PITOU

The "silent" resurrection of the notorious Srizbi kernel spambot

TLP: WHITE

CONTENTS

We began monitoring the development of a mysterious malware that first emerged in early April 2014 when we noticed some intriguing features in the threat's technical aspects. Further analysis revealed a close link to an old threat known as **Srizbi**, which infected machines and used them to send out spam email messages (in other words, a spambot). The new threat has the same general purpose - to infect a machine, download the necessary data from a command and control (C&C) server to create spam email messages, and then send the spam out using the machine - but the methods it uses differ notably.

Due to extensive changes in the new malware's code that made this latest distinctly separate from the older Srizbi variants, we named this new threat **Pitou**. In this whitepaper, we outline Pitou's distribution methods, the kernel payload delivered by its droppers, how its bootkit functions and how it communicates with its C&C server.

F-SECURE LABS SECURITY RESPONSE

Malware Analysis Whitepaper

F-Secure

INTRODUCTION

We have been monitoring the development of a mysterious malware family that first emerged in early April 2014. This malware caught our attention when we noted the following details:

I. The malware's dropper is well designed to check the Microsoft Windows operating system (OS) version it is executing on before selecting one of two different payload dropping mechanisms. This is explored further in the Droppers section

II. The payload is a kernel-mode driver (referred to as the *kernel payload* in the rest of this document) that is is protected and obfuscated by virtual machine (VM) code, which cannot be executed natively by Windows. The VM code is a series of byte code that requires an interpreter to translate it to native machine code that Windows can understand. In other words, this byte code cannot be disassembled by a common disassembler tool. This can be very effective at protecting the malware code, by preventing researchers from understanding the program's functionality, or at least increasing the analysis time needed to investigate the program

III. Some vendors identified the kernel payload using the detection name **Turla** (also known as **Uroburos**). Though the use of the name piqued our attention, we were unable to find any relation between the known espionage-related rootkit and the new malware

After further investigations into the kernel payload, we identified it as a new variant of **Srizbi**, a well-known kernel spam bot back in 2008 [1] . Certain functionalities of the kernel payload remain relatively unchanged, which eventually lead us to determine its close relation to Srizbi. Rather than reusing the old name however, we dubbed this variant **Pitou**; we believe it deserves a new name because the malware code has been completely rewritten with more robust features, including now being equipped with a bootkit.

SRIZBI VS PITOU

During our analysis, we noted the following similarities between Srizbi and Pitou, which first suggested to us that Pitou is tied to Srizbi:

- The kernel payload attempts to hide "docker19.sys", a legacy kernel-mode driver filename used by Srizbi variants
- The kernel payload for both Srizbi and Pitou has the ability to upload kernel-mode crash dump file (Minidump) to the command-and-control (C&C) server
- The kernel payload also contains Domain Generation Algorithm (DGA) code (discussed further in the DGA section)
- The main objective of the kernel payload is to spread spam email messages. Investigation of the spam emails sent by Pitou suggest that they are related to Internet pharmacy websites, which is in line with Srizbi's goal

Due to the similarities, we were convinced that this new malware was a revival of Srizbi spambot. One obvious difference between Pitou and Srizbi however is that according to our telemetry systems, Pitou is not yet anywhere near as widely propagate as its predecessor (at least at the time of writing). It is possible that the malware is still in the 'brewing' stage as we realize that the C&C server used by the earlier versions of the kernel payload (found in the early April 2014) now no longer respond to the bots when contacted, even though the C&C server is active.

In addition, starting in April 2014 and within a period of two months, we found three different versions of the dropper Pitou uses to deliver the kernel payload. Among these three updates, we found a couple of code changes in the droppers, as well as in the kernel payload. In considering the kernel payload's code, there are multiple software interrupt (INT3) code instruction placed in some functional code block, as well as debug messages that are most likely used for debugging purposes.

