情報保護學會論文誌
第18卷 第2號, 2008. 4

응용프로그램 역분석 방지를 위한 코드블록 암호화 방법*

정 동 우[1†], 김 형 식[1‡], 박 중 길[2]
[1]충남대학교, [2]한국전자통신연구원 부설연구소

A Code Block Cipher Method to Protect Application Programs From Reverse Engineering*

Dong-Woo Jung[1†], Hyong-Shik Kim[1‡], Joong Gil Park[2]
[1]Chungnam National University, [2]The Attached Institute of ETRI

요 약

실행코드의 변조와 역분석(reverse engineering)을 방지하기 위한 대표적인 방법은 실행코드를 암호화하는 것이다. 본 논문에서는 키체인(key chaining) 방식의 블록암호화 기법을 이용하여 응용프로그램을 암호화하는 방법을 제안한다. 키체인 방식의 블록암호화 기법은 키가 블록의 내부에 은닉되어 있고 각 블록의 키가 서로 다르다는 장점을 갖지만, 제어이동을 필요로 하는 프로그램에 적용하기에는 적합하지 않다고 알려져 있다. 본 논문에서는 실행코드에서의 제어이동 명령어에 대해서도 키체인 방식을 효과적으로 적용할 수 있도록 블록을 변형시키거나 중복시키는 방법을 제시하고, MIPS 명령어집합을 이용하여 가능성을 분석한다.

ABSTRACT

One of the typical methods to prevent tampering and reverse engineering on executable codes is to encrypt them. This paper proposes a code block cipher method based on key chaining to encrypt the code. The block cipher by key chaining has been known to be inadequate for encrypting the code with control transfer, even though the key chaining has advantage of hiding the keys in blocks and making the individual keys different from block to block. This paper proposes a block transformation and duplication method to apply the block cipher by key chaining to the executable codes with control transfer instructions, and shows the idea works with the MIPS instruction set.

Keywords : code block cipher method, key chaining, application program protection, reverse engineering

Ⅰ. 서 론

최근 정보처리체계에서 인터넷에 대한 의존도가 커지면서 악성코드의 종류가 다양해지고 악성코드에 의한 피해도 커지고 있다. 악성코드는 바이러스, 웜, 트로이목마, 스파이웨어 등으로 구분되며 감염대상 코드의 실행구조를 변경하거나 내부구조를 바꿔서 대상 코드를 수행하기 전후에 특정 기능을 수행하는데 보통 자기복제 능력을 갖는다. 인터넷에의 의존도가 커짐에 따라 악성코드의 전파 속도도 증가하는 추세에 있다.

또한, 역어셈블러(disassembler)나 역컴파일러(decompiler) 등 소프트웨어 역분석을 위한 분석도구들이 많이

접수일: 2007년 12월 26일; 채택일: 2008년 3월 16일
* 본 연구는 지식경제부 및 정보통신연구진흥원의 대학IT연구센터 지원사업의 연구결과로 수행되었음 (IITA- 2008-C1090-0801-0016)
† 주저자, dwjung@csal.cnu.kr
‡ 교신저자, hkim@cnu.kr

출현하면서 이러한 도구들을 이용하여 기계어 코드 수준에서 공격하는 형태도 일반화되고 있다. 기계어 코드 수준에서 프로그램을 분석할 수 있다는 것은 소스 코드 수준에서처럼 사용자의 필요에 의하여 기존 코드를 변형하거나 임의 코드를 삽입할 수 있다는 점에서 공격의 수단으로 활용될 위험이 발생되기 때문에 이러한 역분석을 효과적으로 차단하기 위한 방법이 필요하다.

본 논문에서는 실행코드를 보호하기 위해 실행코드를 블록암호화하는 방법을 제안한다. 실행코드를 암호화하면 키를 알지 못하는 상태에서는 실행코드의 원래 내용에 접근할 수 없으므로 역분석을 원천적으로 방지할 수 있다. 더 나아가 실행코드를 암호화하는데 있어 보안의 강화를 위해 두 가지 관점을 더 고려할 것이다. 한 가지는 실시간 복호화이고 다른 한 가지는 암호블록마다 서로 다른 키를 갖도록 하는 것이다.

첫째, 암호화한 실행코드는 실시간으로 복호화할 수 있도록 한다. 암호화한 실행코드는 시스템에서 인식할 수 있도록 실행 시 복호화를 해야 하는데, 실행코드를 실행하기 전에 그 전체를 복호화한 후 실행하는 것은 문제가 있다. 실행코드 전체를 미리 복호화 한다면 복호화한 내용이 메모리에 보존되므로, 디버거와 같은 도구를 이용하면 복호화된 실행코드 전체의 내용을 손쉽게 얻을 수 있기 때문이다. 이에 본 논문에서는 암호화된 실행코드를 실행할 때, 실행코드 중 현재 실행할 부분만 복호화하는 것이 가능하도록 하는 암호화 방법을 제시하여, 실행코드가 외부로 공개되는 것을 최소화하려한다.

둘째, 실행코드를 암호화하는데 있어 서로 다른 키를 사용하도록 한다. 실행코드의 모든 블록들을 동일한 키를 사용하여 암호화하면, 암호화된 블록들을 이용하여 키를 추측해 볼 수 있으며 키를 알아냈을 경우 모든 실행코드의 내용을 볼 수 있게 된다. 따라서 각 블록마다 서로 다른 키를 사용하여 암호화함으로써 위의 취약점들을 방지할 수 있다.

위의 두 가지 사항을 충족하기 위해 본 논문에서는 키체인(key chaining) 방법을 이용한다. 키체인을 이용하면 각 블록마다 서로 다른 키를 가지도록 할 수 있으며, 키 자체를 블록 안에 은닉할 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 소프트웨어 역분석 방지를 위한 기존의 방법들에 대해 간단히 소개한다. 3절에서는 키체인과 실시간 복호화를 지원하는 암호화 알고리즘을 설명하는데 필요한 기본 지식에 대해 설명한다. 그리고 4절에서는 키체인 방식에서 실시간 복호화를 지원하는 암호화 알고리즘을 제시한다. 5절에서는 실시간 복호화를 지원하기 위한 암호화모듈들을 구현할 때, 어떤 모듈들을 어떤 순서로 적용해야하는지 설명한다. 또한 실제 MIPS 코드의 예를 들어 블록중복에 의해 일어나는 코드의 변화를 보인다. 마지막으로 6절에서는 본 논문에서 고안한 암호화 알고리즘을 실제 실행코드에 적용하여, 실시간 복호화를 지원하기 위해 원래의 실행코드의 크기가 얼마만큼 증가하는지 측정한다.

Ⅱ. 관련 연구

소프트웨어 역분석 방지를 위한 방법은 사용하는 목적에 따라 그리고 활용할 수 있는 기술 수준에 따라 다양한 방법들이 제안되었다. 그 중 잘 알려진 대표적인 방법들은 다음과 같이 분류할 수 있다[1][2].

첫째, tamper-proofing 기법이다. 기계어 코드를 변형하는 것을 방지하기 위하여 기계어 코드가 변형 여부를 검사하여 변형되었을 경우 모든 기능 혹은 특정 기능을 제한하는 방법이다. Tamper- proofing은 프로그램에 내장된 방법으로 구현할 수도 있지만 운영체제의 도움을 받아 별도의 외부 서비스로 구현할 수도 있다[3][4][5].

둘째, obfuscation 기법이다. 프로그램의 설계 방식이나 내부를 보호하기 위하여 역분석을 어렵게 만드는 방법으로 구문 분석에 의한 방법, 제어 변조에 의한 방법, 데이터 변조에 의한 방법, 레이아웃 변조에 의한 방법 등이 있다. 예를 들어서 제어 변조에 의한 방법은 제어 조건을 판별하는 방법을 고의적으로 복잡하게 구성하고 불필요한 조건을 포함시킴으로써 자동화된 분석 도구가 제대로 동작하지 못하도록 방해한다[6][7][8][9].

