오류정정 부호 기반 명령어 연관성 기법을 적용한 임베디드 보안 프로세서의 성능평가

정회원 이 승 욱*, 준회원 권 순 규*, 정회원 김 종 태*

Performance Evaluation of Secure Embedded Processor using FEC-Based Instruction-Level Correlation Technique

Seung Wook Lee* *Regular Member*, Soongyu Kwon* *Associate Member*,
Jong Tae Kim* *Regular Member*

요 약

본 논문에서는 명령어 실행 전에 소프트웨어 또는 하드웨어의 공격에 의한 변조된 명령어의 실행을 방지할 수 있는 새로운 명령어 연관성 기법을 제안한다. 암호화 과정의 복잡성과 암호 모듈의 낮은 처리 속도로 인하여 암호화 기반 보안 프로세서는 오버헤드에 의한 심각한 성능 저하가 발생한다. 반면에, 오류정정부호를 이용한 명령어 기법을 적용한 보안 프로세서는 적은 오버헤드로 인해 일반 프로세서와 비교하여 성능 저하가 거의 없다. 실험 결과 일반 프로세서에 비해 보안 프로그램의 코드와 패리티를 포함하여 필요한 총 메모리양은 평균 26.62% 늘었고, 보안 프로그램의 CPI 상승률은 평균 1.20%~1.97% 증가하였다.

Key Words : embedded processor, computer security, FEC

ABSTRACT

In this paper, we propose new novel technique (ILCT: Instruction-Level Correlation Technique) which can detect tempered instructions by software attacks or hardware attacks before their execution. In conventional works, due to both high complex computation of cipher process and low processing speed of cipher modules, existing secure processor architecture applying cipher technique can cause serious performance degradation. While, the secure processor architecture applying ILCT with FEC does not incur excessive performance decrease by complexity of computation and speed of tampering detection modules. According to experimental results, total memory overhead including parity are increased in average of 26.62%. Also, secure programs incur CPI degradation in average of 1.20%~1.97%.

Ⅰ. 서 론

반도체 기술의 끊임없는 진보를 통해 데스크톱 컴퓨터 시스템에서나 이루어질 수 있었던 많은 작업들이 PDA(personal digital assistants) 및 스마트 폰으로 대표되는 소형 모바일 임베디드 시스템들에서 이루어지고 있다. 임베디드 시스템에서도 데스크톱 컴퓨터 시스템과 같이 다양한 형태의 보안 위협에 직면하고 있다. 데스크톱 컴퓨터 환경에서 축적된 대응 기법은 임베디드 시스템도 데스크톱 컴퓨터 시스템과 유사한 소프트웨어/하드웨어 자원을 가지고 있으므로 이제 임베디드 시스템에도 적용 가

※ 본 연구는 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단(KRF-2006-521-D00376)의 지원으로 수행되었습니다.
* 성균관대학교 정보통신공학부(jtkim@skku.edu)
논문번호 : KICS2009-03-127, 접수일자 : 2009년 3월 26일, 최종논문접수일자 : 2009년 4월 20일

능하다고 할 수 있지만, 데스크톱 컴퓨터 시스템 환경에서의 보안 위협이 소프트웨어적-원격적 위협으로 진보했던 반면에 임베디드 시스템에서는 보안 공격 자체가 물리적(하드웨어적)-직접적으로 가해질 수 있다. 이는 임베디드 시스템이 공격자로 하여금 대상 시스템의 직접적 취득을 용이하게 하고 물리적 기법을 동원하여 기존 보안 대책을 무력화 시킬 수 있음을 의미하다. 이러한 문제 인식으로 인하여 근래 여러 연구에서 그 대응 기법이 논의되고 있다. 그 대표적인 대응 기법이 임베디드 시스템에 보안 프로세서의 적용이다[1-3]. 본 논문에서는 프로세서에서 수행되는 명령어의 비정상적 실행을 방지할 수 있는 명령어 연관성 기법(ILCT: Instruction-Level Correlation Technique)을 제안하고, ITCT가 적용된 보안 프로세서 구조 하에서의 성능 분석 결과에 관해 논한다. ILCT는 오류 정정 부호화의 연관 연산 특성을 이용하여 적은 시스템 오버헤드로 강력한 보안 성능을 제공할 수 있다.

