CATCHING THE SILENT WHISPER: UNDERSTANDING THE DERUSBI FAMILY TREE

Micky Pun, Eric Leung & Neo Tan
Fortinet, Canada

Email {mpun, ericleung, ntan}@fortinet.com

ABSTRACT

From stealing sensitive information from *Mitsubishi Heavy Industries* in 2011 to the *Anthem* data breach revealed in February 2015, the complexity of the Derusbi malware family has been the real driving force behind all these espionage campaigns. Upon entering a targeted company through a newly discovered vulnerability, Derusbi would be dropped on the compromised computer with the purpose of setting up a well hidden front-line camp in the targeted organization. Sophisticated in its covert presence and uncommon malicious activity, our records reveal that many security vendors did not start to detect a Derusbi variant until it had been circulating among vendors for half a year. While most articles focus on linking how these attacks are country-sponsored through the venue of passive DNS, we want to focus on the technical aspects of the Derusbi family. In addition, we will look at how it has evolved over time to remain under the radar of most vendors, as well as at its rare and noteworthy approaches to conducting its espionage objectives.

BACKGROUND

Judging by the compilation times of the malware samples we collected, we believe that the history of Derusbi dates back to 2008 (MD5: 338e4deb0be7769ef2c9d7080fb56154). The first reported incident of Derusbi was from the *Mitsubishi* hack [1, 2] in October at 2011. The sample from that attack indicated a compilation time of 15 April 2011. Although we knew of no other major reports of hacks or compromised data between 2011 and 2014, reports about the group(s) using Derusbi, such as ShellCrew [3] and Deep Panda [4], were published by companies with incident response teams during this period. This indicates that the malicious groups were still actively developing and running campaigns utilizing this special tool during this time frame. Through analysing the samples compiled between 2011 to the present, we observed that the malicious beacon URL used by Derusbi has been seen to be associated with military equipment or public infrastructure (see Appendix A). The latest attack to make the news headlines was the *CareFirst BlueCross BlueShield* attack, where personal data of approximately 1.1 million of customers was breached during 2014 [5].

MODUS OPERANDI

The Derusbi samples that we will refer to are Win32 or Win64 DLL files. As a DLL file, it requires other files to load it before it can run its malicious payload. In our collection, we have found malicious droppers that contain both encrypted 32-bit and 64-bit Derusbi samples. Since we did not have access to any Derusbi-infected machines, we do not know exactly how the droppers got onto the infected computers. However, we were able to find other malware samples that share some unique features, helping us link the Derusbi family to other pieces of malware that might be used together during an espionage campaign. Figure 1 shows a couple of malware families that have similarities with Derusbi.

Sakula

The Sakula malware family (MD5: c384e7f567abd9ea50f647715a28661a) consists of executables that were linked to the