셋째, 암호화 기법이다. 프로그램을 암호화함으로써 외부로부터의 공격을 원천적으로 방지할 수 있다. 암호화된 프로그램을 역분석한 결과는 의미 없는 값을 지니기 때문이다. 암호화된 프로그램에 접근하기 위해서는 암호화 알고리즘과 키를 알아야한다. 또한 다양한 암호화 알고리즘이 존재하기 때문에 암호화 목적에 맞는 알고리즘을 선택하여 사용하여야 한다.

앞서 열거한 역분석을 방지하기 위한 방법들은 소프트웨어적으로 구현할 수 있다. 그렇지만 이러한 소프트웨어적 구현에 하드웨어의 지원을 추가함으로써 그 기능을 향상시킬 수 있다. 외부에 공개해서는 안 되는 정보나 빠른 속도를 요구하는 알고리즘 등을 하드웨어 모

듈에 삽입함으로써, 데이터를 보호하는 동시에 속도 향상을 꾀할 수 있다.

Ⅲ. 키체인 방법을 위한 프레임워크

본 논문에서는 선행블록에서 정보를 얻어와 키를 생성하여 현재 블록에 적용하는 키체인 방식을 택한다. 키체인을 이용하면 실행코드 안의 제어이동명령어(control transfer instruction)로 인해 선행블록이 두 개 이상 존재하여 현재 블록이 취할 수 있는 키가 두 개 이상일 경우가 발생하는데 이 문제를 회피하기 위해 현재 블록을 중복하는 방법을 이용한다. 즉, 블록암호화를 하기 전에 선행블록이 두 개 이상 존재하는 문제를 해결하기 위해 실행코드블록을 중복하고 분기명령어를 삽입해야하는데, 이러한 연산들을 단순화하기 위해 실행코드를 기본블록으로 구분하고 이것을 다시 단위블록으로 변환하는 방법을 채택한다.

3.1. 키체인의 구조

[그림 1]은 키체인 방식의 블록암호화를 보인다. N번째 평문블록N을 암호화할 때는, N-1번째 평문블록(N-1)을 이용하여 키를 생성하고 이 키를 이용하여 N번째 평문블록N을 암호화한다. 즉, 선행하는 평문블록에서 키를 생성하고, 그 키를 이용하여 현재의 평문블록을 암호화하는 방식이다. 초기키는 제일 첫 블록을 암호화할 때만 키로 사용하고 별도로 관리한다.

초기키는 프로그램의 실행권한을 부여하는 목적으로 활용된다. 프로그램에서의 진입점에 해당하는 블록은 항상 일정하고 이 블록을 복호화하기 위하여 사용되는 키가 초기키인데, 일단 초기키를 확보하면 키체인을 따라서 다음에 필요한 키를 연쇄적으로 계산할 수 있다는 점에서 초기키가 프로그램에 대한 실행권한을 제어하는 유일한 수단이 된다.

초기키를 제공하고 관리하는 방법은 프로그램에 대한 실행권한을 제어하는 메커니즘의 핵심 기능이다. 프로그램 공통의 초기키는 배포할 때마다 별도로 생성할 수도 있고, 동일한 키로 고정될 수도 있다. 또한 초기키를 배포하는 방식은 기존의 공개키 방식을 활용하여 프로그램 사용자에게 제공할 수도 있고, TPM(Trusted Platform Module)[10]이나 USB와 같은 별도의 하드웨어를 이용하여 관리하는 방법도 있다. 자세한 구현은 본 논문의 범위에서 벗어나기 때문에 여기서는 다루지 않기로 한다.

3.2. 기본블록과 단위블록

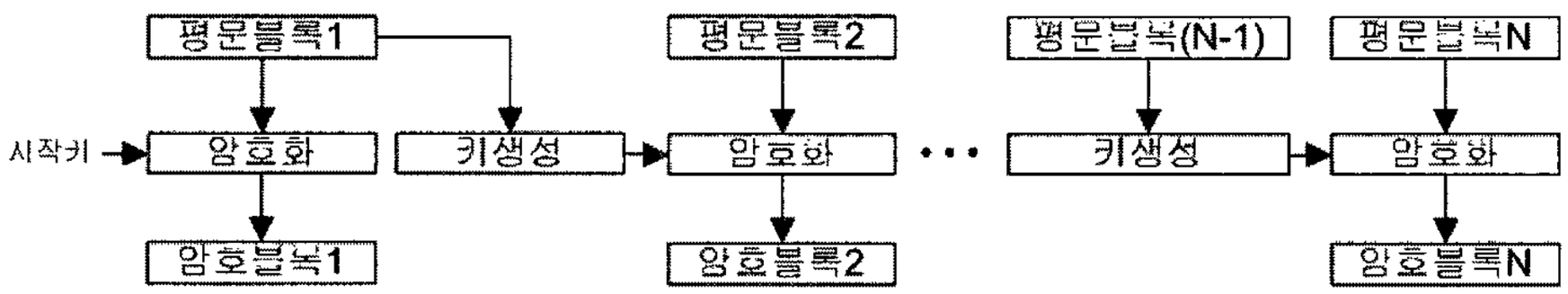
3.2.1. 기본블록

기본블록(basic block)은 "명령어의 연속으로 이루어진 블록으로, 제어의 입출력이 첫 명령어와 마지막 명령어로만 제한되는 블록"이다. 프로그램이 실행되는 도중에 분기나 점프 명령어와 같은 제어이동명령어를 만나면, 제어의 흐름이 바뀌어서 바로 다음 명령어가 실행되지 않고 목적지 주소에 명시된 주소로 제어가 전이하여 그 곳의 명령어가 실행된다. 따라서 기본블록은 이러한 제어이동명령어를 이용하여 나눌 수 있다.

실행코드를 기본블록으로 구분하는 것이 암호화를 위해 반드시 필요한 작업은 아니지만, 이를 통하여 실행코드에 대한 변환 과정을 단순화할 수 있다. 블록을 암호화하면서 발생되는 문제를 해결하기 위해서는 실행코드를 중복하거나 명령어를 추가할 필요가 있는데, 기본블록은 이런 작업을 수행하기에 적절한 단위가 된다.

3.2.2. 단위블록

앞에서 제시된 기본블록을 바로 블록암호화에 적용하기에는 어려움이 있다. 기본블록은 제어이동명령어를 기준으로 분리하였기 때문에 각 블록은 그 크기가 다르기 때문이다. 따라서 블록연산을 위하여 기본블록을 일정크기블록의 정수배로 변환할 필요가 있다. 여기서 일



〔그림 1〕 키체인 방식의 블록암호화

정크기의 블록을 단위블록이라 한다.

기본블록을 단위블록으로 변환하는 이유는 다음과 같다.

1) 안전한 키 생성

키체인 방식의 암호화를 적용할 때 기본블록을 이용하여 암호화키를 생성하면 보안에 문제가 발생할 수 있다. 기본블록의 크기가 충분히 크면 문제가 없지만, 그 크기가 작을 경우 심지어는 1이 될 때도 있기 때문이다. 만약 no operation(NOP) 명령어 하나로 이루어진 기본블록이 있다면 생성된 암호화키는 보안에 취약하게 된다.

이때 단위블록으로 변환을 하면 일정한 크기의 블록이 생성되고, 또한 기본블록의 크기가 작은 경우는 빈 공간을 메우기 위해 임의의 명령어(dummy)를 생성함으로써 암호화키의 보안을 더욱 강화할 수 있다.

