Journal of the Korea Academia-Industrial cooperation Society
Vol. 13, No. 10 pp. 4794-4800, 2012

http://dx.doi.org/10.5762/KAIS.2012.13.10.4794

버퍼오버플로우 공격 방지를 위한 리턴주소 스택

조병태[1], 김형신[1*]
[1]충남대학교 컴퓨터공학과

Return address stack for protecting from buffer overflow attack

Byungtae Cho[1] and Hyungshin Kim[1*]
[1]Department of Computer Science and Engineering, Chungnam National University

요 약 버퍼오버플로우 취약점을 이용한 공격기법을 예방하기 위해서 그 동안 많은 연구가 진행되었으며, 여러 가지 탐지기술과 보안패치 등의 노력이 진행되었음에도 불구하고, 여전히 시스템 보안에 있어 가장 중요한 이슈로 지목되고 있는 이유는 아직까지도 프로그램 개발 시 버퍼오버플로우에 취약한 함수와 라이브러리를 이용하여 프로그램을 개발하고 있다는 점과 실제 버퍼오버플로우 취약점이 노출되어 시스템이 공격받은 후에 패치가 이루어진다는 점이다. 본 연구에서는 버퍼오버플로우 예방을 위한 하드웨어적인 보호기법으로 캐시레벨 기반의 리턴주소 스택을 이용한 버퍼오버플로우 보호기법에 대한 연구를 진행하였다. 제안한 스택구조의 성능평가는 SimpleScalar 시뮬레이터를 이용하여 진행하였으며, 본 연구에서 제안한 보호기법은 기존 버퍼오버플로우 취약점을 완벽히 보호할 수 있으며, 일부의 프로그램을 제외한 대부분의 프로그램에서 성능상의 차이가 발견되지 않은 시스템을 개발하였다.

Abstract Many researches have been performed to resist buffer overflow attacks. However, the attack still poses one of the most important issue in system security field. It is because programmers are using library functions containing security hole and once buffer overflow vulnerability has been found, the security patches are distributed after the attacks are widely spreaded. In this paper, we propose a new cache level return address stack architecture for resisting buffer overflow attack. We implemented our hardware onto SimpleScalar simulator and verified its functionality. Our circuit can overcome the various disadvantages of previous works with small overhead.

Key Words : Buffer overflow, Rreturn address stack, Cache level stack, Stack security

1. 서론

시스템의 발전과 더불어 소프트웨어의 복잡도와 크기가 증가함에 따라, 소프트웨어에 포함된 잠재적 위협요인인 소프트웨어의 취약점도 점차 증가하고 있다. 이러한 소프트웨어의 취약점 중 가장 널리 이용되는 기법은 버퍼오버플로우 취약점을 이용한 시스템 공격기법이다[1, 2]. 버퍼오버플로우 취약점을 이용한 공격기법은 공격자의 악의적인 코드를 메모리에 삽입하여, 정상적인 프로그램의 흐름이 아닌 공격자의 코드로 실행되도록 프로그램의 흐름을 변경하는 행위이다. ICAT의 통계에 의하면[3], 컴퓨터 시스템에 위협을 가하는 바이러스나, 해킹 등의 주요 공격원인으로는 바이러스 제작자나, 해커들이 즐겨 이용하는 취약점을 이용한 방법으로 70% 이상을 버퍼오버플로우로 지적했으며, CERT 리포트[4]에 따르면, 버퍼오버플로우 취약점과 관련되어 발간된 CERT의 보고서는 전체의 50%에 이른다.

버퍼오버플로우 공격기법은 1970년대부터 시스템에 대한 위협을 가해왔다. 1988년 Morris 웜은 finger daemon의 버퍼오버플로우 취약점을 이용한 서비스 거부 공격(DOS: Denial Of Service)로 피해를 끼쳤으며[5], Code Red 웜은 서버에서 특정 모듈의 버퍼오버플로우 취약점을 이용한 공격으로 당시 Computer Economics는 Code Red 웜으로 인한 피해액을 2억 6천만 달러로 추산

*Corresponding Author : Hyungshin Kim
Tel: +82-10-2085-3860 email: hyungshin@cnu.ac.kr
접수일 12년 07월 23일 수정일 (1차 12년 08월 22일, 2차 12년 09월 07일) 게재확정일 12년 10월 11일

했다[6].

