SNOWYAMBER

Malware Analysis Report

13 April 2023

v1.0
Table of Contents

Table of Contents ........................................................................................................................................2
Threat Summary .........................................................................................................................................3
Detailed Technical Analysis ....................................................................................................................4
Delivery ..................................................................................................................................................4
Phishing – Email and Delivery Script .......................................................................................................5
Container File – ZIP ................................................................................................................................7
Container file – ISO ................................................................................................................................7
SNOWYAMBER Analysis .........................................................................................................................9
Obfuscation ...............................................................................................................................................9
Code Flow ...............................................................................................................................................11
Communication with NOTION ..............................................................................................................14
Payloads .................................................................................................................................................16
YARA Rule ..............................................................................................................................................19
Appendix A – IOCs ..................................................................................................................................20
File IoCs ..................................................................................................................................................20
Network IoCs ..........................................................................................................................................21
Appendix B – MITRE ATT&CK ..............................................................................................................22
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 2\(^{nd}\) 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.

---

\(^2\) https://www.mandiant.com/resources/blog/tracking-apt29-phishing-campaigns
\(^4\) 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 script⁶ 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:

![Flowchart](image)

*Figure 1 - SNOWYAMBER delivery chain*

---

Military Counterintelligence Service
CERT.PL

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:

Figure 2 - Example of a phishing email mimicking diplomatic correspondence. The link hidden under “here” leads to the ENVYSCOUT

Figure 3 - ENVYSCOUT banner mimicking correspondence from EU
Figure 4 - ENVYSCOUT banner mimicking diplomatic correspondence

Figure 5 - ENVYSCOUT landing page with a more generic banner

Figure 6 - 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_scheduleexe.pdf
└── vcruntime140.dll
```

Filename `november_scheduleexe.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 7 - SNOWYAMBER loading chain](image)

**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 8 - Content of the LNK file used to start SNOWYAMBER
Military Counterintelligence Service  
CERT.PL

**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.

```c++
#define AY_OBFUSCATE(data) AY_OBFUSCATE_KEY(data, AY_OBFUSCATE_DEFAULT_KEY)

#define AY_OBFUSCATE_KEY(data, key)
[](size_t n) -> ay::obfuscated_data<n, 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++

Figure 9 - Reconstructed decryption routine

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

```c++
void __fastcall ay::obfuscated_data<6ull, 2814183348159999672ull>:::decrypt(Enc_NtOll *data) {
    if (data->m_encrypted ) {
        ay::cipher(data, 6164, 0x2468A525676FA7164);
        data->m_encrypted = 0;
    }
}
```

---

**Figure 9 - Reconstructed decryption routine**

Compare the reconstructed code above with the original source code below:
Due to Obfuscate library implementation, the same key is used to encrypt all strings making decryption easy:

![Obfuscated string code](image1)

Figure 10 - 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:

```cpp
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.

```cpp
hModule = LoadLibraryA(szszAdvapi32);
pProc = GetProcAddress(hModule, szRegOpenKeyExA);
```

Figure 11 - 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\(^1\) 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:

   ```plaintext
   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 (NtProtectVirtualMemory). Those are implemented using direct syscalls. Stubs for syscalls were generated using the SysWhispers3 project:

![Figure 12 - Example of SysWhispers3 generated stubs for direct syscalls used in samples from FEB 2023](image)

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

![Figure 13 - Decryption routine from OCT22 sample. After payload decryption, memory permissions are modified to RX using VirtualProtect.](image)
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\textsuperscript{11} \textsuperscript{12} \textsuperscript{13}.

![Shellcode execution codeblock](https://ntquery.wordpress.com/2014/03/29/anti-debug-fiber-local-storage-fls/)

\textsuperscript{11} http://dronesec.pw/blog/2019/08/12/code-execution-via-fiber-local-storage/

\textsuperscript{12} https://github.com/aahmad097/AlternativeShellcodeExec

\textsuperscript{13} https://ntquery.wordpress.com/2014/03/29/anti-debug-fiber-local-storage-fls/
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 \(https://api.notion.com/v1/databases/<\text{DB}_\text{Id}>/query\) containing the following JSON\(^{15}\):

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

   Where \(<\text{DB}_\text{Id}>\) is the hardcoded database identifier and \(<\text{victimID}>\) 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:

   ```json
   {
     "parent": {
       "database_id": "<\text{DB}_\text{id}>",
       "type": "database_id"
     },
     "properties": {
       "Info": {
         "rich_text": [
           {
             "text": {
               "content": "<\text{EncFingerprint}>"
             },
             "type": "text"
           }
         ]
       },
       "Name": {
         "title": [
           {
         }
       }
   }
   ```

