가상화 기술을 이용한 버퍼 오버플로우 검출 시스템의 설계+

남현우*, 박능수*, 이강우**
*건국대학교 컴퓨터공학과
**동국대학교 정보통신공학과
e-mail:namhw@konkuk.ac.kr

The Design of Buffer-Overflow Detection System using Virtualization Technology

Hyunwoo Nam*, Neungsoo Park*, Kangwoo Lee**
*Dept of Computer Engineering, Konkuk University
**Dept of Information & Communication, Dongguk University

요 약

버퍼 오버플로우 검출을 위한 많은 방법들이 연구되었지만 완벽한 검출을 보장하지는 못한다. 이는 검출하는 도구의 수행 계층이 응용프로그램이나 커널 계층에서 동작하기 때문인데 만약 공격자가 버퍼 오버플로우 검출 도구의 존재를 사전에 인지할 경우 충분히 우회가 가능하다. 본 논문에서는 더욱 정확한 검출을 위하여 커널보다도 더 높은 권한 모드인 가상화 계층에서 검출 도구를 구현하고자 할 때 필요한 요구사항들과 설계방법, 그리고 기존 시스템들과의 비교 분석 결과를 정리하고 있다.

1. 서론

보안 취약성을 발생시키는 이슈 중에 하나로 버퍼 오버플로우(Buffer Overflow)를 말할 수 있다. 버퍼 오퍼플로우는 C 언어에서 포인터를 사용하여 메모리를 제한 없이 접근하기 때문에 발생하며 포인터는 메모리 접근을 자유롭게 하는 강력한 기능을 제공하지만 동시에 보안상의 취약성을 유발하기도 한다.

버퍼 오버플로우를 이용한 악의적인 코드의 수행을 막기 위해서 많은 연구들이 진행 중인데 아직도 완벽한 검출과 방어가 가능하지 않다. 버퍼 오버플로우를 이용한 공격을 방지하기 위해서는 먼저 프로그래머가 코드를 작성할 때 메모리 사용에 대한 정확한 이해를 통해 안전한 코드를 작성해야 한다. 그리고 컴파일러나 운영체제에서도 버퍼 오버플로우 방지를 위한 기술들이 도입되어야 한다.

버퍼 오버플로우의 분석 방법으로는 크게 정적 분석과 동적 분석 방법으로 나누며 기존 컴파일러에서의 정적 분석도구들은 시스템 운영 중에 발생하는 정보들을 사용할 수 없으며 소스 분석만으로 얻은 한정된 정보로는 검출하는데 충분하지 않았다. 이에 반해 동적인 분석 방법은 런타임 정보를 얻을 수 있기 때문에 검출에 충분한 정보를 사용할 수 있었지만 운영체제의 도움을 필요로 한다.

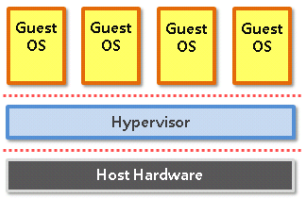
기존의 방법으로는 완벽한 버퍼 오버플로우 공격의 검출과 방어를 보장하지 못했는데 가장 큰 이유로는 검출 도구가 수행되는 계층이 안전하지 못하기 때문이다. 검출 도구들은 일반적으로 응용프로그램 계층이나 또는 더욱 강력한 임무 수행을 위해 커널 계층에서 수행된다. 하지만 커널 계층에서도 버퍼 오버플로우 공격이 가능하며 만약 검출 도구의 존재를 공격자가 인지할 경우 우회할 수 있는 가능성이 존재한다.

본 논문은 기존 방법들의 문제점들의 해결책으로 커널보다 더 높은 권한 모드인 가상화 계층에서 검색 도구를 구현하고자 하며 이를 위해 필요한 요구 사항들과 구현 방법에 대해서 정리하였다. 본 논문의 구성은 1장 서론에 이어 2장에서는 구현에 필요한 관련연구들을 살펴보고 3장에서는 제안하고자 하는 가상화 기술을 이용한 버퍼 오버플로우 검출방법에 대해서 상세히 기술하고 있으며 4장에서는 본 논문에서 제안하는 시스템과 기존 시스템을 비교 분석하고 향후 구현계획에 대하여 정리하였다.

