



Black basta Ransomware Goes Cross-Platform, Now Targets ESXi Systems

Original research by [Siddharth Sharma](#) and [Nischay Hegde](#)

The Uptycs Threat research team recently observed an advancement in the Black basta ransomware, where we saw that the ransomware binaries are now targeting ESXi servers. The Black Basta was first seen this year during the month of April, in which its variants targeted windows systems. This blog highlights the recent addition of the *nix component in the Black Basta ransomware by the ransomware authors.

Threat Attribution

Based on the chat support link and encrypted file extension, we believe that the actors behind this campaign are the same who targeted windows systems earlier with the Black Basta ransomware.

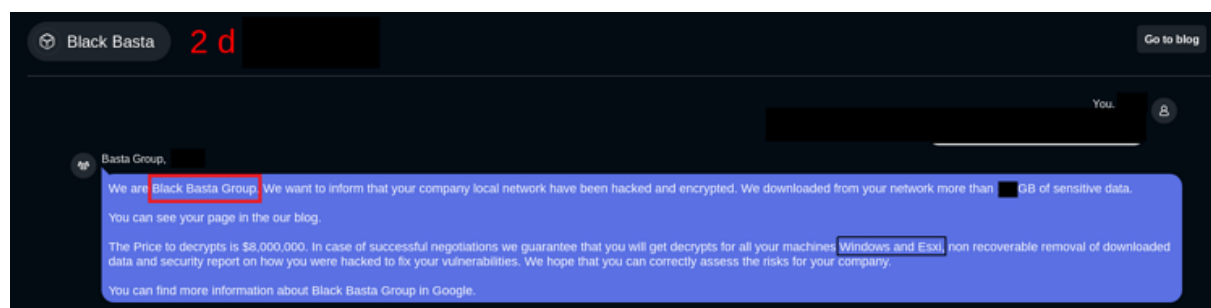


Figure 1: Black basta chat support panel for negotiation

Technical Overview

The ransomware binary(hash: 0d6c3de5aebbbe85939d7588150edf7b7bdc712fceb6a83d79e65b6f79bfc2ef) looks for the /vmfs/volumes directory for encryption in the victim system. The /vmfs/volumes directory stores the virtual machines on the ESXi server. Once it finds the directory it starts encrypting files present inside the volumes folder.

```

loc_40AA48:
lea    rdx, [rsp+0F8h+var_DC]
lea    rdi, [rsp+0F8h+src]
mov    esi, offset aVmfsVolumes ; "/vmfs/volumes"
call   __ZNStC1EPKcRKSaIcE ; std::string::string(char const*,std::allocator<char> const&)
mov    rbx, [rsp+0F8h+src]
mov    r12, [rbx-18h]
lea    rax, [rbx+r12]
cmp    rbx, rax
jz     loc_40AD20

