함수 단위 동적 커널 업데이트를 위한 보안 정책 및 기법의 설계

(Policy and Mechanism for Safe Function-level Dynamic Kernel Update)

박 현 찬 † 유 혁 ††
(Hyunchan, Park) (Chuck Yoo)

요 약 최근 시스템의 복잡도가 증가함에 따라 보안 취약점 문제가 더욱 많이 발생하고 있다. 이를 해결하기 위해 보안 패치가 배포되고 있지만, 시스템 서비스의 중단이 필요하고 패치 자체의 안정성이 검증되지 못해 패치의 적용이 늦어지는 문제가 발생한다. 우리는 이러한 문제를 해결하기 위해 업데이트성이 없는 커널을 위한 함수 단위 동적 업데이트 시스템인 DUNK를 설계하였다. DUNK는 서비스 중단 없는 업데이트를 가능케 하고, 보안 기법인 MAFIA를 통해 안전한 업데이트를 수행한다. MAFIA는 바이너리 패치 코드의 접근 행위를 분석함으로써 패치된 함수가 기존 함수의 접근 권한을 상속받도록 하고, 이를 검증하는 기술을 제공한다. 본 논문에서는 DUNK의 설계와 MAFIA의 알고리즘 및 수행에 대해 기술한다.

키워드 : 동적 커널 업데이트 보안 기법, 접근 행위 분석 기법

Abstract In recent years, the software vulnerability becomes an important problem to the safety in operating system kernel. Many organizations endeavor to publish patches soon after discovery of vulnerability. In spite of the effort, end-system administrators hesitate to apply the patches to their system. The reasons of hesitation are the reboot disruption and the distrust of patches. To solve this problem we propose a dynamic update system for non-updatable kernel, named DUNK. The DUNK provides: 1) a dynamic update mechanism at function-level granularity to overcome the reboot disruption and 2) a safety verification mechanism to overcome the distrust problem, named MAFIA. In this paper, we describe the design of DUNK and detailed algorithm of MAFIA.

Key words : Safety mechanism for dynamic kernel update, code analysis for access behavior

1. 서 론

시스템 규모가 점점 더 커짐에 따라 시스템의 설계 및 구현에서 보다 많은 에러가 발생하고, 이는 보안 취약점(security vulnerability)의 발생으로 이어진다. 최근 이러한 보안 취약점들은 매우 커다란 문제로 인식되고 있어 미국 국립기술 표준원에서 제공하는 NVD[1]와 같은 국가적인 보안 취약점 관리 사이트도 등장하였다. 이런 데이타 베이스는 보안 취약점에 대한 보고 외에도 수정을 위한 패치(patch) 정보도 함께 제공한다. 그러나 [2]에 따르면, 보고되고 패치가 제공된 보안 취약점을 시스템 관리자들이 실제로 수정하기까지는 오랜 시간이 걸리거나, 아예 수정되지 않는 것으로 나타났다. 이 때문에 보안 취약점 문제는 시간이 지나고 패치가 나와도 여전히 해커들의 목표가 되고 있다.

[3]에서는 이렇게 패치가 지연되는 이유 중 하나로 기존 시스템에 패치를 적용하거나 이미 패치가 적용된 새로운 버전의 어플리케이션을 설치할 때, 제공하던 서비스가 중단되는 것이 큰 손해를 가져오기 때문인 것으로 파악하고 있다. 패치가 지연되는 또다른 이유는 패치의 안정성에 대한 시스템 관리자들의 불안감 때문이다. [4]에서는 보안 패치가 신속하게 제공되면서 충분한 테스트를 거치지 않아 시스템의 안정성이 오히려 기존보다 훼손될 가능성을 제시하고 있다. 실제로 마이크로소프트의 윈도우즈-XP 운영체제에 대한 보안 패치에서 이러한 문제가 발생한 사례[5]가 있다.