2. 관련연구

가. 가상화(virtualization)

가상화 기술은 1960년대 IBM의 메인프레임에서부터 적용되었으며 최근 서버 관리의 효율성과 관리의 편리성으로 인해 많은 시스템에서 적용되고 있는 추세이다.



(그림 1) 가상화 시스템 구조

가상화 시스템 구조는 그림 1과 같으며 Hypervisor는 Guest OS와 하드웨어 사이에 위치한다. Hypervisor는 하나의 물리적인 하드웨어에서 다수의 Guest OS를 수행할

+ 본 연구는 서울시 산학연 협력사업(10581)의 일환으로 수행하였음.

수 있도록 Guest OS들 간의 스케줄링 및 하드웨어 자원 할당을 담당하게 된다.

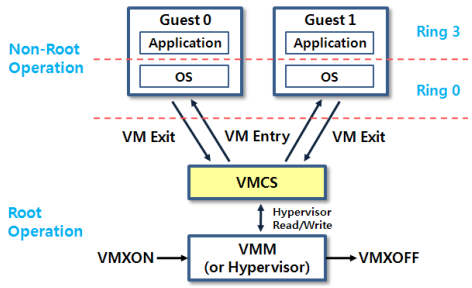
현재 Hypervisor가 위치한 계층은 커널 계층보다도 하위 계층에 위치하여 Guest OS가 사용하는 CPU, 메모리와 같은 모든 하드웨어 자원들을 관리하고 할당 작업을 담당한다. 따라서 설계하려는 버퍼 오버플로우 검출 시스템이 Hypervisor가 동작하는 가상화 계층에서 동작하면 Guest OS에서 사용하는 모든 하드웨어 자원들의 사용 정보를 취득할 수 있게 된다. 뿐만 아니라 제어까지 가능하므로 검출이 된다면 적절한 대응까지도 가능해진다.

하지만 커널이나 사용자 계층이 아닌 가상화 계층에서 악의적인 코드가 수행된다면 이전과 마찬가지로 우회 가능성이 존재하므로 검출 도구는 가상화 계층에서 제일 먼저 수행되어 가상화 계층을 점유하고 있어야 한다.

나. Intel VT-x와 AMD-V

Hypervisor를 구현하기 위해서는 커널에서 Hypervisor에게 서비스를 요청할 수 있도록 Hypercall을 구현하거나 LGDT, CLI, STI와 Sensitive 명령어를 처리할 수 있도록 기존 커널을 수정해야 했다. 이와 같은 수정 작업은 기존 커널이 사용하던 Ring0 권한 모드를 가상화 계층이 사용하고 기존 커널은 Ring1이나 Ring3 권한 모드에서 동작되기 때문이다.

이와 같은 수정작업들은 많은 노력이 필요하며 또한 커널 버전이 업그레이드 될 때 마다 매번 유사한 수정 작업들이 반복되어야 한다. 해결 방법으로 하드웨어에서 가상화 계층을 지원해야 하는데 현재 인텔은 VT-x, AMD는 AMD-V라는 이름으로 가상화 기능을 프로세서에 포함하고 있다.



(그림 2) Hypervisor와 Guest간의 전이 관계

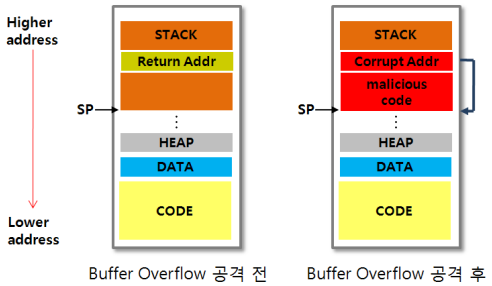
그림 2는 VT-x를 사용했을 경우 Hypervisor와 Guest OS간에 모드 전환이나 스케줄링 할 때 VM(Virtual Machine)의 제어 정보가 담겨 있는 VMCS(Virtual Machine Control Structure)와의 관계를 보여주고 있다.

