공학석사 학위논문

ActiMon: SoC FPGA 장치를 활용한 활성 함수 리스트 기반 ROP/JOP 통합 탐지

2019년 2월

서울대학교 대학원

전기정보공학부

양 명 훈

ActiMon: SoC FPGA 장치를 활용한 활성 함수 리스트 기반 ROP/JOP 통합 탐지

지도교수 백 윤 흥

이 논문을 공학석사 학위논문으로 제출함
2019년 2월

서울대학교 대학원

전기정보공학부

양 명 훈

양명훈의 석사 학위논문을 인준함
2019년 2월

위 원 장 김 장 우 (인)

부위원장 백 윤 흥 (인)

위 원 이 병 영 (인)

국문초록

임베디드 시스템에서 널리 쓰이는 프로그래밍 언어인 C/C++의 문제점 중 하나는 메모리 오버플로우 취약점을 활용한 코드 재사용 공격이 가능하다는 점이다. 이로 인해 공격자는 프로그램의 제어 흐름을 조작함으로써 소프트웨어 개발자가 의도하지 않은 동작을 일으킬 수 있다. 코드 재사용 공격을 막기 위한 방법으로는 제어 흐름 무결성 검증 방법이 가장 대표적이며, 이를 구현하기 위한 다양한 접근법들이 제시되어 왔다. 현재까지 제시된 여러 제어 흐름 무결성 솔루션들 중에서, 소프트웨어 기반 솔루션은 상당한 런타임 오버헤드가 발생한다는 단점이 있으며, 하드웨어 기반 접근법은 오버헤드는 많이 줄어든다는 이점과 동시에, ASIC 설계를 요구하기 때문에 솔루션 탑재에 많은 시간이 소요될 뿐 아니라 초기 개발 비용도 증가된다는 단점이 있다.

이 논문에서는 오버헤드 감소라는 이점을 갖는 하드웨어 기반 솔루션을 채택하면서도 ASIC 설계로 인해 발생하는 단점들을 보완하기 위한 방법으로, FPGA를 활용한 제어 흐름 무결성 검증 방법을 제안한다. 그러나 FPGA는 태생적으로 CPU에 비해 속도가 현저히 느리므로, ASIC 설계가 필요한 기존의 하드웨어 기반 솔루션을 FPGA에 그대로 적용하는데 어려움이 있다. 이를 해결하기 위한 방안으로 바이너리 계측 도구를 활용하여 공격 탐지에 필요한 정보만을 탐지 하드웨어가 전달받을 수 있도록 하였다. 또한, 탐지 하드웨어는 활성 함수 리스트를 기반으로 동시에 여러 정보들이 처리될 수 있도록 하여 속도가 느린 FPGA에서도 CRA 탐지가 충분히 가능하도록 설계하였다.

본 논문의 실험장에서는 FPGA를 활용한 솔루션이 소프트웨어 기반 솔루션과 비교하여 코드 사이즈 및 성능 오버헤드가 줄어들었음을 보이고 있으며, 상용 SoC FPGA 장치를 활용하여 프로그램 동작 중에 발생한 ROP와 JOP 공격들이 성공적으로 탐지되고 있음을 보이고 있다.

주요어 : 코드 재사용 공격, 제어 흐름 무결성 검증, 필드 프로그래머블 게이트 어레이, 바이너리 계측
학 번 : 2017-20741

목 차

표 목 차

그 림 목 차

제 1 장 소개

최근 몇 년 간, 소프트웨어는 활용 영역이 넓어짐에 따라 여러 시스템에 활용되어 점차 다양한 역할 수행이 가능해지고 있다. 그런데 임베디드 시스템용 바이너리를 제작할 때 널리 사용되는 프로그래밍 언어인 C/C++은 메모리 오버플로우 취약점을 보완하기 위해 수행되어야 할 [1] 경계 검사(Boundary checking)를 하지 않아 보안상 심각한 단점이 존재한다. 공격자는 이 취약점을 활용해 제어 흐름을 악의적으로 조작하여 프로그램 인터럽트와 같은 비정상 동작을 야기하거나 조작되어서는 안 될 민감한 정보를 변조할 수 있게 된다. 특히, 코드 재사용 공격은 공격자가 이러한 취약점을 활용해 수행할 수 있는 기술로서, 제어 흐름을 조작해 코드 섹션에 존재하는 명령어들(즉, Gadget)을 연결하여 공격자가 의도한 동작이 이뤄지도록 한다.

코드 재사용 공격은 일반적으로 두 가지 방법을 통해 수행된다. 첫 번째는 Callee 함수에서 Caller 함수로 돌아갈 때 사용하는 return 주소를 조작하는 Return-oriented Programming(ROP) 공격이다. 공격자는 메모리 오버플로우 취약점을 활용함으로써 스택에 쌓인 return 주소를 조작하여 본래 돌아가야 할 Caller 함수가 아닌 공격자가 만든 Gadget의 시작 지점으로 제어 흐름을 변화시킨다. 두 번째는 타겟 주소가 가변적인 간접 분기문을 악용하는 Jump-oriented Programming(JOP) 공격이다. ROP와 마찬가지 방법으로 간접 분기문에 사용되는 레지스터에 담길 타겟 주소를 조작하여 본래 도달해야할 지점이 아닌 공격자가 만든 Gadget의 시작 지점으로 제어 흐름을 변화시킨다.

코드 재사용 공격에 대한 위협을 완화하기 위한 방어 대책은 오랜 시간동안 다양한 측면에서 연구되어 왔다. 특히, 제어 흐름 무결성 검증[1]은 소프트웨어 개발자의 의도의 맞게 정상적으로 프로그램이 동작하는지를 확인하는 가장 강력하고 대표적인 방법이다. 지금까지 제안된 제어 흐름 무결성 검증 수행 방법은 크게 소프트웨어 기반 솔루션[2, 3]과 하드웨어 기반 솔루션[4, 5, 6], 두 가지로 나눌 수 있다. 소프트웨어 기반

1) 변수가 사용되기 전에 해당 변수가 점유하는 메모리 경계 내에 위치하는지를 탐지하는 기법

솔루션은 추가적인 코드 삽입만으로 제어 흐름 무결성 검증을 수행할 수 있다는 점에서 어느 플랫폼에서나 쉽게 탑재될 수 있다는 장점이 있다. 하지만 이를 위해 추가되는 코드의 양이 많으므로 성능 오버헤드 또한 상당히 증가하기 때문에 실제로 활용되기 쉽지 않다. 하드웨어 기반 솔루션은 성능 오버헤드가 큰 소프트웨어 기반 솔루션의 문제점을 해결하고자 제안된 방식으로 제어 흐름 무결성 검증을 별도의 하드웨어가 수행할 수 있도록 설계한다.

