Practical 2: Report TrickBot

3/21/2021
Nicholas Fox
fox.nbrian@ufl.edu
Practical 2, CAP6137

Executive Summary

After analysis of the malware it can be concluded that the malware is some form of BotNet/RAT that connects to a remote command and control (C&C) host. Upon infection of the computer the malware quickly deletes itself and runs behind the scenes. The malware attempts to retrieve the external IP address of the victim by quereying myexternalip.com. If this fails the malware checks other sites for the external IP. After the external IP is established the malware begins to perform reverse DNS queries to identify multiple remote IP addresses, likely used for C&C. The malware communicates with the host server by first encrypting its messages with the ECDSA P348 algorithm and sending them over http. Through the analysis and research it was found that the name of the malware is TrickBot.

Malware MD5 Hash: E05D85ACC62B2795BFB94A681E64E20F
Unpacked MD5 Hash: D86F7FE3B79CACC664B4B4F50BD61273

Static Analysis

(a) Identify the apparent compilation date of the program.

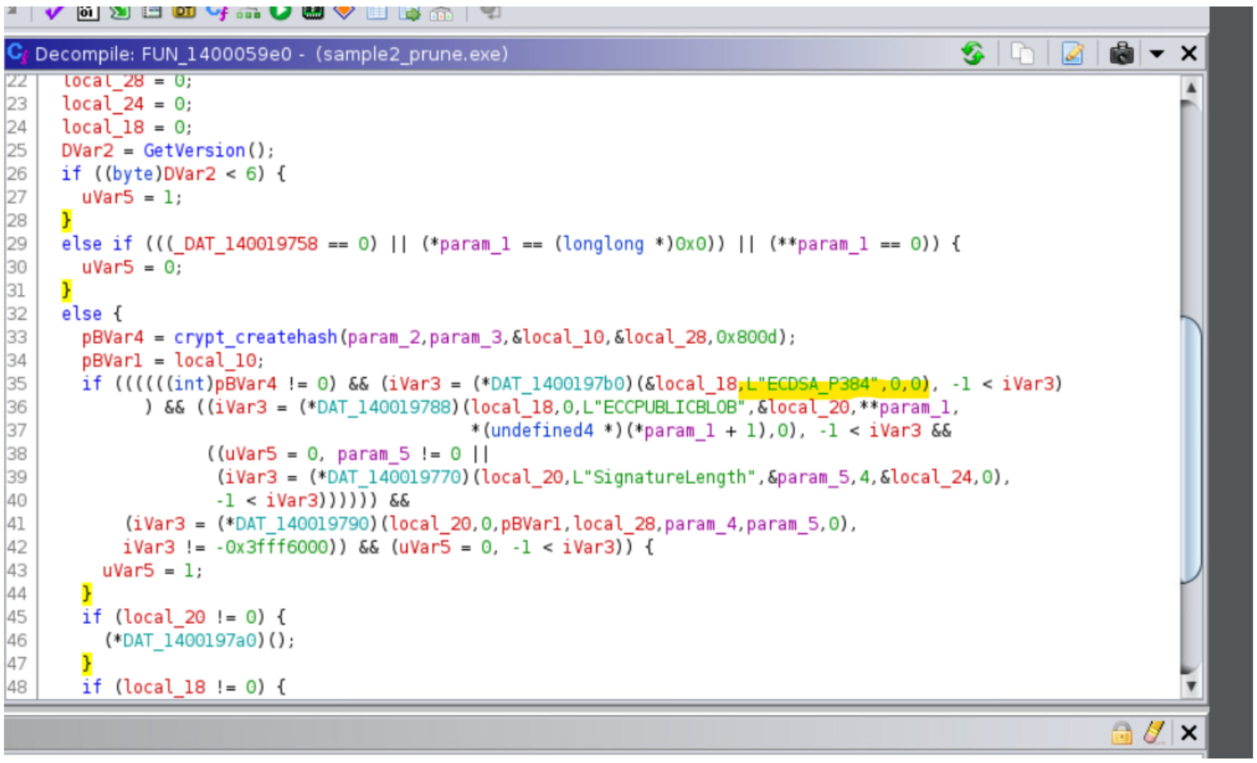
The compilation date of the program appears to be Friday August 19, 13:54:32 2016. After unpacking the code and viewing the compilation date it is noted that the compilation date is later, namely Thursday November 24, 16:57:31 2016. This is suspicious because the resource should have been compiled before the main sample2.exe program. This suggests that the dates may have been manipulated.

(b) Identify any suspicious properties of the program's Imports.

The unpacked file of the program does not have many imports. It mainly includes functions from the KERNEL32 library such as LoadResource, SizeOfResource, LoadLibraryA, and others. The main purpose of most of these imports is to load the resource section and other libraries. Because most of the code is hidden in the resources of the malware the extracted unpacked malware contains some interesting imports. Some of these imports include cryptography functions such as CryptDecrypt, CryptBinaryToStringW, CryptDestroyKey, and others. These imports suggest that the malware does some sort of encrypting of local files. The malware also includes functions from KERNEL32 to create new processes that could allow the malware to exit its main process and function behind the scenes if the program is terminated. Some of these functions include CreateProcessW, TerminateProcess, Create thread and more. The malware also imports ReadFile and WriteFile along with other functions that handle directories and files, suggesting that the malware may edit some of the files on the victim's system. Finally, the malware uses functions from the WINHTTP library such as WinHttpOpenRequest, WinHttpConnect, WinHttpSendRequest, and more. All of these functions suggest that the malware is at least reporting back to a server or acting as an access point for C&C.

