RTM Locker Ransomware as a Service (RaaS) Now on Linux

uptycs.com/blog/rtm-locker-ransomware-as-a-service-raas-linux

Uptycs Threat Research

The Uptycs threat research team has discovered a new ransomware binary attributed to the RTM group, a known ransomware-as-a-service (RaaS) provider. This is the first time the group has created a Linux binary. Its locker ransomware infects Linux, NAS, and ESXi hosts and appears to be inspired by Babuk ransomware's leaked source code. It uses a combination of ECDH on Curve25519 (asymmetric encryption) and Chacha20 (symmetric encryption) to encrypt files.

RTM Locker was identified during Uptycs' dark web hunting. Its malware is specifically geared toward ESXi hosts, as it contains two related commands. Its initial access vector remains unknown. Both asymmetric and symmetric encryption make it impossible to decrypt files without the attacker's private key.

Notable similarities between RTM Locker and Babuk ransomware include random number generation in addition to using ECDH in Curve25519 for asymmetric encryption. Babuk differs slightly from RTM Locker by using sosemanuk for asymmetric encryption, while RTM Locker uses ChaCha20.

The good news is that Uptycs extended detection and response (XDR) provides advanced detection capabilities and YARA rules for detecting RTM Locker malware.

FAQ

Q. How are RTM Locker and Babuk ransomware related?

It appears RTM Locker leverages leaked source code from Babuk ransomware. Both malware types use random number generation, Curve25519 implementation.

Q. How does this new ransomware infect Linux, NAS, and ESXi hosts?

The initial access vector for RTM Locker is unknown at this time.

Q. Can the encrypted files be decrypted without the attacker's private key?

Sorry, no. The combination of asymmetric and symmetric encryption makes decryption impossible without the private key.

Q. What are some unique RTM Locker features compared to other ransomware strains?

RTM Locker specifically targets ESXi hosts, contains two ESXi commands, and is the first Linux binary created by the RTM group. It is also inspired by leaked source code from Babuk ransomware.

Q. How did the Uptycs threat research team discover this threat actor's ransomware?

We identified the RTM Locker threat during our ongoing dark web hunting. Such continual research is imperative to better serve our customers.

Q. What measures can be taken to detect and mitigate RTM Locker?

Organizations can use advanced detection solutions such as Uptycs XDR. Its built-in YARA rules and other advanced detection capabilities identify and mitigate RTM Locker ransomware. To this end the Uptycs threat research team has shared a YARA rule to detect RTM Locker.

Threat Attribution

The threat group RTM Locker was discovered by the Uptycs Threat Intelligence team during our dark web hunting. Figure 1 shows the post made by the RTM group about their Locker, which targets Windows, ESXi/Linux, and NAS systems.

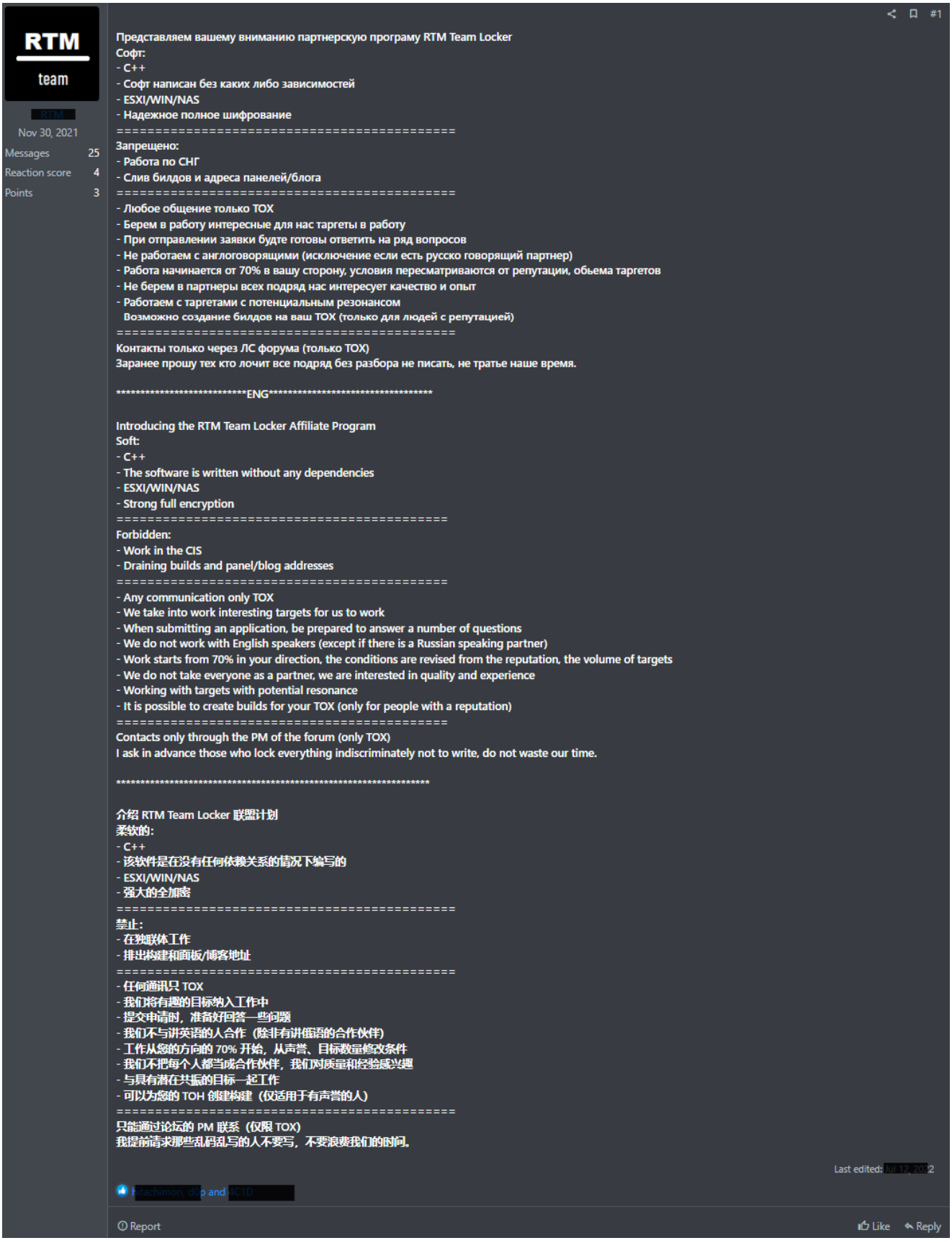

Fig. 1 - The post made by RTM group about its locker

A previous Windows version of this ransomware was reported by Trellix, in which it mentions an onion site link to contact the threat actor. This appears to have prompted the team to move to Tox. The binary for this report contains no mention of the onion site; only a Tox ID is mentioned (Figure 18).

