# 하드웨어 고유 번호 기반 소프트웨어 보호 방식

노한영*, 이현록*, 박재범**, 김광조*

* 한국정보통신대학교 암호와 정보보안 연구실
** SK 텔레콤 Service 기술연구원 Application 개발팀

# A Software Protection Method based on Hardware ID

HanYoung, Noh*, Hyunrok Lee*, JaeBum Park** , Kwangjo Kim*

* Cryptology & Information Security Lab,
Information and Communications University.
** SK Telecom Service Development Team 1,
Service Platform & Core Network R&D Center

**요 약**

소프트웨어는 사용자에게 배포된 후에는 공격자에게 완전히 공개된 상태에 놓이게 되고, 악의적인 사용자는 역공격을 시도 할 수 있다. 따라서 이를 막기 위한 보호 방식의 연구는 매우 중요하다. 본 논문에서는 Fu 등[1] 의 연구 에서 불법복제에 대한 보안이 고려되지 않았다는 점과 변수 테이블을 가지고 있어서 공격에 쉽게 노출되는 취약점을 개선하였다. 불법 복제에 대한 개선으로는 하드웨어의 고유번호를 이용하여 소프트웨어가 하드웨어에 의존적인 상태로 만들어 특정 하드웨어를 가지고 있는 사용자만이 이를 실행할 수 있도록 하는 방식을 제안하였다. 변수 테이블이 공격에 쉽게 노출되는 취약점은 암호학적인 일방향 해쉬 함수와 임의의 변수 값을 이용해 개선하였다.

## I. 서론

매년 소프트웨어 불법 복제로 인한 소프트웨어 업계의 피해는 심각한 상황이다. 정부 차원의 단속과 교육으로 이를 막고자 하지만 이는 아날로그 매체와 다르게 쉽게 복제가 가능하다는 소프트웨어의 특징상 한계가 있다. 또한, 복제 방지 장치를 해놓더라도 한번 공격이 성공하면 쉽게 복제가 된다는 점에서 불법 복제의 피해를 줄이기에는 역부족이며, 특히 소프트웨어의 알고리즘을 도용하거나 악의적으로 변경해서 사용하는 불법적인 개인 사용자들을 막을 수 도 없다.

Barak 등[7]의 연구 결과에 의하면 소프트웨어는 사용자에게 배포된 후에는 완전히 공개된 상태가 되고 사용자는 소프트웨어를 변경할 수 있으며 실행 상태를 파악할 수 있어 악의적인 사용자로부터 소프트웨어를 완벽하게 보호하는 것은 불가능하다. 하지만, 일반적인 바이너리 상태로 배포된 소프트웨어를 역공격(Reverse Engineering)하기 위해서는 기계어코드를 해석해야만 하는데 다양한 분석툴[14, 15, 16]을 활용하더라도 이는 공격자의 관점에서는 매우 어렵고 지루한 작업이다. 그러므로 현실적인 소프트웨어의 보호는 공격자가 얻을 수 있는 이득에 비해, 공격자의 시간과 노력이 더 많이 필요하도록 소프트웨어를 만드는 것이다[2].

본 논문에서 제안하는 방식은 Fu 등이 제안

한 패스워드 기반 소프트웨어 보호법을 개선한, 하드웨어 고유 번호 기반 소프트웨어 보호방식이다. 하드웨어 고유번호란 컴퓨터 하드웨어가 가진 고유한 값으로 CPU ID, 하드 ID, 메인보드 ID, 네트워크 카드의 MAC 주소 등이 있으며 이러한 값들을 조합하게 되면 다른 컴퓨터에서는 얻을 수 없는 값을 만들 수 있다. 이렇게 만들어진 값을 패스워드로 사용하는 소프트웨어 보호 방식이다.

본 논문은 다음과 같이 구성되어있다. Ⅱ장에서는 대표적인 소프트웨어 보호방식과 Fu 등이 제안한 패스워드에 기반 한 소프트웨어 보호방식과 연구 배경을 제공한다. Ⅲ장에서는 본 논문에서 제안하는 하드웨어 고유번호 기반 소프트웨어 보호방식을 소개하고, Ⅳ장에서는 제안한 기술에 대한 성능을 비교 분석한다. Ⅴ장에서는 본 논문을 요약하고 결론을 정리한다.