Ⅱ. 임베디드 시스템의 보안 위협 기법

소프트웨어적 기법을 이용한 공격은 주로 시스템에 내장된 프로그램의 취약성을 악용하여 그 위협이 수행된다. 가장 잘 알려진 프로그램 취약성은 버퍼-오버플로우 취약성으로 버퍼의 경계를 확인하지 않는 프로그램 코드 취약성을 악용하여 공격자가 의도하는 악의적 코드로 오염시키는 기법이다[3]. 이에 대한 대응은 정적 기법으로 사전에 취약성을 가지는 코드를 제거하거나, 동적으로 소프트웨어적 감시를 통한 비정상적 프로그램 코드의 실행 방지 또는 보안 프로세서를 적용을 통한 프로그램 코드의 무결성 확인으로 비정상적 명령어 실행 방지 기법으로 대응 가능하다[3].

하드웨어적 기법은 물리적 위협과 side-channel 위협으로 구분될 수 있다. Side-channel위협은 시스템의 동작 과정에서 노출되는 side-channel information을 악용하여 수행되는 수동적 보안 위협이다. 이는 경우에 따라서 암호화 시스템을 무력화 시킬 수도 있다. 시스템에 가해지는 능동적 기법인 물리적 위협은 공격자의 시스템에 대한 물리적 접근 레벨에 따른 다양한 형태의 경로를 가지고 있지만 본 논문에서는 임베디드 시스템을 구성하는 보드 레벨서 수행 가능한 물리적 위협을 고려 대상으로 한다. 보드 레벨에서 수행 가능한 물리적 위협의 대표적인 기법은 시스템의 데이터 전송라인 변조 및 도청 기법이다. 도청은 메모리와 프로세서 또는 기타 장치 간의 데이터 전송라인의 도청을 통해 공격자가 원하는 정보의 직접적 취득하는 기법이다. 이는 스코프 장비를 통해 쉽게 이루어 질 수 있다. 변조는 데이터 전송라인에 직접적으로 위협이 가해지는 데이터를 강제로 시스템에 주입하는 기법이다. 그 주입 대상은 주로 비정상적 명령어 코드이다. 본 논문에서 제안하는 ILCT는 시스템 데이터 전송라인 변조 문제를 방지/대처 가능하다.

소프트웨어적 기법과 하드웨어적 기법이 복합적으로 적용되어 시스템에 보안 위협을 가할 수 있다. 즉, 소프트웨어적 취약성을 통하여 프로그램의 실행 루틴을 변경하는 것이 아니라 시스템 보드 상에 노출된 시스템 어드레스 라인을 변조하여 프로그램 실행 루틴을 변경 시키거나, 하드웨어적 기법을 동원해 변조한 명령어 코드를 기반으로 소프트웨어적 기법의 공격을 수행하는 형태의 보안 위협이 가능하다. [4]에서 시스템의 데이터 전송라인 조작을 통해 암호화된 메모리의 내용을 암호화-키 없이 유추할 수 있는 기법이 있음을 증명하였다.

Ⅲ. 오류정정부호를 이용한 명령어 연관성 기법

3.1 명령어 연관성의 개념

현재 프로세서에서 수행되는 명령어는 이전에 수행된 명령어와 연관성을 가지고 있고 그 다음에 수행되는 명령어와도 연관성을 가진다. 만일 이러한 연관성이 단절되는 경우 현재 명령어는 비정상적인 명령어로 단정 지을 수 있다[7].

그림 1에서 프로그램 명령어 코드가 정상적인 동작 상황이라면 메모리 주소 0x400390에 저장된 분기 명령어의 조건식 결과에 따라 두 가지의 실행 순서를 가진다. 그림 1의 프로그램에서 이 두 가지 이외의 명령어 실행 순서는 명령어 수준에서 침해를 받았다고 단정 지을 수 있다. 따라서 각 명령어의 실행 시 명령어 변조 여부와 그 명령어 실행의 정상 실행 순서 유/무를 사전에 판별할 수 있다면, 소프트웨어적 기법 및 하드웨어적 기법을 통해 시스템에 가해지는 보안 위협을 효과적으로 방지 가능하다. 그림 2(a)에서 그림 1의 정상 프로그램이 소프트웨어적 기법을 이용하여 분기 명령에 사용되는 주소가 변조(Loop2→dirty)된 경우라면, 이 실행 코드를 실행 시 그림 1에서와는 다른 실행 순서로 명령어가 실행 될 것이다. 정상적인 경우 0x400390에 저장된 분기 명령의 실행 후에는 조건식의 결과

