

# 기술문서

## DKOM 탐지 기법

행정3 김범연

ccibomb@gmail.com

<http://ccibomb.tistory.com>

2009.9. 1.



## ○ 목차

### I. What is "DKOM" ?

- 가. DKOM이란?
- 나. DKOM의 장단점
- 다. DKOM의 기능

### II. DKOM을 이용한 은닉

- 가. 원리
- 나. 실습 (notepad.exe 은닉)

### III. DKOM 탐지 기법

- 가. 다른 Linked list 이용하기
  - 1. 원리
  - 2. 실습 (Windbg 이용 / tool 이용)
- 나. Process Carving
  - 1. 원리
  - 2. 사전지식
  - 3. 실습 (Windbg 이용 / tool 이용)

### IV. 참고문헌

# I. What is “DKOM” ?

## 가. DKOM 이란?

우리는 흔히 루트킷을 위하여 후킹 기술을 활용하는 보안제품이나 악성코드를 볼 수 있다. 루트킷을 운영체제의 일부분으로 만들 수 없기 때문에 운영체제를 후킹하는 것은 효과적인 방법이다. 그러나 유행처럼 SSDT 후킹기법을 사용함에 따라 이제는 최신기술이 아닌 범용 기술이 되었고 탐지가 쉽다는 문제에 걸렸다.

최근에는 실행 코드 자체만 고민하던 해커들이 실행 코드보다는 코드가 사용하는 데이터에 집중하면서 DKOM (Direct Kernel Object Manipulation)이라는 기법이 새롭게 생겨났다. DKOM 은 의미 그대로 커널 오브젝트를 직접 조작하는 방법이다. 이 기법은 실행 코드가 사용하는 Data(프로세스 목록을 저장하기 위한 링크드 리스트 등)를 변경하여 자신을 감추는 방법 등으로 활용되고 있다.

DKOM을 사용해 프로세스를 숨기는 방법을 간단히 설명하면 다음과 같다. 먼저, 윈도우에서는 Process 목록을 링크드 리스트를 사용하여 관리하게 되는데, 이 링크드 리스트의 각 항목들을 가지고 연결 고리를 조작하는 방법이다.

## 나. DKOM의 장점과 단점

장점 : DKOM을 탐지하기 어렵다.

(일반적인 환경에서는 프로세스나 토큰과 같은 커널 오브젝트 변경은 커널의 오브젝트 관리자를 통해 이루어진다. 그러나 DKOM은 오브젝트 관리자를 거치지 않고 직접 커널 오브젝트에 접근하므로, 관리자에 대해 어떤 권한 체크도 이루어지지 않는다.)

단점 : 커널이 메모리상에서 관리하고 운용하는 오브젝트만을 이용할 수 있다.

(따라서 파일 은닉을 수행하려면 후킹이나 파일시스템 필터 드라이버와 같은 보다 전통적인 방법을 사용해야 한다.)

커널 오브젝트를 변경하기 전, 알아야 할 것들이 많다.

(운영체제의 버전에 따른 오브젝트 변경 여부, 오브젝트가 사용되는 운영체제의 상태, 커널이 해당 오브젝트를 어떻게 사용하는지, 오브젝트는 어떤 구조로 이뤄졌으며 그 구조체 안의 각 멤버들은 어떤 것들인지 등)

## 다. DKOM 의 기능

프로세스 은닉,

드라이버 은닉,

네트워크 포트 은닉,

스레드(Thread) 권한 상승,

프로세스 권한 상승,

포렌식 회피(안티 리버싱) 등의 작업을 수행할 수 있다.

\* 이 중 이번 문서에서는 프로세스 은닉과 탐지에 초점을 맞추어 자세히 알아보기로 한다.

## II. DKOM을 이용한 은닉

### 가. 원리

모든 운영체제는 각종 정보를 구조체나 오브젝트 형태로 메모리상에 저장한다. 유저 모드 프로세스가 현재 동작중인 프로세스나 스레드, 또는 디바이스 드라이버 정보를 요청하면 운영체제는 그에 해당하는 오브젝트를 이용해서 정보를 전달한다. 따라서 그런 오브젝트들은 모두 메모리상에 존재하며, 그 내용을 직접 변경하는 것이 가능하다.

윈도우 NT/2000/XP/2003 운영체제는 프로세스와 스레드 정보를 실행 오브젝트로 저장한다. Taskmgr.exe 같은 툴들은 이 오브젝트를 이용해서 현재 실행중인 프로세스 정보를 구한다.

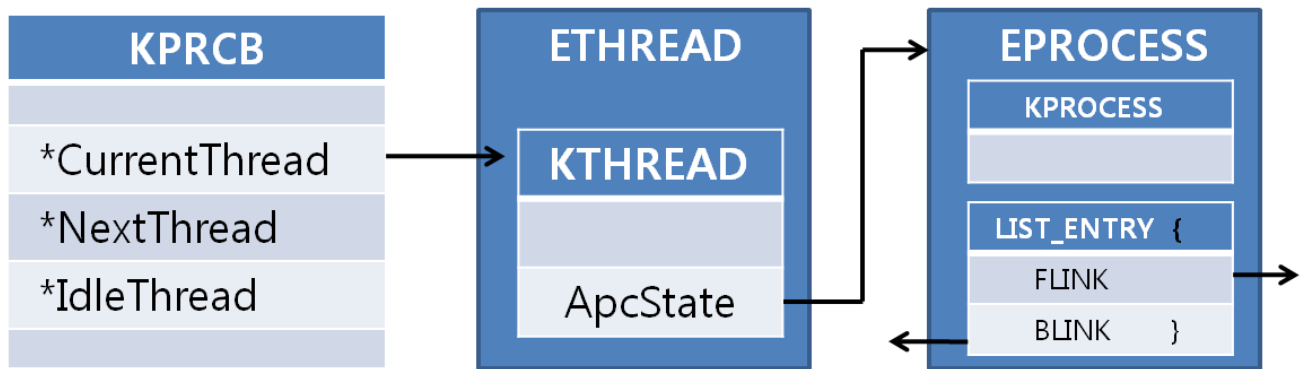
위와 같은 방법으로 구동되고 있는 프로세스 리스트를 알아내기 위한 함수가 바로 **ZwQuery SystemInformation()** 함수이다. 프로세스 리스트는 EPROCESS 구조체라는 Doubly Linked List 형태로 구성되어 있다. 즉, FLINK와 BLINK를 멤버로 하는 LIST\_ENTRY 구조체를 가지고 있다. 현재 구동중인 프로세스의 주소(EPROCESS 구조체)는 PsGetCurrentProcess 함수를 호출하여 찾을 수 있다. 이 함수는 실제 **IoGetCurrentProcess** 함수와 같은데, 이 함수를 디스어셈블하면 다음과 같다.

```

mov  eax, fs:0x00000124; // 현재의 ETHREAD 구조체의 주소
mov  eax, [eax + 0x44]; // _EPROCESS의 오프셋
ret