(c) Identify any suspicious or relevant strings (IPaddresses, urls, process names, file names, etc.).

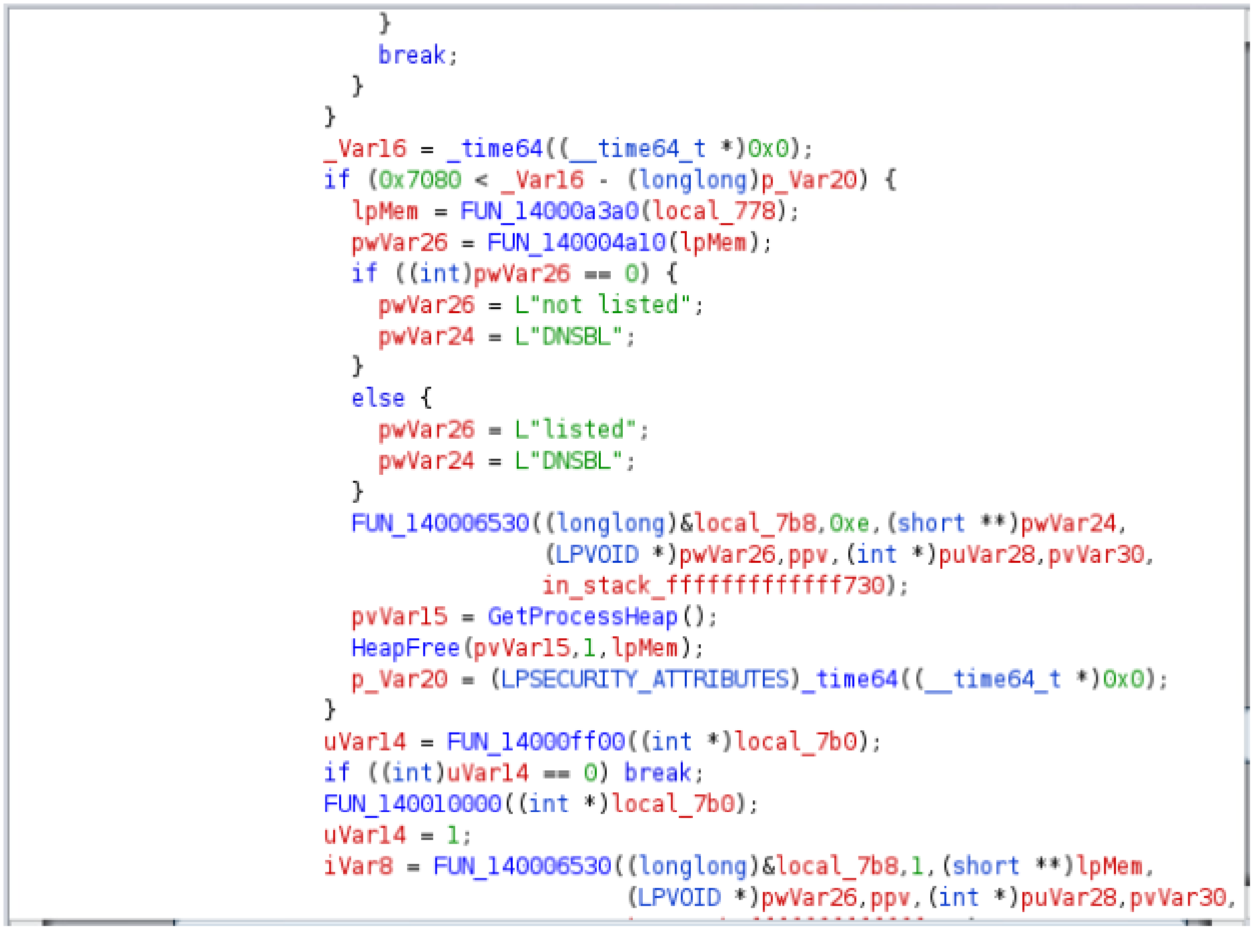
There are many suspicious URLs in the string section of the pe of the unpacked code. Some of these include wtfismyip.com, spam.dsnbl.sorbs.net, myexternalip.com, icanhazip.com, ip.anysrc.net, dsnbl-1.uceprotect.net, cbl.abuseat.org, api.ipify.org, b.barracudacenteral.org. Most of these URLs seem to get the IP of the victim. Along with the URLs there are many strings that refer to WinHttp functions. There are some other interesting strings that deal with cryptography. BCryptImportKeyPair suggests that the malware uses some form of public key cryptography. Other blacklisted strings include function names such as DeleteFile and FindNextFile that suggest the malware alters or even deletes some files. Some other suspicious strings are IDR x86BOT, IDR x64BOT, and IDR X64LOADER. These are names for portions of the .rsrc and are likely the names of programs for specific architectures. This is suggesting that the malware is a botnet and comes equipped with a x64 and x86 version. It is interesting to note the image below (image 1) reveals the type of encryption the malware uses to encrypt its data. It uses ECDSA_P348. This would be useful in further analysis to figure out the contents of the malware’s communication. Further static analysis also revealed that the malware checks if a domain name has been blacklisted by the DNS. This is likely how the malware selects which website to query when attempting to figure out the external IP of the victim (image 2).



