

# #44u6115f

## SDT Hooking 무력화에 대한 연구

By Dual5651  
(<http://dualpage.muz.ro>)

요약 : 이 문서는 Windows 2000/XP/2003 환경에서의 SSDT Hooking을 무력화 시키는 방법에 대한 필자의 연구내용을 다루고 있습니다. 이 문서는 독자가 SSDT Hooking에 대하여 알고 있다는 전제하에 쓰여졌습니다.

# 1. 소개 - SSDT Hooking

SSDT란 System Service Dispatch Table의 약자로써 Window의 API들의 실제 함수의 주소들이 저장되어 있는 Table입니다. Kernel단에서의 처리가 필요한 API들은 해당 API의 Service Index를 이용하여 SSDT에서 해당 함수의 실제 주소를 얻어 호출합니다. SSDT는 Kernel단의 Memory에 존재 하는데, SSDT는 Process 독립적인 부분이 아닙니다. 즉 어떠한 조작을 하지 않는 한 모든 Application들은 모두 같은 Table을 참조하고 있는 것입니다. 그럼으로 SSDT를 수정하여 주면 시스템 전역 적인 API Hooking을 할 수 있습니다. 이러한 이유로 수많은 Rootkit들이 SSDT Hooking을 사용 하고 있으며, Virus Engine의 동작기반 Virus탐지 기술에도 사용되고 있습니다.

# 2. Windows Internals

Ring3에서 Ring0로 Service 호출을 할때, Windows 2000까지는 int 0x2e를 사용 하였으며, Windows XP부터는 sysenter를 사용하여 Ring0로 전환하고 있습니다. Kernel단의 처리가 필요한 API가 실제로 어떠한 호출과정을 거치는지 CreateFile()의 호출 과정을 통해 보도록 하겠습니다.

----- Ring 3 -----

Application에서 CreateFile 호출

**CreateFile(szFileName,,,,,,);**

Kernel32.dll 의 CreateFileA() 호출됨.

```

7C801A24 8BFF          MOV EDI,EDI
7C801A26 55           PUSH EBP
7C801A27 8BEC          MOV EBP,ESP
7C801A29 FF75 08      PUSH DWORD PTR SS:[EBP+8]
7C801A2C E8 43C60000 CALL kernel32.7C80E074
7C801A31 85C0          TEST EAX,EAX
7C801A33 74 1E        JE SHORT kernel32.7C801A53
7C801A35 FF75 20      PUSH DWORD PTR SS:[EBP+20]
7C801A38 FF75 1C      PUSH DWORD PTR SS:[EBP+1C]
7C801A3B FF75 18      PUSH DWORD PTR SS:[EBP+18]
7C801A3E FF75 14      PUSH DWORD PTR SS:[EBP+14]
7C801A41 FF75 10      PUSH DWORD PTR SS:[EBP+10]
7C801A44 FF75 0C      PUSH DWORD PTR SS:[EBP+C]
7C801A47 FF70 04      PUSH DWORD PTR DS:[EAX+4]
7C801A4A E8 11ED0000 CALL kernel32.CreateFileW
7C801A4F 5D           POP EBP
7C801A50 C2 1C00      RETN 1C
7C801A53 93C8 FF      OR EAX,FFFFFFFF
7C801A56 EB F7        JMP SHORT kernel32.7C801A4F
7C801A59 90           NOP
  
