

Cavern Manticore: Exposing Iran-Linked Modular C2 Framework

By stcpresearch

Published: 2026-07-06 · Archived: 2026-07-28 02:01:08 UTC

Note: SysAid was not compromised, and no SysAid vulnerability was involved. The attacker had already gained access to the victim environment and abused a legitimate software-deployment feature to deploy malware onto another machine within it.

Key Points

- Check Point Research (CPR) tracks ‘**Cavern Manticore**’ as an Iran-nexus threat actor operating against **Israeli targets**, with a focus on the government and IT sectors.
- Cavern Manticore shares technical overlaps with other Iranian MOIS (Ministry of Intelligence and Security)-linked threat actors, including [MuddyWater](#) and [Lyceum](#).
- CPR observed a **modular C2 framework** in the wild, with all samples built on top of .NET but compiled into different output formats. These components are used as **Cavern agent** and **Cavern modules**.
- The framework’s **anti-analysis** posture relies on uncommon .NET compilation formats (**Mixed-Mode C++/CLI** and **Native AOT**) that force reverse engineers into multiple toolsets and metadata-reconstruction workflows, together with per-module [AppDomain isolation](#) as an **anti-forensics** measure.
- In malware-engine coverage, the majority of observed samples score **zero** or **very low detection** rates on VirusTotal.
- Post-exploitation modules provide the threat actor with **extended capabilities**, including **file system** and **database** browsing, **LDAP** querying, **network** reconnaissance, and tunneling.
- In multiple observed intrusions, the **initial foothold** was achieved through abuse of existing Remote Monitoring and Management (**RMM**) software deployed in the targeted organization.

Introduction

Since early 2026, Check Point Research (CPR) has tracked a **new modular command-and-control framework** used by **Cavern Manticore**, an Iran-nexus APT group primarily targeting Israeli organizations, with a focus on IT providers, and government sectors. Cavern Manticore is an Iran MOIS (Ministry of Intelligence and Security)-linked actor, with links to the OilRig subgroup named [Lyceum](#). The framework reflects a mature and adaptable toolset built around a shared .NET foundation, while using multiple compilation formats across different components, including **.NET Framework**, **.NET Mixed-Mode C++/CLI**, and **.NET Native AOT**. The **compilation format** itself becomes the **anti-analysis layer** that forces reverse engineers into multiple toolsets and metadata-reconstruction workflows.

During our investigation, we observed both **Cavern agents** and **Cavern modules** in the wild, highlighting a modular architecture that separates core communication capabilities from mission-specific post-exploitation functionality. This design allows the operators to tailor deployments per victim environment, limit what defenders and analysts can recover from any single victim and extend access after compromise through specialized modules for reconnaissance, data access, tunneling, and lateral movement.

 Figure 1: Cavern Modules Evade Malware Engines.

Figure 1: Cavern Modules Evade Malware Engines.

Technical Analysis: Cavern – A Modular .NET C2 Framework

1. Cavern at a Glance

Cavern is a modular post-exploitation C2 framework built entirely on .NET, but deliberately compiled into three different binary formats: **.NET Framework (IL-only)**, **Mixed-Mode C++/CLI (IL + Native)**, and **.NET 8 NativeAOT (Native-only)**.

The recovered execution chain **begins** with [SysAid’s software update](#) feature, which the actor leverages to deploy a **WinDirStat DLL sideloading** package to `C:\ProgramData\WinDir\WinDirStat.exe`. The legitimate `WinDirStat.exe` binary loads the trojanized `uxtheme.dll`, which is the **Cavern Agent**, and the agent in turn

loads a dedicated **native communication** module `n-HTComp.dll` to reach the **C2** and then pulls down additional post-exploitation modules on operator command.

 Figure 2: Cavern Agent Execution Chain.

Figure 2: Cavern Agent Execution Chain.

The table below provides an overview of the modules.

Component	Internal Name	Format	Role
Cavern Agent	<code>uxtheme.dll</code>	Mixed-Mode C++/CLI (.NET 4.7.2, IL + Native)	Core backdoor, module orchestrator
Communication Module	<code>n-HTComp.dll</code>	NativeAOT (.NET 8, Native-only)	HTTPS/WebSocket transport, XOR-encrypted traffic
File Manager	<code>mhm.dll</code>	.NET Framework 4.7.2 (IL-only)	File ops, DPAPI decrypt, archive handling
SQL Browser	<code>db.dll</code>	.NET Framework 4.7.2 (IL-only)	Database enumeration, query, export, manipulation
LDAP Module	<code>ode.dll</code>	.NET Framework 4.7.2 (IL-only)	AD recon, user/group enumeration, LDAP brute-force
Network Module	<code>n-ten.dll</code>	NativeAOT (.NET 8, Native-only)	Net recon, port scan, share enum, SMB brute-force
Tunnel Module	<code>n-sws.dll</code>	NativeAOT (.NET 8, Native-only)	SOCKS5 proxy, WebSocket/WSS tunneling

2. Three Compilation Formats as Anti-Analysis

The most distinctive architectural decision in **Cavern** is the deliberate use of **three different .NET compilation targets** across its components. This is not obfuscation in the traditional sense; there is no packer, no control-flow flattening, and no string encryption anywhere in the framework. Instead, **the compilation format itself becomes the anti-analysis layer**, since each of the three formats has to be reversed with a different toolchain and a different workflow, and the analyst has to context-switch between them across components.

- **Pure .NET Framework (IL-only)** modules (`mhm.dll` , `db.dll` , `ode.dll`) retain full symbol metadata, including the shared `Command.Type` enum with all 61 command IDs, readable class names like `ApiEx.DatabaseBrowser` , and meaningful method signatures. These modules are trivially decompilable with tools such as [ILSpy](#) or [dnSpyEx](#). The developers chose this format for the modules that run inside the agent's managed **AppDomain**, where **IL code** is actually required for reflection-based loading.
- **Mixed-Mode C++/CLI (IL + Native)** agents (`uxtheme.dll`) combine **managed .NET code** with **native C++** in a single PE. Its exports are not regular native functions: each one is a tiny native stub (a `jmp` followed by `ud2` padding) in the `.nep` section that forwards the call to a managed method behind it. Reversing this format takes both a .NET decompiler for the managed logic and a native disassembler for the export stubs and the C++ marshaling code, so the **analyst has to reverse the same binary twice in two different toolchains**.
- **NativeAOT .NET 8 (Native-only)** modules (`n-HTComp.dll` , `n-ten.dll` , `n-sws.dll`) compile the **entire .NET runtime statically** into a **single native PE**. The result is usually a 3-6 MB binary with **thousands of stripped** framework functions, a `.managed` executable section, and a `hydrated` BSS-like section where string objects are materialized only at runtime. Security-sensitive **P/Invoke** calls to **APIs** like `WNetAddConnection2` , `NetShareEnum` , or `NetLocalGroupGetMembers` are resolved through runtime descriptor tables instead of appearing in the PE import table, which **hides** the module's real capabilities from import-based triage.

2.1 Tooling Notes for NativeAOT Analysis

NativeAOT is the format that pushed back the hardest during analysis, so it is worth saying a few words on the tooling we put together for it.

To pull useful metadata back out of the NativeAOT samples, we ported Washi’s Ghidra NativeAOT plugin ([ghidra-nativeaot](#); write-up: [Recovering Metadata from .NET Native AOT Binaries](#)) to IDA Pro. The port reconstructs the .NET type system from the runtime’s **ReadyToRun** metadata, rebuilds the `MethodTable` / `EEType` hierarchy, recovers virtual methods, materializes the frozen string literals from the `hydrated` section, and exposes a metadata browser for navigation. It is available at [ida-nativeaot](#).

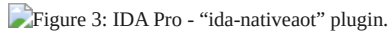
Figure 3: IDA Pro - “ida-nativeaot” plugin.

Figure 3: IDA Pro – “ida-nativeaot” plugin.

To **recover symbols** from the **stripped NativeAOT** .NET 8 modules, we then built a matching .NET 8.0.25 NativeAOT win-x64 “**coverage**” DLL (compiled with PDB) that deliberately exercises the same .NET runtime and class library code the **Cavern** samples rely on, and generated **IDA FLIRT signatures** from it. Applied to the Cavern samples, the signatures matched roughly 60% of all functions, with the matches concentrated on the parts that mattered most for the analysis, e.g., `System.Diagnostics.*`, `System.IO.*`, `System.Net.*`, `System.Security.*`, and `System.Text.*`.