Decompile: FUN_1400059e0 - (sample2_prune.exe)

```
  local_28 = 0;
  local_24 = 0;
  local_18 = 0;
  DVar2 = GetVersion();
  if ((byte)DVar2 < 6) {
    uVar5 = 1;
  }
  else if (((_DAT_140019758 == 0) || (*param_1 == (longlong *)0x0)) || (**param_1 == 0)) {
    uVar5 = 0;
  }
  else {
    pBVar4 = crypt_createhash(param_2,param_3,&local_10,&local_28,0x800d);
    pBVar1 = local_10;
    if ((((((int)pBVar4 != 0) && (iVar3 = (*DAT_1400197b0)(&local_18,L"ECDSA_P384",0,0), -1 < iVar3)
         ) && ((iVar3 = (*DAT_140019788)(local_18,0,L"ECCPUBLICBLOB",&local_20,**param_1,
                                          *(undefined4 *)(*param_1 + 1),0), -1 < iVar3 &&
               ((uVar5 = 0, param_5 != 0 ||
                (iVar3 = (*DAT_140019770)(local_20,L"SignatureLength",&param_5,4,&local_24,0),
                -1 < iVar3)))))) &&
       (iVar3 = (*DAT_140019790)(local_20,0,pBVar1,local_28,param_4,param_5,0),
       iVar3 != -0x3fff6000)) && (uVar5 = 0, -1 < iVar3)) {
      uVar5 = 1;
    }
    if (local_20 != 0) {
      (*DAT_1400197a0)();
    }
    if (local_18 != 0) {
```

Image 1.



```
      }
      break;
    }
  }
  _Var16 = _time64((__time64_t *)0x0);
  if (0x7080 < _Var16 - (longlong)p_Var20) {
    lpMem = FUN_14000a3a0(local_778);
    pwVar26 = FUN_140004a10(lpMem);
    if ((int)pwVar26 == 0) {
      pwVar26 = L"not listed";
      pwVar24 = L"DNSBL";
    }
    else {
      pwVar26 = L"listed";
      pwVar24 = L"DNSBL";
    }
    FUN_140006530((longlong)&local_7b8,0xe,(short **)pwVar24,
                  (LPVOID *)pwVar26,ppv,(int *)puVar28,pvVar30,
                  in_stack_ffffffffffff730);
    pvVar15 = GetProcessHeap();
    HeapFree(pvVar15,1,lpMem);
    p_Var20 = (LPSECURITY_ATTRIBUTES)_time64((__time64_t *)0x0);
  }
  uVar14 = FUN_14000ff00((int *)local_7b0);
  if ((int)uVar14 == 0) break;
  FUN_140010000((int *)local_7b0);
  uVar14 = 1;
  iVar8 = FUN_140006530((longlong)&local_7b8,1,(short **)lpMem,
                        (LPVOID *)pwVar26,ppv,(int *)puVar28,pvVar30,
```

Image 2.

(d) Identify the program section(s) and their likely contents.

The original file has four sections. The .text section is the section that contains the loading and unpacking code. This section has code that loads libraries and resources. It also has the necessary code to unpack the remainder of the program. The .rdata section is a read only data section which may contain various strings and other read only items. The .data section as a place where the malware can read and write from. Finally the .rsrc section contains the resources of the malware. It is also important to note that the .rsrc section contains 94.32% of the file and has an entropy of 7.997 suggesting the most of the malware is packed in this section.

(e) Identify any anti-disassembly techniques you discover the program employing.

