(19)대한민국특허청(KR)
(12) 등록특허공보(B1)

(51) 。Int. Cl.
G06F 12/14 (2006.01)

(45) 공고일자 2006년10월18일
(11) 등록번호 10-0636162
(24) 등록일자 2006년10월12일

(21) 출원번호 10-2004-0067190
(22) 출원일자 2004년08월25일

(65) 공개번호 10-2006-0018693
(43) 공개일자 2006년03월02일

(73) 특허권자 삼성전자주식회사
경기도 수원시 영통구 매탄동 416

(72) 발명자 남수현
서울특별시 서초구 방배2동 435-26 102호

김명선
경기도 의왕시 삼동 대우아파트 105동 104호

장용진
경기도 의왕시 이동 218-74

(74) 대리인 리앤목특허법인
이해영

(56) 선행기술조사문헌
JP2003050640 A
KR1020030092850 A
WO03071544 A1
KR100176131 B1
KR1020040027826 A
1001761310000
* 심사관에 의하여 인용된 문헌

심사관 : 최봉묵

(54) 소프트웨어 보호 방법 및 그 장치

요약

본 발명은 소프트웨어의 보호 방법에 관한 것으로, 보다 구체적으로는 소프트웨어 코드를 무작위로 분산시켜 소프트웨어를 보호하는 방법 및 그 장치에 관한 것이다. 본 발명에 따른 소프트웨어 보호 방법은, 소프트웨어의 코드를 복수개의 보호할 코드 영역과 일반 코드 영역으로 분할하는 단계; 보호할 코드 영역 중 적어도 하나의 셔플링할 대상 영역을 선택하고, 일반 코드 영역 중 적어도 하나의 씨드 영역을 선택하는 단계; 및 선택된 씨드 영역의 코드 값을 기초로 난수발생기를 이용하여 생성된 셔플링 규칙에 따라, 선택된 셔플링할 대상 영역의 코드를 셔플링하는 단계를 포함하는 것을 특징으로 한다.

이에 따라, 불법 크래커의 원본 코드에 대한 변조를 방지하고, 디버거 또는 디스어셈블러를 통한 공격이나 메모리 덤프 공격으로부터 소프트웨어를 효과적으로 보호할 수 있다.

대표도

도 5

명세서

도면의 간단한 설명

도 1은 본 발명의 바람직한 실시예에 따라 소프트웨어 코드를 셔플링하는 과정을 설명하기 위한 도면,

도 2는 본 발명의 바람직한 실시예에 따라 셔플링된 소프트웨어 코드를 디코딩하는 과정을 설명하기 위한 도면,

도 3은 본 발명의 일 실시예로서, 소프트웨어 코드를 셔플링하는 장치를 나타내는 블록도,

도 4는 본 발명의 일 실시예에 따라 셔플링하기 전의 원본 소스 코드와 셔플링한 후의 수정된 바이너리 코드를 나타내는 참고도,

도 5는 본 발명의 바람직한 실시예에 따라 소프트웨어 코드를 셔플링하는 방법을 나타내는 플로차트이다.

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 소프트웨어의 보호 방법에 관한 것으로, 보다 구체적으로는 소프트웨어 코드를 무작위로 분산시켜 소프트웨어를 보호하는 방법 및 그 장치에 관한 것이다.

컴퓨터의 사용이 급속도로 증가됨에 따라 비약적으로 발전한 소프트웨어 개발 기술은 이제 핸드폰이나 가정용 가전기기(Consumer Electronics: 이하 CE)에까지 적용되고 있다. 이러한 경향에 발맞춰 소프트웨어 산업은 중요한 부가가치 사업으로 발전하고 있으며, 법령으로도 소프트웨어에 대한 저작자의 저작권을 보호해주고 있다.

그러나, 일부 불법 크래커(Cracker)들에 의해 이러한 소프트웨어의 저작권은 제대로 보호받지 못하고 있다. 크래킹 기술의 발전 속도에 비해 소프트웨어 보호기술이 이를 제대로 따라가고 있지 못하기 때문이다. 특히, 컴퓨터나 CE 기기 및 기타 자동화 기기에 소프트웨어가 내장되면서 불법 크래커의 공격에 의해 소프트웨어 내의 중요 정보나 사용권한이 불법적으로 도용되어 원 저작자에게 막대한 손해를 입히고 있다.

발명이 이루고자 하는 기술적 과제

따라서 본 발명이 이루고자 하는 기술적 과제는 전술한 문제점을 해결하기 위하여, 소프트웨어 코드를 무작위로 분산시켜 소프트웨어를 보호하는 방법 및 그 장치를 제공하는 것이다.

