ISR 코드 패치를 통한 IDT 후킹 무력화 방법

08. 10. 31.

지 현 석
(binish@binish.or.kr)

Header

IDT 후킹을 통해 키로깅을 수행하는 프로그램을 무력화하는 방법에 대해 소개합니다. 이 방법은 모 특허 내용을 바탕으로 제가 이해한 내용을 덧붙여 구체화 한 내용이며 키보드 ISR의 주소를 원래의 주소로 치환함으로써 방어하는 일반적인 아이디어가 아니라 ISR의 주소는 그대로 유지하되 ISR의 코드를 패치함으로써 후킹을 통한 키로거를 무력화 할 수 있다는 내용입니다.

Body

Windows XP의 경우 일반적인 키보드 벡터는 0x93(147)이며 IDT(Interrupt Descriptor Table)의 인덱스로 사용됩니다.

[표 1] IDT[0x93] information by kernelChecker

```
c:₩binish>kc.exe -ikd idt 147
IDT table information__
 - base address_______ : 0x8003F400
 - limit_____________ : 0x07FF
 - number of IDT gates_ : 256

vector number_________ : 147 (0x93)
 - type_____________ : INTERRUPT GATE
 - description_______ : Keyboard Interface
 - DPL_____________ : 0
 - segment___________ : 0x0008
 - offset____________ : 0x861A1274
 - module information__ : not found.
```

[표 1]과 같이 kc(KernelChecker by 여리, http://zap.pe.kr)를 통해 키보드 벡터의 IDT 정보를 확인하실 수 있습니다. 기타 자세한 내용은 멋쟁이 chpie의 IDT 후킹 관련 문서(http://chpie.tistory.com)를 살펴보시길 바라며 참고로 IOAPIC를 통해 키보드 벡터를 알아오는 방법이 아주 가끔 충돌나면서 윈도우가 재부팅 될 경우가 있는데 여러 테스트 결과 '메인보드'의 문제점으로 생각됩니다.
(근거는 없습니다. 몸으로 느낀 겁니다 -_-)

ISR을 교체하는 방법은 [표 2]와 같은 가상 코드(Pseudo-Code)를 통해 가능합니다.

[표 2] ISR 후킹 가상 코드

```
_asm cli
OldISR = PKInt->ServiceRoutine;
PKInt->ServiceRoutine = (ULONG)(NewHandler);
_asm sti
```

chpie 군의 idt hook 소스에서 소개된 내용으로 커널 인터럽트(PKINTERRUPT) 구조체의 서비스 루틴 주소를 바꿔치기 하는 정석적인 IDT 후킹 방법입니다.

즉!

kc를 통해 키보드 벡터의 IDT를 조사해도 인터럽트 핸들러가 후킹되었는지? 안되었는지?를 알 수 없다는 이야기입니다. 키보드 벡터의 IDT 전체를 후킹 한다면 'module information' 부분에 해당 드라이버가 표시되어 바로 탐지할 수 있지만 kc의 'offset' 주소는 바뀌지 않고 해당 주소의 구조체에 있는 서비스 루틴 주소 값이 바뀌기 때문에 바로 알지 못합니다. (암튼 저는 바로 알지 못해요..)

하지만 만약 후킹된 ISR의 주소값을 확인해보면 kc의 'offset' 주소값과 ISR의 값이 매우 상이함을 알 수 있습니다. 정상적인 경우라면 offset 값과 ISR의 값이 같은 주소 영역으로 시작되는데(메모리 특성상 연속적인 특징을 나타냄) 후킹할 경우에는 키로거의 특정 메모리 주소가 ISR로 저장되기 때문에 상이합니다. 따라서 만약 특정 포인터 변수를 통해 키보드 인터럽트가 발생하는 과정의 주소를 따라가게 한다면 갑자기 엉뚱한 주소(후킹된 ISR 주소)로 뛰어버릴 때 '아! 젠장.. 내 돈.. ' 라고 확인할 수 있을 겁니다.

이제 곰곰이 생각을 해봅시다.

정확히 키보드 벡터의 ISR 후킹을 막을 수 있는 방법은 무엇일까요?
이에 대한 나름의 질문과 답변을 정리했습니다.

