SYSCON Backdoor Uses FTP as a C&C Channel

blog.trendmicro.com/trendlabs-security-intelligence/syscon-backdoor-uses-ftp-as-a-cc-channel/

October 5, 2017

Malware

Bots can usually establish communication with their C&C server via HTTP or other TCP/IP connections. However, we recently encountered a botnet that uses a more unusual method: an FTP server that, in effect, acts as a C&C server.

By: Jaromir Horejsi October 05, 2017 Read time: (words)

Bots can use various methods to establish a line of communication between themselves and their command-and-control (C&C) server. Usually, these are done via HTTP or other TCP/IP connections. However, we recently encountered a botnet that uses a more unusual method: an FTP server that, in effect, acts as a C&C server.

Using an FTP server has some advantages. It is less common, and this fact may allow it to slip unnoticed by administrators and researchers. However, this also leaves the C&C traffic open for monitoring by others, including security researchers. In addition, thanks to a coding mistake by the attackers, this particular backdoor does not always run the right commands.

Infection Chain The infection chain starts with a malicious document with macros. The examples of delivery documents are shown below. The documents suggest that targeted individuals may be connected to the Red Cross and the World Health Organization. All the documents mention North Korea as well. We detect these delivery documents as W2KM_SYSCON.A.

Figures 1-3. Delivery documents with macros (Click to enlarge)

Each document contains two long strings, with Base64 encoding using a custom alphabet. This same technique was used to deliver the Sanny malware family in late 2012.

Figure 4. Base64 decoded function with custom alphabet highlighted

Its similarities with the earlier Sanny attack are interesting. Both attacks used relatively unusual techniques for their C&C server, their structure is similar, and the encoding key is identical. Documents somehow tied to North Korea were also used. We cannot eliminate the possibility that both Sanny and this new malware family were the work of the same threat actor.

Decoding each Base64 string results in a cabinet file. One string contains a 32-bit version of the malware; the other contains a 64-bit version. The appropriate version (based on OS version) is extracted using the *expand* command into the *%Temp%* folder, and *uacme.exe* (one of the files in the cabinet file) is executed. (We detect the malicious files in these cabinet files under the following detection names: BAT_SYSCON.A, BKDR_SYSCON.A, and TROJ_SYSCON.A.)

Figure 5. Command to extract cabinet archive and execute its contents (Click to enlarge)

Figure 6. Listing of all files in cabinet archive

UAC bypass and installation*Uacme.exe*, as the name suggests, determines the operating system version. Based on that information, it either directly executes *install.bat* (for older Windows versions) or injects *dummy.dll* into the *taskhost(ex)* process, which attempts to execute *install.bat* without a UAC prompt appearing.

Figure 7. uacme.exe and UAC bypass

Install.bat copies two files: *ipnet.dll* (the main file) and *ipnet.ini* (configuration file) into *%Windows%\System32*, configures new malicious *COMSysApp* service using the *sc* command line utility, adds the service parameters into the registry, starts the malicious service, and deletes all previously created files in the *%Temp%* directory. This does two things: it sets up the backdoor's autostart routine, and deletes some traces of its previous activity, making detection more difficult.

Figure 8. Service installation

Important parameters for the service configuration are "*type=own"*, which according to MSDN documentation means that it has its own *svchost.exe* process in which the *ipnet.dll* runs. The parameter "*start = auto*" starts the service every time the computer is restarted.

Ipnet.ini is a text file encoded in the same way as the two cabinet files.

Figure 9. Configuration file

Decoding the configuration reveals a URL for the byethost free FTP service provider, as well as a set of login credentials.

The backdoor The malware first gets the computer name, which it uses as the affected machine's identifier. It then logs into the FTP server using the credentials in the configuration file, enters the */htdocs/* directory, and monitors existing *.txt* file names. If the file name contains "To EVERYONE", it means that the file should be processed by everyone. If it contains "to computer_name", then the file should be processed (and later deleted) only by the victim computer with the matching computer_name.