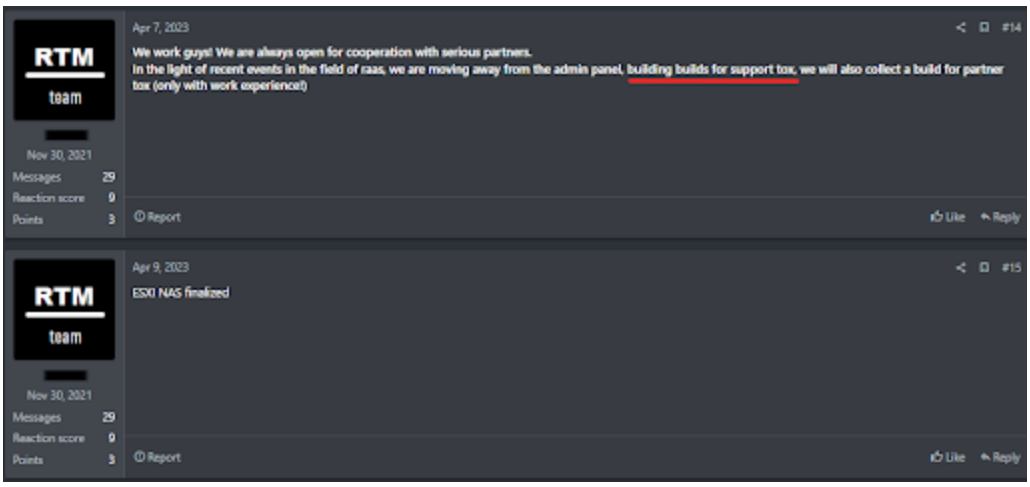

Fig. 2 - Attacker update about moving from an onion site to Tox

Technical Analysis

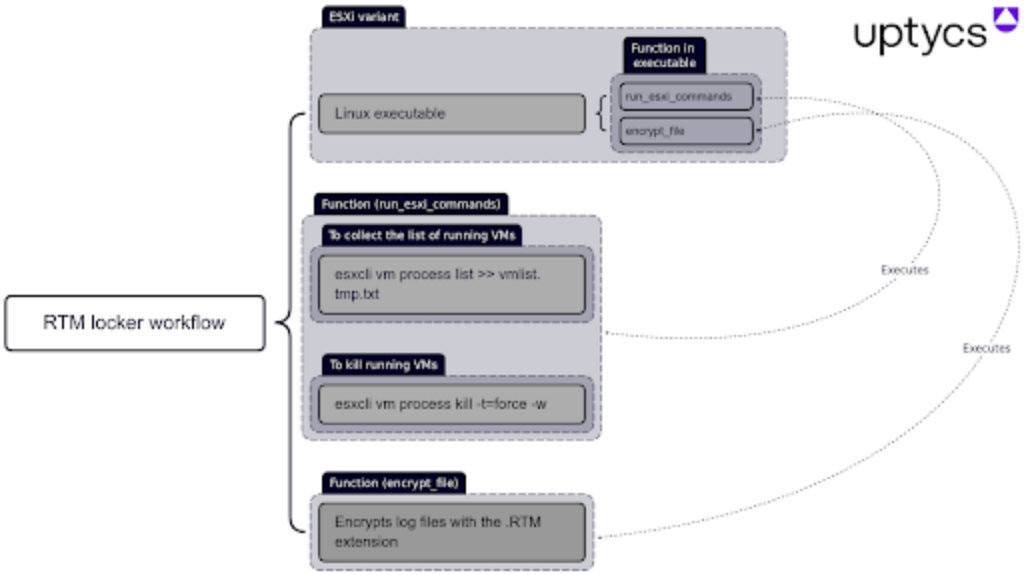

Fig. 3 - Mind map of the Linux executable

The ransomware binary seems to be geared towards ESXi, because of the two ESXi commands that were noticed at the start of the program. It is statically compiled and stripped, making reverse engineering more difficult and allowing the binary to run on more systems. The initial access vector is unknown.

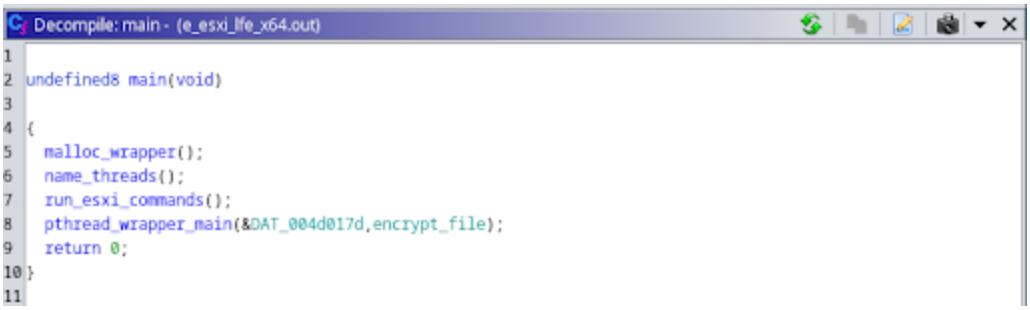

Fig. 4 - Main procedure of the ransomware

name_threads, run_esxi_commands and pthread_wrapper_main are the important functions in this binary. name_threads uses sysconf(3) with _SC_NPROCESSORS_ONLN as argument to find out the number of threads to use in the program, and calls name_thread_routine in the pthread_wrapper routine to name each thread as shown in Figure 5.

