



# HACKTHEBOX

## Safecracker Writeup

---



20<sup>th</sup> July 2023

Prepared by: blitztide

Machine Author(s): Blitztide, Sebh24

Difficulty: Insane

## Scenario

---

We recently hired some contractors to continue the development of our Backup services hosted on a Windows server. We have provided the contractors with accounts for our domain. When our system administrator recently logged on, we found some pretty critical files encrypted and a note left by the attackers. We suspect we have been ransomware. We want to understand how this attack happened via a full in-depth analysis of any malicious files out of our standard triage. A word of warning, our tooling didn't pick up any of the actions carried out - this could be advanced.

### Warning

This is a warning that this Sherlock includes software that is going to interact with your computer and files. This software has been intentionally included for educational purposes and is NOT intended to be executed or used otherwise. Always handle such files in isolated, controlled, and secure environments.

One the Sherlock zip has been unzipped, you will find a DANGER.txt file. Please read this to proceed.

## Artefacts Provided

---

Enter the artefacts provided along with their file hash here.

- safecracker.zip - 61EA7F259763C0937BEE773A16F4CE84

## Initial Analysis

---

Due to the fact that this Sherlock contains live malware, a textfile called `DANGER.txt` is included in the outer `.zip` file, this outlines the risk of analysis and gives the password for the internal `.zip` file.

## MFT Analysis

---

Exporting all of the data to CSV by using the command

```
MFTECmd.exe -f $MFT --csv Safecracker
```

We can now open the CSV file generated in `Timeline Explorer` which will allow us to navigate around the filesystem.

We can see that there are two User profiles:

- contractor01
- Administrator

Looking into the Administrator user's directory we can find a `Backups` folder that contains what appears to be `.31337` files and `.note` files.

Opening the `.note` file we can find the following message:

```
You have been hacked by Cybergang31337
```

```
Please can you deposit $200,000 in BTC to the following address:
```

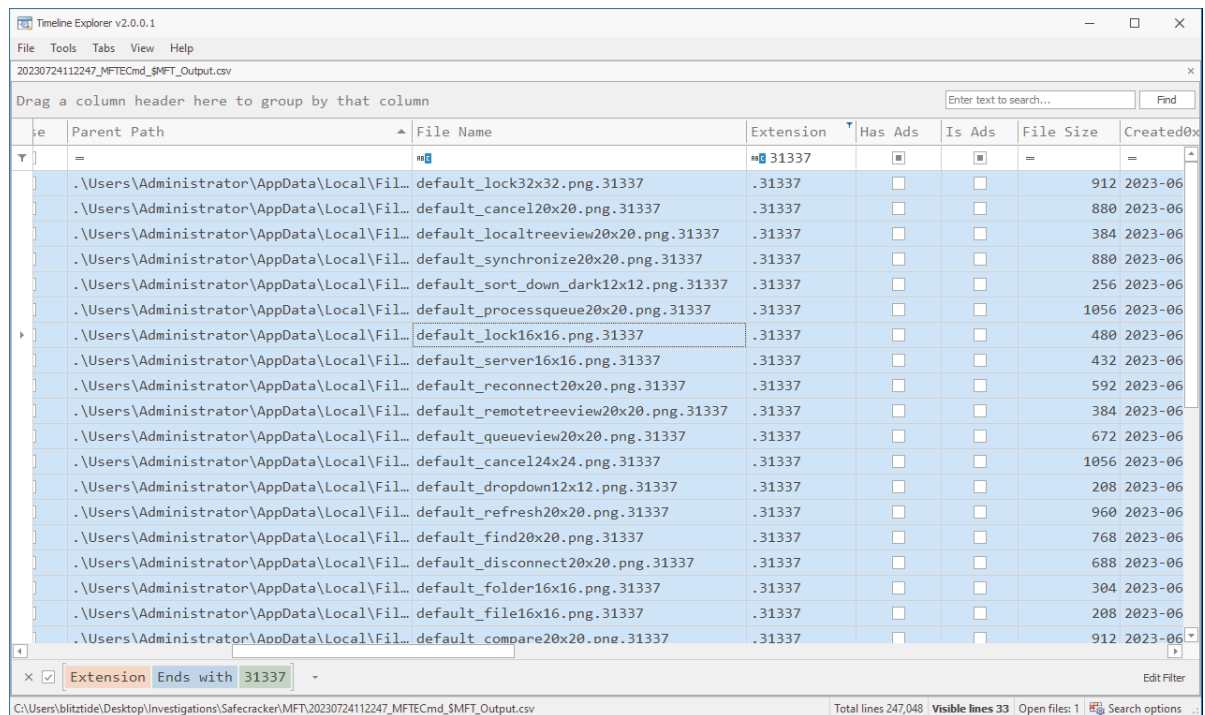
```
- 16ftSEQ4ctQFDtVZiUBusQUjRrGhM3JYwe
```

```
Once you have done so please email: decryption@cybergang31337.hacker  
indicating your source BTC address and we will confirm and release decryption  
keys.
```

```
Regards
```

```
-Cybergang31337
```

We can now filter across the whole filesystem and look for files that have the extension `.31337` to find the number of files encrypted by the ransomware.



