(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2023년01월18일
(11) 등록번호 10-2490331
(24) 등록일자 2023년01월16일

(51) 국제특허분류(Int. Cl.)
G06F 21/71 (2013.01) *G06F 21/53* (2013.01)
G06F 21/62 (2013.01) *G06F 9/30* (2018.01)
G06F 9/455 (2018.01) *H04L 9/06* (2006.01)
(52) CPC특허분류
G06F 21/71 (2013.01)
G06F 21/53 (2013.01)
(21) 출원번호 10-2017-7016354
(22) 출원일자(국제) 2015년12월16일
심사청구일자 2020년12월16일
(85) 번역문제출일자 2017년06월14일
(65) 공개번호 10-2017-0101912
(43) 공개일자 2017년09월06일
(86) 국제출원번호 PCT/US2015/066080
(87) 국제공개번호 WO 2016/100506
국제공개일자 2016년06월23일
(30) 우선권주장
62/092,570 2014년12월16일 미국(US)
(56) 선행기술조사문헌
US20130007469 A1
US20020046092 A1
US20060294238 A1

(73) 특허권자
킨디 인코포레이티드
미국 캘리포니아주 94402 샌 마티오 스위트 500 사우스 엘 카미노 리얼 1300
(72) 발명자
마준다 아룬
미국 캘리포니아주 94063 레드우드 시티 브로드웨이 스트리트 425 킨디 인코포레이티드
람세이 마틴 에스
미국 캘리포니아주 94063 레드우드 시티 브로드웨이 스트리트 425 킨디 인코포레이티드
(74) 대리인
제일특허법인(유)

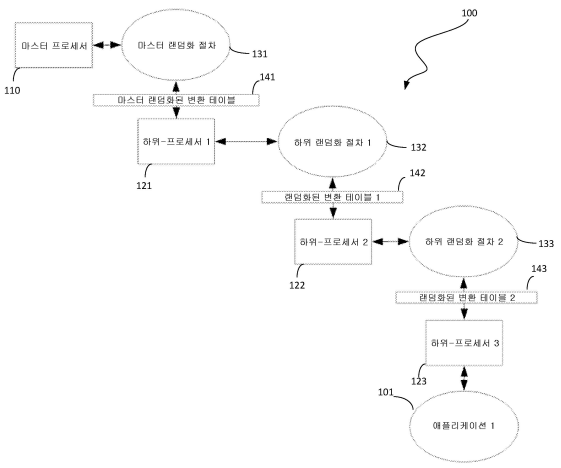
전체 청구항 수 : 총 33 항 심사관 : 구대성

(54) 발명의 명칭 **컴퓨터 인스트럭션 세트, 메모리 레지스터 및 포인터를 랜덤화하기 위한 방법 및 장치**

(57) 요 약

본원에서는, 악의적인 소프트웨어, 악의적인 컴퓨터 사용자, 또는 리버스 엔지니어가 새로운 인스트럭션 세트, 메모리 레지스터 및 포인터의 의미를 이해하기 위한 자원 사용 요구사항들을 증가시킴으로써 컴퓨터 보안을 향상시키도록 인스트럭션 세트, 메모리 레지스터 및 포인터를 랜덤화하는 방법 및 장치가 기술된다.

대 표 도



(52) CPC특허분류
G06F 21/6209 (2013.01)
G06F 9/30178 (2013.01)
G06F 9/45558 (2013.01)
H04L 9/06 (2013.01)
G06F 2009/45587 (2019.08)
G06F 2009/45591 (2019.08)

명 세 서

청구범위

청구항 1

컴퓨터 보안을 향상시키기 위해 인스트럭션을 랜덤화하기 위한 방법으로서,

프로세싱 회로에서 실행되는 내측 프로세싱 시스템(IPS)에서 인스트럭션을 암호화하는 단계- 상기 프로세싱 회로는 상기 프로세싱 회로 상에서 실행되는 외측 프로세싱 시스템(OPS)을 포함하고, 메모리에 대한 상기 IPS의 액세스는 상기 메모리의 제1 부분으로 제한되고, 상기 OPS는, 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능한 상기 메모리의 적어도 제2 부분을 사용함 -와,

상기 암호화된 인스트럭션을 상기 IPS로부터 상기 OPS로 전송하는 단계와,

상기 OPS에 의해서, 수신된 상기 암호화된 인스트럭션을 암호해독하는 단계 - 상기 암호해독된 인스트럭션은 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능함 -와,

상기 암호해독된 인스트럭션을 OPS 인스트럭션의 세트로 변환(translating) 하는 단계- 상기 변환은 상기 암호해독된 인스트럭션을 난독화하고, 상기 OPS 인스트럭션의 세트의 예측가능성을 감소시키는 2-레벨 문법을 사용하여 수행됨 - 와,

상기 OPS 인스트럭션의 세트를, 상기 IPS가 아닌 가상 머신(VM)에서 스텁 루틴(stub routine)을 통해 실행하는 단계를 포함하되,

상기 OPS 인스트럭션의 세트를 실행하는 상기 VM은 상기 메모리의 상기 제1 부분에 저장된 데이터에 액세스하고, 상기 메모리의 상기 제2 부분 내의 데이터에는 액세스하지 않으며, 상기 IPS에 의해 상기 IPS 내의 인스트럭션을 실행함으로써 달성되는 것과 동일한 결과를 실현하는

인스트럭션 랜덤화 방법.

청구항 2

제 1 항에 있어서,

상기 2-레벨 문법은 Van Wijngaarden 문법인,

인스트럭션 랜덤화 방법.

청구항 3

제 2 항에 있어서,

상기 Van Wijngaarden 문법에 기초하여, 상기 VM을 위한 메모리 레지스터 및 포인터를 랜덤하게 생성하는 단계를 더 포함하는,

인스트럭션 랜덤화 방법.

청구항 4

제 1 항에 있어서,

상기 OPS에 의해, 대칭적 암호화 알고리즘에 기초하여, SDE(semantic dictionary encryption)를 생성하는 단계를 더 포함하는,

인스트럭션 랜덤화 방법.

청구항 5

제 1 항에 있어서,

상기 스텁 루틴은 상기 OPS 내에 포함되고,

상기 IPS에 의해 수행되는 암호화는 비대칭적 암호화인,

인스트럭션 랜덤화 방법.

청구항 6

제 5 항에 있어서,

상기 비대칭적 암호화는 RSA 암호화 및 ECC(Elliptic Curve Cryptography) 암호화 중 하나인,

인스트럭션 랜덤화 방법.

청구항 7

제 4 항에 있어서,

상기 대칭적 암호화 알고리즘은 RC4, RC5, SHA-1 및 MD5 알고리즘 중 하나 인,

인스트럭션 랜덤화 방법.

청구항 8

제 1 항에 있어서,

상기 OPS에 의해, 상기 실행된 인스트럭션을 다른 가상 머신으로 전송하는 단계를 더 포함하며,

상기 다른 가상 머신은 가상 머신들의 계층에서 상기 VM의 바로 하위에 있는 가상 머신인,

인스트럭션 랜덤화 방법.

청구항 9

제 8 항에 있어서,

상기 가상 머신들의 계층에서 각각의 가상 머신은 고유 2-레벨 문법을 사용하여 변환된 상기 암호해독된 인스트럭션의 일부를 제공받는,

인스트럭션 랜덤화 방법.

청구항 10

제 8 항에 있어서,

상기 가상 머신들의 계층에 포함된 가상 머신들의 총 개수는 애플리케이션 실행 동안 랜덤하게 선택되는,

인스트럭션 랜덤화 방법.

청구항 11

제 10 항에 있어서,

상기 가상 머신들의 계층에 포함된 랜덤한 개수의 추가 가상 머신들을 생성하는 단계를 더 포함하는,

인스트럭션 랜덤화 방법.

청구항 12

프로그램이 저장된 비일시적 컴퓨터 판독 가능 저장 매체로서,

상기 프로그램은 컴퓨터에 의해 실행될 때, 상기 컴퓨터로 하여금, 컴퓨터 보안을 향상시키기 위해 인스트럭션을 랜덤화하기 위한 방법을 실행하게 하며,

상기 방법은,

메모리의 제1 부분만을 액세스하는 내측 프로세싱 시스템(IPS)에서 인스트럭션을 암호화하는 단계와,

상기 암호화된 인스트럭션을 상기 IPS로부터 외측 프로세싱 시스템(OPS)으로 전송하는 단계- 상기 OPS는, 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능한 상기 메모리의 적어도 제2 부분을 사용함 -와,

상기 OPS에 의해서, 수신된 상기 암호화된 인스트럭션을 암호해독하는 단계 - 상기 암호해독된 인스트럭션은 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능함 -와,

상기 암호해독된 인스트럭션을 OPS 인스트럭션의 세트로 변환하는 단계- 상기 변환은 상기 암호해독된 인스트럭션을 난독화하고, 상기 OPS 인스트럭션의 세트의 예측가능성을 감소시키는 2-레벨 문법을 사용하여 수행됨 -와,

상기 OPS 인스트럭션의 세트를, 상기 IPS가 아닌 가상 머신(VM)에서 스텁 루틴을 통해 실행하는 단계를 포함하되,

상기 OPS 인스트럭션의 세트를 실행하는 상기 VM은 상기 메모리의 상기 제1 부분에 저장된 데이터에 액세스하고, 상기 메모리의 상기 제2 부분 내의 데이터에는 액세스하지 않으며, 상기 IPS에 의해 상기 IPS 내의 인스트럭션을 실행함으로써 달성되는 것과 동일한 결과를 실현하는

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 13

제 12 항에 있어서,

상기 2-레벨 문법은 Van Wijngaarden 문법인,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 14

제 13 항에 있어서,

상기 방법은 상기 Van Wijngaarden 문법에 기초하여, 상기 VM을 위한 메모리 레지스터 및 포인터를 랜덤하게 생성하는 단계를 더 포함하는,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 15

제 12 항에 있어서,

상기 방법은 상기 OPS에 의해, 대칭적 암호화 알고리즘에 기초하여, SDE(semantic dictionary encryption)을 생성하는 단계를 더 포함하는,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 16

제 12 항에 있어서,

상기 스텁 루틴은 상기 OPS 내에 포함되고,

상기 IPS에 의해 수행되는 암호화는 비대칭적 암호화인,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 17

제 16 항에 있어서,

상기 비대칭적 암호화는 RSA 암호화 및 ECC(Elliptic Curve Cryptography) 암호화 중 하나인,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 18

제 15 항에 있어서,

상기 대칭적 암호화 알고리즘은 RC4, RC5, SHA-1 및 MD5 알고리즘 중 하나 인,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 19

제 12 항에 있어서,

상기 방법은 상기 OPS에 의해, 상기 실행된 인스트럭션을 다른 가상 머신으로 전송하는 단계를 더 포함하며,

상기 다른 가상 머신은 가상 머신들의 계층에서 상기 VM의 바로 하위에 있는 가상 머신인,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 20

제 19 항에 있어서,

상기 가상 머신들의 계층에서 각각의 가상 머신은 고유 2-레벨 문법을 사용하여 변환된 상기 암호해독된 인스트럭션의 일부를 제공받는

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 21

제 15 항에 있어서,

상기 가상 머신들의 계층에 포함된 가상 머신들의 총 개수는 애플리케이션 실행 동안 랜덤하게 선택되는,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 22

제 21 항에 있어서,

상기 방법은 상기 가상 머신들의 계층에 포함된 랜덤한 개수의 추가 가상 머신들을 생성하는 단계를 더 포함하는,

비일시적 컴퓨터 판독 가능 저장 매체.