\(^{14}\) https://github.com/nlohmann/json

\(^{15}\) https://developers.notion.com/reference/post-database-query
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.

```json
{
"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:

```json
{
"File": {
"id": "tCO~",
"type": "files",
"files": [
{
"name": "hXaIk2508.pdf",
"type": "file",
"file": {
"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:

```
<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>37000</td>
</tr>
<tr>
<td>MaxGetSize</td>
<td>1048576</td>
</tr>
<tr>
<td>Jitter</td>
<td>33</td>
</tr>
<tr>
<td>MaxDNS</td>
<td>Not Found</td>
</tr>
<tr>
<td>PublicKey_MD5</td>
<td>92b602d1008704ef300d29ca014c6e21</td>
</tr>
<tr>
<td>C2Server</td>
<td>humanecosmetics.com,/category/noteworthy/6426-7346-9789</td>
</tr>
<tr>
<td>UserAgent</td>
<td>Mozilla/5.0 (Windows NT 10.0; Win64; x64)</td>
</tr>
<tr>
<td>AppleWebKit/537.36 (KHTML, like Gecko)</td>
<td>Chrome/89.0.4389.133 Safari/537.36</td>
</tr>
<tr>
<td>HttpPostUri</td>
<td>/category/noteworthy/8264-1537-9826</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>Metadata</td>
</tr>
<tr>
<td></td>
<td>base64</td>
</tr>
<tr>
<td></td>
<td>prepend &quot;X-ID-CONTENT=&quot;</td>
</tr>
<tr>
<td></td>
<td>header &quot;Cookie&quot;</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>prepend &quot;X-ID-CONTENT=&quot;</td>
</tr>
<tr>
<td></td>
<td>header &quot;Cookie&quot;</td>
</tr>
<tr>
<td></td>
<td>Output</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\dllhost.exe</td>
</tr>
<tr>
<td>Spawnto_x64</td>
<td>%windir%\sysnative\dllhost.exe</td>
</tr>
<tr>
<td>CryptoScheme</td>
<td>0</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>
</tbody>
</table>
```

16 Thanks to the amazing work of SentinelOne (https://github.com/Sentinel-One/CobaltStrikeParser)
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
0002F570: 33 23 4D 3F-3A 58 79 4D-42 46 00 00-5B 00 28 00
0002F580: 5D 00 20 00-44 00 6F 00-74 00 6E 00-65 00 74 00
0002F590: 3A 00 20 00-76 00 00 00-25 00 6C 00-73 00 25 00
0002F5A0: 6C 00 75 00-0A 00 5B 00-2B 00 5D 00-20 00 43 00
0002F5B0: 4C 00 52 00-3A 00 20 00-25 00 6C 00-73 00 0A 00
0002F5C0: 00 00 5B 00-2D 00 5D 00-20 00 45 00-3A 00 20 00
```

```
0319 bYXJM/
```

```
3#M?:XyMBF [ +]
```

```
Dot net
```

```
; v % ls %
```

```
l u x [ + ] C
```

```
LR: % ls %
```

```
[-] E:
```
According to one of the patch notes\footnote{https://bruteratel.com/release/2022/07/20/Release-Staffels-Escape/}, 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|||eyJ0eXBlIjogInJlcXVlc3QiLCAiaWQiOiI3MTI1LTgxNzMtOTQ2MS00NTIxLTwiZGF0YSI6IQ==|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|Hfi1836b3linfgsifs6e365425|HHmfo34i836bVFbkn28ny2fowyfsgyf|/api/2/user/1623-2441-6632-3243|/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
```

\footnote{https://bruteratel.com/release/2022/07/20/Release-Staffels-Escape/}
YARA Rule