Parent Path	File Name	Extension	Has Ads	Is Ads	File Size	Created
.\Users\Administrator\AppData\Local\Films	default_lock32x32.png.31337	.31337			912	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_cancel20x20.png.31337	.31337			880	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_localtreeview20x20.png.31337	.31337			384	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_synchronize20x20.png.31337	.31337			880	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_sort_down_dark12x12.png.31337	.31337			256	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_processqueue20x20.png.31337	.31337			1056	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_lock16x16.png.31337	.31337			480	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_server16x16.png.31337	.31337			432	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_reconnect20x20.png.31337	.31337			592	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_remotetreeview20x20.png.31337	.31337			384	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_queueview20x20.png.31337	.31337			672	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_cancel24x24.png.31337	.31337			1056	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_dropdown12x12.png.31337	.31337			208	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_refresh20x20.png.31337	.31337			960	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_find20x20.png.31337	.31337			768	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_disconnect20x20.png.31337	.31337			688	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_folder16x16.png.31337	.31337			304	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_file16x16.png.31337	.31337			208	2023-06-21 13:01:00
.\Users\Administrator\AppData\Local\Films	default_compare20x20.png.31337	.31337			912	2023-06-21 13:01:00

This search returns 33 visible files.

We can identify the first file to be encrypted by the ransomware and use this to get a timestamp for when to look for execution.

## Initial Entry

Using `EVTXECmd .\EvtXECmd.exe -d $LOG_FOLDER --csv $EVENTS` we are able to export all relevant log files for the machine.

We can find the file `ConsoleHost_history` in the `C:\Users\contractor01` directory where the user has ran the command `.\PsExec64.exe -s -i cmd.exe` which gives them a cmd prompt with `NT\SYSTEM` privileges.

We can filter for all events prior to the first instance of a `.31337` file and for Event ID 4624 which is a log that is generated when a login is performed.

```
[Time Created] < #2023-06-21 13:06:14# And [Event Id] = 4624
```

We can see that many logins happen prior to the event, but we can see that a user logged in with Logon type 10 (Remote Desktop) at:

- 2023-06-21 13:01:00

## WSL

There are a few logs identified between 13:01:00 and 13:06:14 which indicate that a `WSL2` instance was started as we can see below from the `Microsoft-Windows-Hyper-V-Compute-Operational` log at 13:01:07.

