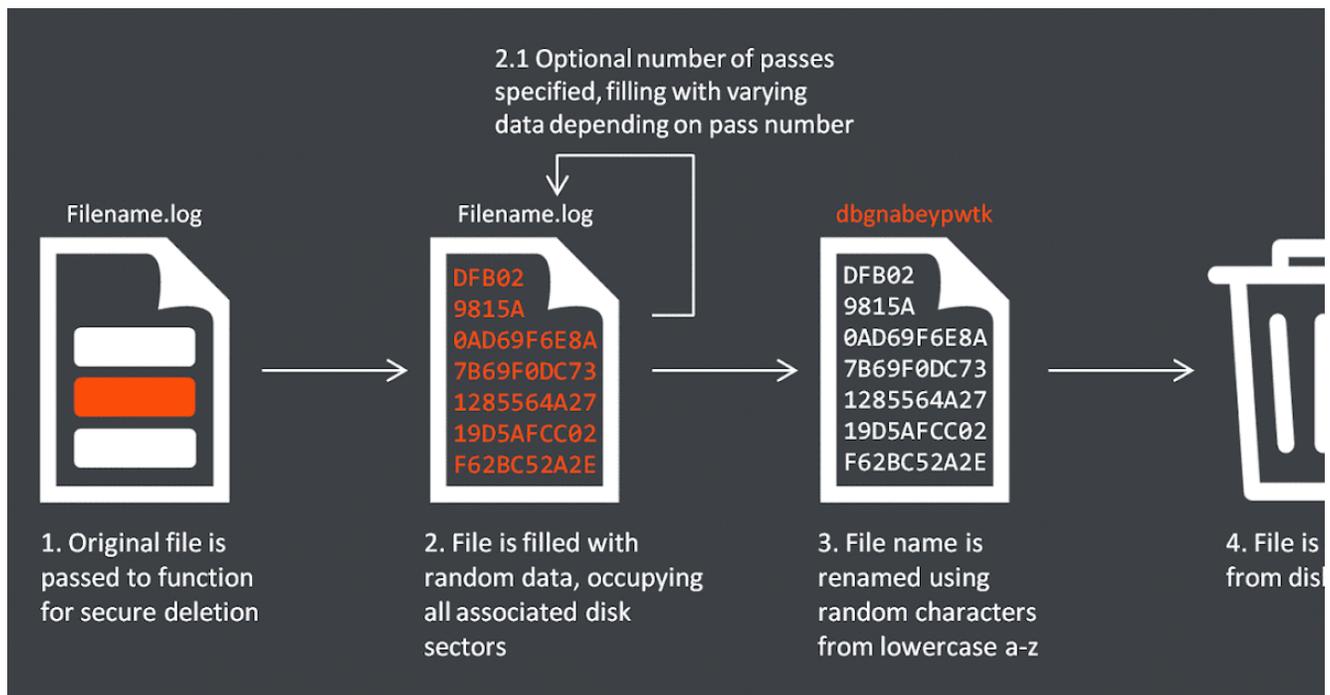# Cyber Heist Attribution

baesystemsai.blogspot.de/2016/05/cyber-heist-attribution.html

Written by Sergei Shevchenko and Adrian Nish

BACKGROUND

Attributing a single cyber-attack is a hard task and often impossible. However, when multiple attacks are conducted over long periods of time, they leave a trail of digital evidence. Piecing this together into a campaign can help investigators to see the bigger picture, and even hint at who may be behind the attacks.

Our research into malware used on SWIFT based systems running in banks has turned up multiple bespoke tools used by a set of attackers. What initially looked to be an isolated incident at one Asian bank turned out to be part of a wider campaign. This led to the identification of a commercial bank in Vietnam that also appears to have been targeted in a similar fashion using tailored malware, but based off a common code-base.

In the bank malware cases we know of, the coders used a unique file wipe-out function. This implementation was so distinctive that it further drew our attention – and so we began to look for other instances of code which had used the same function. Using disassembled machine opcodes (with masked out dynamic virtual addresses) we generated signatures to scan a large malware corpus.