VT-x 기술[1]을 사용하면 기존 구현방법에서 야기됐던 Ring 권한모드 문제점들을 해결할 수 있다. 그림 2에서 Hypervisor는 가상화를 위해 추가된 Root Operation 모드에서 Guest는 Non-Root Operation 모드에서 수행되는 것을 확인할 수 있다. Guest OS의 커널은 이전과 같은 Ring0에서 작동하고 Hypervisor는 새롭게 추가된 계층에서 작동하므로 따라서 커널의 수정이 필요하지 않다.

VMCS 에는 Hypervisor에 의해 스위칭이 VM간에 발생할 때마다 각 VM의 CPU 레지스터 정보들이나 관련 제어정보들을 저장한다. 이는 운영체제에서 멀티태스킹을 할 때 Context Switching이 발생할 때마다 프로세스의 CPU 레지스터나 제어정보들을 저장하는 것과 유사하다.

다. 버퍼 오버플로우(Buffer Overflow)

버퍼의 크기에 비해 많은 양의 데이터가 입력되어 버퍼가 넘치게 될 경우 오버플로우가 발생되며 의도하지 않은 메모리 영역에 데이터가 써지게 된다. 공격자들은 이러한 버퍼의 특성을 이용하여 고의적으로 오버플로우를 발생시킨 후 오버플로우 된 공간에 악성코드를 삽입한다.



(그림 3) 버퍼 오버플로우 공격시 메모리 맵

그림 3은 버퍼 오버플로우 공격이 발생했을 때의 메모리 맵이다. 왼쪽 메모리 맵은 공격전의 모습인데 함수 호출이 발생했을 경우 스택에는 인자 값들과 복귀주소 등을 저장한다. 하지만 공격 후에는 스택공간에 악성 코드가 삽입되고 복귀주소에는 삽입된 악성코드의 시작 주소가 기입되게 되는데 함수에서 모든 작업을 끝마치면 호출한 함수로 복귀할 때 악성코드로 점프하게 된다.

버퍼 오버플로우 공격에 대응하는 방법으로는 컴파일 타임에 소스코드만을 가지고 분석하는 정적인 방법과 실행중에 발생되는 정보들을 사용하여 분석하는 동적인 방법이 있다. 본 논문에서 구현하고자 하는 것은 동적인 방법으로 모든 자원들의 사용여부 탐지와 제어가 가능한 Hypervisor 계층에서 검출 도구가 실행되어 기존 커널기법의 보안 도구들 보다 강력한 기능을 제공한다.

실제 대응 방법으로는 함수 호출시 스택에 저장된 복귀주소 주위에 Guard Value를 함께 기입하는 방법[2]이다. 만약 버퍼 오버플로우가 발생된다면 Guard Value는 바뀌게 되므로 함수 호출 복귀시 이 값이 원래의 값인지를 확인하여 공격 여부를 판단할 수 있다. 하지만 대응 방법을 공격자가 인지할 경우 충분히 우회가 가능하며 컴파일을 할 때 함수 호출을 할 때마다 Guard Value를 삽입해야 하므로 컴파일러의 도움이 반드시 필요하다.

다음은 바운드 검사를 통해 스택 오버플로우를 검출하는 방법으로 메모리 맵에 대한 정보를 바탕으로 스택이나 힙 영역에서 코드의 실행이 발생하였을 경우 이를 검출하는 방법이다. 이러한 방법을 적용하기 위해서는 현재 수행되는 스레드들의 메모리 맵 정보를 알아야 하고 모든 메모리 접근시에 올바른 메모리 영역 내에서의 접근인지를 검사해야 한다.