3. The Cavern Agent

3.1 UxTheme Facade and Side-Load Trigger

The **Cavern Agent** is compiled as a **64-bit Mixed-Mode C++/CLI DLL** named `uxtheme.dll` and exports 83 functions that mimic the legitimate Windows theming library. Of these 83 exports, **82 are empty stubs**, single-instruction managed methods that return immediately. The one live export is `EnableThemeDialogTexture`, which serves as the operational entry point for the entire C2 loop.

This design creates a deliberate sandbox trap. Any automated analysis tool that invokes ordinal `#1`, or any other default export, will observe only inert DLL loading behavior and conclude the sample is benign. The real backdoor personality sits entirely behind export ordinal `#20` (`0x14`).

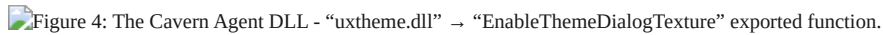
Figure 4: The Cavern Agent DLL - “uxtheme.dll” → “EnableThemeDialogTexture” exported function.

Figure 4: The Cavern Agent DLL – “uxtheme.dll” → “EnableThemeDialogTexture” exported function.

3.2 C2 Polling Loop

Upon invocation, `EnableThemeDialogTexture` creates a singleton mutex (`MYMUTEX123HELLP02` or `MYMUTEX123HELLP04`, depending on the build), initializes the local configuration from `config.txt`, and enters an infinite polling loop. Each iteration builds a command string using the framework’s custom delimiter grammar (`_;` separates fields, `_:` separates arguments) and hands the actual HTTP transport to `n-HTComp.dll`.

Figure 5: The Cavern Agent - Main C2 beacon loop.

Figure 5: The Cavern Agent – Main C2 beacon loop.

3.3 Custom AppDomain Isolation with Post-Execution Unload

One of the most technically interesting mechanisms in the Cavern Agent is its module hosting strategy. Rather than loading .NET modules into the default `AppDomain` via `Assembly.Load` (the common approach in most .NET loaders), **Cavern creates a dedicated AppDomain for each module execution, marshals a proxy object across the domain boundary, invokes the module, and then unloads the entire AppDomain.**

The reason this design choice is operationally relevant is that .NET assemblies loaded into the **default AppDomain cannot be unloaded** without terminating the host process. By isolating each module in its own AppDomain, Cavern gets two things: loaded modules **can be cleanly removed** from memory after execution, leaving no analyzable assembly artifacts behind, and different versions of the same module can be loaded and run one after another without conflict.

Figure 6: The Cavern Agent - “runAssembly” method → AppDomain isolation.

Figure 6: The Cavern Agent – “runAssembly” method → AppDomain isolation.

The `DotNetProxy` class inherits from `MarshalByRefObject`, which allows it to **exist in one AppDomain while being invoked from another**. Inside the isolated domain, it performs standard reflection-based loading (via the `DotNetProxy.runDll` method).

Figure 7: The Cavern Agent - “DotNetProxy.runDll” method → inside the isolated AppDomain.

Figure 7: The Cavern Agent – “DotNetProxy.runDll” method → inside the isolated AppDomain.

3.4 Dual Module Dispatch: Native vs. Managed

The **unified module dispatcher** is `<Module>.run_DLL`, a free function on the global `<Module>` type. The name looks similar to the `DotNetProxy.RunDll` method shown in the previous section, but the two have **different roles**: `<Module>.run_DLL` is the **outer dispatcher** invoked by the agent **for every module load**, and it is also the one that calls into `DotNetProxy.RunDll` (via `<Module>.runAssembly` method) whenever the module turns out to be a managed assembly. The dispatcher itself uses a simple filename convention: modules whose names start with `n-` are treated as native DLLs and loaded via `LoadLibraryA` / `GetProcAddress`, while everything else is treated as a managed .NET assembly and loaded through the AppDomain isolation mechanism described above. Whichever path is taken, the **agent ends up calling the same entry point** on the loaded module: a function named `get_version`.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
// Cavern Agent - <Module>.run_DLL: Unified Module Dispatcher

// Simplified C# reconstruction of the dnSpyEx decompilation

string <Module>.run_DLL(string moduleName, string arguments)
{
    string resolvedPath = get_latest_dll(moduleName); // finds highest-numbered version

    string fileName = Path.GetFileName(resolvedPath);

    if (fileName.StartsWith("n-"))
    {
        // Native module path (NativeAOT compiled)

        IntPtr hModule = LoadLibraryA(resolvedPath);

        if (hModule == IntPtr.Zero)

            return "DLL not found...Maybe you didn't upload it!!!";

        IntPtr pGetVersion = GetProcAddress(hModule, "get_version");

        if (pGetVersion == IntPtr.Zero)

            return "What is this sh*t?! where is get_version?!?";

        var getVersion = Marshal.GetDelegateForFunctionPointer<GetVersionFn>(pGetVersion);

        IntPtr resultPtr = getVersion(Marshal.StringToHGlobalUni(arguments));

        return Marshal.PtrToStringUni(resultPtr);
    }
    else
    {
        // Managed module path (.NET Framework) - loaded in isolated AppDomain

        List<string> argList = new List<string> { arguments };

        return (string)<Module>.runAssembly(

            "mydomain",
```

```

new List<byte>(File.ReadAllBytes(resolvedPath)),

resolvedPath,

string.IsNullOrEmpty(arguments), // noArgs flag

argList,

"MyClass.Program", // fixed class name

"get_version" // fixed method name - the universal interface

);

}

}

// Cavern Agent - <Module>.run_DLL: Unified Module Dispatcher // Simplified C# reconstruction of the dnSpyEx
decompilation string <Module>.run_DLL(string moduleName, string arguments) { string resolvedPath =
get_latest_dll(moduleName); // finds highest-numbered version string fileName = Path.GetFileName(resolvedPath); if
(fileName.StartsWith("n-")) { // Native module path (NativeAOT compiled) IntPtr hModule = LoadLibraryA(resolvedPath);
if (hModule == IntPtr.Zero) return "DLL not found...Maybe you didn't upload it!!!"; IntPtr pGetVersion =
GetProcAddress(hModule, "get_version"); if (pGetVersion == IntPtr.Zero) return "What is this sh*t?! where is
get_version?!?"; var getVersion = Marshal.GetDelegateForFunctionPointer<GetVersionFn>(pGetVersion); IntPtr resultPtr =
getVersion(Marshal.StringToHGlobalUni(arguments)); return Marshal.PtrToStringUni(resultPtr); } else { // Managed
module path (.NET Framework) - loaded in isolated AppDomain List<string> argList = new List<string> { arguments };
return (string)<Module>.runAssembly( "mydomain", new List<byte>(File.ReadAllBytes(resolvedPath)), resolvedPath,
string.IsNullOrEmpty(arguments), // noArgs flag argList, "MyClass.Program", // fixed class name "get_version" // fixed
method name - the universal interface ); } }

```

```

// Cavern Agent - <Module>.run_DLL: Unified Module Dispatcher
// Simplified C# reconstruction of the dnSpyEx decompilation

string <Module>.run_DLL(string moduleName, string arguments)
{
    string resolvedPath = get_latest_dll(moduleName); // finds highest-numbered version
    string fileName = Path.GetFileName(resolvedPath);

    if (fileName.StartsWith("n-"))
    {
        // Native module path (NativeAOT compiled)
        IntPtr hModule = LoadLibraryA(resolvedPath);
        if (hModule == IntPtr.Zero)
            return "DLL not found...Maybe you didn't upload it!!!";

        IntPtr pGetVersion = GetProcAddress(hModule, "get_version");
        if (pGetVersion == IntPtr.Zero)
            return "What is this sh*t?! where is get_version?!?";

        var getVersion = Marshal.GetDelegateForFunctionPointer<GetVersionFn>(pGetVersion);
        IntPtr resultPtr = getVersion(Marshal.StringToHGlobalUni(arguments));
        return Marshal.PtrToStringUni(resultPtr);
    }
    else
    {
        // Managed module path (.NET Framework) - loaded in isolated AppDomain
        List<string> argList = new List<string> { arguments };

        return (string)<Module>.runAssembly(
            "mydomain",
            new List<byte>(File.ReadAllBytes(resolvedPath)),
            resolvedPath,
            string.IsNullOrEmpty(arguments), // noArgs flag

```

