

# DarkSide on Linux: Virtual Machines Targeted

By By: Mina Naiim May 28, 2021 Read time: 5 min (1371 words)

Published: 2021-05-28 · Archived: 2026-04-05 16:06:27 UTC

## Ransomware

We focus on the behavior of the DarkSide variant that targets Linux. We discuss how it targets virtual machine-related files on VMware ESXI servers, parses its embedded configuration, kills virtual machines (VMs), encrypts files on the infected machine, collects system information, and sends it to the remote server.

---

**Updated June 1, 2021, 12:02 am ET: This article has been updated to remove the Command-and-Control (C&C) URI String field in Table 1. Further study showed that it does not apply consistently to a number of samples.**

As we discussed in our [previous blog](#), the DarkSide ransomware is targeting organizations in manufacturing, finance, and critical infrastructures in regions such as the United States, France, Belgium, and Canada. The DarkSide ransomware targets both Windows and Linux platforms. We also noticed that the Linux variant, in particular, targets ESXI servers.

In this blog, we focus on the behavior of the variant that targets Linux. This entry also discusses how this variant targets virtual machine-related files on VMware ESXI servers, parses its embedded configuration, kills virtual machines (VMs), encrypts files on the infected machine, collects system information, and sends it to the remote server.

This table summarizes some of the differences between the behavior of the DarkSide ransomware on Windows and on Linux:

Table 1. Comparison of DarkSide variants on Windows and Linux

	<b>Windows Variant</b>	<b>Linux Variant</b>
<b>Encryption Mechanism</b>	Salsa20 with RSA-1024	ChaCha20 with RSA-4096
<b>Cipher Blocks</b>	Salsa20 matrix is custom and randomly generated using “RtlRandomExW”	ChaCha20 initial block is standard, built using “ <b>expand 32-byte k</b> ” as a constant string
<b>Configuration</b>	Encrypted	Not encrypted
<b>Terminates VMs?</b>	No	Yes
<b>Target Files</b>	All files on the system except the files, folders, and file extensions mentioned in the configuration	VM-related files on VMware ESXI servers, with specific file extensions mentioned in the configuration
<b>New Extension</b>	Generated by applying CRC32 several times on the HWID of the victim machine as “.4731c768”	Hard-coded in the embedded configuration as “.darkside” or passed by

		execution parameters
<b>Ransom Note File Name</b>	Consists of hard-coded part in the configuration as “ <b>README.</b> ” and the generated ID mentioned previously: for example, “ <b>README.4731c768.TXT</b> ”	Hard-coded in the embedded configuration as “ <b>darkside_readme.txt</b> ” or passed by execution parameters

Analysis of the Linux Variant

**Targets**

As we noted earlier, DarkSide also has a Linux variant to infect more machines and cause more damage in the victim network. However, this variant is quite specific, as its main configuration targets VM-related files on VMware ESXI servers as seen in the following figure:

```
0000000004231F1 mov     esi, offset aUmdkUmemUswpLo ; "umdk,vmem,uswp,log,umsn"
0000000004231F6 mov     rbx, rdi
0000000004231F9 sub     rsp, 10h
0000000004231FD lea    rdx, [rsp+18h+var_9]
000000000423202 call   __ZN5C1EPKcRKSa1CE ; std::string::string(char const*,std::allocator<char> const&)
000000000423207 add     rsp, 10h
00000000042320B mov     rax, rbx
```

Figure 1. Target file extensions

**Configuration**

Unlike the Windows variant, the Linux variant’s strings and configuration are not obfuscated. The configuration of the Linux variant specifies features of the sample, such as the extension for encrypted files, C&C URL, number of threads, and a constraint on a minimum size of the target files to be encrypted.