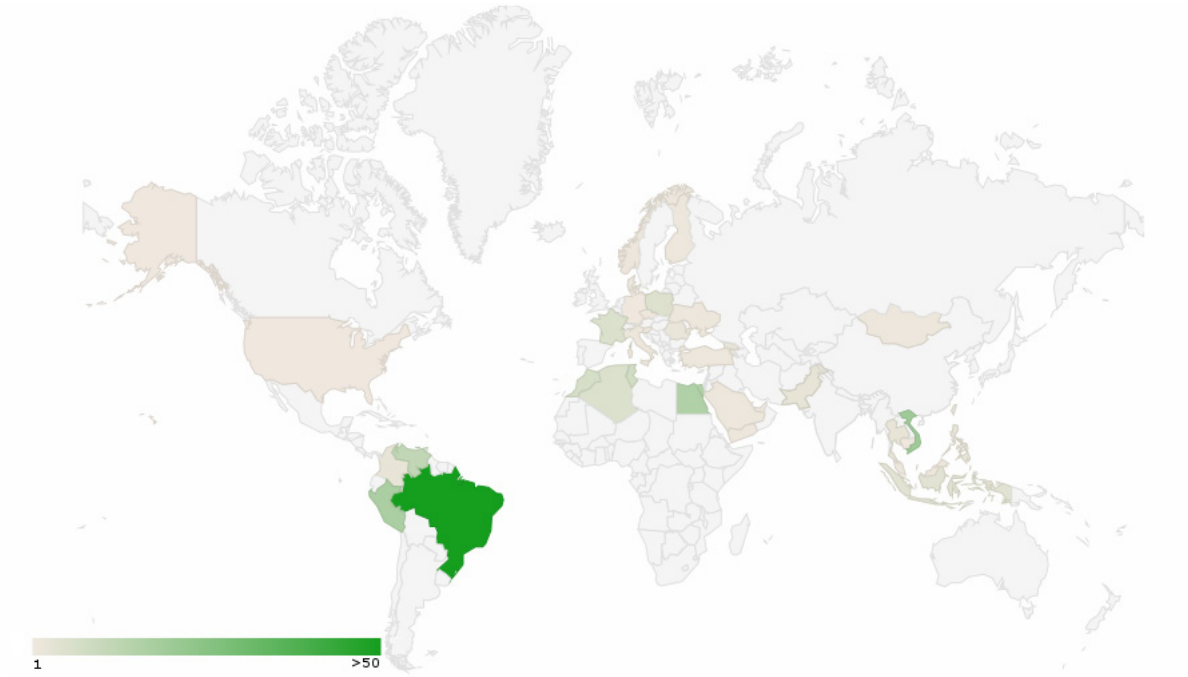


IMAGE 1
Visualization of Pitou-related detections reported (May to Aug 2014)

INFECTION VECTOR

At the time of writing, we have not yet directly seen how Pitou arrives on a compromised machine. Based on the findings of our automated analytical systems however, that the droppers seem to have been distributed in two separate 'batches', using two different distribution channels.

From April to May, the droppers were distributed to victim machines via drive-by download from exploited websites. Later on, from June to July, the distribution was changed so that the droppers were included as part of the payload of different, in-the-wild trojan-downloaders. Based on data from our telemetry systems, **IMAGE 1** is a visualization of countries where detections for Pitou-related malware was reported from May to August 2014. **DIAGRAM 1** (overleaf) shows a timeline of the dropper's distributions based on the reports from our analytical systems, along with the SHA1s for the dropper files and their corresponding kernel payloads.

During deeper, manual analysis of the malware, we manage to trace a spam email message to a victim machine, and discovered that it included as an attachment a malicious, exploit-embedded PDF document named "invoice 449104.pdf", if the PDF was opened by an unsuspecting user and their machine is successfully exploited, the trojan-downloader Upatre is downloaded and installed on the compromised machine (**IMAGE 2**).

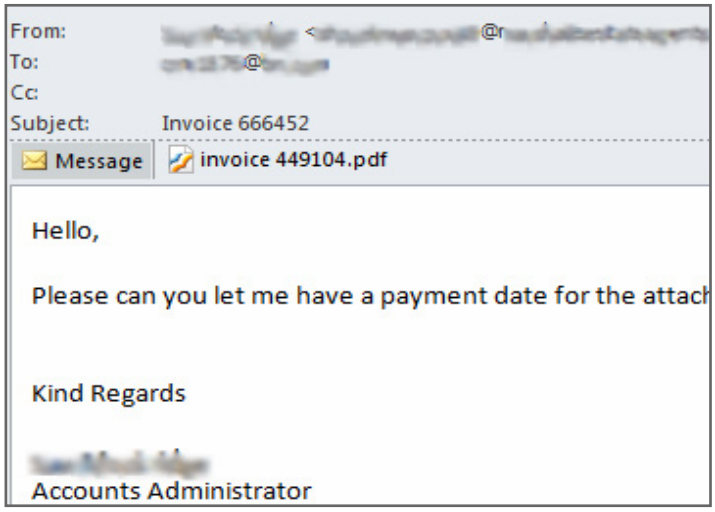
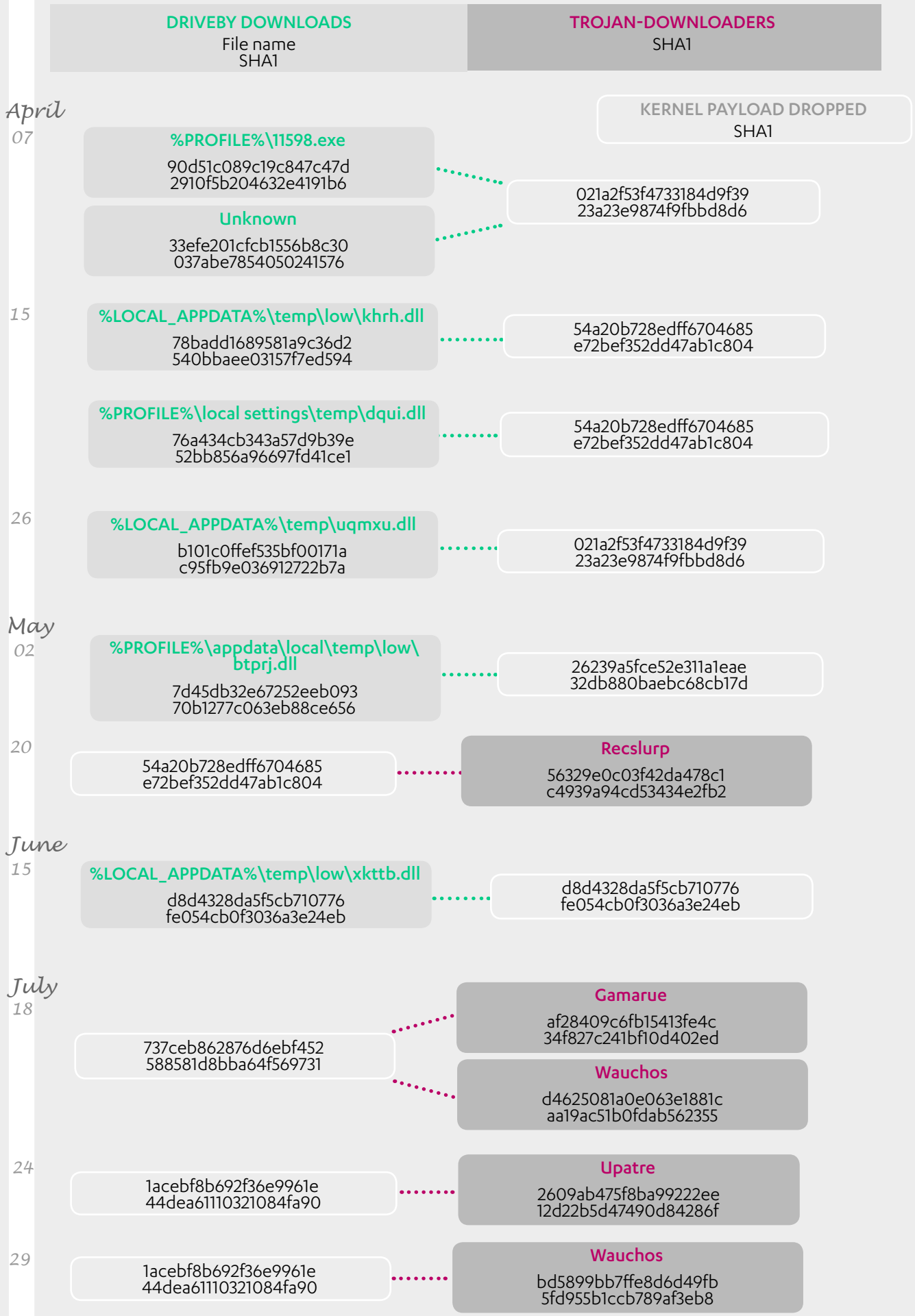


