情報保護學會論文誌
第 23 卷 第 1 號, 2013. 2

MAC함수와 동적 링크키를 이용한 소프트웨어 변조 방지 기법*

박 재 홍,† 김 성 훈, 이 동 훈‡
고려대학교 정보경영공학전문대학원

A tamper resistance software mechanism using MAC function and dynamic link key*

Jae-hong Park,† Sung-hoon Kim, Dong-hoon Lee‡
Graduate School of Information Management & Security, Korea University

요 약

실행코드의 변조와 역분석 방지를 위해 단순히 선행블록에서 암, 복호화 키를 얻던 기존의 기법과 달리, 암호학적 MAC함수를 이용한 암호화기법과 코드 블록 간 중요도에 따라 상관관계를 설정하고 상관관계에 따라 암, 복호화 키를 생성하는 새로운 변조 방지 기법을 제안한다. 본 논문에서는 기존의 해쉬함수 대신 암호학적 MAC함수를 사용하고 MAC함수의 키를 동적으로 생성하는 방법을 소개한다. 또 단순히 선행블록의 해쉬 값에서 키를 얻는 것이 아니라 실행 코드 블록을 중요도에 따라 중요도 높음, 중간, 낮음으로 분류하고 중요도 높음 블록은 암호화하고 중요도 중간 블록은 중요도 높음 블록의 키를 생성시키는 블록으로 분류한다. 또 중요도 낮음 블록은 아무 처리도 하지 않음으로 소프트웨어 효율성을 고려한다. 기존 해쉬함수 대신 동적으로 생성되는 링크키를 이용한 MAC함수와 블록상관관계를 함께 사용함으로써 공격자의 분석을 어렵게 한다.

ABSTRACT

In order to prevent tampering and reverse engineering of executive code, this paper propose a new tamper resistant software mechanism. This paper presents a cryptographic MAC function and a relationship which has its security level derived by the importance of code block instead of by merely getting the encryption and decryption key from the previous block. In this paper, we propose a cryptographic MAC function which generates a dynamic MAC function key instead of the hash function as written in many other papers. In addition, we also propose a relationships having high, medium and low security levels. If any block is determined to have a high security level then that block will be encrypted by the key generated by the related medium security level block. The low security block will be untouched due to efficiency considerations. The MAC function having this dynamic key and block relationship will make analyzing executive code more difficult.

Keywords: application program protection, reverse engineering, software protection, code obfuscation

접수일(2012년 8월 7일), 수정일(1차: 2012년 10월 12일, 2차: 2012년 12월 24일), 게재확정일(2012년 12월 24일)
* 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.
† 주저자, magogos@hanmail.net
‡ 교신저자, donghlee@korea.ac.kr

I. 서 론

소프트웨어는 미리 컴파일 된 이진실행코드 형태로 배포된다. 공격자는 이와 같은 형태의 이진실행코드를 역 컴파일 과정이나 역 공학 기법과 같은 방법으로 분석하여 소프트웨어의 논리적 부분에 대한 정보를 추출해 낼 수 있다[1]. 또한 이러한 공격 기법들을 이용하여 사용자 요금 청구 부분이나 프로그램상의 중요 데이터 제어 부분 등 코드의 중요 부분을 변조함으로써 공격자는 프로그램을 원래의 의도와 다르게 행동하도록 할 수 있다.

이러한 역 공학 기법을 이용한 분석이나 코드 변조로부터 소프트웨어를 보호하기 위한 기술로 대표적인 방법[2]은 기계어 코드의 변형 방지를 위하여 기계어 코드의 변형 여부를 검사하여 만약 변형되었을 경우 프로그램상의 모든 기능 혹은 특정 기능을 제한하는 tamper-proofing기법[3]이 있고, 데이터를 모호화 하거나 프로그램상 제어의 흐름을 모호화 하거나 레이아웃을 모호화 하는 코드 모호화(Code Obfuscating)기법 그리고 암호화 기법이 있다[4]. 이중 가장 강력한 것은 암호화 기법인데 프로그램을 암호화함으로써 외부로부터의 공격을 원천적으로 방지 할 수 있다.

기존에 연구되었던 암호화 방법은 보호하고자 하는 응용 프로그램을 여러 블록으로 나누어 순차적으로 선행 블록의 해쉬 함수 값을 다음 실행블록의 암, 복호화 키로 사용하는 방법이었다[5][6].