```

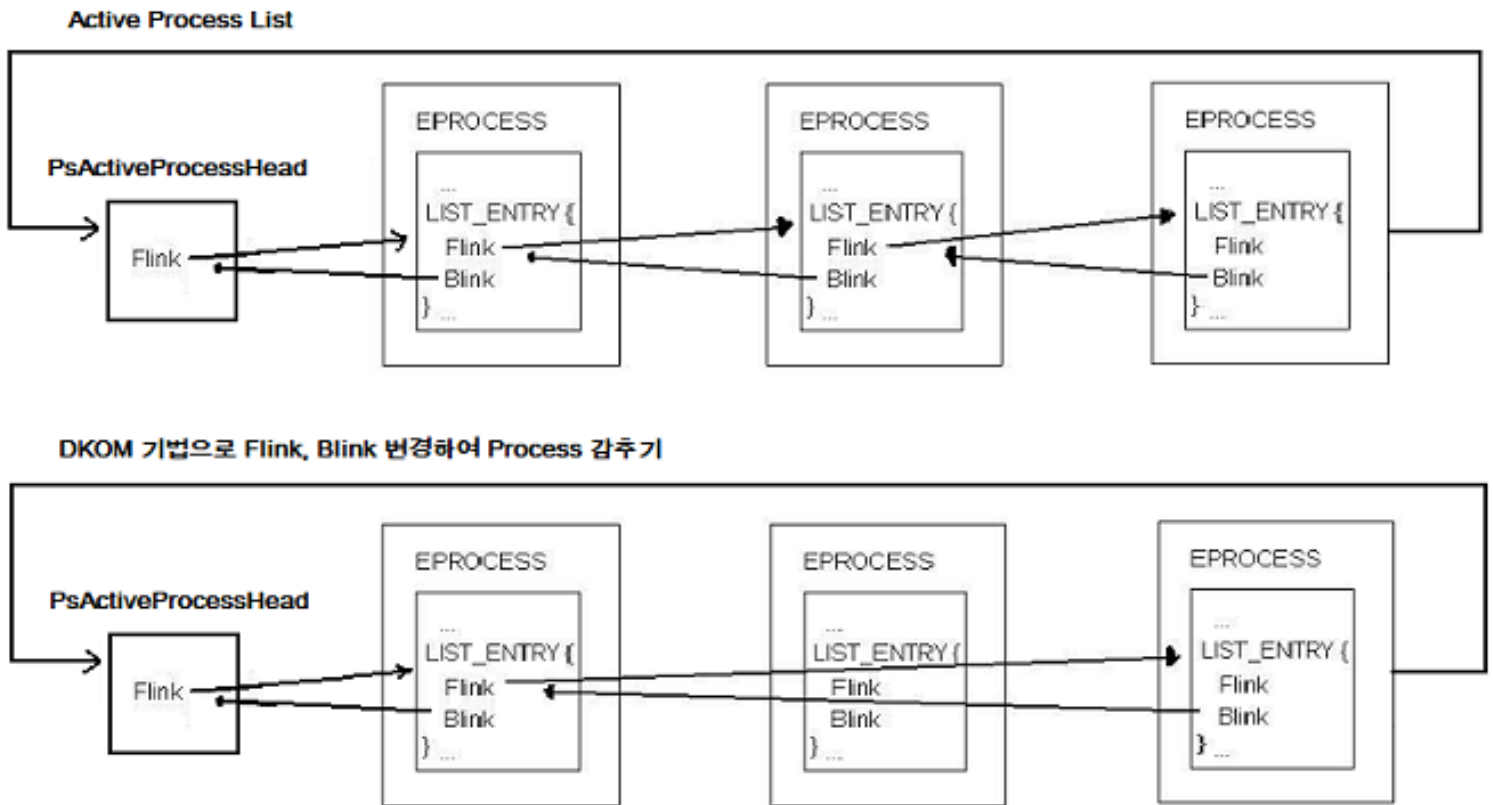
윈도우에는 커널 프로세스 컨트롤 블록(KPRCB , Kernel's Processor Control Block)이라고 하는 것이 있으며 그것은 커널 메모리 공간의 0xffdff120 번지에 존재한다. 위의 어셈블리 코드를 보면 fs 레지스터의 0x124 오프셋의 값을 가져오는데 이는 현재의 ETHREAD 구조체 주소이다. ETHREAD 블록을 통해 KTHREAD 구조체의 주소를 구하고 또 KTHREAD 구조체를 통해서 현재 프로세스의 EPROCESS 구조체 주소를 구한다. 그 다음 EPROCESS의 Doubly Linked list로 연결된 프로세스들을 모두 조사하여 원하는 프로세스를 찾아 숨기면 된다.



<그림 1. KPRCB로부터 프로세스 리스트까지의 링크>

요약해보면, EPROCESS의 BLINK와 FLINK를 포인터를 조작해서 Rootkit 코드의 프로세스를 Skip하도록 FLINK, BLINK 포인터를 조작하여 윈도우즈의 작업관리자(taskmgr.exe)에서 해당 Rootkit 프로세스가 보이지 않도록 하는 것이다.

보통 SSDT 후킹으로 프로세스를 숨기는 경우에는 SSDT 후킹 여부를 탐지하여 복구가 쉽게 가능하지만 DKOM은 탐지가 보다 어렵다.



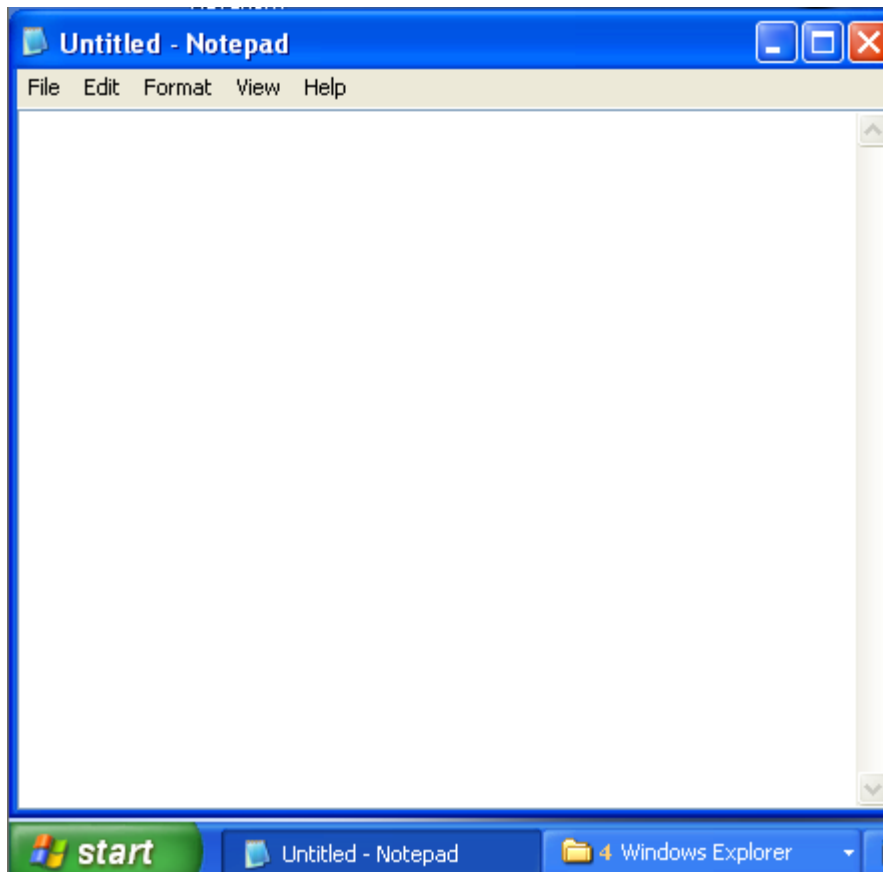
<그림 2. DKOM 을 이용한 Process 은닉 원리>

이처럼 FLINK, BLINK 값을 변경하여 프로세스 리스트에서 해당 EPROCESS 구조체를 제거함으로써 프로세스를 은닉하는 경우, 해당 Process가 실행되는데 문제가 없는가 하는 의문이 들 수 있다. 결론부터 말하자면, 아무 문제가 없다. 실행을 위한 스케줄링에서 제외되지 않는다. 오늘날의 OS는 Process (Thread의 Container 역할) 단위가 아닌 Thread 단위로 실행되며 우선순위 기반의 선점형 스케줄링을 수행한다. 즉, 프로세스는 여러 개의 Thread로 이루어지며 각 Thread는 ETHREAD 구조체로 표현된다.