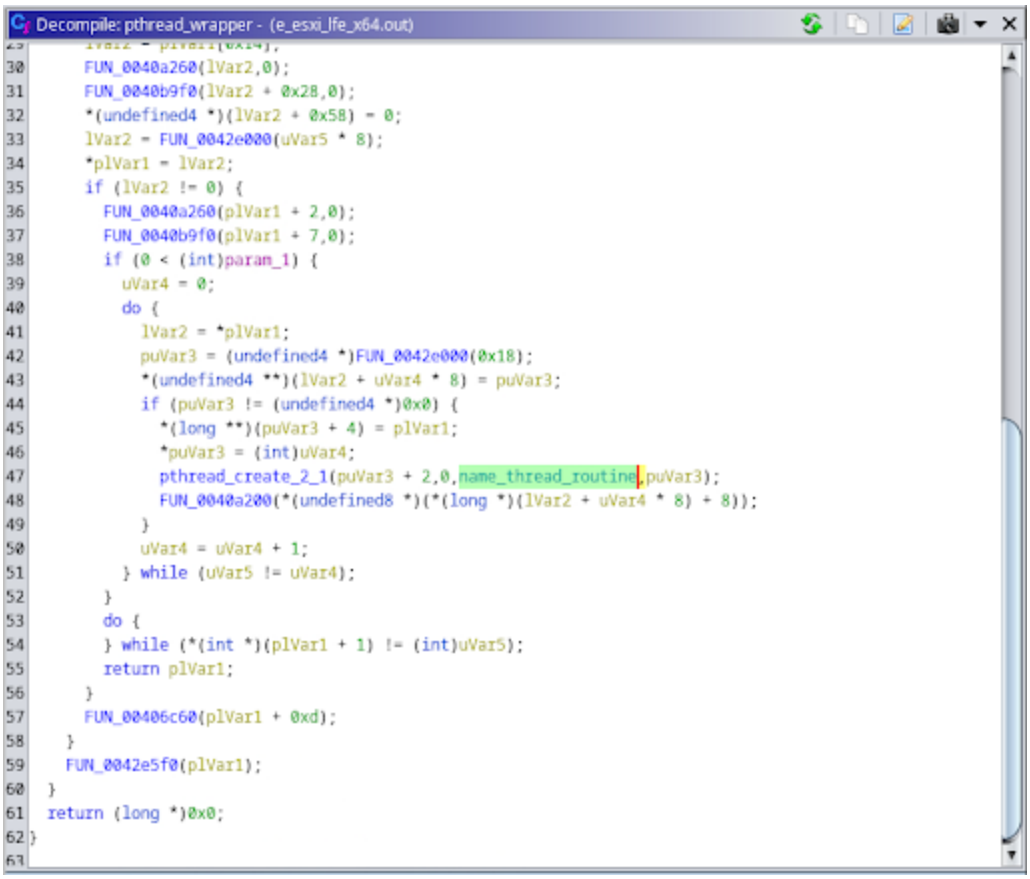

FIg. 5 - pthread_wraper calls pthread_create with name_thread_routine as an argument

name_thread_routine names each thread to use later in the encryption process. The threads are named “Thread-pool-%d”, with the decimal number representing the index of the thread. Shown in Figure 6, this is done using prctl(2) with PR_SET_NAME as its argument.

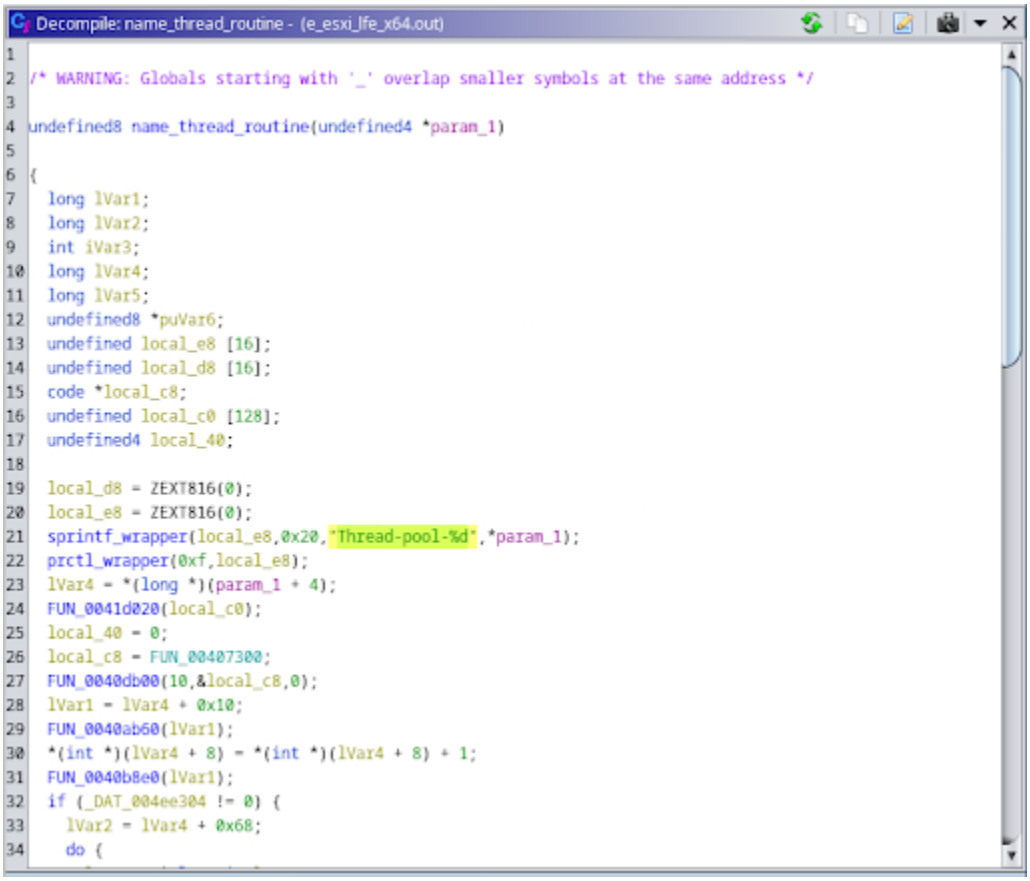

Fig. 6 - Threads being named inside name_thread_routine

After naming each thread, the run_esxi_commands routine is called. Notably, this is not called on the NAS variant of the binary, since a NAS does not run ESXi.