Note that the root path — the starting point for encryption — in the following figure is “/vmfs/volumes/”, which is the default location for the VM files on ESXI hosts.

```
[CFG] Root Path...../vmfs/volumes/
[CFG] Key Size.....548 Bytes
[CFG] Public Key.....VALID
[CFG] Part Size.....500mb
[CFG] Space Size.....0mb
[CFG] Min Size.....1mb
[CFG] Search Extension.....vmdk,vmem,vswp,log
[CFG] New Extension.....darkside
[CFG] Thread Count.....1
[CFG] ReadMe File.....darkside_readme.txt
[CFG] ReadMe Size.....1969 Bytes
[CFG] Landing URL#[01].....http://securebestapp20.com/daecdcbecac
[CFG] User ID.....75fb1970b674cc4
[CFG] RC2 Key.....OK
```

Figure 2. Configuration of the Linux variant

In addition to the hard-coded configuration, the ransomware executable can accept parameters to infect more files and change its default settings. Figure 3 shows where the malware parses execution parameters.

```

v27 = sub_4309E0(0LL);
v28 = sub_42F370(v27, &v156); |
v153 = sub_455A20(&v158);
v29 = sub_456CD0(&v153, "help,h", "Help Screen");
v30 = sub_4571D0(v29, "size,s", v28, "Part Size to Process");
v31 = sub_4571D0(v30, "space,S", v26, &kunk_5BC761);
v32 = sub_4571D0(v31, "dir,d", v24, "Root Directory Path to Process");
v33 = sub_4571D0(v32, "ext,x", v22, "Extension To Apply For Renaming");
v34 = sub_4571D0(v33, "new,n", v20, "Extension To Apply For Encrypted files");
v35 = sub_4571D0(v34, "log,l", v18, "Log File Path");
v36 = sub_4571D0(v35, "thread,t", v16, "Worker Threads Count, 0 - dynamic");
v37 = sub_4571D0(v36, "key,k", v14, "RSA Public Key File Paths");
v38 = sub_4571D0(v37, "rc2,e", v12, "RC2 Key as HEX string");
v39 = sub_4571D0(v38, "content,c", v10, "ReadMe File Path");
sub_4571D0(v39, "readme,r", v8, "ReadMe File name");
    
```

Figure 3. Linux variant parameter parsing

**ESXCLI Commands**

DarkSide runs several ESXCLI commands (such as the command- line interface framework in vSphere) in order to collect information about the infected ESXI host, such as the running virtual machines VMs, storage- related information, and vSAN- related information.

Table 2 shows a list of ESXCLI commands run by DarkSide on the victim machine.

Table 2. ESXCLI Commands

Commands	Description
<b>esxcli --formatter=csv --format-param=fields=="Device,DevfsPath" storage core device list</b>	List the Devfs Path of the devices currently registered with the storage
<b>esxcli --formatter=csv storage filesystem list</b>	List the logical sections of storage currently connected to the ESXI host
<b>esxcli --format-param=fields=="WorldID,DisplayName" vm process list</b>	List the running VMs on the ESXI host
<b>esxcli vsan debug vmdk list</b>	List the status of VMDKs in vSAN
<b>esxcli --format-param=fields=="Type,ObjectUUID,Configuration" vsan debug object list</b>	List the UUID of the vSAN objects

Figure 4 shows how the DarkSide ransomware lists the running virtual machines on the ESXI.

```

00000000427D84 call    __ZNStC1EPKcRKKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
00000000427D89 lea    rdi, [rbx+8]
00000000427D8D lea    rdx, [rsp+98h+var_7E]
00000000427D92 mov    esi, offset aFormatParamFie ; "--format-param=fields=="WorldID,Displa"...
00000000427D97 call    __ZNStC1EPKcRKKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
00000000427D9C lea    rdi, [rbx+10h]
00000000427DA0 lea    rdx, [rsp+98h+var_7D]
00000000427DA5 mov    esi, offset aUm ; "vm"
00000000427DAA call    __ZNStC1EPKcRKKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
00000000427DAF lea    rdi, [rbx+18h]
00000000427DB3 lea    rdx, [rsp+98h+var_7C]
00000000427DB8 mov    esi, offset aProcess ; "process"
00000000427DD0 call    __ZNStC1EPKcRKKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
00000000427DD2 lea    rdi, [rbx+20h]
00000000427DD6 lea    rdx, [rsp+98h+var_7B]
00000000427DDB mov    esi, offset aList ; "list"
00000000427DD8 call    __ZNStC1EPKcRKKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
00000000427DD5 lea    rcx, [rsp+98h+var_7A]
    
```

Figure 4. Listing running VMs

## Killing Virtual Machines

Before encryption, the Linux variant of the DarkSide ransomware can power off running VMs on the ESXI server using the following ESXI command:

**“esxcli vm process kill --type= force --world-id= <WorldNumber>”**

```

000000000426FA9      nov     esi, offset aWorldId ; "--world-id="
000000000426FAE      push   r12
000000000426FB0      push   rbp
000000000426FB1      mov    rbp, rdi
000000000426FB4      push   rbx
000000000426FB5      sub    rsp, 0F8h
000000000426FBC      lea   r15, [rsp+128h+var_58]
000000000426FC4      lea   rdi, [rsp+128h+var_108]
000000000426FC9      mov    rdx, r15
000000000426FCC      call  __ZNStC1EPlKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&
000000000426FD1      lea   rax, [rsp+128h+var_E8]
000000000426FD6      lea   rdi, [rsp+128h+var_108] ; this
000000000426FD8      mov    rsi, r13 ; std::string *
000000000426FDE      mov    [rsp+128h+var_120], rax
000000000426FE3      call  __ZNSt6appendERKStS ; std::string::append(std::string const&)
000000000426FE8      lea   rax, [rsp+128h+var_E8]
000000000426FED      lea   rdx, [rsp+128h+var_109]
000000000426FF2      mov    esi, offset aUm ; "um"
000000000426FF7      mov    rdi, r15
000000000426FFA      mov    [rsp+128h+var_120], rax
000000000426FFF      call  __ZNStC1EPlKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&
000000000427004      lea   rax, [rsp+128h+var_E8]
000000000427009      lea   rdi, [r15+8]
000000000427000      lea   rdx, [rsp+128h+var_F8]
000000000427012      mov    esi, offset aProcess ; "process"
000000000427017      mov    [rsp+128h+var_120], rax
00000000042701C      call  __ZNStC1EPlKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&
000000000427021      lea   rax, [rsp+128h+var_E8]
000000000427026      lea   rdi, [r15+10h]
00000000042702A      mov    esi, offset aKill ; "kill"
00000000042702F      mov    rdx, rax
000000000427032      mov    [rsp+128h+var_120], rax
000000000427037      call  __ZNStC1EPlKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&
00000000042703C      lea   rdi, [r15+18h]
-----

```

Figure 5. Terminating running VMs

```

0000000004378B5      mov    rdi, r12
0000000004378B8      mov    [rsp+0A8h+var_68], rax
0000000004378BD      call  sub_498940
0000000004378C2      mov    rbx, rax
0000000004378C5      mov    [rsp+0A8h+var_60], rax
0000000004378CA      call  __cxa_get_globals
0000000004378CF      mov    eax, [rax+8]
0000000004378D2      lea   rdi, [rbx+8]
0000000004378D6      mov    esi, offset aEsxiKillUms___ ; "[ESXi] Kill UMs....."
0000000004378DB      mov    [rsp+0A8h+var_58], eax
0000000004378DF      call  sub_418930
0000000004378E4      lea   rdi, [rbx+70h]
0000000004378E8      mov    esi, r13d
0000000004378EB      call  __ZNSolsEi ; std::ostream::operator<<(int)
0000000004378F0      mov    rdi, rbp

```

Figure 6. Reporting on VM killing status

## Encryption

The Linux variant of the DarkSide ransomware uses a [ChaCha20](#) stream cipher with RSA-4096 to encrypt targeted files on the victim machine.

It loops across the files on the root path mentioned in the embedded configuration or in the given parameter, as shown in Figure 7.

```

15  v4 = opendir(*a2);
16  v5 = v4;
17  if ( v4 )
18  {
19  LABEL_2:
20  while ( 1 )
21  {
22  v6 = readdir(v5);
23  if ( !v6 )
24  break;
25  while ( 1 )
26  {
27  v7 = v6->d_name;
28  if ( !memcmp(v6->d_name, ".", 2uLL) || !memcmp(v6->d_name, "..", 3uLL) )
29  break;
30  v8 = v6->d_type;
31  if ( v8 == 4 )
32  {
33  v11 = byte_8A2478;
34  std::string::assign((std::string *)&v11, (const std::string *)v3);
35  std::string::append((std::string *)&v11, "/");
36  std::string::append((std::string *)&v11, v7);
37  sub_435B80(v2, (const char **)&v11);
38  std::string::Rep::_M_dispose(v11 - 24, &v10);
39  goto LABEL_2;
40  }
41  if ( v8 != 8 )