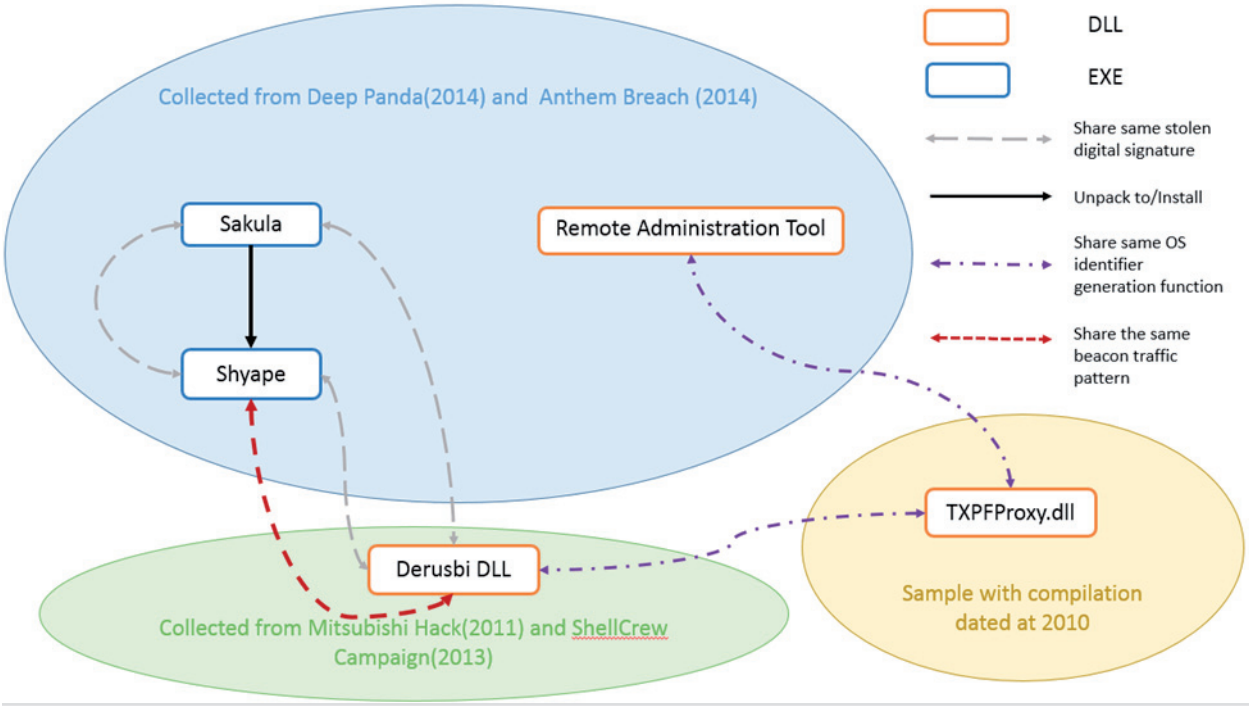

Figure 1: Relationship between different malware families and Derusbi.

Deep Panda campaign in early 2014 and the *Anthem* attack revealed in February 2015. Sakula is a packer that contains an embedded downloader in its resource section, and during execution, the packer will decrypt and drop the downloader. From the samples we have seen, the dropped downloaders are from the Shyape family. Some Sakula samples have the same stolen DTOPTOOLZ Co. [6] digital signature as Derusbi samples (MD5: 0a9545f9fc7a6d8596cf07a59f400fd).

Shyape

The Shyape family of malware (example MD5: 528963946e5040ebcda9d8982f911f4a) are generally downloaders which are distinguished by their traffic. The downloaders' traffic routinely uses POST requests pointing to a photo subdirectory, similar to the traffic created by Derusbi. In addition, some Shyape samples contain the same stolen digital signatures as were used by Derusbi clients.

Remote administration tools infoadmin.dll and sqlsrv32.dll

The remote administration tools (RATs) infoadmin.dll (MD5: 47619fca20895abc83807321cbb80a3d) and sqlsrv32.dll (MD5: e1b09815bacaa7ca3fed0d3aca849055) can be unpacked from the Deep Panda campaign (MD5: 528963946e5040ebcda9d8982f911f4a) and *Anthem* breach IOCs. These RATs would allow the attackers to perform actions such as keylogging and accessing the file systems of comprised computers.

TXPFProxy.dll

The samples we have seen of the TXPFProxy.dll (MD5: 50b6dee51ef2d9ec1a81dbfb62d51448) were compiled around 2012, and it seems to be a more primitive version of the RATs mentioned above. We have observed that TXPFProxy.dll shares similarities with Derusbi samples collected from the ShellCrew campaign and the RAT samples from the Panda campaign. We encountered it because the way it constructed the OS version identifier (PCC_MISC System Info, described below) seems consistent in style (demonstrated in Figure 1). Hence it gives us reason to believe that all these incidents are from the hands of the same malware developer.

VARIANTS OF DERUSBI

Unlike common polymorphic malware that has similar features and behaviours amongst different samples, we can see that the authors of Derusbi added in a high degree of customization to the builds, as the various Derusbi samples we analysed contained different combinations of the available modules. Because the previous reported victims were from pretty specific industries or companies, we believe that Derusbi samples are used only in targeted attacks, and that a specific version of Derusbi was compiled for each of its targets depending on the needs of the campaign. We believe that the samples were dropped as a post-exploitation tool to gather and steal information.

In Figure 3 we outline the relationship between various features present in different samples since 2008. The features are included selectively depending on need, and because of this, newer samples do not have a higher number of features than older samples, even though new features have continuously been developed. Combined with a relatively low sample collection rate, this reinforces the idea that the

