New RAD : 버퍼 오버플로우 공격에 대한 해결책

이민재°, 한경숙
한국산업기술대학교
{wizz°, khan}@kpu.ac.kr

New RAD(Return Address Defender) : The Solution of Buffer Overflow Attack

Minjae Lee°, Kyungsook Han
Korea Polytechnic University

요 약

"버퍼에 대한 바운드 체크를 하지 않는다"라는 작은 취약성 하나가 버퍼 오버플로우라는 큰 위협을 만들어냈다. 그러나 그것을 단지 C언어의 무결성 문제로 단정 지어 버릴 수도 없는 문제이다. 일반적으로 버퍼 오버플로우 공격은 메모리에 할당된 버퍼의 경계를 넘는 데이터를 입력하여 프로그램의 함수복귀주소를 조작하고 공격자가 원하는 코드를 삽입하여 이루어진다. 이러한 버퍼 오버플로우에 대한 여러가지 대응책들이 나왔지만 약간의 문제점들을 가지고 있다. 그래서 본 논문에서는 운영체제의 세그멘테이션 기법을 이용하여 그 공격에 대응할 수 있는 한 가지 방법을 제시하고자 한다. 기존의 스택가드(카나리아 버전)의 문제점인 우회공격과 카나리아 워드를 추측하여 이루어지는 공격 그리고 MineZone RAD의 문제가 되는 DRAMA 등에 있어서도 효과적으로 방어할 수 있을 것으로 기대한다. 또한 스택가드(Memguard 버전)에서 이곳저곳에 산재되어 관리하기 어려운 함수복귀주소를 별도의 세그먼트 테이블로 쉽게 관리할 수 있을 것이다.

1. 소 개

버퍼 오버플로우 공격[1]은 메모리에 할당된 버퍼의 경계를 넘는 데이터를 입력하여 프로그램의 함수 복귀 주소(return address)를 조작하고 궁극적으로는 공격자가 원하는 코드를 삽입하여 실행하는 것이며, 그 원인은 "버퍼에 대한 바운드 체크를 하지 않는다"라는 작은 취약성 하나에서 비롯되었다.

버퍼 오버플로우는 1973년경 C언어의 데이터 무결성 문제로 그 개념이 처음 소개 되었으며 처음에는 해킹 기법이 아닌 단순한 프로그램상의 문제였다. 그러나 1988년 Robert T. Morris에 의해 인터넷 웜이라고 불려 지게 되면서 문제점으로 제기되기 시작하였다.

버퍼 오버플로우가 많이 알려졌음에도 불구하고 계속하여 보안성의 위협을 받는 이유는 첫째, 프로그래머들이 배열의 범위를 위한 프로그램에서의 메모리 할당 규율을 가지고 있지 않으며, 컴파일러 역시 대부분 데이터 영역을 연속적으로 생성하는데 있고, 둘째, 이 취약성에 대한 모든 적용 사례가 발견되지 않았을 뿐만 아니라 이것을 방어하기 위해 전체적으로 시스템과 컴파일러를 바꾸는 것 역시 쉬운 일이 아니기 때문이다[2].

본 논문에서는 버퍼 오버플로우의 공격원리와 방어 메소드의 유형을 살펴보고 기존의 스택가드와 실행불가스택 등의 문제점을 제기하고 다른 방면에서의 버퍼 오버플로우에 대한 대응책을 하나 제시하려고 한다.

본 논문의 구성은 2절에서는 버퍼 오버플로우의 공격 원리를 살펴본고 3절에서는 방어 메소드의 유형에 대해 알아본다. 그리고 4절은 방어 메소드에 대한 관련 연구로 스택가드, 실행불가스택, RAR 개념과 문제점에 대해 논하며, 5절은 New RAD라는 새로운 방법을 제시한다. 6절은 전체적인 결론을 내린다.

2. 관련 연구

2.1. 버퍼 오버플로우의 공격 원리

대부분의 프로그램은 버퍼의 바운드를 체크하지 않는다. 그로 인하여 만일 버퍼 크기보다 큰 데이터가 들어오면 그 버퍼에 인접한 영역은 그 여분의 데이터에 의해 덮여 쓰여 질 수 있다. 이와 같은 특성을 이용하여 버퍼 오버플로우 공격이 이루어진다. 유사한 속성을 가진 프로그램 변수들은 일반적으로 같은 메모리 영역에 할당되고, 데이터 구조의 경계를 넘어서 쓰게 되면 이웃한 데이터 구조에 덮여 쓰게 되므로 그 값들이 변경될 것이다. 만일 덮여 쓰이는 부분에 함수의 복귀 주소를 포함하고 있었다면 그 함수는 복귀할 때 새로운 값을 다음 명령의 주소로 사용하게 될 것이다.