본 논문에서는 이러한 문제의 해결을 위해 먼저 보안 패치를 위한 커널의 함수 단위 동적 업데이트 시스템인 DUNK(Dynamic Update in Non-updatable Kernel)를 제안한다. 커널의 동적인 업데이트는 시스템이 수행 중인 상태에서도 재컴파일이나 재시작없이 커널을 수정할 수 있는 기법이다. 따라서 서비스의 중단 없이도 보안 패치를 적용할 수 있다. 동적 업데이트 기법을 커널의 보안 패치를 위해 활용하고자 할 때 특히 중요한 점은 시스템의 안정성을 유지하는 정책과 기법이 필요하다는 것이다. 우리는 DUNK의 보안 기법으로 간접 접근 경

· 이 연구에 참여한 연구자는 '2단계 BK21사업'의 지원비를 받았음
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '함수 단위 동적 커널 업데이트를 위한 보안 정책 및 기법의 설계'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 고려대학교 컴퓨터학과
hcpark@os.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
hxy@os.korea.ac.kr

논문접수 : 2008년 8월 28일
심사완료 : 2008년 10월 20일

로의 매칭(MAFIA: MAtching the Footprints of Indirect Accesses) 기법을 설계하였다. MAFIA 기법은 기존 격리 정책들이 메모리 주소 영역을 접근 가능 구역과 불가능 구역으로 나누어 관리하는 것에 반해 업데이트 코드의 접근 행위(access behavior)를 분석하여 기존 코드의 접근 행위와 동일한지 검사하는 기법이다. MAFIA를 활용하면 업데이트하려는 코드가 공유 데이타의 접근에 있어서 기존의 코드와 동일한 동작을 한다는 것을 보장할 수 있다. 따라서 업데이트에 의해 커널의 안정성이 저하되는 것을 최소화할 수 있다. 그리고 간접 접근에 대한 권한 검사를 코드가 커널에 삽입될 때 한 번만 수행하면 되므로 실행 시간에 발생하는 부가적인 검사를 제거할 수 있다. 본 논문에서는 MAFIA 기법의 상세한 알고리즘의 설계를 제시하고 예를 들어 그 동작을 설명한다.

본 논문은 다음과 같이 구성된다. 2장에서는 동적 커널 업데이트 시스템인 DUNK의 설계에 관해 기술하고, 3장에서는 DUNK에서 사용되는 보안 정책 및 기법의 동작을 설명한다. 4장에서는 관련 연구를 소개하고 5장에서는 결론과 함께 향후 연구 내용을 제시한다.

2. 동적 커널 업데이트 시스템(DUNK) 설계

2.1 설계 목표

- 서비스 중단 없는 업데이트
  DUNK의 핵심 기능은 업데이트를 서비스의 중단없이 수행하는 것이다. 특히 보안 패치의 동적인 적용을 주요 목표로 한다. 이를 위해 우리는 함수 단위의 동적 업데이트 기법을 제안한다.
- 시스템 안정성 저하 최소화를 위한 보안 정책 수립
  DUNK의 설계에서 가장 중요한 목표는 보안 패치의 업데이트로 인해 시스템의 안정성이 저하되는 것을 막는 것이다. 동시에 간단한 수정을 수행하는 보안 패치가 가능할 정도의 업데이트성을 제공해야 한다.
- 동일한 개발 환경의 제공
  DUNK는 업데이트 코드 생성을 위해 기존의 컴파일러와 기타 개발 도구를 그대로 활용할 수 있다. 따라서 기존 커널 개발 환경에 있어서 어떠한 추가나 수정도 필요로 하지 않는다.

2.2 패치 모델

DUNK에서의 패치는 함수 단위의 동적 업데이트 기법을 이용해 수행된다. 함수 단위 업데이트를 사용하면 패치 제작자는 커널과 패치의 개발을 위해 같은 환경을 사용할 수 있다. 이것은 패치의 제작을 더욱 쉽게 만들어준다. 그리고 시스템의 명령어 세트 구조에 독립적일 수 있다. 함수 단위 외에 기본 블록(basic block) 단위의 업데이트[6]가 사용될 수도 있다. 그러나 이 기법을 사용하면 패치 개발자가 어셈블리 수준에서 함수를 분석하고 패치를 제작하여야 한다. 따라서 개발환경도 변경되어야 하고 패치 개발도 어려워진다.