```

Kernel32.dll의 CreateFileW() 호출됨.

```

7C801000 85D0          TEST EDX,EDX
7C801002 74 C3        JNZ kernel32.7C8030700
7C801004 85C8          TEST EAX,EAX
7C801006 74 00        JNZ kernel32.7C80306E3
7C801008 93D0 20 00   CMP DWORD PTR SS:[EBP+20],0
7C80100A 93D0 21 FE 01 00 CMP kernel32.7C8030707
7C80100C 93D0 40      OR EAX,40
7C80100E 91 E3        AND EBX,PTR SS:[EBP+14]
7C801010 8B 77        MOV ESI,PTR [EAX]
7C801012 55           PUSH EBP
7C801014 57           PUSH EDI
7C801016 56           PUSH ESI
7C801018 54           PUSH EAX
7C80101A 53           PUSH EBX
7C80101C 52           PUSH ECX
7C80101E 51           PUSH EDX
7C801020 50           PUSH EAX
7C801022 50           PUSH ECX
7C801024 50           PUSH EDX
7C801026 50           PUSH EAX
7C801028 50           PUSH ECX
7C80102A 50           PUSH EDX
7C80102C 50           PUSH EAX
7C80102E 50           PUSH ECX
7C801030 50           PUSH EDX
7C801032 50           PUSH EAX
7C801034 50           PUSH ECX
7C801036 50           PUSH EDX
7C801038 50           PUSH EAX
7C80103A 50           PUSH ECX
7C80103C 50           PUSH EDX
7C80103E 50           PUSH EAX
7C801040 50           PUSH ECX
7C801042 50           PUSH EDX
7C801044 50           PUSH EAX
7C801046 50           PUSH ECX
7C801048 50           PUSH EDX
7C80104A 50           PUSH EAX
7C80104C 50           PUSH ECX
7C80104E 50           PUSH EDX
7C801050 50           PUSH EAX
7C801052 50           PUSH ECX
7C801054 50           PUSH EDX
7C801056 50           PUSH EAX
7C801058 50           PUSH ECX
7C80105A 50           PUSH EDX
7C80105C 50           PUSH EAX
7C80105E 50           PUSH ECX
7C801060 50           PUSH EDX
7C801062 50           PUSH EAX
7C801064 50           PUSH ECX
7C801066 50           PUSH EDX
7C801068 50           PUSH EAX
7C80106A 50           PUSH ECX
7C80106C 50           PUSH EDX
7C80106E 50           PUSH EAX
7C801070 50           PUSH ECX
7C801072 50           PUSH EDX
7C801074 50           PUSH EAX
7C801076 50           PUSH ECX
7C801078 50           PUSH EDX
7C80107A 50           PUSH EAX
7C80107C 50           PUSH ECX
7C80107E 50           PUSH EDX
7C801080 50           PUSH EAX
7C801082 50           PUSH ECX
7C801084 50           PUSH EDX
7C801086 50           PUSH EAX
7C801088 50           PUSH ECX
7C80108A 50           PUSH EDX
7C80108C 50           PUSH EAX
7C80108E 50           PUSH ECX
7C801090 50           PUSH EDX
7C801092 50           PUSH EAX
7C801094 50           PUSH ECX
7C801096 50           PUSH EDX
7C801098 50           PUSH EAX
7C80109A 50           PUSH ECX
7C80109C 50           PUSH EDX
7C80109E 50           PUSH EAX
7C8010A0 50           PUSH ECX
7C8010A2 50           PUSH EDX
7C8010A4 50           PUSH EAX
7C8010A6 50           PUSH ECX
7C8010A8 50           PUSH EDX
7C8010AA 50           PUSH EAX
7C8010AC 50           PUSH ECX
7C8010AE 50           PUSH EDX
7C8010B0 50           PUSH EAX
7C8010B2 50           PUSH ECX
7C8010B4 50           PUSH EDX
7C8010B6 50           PUSH EAX
7C8010B8 50           PUSH ECX
7C8010BA 50           PUSH EDX
7C8010BC 50           PUSH EAX
7C8010BE 50           PUSH ECX
7C8010C0 50           PUSH EDX
7C8010C2 50           PUSH EAX
7C8010C4 50           PUSH ECX
7C8010C6 50           PUSH EDX
7C8010C8 50           PUSH EAX
7C8010CA 50           PUSH ECX
7C8010CC 50           PUSH EDX
7C8010CE 50           PUSH EAX
7C8010D0 50           PUSH ECX
7C8010D2 50           PUSH EDX
7C8010D4 50           PUSH EAX
7C8010D6 50           PUSH ECX
7C8010D8 50           PUSH EDX
7C8010DA 50           PUSH EAX
7C8010DC 50           PUSH ECX
7C8010DE 50           PUSH EDX
7C8010E0 50           PUSH EAX
7C8010E2 50           PUSH ECX
7C8010E4 50           PUSH EDX
7C8010E6 50           PUSH EAX
7C8010E8 50           PUSH ECX
7C8010EA 50           PUSH EDX
7C8010EC 50           PUSH EAX
7C8010EE 50           PUSH ECX
7C8010F0 50           PUSH EDX
7C8010F2 50           PUSH EAX
7C8010F4 50           PUSH ECX
7C8010F6 50           PUSH EDX
7C8010F8 50           PUSH EAX
7C8010FA 50           PUSH ECX
7C8010FC 50           PUSH EDX
7C8010FE 50           PUSH EAX
7C801100 50           PUSH ECX
7C801102 50           PUSH EDX
7C801104 50           PUSH EAX
7C801106 50           PUSH ECX
7C801108 50           PUSH EDX
7C80110A 50           PUSH EAX
7C80110C 50           PUSH ECX
7C80110E 50           PUSH EDX
7C801110 50           PUSH EAX
7C801112 50           PUSH ECX
7C801114 50           PUSH EDX
7C801116 50           PUSH EAX
7C801118 50           PUSH ECX
7C80111A 50           PUSH EDX
7C80111C 50           PUSH EAX
7C80111E 50           PUSH ECX
7C801120 50           PUSH EDX
7C801122 50           PUSH EAX
7C801124 50           PUSH ECX
7C801126 50           PUSH EDX
7C801128 50           PUSH EAX
7C80112A 50           PUSH ECX
7C80112C 50           PUSH EDX
7C80112E 50           PUSH EAX
7C801130 50           PUSH ECX
7C801132 50           PUSH EDX
7C801134 50           PUSH EAX
7C801136 50           PUSH ECX
7C801138 50           PUSH EDX
7C80113A 50           PUSH EAX
7C80113C 50           PUSH ECX
7C80113E 50           PUSH EDX
7C801140 50           PUSH EAX
7C801142 50           PUSH ECX
7C801144 50           PUSH EDX
7C801146 50           PUSH EAX
7C801148 50           PUSH ECX
7C80114A 50           PUSH EDX
7C80114C 50           PUSH EAX
7C80114E 50           PUSH ECX
7C801150 50           PUSH EDX
7C801152 50           PUSH EAX
7C801154 50           PUSH ECX
7C801156 50           PUSH EDX
7C801158 50           PUSH EAX
7C80115A 50           PUSH ECX
7C80115C 50           PUSH EDX
7C80115E 50           PUSH EAX
7C801160 50           PUSH ECX
7C801162 50           PUSH EDX
7C801164 50           PUSH EAX
7C801166 50           PUSH ECX
7C801168 50           PUSH EDX
7C80116A 50           PUSH EAX
7C80116C 50           PUSH ECX
7C80116E 50           PUSH EDX
7C801170 50           PUSH EAX
7C801172 50           PUSH ECX
7C801174 50           PUSH EDX
7C801176 50           PUSH EAX
7C801178 50           PUSH ECX
7C80117A 50           PUSH EDX
7C80117C 50           PUSH EAX
7C80117E 50           PUSH ECX
7C801180 50           PUSH EDX
7C801182 50           PUSH EAX
7C801184 50           PUSH ECX
7C801186 50           PUSH EDX
7C801188 50           PUSH EAX
7C80118A 50           PUSH ECX
7C80118C 50           PUSH EDX
7C80118E 50           PUSH EAX
7C801190 50           PUSH ECX
7C801192 50           PUSH EDX
7C801194 50           PUSH EAX
7C801196 50           PUSH ECX
7C801198 50           PUSH EDX
7C80119A 50           PUSH EAX
7C80119C 50           PUSH ECX
7C80119E 50           PUSH EDX
7C8011A0 50           PUSH EAX
7C8011A2 50           PUSH ECX
7C8011A4 50           PUSH EDX
7C8011A6 50           PUSH EAX
7C8011A8 50           PUSH ECX
7C8011AA 50           PUSH EDX
7C8011AC 50           PUSH EAX
7C8011AE 50           PUSH ECX
7C8011B0 50           PUSH EDX
7C8011B2 50           PUSH EAX
7C8011B4 50           PUSH ECX
7C8011B6 50           PUSH EDX
7C8011B8 50           PUSH EAX
7C8011BA 50           PUSH ECX
7C8011BC 50           PUSH EDX
7C8011BE 50           PUSH EAX
7C8011C0 50           PUSH ECX
7C8011C2 50           PUSH EDX
7C8011C4 50           PUSH EAX
7C8011C6 50           PUSH ECX
7C8011C8 50           PUSH EDX
7C8011CA 50           PUSH EAX
7C8011CC 50           PUSH ECX
7C8011CE 50           PUSH EDX
7C8011D0 50           PUSH EAX
7C8011D2 50           PUSH ECX
7C8011D4 50           PUSH EDX
7C8011D6 50           PUSH EAX
7C8011D8 50           PUSH ECX
7C8011DA 50           PUSH EDX
7C8011DC 50           PUSH EAX
7C8011DE 50           PUSH ECX
7C8011E0 50           PUSH EDX
7C8011E2 50           PUSH EAX
7C8011E4 50           PUSH ECX
7C8011E6 50           PUSH EDX
7C8011E8 50           PUSH EAX
7C8011EA 50           PUSH ECX
7C8011EC 50           PUSH EDX
7C8011EE 50           PUSH EAX
7C8011F0 50           PUSH ECX
7C8011F2 50           PUSH EDX
7C8011F4 50           PUSH EAX
7C8011F6 50           PUSH ECX
7C8011F8 50           PUSH EDX
7C8011FA 50           PUSH EAX
7C8011FC 50           PUSH ECX
7C8011FE 50           PUSH EDX
7C801200 50           PUSH EAX
7C801202 50           PUSH ECX
7C801204 50           PUSH EDX
7C801206 50           PUSH EAX
7C801208 50           PUSH ECX
7C80120A 50           PUSH EDX
7C80120C 50           PUSH EAX
7C80120E 50           PUSH ECX
7C801210 50           PUSH EDX
7C801212 50           PUSH EAX
7C801214 50           PUSH ECX
7C801216 50           PUSH EDX
7C801218 50           PUSH EAX
7C80121A 50           PUSH ECX
7C80121C 50           PUSH EDX
7C80121E 50           PUSH EAX
7C801220 50           PUSH ECX
7C801222 50           PUSH EDX
7C801224 50           PUSH EAX
7C801226 50           PUSH ECX
7C801228 50           PUSH EDX
7C80122A 50           PUSH EAX
7C80122C 50           PUSH ECX
7C80122E 50           PUSH EDX
7C801230 50           PUSH EAX
7C801232 50           PUSH ECX
7C801234 50           PUSH EDX
7C801236 50           PUSH EAX
7C801238 50           PUSH ECX
7C80123A 50           PUSH EDX
7C80123C 50           PUSH EAX
7C80123E 50           PUSH ECX
7C801240 50           PUSH EDX
7C801242 50           PUSH EAX
7C801244 50           PUSH ECX
7C801246 50           PUSH EDX
7C801248 50           PUSH EAX
7C80124A 50           PUSH ECX
7C80124C 50           PUSH EDX
7C80124E 50           PUSH EAX
7C801250 50           PUSH ECX
7C801252 50           PUSH EDX
7C801254 50           PUSH EAX
7C801256 50           PUSH ECX
7C801258 50           PUSH EDX
7C80125A 50           PUSH EAX
7C80125C 50           PUSH ECX
7C80125E 50           PUSH EDX
7C801260 50           PUSH EAX
7C801262 50           PUSH ECX
7C801264 50           PUSH EDX
7C801266 50           PUSH EAX
7C801268 50           PUSH ECX
7C80126A 50           PUSH EDX
7C80126C 50           PUSH EAX
7C80126E 50           PUSH ECX
7C801270 50           PUSH EDX
7C801272 50           PUSH EAX
7C801274 50           PUSH ECX
7C801276 50           PUSH EDX
7C801278 50           PUSH EAX
7C80127A 50           PUSH ECX
7C80127C 50           PUSH EDX
7C80127E 50           PUSH EAX
7C801280 50           PUSH ECX
7C801282 50           PUSH EDX
7C801284 50           PUSH EAX
7C801286 50           PUSH ECX
7C801288 50           PUSH EDX
7C80128A 50           PUSH EAX
7C80128C 50           PUSH ECX
7C80128E 50           PUSH EDX
7C801290 50           PUSH EAX
7C801292 50           PUSH ECX
7C801294 50           PUSH EDX
7C801296 50           PUSH EAX
7C801298 50           PUSH ECX
7C80129A 50           PUSH EDX
7C80129C 50           PUSH EAX
7C80129E 50           PUSH ECX
7C8012A0 50           PUSH EDX
7C8012A2 50           PUSH EAX
7C8012A4 50           PUSH ECX
7C8012A6 50           PUSH EDX
7C8012A8 50           PUSH EAX
7C8012AA 50           PUSH ECX
7C8012AC 50           PUSH EDX
7C8012AE 50           PUSH EAX
7C8012B0 50           PUSH ECX
7C8012B2 50           PUSH EDX
7C8012B4 50           PUSH EAX
7C8012B6 50           PUSH ECX
7C8012B8 50           PUSH EDX
7C8012BA 50           PUSH EAX
7C8012BC 50           PUSH ECX
7C8012BE 50           PUSH EDX
7C8012C0 50           PUSH EAX
7C8012C2 50           PUSH ECX
7C8012C4 50           PUSH EDX
7C8012C6 50           PUSH EAX
7C8012C8 50           PUSH ECX
7C8012CA 50           PUSH EDX
7C8012CC 50           PUSH EAX
7C8012CE 50           PUSH ECX
7C8012D0 50           PUSH EDX
7C8012D2 50           PUSH EAX
7C8012D4 50           PUSH ECX
7C8012D6 50           PUSH EDX
7C8012D8 50           PUSH EAX
7C8012DA 50           PUSH ECX
7C8012DC 50           PUSH EDX
7C8012DE 50           PUSH EAX
7C8012E0 50           PUSH ECX
7C8012E2 50           PUSH EDX
7C8012E4 50           PUSH EAX
7C8012E6 50           PUSH ECX
7C8012E8 50           PUSH EDX
7C8012EA 50           PUSH EAX
7C8012EC 50           PUSH ECX
7C8012EE 50           PUSH EDX
7C8012F0 50           PUSH EAX
7C8012F2 50           PUSH ECX
7C8012F4 50           PUSH EDX
7C8012F6 50           PUSH EAX
7C8012F8 50           PUSH ECX
7C8012FA 50           PUSH EDX
7C8012FC 50           PUSH EAX
7C8012FE 50           PUSH ECX
7C801300 50           PUSH EDX
7C801302 50           PUSH EAX
7C801304 50           PUSH ECX
7C801306 50           PUSH EDX
7C801308 50           PUSH EAX
7C80130A 50           PUSH ECX
7C80130C 50           PUSH EDX
7C80130E 50           PUSH EAX
7C801310 50           PUSH ECX
7C801312 50           PUSH EDX
7C801314 50           PUSH EAX
7C801316 50           PUSH ECX
7C801318 50           PUSH EDX
7C80131A 50           PUSH EAX
7C80131C 50           PUSH ECX
7C80131E 50           PUSH EDX
7C801320 50           PUSH EAX
7C801322 50           PUSH ECX
7C801324 50           PUSH EDX
7C801326 50           PUSH EAX
7C801328 50           PUSH ECX
7C80132A 50           PUSH EDX
7C80132C 50           PUSH EAX
7C80132E 50           PUSH ECX
7C801330 50           PUSH EDX
7C801332 50           PUSH EAX
7C801334 50           PUSH ECX
7C801336 50           PUSH EDX
7C801338 50           PUSH EAX
7C80133A 50           PUSH ECX
7C80133C 50           PUSH EDX
7C80133E 50           PUSH EAX
7C801340 50           PUSH ECX
7C801342 50           PUSH EDX
7C801344 50           PUSH EAX
7C801346 50           PUSH ECX
7C801348 50           PUSH EDX
7C80134A 50           PUSH EAX
7C80134C 50           PUSH ECX
7C80134E 50           PUSH EDX
7C801350 50           PUSH EAX
7C801352 50           PUSH ECX
7C801354 50           PUSH EDX
7C801356 50           PUSH EAX
7C801358 50           PUSH ECX
7C80135A 50           PUSH EDX
7C80135C 50           PUSH EAX
7C80135E 50           PUSH ECX
7C801360 50           PUSH EDX
7C801362 50           PUSH EAX
7C801364 50           PUSH ECX
7C801366 50           PUSH EDX
7C801368 50           PUSH EAX
7C80136A 50           PUSH ECX
7C80136C 50           PUSH EDX
7C80136E 50           PUSH EAX
7C801370 50           PUSH ECX
7C801372 50           PUSH EDX
7C801374 50           PUSH EAX
7C801376 50           PUSH ECX
7C801378 50           PUSH EDX
7C80137A 50           PUSH EAX
7C80137C 50           PUSH ECX
7C80137E 50           PUSH EDX
7C801380 50           PUSH EAX
7C801382 50           PUSH ECX
7C801384 50           PUSH EDX
7C801386 50           PUSH EAX
7C801388 50           PUSH ECX
7C80138A 50           PUSH EDX
7C80138C 50           PUSH EAX
7C80138E 50           PUSH ECX
7C801390 50           PUSH EDX
7C801392 50           PUSH EAX
7C801394 50           PUSH ECX
7C801396 50           PUSH EDX
7C801398 50           PUSH EAX
7C80139A 50           PUSH ECX
7C80139C 50           PUSH EDX
7C80139E 50           PUSH EAX
7C8013A0 50           PUSH ECX
7C8013A2 50           PUSH EDX
7C8013A4 50           PUSH EAX
7C8013A6 50           PUSH ECX
7C8013A8 50           PUSH EDX
7C8013AA 50           PUSH EAX
7C8013AC 50           PUSH ECX
7C8013AE 50           PUSH EDX
7C8013B0 50           PUSH EAX
7C8013B2 50           PUSH ECX
7C8013B4 50           PUSH EDX
7C8013B6 50           PUSH EAX
7C8013B8 50           PUSH ECX
7C8013BA 50           PUSH EDX
7C8013BC 50           PUSH EAX
7C8013BE 50           PUSH ECX
7C8013C0 50           PUSH EDX
7C8013C2 50           PUSH EAX
7C8013C4 50           PUSH ECX
7C8013C6 50           PUSH EDX
7C8013C8 50           PUSH EAX
7C8013CA 50           PUSH ECX
7C8013CC 50           PUSH EDX
7C8013CE 50           PUSH EAX
7C8013D0 50           PUSH ECX
7C8013D2 50           PUSH EDX
7C8013D4 50           PUSH EAX
7C8013D6 50           PUSH ECX
7C8013D8 50           PUSH EDX
7C8013DA 50           PUSH EAX
7C8013DC 50           PUSH ECX
7C8013DE 50           PUSH EDX
7C8013E0 50           PUSH EAX
7C8013E2 50           PUSH ECX
7C8013E4 50           PUSH EDX
7C8013E6 50           PUSH EAX
7C8013E8 50           PUSH ECX
7C8013EA 50           PUSH EDX
7C8013EC 50           PUSH EAX
7C8013EE 50           PUSH ECX
7C8013F0 50           PUSH EDX
7C8013F2 50           PUSH EAX
7C8013F4 50           PUSH ECX
7C8013F6 50           PUSH EDX
7C8013F8 50           PUSH EAX
7C8013FA 50           PUSH ECX
7C8013FC 50           PUSH EDX
7C8013FE 50           PUSH EAX
7C801400 50           PUSH ECX
7C801402 50           PUSH EDX
7C801404 50           PUSH EAX
7C801406 50           PUSH ECX
7C801408 50           PUSH EDX
7C80140A 50           PUSH EAX
7C80140C 50           PUSH ECX
7C80140E 50           PUSH EDX
7C801410 50           PUSH EAX
7C801412 50           PUSH ECX
7C801414 50           PUSH EDX
7C801416 50           PUSH EAX
7C801418 50           PUSH ECX
7C80141A 50           PUSH EDX
7C80141C 50           PUSH EAX
7C80141E 50           PUSH ECX
7C801420 50           PUSH EDX
7C801422 50           PUSH EAX
7C801424 50           PUSH ECX
7C801426 50           PUSH EDX
7C801428 50           PUSH EAX
7C80142A 50           PUSH ECX
7C80142C 50           PUSH EDX
7C80142E 50           PUSH EAX
7C801430 50           PUSH ECX
7C801432 50           PUSH EDX
7C801434 50           PUSH EAX
7C801436 50           PUSH ECX
7C801438 50           PUSH EDX
7C80143A 50           PUSH EAX
7C80143C 50           PUSH ECX
7C80143E 50           PUSH EDX
7C801440 50           PUSH EAX
7C801442 50           PUSH ECX
7C801444 50           PUSH EDX
7C801446 50           PUSH EAX
7C801448 50           PUSH ECX
7C80144A 50           PUSH EDX
7C80144C 50           PUSH EAX
7C80144E 50           PUSH ECX
7C801450 50           PUSH EDX
7C801452 50           PUSH EAX
7C801454 50           PUSH ECX
7C801456 50           PUSH EDX
7C801458 50           PUSH EAX
7C80145A 50           PUSH ECX
7C80145C 50           PUSH EDX
7C80145E 50           PUSH EAX
7C801460 50           PUSH ECX
7C801462 50           PUSH EDX
7C801464 50           PUSH EAX
7C801466 50           PUSH ECX
7C801468 50           PUSH EDX
7C80146A 50           PUSH EAX
7C80146C 50           PUSH ECX
7C80146E 50           PUSH EDX
7C801470 50           PUSH EAX
7C801472 50           PUSH ECX
7C801474 50           PUSH EDX
7C801476 50           PUSH EAX
7C801478 50           PUSH ECX
7C80147A 50           PUSH EDX
7C80147C 50           PUSH EAX
7C80147E 50           PUSH ECX
7C801480 50           PUSH EDX
7C801482 50           PUSH EAX
7C801484 50           PUSH ECX
7C801486 50           PUSH EDX
7C801488 50           PUSH EAX
7C80148A 50           PUSH ECX
7C80148C 50           PUSH EDX
7C80148E 50           PUSH EAX
7C801490 50           PUSH ECX
7C801492 50           PUSH EDX
7C801494 50           PUSH EAX
7C801496 50           PUSH ECX
7C801498 50           PUSH EDX
7C80149A 50           PUSH EAX
7C80149C 50           PUSH ECX
7C80149E 50           PUSH EDX
7C8014A0 50           PUSH EAX
7C8014A2 50           PUSH ECX
7C8014A4 50           PUSH EDX
7C8014A6 50           PUSH EAX
7C8014A8 50           PUSH ECX
7C8014AA 50           PUSH EDX
7C8014AC 50           PUSH EAX
7C8014AE 50           PUSH ECX
7C8014B0 50           PUSH EDX
7C8014B2 50           PUSH EAX
7C8014B4 50           PUSH ECX
7C8014B6 50           PUSH EDX
7C8014B8 50           PUSH EAX
7C8014BA 50           PUSH ECX
7C8014BC 50           PUSH EDX
7C8014BE 50           PUSH EAX
7C8014C0 50           PUSH ECX
7C8014C2 50           PUSH EDX
7C8014C4 50           PUSH EAX
7C8014C6 50           PUSH ECX
7C8014C8 50           PUSH EDX
7C8014CA 50           PUSH EAX
7C8014CC 50           PUSH ECX
7C8014CE 50           PUSH EDX
7C8014D0 50           PUSH EAX
7C8014D2 50           PUSH ECX
7C8014D4 50           PUSH EDX
7C8014D6 50           PUSH EAX
7C8014D8 50           PUSH ECX
7C8014DA 50           PUSH EDX
7C8014DC 50           PUSH EAX
7C8014DE 50           PUSH ECX
7C8014E0 50           PUSH EDX
7C8014E2 50           PUSH EAX
7C8014E4 50           PUSH ECX
7C8014E6 50           PUSH EDX
7C8014E8 50           PUSH EAX
7C8014EA 50           PUSH ECX
7C8014EC 50           PUSH EDX
7C8014EE 50           PUSH EAX
7C8014F0 50           PUSH ECX
7C8014F2 50           PUSH EDX
7C8014F4 50           PUSH EAX
7C8014F6 50           PUSH ECX
7C8014F8 50           PUSH EDX
7C8014FA 50           PUSH EAX
7C8014FC 50           PUSH ECX
7C8014FE 50           PUSH EDX
7C801500 50           PUSH EAX
7C801502 50           PUSH ECX
7C801504 50           PUSH EDX
7C801506 50           PUSH EAX
7C801508 50           PUSH ECX
7C80150A 50           PUSH EDX
7C80150C 50           PUSH EAX
7C80150E 50           PUSH ECX
7C801510 50           PUSH EDX
7C801512 50           PUSH EAX
7C801514 50           PUSH ECX
7C801516 50           PUSH EDX
7C801518 50           PUSH EAX
7C80151A 50           PUSH ECX
7C80151C 50           PUSH EDX
7C80151E 50           PUSH EAX
7C801520 50           PUSH ECX
7C801522 50           PUSH EDX
7C801524 50           PUSH EAX
7C801526 50           PUSH ECX
7C801528 50           PUSH EDX
7C80152A 50           PUSH EAX
7C80152C 50           PUSH ECX
7C80152E 50           PUSH EDX
7C801530 50           PUSH EAX
7C801532 50           PUSH ECX
7C801534 50           PUSH EDX
7C801536 50           PUSH EAX
7C801538 50           PUSH ECX
7C80153A 50           PUSH EDX
7C80153C 50           PUSH EAX
7C80153E 50           PUSH ECX
7C801540 50           PUSH EDX
7C801542 50           PUSH EAX
7C801544 50           PUSH ECX
7C801546 50           PUSH EDX
7C801548 50           PUSH EAX
7C80154A 50           PUSH ECX
7C80154C 50           PUSH EDX
7C80154E 50           PUSH EAX
7C801550 50           PUSH ECX
7C801552 50           PUSH EDX
7C801554 50           PUSH EAX
7C801556 50           PUSH ECX
7C801558 50           PUSH EDX
7C80155A 50           PUSH EAX
7C80155C 50           PUSH ECX
7C80155E 50           PUSH EDX
7C801560 50           PUSH EAX
7C801562 50           PUSH ECX
7C801564 50           PUSH EDX
7C801566 50           PUSH EAX
7C801568 50           PUSH ECX
7C80156A 50           PUSH EDX
7C80156C 50           PUSH EAX
7C80156E 50           PUSH ECX
7C801570 50           PUSH EDX
7C801572 50           PUSH EAX
7C801574 50           PUSH ECX
7C801576 50           PUSH EDX
7C8015
```