```

        arglist,
        "MyClass.Program",           // fixed class name
        "get_version"                // fixed method name - the universal interface
    );
}
}

```

The **native path** contains two **error strings** worth flagging: "What is this sh*t?! where is get_version?!?" and "DLL not found...Maybe you didn't upload it!!!".

Figure 8: The Cavern Agent - native path of dual module dispatch → error strings.

Figure 8: The Cavern Agent – native path of dual module dispatch → error strings.

These **are not** the kind of **polished, neutral diagnostics** a code generator tends to emit. They are written **in the first person**, with **frustration**, profanity and exclamation marks, and they read exactly like an operator talking to themselves while debugging their own tooling. We come back to what this tells us about authorship in the “**Authorship and the Human Factor**” section below.

3.5 Module Versioning and Self-Update

Cavern implements a **numbered DLL versioning scheme**. The function `get_latest_dll` scans the working directory for files matching a base module name with appended numeric suffixes (e.g., `n-HTComp0.dll`, `n-HTComp1.dll`) and **loads the highest-numbered** variant. This allows the operator to push module updates via the C2 without file-name conflicts.

Figure 9: The Cavern Agent - module versioning.

Figure 9: The Cavern Agent – module versioning.

The self-command `002` (exposed via `self_execute` method) accepts a **Base64+GZip-compressed** module payload from the C2, writes it to disk as a new numbered DLL, and, in the case of `uxtheme.dll` itself, **executes a hot-swap**: the running agent renames its own DLL, writes the new version, loads it, calls its `EnableThemeDialogTexture` with `signalCode=200` to signal the update-return path, and terminates. All implemented self-commands are detailed in the next section.

Figure 10: The Cavern Agent - “self_execute” method → self-commands processing.

Figure 10: The Cavern Agent – “self_execute” method → self-commands processing.

3.6 Agent Self-Commands

The agent handles **six built-in self-commands** before reaching the module dispatcher:

Command	Action
001	Update polling interval
002	GZip+Base64 module update (including self-update of <code>uxtheme.dll</code>)
003	Toggle debug logging
004	Activate WebSocket communication mode
005	Close WebSocket connection
006	Reconnect WebSocket

3.7 Startup Cleanup as Anti-Forensics

Newer agent builds perform aggressive directory cleanup on first startup: they enumerate all files and subdirectories in the working directory and **delete everything except** the **Communication Module** (`n-HTComp.dll`), the **configuration file** (`config.txt`), and **log** files. This means any modules delivered by the C2 in a previous session are wiped before the next execution cycle, and the agent reports `"cleared"` to the C2 upon completion.

3.8 Variant Evolution

Three agent builds were recovered, showing **clear iterative** development:

Attribute	Oldest Build	Build 02	Build 04
Mutex	MYMUTEX123HELLP	MYMUTEX123HELLP02	MYMUTEX123HELLP04
C2 Domain	auth.hospitalinstallation.com	google.com.hospitalinstallation.com	google.com.hospitalinstallation.
Config Storage	id.txt (plain 7-char ID)	config.txt (JSON)	config.txt (JSON)
Self-Commands	001-003	001-006 (adds WebSocket)	001-006
Cleanup	None	Working-dir wipe	Working-dir wipe
Debug Default	true	false	true

4. The Communication Module – “n-HTCommp.dll”

The communication module is compiled as a **NativeAOT .NET 8 DLL** (~5.5 MB, with about **21k stripped** framework functions) and exposes a single operational export, `get_version`. Despite the name, this exported function is a **full multi-verb HTTP and WebSocket command dispatcher**. The **agent** passes transport **commands** as delimited strings, and `n-HTCommp.dll` parses the **verb**, performs the **network operation**, and returns the result.

The verb matching is the first place where the **NativeAOT format** makes analysis visibly **harder**. In a **normal .NET** build, a check like `verb == "get"` calls `String.Equals`, and the literal `"get"` lives in the string heap (`#US`), where any `strings` scan will find it. NativeAOT instead compiles the comparison inline: it first checks the length of the verb string, then loads the verb’s UTF-16 characters straight from memory and compares them against hard-coded integer constants. Those **constants** are simply the **verb’s characters** packed together as **numbers**. For `"get"`, the three UTF-16 characters `g` (`0x0067`), `e` (`0x0065`) and `t` (`0x0074`) become the constants `0x650067` and `0x740065` that show up in the comparison.

 Figure 11: The Cavern’s “n-HTCommp.dll” module - verb matching → command dispatching.

Figure 11: The Cavern’s “n-HTCommp.dll” module – verb matching → command dispatching.

This is a **real triage problem** because **every readable string** in this module behaves **differently** than in a **normal .NET** binary. **Frozen string literals** like `https`, `wss`, `text/plain`, the `WebSocket` URL fragments and a handful of error messages live in the `hydrated` section, which is **materialized at runtime** by the **NativeAOT runtime** and only becomes a readable UTF-16 string at that point. A `strings` pass over the DLL on disk does not see them, since on disk that section is a **compressed initialization blob**. They become visible only after the section is **rehydrated**, either by running the sample or by **reconstructing it statically** with the kind of [plugin](#) described in [section 2.1](#).

 Figure 12: The Cavern’s NativeAOT “n-HTCommp.dll” module - “ida-nativeaot” plugin → section rehydrated → strings reconstructed (e.g. User-Agent).

Figure 12: The Cavern’s NativeAOT “n-HTCommp.dll” module – “ida-nativeaot” plugin → section rehydrated → strings reconstructed (e.g. User-Agent).

The **packed verb constants** are even further out of reach: they are not strings at all, they are **integer immediates** baked into the `cmp` instructions of the dispatcher. So in practice a `strings`-based triage of this DLL on disk returns almost **nothing usable**, neither the verb set, nor the URL fragments, nor the user-agent header. The command grammar simply does not exist in any byte sequence that a string scan can pick up.

The dispatcher first marshals the inbound command to a managed string, then splits it on the framework’s two delimiters (`;;_` for the **verb/argument** boundary and `_;` between **arguments**), and dispatches to a **verb handler**.

 Figure 13: The Cavern’s “n-HTCommp.dll” module - command dispatcher → verb/argument separation.

Figure 13: The Cavern’s “n-HTCommp.dll” module – command dispatcher → verb/argument separation.

Each **verb** maps to a **distinct network operation**, and the handlers differ in three operationally meaningful ways: whether the payload is **XORed** with key `0x48` (the in-place traffic transform), whether it is then **Base64-encoded** for the **HTTP body**, and which **HTTP/WS** headers and endpoints they touch. Every **HTTP-based verb** sends a **fixed Microsoft Edge** User-Agent (`Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)`)

Chrome/146.0.0.0 Safari/537.36 Edg/146.0.0.0), and the two **C2-bound verbs** (`get` and `send`) additionally attach a custom `X-User-token` header whose value is the **agent ID** with the literal suffix `00` appended. The summary below was reconstructed by following **each verb handler** through its full **HTTP/WS** request build path:

Verb	Network	Endpoint built from arguments	XOR (0x48)	Base64	User-Agent	X-User-token
<code>get</code>	HTTP GET	<code>args[1] + "/profile"</code>	yes (response body, after Base64 decode)	yes	yes	yes (<code>args[0]</code> + <code>"00"</code>)
<code>send</code>	HTTP POST <code>text/plain</code>	<code>args[1] + "/gallery"</code>	yes (request body, before Base64 encode)	yes	yes	yes (<code>args[0]</code> + <code>"00"</code>)
<code>cget</code>	HTTP GET	<code>args[0]</code> (raw URL)	no	no	yes	no
<code>cpost</code>	HTTP POST	<code>args[0]</code> (raw URL), body <code>args[1]</code> , content-type <code>args[2]</code> (default <code>text/plain</code>)	no	no	yes	no
<code>upload</code>	HTTP POST <code>multipart/form-data</code>	<code>args[0]</code> (raw URL), file <code>args[1]</code> from disk as form field file (<code>application/octet-stream</code>)	no	no	yes	no
<code>ws</code>	WS Open + initial WS Send	<code>wss://<host>/socket</code> if <code>args[0]</code> starts with <code>https</code> , otherwise <code>ws://<host>/socket</code> ; immediately sends <code>args[1] + "00"</code> as the first text frame	yes (initial frame only)	no	n/a	n/a (sent inside first frame instead)
<code>getws</code>	WS Recv	active socket	yes (whole accumulated payload, then UTF-8 decoded)	no	n/a	n/a
<code>sendsws</code>	WS Send	active socket	yes (UTF-8 bytes, then framed as text)	no	n/a	n/a
<code>closews</code>	WS Close	active socket	n/a	n/a	n/a	n/a

A few practical observations follow directly from the table. First, the **XOR transform** with key `0x48` is the framework's traffic-encoding layer, and it applies to **every C2-bound** channel: it is on both directions of the **HTTP** path (`get` / `send`)

and on both directions of the **WebSocket** path (`getws` / `sendws`), plus the **initial WS** handshake frame. The only **verbs** that **bypass** it are `cget` , `cpost` and `upload` , which talk to **operator-supplied URLs** that have **nothing** to do with the **Cavern C2**. Second, **Base64** is applied on top of XOR only for the HTTP transport (`get` and `send`), where the body has to survive as text/plain; the WebSocket path skips Base64 because it can carry the raw XORed bytes inside a text frame directly. Third, the `User-Agent` header is **fixed** across every **HTTP verb**, including the operator-driven ones, which **makes the UA itself a stable host artifact for detection**.

 Figure 14: The Cavern’s “n-HTCommp.dll” module - “send” command handler.

Figure 14: The Cavern’s “n-HTCommp.dll” module – “send” command handler.

5. Post-Exploitation Modules

All **Cavern modules**, regardless of **compilation format**, share a uniform interface contract:

the **agent** invokes `get_version(List<string> args)` for **managed** modules or `get_version(wchar_t* args)` for **native** modules. The **first argument** carries a newline-delimited **command string** using numeric **command IDs** from the shared `Command.Type` **enum**, with `_;` and `_,` as **field/argument** delimiters.

 Figure 15: Post-exploitation modules - example “mhm.dll” managed module → arguments processing.

Figure 15: Post-exploitation modules – example “mhm.dll” managed module → arguments processing.

The **full command set** is defined once in that shared **enum** and reused across **every module**. We recovered it intact from the **.NET Framework modules**, which keep their symbols, and it is worth showing in full because the **IDs are grouped by capability area**. The grouping itself is informative: each block of numbers maps to one **functional** category, and the **gaps** between blocks line up neatly with the **individual modules** that **implement** them.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