```

Figure 2: Ransomware binary looking for /vmfs/volumes folder

For encryption the ransomware author seems to be using the chacha20 algorithm as a part of the encryption mechanism, probably because chacha20 is fast.

```

Decompile: EncryptFileBytes -
130  undefined local_53;
131  undefined local_52;
132  undefined local_51;
133  undefined local_50;
134  undefined local_4f;
135  undefined local_4e;
136  undefined local_4d;
137  undefined local_4c;
138  undefined local_4b;
139  undefined local_4a;
140  undefined local_49;
141  ulong local_48;
142
143  local_c8 = 0x61707865;
144  local_c4 = 0x3320646e;
145  local_c0 = 0x79622d32;
146  local_bc = 1797285236;
147  local_b8 = (uint)dest[3] << 0x18 | (uint)*dest | (uint)dest[2] << 0x10 | (uint)dest[1] << 8;
148  local_b4 = (uint)dest[7] << 0x18 | (uint)dest[4] | (uint)dest[6] << 0x10 | (uint)dest[5] << 8;
149  local_b0 = (uint)dest[11] << 0x18 | (uint)dest[8] | (uint)dest[10] << 0x10 | (uint)dest[9] << 8;
150  local_ac = (uint)dest[15] << 0x18 | (uint)dest[12] | (uint)dest[14] << 0x10 | (uint)dest[13] << 8;
151  local_98 = 0;
152  local_94 = 0;
153  local_48 = 0x40;
154  local_a8 = (uint)dest[0x13] << 0x18 |
155           (uint)dest[0x10] | (uint)dest[0x12] << 0x10 | (uint)dest[0x11] << 8;
156  local_a4 = (uint)dest[0x17] << 0x18 |
157           (uint)dest[0x14] | (uint)dest[0x16] << 0x10 | (uint)dest[0x15] << 8;
158  local_a0 = (uint)dest[0x1b] << 0x18 |
159           (uint)dest[0x18] | (uint)dest[0x1a] << 0x10 | (uint)dest[0x19] << 8;
160  local_9c = (uint)dest[0x1f] << 0x18 |
161           (uint)dest[0x1c] | (uint)dest[0x1e] << 0x10 | (uint)dest[0x1d] << 8;
162  local_90 = (uint)dest_0[3] << 0x18 |
163           (uint)*dest_0 | (uint)dest_0[2] << 0x10 | (uint)dest_0[1] << 8;
164  local_8c = (uint)dest_0[7] << 0x18 |
165           (uint)dest_0[4] | (uint)dest_0[6] << 0x10 | (uint)dest_0[5] << 8;

```

'expand 32-byte k'

Figure 3: chacha20 algorithm

It also uses multithreading for encryption to utilize multiple processors and further make it faster and harder to detect. As shown in below figure(see figure: 4), the function `EncryptionThread` is run in parallel to increase throughput of the ransomware.

```

C:\Decompile: EncryptFolder -
150 if (local_158[0] + -0x18 !=
151     std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Rep::
152     _S_empty_rep_storage) {
153     std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Rep::_M_dispose
154         (local_158[0] + -0x18);
155 }
156 iVar4 = std::thread::hardware_concurrency();
157 uVar3 = iVar4 * 3;
158 if (iVar4 * 3 < minimumThreads) {
159     uVar3 = minimumThreads;
160 }
161 uVar18 = 0;
162 if (uVar3 != 0) {
163     do {
164         local_168[0] = 0;
165         // try { // try from 0040952c to 00409530 has its CatchHandler @ 0040a613
166         // } // end try from 0040952c to 00409530
167         puVar8 = operator.new(0x30);
168         *(puVar8 + 1) = 1;
169         *(puVar8 + 0xc) = 1;
170         *puVar8 = &PTR_~_Sp_counted_ptr_inplace_004256d0;
171         puVar8[3] = 0;
172         puVar8[4] = 0;
173         puVar8[2] = &PTR_~_Impl_00425650;
174         puVar8[5] = EncryptionThread; // function that is executed in every thread
175         local_88 = std::
176             _Sp_counted_ptr_inplace<std::thread::_Impl<std::_Bind_simple<void(*)()>>,std::a
177             llocator<std::thread::_Impl<std::_Bind_simple<void(*)()>>>,(_gnu_cxx::_Lock_po
178             licy)2>
179             ::_M_get_deleter(puVar8,&std::_Sp_make_shared_tag::typeinfo);
180         local_80 = puVar8;
181         // try { // try from 00409593 to 00409597 has its CatchHandler @ 0040a62f
182         // } // end try from 00409593 to 00409597
183         std::thread::_M_start_thread(local_168); // Start the thread
184         if (local_80 != 0x0) {
185             std::_Sp_counted_base<(_gnu_cxx::_Lock_policy)2>::_M_release(local_80);
186         }
187     }
188 }