ntdll.dll의 NtCreateFile() 호출됨.

```
7C93D682  B8 25000000  MOV EAX, 25
7C93D687  BA 0003FE7F  MOV EDX, 7FFE0300
7C93D68C  FF12        CALL DWORD PTR DS:[EDX]
7C93D68E  C2 2C00     RETN 2C
```

위의 코드는 ntdll의 NtCreateFile()를 Disassemble한 것인데, eax의 값은 호출하고자 하는 Native API의 Service Index이고, edx의 값은 KiSystemCallEntry()의 주소를 가진 포인터 값입니다.

ntdll.dll의 KiFastSystemCall() 호출됨.

```
7C93EB88  8B04        MOV EDX, ESP
7C93EB8D  0F34       SYSENTER
7C93EB91  90         NOP
7C93EB92  90         NOP
7C93EB93  90         NOP
7C93EB94  C3         RETN
```

현재 ESP값을 EDX에 저장 시키고 SYSENTER 명령어를 실행 시킨다. SYSENTER 명령어는 MSR중 IA32\_SYSENTER\_EIP(0x176)번에 저장되어 있는 함수를 실행 시킵니다.

----- Ring 0 -----

ntoskrnl.exe의 KiFastCallEntry() 호출됨.

```
804E06F0 | B9 23000000 | MOV ECK, 23
804E06F5 | 6A 30       | PUSH 30
804E06F7 | 0FA1       | POP FS
804E06F9 | 8ED9       | MOV DS, CX
804E06FB | 8EC1       | MOV ES, CX
804E06FD | 8B0D 40F0DFFF | MOV ECK, DWORD PTR [FFDFF040]
804E0703 | 8B61 04     | MOV ESP, DWORD PTR [ECK+4]
804E0706 | 6A 23      | PUSH 23
804E0708 | 52        | PUSH EDX
804E0709 | 9C        | PUSHFD
804E070A | 6A 02     | PUSH 2
```

어떠한 조작을 가하지 않는 한 IA32\_SYSENTER\_EIP에는 KiFastCallEntry()의 주소가 저장되어 있습니다. KiFastCallEntry()는 Kernel단에서 필요한 작업을 처리하기 위한 준비 작업을 합니다. KiFastCallEntry()는 현재 Thread의 ETHREAD 구조체의 ServiceTable를 읽어서 Table의 주소를 구합니다. 조작을 가하지 않는한 EPROCESS의 ServiceTable에는 System Thread라면 ntoskrnl.exe에 의해 export되어 지는 KeServiceDescriptorTable의 주소가, GUI Thread라면 KeServiceDescriptorTableShadow의 주소가 담겨 있습니다.

ntoskrnl.exe의 ZwCreateFile() 호출됨.

```
80572D48 | 8BFF          | MOU    EDI, EDI
80572D4A | 55           | PUSH   EBP
80572D4B | 8BEC        | MOU    EBP, ESP
80572D4D | 33C0        | XOR    EAX, EAX
80572D4F | 50          | PUSH   EAX
80572D50 | 50          | PUSH   EAX
80572D51 | 50          | PUSH   EAX
80572D52 | FF75 30     | PUSH   DWORD PTR [EBP+30]
```

eax레지스터로 넘어온 Service Index를 참조하여 KiFastCallEntry()는 SSDT에서 실제 함수의 주소를 구해 호출합니다.

----- Ring 3 -----

ntdll.dll의 KiSystemCallRet() 호출됨

```
7C98EB94 | C3          | RETN
```

KiFastCallEntry()에 의해 실제 함수의 Call처리가 완료된 후, SYSEXIT명령을 이용하여 Ring3로 돌아오게 됩니다.

ntdll.dll의 NtCreateFile()로 리턴됨.

```
7C93E10E | C2 0800    | RETN 8
```

kernel32.dll의 CreateFileW()로 리턴됨.

kernel32.dll의 CreateFileA()로 리턴됨.

Application의 코드로 리턴됨.

Kernel단의 처리가 필요한 API들은 대부분 위와 같은 과정을 통해 수행 되어 집니다. 이 때 Hooking할 수 있는 기회는 다음과 같습니다.

### Ring 3 :

- 1) Application의 Import Descriptor Table에서 CreateFileA()의 주소 수정
- 2) kernel32.dll의 Export Descriptor Table에서 CreateFileA()의 주소 수정
- 3) kernel32.dll의 Import Descriptor Table에서 NtCreateFile()의 주소 수정
- 3) kernel32.dll의 CreateFileA() Inline Hooking
- 4) kernel32.dll의 CreateFileW() Inline Hooking
- 5) ntdll.dll의 Export Descriptor Table에서 NtCreateFile()의 주소 수정
- 6) ntdll.dll의 NtCreateFile() Inline Hooking or HotByte Hooking
- 7) ntdll.dll의 KiFastSystemCall() Inline Hooking

## Ring 0 :

- 1) MSR의 IA32\_SYSENTER\_EIP(0x176)번 수정 - Sysenter Hooking  
or IDT의 0x2e번 수정 - KiSystemService() Hooking
- 2) KiFastCallEntry() Inline Hooking
- 3) KiSystemService() Inline Hooking
- 4) SSDT Hooking
- 5) NTAPI Inline Hooking

위와 같이 공격자 입장에서는 Hooking할 기회가 많이 있습니다.  
이 글에서 위의 모든 내용을 다루는 방법이나 방어법에 관하여 모두  
다룰 수는 없습니다. 하지만 이미 NET상에는 해당 방법에 관한 많은  
문서들이 올라와 있음으로 어렵지 않게 자료를 구하실 수 있을 겁니다.

## 3. SSDT Hooking 무력화의 원리와 코드

### 1) SDT-RESTORE by Chew Keong TAN

SIG<sup>2</sup> G-TEC Lab에 의해 발표된 방법입니다.  
핵심적인 것은 Ring3에서 WdeviceWphysicalmemory Section을  
NtOpenSection()로 열어 Kernel Memory에 접근할 수 있다는 점과,  
NtQuerySystemInformation()을 이용하여 ntoskrnl.exe가 로드되어 있는  
Base주소를 구할 수 있다는 점, ntdll.dll의 Export Table에는 KiServiceTable  
에 존재하는 함수들의 이름이 모두 존재한다는 것입니다.

코드의 내용 중 중요한 부분을 살펴보도록 하겠습니다.

```
BOOL getNativeAPIs(void)
{
    HMODULE hntdll;

    hntdll = GetModuleHandle("ntdll.dll");

    *(FARPROC *)&_RtlAnsiStringToUnicodeString =
        GetProcAddress(hntdll, "RtlAnsiStringToUnicodeString");

    *(FARPROC *)&_RtlInitAnsiString =
        GetProcAddress(hntdll, "RtlInitAnsiString");

    *(FARPROC *)&_RtlFreeUnicodeString =
        GetProcAddress(hntdll, "RtlFreeUnicodeString");

    *(FARPROC *)&_NtOpenSection =
        GetProcAddress(hntdll, "NtOpenSection");

    *(FARPROC *)&_NtMapViewOfSection =
```

```

        GetProcAddress(hntdll, "NtMapViewOfSection");

*(FARPROC *)&_NtUnmapViewOfSection =
        GetProcAddress(hntdll, "NtUnmapViewOfSection");

*(FARPROC *)&_NtQuerySystemInformation =
        GetProcAddress(hntdll, "ZwQuerySystemInformation");

if(_RtlAnsiStringToUnicodeString && _RtlInitAnsiString &&
_RtlFreeUnicodeString &&
        _NtOpenSection && _NtMapViewOfSection &&
_NtUnmapViewOfSection && _NtQuerySystemInformation)
{
        return TRUE;
}
return FALSE;
}

```

GetModuleHandle()을 이용하여 현재 Application에 Load되어 있는 ntdll.dll의 Base주소를 구해온 후, GetProcAddress()를 사용하여 사용할 함수들의 주소를 구해오는 함수입니다.

```

HANDLE openPhyMem()
{
        HANDLE hPhyMem;
        OBJECT_ATTRIBUTES oAttr;

        ANSI_STRING aStr;

        _RtlInitAnsiString(&aStr, "WWdeviceWWphysicalmemory");

        UNICODE_STRING uStr;

        if(_RtlAnsiStringToUnicodeString(&uStr, &aStr, TRUE) != STATUS_SUCCESS)
        {
                return INVALID_HANDLE_VALUE;
        }

        oAttr.Length = sizeof(OBJECT_ATTRIBUTES);
        oAttr.RootDirectory = NULL;
        oAttr.Attributes = OBJ_CASE_INSENSITIVE;
        oAttr.ObjectName = &uStr;
        oAttr.SecurityDescriptor = NULL;
        oAttr.SecurityQualityOfService = NULL;

        if(_NtOpenSection(&hPhyMem, SECTION_MAP_READ |
SECTION_MAP_WRITE, &oAttr) != STATUS_SUCCESS)
        {
                return INVALID_HANDLE_VALUE;
        }
}

```

```

    }

    return hPhyMem;
}

```

ntdll의 NtOpenSection()를 이용하여 \Device\PhysicalMemory를 READ & WRITE권한으로 열어 줍니다.

```

LPVOID loadDLL(char *dllName)
{
    char moduleFilename[MAX_PATH + 1];
    LPVOID ptrLoc = NULL;
    MZHeader mzH2;
    PE_Header peH2;
    PE_ExtHeader peXH2;
    SectionHeader *secHdr2;

    GetSystemDirectory(moduleFilename, MAX_PATH);
    if((myStrlenA(moduleFilename) + myStrlenA(dllName)) >= MAX_PATH)
        return NULL;

    strcat(moduleFilename, dllName);

    // load this EXE into memory because we need its original Import Hint Table

    HANDLE fp;
    fp = CreateFile(moduleFilename, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);

    if(fp != INVALID_HANDLE_VALUE)
    {
        BY_HANDLE_FILE_INFORMATION fileInfo;
        GetFileInformationByHandle(fp, &fileInfo);

        DWORD fileSize = fileInfo.nFileSizeLow;
        //printf("Size = %d\n", fileSize);
        if(fileSize)
        {
            LPVOID exePtr = HeapAlloc(GetProcessHeap(), 0, fileSize);
            if(exePtr)
            {
                DWORD read;

                if(ReadFile(fp, exePtr, fileSize, &read, NULL)
&& read == fileSize)
                {
                    if(readPEInfo((char *)exePtr, &mzH2,
&peH2, &peXH2, &secHdr2))