본 논문에서 소프트웨어 변조 방지 기법 중 암호화 기법을 이용하면서도 기존의 방식인 단순히 선행블록의 해쉬 함수 값에서 키를 얻는 대신 암호학적 MAC(Massage Authentication Code)함수를 사용하고 또 MAC함수에 필요한 키를 동적으로 생성시켜 공격자가 정적인 분석을 통해서는 키를 계산할 수 없게 만든다.

또한 소스 코드를 작성한 후, 코드 블록의 중요도에 따라 3가지로 블록을 분류하여 중요도가 가장 높은 코드 블록에 대해서는 암호화하여 기밀성을 제공함과 동시에 변조 방지를 가능하게 하며, 중요도 중간인 코드 블록은 중요도가 높은 블록의 암호화 키 생성 블록으로 사용하게 하여 변조 방지 기능을 제공한다. 만약 중요도 중간 블록에서 변조가 발생하면 암, 복호화 키가 달라져 암호화된 중요도 높음 블록을 복호화 할 수 없게 된다. 중요도가 낮은 코드 블록에 대해서는 아무것도 적용하지 않음으로써 효율성 측면을 함께 고려하였다.

II. 관련 연구

암호화 기법을 사용하고 있는 기존 연구 중에는 본 논문에서 제안하는 암호학적 MAC함수의 사용여부와 실행블록을 중요도에 따라 나누는 방법과 같은 유사한 연구는 찾을 수 없었다. 기존 연구의 주제를 역 공학 공격에 안전한 코드 암호화 기법에 초점을 맞추면서 암, 복호화 기법에 사용되는 키 생성 기법을 분류해 보면 다음과 같이 나눌 수 있다.

• Cappaert 기법

Cappaert는 코드 암호화기법에 기초한 부분 암호화 기법을 제안 하였다. 이 기법에 따르면 바이너리 코드는 여러 작은 부분으로 나누어져 암호화되고 암호화된 이진코드는 사용자에 의해 런타임시에 복호화 된다. 이런 방법으로 부분 암호화 기법은 한 번에 전부 노출 될 수 있는 위험을 피할 수 있다. 그러나 실행블록이 논리적인 분기점을 가진 2개 이상의 선행블록을 가지게 되면 키를 생성하는 선행블록을 유일하게 결정하기 어렵다는 지적이 있다. Cappaert는 2008년 제안한 논문에서 암, 복호화 키를 계산하는 원리를 integrity-checking 기법이라고 이름 붙이고 있고 체크썸이나 해쉬 함수로 구성된다고 설명하고 있다[5].

• 키 생성 선행 블록이 2개 이상 일 때

논리적 분기점을 가진 2개 이상의 선행블록에서 호출 할 때 키 생성 블록을 유일하게 결정하기 위해 호출하는 블록을 2개로 복제하는 방법이 있다[4]. 이 방법은 두 개 이상의 블록에 의해 호출되는 문제를 해결해 주지만 복제되는 만큼 코드 길이가 길어져 효율성 측면에서 약점을 가진다는 지적이 있다[6].

• 인덱스 테이블을 이용한 방법

논리적 분기점을 가진 2개 이상의 선행블록이 있을 때 위 방법들의 대안으로 제시된 방법이다. 2개 이상의 선행블록에서 호출 블록을 2개로 복제 하지 않고 호출 블록의 주소 값을 인덱스 테이블에 기록하여 키가 필요 할 때 인덱스 테이블의 주소 값을 참조하여 블록을 호출하여 키를 생성하는 방법이다[6].

위의 방법들은 모두 암, 복호화 키를 생성 시키는 방법으로 선행블록의 해쉬 함수 값을 사용하고 있고, 블록 자체를 중요도에 따라 구분하고 있지 않다.

III. 동적 링크키 생성 및 상관관계 설정

3.1 용어 정리

본 논문에서 사용하고 있는 용어와 용어의 정의는 다음과 같다.