함수 단위 업데이트를 위한 가장 단순한 기법은 기존 코드의 첫 부분을 업데이트 코드로 분기하는 명령어로 대치하는 것이다. 따라서 분기 명령어를 단순히 덮어쓰는 것으로 업데이트된 함수가 수행되도록 할 수 있고, 해당 영역을 본래의 데이타로 복구함으로써 패치의 취소도 간단히 수행할 수 있다. 보안 패치는 특별한 기능의 추가가 없기 때문에 가장 단순한 기법으로도 구현 가능하다. 그리고 안전에 대한 분석이나 검증을 위해서는 보다 단순한 기법이 적합하다.

DUNK를 위한 보안 패치는 로드 가능한 오브젝트 코드 형태를 취해야 한다. 이를 위한 개발 과정은 전혀 제한하지 않기 때문에 기존 커널 개발을 위해 사용하던 개발 환경을 그대로 활용할 수 있다. 이는 컴파일러나 링커, make와 같은 유틸리티의 사용을 제한하거나 변경할 필요가 없다는 뜻이다. DUNK의 패치 오브젝트 코드는 재배치와 커널 심볼로의 링크를 위한 정보만을 포함한다. DUNK의 패치 로더는 커널 영역으로 전달된 업데이트 코드의 배치를 위해 커널 메모리를 할당받고, 재배치를 수행하며, 커널 심볼에 대한 링크를 설정하는 역할을 담당한다.

2.3 보안 목표

DUNK의 보안 목표는 기존 함수와 패치된 함수의 커널 전역 데이타에 대한 접근 행위(access behavior)를 분석하여 시스템 관리자가 채택한 정책에 따라 업데이트의 수행 여부를 판별할 수 있도록 하는 것이다. 여기서 접근 행위란 데이타에 대한 일련의 접근들(읽기, 쓰기, 수행)이 이루어지는 과정을 뜻한다. 접근 행위는 단순히 어떤 영역에 대한 접근만이 아닌, 여러 접근들의 순서 또한 포함한다. 이런 접근 행위를 분석하면 해당 코드가 커널에 대해 어떤 접근을 하는지 알 수 있고, 만약 두 코드의 접근 행위가 완벽하게 일치한다면, 커널에 미치는 영향도 일치한다고 판단할 수 있다. [7]에서 증명되었듯이, 동적으로 삽입되는 코드의 모든 동작은 완벽하게 파악할 수 없기 때문에 우리는 메모리 액세스에 관련하는 코드에 대해 검사를 수행함으로써 기존 커널의 안정성을 해칠 수 있는 수행을 막고, 동시에 보안 패치를 위한 수정의 여지를 남겨두고자 하였다.

DUNK에서는 기존 함수의 접근 행위를 분석하고 이를 접근 가능 리스트(access control list)로 작성한다. 패치된 함수는 기존 함수의 접근 가능 리스트를 상속받고, 리스트에 명시된 접근만이 허용된다. 이는 보안 패치로 인해 기존 코드가 접근하던 커널 영역이 아닌 다른 영역에 접근하는 동작을 막을 수 있다. 다른 영역에

대한 접근은 의도적으로 정보를 얻거나 수정하고자 하는 접근일 수도 있고, 패치 개발자의 실수로 인한 잘못된 접근일 수 있다. 따라서 이러한 접근을 차단함으로써 커널의 안정성을 저해하지 않을 수 있다. 그러나 간혹 보안 패치의 개발을 위해 특정 권한에 대한 추가적인 정보를 필요로 할 수 있다. 따라서 제한적인 허용 방안이 필요한데, 이를 위해 DUNK는 시스템 관리자가 적절한 보안 정책을 취할 수 있도록 접근 가능 리스트를 이용해 새로운 접근 권한을 부여할 수 있도록 한다.

위와 같은 검증 과정은 모두 오브젝트 코드 레벨에서 이루어진다. 따라서 앞서 언급한대로 컴파일러 등의 개발 환경을 전혀 수정하지 않아도 된다. 또한 커널 내부에 코드가 로드된 이후, 업데이트가 수행되기 전에 검증하고 시스템 관리자가 보안 정책을 설정할 수 있도록 함으로써 패치 과정의 신뢰성을 높일 수 있다.

