버퍼오버플로우 방지를 위한 효율적인 이진코드 재작성 기법[1)]

김윤삼, 조은선
충북대학교 전자계산학과
e-mail: bijak1@hotmail.com, eschough@chungbuk.ac.kr

Efficient Binary Code Rewriting Technique for Buffer-Overflow Prevention

Yun-Sam Kim, Eun-Sun Cho
Dept of Computer Science, Chung-buk University

요 약

버퍼 오버플로우 공격의 방어는 그 심각한 위험성 때문에 많은 연구가 되고 있지만 방어에 의한 오버헤드의 발생으로 인해 실제 적용되기 어려운 면이 있다.

본 논문은 이진 코드를 재작성 하여 스택의 리턴 주소 사본을 지역변수 아래 부분에 두고 함수 반환시 비교 검사를 하는 것으로써, 소스코드가 없는 경우에도 버퍼 오버플로우 공격을 막는 동시에 오버헤드를 줄일 수 있는 방법을 제안하였다.

1. 서론

몇 년 사이 인터넷 해킹 등 피해사례 조사를 보면 버퍼 오버플로우 공격에 의한 사례가 급증하고 있다. 버퍼 오버플로우 공격은 C와 같은 프로그래밍 언어에서 strcpy등 문자열에 관련된 함수가 경계조건을 검사하지 않기 때문에 발생한다.

버퍼 오버플로우 공격은 악성 코드의 삽입과 악성 코드가 실행되도록 리턴 주소를 조작하는 두가지 과정을 갖는다. 특정한 위치에 악성 코드를 삽입하고, 그림 1과 같이 스택 메모리에 존재하는 배열을 strcpy등의 취약한 함수로 기록하면서 리턴 주소를 악성 코드가 존재하는 주소로 수정하여 함수가 종료하여 원래 위치로 돌아갈 때 악성 코드를 실행시킨다.

2. 관련 연구

버퍼 오버플로우 공격을 막기 위한 방법으로는 소스 코드를 스캔하여 취약한 부분을 밝혀내거나 수정된 컴파일러를 사용하는 방법, 실시간 탐지 기법 등이 있다.[1]

그 중 소스코드가 주어지는 경우에 사용되는 대

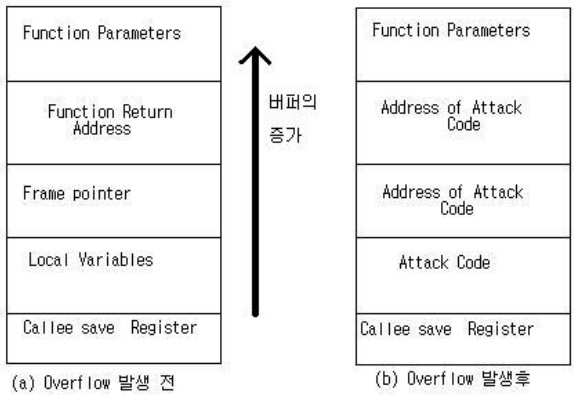

그림 1 오버플로우의 발생

표적인 방법으로 컴파일러를 수정하여 버퍼 오버플로우를 탐지하는 기법이 있다. Immunix의 StackGuard[2]는 버퍼의 리턴 주소 저장 공간과 프레임 포인터 공간 사이에 캐너리 워드를 삽입하고 함수 리턴시 캐너리 워드가 변경되었는지 조사하여 캐너리 워드가 변경되었으면 버퍼 오버플로우가 발생한다고 판단을 하게된다. 그러나 StackGaurd은 스택안에 존재하는 캐너리 워드 값을 알아내어 존재하는 캐너리 워드값과 같은 값을 기록하거나 캐너리 워드부분을 건너띄는 경우 버퍼 오버플로우 공격을 탐지할 수 없다.

1) 본 연구는 학술진흥재단 과제(2003-002-d00312)의 지원에 의해 수행되었음.