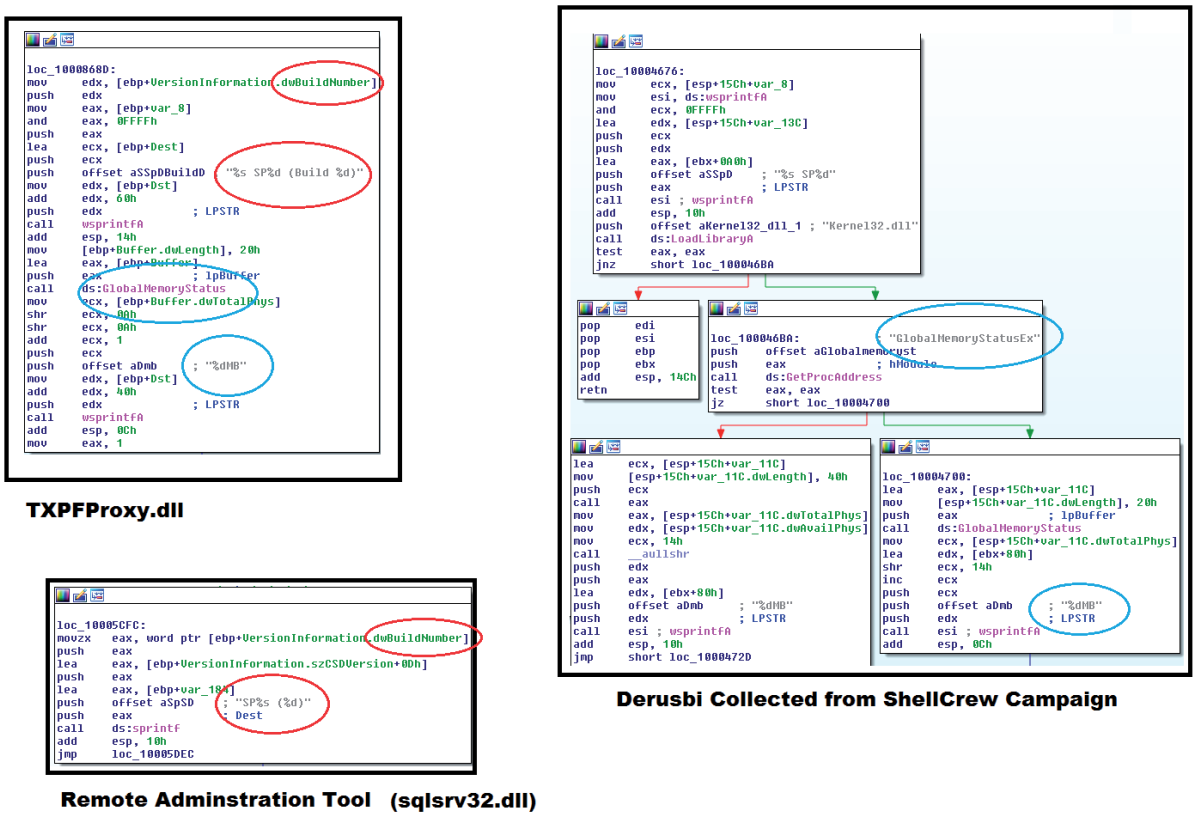

Figure 2: TXPFProxy.dll has similarities with both RAT and Derusbi samples collected from the ShellCrew campaign.

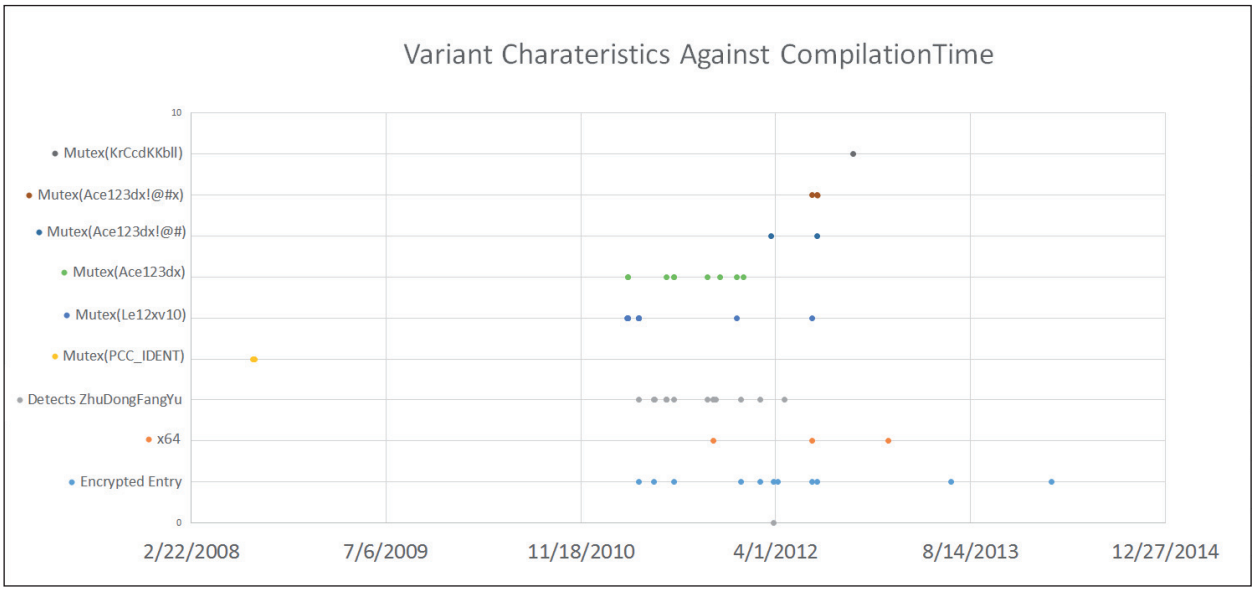

Figure 3: Different feature sets in Derusbi variants, plotted chronologically.