After the backdoor processes the command, it lists all the currently running processes by calling "*cmd /c tasklist >%ws*", which is then packed, encoded and uploaded to FTP under following name pattern "*From %s (%02d-%02d %02d-%02d-%02d).txt*", i.e. computer name followed by date and time of the task execution.

Figure 10. Construction of name of the uploaded file

This shows that the communication between the victim's computer and the bot master is done via uploaded files. However, the files are generally zipped and encoded with the same custom Base64 encoding used earlier.

Compression of the files is done using the Shell Automation Service, which is part of Windows itself. No external library is needed. The malware first creates a 22-byte long empty *.zip* file by executing the instructions below. The content of the newly created *.zip* file in hex is "*504b050600000000000000000000000000000000*", which is basically an empty *.zip* archive.

Figure 11. Instructions creating empty .zip file

Figure 12. Empty .zip file opened in Explorer

The malware then uses the *Folder.CopyHere* method to copy files into the empty *.zip* archive, so these files are compressed by the operating system. According to MSDN documentation, these are the flags used while copying:

- 0x1000 = Only operate in the local directory. Do not operate recursively into subdirectories.
- 0x0400 = Do not display a user interface if an error occurs.
- 0x0010 = Respond with "Yes to All" for any dialog box that is displayed.
- 0x0004 = Do not display a progress dialog box.

Figure 13. Flags used by Folder.CopyHere method

These flags ensure that no dialog box appears and no errors are shown to the victim. All these operations remain hidden in the background. After compression is finished, the compressed files are encoded with Base64 and uploaded to the FTP server. For downloading files from FTP server, the reverse operations would be performed by the threat actor.

C&C communication protocol Bots listen to and can process several supported commands:

Command	Meaning of command
cmd /c pull /f <file_name>	copy <file_name> to *temp.ini*, pack it to *temp.zip*, encode and upload
cmd /c pull <file_name>	pack <file_name> to *temp.zip*, encode and upload
cmd /c chip <string>	delete config file, write <string> to the new config file
cmd /c put <new_file_name>	put file from #<content># to the given path on infected system
cmd /c <command> > <file>	execute command and redirect its output to file, file gets zipped, encoded, uploaded
cmd /c <command>	execute command; do not report about it back to c2
<parameters>	parameters to previously downloaded <file>, called <file> <parameters> using Winexec API, flag Show=SW_HIDE
/user <parameters>	execute previously downloaded <file>, called <file> <parameters>; do not report about it back to c2; use CreateProcessAsUser
/user <parameters> /stext	execute previously downloaded <file>, called <file> <parameters> /stext "*%APPDATA%\Temp\Temp.ini*", result gets zipped, encoded, uploaded; misuses parameters from Nirsoft's utilities; use CreateProcessAsUser
#<content>#	<file>, which is unzip(base64decode (<content>))

Table 1. C&C commands

Figure 14. Processing C&C commands

Here are some examples of commands that could be issued using this method: *Example 1* Command:

Encoded as:

> lxXDK=NK2KKQK=zK2KXxKB-K0KXTKUKKIxKeKPxKINX8KBNK0xXpKBWKnNX6KUjKIWXZKBNK

Result:

> -yT/XXNKKKKKKyD8J6G=UtS6=KKKKXWKKKKzKKKKIVSH0hSjnbz1IFPH-OSi2yR80b8M2=nA0=-WnFYJ2yM-SORi-
> yTXKZNKPKKKKKKKu4VtdTI2AbzIKKKK=KKKKKxKKKKKKKKKKNKWKKKKKKKKK=P8C=Ypr=S6LF9ZCPXLXNIKKKKKKNKXK/7KKKXUK

This result contains the contents of the *autoexec.bat* file. *Example 2* Command:

Encoded as:

> lxXDK=NK2KKQK=zK2KX6K=NK2KKQK=NK2KX/K/7KVKKWKUIK9WKWK=NKwNXJKUKKLxXQKUHKnKKQKBzK2KK5KUjKtWKWK/jKKKK

Result:

> -yT/XXNKKKK2KyHRLOQ26PNrulo0KPtgJWK2KKKKC=SDoUMA0FczVC81jHAJQpYW4j=VU-LAYKl……..