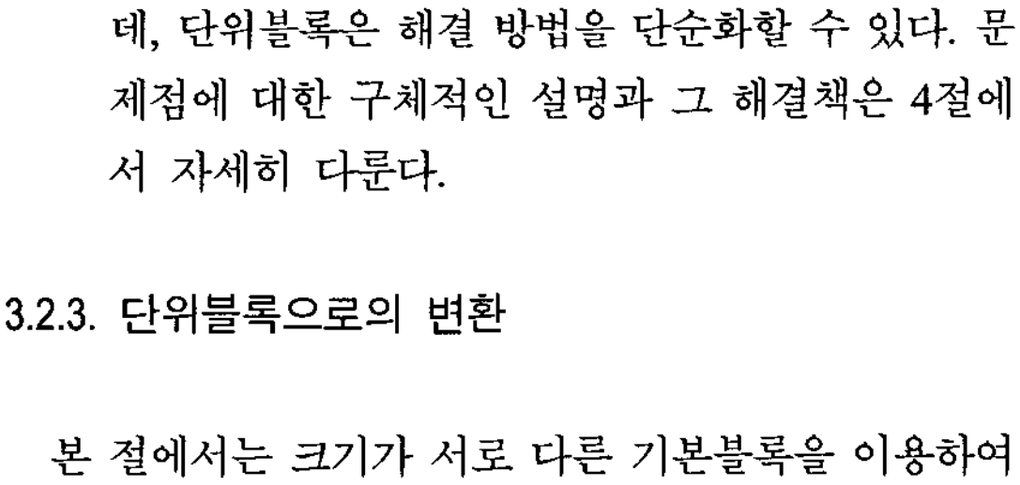
2) 분기로 인한 문제 해결에 도움

키체인 방식으로 키를 생성할 때, 분기나 점프 명령어를 이용하면 한 블록으로 두 개의 제어의 흐름이 들어오는 경우가 발생한다. 이때 여러 선행 블록 중 어떤 블록의 정보를 이용해야 하는가의 문제가 발생하고 이에 대한 해결 방법이 필요한데, 단위블록은 해결 방법을 단순화할 수 있다. 문제점에 대한 구체적인 설명과 그 해결책은 4절에서 자세히 다룬다.

3.2.3. 단위블록으로의 변환

본 절에서는 크기가 서로 다른 기본블록을 이용하여 일정크기의 단위블록으로 만드는 방법을 설명한다. [그림 2]는 기본블록을 크기가 4인 단위블록으로 변환하는 것을 도시한 것이다.

기본블록을 감싸고 있는 크기가 일정한 각 블록들이 단위블록이고, 그 크기에 맞게 기본블록을 확정하는 것이 단위블록으로의 변환이다. 한 단위블록 안의 개수를 일정하게 하기 위해, 기본블록에 임의의 명령어를 추가하거나, 때에 따라서는 기본블록을 두 개 이상의 단위블록으로 분할할 필요가 있다.



〔그림 2〕 단위블록으로 변환

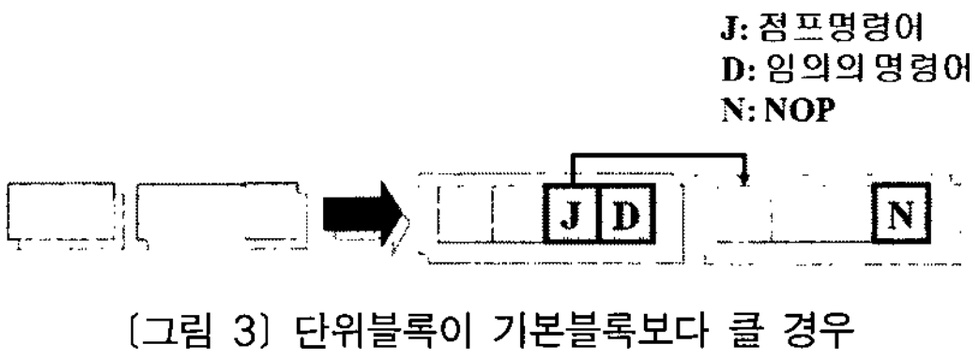
기본블록을 단위블록으로 변환할 때는 크기에 따라 세 가지 경우가 발생된다.

1) 단위블록과 기본블록의 크기가 같을 경우

기본블록이 그대로 단위블록이 된다

2) 단위블록의 크기가 기본블록의 크기보다 클 경우

빈 공간을 하나의 점프 명령어와 하나 이상의 임의의 명령어로 메운다. [그림 3]은 단위블록이 기본블록보다 클 경우 변환하는 방법을 보인다. 화살표의 왼쪽은 변환하기 전의 기본블록이고, 화살표의 오른쪽은 단위블록으로 변환한 것이다.

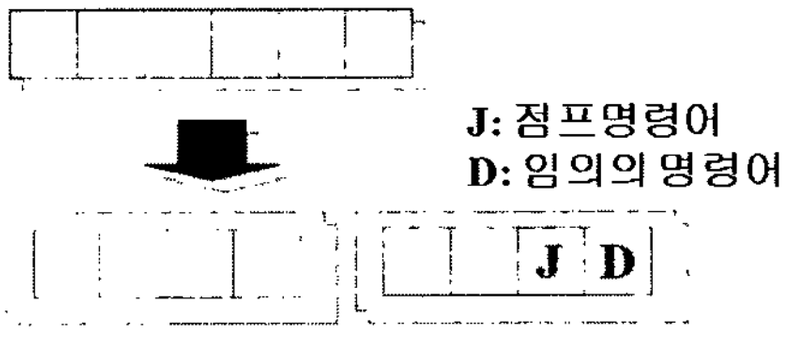


〔그림 3〕 단위블록이 기본블록보다 클 경우

빈 공간을 메우는 방법은 다음과 같다. 우선 기본블록을 단위블록에 대입하고 빈 공간이 하나 남을 경우(그림 3의 두 번째 단위블록 참조)에는 NOP를 대입한다. 그렇지만 빈 공간이 두 개 이상 남을 경우(그림 3의 첫 번째 단위블록 참조)에는 제일 첫 빈 공간에는 점프 명령어를 채우고 나머지 공간에는 임의의 명령어를 채운다. 마지막으로 제어의 흐름이 임의의 명령어로 가지 않도록 하기 위해, 점프 명령어의 목적지주소를 다음 단위블록의 첫 명령으로 설정한다.

3) 단위블록의 크기가 기본블록의 크기보다 작을 경우

기본블록을 두 개 이상의 단위블록에 배치한다. [그림 4]는 기본블록이 두 개 이상의 단위블록으로 분할하는 방법을 보인다. 화살표 윗부분이 변환하기



〔그림 4〕 단위블록이 기본블록보다 작을 경우

전의 기본블록이고, 화살표 아랫부분이 단위블록으로 변환한 이후에 기본블록이 단위블록의 연속으로 분할된 형태이다.

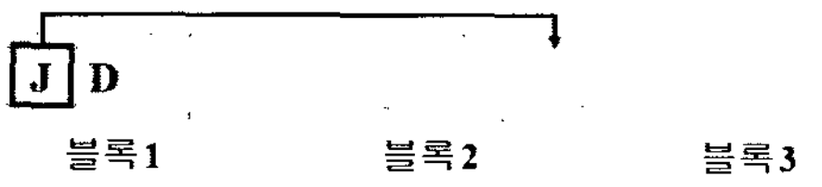
기본블록의 크기가 큰 경우 여러 개의 단위블록으로 분할하는데 이 때 단위블록의 개수는 다음과 같이 계산된다.

$$\text{사용될 단위블록의 개수} = \lceil \frac{\text{기본블록의 크기}}{\text{단위블록의 크기}} \rceil$$

기본블록을 단위블록의 연속으로 변환하면 마지막 단위블록은 빈 공간을 가지게 된다. 이 빈 공간은 앞에서 설명한 빈 공간을 메우는 방법을 이용하여 처리한다.

3.3. 선형 키체인의 한계

실행코드를 기본블록으로 구분하고 기본블록을 단위블록으로 변환하는 과정을 거치면, 실행코드는 일정한 크기 단위로 변환되어 쉽게 암호화할 수 있다. 이렇게 생성된 단위블록에 대하여 키체인 방식을 적용하여 암호화할 경우 문제가 발생될 수 있다. [그림 5]는 문제를 설명하기 위한 것이다.



〔그림 5〕 선형 키체인의 한계

그림은 제어이동명령어를 포함하고 있는 실행코드의 일부분을 보인다. 각 단위블록은 4개의 명령어를 포함하고 있으며 블록1으로부터 점프 명령어를 통하여 제어를 블록3으로 전이될 수 있음을 볼 수 있다. 즉, 제어이동명령어로 인해 블록3의 선행블록이 두 개가 될 수 있다. 이 때 블록3를 암호화하기 위해서는 선행블록을 결정하여야 하는데 어느 블록을 선행블록으로 보고 키를 생성하여야 하는지의 문제가 발생된다. 본 논문에서는 단위블록의 중복을 통하여 이 문제를 회피하는데 그 방법은 4절에서 자세히 설명된다.

