가상화 환경에서의 버퍼 오버플로우 공격 방어 시스템

권용휘, 박능수
건국대학교 컴퓨터공학부

Buffer overflow detection system on Virtual Machine

Yonghwi Kwon, Neungsoo Park
Department of Computer Science and Engineering, Konkuk University

Abstract. 버퍼 오버플로우 공격은 그 방식이 간단하고 효과적이기 때문에, 오랜 시간 동안 널리 사용되고 있는 소프트웨어 공격 방법 중 하나이다. 오랜 시간 동안, 버퍼 오버플로우 공격을 방어하는 방법들에 대한 다양한 기법들이 제안되었지만, 여전히 많은 소프트웨어들이 버퍼 오버플로우 공격의 위협에 노출되어 있다. 이는 대부분의 소프트웨어 기반의 버퍼 오버플로우 공격에 대한 방어 기법들은, 적용 대상 소프트웨어에 대한 재 컴파일을 필요로 하고, 하드웨어 기반의 방어 기법들은, 추가적인 적용 및 업데이트 비용이 발생하여 현실적으로 적용이 어렵기 때문이다. 이러한 문제를 해결하고자, 본 논문에서는 버퍼 오버플로우 공격에 대한 가상화 환경에서의 방어 시스템을 제안한다. 본 방어 시스템은, 보호하고자 하는 소프트웨어에 대한 수정 없이 전체 시스템에 적용할 수 있으며, 버퍼 오버플로우 공격으로부터 효과적으로 시스템을 방어할 수 있다.

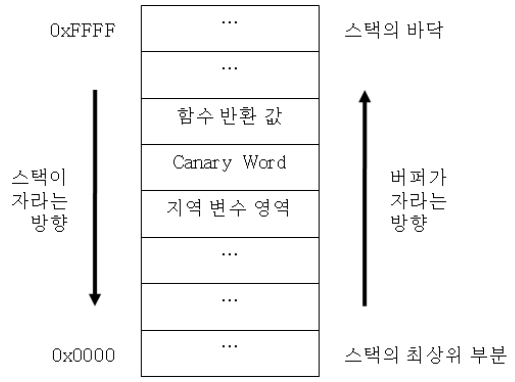
1. Introduction

C프로그래밍 언어와 관련된 버퍼 오버플로우 공격에 대한 문제점이 지적된 이후로, 버퍼 오버플로우 공격은 1988년 Morris 웜 바이러스를 통하여 처음으로 그 심각성이 대중에 알려졌다. 그 이후, 2001년의 Code Red 웜 바이러스와 2003년의 SQL Slammer 웜 바이러스를 통하여 버퍼 오버플로우 공격은 강력하고 치명적인 소프트웨어 공격 방법으로 알려졌다. 지난 수년간, 버퍼 오버플로우 공격을 방어하기 위하여 수많은 방법이 제시되었지만, 아직도 많은 소프트웨어들이 버퍼 오버플로우 공격에 노출되어있다. 이는 지금까지 발표된 대부분의 버퍼 오버플로우 공격에 대한 방어 방법들이 재 컴파일을 필요로 하거나, 추가적인 비용을 필요로 하기 때문이다. 이러한 문제를 해결하고자, 본 논문에서는 방어 대상이 되는 소프트웨어에 대한 변형 없이 적용할 수 있는 가상화 기반의 버퍼 오버플로우 방어 시스템을 제안하였다. 본 논문에서 제안한 방어 시스템은 기존 소프트웨어 및 운영체제에 대한 수정 없이 적용이 가능하며, 알려진 여러 방법의 버퍼 오버플로우 공격에 대한 방어가 가능하였다.

2. Related Work

지금까지의 버퍼 오버플로우 공격에 대한 방어 방법은 크게 소프트웨어 방식과 하드웨어 방식으로 나눌 수 있다. 소프트웨어 방식은, 컴파일 시점에서 함수의 시작과 끝 부분에 공격 탐지를 위한 추가적인 명령을 삽입하며, 대표적인 방법으로 StackGuard[1]와 StackShield[2]가 있다. 하드웨어 방식으로는 함수 호출과 복귀에 사용되는 명령들을 확장한 SmashGuard[3]가 있다.

