Main subroutine

The screenshot below shows the main function, we can see this is a quite basic main function that only calls 1 function `GetRemoteProcessHandle()` which we will reverse next before we continue with the rest of the `main()` subroutine.

First it's going to start with the prologue, making space onto the stack for the parameters/variables.

The main function will load the handle to the process, and handle to the PID. In order to know what will happen when those parameters are provided we have to dive into the `GetRemoteProcessHandle()` subroutine.

```
; int __cdecl main(int argc, const char **argv, const char **envp)
main proc near

Pid= dword ptr  8
hProcess= qword ptr  10h

sub     rsp, 40
xor     eax, eax        ; eax hold 0
lea     r8, [rsp+28h+hProcess] ; void **
lea     rdx, [rsp+28h+Pid] ; unsigned int *
mov     [rsp+28h+Pid], eax ; moves eax register into memory location by offset 28 also add  the value of Pid to RSP (Stackpointer)
mov     [rsp+28h+hProcess], rax ; moves rax register into memoryby offset 28, add value of hProcess to RSP
call    GetRemoteProcessHandle
test    eax, eax
jnz     short loc_14000130C
```

```
mov     eax, 0FFFFFFFFh
add     rsp, 28h
retn
```

```
loc_14000130C:
mov     r8d, [rsp+28h+Pid]
lea     rdx, aSvchostExe ; "svchost.exe"
lea     rcx, aFoundSOfPidD ; "[+] FOUND \"%s\" - Of Pid : %d \n"
call    wprintf
lea     rcx, aPressEnterToQu ; "[#] Press <Enter> To Quit ... "
call    printf
call    cs:__imp_getchar
xor     eax, eax
add     rsp, 28h
retn
main endp
```