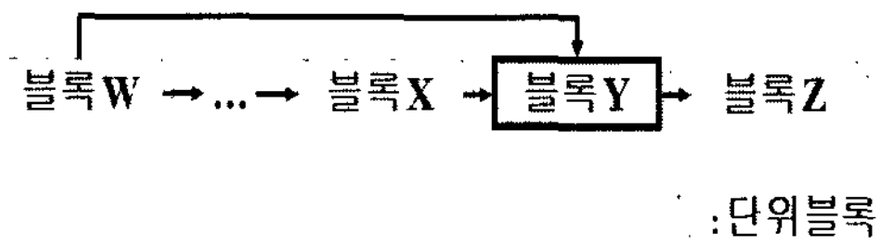
3.4. 실행코드로의 적용 순서

실행코드는 다음의 과정을 거쳐서 암호화된다. 첫째 분기 명령어나 점프 명령어 등의 제어이동명령어 중심으로 실행코드를 기본블록으로 분할한다. 둘째 기본블록들을 단위블록으로 변환한다. 셋째 블록암호화를 하기 전에 제어이동명령어에 의해 발생되는 문제를 회피하기 위해 단위블록을 중복한다. 마지막으로 블록별로 암호화 한다. 다음 절에서는 키가 두 개 이상 생성되는 문제를 해결하기 위해 단위블록을 중복하는 방법에 대해 구체적으로 설명한다.

Ⅳ. 분기명령어를 고려한 키체인 방법

4.1. 단위블록간 제어이동

[그림 6]은 단위블록간의 제어이동 시 선행블록이 두 개 이상일 경우를 보인다. 각각의 사각형은 단위블록을 나타내고, 단위블록사이의 화살표는 제어의 흐름을 나타낸다. 이 그림과 같이 블록Y의 선행블록이 두 개일 경우, 제어의 흐름 역시 두 개(블록W→블록Y, 블록X→블록Y)가 된다. 따라서 이 단위블록들을 키체인 방식으로 암호화를 할 경우, 키 생성 관점에서 블록W와 블록X는 블록Y에게 서로 다른 키를 제공하는 문제가 발생된다.

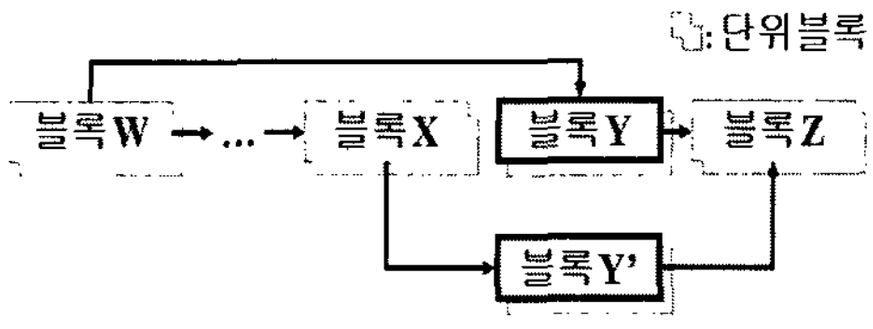


〔그림 6〕 키체인에서의 문제점

블록W가 생성한 키를 이용하여 다음 블록과 블록Y를 암호화한다고 가정하자. 그 후 블록단위로 복호화하면서 프로그램을 실행하면 블록W에서 블록Y로의 흐름은 문제가 없이 실행되지만, 블록X 다음에 블록Y로 제어가 이동하는 경우에는 문제가 발생된다. 왜냐하면 암호화 과정에서 블록Y의 선행 블록이 블록W라고 보았으므로 블록X에서 생성된 키는 블록Y를 복호화하는데 적합하지 않기 때문이다.

4.2. 단위블록 중복 방법

[그림 7]은 블록을 중복하여 문제를 회피하는 방법을 보인다. 굵게 표시된 부분(블록Y, 블록Y')이 단위블록



〔그림 7〕 단위블록의 중복

을 중복하여, 블록W와 블록X에 의해 서로 다른 두 개의 키가 생성되는 것을 해결하기 위한 부분이다. 블록Y에서 생성하는 키와 동일한 키를 생성하도록 블록Y'를 추가한다. 즉 키 생성 관점에서 블록Y와 블록Y'는 동일하도록 블록Y로부터 블록Y'을 생성하고 블록Y의 선행블록중의 하나인 블록X 이후의 제어가 블록Y'으로 이전되도록 조정한다. 이때 블록Y와 블록Y'은 서로 같은 키를 생성하므로 어떤 제어 경로를 따르더라도 블록Z를 위한 키는 항상 동일하다.

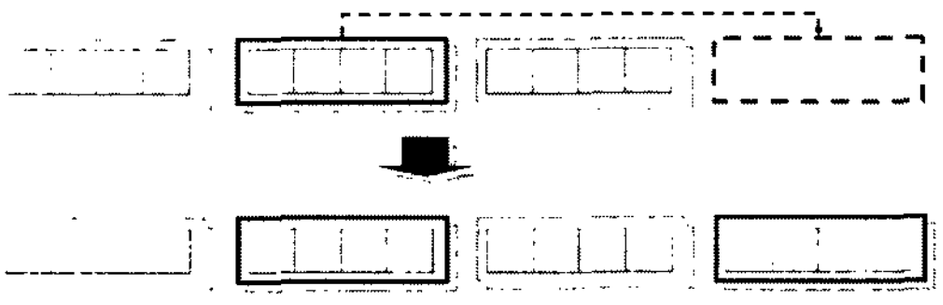
즉, 블록W와 블록X는 서로 다른 키를 생성하여 각각 블록Y와 블록Y'에서 사용되고, 블록Y와 블록Y'은 비록 서로 다른 블록이지만 동일한 키를 생성하여 블록Z에서 사용된다. 앞에서 "키 생성 관점에서 두 개의 블록이 동일하다"는 것은 이 두 개의 블록이 키를 생성하는 부분에서는 정확하게 동일한 내용을 갖게 됨을 의미한다.

4.3. 단위블록 중복을 위한 연산

단위블록을 중복하기 위한 연산에는 블록복사 연산과 제어변환 연산 그리고 블록분할 연산이 있다.

4.3.1. 블록복사

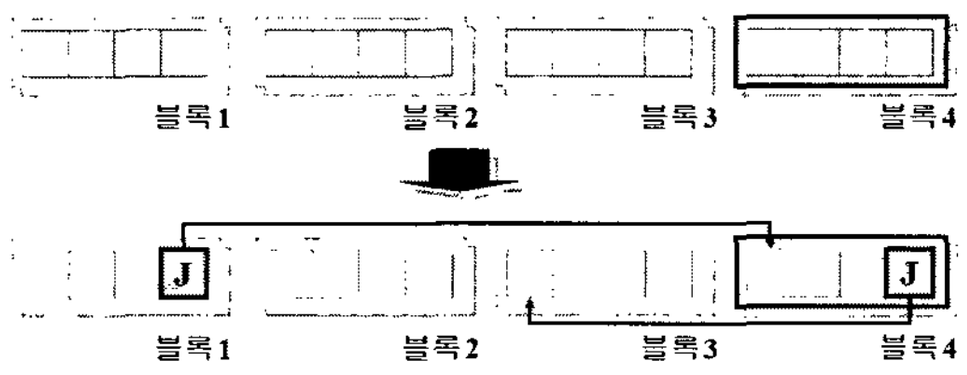
블록복사 연산은 선택한 블록을 프로그램의 제일 끝에 복사한다. 단순히 블록의 내용만 복사하기 때문에 제어의 흐름은 바꾸지 않는다. [그림 8]은 블록복사 연산을 이용하여 블록을 중복하는 예를 보인다. 복사할 블록을 선택하고 블록복사 연산을 취하면 선택한 블록이 프로그램의 제일 끝에 복사된다. 이때 프로그램의 제어는 이동하지 않고 단순히 블록의 내용만 복사된다.



