인텔® 마이크로 프로세서의 영역분리 메커니즘

성윤기* 이은경* 최용준*

*한국정보보호진흥원

The Domain Separation Mechanism of the Intel® Microprocessor

Yune Gie Sung* Eun-kyoung Yi* Yong Jun Choi*

*Korea Information Security Agency

요 약

현재 공통평가기준을 이용하여 작성된 대부분의 파이어월과 VPN, 스마트 카드의 보호 프로파일의 기능요구사항에 영역분리기능이 포함되어 있다. 공통평가기준의 영역분리기능은 미국 국방성 운영체제 평가 기준인 TCSEC이 요구하는 보증수단에서 유래하였다.

8086프로세서는 리얼 모드라는 세그멘테이션 메커니즘을 처음으로 이용하여 향상된 메모리 주소관리를 제공하고 있으며, 80x86은 리얼 모드이외에 보호모드를 제공하여 시스템 영역과 응용프로그램영역을 분리시킬 수 있는 메커니즘을 제공한다. 인텔 80x86 프로세서의 구조적인 발전을 이용하여 구현된 Trusted OS는 링 상태(ring state)라고 알려져 있는 영역 분리기능을 제공하여 시스템의 영역을 응용 프로그램 영역으로부터 보호하는 메커니즘을 구현하고 보증하고 있다.

본 논문에서는 인텔 마이크로 프로세서 8086과 80x86의 구조와 메모리 관리방법을 고찰하여 8086과 비교한 80x86의 발전된 보호모드 메커니즘을 연구하여 시스템영역을 보호할 수 있는 영역분리 메커니즘을 연구하였다.

I. 서론

윈도 NT와 Trusted Solaris와 같은 대부분의 신뢰받는 운영체제는 미국의 TCSEC이나 유럽의 ITSEC 평가기준에 의해 평가를 받은 운영체제이다. 특히 미국의 TCSEC기준은 처음 운영체제의 커널을 평가하기 위해 만들어진 기준이며, 임의적 접근통제, 강제적 접근통제, 식별과 인증, 감사기능 등 보안기능에 대한 요구수준과 보증문서의 수준에 따라 C1등급에서 A1등급까지의 보안등급을 정의한 기준이다[1].

TCSEC은 커널의 안정성과 보안성의 보증을 위해 C1등급에 A1등급까지 모든 등급에서 영역분리(domain separation)를 요구하고 있다. 그리고 이것은 공통평가기준(Common Criteria)의 보안기능의 보호 클래스의 영역분리(FPT_SEP)컴포넌트로 계승되어 존재하고 있다[2].

현재까지 개발된 대부분의 파이어월 시스템, VPN(Virtual Private Network), 스마트 카드의 보호 프로파일을 살펴보면, 자체의 보안기능을 외부의 방해나 변경시도로부터 보호하기 위해 공통평가기준의 기능요구사항인 영역분리기능(FPT_SEP)을 포함하고 있다. 하지만 국내에서는 그동안 영역분리에 대한 지식과 경험이 부족하여 앞으로 공통평가기준 기반 평가수행 시 영역분리에 대한 보증 시 어려움이 예상된다.

시스템 보호 메커니즘이 구현된 신뢰받는 운영체제는 프로세서 아키텍처의 특성에 의존적이기 때문에 프로세서의 구조와 특성을 파악하는 것이 운영체제의 보안 메커니즘을 이해하는 초석이 된다. 그래서 본 논문에서는 인텔 프로세서 8086과 80x86의 메모리 관리 방법을 알아봄으로써 이에 대한 안전한 운영체제의 시스템보호 메커니즘을 파악하고자 한다. 이를 이용하여 임베디드 시스템(Embedded System)이나 전용시스템을 구현할 경우 시스템 보호를 메커니즘으로 구현할 수 있다.

II. 8086 프로세서의 구조

8086은 16비트 프로세서로서 크게 EU(Execution Unit)와 BIU(Bus Interface Unit)와 같이 2개의 모듈로 구성되어 있다. BIU는 EU와 외부사이의 데이터 전송부분을 위해 메모리와 I/O 주소관리를 포함하여 하드웨어 기능을 제공하고 있다.