StackGuard는 버퍼 오버플로우 공격에 대한 방어 방법 중 가장 잘 알려진 방법 중 하나이다. [그림 1]과 같이, StackGuard는 스택에 Canary Word를 함수의 복귀 주소의 다음에 추가한다. 대부분의 버퍼 오버플로우 공격이, 함수의 복귀 주소 보다 낮은 주소에 위치한 지역 변수의 버퍼가 넘쳐서 발생하기 때문에, 함수의 복귀 주소가 변경될 때, Canary Word도 함께 변경되는 현상을 이용한 방법이다.



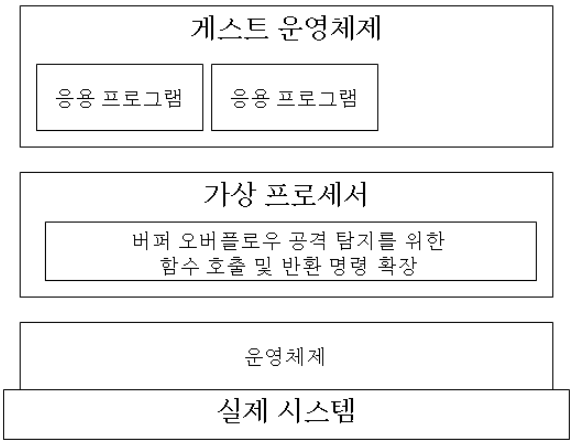
[그림 1] StackGuard의 Canary Word

StackShield는 함수 호출 명령이 실행된 직후, 저장된 함수 복귀 주소를 안전한 장소에 복사하여 저장하고, 함수가 종료될 때 저장된 복귀 주소와 현재 스택에 존재하는 복귀 주소를 비

교하여 버퍼 오버플로우 공격을 탐지한다. StackShield는 효과적인 버퍼 오버플로우 공격 방어 방법으로, 본 논문에서 제시하는 시스템과 기본적으로 유사한 방식을 사용하고 있다. 하지만, StackShield를 적용하기 위해서는 소프트웨어의 재 컴파일이 필요하므로, 실질적으로 적용하기에는 그 범위가 제한적이다.
SmashGuard는 하드웨어 기반의 버퍼 오버플로우 방어 시스템으로, StackShield와 유사한 방식을 사용하고 있다. SmashGuard는 함수 호출 명령과 복귀 명령을 확장하여, 함수 호출 명령이 실행될 때, 함수 복귀 주소를 고유의 하드웨어 스택에 저장하고, 함수 복귀 명령이 실행될 때, 저장된 복귀 주소와 실제 시스템의 스택상에 존재하는 복귀주소를 비교하여 공격을 탐지한다. SmashGuard는 버퍼 오버플로우 공격의 방어에 적합한 시스템이지만, 적용하기 위해서는 하드웨어를 구축하여야 하는 추가적인 비용이 발생하고, SmashGuard자체를 다른 시스템으로 이식하거나, 업데이트 하기 어렵다는 문제가 있다.

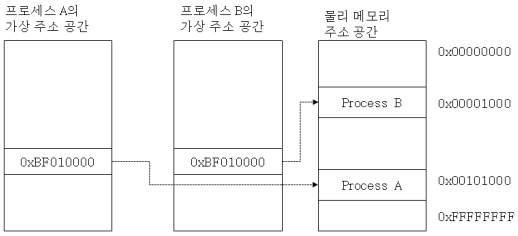
3. Implementation

본 논문에서 제안하는 가상화 기반에서 버퍼 오버플로우 공격을 방어하는 시스템의 전체적인 구조는 [그림 2]와 같다.