3. 보안 정책 및 기법

DUNK의 보안 목표를 달성하기 위한 기법으로 우리는 MAFIA를 고안하였다. MAFIA는 접근 행동을 분석하여 접근 가능 리스트를 구하는 기법으로 DUNK의 보안 정책을 구현하고 적용하는 발판이 된다.

이번 장에서는 MAFIA의 동작을 오브젝트 코드 분석 기법, 분석을 통한 접근 권한 작성, 그리고 업데이트 코드를 분석하여 접근 권한과 일치하는지 검사하는 방식에 대해 예를 들어 설명한다. 예제 코드는 그림 1에 C 언어 소스코드와 이를 컴파일한 ARM 어셈블리 오브젝트 코드가 제시되어 있다. 해당 함수는 task_t 구조체에서 flag 멤버를 보고 큰 쪽의 다음 태스크 pid를 리턴하는 간단한 함수이다. New_who_is_larger() 함수는 간단하게 패치된 함수로서, 기존 함수에서 flag 멤버를 비교할 때 같은 경우 a 태스크를 선택하는데 비해 패치된 함수는 b 태스크를 선택하도록 되어있다. 이는 매우 간단한 예제이지만 실제로 비슷한 이유로 보안 취약점이 발생한 경우[8]가 있고, 이렇게 간단한 실수이지만 치명적인 결과를 초래할 수 있다.

3.1 오브젝트 코드 분석 기법

MAFIA는 오브젝트 코드를 분석하여 접근 권한을 도출해낸다. 대상은 실제 수행되는 커널 코드이다. MAFIA는 우선 어셈블리 코드를 CPU 명령어로 표현한다. 각 명령어를 디스어셈블하여 의미를 파악하고 나면 전체 명령어에 대해 그림 2와 같은 알고리즘을 통해 접근 경로를 분석한다.

접근 경로는 연결된 문자열 리스트로 표현된다. 하나의 문자열을 블록이라고 하고, 한 문자열 내에서 한 명령어는 세미콜론으로 구분되고 각각을 필드라 한다. 이러한 구분을 통해 MAFIA에서는 연산이 이루어지는 시점을 알 수 있다. 표 1은 이렇게 표현되는 문자열 리스트에 어떤 내용이 쓰여지는지를 나타낸다. 표와 같이 분류된 명령어 별로 각각의 내용이 문자열에 저장되고 세미콜론으로 구분된다. 해당 알고리즘은 우선 각 레지스터의 상태를 "INITnr"로 기록하는 것으로 시작한다. 우리는 기존 코드와 업데이트 코드가 동일한 레지스터/메모리 상황에서 시작한다고 가정한다. 이중 몇몇 레지스터는 파라미터를 저장하고 있을 것이고 이는 알고리즘에 영향을 미치지 않는다. 만약 수행 중 어떤 레지스터가 업데이트된다면 새로운 블록을 할당하고 해당 내용을 첫 부분에 기록한다. 그리고 각각의 명령어에 대해 실제로 수행되는 내용은 표 1을 따른다.

위에서 설명한 알고리즘을 이용하여 그림 1의 예제를 분석하면 그림 3과 같은 접근 행위 리스트가 나온다. 이해를 돕기 위해 3번 레지스터(R3)를 예로 들어 설명하면, 첫 번째 블록에는 초기값 INIT3이 기록되었고, 다음 블록에서는 R1에 기록된 주소에서 4 바이트 뒤의 값을 메모리로부터 읽어온다. 이는 C 코드에서 'b->flag'에