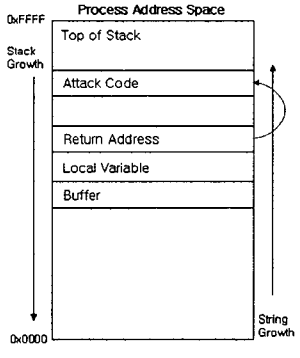

그림 1. 스택 스매싱 버퍼 오버플로우 공격

공격자는 그림 1과 같이 메모리에 코드를 삽입하고 삽입된 코드의 포인터를 함수의 복귀 주소를 변경하여 그 삽입한 코드를 수행하여 루트 권한을 얻어낼 수도 있다[2].

2.2. 방어 메소드의 유형

일반적으로 버퍼 오버플로우에 대한 방어 메소드는 세 가지로 분류될 수 있다[2].

첫째는 악의적 코드의 삽입을 막는다. 그 예로 Richard Johns과 Paul Kelly는 프로그램을 보호하기 위해 배열의 바운드와 포인트를 체크하는 컴파일러(gcc)와 커널을 패치하였다[4].

둘째, 코드의 삽입과 복귀 주소의 수정은 할 수 있으나 허가되지 않는 제어 흐름을 막는다.

셋째, 삽입된 코드를 실행할 수 없게 만든다. 즉, 외부 침입을 감지하여 실행할 수 없게 만든다. 그 예는 다음과 같다.

① R. Sekar와 P. Uppuluri 메소드

각 프로그램에 대한 정상/비정상의 동작 패턴이나 리소스들을 규정짓고 보호를 위해 각 프로세스나 리소스의 실행시간에서의 동작을 비교하면서 침입 패턴을 찾아낸다[5].

② Wenke Lee와 Sal Stolfo 메소드

감시(inspection) 데이터에 대한 데이터 마이닝 접근을 통해 리소스의 패턴으로 시스템을 보호한다. 침입검출시스템을 위해 많은 패턴이 수집되어야 하므로 전략적인 효과는 프로그램들의 정확한 패턴의 유무에 좌우된다[6].

③ Solar Designer's Non-Executable Stack

스택 영역을 실행이 불가능한 상태로 만들어 비록 제어 흐름이 삽입된 코드로 옮겨져도 코드를 실행할 수 없다. 자세한 내용은 2.4절에서 다루고 있다[7].

2.3. 스택가드(StackGuard)

스택가드[3]는 함수 복귀전에 복귀 주소의 변경이 발생했는지를 감지하는 효과적인 방법 중에 하나이다. 그림 2와 같이 스택에 카나리아 워드(canary word)를 두고 그 뒤에 함수의 복귀 주소를 저장한다. 함수가 복귀 주소에 의해 분기되기 전에 카나리아 워드가 온전한지 살펴본다. 그래서 카나리아 워드가 변경되지 않으면 복귀 주소도 변경되지 않았다고 간주한다. 카나리아 워드가 있다는 사실을 알지 못했다는 가정하에서 큰 효과를 볼 수 있으나 다음과 같은 문제점을 안고 있다.

첫째, 카나리아 워드를 우회하여 공격이 이루어질 수 있다. 만일 공격자에게 구조체 배열의 복사본이 체크 되어서 그 위치를 아는 것이 가능하고 그 배열의 정렬(alignment) 요구가 있어서 배열이 밀집된 팩킹하는 중에 요구를 수용할 만큼 크지 않으면 카나리아 워드 부분이 배열의 구멍으로 남게 된다.

둘째, 카나리아 워드를 추측하여 공격이 이루어질 수 있다.