```

Figure 7. Linux variant looping across files/directories

Before encryption, the ransomware performs a file size check to make sure that this is more than the minimum file size given in the embedded configuration or in the parameters.

```

125  v33 = std::operator<<<std::char_traits<char>>(v2 + 296, "[INFO] ");
126  v34 = (std::ostream *)std::operator<<<std::char_traits<char>>(v33, "File Size.....");
127  v35 = sub_5B0CB0(v34);
128  v36 = (std::ostream *)std::operator<<<std::char_traits<char>>(v35, "mb (");
129  v37 = sub_5B0CB0(v36);
130  v38 = std::operator<<<std::char_traits<char>>(v37, " Bytes)");
131  std::endl<char,std::char_traits<char>>(v38, " Bytes)", v39);
132  v40 = *(_QWORD *) (v2 + 120);
133  v41 = *(_QWORD *) (v2 + 128);
134  *(_QWORD *) (v2 + 152) = 0x100000LL;
135  v42 = v40 >> 20;
136  if ( v42 < v41 >> 20 )
137  {
138  if ( v42 < *(_QWORD *) (v2 + 160) )
139  {
140  LODWORD(v52) = 2;
141  v57 = &v52;
142  v47 = (const char *)sub_418430(v38, " Bytes)");
143  v48 = (_int64 **)&v53;
144  sub_4178B0(&v53, v47, &v57);
145  while ( v53 )
146  {
147  v49 = sub_418430(v48, v47);
148  sub_418610(&v57, v49, &v53);
149  v47 = "File Too Small, Ignored";

```

Figure 8. Linux variant performing a file size check

The malware then opens the target file, reads the content based on the part and space size given in the configuration or in the parameters, encrypts them, and writes to the file as shown in the following code:

```

184  do
185  {
186  std::istream::read((std::istream *)&v107, v68, v11); // Read_file
187  if ( v109 )
188  {
189  v54 = (std::runtime_error *)__cxa_allocate_exception(32LL);
190  std::string::string(&v59, "File Reading Failed", &v84);
191  sub_5B4100(&v59, "File Reading Failed");
192  v55 = *_errno_location();
193  sub_416B60(v54);
194  std::string::Rep::_M_dispose(v59 - 24, &v85);
195  __cxa_throw(v54, &off_8991C0, sub_5B4050);
196  }
197  Encryption_routine_sub_510EE0(&v88, v71, v68, v49 - (_QWORD)v68); // Encryption Routine
198  v10 = v71;
199  std::ostream::write((std::ostream *)&v102, v71, *(_QWORD *) (v2 + 152)); // Write_the_encrypted_data
200  if ( v106 )
201  {
202  v52 = (std::runtime_error *)__cxa_allocate_exception(32LL);
203  std::string::string(&v60, "File Writing Failed", &v84);
204  sub_5B4100(&v60, "File Writing Failed");
205  v53 = *_errno_location();
206  sub_416B60(v52);