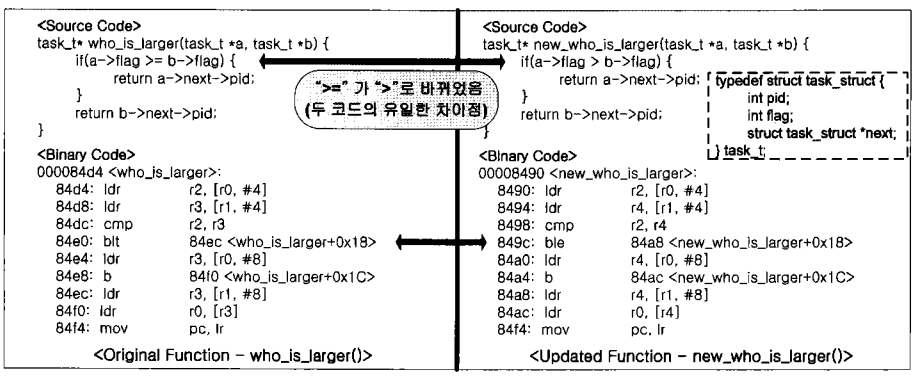


그림 1 예제 코드 - who_is_larger()와 간단하게 패치된 new_who_is_larger()

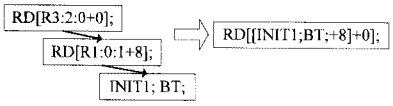
```
//Rd : Destination register, Rs: Source register
//nr : Number of current Rd

lists regs[ALL_REGS]      //linked list of strings
string node[Rd] = current node of regs[Rd]

main() {
    for (each reg.) node[nr] = "INITnr;"; //initializing
    core(address_of_first_instruction);
    for(each reg.) { allocate new node to regs[nr]; write "END;";}
    terminate;
}

core(start_address) {
    for(each instruction from start_address) {
        if(Rd will be updated) allocate new node to regs[nr];

        if(it is branch instruction) {
            for(each reg.) regs[nr] = expression from table 1;
            if(in-bound branch is taken)
                Recursively call core() from current instruction;
        } else regs[nr] = expression from table 1;
    }
}
```

그림 2 MAFIA의 수행 알고리즘

해당하는 연산이다. 이 시점에서 R1에 대해서는 아무런 동작이 수행되지 않았으므로 INIT1에 해당하는 'R1:0:0' 필드가 기록된다. 이는 첫 번째 블록의 첫 번째 필드라는 뜻이다. 그리고 0x84e0 번지의 분기명령어에서 분기했을 경우가 기록되었다. 분기에 의해 0x84ec의 명령이 수행되며 'b->next'에 해당하는 수행이 이루어진다. 현재 R1은 BT;가 추가로 기록된 상태이므로 'R1:0:1' 필드가 기록된다. 이후의 수행은 분기가 이루어진 주소로 복귀하여 위와 유사하게 이루어진다. 각각의 레지스터에 관해 위와 같은 과정을 거쳐 접근 행위 리스트를 생성할 수 있으며, 그림 3은 기존 코드와 업데이트 코드에 대한 접근 행위 리스트를 나타내고 있다.

3.2 접근 권한 리스트의 생성

접근 행위 리스트에서 우리는 접근 권한을 얻어낼 수 있다. 이것은 읽기나 쓰기, 실행이 이루어진 필드를 해석함으로써 이루어진다. 해석은 각 명령이 가리키는 필드 내용을 그대로 복사해오는 것이며, 내부에 레지스터로 표현되는 항목이 없어질 때까지 수행된다. 예를 들어 R0의 두 번째 블록에 있는 RD[R3:2:0+0];은 하나의 접근 행위이며, 이에 관한 접근 권한은 아래와 같다.



이것은 최종적으로 R0을 통해 전달되는 'b->next->pid'를 나타낸다. 그리고 주목할 것은 최종적으로 표현되는 문자열에 어떤 레지스터도 포함되지 않기 때문에, 해당 접근이 어떤 레지스터를 통해 이루어져도 무관하다는 것이다. 이는 내용의 변경 내지는 컴파일러의 변경, 컴파일 옵션의 변경으로 인해 사용되는 레지스터가 변경되었을 때도 같은 접근 행위로 인식할 수 있음을 보인다. 그리고 내부에 'BT;'가 포함됨으로써 함수의 수행 흐름 또한 일치시킬 수 있다. 이러한 동작을 모든 레지스터에 대해 반복함으로써 전체 접근 권한 리스트를 얻을 수 있다.