하드웨어 기반 솔루션은 두 가지 접근법으로 구현될 수 있다. 첫 번째는 제어 흐름 무결성 검증을 담당하는 전용 하드웨어 모듈을 프로세서 내부에 삽입하는 내부 개조식 접근법이다. 이 접근법은 프로세서 구조와 연계하여 제어 흐름 무결성 검증 하드웨어를 프로세서 내에 삽입하는 방법으로써 여러 정보들을 공격 탐지를 위한 기반 정보로 사용할 수 있다는 장점이 있다. 하지만 프로세서 내부 구조의 수정이 필요하기 때문에 이미 설계된 기존 프로세서에는 활용할 수 없을 뿐 아니라, 프로세서가 새로 설계되어야 한다는 단점이 있다. 두 번째는 외부 감시 접근법으로써 내부 개조식 접근법이 상용 프로세서에 탑재될 수 없다는 단점을 해결하기 위해 제안되었다. 이 접근법은 코드 재사용 공격 탐지에 필요한 정보들을 외부로 추출하고, 이 정보를 분석하여 별도의 하드웨어 모듈이 제어 흐름 무결성 검증을 수행하는 방식이다.

하지만 하드웨어 기반 솔루션의 두 가지 접근법은 모두 ASIC 설계가 필요하다는 특징이 있다. ASIC 설계는 성능, 물리적 크기 및 전력 측면에서 상당한 이점을 갖지만, 실제 보드에 회로를 구현하는 제조 공정 기간이 고려되어야 하므로 탑재되기까지 시간이 길어질 수밖에 없다. 또한 설계가 실패하거나 갑작스런 수정이 필요한 경우에 발생하는 비용까지 고려되어야 하므로, 개발 비용이 대폭 증가될 수 있다는 단점이 존재한다. ASIC 설계가 갖는 이러한 단점은 하드웨어 기반 솔루션을 실제로 활용하기 어렵게 만드는 치명적인 요인이다.

본 논문에서는 ASIC 설계로 인해 발생하는 기존 하드웨어 기반 솔루션들의 단점을 보완하고자 FPGA를 활용한 하드웨어 기반 솔루션인 ActiMon을 제안한다. 일반적으로 FPGA는 고정적인 기능을 수행하는 ASIC으로 구현되기 전에, 하드웨어 설계가 적절하게 되었는지를 검증하는 프로토타입용으로 사용되는 것이 일반적이다. 하지만 기술이 발전함

에 따라, FPGA는 구현 방식 또는 용도의 변경 가능성이 있는 프로그래머블리티가 요구되는 제한적인 분야에서 ASIC의 대안으로 활용되고 있다. 만약 FPGA가 코드 재사용 공격 탐지를 위한 용도로 활용된다면, 오버헤드가 적은 하드웨어 기반 솔루션이 갖는 장점을 유지할 수 있을 뿐 아니라 ASIC 설계에 비해 개발 비용도 낮출 수 있어 이전 솔루션들에 비해 활용 가능성이 높아진다. 또한 상용 FPGA을 구입하여 하드웨어 모듈을 탑재만 하면 되므로 솔루션 도입 시기를 앞당길 수도 있다.

ActiMon은 프로세서 구조의 변경을 요구하지 않으므로 기존 하드웨어 기반 솔루션 중에서 외부 감시 접근법을 채택한 [4]와 비슷하다. [4]는 ARM 플랫폼에서 프로그램 디버깅 인터페이스로 사용되는 Program Trace Macrocell(PTM)을 통해 프로세서에서 구동되는 프로그램의 제어 흐름 정보를 추출한다. 이 정보를 바탕으로 코드 재사용 공격 탐지 방안을 다음과 같이 제시하였다. 첫 번째로 ROP 공격 탐지는 Shadow stack을 운용하여 Function call과 return 정보를 전달받아 모든 return 주소에 대해 합법성을 검증하였다. 두 번째로 JOP 공격 탐지는 [6]에서 제시한 탐지 기법을 수용하여, 모든 간접 분기문의 타겟 주소가 다른 함수의 Entry 주소를 가리키는지를 검증한다. 이를 위해 바이너리로부터 모든 함수의 경계 주소 정보가 담긴 메타데이터를 제작하여 간접 분기문 실행 시에 타겟 주소와 메타데이터간의 비교 작업을 수행하는 방식으로 제어 흐름의 무결성을 검증한다.

그런데 [4]에서 제시한 탐지 기법을 그대로 FPGA에 적용하기는 쉽지 않다. 앞서 언급했듯, [4]의 구현은 ASIC 설계를 통해 이뤄지는데 ASIC과 FPGA는 활용 분야가 다르기 때문에 각 장치가 갖는 특성 또한 다르다. 특히, ASIC은 고정적인 기능을 갖도록 설계되기 때문에, 그 기능을 수행할 수 있는 가장 최적화된 제작이 가능하므로 CPU와 같은 속도를 갖도록 설계하는 것이 가능하다. 하지만 FPGA는 프로그래머블리티 특성이 중시된 장치로서 상황에 따라 다양한 동작이 가능하도록 설계되어야 하므로, CPU만큼의 속도를 내기 어렵다. 활용성 차이로 인해 발생한 FPGA의 느린 속도는 CPU와의 속도 불균형을 초래하여 다음과 같은 두 가지 문제를 야기한다. 첫 번째는 코드 재사용 공격 탐지에 필요한 정보가 손실될 수 있다는 점이다. ASIC은 CPU와 같은 속도를 가질 수 있어 PTM에서 배출하는 프로그램 동작 정보를 빠르게 전달받을 수 있다. 그

러나 FPGA는 PTM으로부터 정보를 받아들이는 속도가 느리기 때문에 PTM 내부에 프로그램 동작 정보를 일시적으로 담아두는 FIFO에 오버플로우가 발생한다. 이로 인해 공격 탐지에 반드시 필요한 정보들이 손실되므로 보안적 결함이 야기될 수 있다. 두 번째 문제는 [4]에서 구현한 공격 탐지 기법이 FPGA에 활용되기에 효율적이지 않다는 점이다. ROP 탐지를 위해 운용되는 Shadow stack은 정보가 순차적으로 입력되어 처리되어야 하며, JOP 탐지는 모든 간접 분기문에 검증 절차가 수행되어야 할 뿐 아니라 제어 흐름 검증 절차에 메타데이터가 활용되기 때문에 탐지 속도가 매우 느리다. 그러므로 이러한 탐지 기법은 느린 속도를 가진 FPGA 플랫폼에 탑재하기에 적절하지 않다.

이러한 FPGA의 태생적 단점을 보완하기 위해 ActiMon은 두 가지 해법을 활용한다. 첫 번째는 바이너리 계측 도구를 활용하여 공격 탐지에 반드시 필요한 정보만이 추출될 수 있도록 하여 PTM으로 추출되는 정보를 최소화하였다. 예를 들어, JOP 탐지는 [4]와 마찬가지로 [6]에 제시된 법칙을 활용하나, 모든 간접 분기문의 타겟 주소에 대해 합법성 검증을 수행했던 [4]와는 달리 타겟 주소가 해당 함수의 경계를 넘어가는 경우에만 선별적으로 합법성을 검증한다. 이러한 방법으로 공격 탐지에 필요한 정보를 줄임으로서 PTM 내부의 FIFO 오버플로우 발생 가능성을 최대한 낮춘다. 두 번째는 활성 함수 리스트를 활용하여 여러 정보가 동시에 처리될 수 있도록 하였다. 활성 함수란 call이 된 적 있지만 아직 return이 되지 않은 함수를 의미한다. 즉, 현재까지 각 함수별로 call 및 return이 된 횟수를 활성 함수 리스트에 기록하는 것이다. 이를 위해 본 솔루션에서는 모든 함수의 Entry와 Exit 지점에 바이너리 계측을 수행하여 어떤 함수에 진입했고, 퇴장했는지를 외부에서 파악할 수 있게 하였다. [4]에서 활용한 Shadow stack은 정보가 순차적으로 처리될 수 있도록 설계해야 했지만, 본 솔루션에서는 동시에 다수의 정보 처리가 가능해졌기 때문에 CPU에 비해 느린 FPGA 속도를 극복할 수 있게 되었다.

