Frankenstein Variant of the ToneShell Backdoor Targeting Myanmar

⋮ 9/10/2025

ToneShell is a lightweight backdoor tied to the China-nexus group Mustang Panda. Typically delivered via DLL sideloading inside compressed archives with legitimate signed executables and often spread through cloud-hosted lures. Zscaler's 2025 analysis described updates to its FakeTLS C2 (shifting from TLS 1.2- to 1.3-style headers), use of GUID-based host IDs, a rolling-XOR scheme, and a minimal command set for file staging and interactive shell access. Notably, some of this activity was observed in Myanmar, a region of strategic importance to China. Targeting Myanmar is particularly interesting as it reflects China's broader geopolitical interests, spanning border security, infrastructure projects, and political developments, and highlights how cyber operations are leveraged to maintain influence in neighboring states.

This blog is a technical analysis of another variant of the backdoor. While this variant does not introduce major new features, it is worth highlighting the anti-analysis techniques it employs and the new indicators that can support threat hunting and detection. In addition, the continued targeting of Myanmar underscores China's sustained use of cyber operations to maintain its interests abroad.

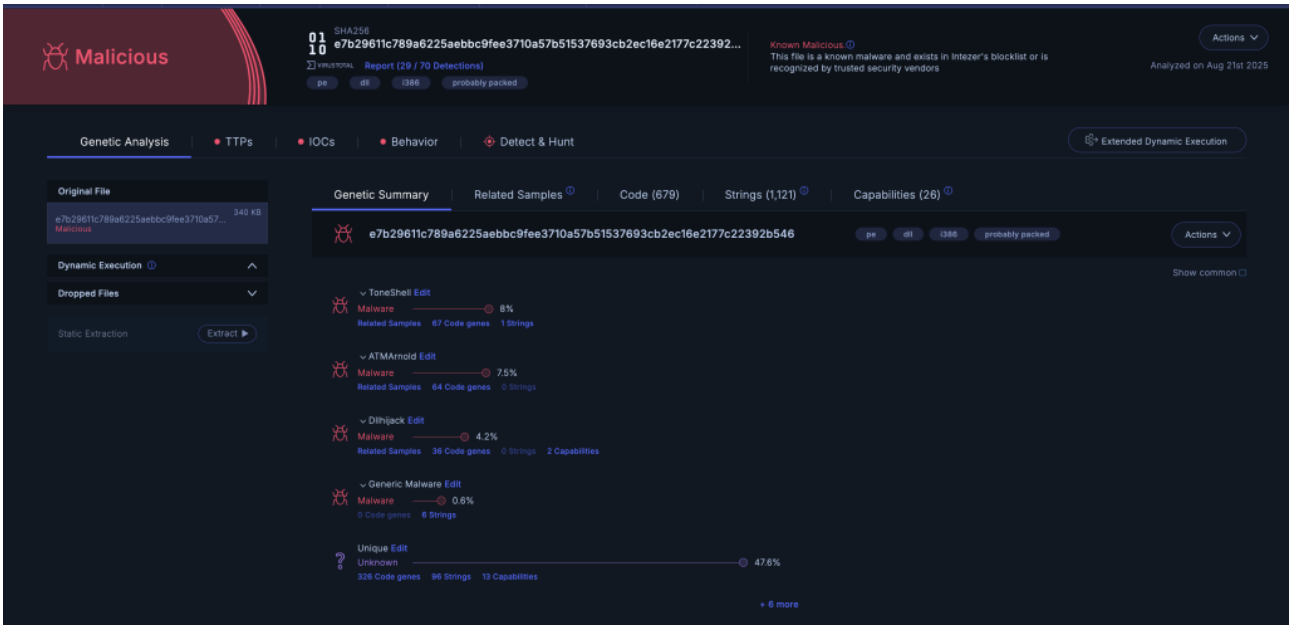
Technical Analysis

As previously documented, ToneShell is a dll that is sideloded by a legitimate executable, usually bundled in the archive file. In this case it is a ZIP file ,named: TNLA □□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□. Which translate to: TNLA and other revolutionary forces.

```
sha256: 1272a0853651069ed4dc505007e8525f99e1454f9e033bcc2e58d60fdafa4f0
```

The backdoor is a DLL named: SkinH.dll, compiled on 2025-07-14 22:38:08.

```
sha256:e7b29611c789a6225aebbc9fee3710a57b51537693cb2ec16e2177c22392b546
```



