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.

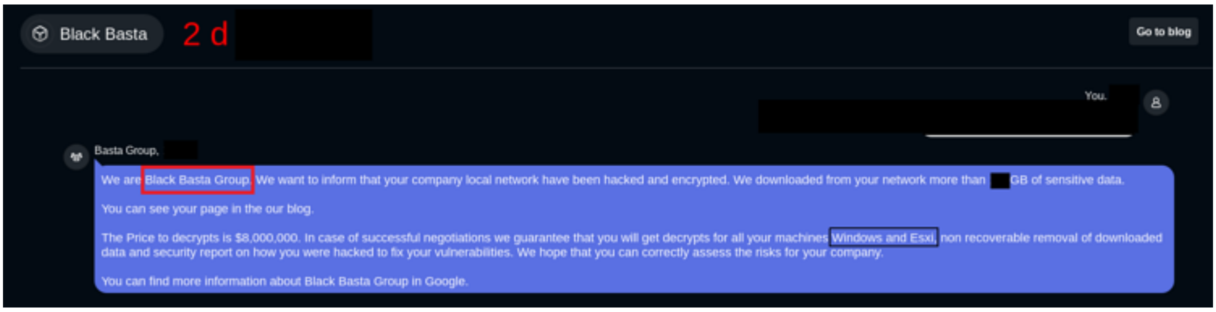

Figure 1: Black basta chat support panel for negotiation

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.

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.