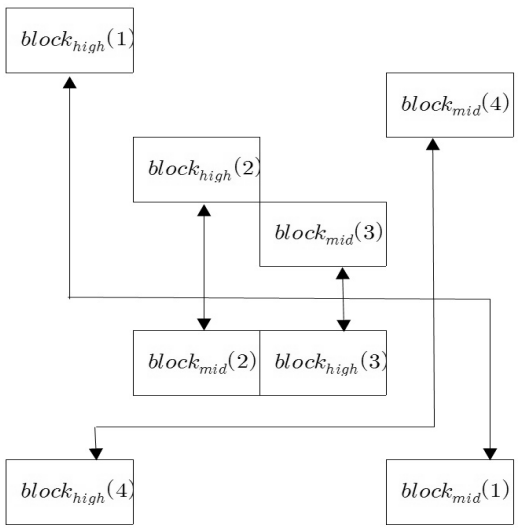
〔표 1〕 용어 설명

용 어	설 명
k	동적 링크 방식으로 생성된 키 값
$klen$	k의 바이트 길이
n	동적 링크키를 생성할 때 사용되는 배열의 크기
f	동적 링크키를 계산할 때 사용되는 함수
a	동적 링크키를 생성할 때 사용되는 배열
$a_i (0 \le i \le n-1)$	동적 링크키를 생성할 때 사용되는 배열 중 i번째 바이트 값
k_i	동적 링크 방식으로 생성된 키 값 중 i번째 바이트 값
m	메시지
$MAC_{key}\{m\}$	메시지m과 키 값 k를 입력 값으로 한 암호학적 MAC 함수
$H\{m\}$	메시지 m을 입력 값으로 하는 암호학적 해쉬 함수
$block_{high}(i)$	코드 영역 중 중요도가 높은 부분 중 i번째 블록
$block_{mid}(i)$	코드 영역 중 중요도가 중간인 부분 중 i 번째 블록
$block_{low}(i)$	코드 영역 중 중요도가 낮은 부분 중 i번째 블록
$bkey_i$	$block_{high}(i)$에 해당하는 블록을 암호화 또는 복호화 할 때 사용하는 대칭키 값
$E_{key}\{m\}$	메시지 m과 키 값 key를 입력 값으로 한 대칭키 암호화 함수
$D_{key}\{m\}$	메시지 m과 키 값 key를 입력 값으로 한 대칭키 복호화 함수

우선 응용 프로그램 개발자가 소스 코드를 작성한 후, 코드의 중요도에 따라 3가지로 코드를 분류한다. 개발자 또는 프로그램 관계자가 판단하여 어떤 실행 코드의 블록이 중요한 부분이고 노출이 되어서는 안 된다고 판단하면 중요도 높음으로 분류한다. 이 코드 블록에 대해서는 암호화 하여 기밀성을 보장한다. 또 어떤 코드 블록이 노출되어도 상관없지만 변조되어서는 안 된다고 판단되면 중요도 중간으로 분류하여 암호화는 하지 않지만 중요도 높음 블록의 암호화키로 사용되게 하여 무결성을 부여한다. 만약 중요도 중간 블록이 변조되면 키가 바뀌게 되어 암호화된 중요도 높음 블록을 복호화 할 수 없게 된다.

중요도가 낮은 코드 블록에 대해서는 아무것도 적용하지 않음으로써 효율성 측면을 고려한다. 세부 방법은 아래와 같다.

3.2 소스코드 중요 부분 설정 및 상관관계 설정 단계



〔그림 1〕 블록간의 상관관계

가. 소스 코드 작성자 또는 관계자는 코드의 중요도에 따라 높음, 중간, 낮음으로 구분한다.

- 코드 블록이 노출되기를 원하지 않고 변조되어서도 안 된다고 판단되면 중요도 높음으로 암호화한다. 기밀성보장
- 코드 블록이 노출되어도 상관없으나 변조되어서는 안 된다고 판단되면 중요도 중간으로 중요도 높음 블록의 키로 사용한다. 만약 이 중요도 중간 블록이 변조되면 암, 복호화 키가 달라져 중요도 높음의 암호화 된 블록이 복호 화되지 않는다. 무결성 보장
- 코드 블록이 별로 중요하지 않다고 판단되면 그냥 둔다. 효율성 보장

나. 중요도 높음과 중간 부분에 대해서 상호간에 상

관(대응)관계를 준다.