IMAGE 2
Email spam with exploited PDF delivers Upatre downloader

In addition, we found other Pitou-infected machines that were also infected with such trojan-downloaders as Gamarue, Wauchos and Onkods. So though we have not directly observed these malware delivering Pitou droppers, it is highly possible that they download other malware components that eventually deliver the droppers.

DIAGRAM 1
TIMELINE OF DROPPER DISTRIBUTION



DROPPERS

At the time of writing, we have discovered three different versions of the droppers; we have analyzed only the first and latest versions that we encountered to note the differences, even if the changes are insignificant.

Generally, the droppers acts like an installer. The droppers are responsible for installing the kernel payload, which is embedded inside the dropper itself. If the dropper is executed on a Windows XP machine however (referred to as **NT5** in the rest of this document), it will extract and install the kernel payload as a Windows service (*SERVICE_KERNEL_DRIVER*) via the *CreateService* API.

Otherwise, on Windows 7 and above (referred to as **NT6** in the rest of this document), it will install its bootkit by infecting the machine's Master Boot Record (MBR) in order to load its kernel payload. By infecting the MBR, it allows the kernel payload to be loaded during Windows' boot process without violating the Kernel Mode Code Signing (KMCS) policy. Under the KMCS policy, only a digitally signed kernel-mode driver is allowed to be executed on NT6, which is fully enforced on the 64-bit platform.

Much like other malware, the droppers can be packed with different obfuscators, such as the Visual Basic (VB) obfuscator we saw in one of the earlier dropper versions. After unpacking the obfuscator code, we can see the droppers utilizing multiple code obfuscation techniques, including (but not limited to) resolving Windows API during runtime, decoding encoded strings based on a shuffled jump table and branching a function into multiple code blocks with multiple jump instructions to make static code analysis harder.

EXPLORER.EXE CODE INJECTION WITH COM ELEVATION

All versions of the droppers use the same privilege escalation technique – a publicly known COM elevation technique documented by Leo Davidson [2].