〔그림 8〕 블록복사

4.3.2. 제어변환

제어변환 연산은 특정 블록의 제어를 변환한다. 이 연산은 단위블록을 중복한 후에 수반되는 연산으로, 블록복사 이후 복사한 블록으로 제어가 이동하도록 한다. [그림 9]에 제어변환의 방법을 보인다. 화살표 윗부분의 블록들은 제어변환 연산을 수행하기 전의 블록들의 모습으로 블록복사 연산으로 블록4가 추가된 상태이다. 화살표 아래의 블록들은 제어변환을 수행한 이후의 모습으로 블록4의 제어 흐름이 블록3으로 전이될 수 있도록 변형된 것을 볼 수 있다.



〔그림 9〕 제어변환

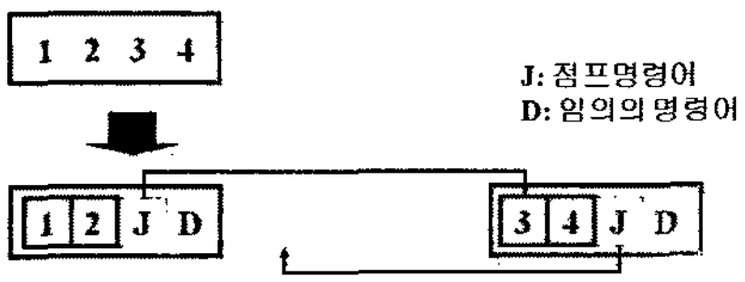
제어를 변환하기 위해서는 블록4와 같이 제어를 변환할 블록을 선택하고, 블록4로 제어를 연결할 선행블록(블록1)과 후행블록(블록2)을 정해야 한다. 제어변환 연산은 블록1과 블록4에 점프 명령어를 삽입하여 블록4의 선행블록과 후행블록으로의 제어를 생성하는 역할을 한다.

각 블록에 점프 명령어를 삽입하는 방법은 다음과 같다. 선행블록으로 만들 블록과 제어를 연결할 블록이 점프 명령어를 포함하고 있다면, 그 점프 명령어의 목적지 주소를 수정한다. 각 블록이 점프 명령어를 포함하지 않는다면, 임의의 명령어(dummy)를 점프 명령어로 대체한다. 만일 블록에 점프 명령어를 삽입할 공간적 여유가 없다면, 다음으로 설명할 블록분할 연산을 이용하여 블록을 분할한 후 제어를 변환한다.

4.3.3. 블록분할

블록분할 연산은 블록 안에 특정한 명령어를 추가하고 싶지만 블록에 여유 공간이 없을 경우 블록을 분할하기 위해 사용한다.

[그림 10]은 블록분할 연산의 결과를 보인다. 블록이



〔그림 10〕 블록분할

분할되는 것을 자세히 보이기 위해, 분할되는 블록의 명령어들에 차례대로 숫자를 붙였다.

블록을 분할하는 방법은 다음과 같다. 우선 프로그램의 제일 뒷부분에 새로운 블록을 생성하고 선택한 블록 안에 있는 명령어들의 일부를 새로 생성한 블록으로 이동한다. 그 후 점프 명령어와 임의의 명령어를 이용하여 원래의 프로그램과 같이 동작하도록 제어의 흐름을 조정한다.

4.3.4. 블록중복으로의 적용

키 생성 문제를 회피하기 위한 목적으로 블록을 중복하기 위해서는 앞에서 설명한 블록복사 연산, 제어변환 연산 그리고 블록분할 연산을 조합하여야 한다. 블록분할 연산의 처리 과정은 다음과 같다.

① 키 생성 문제를 회피하기 위해 중복할 블록을 선택한다.

② '블록복사 연산'을 사용하여 선택한 블록을 프로그램의 제일 마지막으로 복사한다.

③ 복사한 블록의 선행블록과 후행블록을 정한 후 '제어변환 연산'을 사용하여 복사한 블록으로 제어가 이동하도록 관련된 블록들의 제어를 변환한다. 만일 제어변환 연산을 실행할 때 블록에 점프 명령어를 대체하거나 삽입할 공간적 여유가 없다면, '블록분할 연산'을 사용하여 그 공간을 확보한 후 제어변환 연산을 실행한다.

V. 구 현

본 절에서는 앞서 설명한 알고리즘을 이용하여 암호화 코드를 생성하는 방법과 암호화한 코드를 실시간 복호화하는 방법을 설명한다.

5.1. 키체인을 적용하기 위한 프로그램 재구성 절차

프로그램의 실행코드 부분은 암호화 프로그램에 의해 암호화된다. 암호화 과정은 다음과 같은 4단계를 거친다.

1) 기본블록분석 단계
실행코드를 입력으로 받아 기본블록으로 구분한다. 결과물로 기본블록들의 시작과 끝을 가리키는 주소 정보를 생성한다.

2) 변환 단계
기본블록은 제어이동명령어를 중심으로 실행코드를 구분한 것이기 때문에 그 크기가 서로 다르다. 변환모듈은 블록의 조작이 용이하도록 하기 위해, 입력으로 받은 기본블록 정보를 이용하여 일정한 크기의 단위블록으로 변환한다. 결과물로 변환한 단위블록을 생성한다.

3) 블록중복 단계
블록암호화 시 선행블록이 두 개 이상 존재함으로써 두 개 이상의 키를 생성하는 문제를 해결하기 위해 단위블록을 중복시킨다. 변환된 단위블록을 입력으로 받아 문제가 발생하는 블록을 중복시킨다. 결과물로 선행블록으로 인한 문제점이 제거된 단위블록을 생성한다.

4) 블록암호화암호화 단계
선행블록으로 인한 문제점이 제거된 단위블록을 이용하여 실제 블록암호화한다. 실시간 복호화가 가능하도록 제어의 흐름을 따라 단위블록들을 암호화한다. 결과물로 암호화된 실행코드를 생성한다.

5.2. MIPS 코드 예제

본 절에서는 앞서 설명한 암호화를 위한 절차 중 가장 중요한 과정인, 선행블록이 두 개 이상일 경우에 대하여 그 문제를 회피하는 방법을 실제 MIPS 코드에 적용한 예를 보인다.

[그림 11]은 제어이동명령어를 포함한 실행코드를 키체인 방식으로 블록암호화할 경우로서, 제어이동명령어의 목적지에 해당하는 블록에서 키가 두 개 이상 존재하는 경우를 보인다. 예제는 MIPS 명령어를 사용하여 작성되었고 기본블록을 명령어 네 개 크기(16바이트)의 단위블록으로 변환한 후의 결과이다. 그림에서와 같이 블록(4)뿐만 아니라 블록(5)에서도 블록(3)으로 제어가 전이되는 것을 볼 수 있다. 즉 블록(3)을 암호화할 시점에서 서로 다른 두 개의 키를 생성하는 선행블록이 존재하기 때문에 키를 선택하는 문제가 발생한다.

```
(0)   [0x00400000] ori $16, $0, 0
      [0x00400004] ori $17, $0, 0
      [0x00400008] ori $18, $0, 0
      [0x0040000c] nop
(1)   [0x00400010] addi $16, $16, 1
      [0x00400014] addu $17, $17, $16
      [0x00400018] sltiu $8, $16, 10
      [0x0040001c] bgtz $8, -16
(2)   [0x00400020] sltiu $8, $17, 100
      [0x00400024] bgez $8, 24
      [0x00400028] j 0x00400050
      [0x0040002c] dummy
(3)   [0x00400030] ori $2, $0, 10
      [0x00400034] syscall
      [0x00400038] dummy
      [0x0040003c] dummy
(4)   [0x00400040] ori $18, $0, 1
      [0x00400044] j 0x00400030
      [0x00400048] dummy
      [0x0040004c] dummy
(5)   [0x00400050] ori $18, $0, 0
      [0x00400054] j 0x00400030
      [0x00400058] dummy
      [0x0040005c] dummy
```