## 나. 실습

- DKOM 기법으로 notepad.exe 를 숨겨보자!

1. notepad.exe 를 실행시킨다.



<그림 3 Windows에서 실행된 Notepad.exe의 모습>

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\xstone>cd \

C:\>tasklist /FI "ImageName eq notepad.exe"

Image Name                PID Session Name        Session#    Mem Usage
-----
notepad.exe                1192 Console              0           2,496 K

C:\>
```

<그림 4 tasklist 명령어를 이용하여 실행중인 Process List에서 notepad.exe 확인>



## 2. DKOM(ActiveProcessLinks Traverse) 으로 notepad.exe 를 숨긴다.

### 2-1. ActiveProcessLinks Traverse 를 이용하여 프로세스 목록 확인

#### ActiveProcessLink Traverse 자동화 Windbg Script :

```

type c:\ProcessList.txt

.block{
    r $t1 = PsActiveProcessHead // r : register. $ : 변수. @ : 주소
    r $t2 = @$t1

    .while(@$t1){
        r $t2 = poi(@$t2)
        da @$t2 - 0x88 + 0x174
        .if(@$t1 == @$t2){
            .break
        }
    }
}

```

#### 자동화 Windbg Script 실행 결과 :

```

820fa534 "VMwareUser.exe"
820fc7cc "wuauclt.exe"
8215c194 "notepad.exe"
821cc344 "wmiprvse.exe"
80560cc4 "..."

```

// 위의 결과는 ImageFileName (프로세스이름)이 있는 주소를 찾아낸 것이기 때문에, FLINK, BLINK 변경 시에는 - 0x174 (ImageFileName 의 offset) 를 통해 해당 Process 의 맨 앞을 가리키도록 해야 한다.

2-2. FLINK(forward link), BLINK(backwark link) 값 변경하여 숨긴다.

```

da 8215c194
    // 8215c194 : notepad.exe 의 ImageFileName 주소
ed 820fc7cc-174+88 poi(8215c194-174+88)
    // forward 링크 변경.
    -0x174 : Process 맨 앞을 가리킨다.
    +0x88 : flink offset
    ed : doubleword 만큼 해당주소에 해당값으로 채우기
ed 821cc344-174+88+4 poi(8215c194-174+88+4)
    // backward 링크 변경.
    -0x174 : Process 맨 앞을 가리킨다.
    +0x88+4 : blink offset
    
```

3. 실제로 notepad.exe 가 프로세스 목록에서 숨겨졌는지 확인해보자.

실행 결과 :

```

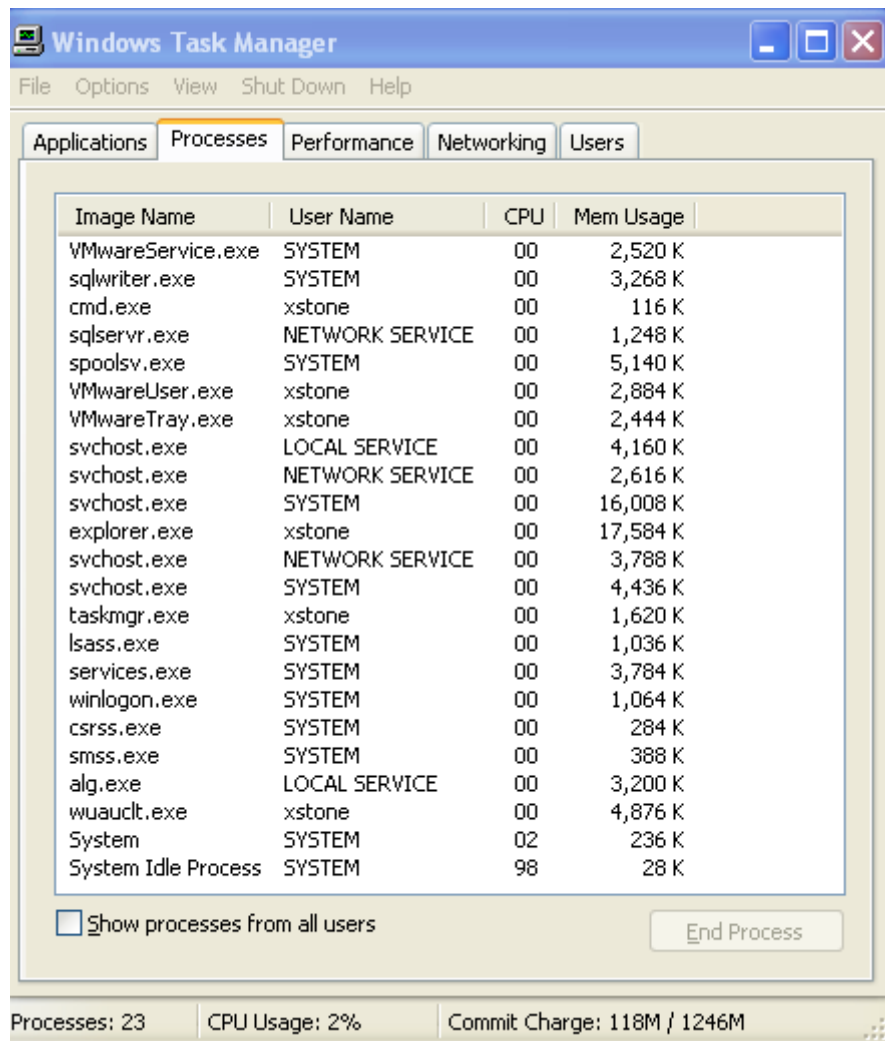
C:\Documents and Settings\xstone>tasklist /FI "ImageName eq notepad.exe"
INFO: No tasks running with the specified criteria.
    