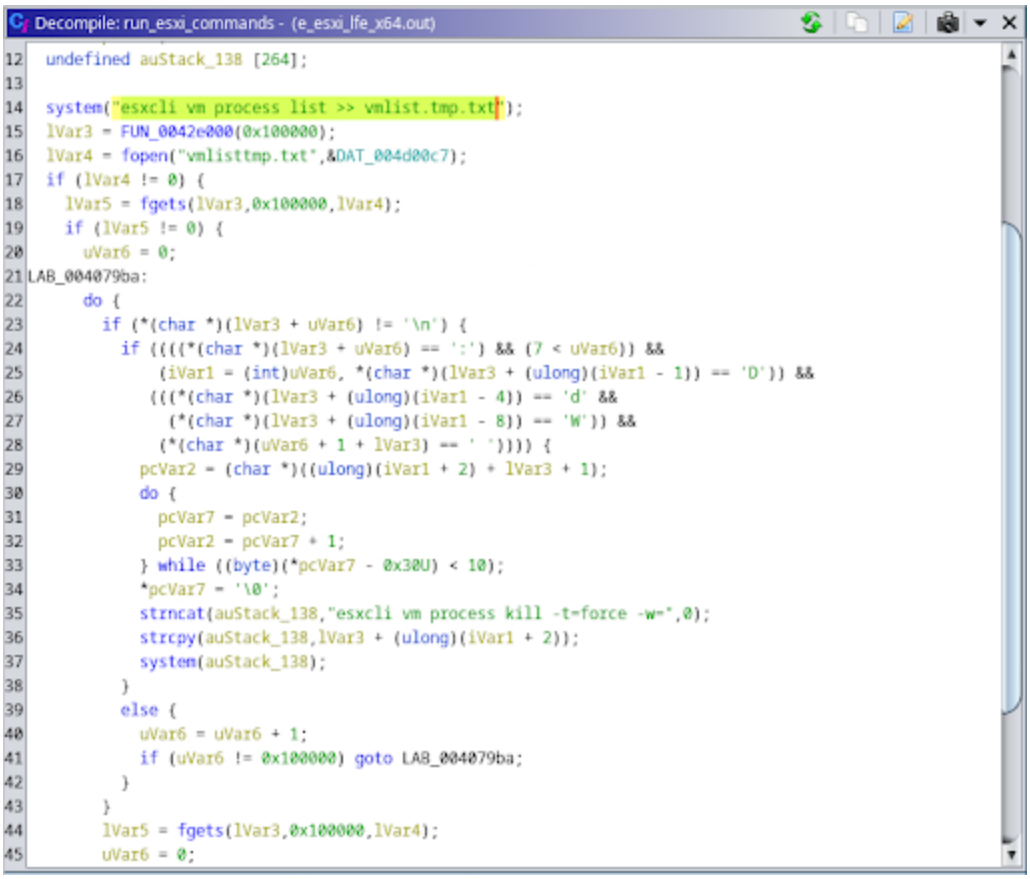

Fig. 7 - Two ESXi commands are run using this program

The two ESXi commands are:

1. “esxcli vm process list >> vmlist.tmp.txt”
   This command lists all the ESXi VMs currently running on the system.
2. “esxcli vm process kill -t=force -w”
   This command kills all the ESXi VMs that were found by the previous command

Interestingly, the file read by the program, vmlisttmp.txt, isn’t the file that it writes to(vmlist.tmp.txt). The differing filenames are a mistake made by the ransomware author, which suggests this ransomware might still be under development.

After the binary successfully kills all the running ESXi VMs, it begins the encryption routine by calling pthread_wrapper_main.

pthread_wrapper_main seems to be a custom function that calls multiple pthread commands to run the encryption process more efficiently. Figure 8 shows a snippet of FUN_00407580, a function that is used to read the entire system using opendir(3), after which it performs lstat(2) on the file descriptor and progresses through the function based on the results of the system call.

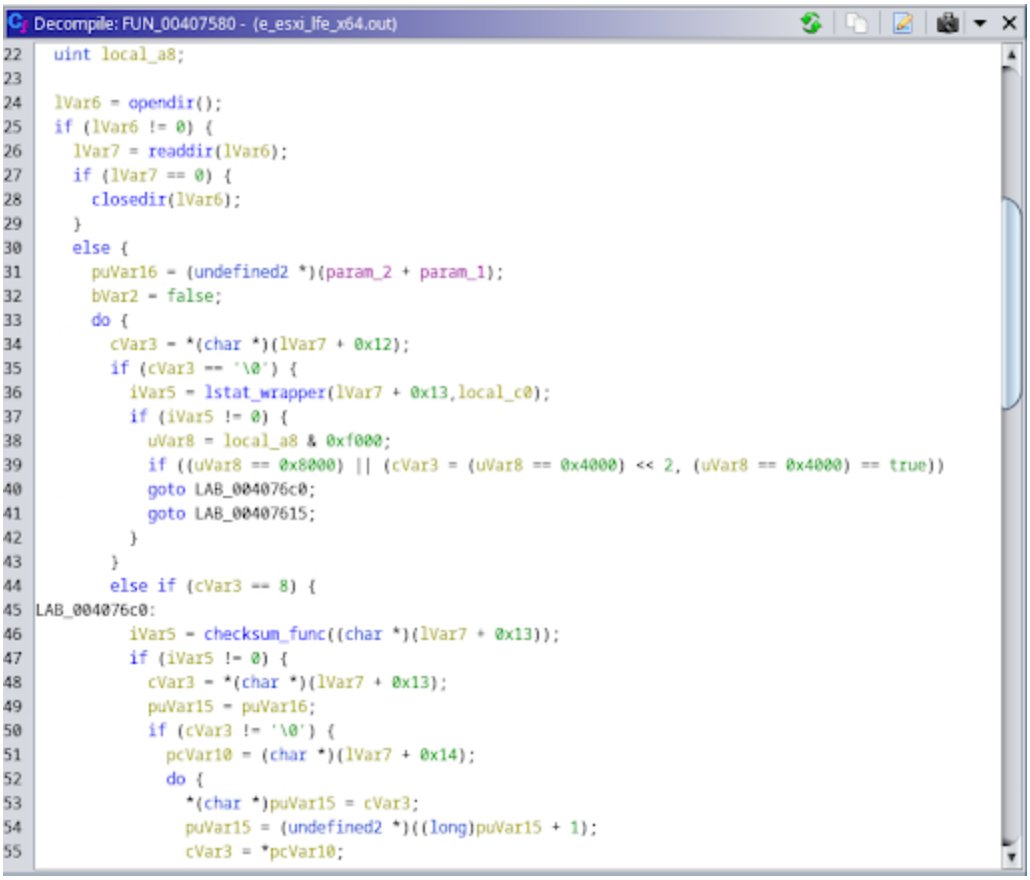

Fig. 8 - A FUN_00407580 excerpt

Two parts of this function are intriguing: 1) the call to the actual encryption routine (i.e.,encrypt_file referenced in the main function, and 2) how it finds which file to encrypt.