```

Figure 9. File encryption

Unlike the Windows variant that randomly generates its custom Salsa20 matrix by calling “RtlRandomExW” several times, the malware uses the standard constant “expand 32-byte k” in the Chacha20 cipher used to encrypt files on the victim machine, as shown in the next figure.

```

00007FF158DBA4A0 65 78 70 61 6E 64 20 33 32 2D 62 79 74 65 20 6B expand-32-byte-k
00007FF158DBA4B0 4B 86 93 6C 50 E4 ED 94 67 03 86 CE B6 49 D0 E7 KãôlPSfüg.ã+!I-t
00007FF158DBA4C0 2B 8D 00 AD 3A 8C 6B 33 FF A9 10 25 CB F6 4C DD +...:îk3~.%-÷L!
00007FF158DBA4D0 00 40 00 00 00 00 00 00 BA 3D F3 43 08 8D 6F 18 .@.....!|=C..o.
00007FF158DBA4E0 00 01 00 00 00 00 00 00 FF FF FF FF FF FF FF 3F .....?
00007FF158DBA4F0 10 00 00 00 00 00 00 00 A0 A4 DB 58 F1 7F 00 00 .....ãñ!X±...
00007FF158DBA500 14 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00007FF158DBA510 F8 2D 8A 00 00 00 00 00 70 2D 8A 00 00 00 00 00 °-è.....p-è....
00007FF158DBA520 80 5E 00 4C F1 7F 00 00 80 5E 00 4C F1 7F 00 00 Ç^.L±...Ç^.L±...
00007FF158DBA530 80 5E 00 4C F1 7F 00 00 80 5E 00 4C F1 7F 00 00 Ç^.L±...Ç^.L±...
00007FF158DBA540 80 5E 00 4C F1 7F 00 00 7F 7E 00 4C F1 7F 00 00 Ç^.L±...~.L±...
00007FF158DBA550 A0 C6 2E 5A F1 7F 00 00 00 00 00 00 00 00 00 00 ä|.Z±.....
00007FF158DBA560 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00007FF158DBA570 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00007FF158DBA580 60 2D 00 4C F1 7F 00 00 01 00 00 00 00 00 00 00 `-.L±.....
00007FF158DBA590 1C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00007FF158DBA5A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00007FF158DBA5B0 80 5E 00 4C F1 7F 00 00 00 20 00 00 00 00 00 00 Ç^.L±.....
00007FF158DBA5C0 01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 .....
    
```

Figure 10. Using “expand 32-byte k” as a constant in the Chacha20 cipher

After encryption, the malware then adds a header and a cipher at the end of the encrypted files as shown in Figure 11.

```

89  *(_QWORD *)v27 = -7988539261020200278LL; ← Hardcoded header
90  v28 = v7;
91  std::ostream::write((std::ostream *)&v41, v27, 12LL);
92  if ( v42 )
93  {
94  v17 = (std::runtime_error *)__cxa_allocate_exception(32LL);
95  std::string::string(&v24, "Writing Header Failed", &v26);
96  sub_5B4100(&v24, "Writing Header Failed");
97  v18 = *_errno_location();
98  sub_416B60(v17);
99  std::string::_Rep::_M_dispose(v24 - 24, &v34);
100  __cxa_throw(v17, &0FF_0991C0, sub_5B4050);
101  }
102  std::ostream::write((std::ostream *)&v41, v32, v33 - (_QWORD)v32);
103  if ( v42 )
104  {
105  v15 = (std::runtime_error *)__cxa_allocate_exception(32LL);
106  std::string::string(&v25, "Cipher Writing Failed", &v26);
107  sub_5B4100(&v25, "Cipher Writing Failed");
    
```

Figure 11. Adding code to header



Figure 12. Hex view of the encrypted file

The ransomware output console shows the results of the encryption, the encrypted filenames, the discarded files after size check, the time of encryption, and more.

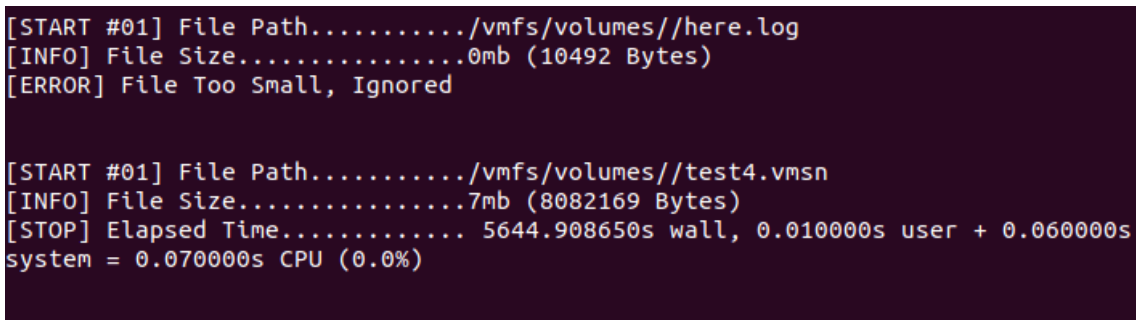


Figure 13. Ransomware output console

**Ransom note and added extensions**

The Linux variant drops a ransom note on the victim machine and adds a new file extension to the encrypted files.

Unlike the Windows variant, the ransom note file name and the new extension for encrypted files are hard-coded in the malware configuration file or given in a parameter, and the malware does not add any ID at the end of it.