```

```

C:\Documents and Settings\xstone>tasklist
Image Name
=====
System
smss.exe
csrss.exe
winlogon.exe
services.exe
lsass.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
svchost.exe
spoolsv.exe
sqlservr.exe
sqlwriter.exe
VMwareService.exe
alg.exe
explorer.exe
VMwareTray.exe
VMwareUser.exe
wuauclt.exe
cmd.exe
tasklist.exe
wmiprvse.exe
PID
=====
4
556
620
644
696
708
860
968
1092
1256
1288
1412
1560
1656
1700
368
1216
1308
1328
1948
748
1420
448
Session Name
=====
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Console
Session#
=====
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
Mem Usage
=====
212 K
372 K
3,396 K
708 K
3,708 K
1,428 K
4,288 K
3,724 K
16,080 K
2,572 K
4,092 K
5,072 K
1,472 K
3,220 K
2,372 K
3,156 K
15,036 K
2,404 K
2,860 K
4,804 K
2,288 K
3,984 K
5,228 K
    
```

<그림 5 DOS의 tasklist 명령어로 notepad.exe 가 보이지 않는다.>

**실행 결과 :**



<그림 6 Windows의 Task Manager 상에도 notepad.exe 가 보이지 않는다.>

## III. DKOM 탐지 기법

### 가. 다른 Linked list 이용하기

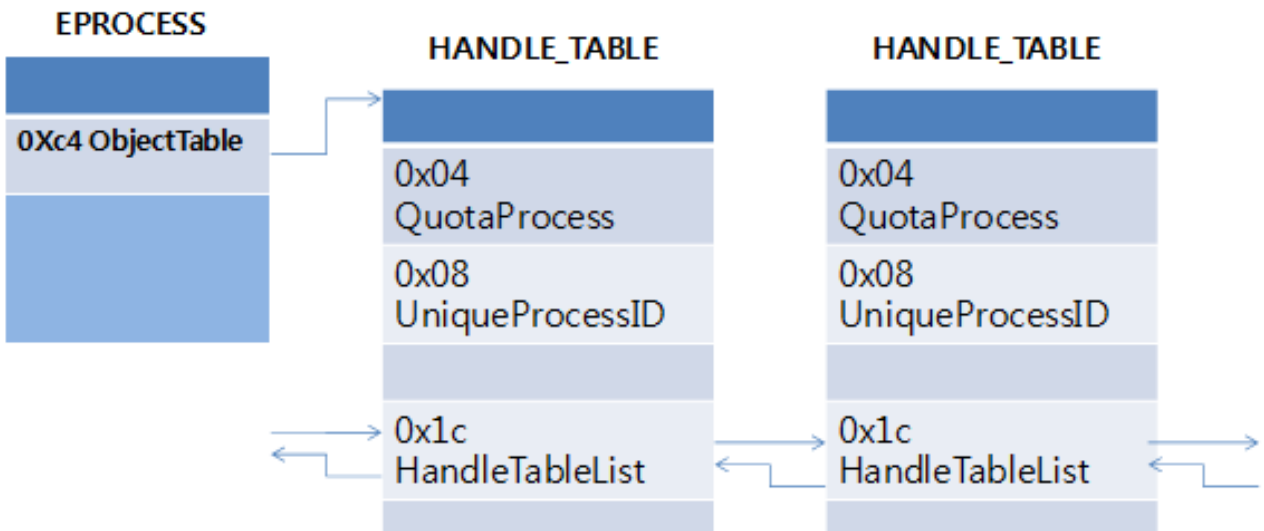
#### 1. 원리

DKOM(루트킷)이 조작하지 못한 Linked list 를 찾아 walking 하는 방법이다. 감추려했던 오브젝트를 탐지하기 위해서는 몇 개의 Linked list 를 탐지한 결과를 비교 분석하는 작업이 필요하다.

본문의 상기 기술( II. DKOM 을 이용한 은닉)에서는 ActiveProcess Links 의 List 를 변경한 것이므로, 이와는 다른 Linked list 구조인 **HandleTableList** 를 이용하여 DKOM 으로 숨겨진 프로세스를 찾아낼 수 있다.

EPROCESS 의 0x0c4 위치에 Object Table 이 HANDLE TABLE 구조체의 포인터로 프로세스의 핸들정보를 담고 있다. HANDLE TABLE 의 QuotaProcess 를 확인하여 해당 프로세스를 확인할 수 있다. 따라서 이렇게 QuotaProcess 를 따라 확인해 보면 지금 실행중인 프로세스를 확인하여 감춰진 프로세스를 찾아낼 수 있다.

HandleTableList 를 사용할 경우 ActiveProcessLinks 가 손상된 프로세스도 탐지할 수 있다.



<그림 7 HandleTableList의 구조>

// HandleTableList 를 이용한 프로세스 목록 조사

System Idle Process 와 System 프로세스는 EPROCESS 구조체의 주소가 없음

## 2. 실습

### 2-1. 직접 Windbg Script 를 작성하여 은닉된 프로세스 찾기

#### Process Handle Linked List 이용해 Process List 얻는 Windbg Script :

```
.block
{
    r $t1 = poi(poi(poi(ffdff12c)+0x44)+0xc4)

    .while(true){
        r $t1=poi(@$t1 + 0x1c) - 0x1c
        .if(poi(@$t1+0x4) == 0){.break}
        da poi(@$t1+0x4) + 0x174
    }
}
```

// 위의 script 를 c:\WProcessList.txt 에 저장한다면,  
Windbg 에서 실행시 " kd> \$\$>a<c:\WProcessList.txt " 라고 입력하면 된다.

이렇게 다른 linked list 를 이용하여 숨겨진 프로세스를 찾아내는 것이고 아래의 툴들도 linked list 를 이용한다. 툴이 사용하는 링크를 우회한다면 툴도 찾아내지 못할 수도 있다.

## 2-2. 툴을 사용하여 은닉된 프로세스 찾기

### 2-2-1. Volatility의 pslist

ActiveProcessLinks 를 따라 Linked list 를 이어가면서 실행중인 프로세스 목록을 조사하므로 위와 같이 ActiveProcessLinks 를 조작한 경우 찾아내지 못한다. 그러나 Hooking 에 의해 감춰진 프로세스는 출력 가능하다.

**실행 방법 :**