버퍼오버플로우 취약점을 이용한 공격기법을 예방하기 위해서 그 동안 많은 연구가 진행되었으나, 여전히 시스템 보안에 있어 가장 중요한 이슈로 지목되고 있다. 그 이유는 아직까지도 프로그램 개발 시 버퍼오버플로우에 취약한 함수와 라이브러리를 이용하여 프로그램을 개발하고 있다는 점과 이런 버퍼오버플로우 취약점을 잠재적으로 내포한 소프트웨어들이 사용자들에게 배포되고 있으며, 실제 버퍼오버플로우 취약점이 발견되어 이미 시스템이 공격받은 후에 패치가 이루어진다는 점 때문이다.

이러한 문제점을 극복하기 위해 제안된 하드웨어적인 보호기법은 상대적으로 소프트웨어적인 해결기법에 비해 적은 오버헤드를 갖는다. 또한 기존의 소프트웨어의 수정, 재 컴파일이 필요하지 않아 레거시 코드에도 모두 적용될 뿐만 아니라, 패치 등을 통해 수정되지 않은 소프트웨어를 모두 예방할 수 있다는 장점을 가지고 있다.

본 연구에서는 버퍼오버플로우 예방을 위한 하드웨어적인 보호기법으로 캐시레벨 기반의 리턴주소 스택을 이용한 버퍼오버플로우 보호기법에 대한 연구를 진행하였다. 제안한 버퍼오버플로우 보호기법은 SimpleScalar[7]를 이용하여 성능을 검증하였다. 본 연구에서 제안한 보호기법은 기존 버퍼오버플로우 취약점을 완벽히 보호할 수 있으며, 대부분의 프로그램에서 성능상의 차이가 거의 없었다.

이 논문의 구조는 다음과 같다. 2장에서는 지금까지 버퍼오버플로우를 예방하기 위해 진행된 관련 연구를 정리하였으며, 3장은 캐시레벨 기반의 하드웨어 보호기법인 리턴주소 스택의 시스템 설계와 동작과정에 대해 기술하였다. 4장에서는 본 논문에서 제안한 시스템에 대한 성능평가를 기술하고, 5장에서 결론을 맺는다.

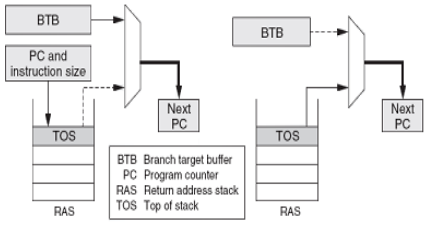
2. 배경지식

2.1 리턴 주소 스택

일반적으로 프로세서는 인스트럭션 캐시로부터 가져온 명령어를 BTB(Branch Target Buffer)를 거쳐 다음 실행할 명령어의 주소로 결정한다. 그런데 함수의 호출 및 반환 같은 경우, 일반적으로 함수는 LIFO(Last In, First Out) 구조로 실행되기 때문에, 이 정보를 미리 가지고 있다면 분기예측기의 정확도도 높일 수 있을 뿐만 아니라, 분기예측시간도 크게 단축할 수 있다. 즉 프로세서가 함수를 호출하는 명령어를 실행할 때, 프로세서는 함수의 반환주소를 리턴주소 스택에 저장하여, 함수가 반환될 때 이 리턴주소를 참조하여 파이프라인의 성능을 높인다. 리턴주소 스택은 함수의 호출과 반환시 리턴주소 정보를 모두 유지하고 있으며, 이 정보는 프로세서에 의해 함수의 호출과 반환 명령어에 의해서만 갱신되기 때문에 버퍼오버플로우 취약점을 이용한 시스템 공격에도 이곳에 저장된 리턴주소는 변조되지 않는다.

이를 고려하여 개선된 SRAS(Secure Return Address Stack)의 동작과정은 [그림 1]과 같다. 리턴주소 스택을 지원하는 프로세서에서는 명령어 캐시로부터 가져온 명령어가 함수 호출 명령인 경우, 현재 실행중인 명령어의 다음 명령어 주소, 즉 프로그램 카운터를 리턴주소 스택에 저장한다.