Code reuse with ToneShell

The backdoor first checks the current module path for the string **Google\DriveFS\Scratch**. If not found, it enumerates processes to identify the parent process, retrieves its image path, and verifies whether it contains **GoogleDrive**. If found, the backdoor will terminate its execution. This check might be an attempt by the threat actor to prevent infecting themselves. If this validation fails, the code enforces a single-instance policy using a named mutex, **Global\SingleCorporation12AD8B**. It attempts to create the mutex, storing the handle globally and setting a flag to indicate success or failure. If creation fails or the mutex already exists, it releases any handles and marks it as not created.
Next, the backdoor checks its persistence location. It compares its module path with the user profile directory (CSIDL_APPDATA, 0x1A). If the binary is not under the user profile, it copies itself and the following files: msvcr100.dll, msvcp100.dll, mfc100.dll to a new folder it creates. The new directory name consists of a 6-character random uppercase folder name (e.g., **C:\Users\<user>\AppData\<random-6-chars>**).

```
10006235
10006255        for (int32_t i = 0; i < 6; i += 1)
1000624d            // Generate random uppercase directory name.
10006247            random_dir_name[i] = (int16_t)((int64_t)_rand() % 0x1a) + 0x41;
10006247
10006261        random_dir_name[6] = 0;
10006265        // Copy random directory name to global buffer.
10006265        _wcscpy_s(&g_AppDataFolderName, 7, random_dir_name);
1000626f        PWSTR random_dir_name_2 = _malloc(0x208);
10006279        wchar16* random_dir_name_1 = random_dir_name;
10006279
1000627c        // Format full path for new directory.
1000627c        if (random_dir_name_2)
1000627c        {
10006291            wsprintfW(random_dir_name_2, %s\%s", &appdata_path, random_dir_name_1);
10006294            _free(random_dir_name);
10006294
100062a7            // Create directory and set as hidden if successful.
100062a7            if (CreateDirectoryW(random_dir_name_2, nullptr))
100062a7            {
100062c8                SetFileAttributesW(random_dir_name_2, FILE_ATTRIBUTE_HIDDEN);
100062d3                PWSTR eax_8 = _malloc(0x208);
100062d3
100062df                if (eax_8)
100062df                {
100062e9                    // Get current executable's directory.
100062e9                    GetModuleFileNameW(nullptr, eax_8, 0x104);
100062f0                    PathRemoveFileSpecW(eax_8);
100062f6                    wchar16 const* const i_1 = u"SkinH.dll";
100062fb                    wchar16 const* const dll_names[0x4];
100062fb                    dll_names[0] = u"SkinH.dll";
10006305                    dll_names[1] = u"msvcr100.dll";
1000630f                    dll_names[2] = u"msvcp100.dll";
10006319                    dll_names[3] = u"mfc100.dll";
10006323                    int32_t var_634 = 0;
1000632d                    void* result = 1;
10006337                    int32_t var_628 = 0;
10006341                    wchar16 const* const i_2 = u"SkinH.dll";
10006341
100063ab                    do
100063ab                    {
10006355                        // Copy DLLs from original directory to
10006355                        // new directory.
10006355                        void var_418;
10006355                        wsprintfW(&var_418, u"%s\%s", eax_8, i_1);
1000636a                        wsprintfW(&appdata_path, u"%s\%s", random_dir_name_2, i_2);
```

The function that copies the DLLs to a new folder in AppData

Once installed, it attempts to spin up the Task Scheduler COM service, connect to the local scheduler, open the root folder (“\”) and obtain an IRegisteredTask for a task named **dokanctl**. If the task doesn’t exist, it will create it using RegisterTaskDefinition. The task is set to execute every minute, it sets the execution path to the folder in AppData created earlier: **%APPDATA%\<random-6-chars>\svchosts.exe**, and sets it as the action Path. The task is registered in the root folder with the name **dokanctl**.

```
10006af7        if (ITaskServiceVtbl->NewTask(ppFolder, flags, ppDefinition)
10006af7            >= 0)
10006af7        {
10006b37            struct ITaskService* eax_14 = taskDef;
10006b43            struct ITaskService* eax_1 = nullptr;
10006b4d            BSTR path = &eax_1;
10006b4e            struct ITaskService* eax = eax_14;
10006b4e            struct ITaskService** esp_6 = &eax;
10006b4e
10006b56            if (eax_14->vtable->GetFolder(eax, path, ppFolder) >= 0)
10006b56            {
10006b63                struct BstrWrapper* rootPathWrap_7 =
10006b63                    mw_BSTR_create(0xc);
10006b65                esp_3 = &eax;
10006b68                rootPathWrap_1 = rootPathWrap_7;
10006b6e                int32_t var_14_7 = 6;
10006b6e
10006b77                if (!rootPathWrap_7)
10006ba8                    rootPathWrap_7 = nullptr;
10006b77                else
10006b77                {
10006b7c                    rootPathWrap_7->str = 0;
10006b7c                    rootPathWrap_7->aux = 0;
10006b80                    rootPathWrap_7->refs = 0;
10006b87                    wchar16 const* const psz = u"dokanctl";
10006b8c                    rootPathWrap_7->aux = nullptr;
10006b93                    rootPathWrap_7->refs = 1;
10006b9a                    BSTR eax_18 = SysAllocString(psz);
10006b9a                    esp_3 = &eax;
10006b9c                    rootPathWrap_7->str = eax_18;
10006b9c
10006ba0                    if (!eax_18)
10006ba0                        goto label_10007234;
10006b77                }
```