Derusbi-related espionage campaigns use malware tailored to the targeted victims.

Figure 3 shows some of the notable characteristics of some Derusbi variants. Samples targeted for different campaigns will have different features/characteristics. The following are some notable insights regarding the evolution of Derusbi:

- From the samples that we have gathered, it seems that a 64-bit version was only introduced in late 2011 and that 64-bit samples are a rarity in our collection of samples.
- One of the classes/modules used in gathering PC information (PCC_MISC) does not report Win8 and beyond, even from a sample that had a compilation date of November 2014.
- Different versions and variants of classes/modules are available, and newer samples do not necessarily use a newer version of a certain class, even if the newer class has been available for over a year.
- We can see significant differences between the samples from 2008 and the samples from 2013/2014. The malware has much more functionality in the more recent samples.

From the different combinations of classes and embedded tools that we have seen bundled with Derusbi, we generalize the Derusbi family as an underlying framework that provides a well-hidden backdoor connection between the infected host and the outside world. In the following sections, we will take a detailed dive into its code structure and reveal all the available functionality provided by this tool.

TECHNICAL ANALYSIS

DLL entries

Hiding itself as a service on the infected machine, before Derusbi's export function ServiceMain/DllRegisterServer/DllUnregisterServer runs, the DLL file is first loaded by svchost or regsvr32. In some older samples, the malicious functions can be found right at the DLLEntryPoint. In other samples, the DLLEntryPoint function is simply used for initialization and the main malicious subroutines are within the exported functions. To complicate things, we have also observed a significant number of samples that are packed, with the DLLEntryPoint subroutine containing code to unpack the other export functions during DLL loading. In addition, although Derusbi comes with a number of different export functions, only the export functions DllRegisterServer, DllUnregisterServer and ServiceMain are of importance. The other export functions seem to be a mere distraction.

Persistence management

When the DLLRegisterServer export function is initially invoked during service registration, Derusbi will decrypt built-in configuration data that has been XOR-encrypted with a hard-coded DWORD key. It will then proceed to move the original DLL to a hard-coded location in the Windows directory or as specified in the built-in configuration data. The DLL will be stored with the filename '[prefix][random_string].[extension]', where the prefix and extension are hard-coded. The original file will then be set for deletion after a reboot (using the MOVEFILE_DELAY_UNTIL_REBOOT flag).

After saving itself in the file system, Derusbi will register itself as a system service with the name 'svc_name' (a list of the service names that Derusbi uses is given in Appendix B). First, it will set the registry value 'netsvcs' with data 'svc_name' (the actual service name it is pretending to be) at the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost. Then it will alter the path of the legitimate service that is loaded by setting the registry value 'ServiceDLL' to the local path of the Derusbi sample at the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\svc_name\Parameters. In the final step to ensure a successful set-up before it starts running the malicious part of the malware, Derusbi will check whether certain anti-virus processes (vstskmgr.exe and mcshield.exe) are running in the background. If they do exist, Derusbi will make a copy of

regsvr.exe and rename the copied regsvr.exe to update.exe. Then, through this 'update.exe', the DllUnregisterServer export function of Derusbi can be run without triggering detection by anti-virus products.

When the DllUnregisterServer export function is called by the DllRegisterServer subroutine, the malware checks whether there are other instances of Derusbi running by checking for the existence of a specific mutex. If the specific mutex has not been created, it will proceed to execute the main payload.

Payload

The rest of the ServiceMain export function can be divided into two major parts:

1. Escalation of privileges of the current process and SeDebugPrivilege, SeLoadDriverPrivilege, SeShutdownPrivilege and SeTcbPrivilege, and creation of a 'Connection Thread' which contains all the malicious communication and packet handling for Derusbi.
2. Patching an original service such that it will be run after the malicious routine. This ensures that the *Windows* environment will still function normally without raising any suspicion. (Appendix B contains a list of service options for patching.)

Configuration

Throughout the Derusbi client, a configuration is used to control the client's behaviour. The default configuration of the Derusbi client is encrypted and hard-coded into each client. Once the DLL is loaded/registered, the configuration can be decrypted by a four-byte XOR key that is also hard-coded in the sample. The configuration settings will be stored in the registry for persistency, encrypted using a combination of NOT and XOR with a different key. The client usually picks an existing HKLM registry subkey to store the encrypted configuration settings. Whenever the client is initialized, it will attempt first to extract the configuration from this registry key and only fall back to using the hard-coded configuration in the malware sample if configuration data extraction from the registry key fails. From looking at different configuration data in various samples, some samples' configurations have 0s between various fields, while other samples have random/junk data between various fields.

We compared the configuration data from different samples and came up with a data structure that will apply to most Derusbi samples. Table 1 shows the important members of the built-in configuration structure.