청구항 23

컴퓨터 보안을 향상시키기 위해 인스트럭션을 랜덤화하기 위한 장치로서,

메모리- 상기 메모리는,

내측 프로세싱 시스템(IPS)에 액세스 가능한 제1 부분과,

외측 프로세싱 시스템(OPS)에 의해 사용되고, 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능한 제2 부분을 포함함 -와,

상기 IPS 및 상기 OPS를 실행시키는 회로를 포함하되,

상기 회로는,

상기 IPS에서 인스트럭션을 암호화하는 동작과,

상기 암호화된 인스트럭션을 상기 IPS로부터 상기 OPS로 전송하는 동작과,

상기 OPS에 의해서, 수신된 상기 암호화된 인스트럭션을 암호해독하는 동작 - 상기 암호해독된 인스트럭션은 상기 IPS의 대역 밖에 있으며 상기 IPS에 의해 액세스불가능함 -과,

상기 암호해독된 인스트럭션을 OPS 인스트럭션의 세트로 변환하는 동작- 상기 변환은 상기 암호해독된 인스트럭션을 난독화하고, 상기 OPS 인스트럭션의 세트의 예측가능성을 감소시키는 컨텍스트 2-레벨 문법을 사용하여 수행됨 -과,

상기 OPS 인스트럭션의 세트를, 상기 IPS가 아닌 가상 머신(VM)에서 스텁 루틴을 통해 실행하는 동작을 수행하도록 구성되고,

상기 OPS 인스트럭션의 세트를 실행하는 상기 VM은 상기 메모리의 상기 제1 부분에 저장된 데이터에 액세스하고, 상기 메모리의 상기 제2 부분 내의 데이터에는 액세스하지 않으며, 상기 IPS에 의해 상기 IPS 내의 인스트럭션을 실행함으로써 달성되는 것과 동일한 결과를 실현하는

인스트럭션 랜덤화 장치.

청구항 24

제 23 항에 있어서,

상기 2-레벨 문법은 Van Wijngaarden 문법인,

인스트럭션 랜덤화 장치.

청구항 25

제 24 항에 있어서,

상기 회로는, 상기 Van Wijngaarden 문법에 기초하여, 상기 VM을 위한 메모리 레지스터 및 포인터를 랜덤하게

생성하는 동작을 수행하도록 더 구성된,

인스트럭션 랜덤화 장치.

청구항 26

제 23 항에 있어서,

상기 회로는, 상기 OPS에 의해, 대칭적 암호화 알고리즘에 기초하여, SDE(semantic dictionary encryption)를 생성하는 동작을 수행하도록 더 구성된,

인스트럭션 랜덤화 장치.

청구항 27

제 23 항에 있어서,

상기 스텁 루틴은 상기 OPS 내에 포함되고,

상기 IPS에 의해 수행되는 암호화는 비대칭적 암호화인,

인스트럭션 랜덤화 장치.

청구항 28

제 27 항에 있어서,

상기 비대칭적 암호화는 RSA 암호화 및 ECC(Elliptic Curve Cryptography) 암호화 중 하나인,

인스트럭션 랜덤화 장치.

청구항 29

제 26 항에 있어서,

상기 대칭적 암호화 알고리즘은 RC4, RC5, SHA-1 및 MD5 알고리즘 중 하나 인,

인스트럭션 랜덤화 장치.

청구항 30

제 23 항에 있어서,

상기 회로는 상기 OPS에 의해, 상기 실행된 인스트럭션을 다른 가상 머신으로 전송하는 동작을 수행하도록 더 구성되며,

상기 다른 가상 머신은 가상 머신들의 계층에서 상기 VM의 바로 하위에 있는 가상 머신인,

인스트럭션 랜덤화 장치.

청구항 31

제 30 항에 있어서,

상기 가상 머신들의 계층에서 각각의 가상 머신은 고유 2-레벨 문법을 사용하여 변환된 상기 암호해독된 인스트

럭션의 일부를 제공받는,

인스트럭션 랜덤화 장치.

청구항 32

제 26 항에 있어서,

상기 가상 머신들의 계층에 포함된 가상 머신들의 총 개수는 애플리케이션 실행 동안 랜덤하게 선택되는,

인스트럭션 랜덤화 장치.

청구항 33

제 32 항에 있어서,

상기 회로는, 상기 가상 머신들의 계층에 포함된 랜덤한 개수의 추가 가상 머신들을 생성하는 동작을 수행하도록 더 구성되는,

인스트럭션 랜덤화 장치.

발명의 설명

기 술 분 야

[0001] **관련 출원에 대한 상호-참조**

[0002] 본원은 2014년 12 월 16 일자로 출원된 미국 가출원 제 62/092,570호에 기초하며 이에 대한 우선권의 이익을 주장하며, 그 전체 내용은 본 명세서에 참조로서 포함된다.

[0003] 본 명세서에서 기술된 실시예들은 전반적으로 컴퓨팅 시스템의 인스트럭션 세트, 메모리 레지스터 및 포인터를 랜덤화하기 위한 프레임워크에 관한 것이다.

배 경 기 술

[0004] 본 명세서에서 제공된 배경 설명은 일반적으로 본 개시내용의 문맥을 제시하기 위한 것이다. 본 발명자들의 성과는, 이러한 성과가 본 배경 기술 부분에 설명되는 한도에서, 명시적으로 또는 암시적으로 본 발명에 대한 선행 기술로서 인정되어서는 안되며, 또한 본원의 출원 시에 선행 기술로서 인정되지 않을 수 있는 본 개시 내용의 측면들도 역시 본 발명에 대한 선행 기술로서 명시적으로 또는 암시적으로 인정되서는 안된다.

[0005] 대부분의 컴퓨팅 시스템에서 사용되는 인스트럭션 세트, 메모리 레지스터 및 포인터는 상당히 표준화되어 있다. 표준화된 머신 인스트럭션 세트는 소프트웨어와 하드웨어 간에 일관된 인터페이스를 제공하지만 장단점이 있다. 표준화된 머신 인스트럭션 세트는 하드웨어와 소프트웨어의 독립적인 개발을 가능하게 함으로써 큰 생산성 이득을 발생시킬지라도, 잘 알려진 인스트럭션 세트의 편재성(ubiquity)은 이용될 수 있는 소프트웨어 결함을 공격하도록 설계된 단일 공격으로 수천 또는 수백만 개의 시스템을 제어하는 것을 가능하게 한다. 따라서, 표준화된 인스트럭션 세트를 갖는 것은 지적 재산 침해, 컴퓨터 악용, 해킹 등을 용이하게 한다.

[0006] ASLR(Address Space Layout Randomization)은 시스템 실행가능한 인스트럭션이 메모리에 로딩되는 위치를 무작위로 지정함으로써, 버퍼-오버플로우 어택(buffer-overflow attack)을 막는 운영 체제(OS)의 메모리 보호-프로세스이다. ASLR은 라이브러리 코드 내에서 인스트럭션 어드레스를 랜덤화하는 방식이고 라이브러리 삽입 코드 공격을 저지하는 온톨로지 인코딩(ontology encoding)의 일 형태이지만, ASLR은 바이너리 코드 정적 인스트럭션 어드레스를 기반으로 하는 클라우드 애플리케이션이 직면한 문제를 해결하지 못한다.

[0007] 따라서, 당해 기술 분야에서의 상술한 문제들을 해결하고, 사이드 채널에 의해 코드가 해독되거나 공격될 수 없도록 소프트웨어 보호를 더 제공하는 기술이 요구된다.

발명의 내용

[0008] 컴퓨터 시스템은 사용된 칩 세트에 관계없이, 표준화된 인스트럭션 세트, 및보다 적은 정도로, 메모리 레지스터 및 포인터를 이용한다. 이러한 산업 표준은 소프트웨어 개발을 촉진한다. 본 개시의 일 양태는 인스트럭션 세트, 메모리 레지스터 및 포인터의 랜덤화를 가능하게 하며, 이로써 리버스 엔지니어링, 사이드-채널 인터셉트 및 분석, 및 데이터 분석의 다른 방법들을 막는 보안을 제공한다. 본 개시의 일 양태는 소프트웨어 개발, 칩 세트 설계 또는 제조 시에 변경을 요구하지 않으면서, 인스트럭션 세트, 메모리 레지스터 및 포인터의 랜덤화를 가능하게 한다. 따라서, 본 명세서에서 기술된 보안 기술은 랜덤화 및 키 인덱스 관리 모두에서 기술적 한계를 극복한다.

[0009] 본 개시의 일 양태는 소프트웨어 코드가 사이드-채널에 의해 해독되거나 공격받을 수 없도록 소프트웨어 보호를 위한 방법 및 장치를 실현한다.

[0010] 또한, 일 실시예에 따르면, 본 발명은 표준화된 인스트럭션 세트, 메모리 레지스터 및 포인터 각각의 생성 시에 무작위성의 개념을 도입함으로써, 이러한 표준화된 인스트럭션 세트, 메모리 레지스터 및 포인터와 관련된 한계 사항을 극복한다. 또한, 본 개시는 다른 것들 중에서도, FOE(Full Ontological Encyprtion), 인공 지능, 머신 학습, 반자동 에이전트 네트워크, SMIRC(Self-Modifying Instruction Randomization Code) 및 SDE(Semantic Dictionary Encryption) 분야에서 새로운 발견사항을 활용한다는 점에서, 통상적으로 코드의 본문이 암호화되게 유지되는, 통상적인 RPCDD(run-time per call dispatch decryption)에 대한 개선 기법을 제공한다.

[0011] 전술한 단락들은 전반적인 도입부로서 제공되었으며, 이하의 청구항의 범위를 제한하고자 하는 것은 아니다. 설명된 실시예들은, 다른 이점들과 함께, 첨부 도면과 관련하여 취해진 다음의 상세한 설명을 참조함으로써 가장 잘 이해될 것이다.