```
python volatility pslist -f c:\wkom.dd
// c:\wkom.dd : Memory dump 한 파일
-f : 현재 실행중인 Process 목록을 출력한다.
```

**실행 결과 :**

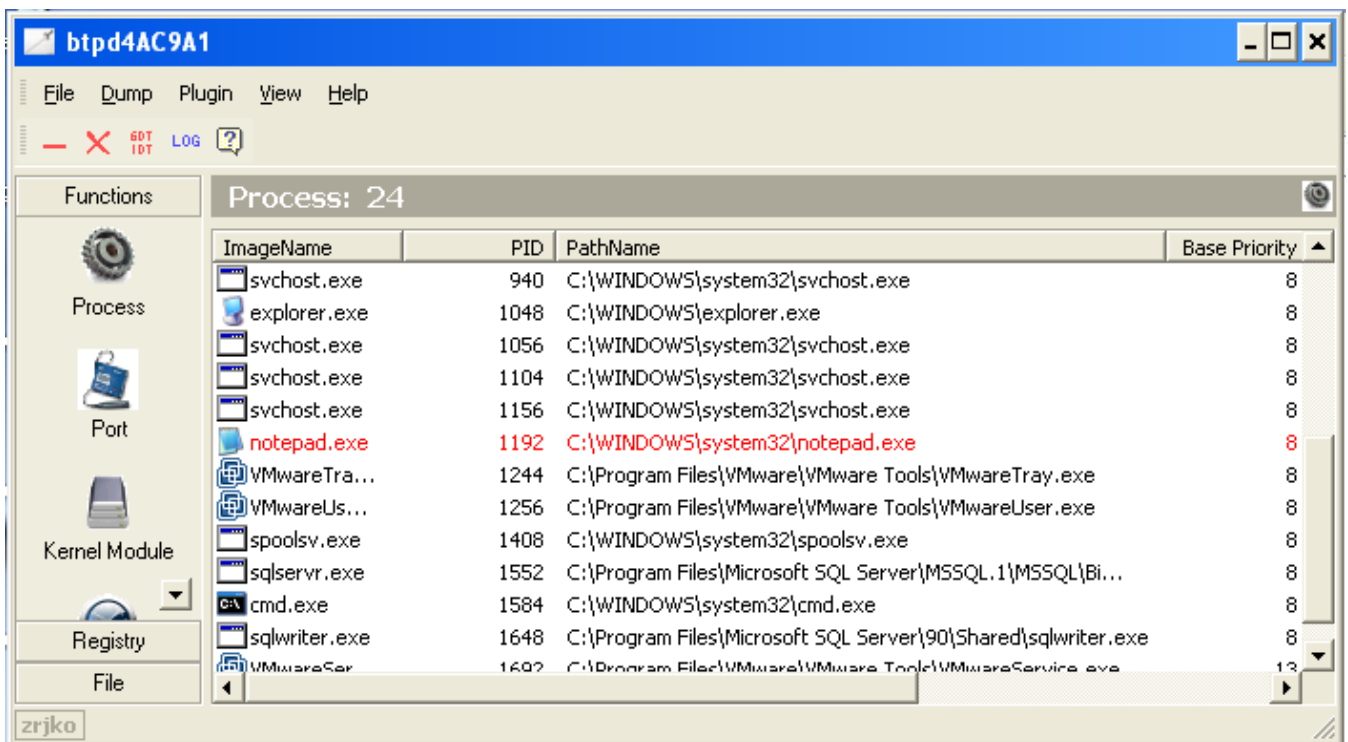
```
C:\Python25\Tools>cd Volatility-1.3_Beta
C:\Python25\Tools\Volatility-1.3_Beta>python volatility pslist -f c:\dkom.dd
*** Unable to load module cachedump: No module named Crypto.Hash
*** Unable to load module hashdump: No module named Crypto.Hash
*** Unable to load module lsadump: No module named Crypto.Hash
*** Unable to load module cachedump: No module named Crypto.Hash
*** Unable to load module hashdump: No module named Crypto.Hash
*** Unable to load module lsadump: No module named Crypto.Hash
Name      Pid      PPid     Thds     Hnds     Time
System    4         0        57       248      Thu Jan 01 00:00:00 1970
smss.exe  520       4         3         21       Wed Jul 22 05:00:28 2009
csrss.exe 628       520      11        357      Wed Jul 22 05:00:28 2009
winlogon.exe 652      520      17        496      Wed Jul 22 05:00:29 2009
services.exe 696      652      16        267      Wed Jul 22 05:00:29 2009
lsass.exe 708       652      18        409      Wed Jul 22 05:00:29 2009
svchost.exe 860       696      18        191      Wed Jul 22 05:00:29 2009
svchost.exe 928       696       9        239      Wed Jul 22 05:00:30 2009
svchost.exe 1044      696      56       1116     Wed Jul 22 05:00:30 2009
svchost.exe 1136      696       4         55       Wed Jul 22 05:00:32 2009
svchost.exe 1192      696      13        201      Wed Jul 22 05:00:32 2009
spoolsv.exe 1400      696      11        132      Wed Jul 22 05:00:33 2009
sqlservr.exe 1548     696      20        296      Wed Jul 22 05:00:39 2009
sqlwriter.exe 1636     696       3         84       Wed Jul 22 05:00:42 2009
VMwareService.e 1676     696       3         58       Wed Jul 22 05:00:42 2009
alg.exe   288       696       5        101      Wed Jul 22 05:00:50 2009
explorer.exe 820      1260     14        497      Wed Jul 22 05:04:22 2009
VMwareTray.exe 1008     820       1         24       Wed Jul 22 05:04:23 2009
VMwareUser.exe 1432     820       3         65       Wed Jul 22 05:04:23 2009
wuauc.lt.exe 1672     1044     3         132      Wed Jul 22 05:04:36 2009
cmd.exe   1244     820       1         31       Thu Jul 23 01:51:00 2009
cmd.exe   964      820       1         28       Thu Jul 23 02:10:48 2009
mdd.exe  1660     1244     1         23       Thu Jul 23 02:18:01 2009
C:\Python25\Tools\Volatility-1.3_Beta>_
```

<그림 8 Volatility pslist 로는 notepad.exe가 보이지 않는다.>

### 2-2-2. IceSword

ActiveProcessLinks 이외에도 HandleTableList 등 다른 Linked list 를 조사하므로 notepad.exe 가 발견된다. IceSword 에서 탐지가능한 모든 Linked list 를 조작한다면 이 툴에서도 DKOM 탐지는 불가능하다.

**실행 결과 :**



<그림 9 IceSword로 Process를 탐지한 결과 은닉된 notepad.exe가 빨간색으로 나타났다.>



## 나. Process Carving

### 1. 원리

Memory Dump 를 한 후 Signature 를 찾아내어 EPROCESS 를 모두 추출하는 방법이다.

Linked List 를 이용하지 않고, 메모리 덤프에서 데이터를 읽어들이어 EPROCESS 구조체인지 검증한다. EPROCESS 구조체에 특별한 Signature 가 없으므로 Pool Tag, Object Header 값 등을 확인한다.

실행중인 프로세스, 감춰진 프로세스, 이미 종료된 프로세스에 대한 정보를 모두 수집 가능하다.

### 2. 실습을 위해 필요한 사전지식

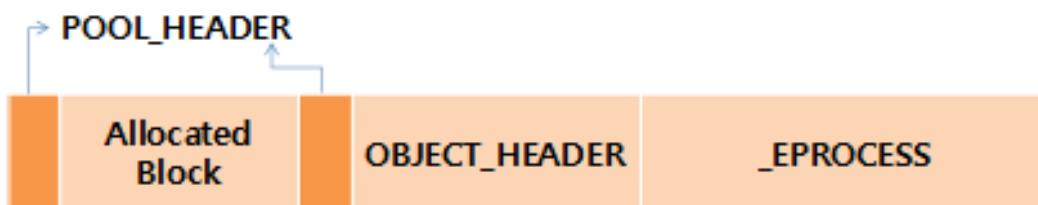
#### 2-1. Memory Pool

Hardware Level 에서 메모리 할당의 최소단위는 : Page(4k)

커널이 사용하는 메모리 영역의 일부를 예약을 해두고, 해당 영역에서 메모리를 조금씩 할당 받아서 사용하는 방식이다.

하나의 커널 오브젝트를 위해 전체 페이지를 할당하는 것은 메모리 낭비이다. 윈도우의 경우 메모리 풀(memory pool)에서 8 바이트 단위로 메모리 할당이 가능하며, ETHREAD, EPROCESS 와 같은 커널 오브젝트는 메모리 풀에 존재한다.

#### 2-2. Pool\_Header



<그림 10 Memory Pool에 할당된 Memory Block>

메모리 풀에 할당된 메모리 블록은 관리를 위한 자료구조 (POOL\_HEADER)를 포함한다.