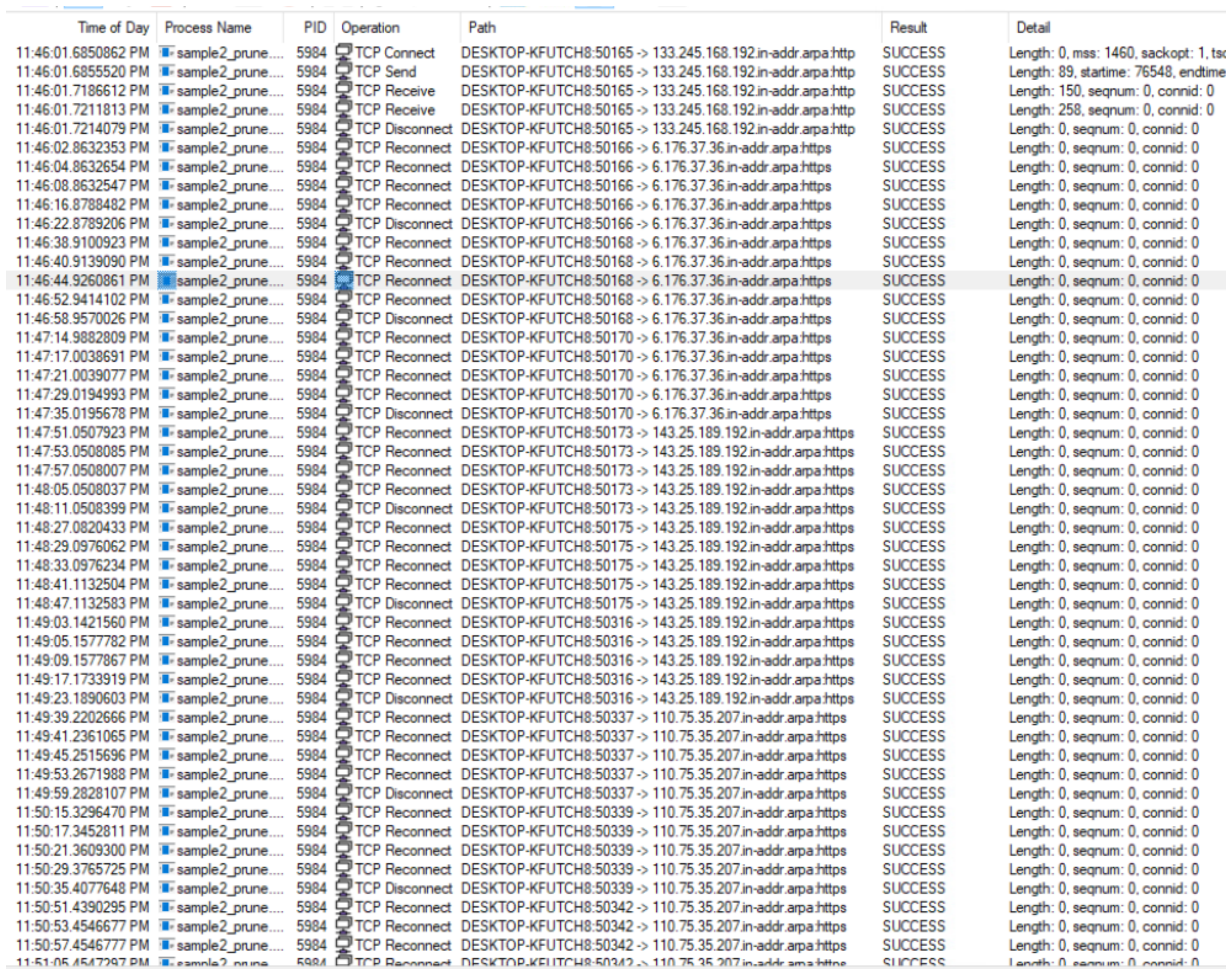
Outside of the program being packed There were no anti-disassembly techniques discovered.

(f) Is the program obfuscated? Use multiple indicators, explaining the significance of each.

Yes the program is obfuscated. The largest indicator of obfuscation is the entropy of the file. The total entropy of the file is 7.971 and the entropy of the .rsrc section, which contains 94% of the file, is 7.997 of 8. The closer the entropy of a file is to 8 the more random the contents of the file are. A normal file would have an entropy of about 6. Another indicator that the program is packed is in the strings of the program there is a string towards the end of the file that reads: “PADDINGXXPADDINGXX…”. This string is suggestive that the program was packed and needed some padding to fill in the blank space. Another indicator of the program being packed is that it only reports using 4 libraries in the PE header, suggesting that only functions needed for loading and unpacking code are included. Most unpacked programs import more libraries because of the need for more functionality. All of these factors contribute to the conclusion that the program is packed.

Dynamic Analysis
(a) Interesting behaviors that occur after the malware has executed.

After the malware has been executed it deletes itself but copies itself to the AppData folder and deletes itself again. While in the AppData folder, the malware makes a Modules directory and two files. The created files are a client_id file and a group_tag. The client id contains the string: DESKTOP-KFUTCHB_W629200.2003545586CB84B29E440898202CD1E1 which is likely an identifier for the machine running the software, the other file contains the string: ser9. The malware also spawns several threads. The threads mostly deal with cryptography. It is hypothesized that the cryptography is used to encrypt data before sending them over http. There is a lot of network activity as well (image 3). It is hypothesized that the malware is a form of a botnet and is attempting to connect for C&C to these IP addresses. Because the VM testing the malware is not connected to the internet the connections do not make it through to the actual servers, although the malware continues to attempt to connect to these servers.



Time of Day	Process Name	PID	Operation	Path	Result	Detail
11:46:01.6850862 PM	sample2_prune....	5984	TCP Connect	DESKTOP-KFUTCH8:50165 -> 133.245.168.192.in-addr.arpa:http	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsc
11:46:01.6855520 PM	sample2_prune....	5984	TCP Send	DESKTOP-KFUTCH8:50165 -> 133.245.168.192.in-addr.arpa:http	SUCCESS	Length: 89, startime: 76548, endtime
11:46:01.7186612 PM	sample2_prune....	5984	TCP Receive	DESKTOP-KFUTCH8:50165 -> 133.245.168.192.in-addr.arpa:http	SUCCESS	Length: 150, seqnum: 0, connid: 0
11:46:01.7211813 PM	sample2_prune....	5984	TCP Receive	DESKTOP-KFUTCH8:50165 -> 133.245.168.192.in-addr.arpa:http	SUCCESS	Length: 258, seqnum: 0, connid: 0
11:46:01.7214079 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50165 -> 133.245.168.192.in-addr.arpa:http	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:02.8632353 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50166 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:04.8632654 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50166 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:08.8632547 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50166 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:16.8788482 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50166 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:22.8789206 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50166 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:38.9100923 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50168 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:40.9139090 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50168 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:44.9260861 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50168 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:52.9414102 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50168 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:46:58.9570026 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50168 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:14.9882809 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50170 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:17.0038691 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50170 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:21.0039077 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50170 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:29.0194993 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50170 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:35.0195678 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50170 -> 6.176.37.36.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:51.0507923 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50173 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:53.0508085 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50173 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:47:57.0508007 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50173 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:05.0508037 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50173 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:11.0508399 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50173 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:27.0820433 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50175 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:29.0976062 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50175 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:33.0976234 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50175 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:41.1132504 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50175 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:48:47.1132583 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50175 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:03.1421560 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50316 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:05.1577782 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50316 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:09.1577867 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50316 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:17.1733919 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50316 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:23.1890603 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50316 -> 143.25.189.192.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:39.2202666 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50337 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:41.2361065 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50337 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:45.2515696 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50337 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:53.2671988 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50337 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:49:59.2828107 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50337 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:15.3296470 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50339 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:17.3452811 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50339 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:21.3609300 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50339 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:29.3765725 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50339 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:35.4077648 PM	sample2_prune....	5984	TCP Disconnect	DESKTOP-KFUTCH8:50339 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:51.4390295 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50342 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:53.4546677 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50342 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:50:57.4546777 PM	sample2_prune....	5984	TCP Reconnect	DESKTOP-KFUTCH8:50342 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0
11:51:05.4547297 PM	sample2_prune	5984	TCP Reconnect	DESKTOP-KFUTCH8:50342 -> 110.75.35.207.in-addr.arpa:https	SUCCESS	Length: 0, seqnum: 0, connid: 0