```
public enum Command.Type
{
    NONE = 0, // 0x0 - sentinel / no command (Agent: uxtheme.dll)

    CHANGE_ALIVE_TIME = 1, // 0x1 - update polling interval (Agent: uxtheme.dll)

    INFO = 101, // 0x65 - host information (mhm.dll)

    CRYPT_DECRYPT = 102, // 0x66 - DPAPI decrypt (mhm.dll)

    TOKEN_INFO = 103, // 0x67 - token information (mhm.dll)

    TIME_INFO = 104, // 0x68 - time information (mhm.dll)

    SQL_QUERY = 201, // 0xC9 - SQL query (db.dll)

    COPY_DIR = 301, // 0x12D - copy directory (mhm.dll)

    COPY_FILE = 302, // 0x12E - copy file (mhm.dll)

    DRIVES_LIST = 305, // 0x131 - list drives (mhm.dll)

    FILES_FOLDERS_INFO = 306, // 0x132 - files / folders info (mhm.dll)

    MOVE_FILE = 307, // 0x133 - move file (mhm.dll)

    MOVE_FOLDER = 308, // 0x134 - move folder (mhm.dll)

    DEL_FILE = 309, // 0x135 - delete file (mhm.dll)

    DEL_FOLDER = 310, // 0x136 - delete folder (mhm.dll)
```

CREATE_FOLDER = 311, // 0x137 - create folder (mhm.dll)

FILE_FOLDER_LIST = 312, // 0x138 - list files / folders (mhm.dll)

SEARCH_FILE = 313, // 0x139 - search files (mhm.dll)

MOVE = 314, // 0x13A - move (mhm.dll)

LDAP_TEST = 401, // 0x191 - LDAP bind test (ode.dll)

LDAP_ALL_GROUPS = 402, // 0x192 - enumerate all groups (ode.dll)

LDAP_ALL_USERS = 403, // 0x193 - enumerate all users (ode.dll)

LDAP_GROUP_MEMBER = 404, // 0x194 - group members (ode.dll)

LDAP_SEARCH = 405, // 0x195 - LDAP search (ode.dll)

LDAP_USER_PROPS = 406, // 0x196 - user properties (ode.dll)

LDAP_BRUTE = 407, // 0x197 - LDAP brute-force (ode.dll)

PROC_KILL = 501, // 0x1F5 - kill process (not in modular set; older Cav3rn)

PROC_LIST = 502, // 0x1F6 - list processes (not in modular set; older Cav3rn)

REG_ADD = 601, // 0x259 - registry add (not in modular set; older Cav3rn)

REG_DEL = 602, // 0x25A - registry delete (not in modular set; older Cav3rn)

REG_QRY_SUBKEYS = 603, // 0x25B - registry query subkeys (not in modular set; older Cav3rn)

REG_QRY_VALUE = 604, // 0x25C - registry query value (not in modular set; older Cav3rn)

SRV_LIST = 701, // 0x2BD - list services (not in modular set; older Cav3rn)

SRV_RESET = 702, // 0x2BE - reset service (not in modular set; older Cav3rn)

SRV_START = 703, // 0x2BF - start service (not in modular set; older Cav3rn)

SRV_STOP = 704, // 0x2C0 - stop service (not in modular set; older Cav3rn)

GZ_READ = 801, // 0x321 - GZip read (download) (mhm.dll)

GZ_WRITE = 802, // 0x322 - GZip write (upload) (mhm.dll)

COMPRESS_DIR = 803, // 0x323 - compress directory (mhm.dll)

DECOMPRESS_DIR = 804, // 0x324 - decompress directory (mhm.dll)

DECOMPRESS_FILE = 805, // 0x325 - decompress file (mhm.dll)

LIST_ARCHIVE_ITEMS = 806, // 0x326 - list archive items (mhm.dll)

DBBrowser = 901, // 0x385 - SQL database browser (db.dll)

NET_DNS_RESOLVE = 1101, // 0x44D - DNS resolve (n-ten.dll)

NET_INTERFACES = 1102, // 0x44E - network interfaces (n-ten.dll)

NET_IP_CONFIG = 1103, // 0x44F - IP configuration (n-ten.dll)

NET_PING = 1104, // 0x450 - ping host (n-ten.dll)

NET_STAT = 1106, // 0x452 - netstat / connections (n-ten.dll)

NET_USE_GET_MAP_DRV = 1201, // 0x4B1 - list mapped drives (n-ten.dll)

NET_USE_MAP_DRV = 1202, // 0x4B2 - map network drive (n-ten.dll)

NET_USE_UNMAP_DRV = 1203, // 0x4B3 - unmap network drive (n-ten.dll)

NET_USE_BRUTE = 1204, // 0x4B4 - SMB credential brute-force (n-ten.dll)

NET_USR_GET = 1301, // 0x515 - user info (n-ten.dll)

NET_USR_GET_ALL = 1302, // 0x516 - enumerate users (n-ten.dll)

NET_LOCAL_GROUP = 1401, // 0x579 - enumerate local groups (n-ten.dll)

NET_LOCAL_GROUP_MEMBERS = 1402, // 0x57A - local group members (n-ten.dll)

NET_ARP_TABLE = 1501, // 0x5DD - ARP table (n-ten.dll)

NET_GET_DOMAIN = 1601, // 0x641 - current domain / workstation (n-ten.dll)

NET_VIEW_SHARE_LIST = 1602, // 0x642 - list shares on a host (n-ten.dll)

NET_DOMAIN_COMPUTERS = 1603, // 0x643 - list computers in domain (n-ten.dll)

NET_PORT_SCN = 1701 // 0x6A5 - TCP port scan (n-ten.dll)

}