대책 1. 보안 프로그램에서 키보드 벡터의 ISR 값이 계속 바뀌는지 검사한다.
Why Not? 키로거 또한 ISR 값을 더 빠르게 바꾸면 됩니다. 이럴 경우 정상적인 스캔코드 처리 또한 안될 가능성이 높아집니다.

대책 2. 보안 프로그램에서 키보드 벡터의 ISR 값을 바뀌었을 때 해당 키로거를 찾아서 프로세스를 죽인다.
Why Not? 요즘은 기술이 발달해서(?) 프로세스 Kill 혹은 Suspend 모드로의 요청이 발생하면 스스로를 다시 실행시키거나 자신의 프로세스를 숨기거나(EPROCESS라면 쉽게 찾아집니다) 혹은 접근을 금하게 합니다. (안티 키로거, e2e 메모리 해킹 방지 프로그램의 경우 접근조차 안되더군요. 샤샤샥 피해다님)

대책 3. 보안 프로그램에서 키보드 벡터 테이블을 보호해서 아예 접근을 금하게 한다.
Why Not? 효율적으로 보호하는 기술을 저자는 아직 모릅니다. (대체 뭐니?!)

대책 4. 보안 프로그램에서 후킹 시점보다 먼저 키 값을 암호화 한다.
Yes! 실제 안티 키로거 제품들의 기술적 동향을 보면 보안 채널을 생성해서 암호화 전송을 합니다. 이와 같은 기술을 특허로 등록했습니다. 한 발 늦었습니다.

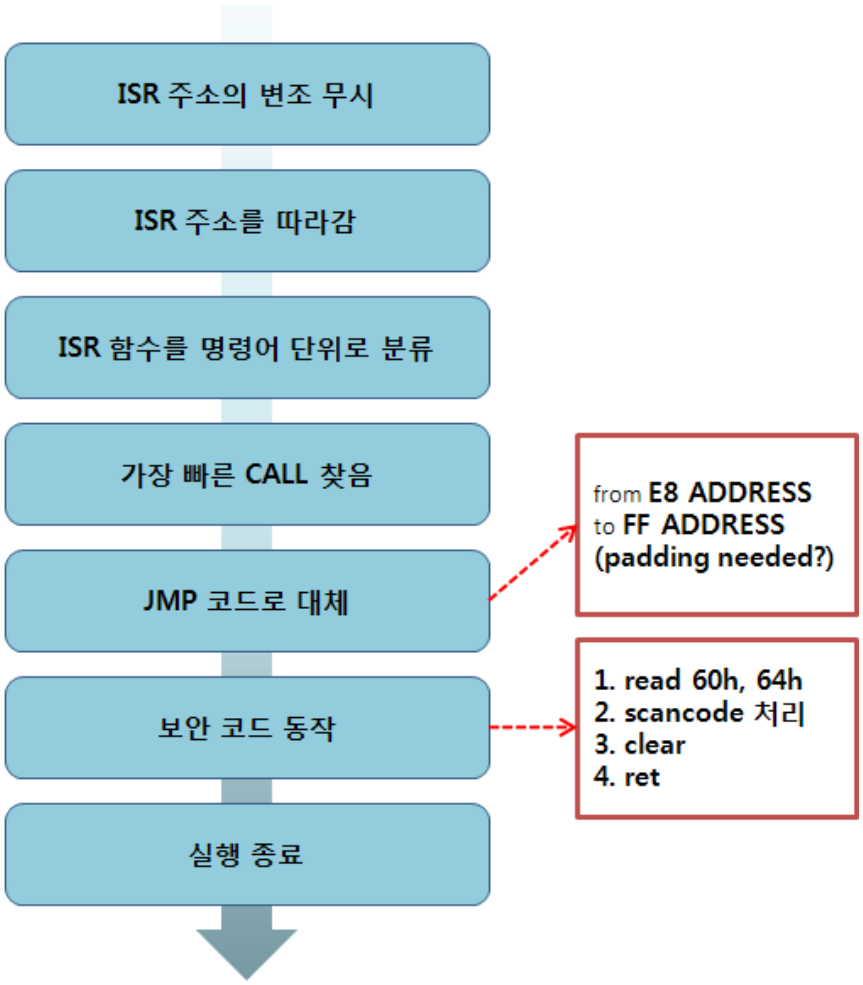
대책 5. 보안 프로그램에서 키로거의 우선순위를 낮춘다.
Why Not? 우선 순위를 낮추어도 키로거가 실행 되기 때문에 키로깅이 됩니다.
or Yes! 키로거의 우선 순위를 낮춰버리고 보다 높은 우선순위의 보안 프로그램이 스캔 코드를 먼저 처리해버리면 됩니다. 이 기법 또한 이미 알려진 걸로 알고 있습니다.