- 상관관계는 중요도 중간 블록과 중요도 높음 블록에 대한 상관관계를 의미한다.
- 중요도 중간 코드 블록을 중요도 높음 블록과 대응시켜 중요도 높음 블록의 암, 복호화 키로 사용한다. 만약 중요도 중간의 코드 블록이 변조되었을 경우 올바른 키가 계산되지 않기 때문에 중요도 높음 블록의 실행 시에 제대로 복호화 되지 않는다.
- 또한 중요도 높음 코드 블록이 변조 되었을 경우에도 복호화가 제대로 이루어지지 않는다.

3.3 동적 링크키 생성 방법

가. 동적 링크 방식으로 키를 생성할 때 사용할 배열 a의 크기 n값을 설정한다.

- 배열 a의 크기는 임의의 크기가 가능하지만 MAC함수의 키의 크기 $klen$보다 크거나 같을 것을 권장한다.

나. 배열 a를 바이트 단위로 길이n을 생성한다.

- 배열의 크기를 결정하면 n바이트 값을 생성한다. 이 n바이트 값은 MAC함수의 키 k를 생성시킬 종자값(seed value)의 의미로 사용된다.

다. 동적 링크키 생성 방법은 아래와 같다.

- $0 \le k_0 \le n-1$ 범위에서 k_0 값을 선택한다.
- $i=1$부터 $i<klen$ 까지 i 값을 1씩 증가 시키면서 아래를 계산한다.
- $k_i = a_{(f(k_{i-1}) \bmod n)}$
- 함수 f는 키 k를 생성시킬 배열 a의 값을 모듈러 연산을 통해 순서를 섞는 기능을 한다. 본 논문에서는 함수 f을 특정하지 않는다. 키 k의 값을 특성 좋게 생성 시킬 수 있으면 어떤 함수도 사용 가능하다.

〔예 1〕 간단한 함수f를 이용한 동적 링크키 생성 예

가) 배열 a의 크기: 8바이트 ($n=8$)

나) $a=\{114,173,220,31,238,91,9,192\}$로 a값을 임의로 생성한다.

다) 함수 f를 $f(x)=x$ 로 임의로 정의한다.
f는 설계자 각자가 정의하여 사용한다.

라) k_0값을 먼저 결정한다.
여기서는 $i=1$인 a값 즉, 173로 가정한다.
따라서 $k_0=173$

$k_i = a_{(f(k_{i-1}) \bmod n)}$ 이므로

$i=1$ 에서

$k_1 = a_{(x \bmod 8)} = a_{(173 \bmod 8)} = a_{(5)} = 91$

$i=2$ 에서

$k_2 = a_{(x \bmod 8)} = a_{(91 \bmod 8)} = a_{(3)} = 31$

$i=3$ 에서

$k_3 = a_{(x \bmod 8)} = a_{(31 \bmod 8)} = a_{(7)} = 192$

$i=4$ 에서

$k_4 = a_{(x \bmod 8)} = a_{(192 \bmod 8)} = a_{(0)} = 114$

$i=5$ 에서

$k_5 = a_{(x \bmod 8)} = a_{(114 \bmod 8)} = a_{(2)} = 220$

$i=6$ 에서

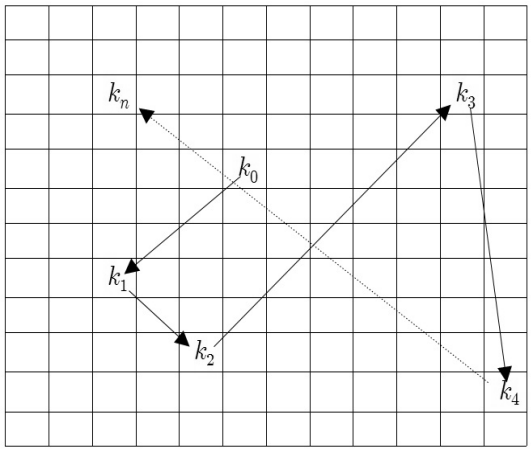
$k_6 = a_{(x \bmod 8)} = a_{(220 \bmod 8)} = a_{(4)} = 238$

$i=7$ 에서

$k_7 = a_{(x \bmod 8)} = a_{(238 \bmod 8)} = a_{(6)} = 9$

〔표 2〕 배열a와 키k 사이의 관계

i	0	1	2	3	4	5	6	7
a	114	173	220	31	238	91	9	192
k	173	91	31	192	114	220	238	9



〔그림 2〕 동적 링크키 동작 예

IV. 암 · 복호화 과정

4.1 암호화 과정

중요도 높음 블록의 암호 키 생성 및 암호화 단계는 MAC함수를 이용하는 방법, 해쉬 함수 h를 이용하는 방법으로 나눌 수 있다.