Optional functionalities

Other than the operating client, the Derusbi malware can be embedded with different kinds of executables and install them onto the system at runtime.

Keylogger DLL

The samples (e.g. MD5: a4df6ca8904f1073c3de09bf77aa3bed) that contain a keylogging driver as an embedded file will check if the Zhudongfanyu.exe process (a background process of an anti-virus product from *360 Total Security*) is currently running using the ZwQuerySystemInformation API. If Zhudongfanyu.exe does not exist, this driver will be dropped.

USB drive infector

Some of the samples (e.g. MD5: 92d18d1ca7e66539873be7f5366b04d1) contain a USB-driver-infecting function that can plant and invoke a copy of the Derusbi client in other drives. The infector will iterate the whole directory and drop a copy of Derusbi where it contains files which also have export functions 'DllRegisterServer', 'DllUnregisterServer' and 'SvchostPushServiceGlobals'. After that, it will create an autorun.inf file that will silently register the Derusbi client stored in the drive when a device is plugged in and auto-played.

In one sample of the USB drive infector, a path for program debugging database 'G:\Work\hack\BackDoor\Ctrl11Sum\i386\UdiskExe.pdb', was found. This path also corresponds to other .pdb file paths we have seen inside Derusbi samples (e.g. S:\Work\Hack\Backdoor\Ctrl12Nov\trunk_OUT\MainDLL64.pdb and G:\Work\hack\BackDoor\RkBios\i386\DoorInst.pdb). These paths provides some interesting insight into the malware authors' development environment. One thing we can note here is that 'U disk', translated into Chinese, is actually the usual way to refer to a USB drive.

Network hooking driver

Some Derusbi samples (MD5: a1fb51343f3724e8b683a93f2d42127b) contain an embedded

Size	Description
N/A	Infected machine identifier: (Computer name + random number) or just random number. (Runtime generated)
N/A	Beacon URL
4 bytes	Sleep time/interval
N/A	Persistence service name (e.g. hlpsrv), which is what the Dersubi client is patched to run its malicious payload before loading the actual legitimate driver (e.g. Still Image Drivers, Microsoft Windows Updater Module or Microsoft PCHealth Service)
4 bytes	Unknown flag
N/A	Proxy server (if it exists) (e.g. 172.22.161.19:8080)
N/A	Persistence file path where the Derusbi client is stored on the computer under a different name (e.g. C:\....).

Table 1: Derusbi configuration structure.

device driver that will be decrypted and installed on the system. It performs SSDT hooking on the NSI proxy driver. It also uses a special keyword, 'KDTR', to name the pool tag for allocation.

Connection thread

Most of the samples contain code in the beginning which delays the start of the network connection.

1. The malware loads the existing or built-in configuration and connects to the URL if it starts with http. Otherwise, it will use the IP address to follow instead.
2. The malware will query and check whether the stored URL has the same IP address as expected. If it doesn't, it will use the hard-coded IP address, which points to the malicious site, as a last resort.
3. The malware will attempt to initialize a connection with the IP address, if this IP address gives a response. It will attempt to use this IP address as a proxy to connect to a malicious URL.
4. Once the malware has verified that it can reach the malicious URL, it will attempt to send an encrypted data packet which contains the OS environment information of the client.
5. The client will establish itself a select server to handle traffic between the C&C server and start to handle the incoming traffic/packets with its C++ classes described in the following section.

CLASS ANALYSIS

Thanks to the Derusbi samples being written in C++, we were able to gain some insight into what classes were present in its source code. For each sample, it appears that the malware authors made a deliberate choice of which classes to include. We made a chart of what classes were included in the samples if we organize them chronologically. Our findings match what others have found previously in [7] and [4].

Command classes

Through static analysis of different samples, we were able to get a sense of what types of commands are available from the C&C servers. Each type of command is handled by a different processing class that usually starts with the prefix 'PCC_'. The following subsections are a summary of these classes.

PCC_ class commands

All the PCC_ classes that handle commands from the C&C servers are child classes of a base class called 'PCC_BASEMOD'. These child classes all implement a handler function that allows the corresponding class to conveniently be called when a message is received from the C&C server. Interestingly, the enums or source code had gone through a significant change, as we encountered two different versions of the command ID or command type corresponding to the PCC_ classes. This change might be due to having added the INTERNAL_CMD classes that sometimes replaced the PCC_CMD class. Even though newer samples have the PCC_CMD class in the code, it seems the PCC_CMD class is no longer derived from the PCC_BASEMOD class, and that the class only contains a destructor member function.

We list the IDs of the child classes of PCC_BASEMOD in Table 2. Note that not all samples will have all the classes listed. We usually see at least the PCC_CMD/INTERNAL_CMD class along with the PCC_SYS and PCC_FILE classes.