ActiMon은 ARM 프로세서와 FPGA가 결합된 상용 SoC FPGA 장치인 Zynq-7000 모델에 구현되었다. 이 모델은 PTM으로부터 추출된 정보가 곧바로 FPGA의 Programmable Logic(PL)에 입력되는 구조로서 PL에 설계된 탐지 하드웨어는 이 정보를 전달받아 공격 탐지를 수행한다. 본 논문의 실험장에서는 ActiMon의 탐지 기법을 소프트웨어로만으로 구

현했을 때와 비교하여 소프트웨어 기반 솔루션 대비 저하된 코드 크기 및 성능 오버헤드를 보인다. 동시에 자체 설계한 ROP 및 JOP 공격 프로그램을 실행했을 때, ActiMon에 의해 공격이 성공적으로 탐지되었음을 보인다.

제 2 장 가정 및 공격 모델

본 논문에서는 방어의 대상이 되는 임베디드 장치가 신뢰할 수 있는 OS에서 동작한다고 가정한다. 또한 공격자는 다음과 같은 능력이 있다고 가정한다.

1) 공격자는 권한 상승을 일으켜 추가적으로 발생할 수 있는 공격 요소를 통해 보안상 결함을 일으킬 수 없다.

2) 공격자는 목표 어플리케이션에 대한 구현 세부 사항에 대해 잘 알고 있어 Address Space Layout Randomization(ASLR)과 같은 code randomization 기술을 회피할 수 있다.

3) 공격자는 Buffer overflow와 같이 메모리 오버플로우 취약점을 충분히 활용할 수 있을 정도의 stack과 heap에 대한 제어 권한을 갖고 있다.

제 3 장 세부 구현

제 1 절 바이너리 계측

본 절에서는 원본 바이너리에 바이너리 계측을 수행하여 공격 탐지에 필요한 정보가 PTM을 통해 추출되는 과정을 서술한다. PTM은 간접 분기문의 타겟 주소만을 외부로 배출하도록 설정되었으며, 이러한 PTM의 특성을 활용하여 바이너리 계측을 통해 간접 분기문을 생성하고 타겟 주소에 제어 흐름 정보를 담아 탐지 하드웨어에 전달한다. 바이너리 계측 수행 방법은 ROP 탐지와 JOP 탐지, 두 가지 측면에서 서술한다.

1. ROP 탐지

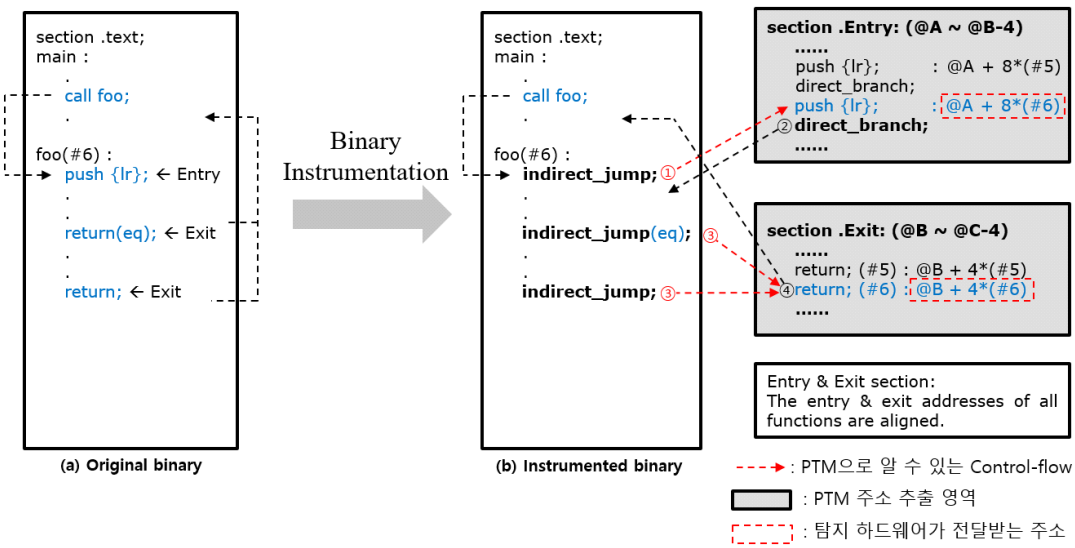

그림 1 원본 코드(a)와 ROP 탐지를 위한 바이너리 계측이 수행된 코드(b)간의 비교

그림 1의 (b)는 ROP 공격 탐지를 위해 원본 바이너리 코드 (a)에서 바이너리 계측을 통해 추가된 부분을 나타낸다. ROP 공격은 특정 함수에 불법적으로 진입한 후, return을 마지막 명령어로 한 공격자의 Gadget을 여러 개 연결하는 방식으로 수행된다. 그러므로 return 주소 조작을 통해 특정 함수에 불법적으로 진입했다면, 함수의 Entry를 거치지 않은 채로 Exit 지점, 즉 return에 도달하게 된다. 이러한 사실을 기반으로 모든 함수의 Entry 지점과 Exit 지점에 바이너리 계측을 수행하여 특정 함수가 Entry 지점의 통과없이 Exit 지점에 도달하였다면 이를

ROP 공격으로 판단한다. 이를 위해 Entry와 Exit 지점에 있는 기존 명령어를 간접 분기문으로 대체하는 방식으로 모든 함수의 진입 및 퇴장을 감시한다. 또한 어떤 함수로 진입하였고 어떤 함수에서 퇴장을 했는지를 알기 위해, 삽입한 간접 분기문의 타겟 주소에 의미를 부여하였다. 예를 들어, 그림 1과 같이 6번째 함수인 *foo*에 바이너리 계측을 수행한다고 가정해보자. *foo* 함수의 Entry에 있었던 *push {lr}*을 간접 분기문으로 대체하고, 이 간접 분기문의 타겟 주소를 *Entry section* 내의 특정 주소로 변경(①)한다. 특정 주소란, *Entry section*의 base 주소에 함수의 번호(*#6*)에 8을 곱한 수가 더해진 만큼을 의미한다. 즉, 6번째 함수인 *foo*의 Entry에 진입했다면 PTM을 통해 *@A+8*(#6)* 값이 배출될 것이며, 탐지 하드웨어는 미리 설정된 *Entry section*의 주소 정보(*@A*)를 활용하여 6번째 함수로 진입했음을 인지할 수 있게 된다. 바이너리 계측으로 인해 기존 제어 흐름을 변경하여 *Entry section*에 진입하였으므로, 다시 정상적인 제어 흐름으로 돌아갈 수 있도록 직접 분기문을 수행(②)한다. 마찬가지로 *return*에도 *Exit section*의 *foo*에 맞는 특정 주소(*@B+4*(#6)*)를 타겟 주소로 한 간접 분기문(③)이 수행되도록 설계한다. 그러면 탐지 하드웨어는 미리 설정된 *Exit section* 주소 정보(*@B*)를 활용하여 PTM으로 배출된 주소 정보를 해석함으로써 6번째 함수에서 퇴장했음을 인지하게 된다.