Image 3.

(b) Machines and services the malware attempts to identify or contact by IPor domain/host name.

The malware attempts to connect to many hosts through IP addresses on an http connection. This is known by setting up a fake dns server on the remnux machine and allowing it to communicate and serve DNS requests sent from the malware being run on the windows 10 machine. The fake DNS has many reverse DNS queries in an attempt to locate some of the IP addresses that the malware was requesting. The malware also made a request to myexternalip.com likely to figure out the external IP of the victim machine. Procmon was used to view the activity on the victim machine while the malware was running. Procmon identified many attempts to communicate over the network as seen in image 3.

(c) Registry Keys created/modified by the malware.

Procmon located many attempts for the malware to view registry keys, but it did not recognize any activity of the malware changing registry keys or values.

(d) Files created/modified by the malware.

The malware creates two files as mentioned earlier. There is a client ID file and a group_tag file. These files are used for identifying this machine.

(e) Processes started by the malware.

After the malware is run it promptly copies then deletes itself and starts running again. The malware also starts a service which allows it to run if it is closed. There are also numerous threads spawned by the malware. These threads are used for making http connections and encrypting data before communicating with an external host.

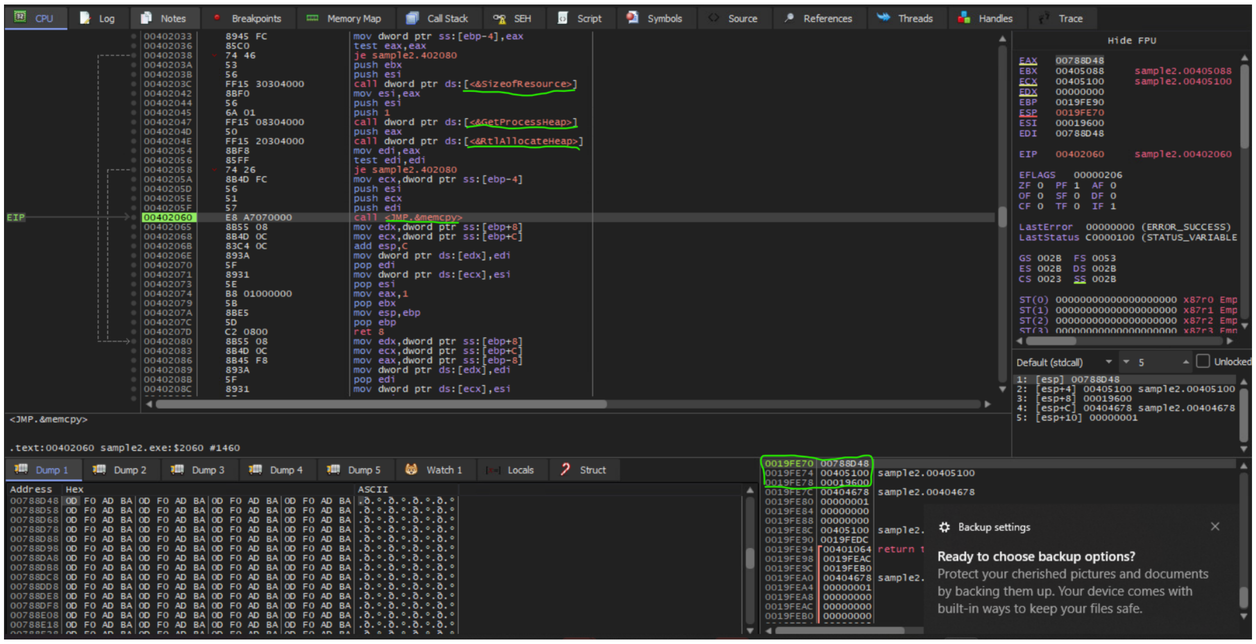
(f) Persistence mechanisms employed by the malware.