For the analyzed samples, the new extension was **“.darkside”** and the hard-coded ransom note file name was **“darkside\_readme.txt”**.

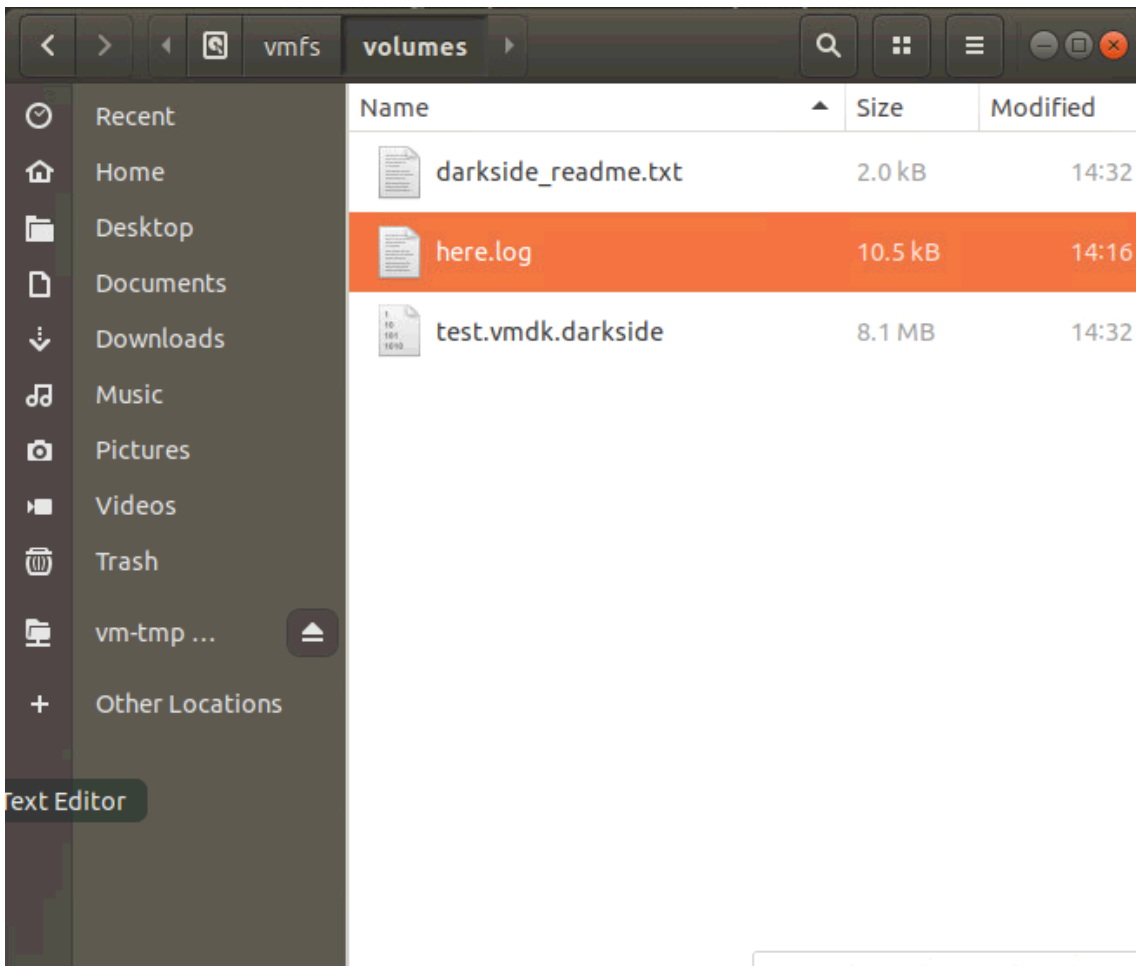


Figure 14. Encrypted folder with ransom note

### C&C Beaconing

The DarkSide ransomware can send a C&C beaconing message with the collected system information to a remote server hardcoded in the configuration. It collects system information on the victim machine, such as host name, domain, and disk information, as evidenced in Figure 15.

```

174  v91 = v8;
175  v8 += 8LL;
176  v92 = *(_DWORD *)(__cxa_get_globals() + 8);
177  sub_418930(v8, "username: ");
178  v7 = v119;
179  sub_418630(v8, v119, *(_DWORD *)v119 - 3));
180  v6 = (__int64 *)&v90;
181  sub_417930(&v90);
182  }
183  LODWORD(v117) = 0;
184  *(_DWORD *)&name.sysname[0] = &v117;
185  v9 = sub_418430(v6, v7);
186  v10 = &v125;
187  v11 = (char *)v9;
188  sub_417880(&v125, v9, &name);
189  while ( v125 )
190  {
191  v93 = sub_418430(v10, v11);
192  v12 = sub_498940(&v125);
193  v94 = v12;
194  v12 += 8LL;
195  v95 = *(_DWORD *)(__cxa_get_globals() + 8);
196  sub_418930(v12, "group: ");
197  v11 = v122;
198  sub_418630(v12, v122, *(_DWORD *)v122 - 3));
199  v10 = (__int64 *)&v93;

```

Figure 15. System information collection

The ransomware then puts the collected system information of the victim machine with a hard-coded UID value in the following format:

```
debug006:00007F5C200076C8 aIdAa21bbc2aa21abacab2cUid46017379a796803 db '{',0Ah
debug006:00007F5C200076C8 db '  "id": "aa21bbc2aa21abacab2c",',0Ah
debug006:00007F5C200076C8 db '  "uid": "46017379a796803",',0Ah
debug006:00007F5C200076C8 db '  "hostname": "ComputerNameUbuntu",',0Ah