4.1.1 MAC함수를 이용하는 방법

가. 응용 프로그램을 중요도에 따라 높음, 중간, 낮음으로 구분하여 상관관계를 준다.

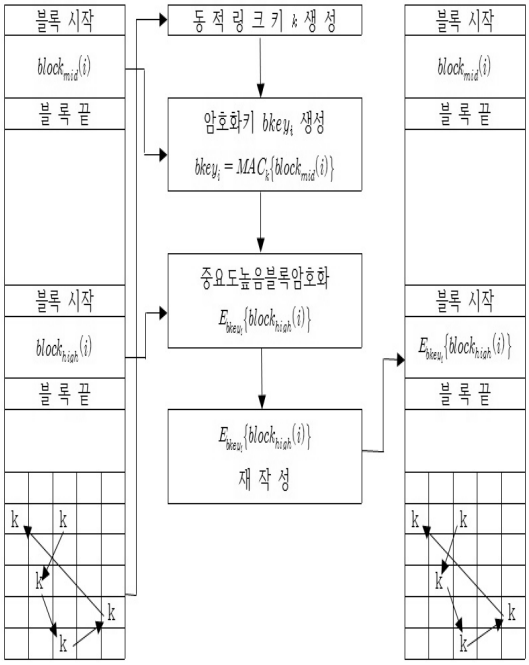
나. 위의 상관관계에서 설정한 임의의 중요도 높음 블록 $block_{high}(i)$에 대응하는 중요도 중간 블록 $block_{mid}(i)$을 결정하고 동적 링크키 k를 생성한다.

다. 중요도 높음 블록 $block_{high}(i)$에 대해 상관관계를 준 $block_{mid}(i)$와 동적 링크키 k로 다음과 같이 $bkey_i$을 생성한다.

$bkey_i = MAC_k\{block_{mid}(i)\}$

라. 위에서 구한 암호화키 값 $bkey_i$로 $block_{high}(i)$을 아래와 같이 암호화 한다.

$E_{bkey_i}\{block_{high}(i)\}$



〔그림 3〕 MAC함수를 이용한 암호화 과정

4.1.2 해쉬 함수 h를 이용하는 방법

가. 응용 프로그램을 중요도에 따라 높음, 중간, 낮음으로 구분하여 상관관계를 준다.

나. 위의 상관관계에서 설정한 임의의 중요도 높음 블록 $block_{high}(i)$에 대응하는 중요도 중간 블록 $block_{mid}(i)$을 결정하고 동적 링크키 k를 생성한다.

다. 중요도 높음 블록 $block_{high}(i)$에 대해 상관관계를 준 $block_{mid}(i)$와 동적 링크키 k로 다음과 같이 $bkey_i$을 생성한다.

$bkey_i = hash\{k \parallel block_{mid}(i)\}$

즉, 중요도 중간 블록과 동적 링크키 k를 패딩하여 해쉬 값을 취한 값이다.

라. 위에서 구한 암호화키 값 $bkey_i$로 $block_{high}(i)$을 아래와 같이 암호화 한다.

$E_{bkey_i}\{block_{high}(i)\}$

〔그림 3〕에서 $bkey_i$ 생성부분을 $MAC_k\{block_{mid}(i)\}$ 대신 $hash\{k \parallel block_{mid}(i)\}$으로 대신하면 된다.

4.2 복호화 및 재 암호화 단계

암호화 키 생성 및 암호화 단계와 마찬가지로 실행코드블록의 실행 전 암호화 된 블록의 복호화 및 재 암호화 단계에서도 MAC함수를 이용하는 방법, 해쉬함수 h를 이용하는 방법이 있다.

4.2.1 MAC함수를 이용하는 방법

가. 위의 동적 링크키 생성 방법으로 동적 링크키 k를 생성한다.