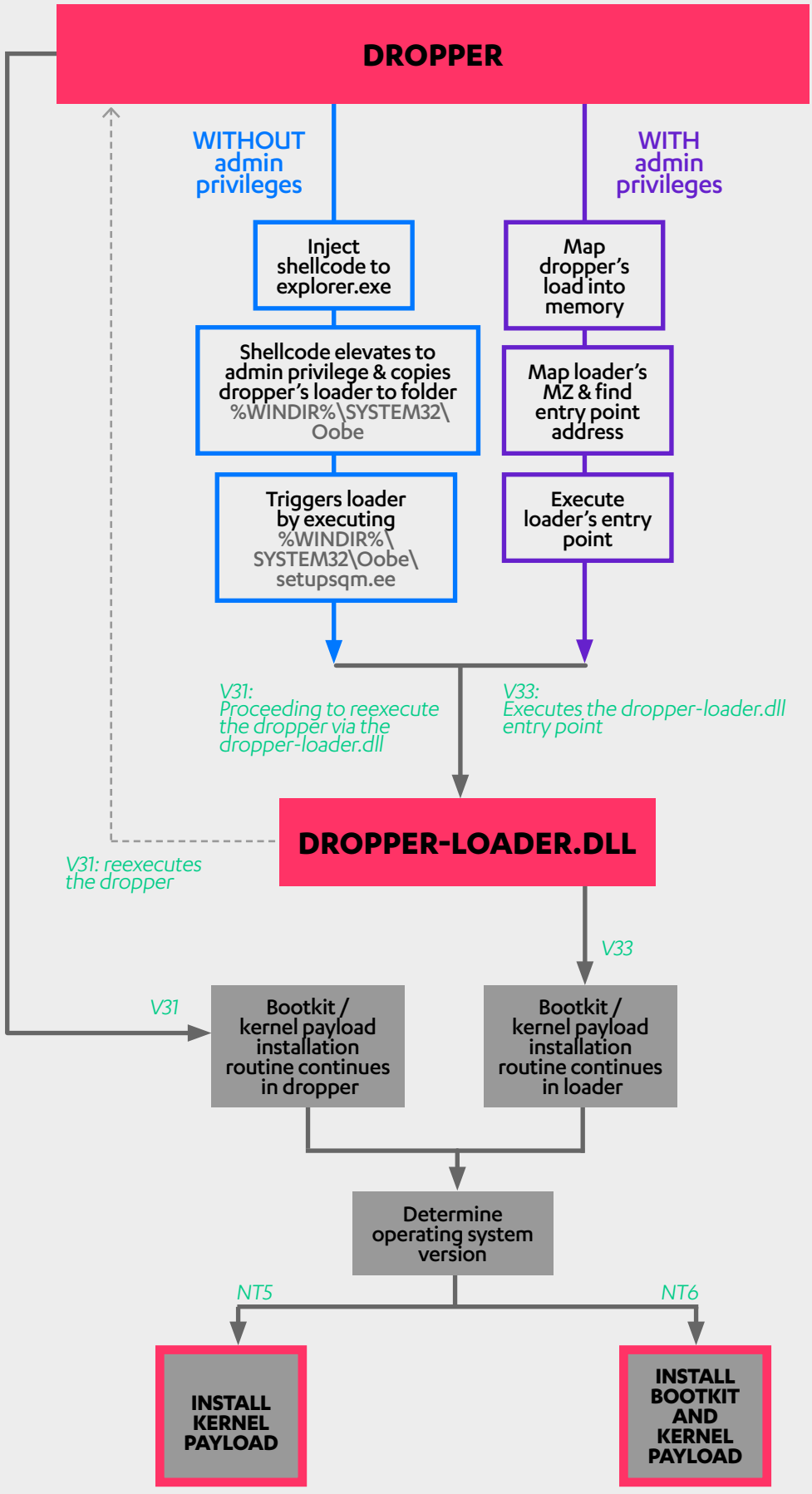
The droppers first prepare a shellcode that will be injected to the explorer.exe process. The shellcode then attempts to escalate the privilege of explorer.exe process by leveraging the COM elevation trick. Once the elevation code execution is done, the shellcode running in the explorer.exe process context with elevated privilege will drop a DLL file into **%WINDIR%\SYSTEM32\Oobe** using the filename **wdscore.dll**. This DLL is also one of the components embedded in the dropper, and is referred to as the dropper's loader for the rest of this document.

Afterwards, the shellcode triggers an executable under **%WINDIR%\SYSTEM32\Oobe\setupsqm.exe** via ShellExecuteEx that is supposed to load one of its DLL dependencies, **wdscore.dll** (found in the %WINDIR%\SYSTEM32 directory). Because of the nature of Windows' DLL search order [3] however, the dropper's loader will be loaded and executed first. This is also a well-known technique, "DLL hijacking", abused by many other malware.

When the dropper's loader gets control, depending on the version of the droppers, the actual kernel payload installation will take place. Basically, three different version of the dropper are identified as **v31**, **v32** and **v33** respectively by the author. For v31, the actual kernel payload installation routine can be found in the droppers, whereas from v33, the kernel payload installation routine has been shifted to the loader.

DIAGRAM 2 (overleaf) depicts the dropper's overall installation workflow.

DIAGRAM 2:
KERNEL PAYLOAD INSTALLATION WORKFLOW



KERNEL PAYLOAD FILE NAME GENERATION

On NT5, the kernel payload will be installed as a kernel system driver on the infected machine. The droppers generate a unique filename for the kernel payload on each infected machine. The droppers fabricate an exclusive identifier using the following data from the infected machine:

I. Get disk drive's serial number and model name via *I/O control code SMART_RCV_DRIVE_DATA*, which is available only from driver with SMART supported

II. Get ProductID from the registry *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion*

III. Get InstallDate from the registry *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion*

IV. Generate 16-byte hexadecimal value from the retrieved data. A filename with a minimum filename length of four will be generated based on the 16-byte hexadecimal value

V. The resulting filename can be expressed using the regular expression:

[a-z]{4,7}[0-9]?.sys

On NT6, the droppers write the bootkit's MBR and kernel payload to the raw disk via SCSI Pass Through Interface (SPTI). Hence these files cannot be navigated or viewed directly using regular Windows Explorer. Later on in the Bootkit section, we will discuss the kernel payload loading process carried out by the bootkit's MBR.

SIGNS OF INFECTION

KERNEL PAYLOD DEVICE OBJECT NAME

On NT5, once the kernel payload has been successfully installed, the kernel payload will create a corresponding device object using a specially crafted device name, consisting of an alphabet from a-p, based on the disk device's vendor string retrieved via I/O control code *IOCTL_STORAGE_QUERY_PROPERTY*.