The lstat(2) system call returns 4 for a directory or 8 for a file. Figure 9 shows a function excerpt where a checksum is performed, after which the encrypt_file function is called. This checksum seems to only check file extensions and, like the source code that inspired it, currently works for the following extensions:

.log .vmdk .vmem .vswp .vmsn

Fig. 9 - Excerpt from FUN_00407580

The encryption function also usespthreadsto speed up execution. It obtains locks on particular threads to prevent race conditions, then runs another function that encrypts a single file.

Figure 10 shows the function called by encrypt_file. It has two constants, `expand 16-byte k` and `expand 32-byte k` related to the Salsa20/ChaCha family of ciphers. This leads us to believe the file is encrypted using the same cipher family. Figure 11 shows the constants as found in the function.

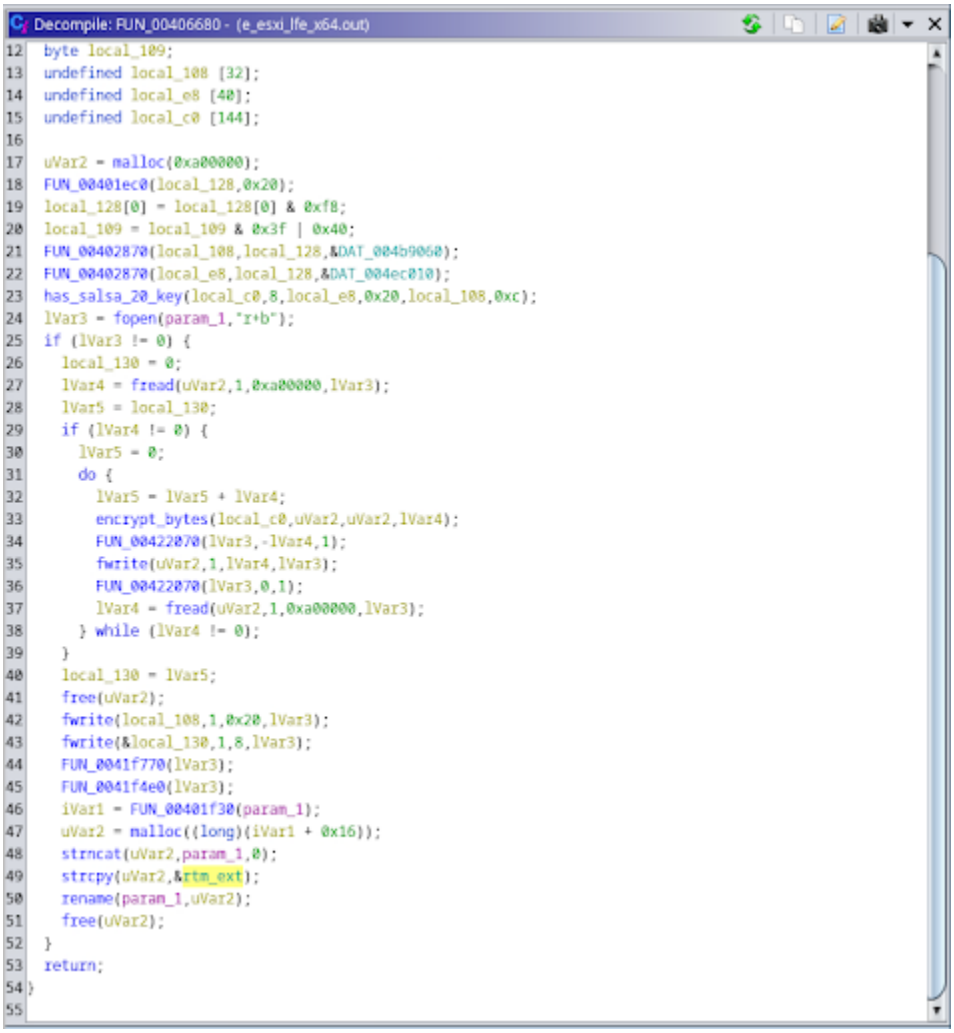

Fig. 10 - The FUN_00406680 function that encrypts a single file

The function in Figure 10 essentially encrypts a chunk of bytes read from fread(3) and writes that, after which it probably seeks to the next chunk before reading it and encrypting the next chunk of bytes.