A rule that can be used to scan for SNOWYAMBER:

```yara
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 4B 89 C8 31 D2 4C 8B 4C 24 ?? 4B F7 74 24 ?? 49 8B 45 00 41 8A 14 11 32 54 0B 10 89 C8 41 0F AF C0 31 C2 88 14 0B 4B FF C1}
        // Decryption routine generated by Obfuscate library
        $op_decrypt_string = {48 39 00 74 19 4B 89 C1 4D 89 C2 B3 E1 07 48 C1 E1 03 49 D3 EA 45 30 14 01 4B FF C0 6B E2}
        // Hardcoded initial value used as beaconing counter
        // 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

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample dated 24/10/2022</td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>7za.dll</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>270,336B</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>d0efe94195eb4923eb644ec0b53d226cc</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>c938934c0f5304541087313332aee153e0c5239c</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>381a3c6c7e1f9f58dfde6f03a9890353a20badf01bfa7c38ede62c6b0692103c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample dated 8/02/2023</td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>BugSplatRc64.dll</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>271,360B</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>cf36bf564fbb7d5ec4cec9b0f185f6c9</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>8eb64670c10505322d45f6114bc9f7de0826e3a1</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>e957326b2167fa7ccd508cbf531779a28bfc75eb2635ab81826a522979aeb98</td>
</tr>
<tr>
<td><strong>Additional remarks</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample dated 7/02/2023</td>
<td></td>
</tr>
<tr>
<td><strong>File Name</strong></td>
<td>BugSplatRc64.dll</td>
</tr>
<tr>
<td><strong>File Size</strong></td>
<td>301,056B</td>
</tr>
<tr>
<td><strong>MD5</strong></td>
<td>82ecb8474efe5fedcb8f57b8aaf93d2</td>
</tr>
<tr>
<td><strong>SHA1</strong></td>
<td>3fd43de3c9f7609c52da7c1fc4d01ec0b5a9c74c</td>
</tr>
<tr>
<td><strong>SHA256</strong></td>
<td>4d92a4cecb62d237647a20d2cddf944d5a29c1a14b274d729e9c8ccca1f0b68b</td>
</tr>
</tbody>
</table>
**2nd stage - CobaltStrike beacon (decrypted)**

- **File Name**: hXaIk1725.pdf
- **File Size**: 261,635B
- **MD5**: 800db035f9b6f1e86a7f446a8a8e3947
- **SHA1**: aaf973a56b17a0a82c1b3a49ff68da1c50283d4
- **SHA256**: 032855b043108967a6c2de154624c16b70a0b7d0d0a0e93064b387f59537cc1e

**2nd stage - BruteRatel stageless badger (decrypted)**

- **File Name**: hXaIk1314.pdf
- **File Size**: 347,837
- **MD5**: 0e594576bb36b025e80eab7c35dc885e
- **SHA1**: a8a82a7d2d979b28cbcedef4e70f9d5725ef66b
- **SHA256**: ec687a447ca036b10c28c1f9e1e9ceef9f2078fd2bc2ffdb4d8dd32e834b310c0d

**Network IoCs**

<table>
<thead>
<tr>
<th>URL</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalmassasje[]no/schedule.php</td>
<td>ENVYSCOUT delivering SNOWYAMBER ZIP</td>
</tr>
<tr>
<td>signitivelogics[]com/Schedule.html</td>
<td>ENVYSCOUT delivering SNOWYAMBER ISO</td>
</tr>
<tr>
<td>humanecosmetics[]com/category/noteworthy/6426-7346-9789</td>
<td>Cobalt Strike Team Server</td>
</tr>
<tr>
<td>signitivelogics[]com/BMW.html</td>
<td>ENVYSCOUT delivering SNOWYAMBER ISO</td>
</tr>
<tr>
<td>badriatiminii[]com</td>
<td>BRUTERATEL C2</td>
</tr>
<tr>
<td>literaturaelsalvador[]com/Instructions.html</td>
<td>ENVYSCOUT delivering SNOWYAMBER ZIP</td>
</tr>
<tr>
<td>literaturaelsalvador[]com/Schedule.html</td>
<td>ENVYSCOUT delivering SNOWYAMBER ISO</td>
</tr>
<tr>
<td>parquesanrafael[]hc/.note.html</td>
<td>ENVYSCOUT URL</td>
</tr>
<tr>
<td>inovaaoftalmologia.com[]br/form.html</td>
<td>ENVYSCOUT URL</td>
</tr>
</tbody>
</table>

**Emails**

<table>
<thead>
<tr>
<th>Email</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:miodrag.sekulic@mod.gov">miodrag.sekulic@mod.gov</a>[]rs</td>
<td>Used to distribute phishing emails with a link to ENVYSCOUT</td>
</tr>
<tr>
<td>bohuslava.kopalova@seznam[]cz</td>
<td>Used to distribute phishing emails with a link to ENVYSCOUT</td>
</tr>
<tr>
<td>Navratilova.Lucie.etnologie@seznam[]cz</td>
<td>Used to distribute phishing emails with a link to i.php (reconnaissance?)</td>
</tr>
<tr>
<td>zdenek.holych@seznam[]cz</td>
<td>Used to distribute phishing emails with a link to ENVYSCOUT</td>
</tr>
</tbody>
</table>
## 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 |