```

```

        {
            int imageSize
            = calcTotalImageSize(&mzH2, &peH2, &peXH2, secHdr2);

            //ptrLoc =
VirtualAlloc(NULL, imageSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
            ptrLoc =
HeapAlloc(GetProcessHeap(), 0, imageSize);
            if(ptrLoc)
            {
loadPE((char *)exePtr, &mzH2, &peH2, &peXH2, secHdr2, ptrLoc);
                }
            }

            HeapFree(GetProcessHeap(), 0, exePtr);
        }
    }
    CloseHandle(fp);
}

return ptrLoc;
}

```

시스템 디렉토리에서 인자로 넘어온 프로그램을 열어 메모리를 할당하여 EXECUTE & READ & WRITE 권한으로 메모리를 할당하고 로드하는 역할을 합니다.

```

DWORD procAPIExportAddr(DWORD hModule, char *apiName)
{
    if(!hModule || !apiName)
        return 0;

    char *ptr = (char *)hModule;
    ptr += 0x3c;          // offset 0x3c contains offset to PE header

    ptr = (char *)*(DWORD *)ptr + hModule + 0x78;
        // offset 78h into PE header contains addr of export table

    ptr = (char *)*(DWORD *)ptr + hModule;
        // ptr now points to export directory table

    // offset 24 into the export directory table == number of entries
    // in the Export Name Pointer Table

    // table
    DWORD numEntries = *(DWORD *)ptr + 24;
    //printf("NumEntries = %d\n", numEntries);
}

```

```

        DWORD *ExportNamePointerTable = (DWORD *)
            (*(DWORD *)(ptr + 32) + hModule);
// offset 32 into export directory contains offset to Export Name Pointer Table

        DWORD ordinalBase = *((DWORD *)(ptr + 16));
//printf("OrdinalBase is %dWn", ordinalBase);

        WORD *ExportOrdinalTable = (WORD *)
            (*(DWORD *)(ptr + 36) + hModule);
// offset 36 into export directory contains offset to Ordinal Table
        DWORD *ExportAddrTable = (DWORD *)
            (*(DWORD *)(ptr + 28) + hModule);
// offset 28 into export directory contains offset to Export Addr Table

        for(DWORD i = 0; i < numEntries; i++)
        {
            char *exportName = (char *)(ExportNamePointerTable[i]
+ hModule);

            if(myStrcmpA(exportName, apiName) == TRUE)
            {
                WORD ordinal = ExportOrdinalTable[i];
                //printf("%s (i = %d) Ordinal = %d at %XWn",
                    // exportName, i, ordinal, ExportAddrTable[ordinal]);

                return (DWORD)(ExportAddrTable[ordinal]);
            }
        }

        return 0;
}

```

첫 번째 인자로 넘어온 Base주소에 Load되어 있는 PE 포맷을 가진 파일로부터 두 번째 인자로 넘어온 이름을 Export Descriptor Table에서 찾아서 리턴하여 줍니다.

```

DWORD getKernelBase(void)
{
    HANDLE hHeap = GetProcessHeap();

    NTSTATUS Status;
    ULONG cbBuffer = 0x8000;
    PVOID pBuffer = NULL;
    DWORD retVal = DEF_KERNEL_BASE;

    do
    {
        pBuffer = HeapAlloc(hHeap, 0, cbBuffer);

```

```

        if (pBuffer == NULL)
            return DEF_KERNEL_BASE;

        Status = _NtQuerySystemInformation(SystemModuleInformation,
            pBuffer, cbBuffer, NULL);

        if(Status == STATUS_INFO_LENGTH_MISMATCH)
        {
            HeapFree(hHeap, 0, pBuffer);
            cbBuffer *= 2;
        }
        else if(Status != STATUS_SUCCESS)
        {
            HeapFree(hHeap, 0, pBuffer);
            return DEF_KERNEL_BASE;
        }
    }
    while (Status == STATUS_INFO_LENGTH_MISMATCH);

    DWORD numEntries = *((DWORD *)pBuffer);
    SYSTEM_MODULE_INFORMATION *smi
= (SYSTEM_MODULE_INFORMATION *)((char *)pBuffer + sizeof(DWORD));

    for(DWORD i = 0; i < numEntries; i + )
    {
        if(strcmpi(smi->ImageName, "ntoskrnl.exe"))
        {
            //printf("%.8X - %sWn", smi->Base, smi->ImageName);
            retVal = (DWORD)(smi->Base);
            break;
        }
        smi++;
    }

    HeapFree(hHeap, 0, pBuffer);

    return retVal;
}

```

NtQuerySystemInformation()을 이용하여 Process목록을 열거하고 그 중, "ntoskrnl.exe"를 찾아보고, 찾는데 성공하면 Base주소를 리턴하여 줍니다.

```

BOOL mapPhyMem(HANDLE hPhyMem, DWORD *phyAddr, DWORD *length,
    PVOID *virtualAddr)
{
    NTSTATUS ntStatus;
    PHYSICAL_ADDRESS viewBase;

    *virtualAddr = 0;
}

```

```

viewBase.QuadPart = (ULONGLONG) (*phyAddr);

ntStatus = _NtMapViewOfSection(hPhyMem, (HANDLE)-1, virtualAddr, 0,
    *length, &viewBase, length, ViewShare, 0, PAGE_READWRITE );

if(ntStatus != STATUS_SUCCESS)
{
    printf("Failed to map physical memory view of length %X at %X!",
        *length, *phyAddr);
    return FALSE;
}

*phyAddr = viewBase.LowPart;
return TRUE;
}

```

할당했던 메모리를 물리적 주소를 얻기 위해 메핑 하는 역할을 합니다.

```

BOOL buildNativeAPITable(DWORD hModule, char *nativeAPINames[], DWORD
numNames)
{
    if(!hModule)
        return FALSE;

    char *ptr = (char *)hModule;
    ptr += 0x3c;          // offset 0x3c contains offset to PE header

    ptr = (char *)*(DWORD *)ptr + hModule + 0x78;
// offset 78h into PE header contains addr of export table

    ptr = (char *)*(DWORD *)ptr + hModule;
// ptr now points to export directory table

    // offset 24 into the export directory table == number of entries in the
//Name Pointer Table
// table
    DWORD numEntries = *(DWORD *) (ptr + 24);

    DWORD *ExportNamePointerTable = (DWORD *)*(DWORD *)
        (ptr + 32) + hModule;
// offset 32 into export directory contains offset to Export Name Pointer Table

    DWORD ordinalBase = *((DWORD *) (ptr + 16));

    WORD *ExportOrdinalTable = (WORD *)*((DWORD *)
        (ptr + 36)) + hModule;
// offset 36 into export directory contains offset to Ordinal Table
    DWORD *ExportAddrTable = (DWORD *)*(DWORD *)

```

```

(ptr + 28) + hModule);
// offset 28 into export directory contains offset to Export Addr Table

for(DWORD i = 0; i < numEntries; i++)
{
    // i now contains the index of the API in the Ordinal Table
    // ptr points to Export directory table

    WORD ordinalValue = ExportOrdinalTable[i];
    DWORD apiAddr = (DWORD)ExportAddrTable[ordinalValue]
        + hModule;
    char *exportName = (char*)(ExportNamePointerTable[i]
        + hModule);

    // Win2K
    if(gWinVersion == 0 &&
        *((unsigned char *)apiAddr) == 0xB8 &&
        *((unsigned char *)apiAddr + 9) == 0xCD &&
        *((unsigned char *)apiAddr + 10) == 0x2E)
    {
        DWORD serviceNum = *(DWORD*)((char *)apiAddr + 1);
        if(serviceNum < numNames)
        {
            nativeAPINames[serviceNum] = exportName;
        }
        //printf("%X - %sWn", serviceNum, exportName);
    }

    // WinXP
    else if(gWinVersion == 1 &&
        *((unsigned char *)apiAddr) == 0xB8 &&
        *((unsigned char *)apiAddr + 5) == 0xBA &&
        *((unsigned char *)apiAddr + 6) == 0x00 &&
        *((unsigned char *)apiAddr + 7) == 0x03 &&
        *((unsigned char *)apiAddr + 8) == 0xFE &&
        *((unsigned char *)apiAddr + 9) == 0x7F)
    {
        DWORD serviceNum = *(DWORD*)((char *)apiAddr + 1);
        if(serviceNum < numNames)
        {
            nativeAPINames[serviceNum] = exportName;
        }
        //printf("%X - %sWn", serviceNum, exportName);
    }
}

return TRUE;
}

```

NativeAPI들의 Name배열을 만듭니다.

```

for(DWORD i = 0; i < sdtCount; i + + )
{
    if((serviceTable[i] - PROT_MEMBASE - kernelOffset)
        != fileServiceTable[i])
    {
        printf("%-25s %3X --[hooked by unknown at %X]--\n",
            (nativeApiNames[i] ? nativeApiNames[i] : "Unknown API"),
            i, serviceTable[i]);

        hookCount+ + ;
    }
}

```

메모리에 새롭게 매핑한 ntoskrnl.exe의 함수주소와 기존의 ntoskrnl.exe의 함수주소가 같은지 한 개씩 비교 합니다.

```

for(DWORD i = 0; i < sdtCount; i + + )
{
    if((serviceTable[i] - PROT_MEMBASE - kernelOffset)
        != fileServiceTable[i])
    {
        serviceTable[i] = fileServiceTable[i] + PROT_MEMBASE
            + kernelOffset;
        printf("[+ ] Patched SDT entry %.2X to %.8X\n", I,
            fileServiceTable[i] + PROT_MEMBASE + kernelOffset);
    }
}

```

기존의 ntoskrnl.exe의 함수주소를 한 개씩 비교하면서 새로 할당한 ntoskrnl.exe의 함수 주소와 다르다면, 새로 할당한 ntoskrnl.exe의 함수주소로 바꿔줍니다.

## 정리

실행 순서를 다음과 같이 정리하여 볼 수 있습니다.

- 1) Kernel Memory를 Open 한다.
- 2) Memory에 ntoskrnl.exe를 Load한다.
- 3) Load한 메모리에서 KeServiceDescriptorTable의 주소를 구한다.
- 4) 커널의 베이스주소를 구해온다.
- 5) Load한 메모리를 매핑한다.
- 6) NativeAPI Name 배열을 만든다.
- 7) Hooking된 함수가 있는지 검사한다.
- 8) Hooking된 함수를 정상주소로 돌린다.

### 장점

Ring3에서 ntdll의 함수들을 이용하여 Kernel Memory에 존재하는 SSDT를 검사하고, 고쳐 준다는 점에서 획기적입니다.

### 단점

Rootkit에 의해 WWdeviceWWphysicalmemory을 Open하는 것이 이전에 막혀 있을 경우나 Administrator의 권한이 아닐 경우 작동할 수 없습니다.

## 2) ServiceTable Relocation by Yeori

System Thread의 경우 ServiceTable로써, KeServiceDescriptorTable을, GUI Thread의 경우 KeServiceDescriptorTableShadow을 사용한다고 하였습니다.

이를 확인해 보기 위해 System Process의 Thread와 A.exe라는 User Application의 Thread에 ServiceTable이 각각 가르키는 주소를 덤프하여 보았습니다.

KeServiceDescriptorTable :

```
8055B480  A8 46 4E 80 00 00 00 00 1C 01 00 00 B8 1A 51 80
8055B490  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8055B4A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8055B4B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

KeServiceDescriptorTableShadow :

```
8055B440  A8 46 4E 80 00 00 00 00 1C 01 00 00 B8 1A 51 80
8055B450  00 83 99 BF 00 00 00 00 9B 02 00 00 10 90 99 BF
8055B460  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
8055B470  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

KeServiceDescriptorTableShadow의 경우 두 번째 줄에 KeServiceDescriptorTable의 경우 0으로 되어 있는 부분에 값이 더 들어 있는 것을 볼 수 있습니다. 이는 GUI관련 처리를 하기 위해 필요한 WINAPI Table에 대한 것입니다.