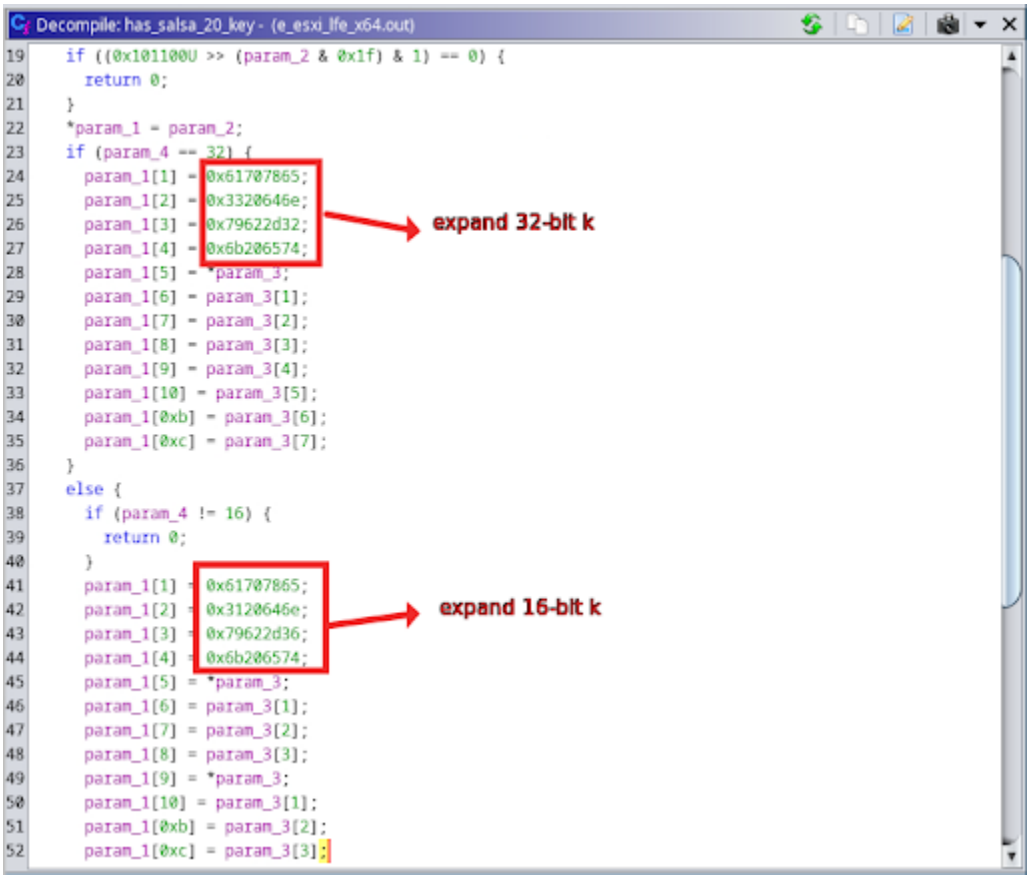

Fig. 11 - Constants related to the Salsa20/ChaCha cipher family

After searching through the entire file, the filename has an .RTM extension appended to it.

File encryption on Windows and Linux versions

The encryption algorithm has two steps:

1. Asymmetric encryption is initially used. The bad actor embeds a public key in the file, with its corresponding private key remaining with the attacker. It generates a 32-byte shared secret between the attacker's public key and the file ephemeral keys using the Diffie-Hellman key exchange protocol.
2. It then uses ChaCha20 symmetric encryption. The shared secret is hashed to obtain a 32-byte key to be used with an asymmetric encryption algorithm. After encryption, each public key is written at the end of its corresponding file (as with Linux) or appended as an extension for Windows.

Both ECDH on Curve25519 and ChaCha are statically implemented without using any libraries or crypt function.

The Encryption Process

1. An ephemeral key is generated by using:

- Windows – SystemFunction36 resolves to bcryptprimitives.ProcessPrng, which generates a specified number of random bytes from the user-mode per-processor random number generator.
- Linux – By reading /dev/urandom to generate a random sequence.

These random bytes are used as a private key during the Elliptic-Curve Diffie-Hellman (ECDH) algorithm implemented on Curve25519.

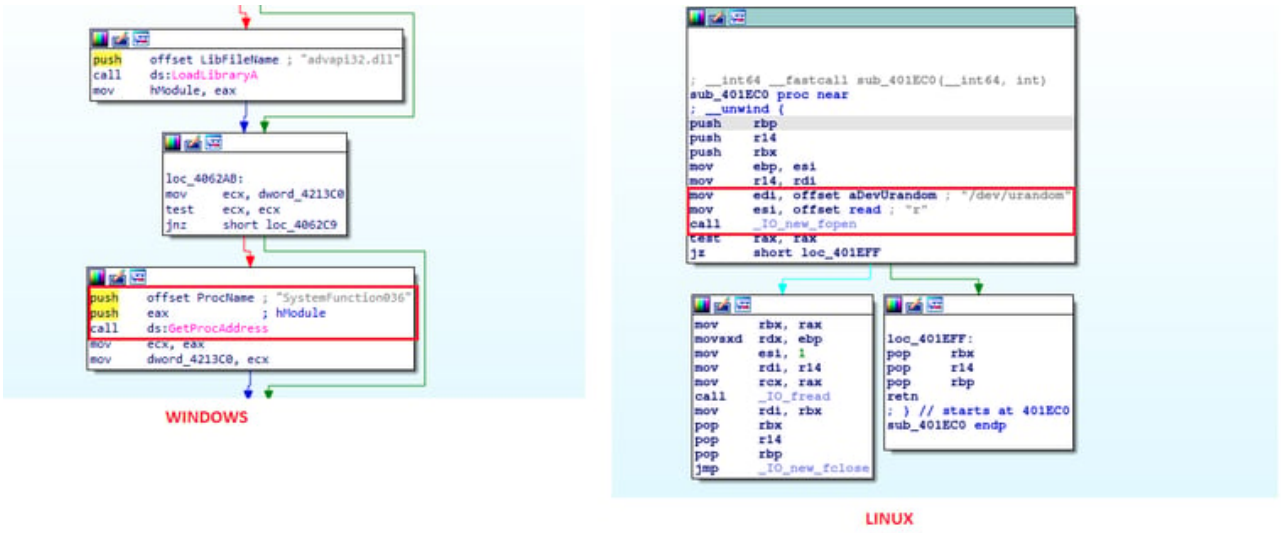

Fig. 12 - Random number generator as ephemeral key

2. The private key is now used to generate the public key on Curve25519.

- Windows – The public key is appended as an extension to the encrypted file.
- Linux – The public key is appended to the end of the encrypted file. This public key is used for decryption in the event of a victim paying ransom.

Fig. 13 - Encrypted files

3. A shared key is now generated, using the private key from step 1 and the attacker's public key hardcoded in the file on Curve25519. This shared secret is now used in symmetric ChaCha20 encryption.