GetRemoteProcessHandle

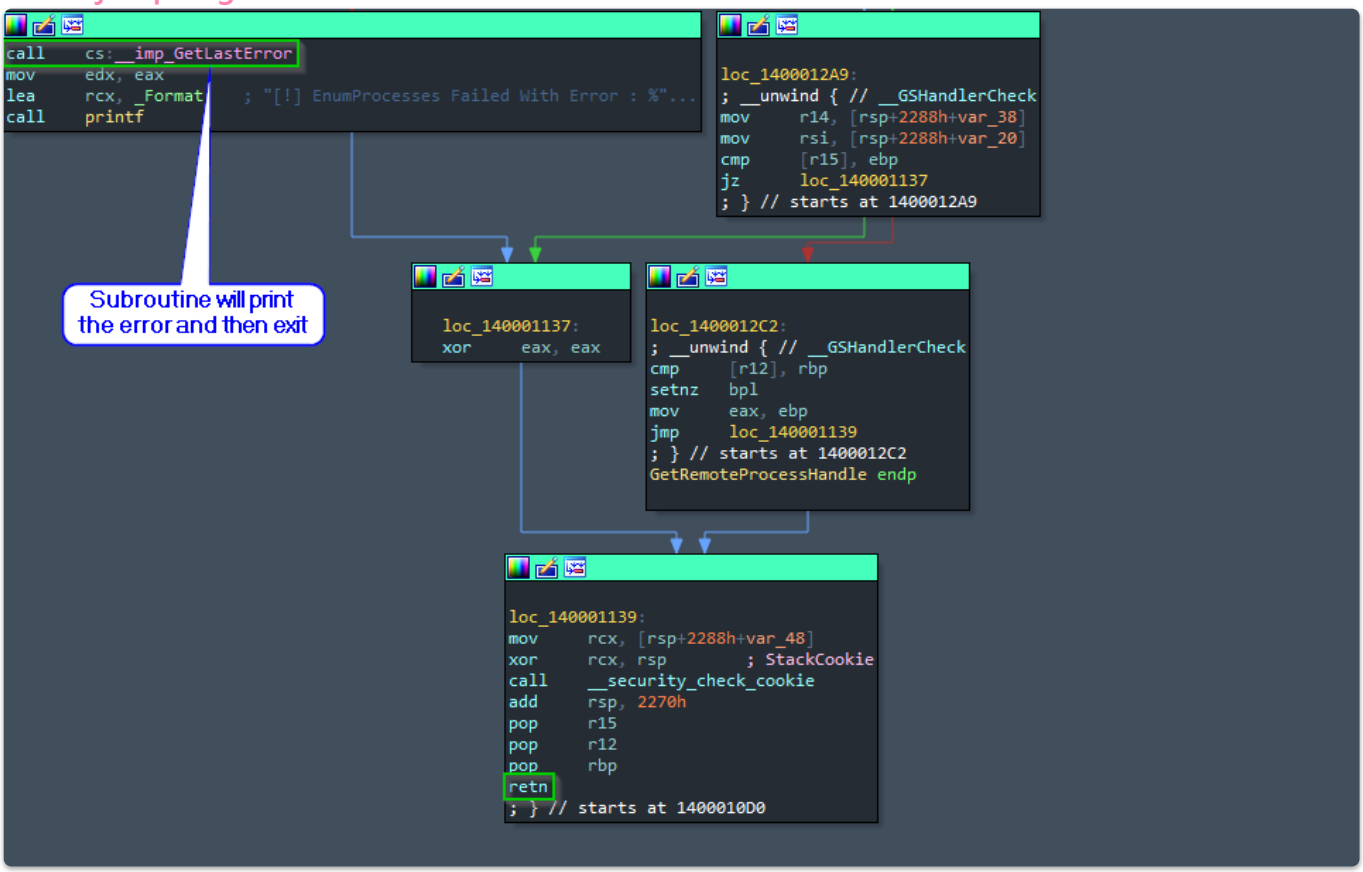
The function is going to make a call to EnumProcess which allows the program to gather processes that are present on the target machine.

```
; int __fastcall GetRemoteProcessHandle(const wchar_t *, unsigned int *, void **)
GetRemoteProcessHandle proc near

hModule= qword ptr -2268h
cbNeeded= dword ptr -2260h
var_225C= dword ptr -225Ch
idProcess= dword ptr -2258h
BaseName= word ptr -258h
var_48= qword ptr -48h
var_38= qword ptr -38h
var_30= qword ptr -30h
var_28= qword ptr -28h
var_20= qword ptr -20h
arg_0= qword ptr  8

; __unwind { // __GSHandlerCheck
push    rbp
push    r12
push    r15
mov     eax, 2270h
call    _alloca_probe
sub     rsp, rax
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+2288h+var_48], rax
xor     ebp, ebp
lea     rcx, [rsp+2288h+idProcess] ; lpidProcess
                        ; Load the address to store the process ID
mov     r12, r8
mov     [rsp+2288h+cbNeeded], ebp ; Initialize the bytes required to store PIDs
mov     r15, rdx
mov     [rsp+2288h+var_225C], ebp
lea     r8, [rsp+2288h+cbNeeded] ; lpcbNeeded
mov     [rsp+2288h+hModule], rbp
mov     edx, 8192       ; cb
                        ; moves 9182 bytes as size of the PIDs array
call    cs:__imp_K32EnumProcesses
test    eax, eax
jnz     short loc_140001156
```

Based on the return value, it will either exit the program or will continue with the next routine.