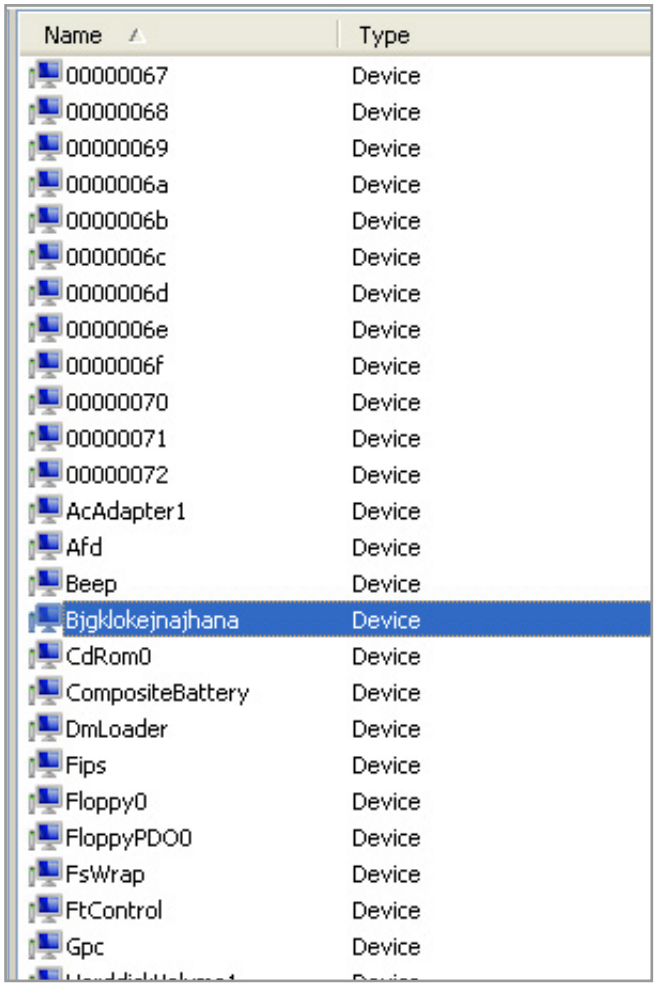


IMAGE 3
A specially crafted kernel payload's device name can be inspected using WinObj

On NT6, no device object will be created.

TABLE 1
CRASHDUMPENABLED DEFAULT VALUE ON DIFFERENT OPERATING SYSTEMS

OPERATING SYSTEM	DEFAULT VALUE	ACTION
Windows XP	3	Write small memory dump to file
Windows 7	2	Write kernel memory dump to file
Windows 8/8.1	7	Automatic memory dump

TABLE 2
HIJACKED SYSTEM MODULES AND FUNCTIONS IN BOOT PROCESS

TARGETED SYSTEM MODULES	HOOKED FUNCTION
BOOTMGR	Archx86TransferTo32BitApplicationAsm/ Archx86TransferTo64BitApplicationAsm
WINLOAD.EXE	OslArchTransferToKernel
NTOSKRNL.EXE	InbvIsBootDriverInstalled or ExNotifyCallback

MINIDUMP/CRASHDUMP ENABLER VIA REGISTRY

Pitou is a full-kernel mode malware that performs all of its tasks from kernel-mode, including its network operations; spreading spam email messages. Because Windows does not provide handy APIs to let kernel-mode code interact with the physical network adapter to carry out network activities directly, the kernel payload installs multiple hooks on the lowest-level Windows network driver - NDIS.

Kernel-mode hook is known to cause stability issues if it is not properly implemented. When Windows encounters a kernel-mode code error, it forces the machine to reboot and show the infamous blue screen, also known as the Blue Screen of Death (BSOD).

Because of this, the malware author needs to keep track of the crashes that might be caused by his work. Thus the kernel payload will set CrashDumpEnabled to value 3 (meaning Windows will write small memory dump to a file in the event of a BSOD) from Windows registry key under all control sets:

I. HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\CrashControl

II. HKEY_LOCAL_MACHINE\SYSTEM\ControlSet002\Control\CrashControl

III. HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl

Windows has pre-configured CrashDumpEnabled to default value depends on the OS version [4], (listed in **TABLE 1**). Hence the CrashDumpEnabled registry modification can be used as an indicator to tell if the machine has been infected, especially on Windows 7 or 8.

BOOTKIT

The bootkit's purpose is not only to provide the kernel payload persistency upon restarting the infected machine, it also can be used to bypass the KMCS to allow a malware to execute its unsigned kernel-mode driver.

Similar to other classic bootkits in the wild, Pitou's bootkit hijacks the BIOS interrupt handler (INT 13h) to monitor all read operation to the disk's sectors. The hooked interrupt handler will set up hooks on system modules involved in the boot process in order to intercept Windows boot sequence. **TABLE 2** shows the system modules targeted by the bootkit and the corresponding function that will be hooked.

To ensure the hook's compatibility with the latest OS, Pitou's bootkit picks different target functions - *InbvIsBootDriverInstalled* or *ExNotifyCallback* from notskrnl.exe - for hooks, rather than *IoInitSystem*, which is the favored target of more conventional bootkits.