EU에서는 주로 연산에 사용되는 AX(연산 레지스터), BX(베이스 레지스터), CX(카운터 레지스터), DX(데이터 레지스터)와 주로 메모리의 번지 오프셋 값으로 사용되는 레지스터인 SP(스택 포인터), BP(베이스 포인터), SI(소스 인덱스), DI(데스티네이션 인덱스) 등으로 구성된다.

EU는 BIU로부터 프로그램 명령 코드와 데이터를 수신하여 명령을 수행한다. 명령을 수행한 후 결과 값은 레지스트리에 저장한다. 수행한 결과를 BIU에서 다시 보내어 메모리나 I/O 장치에 데이터가 저장된다. 한가지 명심할 점은 EU는 시스템 버스와 연결점이 없고 단지 BIU를 통해서만 EU의 모든 데이터를 수신하고 송신한다.

BIU는 세그먼트 레지스터인 CS(코드 세그먼트), DS(데이터 세그먼트), ES(여분 세그먼트), SS(스택 세그먼트)와 IP(명령 포인터)레지스터로 구성된다. CS는 프로그램의 명령코드를 가지고 있으며, DS는 프로그램의 데이터를 저장한다. ES는 여분의 데이터를 저장한다. SS는 인터럽트와 서브루틴 반환주소를 저장한다.

세그먼트 레지스터는 각각 포인터 레지스터와 함께 물리적인 메모리 주소 지정을 위해 동작한다. 예를 들어 CS:IP를 이용한 20비트의 주소는 메모리에 실행할 코드의 물리적인 주소를 지정한다.

8086 프로세서는 16비트 프로세서이지만, 리얼 모드의 메모리 세그멘테이션 방법을 사용하여 20비트의 주소 관리를 할 수 있어 최대 1 MB의 메모리 공간을 가질 수 있다. 그리고 메모리는 64KB 크기의 메모리 블록으로 나누어진다.

이 부분은 앞으로 8086 프로세서의 리얼 모드에서 설명한다.

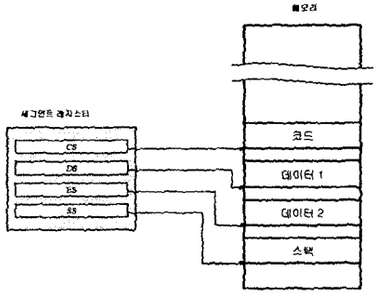


그림 1 : 세그먼트 레지스터의 베이스주소 지정

과거 8비트 프로세서인 8080과 Z-80은 단 한 개의 64KB 크기의 메모리를 사용하여 같은 세그먼트에 접근할 수 있었다. 이 말은 프로그램 코드, 데이터, 서브루빈 스택 등이 모두 같은 메모리를 공유하여 사용했다. 이것으로 인해 저장공간이 부족하였을 뿐만 아니라, 스택이 데이터다 프로그램 영역에 덮어쓰는 경우에는 시스템이 다운되기도 한다.

세그먼트 레지스터는 그림 1과 같이 각 세그먼트 영역의 메모리의 베이스 주소를 지정한다. 8086에서 기억해야 할 점은 세그먼트의 시작 주소는 임의적으로 지정할 수 없다. 주소는 16의 배수로 시작해야 한다. 또 하나는 네 개의 세그먼트는 반드시 분리될 필요는 없으며 완전히 겹칠 수도 있다(즉 CS = DS = ES = SS). 그리고 2개의 세그먼트가 겹칠 수도 있으며 겹칠 경우 2개의 논리적인 주소가 같은 물리적인 주소를 지정하게 된다. 이것으로 인해 데이터가 서브루틴 스택 영역에 데이터가 쓰이지게 되면 참담한 결과를 가져다 줄 수 있다.