```
public enum Command.Type { NONE = 0, // 0x0 - sentinel / no command (Agent: uxtheme.dll) CHANGE_ALIVE_TIME =
1, // 0x1 - update polling interval (Agent: uxtheme.dll) INFO = 101, // 0x65 - host information (mhm.dll)
CRYPTO_DECRYPT = 102, // 0x66 - DPAPI decrypt (mhm.dll) TOKEN_INFO = 103, // 0x67 - token information (mhm.dll)
TIME_INFO = 104, // 0x68 - time information (mhm.dll) SQL_QUERY = 201, // 0xC9 - SQL query (db.dll) COPY_DIR =
301, // 0x12D - copy directory (mhm.dll) COPY_FILE = 302, // 0x12E - copy file (mhm.dll) DRIVES_LIST = 305, // 0x131
- list drives (mhm.dll) FILES_FOLDERS_INFO = 306, // 0x132 - files / folders info (mhm.dll) MOVE_FILE = 307, //
0x133 - move file (mhm.dll) MOVE_FOLDER = 308, // 0x134 - move folder (mhm.dll) DEL_FILE = 309, // 0x135 - delete
file (mhm.dll) DEL_FOLDER = 310, // 0x136 - delete folder (mhm.dll) CREATE_FOLDER = 311, // 0x137 - create folder
(mhm.dll) FILE_FOLDER_LIST = 312, // 0x138 - list files / folders (mhm.dll) SEARCH_FILE = 313, // 0x139 - search
files (mhm.dll) MOVE = 314, // 0x13A - move (mhm.dll) LDAP_TEST = 401, // 0x191 - LDAP bind test (ode.dll)
LDAP_ALL_GROUPS = 402, // 0x192 - enumerate all groups (ode.dll) LDAP_ALL_USERS = 403, // 0x193 - enumerate
all users (ode.dll) LDAP_GROUP_MEMBER = 404, // 0x194 - group members (ode.dll) LDAP_SEARCH = 405, // 0x195 -
LDAP search (ode.dll) LDAP_USER_PROPS = 406, // 0x196 - user properties (ode.dll) LDAP_BRUTE = 407, // 0x197 -
LDAP brute-force (ode.dll) PROC_KILL = 501, // 0x1F5 - kill process (not in modular set; older Cav3rn) PROC_LIST =
502, // 0x1F6 - list processes (not in modular set; older Cav3rn) REG_ADD = 601, // 0x259 - registry add (not in modular
set; older Cav3rn) REG_DEL = 602, // 0x25A - registry delete (not in modular set; older Cav3rn) REG_QRY_SUBKEYS =
603, // 0x25B - registry query subkeys (not in modular set; older Cav3rn) REG_QRY_VALUE = 604, // 0x25C - registry
query value (not in modular set; older Cav3rn) SRV_LIST = 701, // 0x2BD - list services (not in modular set; older Cav3rn)
SRV_RESET = 702, // 0x2BE - reset service (not in modular set; older Cav3rn) SRV_START = 703, // 0x2BF - start service
(not in modular set; older Cav3rn) SRV_STOP = 704, // 0x2C0 - stop service (not in modular set; older Cav3rn) GZ_READ
= 801, // 0x321 - GZip read (download) (mhm.dll) GZ_WRITE = 802, // 0x322 - GZip write (upload) (mhm.dll)
COMPRESS_DIR = 803, // 0x323 - compress directory (mhm.dll) DECOMPRESS_DIR = 804, // 0x324 - decompress
directory (mhm.dll) DECOMPRESS_FILE = 805, // 0x325 - decompress file (mhm.dll) LIST_ARCHIVE_ITEMS = 806, //
0x326 - list archive items (mhm.dll) DBBrower = 901, // 0x385 - SQL database browser (db.dll) NET_DNS_RESOLVE =
1101, // 0x44D - DNS resolve (n-ten.dll) NET_INTERFACES = 1102, // 0x44E - network interfaces (n-ten.dll)
NET_IP_CONFIG = 1103, // 0x44F - IP configuration (n-ten.dll) NET_PING = 1104, // 0x450 - ping host (n-ten.dll)
NET_STAT = 1106, // 0x452 - netstat / connections (n-ten.dll) NET_USE_GET_MAP_DRV = 1201, // 0x4B1 - list mapped
drives (n-ten.dll) NET_USE_MAP_DRV = 1202, // 0x4B2 - map network drive (n-ten.dll) NET_USE_UNMAP_DRV =
1203, // 0x4B3 - unmap network drive (n-ten.dll) NET_USE_BRUTE = 1204, // 0x4B4 - SMB credential brute-force (n-
ten.dll) NET_USR_GET = 1301, // 0x515 - user info (n-ten.dll) NET_USR_GET_ALL = 1302, // 0x516 - enumerate users
(n-ten.dll) NET_LOCAL_GROUP = 1401, // 0x579 - enumerate local groups (n-ten.dll)
NET_LOCAL_GROUP_MEMBERS = 1402, // 0x57A - local group members (n-ten.dll) NET_ARP_TABLE = 1501, //
0x5DD - ARP table (n-ten.dll) NET_GET_DOMAIN = 1601, // 0x641 - current domain / workstation (n-ten.dll)
NET_VIEW_SHARE_LIST = 1602, // 0x642 - list shares on a host (n-ten.dll) NET_DOMAIN_COMPUTERS = 1603, //
0x643 - list computers in domain (n-ten.dll) NET_PORT_SCN = 1701 // 0x6A5 - TCP port scan (n-ten.dll) }
```

```
public enum Command.Type
{
    NONE = 0, // 0x0 - sentinel / no command (Agent: uxtheme.dll)
    CHANGE_ALIVE_TIME = 1, // 0x1 - update polling interval (Agent: uxtheme.dll)
```

INFO	= 101, // 0x65 - host information	(mhm.dll)
CRYPT_DECRYPT	= 102, // 0x66 - DPAPI decrypt	(mhm.dll)
TOKEN_INFO	= 103, // 0x67 - token information	(mhm.dll)
TIME_INFO	= 104, // 0x68 - time information	(mhm.dll)
SQL_QUERY	= 201, // 0xC9 - SQL query	(db.dll)
COPY_DIR	= 301, // 0x12D - copy directory	(mhm.dll)
COPY_FILE	= 302, // 0x12E - copy file	(mhm.dll)
DRIVES_LIST	= 305, // 0x131 - list drives	(mhm.dll)
FILES_FOLDERS_INFO	= 306, // 0x132 - files / folders info	(mhm.dll)
MOVE_FILE	= 307, // 0x133 - move file	(mhm.dll)
MOVE_FOLDER	= 308, // 0x134 - move folder	(mhm.dll)
DEL_FILE	= 309, // 0x135 - delete file	(mhm.dll)
DEL_FOLDER	= 310, // 0x136 - delete folder	(mhm.dll)
CREATE_FOLDER	= 311, // 0x137 - create folder	(mhm.dll)
FILE_FOLDER_LIST	= 312, // 0x138 - list files / folders	(mhm.dll)
SEARCH_FILE	= 313, // 0x139 - search files	(mhm.dll)
MOVE	= 314, // 0x13A - move	(mhm.dll)
LDAP_TEST	= 401, // 0x191 - LDAP bind test	(ode.dll)
LDAP_ALL_GROUPS	= 402, // 0x192 - enumerate all groups	(ode.dll)
LDAP_ALL_USERS	= 403, // 0x193 - enumerate all users	(ode.dll)
LDAP_GROUP_MEMBER	= 404, // 0x194 - group members	(ode.dll)
LDAP_SEARCH	= 405, // 0x195 - LDAP search	(ode.dll)
LDAP_USER_PROPS	= 406, // 0x196 - user properties	(ode.dll)
LDAP_BRUTE	= 407, // 0x197 - LDAP brute-force	(ode.dll)
PROC_KILL	= 501, // 0x1F5 - kill process	(not in modular set; older Cav3r)
PROC_LIST	= 502, // 0x1F6 - list processes	(not in modular set; older Cav3r)
REG_ADD	= 601, // 0x259 - registry add	(not in modular set; older Cav3r)
REG_DEL	= 602, // 0x25A - registry delete	(not in modular set; older Cav3r)
REG_QRY_SUBKEYS	= 603, // 0x25B - registry query subkeys	(not in modular set; older Cav3r)
REG_QRY_VALUE	= 604, // 0x25C - registry query value	(not in modular set; older Cav3r)
SRV_LIST	= 701, // 0x2BD - list services	(not in modular set; older Cav3r)
SRV_RESET	= 702, // 0x2BE - reset service	(not in modular set; older Cav3r)
SRV_START	= 703, // 0x2BF - start service	(not in modular set; older Cav3r)
SRV_STOP	= 704, // 0x2C0 - stop service	(not in modular set; older Cav3r)
GZ_READ	= 801, // 0x321 - GZip read (download)	(mhm.dll)
GZ_WRITE	= 802, // 0x322 - GZip write (upload)	(mhm.dll)
COMPRESS_DIR	= 803, // 0x323 - compress directory	(mhm.dll)
DECOMPRESS_DIR	= 804, // 0x324 - decompress directory	(mhm.dll)
DECOMPRESS_FILE	= 805, // 0x325 - decompress file	(mhm.dll)
LIST_ARCHIVE_ITEMS	= 806, // 0x326 - list archive items	(mhm.dll)
DBBrowser	= 901, // 0x385 - SQL database browser	(db.dll)
NET_DNS_RESOLVE	= 1101, // 0x44D - DNS resolve	(n-ten.dll)
NET_INTERFACES	= 1102, // 0x44E - network interfaces	(n-ten.dll)
NET_IP_CONFIG	= 1103, // 0x44F - IP configuration	(n-ten.dll)
NET_PING	= 1104, // 0x450 - ping host	(n-ten.dll)
NET_STAT	= 1106, // 0x452 - netstat / connections	(n-ten.dll)
NET_USE_GET_MAP_DRV	= 1201, // 0x4B1 - list mapped drives	(n-ten.dll)
NET_USE_MAP_DRV	= 1202, // 0x4B2 - map network drive	(n-ten.dll)
NET_USE_UNMAP_DRV	= 1203, // 0x4B3 - unmap network drive	(n-ten.dll)
NET_USE_BRUTE	= 1204, // 0x4B4 - SMB credential brute-force	(n-ten.dll)
NET_USR_GET	= 1301, // 0x515 - user info	(n-ten.dll)
NET_USR_GET_ALL	= 1302, // 0x516 - enumerate users	(n-ten.dll)