The malware has a couple of persistence mechanisms in place. One of them is to delete itself. This makes the malware more stealthy by not leaving a footprint. The malware also has startup privileges. When the computer is rebooted the malware is started as soon as the reboot is complete. The malware also creates a service to allow for persistent running.

(g) Deobfuscation, anti-debugging, and/or anti-VM techniques.

Deobfuscation

Because the malware is packed, it cannot be directly analyzed. According to the PE header file of the original file 95% of the program is stored in the .rsrc section of the program and it has an entropy of 7.99 which means it is packed. To depack it the malware was opened with x32dbg. Because the malware is mostly in the .rsrc section and it is backed there must be a small chunk of code in the .text section to unpack it and run it. First the malware must load the code from the resource section and then unpack it so a breakpoint for LoadResource is set. After the breakpoint is hit and stepped out to the code in the .text section, there are a couple interesting calls to take note of such as the calls to RtlAllocateHeap and SizeOfResource (image 4). This section of code is taking the packed resource allocating space on the heap and copying it to the allocated space. Looking at the parameters for the memcpy are seen on the stack (image 4 lower right) and are noted as the destination, the source and the amount of bytes to copy. The destination is then followed in the dump. Stepping through a couple more instructions leads to a loop with an xor in it. This loop is unpacking the copied code in the heap (image 5). A breakpoint can be set after this loop to unpack all of the code. Next, the section containing the upacked code is dumped to a file. The offset of the section is calculated by taking the heap address of the copy destination and subtracting the section containing the copied code's base address. The length of the unpacked code can be found on the stack when resource size was called. A tool such as dd can be used to carve out the unpacked executable using the offset and the length of the code. Finally we have an unpacked binary that we can further analyze.



```
    00402033  8945 FC          mov dword ptr ss:[ebp-4],eax
    00402036  85C0             test eax,eax
    00402038  74 46            je sample2.402080
    0040203A  53               push ebx
    0040203B  56               push esi
    0040203C  FF15 30304000    call dword ptr ds:[<&SizeofResource>]
    00402042  8BF0             mov esi,eax
    00402044  56               push esi
    00402045  6A 01            push 1
    00402047  FF15 08304000    call dword ptr ds:[<&GetProcessHeap>]
    0040204D  50               push eax
    0040204E  FF15 20304000    call dword ptr ds:[<&RtlAllocateHeap>]
    00402054  8BF8             mov edi,eax
    00402056  85FF             test edi,edi
    00402058  74 26            je sample2.402080
    0040205A  8B4D FC          mov ecx,dword ptr ss:[ebp-4]
    0040205D  56               push esi
    0040205E  51               push ecx
    0040205F  57               push edi
EIP 00402060  E8 A7070000      call <JMP.&memcpy>
    00402065  8B55 08          mov edx,dword ptr ss:[ebp+8]
    00402068  8B4D 0C          mov ecx,dword ptr ss:[ebp+C]
    0040206B  83C4 0C          add esp,C
    0040206E  893A             mov dword ptr ds:[edx],edi
    00402070  5F               pop edi
    00402071  8931             mov dword ptr ds:[ecx],esi
    00402073  5E               pop esi
    00402074  B8 01000000      mov eax,1
    00402079  5B               pop ebx
    0040207A  8BE5             mov esp,ebp
    0040207C  5D               pop ebp
    0040207D  C2 0800          ret 8
    00402080  8B55 08          mov edx,dword ptr ss:[ebp+8]
    00402083  8B4D 0C          mov ecx,dword ptr ss:[ebp+C]
    00402086  8B45 F8          mov eax,dword ptr ss:[ebp-8]
    00402089  893A             mov dword ptr ds:[edx],edi
    0040208B  5F               pop edi
    0040208C  8931             mov dword ptr ds:[ecx],esi

<JMP.&memcpy>

.text:00402060 sample2.exe:$2060 #1460
```

```
EAX     00788D48
EBX     00405088      sample2.00405088
ECX     00405100      sample2.00405100
EDX     00000000
EBP     0019FE90
ESP     0019FE70
ESI     00019600
EDI     00788D48

EIP     00402060      sample2.00402060

EFLAGS    00000206
ZF 0  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000100 (STATUS_VARIABLE

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) 00000000000000000000 x87r0 Emp
ST(1) 00000000000000000000 x87r1 Emp
ST(2) 00000000000000000000 x87r2 Emp
ST(3) 00000000000000000000 x87r3 Emp
```

```
1: [esp] 00788D48
2: [esp+4] 00405100 sample2.00405100
3: [esp+8] 00019600
4: [esp+C] 00404678 sample2.00404678
5: [esp+10] 00000001
```