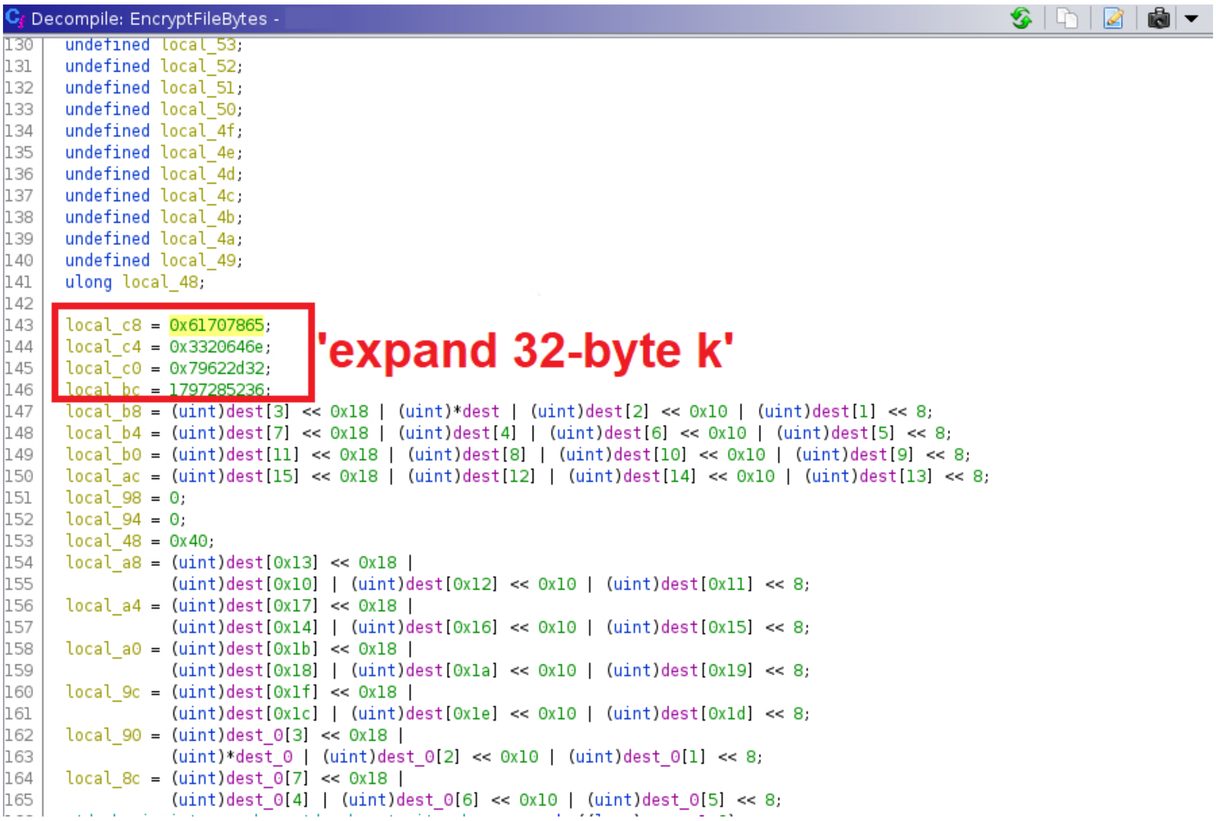

Figure 3: chacha20 algorithm

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.

```
Decompile: EncryptFolder -
    if (local_158[0] + -0x18 !=
        std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Rep::
        _S_empty_rep_storage) {
      std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Rep::_M_dispose
                ((local_158[0] + -0x18);
    }
    iVar4 = std::thread::hardware_concurrency();
    uVar3 = iVar4 * 3;
    if (iVar4 * 3 < minimumThreads) {
      uVar3 = minimumThreads;
    }
    uVar18 = 0;
    if (uVar3 != 0) {
      do {
        local_168[0] = 0;
                    // try { // try from 0040952c to 00409530 has its CatchHandler @ 0040a613
                    // } // end try from 0040952c to 00409530
        puVar8 = operator.new(0x30);
        *(puVar8 + 1) = 1;
        *(puVar8 + 0xc) = 1;
        *puVar8 = &PTR_~_Sp_counted_ptr_inplace_004256d0;
        puVar8[3] = 0;
        puVar8[4] = 0;
        puVar8[2] = &PTR_~_Impl_00425650;
        puVar8[5] = EncryptionThread; // function that is executed in every thread
        local_88 = std::
                   _Sp_counted_ptr_inplace<std::thread::_Impl<std::_Bind_simple<void(*())()>>,std::a
                   llocator<std::thread::_Impl<std::_Bind_simple<void(*())()>>>,(__gnu_cxx::_Lock_po
                   licy)2>
                   ::_M_get_deleter(puVar8,&std::_Sp_make_shared_tag::typeinfo);
        local_80 = puVar8;
                    // try { // try from 00409593 to 00409597 has its CatchHandler @ 0040a62f
                    // } // end try from 00409593 to 00409597
        std::thread::_M_start_thread(local_168); // Start the thread
        if (local_80 != 0x0) {
          std::_Sp_counted_base<(__gnu_cxx::_Lock_policy)2>::_M_release(local_80);
        }
```

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", O_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\\-        Encrypted data getting written to .doc file inside
                                                         volumes folder
23`\350\306O[\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^
lb_\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
409b\324\201@|\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.