8086 리얼 모드

세그먼트 구조에서 주 메모리에 주소를 지정하는 방법이 8080과 비교해서 약간 복잡하다. 8086의 모든 레지스트리는 16비트이다. 그래서 모든 프로그램은 최고 64K바이트의 메모리 크기 초과할 수 없다. 하지만 8086에서는 독특한 방법으로 20비트의 메모리 주소를 지정할 수 있도록 구현하여 최대 1MB 크기의 메모리 주소를 관리할 수 있다.

논리적인 주소관리 방법을 도입하여, 16비트의 레지스터 2개를 이용하여 20비트의 주소를 만들어낸다. 세그먼트 레지스터의 값에 16을 곱한 값을 베이스 주소로 한 후, 다른 레지스터 값을 오프셋 값으로 하여 두 값을 합하여 20비트의 물리적인 주소를 지정하는 방식이다.

그러므로 세그먼트 베이스 주소는 16의 배수로 사용되어야 한다. 왜냐하면 세그먼트 레지스트리는 16비트인데 이것을 메모리의 베이스 주소로 사용할 경우 16을 곱한 후 그 주소를 세그먼트의 베이스 주소로 설정하기 때문이다.

예를 들어 이러한 방법을 이용하여 80100H라는 메모리 주소(물리적인 주소)에서 어떤 프로그램을 실행시키고자 한다면, 일단 CS = 8000H 값을 저장시키면 베이스 주소는 80000H가 된다. 여기에 IP에 0100H값을 넣고서 2개의 값을 더하면 80100H의 실제 주소를 가리키게 된다. 그림 2는 리얼 모드의 메모리 주소 지정방식을 설명한 그림이다.

```
CS = 8000H × 16 = 80000H      → 베이스주소
IP = 0100H                    → 오프셋 값
베이스주소 + 오프셋 = 80100H   → 물리적인 주소
```

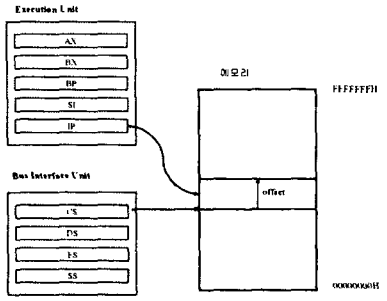


그림 2 : 리얼모드의 주소 지정 방식

모든 8086 레지스트리는 16비트 크기이므로 각 세그먼트들의 크기는 2^{16} = 64K바이트가 가능하다. 다른 데이터 세그먼트도 똑 같은 방식으로 동작한다.

이것이 8086 프로세서의 동작이며, 80x86 프로세서가 리셋이나 파워를 넣은 후 동작을 개시하는 모드이다. 이와 같은 것을 리얼 모드(Real Mode)라 한다[3].

요약하자면, 8086 프로세서는 앞으로 80x86에서 지원하게 될 메모리간의 보호 메커니즘은 지원하지 않는다. 이것을 기반으로 구현된 마이크로소프트사의 DOS와 같은 운영체제는 리얼모드만 지원하므로 시스템 보호를 위한 어떠한 보안성도 제공하지 않는다는 것을 의미한다.

III. 80x86의 구조

8086 프로세서는 전술(前述) 하였듯이 EU와 BIU의 두 가지 모듈로 구성되어 있는 반면 80x86에서는 독립적으로 동작하는 4가지 모듈(버스 유니트, 어드레스 유니트, 명령 유니트, 실행 유니트)로 구성되어 있다. 그리고 8086은 리얼 모드만 지원하는 반면에, 80x86에서는 리얼 모드이외에 보호모드(Protected mode)[1]라는 특징이 추가되었다. 이것으로 인해 80x86은 훨씬 강력해졌으며 현재 대부분의 Trusted OS에 구현되어 있는 링 상태를 통한 **영역 분리, 프로세스간 메모리 보호**와 가상 메모리에 대한 기능을 제공하게 되었다.

그럼 여기서 8086과 비교하여 80x86의 구조를 먼저 살펴보자.