```

NET_LOCAL_GROUP      = 1401, // 0x579 - enumerate local groups      (n-ten.dll)
NET_LOCAL_GROUP_MEMBERS = 1402, // 0x57A - local group members      (n-ten.dll)

NET_ARP_TABLE        = 1501, // 0x5DD - ARP table                (n-ten.dll)

NET_GET_DOMAIN        = 1601, // 0x641 - current domain / workstation (n-ten.dll)
NET_VIEW_SHARE_LIST   = 1602, // 0x642 - list shares on a host         (n-ten.dll)
NET_DOMAIN_COMPUTERS  = 1603, // 0x643 - list computers in domain      (n-ten.dll)

NET_PORT_SCN          = 1701 // 0x6A5 - TCP port scan                (n-ten.dll)
}

```

The **enum** defines **61 command IDs** in total. Most map directly to a handler in one of the recovered modules, but a handful (such as the **5xx process** and **6xx / 7xx registry and service** ranges) have no implementation in any sample we obtained, which suggests at least one module **was never delivered to the victim** and is still missing from our set.

Two **older Cav3rn** -era samples found on VirusTotal during this writeup also help frame that gap. They **predate the rename**, are nearly identical to each other, and are **not part of the modular intrusion** documented here, but each ships every **ApiEx.*** capability (**ApiEx.Proc**, **ApiEx.Reg**, **ApiEx.Serv** included – related to the **5xx / 6xx / 7xx** command IDs) inside a **single .NET DLL** under namespace **CAV3RN_APIEX_Module** rather than across separate modules. Transport in those builds is split: the **Cav3rn agent** itself only reads **steganographic command PNGs** from a local **inpt** directory and writes result PNGs into **outpt**, while the HTTP exchange against the C2 is performed by a **separate HTTP companion** module (**CAV3RN_Http_Module**), which we later recovered as a third **Cav3rn** -era sample. The companion consumes the same **Domain[]** and **PageName = "cac.aspx"** constants the agent carries, **POSTs** `s=<timestamp>&id=<AgentID>&q=<XOR+Base32 telemetry>` to `https://adserviceupdate[.]com|hygienehistory[.]com/cac.aspx`, and expects a response whose body starts with a fixed **21-byte JPEG magic header** and whose **Content-Disposition: filename=** value is **XOR+Base32-encrypted** with the **AgentID**, then drops the carved payload into the same local **inpt** directory the agent reads from. Two details in that exchange show that **cac.aspx** is an **operator-deployed handler** rather than an abused legitimate page: the request and response shape is a **custom protocol** no clean IIS server would understand or produce, and the companion's **ServerCertificateValidationCallback** is hard-coded to **always return true**, meaning the operator is explicitly not relying on a properly-issued certificate for the C2 endpoint. Whether the underlying IIS server is **attacker-stood-up** or **cac.aspx** was planted on a **third-party host** the operator does not fully control is not something the binary distinguishes.

The **modern framework** collapses both halves into **n-HTComp.dll** with direct **HTTPS / WebSocket**. The command set is also smaller and clearly under active development, and there is **no NativeAOT**, **no Mixed-Mode** wrapper, and **no AppDomain** isolation. **Today's Cavern** is a **refactor** of that **same project**, split across separate modules and rebuilt around three different compilation formats to harden the analysis. The three hashes (**Cav3rn** -era samples) are listed in the **IOC section** as the **older Cav3rn agent** (two near-identical builds) and the **older Cav3rn HTTP module**; the rest of this publication stays focused on the **modular generation** actually used in the intrusion.

5.1 File Manager – “mhm.dll”

The **file manager module** implements the broadest command surface across three of the **enum** blocks (the **1xx information** block **101-104**, the **3xx file/directory** block **301-314**, and the **8xx archive** block **801-806**): host information collection, DPAPI decryption, drive/file/directory enumeration, recursive file search with content matching, GZip+Base64 file transfer in both directions, ZIP archive creation/extraction, and file/directory manipulation. It **does not implement** the **5xx**, **6xx**, or **7xx** ranges even though those **IDs are present** in the **shared enum** it ships.

Its most notable capability is **DPAPI decryption of operator-supplied blobs**. The **CryptDecrypt** function takes a **Base64-encoded DPAPI-protected** blob, calls **ProtectedData.Unprotect** with **DataProtectionScope.CurrentUser**, and returns the decrypted plaintext. Because the module runs inside the victim's process under their user token, this lets the operator decrypt any **DPAPI-protected** secret that belongs to the **compromised user**.

 Figure 16: The Cavern's “mhm.dll” module - “CryptDecrypt” DPAPI decryption.

Figure 16: The Cavern's “mhm.dll” module – “CryptDecrypt” DPAPI decryption.

An **older variant** of **mhm.dll** retains **legacy “Cav3rn” naming artifacts** in its static configuration: file extensions **.CvnC.png**, **.CvnA.png**, **.CvnR.png** for command, API, and result files, respectively, a config filename **Cvn.cfg**, a hardcoded page name **cac.aspx**, and embedded JPEG header magic bytes. These artifacts point to

an **earlier webshell-style transport layer** (the HTTP side fronted by an ASP.NET page on a separate IIS server, invoked by the **older Cav3rn HTTP module** covered in **Section 5**, not by this module or by the older Cav3rn agent itself) that was **retired** when the framework **evolved** from “Cav3rn” to “Cavern” and moved to the `n-HTComp.dll` native communication module.

 Figure 17: The Cavern’s “mhm.dll” module - older variant → legacy “Cav3rn” configuration.

Figure 17: The Cavern’s “mhm.dll” module – older variant → legacy “Cav3rn” configuration.

5.2 SQL Database Browser – “db.dll”

The **database module** implements a REST-like route dispatcher that accepts **JSON commands** with operator-supplied **SQL Server credentials** passed through **pseudo-HTTP** headers. It supports **SQL database enumeration, query, export, and manipulation**.

 Figure 18: The Cavern’s “db.dll” module - SQL database browser.

Figure 18: The Cavern’s “db.dll” module – SQL database browser.

The connection pool caches SQL connections keyed by connection string. **Credentials** are supplied per-request via `x-db-user`, `x-db-password`, `x-db-host`, with optional `x-db-encrypt` and `x-db-trust-cert` fields, a convention borrowed from HTTP header-based authentication patterns.

5.3 LDAP / Active Directory Module – “ode.dll”

The **LDAP module** provides **Active Directory reconnaissance** and **credential testing**. It auto-discovers the LDAP server and base DN from `LDAP://RootDSE` when not explicitly supplied, performs paged searches with a page size of 1,000, and always accepts TLS certificates without validation.

The most operationally significant function is `LdapBrute`, which accepts semicolon-delimited **username** and **hex-encoded password lists**, supports **file-based input** via the `<path` prefix convention, and includes a **configurable** inter-attempt delay with **break-on-success** logic.

 Figure 19: The Cavern’s “ode.dll” LDAP module → “LdapBrute” method.

Figure 19: The Cavern’s “ode.dll” LDAP module → “LdapBrute” method.

5.4 Network Reconnaissance Module – “n-ten.dll” (NativeAOT)

The **network module** is compiled as **NativeAOT** and provides **network reconnaissance, port scan, share enumeration, and SMB brute-force**. It resolves its **security-sensitive** Windows APIs at **runtime** through **P/Invoke** descriptor tables, which keep them **out of the PE import** table. Static analysis of the P/Invoke resolution data recovered **21 dynamically-loaded** API descriptors. A selection of the most **security-relevant** ones is shown below:

P/Invoke Target	Library	Purpose
<code>WNetAddConnection2</code>	<code>mpr.dll</code>	Map network drive with credentials
<code>WNetCancelConnection2</code>	<code>mpr.dll</code>	Unmap network drive
<code>WNetOpenEnum</code> / <code>WNetEnumResource</code>	<code>mpr.dll</code>	Enumerate network resources
<code>NetUserEnum</code> / <code>NetUserGetInfo</code>	<code>netapi32.dll</code>	User enumeration
<code>NetLocalGroupEnum</code> / <code>GetMembers</code>	<code>netapi32.dll</code>	Local group enumeration
<code>NetServerEnum</code>	<code>netapi32.dll</code>	Domain computer discovery
<code>NetShareEnum</code>	<code>netapi32.dll</code>	Share enumeration
<code>NetWkstaGetInfo</code>	<code>netapi32.dll</code>	Domain/workstation info

The `NetUseBrute` function iterates over **operator-supplied credential** pairs, calling `WNetAddConnection2` against a target share with each pair and immediately disconnecting successful connections via `WNetCancelConnection2`, which gives the operator an **SMB-based credential spraying primitive**.

 Figure 20: The Cavern’s “n-ten.dll” module - “NetUseBrute” function → “WNetAddConnection2”.

Figure 20: The Cavern’s “n-ten.dll” module – “NetUseBrute” function → “WNetAddConnection2”.

5.5 SOCKS5 / WebSocket Tunnel – “n-sws.dll” (NativeAOT)

The **tunnel module** implements a full **SOCKS5 proxy** and **WebSocket/WSS tunnel** in both **server** and **client** modes. Its `get_version` export parses operator-supplied configuration, constructs a command-line argument vector, and dispatches to the internal argument parser, which supports:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

Server: `-s -tp <tunnel_port> -sp <socks5_port> -u <user> -p <pass> [-i <info_url>]`

Client: `-c -ti <tunnel_ip|domain> -tp <tunnel_port> [-ll <log_level>]`

Server: `-s -tp <tunnel_port> -sp <socks5_port> -u <user> -p <pass> [-i <info_url>]` Client: `-c -ti <tunnel_ip|domain> -tp <tunnel_port> [-ll <log_level>]`