다른 방법으로는 하드웨어적인 지원이 필요한 경우로 SecureBit를 사용하는 방법[3]이다. 이 방법은 메모리 디스크립터에 1 bit를 SecureBit로 할당하여 CALL, RET, 그리고 JMP 같은 명령어가 수행되었을 때 Secure 영역이라면 비트를 1로 설정해준다. 이렇게 함으로써 도메인의 Secure 한 영역으로부터 수행되었다는 것을 알 수 있게 해준다. 하지만 이 방법은 CPU에서 메모리 접근 명령이 발생할 때마다 추가적으로 SecureBit를 처리해야 하므로 적용결과 30배 이상 느려졌으며 CPU에서 이 기술을 적용해주지 않으면 사용이 불가능한 기술이다.

MS Windows에 적용된 Data Execution Prevention(DEP)이라는 기능은 코드 영역을 제외한 메모리 영역의 Page Table의 NX(No eXecute) 비트를 설정하여 실행을 방지하는 방법이다. 이 비트의 수정작업은 전체적인 메모리맵을 관리하는 커널에서 수행한다.

그리고 다른 방법으로는 스택 공간에 대한 메모리 공간 영역을 Read-Only 속성으로 바꾸어주는 것이다. 스택 공간은 데이터를 보관하는 공간으로 코드가 실행되면 안된다. 따라서 코드영역 외에 메모리의 속성을 읽기 전용으로 변경하면 데이터 공간인 버퍼 공간 내에서 악성코드가 수행되는 것을 방지할 수 있다.

위와 같은 기존의 여러 대응 방법들은 가상화 계층에서 구현되었을 경우 더욱 강력해질 것이다. 상세한 설계 방법에 대해서는 3장에서 언급하겠다.

위와 달리 Sandboxing 기법은 오버플로우가 발생하였을 경우 이를 직접적으로 막지는 않지만 공격이 더 이상 다른 프로세스들에게 영향을 끼치지 못하도록 악성코드의 실행 환경을 제한하는 보안 기법이다. 추가적으로 가상화 계층은 Sandboxing을 가능하게 하는 매개체로서 활용 가능성이 예상된다.

라. Virtualized Rootkit

이 논문은 버퍼 오버플로우 공격 탐지 도구가 가상화 계층에서 수행되기 때문에 기존 커널 계층에서 구현된 기존 방법들보다 효과적임을 증명하고자 한다. 하지만 역으로 가상화 계층에서 공격자의 악의적인 코드가 먼저 수행된다면 기존 검출 도구로는 검출조차 할 수 없을 정도로 보안상 큰 위험이 가해질 수도 있다.[7]

실제 MS와 미시간 대학에서는 가상화 기술을 이용한 해킹도구의 위험성을 알아보기 위하여 Subvirt[4] 라는 루트킷을 제작하였다. 이 논문은 기존의 루트킷보다 더욱 치명적인 공격능력을 가지고 있음을 증명하였다. Subvirt 외에도 Intel VT-x 기술을 사용한 DBvm[6] 이라는 게임 치트 엔진의 소스도 공개되어 있다. 이 엔진은 하드웨어 가상화 기술을 이용하여 기존 커널의 수정 없이도 운영체제상의 모든 메모리의 접근과 조작을 통해 게임을 해킹하는 기능을 제공한다.

3. 제안하는 버퍼 오버플로우 검색 도구의 설계

이장에서 제한하는 방법들은 기존에 버퍼 오버플로우 공격을 방지하기 위해 고안된 방법들이 가상화 계층에서 어떻게 적용될 수 있는지에 대해서 기술하고 있다.