debug006:00007F5C200076C8 db '  "domain": "-",',0Ah
debug006:00007F5C200076C8 db '  "version": "1.0",',0Ah
debug006:00007F5C200076C8 db '  "username": "username",',0Ah
debug006:00007F5C200076C8 db '  "group": "1000",',0Ah
debug006:00007F5C200076C8 db '  "os_type": "Linux",',0Ah
debug006:00007F5C200076C8 db '  "os_version": "Linux #42-Ubuntu SMP Tue Oct 23 15:48:01 UTC 2'
debug006:00007F5C200076C8 db '018",',0Ah
debug006:00007F5C200076C8 db '  "os_build": "4.15.0-39-generic",',0Ah
debug006:00007F5C200076C8 db '  "os_arch": "x86_64",',0Ah
debug006:00007F5C200076C8 db '  "disks": [' ,0Ah
debug006:00007F5C200076C8 db '    {' ,0Ah
debug006:00007F5C200076C8 db '      "MountPoint": "\/',',0Ah
debug006:00007F5C200076C8 db '      "Type": "ext4",',0Ah
debug006:00007F5C200076C8 db '      "Device": "\/dev/sda1",',0Ah
debug006:00007F5C200076C8 db '      "Size": "50138",',0Ah
debug006:00007F5C200076C8 db '      "Available": "38679",',0Ah
debug006:00007F5C200076C8 db '      "Free": "41255"',0Ah
debug006:00007F5C200076C8 db '    }',0Ah
debug006:00007F5C200076C8 db '  ]',0Ah
debug006:00007F5C200076C8 db '}',0Ah,0
debug006:00007F5C2000790F db 0
```

Figure 16. System information format

It hashes the collected information before sending it to the URL mentioned in the embedded configuration of the sample. DarkSide also uses a random parameter of eight characters in the request body to make its C&C traffic more difficult to detect by IPS/IDS devices on the victim network. The request body has the following format:

**<Random 8-character variable> = <Encrypted collected information> & <Random 8-character variable> = <hardcoded UID>**

Figure 17 shows the HTTP POST request sent by the malware to the remote server with the collected information.