도면의 간단한 설명

[0012] 본 발명 및 이의 부수적인 이점들은 첨부된 도면과 함께 다음의 상세한 설명을 참조함으로써 더 잘 이해될 수 있기에, 본 발명 및 이의 부수적인 이점들은 보다 완전하면서 용이하게 인정될 것이다.

도 1은 일 실시예에 따른 프로세스들의 예시적인 계층을 도시한다.

도 2는 프로세서에 의해 수행되는 랜덤화를 묘사하는 예를 도시한다.

도 3은 본 발명에 따른 프레임워크의 일 실시예를 도시한다.

도 4는 본 발명에 따른 방법의 일 실시예를 도시한다.

도 5는 일 실시예에 따른 컴퓨팅 디바이스의 블록도를 도시한다.

발명을 실시하기 위한 구체적인 내용

[0013] 이제 도면을 참조하며, 도면에서 동일한 참조 번호는 여러 도면에 걸쳐서 동일하거나 대응하는 부분을 나타낸다. 따라서, 전술한 설명은 단지 본 발명의 예시적인 실시예를 개시하고 설명한다. 당업자가 이해할 수 있는 바와 같이, 본 발명은 본 발명의 사상 또는 본질적인 특성을 벗어나지 않고서 다른 특정 형태로 실시될 수 있다.

[0014] 따라서, 본 개시는 예시를 위한 것이며 본 발명의 범위 및 다른 청구항들의 범위를 제한하려는 것은 아니다. 본 개시는 본 명세서에 암시된 내용에 대해 쉽게 알 수 있는 변형들을 포함하며, 본 발명의 대상이 공중에게 전용되지 않도록 전술한 청구항 용어의 범위를 부분적으로 정의한다.

[0015] 다시 도 1로 돌아가면, 도 1에는 예시적인 프로세서들의 상호연동 세트가 도시되어 있다. 일 실시예에 따르면, 애플리케이션 프로그램은 본 명세서에서 서브-루틴 또는 태스크로 지칭되는 다양한 작업(job)으로 분할될 수 있다. 각 작업은 마스터 프로세서에 의해 하위 프로세서로 분배될 수 있다. 그 후, 이 하위 프로세서는 자신의 마스터 프로세서가 될 것을 선택하고, 그에 할당된 태스크를 다양한 하위 작업들로 세분하고, 각 하위 작업을 그의 하위 프로세서에 할당한다. 따라서, 이러한 방식으로, 프로그램은 다수의 상이한 태스크로 분할될 수 있고, 각각의 태스크는 하위 프로세서에 할당된다. 각 하위 프로세서는 이 하위 프로세서가 작업을 할당하는 프로세서들에 대한 마스터 프로세서라는 점을 이해해야 한다. 따라서, 프로그램의 실행은 도 1에 도시된 바와 같은 프로세서들의 계층에 의해 수행되는 것으로 간주될 수 있다.

[0016] 문법(grammar)은 구현되는 문법 유형에 의존하는, 규칙들의 세트를 기반으로 문장 구조를 기술하는데 사용된다. 예를 들어, CFG(context-free grammar)는 문자열 패턴을 생성하는데 사용되는 반복 재작성 규칙들(또는 프로덕

션(production)들)의 세트이다. CFG는 다음과 같은 구성요소들, 즉, 문법에 의해 생성된 문자열들에서 나타나는 알파벳의 문자인 종단 기호들의 세트, 비종단 기호들에 의해 생성될 수 있는 종단 기호들의 패턴에 대한 플레이스홀더인 비종단 기호들의 세트, 문자열에서(프로덕션의 좌측에 있는) 비종단 기호를(프로덕션의 우측에 있는) 다른 비종단 기호 또는 종단 기호로 대체(또는 재작성)하기 위한 규칙들인 프로덕션들의 세트, 및 문법에 의해 생성된 초기 문자열에 나타나는 특정 비종단 기호인 시작 기호를 포함한다.

[0017] 본 발명의 일 실시예에 따르면, 마스터 프로세서는 애플리케이션 프로그램을 실행하는 프로세서들의 계층구조를 생성한다. 구체적으로, 앞에서 설명한 것처럼, 마스터 프로세서는 임의의 수의 하위-프로세서들을 생성한다. 이어서, 각 하위-프로세서는 마스터 프로세서 역할을 수행하고 그 자신의 하위-프로세스들을 생성할 수 있다. 하위-프로세서들 각각에 고유 랜덤 인스트럭션 세트, 메모리 레지스터 및 포인터가 마스터 프로세서에 의해 할당된다. 일 실시예에 따르면, 각각의 하위-프로세서에 대해 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터를 결정하는데 문법들이 사용될 수 있다. 그렇게 함에 있어서, 본 발명은 이 프로그램에 대한 잠재적인 악용 위협에 대한 보안 메카니즘을 제공하는 효과적인 기능을 갖는다.

[0018] 따라서, 프로그램의 취약점을 악용하기 위해 만들어진 악의적인 소프트웨어 프로그램이나 리버스 엔지니어링 기법에 의한 시도들이 방지된다. 특히, 해당 프로그램을 실행하는데 사용되는 프로세서들의 수가 무작위적이며, 각 프로세서에 할당된 인스트럭션 세트는 (문법에 기초하여) 랜덤하게 결정되므로, 현재 및 미래의 리버스 엔지니어링 프로그램은 인스트럭션 세트에 대한 변환 테이블 또는 포인터 및 메모리 레지스터의 위치를 획득하기 위한 메커니즘을 갖지 못한다.

[0019] 또한, 보안 부가 층으로서, 애플리케이션이 턴온될 때마다, 상위 층 프로세서에 의해서 새로운 변환 테이블이 제공되고, 이 변환 테이블은, 프로세서 내의 다양한 실행중인 애플리케이션이 표준화된 인스트럭션, 메모리 레지스터 또는 포인터를 따르지 않음으로 인한 프로세서 고장을 해결하고자 시도한다. 따라서, 일 실시예에 따르면, 사용되는 프로세서의 수는 랜덤이며, 이로써, 리버스 엔지니어 또는 악성 소프트웨어는 각 프로세서를 벗어나고자 시도해야 하는 시도 횟수를 알지 못할 것이고, 해당 프로세서를 벗어나려는 시도는, 감독 프로세서로 하여금 하위-프로세서를 종료시키고 전적으로 새로운 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터를 사용하여 상기 하위-프로세서를 재부팅하게 한다.

[0020] 본 발명의 일 실시예에 따라, VW(Van Wijngaarden) 문법은 각 하위-프로세서에 할당된 랜덤 인스트럭션 세트, 메모리 레지스터 및 포인터를 생성하는 데 사용된다. VW 문법은 2 개의 CFG(context-free grammar)(즉, VW가 2-레벨 문법임)의 합성으로서 시각화될 수 있다. 제 1 CFG은 제 2 CFG에 대해 비종단 기호 역할을 하는 종단 기호들의 세트를 생성하는 데 사용된다.

[0021] 온톨로지 암호화는 데이터의 특정 요소들의 내용을 드러내지 않으면서 이러한 데이터의 특정 요소들에 대해 연산들이 수행될 수 있도록, 데이터의 내용을 모호하게 하는 암호화 시스템이다. 동형적 암호화 기술(homomorphic encryption technique)은 데이터 그룹의 구조를 유지하는 데이터 맵으로서 규정될 수 있는 동형성(homomorphsim)에 의존하는데, 달리 말하면, 이 기법은 단일 문법에 대해서 데이터를 시프트한다. 반대로, 온톨로지 암호화는 다수의 문법들에 대해서 데이터를 시프트한다는 점에서 상이하다. 구체적으로, 본 발명의 일 실시예에 따르면, 데이터는 단일 문법 내에서, 2-층 문법(VW 문법) 내에서 및/또는 N 개의 문법들 중 M 개의 문법들 내에서 시프트될 수 있다. N 개의 문법들 중 M 개의 문법들은 병렬형, 직렬형 또는 단계형(cascading)일 수 있다. 따라서, 온톨로지 암호화는 많은 방법들을 사용함으로써 발생할 수 있으며, 본 개시의 실시예들은 모든 형태들의 온톨로지 암호화를 포함하고 활용할 수 있다.

[0022] 도 1은 프로세서들(100)의 계층 구조를 도시하는, 비제한적인 예를 나타낸다. 도 1에 도시된 바와 같이, 애플리케이션 프로그램(101)은 3 개의 레벨들로 존재하는 프로세서들의 계층에서 실행된다. 이러한 일련의 계층적 프로세서들(100)은 하위-프로세서들(121, 122 및 123)을 제어하는 마스터 프로세서(110)를 포함한다. 각 프로세서에는 랜덤화된 인스트럭션 세트, 메모리 포인터 및 레지스터를 할당되며, 이러한 랜덤화된 인스트럭션 세트, 메모리 포인터 및 레지스터는, 보안, 데이터, 인스트럭션 세트, 메모리 포인터 및 레지스터를, 리버스 엔지니어링을 시도하는 공격자로부터 보호하거나, 또는 해당 프로세서를 파악하고자 하는 공격자로부터 보호하거나, 해당 프로세서로 하여금 승인되지 않은 절차를 수행하거나 공격자들이 선택한 방식으로 공격자를 돕지 못하게 한다.

[0023] 도 1에 도시된 바와 같이, 마스터 프로세서(110)는 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터를 제 1 하위-프로세서(121)에 할당하는 변환 테이블(141)(인스트럭션 세트의 변환 테이블)을 생성하도록 마스터 랜덤화 프로세스(131)(즉, 문법 생성 프로세스)를 구현한다. 유사한 방식으로, 하위-프로세서(121)는 그 자신의 마

스터 프로세서로서 동작할 수 있고, 마스터 프로세서(110)에 의해 자신에게 할당된 태스크들의 일부 또는 전부를 제 2 하위-프로세서(122)에 할당할 수 있다. 그렇게 함에 있어서, 상기 제 1 하위-프로세서(121)는 또한 제 2 하위-프로세서(122)에 대한 변환 테이블(142)을 생성하도록 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터를 (하위-랜덤화 프로세스(132)를 구현함으로써) 할당한다.

[0024] 또한, 제 2 하위-프로세서(122)는 제 3 변환 테이블을 통해 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터를 제 3 하위-프로세서(123)에 할당하도록 하위-랜덤화 프로세스(즉, VW 문법에 기초한 문법 생성 프로세스)를 구현할 수 있다. 또한, 도 1에 도시된 예시는 오직 3 개의 레벨(반복)만의 하위-프로세서 생성을 포함하지만, 본 명세서에서 기술된 기법들은 임의의 수의 계층적 프로세서들에 적용 가능하다는 점을 이해해야 한다.