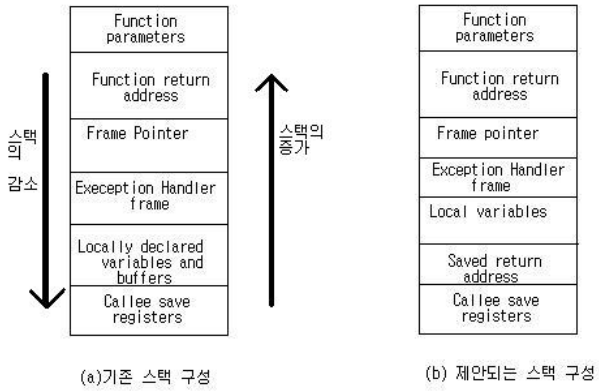

그림 2 기존 스택과 제안되는 스택 구성

미 Stony Brook 대학의 RAD[3]는 소스 코드가 공개되지 않은 윈도우즈용 프로그램의 오버플로우 공격을 막기 위한 도구로서 이진파일을 수정하는 방식을 취한다. 여기서는 .RAR이라는 리턴 주소 저장 공간을 만들어 함수 리턴시 스택의 리턴 주소와 .RAR 공간에 저장되어 있는 리턴 주소가 일치하는 것이 있는지 찾아내 만일 있다면 리턴 주소가 변경되지 않았다고 판단을 하게 된다.

그러나, RAD는 .RAR 공간에서 리턴 주소를 저장하고 일치하는 리턴 주소가 있는지 검색을 해야 하고 이러한 과정은 사용자가 정의하는 루틴에 의하여 실행되어야 하므로 리턴 주소 관리에 대한 오버헤드가 발생한다. 또한 .RAR 공간을 포시하기 위해 사용되는 VirtualProtect()함수에 의한 속도의 감소 또한 나타난다.

```
push    ebp                 mov     eax, dword[ebp+4]
mov     ebp, esp            cmp     eax, dword[ebp-4]
sub     esp, 040            jmp     02
mov     eax, dword[ebp+4]   jne     09
mov     dword[ebp-4], eax   mov     esp, ebp
call    004010A6            pop     ebp
                            ret
```

(a) Stack Prologe　　(b) Stack Epiloge

그림 3 스택 프롤로그와 에필로그

기타 국내에서도 RAD와 유사한 방식으로 이진파일을 재작성 하는 기법에 대한 연구가 진행된 바 있는데[4], 리턴 주소가 코드 부분이 아닌 데이터 부분일 경우에 대해 버퍼 오버플로우 공격이 일어났다고 판단을 하게 된다. 하지만 Lisp나 Object C등의 Nested 함수 또는 예외 핸들러의 경우 스택에서 실행되어야 하므로, 이와 같이 리턴 주소를 코드 부분으로 한정시키는 것은 문제가 있다.[5]

3. 제안 프로그램

3.1 구성

본 논문에서는 커널 및 라이브러리와 소스 코드가 공개되지 않는 윈도우즈에서 버퍼 오버플로우 공격을 막는 것을 목적으로 하는 효율적인 기법을 제안한다. RAD와 유사하게 이진 실행 파일을 이용하게 되나 리턴 주소 사본을 관리하는 오버헤드를 줄인다. 즉, RAD이 별도의 저장 공간을 이용하는 반면 여기서는 그림 2와 같이 스택 내에 리턴 주소를 복사하여 저장하게 된다. 이 방법은 또한 버퍼 오버플로우 공격시 문자열 버퍼가 리턴 주소가 존재하는 방향으로 증가하는 방향성을 이용하여 리턴 주소의 사본의 위치를 스택의 윗부분에 둠으로써 안전성을 확보한다.