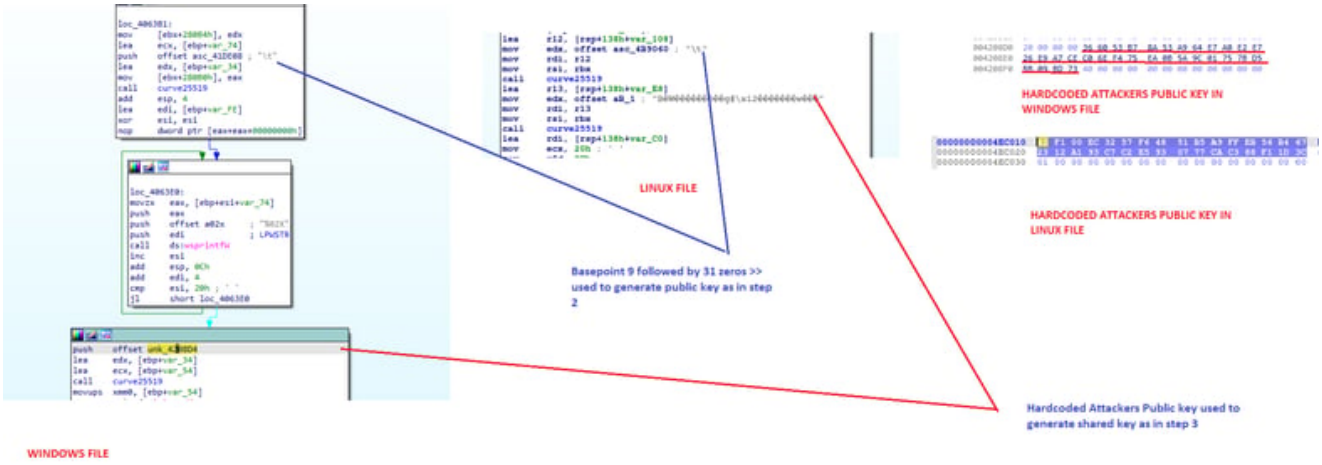

Fig. 14 - Code snippet showing shared key generation Curve25519

ChaCha20 is a symmetric encryption where:

- Key – 32-byte shared key from step 3
- Nonce – 8 bytes 0000000000000000
- Counter – 0
- ChaCha20 is used for symmetric encryption in both Windows and Linux
- For Windows, only the first 8000 hex bytes are encrypted, and the remaining bytes remain intact
- For Linux, the entire file is encrypted

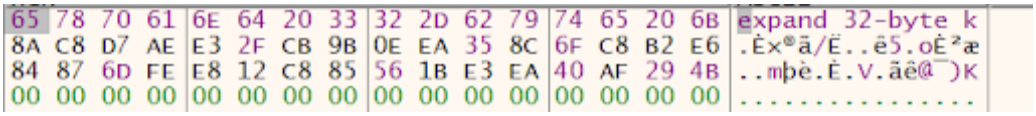

Fig. 15 - ChaCha key structure along with constants, key, counter, and nonce

File decryption

To decrypt the file, the public key, which is present in extension (WIndows) / end of the file (Linux), is read and along with the attacker's private key the shared secret is obtained allowing file decryption. Use of both asymmetric and symmetric encryption makes it impossible to decrypt the encrypted files without the attacker's private key.

Similarities with Babuk ransomware

As mentioned, RTM Locker was likely inspired from leaked source code of Babuk ransomware.

- Linux random number generation is done by reading /dev/urandom, the same as for Babuk ransomware
- Windows and Linux Curve25519 implementation is based on Babuk ransomware
- Both Linux versions encrypt files using the .log, .vmdk, .vmem, **.vswp,** and .vmsn file extensions
- Both use ECDH in Curve25519 for asymmetric and ChaCha for symmetric encryption.

Fig. 16 - Similarities between RTM and Babuk ransomware

After the entire directory is read, FUN_0047580 leaves a ransom note in the current directory that has a !!! Warning!!! filename (Figure 18).

Fig. 17 - Excerpt from FUN_0047580 that writes the ransom note

Figure 18 shows the RTM Locker ransom note. They group has left a Tox ID to contact it to decrypt the files after paying the ransom.

```
$ cat \!\!\!\ Warning\!\!\!
!!! Your network is infected by the RTM Locker command!!!
All your documents, photos, reports, customer and employee data, databases and other
important files are encrypted and you cannot decrypt them yourself. They are also on
our servers! But don't worry, we will help you recover all your files!
The only way to recover your files is to buy our dedicated software. Only we can prov
ide you with this software, and only we can recover your files!
We value our reputation. If we do not fulfill our work and obligations, no one will p
ay us. It's not in our interest.
All of our decryption software is perfectly tested and will decrypt your data. We wil
l also provide support in case of problems.
============================================== ==================================
========

To contact us, you need to install TOX (https://tox.chat/download.html), write to our
 support.
Contact: A0FE105A82525ECB94DD2977B4A1F8A5A7CF82F12D720DD8C8D9CCA3F98B6F52D911126AC1DF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Warning!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!
If you do not contact the support team within 48 hours, your data will be published i
n the public domain, and compromising data will be sent to your competitors, as well
as to the relevant regulatory authorities.


============================================== ==================================
==========

DO NOT ATTEMPT TO RECOVER THE FILES YOURSELF!
DO NOT MODIFY ENCRYPTED FILES!
OTHERWISE YOU MAY LOSE ALL YOUR FILES FOREVER!
```

Fig. 18 - Ransom note

```
$ sudo find / -type f -iname "*.RTM" 2>/dev/null
/run/cloud-init/cloud-init-generator.log.RTM
/run/cloud-init/ds-identify.log.RTM
/run/initramfs/overlayroot.log.RTM
/run/initramfs/fsck.log.RTM
/usr/share/doc/python3.8/pybench.log.RTM
/var/lib/mysql/tc.log.RTM
/var/lib/docker/containers/1142c893664baa9d1742b43d63c962291164b2979196d7af5781fa3c60
fd8040/1142c893664baa9d1742b43d63c962291164b2979196d7af5781fa3c60fd8040-json.log.RTM
/var/lib/docker/containers/aded7326a8b68a12cc56ca455736aed54d510bbacbd989787fa2a32dd0
46c094/aded7326a8b68a12cc56ca455736aed54d510bbacbd989787fa2a32dd046c094-json.log.RTM
/var/lib/docker/containers/c2d28ae410054a992fbf310ce2e6dfb7135849ddff6239bf02c1e1785f
de5951/c2d28ae410054a992fbf310ce2e6dfb7135849ddff6239bf02c1e1785fde5951-json.log.RTM
/var/lib/docker/containers/aef13f2baae9cd6691b527545f4ba3d7ba73acc4deda3b4e65a600d811
3025e7/aef13f2baae9cd6691b527545f4ba3d7ba73acc4deda3b4e65a600d8113025e7-json.log.RTM
/var/lib/docker/containers/ffa9098b66da87e57b8ca9ec1e175c4b318f715bd1a2eb57486edeeac6
a47273/ffa9098b66da87e57b8ca9ec1e175c4b318f715bd1a2eb57486edeeac6a47273-json.log.RTM
/var/lib/docker/containers/44f3b6c5fabf87b60c82fa01478625276be862f6422916689a765b382e
5aaab2/44f3b6c5fabf87b60c82fa01478625276be862f6422916689a765b382e5aaab2-json.log.RTM
/var/lib/docker/containers/aa877baa1eaf7519632d3463b2d1e2f79c818ff4a6d3fbbc4c48c86cde
f746a6/aa877baa1eaf7519632d3463b2d1e2f79c818ff4a6d3fbbc4c48c86cdef746a6-json.log.RTM
/var/lib/docker/containers/b57bbf19389b49349b13cec280ffd5a0ccb8b0ebacba26995e5a1eca5d
0e0fd8/b57bbf19389b49349b13cec280ffd5a0ccb8b0ebacba26995e5a1eca5d0e0fd8-json.log.RTM
/var/lib/docker/containers/828408cd493ea59b6ac25cd9a034378ec257835d7e4cacae287482cb4e
a4a794/828408cd493ea59b6ac25cd9a034378ec257835d7e4cacae287482cb4ea4a794-json.log.RTM
/var/lib/docker/containers/1d9b979a0ffaa72b99835d44256facb8237403f0a3fc62e2bfc20e0f5c
```