표 1 명령어에 따른 표현 방법

명령어 종류	표현 방법
데이타 처리 (읽기/쓰기)	<레지스터/메모리에 대한 직접 접근> 읽기: RD(where) 쓰기: WR(where) <레지스터/메모리에 대한 간접 접근> 읽기: RD[where±@] 쓰기: WR[where±@] * where = Rd:블록번호:필드번호
분기 명령어	<코드 내부로의 분기> 무조건 분기: BAW (Always) 분기시: BT (Taken) 비분기시: BNT (Not-Taken) <코드 외부로의 분기> 실행/복귀 등: BOUT * 모든 레지스터에 기록함
연산 명령	① 읽기나 쓰기가 포함된 경우, 이에 대한 표현 처리 수행 ② "명령어 이름 및 피연산자" 기록
스택 명령	POP: PP Rs PUSH: PS Rs * 모든 대상 레지스터에 대해 기록
기타 명령	"명령어 이름 및 피연산자" 기록

3.3 업데이트 코드의 분석 및 접근 권한 검증

업데이트 코드의 분석 및 접근 권한 리스트 생성은 위와 동일한 과정을 거쳐 이루어진다. 예제에서 업데이트 코드의 경우 R3 대신 R4를 사용하도록 임의로 수정하였는데, 이에 따라 접근 권한을 분석해보면 R0의 RD[R4:2:0+0]은 비록 R4를 사용하지만 기존 코드와 동일하게 RD[[INIT1; BT;+8]+0]; 이 도출된다. 이 접근 행위는 기존 코드에도 존재하기 때문에 해당 접근 행위는 허용될 것이다. 시스템 관리자는 이러한 접근 행위 검증 과정에 특별한 보안 정책을 적용할 수 있다. 예를 들어 현재 구현에서는 a, b 태스크의 pid 멤버는 접근하고 있지 않은데, 여러 코드에서 이미 R0, R1으로 전달된 인자에 대한 간접 접근을 많이 수행하고 있기 때문에 해당 인자에 대한 읽기 권한을 모두 허용하는 정책을 사용할 수도 있다. 이러한 정책의 사용은 안전성을 위협할 수도 있지만 동시에 보안 패치의 구현을 자유롭게 만들어줄 수 있을 것이다.

메모리로의 직접 접근을 제한하는 방식은 데이타 처리 명령에 대해 "READ/WRITE(메모리 주소)"와 같은 방식으로 표현한 후, 만약 기존 코드에 동일한 접근 행위가 있다면 허용하는 방식이다. DUNK의 보안 기법에서는 동일한 분석 기법을 이용하여 이와 같이 직접/간접 접근을 모두 분석하고 통제할 수 있다.