나. 위의 상관관계에서 설정한 암호화된 임의의 중요도 높음 블록 $E_{bkey_i}\{block_{high}(i)\}$에 대해서 복호화 키 값을 생성한다.

다. 중요도 높음 블록 $block_{high}(i)$에 대해 상관관계를 준 $block_{mid}(i)$와 동적 링크키 k로 다음과 같이 $bkey_i$을 생성한다.

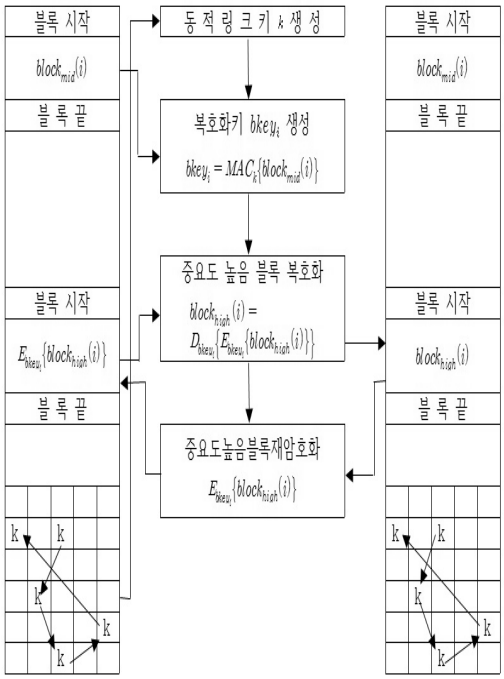
$bkey_i = MAC_k\{block_{mid}(i)\}$

라. 위에서 구한 복호화 키 $bkey_i$로 암호화 된 블록 $E_{bkey_i}\{block_{high}(i)\}$을 아래와 같이 복호화 한다.

$block_{high}(i) = D_{bkey_i}\{E_{bkey_i}\{block_{high}(i)\}\}$

마. 복호화 되어 실행된 블록은 재 암호화 된다.

$E_{bkey_i}\{block_{high}(i)\}$



〔그림 4〕 MAC함수를 이용한 복호화 과정 및 재 암호화

4.2.2 해쉬 함수 h를 이용하는 방법

가. 위의 동적 링크키 생성 방법으로 동적 링크키 k를 생성한다.

나. 위의 상관관계에서 설정한 암호화된 임의의 중요도 높음 블록 $E_{bkey_i}\{block_{high}(i)\}$에 대해서 복호화 키 값을 생성한다.

다. 중요도 높음 블록 $block_{high}(i)$에 대해 상관관계를 준 $block_{mid}(i)$와 동적 링크키 k로 다음과 같이 $bkey_i$을 생성한다.

$$bkey_i = hash\{k \| block_{mid}(i)\}$$

라. 위에서 구한 복호화 키 $bkey_i$로 암호화 된 블록 $E_{bkey_i}\{block_{high}(i)\}$을 아래와 같이 복호화 한다.

$$block_{high}(i) = D_{bkey_i}\{E_{bkey_i}\{block_{high}(i)\}\}$$

마. 복호화 되어 실행된 블록은 재 암호화 된다.

$E_{bkey_i}\{block_{high}(i)\}$

〔그림 4〕에서 $bkey_i$ 생성부분을 $MAC_k\{block_{mid}(i)\}$ 대신 $hash\{k \| block_{mid}(i)\}$으로 대신하면 된다.

4.3 본 논문에서 제안하는 기법의 특성 비교 분석

본 논문에서는 블록의 중요도에 따라 상관관계를 주고 또 암,복호화 키로 MAC함수를 이용하였다. 그러나 기존의 연구들은 선행 블록의 해쉬값만을 암, 복호화 키로 사용하고 있다. 본 논문에서는 특히 선행블록의 해쉬 값에서 암호키를 얻으면서 키체인 방식을 사용하여 응용 프로그램 전체를 암호화하는 기존의 알고리즘과 간략히 비교한다〔4〕〔5〕.

가. 코드의 무결성(integrity): 제안하는 기법이나 기존 기법 모두 무결성을 제공한다.

나. 코드의 기밀성(security): 제안하는 기법이나 기존 기법 모두 기밀성을 제공한다.