KeServiceDescriptorTable과 KeServiceDescriptorTableShadow를 각각 C구조체로 나타내보면 다음과 같습니다.

```
typedef struct ServiceDescriptorEntry {
    unsigned int *ServiceTableBase; //서비스테이블
    unsigned int *ServiceCounterTableBase; //Debug모드에만 사용됨
    unsigned int NumberOfServices; //서비스의 갯수
    unsigned char *ParamTableBase; //각 서비스당 Param의 총 크기
} ServiceDescriptorTableEntry_t, *PServiceDescriptorTableEntry_t;
```

```
typedef struct ServiceDescriptorShadowEntry {
    unsigned int *ServiceTableBase; //서비스테이블
    unsigned int *ServiceCounterTableBase; //Debug모드에만 사용됨
    unsigned int NumberOfServices; //서비스의 갯수
    unsigned char *ParamTableBase; //각 서비스당 Param의 총 크기
    unsigned int *Win32kTableBase; //WINAPI 서비스 테이블
```

```

    unsigned int *Win32kCounterTableBase; //Debug모드에만 사용됨
    unsigned int NumberOfWin32kServices; //WINAPI서비스의 갯수
    unsigned char *Win32kParamTableBase; //각 서비스당 Param의 총크기
} ServiceDescriptorTableShadowEntry_t,
*PServiceDescriptorTableShadowEntry_t;

```

KiFastCallEntry()는 요구된 NativeAPI를 처리하기 위해 직접적으로 위의 ServiceDescriptorTable들을 사용하는 것이 아니라, 요청한 ETHREAD의 ServiceTable를 참조한다고 하였습니다. 그럼으로 ServiceDescriptorTable을 별도로 만들고, ETHREAD의 ServiceTable을 별도로 만든 ServiceDescriptor Table로 연결시킨다면, SSDT Hooking을 해당 Thread에 한하여 무력화 시킬 수 있을 것입니다. 이 가정을 전제로 만든 코드의 핵심부분을 보겠습니다.

```

SDT = &KeServiceDescriptorTable;

NewServiceTable = ExAllocatePool(NonPagedPool,(SDT->NumberOfServices) * 4);
memcpy(NewServiceTable,SDT->ServiceTableBase,(SDT->NumberOfServices) * 4);

```

Service 개수 \* 4의 크기를 가진 메모리를 할당하고, 그곳에 본래 ServiceTable에 있던값들을 전부 복사하여 넣습니다.

```

OrgSDT = (PServiceDescriptorTableEntry_t)*
        (eThread + ThrdOffset_ServiceTable);

//Thread가 가지고 있던 Original ServiceTable주소 저장.
DbgPrint("Original SDT : 0x%X",OrgSDT);

SDT_s[ThreadCount] = OrgSDT; //관리를 위한 테이블에 저장해둔다.
ETHREAD_s[ThreadCount] = eThread; //“ ”

NewSDT[ThreadCount] = ExAllocatePool(NonPagedPool,128); //새로운 메모리 할당
memcpy(NewSDT[ThreadCount],OrgSDT,128); //복사한다.

ServiceTable[ThreadCount] //서비스 테이블을 위한 메모리 할당.
= ExAllocatePool(NonPagedPool,(SDT->NumberOfServices) * 4);

memcpy(ServiceTable[ThreadCount],NewServiceTable,
        (SDT->NumberOfServices) * 4); //메모리를 복사한다.

DbgPrint("New Service Table : 0x%X",ServiceTable[ThreadCount]);

NewSDT[ThreadCount]->ServiceTableBase = ServiceTable[ThreadCount];
//관리를 위한 테이블에 저장해 둔다.

g_pmdl_KeServiceTable[ThreadCount]
        = MmCreateMdl(NULL,NewSDT[ThreadCount],128);
//MDL을 만든다.

```

```

MmBuildMdlForNonPagedPool(g_pmdl_KeServiceTable[ThreadCount]);

g_pmdl_KeServiceTable[ThreadCount]->MdlFlags
=g_pmdl_KeServiceTable[ThreadCount]->MdlFlags
| MDL_MAPPED_TO_SYSTEM_VA | MDL_WRITE_OPERATION
| MDL_IO_PAGE_READ;
//MDL속성을 변환한다.

Mapped_KeServiceTable[ThreadCount]
= MmMapLockedPages(g_pmdl_KeServiceTable[ThreadCount], KernelMode);

DbgPrint("New SDT : 0x%X",Mapped_KeServiceTable[ThreadCount]);

pmdl_SDT_Pointer = MmCreateMdl(NULL, SDT_Pointer, 4);
MmBuildMdlForNonPagedPool(pmdl_SDT_Pointer);
pmdl_SDT_Pointer->MdlFlags =
pmdl_SDT_Pointer->MdlFlags | MDL_MAPPED_TO_SYSTEM_VA |
MDL_WRITE_OPERATION | MDL_IO_PAGE_READ;

Mapped_SDT_Pointer = MmMapLockedPages(pmdl_SDT_Pointer, KernelMode);

*Mapped_SDT_Pointer = Mapped_KeServiceTable[ThreadCount];
//새로 만든 ServiceTable을 가르키도록 한다.
MmUnmapLockedPages(Mapped_SDT_Pointer,pmdl_SDT_Pointer);

IoFreeMdl(pmdl_SDT_Pointer); //MDL을 해제한다.

```

위의 코드가 SDT Relocation의 핵심적인 부분입니다. 위의 코드는 완성 되지는 않았습니니다. 그 이유는 Thread의 생성단계에서는 Thread의 ETHREAD에 ServiceTable은 모두 KeServiceDescriptorTable을 포인트 하다가, KiFastCallEntry()에 의해 Service를 처리할 때, KiFastCallEntry()내에서 내부적으로 호출하는 PsConvertToGuiThread()라는 함수에 의해서 WINAPI의 처리가 필요할 경우 그때에 KeServiceDescriptorTable Shadow를 포인트 하도록 변하기 때문입니다.

## 정리

**실행 순서를 다음과 같이 정리하여 볼 수 있습니다.**

- 1) 기존의 ServiceTable로부터 함수들의 주소들을 복사한 새로운 ServiceTable을 만든다.
- 2) 지정한 Process의 Thread들을 트레이싱 하면서 각 ETHREAD에 대해 새로운 SDT를 만들어 준다.
- 3) PsSetCreateThreadNotifyRoutine()에 의해 THREAD가 동적생성이 Notify되면, 해당 Thread에 대한 SDT를 만들어 준다.
- 4) 사용자가 기능을 중지하기 원하는 경우, 각 Thread의 ServiceTable을 저장해두었던 것으로 복구시키고, 사용했던 메모리들을 모두 Release시킨다.

## 장점

ServiceTable을 새로 만드는 것임으로, SSDT Hooking에 영향을 받지 않습니다.

## 단점

현재 Code로는 동적으로 생성된 GUI Thread에 대해서는 처리할 수 없습니다.

### 3) KiFastCallEntry Imitation by Dual

Kernel단의 처리가 필요한 함수들은 SYSENTER에 의해 Ring0로 변환된 후, 처리된다고 하였습니다. 이때 SYSENTER는 MSR중, IA32\_SYSENTER\_EIP를 참조하여 실행할 함수를 결정하는데, 이 IA32\_SYSENTER\_EIP의 값을 WRMSR이라는 명령어를 통해 덮어쓸 수 있습니다. 만약 덮어씀으로 써, 새롭게 연결된 함수에서 KiFastCallEntry()의 기능을 수행할 수 있지만, ETHREAD의 ServiceTable을 참조하여, 그 ServiceTable을 사용하는 것이 아니라, 내부적인 ServiceTable을 사용한다면, SSDT Hooking을 전역적으로 무력화 시킬 수 있을 것입니다. 그럼 핵심 코드를 보도록 하겠습니다.

```
VOID InitializeEngine()
{
    UNICODE_STRING y;
    IDTINFO idt_info;
    IDTENTRY* idt_entries;
    unsigned int i;

    DbgPrint("KiFastCallEntry Imitation coded by Dual");

    DbgPrint("ProcessorCount : 0x%X",KeNumberProcessors);
    //프로세서의 갯수를 구해온다.

    gDPCP1 = ExAllocatePool(NonPagedPool,sizeof(KDPC));
    KeInitializeDpc(gDPCP1,WRMSRDPC,NULL);

    gDPCP2 = ExAllocatePool(NonPagedPool,sizeof(KDPC));
    KeInitializeDpc(gDPCP2,RestoreMSR,NULL);
    //DPC를 위한 메모리들을 할당해 둔다.

    GetProcessNameOffset();
    //EPROCESS에서 Name까지의 Offset
    DbgPrint("ProcessName Offset : 0x%X",gProcessNameOffset);

    DbgPrint("Engine_KiFastCallEntry : 0x%8X",MyKiFastCallEntry);

    __asm
    {
        mov ecx, 0x176
        rdmsr
        mov d_origKiFastCallEntry, eax
    }
    //KiFastCallEntry의 주소를 구해온다.

    DbgPrint("KiFastCallEntry : 0x%8X",d_origKiFastCallEntry);
}
```

```

PsConvertToGuiThread = findAddressofPsConvertToGuiThread();
//PsConvertToGuiThread의 주소를 구해온다.
DbgPrint("PsConvertToGuiThread : 0x%8X",PsConvertToGuiThread);

_MmUserProbeAddress = *MmUserProbeAddress;
DbgPrint("MmUserProbeAddress : 0x%8X",_MmUserProbeAddress);

DbgPrint("KeServiceDescriptorTable : 0x%8X",&KeServiceDescriptorTable);
DbgPrint("KiServiceTable :
    0x%8X",KeServiceDescriptorTable.ServiceTableBase);
DbgPrint("ArgumentTable :
    0x%8X",KeServiceDescriptorTable.ParamTableBase);
DbgPrint("Service Limit : 0x%X",KeServiceDescriptorTable.NumberOfServices);

//Make New Service Table
OrgServiceTable = KeServiceDescriptorTable.ServiceTableBase;
NewServiceTable = ExAllocatePool(NonPagedPool,
    (KeServiceDescriptorTable.NumberOfServices) * 4);
memcpy(NewServiceTable,
    KeServiceDescriptorTable.ServiceTableBase,
    (KeServiceDescriptorTable.NumberOfServices) * 4);
DbgPrint("NewServiceTableBase : 0x%8X",NewServiceTable);

KeServiceDescriptorTableShadow =
(PServiceDescriptorTableShadowEntry_t)findAddressofShadowTable();
ShadowSDT = (ULONG)KeServiceDescriptorTableShadow;
KeServiceDescriptorTableShadow += 0x1;
DbgPrint("KeServiceDescriptorTableShadow : 0x%8X",
    KeServiceDescriptorTableShadow);
DbgPrint("Win32KServiceTable : 0x%8X",
    KeServiceDescriptorTableShadow->Win32kTableBase);
DbgPrint("Win32kArgumentTable : 0x%8X",
    KeServiceDescriptorTableShadow->Win32kParamTableBase);
DbgPrint("Win32kService Limit : 0x%X",
    KeServiceDescriptorTableShadow->NumberofWin32kServices);

RtlInitUnicodeString(&y,L"KeRaiseIrql");
_KeRaiseIrql = MmGetSystemRoutineAddress(&y);
DbgPrint("KeRaiseIrql : 0x%8X",_KeRaiseIrql);

RtlInitUnicodeString(&y,L"KeLowerIrql");
_KeLowerIrql = MmGetSystemRoutineAddress(&y);
DbgPrint("KeLowerIrql : 0x%8X",_KeLowerIrql);

RtlInitUnicodeString(&y,L"KiDeliverApc");
_KiDeliverApc = MmGetSystemRoutineAddress(&y);
DbgPrint("KiDeliverApc : 0x%8X",_KiDeliverApc);

RtlInitUnicodeString(&y,L"KeGetCurrentIrql");
_KeGetCurrentIrql = MmGetSystemRoutineAddress(&y);

```

```

DbgPrint("KeGetCurrentIrql : 0x%8X",_KeGetCurrentIrql);

__asm
{
    sidt idt_info
}
idt_entries = (IDTENTRY*) MAKELONG(idt_info.LowIDTbase,idt_info.HiIDTbase);
_KiTrap6 = MAKELONG(idt_entries[0x6].LowOffset,idt_entries[0x6].HiOffset);
DbgPrint("KiTrap06 : 0x%8X",_KiTrap6);
}

```

KiFastCallEntry()를 구현하는데 있어서 필요한 함수들의 주소를 구해오는 기능을 합니다.

```

Case StartHook:
    for(i = 0; i < KeNumberProcessors; i++)
    {
        KeSetTargetProcessorDpc(gDPCP1,i);
        KeInsertQueueDpc(gDPCP1,&tmp,&tmp);
    }

```

미리 구해둔 프로세서의 개수 만큼 반복하면서, 각 프로세서의 MSR에 IA32\_SYSENTER\_EIP을 새로 만든 KiFastCallEntry()의 주소로 덮어 씁니다. 이유는 멀티프로세서 환경에서 각 프로세서 마다 별도의 MSR들을 가지고 있기 때문에 정상적인 처리를 위해선 전부 처리해주어야 합니다.

```

VOID WRMSRDPC(IN PKDPC Dpc,
              IN PVOID DeferredContext,
              IN PVOID sys1,
              IN PVOID sys2)
{
    ULONG ProcessorFastCallEntry;

    DbgPrint("Processor Number : 0x%X",KeGetCurrentProcessorNumber());

    __asm
    {
        mov ecx,0x176
        mov edx,0
        mov eax,MyKiFastCallEntry
        wrmsr
    }

    __asm
    {
        mov ecx,0x176
        rdmsr
        mov ProcessorFastCallEntry,eax
    }
}

```

```

if(ProcessorFastCallEntry == (ULONG)MyKiFastCallEntry)
{
    DbgPrint("Install Success");
}
else
{
    DbgPrint("Install Faild");
}
}

VOID RestoreMSR(IN PKDPC Dpc,
                IN PVOID DeferredContext,
                IN PVOID sys1,
                IN PVOID sys2)
{
    ULONG ProcessorFastCallEntry;

    DbgPrint("Processor Number : 0x%X",KeGetCurrentProcessorNumber());

    __asm
    {
        mov ecx,0x176
        mov edx,0
        mov eax,d_origKiFastCallEntry
        wrmsr
    }

    __asm
    {
        mov ecx,0x176
        rdmsr
        mov ProcessorFastCallEntry,eax
    }
    if(ProcessorFastCallEntry == d_origKiFastCallEntry)
    {
        DbgPrint("Restore Success");
    }
    else
    {
        DbgPrint("Restore Faild");
    }
}