리턴주소 스택은 간단하면서, 버퍼오버플로우 공격의 리턴주소 변조를 완벽히 보호하지만, 실제 아키텍처에서는 몇 가지 문제점으로 인해 사용하지 않는다. 리턴주소 스택의 문제점 중 하나는 리턴주소 스택의 오버플로우 현상으로, 재귀호출 함수와 같은 경우, 연속적으로 함수가 호출되기 때문에 쉽게 리턴주소 스택의 크기를 초과해 버리게 되며, 이 같은 경우 더 이상의 함수를 호출하게 되면 이전에 저장되어 있는 리턴주소들이 훼손된다.



[그림 1] 보안 리턴주소 스택
[Fig. 1] Secure Return address stack

리턴주소 스택의 다른 문제점은 Non-LIFO 함수 실행 흐름이 발생하는 리턴주소 스택의 오작동이다. 일반적으로 함수 호출은 LIFO의 구조를 갖지만, 반드시 LIFO 구조로 실행되는 것은 아니다. C++의 예외처리나 setjmp/longjmp API는 비 LIFO의 구조로 실행되는 전형적인 예로, 이 경우 리턴주소 스택은 잘못된 리턴주소를 반환한다.리턴주소 스택에서 발생하는 문제들을 해결한 쉐도우 상태 레지스터(SSR: Shadow State Register)를 이용한 하드웨어 스택 관리엔진(RASE: Return Address Stack Engine)은 앞서 열거한 문제점들을 해결하였다[18]. 그러나 프로세서의 분기예측 오류가 발생하는 경우 리턴주소 스택이 오염되는 문제를 방지하기 위해서는 파이프라인 하드웨어의 상태를 지속적으로 확인해야 한다. 그

결과, 쉐도우 상태 레지스터(SSR)의 빈번한 업데이트로 인한 오버헤드가 크며 복잡한 하드웨어 구조를 갖는 문제점을 가지고 있다. 또한 RASE는 Nonlocal jump의 문제점을 해결하기 위해, 컴파일러를 수정해야 하는 단점도 가지고 있다.

지금까지 열거된 연구들이 프로세서 내부에 보호유닛이 설계된 것에 비해, SCache는 캐시레벨에서 버퍼오버플로우 공격을 예방한다[19]. SCache는 함수를 호출하여 현재 프로그램 카운터 값을 메모리에 저장할 때 캐시에 값을 복제하여 함수 호출 종료시 메모리 스택에 저장된 리턴주소와 캐시에 복제된 원본 값을 비교하여 리턴주소의 변조여부를 판단한다. SCache에 저장되는 리턴주소는 프로세서의 동작과는 관계없이 저장되기 때문에, 앞서 제기한 리턴주소 스택의 문제점을 모두 해결할 수 있으며, 구조도 매우 단순하다. 그러나 SCache는 리턴주소 저장시 복제 값을 저장함으로써, 캐시의 히트타임이 감소하고 캐시공간을 낭비하기 때문에 캐시 부적중률이 증가한다. 또한 리턴 주소를 조작할 때, 스택 메모리의 리턴 주소와 복제된 원본 리턴주소까지 모두 조작하는 경우 버퍼오버플로우 공격에 취약한 단점을 갖고 있다.

본 연구는 기존 연구에 비해 낮은 하드웨어 복잡도를 가지면서 버퍼오버플로우를 예방하는 하드웨어 보호 기법을 개발하였으며, [표 1]은 기존 연구와 본 연구에서 제안하는 연구결과를 비교한 것이다. 우리가 개발한 리턴주소 스택은 기존의 리턴주소 스택 하드웨어의 문제점으로 지적된 setjmp/longjump 와 같은 비정상적인 함수 호출과 리턴주소 스택의 오버플로우, 그리고 프로세서의 추측실행을 모두 지원하며, 컴파일러나 라이브러리의 수정이 필요하지 않다.