80x86의 레지스터 구조

8086이 16비트 프로세서인 것에 반해 80386은 32비트 프로세서이다. 이 말은 레지스터와 데이터 버스가 32비트로 구성되어 있다는 것을 의미한다. 80x86프로세서의 외형적인 발전사항을 살펴보면, 세그먼트 레지스터의 크기는 모두 16비트로 변경이 없지만 80x86에서는 기존의 CS, DS, ES SS 이외에 추가데이터용 레지스터인 FS와 GS가 추가되었다.[2] 범용 레지스터인 AX, BX, CX, DX는 모두 32비트가 되면서 EAX, EBX, ECX, EDX로 이름이 각각 변경되었다. 그리고 SP, BP, DI, SI도 32비트 크기의 ESP, EBP, EDI, ESI로 변경되었다.

전체적인 레지스터의 구조는 크게 변경된 것은 없지만, 8086 프로세서는 세그먼트 레지스터 값이 직접 메모리의 베이스 주소를 가리키는 것과는 다르게 80x86에서는 디스크립터 테이블(Descriptor table)을 이용한 베이스 주소 지정방식으로 변경된 것과 프로세서에서 메모리를 시스템 영역과 응용프로그램 영역으로 분리하여 시스템 영역이 외부의 악의적인 데이터로 인한 변경으로부터 보호하는 매커니즘이 추가되었다.

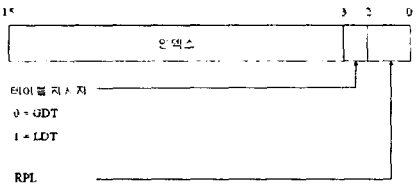


그림 3 : 80x86 프로세서의 셀렉터 구조

가상메모리 보호모드

과거의 8086에서는 리얼 모드만이 제공하였지만 80x86부터는 리얼 모드이외에 가상메모리 보호모드(Virtual memory protection mode), 즉 보호모드(Protected mode)가 추가되었다.

보호모드를 이용하여 메모리 세그먼트를 관리하면 확장된 메모리를 관리해줄 수 있을 뿐만 아니라, 8086에서 지원하지 않았던 영역분리기능을 구현할 수 있다.

다음은 보호모드를 이용한 메모리 주소 지정방법에 대한 설명이다.

주소지정방식

80x86에서는 각 세그먼트 레지스터는 셀렉터(Selector)라는 이름으로 바뀌었으며 레지스터 구성은 그림 3과 같이 디스크립터 테이블을 선택하

1) 80286에서 보호모드가 처음으로 나왔지만, 여기서 설명하는 80x86 프로세서는 80386이상의 프로세서에 대해서 설명한다.

2) FS와 GS는 80386이상의 프로세서에서만 추가된 레지스터이다.

기 위해 테이블 지시자(Table Indicator)비트와 메모리의 영역을 보호하기 위해서 사용되는 RPL(Requested Privilege Level)필드, 그리고 디스크립터 테이블의 주소를 지시하는 인덱스 필드로 구성되어 있다. 셀렉터의 인덱스는 디스크립터 테이블로 가는 포인터 역할을 한다. 세그먼트 레지스터의 인덱스 값은 해당 디스크립터 테이블은 메모리의 주소를 가리키고, 지정된 디스크립터 테이블에 저장된 주소가 메모리의 베이스 주소가 되고 해당 레지스터의 오프셋 값을 이용하여 메모리의 물리적인 주소를 가리키게 된다.