## II. 관련 연구

일단 배포된 소프트웨어는 사용자에게 완전히 공개된 상태가 되고 공격자는 소프트웨어를 공격할 수 있는 최고의 권한을 가지게 된다. 따라서 사용자는 실행되고 있는 코드를 모두 파악 할 수 있으며, 이를 수정할 수도 있다. 이를 막기 위해 신뢰할 수 있는 물리적인 장치를 활용해서 코드를 보호하는 방식들이 제안이 되었지만, 이는 선의의 사용자에게 추가적인 비용을 발생시키게 되고 사용하는데도 불편함이 따르게 된다[3]. 따라서 추가적인 장비를 사용하지 않고 소프트웨어만을 가지고 보호를 하는 보호방식들이 제안이 되었고, 그 중 대표적인 기술이 프로그램 변조방지(Tamper-proofing)[4, 6, 8, 13], 워터마킹(Watermarking)[6, 10], 프로그램 난독화(Obfuscation)[5, 6, 11, 12] 기술이다. 프로그램 변조방지는 프로그램을 쉽게 수정하지 못하도록 코드를 감시하는 방식이고, 워터마킹은 프로그램을 임의로 변경했을 때 이를 감지할 수 있는 코드를 삽입하는 기술이다. 마지막으로 프로그램 난독화는 프로그램의 코드를 쉽게 해석하지 못하도록 코드 사이에 더미코드를 입력하거나, 논리적인 흐름 해석이 어렵도록 다양한 수식을 소스 코드나 머신코드에 적용하는 방식이다. 이 방식은 앞서 소개한 프로그램 변조 방지 및 워터마킹 방식에도 추가적으로 적용할 수 있는 방식으로 공격자의 분석시간을 늘일 수 있는 현실적인 소프트웨어 보호 방식이다.

Fu 등이 제안한 소프트웨어 보호 방식은 기존의 접근과는 다른 새로운 방식으로 해쉬 함수를 통해 변수를 보호하는 방식이다. 이 방식은 공격자가 변수들의 값을 쉽게 확인할 수 없도록 보호하는 방식이다. 특정 변수를 패스워드(Password)로 사용하여, 패스워드와 임의의 값 α 를 이용해 *Hash*(Password + α)의 값을 미리 계산한다. 여기서 사용한 *Hash*()는 일반적인 암호학적인 일방향 함수가 아닌 맵핑 테이블의 역할을 하는 해쉬 함수이다. 계산된 값을 변수 테이블에서 찾아 이 값을 변수 값으로 대체한다. 이를 효과적으로 구현하기 위해 고려해야할 사항은 아래와 같다.

1. 알고리즘에서 사용되는 핵심값을 선택한다.
2. 공통적으로 사용되는 값을 사용한다.
3. 새로운 변수를 만든다.

이렇게 생성된 결과 값은 변수를 대체할 수도 있고, 새롭게 만들어진 변수를 이용해 기존에 제시되었던 프로그램 난독화 기술에 적용할 수도 있다. 만약 잘못된 패스워드가 입력이 될 경우에는 변수 값이 정상적인 값이 아니더라도 실행은 지속적으로 이뤄지기 때문에 공격자는 공격에 어려움을 겪게 된다. 하지만, 이 방식은 변수 테이블이 메모리상에 저장되어있기 때문에 변수 테이블이 충분히 계산 가능한 크기라면 공격자는 쉽게 이 값들을 예측할 수 있다. 만약에 변수 테이블이 매우 크다고 하더라도 메모리상에 저장이 되어있으므로 공간적인 한계가 있다는 약점이 있다.

## III. 하드웨어 고유번호 기반 소프트웨어 보호

본 논문에서는 크게 2가지 방식을 제안한다. 첫 번째는 특정 컴퓨터에서만 실행 가능한 프

로그램을 만들어 불법복제를 방지하는 것으로 기존에는 일반적으로 추가 하드웨어를 이용하였으나[9], 본 논문에서는 하드웨어 고유번호에 기반한 소프트웨어 보호 방식을 제안한다. Fu 등의 방식에서 사용된 패스워드 매개 값을 하드웨어 고유번호로 변경한 방식으로, 정상적인 경로로 받은 프로그램을 다른 컴퓨터에서 실행시키면 하드웨어 고유번호가 다르기 때문에 결과가 정상적으로 나오지 않는다. 따라서 이 방식은 효과적인 오프라인 인증 시스템이 될 수 있다.

두 번째는 Fu 등의 방식에서 변수 테이블이 존재한다는 취약점을 보완한 방식으로 임의의 값을 암호학적인 일방향 해쉬 함수를 적용해 공격자가 이 결과 값을 예측할 수 없도록 개선하였다.