〔그림 11〕 블록암호화시 문제점

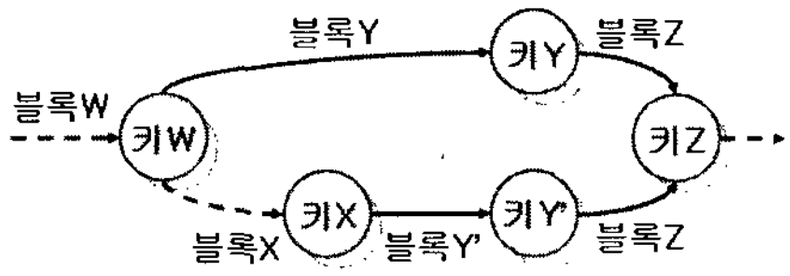
[그림 12]는 단위블록을 중복하는 방법에 대한 예를 보인다. 그림에서는 블록(3)을 블록(6)으로 중복하였다. 또한 블록(5)에서 블록(3)으로 제어가 이동하던 원래의 흐름을, 블록(5)에서 블록(6)으로 이동하도록 변환함으로써 블록(3)의 선행블록이 하나만 존재하도록 하였다. 변환된 제어의 흐름을 살펴보면 블록(3)은 암호화를 위해 블록(4)로부터 키를 얻어오고, 평문블록(6)은 평문블록(5)로부터 키를 얻어온다. 결과적으로 평문블록(3)에서 발생하던 키가 두 개 이상 생성되는 문제가 해결된다.

5.3. 복호화

암호화된 실행코드는 실행하기 전에 복호화되어야 한다. 실행코드를 실행할 때에는 전체 실행코드를 사전에 모두 복호화하지 않고, 실행코드의 실행흐름을 따라가면서 현재 실행할 명령어가 포함되어 있는 암호블록에 대해서만 복호화한다. 다음 블록을 복호화하기 위한 키는 이미 복호화한 현재 블록을 이용하여 생성한다. 4절에서 설명된 키체인 방법에 의하면 분기명령어에 의해 제어의 흐름이 바뀔 경우에도 먼저 실행되는 블록은 다음에 실행될 블록에서 필요로 하는 키를 제공할 수 있다. 즉 [그림 7]에서 모든 블록은 다음에 수행될 블록에 대한 키를 생성할 수 있다. [그림 7]에서의 블록들을

```
(0)   [0x00400000] ori $16, $0, 0
      [0x00400004] ori $17, $0, 0
      [0x00400008] ori $18, $0, 0
      [0x0040000c] nop
(1)   [0x00400010] addi $16, $16, 1
      [0x00400014] addu $17, $17, $16
      [0x00400018] sltiu $8, $16, 10
      [0x0040001c] bgtz $8, -16
(2)   [0x00400020] sltiu $8, $17, 100
      [0x00400024] bgez $8, 24
      [0x00400028] j 0x00400050
      [0x0040002c] dummy
(3)   [0x00400030] ori $2, $0, 10
      [0x00400034] syscall
      [0x00400038] dummy
      [0x0040003c] dummy
(4)   [0x00400040] ori $18, $0, 1
      [0x00400044] j 0x00400030
      [0x00400048] dummy
      [0x0040004c] dummy
(5)   [0x00400050] ori $18, $0, 0
      [0x00400054] j 0x00400060
      [0x00400058] dummy
      [0x0040005c] dummy
(6)   [0x00400060] ori $2, $0, 10
      [0x00400064] syscall
      [0x00400068] dummy
      [0x0040006c] dummy
```

〔그림 12〕 문제점 해결



〔그림 13〕 키 그래프

복호화하는데 필요한 키들의 상관 관계는 [그림 13]와 같다. 블록W를 실행하면서 생성된 키W는 블록Y나 그림에서는 표시되지 않은 다른 블록을 복호화하는데 사용된다. 또한 그림은 키Y와 키Y'이 모두 블록Z를 복호화할 수 있음을 보인다.

Ⅵ. 시 험

실행코드를 키체인 방식으로 암호화하기 위해서는 제어가 이동되는 블록들은 중복되어야 한다. 실행코드에 이런 조작을 가하면 블록의 개수가 증가하게 되는데, 이것은 프로그램을 암호화하는데 소요되는 시간이 늘어나고 실행되는 명령어의 개수가 늘어나는 것을 의미한다.

암호화는 프로그램 실행 횟수와 상관없이 한번만 필요하므로 본 논문에서 제시된 방법이 성능에 미치는 영

향은 제한적이지만 그 정도는 늘어나는 블록의 수와 비례할 것으로 추정된다. 한편 각 블록을 복호화하는 오버헤드는 프로그램을 실행할 때마다 반복되므로 암호화보다는 성능에 미치는 영향이 크지만, 이는 본 논문에서 지향하는 실시간 복호화를 위해서는 불가피하다. 복호화에 수반되는 오버헤드는 두 가지로 분류될 수 있는데, 각 블록을 복호화하는데 소요되는 시간과 암호화 과정에서 추가된 블록을 실행하는데 소요되는 시간이 그것이다. 각 블록에 대한 복호화 시간은 앞에서의 설명처럼 불가피한 측면이 있으므로 여기서는 추가된 블록에 의하여 유발되는 실행시간의 증가 정도를 분석한다.

[그림 7]에서의 암호화 과정에서 블록Y'와 같이 새로운 블록이 추가될 수 있지만, 실제 실행 경로를 보면 블록W에서 블록Y로 제어가 이동되는 경우와 블록X에서 블록Y로 이동되는 경우 모두 실제로 수행되는 블록의 수는 증가하지 않는다. 이것은 정적인 관점에서 보면 블록Y'이 추가되었지만 동적으로 실행될 때는 블록Y'은 블록Y를 대체하는 목적으로 활용되기 때문이다. 그렇지만 단위블록을 중복시키는 과정에서 추가된 블록은 실제 수행될 때도 실행시간의 증가를 유발시킨다. 따라서 실행시간의 증가 정도는 블록의 개수가 늘어나는 정도보다 작다는 알 수 있다.

본 절에서는 제안된 키체인 방식으로 실행코드를 보호하기 위한 오버헤드를 분석하기 위한 목적으로 실제 동작하는 실행코드를 분석하여, 실행코드에 블록암호화를 위한 기법을 적용하였을 경우 코드 크기의 증가 정도를 측정하고, 이를 통하여 암호화 과정과 복호화 과정에서의 성능 영향을 추정하기로 한다.

6.1. 시험환경

본 논문에서는 실행코드 증가량을 예측하기 위해, 고정길이 명령어를 사용하는 아키텍처를 이용하였다. 그 중 많이 사용하는 아키텍처인 MIPS를 선택하였고, 운영체제에서 실제 사용하는 프로그램을 이용하여 블록암호화로 인한 실행코드 증가량을 측정하였다.

[표 1]은 블록암호화로 인하여 증가하는 실행코드의 크기를 측정하기 위해 사용한 시험환경이다. 시스템 사양 중 파일증가에 영향을 미치지 않는 다른 요소들은 생략한다. 시험환경은 MIPS R12000을 CPU로 사용하고 IRIX 6.5를 운영체제로 이용하는 시스템을 이용하였다. 블록암호화시 파일크기의 증가량을 측정할 시험

〔표 1〕 시험환경

CPU	MIPS R12000
FPU	MIPS R12010
운영체제	IRIX 6.5
시험대상 프로그램	diff, echo, find, gcc, grep, gzip, ls, mkdir, more, nawk, pwd, rm, rmdir, tar

대상 프로그램은 실제 IRIX에서 제공하는 프로그램 중 흔히 사용하는 프로그램을 표와 같이 선정하였다.

6.2. 시험방법

크기가 다른 기본블록들을 단위블록으로 변환하면 변환에 프로그램의 크기가 증가한다. 선정한 프로그램들을 기본블록으로 구분하여 늘어나는 정도를 측정하였다.