발명의 구성 및 작용

전술한 기술적 과제는 소프트웨어의 코드를 복수개의 보호할 코드 영역과 일반 코드 영역으로 분할하는 단계; 보호할 코드 영역 중 적어도 하나의 셔플링할 대상 영역을 선택하고, 일반 코드 영역 중 적어도 하나의 씨드 영역을 선택하는 단계; 및 선택된 씨드 영역의 코드 값을 기초로 난수발생기를 이용하여 생성된 셔플링 규칙에 따라, 선택된 셔플링할 대상 영역의 코드를 셔플링하는 단계를 포함하는 것을 특징으로 하는 소프트웨어 보호 방법에 의해 달성된다.

또한, 상기 선택하는 단계는, 일반 코드 영역 중 셔플링된 코드를 디코딩할 함수를 삽입할 소정의 영역을 선택하는 단계를 더 포함하는 것이 바람직하다.

또한, 상기 일반 코드 영역은 복수개의 더 작은 영역으로 분할할 수 있으며, 씨드 영역 또는 디코딩할 함수를 삽입할 소정의 영역은 더 작은 영역 중 선택되는 것이 바람직하다.

또한, 상기 씨드 영역 또는 디코딩할 함수를 삽입할 소정의 영역을 선택하는 경우, 난수 발생기를 이용하여 랜덤한 영역을 선택하는 것이 바람직하다.

또한, 상기 셔플링 규칙은 선택된 셔플링할 대상 영역의 코드를 랜덤하게 섞기 위해 사용되는 규칙이며, 씨드 영역의 코드 값을 초기값으로 하여 랜덤하게 생성되는 것이 바람직하다.

또한, 상기 난수 발생기는 리니어 피드백 쉬프트 레지스터(LFSR)을 포함하는 것이 바람직하다.

한편, 본 발명의 다른 분야에 따르면 전술한 기술적 과제는, 전술한 방법을 수행하는 프로그램을 기록한 컴퓨터 판독 가능한 기록 매체에 의해 달성된다.

한편, 본 발명의 다른 분야에 따르면 전술한 기술적 과제는, 소프트웨어의 코드를 복수개의 보호할 코드 영역과 일반 코드 영역으로 분할하고, 보호할 코드 영역 중 적어도 하나의 셔플링할 대상 영역과, 일반 코드 영역 중 적어도 하나의 씨드 영역을 선택하는 코드 선택기; 선택된 씨드 영역의 코드 값을 기초로 난수 발생기를 이용하여 소정의 셔플링 규칙을 생성하는 셔플링 규칙 생성기; 및 생성된 셔플링 규칙에 따라, 선택된 셔플링할 대상 영역의 코드를 셔플링하는 코드 셔플러를 포함하는 것을 특징으로 하는 소프트웨어 보호 장치에 의해 달성된다.

이하, 첨부한 도면을 참조하여 본 발명의 바람직한 실시예를 상세히 설명한다.

먼저, 셔플링(shuffling)이란, 디지털 데이터의 기록시 데이터의 기록 순서를 시간 축 방향으로 전후를 바꾸어 기록하는 것을 말한다. 즉, 디지털 데이터를 무작위로 뒤섞어서 기록하는 것을 의미한다.

본 발명은 소프트웨어 코드를 쉐플링하는 기술을 이용한 소프트웨어 보호방법 및 그 장치에 관한 것으로, 불법적인 크래커의 공격으로부터 소프트웨어를 보호하기 위하여 소프트웨어 코드를 쉐플링하여 무작위로 분산시킴으로써, 크래커의 소프트웨어 코드에 대한 해석 및 접근을 어렵게 만드는 것을 그 목적으로 한다.

도 1은 본 발명의 바람직한 실시예에 따라 소프트웨어 코드를 셔플링하는 과정을 설명하기 위한 도면이다.

도 1을 참조하면, 셔플링 인코더(1)는 소프트웨어 원본 코드(102)의 일부분을 셔플링을 위한 씨드(seed)로 사용하여 원본 코드(102)를 셔플링한다. 셔플링 결과, 수정된 코드와 셔플링에 사용된 셔플링 씨드, 그리고 디코딩시에 사용될 셔플링 디코더(104)(이하, 설명의 편의상 수정된 코드라고 약칭한다)가 출력된다. 수정된 코드(104)는 소프트웨어의 코드가 무작위로 뒤섞여서, 후술하는 셔플링 디코딩 과정을 거치지 않는 한, 정상적으로 실행될 수 없다. 따라서, 불법적 크래커의 접근을 방지할 수 있다.

도 2는 본 발명의 바람직한 실시예에 따라 셔플링된 소프트웨어 코드를 디코딩하는 과정을 설명하기 위한 도면이다.

도 2 및 도 1을 참조하면, 셔플링 인코더(1)에 의해 수정된 코드(104)는 실행을 위해 시스템 메모리(도시하지 않음)에 로드된다. 특히, 수정된 코드(104)에 포함된 셔플링 디코더(2)는 시스템 메모리에 로드되어 수정된 코드를 디코딩한다. 디코딩을 위하여 수정된 코드(104)에 포함된 셔플링 씨드가 사용된다. 디코딩 결과, 셔플링되었던 수정된 코드가 원본 코드(106)로 복원된다.