Task creation

If the backdoor is already under the user profile, it performs junk arithmetic before entering its C2 loop, followed by a bounded timing/zero-check loop that appears to serve as anti-analysis noise without affecting the main flow.

API Function Hashing

The malware uses a custom API-resolving scheme based on a simple rolling hash, applying it to module and function names instead of storing them in cleartext. This technique, previously analyzed in previous blogs, is a common evasion method seen in different malware types. With the HashDB project, resolving these hashes back to the original imported functions is straightforward, making analysis and detection easier.

```
if (ecx)
{
    while (true)
    {
        void* esi_2 = *(uint32_t*)((char*)eax_3 + (edi << 2)) + moduleBase;

        if (edx_1 > (char*)esi_2 + 0xdc)
        {
            void* edx_2 = esi_2;
            int32_t eax_5 = 0;
            (uint8_t)ecx = *(uint8_t*)edx_2;

            while ((uint8_t)ecx)
            {
                edx_2 += 1;
                eax_5 = eax_5 * 0xc85e31 + (int32_t)(uint8_t)ecx;
                (uint8_t)ecx = *(uint8_t*)edx_2;
            }

            if (eax_5 == hash)
                return pGetProcAddress(moduleBase, esi_2);

            ecx = var_c;
            edx_1 = var_10;
        }

        edi += 1;

        if (edi >= ecx)
            break;

        eax_3 = var_14;
    }
}
```

Hash calculation

```
struct apiHashTable* esi_1 = ppBootstrapHdr;
int32_t dllName;
int32_t* dllName_1 = &dllName;
__builtin_strncpy(&dllName, "ws2_32.dll", 0xb);
void* eax_16 = esi_1->GetProcAddress(dllName_1);
esi_1->ws2_32_base = eax_16;
int32_t var_10;

if (eax_16)
{
    int32_t ws_WSAStartup =
        mw_get_proc_by_hash(eax_16, WSAStartup, eax_16, esi_1->Kernel32Base);
    esi_1->ws_WSAStartup = ws_WSAStartup;

    if (ws_WSAStartup)
    {
        int32_t ws_socket = mw_get_proc_by_hash(ws_WSAStartup, socket,
            esi_1->ws2_32_base, esi_1->Kernel32Base);
        esi_1->ws_socket = ws_socket;

        if (ws_socket)
        {
            int32_t ws_setscockpot = mw_get_proc_by_hash(ws_socket, setsockopt,
                esi_1->ws2_32_base, esi_1->Kernel32Base);
            esi_1->ws_setscockpot = ws_setscockpot;

            if (ws_setscockpot)
            {
                int32_t ws_connect = mw_get_proc_by_hash(ws_setscockpot, connect,
                    esi_1->ws2_32_base, esi_1->Kernel32Base);
                esi_1->ws_connect = ws_connect;
```

Resolved Windows API functions

Anti-Analysis and Anti-Sandboxing

This ToolShell variant employs several stalling and anti-sandboxing tricks designed to waste time, confuse automated analysis, and evade lightweight sandboxes. These include:

1. Repeated file churn: creating, writing, closing, and deleting a temporary file in a loop with Sleep(100) delays, which burns execution time and stresses filesystem emulation.
2. Randomized sleep loops: several loops sleep for pseudo-random intervals ranging from ~800 ms to over a second per iteration, accumulating 20+ seconds of startup delay before meaningful behavior begins.
3. Tick count checks: uses GetTickCount64() combined with jittered sleeps, waiting until at least 10 seconds of wall-clock time has elapsed. This ensures that emulators that don’t advance the system clock realistically can get stuck.
4. Opaque string comparisons: slides across a large embedded wide-string buffer, repeatedly calling lstrcmpW() between overlapping substrings until a break condition. The comparison results are discarded; the loop simply consumes cycles and obfuscates control flow.