본 바이너리 계측 방법을 수행함에 있어 두 가지 유의점 존재한다. 첫 번째는 모든 함수의 시작점은 Entry 지점일 수밖에 없지만, Exit 지점인 *return* 명령어는 소프트웨어 설계자의 코드 구성에 따라 여러 개가 존재할 수 있다. 그러므로 바이너리 계측을 통해 한 함수에 존재하는 모든 *return*을 간접 분기문으로 대체해야 하며, 한 함수 내에서 대체된 모든 간접 분기문의 타겟 주소는 *Exit section*의 한 지점을 공통적으로 가리켜야 한다. 즉, 한 함수 내에서 서로 다른 *return*이 수행되었다고 하더라도 같은 주소가 PTM으로 배출되어야 탐지 하드웨어는 동일한 함수에서 퇴장되었다고 인지할 수 있다. 두 번째는 ARM ISA가 다른 플랫폼의 ISA와 다른 점 중 하나는 *return* 명령어에 조건 설정이 가능하다는 점이다. 예를 들어, 그림 1과 같이, *return(eq)*가 있다면 이전 *cmp* 연산을 통해 조건이 *eq*로 결과가 나온 경우에만 *return*을 수행할 수 있다. 이 경우, *return(eq)*로부터 대체된 간접 분기문에도 *eq* 조건을 부여함으로

써 조건이 맞는 경우에만 *Exit section*으로 이동할 수 있도록 바이너리 계측을 설계하였다.

2. JOP 탐지

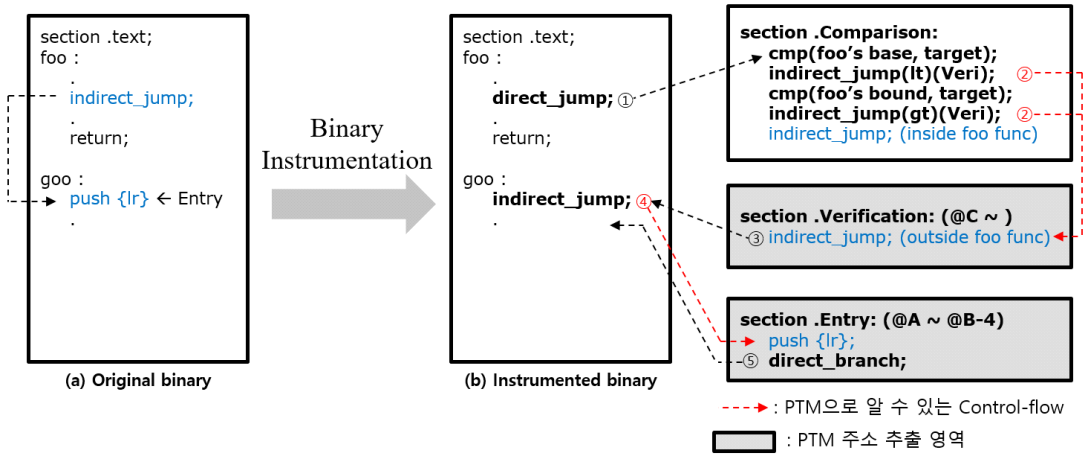

그림 2 원본 코드(a)와 JOP 탐지를 위한 바이너리 계측이 수행된 코드(b)간의 비교

JOP 공격의 탐지는 [6]에 제시된 규칙에 따라, 간접 분기문의 타겟 주소가 해당 함수 경계를 벗어난다면 다른 함수의 Entry 주소가 아닐 때를 JOP 공격으로 규정한다. 그림 2의 (b)는 JOP 공격 탐지를 위해 원본 코드(a)에서 추가된 부분을 나타낸다. 먼저, 원본 바이너리 코드의 간접 분기문에 바이너리 계측으로 직접 분기문으로 고쳐 제어 흐름이 *Comparison section*을 향하도록 만든다(①). *Comparison section*에서는 간접 분기문의 타겟 주소가 해당 함수의 경계를 벗어나는 영역에 속하는지를 판단한다. 주소 영역 판단 절차는 해당 함수의 base 주소보다 작은지를 확인하고, 작지 않으면 bound 주소보다 큰지를 비교하는 순서로 진행된다. 만약 타겟 주소가 함수 경계의 외부에 속한다고 판단되면, JOP 공격 가능성이 있으므로 *Verification section*으로 간접 점프를 수행(②)한다. 반대로 타겟 주소가 해당 함수의 경계 내부라면, [6]의 규칙에 따라 공격 가능성이 없으므로 *Verification section*을 지날 필요 없이 *foo* 함수로 다시 진입한다. 만약 *Verification section*으로 진입했다면 정상 간접 분기문인 경우, 다른 함수의 Entry로 진입할 것이다. 이 때, 3장 1절에서 언급했듯 모든 함수의 Entry 지점에는 *Entry section*으로 제어 흐름을 변경시키는 바이너리 계측이 수행되어져 있으므로 *Verification section* 진입 직후에는 반드시 *Entry section*으로 진입하는 정보가 배출

되어야 한다. 즉, 공격자가 간접 분기문의 타겟 주소를 조작하여 간접 분기문이 여러 개 연결되도록 공격을 설계했다면, *Verification section* 진입 이후에 *Verification section* 영역에 계속 진입할 것이다. 탐지 하드웨어는 이러한 section 간 이동을 감지하여 JOP 공격을 탐지한다.