[표 1] 기존 연구와의 비교
[Fig. 1] Comparison of our method

	라이브러리 수정	완벽히 보호하는가?	코드 사이즈 증가	성능 저하 발생	리턴 주소 오버플로우 지원	setjmp/longjmp 허용	추측 실행 허용
StackGuard	○	○	○	○	No	No	No
StackGhost	×	×	×	×	No	No	No
SRAS	×	○	×	×	×	×	×
SCache	×	×	×	×	No	No	No
RASE	○	○	×	×	○	○	○
제안 연구		○	×	×	○	○	○

2.2 관련연구

시스템에서 발생하는 버퍼오버플로우를 예방하기 위해 현재까지 진행된 연구는 정적 분석을 통한 보호기법(Static software based techniques)과 프로그램을 실시간으로 감시하며 버퍼오버플로우를 탐지 및 예방하는 동적 보호기법(Dynamic software techniques), 그리고 하드웨어 기반의 보호기법(Hardware protection)으로 구분할 수 있다.

정적분석을 통한 보호기법은 코드 상에서 개발자가 직접 혹은 자동화된 도구를 이용하여 보안상 취약한 부분을 찾고 해당 부분을 수정한다. 일반적으로 정적분석 작업은 개발자가 프로그램 개발 도중에 수작업으로 하는 방법과, lint와 같은 자동화된 도구를 이용한 방법이 있다[10]. 그러나 정적분석 기법은 실제 프로그램을 실행하지 않은 상태에서 코드 분석을 통해 취약성을 찾기 때문에 정확도가 낮고 부정확한 탐지의 가능성을 배제할 수 없다[11].

동적 보호기법은 정적분석을 통한 보호기법과는 달리 컴파일된 바이너리 코드를 수정하거나, 프로그램의 실행시 실시간으로 버퍼오버플로우를 탐지 및 보호하는 기법이다. 이 기법은 프로그램 코드에 리턴주소를 체크하는 루틴을 삽입하거나[12], 별도의 모니터 프로그램을 이용하여 시스템 콜의 비정상적인 흐름을 탐지하거나, 프로세서의 레지스터 값을 확인한다[13]. 그러나 동적 보호기법은 종종 프로그램의 재 컴파일을 필요로 하기 때문에, 프로그램의 크기가 증가하고, 실행할 명령어의 개수가 증가하여 전체 프로그램 실행시간이 길어지며, 레거시 코드(legacy code)에는 적용할 수 없는 단점을 가지고 있다. 또한 실시간 탐지로 인해 시스템의 전체적인 성능저하와 에너지 낭비를 가져온다.

하드웨어 기반의 보호기법은 하드웨어 회로를 통해 버퍼오버플로우를 예방하며, 소프트웨어적인 보호기법에 비해 낮은 성능 오버헤드와 에너지 소모를 갖는다. 또한, 소프트웨어적인 보호기법은 적용된 특정 프로그램에 한해 보호되는 반면, 하드웨어 기반의 보호기법은 실행되는 모든 프로그램이 보호할 수 있다.

버퍼오버플로우와 같이 소프트웨어의 특정 취약점을 예방하는 하드웨어적인 기법은 AMD의 NX(Non eXecution) 플래그와 같이 동적 보호 기법에서 제안된 방법을 하드웨어로 구현하거나[36], 리턴주소를 위한 하드웨어 스택(RAS: Return Address Stack) 등이 있다[8, 14]. AMD의 NX 플래그는 하드웨어차원에서 프로그램을 실행할 명령어가 텍스트 세그먼트가 아닌 이외의 영역에 있는 프로그램 코드는 실행을 하지 않는다. 이는 일반적인 버퍼오버플로우 공격으로부터 시스템을 보호하는데 매우 효과적이나, RTL(Return To Libc)공격처럼 텍스트

세그먼트 영역의 코드를 악의적으로 활용하는 공격에는 아무 효과가 없다[16].

일부 슈퍼스칼라 프로세서는 리턴주소의 무결성을 검사하기 위한 하드웨어 모듈인 리턴주소 스택(RAS:Return Address Stack)을 가지고 있다. 리턴주소 스택은 원래 파이프라인을 지원하는 프로세서에서 함수가 반환될 때 분기예측기의 정확도를 높이기 위해 제안된 방법이고, SRAS(Secure Return Address Stack)는 이를 개선하여 버퍼오버플로우와 같은 리턴주소를 변조한 공격을 예방하기 위한 방법이다[17].