When the hooked handler of *OslArchTransferToKernel* kicks in, at this point the ntoskrnl.exe image has been loaded into memory, and Pitou's bootkit starts parsing the ntoskrnl.exe image to look for the INIT image section. Once the INIT section has been found, it searches for the CALL instruction to function *InbvIsBootDriverInstalled* or *ExNotifyCallback* within the INIT section. Based on our analysis, the export function *InbvIsBootDriverInstalled* is referenced by code in the INIT section on Windows 8, while *ExNotifyCallback* is referenced by code in the INIT section on Windows 7. In other words, the bootkit will set up a hook on the CALL instruction to *InbvIsBootDriverInstalled* for Windows 8, or on the CALL instruction to *ExNotifyCallback* for Windows 7.

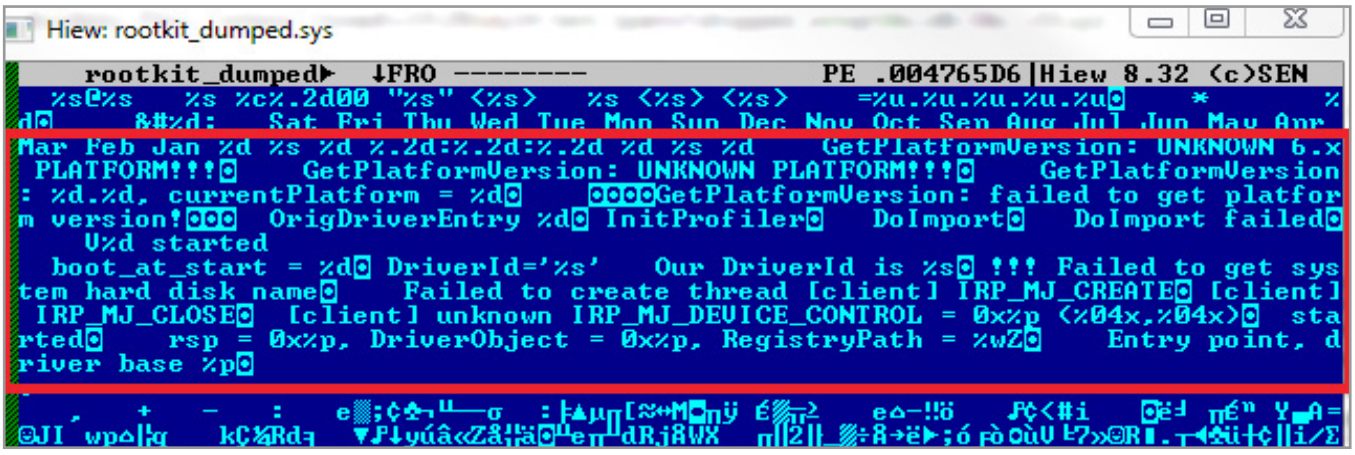


IMAGE 4
Debug string from kernel payload

After the ntoskrnl.exe hook handler gets control, the bootkit resolves the necessary function APIs by traversing the export table from ntoskrnl.exe, calculating the export function name in hash and comparing it against the hash of the desired function. Afterwards, the bootkit creates a system thread via *PsCreateSystemThread* that is responsible for loading the kernel payload by reading from the raw disk's sector via *ZwOpenFile* and *ZwReadFile* and eventually executing the kernel payload's entry point. At the same time, the bootkit will also restore the previous hooked code in the ntoskrnl.exe image and resume execution from the hijacked code, so that Windows will boot normally.

We tested the bookits on desktop systems running Windows 8.1 (32-bit and 64-bit) with a legacy BIOS and Unified Extensible Firmware Interface (UEFI). Our results show that the bootkit only works on the legacy BIOS; systems using UEFI will not be affected.

KERNEL PAYLOAD

DEBUG STRING

The kernel payload supports both 32-bit and 64-bit platforms. Both kernel payloads are also embedded in the droppers. Based on the findings from our sample collection systems, at the time of writing, we found three different version of kernel payloads, namely v31, v32, and v33. This version number is part of the encoded debug string from the binary.

COMPILATION TIMESTAMP

Based on our analysis of the samples, it seems that the compilation date of the samples are not synchronized with the version number that we see from the debug string. For example, a v33's kernel payload (SHA1: 1acebf8b692f36e9961e44dea61110321084fa90) has the compilation date 11 August 2013, while our sample collection indicates the sample was first found on 24 July 2014. However on 7 April 2014, we encountered the first kernel payload (SHA1: 021a2f53f4733184d9f3923a23e9874f9fbbd8d6) with a compilation date of 21 Jan 2014. This indicates that the authors has tampered with the compilation date to mislead researchers.

SANDBOX DETECTION

To keep the kernel payload from being reviewed by a sandboxed analysis system, the kernel payload includes a couple sandbox detection routines. The sandbox detection routine has been improving since the first version of the kernel payload. Some of the sandbox detection routine is protected by VM code. In general, the sandbox detection routine is as follows:

I. Check the existence of sandbox name from the registry key, *SYSTEM\CurrentControlSet\Services\Disk\Enum*.

II. Look for sandbox's kernel module name. For example in VMware, it looks for the following VMware modules:

   a. vmx_svga.sys
   b. vmx_fb.sys
   c. vmxnet.sys
   d. other vmware kernel module name