**(방식 1) 하드웨어 고유 번호를 적용한 오프라인 인증 방식**

Fu 등의 연구에서 사용되던 패스워드를 하드웨어 고유번호로 바꿔서 사용한다. 즉, 단순한 문자열인 패스워드에서 의미를 가지는 값으로 대체하는 것이다. 여러 하드웨어 고유 번호를 조합해서 사용하면 다른 컴퓨터에서 얻을 수 없는 값을 얻을 수 있으며 이를 이용하면 불법 복제로부터 안전할 수 있다. 또한, 하드웨어 고유 번호는 하드웨어를 교체하는 방법 외에는 바뀌지 않는 값이다. 그리고 추가적인 하드웨어를 사용하는 방식에 비해, 사용자에게 주는 불편함과 비용이 줄어든다.

또한 Fu 등의 방식에서는 패스워드가 코드상에 나타나 있으므로 공격자가 이를 쉽게 분석할 수 있지만 (방식 1)의 하드웨어 고유번호는 프로그램 실행시에 실시간으로 얻어지는 값이므로 코드 상에 하드웨어 고유번호가 나타나지 않는다. 따라서, 정상적으로 소프트웨어를 배포 받은 사용자가 아닌 불법적으로 소프트웨어를 받은 사용자는 어떤 하드웨어 고유번호가 정상적인 키인지 알 수가 없으므로 해쉬 함수의 범위를 전수 조사하는 방법으로만 이를 공격할 수 있다. 하드웨어 고유번호는 사용자로부터 신뢰할 수 있는 값을 얻을 수 있다고 가정한다.

**(방식 2) 암호학적인 일방향 해쉬 함수를 이용한 변수 값 대체 방식**

Fu 등은 변수 테이블과 맵핑 테이블 역할을 하는 해쉬 함수를 사용해 원하는 변수 값을 반환 받았으나, 본 논문에서는 (방식 1)에서 제안한 하드웨어 고유 번호와 임의의 값을 암호학적인 일방향 해쉬 함수에 적용하고, 그 결과 값을 이용하여 원하는 값을 얻을 수 있게 하였다. 변수 테이블 대신에 임의의 값을 사용함으로써 메모리 테이블이 사라지게 되었고, 어떤 값이 결과 값이 될 지 예측할 수가 없게 되었다. 제안한 방식에서 사용한 표기법은 표 1 과 같다.

표 1 : 표기법

| 표기법 | 설명 |
|---|---|
| $H\_ID$ | 하드웨어 고유 번호 |
| $Value$ | 소스 코드상의 변수값 |
| $R$ | 임의의 값 |
| $S$ | $H(H\_ID \cdot R) \oplus Value$ |
| $O$ | $H(GetH\_ID() \cdot R) \oplus S$ |
| $\cdot$ | 사용자가 정의한 연산 |
| $\oplus$ | 일반적인 XOR 연산 |
| $H()$ | 암호학적인 일방향 해쉬 함수 |
| $GetH\_ID()$ | 하드웨어 고유 번호를 반환하는 함수 |

위 표기법에서 $O$ 는 $H(GetH\_ID() \cdot R) \oplus S$ 의 역할을 수행하며 $H(H\_ID \cdot R) \oplus S$ 와 같은 기능을 수행하고, 그 결과값 $Value$ 를 반환한다. 다만 하드웨어 고유번호를 프로그램 실행시에 얻어 온다는 점만이 다르다. 제안한 방식을 적용하는 시나리오는 다음과 같다.

1) 사용자의 $H\_ID$ 를 수집
2) 프로그램 소스의 $Value$ 를 수집
3) $H\_ID$, $Value$, $R$ 를 이용해 $S$ 를 계산
4) $Value$ 를 $O$ 로 변경
5) 컴파일 후 배포

(단, $R$ 값은 재사용되지 않는다.)

간단한 수식인 $x^2+bx+c=0$ 을 계산하는 프로그램에 이를 적용해보자. 결과 값은 $\frac{-b+\sqrt{(b^2-4c)}}{2}$, and $\frac{-b-\sqrt{(b^2-4c)}}{2}$ 이다. C 언어 소스 코드를 아래와 같이 만들었다.

```
#include <stdio.h>
#include <math.h>

void quadratic(double b, double c,
          double *root1, double *root2){

     double temp = sqrt(b * b - 4 * c);

     *root1 = (-b + temp ) / 2;
     *root2 = (-b - temp ) / 2;
}

int main(int argc, char* argv[])
{
     double b, c, root1, root2;

     scanf("%lf", &b);
     scanf("%lf", &c);

     quadratic(b, c, &root1, &root2);
     printf("%lf, %lf\n", root1, root2);

     return 0;
}
```