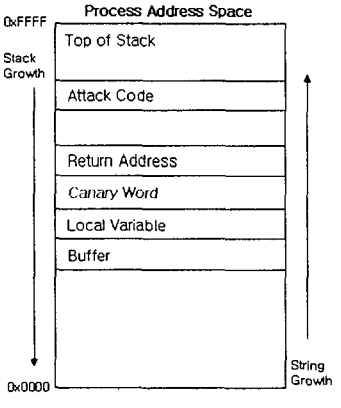

그림 2. 카나리아 워드의 위치

2.3.1. Random Canary Version

스택가드의 Random Canary 버전은 crt0라이브러리를 수정한 임의의 카나리아 워드를 사용한다. 그러나 카나리아 워드의 값을 추측하는 것이 어렵지는 않다는 데에 문제점이 있다. 그리고 공격자가 선택되어진 워드를 얻기 위해 실행 프로세스의 메모리 이미지 가로채기를 시도한다면 그 값을 알아낼 수도 있다.

2.3.2. Memguard Version

스택가드의 Memguard 버전은 함수가 호출될 때 복귀 주소를 읽기 전용 속성으로 만들어 보호하고 반환시 읽기 전용 속성을 해제하는 방식으로 버퍼 오버플로우 공격에 대항한다. 그러나 함수 복귀 주소는 스택 안에 이곳저곳에 산재되어 있다. 함수복귀주소를 보호하기 위해서는 별도의 테이블 형태로 관리되어질 필요가 있다. 각 테이블에 각 항목에 대한 보호 여부에 대한 처리가 이루어져야 하므로 복잡할 수밖에 없다.

2.4. 실행불가스택(Non-Executable Stack)

Solar Designer가 개발한 리눅스 패치로 스택 스매싱 문제에 대항하여 공격자가 스택에 공격 코드가 삽입하더라도 사용자 프로세스의 가상 어드레스 공간의 스택부분을 실행할 수 없게 만들어서 버퍼 오버플로우 공격에 대항한다.

이 패치의 장점은 성능 저하가 없다는 것이다. 그리고 해당 프로그램의 다시 컴파일하지 않아도 보호된다. 그러나 이 확장이 표준으로 채용되지 않는 한 특별한 패치된 커널을 실행해야 하고 다음과 같은 제약 조건이 따른다.

첫째, gcc는 중첩된 함수들의 호출과 반환 작용을 위해 실행 가능한 스택을 필요로 한다.

둘째, 리눅스 시그널 처리를 위해 실행 가능한 사용자 스택을 필요로 한다.

셋째, 함수형 언어와 몇몇 다른 프로그램들이 실행시간에 코드 생성하므로 실행 가능한 스택을 필요로 한다.

위와 같은 제약 사항 때문에 무조건 스택을 실행불가로 만들 수 없다. 그래서 시그널을 전달하여 시그널에 따라 실행할 수 있게 하는 방법과 gcc를 위해 중첩된 함수들의 호출과 반환하는 작용들이 가능하도록 스택에 실행코드를 둔다. 그러나 이러한 방법들은 시그널 핸들러 오버플로우라는 또 다른 취약성을 초래할 수 있으며 포인터를 이용한 우회 공격이나 힙이나 정적 데이터 세그먼트에 할당받지 않은 버퍼에 대한 공격에는 취약할 수밖에 없다[7].

3. RAD(Return Address Defender)

RAD[2]는 함수 프로토콜의 보호 코드와 그것에 의해 컴파일되는 프로그램의 에필로그를 자동적으로 추가하는 gcc 2.95.2 패치로 소스 코드를 수정할 필요가 없다. RAR(Return Address Repository)이라는 데이터 세그먼트의 특정지역에 함수의 복귀 주소의 복사본을 저장하고 읽기전용 속성으로 만들어 보호한다. 그래서 공격자가 코드를 삽입했다고 하더라도 스택에 있는 함수 복귀 주소에 의해 분기가 이루어질 때마다 RAR에 복사된 주소와 비교함으로 버퍼 오버플로우 공격으로부터 보호할 수 있다.

3.1. MineZone RAR

그림 3과 같이 RAR 영역의 앞뒤에 MineZone이라고 불리는 일정한 영역을 읽기전용으로 만들고 MineZone에 쓰기 시도가 될 경우 *mprotect()*를 호출하여 RAR을 읽기전용으로 하여 함수의 복귀 주소를 보호한다. 그러나 DRAMA(Direct Return Address Memory Attack)에 공격당할 수 있다.