This results in a file with a list of all files in C:\, sorted from newest to oldest. *Example 3* Command:

> *#…encoded file…#LxX8KBzKnNXJKUKKLxXhK=-oWX3K=OK0xX1KUKKzNKHKUKKLxX3KBNKnNXjKBNKKKKv*

Decoded as:

Result:
In this case, attacker sent an encoded version of Nirsoft's utility *mailpv.exe*, which was then executed with parameters above.

Mistakes in malware code The command processing loop contains what appears to be a typo or mistake. The malware treats the command-and-control commands as strings in wide character format. The second *wsprintfA* function in the snippet below has the first format parameter *"%s*", which should actually be *"%ws".* Because of this, only the first character of *lpExistingFileName* is propagated into *&CommandLine*, preventing the process from executing because of the incorrect file name.

Figure 15. Code snippet with a typo (Click to enlarge)

ConclusionIt is interesting to see something atypical, like C&C communication via FTP. While the malware authors probably used this method in an attempt to avoid security solutions inspection and/or blocking, they may not have realized this would make it very easy to monitor their actions and victims' data.

IT administrators should be aware that connections to external FTP servers can signify not just data extraction, but C&C activity as well. Either way, if this kind of network activity is not necessary for business functions, blocking it should be considered.

Indicators of Compromise Files with the following SHA256 hashes are connected to this attack, and are detected as W2KM_SYSCON.A:

- 34e968c067f6a360cc41a48b268c32a68421567f0329d4f9f8e2850fb4e27c8c
- 63ca182abb276e28aec60b9ef1eab5afc10bfb5df43f10a11438d8c0f7550c5c
- a07251485a34dd128d80860737b86edd3eb851f57797f2f8fb6891a3cb7a81b3
- cff8d961f3287f9ca75b65303075343bdbe63bb171d8f5b010bbf4fa30450fc4
- f4987d127320cb5bfb8f49fc26435e01312bdd35a4e5e60db13546046584bd4e

Files with the following SHA256 hashes are detected as BAT_SYSCON.A:

- 2c958cd3838fcae410785acb0acf5a542d281524b7820d719bb22ad7d9fcdc7c
- e4226645bad95f20df55ef32193d72c9dafcf060c3360fd4e50b5c08a986a353
- f01e440764b75b72cab8324ba754d89d50d819a1b2db82ca266f1c307541a2b0

Files with the following SHA256 hashes are detected as BKDR_SYSCON.A:

- 1f9afb142827773cefdb29f06ed90e0476c0185d4c8b337439b3be27e61ed982
- 65e4212507bb52e72e728559df5ad38a4d3673b28104be4b033e42b1c8a264e8
- 9b62a013b579f01e3c4c3caf3c9bc02eb338ce9859496e02016ba24b8908d59a
- 9be95f5954202d7b159c5db928851102f23eae88c087892663781cf8edc0753a
- bec437d1979d16505ca8fc896fa8ce9794f655abd39145a82330343b59c142c5
- cfb2161b5aebf0c674c845e2428e24373edd4c74a2fb15de527d6763a62dd74e

Files with the following SHA256 hashes are detected as TROJ_SYSCON.A:

- 25c08d5e77fada975f31a0e0807b7ea1064aae80f5de43790f6ada16159ae1c2
- 2d261eb478bafaabd7dc12752b1c0aadba491d045573fe2e24cdac5588e2c96b
- 2f6df307dbe54b8a62a35ea2941a7d033bfdfbb545a7872cb483aea77ec6a10b
- 3319a156c84e85a4447fa40b0f09aabb84092b5c3a152ad641ee5692741b9194
- 3fcda66e87eec4f90b50f360460fa46448249e6e177de7ff8f35848353acfaaa
- 65380ab72bb6aa6ffcd2ea781fe2fa4f863a1b4a61073da7da382210c163b0f9
- 7daec65f8fee86227d9f9c81ed00d07c46b44e37968bd2894dc74bf311c63651
- b7c970f1f65850fa859549f2cf3c2284b80ec464496b34f09bc53c4456e10d1f
- d495295466428a52263c8725070a9cf7c2446c6115bddc2de662949afd39f9a9