[0025] 도 1에서, 각각의 하위-프로세서에는 각 하위-프로세서를 제어하는 관리 프로세서에 의해서 랜덤화된 인스트럭션 세트, 메모리 레지스터 및 포인터가 제공된다. 예를 들어, X + Y를 합산하는 대신, 하위-프로세서는 Z mod X를 A mod Y와 합산하도록 지시될 수 있다. 진정한 알고리즘을 파악하고 (하위-프로세서에 의해) 답을 받을 때에, 상기 알고리즘(변환 알고리즘)을 적용하여서 올바른 해답을 도출하는 것은 제어 프로세서(즉, 해당 하위-프로세서의 마스터 프로세서/관리 프로세서)의 임무이다. 따라서, 이러한 방식으로, 각 프로세서는 프로그램의 부분들만을 프로세싱할 것이고, 해당 프로그램의 이러한 부분들 각각은 몇몇 난독화 단계들을 거칠 것이다. 또한, 일 실시예에 따르면, 상기 프로그램의 대응하는 부분이 실행될 때마다, 이 부분은 상이한 가상 머신 내의 상이한 프로세서 상에서 실행될 수 있다. 구체적으로, 특정 프로그램의 실행 시에 사용할 프로세서들의 수를 결정할 경우의 무작위성 때문에, 이러한 특정 프로그램의 각 부분은 애플리케이션의 각 실행 반복 동안 상이한 프로세서들 상에서 실행될 수 있다.

[0026] 이러한 방식으로, 공격자는 특정 시점에 프로세싱되고 있는 정확한 데이터와, 또한 전체 프로그램이 상주하는 위치를 알기 어려울 것이다. 또한 공격자는 암호화되지 않은 데이터가 상주하는 위치(인스트럭션 세트, 메모리 포인터 또는 레지스터) 또는 해당 시점에 공격자가 알고자 하는 난독화 층(layer of obfuscation)을 알 수 없을 것이며, 또한 어느 층에서 마스터 프로세서가 마스터 변환 테이블을 생성하고 프로세서들에 대한 키 관리 및 랜덤화 방법을 마스터 프로세서의 제어 하에서 즉시 저장하는지를 알 수 없을 것이다.

[0027] 또한, 각 프로세서가 그의 임시 제어 하에서 하위-프로세서를 활성화할 때, 해당 프로세서는 제어되는 프로세스 내로 랜덤화를 도입하고 해당 프로세서에 대해 생성된 마스터 인덱스를 보유한다. 따라서, 전체 프로그램이 보다 완벽하게 보호된다. 마스터 프로세서에 의해 제어되는 하위-프로세서들 자체들은 이들보다 하위에 있는 또 다른 하위-프로세서들을 제어하고, 이러한 하위-프로세서들에 대해 마스터 프로세서 역할을 한다. 따라서, 해당 프로그램은 랜덤한 방식으로 확장 및 수축되고, 각 확장부분은 새롭게 지정된 마스터 제어 프로세서에 할당된다. 이러한 방식으로, 해당 프로그램은 상태를 유지하며, 컴퓨터 시스템 내에서의 해당 프로그램의 위치는 오직 자신에게만 알려지며, 컴퓨터 시스템 전체에 걸쳐서 프로그램이 할당되어 오직 다시 수축되고 새로운 마스터 제어 프로세서를 재구성 및 할당하고 확장 사이클을 반복할 때에, 해당 프로그램의 각 부분은 보호 스터브 상태(protected stub state)로 포함된다.

[0028] 인스트럭션 세트, 메모리 레지스터 및 포인터의 랜덤화뿐만 아니라 프로세서 수의 랜덤화를 수행할 때 발생하는 유리한 능력은, 악성 소프트웨어 또는 리버스 엔지니어 공격자가 오직 특정 순간에 실행 중인 명령이 예를 들어, 2 + 2라는 것을 결정할 수만 있을 것이라는 것이다. 이러한 명령을 보는 공격자는 숫자 2가 암호화되고 랜덤화된 숫자인지를 알 수 없으며, 이러한 가산 인스트럭션이 실제로 참(true) 인스트럭션인지를 알 수 없으며, 컴퓨터 시스템 내의 어느 프로세서가 현재 마스터 제어 하위-프로세서인지를 공격자는 알 수 없을 것이다.

[0029] 마지막으로, 공격자는 인스트럭션 세트가 특정 순간에 실행 중인 이유를 알 수 없다. 공격자가 컴퓨터 시스템 내의 각 프로세서를 열화시키고 이를 분석하려고 시도할 때, 프로세서들의 수는 무작위로 선정되기 때문에, 공격자는 컴퓨터 시스템 내에 실제로 얼마나 많은 프로세서가 있는지 알 수 없다. 또한, 일 실시예에 따르면, 유효한 프로세싱인 것처럼 보이지만 프로그램에 의해서 무시되는 노이즈를 생성하기 위해, 랜덤한 개수의 추가 프로세서들이 도입된다.

[0030] 또한, 악의적인 소프트웨어나 리버스 엔지니어는 명령 2 + 2가 마스터 프로세서에 의해서 변환될 때에, 명령 2 + 2가 무엇을 지정하는지 알지 못한다. 또한, 상기 명령이 다음 회에 실행되면, 이 명령은 5/6으로 보일 수 있는데, 그 이유는 (특정 하위-프로세서의) 마스터 프로세서가 각 실행 사이클 시에 인스트럭션 세트를 랜덤화시키기 때문이다.

[0031] 본 명세서에 기술된 실시예들은 단일 컴퓨터 시스템으로 한정되지 않고, 분산형 저장 장치들 및 프로세싱 컴퓨터 시스템들 등을 갖는 네트워킹된 컴퓨터 시스템에도 적용 가능하다는 것을 알아야 한다. 또한, 상술한 실시예들은 '클라우드 네트워크'로 지칭되는 분산형 프로세싱 및 저장 시스템에도 동일하게 적용가능하다.

[0032] 또한, 분산형 프로세싱 시스템 및 저장 시스템을 이용하든 이용하지 않든, 전술한 실시예들의 프로세스가 여전히 적용 가능한데, 즉, 랜덤한 개수의 프로세서들이 마스터 프로세서에 의해 선택되고 태스크에 필요한 프로세서들의 임시 클러스터에 할당된다. 이러한 N 개의 프로세서들 중 M 개를 선택하는 방식(즉, 랜덤한 개수의 프로세서들을 선택하고, 각 프로세서는 랜덤한 개수의 프로세싱 사이클을 가짐)은 해당 프로세스와 데이터를 더욱 난독화시키며, 이로써 공격자는 해당 프로세스 및 데이터를 완전히 분석하기 위해 얼마나 많은 프로세서를 검사해야 하는지를 알지 못할 것이다. 마스터 프로세서는 각 층-1 프로세서에 태스크를 할당할 수 있으며, 이어서, 상기 층-1 프로세서는 그 하위에 추가 층들을 생성하고 하위-마스터 프로세서 역할을 할 수 있다. 이러한 절차는 프로세서들의 클러스터가 새로운 마스터 프로세서와 새로운 층들로 자신을 재구성시키도록 지시될 때까지 계속된다.

[0033] 이제 도 2를 참조하면, 도 2는 상기 실시예들에서 설명된 랜덤화 프로세스를 나타내는 비제한적인 예를 도시한다. 도 2는 컴퓨터 객체 코드에 대응하는 바이트 코드 파일(230) 및 프로세서/가상 머신(220)에 연결된 프로그램 로딩부(program loder)(210)의 인스턴스를 도시한다. 본 발명의 랜덤화 프로세스에 의해 바이트코드 파일(230)에 대해 수행되는 프로세싱이 240로 표시된다.

[0034] 프로그램 로딩부(210)는 오프라인 저장 장치(예를 들어, 하드 디스크) 내에서 소정의 프로그램(애플리케이션이거나 또는 경우에 따라, 컴퓨터 자체의 운영 체제의 일부일 수 있음)의 위치를 파악하여 이 프로그램을 실행을 위해서 주 저장 장치(예를 들어, 퍼스널 컴퓨터 내의 랜덤 액세스 메모리) 내에 로딩한다.

[0035] 또한, 바이트코드 파일(230)은 프로그램(예를 들어, 가상 머신)에 의해 프로세싱되는 컴퓨터 객체 코드를 포함한다. 도 2에 도시된 바와 같이, 바이트코드 파일은 헤더 및 인스트럭션들에 대한 인덱스들의 시퀀스(250)를 포함한다. 본 발명의 일 실시예에 따라서, (인스트럭션에 대응하는) 각각의 인덱스에 대해, 특정 키(260)(즉, 연산코드)가 랜덤 방식으로 생성된다. 인스트럭션에 대해 생성된 연산 코드는 수행될 연산을 특정한다는 것이 주목된다. 따라서, 인스트럭션 세트에 대한 연산코드를 무작위로 생성함으로써, 본 발명은 악의적인 소프트웨어 또는 리버스 엔지니어가 애플리케이션의 정확한 실행을 추적하는 것을 방지할 수 있다. 달리 말하면, 바이트코드 파일은 (실행 시에 랜덤 방식으로) 암호화되며 이로써 상기 바이트코드 파일이 암호해독되는 때에, 런타임 프로세싱이 재구성된다.

[0036] 본 발명의 일 실시예에 따라, 각 프로세서가 랜덤화 프로세스에서 수행하는 단계들은 다음을 포함하는데, 즉, 1) 바이너리 코드가 SMIRC(Self-Modifying-Instruction-Randomization-Code) 및 SDE(Semantic Dictionary Encryption)(즉, SMIRC + SDE) 표현으로 변환되며, 2) 상기 SMIRC + SDE 표현은 VWGS(Van Wijngaarden Grammar Synthesizer)로 더 변환되며, 상기 VWGS는 라이트-백(write-back) 프로세스를 통해 생성기 스텁(generator stub)(인스트럭션들의 블록)을 생성하며, 상기 생성기 스텁은 소형 바이너리 실행가능한 파일을 생성한다. 상기 스텁은 코드의 본문이 암호화된 상태로 유지되는 동안에, RPCDD(run-time per call dispatch decryption)을 통해서 보호된다는 것이 주목된다. 이러한 프로세스는 최초의 애플리케이션을 바이너리 형태로 추가로 재생성할 수 있다. 따라서, 완전한 온톨로지 암호화는 상기 스텁으로부터 생성된 온톨로지 암호화를 사용하는 바이너리 코드의, 랜덤한 시간에서의, 실시간 랜덤화를 통해, 본 명세서에 기술된 실시예에 의해 채용된다.

[0037] **암호화 프로토콜**

[0038] 본 프로세싱 시스템(예를 들어, 가상 머신(VM))은 다음과 같은 2 개의 부분으로 구성되고, 이는 실제 CPU 하드웨어 상에서 실행되는 OPS(Outer Processing System) 호스트, 및 예를 들어, (예를 들어, qemu와 같이) 가상화된 가상 머신 컨테이너 내의 IPS(Inner Processing System) 내에서 실행되는 내부 게스트 부분이다. OPS는 사용자의 실제 정체 및 사용자의 프로세스(즉, 컴퓨터)에 대한 모든 정보를 유지 관리한다. 모든 애플리케이션 하위-루틴은 IPS로부터 실행된다. 쓰기가능한 액세스 제어는 오직 OPS가 제공하는 가상 장치와 같은 하위 프로세서로서의 IPS에 의해서만 허용된다. 내측 머신이 키에 액세스할 수 없도록 암호화가 OPS 상에서 실행된다.