제안하는 기법은 다음과 같은 세가지 절차로 코

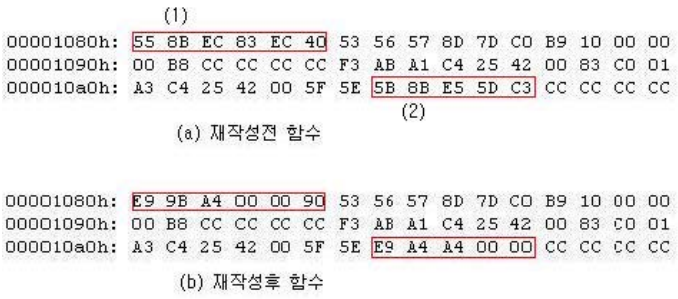

그림 4 기존 함수 및 재 작성후 함수의 바이너리 구조

드를 재작성 한다.

1. 역어셈블[6]
2. 함수의 시작 및 끝을 조사
3. 함수의 시작과 끝을 프롤로그와 에필로그로 이동하는 코드로 재작성

프롤로그는 그림 3과 같이 JMP 명령어에 의하여 덮어 쓰인 명령어와 리턴 주소를 리턴 복사 공간으로 복사하는 부분으로 이루져 있으며, 에필로그는 리턴 주소와 리턴 복사 공간의 값을 비교하여 같을 경우 스택을 비우고 원래의 함수로 리턴하며 다를 경우 에러를 일으키며 프로그램은 종료한다.

재작성은 그림 4와 같이 기존 이진 코드에서 존

재하는 6바이트 크기의 코드를 프롤로그로 이동하는 명령과 에필로그로 이동하는 명령으로 각각 대체한다. 그림 4의 1에서 스택의 크기를 증가 시키는 부분이 프롤로그 부분으로 이동하며 그림 4의 2에서 스택 크기를 감소 시키고 호출 함수로 되돌아가는 부분이 에필로그로 이동하는 것을 볼 수 있다.

3.2 특징

본 논문에서 제안하는 기법은 스택에 리턴 주소 복사 공간을 생성한다. 따라서, 리턴 주소사본의 관리를 특별한 공간이나 함수를 이용하지 않으므로, RAD등에서 발생하는 복사된 리턴주소의 관리에 소요되는 비용을 최소화 시킬 수 있다. 또한 저장된 리턴 주소 값을 찾을 때 모든 주소 값을 비교 하지 않고 스택에서 한번에 주소 값을 알아 낼 수 있기 때문에 리턴 주소 탐색에 의해 소모되는 시간 또한 줄일 수 있다.

하지만 스택에 저장된 리턴 주소값 사본은 또다시 버퍼오버플로우의 위험성을 갖는다고 볼 수도 있다. 그러나, 버퍼 오버플로우 공격에 이용되는 문자열 기록은 방향성을 가지므로 이것은 문제가 되지 못한다. 즉, 문자열에 기록할 경우에는 메모리 주소가 높은 쪽에서 낮은 쪽으로 순차적으로 기록하므로, 문자열이 들어있는 변수 공간보다 메모리 주소가 높은 곳에 있는 값들은 버퍼 오버플로우 공격에서 안전하다. 이를 이용하여 그림 2와 같이 스택의 범용 레지스터 저장 공간과 변수 공간 사이에 4바이트 크기의 리턴 주소 복사 공간을 만들어 리턴 주소 사본을 저장하게 되고, 따라서 리턴 주소 복사 공간이 버퍼 오버플로우 공격에 대하여 안전성을 보장하게 된다.

3.3 실험

제안된 기법의 성능을 실험하기 위해 AMD 2400+, RAM 512MB, 윈도우즈 2003 환경에서 테스트를 하였으며, 테스트를 위하여 4개의 실행 파일을 만들어 30번의 테스트 후 평균적인 값을 도출하였다. 모든 실행 파일은 1부터 50000000까지 카운트하는 방식을 사용하였다. 첫 번째 실행 파일은 메인 함수에서 i++를 이용하여 50000000까지 증가시키며, 나머지 실행 파일들은 각각 전역 변수를 이용한 void increment(void)와 void increment(int *) 그리고 int increment(int)를 이용한 함수들을 50000000번 실행시켰다. 또한 이러한 함수들의 오버헤드를 알아내기 위하여 다음과 같은 식을 사용하였다.