Our initial search turned up an additional sample which implemented the same wipe-out function.

This sample was uploaded from a user in the US on 4th March 2016:

SHA1	Compile time	Size (bytes)	Name	Country
c6eb8e46810f5806d056c4aa34e7b8d8a2c37cad	2014-10-24 09:28:55	45,056	msoutc.exe	US

ANALYSIS

The msoutc.exe functionality

`msoutc.exe` accepts a number of parameters passed with the command line. When executed, it checks if there is another instance of itself already running on a system, by attempting to create a mutex called:

`Global\FwtSqmSession106839323_S-1-5-20`

If the mutex creation attempt fails, returning the 'already exists' error code, the bot will quit and delete itself by executing a self-delete batch file. The bot then copies itself into a temporary directory as:

`%TEMP%\sysman\svchost.exe`

Next, it recreates its configuration file `wmplog09c.sqm` in the same directory. As in case with the malware described in our blogpost, it will also install itself as a system service.

The service will be called *"Indman"*, *"Indexing Manager"*, and described as *"Provides fast indexing service"*.

The malware keeps its logs within an encrypted file `wmplog15r.sqm` and/or `wmplog21t.sqm`, located in the same directory. The logged messages are encrypted with a key:

`y@s!11yid60u7f!07ou74n001`

Back in 2015, PwC reported a malware with an identical encryption key as above and a very similar unique mutex name. The `msoutc.exe` bot matches the description of other tools from the same toolkit, as described in the US CERT Alert TA14-353A:

US Cert Alert	PwC	msoutc.exe
FwtSqmSession106829323_S-1-5-19	FwtSqmSession106829323_S-1-5-19	FwtSqmSession106839323_S-1-5-20
y0uar3@s!llyid!07,ou74n60u7f001	y@s!11yid60u7f!07ou74n001	y@s!11yid60u7f!07ou74n001

If the configuration file is missing, the bot initiates a default configuration and saves it to the `wmplog09c.sqm` file, located in the same directory. The default parameters specify 127.0.0.1 (local host) and port 443 as the default address for the C&C server.

In this case, the bot expects the C&C server's IP address and port number to be passed as the command line parameters. These parameters are then used to update the configuration file.

Since the configuration file is not available for analysis, the C&C IP remains unknown, but the port number is likely to be 443, same as the default value.

Every 5 minutes it initiates a communication to the remote C&C. All communications are encrypted. The beacon signal it sends contains the local IP address. The data it receives from the C&C is first decrypted, and then used to update its own executable and the `wmplog09c.sqm` configuration file. The updated executable may have additional functionality that is missing in the analysed sample.

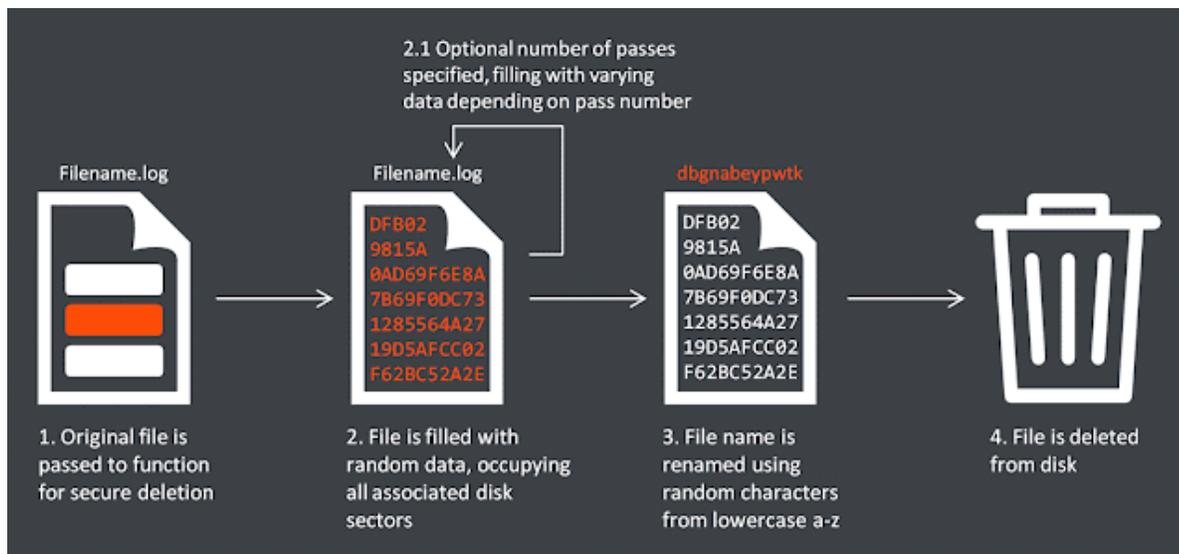The msoutc.exe 'wipe-out' and 'file-delete' functions