이 때, 불법적 크래커의 공격으로 수정된 코드의 일부분이 변조되었다면, 셔플링 씨드 값도 변하게 되고, 이에 따라 원본 코드와는 다른 바이너리 코드로 복원된다. 따라서, 원본 코드의 복원 및 정상적인 실행에 실패하게 된다.

반면, 불법적 크래커의 공격이 없었다면, 수정된 코드에 포함된 셔플링 씨드의 값도 변함이 없으며, 이에 따라 원본 코드의 복원 및 정상적인 실행에 성공하게 된다.

도 3은 본 발명의 일 실시예로서, 소프트웨어 코드를 셔플링하는 장치를 나타내는 블록도이다.

도 3을 참조하면, 소프트웨어 코드를 셔플링하는 장치로서 도 1에 도시된 셔플링 인코더(1)는, 코드 선택기(322), 난수 발생기(324), 셔플링 규칙 생성기(326), 및 코드 셔플러(328)를 포함한다.

원본 소스 코드(300)는 저작자가 생성한 소프트웨어 코드로서 C, C++, 어셈블러, 마크업 언어 등 다양한 언어로 표현된 소스 코드이다. 원본 소스 코드(300)는, 크래커의 공격에 취약하여 보호할 코드 영역(S)과, 일반 코드 영역(A)으로 분할되며, 코드 분할에 관한 정보는 코드 선택기(322)로 전달된다. 코드 분할을 사용자가 선택하게 할 수도 있고, 셔플링 장치에서 선택하게 할 수도 있음은 물론이다. 이 때, 일반 코드 영역은 복수개의 더 작은 영역으로 분할할 수 있다.

코드 선택기(322)는, 코드 분할에 관한 정보를 전달 받아, 셔플링에 필요한 영역을 선택한다. 코드 선택기는 다음과 같이 크게 3가지 동작을 수행한다.

첫째, 전달받은 코드 분할에 관한 정보를 바탕으로, 보호할 코드 영역 중에서 셔플링할 영역(Si)을 선택한다. 또한, 일반 코드 영역 중에서 디코딩할 함수를 삽입할 영역(Aj)를 선택한다. 나아가, 셔플링 규칙을 생성하기 위하여 난수 발생기(324)에서 초기값으로 사용될 씨드 영역(Ck)을 선택한다. 이 때, 디코딩할 함수를 삽입할 영역(Aj) 및 씨드 영역(Ck)은 난수 발생기(324)를 이용하여 생성된 난수값을 바탕으로 랜덤하게 선택할 수 있다. 또한, 씨드 영역 또는 디코딩할 함수를 삽입할 영역은 전술한 바와 같이 일반 코드 영역을 더 작은 영역을 분할한 경우, 이 중 적어도 하나를 선택할 수 있다.

둘째, 원본 소스 코드에 대하여 디코딩할 함수를 삽입할 영역(Aj)에 도 1에 도시한 셔플링 디코더, 즉 디코딩 함수를 삽입한다. 또한, 원본 소스 코드에 대하여 셔플링할 영역(Si) 및 씨드 영역(Ck)의 위치를 나타내기 위한 플래그를 설정한다. 이제 디코딩 함수가 삽입되고, 플래그가 설정된 원본 소스 코드는 컴파일러(310)로 전달된다. 컴파일러(310)는 원본 소스 코드를 컴파일하여, 시스템에서 실행할 수 있는 바이너리 코드(binary code)를 생성한다.

셋째, 셔플링 규칙을 생성하기 위하여, 난수 발생기(324)의 초기값으로 사용될 씨드 영역(Ck)의 코드를 난수 발생기(324)로 전달한다.

난수 발생기(324)는, 디코딩할 함수를 삽입할 영역(Aj) 및 씨드 영역(Ck)을 랜덤하게 선택하는데 사용될 난수값을 생성한다. 또한, 씨드 영역(Ck)의 코드를 초기값으로 이용하여 셔플링 규칙을 생성하는데 사용될 난수값을 생성한다. 난수 발생기는 피드백 리니어 쉬프트 레지스터(Feedback Linear Shift Register: 이하 LFSR이라 약칭한다)가 대표적이다. 물론, 그 밖에 다양한 종류의 난수 발생기를 사용할 수도 있다.

셔플링 규칙 생성기(326)는, 난수 발생기(324)에서 생성된 난수값을 바탕으로 셔플링 규칙을 생성한다. 셔플링 규칙(Ri)은 선택된 셔플링할 대상 영역(Si)의 코드를 랜덤하게 섞기 위해 사용되는 규칙이며, 씨드 영역(Ck)의 코드 값을 초기값으로 하여 랜덤하게 생성되는 것을 특징으로 한다.