[그림 2] 시스템의 전체적인 구조

가상 머신에 포함된 가상 프로세서의 명령을 확장하는 방법을 사용하여, 버퍼 오버플로우 공격을 탐지하는 방식으로 구현되었으며, StackShield와 같이, 함수가 호출될 때 복귀 주소를 안전한 공간에 저장하고, 함수가 종료될 때, 저장된 복귀 주소와 스택에 존재하는 복귀 주소를 비교하여 공격을 탐지하는 방식을 사용한다. 가상화 환경에서 본 시스템을 구현하기 위하여 반드시 해결하여야 할 문제로는 서로 다른 실제 메모리를 가리키는 같은 가상 주소를 가지는 명령들에 대한 처리와 setjmp와 longjmp로 인한 비 대칭적인 함수 호출 및 복귀에 대한 문제[4]가 있으며, 본 시스템에서는 이러한 문제들을 아래와 같이 해결하고 있다.
먼저, 80x86프로세서는 가상 주소 공간을 사용하여 각 프로세스의 주소 공간을 독립적으로 유지한다. 즉, [그림 3]와 같이, 프로세스A의 0xBF010000에 있는 값과, 프로세스B의 0xBF010000에 있는 값이 가상 주소는 동일하지만, 서로 다른 물리 메모리 위치로 연결되어 있다.



[그림 3] 가상 주소 공간과 실제 주소 공간

StackGuard나 StackShield와 같이 컴파일시에 적용되는 방법들은, 가상 주소 공간에 대한 고려가 필요가 없으나, 가상화 환경에서는 서로 다른 가상 주소 공간을 가지는 소프트웨어를 다루어야 하므로, 가상 주소 공간에 대한 고려가 필요하며, 본 논문에서 제시한 시스템에서는 각 함수 복귀 주소를 저장할 때, 프로세스 별로 구분하여 저장하는 방식을 사용하여 해결하고 있다.
또 다른 문제점인 setjmp와 longjmp는 함수 종료시, 스택에 가장 마지막으로 입력된 함수 복귀 주소를 사용하는 일반적인 형태의 함수 호출 흐름을 바꾸는 기능을 한다. 이는, 시스템 전체의 입장에서 버퍼 오버플로우 공격 방어 시스템을 구현할 때 해결하여야 할 문제들 중 가장 복잡한 부분 중 하나이며, 본 논문에서 제시한 시스템에서는 함수 복귀 주소를 저장하고 비교할 때, 비교 대상이 되는 함수 복귀 주소에 대한 스택에서의 위치를 함께 저장하여 비교하는 것으로 해결하고 있다. 기존의 다른 방어 시스템들은, setjmp와 longjmp에 대한 문제를, 일치하는 복귀 주소가 나타날 때까지 검색을 지속하는 방법을 사용하고 있는데, 이 방법은 본 논문에서 사용한 방법보다 오버헤드가 크고, 서로 다른 스택의 위치에서 사용된 주소들을 구분 없이 사용하기 때문에, 거짓 음성 문제를 야기할 수 있다.

4. Evaluation

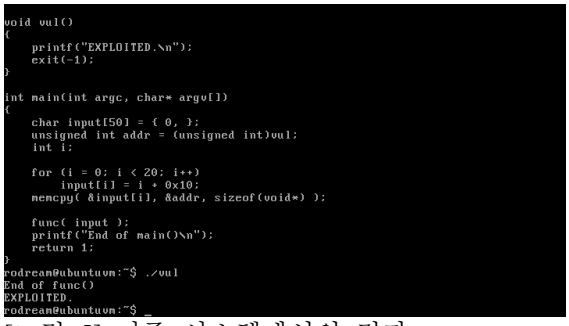
본 논문에서 제안한 가상화 환경에서의 버퍼 오버플로우 방어 시스템이 실질적인 버퍼 오버플로우 공격을 방어할 수 있는지 확인하기 위하여, [그림 4]와 같은 코드를 사용하여 시스템

의 유효성을 검증하였다.