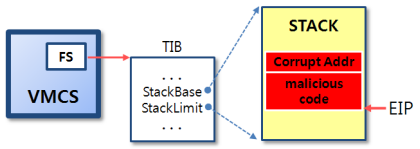
가. 바운드 검사를 이용한 검출 방법

기존 방법 중에 바운드 검사를 통한 버퍼 오퍼플로우 검출방법을 가상화 계층에서 구현하고자 한다. 우선 알아야 할 정보들은 현재 수행되고 있는 스레드의 스택의 위치와 크기에 대한 정보이다. 이러한 정보들은 윈도우 운영체제의 경우 TIB(Thread Information Block)에 기록되어 있으며 이 자료구조는 CPU의 FS 레지스터가 가리키고 있다. 여기서 TIB는 스레드의 SEH(Structured Exception Handling)나 스택 정보등을 담고 있는 커널 자료구조이다.

<표 1> TIB(Thread Information Block) 자료구조

```
typedef struct _NT_TIB {
    struct _EXCEPTION_REGISTRATION_RECORD
*ExceptionList;
    PVOID StackBase;  // 스택의 시작 주소
    PVOID StackLimit;  // 스택의 크기
    PVOID SubSystemTib;
    union {
        PVOID FiberData;
        DWORD Version;
    };
    PVOID ArbitraryUserPointer;
    struct _NT_TIB *Self;
} NT_TIB;
```

만약 특정 프로세스의 버퍼 오버프로우 공격을 검출하고자 한다면 스택 정보를 알아온 다음 현재 수행하는 코드의 주소를 나타내는 EIP 레지스터가 스택영역에 있는지를 체크하는 것이다. 만약 EIP가 스택 내부의 주소를 가리키고 있다면 이는 버퍼 오버플로우 공격이라고 판단하며 Fault를 발생시켜 작업을 중단 시킨다.



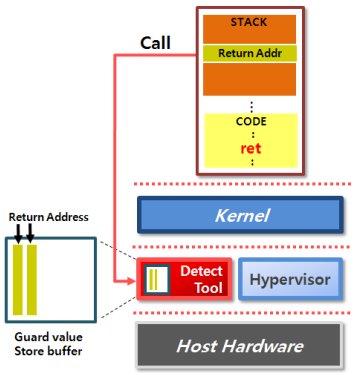
(그림 4) 제안 시스템의 공격 검출 구조

그림 4는 제안하고자 하는 공격 검출 시스템의 작동구조를 보여주는데 VMCS에는 각 VM의 제어 정보와 CPU 레지스터 정보 등의 데이터들이 들어있다. 스택에 대한 정보를 알아내기 위해 먼저 VMCS에서 FS 레지스터를 통해 현재 윈도우즈 운영체제에서 작동하고 있는 스레드의

TIB에 접근한다. TIB에는 스택 정보가 들어있으며 현재 해당 코드의 실행 위치인 EIP 레지스터의 값이 스택 내부에 위치한다면 버퍼 오버플로우 공격으로 판단한다.

나. 복귀주소 비교를 통한 버퍼 오버플로우 검출 방법

이 방법은 함수 호출시 저장된 복귀주소가 변경되었는지를 검사함으로서 악성코드의 실행을 방지하는 방법이다. 그림 5는 제안하는 시스템에서 CALL 명령어를 통해 함수를 호출하였을 경우의 수행 과정을 보여준다. CALL 명령어가 실행되면 복귀주소를 가상화 계층의 검출 도구에 저장한다. 그리고 RET 명령어가 실행되면 복귀주소로 이동하기 전에 저장된 복귀주소와 비교하여 달라졌는지를 검사한다. 만약 CALL을 했을 때 저장된 복귀주소와 RET을 했을 때 사용하는 복귀주소가 달라졌다면 이를 오버플로우 공격으로 인식하고 적절한 대응을 해주게 된다.