코드 셔플러(328)는, 랜덤하게 생성된 셔플링 규칙(Ri)을 해당 셔플링할 영역(Si)에 적용하여 셔플링할 영역의 코드를 셔플링한다. 이 때, 셔플링은 컴파일러(310)를 통해 컴파일된 바이너리 코드에 대해 수행된다. 이에 따라, 크래커의 공격에 취약한 보호할 영역이 랜덤하게 셔플링된, 수정된 바이너리 코드(330)를 얻을 수 있다. 수정된 바이너리 코드(330)에는 디코딩할 함수를 삽입할 영역(Aj)에 디코딩 함수가 삽입되며, 디코딩 시에 사용될 셔플링 씨드로서 씨드 영역(Ck)에 대한 정보가 함께 포함된다.

전술한 셔플링 인코더(1)의 동작을 살펴보면, 셔플링 인코더(1)는 원본 소스 코드(300)와 전술한 코드 분할에 대한 정보를 전달받아 코드 선택기(333)를 통해 셔플링할 대상 영역(Si)과 디코딩할 함수를 삽입할 영역(Aj) 및 셔플링 씨드 영역(Ck)을 선택한다. 이 때, 난수 발생기(324)로부터 난수값을 받아 상기 영역들을 선택할 수 있으며, 소프트웨어의 복잡도에 따라 초기에 설정된 회수(N)만큼 반복할 수 있다. 선택된 정보를 기초로 하여 원본 소스 코드에 디코딩 함수 및 각 영역의 위치 정보를 삽입하고 컴파일러(310)에게 전달한다. 컴파일러(310)를 통해 컴파일된 바이너리 코드는 코드 셔플러(328)에게 전달된다. 코드 셔플러는 셔플링 씨드 영역(Ck)의 코드값을 초기값으로 랜덤하게 생성된 셔플링 규칙에 따라 전달 받은 바이너리 코드 중 셔플링할 대상 영역(Si)의 바이너리 코드를 셔플링한다. 초기에설정된 반복회수만큼 반복 수행할 수 있다.

한편, 수정된 바이너리 코드를 디코딩하는 동작을 살펴보자. 수정된 바이너리 코드를 시스템 메모리에 로드하여 실행하는 중, 디코딩할 함수를 삽입할 영역(Aj)에 삽입된 디코딩 함수가 자동으로 실행된다. 디코딩 함수는 삽입된 씨드 영역(Ck)과 셔플링할 대상 영역(Si)에 대한 위치 정보를 바탕으로 난수 발생기의 초기값을 생성하여 디코딩 규칙을 생성하고, 생성된 디코딩 규칙에 따라 셔플링할 대상 영역(Si) 위치의 코드를 원래의 상태로 복원한다. 즉, 대상 영역의 셔플링된 코드들을 제자리로 되돌려 놓아 원본 코드로 복원하는 것이다. 특히, 수정된 바이너리 코드의 런타임(run-time)시에는 실행에 필요한 부분만 복원되었다가 실행이 끝난 후 바로 셔플링하도록 하면, 크래커의 메모리 덤프 공격을 막을 수 있다.

이 때, 불법적 크래커의 공격으로 수정된 코드의 일부분이 변조되었다면, 셔플링 씨드 값도 변하게 되고, 셔플링 씨드 값을 기초로 생성된 디코딩 규칙도 달라지게 된다. 이에 따라, 원본 코드와는 다른 바이너리 코드로 복원되며, 원본 코드의 복원 및 정상적인 실행에 실패하게 된다. 반면, 불법적 크래커의 공격이 없었다면, 수정된 코드에 포함된 셔플링 씨드의 값도 변함이 없으며, 생성되는 디코딩 규칙도 변함이 없게되어 원본 코드와 동일한 바이너리 코드로 복원된다. 즉, 원본 코드의 복원 및 정상적인 실행에 성공하게 된다. 따라서, 전술한 셔플링 기술을 이용하여 불법적인 크래커의 공격을 방어하고 소프트웨어를 보호할 수 있다.

도 4는 본 발명의 일 실시예에 따라 셔플링하기 전의 원본 소스 코드와 셔플링한 후의 수정된 바이너리 코드를 나타내는 참고도이다.

도 4를 참조하면, 좌측에는 셔플링하기 전의 원보 소스 코드(300)의 일 예가, 우측에는 셔플링한 후의 수정된 바이너리 코드(330)의 일 예가 도시된다.

먼저, 원본 소스 코드는 저작자 또는 셔플링 인코더(1)에 의해서 복수개의 보호할 코드 영역(S)과, 복수개의 일반 코드 영역(A)으로 분할된다. 이 때, 일반 코드 영역은 복수개의 더 작은 영역으로 분할할 수 있다. 셔플링 인코더(1)는 분할된 복수의 보호할 코드 영역 중 셔플링할 영역(Si)을 선택한다. 또한, 일반 코드 영역 중에서 디코딩할 함수를 삽입할 영역(Aj)과 씨드 영역(Ck)을 선택한다. 이 때, 디코딩할 함수를 삽입할 영역(Aj) 및 씨드 영역(Ck)은 난수 발생기(324)를 이용하여 생성된 난수값을 바탕으로 랜덤하게 선택할 수 있다.