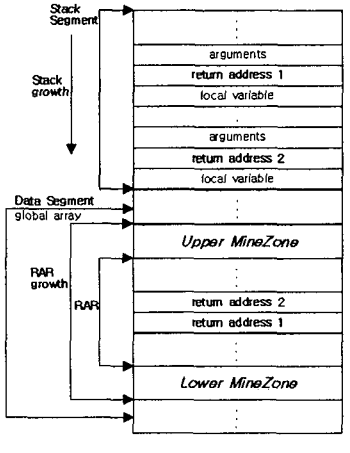

그림 3. RAR과 MineZone

3.2. Read-Only RAR

RAR 자체를 읽기 전용 속성으로 만들어서 보호하며, 유일시간에 함수 프롤로그에서 함수복귀주소를 RAR에 푸쉬(push)하게 되므로 DRAMA에도 대항할 수 있다. 그러나 RAR이 읽기전

용이므로 업데이트하기 위한 함수 프롤로그에 두 개의 시스템 호출이 추가되어야 하므로 막대한 성능저하를 가져올 수 있다.

4. New RAD(Return Address Defender)

버퍼 오버플로우에 대한 대응책은 앞에서 말한 방법 외에도 여러 가지가 있지만 본 논문에서는 앞에서 제시한 방법의 문제점을 극복할 수 있는 한 가지 방법을 제시하고자 한다. 그것은 바로 운영체제의 세그멘테이션 기법을 이용하는 것이다.

우선 함수복귀주소만을 저장하는 세그먼트(Return Address Segment, 이하 RAS)를 만든다. 그리고 그림 4와 같이 스택에 함수복귀주소가 저장되는 부분에 RAS에 있는 실제 함수복귀주소가 저장된 부분을 가리키게 한다. 즉, 가상의 함수복귀주소를 쓰는 것이라고 할 수 있다.

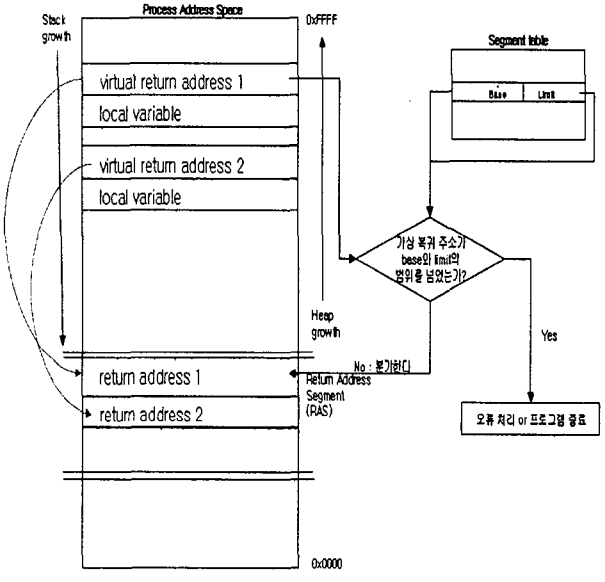

그림 4. New RAD

함수가 분기할 때에 그 가상함수복귀주소를 가지고 세그먼트테이블을 참조하여 RAS상의 실제 함수복귀주소를 가지고 함수 분기가 일어나게 된다. 그러므로 공격자가 버퍼 오버플로우 공격을 시도하여 함수복귀주소가 위조되더라도 그 부분은 가상함수복귀주소이고 그것이 변경되었다고 할지라도 실제 분기가 일어나기 전에 가상함수복귀주소와 세그먼트 테이블의 기본(base), 한계(limit) 주소와 비교가 이루어지게 된다. 따라서 그 범위를 넘으면 분기하지 않고 오류를 발생하여 프로그램을 종료하므로 그 공격을 저지할 수 있다.

앞에서 기본의 여러 가지 대응책들이 약간의 문제점을 가지고 있다고 언급했었다. 그러한 문제점들에 대한 해결은 다음과 같다.

첫째, 스택가드의 카나리아 버전을 사용할 경우 카나리아 워드의 위치를 알게 되어 이루어지는 우회 공격이나 카나리아 워드의 값을 추측하여 함수의 복귀 주소를 위조하는 방식으로 공격이 이루어지게 된다. 그러나 New RAD는 만약 가상 복귀 주소가 위조되더라도 세그먼트 테이블의 범위를 벗어나면 분기하지 않으므로 이런 패턴의 공격은 성공할 수 없다.