[0039] 이러한 방식으로, OPS는 마스터 프로세서로서 기능하고 IPS는 하위 프로세서로서 기능한다(또한, 그 자신의 하위 프로세서들에 대해서는 마스터 프로세서로서 기능할 수 있음). OPS(예를 들어, IPSOPS 가상 시스템 이미지)는 IPS를 하우징하는 호스트 VM(예를 들어, 내부 OPS VM)의 일부이며, 이로써 OPS의 각 인스턴스는 내부에서 볼 수 있는 모든 방식으로 동일하게 나타난다(즉, 사용자 이름, MAC 어드레스 및 IP 어드레스). 외측 머신에 의해

내측 머신에 프리젠테이션되는 인터페이스는 또한 항상 동일하며, 예를 들어, 내측 머신은 항상 동일한 이름들을 사용하여 외측 머신을 참조한다.

[0040] 외측 머신은 하이퍼바이저로부터 실행되며 모든 코드는 "실제" 네트워크와 통신해야 한다. 내측 머신은 외측 머신의 할당된 암호화 포트에만 연결될 수 있다. 외측 머신은 트래픽이 내측 머신으로부터 네트워크로 직접 릴레이될 수 없도록 구성된 방화벽을 갖는다. 내측 머신이 다른 프로세서와 통신할 수 있는 유일한 방법은 외측 머신에 의해 할당된 포트를 통해서만이다.

[0041] 암호화에는 다음과 같은 두 가지 유형이 있다.

[0042] 1. 대칭적 암호화(예를 들어, RC4, RC5, SHA-1 및 MD5) 및

[0043] 2. 비대칭적 암호화(예를 들어, RSA 및 ECC).

[0044] OPS는 도 4에서 도시된 바와 같은, 암호화 메커니즘을 통해 SMIRC + SDE를 생성한다. 이 예에서, 대칭적 RC5 알고리즘이 사용되지만, 임의의 대칭적 암호화 알고리즘이 애플리케이션 서브루틴들 간의 통신을 위해서 사용 및 채택될 수 있다(즉, 마스터 애플리케이션의 구성요소로 실행되는 공동-루틴들 또는 자원 제한 노드가 존재한다). OPS와 IPS 간의 통신에는 비대칭적 암호화(예를 들어, RSA 알고리즘)가 적용된다.

[0045] 2 개의 보호 메커니즘들이 비신뢰성 시스템(즉, 에뮬레이션되는 중인 시스템)의 대역 밖에 있으면서 신뢰성 에뮬레이터들 내에서 실행된다. 시스템의 무결성과 신뢰성은 에뮬레이션된 환경 내의 샌드박스 내에 공격자를 유지하는 것에 달려 있다. OPS 와 IPS는 다음 규칙을 사용하여 상호 작용한다.

[0046] 1. OPS는 IPS 인스트럭션을 IPS 메모리로부터 OPS 메모리로 복사한다.

[0047] 2. IPS 인스트럭션은 OPS 인스트럭션 세트로 변환된다. 이러한 변환된 OPS 인스트럭션 세트가 실행되면, IPS 메모리 및 레지스터의 상태가 최초의 IPS 인스트럭션이 실행된 것처럼 보이도록 수정된다.

[0048] 3. 변환 프로세스는 IPS 인스트럭션이 IPS 메모리만을 판독하고 기록하도록 보장한다. OPS 메모리는 IPS 인스트럭션에 의해 액세스될 수 없으며 변환된 OPS 인스트럭션 세트는 OPS 메모리를 판독하거나 기록할 수 없다. 이로써, 변환된 인스트럭션 세트는 자기-판독(self-reading)되지 않을 것이다. IPS 인스트럭션 샌드박스는 제한된 메모리 공간이다.

[0049] 클라우드 프레임워크 내의 매우 많은 수의 통신 노드들로 인해 엔드-투-엔드 암호화는 비실용적인데, 그 이유는 매우 많은 수의 암호화 키를 관리, 저장 및 회수해야 하기 때문이다. 본 방법에서는, 컴퓨팅 노드의 수를 N이라고 가정하고, IPS 노드로 홉-바이-홉(hop-by-hop) 암호화를 수행하도록 OPS를 할당하며, IPS 노드들 내에서, 각 IPS 하위노드는 그의 바로 인접하는 하위노드들과 공유하는 암호화 키들을 저장한다. OPS 노드 내에 저장된 키들은 그의 이웃하는 노드들 및 IPS 노드와 공유하는 키들을 포함하며, IPS 노드는 그의 하위노드들과 공유하는 키들 및 OPS와 공유하는 하나의 키를 갖는다. 이러한 방법은 키를 전송하기 위해서 필요한 전력 소비 및 키에 대해서 요구되는 메모리를 최소화한다.

[0050] **방법 프로토콜**

[0051] 1. OPS 클론들(clones)이 IPS SMIRC + SDE 인스트럭션을 OPS 메모리 내로 암호화한다. 암호화된 IPS는 OPS 메모리 내에서 암호해독된다. 따라서, 이러한 인스트럭션은 IPS의 대역 밖에 유지되며 IPS에 의해 액세스될 수 없다.

[0052] 2. 암호해독된 IPS SMIRC + SDE 인스트럭션은 목표 IPS의 단순한 2-층 변환기(도 4 참조)로서 특정된 Van Wijngaarden 문법을 사용하여 OPS 인스트럭션 세트로 변환(또는 해석)된다.

[0053] 3. 변환된 OPS 인스트럭션 세트는 최초의 IPS 인스트럭션이 실행된 것처럼 보이도록 IPS 상태를 실행시킨다. 이러한 변환 프로세스는 IPS 인스트럭션이 암호해독된 IPS 인스트럭션을 판독하지 않도록 보장한다.

[0054] 4. 암호화된 인스트럭션은 OPS 루틴을 사용하여 OPS 메모리 내에서 암호해독된다. 에뮬레이션 샌드박스는 IPS가 OPS 메모리를 액세스할 수 없도록 보장한다(즉, 암호해독된 인스트럭션 및 암호해독 루틴은 대역 외에 있음). 암호화된 IPS 실행가능한 인스트럭션들은 암호해독 프로세스를 받지 않으며 인스트럭션들을 암호해독하는데 필요한 키들을 만나지 않으며, 암호해독은 항상 대역 외로 유지된다.

[0055] 5. OPS의 변환 메커니즘은 자신이 실행한 인스트럭션을 2 레벨 문법을 사용하여 IPS 내로 재작성할 것이며, 이는 IPS가 동일한 이미지를 2회 연속으로 가지지 않으며 이로써 공격 서피스(attack surface)이 2 배가 되지 않

을 것임을 의미한다.

[0056] **방법 : 키 - 코드해시 페어링 화이트리스트(CodeHash Pairing Whitelist)**

[0057] 암호화 키 자체는 IPS 코드가 OPS 해시를 상호적으로 가지고 있는 경우에는 IPS 코드의 해시 또는 서브루틴의 해시(예를 들어, MD5)에, 간단한 해시맵(hashmap)을 사용하여 연관되고, 이러한 인코딩은 오직 IPS와 페어링되고(paired) 서명된(signed) OPS, 및 그의 하위노드들을 갖는 IPS만을 실행하는 실행 모델을 규정하며, 따라서, 루트킷(rootkit) 및 익스플로잇(exploit)과 같은 비페어링되고 비서명된 악성 코드는 실행되지 않고, 이의 주요한 이유는 새로운 변환이 완료될 때마다 해시(예를 들어, MD5)가 재컴퓨팅되어야 하며, 이는 조작 증거 및 조작 방지를 제공하기 때문이다.

[0058] **Van Wijngaarden(VW) 문법 규칙**

[0059] VW 문법은 컨텍스트-감지형이며 따라서 당업자에게는 이러한 VW 문법은 그의 컨택스트의 정보인 그의 시맨틱을 유지하면서, (예를 들어, x86 인스트럭션 세트의) 일 형태의 코드를 다른 형태로 변환하는 규칙을 작성하는데 사용될 수 있다. VW는 장범위(long-range) 관계를 규정할 수 있으며, 이는 코드의 표현의 문장적 형식(예를 들어, x86 어셈블러)을 통해 정보가 컨텍스트적으로 유동하는 것을 의미한다. VW가 OPS 및 IPS의 맥락에서 사용될 때, 정보는 일 코드 형식을 다른 코드 형식으로 재작성하기 위해 그들의 이웃하는 하위노드들을 지켜보는 하위노드들로 유동해야 한다. 따라서, VW 문법을 코드 변환 엔진으로 사용한다는 것은, 거의 모든 규칙이 거의 모든 다른 규칙에 대해 어떠한 것을 아는 것을 요구한다는 것을 암시하며, 이는 공격 서피스를 매우 어렵게 만든다.

[0060] **제 1 레벨 변환기 문법**

[0061] 제 1 레벨 문법은 컴퓨터 코드를 생성한다. 구체적인 예로서, x86 인스트럭션 세트를 위한 제 1 레벨 문법은 기호들 S, T, U 및 V를 사용하여 다음과 같이 주어질 수 있다.

S -> mov eax, key T

T -> xor [ebx], eax U

[0062] U -> inc ebx V

[0063] 이러한 제 1 레벨 문법은 다음과 같은 실행가능한 인스트럭션 코드를 생성한다.

mov eax, key

xor [ebx], eax

[0064] inc ebx

[0065] 이러한 코드들은 비-종단 기호들의 시퀀스 S -> T -> U -> V에 의해 생성된다. 구체적으로, S와 같은 비-종단 기호는 "mov eax, key T"로 재작성될 수 있으며, 이는 다시 "mov eax, key xor[ebx], eax U" 등으로 재작성될 수 있다. 규정된 프로덕션 규칙은 다음과 같이 임의의 프로그램들에 대해 균등한 인스트럭션 시퀀스를 생성할 수 있다.

S -> mov eax, key T | push key; pop eax T

T -> xor [ebx], eax U | mov ecx, [ebx];

and ecx, eax; not ecx; or [ebx], eax;

and [ebx], ecx U

[0066] U -> inc ebx V | add ebx, 1 V

[0067] 마찬가지로, OPS가 IPS를 변경하기 위해 어떠한 기능적 효과도 없는 코드를 추가할 수 있다. 이는 하기와 같이 작은 오버헤드로 "비-작용적 시퀀스(non-operational sequences)" 인스트럭션을 생성하는 새로운 비종단 기호들을 부가함으로써 이루어질 수 있다. S -> G mov eax, key T | G push key; pop eax T

[0068] 이는 IPS 및 하위노드들에서 변화량을 증가시키기 위해 생성될 수 있는 인스트럭션 시퀀스들의 수를

증가시킨다.