5. Nonsense randomization: calculates values like 0xBABE or 0xCAFE via contrived branches using rand(). These values are not security checks, but serve as junk arithmetic to make the code harder to follow.
6. Decoy API use: occasionally calls APIs such as OutputDebugStringA and sets LastError to unusual constants (0xBADF00D), further muddying behavior without altering core logic.

```
// Create, write, and close temp file multiple times with
// delays.
for (int32_t i_1 = 0; i_1 < loopCounter; i_1 += 1)
{
    HANDLE fileHndl = CreateFileA(&temp_filename, 0xc0000000, FILE_SHARE_READ,
        nullptr, 0, FILE_ATTRIBUTE_NORMAL, nullptr);

    if (fileHndl != 0xffffffff || !fileHndl)
    {
        uint32_t numberOfBytesWritten = 0;
        WriteFile(fileHndl, &temp_filename, lstrlenA(&temp_filename),
            &numberOfBytesWritten, nullptr);
        CloseHandle(fileHndl);
    }

    Sleep(0x64);
}

DeleteFileA(&temp_filename);  // Delete temp file after use.
```

File creation loops

Notably, the large embedded string buffers used in the comparison loop contain text copied from OpenAI's blog on image generation and from Pega AI's website. Pega AI refers to the artificial intelligence features integrated into the Pega platform, a software suite for workflow automation and customer relationship management. The backdoor does not use this content functionally; instead, it serves as filler to inflate the binary and supply meaningless strings for comparison.

```
10052110  wchar16 junk_string[0x13b0] = "As an AI-powered platform, decisioning is in our DNA. It"
10052110      "\xe2\x80\x99s small wonder, then, that Pega is the ideal environment for AI to thriv"
10052110      "e \xe2\x80\x93 where decisions and workflows live at the center. Silo-free. Solution"
10052110      "-ready. And the enterprise-grade experts we look to? Well, they all work here. Here"
10052110      "\'s how they think about practical application today \xe2\x80\x93 and how to drive b"
10052110      "usiness-defining outcomes tomorrow\x00From the first cave paintings to modern infogr"
10052110      "aphics, humans have used visual imagery to communicate, persuade, and analyze"
10052110      "\xe2\x80\x94not just to decorate. Today\'s generative models can conjure surreal, br"
10052110      "eathtaking scenes, but struggle with the workhorse imagery people use to share and c"
10052110      "reate information. From logos to diagrams, images can convey precise meaning when au"
10052110      "gmented with symbols that refer to shared language and experience.GPT\xe2\x80\x914o "
10052110      "image generation excels at accurately rendering text, precisely following prompts, a"
10052110      "nd leveraging 4o\xe2\x80\x99s inherent knowledge base and chat context\xe2\x80\x94in"
10052110      "cluding transforming uploaded images or using them as visual inspiration. These capa"
10052110      "bilities make it easier to create exactly the image you envision, helping you commun"
10052110      "icate more effectively through visuals and advancing image generation into a practic"
10052110      "al tool with precision and power.\x00From the first cave paintings to modern infogra"
10052110      "phics, humans have used visual imagery to communicate, persuade, and analyze"
10052110      "\xe2\x80\x94not just to decorate. Today\'s generative models can conjure surreal, br"
10052110      "eathtaking scenes, but struggle with the workhorse imagery people use to share and c"
10052110      "reate information. From logos to diagrams, images can convey precise meaning when au"
10052110      "gmented with symbols that refer to shared language and experience.GPT\xe2\x80\x914o "
10052110      "image generation excels at accurately rendering text, precisely following prompts, a"
10052110      "nd leveraging 4o\xe2\x80\x99s inherent knowledge base and chat context\xe2\x80\x94in"
10052110      "cluding transforming uploaded images or using them as visual inspiration. These capa"
10052110      "bilities make it easier to create exactly the image you envision, helping you commun"
10052110      "icate more effectively through visuals and advancing image generation into a practic"
10052110      "al tool with precision and power.\x00As an AI-powered platform, decisioning is in ou"
10052110      "r DNA. It\xe2\x80\x99s small wonder, then, that Pega is the ideal environment for AI"
10052110      " to thrive \xe2\x80\x93 where decisions and workflows live at the center. Silo-free."
10052110      " Solution-ready. And the enterprise-grade experts we look to? Well, they all work he"
10052110      "re. Here\'s how they think about practical application today \xe2\x80\x93 and how to"
10052110      " drive business-defining outcomes tomorrow\x00As an AI-powered platform, decisioning"
10052110      " is in our DNA. It\xe2\x80\x99s small wonder, then, that Pega is the ideal environme"
10052110      "nt for AI to thrive \xe2\x80\x93 where decisions and workflows live at the center. S"
```