Subroutine Flow

When jump flag has the value: 1



Subroutine Flow

When jump flag has the value: 0

Print Number Processes

```
loc_140001156:          ; This routine will print the number of PIDs found
; __unwind { // __GSHandlerCheck
mov     [rsp+2288h+var_20], rsi
lea     rcx, aINumberOfProce ; "[i] Number Of Processes Detected : %d "...
mov     [rsp+2288h+var_38], r14
mov     r14d, [rsp+2288h+cbNeeded] ; Load num of bytes required to store PIDs
shr     r14d, 2         ; Devides value r14d by 2 giving the number of DWORD (32 but unsigned)
mov     edx, r14d       ; Moves num of PIDs to edx
call    printf
mov     esi, ebp
test    r14d, r14d
jz      loc_1400012A9
; } // starts at 140001156
```

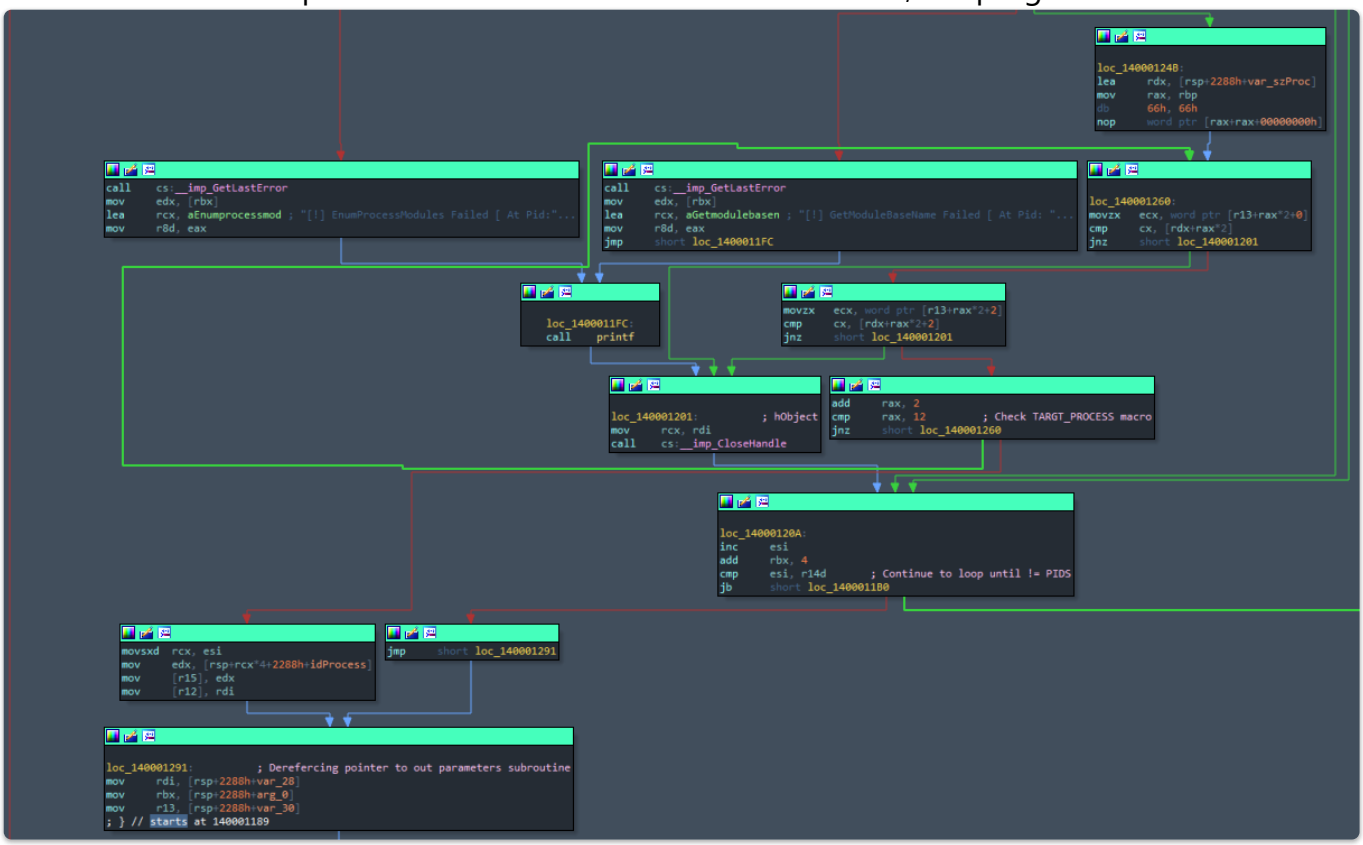
Seeking to Find Target Process

It irates over each process, with full access. After that the it will gather `GetModuleBaseName()` because that is required by the next called WinAPI.

```
loc_1400011B0:          ; dwProcessId
mov     r8d, [rbx]      ; Start for-loop over PIDs, that are collected EnumProcess routine
test    r8d, r8d
jz      short loc_14000120A
```

```
xor     edx, edx        ; bInheritHandle
                        ; EDX = 0 == False
mov     ecx, 1FFFFFh    ; dwDesiredAccess
                        ; PROCESS_ALL_ACCESS
call    cs:__imp_OpenProcess
mov     rdi, rax
test    rax, rax
jz      short loc_14000120A
```

It will continue to loop until there is a match if there is no match, the program will exit.




```
__int64 __fastcall GetRemoteProcessHandle(const wchar_t *a1, unsigned int *p_dwPid,
void **p_hProcess)
{
  unsigned int b_TRUE; // ebp
  DWORD LastError; // eax
  unsigned int v8; // r14d
  unsigned int v9; // esi
  DWORD *v10; // rbx
  HANDLE v11; // rax
  void *hProcess; // rdi
  DWORD v13; // eax
```

```
  DWORD v14; // eax
  __int64 v15; // rax
  HMODULE hModule; // [rsp+20h] [rbp-2268h] BYREF