3. 캐시수준 리턴주소 스택

본 장에서는 버퍼오버플로우 취약점을 예방하기 위해 제안한 캐시레벨 기반의 하드웨어 보호기법인 리턴주소 스택에 대해 기술하였다. 본 연구에서 제안하는 리턴주소 스택은 버퍼오버플로우 취약점을 통한 리턴주소의 변조를 예방하기 위해 프로세서와 L1 캐시 사이에서 동작하는 하드웨어 유닛으로써, 함수의 호출과 복귀시 L1 데이터 캐시로 요청하는 리턴주소의 무결성을 검사하여 오버플로우의 발생을 감지한다.

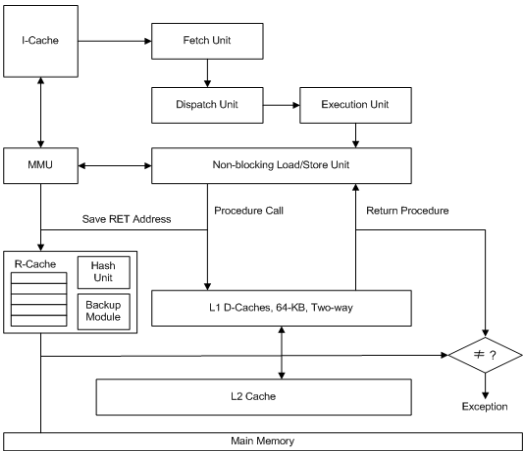
기존의 리턴주소 스택은 프로세서의 내부에 위치하기 때문에, 프로세서의 복잡한 파이프라인 동작에 의해 여러 문제점들이 발견되었다. 이를 해결하기 위해 프로세서의 동작에 영향을 받지 않는 캐시 레벨 기반의 하드웨어 보호기법을 설계하였다. 이와 같은 구조를 채택한 이유는 앞서 리턴주소 스택의 문제점으로 지적한 프로세서의 추측 실행시 발생하는 리턴주소 스택의 훼손을 방지하기 위해, 프로세서의 파이프라인과 캐시 사이에 리턴주소 스택을 위치시킴으로써, 상호간의 동작에 전혀 간섭을 받지 않는 투명한 구조로 만들기 위해서이다.

일반적으로 프로세서는 프로세서의 효율을 높이기 위해 파이프라인을 지원하여, 여러 명령어를 연속적으로 처리하고 있다. 파이프라인 프로세서는 명령어를 명령어 캐시 메모리로부터 가져와 패치하고, 해당 명령어를 분석하고 처리한다. 그러나 파이프라인 프로세서에서 발생하는 해저드를 극복하기 위해 함수의 호출과 복귀, 점프 같은 분기 명령어들은 분기 예측기를 통해 명령어를 예측한 후, 미리 파이프라인에서 명령어를 처리하고, 예측이 잘못된 경우 파이프라인을 초기화시키고 올바른 명령어를 처리한다. 그리고 이러한 예측이 잘못될 경우를 대비하여 명령어가 처리되기 위해서는 프로세서의 완료(Commit)가 필요하다.

기존의 리턴주소 스택에 관한 연구들은 이러한 파이프라인 내부의 동작과정 중 함수의 호출과 복귀시마다 잘못 예측되는 경우 리턴주소 스택을 수정하고, 원래의 데이터를 복귀시키는 등 테이블의 빈번한 업데이트가 발생하지만, 캐시는 프로세서에 투명하기 때문에 프로세서에서 잘못 예측되어 실행되더라도 실제 캐시에는 영향을 미치지 않기 때문에 간단한 구조를 갖게 된다.

그리고 본 논문에서 제안하는 리턴주소 스택은 함수의 호출 및 반환 명령어가 실행 중일 때만 동작하고 일반적인 분기명령어이나 산술·논리 연산을 수행할 때에는 아무 동작도 하지 않기 때문에 리턴주소 스택 참조로 인한 오버헤드는 기존의 제안된 연구와 비교하여 동일하거나, 낮은 수준이다.