```

위에것은 설치 함수이고, 밑에것은 복구 함수입니다.

```

DWORD findAddressofShadowTable(void)
{
    int i;
    unsigned char *p;
    DWORD val;

```

```

p = (unsigned char *)KeAddSystemServiceTable;

for (i = 0; i < PAGE_SIZE; i++, p++)
{
    __try
    {
        val = *(unsigned int *)p;
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        return 0;
    }

    if (MmIsAddressValid((PVOID)val))
    {
        if (memcmp((PVOID)val, &KeServiceDescriptorTable, 16) == 0)
        {
            if((PVOID)val != &KeServiceDescriptorTable)
                return val;
        }
    }
}
return 0;
}

```

```

DWORD findAddressofPsConvertToGuiThread(void)
{
    int i;
    unsigned char *p;
    ULONG val;
    static char Adr[4];
    char CodePattern[3] = {0x52,0x53,0xE8};
    char PsCovertPattern[3] = {0x6A,0x38,0x68};

    p = (unsigned char *)d_origKiFastCallEntry - PAGE_SIZE;
    for(i = 0; i < PAGE_SIZE*2; i++, p++)
    {
        if(!memcmp(p,CodePattern,3))
        {
            val = *(ULONG *)(p + 3) + (ULONG)p + 7;
            if(MmIsAddressValid((PVOID)val))
            {
                if(!memcmp((PVOID)val,PsCovertPattern,3))
                {
                    return val;
                }
            }
            else

```

```

    {
        return 0;
    }
}
}
return 0;
}
}

```

위에 것은 ShadowTable의 주소를 구해오는 함수이고, 밑에 것은 PsConvertToGuiThread의 주소를 구해오는 함수입니다. KeServiceDescriptorTableShadow 와 PsConvertToGuiThread의 경우 export되지 않기 때문에, 제가 아는 한 예서는 위와 같은 방법으로 구할 수 밖에 없었습니다. 더 좋은 방법을 알고 계시다면 저에게 연락해 주세요. :)

```

_declspec(naked) MyKiFastCallEntry()
{
    __asm
    {
        //세그먼트 선택터 값 지정
        mov ecx,0x23

        //FS를 PCR로 지정
        push 0x30
        pop fs

        mov ds,cx
        mov es,cx

        //현 스택을 커널 스택으로 변경
        mov ecx,dword ptr fs:0x40 //KPCR_TSS
        mov esp, ss:[ecx+0x4]

        //가짜 INT 스택을 만든다.
        push 0x23 //KGDT_R3_DATA + RPL_MASK(0x3)
        push edx //Ring3 ESP
        pushfd //Ring3 EFLAGS
        push 2 //Ring 0 EFLAGS
        add edx,8 //Skip user param
        popfd //Set EFLAGS

        or byte ptr [esp+0x1],0x2 //가짜 INT로 IRQ재활성화

        push 0x1B //RGDT_R3_CODE + RPL_MASK
        push KiFastSystemCallRet //sysenter리턴어드레스
        push 0
        push ebp
        push ebx
        push esi
    }
}

```

```

push edi

//우리의 PCR로 포인터 저장
mov ebx,dword ptr fs:0x1C

push 0x38 + 0x3    //KGDT_R3_TEB + RPL_MASK

//현재 스레드의 포인터 구함
mov esi,[ebx+0x124]

//예외 핸들러 체인 종료자 지정
push dword ptr[ebx]
mov dword ptr[ebx],-1

//스레드의 스택 사용
mov ebp,[esi+0x18] //KTHREAD_INITIAL_STACK

push 0x1    //Usermode
//push dword ptr[esi+0x140] //KTHREAD_PREVIOUS_MODE

//다른 레지스터들을 넘긴다.
sub esp,0x48

//mov dword ptr [esp+0x38], 0x23
//mov dword ptr [esp+0x34], 0x23

//스택에 우리의 공간을 만든다.
sub ebp,0x29C

//모드를 썬는다.
mov byte ptr[esi+0x140],0x1    //KTHREAD_PREVIOUS_MODE

//Sanity check
cmp ebp,esp
jne BadStack

//Flush DR7
and dword ptr[ebp+0x2C],0

//스레드가 디버깅 당하는 중인가?
test byte ptr[esi+0x2C],0xFF //KTHREAD_DEBUG_ACTIVE

//스레드의 트랩 프레임 설정
mov [esi+0x134],ebp //KTHREAD_TRAP_FRAME 0x134? or 0x110?

jnz Dr_FastCallDrSave

```

DEBUG\_STATUS:

```

//트랩 프레임 디버그 헤드 설정
mov ebx,dword ptr[ebp+ 0x60] //KTRAP_FRAME_EBP
mov edi,dword ptr[ebp+ 0x68] //KTRAP_FRAME_EIP

//커널 데이터 기록
mov dword ptr[ebp+ 0xC],edx
//KTRAP_FRAME_DEBUGPOINTER
mov dword ptr[ebp+ 0x8],0xBADB0D00
//KTRAP_FRAME_DEBUGARGMARK
mov dword ptr[ebp],ebx //stack 세이브
mov dword ptr[ebp+ 0x4],edi //KTRAP_FRAME_DEBUGEIP

```

```

//인터럽트 활성화
sti

```

```

/*

```

여기서 SDT를 이용하여 함수 호출

```

*/
////////////////////////////////////

```

SysCallEntry:

```

mov edi,eax
shr edi,0x8 //SERVICE_TABLE_SHIFT
and edi,0x30 //SERVICE_TABLE_MASK
mov ecx,edi

//add thread`s base system table to offset
add edi,dword ptr[esi+ 0xE0] //KTHREAD_SERVICE_TABLE

//GetSyscallID
mov ebx,eax
and eax,0xFFF//SERIVCE_NUMBER_MASK

//check syscallID
cmp eax,dword ptr[edi+ 0x8] //SERVICE_DESCRIPTOR_LIMIT
//Invalid?
jnb UnexpectedRange

//Check Win32K
cmp ecx,0x10 //SERVICE_TABLE_TEST
jnz NotWin32K

//Get TEB
mov ecx,dword ptr fs:0x18 //KPCR_TEB

//check flush?
xor ebx,ebx
or ebx,dword ptr[ecx+ 0xF70] //TEB_GDI_BATCH_COUNT

```

```

je NotWin32K //jz

//Flust it
push edx
push eax
call NtGdiFlushUserBatch
pop eax
pop edx

NotWin32K:
//inc dword ptr fs:[0x638] //KPCR_SYSTEM_CALLS
inc dword ptr fs:0x638 //KPCR_SYSTEM_CALLS

//NoCountTable:
mov esi,edx

//인자들 공간 확보
mov ebx,dword ptr [edi+ 0xC] //SERVICE_DESCRIPTOR_NUMBER
xor ecx,ecx
mov cl,byte ptr [eax+ ebx]

//Get the function point
mov edi,dword ptr [edi] //Service_Descriptor_Base

//Blocking SSDT Hooking
cmp edi,OrgServiceTable
jne Shadow

mov edi,NewServiceTable
jmp NotShadow

Shadow:
mov edi,NewShadowTable

NotShadow:
mov ebx,dword ptr [edi+ eax*4]

//우리 스택 공간 확보
sub esp,ecx

//인자 & 목적지 크기 지정
shr ecx,2
mov edi,esp

//MmUserProbeAddress인지 확인
cmp esi,_MmUserProbeAddress //0x7FFF0000
jnb AccessViolation

CopyParams:

```

```

rep movsd

//call syscall
call ebx

AfterSyscall:
mov esp,ebp

////////////////////////////////////

KeReturnFromSystemCall:
mov ecx,dword ptr fs:0x124 //KPCR_CURRENT_THREAD

//프레임 포인터 복구
mov edx,dword ptr[ebp+ 0x3C] //KTRAP_FRAME_EDX
mov dword ptr[ecx+ 0x134],edx
//KTHREAD_TRAP_FRAME 0x110? 0x134?

//인터럽트 비활성화
cli

test dword ptr[ebp+ 0x70],0x20000

//KTHREAD_COMBINED_APC_DISABLE(0x70),EFLAGS_V86_MASK(0x20000)
jnz SKIP_CS_TEST

test byte ptr[ebp+ 0x6c],0x1 //KTRAP_FRAME_CS
je SKIP_SAVE_FRAME

SKIP_CS_TEST:
//현재 스레드 구함
//mov ebx,dword ptr fs:edx+ 0x124 //KPCR_CURRENT_THREAD
mov ebx,dword ptr fs:0x124

mov byte ptr [ebx+ 0x2E],0 //KTHREAD_ALERTED 0x5E? 0x2E?

cmp byte ptr[ebx+ 0x4A],0 //KTHREAD_PENDING_USER_APC
je SKIP_SAVE_FRAME

//스택 포인터를 트랩 프레임에 저장
mov ebx,ebp

mov dword ptr[ebx+ 0x44],eax //KTRAP_FRAME_EAX
mov dword ptr[ebx+ 0x50],0x3B
//KTRAP_FRAME_FS<-KGDT_R3_TEB + RPL_MASK
mov dword ptr[ebx+ 0x38],0x23
//KTRAP_FRAME_DS<-KGDT_R3_DATA + RPL_MASK
mov dword ptr[ebx+ 0x34],0x23
//KTRAP_FRAME_ES<-KGDT_R3_DATA + RPL_MASK

```

```

mov dword ptr[ebx+0x30],0x0 //KTRAP_FRAME_GS

//IRQL을 APC LEVEL로
mov ecx,1

//과거 IRQL저장
call _KeGetCurrentIrql//Hack for Hyper Thread
push eax

push offset TmpOldIrql //Temp for OldIrql
push 1 //APC_LEVEL
call _KeRaiseIrql

//인터럽트 활성화
sti

push ebx
push 0x0
push 0x1 //유저모드
call _KiDeliverApc

//과거 IRQL로 복귀
pop ecx

push ecx //:)
call _KeLowerIrql

//EAX 복구
mov eax,dword ptr[ebx+0x44] //KTRAP_FRAME_EAX

//인터럽트 비활성화
cli
jmp SKIP_CS_TEST
SKIP_SAVE_FRAME:
//mov ds,[esp+0x38]
//mov es,[esp+0x34]
mov edi,edi
mov edx,dword ptr[esp+0x4C]
//KTRAP_FRAME_EXCEPTION_LIST
mov ebx,dword ptr fs:[0x50] //KTRAP_FRAME_FS
mov dword ptr fs:[0],edx
mov ecx,dword ptr[esp+0x48]
//KTRAP_FRAME_PREVIOUS_MODE
mov esi,dword ptr fs:[0x124]//KPCR_CURRENT_THREAD
mov byte ptr[esi+0x140],cl //KTHREAD_PREVIOUS_MODE

test ebx,0xFF //MAXIMUM_IDTVECTOR?
jnz FullIDTVector

```

```

CHECK_FOR_V86:
    //Check for V86
    test dword ptr[esp+ 0x70],0x20000
    //KTRAP_FRAME_EFLAGS,EFLAGS_V86_MASK
    jnz V86_EXIT

    test word ptr[esp+ 0x6C],0xFFF8
    //KTRAP_FRAME_CS,FRAME_EDITED
    jz RestoreCS

    cmp word ptr[esp+ 0x6C],0x1B    //KGDT_R3_CODE + RPL_MASK
    bt word ptr[esp+ 0x6C],0
    cmc
    ja RestoreEAX

    cmp dword ptr[ebp+ 0x6C],0x8
    //KTRAP_FRAME_CS,KGDT_R0_CODE
    jz SkipDebugInformation

RestoreFS:    //Restore FS
    lea esp,dword ptr[ebp+ 0x50]//KTRAP_FRAME_FS
    pop fs

SkipDebugInformation:
    lea esp,dword ptr[ebp+ 0x54]//KTRAP_FRAME EDI
    pop edi
    pop esi
    pop ebx
    pop ebp

    //ABIOS를 위한 체크?
    cmp word ptr[esp+ 0x8],0x80
    ja ABIOSExit

    //Pop Error Code
    add esp,4

    //Previous CS from usermode?
    test dword ptr[esp+ 4],1
    jnz FastExit

    pop edx
    pop ecx
    popfd
    jmp edx

IntReturn:
    iretd

```

FastExit:

```
test byte ptr[esp+ 0x9],0x1
jnz IntReturn

//클린업 스택
pop edx
add esp,4
and byte ptr[esp+ 1],0xFD
popfd
pop ecx
jmp     short $+ 3
mov     eax,90350FFBh    //sti sysexit
iretd
```

BadStack:

```
mov ecx,fs:0x40    //KPCR_TSS
mov esp,dword ptr[ecx+ 0x4]//KTSS_ESP0

//V86 stack
push 0
push 0
push 0
push 0

push 0x23    //KGDT_R3_DATA + RPL_MASK
push 0
push 0x20202
push 0x1B    //KGDT_R3_CODE + RPL_MASK
push 0

jmp _KiTrap6 //KiTrap06
```

Dr\_FastCallDrSave:

```
test dword ptr[ebp+ 0x70],20000
///KTRAP_FRAME_EFLAGS,EFLAGS_V86_MASK
jnz SKIP_CS_TEST3

test dword ptr[ebp+ 0x6c],1 ///KTRAP_FRAME_CS
je DEBUG_STATUS
```