When the malware reaches out to the C&C server, it may receive a special instruction to terminate itself. In that case, it uninstalls its service, deletes its configuration and log files, then quits and deletes itself by executing a self-delete batch file that has the following format:

.data:0040A400 echoOffD1Del db '@echo off',0Dh,0Ah
.data:0040A400   db ':D1',0Dh,0Ah
.data:0040A400   db 'del /a %1',0Dh,0Ah
.data:0040A400   db 'if exist %1 goto D1',0Dh,0Ah
.data:0040A400   db 'del /a %0',0

The batch file format is identical to the one previously reported by Novetta.

In order to delete its configuration and log files, the bot calls an internal function that wipes these files out so that their contents cannot be forensically restored.

The implementation of this function is very unique - it involves complete filling of the file with the random data in order to occupy all associated disk sectors, before the file is deleted. The file-delete function itself is also unique – the file is first renamed into a temporary file with a random name, and that temporary file is also deleted.

2.1 Optional number of passes specified, filling with varying data depending on pass number

Filename.log

Filename.log
DFB02
9815A
0AD69F6E8A
7B69F0DC73
1285564A27
19D5AFCC02
F62BC52A2E

dbgnabeypwtk
DFB02
9815A
0AD69F6E8A
7B69F0DC73
1285564A27
19D5AFCC02
F62BC52A2E

1. Original file is passed to function for secure deletion

2. File is filled with random data, occupying all associated disk sectors

3. File name is renamed using random characters from lowercase a-z

4. File is deleted from disk

Let's compare the wipe-out function employed by this bot to the wipe-out function within the malware that was involved in the operation against the bank in Vietnam.

Machine Opcode	Disassembled Code
B8 20 10 00 00	mov eax, 1020h
E8 96 EA 04 00	call alloca_probe
53	push ebx
55	push ebp
57	push edi
FF 15 4C F0 45 00	call ds:GetTickCount
...	...

The first instruction is to assign `0x1020` to `EAX` - this instruction is represented with an opcode `B8 20 10 00 00`, and it stays the same across the compared samples.

The second instruction is to call a function located at the address calculated as the address of the next instruction (`push ebx`) plus `0x4EA96`. Depending on a sample, the compiler may place *__alloca_probe()* function at a different offset within the executable, hence the `0x4EA96` offset is dynamic and the opcode bytes `96 EA 04` (representing `0x4EA96`) should be ignored in our comparison.

The API *GetTickCount()* is called with the last instruction, but again, the pointer into the API is stored in this particular sample at the virtual address of `0x45F04C`. In a modified sample with a different image base, the virtual address of this pointer will also be different. Hence, the bytes `4C F0 45` (representing `0x45F04C`) should also be masked out (ignored).

Using this principle, full comparison of this function's implementation can now be done.