다. 코드의 복원성(resilience)

코드 복원성(resilience)은 난독화된 코드가 외부 공격에 의해 원 코드로 복원 시도 될 때 얼마나 잘 견디는지에 따른 분류이다. one-way함수는 복원성이 높은 예이다. 제안하는 기법은 one-way 함수인 MAC함수를 사용하므로 복원성이 높다.

라. 코드의 불명확성(potency)

코드의 불명확성은 난독화된 코드의 가독성 정도에 따른 분류이다. 코드 복잡도(complexity)가 높을수록 불명확성은 증가한다. 제안하는 기법은 MAC함수의 키가 개발자의 설계에 따라 동적으로 생성되므로 MAC 함수값이 고정되지 않고 매번 바뀌어 암, 복호화에 사용되는 키가 매번 바뀌는 반면 기존의 기법은 단순히 선행 블록의 해쉬값으로 암, 복호화 키를 사용하므로 해쉬함수 자체가 바뀌지 않는 한 동일한 응용 소프트웨어를 실행 할 때마다 매번 같은 키를 사용하게 된다.

따라서 제안하는 기법이 코드 불명확성 측면에서 더 효과적이다.

마. 비용(cost)

비용(cost)은 난독화로 인한 실행시간의 증가 또는 응용프로그램 크기의 증가로 인한 비용의 발생을 의미한다. 만약 동일한 응용 프로그램을 동일한 블록으로 나누어 전부 암, 복호화 하여 실행한다면 기존의 기법보다 본 논문에서 제안하는 기법의 비용이 더 든다. 하지만 실제로는 이진 실행코드 형태로 배포되는 응용프로그램 전체를 암호화 할 필요는 없는 것이다. 중요한 부분만 암호화 하면 되는 것이다.

다음은 하나의 동일한 응용 프로그램을 다섯 개의 블록으로 나누어서 본 논문에서 제안하는 기법과 기존의 기법과의 비용 분석을 한다면 다음과 같이 생각해 볼 수 있다. 여기서 암, 복호화 암호 알고리즘은 동일한 것을 사용한다고 가정하고 연산은 각 블록별로 복호화단계, 실행단계, 해쉬, 암호화단계 등 처리 단계별로 나누어 생각한다.

〔표 3〕 제안하는 기법의 비용분석

제안하는 기법		기존의 기법	
블록 중요도 판정	필요한 연산	블록 중요도 판정	필요한 연산
블록1: 무결성 필요 (중요도 중간)	키생성, 해쉬(MAC)	블록1	복호, 실행, 해쉬, 암호
블록2: 보호 불필요	실행	블록2	복호, 실행, 해쉬, 암호
블록3: 기밀성 필요 (중요도 높음)	복호, 실행, 암호	블록3	복호, 실행, 해쉬, 암호
블록4: 보호 불필요	실행	블록4	복호, 실행, 해쉬, 암호
블록5: 보호 불필요	실행	블록5	복호, 실행, 암호
실행을 제외한 연산	키생성: 1 해쉬: 1 복호: 1 암호: 1	실행을 제외한 연산	복호: 5 해쉬: 4 암호: 5

여기서는 본 논문에서 제안하는 기법의 비용이 기존의 기법에 비해 복호 1/5, 해쉬 1/4, 암호 1/5 수준이고 추가로 키생성 연산이 1번 더 사용되었다. 본 논문에 제안하는 기법에서 비용 측정은

첫째. 일반적으로 응용 프로그램 설계자 및 관계자가 판단하여 보호가 필요한 블록이 더 많다면 실행 비용은 더 증가 할 것이고 보호가 필요한 블록이 더 적다면 비용도 더 감소 할 것이다.

둘째. 암호, 복호, 해쉬, 키생성으로 연산을 나누었지만 암, 복호화 알고리즘은 동일하므로 결국 MAC함수와 해쉬 함수의 차이 그리고 동적 링크키 생성 알고리즘의 실행 성능 및 알고리즘을 설계하는데 드는 비용이 될 것이다. 이것 또한 설계자가 임의로 비용을 조절 할 수 있는 부분이다.

다음은 본 논문에서 제안하는 기법의 특성 분석을 표로 정리 한 것이다.