```
Server: -s -tp <tunnel_port> -sp <socks5_port> -u <user> -p <pass> [-i <info_url>]
Client: -c -ti <tunnel_ip|domain> -tp <tunnel_port> [-ll <log_level>]
```

In **server** mode, it binds **HTTP/HTTPS** listeners, accepts incoming **WebSocket** upgrades, enforces **username/password authentication**, and relays **SOCKS5** proxy traffic through the **WebSocket** tunnel. A built-in HTTP status page at `/index.htm` returns a `Server Status` HTML response, a small operational convenience. The **tunnel protocol** handles **five** message opcodes: **connect**, **heartbeat**, **data**, **disconnect**, and **error**.

The binary also preserves **developer typos** such as `"tunnel message received"` and `"handeling connect ms"`. Misspellings like these are another small **human fingerprint**, the kind of thing a person types in a hurry and a code generator generally does not produce. We pull these threads together in the next section.

6. Attribution Indicators

The **recovered artifacts** contain several developer and infrastructure fingerprints:

- **PDB paths** across three modules consistently reference `C:\Users\rick\Desktop\Modules\cavern\`, which establishes “**rick**” as the developer username and “**cavern**” as the **internal project name**.
- **C2 infrastructure** uses subdomains of `hospitalinstallation[.]com`: `auth[.]hospitalinstallation[.]com` (older builds) and `google[.]com[.]hospitalinstallation[.]com` (newer builds, where the `google[.]com[.]` prefix is a simple visual trick aimed at anyone skimming proxy logs).
- **Legacy naming** in the **older** `mhm.dll` variant references `Cav3rn` (with a leetspeak “3”) through field names like `Cav3rnCommandExt`, which suggests the framework was renamed from “**Cav3rn**” to “**Cavern**” during its **development**.
- **Cross-version continuity**. Two **older non-modular** `Cav3rn` samples (listed in **IOCs** as the older `Cav3rn agent`) carry the same `ApiEx.*` capability tree, the same `Command.Type` **enum** and the same idiosyncratic method names that **today’s modular Cavern** is built on top of. The newer framework adds commands (`LDAP_BRUTE`, `CRYPT_DECRYPT`, archive ops and the `NET_PORT_SCN` block), retires the **webshell** + **steganography** transport in favor of `n-HTComp.dll`, and splits the codebase across three different compilation formats – a **refactor of the same project**, not a rewrite.

7. Authorship and the Human Factor

It is worth **pausing on a question** that comes up with almost **every new toolset we look at today**: how much of this **was written by a person**, and how much **by an AI coding assistant**. In 2026 it is genuinely hard to imagine a project of this size being built **with no AI assistance** at all, and we would not claim that **Cavern** was. Boilerplate such as the JSON formatting, the LINQ-heavy collection handling, and the standard `P/Invoke` signatures could easily have been drafted or completed with a model. That kind of help is so **common now** that its presence would tell us very little.

What the artifacts do tell us, and **tell us clearly**, is that a **human** was **significantly** and **substantively involved** in building this framework. The evidence is in the rough edges that a code generator tends to sand off:

- **Error strings written in frustration.** The native module dispatcher of the **Cavern agent** returns "What is this sh*t?! where is get_version?!?" when an export is missing and "DLL not found...Maybe you didn't upload it!!!" when a module is absent. These are **first-person**, **profane**, and **exasperated**. They are the voice of an operator debugging their own tooling, not the neutral phrasing a model defaults to.
- **Typos baked into the binaries.** The tunnel module carries "tunnel message received" and "handeling connect ms", and the SQL module builds a query as `SELECT TOP({0}) *FROM[{1}].[{2}]` with the space dropped before `FROM`. Small slips like these are what a **person produces** while typing quickly.
- **Idiosyncratic, hand-picked names.** Hardcoded markers such as the `MYMUTEX123HELLP02` / `MYMUTEX123HELLP04` mutexes and the leetspeak `Cav3rn` to `Cavern` rename are **personal choices**, the kind of naming a developer reaches for, **not output a model** would converge on.
- **Inconsistencies across modules.** Casing drifts (`netapi32.dll` in some descriptors, `Netapi32.dll` in others), debug strings read like scratch notes (`No Handler for path [...] ++`), and the command grammar is bespoke rather than a library default.

None of these are individually conclusive, but **together** they form a **consistent** picture. The higher-level decisions (the **three-format compilation** strategy, the **per-module AppDomain isolation** with post-execution unload, the numbered **self-update scheme**) reflect **deliberate design** by someone who understood the trade-offs. The low-level texture (the **frustration**, the **typos**, the **personal naming**) reflects **hands-on human coding**. Our assessment is that **Cavern** is a **human-authored framework**, very plausibly built with some AI assistance for routine code, but **driven** and **shaped** throughout by a **developer** rather than generated end to end.

Victimology

Our analysis indicates that **Cavern Manticore** is primarily focused on **Israeli targets**, with particular interest in organizations operating in the government and IT sectors. Recent campaigns suggest that the threat actor possesses a strong understanding of the complex IT supplier chains within Israel's cyber ecosystem. In several cases, we observed evidence of the actor moving from an initial compromised IT provider to a second-hop provider before ultimately reaching the intended target organization. This activity highlights the operational value of trusted service-provider relationships, particularly where Remote Monitoring and Management (**RMM**) solutions are deployed. By abusing these tools, the actor can move laterally between victims and deliver malicious software disguised as legitimate updates. The actor also appears to leverage browser-based remote desktop technologies to access targets of interest and, in some cases, abuse built-in features such as remote printing to exfiltrate data when clipboard-based copy-paste or file-transfer capabilities are restricted.

Attribution