As it turns out, the wipe-out function employed by this bot is byte-to-byte identical to wipe-out function within the malware that was involved in the operation against the bank in Vietnam, as shown below:

Wipe-out function (msoutc.exe, 2014)	Wipe-out function (Vietnam malware, 2015)

```
B820100000E8B64E0000535557FF1500    B820100000E896EA0400535557FF154C
90400050FF154C90400083C404C64424    F0450050FF152CF1450083C404C64424
0CFFFF156890400025FF000080790748    0CFFFF1524F1450025FF000080790748
0D00FFFFFF408844240DB9FF03000033    0D00FFFFFF408844240DB9FF03000033
C08D7C242DC644242C5F33DBF3AB66AB    C08D7C242DC644242C5F33DBF3AB66AB
5368800000006A0353AA8B8424401000    5368800000006A0353AA8B8424401000
0053680000004050C644242AFF885C24    0053680000004050C644242AFF885C24
2BC644242C7EC644242DE7FF15A4AC40    2BC644242C7EC644242DE7FF1548F045
008BE883FDFF7510FF151C9040005F5D    008BE883FDFF7510FF1508F045005F5D
5B81C420100000C3566A02536AFF55FF    5B81C420100000C3566A02536AFF55FF
15D4AC40008D4C242453518D5424386A    1544F045008D4C242453518D5424386A
015255FF15ACAC400055FF15C8AC4000    015255FF1540F0450055FF153CF04500
8D44241C5055FF15D0AC400033F68974    8D44241C5055FF1538F0450033F68974
24188B84243810000083F8067E05B806    24188B84243810000083F8067E05B806
0000003BF00F8DC900000053535355FF    0000003BF00F8DC900000053535355FF
15D4AC40008A4434103CFF75148D4C24    1544F045008A4434103CFF75148D4C24
30680010000051E8C4FDFFFF83C408EB    30680010000051E8C4FDFFFF83C408EB
1C25FF000000B9000400008AD08D7C24    1C25FF000000B9000400008AD08D7C24
308AF28BC2C1E010668BC2F3AB8B4424    308AF28BC2C1E010668BC2F3AB8B4424
208B4C241C33FF33F63BC37C607F0A3B    208B4C241C33FF33F63BC37C607F0A3B
CB765AEB048B4C241C2BCF1BC678157F    CB765AEB048B4C241C2BCF1BC678157F
0881F900100000760BB900100000895C    0881F900100000760BB900100000895C
242CEB048944242C8D4424245350518D    242CEB048944242C8D4424245350518D
4C243C5155FF15ACAC400085C0741E8B    4C243C5155FF1540F0450085C0741E8B
4424243BC3741603F88B44242013F33B    4424243BC3741603F88B44242013F33B
F07CB27F088B4C241C3BF972AC55FF15    F07CB27F088B4C241C3BF972AC55FF15
C8AC40008B44241840894424188BF0E9    3CF045008B44241840894424188BF0E9
1EFFFFFF55FF15A8AC40008B94243410    1EFFFFFF55FF1510F045008B94243410
00005352E847FDFFFF83C4085E5F5D5B    00005352E847FDFFFF83C4085E5F5D5B
81C420100000C3                      81C420100000C3
```

NOTE: the greyed-out areas correspond to the virtual addresses of the called APIs – these addresses would not be expected to be constant across different samples, and hence, should be ignored.

The wipe-out function used in the Bangladesh malware case was slightly different: the author removed one outer loop from the writing into the file, and one randomisation layer, as shown below. However, the core of the function was left intact.

Wipe-out function in the msoutc.exe bot (2014)	Wipe-out function in the Bangladesh case malware (2016)

At the end of the file writing, the wipe-out function makes a call to a file-delete function that removes the file/directory – this is named *removeFileDir()* in our reconstructed code.

This function is implemented identically in all the analysed samples, up to the last character. It's very distinctive as the file is first renamed into a random filename in the same directory, and then deleted, as shown below in the reconstructed code:

```
01    DWORD removeFileDir(LPCSTR lpExistingFileName, bool bDir)
02    {
03    const CHAR *_filePath;
04    char *backslash;
05    char *fileName;
06    char next_char;
07    char _filepath;
08    char buf;
09    _filepath = 0;
10    _filePath = lpExistingFileName;
11    memset(&buf, 0, 0x100u);
12    strcpy(&_filepath, lpExistingFileName);
13    backslash = strrchr(&_filepath, '\\');
14    if (backslash)
15    fileName = backslash + 1;
16    else
17    fileName = &_filepath;
18    if (*fileName)
19    {
20    do
21    {
22    *fileName = rand() % 26 + 'a';
23    next_char = (fileName++)[1];
24    }
25    while ( next_char );
26    }
27    if (MoveFileA(lpExistingFileName, &_filepath))
28    _filePath = &_filepath;
29    if (bDir)
30    {
31    if (!RemoveDirectoryA(_filePath))
32    return GetLastError();
33    }
34    else if (!DeleteFileA(_filePath))
35    {
36    return GetLastError();
37    }
38    return 0;
39    }
```