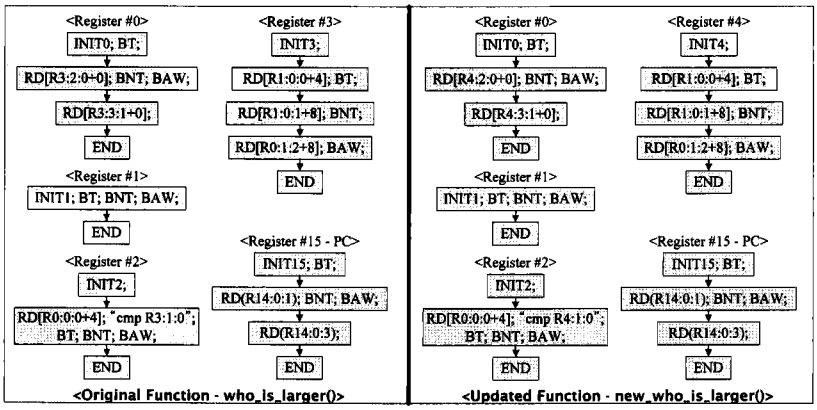


그림 3 두 예제 함수의 레지스터 별 접근 행위 분석

4. 관련 연구

커널에 동적으로 추가되는 코드에 대해 기존 커널의 안정성을 유지하고자 하는 보안 정책과 기법은 [9,10]에서 연구된 바 있다. [9]에서 제시된 SFI 기법은 소프트웨어 적으로 프로세스 간의 주소 공간을 격리시키고자 하며, 명확하게 메모리 주소로 설정된 접근 권한을 통해서만 커널의 다른 영역에 접근할 수 있다. DUNK에서는 기존 함수가 접근하는 권한이 메모리 주소로 명확하게 표현할 수 없기 때문에 이러한 기법을 바로 적용할 수 없다. [10]에서 제시된 PCC는 SFI와 유사하지만 정적인 분석 기법을 이용하여 런타임에 오버헤드가 발생하지 않도록 하는 기법이다. 추상적인 머신 코드 분석을 통해 접근 행위를 체크하는 것은 MAFIA와 유사하지만, 기존 코드에서 접근 권한을 추출하고 업데이트 코드로 권한이 상속되는 동작 등은 지원하지 않는다.

DUNK와 유사하게 동적으로 보안 패치를 시도하는 연구로 OPUS[3]가 있다. OPUS는 커널이 아닌 일반 어플리케이션을 대상으로 하며, 보안 목표로 보수적인 안전성을 지향한다. 이는 업데이트 코드가 기존 함수에 비해 전역 변수에 대한 추가적인 쓰기를 수행하지 않으며, 리턴되는 결과가 동일함을 검증한다. OPUS는 컴파일러를 수정하여 이러한 보안 검증을 수행하며 사용자에게 단순한 경고 메시지를 출력한다. 이는 동적인 코드의 분석에 한계가 있기 때문에 패치 개발자가 최종적인 코드의 검토를 하도록 유도하는 형태이다. 따라서 패치 적용 시점에서의 보안 검증이 수행되지 않는다.

5. 결론 및 향후 과제

본 논문에서는 보안 패치로 인해 시스템 서비스의 중단이 발생하거나 패치로 인해 시스템 안정성이 저하되는 것을 우려하여 패치의 적용이 늦어지는 문제를 해결하기 위해 DUNK와 MAFIA를 제안하였다. DUNK의 설계와 이에 필요한 기술들, 그리고 여러 가지 고려사항들을 제시하였으며, MAFIA 기법의 알고리즘과 구체적인 실행 방식을 기술하였다. 이를 통해 기존 함수 코드의 커널 메모리 접근 권한을 업데이트 함수의 코드가 상속받을 수 있다. 또한 정적으로 분석 및 검증이 가능하기 때문에 성능 상의 저하도 발생하지 않는다. DUNK와 MAFIA의 사용으로 우리는 동적 커널 업데이트 기법을 현재 업데이트성이 지원되지 않는 커널에서 안전하게 수행할 수 있다.

참 고 문 헌

[1] http://nvd.nist.gov/home.cfm
[2] W.A. Arbaugh, et al., "Windows of vulnerability: a case study analysis," ,pp. 52-59, Computer, Vol.33, No.12, 2000.
[3] G. Altekar, et al., "OPUS: Online Patches and Updates for Security," Proc. USENIX SS, 2005.
[4] S. Beattie, et al., "Timing the Application of Security Patches for Optimal Uptime," Proceedings of LISA XVI, 2002.
[5] http://support.microsoft.com/kb/819634/en-us
[6] Tamches, A. and B.P. Miller, "Fine-Grained Dynamic Instrumentation of Commodity Operating System Kernels," , pp. 117-130, Proc. OSDI, 1999.
[7] D. Gupta, et al., "A formal framework for on-linesoftware version change," ,pp. 120-131, IEEE ToSE, Vol.22, No.2, 1996.
[8] http://www.kb.cert.org/vuls/id/715973
[9] R. Wahbe, et al., "Efficient software-based fault isolation," ,pp. 203-216, ACM SIGOPS Operating Systems Review, Vol.27, No.5, 1994.
[10] G.C. Necula and P. Lee, "Safe kernel extensions withoutrun-time checking," ,pp. 229- 243, ACM SIGOPS Operating Systems Review, Vol.30, 1996.