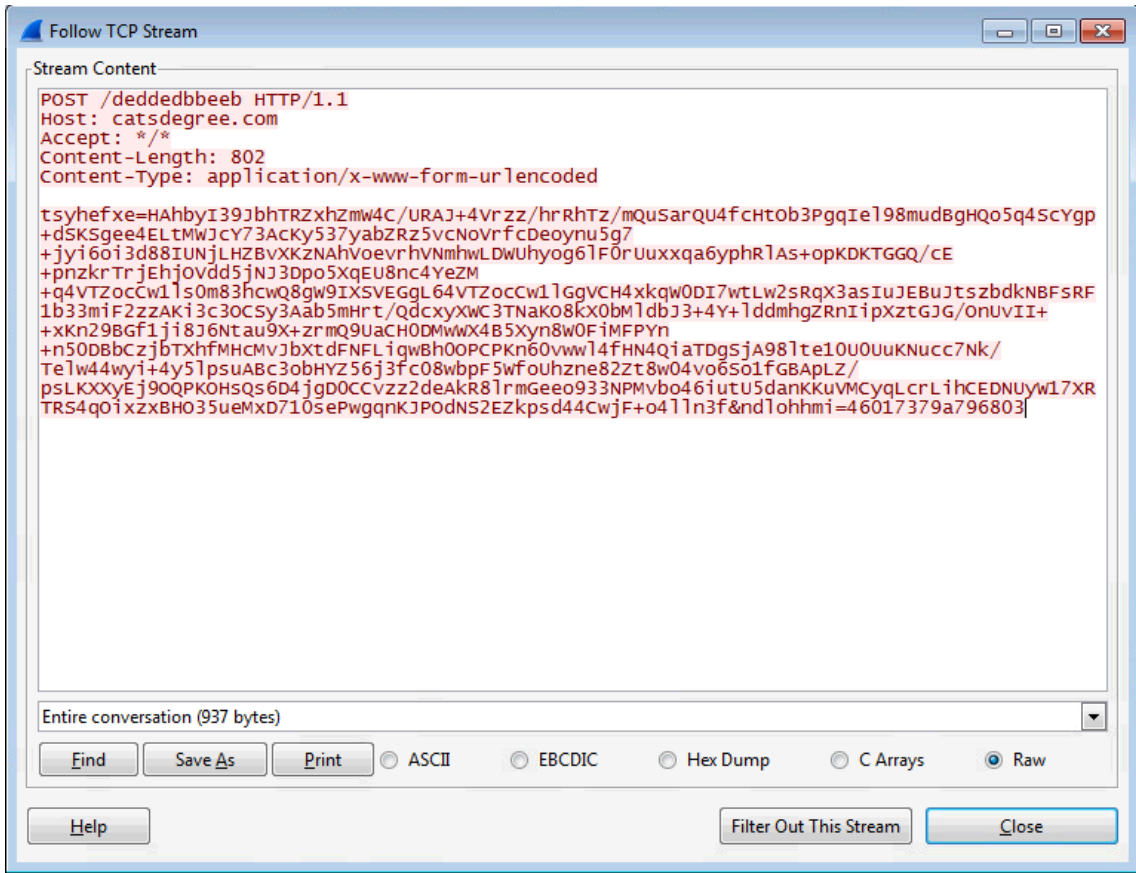


Figure 17. C2 beaconing HTTP traffic

Conclusion

The DarkSide ransomware family targets both Windows and Linux platforms. There are similarities between the Linux and Windows variants, but they are different with regard to some features, such as encryption mechanism, target files, ransom note name, extension, C&C URL, and more.

The Linux variant uses a ChaCha20 stream cipher with RSA-4096 in order to encrypt the files on the victim machine. It mainly targets VM-related files on VMWare ESXI servers, such as VMDK files. It can also accept parameters to infect more files on the victim machine. Additionally, the DarkSide ransomware runs ESXCLI commands to get vSAN and storage information on the victim machine. It also lists and kills running VMs on the infected ESXI host before encryption. Lastly, it drops a ransom note on the encrypted directories on the victim machine.

Indicators of Compromise

C&C servers:

- catsdegree[.]com
- securebestapp20[.]com
- temisleyes[.]com

SHA256	Trend Micro Detection Name
--------	----------------------------

984ce69083f2865ce90b48569291982e786980aeef83345953276adfcbbbeece8	Ransom.Linux.DARKSIDE.THDBGBA
9cc3c217e3790f3247a0c0d3d18d6917701571a8526159e942d0fffb848acffb	
c93e6237abf041bc2530ccb510dd016ef1cc6847d43bf023351dce2a96fdc33b	
da3bb9669fb983ad8d2ffc01aab9d56198bd9cedf2cc4387f19f4604a070a9b5	

## Tags

---

Source: [https://www.trendmicro.com/en\\_us/research/21/e/darkside-linux-vms-targeted.html](https://www.trendmicro.com/en_us/research/21/e/darkside-linux-vms-targeted.html)