During our analysis of an **older Cavern Manticore** toolset, we identified a communication module (`CAV3RN_Http_Module`) that uses a webshell-style ASP.NET handler, `cac.aspx`, hosted on a separate IIS server at one of two attacker-controlled or attacker-deployed domains and used as the command-and-control endpoint. The use of victim-side infrastructure to proxy C2 traffic, combined with XOR-based obfuscation, Base64 encoding, and a fixed verb set per backdoor, is consistent with techniques we have previously observed in operations attributed to [OilRig subgroup named Lyceum](#). Additional overlaps further support a possible Iranian nexus: the targeting of SysAid servers has been observed in past activity linked to Iranian MOIS-aligned actors, including [MuddyWater](#), and this campaign similarly focused on major IT providers in Israel. Finally, WHOIS analysis of the root domain observed in the campaign, `hospitalinstallation[.]com`, showed that it was registered through *Fars Data*, an Iranian hosting provider. Taken together, these technical evidences suggest a connection to Iranian-nexus threat activity.

Conclusion

Cavern Manticore illustrates the continued evolution of Iran-nexus cyber capabilities, exposing a **mature** and **modular C2 framework** that can be rapidly adapted to new campaigns, targets, and operational requirements. The adversary's ability to gain access to organizations in the defense and government sectors during the U.S. military campaign "Operation Epic Fury" demonstrates both a high operational tempo and a disciplined approach to target selection.

This activity also emphasizes the persistent risk posed by supply-chain compromise. In several cases, a compromised IT supplier was not the final objective, but rather the first hop toward a higher-value target. By abusing trusted access relationships, the operators were able to move across organizational boundaries while blending into legitimate administrative workflows.

The campaign further highlights the expanding role of Remote Monitoring and Management tools (RMM) as an evolution of traditional living-off-the-land techniques. For defenders, this reinforces the need to monitor anomalous activity originating from otherwise benign RMM software, enforce strict access controls, limit remote sessions, and reduce the overall attack surface exposed through third-party management infrastructure.

By decoupling its core infrastructure from mission-specific modules, Cavern Manticore’s operators gain both operational agility and durability under defensive pressure. This **modularity** allows them to adjust capabilities per campaign while preserving the underlying framework. For defenders, the key takeaway is clear: detection strategies must move beyond static IOCs and focus on malware behavior patterns, infrastructure, and abuse of trusted administrative channels.

Protections

Check Point Threat Emulation and **Harmony Endpoint** provide comprehensive coverage of this attack and protect against threats described in this report.

Security Recommendation

Conduct a focused review of logs, process execution events, and file activity involving `uxtheme.dll`, as this DLL is known to be abused in DLL sideloading attack chains. Security teams should also examine the `C:\ProgramData` directory for unusual DLL placement, recently created folders, unsigned binaries, or execution patterns that may indicate attempted or successful DLL sideloading.

IOCs

Hashes

SHA-256	Component
37e123bd7998af4eae32718ce254776f36365a80ba56952593dab46f536d4066	<code>uxtheme.dll</code> (Cavern Agent, build 02)
92cae0ad7f98f51a14bcc0ee05e372ebdc29ea96ea7bd161bd3f55198767603b	<code>uxtheme.dll</code> (Cavern Agent, build 04)
5dc08bda6919a57a85e5f38b857985fa71529ca39c8299868d5a49a987e19b18	<code>uxtheme.dll</code> (Cavern Agent, oldest)
a4aa217def4c38f4ecacdf47b1cd687f60cc74c18ab75195be3c4357a790bf41	<code>n-HTComp.dll</code> (communication module)
b630c96d3763182533d4fb9b614134382bd644cb02c6c1c3ade848b6ecc31e86	<code>n-HTComp.dll</code> (communication module)
8e9425c0b46eeb516610ae913d13f2b3f44a023043cb099277031d4ec38a6134	<code>mhm.dll</code> (file manager module)
0a3663648a46771a5a5423ad01e91a4e7ba825595e99fa934cb35cbb4848adc8	<code>mhm.dll</code> (file manager module, older “Cav3rn” variant)
5394d3b220de4695f731647e3a70545f951a8912ceb0c6585efab8d6842e8b42	<code>db.dll</code> (SQL database browser module)
30cb4679c4b8599eeb3d63a551716475c6332bdc4d4b4e3de0964aadb3092a10	<code>ode.dll</code> (LDAP / Active Directory module)
2cb1ad3b22db8e3666ea138fee88034a87a87cf43db3d3265a675ebf221379b0	<code>n-ten.dll</code> (network reconnaissance module)
7d586fb7f94182a8e2a0e53c7e4deb898066da029da5cd9972a94a59ca6d255a	<code>n-sws.dll</code> (SOCKS5 / WebSocket tunnel module)

SHA-256	Component
541b1f417b9e42078c3355693a8a492b6a76048850f6549a429e0be99e6819cb	Older <code>Cav3rn</code> agent (earlier non-modular build)
cbc9485db715e1b8cc384fe94b4cceadca4006cda8a5e28adc8848529cfafc93	Older <code>Cav3rn</code> agent (earlier non-modular build)
ccf218189c3aadb1c761da14bfda3bae686769031e1e1b10007648bd72e34748	Older <code>Cav3rn</code> HTTP module (<code>CAV3RN_Http_Module</code>)

Network

Indicator	Type
hospitalinstallation[.]com	Parent domain
auth[.]hospitalinstallation[.]com	C2 (older agent)
google[.]com[.]hospitalinstallation[.]com	C2 (newer agents)
adserviceupdate[.]com	C2 domain invoked by the older <code>Cav3rn</code> HTTP module at <code>https://adserviceupdate[.]com/cac.aspx</code> ; part of the older <code>Cav3rn</code> agent config
hygienehistory[.]com	C2 domain invoked by the older <code>Cav3rn</code> HTTP module at <code>https://hygienehistory[.]com/cac.aspx</code> ; part of the older <code>Cav3rn</code> agent config

Host Artifacts

Indicator	Context
MYMUTEX123HELLP / MYMUTEX123HELLP02 / MYMUTEX123HELLP04	Mutex names
config.txt with keys <code>i</code> , <code>xd</code> , <code>int</code>	Agent configuration
Cvn.cfg.A / Cvn.cfg.U	Legacy alive-time config
C:\Users\rick\Desktop\Modules\cavern\	PDB path prefix
cac.aspx	Operator-deployed ASP.NET handler at <code>https://<adserviceupdate[.]com hygienehistory[.]com>/cac</code> Carried as configuration by the older <code>Cav3rn</code> agent and the older <code>mhm.dll</code> variant (defined but not invoked by either), and invoked by the older <code>Cav3rn</code> HTTP module.
inpt / outpt working directories	Command / result drop dirs for the older <code>Cav3rn</code> agent
.CvnC.png / .CvnA.png / .CvnR.png (JPEG-magic prefixed)	PNG-styled steganographic command / default-API / result files used by the older <code>Cav3rn</code> agent

Source: <https://research.checkpoint.com/2026/cavern-manticore-exposing-iran-linked-modular-c2-framework/>