SKIP\_CS\_TEST3:

```
mov ebx,DR0
mov ecx,DR1
mov edi,DR2
mov dword ptr[ebp+ 0x18],ebx
mov dword ptr[ebp+ 0x1C],ecx
mov dword ptr[ebp+ 0x20],edi

mov ebx,DR3
mov ecx,DR6
```

```
mov edi,DR7
mov dword ptr[ebp+ 0x24],ebx
mov dword ptr[ebp+ 0x28],ecx
xor ebx,ebx
mov dword ptr[ebp+ 0x2C],edi

mov DR7,ebx
mov edi,dword ptr fs:0x20
mov ebx,dword ptr[edi+ 0x2F8]
mov ecx,dword ptr[edi+ 0x2FC]
mov DR0,ebx
mov DR1,ecx

mov ebx,dword ptr[edi+ 0x300]
mov ecx,dword ptr[edi+ 0x304]
mov DR2,ebx
mov DR3,ecx

mov ebx,dword ptr[edi+ 0x308]
mov ecx,dword ptr[edi+ 0x30C]
mov DR6,ebx
mov DR7,ecx
jmp DEBUG_STATUS
```

UnexpectedRange:

```
cmp ecx,0x10 //SERVICE_TABLE_TEST
jnz InvalidCall

//Setup Win32K Table
push edx
push ebx
call PsConvertToGuiThread

//리턴코드 체크
or eax,eax

//레지스터 리스토어
pop eax
pop edx

//트랩 프레임 리셋
mov ebp,esp
mov [esi+ 0x134],ebp //KTHREAD_TRAP_FRAME

//다시 콜
jz SysCallEntry

//테이블 limit과 베이스 구함
lea edx, KeServiceDescriptorTable + 0x10
```

```
mov ecx,dword ptr[edx+ 0x8] //SERVICE_DESCRIPTOR_LIMIT
mov edx,dword ptr[edx] //SERVICE_DESCRIPTOR_BASE
```

```
//테이블 주소 와 인덱스값 더한 주소 구함
lea edx,dword ptr[edx+ ecx*4]
and eax,0xFFF//SERVICE_NUMBER_MASK
add edx,edx
```

```
//리턴값 구함
movsx eax,byte ptr[edx]
or eax,eax
```

```
jle KeReturnFromSystemCall
```

```
//잘못된 서비스로 리턴값 넘김
mov eax, 0xC000001CL //STATUS_INVALID_SYSTEM_SERVICE
jmp KeReturnFromSystemCall
```

InvalidCall:

```
mov eax, 0xC000001CL //STATUS_INVALID_SYSTEM_SERVICE
jmp KeReturnFromSystemCall
```

AccessViolation:

```
//커널 모드로 부터의 액세스인가?
test byte ptr [ebp+ 0x6C],0x1
//KTRAP_FRAME_CS,MODE_MASK
```

```
//괜찮다면 계속 진행
je CopyParams
```

```
//실패 잘못된 인자
mov eax,0xC0000005L
jmp AfterSyscall
```

FullIDTVector:

```
test dword ptr[ebp+ 0x70],0x20000
//KTRAP_FRAME_EFLAGS,EFLAGS_V86_MASK
jnz Skip_CS_TEST2
```

```
test dword ptr[ebp+ 0x6c],0x1 //KTRAP_FRAME_CS
je CHECK_FOR_V86
```

SKIP\_CS\_TEST2:

```
xor ebx,ebx
mov esi,dword ptr[ebp+ 0x18]
mov edi,dword ptr[ebp+ 0x1C]
mov DR7,ebx
mov DR0,esi
```

```
mov ebx,dword ptr[ebp+ 0x20]
mov DR1,edi
mov DR2,ebx

mov esi,dword ptr[ebp+ 0x24]
mov edi,dword ptr[ebp+ 0x28]
mov edi,dword ptr[ebp+ 0x2c]
mov DR3,esi
mov DR6,edi
mov DR7,ebx
jmp CHECK_FOR_V86
```

RestoreEAX:

```
mov eax,[esp+ 0x44] //KTRAP_FRAME_EAX

//skip registers
add esp,0x30

//Restore segments
pop gs
pop es
pop ds
pop edx
pop ecx

//Jump back to mainline
jmp RestoreFS
```

RestoreCS:

```
mov ebx,[esp+ 0x10] //KTRAP_FRAME_TEMPCS
mov [esp+ 0x6C],ebx //KTRAP_FRAME_CS

mov ebx,[esp+ 0x14] //KTRAP_FRAME_TEMPESP
sub ebx,0xC
mov [esp+ 0x64],ebx //KTRAP_FRAME_ERROR_CODE

//Copy Interrupt Stack
mov esi,[esp+ 0x70] //KTRAP_FRAME_EFLAGS
mov [ebx+ 8],esi
mov esi,[esp+ 0x6C] //KTRAP_FRAME_CS
mov [ebx+ 4],esi
mov esi,[esp+ 0x68] //KTRAP_FRAME_EIP
mov [ebx],esi

//리턴
add esp,0x54 //KTRAP_FRAME EDI
pop edi
pop esi
```

```

        pop ebx
        pop ebp
        mov esp,[esp]
        iretd

V86_Exit:
        add esp,0x3C          //KTRAP_FRAME_EDX

        //Restore
        pop edx
        pop ecx
        pop eax

        lea esp,[ebp+ 0x54] //KTRAP_FRAME_EDI
        pop edi
        pop esi
        pop ebx
        pop ebp

        cmp word ptr [esp+ 8],0x80
        ja BIOS_Exit

SKIP_LSS:
        //skip error code
        add esp,4
        iretd

BIOS_Exit:
        cmp word ptr[esp+ 0x2],0
        je SKIP_LSS
        cmp word ptr[esp],0
        jnz SKIP_LSS
        shr dword ptr[esp],0x10
        mov word ptr[esp+ 0x2],0xF8
        lss sp,dword ptr[esp]
        movzx esp,sp
        iretd
    }
}

```

위의 코드는 Windows XP SP2의 KiFastCallEntry()를 Reverse Engineering하여 만든 것이기 때문에 다른 Windows에서는 동작하지 않을 것입니다. 코드의 내용 중, 기존의 KiFastCallEntry()와 다른 부분이 있는데, 밑의 코드입니다.

```

        //Blocking SSDT Hooking
        cmp edi,OrgServiceTable //edi가 KeServiceDescriptorTable인가?
        jne Shadow //아니라면 KeServiceDescriptorShadow로 인식

        mov edi,NewServiceTable //프로그램 내부의 새 ServiceTable을 사용

```

```

Shadow:
    jmp NotShadow
    mov edi,NewShadowTable //Shadow일 경우 WINAPI에 관한 정보가
                          //추가된 SDT사용
NotShadow:

```

위의 코드를 통해서 프로그램 내부의 ServiceTable을 사용하도록 변경 됨으로, SSDT Hooking을 전역적으로 무력화 시킬 수 있습니다. SYSENTER를 사용하지 않는 int 0x2e에 대해서도 IDT에 0x2e번을 다음 함수의 주소로 수정하여 처리하여 줄 수 있습니다.

```

_declspec(naked) MyKiSystemService()
{
    __asm
    {
        push 0
        push ebp
        push ebx
        push edi

        push fs
        mov ebx,0x30
        mov fs,bx

        push dword ptr fs:0
        mov dword ptr fs:0,-1

        mov esi,dword ptr fs:0x124
        push dword ptr[esi+ 0x140]
        sub esp,0x48
        mov ebx,dword ptr[esp+ 0x6c]
        and ebx,0x1
        mov byte ptr[esi+ 0x140],bl
        mov ebp,esp
        mov ebx,dword ptr[esi+ 0x134]
        mov dword ptr[ebp+ 0x3c],ebx
        mov dword ptr[esi+ 0x134],ebp
        cld
        mov ebx,dword ptr[ebp+ 0x60]
        mov edi,dword ptr[ebp+ 0x68]
        mov dword ptr[ebp+ 0xc],edx
        mov dword ptr[ebp+ 0x8],0xBADB0D00
        mov dword ptr[ebp],ebx
        mov dword ptr[ebp+ 0x4],edi

        test byte ptr[esi+ 0x2c],0xFF
        jnz DebugStatus
    }
ExitService:
    sti
}

```

```
    jmp SyscallEntry

DebugStatus:
    test dword ptr [ebp+ 0x70],0x20000
    jnz SkipCheck
    test dword ptr [ebp+ 0x6c],1
    je ExitService
SkipCheck:
    mov ebx,DR0
    mov ecx,DR1
    mov edi,DR2
    mov dword ptr [ebp+ 0x18],ebx
    mov dword ptr [ebp+ 0x1c],ecx
    mov dword ptr [ebp+ 0x20],edi
    mov ebx, DR3
    mov ecx, DR6
    mov edi, DR7
    mov dword ptr [ebp+ 0x24],ebx
    mov dword ptr[ebp+ 0x28],ecx
    xor ebx,ebx
    mov dword ptr [ebp+ 0x2c],edi
    mov DR7,ebx
    mov edi,dword ptr fs:[0x20]
    mov ebx,dword ptr[edi+ 0x2F8]
    mov ecx,dword ptr[edi+ 0x2FC]
    mov DR0,ebx
    mov DR1,ecx
    mov ecx,dword ptr[edi+ 0x304]
    mov DR2,ebx
    mov DR3,ecx
    mov ebx,dword ptr[edi+ 0x308]
    mov ecx,dword ptr[edi+ 0x30C]
    mov DR6,ebx
    mov DR7,ecx
    jmp ExitService
}
}
```

## 정리

실행 순서를 다음과 같이 정리하여 볼 수 있습니다.

- 1) 구현에 필요한 함수들의 주소를 구해온다.
- 2) 각 프로세서의 MSR에 0x176번을 새로 만든 함수로 연결하여 준다.
- 3) 사용자가 기능중지를 원할 경우 MSR의 0x176번을 기존의 KiFastCallEntry()주소로 복구 시킨다.

## 장점

시스템 전역적으로 SSDT Hooking을 무력화 시킬 수 있으며, 프로그램 내부적으로는 Hooking을 사용할 수 있으므로, 경쟁상태에서 우위를 차지할 수 있습니다.

## 단점

0x176번이 다른 프로세스에 의해 덮어 쓰여질 경우, 재 기능을 발휘 할 수 없습니다.

## 4. 작동여부 테스트

실제 위에서 다루었던 내용들이 작동하는지에 대한 테스트를 하겠습니다.  
시나리오는 간단합니다. SSDT Hooking을 사용하여 notepad.exe를 숨긴 후,  
3번에서 다루었던 무력화 방법들을 사용하여 어떤 작용을 하는지 보겠습니다.

후킹하는 함수들은 다음과 같습니다..

**ZwOpenProcess()**

**ZwWriteVirtualMemory()**

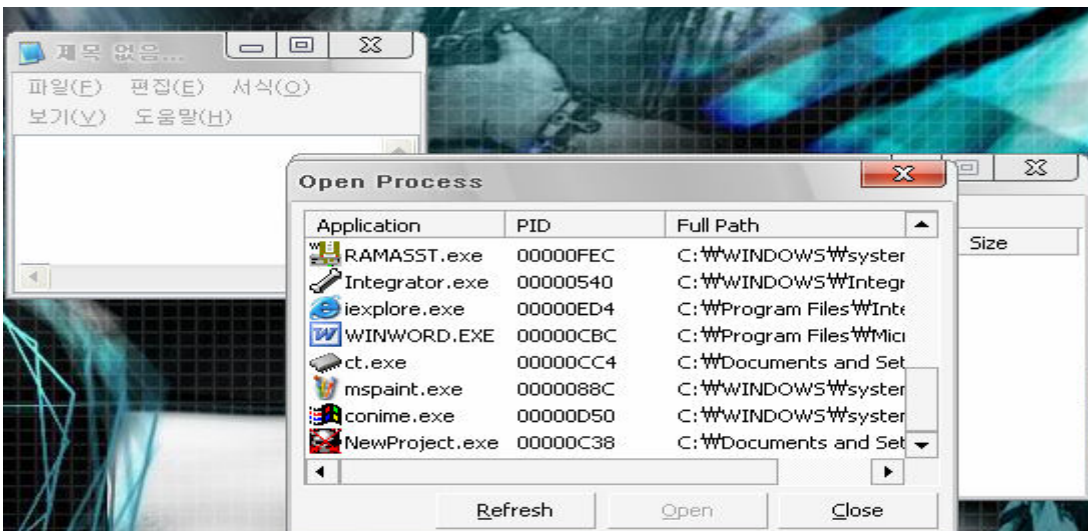
**ZwReadVirtualMemory()**

**ZwQuerySystemInformation()**

### 4-1. SDT-RESTORE



Hooking을 하기 전에는 프로세스 목록에서 notepad.exe를 볼 수 있습니다.