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">  
  <System>
```

```

<Provider Name="Microsoft-Windows-Hyper-V-Compute" Guid="{17103e3f-3c6e-4677-bb17-3b267eb5be57}" />
<EventID>2007</EventID>
<Version>0</Version>
<Level>4</Level>
<Task>0</Task>
<Opcode>0</Opcode>
<Keywords>0x4000000000000000</Keywords>
<TimeCreated SystemTime="2023-06-21T13:01:07.3975352Z" />
<EventRecordID>110</EventRecordID>
<Correlation />
<Execution ProcessID="6584" ThreadID="5864" />
<Channel>Microsoft-Windows-Hyper-V-Compute-Operational</Channel>
<Computer>WinBackUp001.forela.local</Computer>
<Security UserID="S-1-5-18" />
</System>
<UserData>
  <VmEventLog xmlns="http://www.microsoft.com/Windows/Virtualization/Events">
    <SystemId>F855A3D1-B822-46E5-86EF-47CC36B2F551</SystemId>
    <Result>0x00000000</Result>
    <Parameter0>
      {"ResourcePath":"VirtualMachine/Devices/Scsi/0/Attachments/1","RequestType":"Add",
      "Settings":
      {"Type":"VirtualDisk","Path":"C:\\Users\\Administrator\\AppData\\Local\\Packages\\CanonicalGroupLimited.Ubuntu20.04onWindows_79rhkp1fndgsc\\LocalState\\ext4.vhdx",
      "SupportCompressedVolumes":true,"AlwaysAllowSparseFiles":true,"SupportEncryptedFiles":true}}</Parameter0>
    </VmEventLog>
  </UserData>
</Event>

```

Looking at files modified or created after this time will show what possible actions could have been taken by the actor, we can do this with the MFT we exported to CSV earlier.

Sequence Number	Parent Entry Number	Parent Sequence Number	In Use	Parent Path	File Name	Extension	Is Directory
4	97667	2	<input checked="" type="checkbox"/>	.\Users\Administrator\Downloads	MsMpEng.exe	.exe	<input checked="" type="checkbox"/>
5	97723	1	<input checked="" type="checkbox"/>	.\Windows\Logs\WindowsUpdate	WindowsUpdate.20230621.140135.889.1.etl	.etl	<input type="checkbox"/>
6	475	1	<input checked="" type="checkbox"/>	.\ProgramData\USOShared\Logs\System	NotificationUsxBroker.9c307868-d359-411c-b44...	.etl	<input type="checkbox"/>
6	91155	6	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	5b120a24.BUD	.BUD	<input type="checkbox"/>
6	97767	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	B5F9AC62-7C51-4AD2-85C3-D91ABE7C2E95	.BUD	<input checked="" type="checkbox"/>
6	91189	6	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	69b8a4a.BUD	.BUD	<input type="checkbox"/>
6	97767	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	4AA0E07D-084E-437F-93A2-D968A8F79668	.BUD	<input checked="" type="checkbox"/>
5	91864	6	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	9a072afe.BUD	.BUD	<input type="checkbox"/>
6	97767	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	25FF57E4-2277-4D74-B5A4-E0CC94B72EC1	.BUD	<input checked="" type="checkbox"/>
5	31924	19	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	69b8a4a.BUD	.BUD	<input type="checkbox"/>
19	97767	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	AS6F4AA7-A69E-4FAS-8983-548FFA324A2F	.BUD	<input checked="" type="checkbox"/>
16	5858	10	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	5b120a24.BUD	.BUD	<input type="checkbox"/>
10	97767	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	CEF9E5D7-CFA1-4658-8AAS-ED38B1CC146	.BUD	<input checked="" type="checkbox"/>
8	97691	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	4		<input checked="" type="checkbox"/>
4	76124	2	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	MSH1st012023062120230622		<input checked="" type="checkbox"/>
14	30143	3	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	1033		<input checked="" type="checkbox"/>
6	23135	10	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	48F4C442-9B8A-41A8-B380-DD4704D0B28		<input checked="" type="checkbox"/>
10	30124	3	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	Vault		<input checked="" type="checkbox"/>
5	4453	1	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	Microsoft-Windows-VHMP-Operational.evtx	.evtx	<input type="checkbox"/>
2	74877	3	<input checked="" type="checkbox"/>	.\Users\contractor01\AppData\Local\Microsoft\Internet Expl...	CacheStorage		<input checked="" type="checkbox"/>
2	86493	2	<input checked="" type="checkbox"/>	.\Users\contractor01\AppData\Local\Microsoft\Internet Expl...	10.0.0.0		<input checked="" type="checkbox"/>
2	30556	3	<input checked="" type="checkbox"/>	.\Users\contractor01\AppData\Local\Microsoft\Internet Expl...	History.IES		<input checked="" type="checkbox"/>
3	4453	1	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	Microsoft-Windows-TerminalServices-RemoteCo...	.evtx	<input type="checkbox"/>
3	5892	1	<input checked="" type="checkbox"/>	.\Windows\System32\spool\V4Dir\B5F9AC62-7C51-4AD2-85C3-D91...	vmware-vmtoolsd-Administrator.log	.log	<input type="checkbox"/>
9	740	8	<input checked="" type="checkbox"/>	.\Windows\Logs\NetSetup	service.0.etl	.etl	<input type="checkbox"/>

This identifies a key file called **MsMpEng.exe** in the Administrators download folder, this is normally a **Windows Defender** binary, but it lives elsewhere in the file system, this file is available in the triage data.

# Mysterious file

If we upload the file to [VirusTotal](#) we can see that it is a Linux ELF binary and doesn't appear to be malicious.

1  
/ 60

Community Score

1 security vendor and no sandboxes flagged this file as malicious

6b091a4a56a66f858d4c75eb72e16e87169526873147fc78cc4dc261d4cac000  
MsMpEng.exe  
elf 64bits shared-lib

Size  
4.10 MB

Last Analysis Date  
9 days ago

ELF

Reanalyze

Similar

More

DETECTION

DETAILS

BEHAVIOR

COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Security vendors' analysis

Do you want to automate checks?

TrendMicro-HouseCall	❗ TROJ_GEN.R002V01KG23	Acronis (Static ML)	✅ Undetected
Ad-Aware	✅ Undetected	AhnLab-V3	✅ Undetected
ALYac	✅ Undetected	Antiy-AVL	✅ Undetected
Arcabit	✅ Undetected	Avast	✅ Undetected
Avast-Mobile	✅ Undetected	AVG	✅ Undetected
Avira (no cloud)	✅ Undetected	Baidu	✅ Undetected
BitDefender	✅ Undetected	BitDefenderTheta	✅ Undetected
Bkav Pro	✅ Undetected	ClamAV	✅ Undetected
CMC	✅ Undetected	Cynet	✅ Undetected

## Malware Analysis

### Strings!

Using `strings` within `MsMpEng.exe` we can find some clues about what kind of binary this is.

A quick triage to find interesting strings would be `strings MsMpEng.exe | awk 'length($0) > 9' | sort | uniq > /tmp/orig.strings`

This will return all strings found that are greater than 9 characters in length in alphabetical order.

Notable mentions are:

```
/home/blitztide/Projects/Payloads/LockPick3.0/lib/ssl
GCC: (Debian 10.2.1-6) 10.2.1 20210110
SHA512 block transform for x86_64, CRYPTOGAMS by <appro@openssl.org>
crypto/evp/e_aes.c
crypto/evp/e_aria.c
crypto/evp/e_camellia.c
crypto/evp/e_chacha20_poly1305.c
crypto/evp/e_des3.c
inflate 1.2.11 Copyright 1995-2017 Mark Adler
zlib compression
zlib deflate error
zlib inflate error
zlib not supported
```

These show what platform the malware was developed on, the source locations for certain files, and that it has included libraries for OpenSSL and Zlib.

## Elf information

```
>file MsMpEng.exe
```

```
MsMpEng.exe: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),  
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,  
BuildID[sha1]=e2097187e976e108f5b233c0a288cc35ae5886b8, for GNU/Linux  
3.2.0, stripped
```

```
>objdump -s -j .comment MsMpEn  
g.exe
```

```
MsMpEng.exe:      file format elf64-x86-64
```

```
Contents of section .comment:
```

```
0000 4743433a 20284465 6269616e 2031302e  GCC: (Debian 10.  
0010 322e312d 36292031 302e322e 31203230  2.1-6) 10.2.1 20  
0020 32313031 313000      210110.
```

```
> strace ./MsMpEng.exe
```

```
execve("./MsMpEng.exe", ["/MsMpEng.exe"], 0x7ffd9d56d2d0 /* 29 vars */) = 0  
getrandom("\xd1\x9a\x94\x09\xbc\xff\xfa\x3b", 8, GRND_NONBLOCK) = 8  
memfd_create("test", MFD_CLOEXEC) = 3  
write(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0-\0\1\0\0\0\320\242\4\0\0\0\0"...  
execve("/proc/self/fd/3", ["PROGRAM"], 0x7ffe1a1f3e08 /* 29 vars */) = 0  
openat(AT_FDCWD, "/proc/self/status", O_RDONLY) = 3  
read(3, "Name:\t3\nUmask:\t0022\nState:\tR (ru)...", 1024) = 1024  
write(1, "*****DEBUGGED*****\n", 24*****DEBUGGED*****  
) = 24  
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_TKILL, si_pid=1873, si_uid=1000} ---  
+++ killed by SIGSEGV +++  
Segmentation fault
```

This shortened `strace` output shows that the executable opens a file descriptor in memory, writes what appears to be an `ELF` executable to it, then executes.

Inside the second executable we can see that it has detected `strace` as a debugger and has raised a SIGSEGV fault.

## Finding the unpacked executable

`Memfd_create` is a rare syscall to use, we can use this to navigate to the code in `Ghidra`, which leads us to offset `0x0013a50a`.

We rename the function to `main` and give it the proper function signature.