```
kd> dt _POOL_HEADER
nt!_POOL_HEADER
+0x000 PreviousSize      : Pos 0, 9 Bits
+0x000 PoolIndex        : Pos 9, 7 Bits
+0x002 BlockSize       : Pos 0, 9 Bits
+0x002 PoolType        : Pos 9, 7 Bits
+0x000 Ulong1          : Uint4B
+0x004 ProcessBilled   : Ptr32 _EPROCESS
+0x004 PoolTag         : Uint4B
+0x004 AllocatorBackTraceIndex : Uint2B
+0x006 PoolTagHash     : Uint2B
```

<그림 11 Pool\_Header의 구조 확인>

### 2-3. Paged Pool 과 Non Paged Pool

Paged pool - 메모리 부족시 Page in, Page out 을 통해 swap 하여 메모리를 확보한다.

Non paged Pool - 절대로 Page out 이 안 된다.

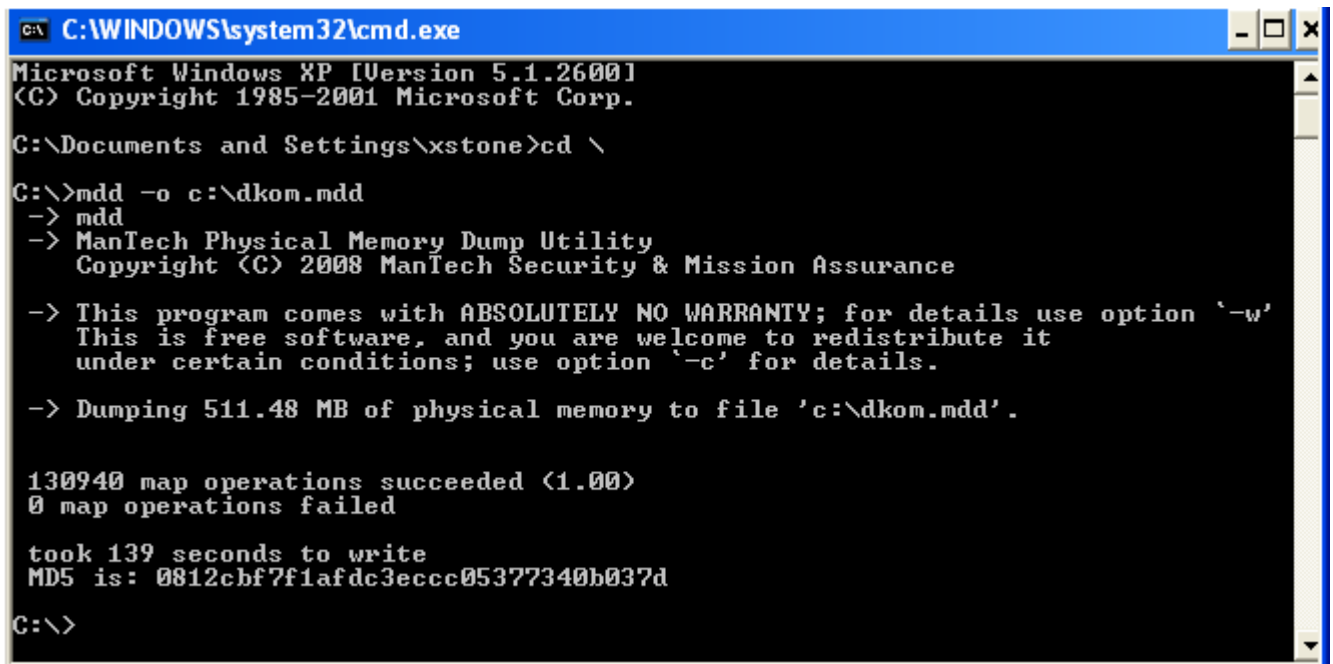
커널은 중요한 데이터는 대부분 Non paged pool 에서 할당 받는다.

### 3. 실습

3-0. 먼저 mdd 를 통해서 메모리를 덤프한다.

Notepad 는 DKOM 으로 감춘 상태이고 calc.exe 는 실행시키고 종료한 상태이다.

c.f) mdd : Mantech Memory DD ([www.mantech.com/msma/mdd.asp](http://www.mantech.com/msma/mdd.asp))



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\xstone>cd \

C:\>mdd -o c:\dkom.mdd
-> mdd
-> ManTech Physical Memory Dump Utility
    Copyright (C) 2008 ManTech Security & Mission Assurance

-> This program comes with ABSOLUTELY NO WARRANTY; for details use option '-w'
    This is free software, and you are welcome to redistribute it
    under certain conditions; use option '-c' for details.

-> Dumping 511.48 MB of physical memory to file 'c:\dkom.mdd'.

130940 map operations succeeded (1.00)
0 map operations failed

took 139 seconds to write
MD5 is: 0812cbf7f1afdc3eccc05377340b037d

C:\>
```

<그림 12 현재 Physical Memory를 c:\dkom.dd 파일로 dump함>

// 원래는 mdd.exe 와 만들어질 dkom.dd 모두 usb 에 넣고 run 해야 해당 시스템을 완전히 보존하여 이미지를 뜯 수 있다.

### 3-1. 직접 Windbg 를 이용하기

#### 3-1-1. PoolTag 값 검증

ExAllocatePoolWithTag 와 같은 함수를 이용하여 메모리를 할당받는 경우 PoolTag 를 파라미터로 전달한다.

PoolTag : Memory Pool 을 사용하는 주체를 식별하려는 목적  
Object 의 성격에 대한 정보를 메모리를 요청할 때  
4byte 의 ASCII 문자열을 붙여서 보낸다.

i. kd> !process 0 0

**실행 결과 :**

```
PROCESS 820f5020 SessionId: none Cid: 0208 Peb: 7ffde000 ParentCid: 0004
DirBase: 07a80020 ObjectTable: e1022188 HandleCount: 21.
Image: smss.exe
```

... (생략)

```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 823c8830 SessionId: none Cid: 0004 Peb: 00000000 ParentCid: 0000
DirBase: 00319000 ObjectTable: e1000cc0 HandleCount: 247.
Image: System
```

<그림 13 현재 모든 프로세스 정보를 보여준다. >

#### **c.f) !process**

The **!process** extension displays information about the specified process, or about all processes, including the EPROCESS block.

This extension can be used only during kernel-mode debugging.

ii. kd> dt\_EPROCESS 820f5020 (EPROCESS 의 시작점)

**실행 결과 :**

```
kd> dt _EPROCESS 820F5020
ntdll!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ProcessLock : _EX_PUSH_LOCK
+0x070 CreateTime : _LARGE_INTEGER 0x1ca0a89`501d4bd8
+0x078 ExitTime : _LARGE_INTEGER 0x0
+0x080 RundownProtect : _EX_RUNDOWN_REF
+0x084 UniqueProcessId : 0x00000208
+0x088 ActiveProcessLinks : _LIST_ENTRY [ 0x821350a8 - 0x823c88b8 ]
+0x090 QuotaUsage : [3] 0x280
+0x09c QuotaPeak : [3] 0x318
+0x0a8 CommitCharge : 0x2a
+0x0ac PeakVirtualSize : 0xbb5000
+0x0b0 VirtualSize : 0x3b6000
+0x0b4 SessionProcessLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
+0x0bc DebugPort : (null)
+0x0c0 ExceptionPort : (null)
+0x0c4 ObjectTable : 0xe1022188 _HANDLE_TABLE
+0x0c8 Token : _EX_FAST_REF
+0x0cc WorkingSetLock : _FAST_MUTEX
+0x0ec WorkingSetPage : 0x779f
+0x174 ImageFileName : [16] "smss.exe"
```

<그림 14 EPROCESS의 시작점 smss.exe 를 EPROCESS의 구조체로 정보 확인>

iii. EPROCESS 앞에 Object Header(size : 0x18)가 있다. 확인해보자.