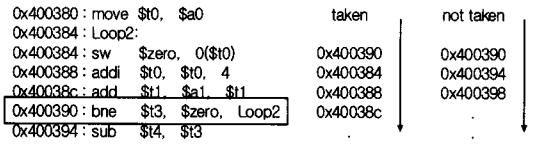


그림 1. 정상적인 명령어 실행 순서의 예시

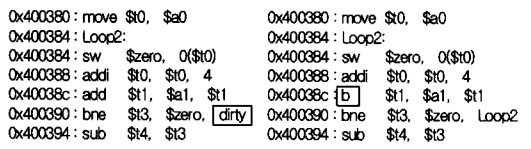


그림 2. 명령어 수준 침해 코드 예시

에 따라 0x400384에 저장된 명령이 실행되거나 0x400394에 저장된 명령이 실행되어야 한다. 하지만 그림 2(a)의 경우는 조건식의 결과가 taken일 경우 그 실행 순서에 변화가 생길 것이다. 그림 2(b)의 경우는 메모리 주소 0x40038c의 명령어 add가 임의의 침해 기법을 통해 명령어 b(Branch Instruction)로 변조된 경우이다. 이 경우는 주소 0x400388의 명령어 다음의 명령어 b가 정상적이지 않으므로 명령어 간의 연관성에 단절 상황이 발생한 경우라 할 수 있다. 이와 같이 ILCT은 논리적 명확성을 통해 현재 실행 명령어의 이전 명령어와 이후 명령어 간의 연관성을 통해 현재 명령어의 변조 및 실행 순서 이상을 확인 가능한 기법이다.

3.2 명령어 연관성 유도 기법

명령어간의 연관성을 유도하는 기법은 오류 정정 부호화 코드 연산 개념과 유사하다. 연관성이 단절되는 상황은 부호화 코드에 오류가 발생한 상황과 동일한 상황이다. 명령어의 변조는 코드에 발생한 오류와 같은 의미이고, 비정상적인 명령어 실행 순서의 변화는 부호화 코드순서의 변화와 같은 상황으로 마찬가지로 오류가 발생한 상황과 같은 의미이다[6]. 그림 3은 각 명령어가 각 단위 시간에 하나의 명령어를 memory로부터 fetch한다고 가정하고, 이 경우 각 명령어가 저장된 주소와 해당 명령어 그리고 선행 명령어의 내용이 연관적으로 사용되어 오류 정정 부호화 코드 연산 과정을 통해 유도된 parity를 가지는 구조를 나타낸다. 현재 시간 $t(i)$을 기준으로 각 명령어와 그 주소를 통해 parity 1과 parity 2를 구하는 연산을 다음 식(1)과 식(2)와 같이 유도한다.

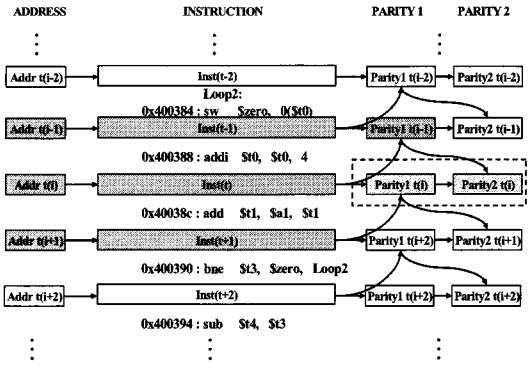


그림 3. 명령어 연관성 유도를 위한 연산 데이터 구조

$$P1\_t(i) = P(A\_t(i), I\_i(i), I\_t(t+1)) \quad \text{식(1)}$$

$$P2\_t(i) = P(A\_t(i), I\_t(i), P1\_t(i), P1 \quad \text{식(2)}$$

여기서 parity 함수 $P1_t(i)$와 $P2_t(i)$는 일반적인 오류 정정 부호화 연산에 사용되는 함수로 본 논문에서는 ILCT의 평가를 위해 Reed-Solomon 오류 정정 부호화 연산[6]을 적용하였다. 식(1)과 식(2)에 따르면 parity를 구하기 위해서는 해당 $t(i)$ 시간 명령어 전 $t(i-1)$/후 $t(i+1)$의 정보가 필요하다.