```

Figure 4: EncryptionThread usage

The ransomware binary also uses chmod utility for giving full permissions to the target files.(see figure 5)

```

chmod("/vmfs/volumes/ff.doc", 0777) = 0
getcwd("/home /intel/sgo.txt_/tt", 4096) = 31
openat(AT_FDCWD, "/vmfs/volumes/ff.doc", 0_RDWR) = 10</vmfs/volumes/ff.doc>
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_END) = 0
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_END) = 0
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_CUR) = 0
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_SET) = 0
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_SET) = 0
lseek(10</vmfs/volumes/ff.doc>, 0, SEEK_END) = 0
write(10</vmfs/volumes/ff.doc>, "+V|2|;;\37H|251\|-
23'\350\3060[\240\204a(\4\244,\177W\222\36\17\274Y\3\261,\324s\366G\234d\357f\304\31\241L\367Y^\215\254-
,\345Z4\336` \230\372^
ib_\23;\316M\371\263\342\204\6e\220f\37\361<\215\253\3133\363e\4\216\204L\252w{\3A\221\223i{\266\3179\227\t\246\
109b\324\2010|\347\235\316>*\320\361F\325\376\v\305\201\273\10\2\322\246\16\220\260\371\300\256\343\266[\rh()-
56\301\3154\233` \24\342.\245\365\242\330ma\231\315\342\351\3%}S\263\300\253\330\333<)-

```

Figure 5: Malware binary writing encrypted content to files inside volumes folder

Below figure shows the encrypted files inside the volumes folder in the victim system. The extension used by the ransomware binary is .basta.

```
root /vmfs/volumes# ls
bb.xlsx.basta 'IDA Freeware 7.6.desktop.basta' readme.txt
bcc           kk.txt.basta  ssd1.pcap.basta
d1e          ll.txt.basta  sss.jpeg.basta
dd.docx.basta logo.png.basta testing.elf.basta
debugf.py.basta pp.elf.basta
ff.doc.basta pp.txt.basta

root(/ :/vmfs/volumes# cat readme.txt
Your data are stolen and encrypted
The data will be published on TOR website if you do not pay the ransom
You can contact us and decrypt one file for free on this TOR site
(you should download and install TOR browser first https://torproject.org)
https://aazsbsgya565vlu2c6bzy6yfiebkcbtvvctvolt33s77xypi7nypxyd.onion/

Your company id for log in: 01e

root(/ :/vmfs/volumes# █
ENCRYPTION
Done time: 14.5620 seconds, encrypted: 0.0016 gb
```

Figure 6: Encrypted files along with the readme file

Inside the readme.txt file, the author puts the link to the chat support panel where the victims can approach for file decryption.