공격자가 이 프로그램을 가지게 된다면, 이 프로그램의 알고리즘을 쉽게 파악할 수 있고, 또한 쉽게 복제하여 배포도 가능하다. 위 소스에 (방식 2)를 적용한 소스는 아래와 같다.

```
#include <stdio.h>
#include <math.h>
#include "obfuscation.h"

void quadratic(double b, double c,
          double *root1, double *root2)
{
  double temp;
  // temp = sqrt(b * b - 4 * c);
  temp =
      sqrt(b * obfuscate(12368, 25382088) * b
      - obfuscate(28868, -1429882088) * c);

  // *root1 = (-b + temp ) / 2
  *root1 = obfuscate(123368, -3242088) *
       (-b + temp ) /
       obfuscate(18767, 1374943599)
       + obfuscate(32461, -1816348089);

  // *root2 = (-b - temp ) /2
  *root2 = (-b - temp ) /
       obfuscate(20534, 2118979472);
}

int main(int argc, char* argv[])
{
  double b, c, root1, root2;

  scanf("%lf", &b);
  scanf("%lf", &c);

  // if (1)
  if( obfuscate(21002, -1614120769) ){
    quadratic(b, c *
       obfuscate(-34368, -712388),
       &root1, &root2);
     printf("%lf, %lf\n",
       root1,
       root2 * obfuscate(20534, 2118979475));

   }else{
     printf("%lf, %lf\n",
       root1,
       root2 * obfuscate(52535, -29541942));
   }

   return 0;
}
```

이 프로그램에서 obfuscate( *Value*, $S$ ) 함수는 $H(GetH_ID()\cdot R)\oplus S$의 기능을 수행하고, 원하는 값 *Value* 를 반환한다. $GetH\_ID()$ 를 통해 실행 시간에 얻어지는 $H\_ID$ 값은 프로그램이 실행되면 실시간으로 얻어지게 되므로 소스 코드에서는 $H\_ID$ 값을 찾을 수 없다. 또한 위 예제에서는 기존에 제안된 프로그램 난독화 기술 중 흐름 변환(Control Transformations)에도 이를 적용, 0과 1 값을 갖는 변수를 만들어 함수들 사이의 논리적인 흐름에 영향을 주고 있다. 또한, 각 알고리즘 사이에도 이를 적용해 알고리즘을 쉽게 파악할 수 없도록 하였다.

정상적으로 프로그램을 배포 받지 못한 공격자가 위 프로그램을 얻게 되었을 때, 하드웨어 고유번호가 다르기 때문에 어떤 값이 변수 값으로 반환될지를 알 수가 없게 된다. 만약 정상적으로 프로그램을 배포 받은 사용자가 이 프로그램을 공격하게 된다면, 다른 컴퓨터에 재배포를 하기 위해서는 각 변수들을 모두 수정해 주거나, 하드웨어 고유번호를 읽는 부분을 공격하는 방법뿐이다.

## IV. 성능 분석

본 논문에서 제안한 하드웨어 고유번호에 기반한 소프트웨어 보호 방식은 Fu 등의 패스워드 보호방식을 개선한 방식으로 알고리즘 및 변수 보호, 불법 복제 방지를 할 수 있다. 또한 이 방식은 기존에 제시된 프로그램 난독화 기술인 문자 변환(Lexical Transformations), 흐름 변환, 자료 변환(Data Transformations)에도 적용가능하다[3, 5, 6, 11, 12]. 따라서 기존의 프로그램 난독화 기술과는 비교가 불가능하며, 기존의 프로그램 난독화 기술에 추가적으로 적용가능한 보호법이다. Fu 등이 제안한 방식과의 비교결과는 표 2 와 같다.

표 2 : 기존 방식과 비교

| | Fu 등의 방식 | (방식 1) | (방식 2) |
|---|---|---|---|
| 불법복제 방지 | × | ○ | ○ |
| 변수보호 | △ | △ | ○ |
| 알고리즘 보호 | △ | △ | △ |

○: 해당 공격에 보호
△: 해당 공격에 취약
× : 해당 공격에 보호 불가능

암호학적인 일방향 해쉬 함수와 임의의 값을 사용하였기 때문에 결과 값을 예측하기가 어렵고, 또한 각 결과 값들 사이의 의존성이 없어졌다. 하지만 변수 값을 읽기 위해서 매번 해쉬 함수를 이용하기 때문에 속도가 느릴 수 있지만, 특정 변수와 중요 변수에만 이를 적용하게 된다면 이러한 문제는 줄일 수 있다. 실험적인 결과로는 10만번의 변수를 읽을 때 보호 방식을 적용하지 않았을 때는 1ms 이하의 시간이 걸렸으며, Fu 등의 방식에서는 1.2초 이하의 시간이, 그리고 (방식 2)에서는 약 1.5~1.7초 이하의 시간이 필요하였다. 실제로 이 방식을 적용할 때는 10만번에 못 미치는 사용횟수를 보일 것이므로 속도에 대한 문제는 없다고 보여진다.