3.3 ILCT의 제반 문제 처리

명령어 연관성의 개념은 그림 3과 같이 공간적 및 시간적으로 선/후 관계인 명령어 연관성을 연속적으로 확인함을 기본 개념으로 한다. 그러나 명령어 연관성이 단절되는 모든 지점을 보안에 문제가 발생한 지점으로 단정 지을 수는 없다. 프로그램의 시작점의 경우 이전 명령어가 존재 하지 않으므로 불가피하게 명령어 연관성의 단절을 야기하고, 분기 명령어의 경우 프로그램의 선/후 관계에 영향을 미치므로 명령어 연관성이 단절될 수 있는 여지를 제공한다. 즉, 명령어 연관성 단절 지점은 보안성 측면에서 볼 때 보안 취약점으로 사전에 이에 대한 처리를 필요로 한다.

Ⅳ. ILCT을 통한 보안 대책

ILCT에서는 현재 시간 $t(i)$의 명령어를 실행하기 위해서는 $t(i-1)$의 명령어와 $t(i+1)$의 명령어의 무결성이 확보되어야 하고, 이러한 연관성은 해당 프로그램의 마지막까지 그 연관성이 보장되어야 한다. 그림 3에 의하면 시간 $t(i)$의 parity는 공간적 및 시간적 선/후의 명령어와 연관성을 가지고 있

고프로그램 내의 하나의 명령어의 연관성에 문제가 생기면 프로그램 전체의 연관성에 문제가 발생하게 되는 구조이다.

4.1. 소프트웨어적 보안 위협 상황 대응

소프트웨어적 공격 기법의 대표적 기법으로 악성 코드를 주입하는 보안 위협을 가정한다. 공격자는 보안 위협 성공을 위해는 해당 시스템에서 어떤 연관성 유도 연산 알고리즘을 사용하였는가를 알고 있어야 한다. 여기서는 이는 공개가 되어 있다고 가정한다. 이유는 암호화 기법이 적용된 시스템과 비교하면 공격자가 해당 암호화 알고리즘을 알고 있는 경우로 볼 수 있다. 암호화 기법이 적용된 시스템의 보안 유지 기법은 암호화에 사용된 암호화-키를 효과적으로 은닉하는 것이다. 이와 마찬가지로 오류 정정 부호화 연산에 사용되는 여러 연산 인자들을 은닉한다면, 암호화 기법을 적용한 시스템과 동일한 효과를 기대 할 수 있다[5]. 예를 들면, 그림 3의 parity연산에서 이 연산을 위한 단위 코드의 위치 또는 크기 및 오류 정정 부호화 코드의 사양을 은닉함으로써 악성 코드의 주입을 위한 parity 연산을 암호 해독 연산 수준의 산술적으로 불가능하게 할 수 있다[5]. 또한 악성 코드를 위한 parity를 획득했다고 하더라도 악성 코드로의 진입을 위해서는 그 진입 시점 이전 명령어의 parity들의 변경을 필요로 한다. 이와 같은 보안 위협을 성공하기 위해서는 해당 시스템에 대한 인증과 해당 프로그램에 대한 정확한 정보의 획득이 필요함을 의미한다. 이는 시스템의 보안 정책이 이미 파괴된 상황으로 어떠한 보안회 기법을 통해서도 그 보안 위협을 방지할 수 없는 상황과 같다.