(그림 5) 함수 호출시 복귀주소 저장

이러한 방법은 복귀주소를 가장 높은 권한 모드인 가상화 계층에서 저장하고 있으므로 커널 계층이나 사용자 계층에서 실행되는 악의적인 코드로부터도 안전하다. 기존에 유사한 검출 방법으로는 Guard Value를 사용하는 경우가 있는데 공격자가 해당 공격 패턴이나 Guard Value의 존재를 알 경우 우회가 가능하다는 단점이 발생한다. 하지만 제안 방법은 Guard Value와 같은 Signature를 스택공간에 저장하지도 않으며 저장하고 있는 복귀주소도 커널보다도 더 높은 권한 계층에서 관리하기 때문에 기존 공격들이 무력화된다. 하지만 call, ret 명령이 발생시 Guard Value를 관리할 코드를 수행할 방법이 존재하지 않는다. 따라서 컴파일러나 프로세서의 도움을 반드시 필요로 하게 된다. 기존의 검출 방법들을 가상화 계층에서 구현했을 경우 장점들을 정리해보면 아래와 같다.

■높은 권한 모드 : 추가된 가상화 계층은 기존의 커널 계층보다도 높은 권한을 가지고 있어 기존에 커널이나 사용자 계층에서 작동하던 공격 프로그램들이 무력화 됨.

■Guest 모니터링 : VM이 사용하는 모든 자원 정보들에 대한 자유로운 접근과 제어가 가능.

■Isolation : 각 VM 간은 서로 영향이 미치지 못하므로 하나의 VM 상에서 발생한 보안 취약성이 다른 VM에 영향을 주지 못하도록 Sandboxing 기능을 제공.

4. 결론

이 논문은 가상화 기술이 버퍼 오버플로우 검출 같은 모니터링 도구로서의 가능성에 대하여 연구한 결과이다. 최근 가상화는 서버의 자원 활용도를 높이고 관리상의 이점을 제공할 뿐 아니라 보안상으로도 새로운 가능성을 제공해주는 플랫폼으로서 부각되고 있다. 그리고 최근 보급된 CPU들은 기본적으로 가상화 기능을 제공하고 있기 때문에 앞으로 가능성이 더 커질 것으로 예상된다.

기존의 보안 관련 도구들은 커널을 수정하거나 일부 제한된 정보와 접근 제어 권한만을 가지고 작동하였는데 Hypervisor 기반의 도구들은 VM상에 돌아가는 OS와 응용프로그램에 대한 모든 메모리의 접근과 제어가 가능하다. 이는 더욱 강력한 기능 구현의 가능성을 제공하며 기존 보안 도구들을 공격하거나 우회하려는 많은 방법들을 무력화 시킬 수 있을 것이다.

이 논문은 버퍼 오버플로우 검출 시스템의 구현에 앞서 구체적인 구현 방법에 대해 그 가능성을 정리하고 있다. 최종적으로 보안 관리를 위해서는 VT-x와 같은 하드웨어의 가상화 기능을 이용한 Thin Hypervisor의 구현을 목표로 하고 있다. Thin Hypervisor는 기존 커널 기반의 IDS를 가상화 계층으로 옮길 수 있는 기반 플랫폼으로써 사용될 것이다.

참고문헌

[1] Yaozu, D et al., "Extending Xen with Virtualization Technology" Intel Technology Journal, volume 10, Issue3, 2006.

[2] Cowan, C et al "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attack", Proceedings of the 7th USENIX Security Symposium, Jan, 1998

[3] Piromsopa K, Richard J, "Secure Bit: Transparent, Hardware Buffer-Overflow Protection", IEEE Transactions on Dependable And Secure Computing, Vol3, No4, October-December 2006.

[4] Samuel T. King et al, "Subvirt: Implementing malware with virtual machines" IEEE Symposium on Security and Privacy., pp. 314-327, 2006

[5] www.intel.com, "Intel 64 and IA-32 Architectures Software Developer's Manual Vol.3 System Programming Guide" Intel Corporation, 2006.

[6] DBvm, http://www.cheatengine.org/aboutdbvm.php

[7] 윤근용, "루트킷의 예고된 위협-가상화" 마이크로소프트웨어 4월호, 2008

[8] 정덕영, "Windows 구조와 원리-OS를 관통하는 프로그래밍의 원리" 한빛미디어, 2006