TABLE 3
SUMMARY OF IRP AND FUNCTION HOOKS FOR HIDING FILES AND REGISTRY

OS	FILES	REGISTRY
NT5	IRP_MJ_DIRECTORY_CONTROL, register filesystem notification callback routine via IoRegisterFsRegistrationChange	NtOpenKey, NtEnumerateKey
NT6	IRP_MJ_DEVICE_CONTROL, IRP_MJ_INTERNAL_DEVICE/ IRP_MJ_SCSI	-

TABLE 4
HIJACKED NDIS HANDLERS IN DIFFERENT VERSIONS OF NDIS DRIVER

DATA STRUCTURE	NDIS 5.X	NDIS 6.X
NDIS_ PROTOCOL_ BLOCK	SendCompleteHandler, TransferDataCompleteHandler, ReceiveHandler, StatusHandler	-
NDIS_OPEN_ BLOCK	SendCompleteHandler, TransferDataCompleteHandler, ReceiveHandler, ReceivePacketHandler, StatusHandler	ReceiveNetBufferLists, StatusHandler
NDIS_M_ DRIVER_ BLOCK	HaltHandler	HaltHandlerEx, PauseHandler, ResetHandlerEx, ShutdownHandlerEx
NDIS_ MINIPORT_ BLOCK	ResetCompleteHandler	-

III. Check the disk device model name using *I/O control code, IOCTL_STORAGE_QUERY_ PROPERTY*

IV. Check CPU tick count ratio via RDTSC

V. Since v33, it has included a new sandbox detection, checkForVmBios. As the function name implies, this routine checks the sandbox name from the BIOS memory. The routine calls MmMapIoSpace function to read the BIOS content located at physical memory at offset 0xE0000 and then start traversing the BIOS memory content.

The results of the sandbox check will be sent to the C&C server and the server will determine if the bot is running inside a sandbox. If that is the case, the C&C server will refuse to respond to the bot with commands to perform further actions, which in turn hides the main purpose of the kernel payload.

FILES, REGISTRY, AND BOOTKIT HIDING

TABLE 3 contains a summary of I/O Request Packets (IRP) or function routines that are hooked by the kernel payload in order to hide files, the infected MBR and the registry, depending on OS version. These are typical tactics used by modern rootkits; they can easily be detected by common rootkit detectors and removed by anti-malware tools.

NDIS HIJACKING

This is the core part of the kernel payload that plays a very important role as a spambot. Pitou's NDIS hijacking has improved significantly, such that it also supports NDIS handlers hooking on NDIS 6.X. Clearly, one of the benefits of hijacking the NDIS layer is to bypass software firewall in order to allow inbound and outbound connections without interception.

Pitou is compatible with Windows 7/8 (32-bit and 64-bit), though some of the implementations still remain the same - for example, the private DNS protocol, the way it finds the correct network adapter to be hooked on and also the way how it forwards the dedicated packets based on the server's port number to its own private TCP/IP stack [5].

In order to be able to intercept received packets, Pitou traverses a network protocol list, which can be obtained from the *ndisProtocolList* NDIS driver, and looks for the desired NDIS handler functions from network protocols like PSCHED, PACER, TCPIP, TCPIP_WANARP and WANARP. **TABLE 4** shows the different hooked NDIS handlers on NDIS 5.X and NDIS 6.X.

Notice that there is no send packets related handler hooked. The kernel payload uses *ndisMSendPacketsX* (on NT5) or *nidsSendNetBufferLists* (on NT6) from NDIS library to send the information collected from the infected machine, for example the sandbox check

result, to the C&C server as in encrypted TCP data. When the server responds to the bot with an encrypted TCP payload, the bot manage to intercept and filter the TCP payload first via the NDIS *ReceiveHandler* hook before it can be processed by the original NDIS ReceiveHandler hook.

The contacted server address is the primary C&C server. This primary C&C server address and port number are hardcoded and encoded in the binary (see **APPENDIX B**).

DOMAIN GENERATION ALGORITHM (DGA)

When the bot fails to contact the primary C&C server in ten minutes, the bot will fall back to the C&C address generated via the DGA. It is trivial for researchers to recover this kind of algorithm and take control of the domain name generated by the DGA. The author has however taken one step further to protect the DGA with VM code. Thus recovering Pitou's DGA becomes very challenging, though not impossible.

The kernel payload first retrieves the current date (day, month and year) using *ExSystemTimeToLocalTime* and *RtlTimeToTimeFields*. The DGA will start generating a default DWORD seed value based on the current date. A detailed explanation of the DGA algorithm is beyond the scope of this document. In short, the DWORD seed value will be used as an index of an alphabet to be retrieved from a list of consonants (bcdfghjklmnpqrstvwxyz) and vowels (aeiou). The same DWORD seed value will also be used as an index to determine which top-level domain name (TLD) to be selected from this list of TLD: com,org,biz,net,info,mobi,us,name,me. A total of 16 different DGA domain names can be generated in one day. There is also a base XOR key that varies in different versions of the spambot - for instance, in v31, the base XOR key is 0xDAFE02D, while in v33, the base XOR key is 0xDAFE02C.

SPREADING EMAILS

At this point, the reader should already know the main objective of kernel payload - receiving email templates from the C&C server and then sending out spam email messages based on the templates. When the bot receives a response from the C&C server via TCP, the TCP data is encrypted and will be decrypted by the bot. After decryption, the TCP data appears to be a list of senders and recipients names, as well as the email addresses and SMTP relay servers to be used by the bot.