4.2. 하드웨어적 보안 위협 상황 대응

하드웨어적 기법은 시스템이 구성된 보드에 노출된 버스라인에 대한 도청 및 변조를 통한 보안 위협을 가정한다[4]. 만일 공격자가 그림 2(b)와 같은 명령어 변조를 원한다고 가정하면, 우선 공격자는 메모리로부터 프로세서로 전송되는 명령어를 버스라인을 통해 별도의 장비를 이용하여 획득할 것이다. 이 획득된 명령어에는 현재 시점 $t(i)$에서 전송되는 명령어와 메모리 주소 그리고 그에 해당하는 parity들이다. 다음으로 공격자는 해당 명령어를 변조하고 그에 따른 parity들을 변조하려고 할 것이다. 이 과정에서 공격자가 parity연산을 위한 연산 과정을 모르고 있는 경우를 가정하면, 공격자는 강제 주입 공격(brute force attack)을 parity들에 가하여 그 값을 추론 하려 할 것이다. 명령어 연관성 측면에서 보면 이와 같은 공격은 논리적, 확률적 및 산술적으로 불가능하다[5][7]. 즉, 현재 시점의 명령어를 변조하기 위해서는 이전 시점 $t(i-1)$의 명령어의 parity들이 변조 되어야 하고, 이는 다시 그 이전 시점 $t(i-2)$의 parity들의 변조를 필요로 한다. 즉, 프로그램의 명령어 연관성이 끝나는 지점까지 계속 반복 되어야 한다. 프로그램의 명령어 연관성이 끝나는 지점은 프로그램의 시작점을 의미할 수도 있고 메모리의 시작점을 의미할 수도 있다. 이와 같은 시도를 통해서 공격이 성공하는 확률적 시간을 계산해 보면 다음의 식(3)과 같다.

$$\frac{I_{th} \cdot 2^{Paritybit \cdot I_{th}}}{InstructionFetchRate}\ [\mathrm{sec}] \qquad \text{식(3)}$$

식(3)에서 I_{th}는 공격 대상이 되는 명령어가 명령어 단절점으로부터 몇 번째 명령어 인가를 의미하고, InstructionFetchRate은 해당 명령어의 parity를 변경해서 다시 실행 시키는 경우 그 수행 시간을 부여하기 위한 인자이다. 즉, 변조 대상 명령어가 버스라인에서 발생하기까지 프로그램 상에서 명령어의 위치와 프로세서의 수행 속도를 나타낸다. 만일 명령어 간에 연관성이 고려되지 않은 경우 parity들이 8-bits로 구성되어 있다고 가정했을 때 강제 주입 공격을 통해 각 명령어를 parity들을 얻기 위해서는 $2^8 \cdot 2^8 = 65{,}536$회의 비교적 적은 시도로 그 값을 추론 할 수 있다는 문제가 발생한다. 이 문제를 해결하기 위해 프로그램의 시작점을 보안 처리 과정에서 이 연관성 단절점을 보하기 위하여 몇 개의 의미 없는 명령어(NOP: No Operation Instruction)를 삽입하고 실제 프로그램의 시작점을 그 후에 위치시키는 방법으로 해결 가능하다.

식(3)에 따르면 parity의 길이는 크면 클수록 확률적으로 안전해지지만 메모리 오버헤드를 고려했을 때 보안 프로세서의 처리 단위를 고려하여 적절한 크기로 설정되어야 한다. 식(3)을 이용하여 32-bits, 64-bits, 128-bits의 처리 능력을 가지는 프로세서를 가정하여 parity를 16bits와 8bits로 가정하였을 때 메모리 오버헤드와 최초 안전성 확보 지점을 계산할 수 있다. 표 1에서 보는 바와 같이 5번째 이후의 명령어는 확률적 시간으로 그 공격이 불가능하다는 결과를 가짐을 알 수 있다. 물론 이 과정에서 프로그램 코드의 증가와 수행 속도의 저하가 발생

표 1. 메모리 오버헤드와 최초 안전성 확보 지점

Processor Instruction Fetch Rate 1000 M [Inst./sec]						
Parity bits	16bit			8bit		
Processor bits	32bits	64bits	128bits	32bits	64bits	128bits
메모리 오버헤드	50%	25%	12.50%	25%	12.50%	6.25%
최초 안전성 확보 지점	5번째 명령어			9번째 명령어		
공격 성공 시간 [sec]	6.04E+15			4.25E+13		
부가 내부 메모리 [bytes]	30	50	90	45	81	153

된다. 하지만 그 크기는 매우 미약하다[5]. 표 1에서 메모리 오버헤드는 프로세서의 처리 bits에 비례하여 단순 연산된 오버헤드로 실제 프로그램의 보안화 처리 과정에서 부가되는 명령어 코드로 인하여 실제 메모리 오버헤드는 약간 더 증가한다. 프로세서의 처리 단위와 최초 안전성 확보 지점간의 관계는 프로그램의 시작점을 은닉하는데 필요한 내부 실행 메모리의 크기와 관련이 있다.