```
int main(int argc, char **argv)  
  
{  
    int file_size;
```

```

uint mem_fd;
void *malloc_1;
void *malloc_2;
ssize_t sVar1;
char *pcVar2;
undefined8 in_R8;
undefined8 in_R9;
char path [256];
uint memfd;

malloc_1 = malloc((ulong)UINT_00518ee0);
malloc_2 = malloc((size_t)&size_t_0036d920);
FUN_0013a29b(&DAT_003893a0,malloc_1,UINT_00518ee0);
file_size =
FUN_0013a3cb(malloc_1,malloc_2,&size_t_0036d920,&size_t_0036d920,in_R8,in_R9,argv
);
if (file_size < 0) {
    FUN_0013a4ac(file_size);
}
free(malloc_1);
mem_fd = memfd_create("test",0b00000001);
if ((int)mem_fd < 1) {
    printf("ERROR FD:%i\n", (ulong)mem_fd);
    /* WARNING: Subroutine does not return */
    exit(-1);
}
sVar1 = write(mem_fd,malloc_2,(long)file_size);
if ((int)sVar1 < 1) {
    pcVar2 = strerror((int)sVar1);
    fprintf(stderr,"Error Writing: %s\n",pcVar2);
    /* WARNING: Subroutine does not return */
    exit(-1);
}
free(malloc_2);
path = {0};
sprintf(path,"/proc/self/fd/%i", (ulong)mem_fd);
execl(path,"PROGRAM",0);
return 0;
}

```

Looking at the function, we can see two large buffers have been created and `FUN_0013a29b` takes data from offset `0x003893a0` and moves it into buffer 1, then the next function call appears to do some extra data manipulation between the two buffers.

The two buffer sizes are:

- MALLOC1 - 1637184
- MALLOC2 - 55834576014

We can see that the second function returns a value for the file size, which is then used in the `write` syscall that can be seen in the `strace` output.

Lets try and extract the original buffer from `MsMpEng.exe` for analysis, we can do this by taking the first 10 hex values in the location pointed to by `DAT_003893a0` and run these commands:

```

> xxd -d MsMpEng.exe | grep "c457 d495"
02655136: c457 d495 8a53 b4ae e8ce 421c 5076 dc07 .W...S....B.Pv..
> dd if=MsMpEng.exe skip=02655136 bs=1 count=1637184 of=inner.bin
1637184+0 records in
1637184+0 records out
1637184 bytes (1.6 MB, 1.6 MiB) copied, 4.59638 s, 356 kB/s
> file inner.bin
inner.bin: data
> ent inner.bin
Entropy = 7.999878 bits per byte.

Optimum compression would reduce the size
of this 1637184 byte file by 0 percent.

Chi square distribution for 1637184 samples is 277.09, and randomly
would exceed this value 16.34 percent of the times.

Arithmetic mean value of data bytes is 127.5425 (127.5 = random).
Monte Carlo value for Pi is 3.143030960 (error 0.05 percent).
Serial correlation coefficient is -0.000898 (totally uncorrelated = 0.0).

```

We can see that the data extracted has an extremely high entropy, this leads us to believe that the data is compressed and/or encrypted, from the strings analysis we know that OpenSSL and Zlib are being used, which means both are probably used in this packer.

Is it more efficient to compress data then encrypt it, rather than encrypting it then compressing, because encryption removes all patterns from the data making it almost impossible to compress. Using this as a starting point, we can theorise that the `inner.bin` file we have extracted needs to be decrypted first, then decompressed.

## Finding the Encryption method

We have identified two values that are used for the encryption:

- a5f41376d435dc6c61ef9ddf2c4a9543c7d68ec746e690fe391bf1604362742f
- 95e61ead02c32dab646478048203fd0b

this indicates that it is probably a Symmetric Cipher, OpenSSL has the following ciphers available:

Cipher	Key Size	Block Size/ IV Size	Modes
AES	256/192/128	128	CBC,CCM,CFB,CTR,ECB,GCM,OCB,OFB,XTS
BLOWFISH	32-448	64	CBC, CFB,ECB,OFB
CAST	40-128	128	ECB,CBC
DES	56	64	CBC,CFB,ECB,OFB,
3DES	112	64	CBC,CFB,ECB,OFB,
IDEA	128	64	CBC
RC2	1-128	64	CBC



Cipher	Key Size	Block Size/ IV Size	Modes
RC5	0-2040	32/64/128	CBC
SEED	128	128	CBC
SM4	128	128	CBC
ARIA	128/192/256	128	GCM
CAMELLIA	128/192/256	128	CBC

Knowing that the key is 256 bit and IV is 16 bytes, we can narrow it down to:

- AES 256
- RC5
- ARIA-256
- CAMELLIA-256

The most common of these algorithms would be AES-256 so we can start our testing on this algorithm.

Loading the extracted file into **CyberChef** we can attempt AES-256 decryption and iterate through the modes of operations.

The screenshot shows the CyberChef interface with the following details:

- Operations:** A sidebar on the left lists various operations like Zlib Deflate, Zlib Inflate, Detect File Type, Extract Files, etc.
- Recipe:** The main panel shows an 'AES Decrypt' operation. The 'Key' is 'a5f41376d435dc6c61e...' and the 'IV' is '95e61ead02c32dab646...'. The 'Mode' is set to 'CBC' and the 'Input' is 'Raw'.
- Input:** The input panel shows a hex dump of the file 'inner.bin'.
- Output:** The output panel shows the file type as 'Zlib Deflate', extension as 'zlib', and MIME type as 'application/x-deflate'.
- Detect File Type:** A panel below the recipe shows checkboxes for 'Images', 'Video', 'Audio', 'Documents', and 'Applications', all of which are checked.

luckily for us, the first mode **CBC** is the correct one and the file is detected as a **Zlib Deflate** which you can easily inflate with **CyberChef**

The screenshot shows a file analysis tool with three main panels. On the left is a sidebar with categories like 'Operations', 'Favourites', 'Data format', 'Encryption / Encoding', 'Public Key', 'Arithmetic / Logic', 'Networking', and 'Language'. The 'Operations' panel is active, showing a list of operations including 'Zlib Deflate', 'Zlib Inflate', 'Detect File Type', and 'Extract Files'. The 'Recipe' panel in the center shows a sequence of operations: 'AES Decrypt' (with Key and IV fields) and 'Zlib Inflate' (with Start index and Initial output buffer fields). The 'Input' panel on the right shows a hex dump of the file. The 'Output' panel on the far right shows file details: 'File type: Executable and Linkable Format', 'Extension: elf,bin,axf,o,prx,so', 'MIME type: application/x-executable', and 'Description: Executable and Linkable Format file. No standard file extension.'

we can now save this file as `inner.elf` and continue our analysis.

## Inner Executable

To be sure that these results can be recreated, the md5 of the `inner.elf` file is

`57c4a3c06a461f96f1ed3a986f4e6016`

## More Strings

running the command `strings inner.elf | awk 'length($0) > 9' | sort | uniq > /tmp/new.strings` we find that there is a large overlap of strings from the original file, maybe libraries have been reused in this one.

We can filter out the new additions with the command `comm -1 -3 /tmp/orig.strings /tmp/new.strings`