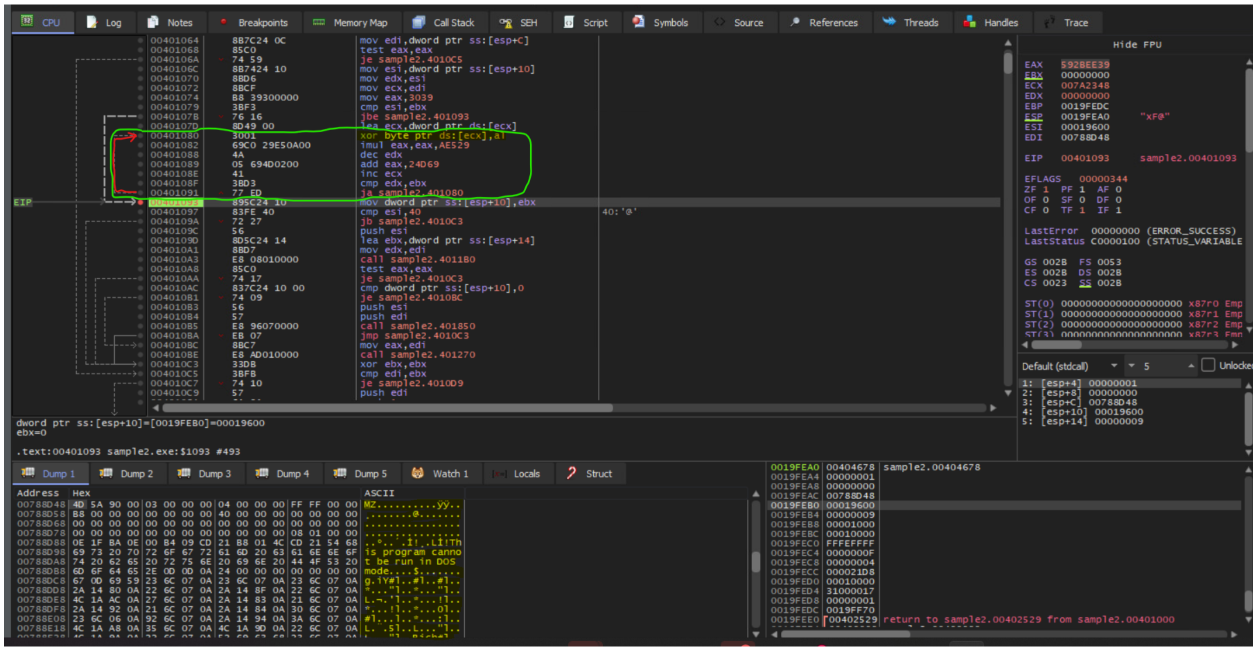
```
Address   Hex                                               ASCII
00788D48  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788D58  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788D68  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788D78  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788D88  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788D98  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DA8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DB8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DC8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DD8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DE8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788DF8  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788E08  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
00788E18  0D F0 AD BA 0D F0 AD BA 0D F0 AD BA 0D F0 AD BA  .ð.º.ð.º.ð.º.ð.º
```

```
0019FE70  00788D48
0019FE74  00405100  sample2.00405100
0019FE78  00019600
0019FE7C  00404678  sample2.00404678
0019FE80  00000001
0019FE84  00000000
0019FE88  00000000
0019FE8C  00405100  sample2.
0019FE90  0019FEDC
0019FE94  00401064  return t
0019FE98  0019FEAC
0019FE9C  0019FEB0
0019FEA0  00404678  sample2.
0019FEA4  00000001
0019FEA8  00000000
0019FEAC  00000000
0019FEB0  00000000
```

Backup settings

Ready to choose backup options?
Protect your cherished pictures and documents by backing them up. Your device comes with built-in ways to keep your files safe.

Image 4.



```
00401064   8B7C24 0C      mov edi,dword ptr ss:[esp+C]
00401068   85C0           test eax,eax
0040106A   74 59          je sample2.4010C5
0040106C   8B7424 10      mov esi,dword ptr ss:[esp+10]
00401070   8BD6           mov edx,esi
00401072   8BCF           mov ecx,edi
00401074   B8 39300000    mov eax,3039
00401079   3BF3           cmp esi,ebx
0040107B   76 16          jbe sample2.401093
0040107D   8D49 00        lea ecx,dword ptr ds:[ecx]
00401080   3001           xor byte ptr ds:[ecx],al
00401082   69C0 29E50A00  imul eax,eax,AE529
00401088   4A             dec edx
00401089   05 694D0200    add eax,24D69
0040108E   41             inc ecx
0040108F   3BD3           cmp edx,ebx
00401091   77 ED          ja sample2.401080
00401093   895C24 10      mov dword ptr ss:[esp+10],ebx
00401097   83FE 40        cmp esi,40                          40:'@'
0040109A   72 27          jb sample2.4010C3
0040109C   56             push esi
0040109D   8D5C24 14      lea ebx,dword ptr ss:[esp+14]
004010A1   8BD7           mov edx,edi
004010A3   E8 08010000    call sample2.4011B0
004010A8   85C0           test eax,eax
004010AA   74 17          je sample2.4010C3
004010AC   837C24 10 00   cmp dword ptr ss:[esp+10],0
004010B1   74 09          je sample2.4010BC
004010B3   56             push esi
004010B4   57             push edi
004010B5   E8 96070000    call sample2.401850
004010BA   EB 07          jmp sample2.4010C3
004010BC   8BC7           mov eax,edi
004010BE   E8 AD010000    call sample2.401270
004010C3   33DB           xor ebx,ebx
004010C5   3BFB           cmp edi,ebx
004010C7   74 10          je sample2.4010D9
004010C9   57             push edi

dword ptr ss:[esp+10]=[0019FEB0]=00019600
ebx=0

.text:00401093 sample2.exe:$1093 #493
```

```
EAX    592BEE39
EBX    00000000
ECX    007A2348
EDX    00000000
EBP    0019FEDC
ESP    0019FEA0     "xF@"
ESI    00019600
EDI    00788D48

EIP    00401093     sample2.00401093

EFLAGS   00000344
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 1  IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus C0000100 (STATUS_VARIABLE

GS 002B  FS 0053
ES 002B  DS 002B
CS 0023  SS 002B

ST(0) 00000000000000000000 x87r0 Emp
ST(1) 00000000000000000000 x87r1 Emp
ST(2) 00000000000000000000 x87r2 Emp
ST(3) 00000000000000000000 x87r3 Emp

1: [esp+4] 00000001
2: [esp+8] 00000000
3: [esp+C] 00788D48
4: [esp+10] 00019600
5: [esp+14] 00000009
```