**실행 결과 :**

```
kd> dt _OBJECT_HEADER 820F5020-18
nt!_OBJECT_HEADER
+0x000 PointerCount : 6
+0x004 HandleCount : 0
+0x004 NextToFree : (null)
+0x008 Type : 0x823c8e70 _OBJECT_TYPE
+0x00c NameInfoOffset : 0 ''
+0x00d HandleInfoOffset : 0 ''
+0x00e QuotaInfoOffset : 0 ''
+0x00f Flags : 0x22 ''''
+0x010 ObjectCreateInfo : 0x00000001 _OBJECT_CREATE_INFORMATION
+0x010 QuotaBlockCharged : 0x00000001
+0x014 SecurityDescriptor : 0xe10003de
+0x018 Body : _QUAD
```

<그림 15 첫 번째 EPROCESS인 smss.exe의 Object Header 정보 확인>

iv. Object Header 앞에는 PoolHeader(size : 0x8)가 있다. 확인해보자.

**실행 결과 :**

```
kd> dt _POOL_HEADER 820F5020-18-8
nt!_POOL_HEADER
+0x000 PreviousSize      : 0y0000000000 (0)
+0x000 PoolIndex        : 0y00000000 (0)
+0x002 BlockSize       : 0y0010100000 (0x50)
+0x002 PoolType        : 0y00000101 (0x5)
+0x000 Ulong1         : 0xa5000000
+0x004 ProcessBilled   : 0xe36f7250 _EPROCESS
+0x004 PoolTag         : 0xe36f7250
+0x004 AllocatorBackTraceIndex : 0x7250
+0x006 PoolTagHash     : 0xe36f
```

<그림 16 첫 번째 EPROCESS인 sms.exe의 Pool Header 정보 확인>

// PoolTag 값 0xe36f7250 은 실제로는 0x636f7250 이다.

최상위 비트를 항상 1로 설정하기 때문이다.

v. POOL\_HEADER 의 PoolTag 값이 e36f7250 인 것을 찾는다.

**실행 방법 :**

```
!poolfind 0xe36f7250 0
// 0xe36f7250 PoolTag 값을 갖는 Pool 중
0(Non paged pool)을 찾으라는 뜻이다.
```

**실행 결과 :**

```
kd> !poolfind 0xe36f7250
unable to get PoolTrackTable - pool tagging is likely disabled or you have the wrong symbols
unable to get large pool allocation table - either wrong symbols or pool tagging is disabled

Searching NonPaged pool (811ed000 : 82400000) for Tag: Proã

...terminating - searched pool to 812fb000
kd> !poolfind 0xe36f7250 0
unable to get PoolTrackTable - pool tagging is likely disabled or you have the wrong symbols
unable to get large pool allocation table - either wrong symbols or pool tagging is disabled

Searching NonPaged pool (811ed000 : 82400000) for Tag: Proã

81f69000 size: 280 previous size: 0 (Allocated) Proc (Protected)
81f86168 size: 280 previous size: 40 (Allocated) Proc (Protected)
```

<그림 17 0xe36f7250 PoolTag를 갖는 해당 Non paged pool 정보 확인>

vi. 위의 결과 여러 주소가 나왔다. 이 주소는 PoolTag 가 proc 값을 가지는 Pool Header 의 주소이므로 여기에 적절하게 offset 값을 더하여 EPROCESS 를 확인하면 다음과 같다.

**실행 결과 :**

```
kd> dt _EPROCESS 81f69000+20
ntdll!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ProcessLock : _EX_PUSH_LOCK
+0x070 CreateTime : _LARGE_INTEGER 0x1ca0a89`5d46269a
+0x078 ExitTime : _LARGE_INTEGER 0x0
+0x080 RundownProtect : _EX_RUNDOWN_REF
+0x168 Filler : 0
+0x170 Session : 0xf8bd0000
+0x174 ImageFileName : [16] "sqlwriter.exe"
+0x184 JobLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
```

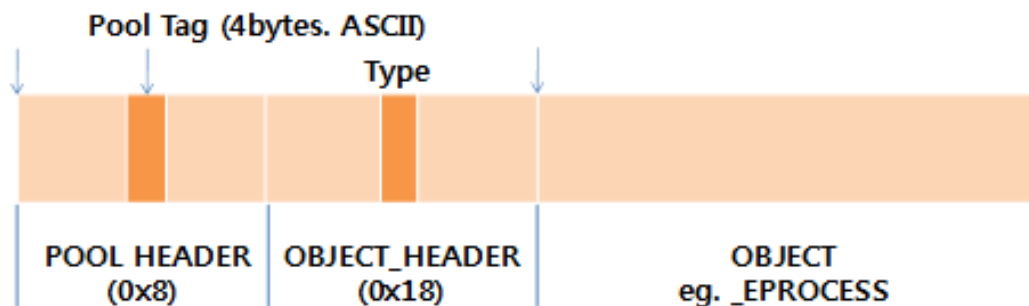
**<그림 18 81f69000 에 +20 하여 EPROCESS 확인>**

// +20 (0x18 : Object Header Size + 0x8 : Pool Header Size)

### 3-1-2. OBJECT\_HEADER 검증

윈도우 커널은 Process, Thread, File, Event 등 커널 내부에서 사용하는 자료 구조들의 접근 방법을 단일화하고 보안성을 높이기 위해 오브젝트 모델을 사용한다.

OBJECT\_HEADER 내의 Type 필드는 대상 오브젝트(eg. \_EPROCESS)의 일반적인 속성에 대한 정보를 담고 있는 OBJECT\_TYPE 구조체를 가리키는 포인터이다. 프로세스의 경우 이 값이 PsProcessType 과 같거나 0xbad0404 값을 가진다.



<그림 19 Object Header 를 포함한 Memory 구조>



i. kd> dt \_OBJECT\_HEADER 820F5000+8  
 (!object 820f5000 도 같은 결과를 보여준다.)