Hooking을 시작하니 Process목록에서 notepad.exe가 사라졌습니다.

```
ZwEnumerateValueKey      49 --[hooked by unknown at F75F5F9A]--
ZwOpenKey                77 --[hooked by unknown at F75F598E]--
ZwOpenProcess           7A --[hooked by unknown at F79DD330]--
ZwQueryKey              A0 --[hooked by unknown at F75F6064]--
ZwQuerySystemInformation AD --[hooked by unknown at F79DD590]--
ZwQueryValueKey        B1 --[hooked by unknown at F75F5EFC]--
ZwReadVirtualMemory     BA --[hooked by unknown at F79DD4B0]--
ZwSetValueKey          F7 --[hooked by unknown at F75F60EC]--
ZwWriteVirtualMemory   115 --[hooked by unknown at F79DD400]--

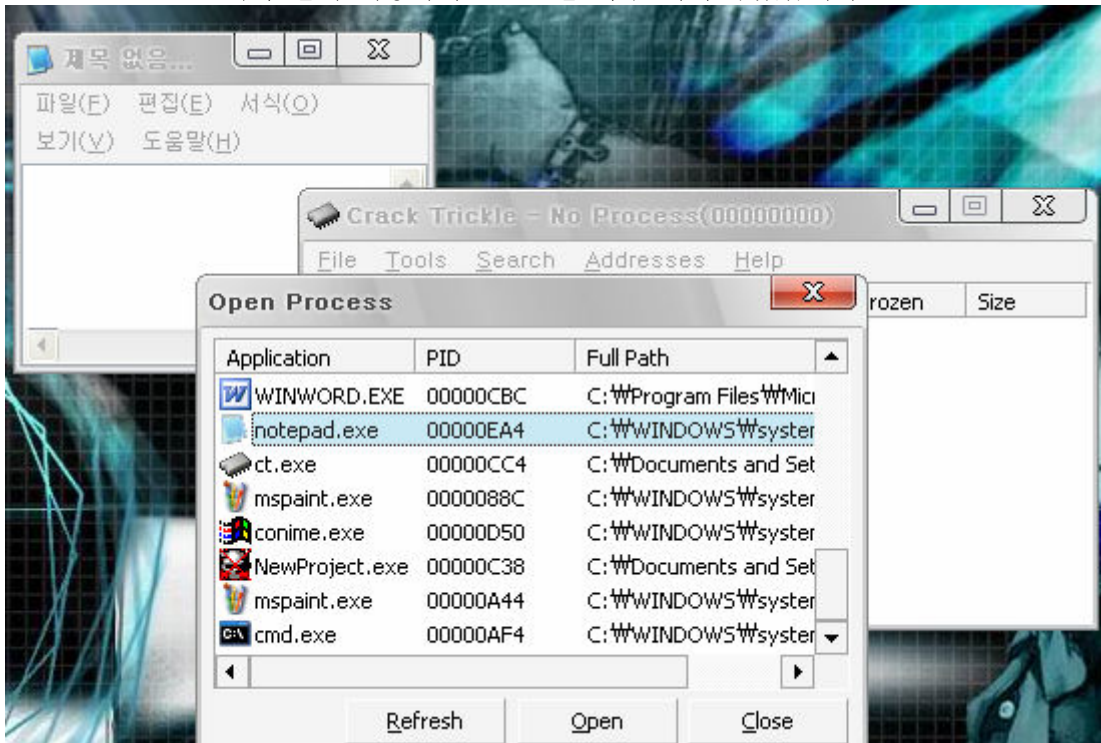
Number of Service Table entries hooked = 12

WARNING: THIS IS EXPERIMENTAL CODE.  FIXING THE SDT MAY HAVE GRAVE
CONSEQUENCES, SUCH AS SYSTEM CRASH, DATA LOSS OR SYSTEM CORRUPTION.
PROCEED AT YOUR OWN RISK.  YOU HAVE BEEN WARNED.

Fix SDT Entries <Y/N>? : y

[+] Patched SDT entry 1F to 8058B4AD
[+] Patched SDT entry 29 to 80570761
[+] Patched SDT entry 47 to 80570E68
[+] Patched SDT entry 49 to 80580B28
[+] Patched SDT entry 77 to 80569AFB
[+] Patched SDT entry 7A to 80575C96
[+] Patched SDT entry A0 to 80570B71
[+] Patched SDT entry AD to 8057E4AA
[+] Patched SDT entry B1 to 8056D0BB
[+] Patched SDT entry BA to 8058048E
[+] Patched SDT entry F7 to 80576C1D
[+] Patched SDT entry 115 to 805805E0
```

SDT-RESTORE를 위와 같이 이용하여 SDT들을 복구 시켜 주었습니다.

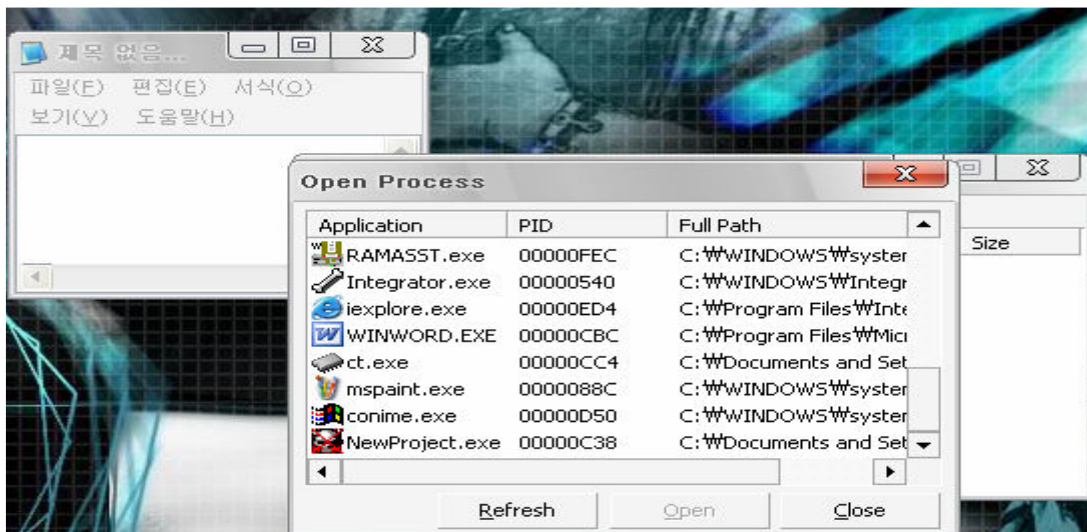


다시 notepad.exe를 볼 수 있게 되었습니다.

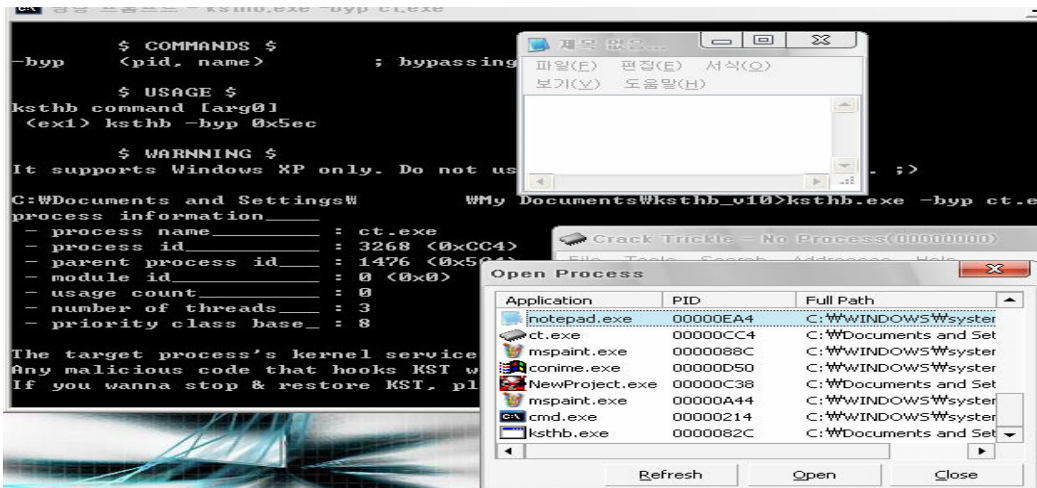
## 4-2. SDT RELOCATION



Hooking 전이라 notepad.exe를 목록에서 볼 수 있습니다.



Hooking을 시작하여서 notepad.exe가 목록에서 사라진 것을 볼 수 있습니다.



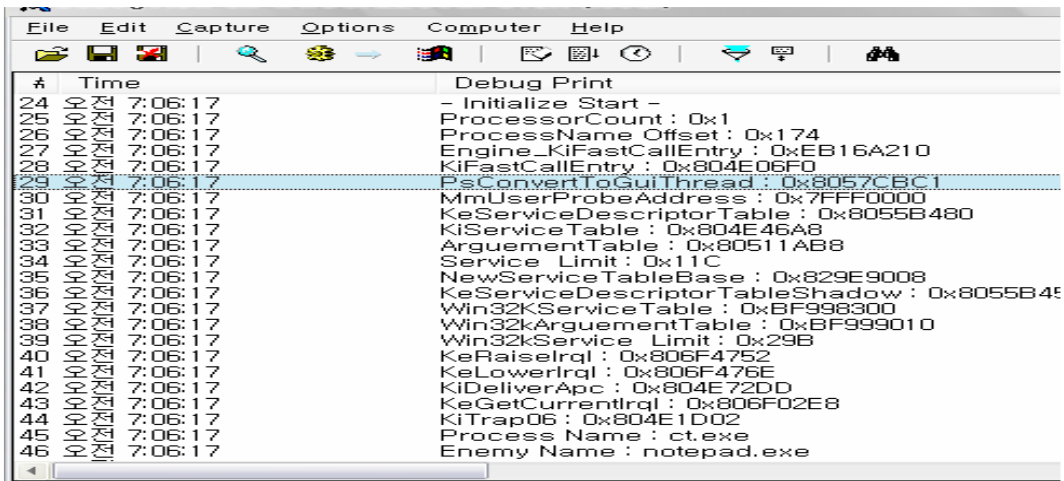
SDT RELOCATION 시키자, SSDT Hooking여부에 상관없이, notepad.exe를 프로세스 목록에서 다시 볼 수 있게 되었습니다.

### 4-3. KiFastCallEntry() Imitation

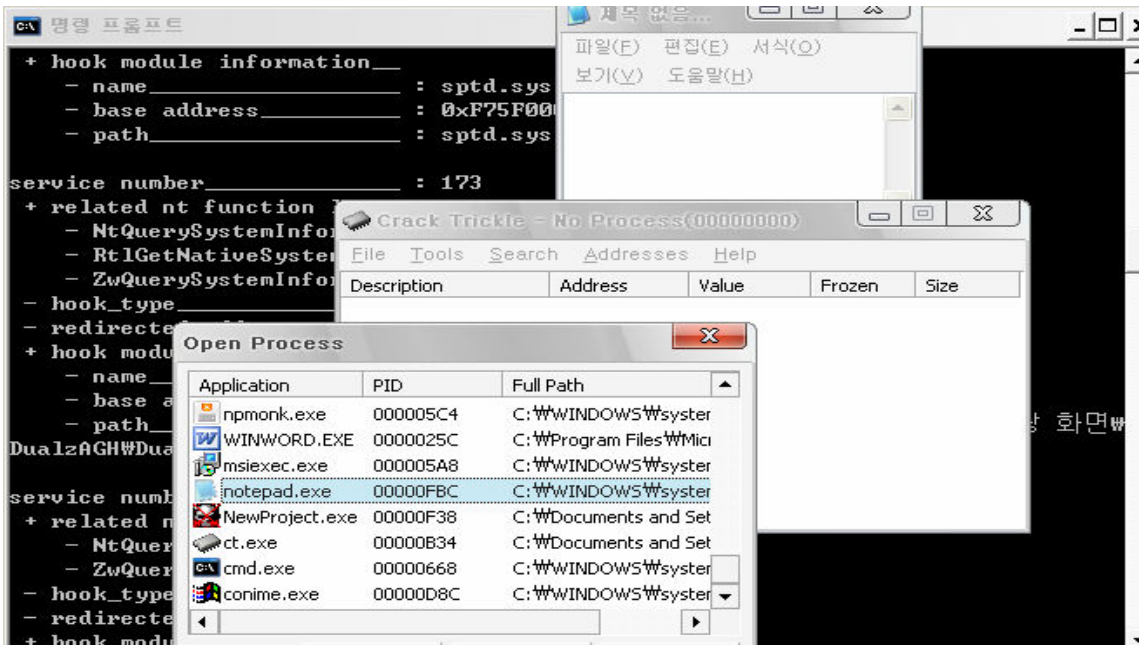
현재의 KiFastCallEntry() Imitation에는 ServiceTable을 기존 시스템의 것을 그대로 사용하고 있습니다. 그럼으로 현재의 코드에서는 KiFastCallEntry() Imitation을 Hooking하는 녀석보다 먼저 실행시켜 주어야 합니다. 만약 ServiceTable을 하드코딩을 하거나, SDT-RESTORE처럼 재 메핑하여 깨끗한 ServiceTable을 제공해 준다면 보다 강력해 질 것입니다.



Hooking 전이라 notepad.exe를 프로세스 목록에서 볼 수 있습니다.



KiFastCallEntry() Imitation을 작동시켰습니다.



Hooking중인 상태이나, notepad.exe를 프로세스 목록에서 볼 수 있습니다.

위의 테스트를 통해 SSDT Hooking 무력화가 실제로 가능한 것을 알 수 있습니다.

제가 다루었던 이 방법 외에도 훨씬 강력한 방법이 존재할 수 있습니다.

읽고 계시는 분께서 무언가 만들고자 하는 의지와 열정만 있으시다면 분명 획기적인 방법을 고안해내실 수 있으리라 믿습니다.

그럼 이만 이 글을 마치도록 하겠습니다. ©

## 5. References

### 참고문헌

- [1] Greg Hogg, James Bulter, Rootkits : Subverting the Windows Kernel
- [2] NativeAPIList : <http://jedi-apilib.sourceforge.net/native/NativeList.html>
- [3] Undocumented Functions : <http://undocumented.ntinternals.net/>
- [4] Win2k Win32k.sys SS : <http://www.fengyuan.com/article/win32ksyscall.html>
- [5] 정덕영, Windows 구조와 원리 그리고 CODES
- [6] Rootkit.com - <http://rootkit.com>
- [7] 여리의 홈페이지 - <http://zap.pe.kr>

- End -