Text taken from PegaAI's website

```
1022ef48  wchar16 const data_1022ef48[0x7a] = "From the fir"
1022ef48      "st cave paintings to modern infographics, hu"
1022ef48      "mans have used visual imagery to communicate"
1022ef48      ", persuade, and analyz"
```

Text taken from OpenAI's website

Together, these techniques do not detect debuggers directly; instead, they stall execution, pollute control flow with junk work, and frustrate automated sandboxing systems that expect short-lived, straightforward malware runs.

GUID Creation

Similar to earlier versions, this variant of ToneShell generates a unique identifier for each infected machine. It first attempts to check if one was already created by reading 16 bytes from C:\ProgramData\SystemRuntimeLag.inc. The malware creates a new seed if the file is missing, unreadable, or does not provide exactly 16 bytes. The primary method is to generate a GUID via CoCreateGuid and use

it as the seed. If that fails, it falls back to producing 16 bytes using an internal linear congruential generator (LCG) seeded from the current PRNG state. Once a new seed is generated, it is written back to the designated file path to ensure persistence across executions.

This logic is similar to Zscelar's blog about the second backdoor variant.

Networking

The malware sets up a socket with a C2 at the address:

```
146.70.29[.]229:443
```

The malware's network protocol wraps its messages in a TLS-like record header to blend in with legitimate traffic. Each packet begins with the fixed bytes 17 03 03 (TLS 1.2 Application Data) followed by a two-byte length field, but only the low byte is honored, effectively capping payloads at 255 bytes. The 5-byte header is read into a temporary stack buffer and discarded; only the payload bytes are stored in the context structure's receive buffer starting at offset 0.

Once the payload is received, the malware XOR-decodes it in place with a 256-byte rolling key. The first decoded byte is treated as a "type/status" field, the second as an additional code, and the remainder (len-2 bytes) as the message body. The decoded buffer pointer is then saved in the structure for later use. This design produces a simple packet format: [TLS-like header][XOR-obfuscated type | code | body…], with the header stripped before the data is available to the rest of the malware.

Interestingly, while the GUID generation logic in this sample aligns with the **first** version of ToneShell described by Zscaler, the communication protocol remains identical to the **second** version. Combined with the anti-analysis techniques and the use of text copied from AI platforms, this sample stands out as a hybrid variant.

Pivoting on The C2 Address

Pivoting on the C2 address, we were also able to find another variant:

```
sha256: a58868b3d50b775de99278eeb14da8b7409b165aa45313c6d9fa35ac30d2cda2
```

This variant was compiled on 2025-03-23 22:42:42, named node.dll. It was part of an archive named: update.zip with the sha256:

```
543024edc9f160cc1cedcffc3de52bfa656daa0ec9ed351331d97faaa67d0d99
```

This variant reuses the same C2 commands and anti-analysis techniques described above, and it stores the GUID under the same file name.

Conclusion

In summary, this analysis of a new ToneShell backdoor variant highlights the evolving tactics of the Mustang Panda group, particularly their sustained targeting of Myanmar. While not introducing revolutionary features, this variant employs anti-analysis techniques, including elaborate stalling mechanisms and obfuscated code. The blend of old and new elements, such as the consistent communication protocol alongside updated GUID generation and the use of seemingly random, copied text for filler, underscores the adaptive nature of this threat. The continuous refinement of these evasion methods, coupled with the geopolitical significance of the targeted region, reinforces the need for ongoing research and threat hunting to counter cyber operations.

IOCs

ToneShell variants:

a58868b3d50b775de99278eeb14da8b7409b165aa45313c6d9fa35ac30d2cda2

e7b29611c789a6225aebbc9fee3710a57b51537693cb2ec16e2177c22392b546

Archive files:

543024edc9f160cc1cedcffc3de52bfa656daa0ec9ed351331d97faaa67d0d99

1272a0853651069ed4dc505007e8525f99e1454f9e033bcc2e58d60fdafa4f0

C2:

146.70.29[.]229:443

Learn more about how Intezer's AI SOC platform can help detect threats like this with our unique combination of battle-tested forensic analysis tools and powerful LLMs

Nicole Fishbein

Nicole is a malware analyst and reverse engineer. Prior to Intezer she was an embedded researcher in the Israel Defense Forces (IDF) Intelligence Corps.