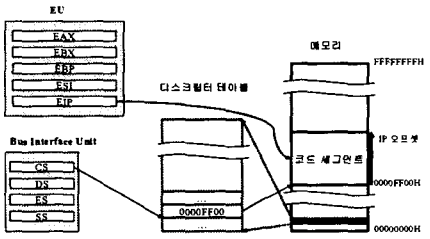


그림 4 : 80x86의 메모리 관리 방법

즉 그림 4와 같이 세그먼트는 먼저 TI(Table Indicator)값을 확인한 후, GDT인지 LDT인지 판단 한 후 인덱스 필드에 들어 있는 해당 디스크립터 테이블 주소를 지정하면 해당 디스크립터 테이블은 메모리 세그먼트의 베이스 주소를 지정하고 다시 레지스터에 저장된 해당 오프셋 값을 이용하여(베이스 주소 + 오프셋 값) 메모리의 세그먼트 영역에 접근할 수 있다. 즉 과거 리얼 모드에서는 세그먼트 레지스터와 오프셋 레지스터 값을 이용한 직접적인 메모리 관리가 보호모드에서는 그 중간에 디스크립터 테이블이 만들어 져서 간접적으로 메모리 주소를 지정하게 되었다. 하지만 이러한 개념을 이용하여 과거 16비트 8086 프로세서에서는 지원할 수 없었던 메모리 보호기능을 만들 수 있게 되었다. 이것은 디스크립터 테이블을 이용하여, 시스템 영역과 응용프로그램 영역을 분리하고 중간에서 통제하는 메커니즘이 추가되었다.

영역분리 메커니즘

디스크립터 테이블은 시스템 사용영역을 지정하는 GDT (Global Descriptor Table)와 사용자 영역을 지정하는 LDT (Local Descriptor Table)와 인터럽트 시에 사용되는 IDT (Interrupt Descriptor Table)의 3종류가 있다. 그리고 동시에 2개의 테이블만 활성화 될 수 있다. 이러한 테이블은 주 메모리에 저장되어 있으며 GDT는 보통 커널 영역을 위해서 한 개가 존재하며, LDT는 각 프로세스별로 하나씩 만들어진다.

디스크립터 테이블은 메모리 주소 지정을 위한 해당 세그먼트의 메모리 베이스 주소뿐만 아니라 추가적인 정보를 가지고 있다. 추가정보의 대표적인 것이 그림 5와 같이 80x86 보호메커니즘을 위한 접근 권한 필드와 세그먼트 한계 필드이다. 그리고 접근권한 필드에 있는 2비트의 DPL은 0 - 3 까지 숫자를 가질 수 있으며 이것이 바로 현재 Trusted OS에서 사용되는 운영체제의 링 상태(ring state)를 나타낸다. 즉 DPL은 권한 등급에 따라 세그먼트마다 할당되어져 있다.

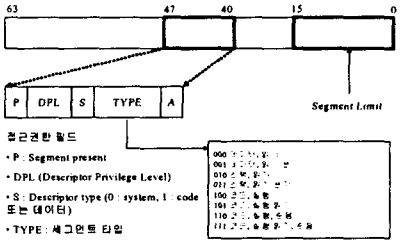


그림 5 : 디스크립터 테이블의 구조

먼저 각 세그먼트 레지스터(Selector)는 GDT인지 LDT인지를 판별하는 TI(Table Indicator)에 의해서 GDT 또는 LDT를 선택한다. 이들 테이블의 시작 주소는 GDTR(GDT Register)과 LDTR(LDT Register)이라는 특별한 레지스터가 가리키고 있다.

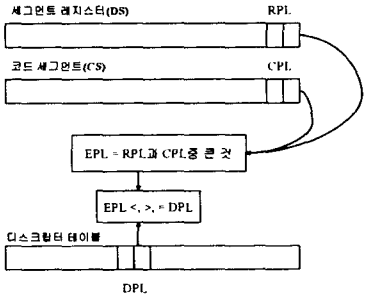


그림 6 : 80x86 프로세서의 권한 등급 결정 방법

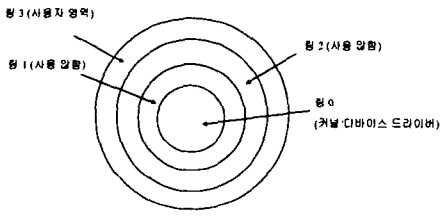
프로세서는 세그먼트에게 접근권한을 주기 전에 셀렉터의 RPL과 접근대상 세그먼트의 DPL을 비교한다. CPL(Current Privilege Level)은 여기서 큰 역할을 담당한다. CPL은 실행되고 있는 코드 세그먼트(셀렉터)의 DPL이다. 일단 CPL과 RPL을 비교하여 둘 중 큰 값이 EPL(Effective