〔표 4〕 제안하는 기법의 특징

	기밀성/ 무결성	코드의 복원성 (resilience)	코드의 불명확성 (potency)	비용 (cost)
제안하는 기법	제공	높음	상대적으로 더 높음	조절가능
기존 기법	제공	높음	높음	조절불가

V. 결 론

본 논문에서는 코드 변조 방지 기법 중 하나인 암호화 기법을 사용하면서 암, 복호화 키가 동적으로 생성되게 설계하여 만약 공격자가 동적 또는 정적인 분석을 시도하면서 자신의 의도대로 프로그램 변조를 할 때 공격에 어려움을 준다. 프로그램 설계자 또는 관계자가 코드 블록의 중요도에 따라 중요도 높음, 중간, 낮음으로 구별하고 서로 상관관계를 주어 기밀성이 필요한 부분은 중요도 높음을 주어 암호화한다. 또 무결성이 필요한 부분은 중요도 중간을 주어 암호화에 필요한 키를 생성시키는 키 생성 영역으로 분류하여 만약 중요도 중간 영역의 코드 블록에서 변조가 일어나면 복호화 키가 맞지 않아 오류를 발생시킨다. 중요도 낮음 영역은 아무것도 처리 하지 않음으로 실행 효율성 측면을 함께 고려하였다. 암호화한 부분을 실행하기 전 복호화 해야 하는 영역에서 키 값을 동적으로 생성한 후, 해당키와 중요 블록부분을 입력 값으로 하여 생성된 MAC함수의 값을 복호화 키로 사용함으로써 공격자가 역 공학 기법으로 공격할 때, MAC함수 자체를 분석해야 하는 문제와 함께 동적으로 계산되는 키 생성 함수의 상관관계를 동시에 분석해야 하는 문제를 만듦으로써 역 공학 기법을 이용한 코드 변조, 분석을 어렵게 하였다.

참고문헌

〔1〕 Hoi Chang and Mikhail J. Atallah, “Protecting software Code By Guards,” LNCS 2320, pp. 160-175, 2002.

〔2〕 Gleb Naumovich and Nasir Memon, “Preventing Piracy, Reverse Engineering, and Tampering,” computer, pp. 64-71, July 2003.

〔3〕 Tao Zhang, Santosh Pande, and Antonio Valverde, “Tamper-Resistant Whole Pro-

gram Partitioning," Proc. Conference on Languages, Compilers, and Tools for Embedded Systems, pp. 209-219, June 2003.
〔4〕 정동우, 김형식, "응용프로그램 역분석 방지를 위한 코드블록 암호화 방법," 정보보호학회논문지, 18(2), pp. 85-96, 2008년 4월.
〔5〕 J. Cappaert et al., "Toward Tamper Resistant Code Encryption:Practice and Experience," LNCS 4991, pp. 86-100, 2008.
〔6〕 Sungkyu Cho et al., "Secure and Efficient Code Encryption Scheme Based on Indexed Table," ETRI Journal, vol. 33, Number 1, February 2011.
〔7〕 J. Cappaert et al., "Self-Encryption Code to Protect Against Analysis and Tampering," 1st Benelux Workshop Inf. Syst. Security, p. 14, November 2006.

〈著者紹介〉

박 재 홍 (Jae Hong Park) 정회원
1994년 2월: 부산대학교 수학과 졸업
2010년 2월: 고려대학교 정보경영공학전문대학원 수료
2000년 7월~현재: 국방부 재직
〈관심분야〉 정보보호이론, 통신공학, 소프트웨어공학

김 성 훈 (Sung Hoon Kim) 정회원
2006년 8월: 서울시립대학교 수학과 졸업
2009년 2월: 고려대학교 정보보호대학원 석사 졸업
2009년 1월~2011년 2월: (주)알티캐스트 CAS개발본부 전임연구원
2011년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 소프트웨어 보안, 소프트웨어 난독화

이 동 훈 (Dong Hoon Lee) 정회원
1983년: 고려대학교 경제학과 학사 졸업
1987년:Oklahoma University 전산학 석사 졸업
1992년:Oklahoma University 전산학 박사 졸업
1993년~1997년: 고려대학교 전산학과 조교수
1997년~2001년: 고려대학교 전산학과 부교수
2001년~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 정보보호이론, 암호 프로토콜, USN, 키 교환, 프라이버시향상기술(PET), 익명성 연구