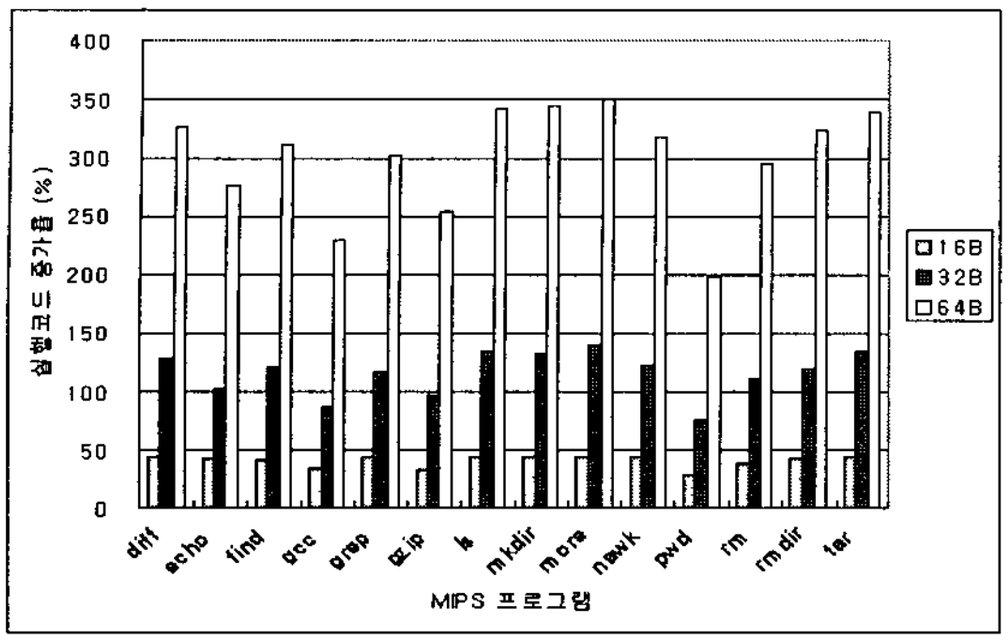
단위블록 안의 제어이동명령어들을 분석하여 선행블록이 두 개 이상인 블록을 찾고 이를 통하여 중복에 의해 늘어나는 블록의 개수를 계산하였다.

6.3. 단위블록으로의 변환에 의한 영향

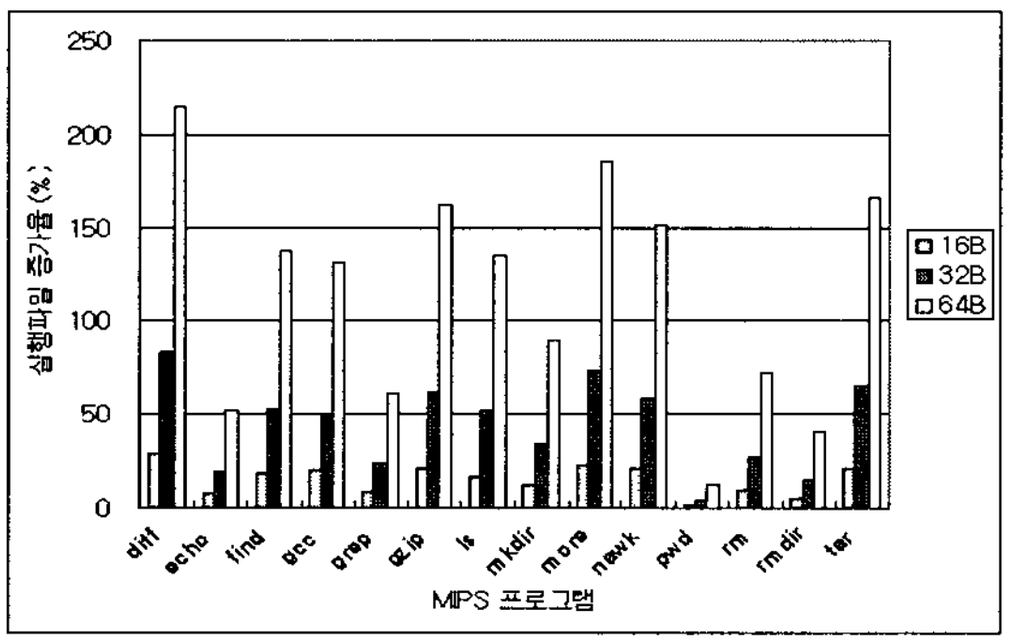
[그림 14]는 기본블록을 단위블록으로 변환할 때 증가되는 실행코드(a)와 실행파일(b)의 양을 각각 비율로 보인다. 다음은 그림 안에 명시한 각 항목에 대한 설명이다.

- 실행코드 영역 크기 증가(그림14(a)): 기본블록을 단위블록으로 변환함으로써 증가하는 실행코드의 크기를 비율로 나타낸다. 즉 텍스트 영역의 증가율이다.
- 전체 실행파일 크기 증가(그림14(b)): 기본블록을 단위블록으로 변환함으로써 증가하는 실행파일의 크기를 비율로 나타낸다. 즉 파일 전체 크기의 증가율이다.
- MIPS 프로그램: 단위블록 증가율을 측정하기 위해 사용한 실제 MIPS 프로그램이다.
- 16B/32B/64B: 단위블록으로 변환할 때 사용하는 단위블록의 크기로 각각 16바이트, 32바이트, 64바이트를 나타낸다.

[그림 14(a)]에서 보는 것과 같이 기본블록을 단위블록으로 변환하였을 경우 단위블록의 크기에 따라 단위블록 크기의 증가율이 상당히 다른 것을 볼 수 있다. 변환할 단위블록의 크기를 16바이트, 32바이트, 64바이트