그림 2는 캐시레벨 기반 리턴주소 스택 하드웨어의 전체 구조를 나타낸 그림으로 리턴주소 스택은 LIFO 구조의 L1 캐시 메모리로 프로세서와 리턴주소가 저장되는 L1 데이터 캐시 메모리 사이에 위치해 있다. 일반적으로 함수의 호출은 Non-LIFO 실행흐름과 같은 예외적인 상황을 제외하고는 먼저 호출된 함수가 먼저 반환되기 때문에 가장 최근에 호출된 함수가 리턴주소 스택의 가장 높은 위치에 저장된다. 그리고 리턴주소 스택의 크기를 넘어 함수가 연속적으로 호출되는 경우 발생할 수 있는 오버플로우로 인해 리턴주소 스택이 훼손되는 것을 방지하기 위해 백업모듈과 백업된 데이터의 무결성을 검증하기 위한 해시 유닛으로 구성되어 있다.



[그림 2] 캐시레벨 기반 리턴주소 스택 하드웨어의 전체 구조

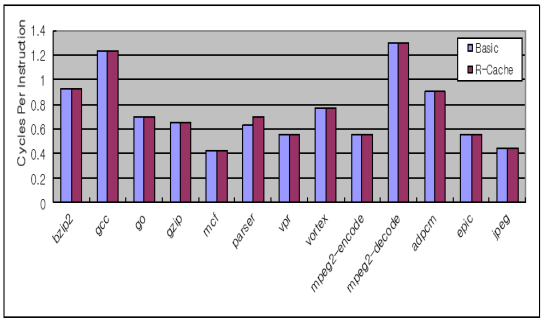
[Fig. 2] Cache-level return address stack architecture

4. 성능평가

본 장에서는 버퍼오버플로우 취약점을 예방하기 위해

제안된 리턴주소 스택을 검증하기 위해 SimpleScalar 시뮬레이터와 SPEC2000[18] 벤치마크를 이용하여 성능평가를 진행하였다. 본 논문에서는 SimpleScalar 중 sim-outorder를 이용하여 테스트를 진행하였다. 본 연구에서의 실험은 Alpha 21264 프로세서의 성능을 기본 지표로 사용하였으며, 리턴주소스택은 32 엔트리를 갖는 R-Cache와 이진 해시트리를 사용하도록 설정하여 실험을 진행하였다.

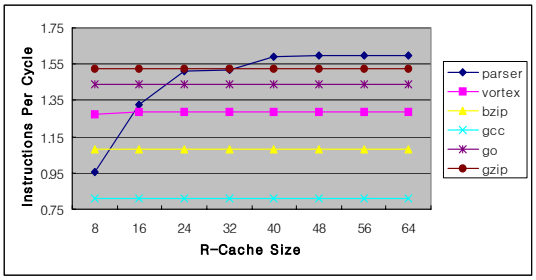
그림 3은 리턴주소 스택 하드웨어에서 SPEC2000 벤치마크를 통해 성능을 측정한 모습이다. 이진 해시 트리의 모든 해시 노드는 메모리에 안전한 곳에 저장되는 정책을 이용하는 것으로 설정되었다. 리턴주소 스택은 함수 호출시 프로세서가 L1 데이터 캐시에 저장하기 위해 지연되는 것 이외에도 리턴주소 스택에 저장하기 위한 추가적인 지연시간이 발생한다. 따라서, 함수의 호출이 빈번한 프로그램일수록 성능이 낮아지고, 함수의 호출 깊이가 길어질수록 리턴주소 스택의 오버플로우로 인한 메모리로 백업과 복구, 그리고 백업된 데이터의 무결성을 검사하기 위한 메모리 인증의 오버헤드가 발생한다.



[그림 3] SPEC2000벤치마크 실험결과
[Fig. 3] SPEC2000 benchmark result

SPEC2000의 벤치마크 프로그램 중 parser의 성능저하가 10%로 가장 높은데, parser는 함수를 연속적으로 35번 호출하고 있기 때문에 리턴주소 스택의 오버플로우가 발생하여 메모리 인증에서 오버헤드가 증가하였기 때문이다. 실제 리턴주소 스택의 크기를 35 이상으로 설정하였을 때에는, 리턴주소 스택의 오버플로우가 발생하지 않아 성능저하는 발생하지 않게 된다. 이외에 성능이 측정된 프로그램들은 프로그램 내부의 함수의 잦은 호출은 있지만, 연속적인 함수 호출의 깊이가 35 이하이기 때문에 성능저하는 없다.