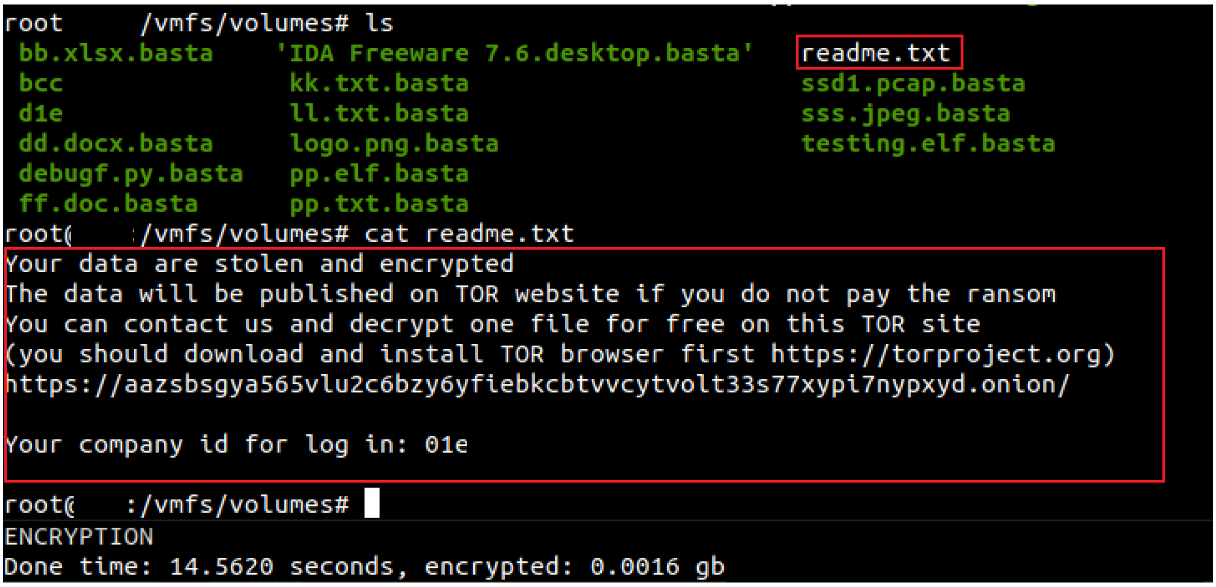

Figure 6: Encrypted files along with the readme file

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

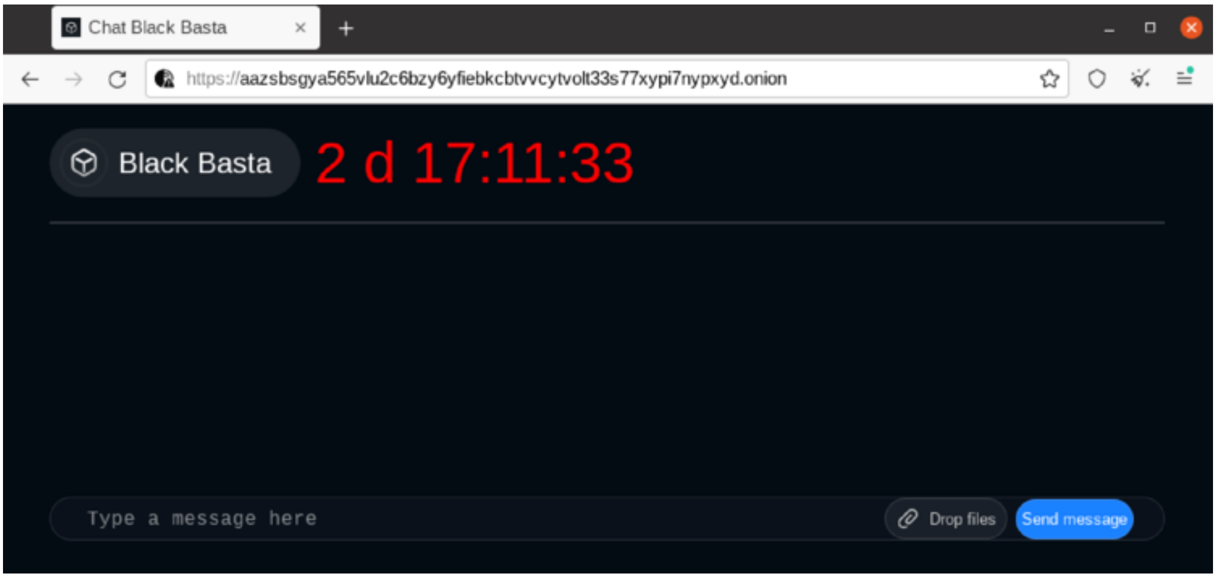

Figure 7: Black Basta panel for chat support

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)

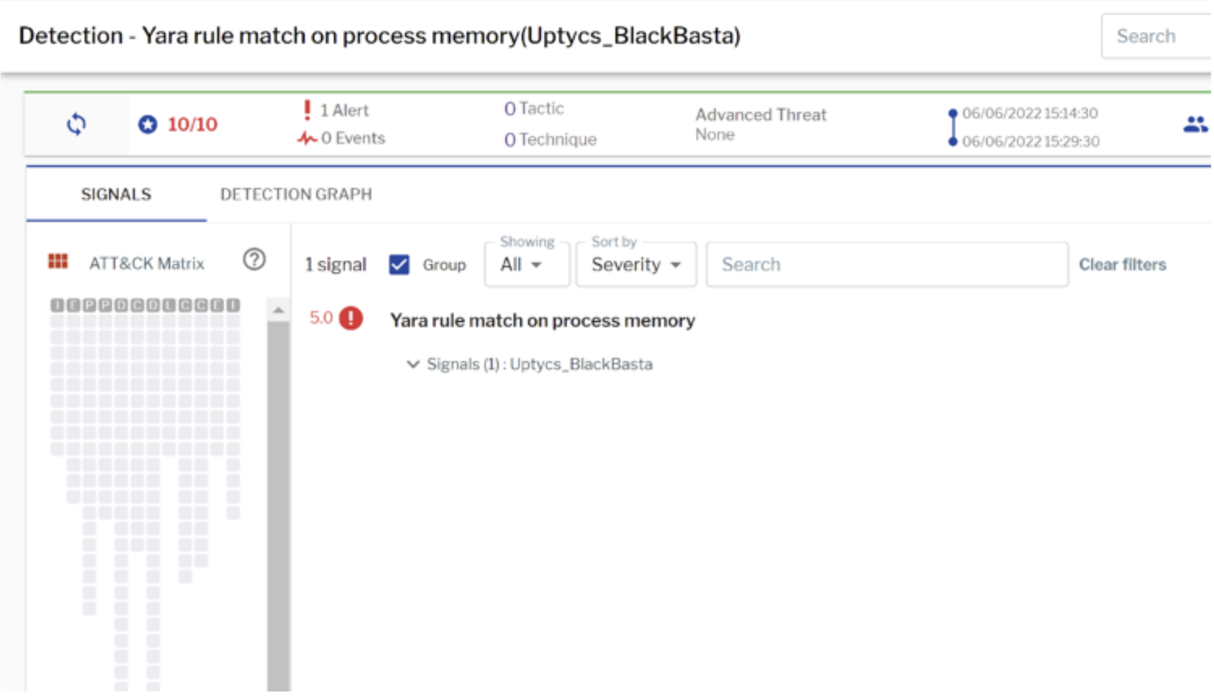

Figure 8: Uptycs EDR detection

IOCs

0d6c3de5aebbbe85939d7588150edf7b7bdc712fceb6a83d79e65b6f79bfc2ef
https[:]//aazsbsgya565vlu2c6bzy6yfiebkcbtvvcytvolt33s77xypi7nypxyd[.]onion