도 4의 좌측에 도시된 예를 살펴보면, 셔플링할 영역(Si)로 S2가 선택되고, 디코딩 함수를 삽입할 영역(Aj)으로 A4가 선택되며, 씨드 영역(Ck)으로 A2가 선택된다. 셔플링 인코더(1)는 씨드 영역으로 선택된 A2 영역의 코드 값을 난수발생기의 초기값으로 사용하여 발생된 난수값을 기초로 하여 셔플링 규칙을 생성하며, 셔플링된 영역을 다시 복원하기 위해 사용할 디코딩 함수를 디코딩 함수를 삽입할 영역으로 선택된 A4 영역에 삽입한다. 또한, 원본 소스 코드는 컴파일러(310)에 의해 바이너리 코드로 컴파일된다. 이제 생성된 셔플링 규칙에 따라 바이너리 코드 중 셔플링할 영역으로 선택된 S2 영역의 코드 값들을 랜덤하게 셔플링한다. 이러한 방식으로 수정된 바이너리 코드(330)의 일 예가 도 4의 우측에 도시되어 있다. 한편, 도 4에는 각 영역이 동일한 크기로 표시되어 있으나, 각 영역의 크기는 다양하게 분할될 수 있다. 또한, 전술한 셔플링은 복수회 반복될 수 있다.

도 5는 본 발명의 바람직한 실시예에 따라 소프트웨어 코드를 셔플링하는 방법을 나타내는 플로차트이다.

도 5를 참조하면, 셔플링 인코더(1)는 원본 소스 코드를 입력받고, 셔플링 반복 회수(N)를 결정한다(510 단계). 셔플링 반복회수는 예를 들어, 프로그램의 복잡도에 따라 저작자로부터 입력받을 수도 있고, 셔플링 인코더(1)가 설정할 수도 있다. 원본 소스 코드 중 불법 크래커의 공격을 받기 쉬운 코드 영역(S)과 일반 적인 코드 영역(A)으로 분할한다(520 단계). 분할된 영역들 중 셔플링할 영역(Si), 디코딩 함수가 삽입될 영역(Aj), 및 셔플링 씨드로 사용될 씨드 영역(Ck)를 선택한다(530 단계 내지 550 단계). 각 영역은 난수 발생기를 통해 임의의 값으로 선택할 수 있다. 셔플링 인코더(1)는 선택된 씨드 영역(Ck)의 코드값을 초기값으로 사용하여 난수발생기에 의해 생성된 난수값을 바탕으로 셔플링 규칙(Ri)을 생성한다. 이제 컴파일러로부터 원본 소스 코드를 컴파일한 바이너리 코드를 전달받아, 선택된 셔플링할 영역(Si)에 대하여, 생성된 셔플링 규칙(Ri)을 적용하여 셔플링을 수행한다. 또한, 선택된 디코딩 함수가 삽입될 영역에 상기 생성된 셔플링 규칙에 기초한 디코딩 함수를 삽입한다(560 단계). 전술한 530 단계 내지 560 단계의 동작을 설정된 반복 회수만큼 반복한다(570 단계). 이제 코드의 중요한 부분이 랜덤하게 뒤섞인 수정된 바이너리 코드를 출력한다(580 단계).

본 발명은 또한 컴퓨터로 읽을 수 있는 기록 매체에 컴퓨터가 읽을 수 있는 코드로서 구현될 수 있다. 컴퓨터가 읽을 수 있는 기록매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다. 컴퓨터가 읽을 수 있는 기록 매체의 예로는 ROM, RAM, CD-ROM, 자기 테이프, 플로피 디스크, 광 디스크 등이 있으며, 또한 캐리어 웨이브(예를 들어, 인터넷을 통한 전송)의 형태로 구현되는 것을 포함한다. 또한, 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어, 분산 방식으로 컴퓨터가 읽을 수 있는 코드로 저장되고 실행될 수 있다.

이상의 설명은 본 발명의 일 실시예에 불과할 뿐, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자는 본 발명의 본질적 특성에서 벗어나지 않는 범위에서 변형된 형태로 구현할 수 있을 것이다. 따라서, 본 발명에 따른 실시예에 한정되지 않고 특허 청구범위에 기재된 내용과 동등한 범위 내에 있는 다양한 실시 형태가 포함되도록 해석되어야 할 것이다.

발명의 효과

전술한 바와 같이 본 발명에 따르면, 소프트웨어 코드를 무작위로 분산시켜 소프트웨어를 보호하는 방법 및 그 장치가 제공된다.

즉, LFSR과 같은 난수 발생기를 이용하여 난수를 발생시켜 소프트웨어의 코드를 랜덤하게 분산시킴으로써 크래커의 공격으로부터 소프트웨어를 보호할 수 있다. 특히, LFSR은 의사 난수의 생성을 위해 사용되는 구조로써 빠른 속도와 좋은 성능을 가지고 있으며, tab 수의 변화나 다중 구조 등을 통해 보다 뛰어난 성능을 보장해준다.