그림 4는 리턴주소 스택의 크기에 따른 성능의 변화를 나타낸 그림으로 프로그램의 호출 깊이가 긴 프로그램은 리턴주소 스택의 오버플로우를 발생시켜, 메모리로 데이터를 백업해야 한다. 그러나 백업된 리턴주소 데이터의 무결성을 유지하기 위해 메모리 인증을 사용하고 있으며, 이 때 해시 연산을 위한 추가적인 오버헤드와 주 메모리로의 액세스 오버헤드가 발생한다.



[그림 4] 리턴주소 스택의 크기에 따른 성능 변화
[Fig. 4] Performance with varying return stack size

그림 4에서 테스트 프로그램 parser의 경우, 리턴주소 스택의 크기가 40개 이상이 되어도 실제로 성능에는 변화가 발생하지 않는다. 이는 parser 프로그램이 실행될 때 연속적으로 호출하는 함수의 깊이가 35이기 때문에 리턴주소 스택의 크기가 40이상이 되면, 오버플로우가 발생하지 않기 때문에 성능저하가 발생하지 않은 것이다. vortex 프로그램은 함수의 연속적인 호출 깊이가 8로 리턴주소 스택의 크기가 8이하인 경우 오버플로우가 발생하여 성능이 저하되지만, 리턴주소 스택의 크기가 16 이상이 되면, 성능에 아무 변화가 없는 것을 확인할 수 있다. 따라서 연속적인 함수 호출 깊이를 넘어서는 리턴주소 스택의 엔트리를 확보하고 있다면, 메모리 인증에 대한 오버헤드는 상쇄된다. 그리고 나머지 프로그램들은 함수의 연속적인 호출 깊이가 8이하이기 때문에 리턴주소 스택의 크기에 따른 성능변화가 없다.

5. 결론

버퍼오버플로우 취약점은 매년 발생하는 바이러스, 해킹 등과 같은 보안위협의 50% 이상을 차지하는 취약점으로, 이를 예방하기 위한 소프트웨어적인 해결기법과 하드웨어적인 기법이 많이 제안되었지만 컴파일러 및 라이브러리의 수정, 그리고 프로세서의 추측실행, Nonlocal jump와 같은 비정상적인 함수 호출시에는 정상적으로 동작하지 않거나, 프로세서 내부의 복잡한 파이프라인에 설계되어 하드웨어 복잡도가 증가하는 문제점이 있다.

본 논문에서는 하드웨어 기반의 보호기법인 리턴주소

스택을 제안하였다. 이 방법은 프로세서와 캐시 메모리 사이에서 함수의 호출과 복귀시 스택 메모리에 저장된 함수의 반환주소의 무결성을 확인하여 버퍼오버플로우 취약점을 예방하는 시스템이다. 그 결과 함수의 반환주소를 조작한 버퍼오버플로우 공격으로부터 시스템을 완벽히 보호할 수 있으며, 일반적인 프로그램을 실행할 때에는 성능저하가 발생하지 않는다. 그러나 본 연구를 실제로 적용하기 위해서는 운영체제의 일부 수정이 필요하다는 단점이 있다. 향후 연구에서는 이 같은 문제점을 해결하고, 버퍼오버플로우 취약점이 아닌 다른 시스템 취약점도 예방할 수 있는 프로세서를 설계할 예정이다.

Refrerences

[1] D.Wagner, J. S. Foster, E. A. Brewer, and A. Aiken. "A first step towards automated detection of buffer overrun vulnerabilities". In Proceedings of Network and Distributed System Security Symposium, pp.3-7, Feb. 2000.

[2] D. Evans and D. Larochelle. "Improving security using extensible lightweight static analysis". IEEE Software, Vol.19, No. 1, pp.42-51, February 2002.