함수 종류	실행 시간 (ms)	제안된 방법 실행 시간 (ms)	오버헤드 (%)	StackGuard 오버헤드 (%)
i++	172	172	0	0
void increment(void)	1485	1723	16.02	125
void increment(int*)	1508	1691	12.14	69
int increment(int)	1547	1721	11.25	80

표 1 실험 결과

$$overhead = \frac{function's\,additional\,cost}{original\,function\,overhead} X100$$

식 1 오버헤드 계산 식[4]

앞서 2장에서 언급한 StackGuard와 성능 비교를 한 결과는 표 1과 같다[2]. StackGuard보다 최대 1/10정도 수준인 15%정도의 오버플로우가 발생하였으며 평균 12%정도의 오버플로우가 발생하는 것을 볼 수 있었다. 이는 StackGuard의 경우 공격자가 캐너리 워드 값을 추측하는 것을 방지하기 위해 함수가 호출될 때마다 캐너리 워드를 난수화 시킴에 따라 발생하며 또한 난수화하여 얻어진 캐너리 워드를 유지하고 비교하기 위하여 배열등의 특별한 자료 공간을 생성하고 유지 관리함으로 인해 오버헤드가 발생하지만, 본 논문에서 제안된 방법은 버퍼 오버플로우 공격이 발생했는지에 대한 정보를 리턴 주소를 직접 이용하기 때문에 난수화 과정에서 오는 오버헤드가 없으며, 최초 스택 생성시와 제거시 리턴 주소 복사 주소 공간이 동시에 생성되고 제거되기 때문에 주소 복사 공간의 유지를 위해 발생하는 오버헤드가 최소화되고, 메모리 접근 횟수의 격감에 의하여 StackGuard에 비하여 오버헤드가 상당히 적게 발생

2) 원래 RAD와 비교하고자 시도하였으나, RAD로 변환된 실행 파일들이 가지는 오류로 부득이하게 StackGuard와의 비교 결과를 얻는 것으로 바꾸었으며, 현재 RAD 저자와 연락을 취하고 있다.

한 것으로 추측된다.

또한, 테스트한 파일들은 실제 함수 내에서 i++ 라는 단순한 작업만을 하였을 때 나온 오버헤드이며 함수의 코드 길이가 충분히 길어지면 오버헤드 발생률이 적어질 것으로 기대된다.

4. 결론 및 향후 연구 과제

현재 버퍼 오버플로우 공격의 방어는 그 심각한 위험성 때문에 많은 연구가 되고 있지만 방어에 의한 오버헤드의 발생이 큰 문제로 작용하고 있었다.

본 논문은 이진 코드를 재작성 하여 스택의 리턴 주소 사본을 지역변수 아래 부분에 두고 함수 반환 시 비교 검사를 하는 것으로써, 소스코드가 없는 경우에도 버퍼 오버플로우 공격을 막는 동시에 오버헤드를 줄일 수 있는 방법을 제안하였다.

RAD 및 기타 기법들과의 비교를 위한 실험을 추진 중이며, 이진 파일 역어셈블을 통한 구현을 개선하여 통합된 자동화 도구로 개선할 계획에 있다.

5. 참고문헌

[1] Crispin Cowan, "Software Security for Open-Source System", IEEE Security & Privacy, Jan. 2003

[2] Crispin Cowan, Calton Pu, Dave Maier, Heather Ginton, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle and Qian Zhang, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", In proceeding of the 7th USENIX Security Conference. 1998

[3] M. Prasad and T. Chiueh, "A Binary Rewriting Defense against Stack-based Buffer Overflow Attacks", In proceedings of the IEEE Symposium on Security and Pricvacy, May 1996

[4] 김종의, 이성욱, 홍만표, “버퍼오버플로우 공격 방지를 위한 컴파일러 기법”, 2002 정보처리학회논문지 9권 4호 p.453-458

[5] A. Baratloo, N. Singh and T. Tsai, "Transparent run-time defense against stack smashing attacks", In proceedings of USENIX Annual Technical Conference, June 2000

[6] 조상, Windows disassembler, http://www.geocities.com/mysimpc/