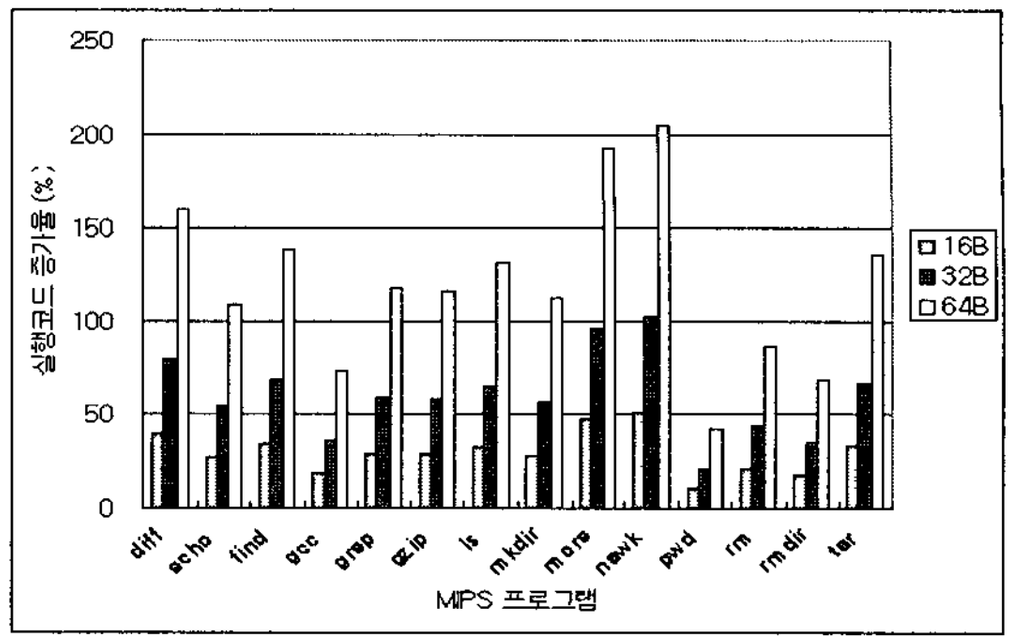


(a) 실행코드 영역 크기 증가

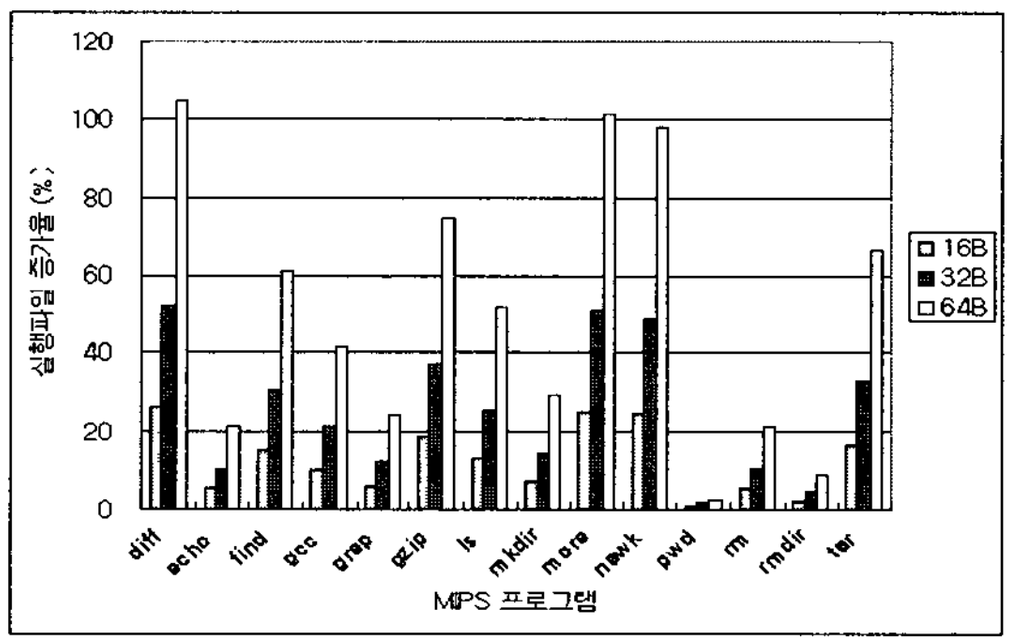


(b) 전체 실행파일 크기 증가

〔그림 14〕 단위블록으로의 변환에 의한 영향



(a) 실행코드 영역 크기 증가



(b) 전체 실행파일 크기 증가

〔그림 15〕 단위블록중복으로 인한 영향

로 하였을 경우 각각 실행코드 영역은 평균 40.8%, 115.6%, 301.1% 증가하였다. [그림 14(b)]는 단위블록으로의 변환에 의한 실행코드 크기의 증가율이 실행파일 전체 크기에 영향을 미치는 정도를 보인다. 그림 14(a)와 마찬가지로 단위블록의 크기(16바이트, 32바이트, 64바이트)에 따라 증가율의 변화가 크다는 것을 볼 수 있다. 그렇지만 실행파일 전체 크기에 대한 단위블록의 증가율이므로 실행코드 증가율보다는 그 크기가 작다. 16바이트 블록을 기준으로 평균 15.6% 증가하였다.

6.4. 단위블록중복으로 의한 영향

[그림 15]는 단위블록을 중복함으로써 증가되는 실행코드의 크기를 보인다.

- 실행코드 영역 크기 증가(그림 15(a)): 블록을 중복함으로써 증가하는 실행코드의 크기를 비율로 나타낸다. 즉 텍스트 영역의 증가율이다.
- 전체 실행파일 크기 증가(그림 15(b)): 블록을 중복함으로써 증가하는 실행파일의 크기를 비율로 나타낸다. 즉 파일 전체 크기의 증가율이다.

단위블록으로 변환했을 때와 마찬가지로 단위블록의 크기에 따라 그 증가율이 다르다. 변환할 단위블록의 크기를 16바이트, 32바이트, 64바이트로 하였을 경우 각각 텍스트영역의 증가율이 약 30.2%, 60.5%, 120.9%이다. 차이가 발생되는 주된 이유는 중복해야 하는 단위블록의 수가 다르기 때문이다. 그림 15(b)는 중복에 의한 실행코드 크기의 증가율이 실행파일 전체 크기에 영향을 미치는 정도를 보인다. 16바이트 블록의 경우 증가율은 평균 12.6%이다.

Ⅶ. 결 론

본 논문은 응용 프로그램의 실행코드를 블록암호화하고, 그 암호화된 실행코드를 코드블록별로 서로 다른 키를 사용하여 실시간으로 복호화하는 절차를 보임으로써 역분석에 효과적으로 대응할 수 있는 방법을 제시하였다. 키체인 방식으로 코드블록들을 암호화하였고, 제어이동명령어에 의해 선행블록이 두 개 이상 존재할 경우의 문제를 코드블록을 중복시키는 방법으로 해결하였다. 이 방법을 MIPS 프로그램에 적용할 경우 단위블록

과 블록중복으로 인한 실행파일 크기의 증가 정도는 16 바이트 블록 크기를 기준으로 평균 28.2% 전후이며 최대 55.6%로 측정되어 실제 적용하기에 큰 부담이 될 정도는 아닌 것으로 보인다.

참고문헌

[1] Gleb Naumovich, Nasir Memon, "Preventing Piracy, Reverse Engineer- ing, and Tampering", *Computer*, pp.64-71, July 2003.

[2] Christian S. Collberg, Clark Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection", *IEEE Transac- tions on Software Engineering*, pp.735-746, August 2002.

[3] Tao Zhang, Santosh Pande, Antonio Valverde, "Tamper-Resistant Whole Program Partitioning", *Proc. Conference on Languages, Compilers, and Tools for Embedded Systems*, June 2003.

[4] Ping Wang, Seok-kyu Kang, Kwangjo Kim, "Tamper Resistant Software Through Dynamic Integrity Checking", *Proc. 2005 Symposium on Cryptography and Information Security*, 2005.

[5] Hoi Chang, Mikhail J. Atallah, "Protecting Software Codes By Guards", CERIAS Tech- nical Report 2001-49, 2001.

[6] Cullen Linn, Saumya Debray, "Obfuscation of Executable Code to Improve Resistance to Static Disassembly", *Proc. 10th ACM Conference on Computer and Communications Security*, pp. 290-299, October 2003.

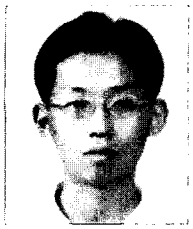
[7] Sonali Gupta, "Code Obfuscation", http://palisade.plynt.com, August 2005.

[8] Sonali Gupta, "Code Obfuscation Part 2: Obfuscating Data Structures", http://palisade.plynt.com, September 2005.

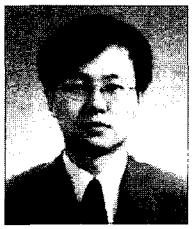
[9] Sonali Gupta, "Code Obfuscation Part 3: Hiding Control Flows", http://pali sade.plynt.com, October 2005.

[10] Trusted Computing Group, "Trusted Platform Module (TPM) Specifica tions", https://www.trustedcomputing grop.org/specs/TPM/, 2007.

〈著者紹介〉



정 동 우 (Dong-Woo Jung) 학생회원
2006년 2월 : 충남대학교 컴퓨터과학과 졸업
2008년 2월 : 충남대학교 컴퓨터공학과 석사
<관심분야> 인터넷보안



김 형 식 (Hyong-Shik Kim) 종신회원
1988년 2월 : 서울대학교 컴퓨터공학과 졸업
1990년 2월 : 서울대학교 컴퓨터공학과 석사
1997년 8월 : 서울대학교 컴퓨터공학과 박사
1997년 12월~1999년 8월 : 미국 UAH, Faculty Research Associate
1999년 9월~현재 : 충남대학교 전기정보통신공학부 컴퓨터전공 부교수
<관심분야> 인터넷보안, 컴퓨터시스템구조

박 중 길 (Joong-Gil Park) 정회원
1986년 2월 : 동국대학교 전산과 졸업
1988년 2월 : 서강대학교 전산과 석사
2002년 2월 : 충남대학교 전산학과 박사
1988년 2월~2000년 1월 : 국방과학연구소 선임연구원
2000년 2월~현재 : 한국전자통신연구원 부설 연구소 책임연구원
<관심분야> 네트워크 보안, 접근제어, 암호프로토콜