V. 보안 프로세서의 성능 측정 및 비교

ILCT을 적용한 보안 프로세서의 성능을 측정하기 위하여 명령어 연관성을 확인 하는 모듈이 기존 프로세서와 L1-cache사이에 위치하는 구조의 보안 프로세서가 구성 되었다. 이의 구성을 위해서 Simplescalar3.0을 사용하였고 성능의 측정은 SPEC2000 벤치마크프로그램이 적용되었다. 또한 명령어 연관성 연산에 사용된 오류 정정 코드는 한 심벌이 8-bits로 구성된 Reed-Solomon코드이다[6].

명령어 연관성을 확인 하는 모듈이 기존 프로세서와 L1-cache사이에 위치하여 매 명령어 fetch시 그 연관성을 확인하게 된다. 이는 완전한 파이프라인 구조로 프로세서의 마이크로-아키텍처 파이프라인에 동기화되어 동작하도록 구성 되었다. 명령어 연관성 모듈은 Reed-Solomon연산과정의 중 오류 검출 연산만을 사용하므로 전체 시스템 측면에서 무시 가능한 정도의 하드웨어 오버헤드를 발생 시킨다[5]. 보안 프로세서의 동작 성능 측정을 위하여 표 2와 같은 사양의 프로세서 모델이 사용되었다. 프로세서 모델에서는 분기 예측(branch prediction) 기법으로 항상 분기 명령어의 조건식을 항상 taken으로 예측하는 기법을 적용하였다. 이는 분기 명령어(연관성 단절점) 처리를 위해 삽입된 NOP로 발생 가능한 프로세서 동작 지연을 최대한 유도하기 위함이다[5]. 측정된 결과를 정리하면 표 3과 같다. 표 3에 따르면 프로그램의 보안화 과정에서 발생한 메모리 오버헤드는 크게는 27.07%에서 작게는 26.41%까지 측정되었다. 또한 각 프로세서 모델에

표 2. 성능 측정용 ILCT 적용 보안 프로세서 모델 사양

Simplescalar 3.0 PISA Architecture Model			
Parameters	Processor Model		
	1-issue	4-issue	8-issue
Fetch queue size	1	4	8
Decoder width	1	4	8
Issue width	1	4	8
Commit width	1	4	8
ALU	1	4	8
Memory port	8	8	8
L1 I-cache	16 kbytes	16 kbytes	16 kbytes
L1 D-cache	16 kbytes	16 kbytes	16 kbytes
L2 unified cache	256 kbytes	256 kbytes	256 kbytes
P-cache	2 kbytes	2 kbytes	2 kbytes
Branch prediction	taken	taken	taken
Parity 1 + Parity 2	8 bits	8 bits	8 bits

표 3. 성능 측정 결과

각 벤치마크 프로그램에서 측정된 오버헤드 결과 (%)						
Benchmark	1 issue		4 issue		8 issue	
	Memory	CPI	Memory	CPI	Memory	CPI
gzip	26.43	0.4	26.43	0.23	26.43	0.62
parser	27.07	0.71	27.07	0.58	27.07	1.52
vortex	26.56	2.43	26.56	3.7	26.56	3.74
twolf	26.41	1.27	26.41	1.66	26.41	1.98
평균	26.62	1.20	26.62	1.54	26.62	1.97

따른 CPI 증가량은 크게는 3.74%에서 작게는 0.23%로 측정되었다. 또한 issue의 수가 증가함에 따라 CPI 오버헤드가 증가하는 경향을 보이는 프로그램들이 있으나, 보안화 과정에서 삽입된 NOP의 동작 지연을 최대로 하기 위하여 설정한 분기 예측 설정(항상 taken)의 영향으로 발생한다.