[0069] **제 2 레벨 변환기 문법**

[0070] 본 특허에서, 사용된 방법은 프로덕션 과정에서, 프로덕션으로 하여금, 다음과 같은 VW 메타룰(metarule)들을 사용하여 시맨틱 및 컨텍스트 유지 코드 시퀀스들을 랜덤하게 생성하게 하는 랜덤 생성기이다(메타룰, 하이퍼룰(hyperrule) 및 프로덕션 규칙들 간을 구별하기 위해, 본 명세서에서는 통상적인 Backus-Nar 신택스 '->"을 ';'으로 변경하고, 메타룰에 대해서는 ::으로 변경하고 하이퍼룰에 대해서는 콜론(':')으로 변경한다. 규칙의 상이한 대안들을 구분하기 위해서, 본 명세서에서는 '|' 대신에 세미콜론 ';'을 사용한다).

N :: 0; 1; 2; … ; 9; 0N; … 9N;

HEXIDECIMAL :: N; a; b ; f; a HEXIDECIMAL; b HEXIDECIMAL; … ; f

HEXIDECIMAL.

ADDRESS :: 0xN.

NUMBER :: MEMORY ADDRESS; HEXIDECIMAL.

INSTRUCTION :: mov; push; pop.

MEMORY REGISTER :: eax; ebx; edx.

STACK :: esp.

MEMORY REGISTERS :: STACK; REGISTER.

REGISTER NUMBER :: REGISTER; NUMBER.

MEMORY :: [REGISTER]; [ADDRESS].

[0071] TO :: ‘,’.

[0072] 본 명세서에서는 다음 표에서 온톨로지 암호화가 바이너리 레벨에서 수행되기 때문에 칩 아키텍처와 무관하게 기능하는 방식을 전문적인 용어가 아닌 일반적인 용어를 사용하여 나타낸다.

실제 인스트럭션	최초의 바이너리	온톨로지 방식으로 암호화된 바이너리	온톨로지 방식으로 암호화된 인스트럭션
2에 2를 가산	01000001 01100100 01100100 00100000 00110010 00100000 01110100 01101111 00100000 00110010	01110011 01110101 01100010 01110100 01110010 01100001 01100011 01110100 00100000 00110011 00100000 01100110 01110010 01101111 01101101 00100000 00110011	3에서 3을 감산
eax를 ebx와 비교	01100011 01101111 01101101 01110000 01100001 01110010 01100101 00100000 01100101 01100001 01111000 00100000 01110100 01101111 00100000 01100101 01100010 01111000	01101101 01101111 01110110 01100101 00100000 00110011 00100000 01110100 01101111 00100000 00110010	3을 2로 이동

[0073]

[0074] 메타노션(metnotion) NUM은 어드레스 또는 16진수를 나타내며 INSTRUCTION은 단지 단일 인스트럭션(예를 들어, mov, push 및 pop)이 아닌 다수의 인스트럭션을 나타낸다. 다음과 같은 변환은 인스트럭션을 다음과 같은 하이퍼룰에 의해 주어진 그의 서브루틴과 균등하게 실행될 수 있는 판독가능하고 재작성가능한 문장으로 수정할 것이다.

mov REGISTERS TO REGISTER NUMBER:

move REGISTER NUMBER in REGISTERS.

push REGISTER NUMBER :

save REGISTER NUMBER.

pop REGISTERS :

[0075] restore REGISTERS.

[0076] 짧은 서브루틴의 구체적인 예로서, 코드들 "mov eax, 0"은 첫 번째 하이퍼룰로 인해서, "move 0 in eax"에 의해서 대체될 것이다. 더 높은 복잡도를 생성하기 위해, 본 명세서에서는 다음과 같이 코드들을 다른 균등한 코드들로 변환하는 하이퍼룰들을 부가할 수 있다.

move REGSTER NUMBER in MEMORY :

mov, MEMORY, TO, REGISTER NUMBER;

move REGISTER NUMBER in REGISTERS :

mov, REGISTERS, TO, REGISTER NUMBER;

[0077] save REGISTER NUMBER, restore REGISTERS.

[0078] 예를 들어, 이전에 얻은 코드들("move 0 in eax")는 "mov, eax, ',',0" 또는 "save 0, restore eax"로 수정될 수 있다. 첫 번째 대안은 생성 프로세스를 중단할 것이다. 그러나, 코드들 "mov", "eax", "','", 및 "0"(이들 중 어떠한 것도 하이퍼룰의 좌측과 일치하지 않음)은 중단될 것이다. 그러나, 다른 대안들도 생성을 계속할 것이며, 코드의 양 부분들 "save 0" 및 "restore eax"는 서로 독립적으로 대체될 것이다. 따라서 "save 0"은 "push, 0" 또는 "substract 4 from esp, move 0 in [esp]", 등에 의해서 대체될 수 있다. 위에 정의된 메타룰들(metarules)은 이들이 바이너리 변환을 통해서 수행되기 때문에, 상이한 프로세서들(예를 들어, ARM, Intel 또는 기타 하드웨어)에 대해서 동일한 인스트럭션 문법들의 테이블을 생성함으로써 보다 확장될 수 있으며, 이는 본 발명이 단지 하나의 바이너리 인스트럭션 칩 아키텍처에만 국한되지 않고 임의의 칩 아키텍처(ARM, Intel 또는 다른 하드웨어)에 의해 사용될 수 있다는 것을 의미한다. VM 문법은 가변성을 위해 무한한 인스트럭션 세트를 생성할 수 있으며, 이로써 하이퍼룰들은 무한한 수의 프로덕션 규칙들을 생성하고, 실제로, 다이버시티를 보장하면서도 런타임의 오버 헤드를 최소화시키는 것을 보장하도록 IPS 단위로 고정형 카운터를 설정함으로써 상기 생성되는 양이 제한된다. 본 명세서에서는 이러한 VW 생성된 목표 코드 시퀀스들 각각을 "스텁(stub)"이라고 칭한다.

[0079] IPS의 다양한 서브루틴들 각각은 개별적인 VW 문법에 의해서 규정될 수 있다. 문법의 시작 기호 자체가 각 프로그램을 스텁들의 구성으로 기술하는 문법의 시작 기호이다. VW 문법은 목표 머신 CPU의 이러한 코드들을 자동으로 생성하는 건설적인 방법을 생성한다. OPS는 전체 운영 프로그램을 제공하지만 OPS 자체는 IPS의 동일한 프로세스에 의해 재규정될 수 있다. 따라서, Van Wijngaarden 문법이 사용되어 IPS의 모든 부분을 생성하는 시작 OPS에서 IPS를 규정할 수 있으며, 이어서, 런타임 복제 또는 복사 프로세스를 통해 OPS 자체가 변환될 수 있고, 예를 들어, 4-레벨 VM 문법 규정을 갖는 OPS에 대해서, 페어런트(parent)인 OPS가 계속적으로 재작성하는 경우에서, 이러한 규정은 다음과 같다.

[0080] OPS : IPS-STUB01, IPS-STUB02, IPS-STUB03, IPS-STUB04

[0081] IPS-STUB01 : OPS-CHILD와 페어링되고 서명된 STUB01의 VW-Grammar

[0082] IPS-STUB02 : OPS-CHILD와 페어링되고 서명된 STUB02의 VW-Grammar

[0083] IPS-STUB03 : OPS-CHILD와 페어링되고 서명된 STUB03의 VW-Grammar

[0084] IPS-STUB04 : OPS-CHILD와 페어링되고 서명된 STUB04의 VW-Grammar

[0085] OPS-CHILD : OPS와 페어링되고 서명된 OPS-CHILD

[0086] 예를 들어, Dick Grune, How to produce all sentences from a two-level grammar Information Processing Letters Volume 19, Issue 4, 12 November 1984, Pages 181-185을 참조하고, 이 문헌은 본 명세서에서 참조로서 인용된다. 구체적으로, 각각의 스텁은 예를 들어, 프로세싱되고 감독 프로세서에게 역으로 보고되기 위해서, 클라우드 환경 또는 일부 다른 컴퓨터 시스템(도 5를 참조하여 기술됨) 내의 상이한 프로세서에 할당된다. 이러한 방식으로, 공격자가 전체 코드 구조를 모으는 것은 어렵게 되고, 코드가 실행될 때마다 코드는 상이한 인스트럭션 세트, 메모리 포인터, 레지스터 및 데이터 암호화를 가질 것이다.

[0087] 도 3은 상술한 바와 같은, 외측 프로세싱 시스템(OPS)(301) 및 내측 프로세싱 시스템(IPS)(302)에서 실행되는 내부 게스트 부분으로 구성된 가상 머신(VM)을 포함하는 프로세싱 시스템의 예를 도시한다. OPS(301)는 메타데이터 2-층 문법 변환기를 포함한다. 비대칭적 암호화가 OPS(301)와 IPS(302) 간의 통신에 적용된다. 또한, 대칭적 암호화가 애플리케이션 서브루틴들 간의 통신을 위해 사용된다.

[0088] 도 4는 컴퓨팅 시스템의 인스트럭션 세트, 메모리 레지스터 및 포인터를 랜덤화하기 위한 프로세스를 도시한다. (실제로 프로그램을 실행하기 위해) 스텁을 실행하는 실행 VM은 무작위로 생성된 코드들을 사용한다---달리 말하면, 실행 코드는 항상 한 번 나타나고 그 다음에 실행 코드는 떠나고 스택(스텁의 일부)은 다음 VM이 컴퓨팅을 계속하도록 유지된다.

[0089] 단계 1에서, 인스트럭션이 내측 프로세스에서 OP-Hash를 사용하여 암호화된다. 단계 2에서, 암호화된 인스트럭션은 외측 프로세스로 전송된다. 단계 3에서, 인스트럭션은 OPS 메모리 내로 암호해독되고 STUB를 실행하는 VM에 의해 실행된다. 이러한 암호해독된 인스트럭션은 IPS의 대역 밖에서 유지되며 IPS에 의해서 액세스될 수 없다. 암호화된 인스트럭션는 OPS 루틴을 사용하여 OPS 메모리에서 암호해독된다. 에뮬레이션 샌드박스는 IPS에 의해서 OPS 메모리가 액세스될 수 없도록 보장한다(즉, 암호해독된 인스트럭션 및 암호해독 루틴은 대역 밖에 있음). 암호화된 IPS 실행가능한 인스트럭션은 암호해독 프로세스를 가지지 않으며 인스트럭션을 암호해독하는데 필요한 비밀 키를 만나지 못하며, 암호해독은 항상 대역 밖에서 유지된다.