둘째, 스택가드의 Memguard 버전은 강력한 방어는 할 서 있는 대신 보호모드의 작동과 해제에 대한 비용이 들어간다. 그러나 이 방법도 보호모드의 작동과 해제 부분이 있어서 구현되지 않은 관계로 그 비용의 차이는 말할 수 없다. 그렇지만 Memguard는 함수복귀주소가 있는 부분을 보호모드를 구성하므로 그 위치가 일정하지 않고 산재되어 있어 관리상의 어려움이 있다. 그렇지만 이 방법은 사용할 경우에는 하나의 별도의 세그먼트에 관리되므로 관리적 차원에서도 용이하다고 말할 수 있다.

셋째, 실행불가스택은 추가비용이 없는 유용한 방법이지만 여러 가지 예외사항을 위해 스택을 실행영역으로 만들어야한다. 그로 인한 다른 취약성이 발생한다. 그렇지만 이 방법은 스택 영역 자체는 실행이 가능하므로 그런 문제는 더 이상 거론될 수 없다.

넷째, MineZone RAD에서 발생하는 DRAMA를 이용하여 가상함수복귀주소를 위조하여 RAS에 들어온다고 할지라도 그 영역은 보호모드 작동시에는 읽기전용모드이고 보호모드까지 무력화 했다고 가정할지라도 버퍼가 없으므로 그 안에서의 버퍼 오버플로우 공격은 원천적으로 불가능하다.

마지막으로, ReadOnly RAD에서 보호모드 작동과 해제에 대한 비용문제가 있다고 언급했다. 이 방법의 세그먼트 테이블의 주소 비용과의 비교 문제는 구현이 되지 않아 논할 수 없지만 New RAD도 유일시간에 RAS에 쓰기가 이루어지므로 방어력의 차원에서는 ReadOnly RAD 수준의 버퍼 오버플로우 공격에 대한 방어는 보장할 수 있을 것이다.

5. 결론

버퍼 오버플로우 공격의 방법은 다양하며 아직 알려지지 않은 방법도 많다. 그렇지만 버퍼 오버플로우라는 취약성이 우리가 그냥 넘어갈 수 있는 작은 문제가 아닌 것은 분명한 사실이다. 효과적인 방어와 성능의 저하는 서로 반비례한다. 예를 들면, Read-Only RAR이나 배열과 포인터에 대한 모든 접근을 체크하는 방법 등의 방법이 방어 차원에서는 우수하나 성능저하를 가져오는 사례가 많다.

본 논문에서 제시하는 방법은 구현이 되지 않아 정확한 성능상의 문제는 알 수 없으나 RAS의 보호 모드를 작동/해제하는 비용이 충분히 들 수 있을 것이다. 그러나 버퍼 오버플로우에 대한 방어적인 차원에서는 간단한 방법(스택가드-카나리아 버전, MineZone RAD 등)들에 비해 보다 훨씬 강력한 보호를 할 수 있다고 말할 수 있다.

향후과제는 이 아이디어를 구현하는 것이고 구현을 위해서는 운영체제에서 RAS를 사용할 수 있도록 세그멘테이션 테이블을 수정하는 커널 패치가 이루어져야 하며, 컴파일러에서는 보호모드의 작동과 해제를 위해서 함수의 프롤로그와 에필로그에 대한 패치가 이루어져야 한다.

5. 참고 문헌

[1] Crispin Cowan et al, "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade"

[2] Tzi-cker Chiueh et al, "RAD: A Compile-Time Solution to Buffer Overflow Attacks"

[3] Crispin Cowan, Calton Pu, et.al. , "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", Proceedings of the 7th USENIX Security Conference, San Antonio, Texas, USA, 1998.

[4] Richard W M Jones and Paul H J Kelly, "Backwardscompatible Bounds Checking for arrays and pointers in C programs", http://www-ala.doc.ic.ac.uk/-phj WBoundsCheckinig.html.

[5] R. Sekar and P. Uppuluri, "Synthesizing Fast Intrusion DetectionPrevention Systems from High-Level Specifications", USENIX Security Symposium, 1999

[6] Wenke Lee and Sal Stolfo, "Data Mining approaches for Intrusion Detection", Proceedings of the Seventh USENIX security Symposium (SECURITY '98), San Antonio, TX, January 1998.

[7] Solar Designer, "Non-Executable User Stack", http://www.openwall.com/.