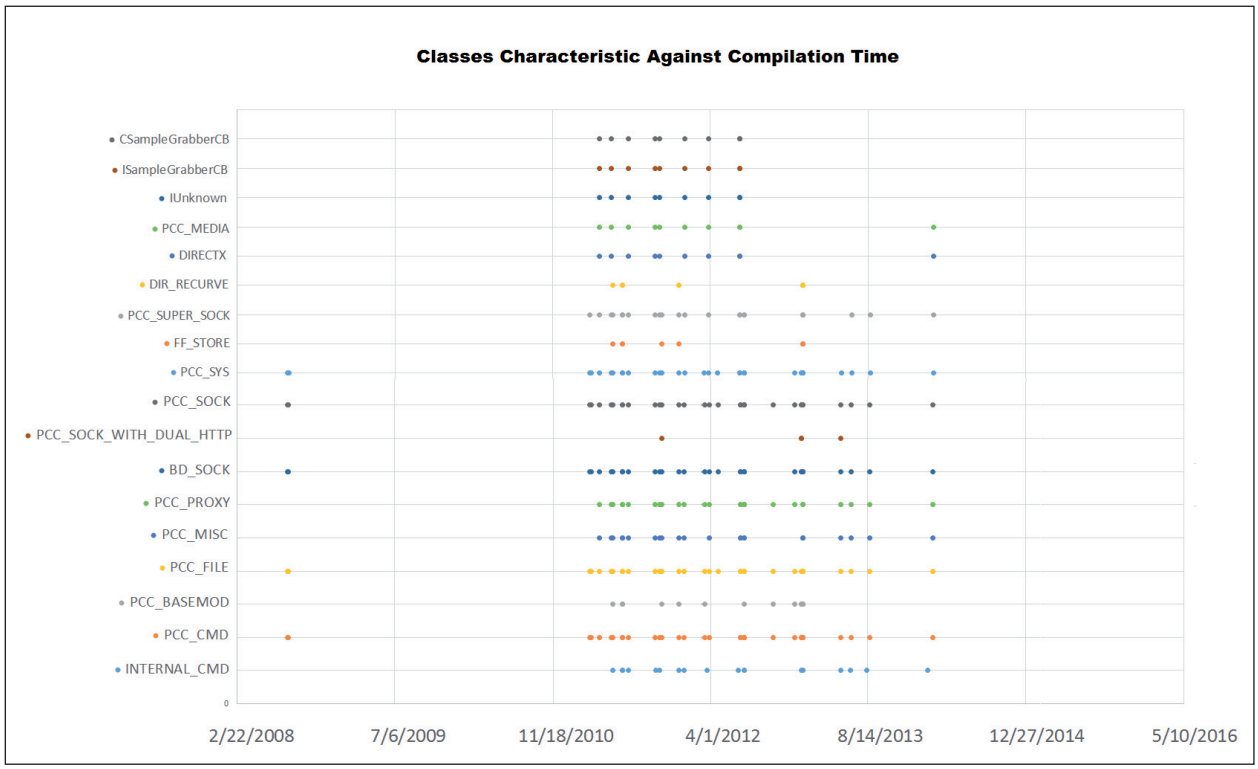

Figure 4: Class characteristics against compilation time.

The PCC_BASEMOD class hierarchy was not seen in the earliest samples that date back to 2008 and was probably not introduced until 2012.

Older samples	Newer samples
PCC_SYS – 80h	PCC_SYS – 4
PCC_CMD – 81h	INTERNAL_CMD – 5
PCC_PROXY – 82h	PCC_PROXY – 6
PCC_FILE – 84h	PCC_FILE – 8
Default_handler – F0h	PCC_MISC – 10
	Default_handler – 100h

Table 2: PCC_ classes and their class IDs.

Default handler class commands

The default handler class is always present in the samples. It is contains a bare-minimum number of actions for remotely controlling the malware such as stopping execution or updating the DLL sample. Tables 3 and 4 are lists of functions along with their command IDs for the newer and older samples.

Command ID	Action
2	Clean up the classes data
3	Back up config to registry, set current file to be deleted on reboot, terminate current process immediately
4	Terminate after current jobs
5	Save file from command data to temp location, load temp file as DLL, and run DllRegisterServer, then delete temp file. This effectively uninstalls the old one while installing a new DLL. Set current file to delete after reboot.

Table 3: Default handler class command IDs (newer).

Command ID	Action
8	Terminate current connection
10h	Clean up all command classes
14h	Back up configuration to registry, set current file to be deleted on reboot, terminate current process immediately
18h	Terminate after current jobs
1Ch	Save file from command data to temp location, load temp file as DLL, and run DllRegisterServer, then delete temp file. This effectively uninstalls the old one while installing a new DLL. Set current file to delete after reboot.
20h	To be analysed

Table 4: Default handler class (older).