또한, 본 발명은 크래커의 공격에 취약한 코드 영역, 예를 들면 비교동작을 수행하는 코드 영역을 선택하고, 원본 코드의 일부분을 씨드 값으로 사용하여 랜덤한 셔플링 규칙을 생성함으로써, 크래커의 원본 코드에 대한 변조를 방지할 수 있다. 왜냐하면, 크래커의 공격 결과는 원본 코드를 변조하며, 이러한 코드의 변화는 원본 코드의 일부분인 씨드의 변화를 이끌어, 잘못된 디코딩 규칙을 생성하게 하고, 결국 원본 코드의 본원에 실패하도록 만들기 때문이다.

또한, 본 발명에 따라 셔플링된 코드는 크래커가 주로 사용하는 디버거(debugger)나 디스어셈블러(disassembler)에서 잘못된 해석을 유도하기 때문에 이러한 공격 툴의 사용을 원천적으로 봉쇄할 수 있다.

나아가, 본 발명에 따른 소프트웨어 보호 방법은 원본 코드를 여러 영역으로 분할하여 적용하기 때문에, 크래커의 메모리 전체에 대한 덤프(dump) 공격을 막을 수 있다. 왜냐하면, 메모리 덤프 공격을 위해서는 코드 전체가 메모리상에 로드되어야 하는데, 본 발명에 따른 소프트웨어 보호 방법은 런타임(runtime)시에만 원본 코드의 일부, 즉 실행에 필요한 일부만을 복원하고 해당 코드의 실행 후에는 바로 다시 셔플링하여, 코드 전체가 메모리에 복원된 상태로 존재하는 경우는 발생하지 않기 때문이다.

(57) 청구의 범위

청구항 1.

소프트웨어의 코드를 복수개의 보호할 코드 영역과 일반 코드 영역으로 분할하는 단계;

상기 보호할 코드 영역 중 적어도 하나의 셔플링할 대상 영역을 선택하고, 상기 일반 코드 영역 중 적어도 하나의 씨드 영역을 선택하는 단계; 및

상기 선택된 씨드 영역의 코드 값을 기초로 난수발생기를 이용하여 생성된 셔플링 규칙에 따라, 상기 선택된 셔플링할 대상 영역의 코드를 셔플링하는 단계를 포함하는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 2.

제1항에 있어서,

상기 선택하는 단계는,

상기 일반 코드 영역 중 셔플링된 코드를 디코딩할 함수를 삽입할 소정의 영역을 선택하는 단계를 더 포함하는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 3.

제2항에 있어서,

상기 일반 코드 영역은 복수개의 더 작은 영역으로 분할할 수 있으며, 상기 씨드 영역 또는 상기 디코딩할 함수를 삽입할 소정의 영역은 상기 더 작은 영역 중 선택되는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 4.

제2항에 있어서,

상기 씨드 영역 또는 상기 디코딩할 함수를 삽입할 소정의 영역을 선택하는 경우, 상기 난수 발생기를 이용하여 랜덤한 영역을 선택하는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 5.

제1항에 있어서,

상기 셔플링 규칙은 상기 선택된 셔플링할 대상 영역의 코드를 랜덤하게 섞기 위해 사용되는 규칙이며, 상기 씨드 영역의 코드 값을 초기값으로 하여 랜덤하게 생성되는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 6.

제1항에 있어서,

상기 난수 발생기는 리니어 피드백 쉬프트 레지스터(LFSR)를 포함하는 것을 특징으로 하는 소프트웨어 보호 방법.

청구항 7.

제1항 내지 제6항 중 어느 한 항의 방법을 수행하는 프로그램을 기록한 컴퓨터 판독 가능한 기록 매체.

청구항 8.

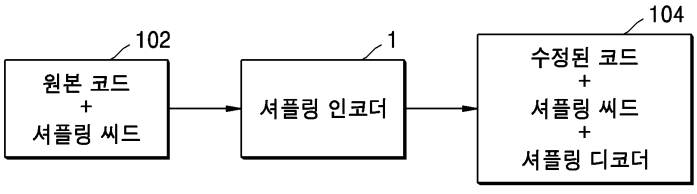
소프트웨어의 코드를 복수개의 보호할 코드 영역과 일반 코드 영역으로 분할하고, 상기 보호할 코드 영역 중 적어도 하나의 셔플링할 대상 영역과, 상기 일반 코드 영역 중 적어도 하나의 씨드 영역을 선택하는 코드 선택기;

상기 선택된 씨드 영역의 코드 값을 기초로 난수 발생기를 이용하여 소정의 셔플링 규칙을 생성하는 셔플링 규칙 생성기; 및

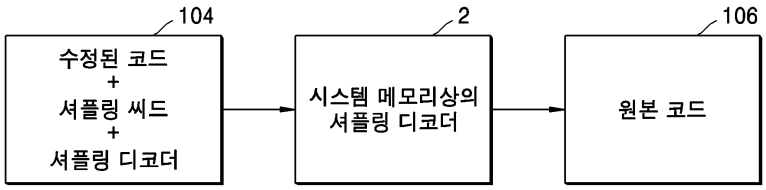
상기 생성된 셔플링 규칙에 따라, 상기 선택된 셔플링할 대상 영역의 코드를 셔플링하는 코드 셔플러를 포함하는 것을 특징으로 하는 소프트웨어 보호 장치.