[0090] 단계 4에서, 암호해독된 인스트럭션이 목표 IPS의 단순한 2-층 변환기로 지정된 Van Wijngaarden 문법을 사용하여 OPS 인스트럭션 세트로 변환(또는 해석)된다. 변환된 OPS 인스트럭션 세트는 마치 최초의 IPS 인스트럭션가 실행된 것처럼 보이도록 IPS 상태를 실행한다. 이러한 변환 프로세스는 IPS 인스트럭션이 암호해독된 IPS 인스트럭션을 판독하지 않도록 보장한다.

[0091] 단계 5에서, OPS의 변환 메커니즘은 2-레벨 문법을 사용하여, 자신이 실행한 인스트럭션을 IPS 내로 재작성한다. 이러한 재작성은 IPS가 동일한 이미지를 2회 연속해서 가지지 않고 공격 서피스가 2 배가 되지 않도록 보장한다.

[0092] 도 4에 도시된 프로세스 체인 전체가 시스템이 동작하는데 필요하지 않는다는 것이 주목된다. 예를 들어, 일 실시예에서는, 시스템이 이러한 단계들 모두를 거치지만, 상이한 실시예에서는, 단계들이 바뀔 수 있거나(재순서화될 수 있거나), 모든 단계가 요구되지 않을 수 있다(일부 단계는 생략될 수 있다).

[0093] 전술한 바와 같이, 전술한 실시예들의 기능들 각각은 하나 이상의 프로세싱 회로에 의해 구현될 수 있다. 프로세싱 회로는 프로세서가 회로를 포함할 경우에, 프로그램된 프로세서(예를 들어, 도 5의 프로세서(503))를 포함한다. 프로세싱 회로는 또한 위에서 인용된 기능들을 수행하도록 구성된 ASIC(application-specific integrated circuit) 및 종래의 회로 컴포넌트들과 같은 디바이스를 포함한다. 일 실시예에 따르면, 도 5에서 기술된 바와 같은 회로는 본 명세서에서 기술된 보안 특징부들을 제공하도록 인스트럭션 세트, 메모리 레지스터 및 포인터의 랜덤화를 수행하는데 사용될 수 있다. 따라서, 랜덤화 프로세스를 구현할 시에, 본 회로는 애플리케이션을 실행하기 위한 보안 프레임워크를 제공할 수 있고, 그에 따라 컴퓨터의 전반적인 기능을 향상시킬 수 있다.

[0094] 상술한 다양한 특징부들은 컴퓨터 시스템(또는 프로그램가능 로직)과 같은 컴퓨팅 디바이스에 의해 구현될 수 있다. 회로가 회로의 프로세싱을 향상시키고 인간 또는 본 실시예들의 특징부들을 가지지 않는 범용 컴퓨터에 의해서는 불가능한 방식으로 데이터가 프로세싱되도록 하는 상술한 기능들 및 특징부들을 구현하도록 특정하게

설계되거나 프로그래밍될 수 있다. 도 5는 그러한 컴퓨터 시스템(501)을 도시한다. 도 5는 특정한 특수 목적용 머신일 수 있다. 일 실시예에서, 컴퓨터 시스템(501)은 프로세서(503)가 벡터 수축을 계산하도록 프로그램될 경우에 특정한 특수 목적용 머신이다.

[0095] 컴퓨터 시스템(501)은 자기 하드 디스크(507), 및 분리식 매체 디바이스(508)(예를 들어, 플로피 디스크 드라이브, 판독 전용 컴팩트 디스크 드라이브, 판독/기록 컴팩트 디스크 드라이브, 컴팩트 디스크 쥬크박스, 테이프 드라이브 및 분리식 광 자기 드라이브)와 같은, 정보 및 인스트럭션을 저장하기 위한 하나 이상의 저장 디바이스를 제어하기 위해, 버스(502)에 연결된 디스크 제어기(506)를 포함한다. 이러한 저장 디바이스는 적절한 디바이스 인터페이스(예를 들어, 소형 컴퓨터 시스템 인터페이스(SCSI), IDE(Integrated device electronics), E-IDE(Ehhanced-IDE), 직접 메모리 액세스(DMA), 또는 울트라-DMA)를 사용하여 컴퓨터 시스템(501)에 부가될 수 있다.

[0096] 컴퓨터 시스템(501)은 또한 특수 목적용 로직 디바이스들(예를 들어, 주문형 집적 회로(ASIC)) 또는 구성가능한 로직 디바이스들(예를 들어, 간단한 프로그램 가능 로직 디바이스(SPLD), 복잡한 프로그램 가능 로직 디바이스(CPLD), 및 필드 프로그램가능한 게이트 어레이(FPGA)를 포함할 수 있다.

[0097] 컴퓨터 시스템(501)은 정보를 컴퓨터 사용자에게 디스플레이하기 위해, 디스플레이(510)를 제어하기 위해 버스(502)에 연결된 디스플레이 제어기(509)를 또한 포함할 수 있다. 컴퓨터 시스템은 컴퓨터 사용자와 상호 작용하고 정보를 프로세서(503)에 제공하기 위한 키보드(511) 및 포인팅 디바이스(512)와 같은 입력 디바이스를 포함한다. 포인팅 디바이스(512)는 예를 들어, 방향 정보 및 명령 선택을 프로세서(503)에게 전달하고 디스플레이(510) 상에서의 커서 이동을 제어하기 위한 마우스, 트랙볼, 터치스크린 센서의 경우에 손가락, 또는 포인팅 스틱일 수 있다.

[0098] 프로세서(503)는 메인 메모리(504)와 같은 메모리에 포함된 하나 이상의 인스트럭션의 하나 이상의 시퀀스를 실행한다. 그러한 인스트럭션은 하드 디스크(507) 또는 착탈식 매체 드라이브(508)와 같은 다른 컴퓨터 판독 가능 매체로부터 메인 메모리(504) 내로 판독될 수 있다. 다중-프로세싱 장치 내의 하나 이상의 프로세서가 또한 메인 메모리(504)에 포함된 인스트럭션들의 시퀀스를 실행하기 위해 사용될 수 있다. 대안적인 실시예들에서, 하드-와이어드 회로가 소프트웨어 인스트럭션 대신에 또는 소프트웨어 인스트럭션과 조합하여 사용될 수 있다. 따라서, 실시예들은 하드웨어 회로 및 소프트웨어의 임의의 특정 조합으로 제한되지 않는다.

[0099] 전술한 바와 같이, 컴퓨터 시스템(501)은 본 명세서의 교시사항들 중 임의의 것에 따라 프로그래밍된 인스트럭션를 보유하기 위한 그리고 본 명세서에서 기술된 데이터 구조, 테이블, 기록사항 또는 다른 데이터를 포함하기 위한 적어도 하나의 컴퓨터 판독 가능 매체 또는 메모리를 포함한다. 컴퓨터 판독 가능 매체의 예는 콤팩트 디스크, 하드 디스크, 플로피 디스크, 테이프, 광-자기 디스크, PROM(EPROM, EEPROM, 플래시 EPROM), DRAM, SRAM, SDRAM 또는 임의의 다른 자기 매체, 콤팩트 디스크(CD-ROM), 또는 임의의 다른 광학 매체, 펀치 카드, 페이퍼 테이프 또는 구멍의 패턴을 갖는 다른 물리적 매체이다.

[0100] 컴퓨터 판독 가능 매체들 중 임의의 하나 또는 조합에 저장되는 본 발명은 컴퓨터 시스템(501)을 제어하고, 본 발명을 구현하기 위한 디바이스 또는 디바이스들을 구동하고, 컴퓨터 시스템(501)이 인간 사용자와 상호 작용할 수 있게 하는 소프트웨어를 포함한다. 이러한 소프트웨어에는 장치 드라이버, 운영 체제 및 애플리케이션 소프트웨어가 포함될 수 있지만 이에 국한되지는 않다. 이러한 컴퓨터 판독 가능 매체는 본 발명의 임의의 부분을 구현할 시에 수행되는 프로세싱의 전부 또는 일부(프로세싱이 분산된 경우)를 수행하기 위한 본 발명의 컴퓨터 프로그램 제품을 더 포함한다.

[0101] 본 실시예들의 컴퓨터 코드 디바이스는 스크립트, 해석 가능 프로그램, 동적 링크 라이브러리(DLL), Java 클래스 및 완전한 실행가능 프로그램을 포함하지만 이에 한정되지 않는, 임의의 해석 가능하거나 실행 가능한 코드 메커니즘일 수 있다. 또한, 본 실시예들의 프로세싱의 부분들은 보다 양호한 성능, 신뢰성 및/또는 비용을 위해 분산될 수 있다.

[0102] 본 명세서에서 사용되는 바와 같은 "컴퓨터 판독 가능 매체"라는 용어는 실행을 위해 프로세서(503)에 인스트럭션을 제공하는데 참여하는 임의의 비일시적인 매체를 지칭한다. 컴퓨터 판독 가능 매체는 비휘발성 매체 또는 휘발성 매체를 포함하지만 이에 한정되지 않는 많은 형태를 취할 수 있다. 비휘발성 매체는 예를 들어, 하드 디스크(507) 또는 분리가능한 매체 드라이브(508)와 같은 광학 디스크, 자기 디스크 및 광학-자기 디스크를 포함한다. 휘발성 매체는 메인 메모리(504)와 같은 동적 메모리를 포함한다. 반대로, 전송 매체는 버스(502)를 구성하는 와이어를 포함하는 동축 케이블, 구리 와이어 및 광섬유를 포함한다. 또한, 전송 매체는 무선파 및 적외선

데이터 통신 동안 생성되는 것과 같은, 음파 또는 광파의 형태를 취할 수도 있다.

[0103] 컴퓨터 판독 가능 매체의 다양한 형태는 실행할 하나 이상의 인스트럭션의 하나 이상의 시퀀스를 프로세서(503)에 전달하는 것에 참여할 수 있다. 예를 들어, 인스트럭션들은 초기에 원격 컴퓨터의 자기 디스크 상에 반송될 수 있다. 원격 컴퓨터는 본 발명의 전부 또는 일부를 원격으로 구현하기 위한 인스트럭션을 동적 메모리에 로딩하고 모뎀을 사용하여 전화선을 통해 인스트럭션을 전송할 수 있다. 컴퓨터 시스템(501)에 속한 모뎀은 전화선 상에서 데이터를 수신하고 버스(502) 상에 데이터를 배치할 수 있다. 버스(502)는 데이터를 메인 메모리(504)에 전달하며, 프로세서(503)는 인스트럭션을 이러한 메인 메모리(504)로부터 검색하여 인스트럭션을 실행한다. 메인 메모리(504)에 의해 수신된 인스트럭션들은 선택 사양적으로 프로세서(503)에 의한 실행 이전 또는 이후에 저장 디바이스(507 또는 508) 상에 저장될 수 있다.