6. 기타 고려해야 할 사항
보안 프로그램이 먼저 설치되는가? 키로거가 먼저 설치되는가? 하는 선점(Preemptive)의 문제도 중요합니다. 키로거가 먼저 설치되어 쓰레드로 ISR의 변조를 검사하면서 방어를 취하면 골치 아픕니다.

그렇다면 어떤 방법이 있을까요?

본인은 초기에 ntoskrnl.exe에서 ssdt를 복구하는데에 idt 또한 복구가 가능하다는 이야기(by 굳사무엘{dual}, http://dualpage.muz.ro)를 접했습니다. 따라서 보안 프로그램에서 변조를 탐지할 경우 idt를 계속 복구시켜주면 된다는 생각을 했지만 1번의 문제의 경우에는 역시 무용지물입니다.

결국 정답은 아니겠지만 좋은 아이디어가 바로 [그림 1]과 같습니다.



[그림 1] ISR 코드 패치를 통한 IDT 후킹 키로거 무력화 과정

즉!

ISR 주소의 변조를 무시하고 무조건 ISR 주소를 따라갑니다. 그리고 메모리에 바이너리로 적재되어 있는 ISR 코드들을 명령어 단위로 분류합니다. 그렇게 분류를 하면 가장 처음에 나오는 CALL ADDRESS 형태의 5바이트가 있을 겁니다. 이 CALL ADDRESS를 보안 프로그램에서 제공하는 특정 함수의 주소로 바꿔치기 합니다. 물론 JMP 시키는 겁니다. 이렇게 되면 보안 프로그램에서 제공하는 특정 함수를 키로거가 실행하게 됩니다. 이후 ret 함으로써 정상 종료 되도록 합니다.

보안 프로그램에서 제공하는 특정 함수는 무슨 일을 하느냐?
특정 함수에서는 스캔코드 처리를 해주고 버퍼를 비워버립니다. 따라서 키로거는 아무런 값을 갖지 못하게 됩니다. 키보드 Enable을 껐다 키면 자동적으로 버퍼가 지워지게 될 것 같습니다. 그냥 제 생각입니다.

이 방법은 ISR을 원래의 주소로 복귀시키기 위한 노력에 따른 문제점들을 해결해 줍니다. 하지만 문제점(의문점)들이 있습니다.

의문 1. 키로거는 계속 설치되어 있다?
그런것 같죠? ioi)/

의문 2. 실제 구현 가능한가?
가능할 겁니다. -_-? 저는 현재 이 사항을 구현중인데 이 아이디어는 기본적으로 모 특허 내용을 바탕으로 구체화되었기 때문에 가능한 내용이라 생각됩니다. 다만 몇 가지 궁금한 사항들이 있긴 하지만 차차 풀어 나갈 수 있을 듯 합니다.

Tail

키보드 해킹 기술은 나날이 발전하면서 아직 저희가 모르는 또 다른 킹왕짱 기법이 있을 수 있습니다. 꾸준한 조사 및 학습과 더불어 해킹의 초점이 아닌 '방어'로의 초점에도 귀 기울여야 하지 않을까 생각됩니다.

가장 중요한 것은 이와 같은 기술 들을 바탕으로 자신만의 아이디어를 만들어 내는 생산적인 연구가 활발히 진행되어져야 합니다. 가장 좋은 아이디어는 가장 심플할 때 최고의 성능을 나타냅니다. 간단한 의문부터 시작해 봅시다.