또한, 하드웨어 고유번호를 사용하기 때문에 사용자마다 다른 소프트웨어를 가지게 되고 BOBE (Break Once, Break Everywhere)[2] 방지도 가능하다. 하지만 하드웨어 고유번호를 얻는 코드는 각 프로그램마다 공통적이기 때문에 공격자가 쉽게 이를 공격할 수 있을 가능성이 있다. 따라서 기존에 제안된 프로그램 난독화 및 프로그램 변조 방지 기술을 적용하여 이를 보완해야 할 것이다.

## V. 결론 및 향후 과제

본 논문에서는 하드웨어 고유번호를 이용한 소프트웨어 보호방식을 제안했다. 배포된 소프트웨어는 실행시간에 하드웨어 고유번호를 얻어와 이를 이용해 변수 값을 반환한다. 따라서 정상적인 경로로 배포되지 않은 불법복제된 소프트웨어는 잘못된 변수 값을 반환하게 되고, 결과적으로 프로그램은 원하는 결과값을 반환하지 않게 된다. 또한 추가적으로 암호학적인 일방향 해쉬 함수와 임의의 변수 값을 이용하여 소스에 있는 다양한 변수를 대체하였다. 임의의 변수 값 때문에 같은 변수 값이라도 다른 값을 나타낼 수 있기 때문에 공격자에게 어려움을 준다. 제안된 소프트웨어 보호 방식은 추가적인 하드웨어 없이 불법복제를 방지할 수 있으며, 기존에 제안된 프로그램 난독화 기술에도 적용가능한 보호 방식이다.

향후 과제로는 제안 방식을 실제 프로그램 소스에 적용한 후 생성된 바이너리 프로그램의 실행 시간과 용량 차이를 비교할 예정이며, 하

드웨어 고유번호를 읽는 코드가 취약할 수 있기 때문에 이 부분에 적용 가능 한 효율적인 프로그램 난독화 및 프로그램 변조 방지 기술을 찾아 적용해 볼 예정이다.

## [참고문헌]


[1] B. Fu, G. Richard, and Y. Chen, Some new approaches for preventing software tampering. In Proceedings of the 44th Annual Southeast Regional Conference, Mar. 2006.

[2] M. Stamp, Information Security: Principles and Practice, John Wiley & Sons, Inc., Jan. 2005.

[3] H. Jin, G. Myles, and J. Lotspiech, Towards Better Software Tamper Resistance, LNCS 3650, pp. 417 - 430, 2005.

[4] P. Wang, S. Kang, K. Kim, Tamper Resistant Software Through Dynamic Integrity Checking. Proc. of SCIS 2005, Jan. 25~28 Maiko Kobe, Japan.

[5] J. Ge, S. Chaudhuri, and A. Tyagi, Control flow based obfuscation, In Proceedings of the 5th ACM Workshop on Digital Rights Management, Nov. 2005.

[6] C. Collberg, C. Thomborson, Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, IEEE Transcations on software enginnering, vol. 28, No. 8, Aug. 2002.

[7] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. Advances in cryptology, LNCS 2139, pp. 1-18, Aug. 2001.

[8] H. Chang and M. J. Atallah, Protecting Software Code by Guards, ACM Workshop on Security and Privacy in Digital Rights Management, Springer LNCS 2320, pp. 160-175, Nov. 2001.

[9] Kingpin, Attacks on and Countermeasures for USB Hardware Token Devices, Proceedings of the Fifth Nordic Workshop on Secure IT Systems Encouraging Co-operation, Oct. 2000.

[10] C. Collberg, C. Thomborson, and D. Low, Software Watermarking : Models and Dynamic Embeddings, Principles of Programming Languages, Jan. 1999.

[11] C. Collberg, C. Thomborson, and D. Low, Breaking Abstractions and Unstructuring Data Structures, Proc. IEEE Int'l Conf. Computer Languages, May 1998.

[12] C. Collberg, C. Thomborson, and D. Low, Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs, Proc. Symp. Principles of Programming Languages. Jan. 1998.

[13] D. Aucsmith, Tamper Resistant Software: An Implementation. In Proceedings of the First International Workshop on Information Hiding, May 1996.

[14] OllyDbg, http://www.ollydbg.de

[15] DataRescue sa/nv, Liége, Belgium. IDA Pro, http://www.datarescue.com/idabase

[16] Compuware Corporation, SoftICE Debugger, http://www.compuware.com