제 2 절 하드웨어 구조

1. 전체 시스템 구조

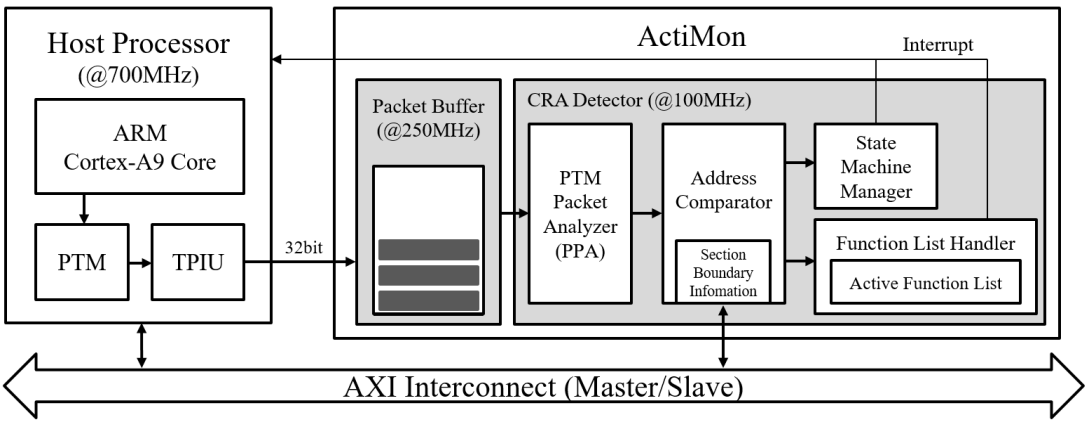

그림 3 전체 시스템 구조

그림 3은 호스트 프로세서와 ActiMon이 포함된 전체 시스템 구조를 나타낸다. 바이너리 계측을 수행한 바이너리가 호스트 프로세서에서 동작하면 PTM은 프로그램 동작 정보를 인코딩한 패킷들을 내부 FIFO에 쌓는다. 쌓인 패킷들은 Packet Buffer의 클럭에 맞추어 TPIU를 통해 배출된다. 배출된 패킷들은 ActiMon의 입력 단에 위치한 Packet Buffer에 순차적으로 쌓인다. Packet Buffer는 패킷의 일시적인 저장소로써 FPGA가 낼 수 있는 가장 빠른 속도로 패킷을 받아들여 PTM내부의 FIFO의 오버플로우 발생 가능성을 최대한 낮추는 역할을 한다. 그래서 Packet Buffer는 본 설계에 사용된 FPGA 모델의 최대 속도인 250MHz에 맞추어 설계되었다. CRA Detector는 Packet Buffer에 저장된 패킷을 순서대로 받아들여 공격 탐지를 진행한다. PTM Packet Analyzer는 인코딩된 패킷을 입력으로 받아 디코딩 작업을 하여 타겟 주소를 계산하고 4개를 한 세트로 하여 Address Comparator에 전달한다. Address Comparator는 바이너리 동작 전에 시스템 버스를 통해 미리 설정된 *Entry, Exit* 및 *Verification section*의 주소 경계 정보를 바탕으로 어떤 section에 진입했는지, 몇 번째 함수의 Entry 혹은 Exit에 진입했는지를 해석한다. 그리

고 Address Comparator의 뒷단에 존재하는 State Machine Manager와 Function List Handler가 정보를 쉽게 처리할 수 있게끔 정보를 가공하는 역할을 한다. State Machine Manager는 section간 transition을 파악하여 *Verification section* 진입 직후에 *Entry section*에 진입했는지를 확인하는 모듈로써 JOP 공격을 탐지하는 역할을 수행한다. 만약 JOP 공격이 탐지되면 인터럽트 신호를 호스트 프로세서에 보내 프로그램 동작을 중단시킨다. 마지막으로 Function List Handler는 모든 함수의 call 및 return 빈도 정보가 담긴 활성 함수 리스트(Active Function List)를 관리하는 모듈로써 ROP 공격을 탐지하는 역할을 수행한다. Function List Handler 또한 State Machine Manager와 마찬가지로 ROP 공격이 탐지되면 호스트 프로세서에 인터럽트 신호를 보내 프로그램 동작을 중단시킨다.

2. Address Comparator

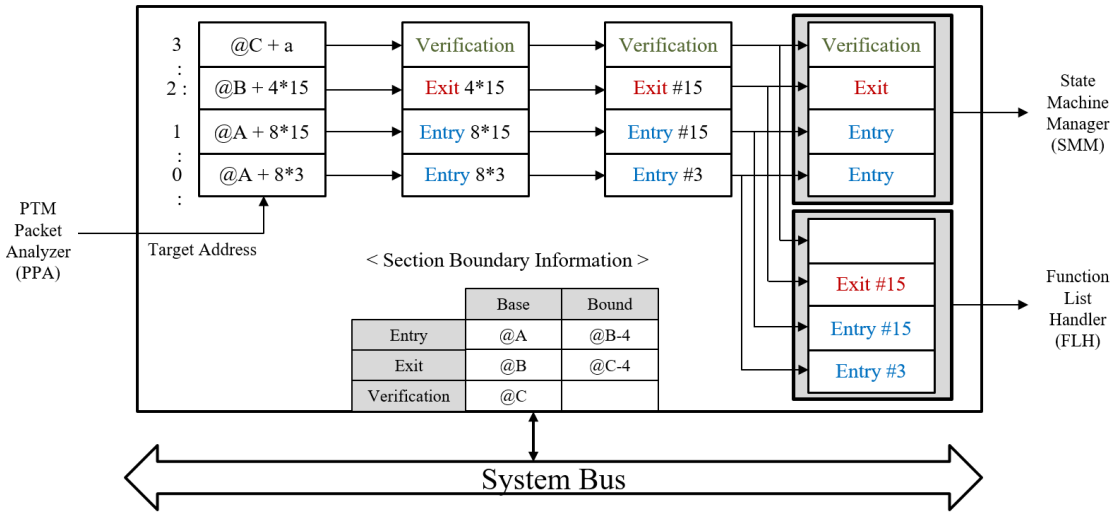

그림 4 Address Comparator의 동작 메커니즘

그림 4는 4개의 동작 정보를 동시에 처리하는 Address Comparator의 동작 메커니즘을 나타낸다. 앞서 설명했듯, Address Comparator는 바이너리의 실행 전에 바이너리 계측으로 인해 생성된 *Entry, Exit* 및 *Verification section*의 주소 경계 정보를 시스템 버스를 통해 미리 입력받는다. 그리고 호스트 프로세서에서 바이너리가 실행되면 PPA로부터 타겟 주소 4개가 한 세트를 이루어 Address Comparator에 입력된다. Address Comparator는 미리 설정된 각 section 별 주소 경계 정보를 참

조하여 4개의 타겟 주소가 의미하는 바를 해석한다. 즉, 어떤 section에 진입했는지, 몇 번 함수에 진입 및 퇴장했는지를 파악하는 과정을 거쳐 state 정보는 Entry, Exit 및 Verification 중 하나로 전환되며, 만약 state가 Entry나 Exit에 해당된다면 몇 번째 함수에 해당하는 것인지를 파악한다. 이 과정을 거치면, State Machine Manager에 입력되는 4개의 state 정보 한 세트가 생성됨과 동시에 Function List Handler에 입력되는 최대 4개의 함수 Entry 및 Exit 진입 정보 한 세트가 생성된다.