PCC_CMD class

This class seems like it is a deprecated class. It seems that whenever the INTERNAL_CMD is present, the PCC_CMD class is also present, but it does not have any member functions. When the PCC_CMD class is alone, it will accept different commands. This class is present in the earliest sample we have, which dates back to 2008.

INTERNAL_CMD class

There are two known versions of the INTERNAL_CMD class, version 1.1 and version 1.2. There are some new additional commands in v1.2. It does not use command code/IDs like the other classes. This class was first seen in a sample with a compilation date in late 2011. If the compilation date is correct, then that means this class would have been available while PCC_CMD was still used.

Commands	Action
help or ?	Print a help menu
cd	Go to directory
dir	List files in dir
md	Make directory
rd	Remove directory
del	Delete file
copy	Copy file
ren	Move file
type	Display information about command type
start	Run a program in the background

Table 5: INTERNAL_CMD v1.1 and 1.2 commands.

Commands	Action
runas	Run a program as the same user as another process with pid
reboot [-f]	Restart computer [force]
shutdown [-f]	Shut down computer [force]
clearlog	Clear event logs
wget [httpurl]	Download and save a file

Table 6: INTERNAL_CMD v1.2 commands.

PCC_MISC class

The PCC_MISC class provides miscellaneous functions. There are two types of commands, numerical and text, as shown in Tables 7 and 8.

Command ID	Action
1	Save file attached in message from C&C to a temp file. Remembers up to 16 files. Then the temp files are loaded as DLLs to install them.
2	Delete a temporary saved file. Filenames of the attached in message. Must match previous filenames from C&C.

Table 7: Numerical command IDs.

Command	Action
pstore	Password stealer function that steals password and account info stored in Internet Explorer (IE) and Mozilla Firefox. For IE stored passwords for versions 4–6, as well as 7 and above. For IE 4–6, MSN messenger and Outlook, passwords are stored at the same location and they are also gathered.
keylog	Send back saved keylog information to C&C
Info	Gathers system information

Table 8: Text commands.

Table 9 shows more details of the gathered system information.

PCC_SYS class

PCC_SYS class offers additional system-related functions. This class, along with PCC_CMD and PCC_FILE, has been seen since the earliest Derusbi samples. The commands can be split into four groups: a screenshot command, process-related commands, service-related commands, and registry-related commands. The screenshot command will take a screenshot of the user desktop. The other commands are described in Tables 10–12.

PCC_FILE class

The PCC_FILE class provides support for common file operations. This class has also undergone changes over the

System info	OS_NAME	BUILD_NUM
	Win32s Win95 Win95 OSR2 Win98 Win98 SE WinMe NT Works Win2000 Server Win2000 Advanced Server WinXP Home/Pro WinXP64 Home/Pro Win2003 (R2) Win2003 (R2) Datacenter Itanium Win2003(R2) Enterprise x64 Win2003 (R2) Enterprise Itanium Win2003 (R2) Standard x64 Win2003 (R2) Web Vista Home/Pro Win2008 Win7 Home/Pro	SPxx (yy) xx = service pack number yy = build number
Adaptor info	For each network adapter, get • Adapter name • Adapter description • MAC address • IP addresses associated with the network adapter	
IE version		
Proxy info	Look for proxy server addresses for all currently active users stored under the 'ProxyServer' value at HKEY_USERS\xxxx\Software\Microsoft\Windows\CurrentVersion\Internet Settings xxxx = SID of active users	
AV info	Recognize if one or more of the following AV processes are running: navapsvc.exe – Norton AntiVirus Auto-Protect Service ccSvcHst.exe – Norton zhudongfangyu.exe – 360 KAVsvc.exe – Kaspersky Anti-Virus application RAVMonD.exe – Rising RealTime monitor Avp.exe – old Kapersky AV TmPfw.exe – Trend Micro NOD32ekrn.exe – NOD32 avguard.exe – Avira Free Antivirus	

Table 9: Details of the gathered system information.

Command ID	Action
0	Enumerate all current processes and get the process info. Gets the number of currently running processes. For each process the following information is recorded: • Process name. If the process can be opened, open the process and get the name using GetModuleFilename(). Otherwise, get it from the SYSTEM_PROCESS_INFO structure (https://msdn.microsoft.com/en-us/library/windows/desktop/ms725506%28v=vs.85%29.aspx). • SID of the user that started the process • Process ID • A reserved DWORD from offset 50h in the structure _SYSTEM_PROCESS_INFORMATION
1	Kill a process by process ID

Table 10: PCC_SYS process-related commands.