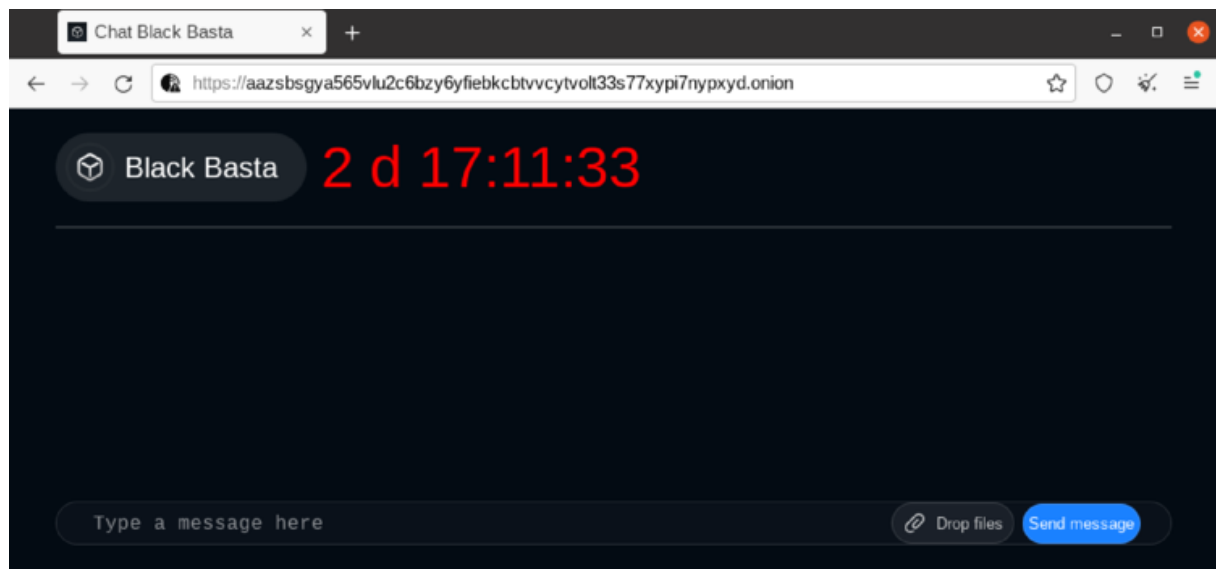


Figure 7: Black Basta panel for chat support

Uptycs EDR detections

The Uptycs EDR armed with YARA process scanning detects the BlackBasta ransomware with a threat score of 10/10.(see figure 8)

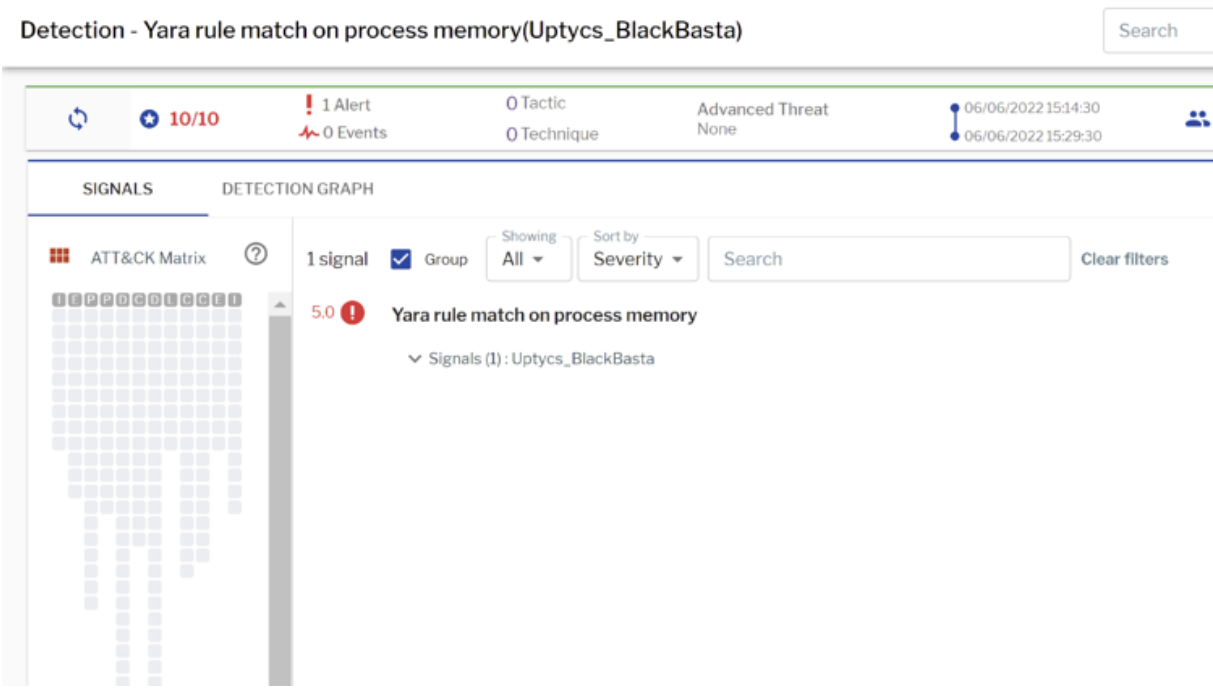


Figure 8: Uptycs EDR detection

IOCs

0d6c3de5aebb85939d7588150edf7b7bdc712fceb6a83d79e65b6f79bfc2ef
https[:]//aazsbsgya565vlu2c6bzy6yfielkcbtvvcyvtolt33s77xypi7nypxyd[.]oni
on