Fig. 19 - Files encrypted by the RTM Locker ransomware

Uptycs XDR Coverage

In addition to having YARA built in and being armed with other advanced detection capabilities, Uptycs XDR users can easily scan for RTM Locker. XDR contextual detection provides important details about identified malware. Users can navigate to the toolkit data section in the detection screen, then click a detected item to reveal its profile (Fig. 20).

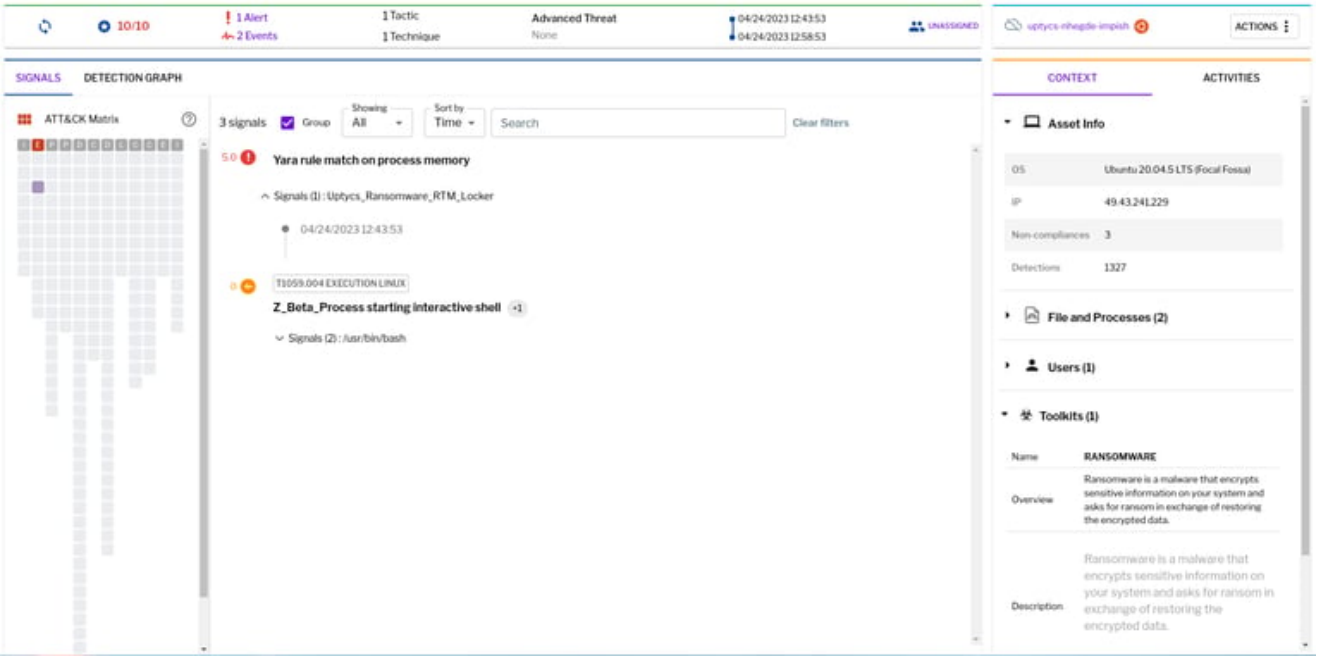

Fig. 20 - Uptycs detection

IOC

SHA256
55b85e76abb172536c64a8f6cf4101f943ea826042826759ded4ce46adc00638
b376d511fb69085b1d28b62be846d049629079f4f4f826fd0f46df26378e398b
d68c99d7680bf6a4644770edfe338b8d0591dfe143278412d5ed62848ffc99e0

YARA

Uptycs XDR scans the memory of newly launched processes and detects any presence of suspicious strings by using YARA rules. The rule for detecting this RTM Locker has already been made available to our customers.

If you're not an Uptycs XDR customer, you can use either the YARA tool or a third-party tool to scan suspicious processes. Here we share the rule for your convenience.

```
rule Uptycs_Ransomware_RTM_Locker
{
   meta:
      malware_name = "RANSOMWARE"
      description = "Ransomware is a malware that encrypts sensitive information on your system and asks for ransom in exchange for restoring the encrypted data."
      author = "Uptycs Inc"
      version = "1"
```

```
    strings:
        $Ransomware_RTM_Locker_0 = "esxcli vm process list"  ascii wide
        $Ransomware_RTM_Locker_1 = "vmlist.tmp.txt"  ascii wide
        $Ransomware_RTM_Locker_2 = "esxcli vm process kill"  ascii wide
        $Ransomware_RTM_Locker_3 = "!!! Warning!!!"  ascii wide
        $Ransomware_RTM_Locker_4 = "Your network is infected by the RTM Locker
command"  ascii wide
    condition:
        all of ($Ransomware_RTM_Locker*)
}
```