ATTRIBUTION

So what else do we know about `msoutc.exe` ? It turns out that this malware exhibits the same unique characteristics, such as mutex names and encryption keys, as other tools from a larger toolkit described in US-CERT Alert TA14-353A.

The US-CERT alert mentions "a major entertainment company" and is widely believed to describe the toolkit used to conduct destructive cyber-attack which took place in late 2014. Further details of this same toolkit were disclosed in the 'Op Blockbuster' report in February 2016.

`msoutc.exe` matches the description of the 'Sierra Charlie' variants in their report. From their analysis this is described as a spreader type of malware, presumably used to gain a foothold on multiple devices within a target environment before launching further actions.

Summarising the overlaps:

Typos

Typos are not uncommon in cyber-attacks, as developers and operators rush through their tasks in an effort to stay ahead of network-defenders. However, this provides another soft link across the different sets of activity.

Instance	Comment
Operation Blockbuster campaign	The Kaspersky Labs team noted a typo in the User-Agent of 'Mozillar' instead of 'Mozilla'
Vietnam case	Attacker code featured the string 'FilleOut' instead of 'FileOut'
Bangladesh case	Attacker code featured the string 'alreay' instead of 'already'
Attacker mis-spelled 'foundation' as 'fandation'	

Development environment

The coder makes exclusive use of Visual C++ 6.0, an older development environment released in 1998. The use of such infrastructure for development is not unheard of, but it does leave a distinctive tool-mark with which to link the malware samples.

The use of this compiler was also noted in the 'Kordllbot' samples described in a recent report by BlueCoat. This report details the same activity as discussed in the Operation Blockbuster reports.

Instance	Comment
Kordllbot malware (BlueCoat report / Op Blockbuster)	Noted to have used Visual C++ 6.0
Vietnam case	Malware compiled in Visual C++ 6.0
Bangladesh case	Malware compiled in Visual C++ 6.0

File-wipe / file-delete functionality

As noted already, the developers coded a specific function for securely erasing files from disk. The author of this function took great care in writing a large amount of random data into a file, before passing it to a file delete command. This was done in order to make sure the deleted file could not be forensically restored as all the original contents of the hard disk sectors associated with the file would be lost.

Instance	Comment
Operation Blockbuster campaign	Described under the 'Secure File Delete' section

`msoutc.exe`	Contains both 'file-wipe' and 'file-delete' functions
Vietnam case	Contains identical functions to the above sample
Bangladesh case	Contains modified 'file-wipe' but identical 'file-delete' function

CONCLUSIONS

The overlaps between these samples provide strong links for the same coder being behind the recent bank heist cases and a wider known campaign stretching back almost a decade.

It is possible that this particular file-delete function exists as shared code, distributed between multiple coders who look to achieve similar results. However, we have noted that this code isn't publically available or present in any other software after searching through tens of millions of files. The unique decision to move and rename the file before deletion after overwriting is unusual, and not a common step we would expect to see when implementing this capability.

A motivated and talented coder could implement the same function and choose to compile their project in Visual Studio 6.0 to achieve the same binary result for the purposes of misattribution. We cannot prove for certain that this is not the case, and further investigation of command infrastructure and related tools is needed before any definitive statements on attribution can be made.

Whilst there are possibilities that exist which may lead to alternative hypotheses, these are unlikely and as such, we believe that the same coder is central to these attacks. Who the coder is, who they work for, and what their motivation is for conducting these attacks cannot be determined from the digital evidence alone. However, this adds a significant lead to the investigation.