3. State Machine Manager

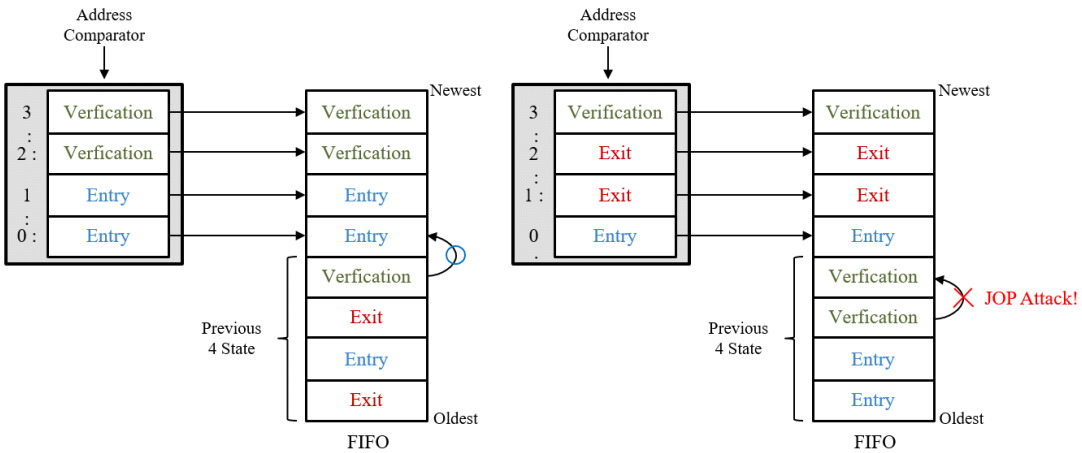

그림 5 State Machine Manager의 동작 메커니즘

그림 5는 Address Comparator로부터 전달받은 한 세트를 동시에 처리하는 State Machine Manager의 동작 메커니즘을 나타낸다. JOP 공격의 탐지는 Verification state의 다음 정보가 Entry state인지를 확인하는 과정으로 이뤄진다. 그런데 한 세트에 존재하는 4개의 정보만으로 공격을 탐지하기에는 마지막 4번째 정보가 Verification state일 때 다음 정보가 아직 입력되지 않아 JOP 공격 여부를 알 수 없다. 그러므로 총 8개의 FIFO Buffer를 만들어 이전 4개의 정보를 저장해두고, 다음 4개의 정보가 입력되어 이전 세트의 모든 정보에 대해 검증이 가능한 상태가 되면 JOP 공격 탐지를 수행한다.

4. Function List Handler

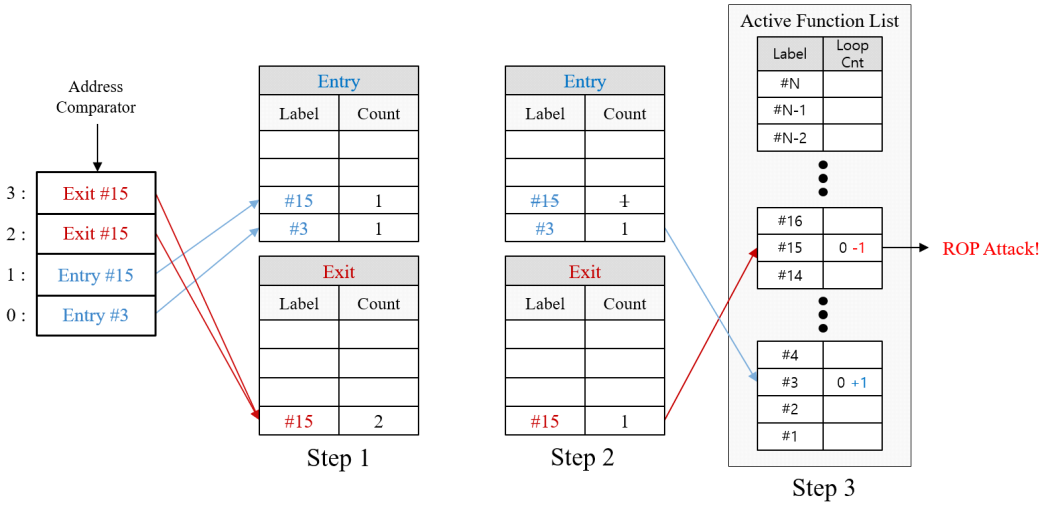

그림 6 활성 함수 리스트를 관리하는 Function List Handler의 동작 메커니즘

그림 6은 Address Comparator로부터 전달받은 4개의 Entry 및 Exit 정보를 활성 함수 리스트에 반영하는 메커니즘과 ROP 공격이 탐지되는 상황을 나타낸 것이다. Function List Handler는 크게 세 단계로 구성되어 있다. 첫 번째 단계에서는 4개의 정보에 대해 Entry 혹은 Exit으로 구분하고, 같은 함수에 대해 Entry와 Exit이 여러 번 나타난다면 빈도수만큼 해당 함수의 Count 값에 중첩 반영한다. 두 번째 단계에서는 같은 함수에 대해 Entry와 Exit 정보가 같이 존재한다면 서로의 Count 값을 상쇄시킨다. 마지막 단계에서는 두 번째 단계까지 연산을 마친 정보들을 바탕으로 활성화 함수 리스트에 값을 반영한다. 단, Entry에 속한 함수의 Count 값은 활성 함수 리스트에 반영될 때 양의 값으로, Exit에 속한 함수의 Count 값을 활성 함수 리스트에 반영될 때 음의 값으로 반영된다. 그런데 두 번째 단계의 Exit 정보가 활성 함수 리스트에 반영될 때, 해당 함수의 LoopCnt가 0이었다면 이는 이전에 *Entry section*으로 진입한 적이 없는데 *Exit section*으로 진입했다는 것을 의미한다. 이 경우, 불법적인 함수 진입으로 판단하여 ROP 공격 발생을 알린다.

제 4 장 실험 결과

본 솔루션의 효과를 평가하기 위해 ActiMon의 탐지 기법을 소프트웨어로만으로 진행했을 때를 비교군으로 두어 바이너리 크기 및 성능 오버헤드를 측정하는 실험을 진행하였다. SW-only는 소스 코드 수정 및 바이너리 계측만으로 ActiMon이 수행하는 ROP 및 JOP 공격 탐지 기법을 그대로 구현하였으므로 ActiMon과 같은 수준의 보안성을 갖는다. 기준 바이너리는 SPEC CPU2006 벤치마크[7] 중 1절과 같이 8개를 선택하였다.

제 1 절 오버헤드 비교

1. 바이너리 크기 오버헤드

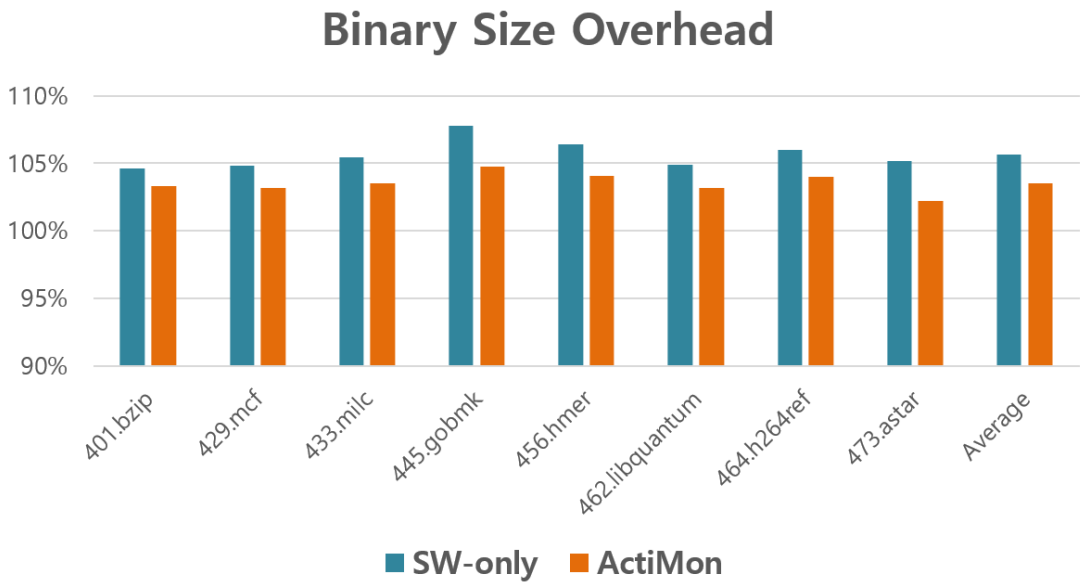

그림 7 ActiMon의 탐지 기법을 소프트웨어로만 수행했을 때(SW-only)와 ActiMon 사용을 위한 바이너리 계측 수행했을 때(ActiMon)의 바이너리 크기 비교