```
# This file was generated by libcurl! Edit at your own risk.
*****DEBUGGED*****

/mnt/c/Users
/proc/self/status
Unsupported HTTP version (%u.%d) in response
You can now connect to the Corporate VPN
compiler: gcc -pthread -m64 -Wa,--noexecstack -Wall -O3 --static -
DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_CPUID_OBJ -DOPENSSL_IA32_SSE2 -
DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -
DSHA256_ASM -DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DAESNI_ASM -
DVPAAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -DPOLY1305_ASM -DNDEBUG
curl failed: %s
daV324982S3bh2
```

## Finding the debugger

It appears that at the start of the main function there is a function call that checks for a debugger. inside that function it appears to conditionally set a signal handler based on the output of another function.

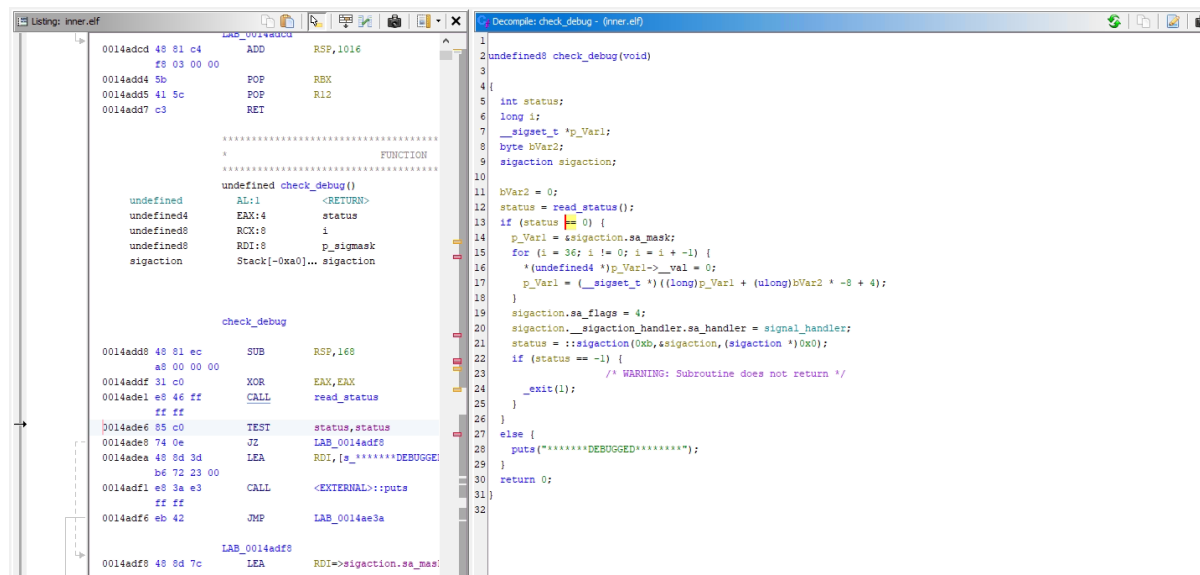
That final function appears to:

- Open `/proc/self/status`
- Find a line that contains `TracerPid`
- Splits the line on the character `:`
- Returns the value after that

This `TracerPid` value is the PID of the process that is currently tracing the running process, this could be `strace`, `gdb` or any other process that has used the `ptrace` syscall.

A value of `0` will indicate that there are no tracing processes.

We can patch the function in `check_debug`



Where the instruction `0014ade8 74 0e JZ LAB_0014aff8` could be changed to unconditional jump, which then makes the program think that there isn't a debugger even if there is.

## Patching the binary

We know the opcode used by the JZ instruction is `74 0e` which is a conditional jump to an 8bit offset.

We can look up a new two byte opcode that will display the behaviour that we need, and that instruction is: `EB 0e` which is an unconditional jump to an 8bit offset, this will mean our function will always return false for detecting a debugger.

We can do a quick online patch with the following command:

```
echo -en "\xEB" | dd of=./inner.elf bs=1 seek=306664 conv=notrunc
```

we can now run the program in `strace`

## Strace

We now have a new strace for the full execution which gives us all syscalls for the binary such as:

- getdents64
- unlink

## Segfault?

Running the program in `gdb` appears to give a large amount of segfault errors.

The programmer appears to have left `raise(SIGSEGV)` calls all around the code, the issue with `gdb` is that it injects its own signal handlers into the code.

To bypass this we can issue the command:

```
handle SIGSEGV nostop noprint
```

This will ignore all SIGSEGV faults within the code and allow it to run to completion.

## What files do we want?

To identify which files will get encrypted we can do the following

```
> sudo umount /mnt/c
> mkdir /mnt/c/Users
> touch /mnt/c/Users/1{.pptx,.pdf,.tar.gz,.tar,.zip,.exe,.mp4,.mp3}
> ./inner.elf
Running update, testing update endpoints
Checking uri: http://google.com ..... [SUCCESS]
Checking uri: http://icanhazip.com ..... [SUCCESS]
Checking uri: http://something.com ..... [SUCCESS]
Checking uri: http://ifconfig.me ..... [SUCCESS]
Checking uri: https://reddit.com ..... [SUCCESS]
Checking uri: https://wikipedia.org ..... [SUCCESS]
Files found:
- /mnt/c/Users/1.pptx
- /mnt/c/Users/1.tar
- /mnt/c/Users/1.zip
- /mnt/c/Users/1.tar.gz
- /mnt/c/Users/1.mp3
- /mnt/c/Users/1.pdf
- /mnt/c/Users/1.mp4