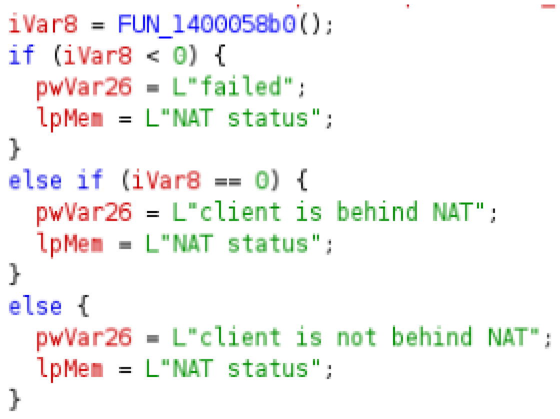
```
Address   Hex                                              ASCII
00788D48  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00788D58  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00788D68  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00788D78  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00  ................
00788D88  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00788D98  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00788DA8  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00788DB8  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00788DC8  67 0D 69 59 23 6C 07 0A 23 6C 07 0A 23 6C 07 0A  g.iY#l..#l..#l..
00788DD8  2A 14 80 0A 22 6C 07 0A 2A 14 8F 0A 22 6C 07 0A  *..."l..*..."l..
00788DE8  4C 1A AC 0A 27 6C 07 0A 2A 14 83 0A 21 6C 07 0A  L.¬.'l..*...!l..
00788DF8  2A 14 92 0A 21 6C 07 0A 2A 14 84 0A 30 6C 07 0A  *...!l..*...0l..
00788E08  23 6C 06 0A 92 6C 07 0A 2A 14 94 0A 3A 6C 07 0A  #l...l..*...:l..
00788E18  4C 1A A8 0A 35 6C 07 0A 4C 1A 9D 0A 22 6C 07 0A  L.¨.5l..L..."l..
```

```
0019FEA0  00404678  sample2.00404678
0019FEA4  00000001
0019FEA8  00000000
0019FEAC  00788D48
0019FEB0  00019600
0019FEB4  00000009
0019FEB8  00001000
0019FEBC  00010000
0019FEC0  FFFEFFFF
0019FEC4  0000000F
0019FEC8  00000004
0019FECC  000021D8
0019FED0  00010000
0019FED4  31000017
0019FED8  00000001
0019FEDC  0019FF70
0019FEE0  00402529  return to sample2.00402529 from sample2.00401000
```

Image 5.

Anti-VM

The malware does use anti-vm techniques to hide itself from an analyst. In order to get past the anti-Vm techniques, first, the Vmware tools service was stopped. This is in an attempt to hide the virtual environment from the malware. Although the Vmware tools are disabled the malware can still find other indicators that it is being run in a vm. Virtual machines use MAC addresses that all lie within a range. This fact can be checked by the malware inorder to identify that it is being run in a virtual environment. In this example the MAC address was changed to AA.AA.AA.AA.AA.AA. Another preventative measure taken was to change the resolution of the display. The final measure to hide the VM is to alter the registry keys so if the malware checks the registry for evidence of a VM it wont find any. The altered key is: HKLM\System\CurrentControlSet\Services\Disk\Enum\0. Most Virtual machines connect through NAT. As seen in image 4 the malware checks if the machine is behind a NAT.



```
iVar8 = FUN_1400058b0();
if (iVar8 < 0) {
  pwVar26 = L"failed";
  lpMem = L"NAT status";
}
else if (iVar8 == 0) {
  pwVar26 = L"client is behind NAT";
  lpMem = L"NAT status";
}
else {
  pwVar26 = L"client is not behind NAT";
  lpMem = L"NAT status";
}
```

Image 4.

Indicators of Compromise

The main indicator of compromise for this sample of malware would be the internet traffic. Although many applications that are not malware communicate over http, this sample makes requests to myexternalip.com along with many other obscure IPaddresses. This is a good indicator of compromise for any malware but not unique to this sample. Other indicators of compromise would be an unknown process running behind the scenes. After finding the unknown process, If the suspicious process is stopped and a service restarts it, it could be this sample as this behavior has been observed. The malware was also observed to delete itself after opening, but remained executing on the machine. This is a strong indicator of infection, but it is still not unique to this sample. One of the final indicators of compromise would be evidence of the files that the malware creates. If a folder named Modules and two files named client_id and group_tag are found then it is highly likely that the machine is infected. If all of the described behavior above is observed the following Yara rule can be run to identify if the malware is in fact the same as this sample. This Yara rule was tested on other executables and only returned positive for this sample. If Yara finds the malware with this rule then it can be confirmed that the machine has been infected with this sample. The rule checks for a string at the entrypoint of the executable.

Yara rule:

```
Rule TrickBot
{
        strings:
                $a = {E8 47 04 00 00}
        condition:
                $a at entrypoint
}
```