  unsigned int cbNeeded; // [rsp+28h] [rbp-2260h] BYREF
  unsigned int v18; // [rsp+2Ch] [rbp-225Ch] BYREF
  int idProcess[2048]; // [rsp+30h] [rbp-2258h] BYREF
  __int16 var_szProc[264]; // [rsp+2030h] [rbp-258h] BYREF

  b_TRUE = 0;
  cbNeeded = 0;
  v18 = 0;
  hModule = 0i64;
  if ( !K32EnumProcesses((DWORD *)idProcess, 0x2000u, &cbNeeded) )
  {
    LastError = GetLastError();
    printf("[!] EnumProcesses Failed With Error : %d \n", LastError);
    return 0i64;
  }
  v8 = cbNeeded >> 2;
  printf("[i] Number Of Processes Detected : %d \n", cbNeeded >> 2);
  v9 = 0;
  if ( v8 )
  {
    v10 = (DWORD *)idProcess;
    do
    {
      if ( *v10 )
      {
        v11 = OpenProcess(0x1FFFFFu, 0, *v10);
        hProcess = v11;
        if ( v11 )
        {
          if ( K32EnumProcessModules(v11, &hModule, 8u, &v18) )
          {
            if ( K32GetModuleBaseNameW(hProcess, hModule, (LPWSTR)var_szProc,
0x104u) )
            {
              v15 = 0i64;
              while ( aSvchostExe[v15] == var_szProc[v15] && aSvchostExe[v15 + 1]
== var_szProc[v15 + 1] )
              {
                v15 += 2i64;
                if ( v15 == 12 )
                {
                  *p_dwPid = idProcess[v9];
                  *p_hProcess = hProcess;
```

```
                goto LABEL_20;
              }
            }
          }
          else
          {
            v14 = GetLastError();
            printf("[!] GetModuleBaseName Failed [ At Pid: %d ] With Error : %d
\n", *v10, v14);
          }
        }
        else
        {
          v13 = GetLastError();
          printf("[!] EnumProcessModules Failed [ At Pid: %d ] With Error : %d
\n", *v10, v13);
        }
        CloseHandle(hProcess);
      }
    }
    ++v9;
    ++v10;
  }
  while ( v9 < v8 );
}
LABEL_20:
  if ( !*p_dwPid )
    return 0i64;
  LOBYTE(b_TRUE) = *p_hProcess != 0i64;
  return b_TRUE;
}
```