-----
Configuration Successful
You can now connect to the Corporate VPN
> ls /mnt/c/Users
ls /mnt/c/Users/
1.exe      1.mp3.note  1.mp4.note  1.pdf.note  1.pptx.note
1.tar.gz.31337 1.tar.note  1.zip.note
1.mp3.31337 1.mp4.31337 1.pdf.31337 1.pptx.31337 1.tar.31337
1.tar.gz.note 1.zip.31337
```

This identifies the files that the malware encrypts, it does not encrypt `.exe` files.

The malware appears to encrypt files and gives them a `.31337` file extension and leaves a note with `filename.note`

The contents of the note are below:

You have been hacked by Cybergang31337

Please can you deposit \$200,000 in BTC to the following address:

- 16ftSEQ4ctQFDtVZiUBusQUjRrGhM3JYwe

Once you have done so please email: [decryption@cybergang31337.hacker](mailto:decryption@cybergang31337.hacker) indicating your source BTC address and we will confirm and release decryption keys.

Regards

- Cybergang31337

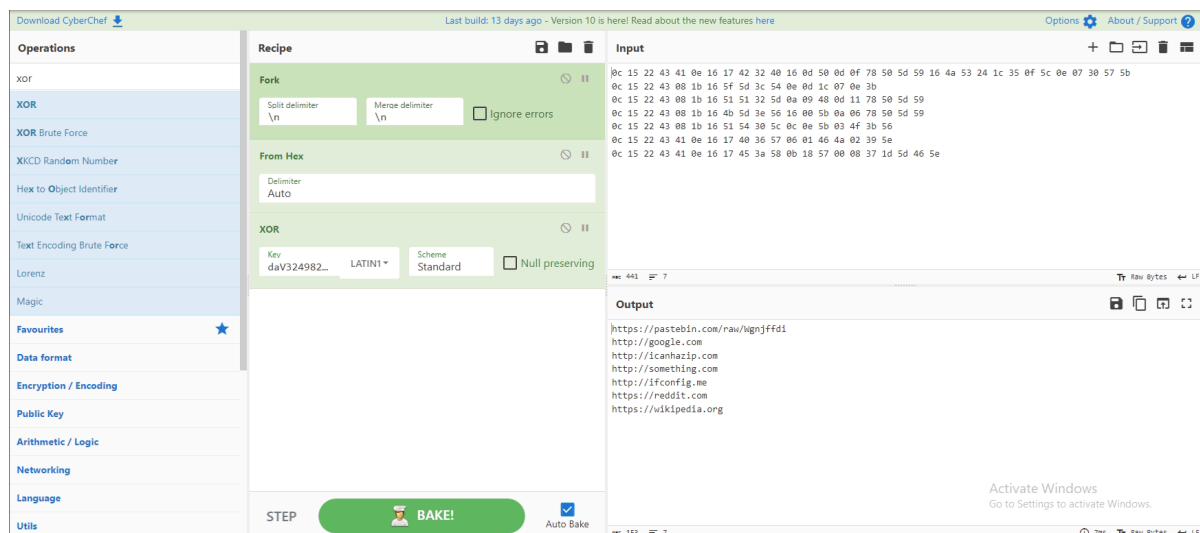
## Where are the URLs?

It appears to have connected to multiple websites however the strings for them do not appear within the file, they must be obfuscated somehow.

We can use the output of the strace command to identify all URLs used.

One of the globals in the disassembly appears to be a blob of 238 random characters with null bytes in.

The data can be extracted and imported into CyberChef to be decoded.



This gives us a new URL that wasn't shown in the text output of the program:

<https://pastebin.com/raw/Wgnjffdi>

Navigating to this URL gives us some sort of hex output:

a5f41376d435dc6c61ef9ddf2c4a9543c7d68ec746e690fe391bf1604362742f:95e61ead02c32dab646478048203fd0b

This appears to be the same encryption key and IV used to encrypt the malware itself.

# Questions

---

1. Which user account was utilised for initial access to our company server?

`contractor01`

2. Which command did the TA utilise to escalate to SYSTEM after the initial compromise?

`.\PsExec64.exe -s -i cmd.exe`

3. How many files have been encrypted by the the ransomware deployment?

`33`

4. What is the name of the process that the unpacked executable runs as?

`PROGRAM`

5. What is the XOR key used for the encrypted strings?

`daV324982S3bh2`

6. What encryption was the packer using?

`AES-256-CBC`

7. What was the encryption key and IV for the packer?

`a5f41376d435dc6c61ef9ddf2c4a9543c7d68ec746e690fe391bf1604362742f:95e61ead02c32da  
b646478048203fd0b`

8. What was the name of the memoryfd the packer used?

`test`

9. What was the target directory for the ransomware?

`/mnt/c/Users`

10. What compression library was used to compress the packed binary?

`zlib`

11. The binary appears to check for a debugger, what file does it check to achieve this?

`/proc/self/status`

12. What exception does the binary raise?

`SIGSEGV`

13. Out of this list, what extension is not targeted by the malware?

`.pptx, .pdf, .tar.gz, .tar, .zip, .exe, .mp4, .mp3`

`.exe`

14. What compiler was used to create the malware?

`gcc`

15. If the malware detects a debugger, what string is printed to the screen?

`*****DEBUGGED*****`

16. What is the contents of the `.comment` section?

`GCC: (Debian 10.2.1-6) 10.2.1 20210110`

17. What file extension does the ransomware rename files to?

`.31337`

18. What is the bitcoin address in the ransomware note?

`16ftSEQ4ctQFDtVZiUBusQUjRrGhM3JYwe`

19. What string does the binary look for when looking for a debugger?

`TracerPid`

20. It appears that the attacker has bought the malware strain from another hacker, what is their handle?

`blitztide`

21. What system call is utilised by the binary to list the files within the targeted directories?

`getdents64`

22. Which system call is used to delete the original files?

`unlink`