그림 7은 SW-only의 바이너리와 ActiMon에 활용된 바이너리 간 크기를 비교한 그래프를 나타낸다. 그래프 상에서 100%는 원본 바이너리의 크기를 의미하며, 100%와 차이가 커질수록 원본 바이너리로부터 많은 양의 바이너리 계측이 삽입되었음을 의미한다. 바이너리 계측 이후 늘어난 바이너리 크기 확인 결과, SW-only는 평균 약 5.6%의 바이너리 크기 오버헤드를 보였으며, ActiMon의 바이너리는 평균 약 3.5%의 바이너리 크기 오버헤드를 보였다. ActiMon은 SW-only에 비해 평균 약

2.1%의 바이너리 크기 오버헤드 감소 효과를 보이고 있다.

2. 성능 오버헤드

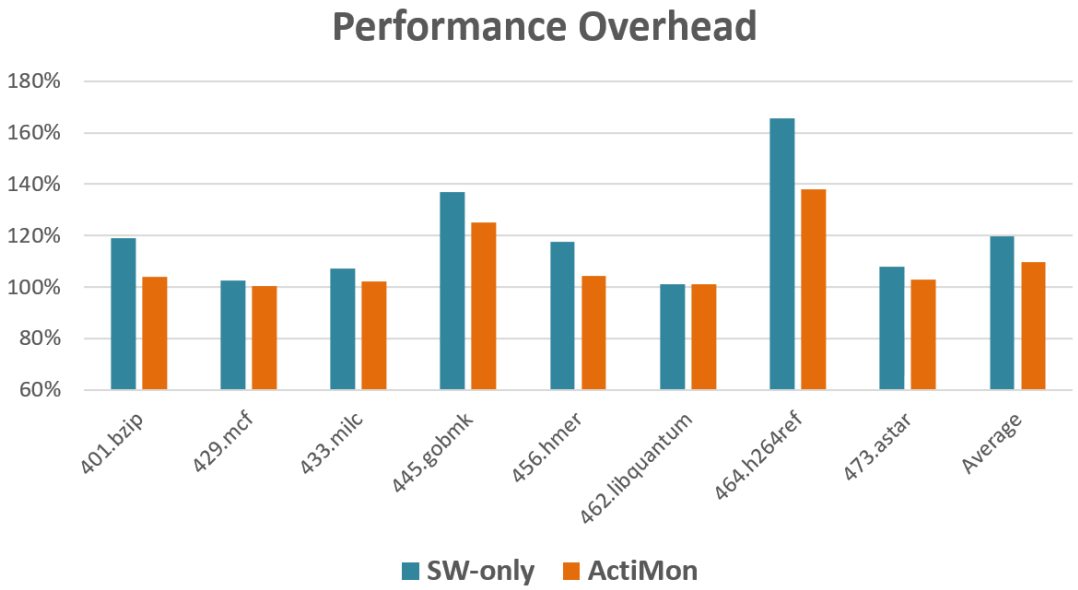

그림 8 ActiMon의 탐지 기법을 소프트웨어로만 수행했을 때(SW-only)와 ActiMon 사용을 위한 바이너리 계측 수행했을 때(ActiMon)와의 성능 오버헤드 비교

그림 8은 SW-only와 ActiMon간에 바이너리 크기를 비교한 그래프를 나타낸다. 바이너리의 실행은 ARM Cortex-A9 프로세서를 탑재한 ZYNQ-7 ZC706 Evaluation Board(xc7z045ffg900-2)[8]에서 이루어졌다. 그래프 상에서 100%는 바이너리 계측을 수행하지 않은 원본 바이너리의 성능 오버헤드를 의미하며, 100%와 차이가 커질수록 성능 오버헤드가 크다는 것을 의미한다. 성능 오버헤드 실험 결과, SW-only는 평균 약 19.71%의 성능 오버헤드를 보였으며 ActiMon에 활용되는 바이너리는 평균 약 9.77%의 성능 오버헤드를 보였다. 결과적으로 ActiMon의 활용으로 인해 평균 약 10%정도의 오버헤드 감소효과를 보였다.

표 1은 ActiMon을 구성하는 각 하드웨어 모듈별 LUT, FF 및 BRAM의 사용량과 xc7z045ffg900-2의 가용한 전체 하드웨어 리소스 대비 ActiMon의 하드웨어 리소스 사용량을 나타낸다. ActiMon의 하드웨어 모듈은 한 클럭당 4개의 주소 정보에 대해 동시 처리가 가능하며, 최대 2047개의 함수를 관리 가능하고 각 함수당 최대 8번의 Loop를 셀 수 있도록 설계하였다. Packet Buffer는 입력과 출력이 클럭을 독립적으로

Components		LUTs	FFs	BRAM
Packet Buffer		115	229	4
CRA Detector	PTM Packet Analyzer	3064	954	0
	Address Comparator	1624	100	0
	State Machine Manager	7	1	0
	Function List Handler	95207	7798	0
	Total	**100017**	**9082**	**4**
	% over FPGA resources	**45.75%**	**2.08%**	**0.73%**

표 1 FPGA의 가용한 전체 하드웨어 리소스 대비 ActiMon의 하드웨어 리소스 사용량

사용할 수 있는 Independent Clock BRAM을 사용하여 설계하였다. xc7z045ffg900-2가 사용할 수 있는 최대 하드웨어 리소스는 LUT 218600개, FF 437200개 및 BRAM 545개이며, ActiMon은 각 하드웨어 요소에 대해 45.75%, 2.08%, 0.73%의 하드웨어 리소스를 활용하고 있다.

제 2 절 공격 탐지

공격 탐지 실험은 자체적으로 ROP 및 JOP 공격을 설계하여 ActiMon의 탐지 성능을 확인하였다. 첫 번째로 ROP 공격 설계를 위해 strcpy로 할당된 배열 크기보다 더 큰 값을 입력할 수 있는 파일을 읽어들여 return 주소가 적힌 스택의 값을 조작하였다. 조작한 주소는 특정 변수의 값이 연산이 시작되는 다른 함수의 중간 지점으로 하여 정상적인 제어 흐름에서는 일어나지 않아야 할 데이터 변조를 일으켰다. ROP 공격 결과, ActiMon은 불법적으로 진입한 함수의 return이 있는 Exit section에서 ROP 공격을 탐지하였다. 두 번째로 JOP 공격 설계는 ROP 공격과 마찬가지 방법으로 스택을 조작하여 간접 분기문에서 참조하는 레지스터의 값을 다른 함수에 있는 간접 분기문이 있는 주소로 변경하였다. JOP 공격 결과, ActiMon은 조작한 간접 분기문을 통해 수행된 다른 함수의 간접 분기문이 *Verification section*에 진입할 때 JOP 공격을 탐지하였다.