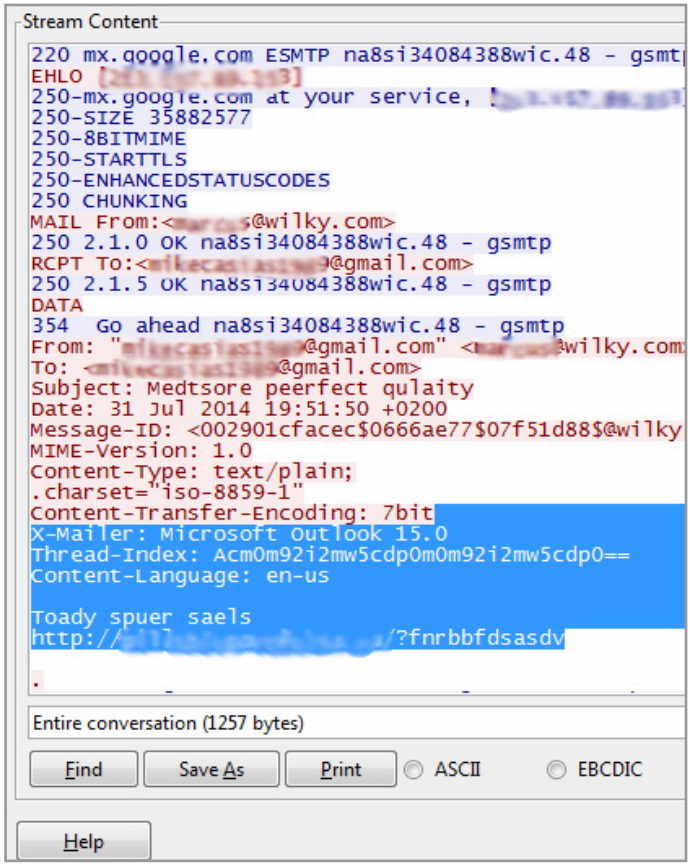


IMAGE 5
Example of spam email spread by Pitou

CONCLUSION

In summary, we have discussed the different ways Pitou can arrive on a victim's machine, as well as the telltale signs when Pitou infects the machine.

The major enhancement of this kernel spambot is clearly in making itself compatible with different Windows operating system versions, as this takes into consideration ongoing changes in the 'target audience' as it were, as more and more users switch to the latest operating system version for various reasons (one of the main reasons definitely being Windows XP's gradual fade out).

From a technical perspective however, the motive behind Pitou's code obfuscation and protection technique is clearly to consume a security researcher's analysis time before the malware's author(s) manage to push a new update.

ACKNOWLEDGEMENT

We would like to express gratitude to Chun Feng for his valuable input with regards to pointing out Pitou's bootkit compatibility on Microsoft Windows 8.1.

CHANGELOG

- **28 Aug 2014:** Original publication of whitepaper.
- **2 Sept 2014:** Minor edit to paragraph (p 2) to clarify (lack of) relationship between Pitou and Turla.

REFERENCES

1. Wikipedia; *Srizibi botnet*; http://en.wikipedia.org/wiki/Srizbi_botnet
2. Pretentious Name; Leo Davidson; *Windows 7 UAC whitelist: Code-injection Issue (and more)*; http://www.pretentiousname.com/misc/win7_uac_whitelist2.html
3. Microsoft; Security Research and Defense Blog; *Load Library Safely*; 13 May 2014; http://blogs.technet.com/b/srd/archive/2014/05/13/load-library-safely.aspx
4. Microsoft TechNet; *CrashDumpEnabled*; http://technet.microsoft.com/en-us/library/cc976050.aspx
5. Virus Bulletin; Kimmo Kasslin & Elia Florio; *Spam from the kernel*; 11 Jan 2007; https://www.virusbtn.com/virusbulletin/archive/2007/11/vb200711-srizbi

APPENDIX A: SAMPLES

DESCRIPTION	SHA1
Exploit PDF delivering Upatre downloader	c14842ba0a4adee820d12c1fce236b22814304ba
V31 droppers	33efe201cfcb1556b8c30037abe7854050241576
V31 dropper's loader (x86)	c51c1d90b2cfc2119bc408423c8239e7451612e4
V31 dropper's loader (x64)	77cbbb7d834796ed6019d72dace94ab412296aee
V33 droppers	002ba587dcb9f3bd249fde7b07a447a270a24567
V33 dropper's loader (x86)	b6d670b8290fd4ee3798ead2e557be72215554ec
V33 dropper's loader (x64)	c7a2acdc22c857a2b75eb4f251c9a6c7655d3259
V31 kernel payload (x86)	021a2f53f4733184d9f3923a23e9874f9fbbd8d6
V32 kernel payload (x86)	4965fe1e3d401fca0748479f2f69191061edb429
V33 kernel payload (x86)	1acebf8b692f36e9961e44dea61110321084fa90

APPENDIX B: C&C SERVER INFO

VERSION	C&C SERVER	PORT NUMBER
V31	195.154.231.114; ternexwestern.biz	3924
V33	5.61.39.33; rgnerignioerjg.com	5940

SWITCH ON FREEDOM

F-Secure is an online security and privacy company from Finland. We offer millions of people around the globe the power to surf invisibly and store and share stuff, safe from online threats.

We are here to fight for digital freedom.

Join the movement and switch on freedom.

Founded in 1988, F-Secure is listed on NASDAQ OMX Helsinki Ltd.

F-Secure