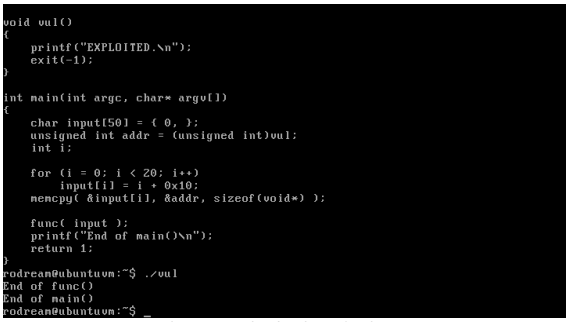
```
1   #include <stdio.h>
2   #include <string.h>
3   #include <stdlib.h>
4
5   void func(char* buf)
6   {
7       char var[16];
8       strcpy( var, buf );
9       printf("End of func()\n");
10  }
11
12  void vul()
13  {
14      printf("EXPLOITED.\n");
15      exit(-1);
16  }
17
18  int main(int argc, char* argv[])
19  {
20      char input[50] = { 0, };
21      unsigned int addr = (unsigned int)vul;
22      int i;
23
24      for(i = 0; i < 20; i++)
25          input[i] = i + 0x10;
26      memcpy( &input[i], &addr, sizeof(void*) );
27
28      func( input );
29      printf("End of main()\n");
30      return 1;
31  }
```

[그림 4] 테스트 프로그램

[그림 4]의 프로그램에서는, 일반적인 버퍼 오버플로우 공격에 취약한 함수인 func에 공격 문자열을 강제로 주입하였다. 또한, [그림 4]의 소스코드의 26번째 줄의 코드는 Return-to-libc[5]형태의 공격을 시도하고 있다는 것을 알 수 있다. [그림 4]의 프로그램을 본 시스템을 사용하지 않은 상태에서 동작시킨 결과와, 본 시스템을 사용한 상태에서 동작시킨 결과는 각각 [그림 5]와 [그림 6]과 같다.



[그림 5] 기존 시스템에서의 결과



[그림 6] 본 시스템에서의 결과

[그림 6]에서의 결과와 같이, 본 논문에서 제안하는 시스템은 효과적으로 버퍼 오버플로우 공격을 탐지하고 방어할 수 있음을 확인하였다.

5. Conclusion

본 논문에서 제안한 가상화 환경에서의 버퍼 오버플로우 방어 시스템은 기존 소프트웨어 기반의 공격 방어 시스템의 단점인 소프트웨어의 재 컴파일 요구에 대한 문제를 해결하였고, 하드웨어 기반의 공격 방어 시스템의 단점인, 추가 하드웨어 비용 및 업데이트를 위한 수정 문제 없이, 공격을 방지할 수 있었다. 본 시스템은 방어하고자 하는 소프트웨어에 대한 사전 지식 없이 공격에 대한 방어가 가능하므로, 다양한 운영체제에 대한 공격 방어가 가능하였다. 본 시스템에 추가적으로 연구할 부분으로는, 성능 개선에 대한 부분과 다른 소프트웨어 공격에 대한 방어 기능을 추가하는 부분이 있다.

References

[1] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang, “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks,” *Proc. Seventh USENIX Security Conf.*, pp. 63-78, Jan. 1998.

[2] Vendicator, "StackShield: A stack smashing technique protection tool for Linux," Jan. 08, 2000. http://www.angelfire.com/sk/stackshield/

[3] H. Ozdoganoglu, C. E. Brodley, T. N. Vijaykumar, B. A. Kuperman and A. Jalote, "SmashGuard: A hardware solution to prevent security attacks on the function return address," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1271 - 1285, November, 2003.

[4] Stevens, W. Richard, Advanced Programming in the UNIX Environment, Addison-Wesley, 1992.

[5] Solar Designer. “return-to-libc" attack. Bugtraq, Aug. 1997.