제 5 장 토의

본 논문에서 사용한 코드 재사용 공격 탐지 기법은 기본적으로 [5], [6]의 탐지 전략을 따르고 있다. 먼저, [5]에서 제안한 HAFIX의 Label State Memory는 ActiMon에 탑재된 활성 함수 리스트와 같은 역할을 한다. HAFIX는 제어 흐름 무결성 검증을 수행하는 하드웨어 모듈이 코어 내부에 삽입된 내부 개조식 접근법이며, 제어 흐름 무결성 검증용 ISA를 추가하는 방식으로 ROP만을 방어한다는 점에서 ActiMon과 노선을 달리한다. 그리고 구현 방법에 있어서 HAFIX는 모든 Function call의 다음 지점에 합법적인 return 발생 여부를 인지할 수 있는 명령어가 추가되어 있다. 하지만 ActiMon에서는 ROP 공격 탐지를 위해 함수의 Entry와 Exit만을 감시한다는 점에서 HAFIX와 차이가 있다. 이러한 차이로 인해, HAFIX는 return 주소가 조작되었다면 해당 return 수행 후에 바로 탐지할 수 있는 반면, ActiMon은 조작된 return 수행 후에 불법적으로 진입한 함수의 return 지점에서 ROP 공격을 탐지한다. 이로 인해 ActiMon은 HAFIX에 비해 보안성이 떨어지기는 하지만, 공격자가 단 한 번의 Gadget 연결만으로 공격을 수행할 수 있어야하므로 이를 만족하는 공격을 구현하기 쉽지 않다. 그러므로 ActiMon의 탑재는 ROP 공격에 대한 억제력을 갖도록 만든다. 또한 ActiMon은 JOP 공격 탐지를 위해 [6]에서 제안한 Branch Regulation의 탐지 전략을 따른다. 그런데 타겟 주소의 합법성을 간접 분기문 실행 시에 즉각적으로 판단하는 Branch Regulation과는 달리, ActiMon은 간접 분기문의 합법성을 판단하기 위해서는 다음 state 정보를 필요로 한다. 이로 인해 ActiMon은 [6]에 비해 보안성이 떨어지기는 하지만, ROP 공격 때와 마찬가지로 공격자가 단 한 번의 Gadget 연결만으로 공격을 수행할 수 있어야하기 때문에 이를 만족하는 JOP 공격 또한 구현하기 쉽지 않다. 그러므로 ActiMon의 탑재는 JOP 공격에 대한 억제력을 갖도록 만든다.

제 6 장 결론

본 논문은 ASIC 설계 없이도 상용 FPGA에 탐지 하드웨어를 탑재하여 ARM 플랫폼에서 동작하는 바이너리의 ROP/JOP 공격을 통합 탐지할 수 있는 ActiMon을 제시하고 있다. ASIC과 FPGA는 활용도가 다르므로 그 특성 또한 다를 수밖에 없는데, 이러한 차이는 ASIC 설계를 요구하는 기존 하드웨어 탐지 전략을 그대로 FPGA에 활용할 수 없도록 만든다. 특히, 속도 측면에서 FPGA는 ASIC에 비해 현저히 느리기 때문에 이 한계점을 보완하기 위한 대책이 필요하다. 이에 대한 대책으로, 첫 번째는 바이너리 계측을 통해 최소한의 정보만으로 코드 재사용 공격을 탐지하는 것이며, 두 번째는 활성화 함수 리스트를 기반으로 다수의 정보가 동시에 처리되도록 하는 것이다. 두 가지 방안을 활용하여 FPGA에 탑재된 ActiMon은 코드 재사용 공격을 성공적으로 탐지해내었다. 또한 실험 결과를 통해 소프트웨어만으로 공격 탐지를 수행해내었을 때에 비해 바이너리 코드 사이즈 및 성능 오버헤드를 크게 감소시킬 수 있었다.

참 고 문 헌

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-ow integrity: Principles, implementations ,and applications. ACM Trans. Inf. Syst. Secur., 13(1), 2009.

[2] P. Chen et al., “DROP: Detecting return-oriented programming malicious code,” in Information Systems Security. Springer, 2009, pp. 163 - 177.

[3] B. Niu and G. Tan, “Per-Input Control-Flow Integrity,” in Proceedings ofthe 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015

[4] Yongje Lee, Jinyong Lee, Ingoo Heo, Dongil Hwang, and Yunheung Paek. 2016. Integration of ROP/JOP Monitoring IPs in an ARM-based SoC. In Proceedings of the 2016 Conference on Design, Automation and Test in Europe (DATE ’16). EDA Consortium, 331 - 336

[5] O. Arias, L. Davi, M. Hanreich, Y. Jin, P. Koeberl, D. Paul,A.-R. Sadeghi, and D. Sullivan. Hafix: Hardware-assisted flow integrity extension. In 52nd Design Automation Conference(DAC), 2015.

[6] M. Kayaalp et al., “Branch regulation: Low-overhead protection from code reuse attacks,” in Computer Architecture (ISCA), International Symposium on, June 2012, pp. 94 - 105.

[7] J. L. Henning, “SPEC CPU2006 benchmark descriptions,” ACM SIGARCH Computer Architecture News, vol. 34, no. 4, pp. 1 - 17, 2006.

[8] Xilinx Zynq-7000 SoC ZC706 Evaluation Kit, https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html

Abstract

ActiMon: Unified JOP and ROP Detection with Active Function Lists on an SoC FPGA

양 명 훈

전기·정보 공학부

The Graduate School

Seoul National University

One of the problems with C / C ++, a widely used programming language in embedded systems, is that it has a memory overflow vulnerability that can lead to code reuse attacks. This allows the attacker to manipulate the control flow of the program, causing malicious behaviors unintentionally. As a method to prevent this attack, Control-Flow Integrity(CFI) enforcement is the most powerful technique and there have been various approaches for implementing this technique. Among these approaches, the software-based approach leads to substantial run-time overhead and the hardware-based approach requires ASIC solutions which generally suffer from very high Entry-barrier in terms of manufacturing time and non-recurring cost.

To overcome the disadvantages of the existing solutions, this paper proposes a CFI enforcement using FPGA. However, since FPGAs are inherently slower than the host processors, it is difficult to directly adopt the existing hardware-based solutions requiring ASIC design to

FPGAs. In order to solve this problem, our solution utilizes the binary measurement tool that allows the information necessary for attack detection to be selectively extracted. Furthermore, the detection hardware manages Active Function List to process multiple information simultaneously, complementing the limitation of slower FPGAs compared to CPUs.

Experimental results show that this solution reduces code size and performance overhead compared to the software-based solution and ROP/JOP attack are successfully detected using a commercial SoC FPGA device.

keywords : Code Reuse Attack(CRA), Control-Flow Integrity(CFI), Field Programmable Gate Array(FPGA), Binary Instrumentation

Student Number : 2017-20741