**실행 결과 :**

```
81f69000 size: 280 previous size: 0 (Allocated) Proc (Protected)
81f86168 size: 280 previous size: 40 (Allocated) Proc (Protected)
81f8ece8 size: 280 previous size: 268 (Allocated) Proc (Protected)
820f5000 size: 280 previous size: 0 (Allocated) Proc (Protected)
```

```
...terminating - searched pool to 82101000
kd> dt _OBJECT_HEADER 820f5000+8
nt!_OBJECT_HEADER
+0x000 PointerCount : 6
+0x004 HandleCount : 0
+0x004 NextToFree : (null)
+0x008 Type : 0x823c8e70 _OBJECT_TYPE
+0x00c NameInfoOffset : 0 ''
+0x00d HandleInfoOffset : 0 ''
+0x00e QuotaInfoOffset : 0 ''
+0x00f Flags : 0x22 ''
+0x010 ObjectCreateInfo : 0x00000001 _OBJECT_CREATE_INFORMATION
+0x010 QuotaBlockCharged : 0x00000001
+0x014 SecurityDescriptor : 0xe10003dd
+0x018 Body : _QUAD
```

**<그림 20 EPROCESS 중 하나를 선택하여 Object Header 검증>**

// !process 0 0 의 결과값 중 하나를 선택하여 OBJECT\_HEADER 를 검증해 본다. PoolHeader Size 인 0x8 을 더하여 오브젝트 헤더의 정보 중 Type 을 확인한다. Type 정보의 값은 0x823c8e70 이다. 이것이 PsProcessType 값과 같을까? 확인해보자.

ii. kd> dd PsProcessType L1

**실행 결과 :**

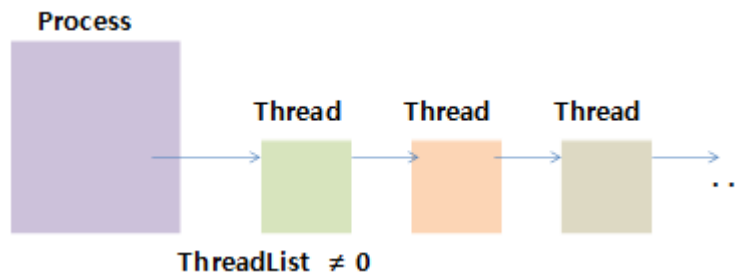
```
|kd> dd PsProcessType L1
|80559358 823c8e70
```

**<그림 21 PsProcessType 확인>**

// L1 : Doubleword 하나를 의미한다.  
 실제로 PsProcessType 값을 확인해보면 823c8e70 이라는 Object\_Type 과 같은 값을 갖는다.

### 3-1-3. \_EPROCESS 검증

#### i. ThreadListHead 의 값을 검증



<그림 22 Process 와 Thread>

프로세스는 반드시 하나 이상의 Thread 를 가진다.

그렇기 때문에 ThreadList 의 시작 값을 가지는 ThreadList Head 가 0 이 아니어야 한다. Thread Control Block 은 커널 메모리 영역에 존재한다.

커널의 가상 주소 공간은 일반적으로 0x00000000~0x7fffffff 안에 위치하며, /3GB 옵션은 0x00000000~0xbfffffff 의 주소를 갖는다.

#### 실행 결과 :

```
kd> dt _EPROCESS 820f5000+8+18 ThreadListHead
nt!_EPROCESS
+0x190 ThreadListHead : _LIST_ENTRY [ 0x8214b24c - 0x8210524c ]
```

<그림 23 ThreadListHead 의 값이 0 이 아님을 확인 >

## ii. DISPATCHER\_HEADER 검증

EPROCESS 는 Dispatcher Object 이다. 구조체의 맨 앞부분에 DISPATCHER\_HEADER 가 존재한다.

### 실행 결과 :

```
kd> dt _KPROCESS
nt!_KPROCESS
+0x000 Header : _DISPATCHER_HEADER

kd> dt _DISPATCHER_HEADER
nt!_DISPATCHER_HEADER
+0x000 Type : UChar
+0x001 Absolute : UChar
+0x002 Size : UChar
+0x003 Inserted : UChar
+0x004 SignalState : Int4B
+0x008 WaitListHead : _LIST_ENTRY

kd> dt _EPROCESS 820f5000+8+18
nt!_EPROCESS
+0x000 Pcb | : _KPROCESS
```

<그림 24 DISPATCHER\_HEADER 의 구조>

관찰을 통해 확인해보면 process 의 경우 DISPATCHER HEADER 의 Type 필드와 Size 필드의 값이 각각 0x3 과 0x1b 로 고정되어 있다.

### 실행 결과 :

```
kd> dt _DISPATCHER_HEADER 820f5000+8+18
nt!_DISPATCHER_HEADER
+0x000 Type : 0x3 ''
+0x001 Absolute : 0 ''
+0x002 Size : 0x1b ''
+0x003 Inserted : 0 ''
+0x004 SignalState : 0
+0x008 WaitListHead : _LIST_ENTRY [ 0x820f5028 - 0x820f5028 ]
```

<그림 25 Type=3, Size=1b 고정>

### iii. DirectoryTableBase 값 검증

가상 주소를 물리 주소로 변환하는데 필요한 Page Directory의 시작 주소를 저장한다.

정상적인 EPROCESS 객체의 경우 이 값이 0 이 아니어야 한다. Page Directory 는 페이지 하나를 통째로 사용하여 시작주소는 페이지 단위로 alignment 되므로 4k 의 배수가 되어야 한다.

#### 실행 결과 :

```
kd> dt _EPROCESS 820f5000+8+18 VadRoot
nt!_EPROCESS
+0x11c VadRoot : 0x8227a0f0
```

#### <그림 26 DirectoryTableBase 확인>

// 우리가 DirectoryTableBase 로 부르지만 windbg 에서는 VadRoot 로 불리우며, 0x8227a0f0 을 십진수 4k 로 나누면 532725 로 딱 떨어진다.

### 3-2. 툴을 사용하기

#### 3-2-1. Volatility의 psscan

Process Carving 의 방법을 사용하므로 DKOM 탐지가 가능하다. 루트킷으로 감춘 프로세스 뿐만 아니라 종료한 프로세스까지도 탐지한다.

#### 실행 방법 :

```
python volatility psscan -f c:\wdkom.dd
// c:\wdkom.dd : Memory dump 한 파일
-f : 현재 실행중인 Process 목록을 출력한다.
```

#### 실행 결과 :

```
C:\Python25\Tools\Volatility-1.3_Beta>python volatility psscan -f c:\dkom.dd
```

1	0	0					0x00551b80
0x00319000 Idle							
2	964	820	Thu	Jul 23	02:10:48	2009	0x02070da0
0x07a80300 cmd.exe							
3	1244	820	Thu	Jul 23	01:51:00	2009	0x0213e020

// 실행중인 프로세스가 나오기 시작한다.

49	1880	820	Wed	Jul 22	05:04:42	2009	0x15285c18
0x07a802c0 notepad.exe							
50	520	4	Wed	Jul 22	05:00:20	2009	0x155ec020
0x07a80020 smss.exe							
51	628	520	Wed	Jul 22	05:00:28	2009	0x155f9020
0x07a80040 csrss.exe							

// 감추어진 notepad 가 나온다.

66	1172	820	Thu	Jul 23	02:15:25	2009	Thu Jul 23 02:17:34 2009	0x197b6be8
0x07a80240 calc.exe								
67	1244	820	Thu	Jul 23	01:51:00	2009		0x1a1cd020
0x07a80220 cmd.exe								

// 종료했던 계산기(calc.exe)가 나타났으며, 종료된 시간까지 나왔다.

<그림 27 Volatility psscan 은 숨겨진 프로세스 및 종료된 프로세스도 출력>

#### 3-2-2. 그 밖의 유명한 툴 : PTFinder(Process&Thread Finder)

## IV. 참고문헌

Greg Hoglund · Jamie Butler, "루트킷 : 윈도우 커널 조작의 미학" ,  
에이콘, July 2008