도면

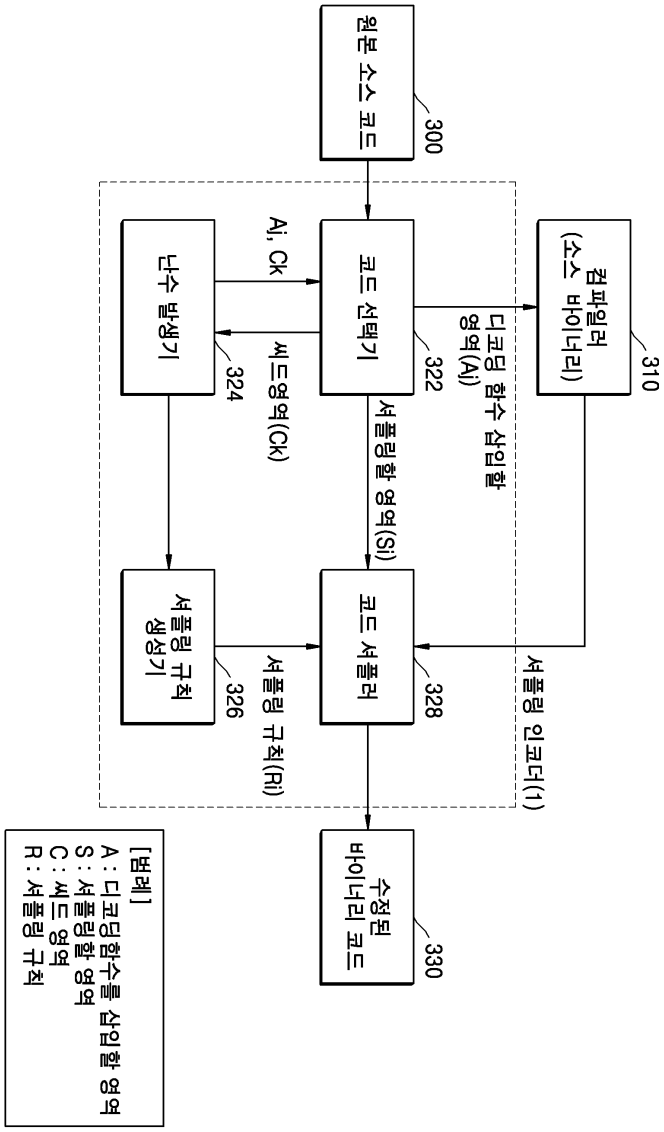
도면1



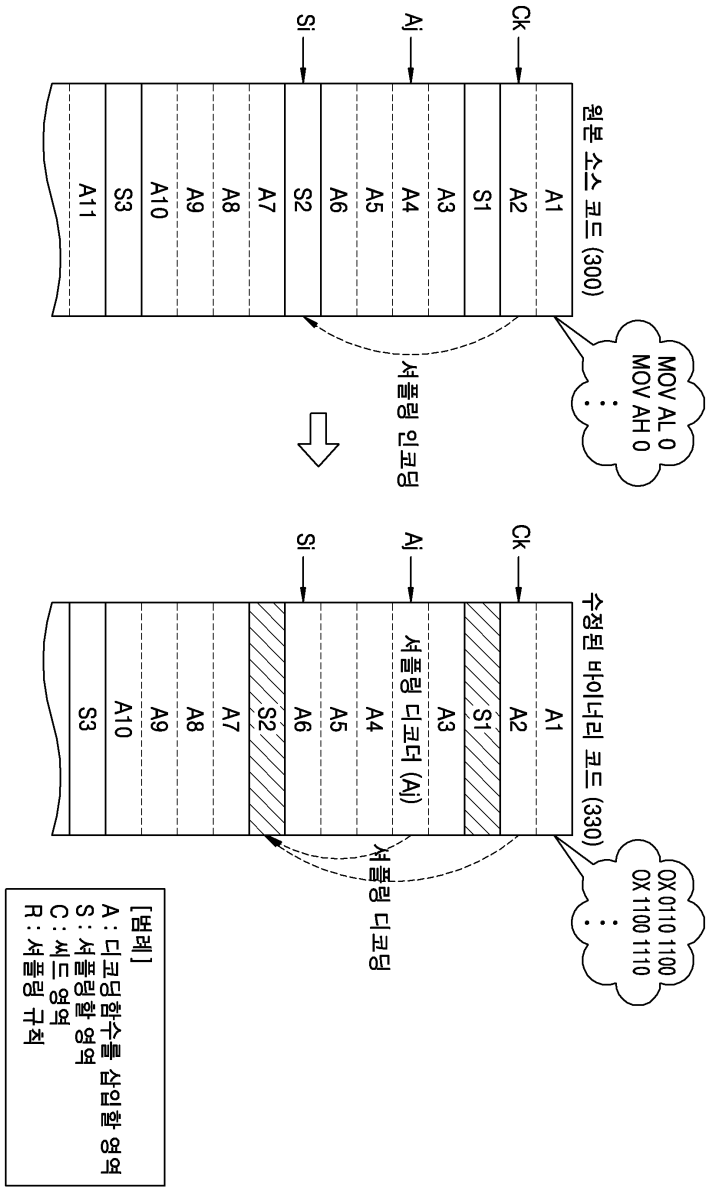
도면2


104
수정된 코드
+
셔플링 씨드
+
셔플링 디코더
2
시스템 메모리상의
셔플링 디코더
106
원본 코드

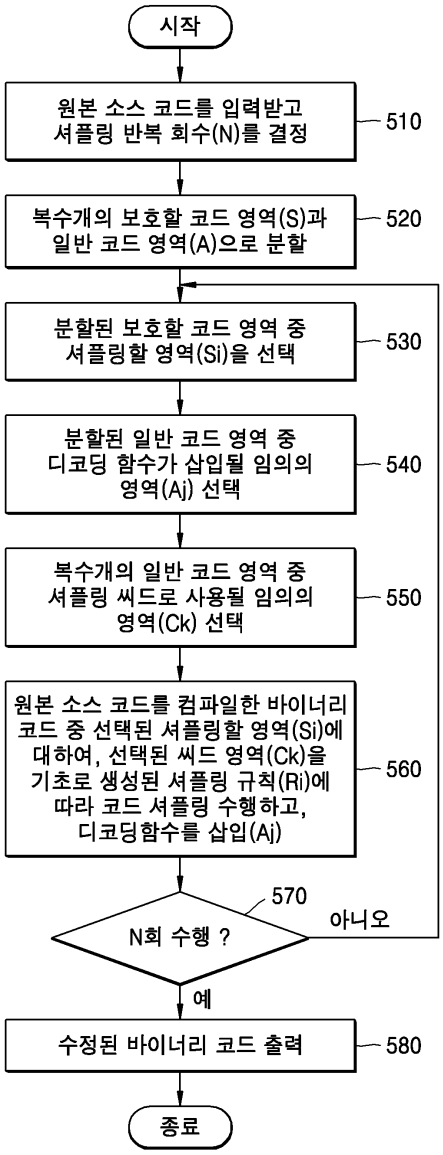
도면3


원본 소스 코드
300
난수 발생기
Ai, Ck
코드 선택기
디코딩 함수 삽입할 영역(Ai)
컴파일러 (소스 바이너리)
310
324
씨드영역(Ck)
322
셔플링할 영역(Si)
셔플링 규칙 생성기
326
셔플링 규칙(Ri)
코드 셔플러
328
셔플링 인코더(1)
수정된 바이너리 코드
330
[범례]
A : 디코딩함수를 삽입할 영역
S : 셔플링할 영역
C : 씨드영역
R : 셔플링 규칙

도면4


원본 소스 코드 (300)
Ck
Aj
Si
A1
A2
S1
A3
A4
A5
A6
S2
A7
A8
A9
A10
S3
A11
MOV AL 0
MOV AH 0
셔플링 인코딩
수정된 바이너리 코드 (330)
OX 0110 1100
OX 1100 1110
셔플링 디코더 (Aj)
셔플링 디코딩
[범례]
A : 디코딩함수를 삽입할 영역
S : 셔플링할 영역
C : 씨드영역
R : 셔플링 규칙

도면5


시작
원본 소스 코드를 입력받고 셔플링 반복 회수(N)를 결정
510
복수개의 보호할 코드 영역(S)과 일반 코드 영역(A)으로 분할
520
분할된 보호할 코드 영역 중 셔플링할 영역(Si)을 선택
530
분할된 일반 코드 영역 중 디코딩 함수가 삽입될 임의의 영역(Aj) 선택
540
복수개의 일반 코드 영역 중 셔플링 씨드로 사용될 임의의 영역(Ck) 선택
550
원본 소스 코드를 컴파일한 바이너리 코드 중 선택된 셔플링할 영역(Si)에 대하여, 선택된 씨드 영역(Ck)을 기초로 생성된 셔플링 규칙(Ri)에 따라 코드 셔플링 수행하고, 디코딩함수를 삽입(Aj)
560
570
N회 수행 ?
아니오
예
수정된 바이너리 코드 출력
580
종료