Command ID	Action
0	Enumerate all services and get their info. Gets the number of services on the system. For each service, all the information about a service stored in a ENUM_SERVICE_STATUS structure (https://msdn.microsoft.com/en-us/library/windows/desktop/ms682651%28v=vs.85%29.aspx) is recorded. A copy of the info is provided below: • Service name • Service display name • Service type • Current state of service • Control code accepted by the service • Error code the service uses to report start or stop errors • Service-specific error code for start/stop errors • Check-point value for the service progress • Estimated wait time for pending start/stop/pause/continue operations
1	Starts a service
2	Stops a service
3	Deletes a service

Table 11: PCC_SYS service-related commands.

Command ID	Action
0	Enumerate all subkeys and values for a given registry key. The number of registry subkeys as well as the number of registry values are recorded. For a subkey, only the subkey name is recorded. For a registry value, the following is recorded: • Registry value name • Registry value type code • Registry value data size • Registry value data
1	Create a registry subkey
2	Do nothing
3	Delete a subkey and all its descendants
4	Set registry value for a key
5	Replace/overwrite a registry value
6	Delete a registry value

Table 12: PCC_SYS registry-related commands.

years since it was present in the earliest Derusbi samples. The recent commands and their IDs are listed in Table 13.

Command ID	Action
1	Clean up class data
2	Enumerate all drives
3	Find file
5	Rename file
6	Delete file
7	Save a file to system
8	Recursively enumerate directory
Ah	Copy file
Bh	Move file
Ch	Start new process
Dh	Recursively enumerate directory
Eh	Recursively enumerate directory
Fh	Recursively enumerate all drives

Table 13: Recent commands and their IDs.

CONCLUSION

After reviewing all the samples we have collected since 2011, we can conclude that Derusbi has successfully laid down a solid framework for conducting covert espionage operations. The low and slow sample forensic recovery rate of Derusbi relative to other related malware families (Sakula or packed RATs) indicates that the operation was carefully planned. Derusbi is an important piece of malware that was not meant to be recovered. Through reversing the class structure, we can gain a better understanding of the traffic pattern, especially the raw packet structure, which can be used as a powerful weapon against Derusbi operations. Meanwhile, any form of behavioural or signature-based detection has been proven to be weak against the Derusbi family and the upkeep is high because the lack of samples. As Derusbi has already hit the headlines and played a vital part in a number of fruitful information breach operations, we believe the adversary will continue to invest more resources in continuing the development of Derusbi.

REFERENCES

[1] Reuters. http://af.reuters.com/article/energyOilNews/idAFL3E7KJ04120110919.

[2] Mistsubishi Heavy Industry Cyberattack TSPY_DERUSBI.A (translated). IXoXI Blog. https://ixoxi.wordpress.com/2011/10/16/.%E4%B8%89%E8%8F%B1%E9%87%8D%E5%B7%A5%E3%82%B5%E3%82%A4%E3%83%90%E3%83%BC%E6%94%BB%E6%92%83-%E3%82%B9%E3%83%91%E3%82%A4%E3%82%A6%E3%82%A7%E3%82%A2%E3%83%BB%E3%82%A6%E3%82%A4%E3%83%AB%E3%82%B9-derusbi/

[3] RSA Incident Response. (2014). Emerging Threat Profile: Shell_Crew. RSA. http://www.emc.com/collateral/white-papers/h12756-wp-shell-crew.pdf.

[4] Deep Panda Intellgence Team Report ver1.0. CrowdStrike. https://www.documentcloud.org/documents/2084641-crowdstrike-deep-panda-report.html.

[5] Vinton, K. Forbes. http://www.forbes.com/sites/katevinton/2015/05/20/data-belonging-to-1-1-million-carefirst-customers-stolen-in-cyber-attack/.

[6] ThreadConnect. http://www.threatconnect.com/news/the-anthem-hack-all-roads-lead-to-china/.

[7] Derusbi (Server Variant) Analysis. http://www.novetta.com/wp-content/uploads/2014/11/Derusbi.pdf.

APPENDIX A – URL FOR BLENDING IN NORMAL TRAFFIC OF THE ATTACK TARGET

The following list consists of websites that resemble URLs that would be consistent with the compromised machines' normal traffic patterns.

URL	Description
sheisme42.jetos.com:80	Flight management system
www.deltekinfo.com:443	SHANGHAI MEICHENG Technology Information Development
moinia.eicp.net:443	Health care
proxy.pl.abb.com:8080	Automation and power company

APPENDIX B – A LIST OF TARGETED SERVICES DERUSBI INTENDS TO PATCH

Service name	Original DLL
stisvc	%SystemRoot%\\System32\\wiaservc.dll
wuauserv	%systemroot%\\system32\\wuaueng.dll
helpsvc	%WINDIR%\PCHealth\HelpCtr\Binaries\pchsvc.dll
iphlpsvc	%SystemRoot%\System32\iphlpsvc.dll
dnscfghlp	dnscfghlp.dll
iphlpsvc	%SystemRoot%\System32\iphlpsvc.dll