Privilege Level)이 된다. 그런 후 다시 EPL과 DPL을 비교한 후 작은 경우에만 세그먼트가 접근할 수 있다. 운영체제는 주로 CPL = 0에서 동작하고(이것을 링 0라 한다)사용자 프로세스는 CPL = 3 (링 3)에서 동작한다. 이러한 방식을 통해서 사용자 모드는 운영체제의 메모리 영역에 접근할 수가 없다. 반면에 사용자 프로세스는 같거나 높은 등급의 서비스를 호출 할 수 있다. 이것이 보안성을 보증하는 링 기반 운영체제의 기본적인 원칙이다[4].

80x86 플랫폼으로 개발되어 TCSEC으로 평가를 받은 윈도 NT 계열의 운영체제에도 그림 7과 같이 4개의 링 상태가 있으며, 0은 가장 높은 권한 상태로 시스템 영역으로 사용되고 3은 가장 낮은 권한 상태로서 응용프로그램이 사용하는 영역이다.

권한 등급(Privilege Level)이외에 접근 타입이 함께 동작한다. S필드의 값은 1이면 시스템 세그먼트를 의미하고 0이면 응용프로그램의 세그먼트를 나타낸다. S필드와 타입필드를 이용해서 세그먼트가 읽기권한(Read only)인지 실행권한(Execute)인지를 명세한다. 읽기 권한만 있는 세그먼트에 쓰기를 시도하면 예외현상이 발생한다. 이러한 특징을 이용해서 프로그램이 자기 자신의 세그먼트 이외에 접근하고자 하는 것은 금지한다.

또한 15비트 크기의 세그먼트의 한계 필드가 있어서 세그먼트의 마지막 바이트 오프셋 값을 가지고 있어서 이것을 초과할 경우 명령이 실행되지 않는다.



[그림 7] 윈도 NT의 링 상태

IV. 결론

이상으로 인텔 마이크로 프로세서 8086과 80x86의 메모리 관리 방법을 통하여 80x86의 영역분리 메커니즘의 구현과 원리에 대해서 살펴보았다.

8086 이전의 인텔 프로세서는 보호 모드를 제공하고 있지 않으므로 이것을 이용한 소프트웨어/펌웨어 구현은 시스템의 데이터를 안전하게 보호하지 못한다. 하지만 80x86 프로세서를 이용하면 보호모드를 이용하여 시스템영역과 응용프로그램 영역을 분리할 수 있으며 시스템 영역을 외부의 방해나 변경시도로부터 좀더 안전하게 보호할 수 있다. 이들 가상 주소공간은 서로 완벽하게 분리되어 있어서, 응용프로그램이 시스템 영역으로 접근 시 시스템 호출의 단계를 거쳐야 한다. 또 하드웨어 가상 메모리 메커니즘은 메모리 영역에 쓰기를 금지 할 수 있게 한다. 이것을 이용하여 시스템 코드와 데이터가 악의적인 프로그램에 의해 덮어 씌어지는 것을 막아준다.

본 논문은 앞으로 공통평가기준의 영역분리기능(FPT_SEP)에 대한 평가 방향을 제시하고, 개발업체나 평가기관에서 전용시스템(dedicated system)을 개발 및 평가할 경우 시스템 보호를 위한 메모리의 관리방법을 제대로 구현이 되었는지 검토할 수 있는 방법론을 제공할 수 있다.

참고문헌

[1] 미국 국방성, DoD 5200.28-STD, DoD TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA, 1983

[2] CCIMB, Common Criteria for Information Technology Security Evaluation Version 2.1, 1999

[3] Les Bell, "Intel Processor Architectures", http://www.lesbell.com.au/os290.nsf/504ca249c786e20f852562840006da7ab/fcacf94ebf24fe254a2565c2000c1c24?OpenDocument, 1998

[4] Prasad Dabak, Milind Borate, Sandeep Phadke, "Memory Management", http://www.windowsitlibrary.com/Content/356/04/toc.html, 1999