[3] ICAT Metabase A CVE Based Vulnerability Database, http://www.icat.nist.gov/icat.cfm

[4] C.C. Center. CERT/cc statistics 1988-2003. http://www.cert.org/stats

[5] Mark W. Eichin and Jon A.Rochlis. With microscope and tweezers: An analysis of the Internet virus of November 1988. Proceeding of the IEEE Symposium on Research in Security and Privacy, 1989.

[6] Roman Danyliw and Allen Householder. CERT Advisory CA-2001-19: Code Red Worm Exploiting Buffer Overflow IN IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, Jul. 2001.

[7] The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com

[8] Lee, R. B., Karig, D. K., McGregor, J. P., and Shi, Z., "Enlisting Hardware Architecture to Thwart Malicious Code Injection," in Proceedings of the Security in Pervasive Computing, Boppard, Germany, 2003, pp.237-252.

[9] SPEC: Standard Performance Evaluation Corporation. http://www.spec.org, September 2000.

[10] clint : A source code checker for C++, http://www.sourceforge.net/projects/clint/

[11] Wagner, D., Foster, J.S, Brewer, E. A., and Aiken, A., "A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities," in Proceedings of the Networks and Distributed System Security Symposium (NDCS), San Diego, CA, USA, 2000.

[12] Cowan, C., Pu, C., Maier, D., Walpole, J., Bakke, P., Beattie, S., Grier, A., Wagle, P., Zhang, Q., and Hinton, H., "Stackguard: Automatic Adaptive Detection and Prevention of Buffer Overflow Attacks," in Proceedings of the 7th USENIX Security Conference, San Antonio, TX, USA, pp. 63-78, 1998.

[23] Oppenheimer, D. L. and Martonosi, M. R., "Performance Signatures: A Mechanism for Intrusion Detection," in Proceedings of the 1997 IEEE Information Survivability Workshop, San Diego, CA, USA, 1997.

[13] Sekar, R., Bendre, M., Dhurjati, D., and Bollineni, P., "A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors," in Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, USA, 144-155, 2001.

[14] Ozdoganoglu, H., Brodley, C. E., Vijaykumar, T. N., Kuperman, B. A., and Jalote, A., "SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address," Purdue University, TR-ECE 03-13, November 22, 2003.

[15] Ye, D. and Kaeli, D., "A Reliable Return Address Stack: Microarchitectural Features to Defeat Stack Smashing," in Proceedings of the Workshop on Architectural Support for Security and Anti-Virus (WASSA), Boston, MA, USA, pp. 69-76, 2004.

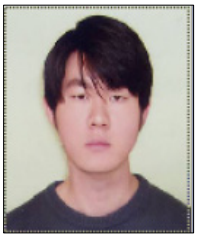
[16] Nergel, "The advanced return-into-lib(c) exploits: PaX case study". Phrack Magazine 11(56), Oct. 2004

[17] Ralph Merkle. “Protocols for public key cryptosystems”. IEEE Symposium on Security and privacy, pp. 122-134. 1980.

[18] Yong-Joon Park, Gyungo Lee, "Microarchitectural Protection Against Stack-Based Buffer Overflow Attacks", IEEE Micro 2006, Vol. 26, No. 4, pp. 62-71

[19] INOUE Koji, "Return address protection on cache memories". IEICE transactions on electronics (IEICE trans. electron.) 2006, vol. 89, no12, pp. 1937-1947

조 병 태(Byungtae Cho) **[정회원]**



- 2008년 2월 : 충남대학교 컴퓨터공학과 (석사)
- 2008년 3월 ~ 현재 : 삼성전자

<관심분야>
내장형 시스템, 시스템 소프트웨어, 컴퓨터 보안

김 형 신(Hyungshin Kim) **[정회원]**

- 1990년 12월 : Univ. of Surrey, 위성통신공학 (석사)
- 2003년 2월 : 한국과학기술원 전산학과 (박사)
- 2003년 4월 ~ 2004년 2월 : Carnegie Mellon Univ. 박사후 연구원
- 2004년 2월 ~ 현재 : 충남대학교 부교수

<관심분야>
내장형 시스템, 시스템 소프트웨어, 우주용 컴퓨터