표 4는 근래에 제안된 암호화 기법을 적용한 보안 프로세서와 ILCT이 적용된 보안 프로세서와의 성능 비교를 나타낸다. 이상의 측정된 성능 측정 결과를 통해 ILCT가 적용된 프로세서의 시스템 오버헤드는 기존 제안된 보안 프로세서[2]의 오버헤드 보다 적으며, 그 발생한 오버헤드도 프로세서의 보안화를 위해서는 허용 가능한 수치라 할 수 있다.

VI. 결 론

본 논문에서는 통신 시스템의 오류 정정 연산 기법을 적용하여 임베디드 시스템에 가해지는 보안

표 4. 기존 암호화 기법을 적용한 보안 프로세서와의 성능 비교

성 능 비 교		
	명령어 연관성 기법	[2]
메모리 오버헤드	평균 26.62%	34.6% ~ 86.6%
CPI 증가율	1.20% ~ 1.97%	평균 6.4%

공격에 대한 대응으로 비정상적으로 변조된 명령어의 실행을 방지할 수 있는 새로운 기법인 명령어 연관성 기법을 제안하고 그 안전성 및 효용성에 대하여 논하였다. 성능 측정 결과에 따르면 ILCT동작을 위해 시스템에 요구되는 오버헤드는 메모리 측면에서 평균 26.62%의 부가 필요로 하고 보안 프로그램 구동 시 1.20%~1.97%의 CPI 증가가 발생하였다. 이는 기존 연구들에서 요구하는 오버헤드보다 감소된 수치이다.

참 고 문 헌

[1] J.P. McGregor, et al., "A Processor Architecture Defense against Buffer Overflow Attacks", *Proc. IEEE Int. Conf. Information Technology: Research and Education (ITRE)*, pp. 243-250, Aug. 2003.

[2] A. Murat Fiskiran, Ruby B. Lee, "Runtime Execution Monitoring (REM) to Detect and Prevent Malicious Code Execution", *ICCD* 2004, pp. 452-457, Oct. 2004.

[3] Andrew Huang, "Keeping Secrets in Hardware: The Microsoft XboxTM Case Study", *Cryptographic Hardware and Embedded Systems - CHES 2002*, pp. 213 - 227, Aug. 2002.

[4] Irving S. Reed and Xuemin Chen, "Error-Control coding for Data Network", *Kluwer Academic Publishers*, 1999

[5] R. B. Lee, et al. "Enlisting hardware architecture to thwart malicious code injection", *In Proceedings of the 2003 International Conference on Security in Pervasive Computing*, 2003.

[6] S. Ravi, et al., "Security in Embedded Systems: Design Challenges" *in ACM Transactions on Embedded Computing Systems*, 2004

[7] Milena Milenkovi, et al., "A framework for trusted instruction execution via basic block signature verification" *ACM Southeast Regional Conference Proc. of the 42nd annual Southeast regional conference*, pp. 191-196, 2004.

이 승 욱 (Seung Wook Lee) 정회원

2000년 2월 성균관대학교 전기전자컴퓨터공학부 졸업

2002년 2월 성균관대학교 전기전자컴퓨터공학과 졸업 석사

2006년 2월 성균관대학교 전자전기공학과 졸업 박사

<관심분야> 임베디드시스템, SoC 설계, secure processor architecture. 상위수준 설계

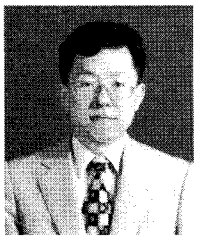
권 순 규 (Soongyu Kwon) 준회원

2008년 2월 성균관대학교 정보통신공학부 전자전기공학과 졸업

2008년 3월~현재 성균관대학교 전자전기컴퓨터공학과 석사과정

<관심분야> 임베디드시스템, 상위수준 설계

김 종 태 (Jong Tae Kim) 정회원



1982년 2월 성균관대학교 전자공학과졸업

1987년 6월 University of California, Irvine Department of Electrical and Computer Engineering 졸업 석사

1992년 12월 University of California, Irvine Department of Electrical and Computer Engineering 졸업 박사

1995년 3월~현재 성균관대학교 정보통신공학부 교수

<관심분야> 임베디드시스템, SoC 설계