[0104] 컴퓨터 시스템(501)은 또한 버스(502)에 연결된 통신 인터페이스(513)를 포함한다. 통신 인터페이스(513)는 예를 들어, 근거리 통신망(LAN)(515) 또는 인터넷과 같은 다른 통신 네트워크(516)에 연결된 네트워크 링크(514)로의 양방향 데이터 통신 접속을 제공한다. 예를 들어, 통신 인터페이스(513)는 임의의 패킷 교환 LAN에 어태치되는 네트워크 인터페이스 카드일 수 있다. 다른 예로서, 통신 인터페이스(513)는 통합 서비스 디지털 네트워크(ISDN) 카드일 수 있다. 무선 링크가 또한 구현될 수도 있다. 임의의 그러한 구현예에서, 통신 인터페이스(513)는 다양한 유형의 정보를 나타내는 디지털 데이터 스트림을 반송하는 전기적, 전자기적 또는 광학적 신호를 송신 및 수신한다.

[0105] 네트워크 링크(514)는 전형적으로 하나 이상의 네트워크를 통해 다른 데이터 디바이스에 데이터 통신을 제공한다. 예를 들어, 네트워크 링크(514)는 통신 네트워크(516)를 통해 통신 서비스를 제공하는 서비스 제공자에 의해 운영되는 장비 또는 로컬 네트워크(515)(예를 들어, LAN)를 통해 다른 컴퓨터로의 연결을 제공할 수 있다. 네트워크 링크(514) 및 통신 네트워크(516)는 예를 들어, 디지털데이터 스트림을 반송하는 전기적, 전자기적 또는 광학적 신호 및 연관된 물리 층(예를 들어, CAT 5 케이블, 동축 케이블, 광섬유 등)을 사용한다. 컴퓨터 시스템(501)으로의/으로부터의 디지털 데이터를 반송하는, 다양한 네트워크를 통한 신호 및 네트워크 링크(514)를 통한 신호 및 통신 인터페이스(513)를 통한 신호는 기저대역 신호 또는 반송파 기반 신호로 구현될 수 있다.

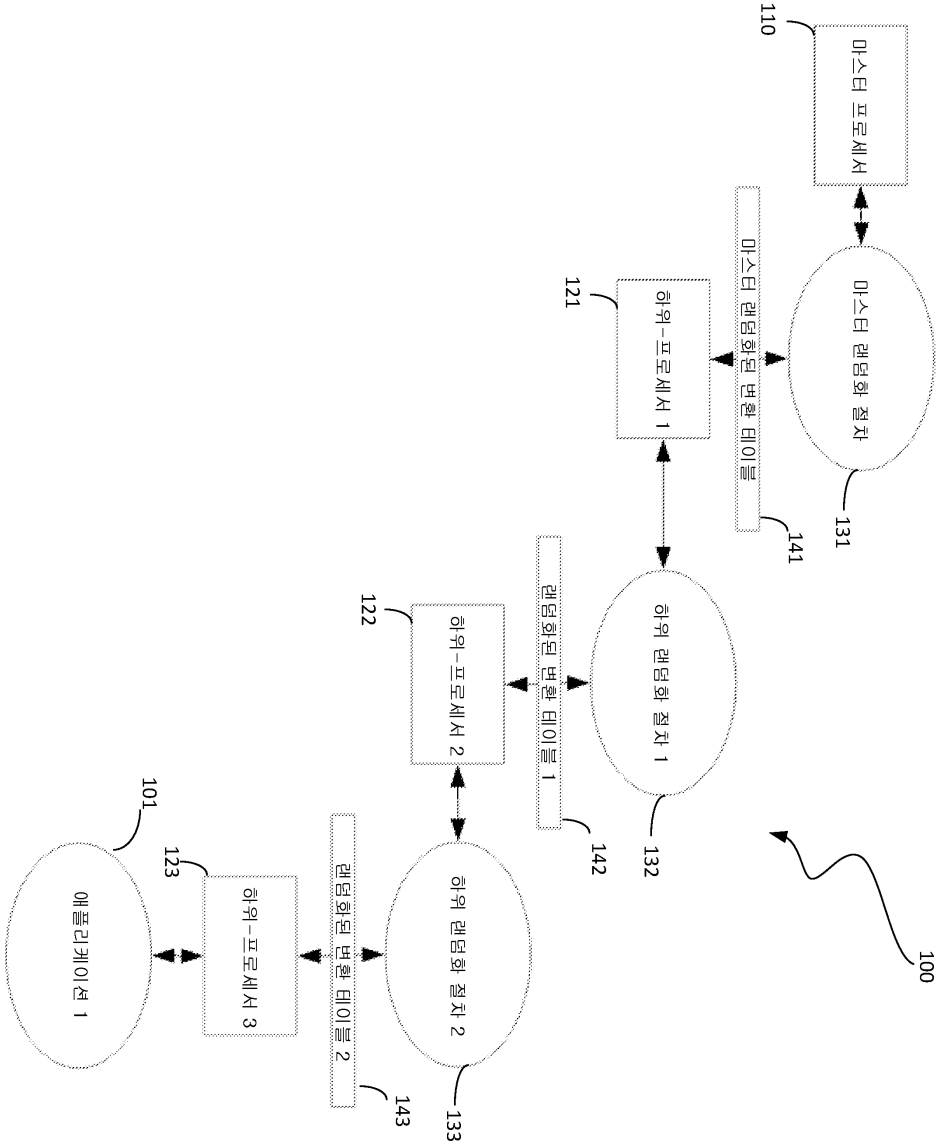
[0106] 기저대역 신호는 디지털 데이터 비트들의 스트림을 기술하는 변조되지 않은 전기적 펄스들로서 디지털 데이터를 반송하며, 여기서, 용어 "비트들"은 심볼을 의미하도록 넓게 해석되어야 하며, 각 심볼은 적어도 하나 이상의 정보 비트를 전달한다. 디지털 데이터는 도전성 매체를 통해 전파되거나 전파 매체를 통해 전자기파로서 전송되는, 진폭, 위상 및/또는 주파수 시프트 키잉된 신호와 같은, 반송파를 변조하는데에도 사용될 수 있다. 따라서, 디지털 데이터는 "유선" 통신 채널을 통해 무변조 기저대역 데이터로서 전송될 수 있고 및/또는 반송파를 변조함으로써 기저대역과 다른 미리결정된 주파수 대역 내에서 전송될 수 있다. 컴퓨터 시스템(501)은 네트워크(들)(515 및 516), 네트워크 링크(514) 및 통신 인터페이스(513)를 통해, 프로그램 코드를 포함하여 데이터를 송신 및 수신할 수 있다. 또한, 네트워크 링크(514)는 LAN(515)을 통해 PDA(personal digital assistant) 랩탑 컴퓨터 또는 셀룰러 전화와 같은 이동 디바이스(517)로의 접속을 제공할 수 있다.

[0107] 일 실시예에 따르면, 컴퓨터 보안을 증가시키기 위해 인스트럭션들을 랜덤화하기 위한 방법 및 장치가 기술된다. 본 프로세스는 외측 프로세싱 시스템(OPS) 및 내측 프로세싱 시스템(IPS)으로 구성되는 가상 머신(VM)을 포함하는 프로세싱 시스템의 내측 프로세싱 시스템(IPS)에서 인스트럭션을 암호화하는 단계와, 암호화된 인스트럭션을 외측 프로세싱 시스템(OPS)에 전송하는 단계와, 암호해독된 인스트럭션이 상기 IPS의 대역 밖에 있고 IPS에 의해서 액세스되지 않도록 상기 암호화된 인스트럭션을 암호해독하는 단계와, 상기 가상 머신 내의 스텁 루틴(stub routine)을 통해 상기 암호해독된 코드를 실행하는 단계와, 상기 IPS에 대해 간단한 2-층 변환기로 지정된 Van Wijngaarden 문법을 사용하여 상기 암호해독된 인스트럭션들을 OPS 인스트럭션 세트로 변환하는 단계, 및 상기 변환된 인스트럭션를 IPS로 전송하는 단계를 포함한다.

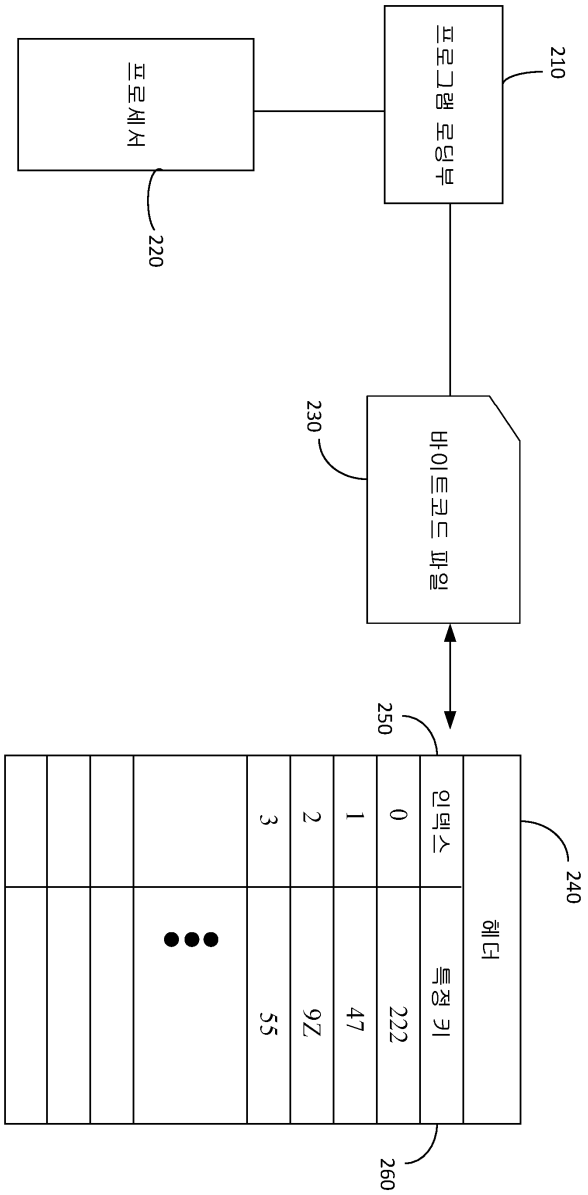
[0108] 본 발명의 양태들이 실례들로서 제안된 그들의 특정 실시예들과 관련하여 기술되었지만, 이러한 실례들에 대한 대체, 변형 및 수정이 이루어질 수 있다. 또한, 명세서 및 첨부된 청구 범위에서 사용된 바와 같이, 명사의 단수형 표현은 해당 명사의 복수형의 존재를 배제하지 않는다는 것이 주목되어야 한다.

도면

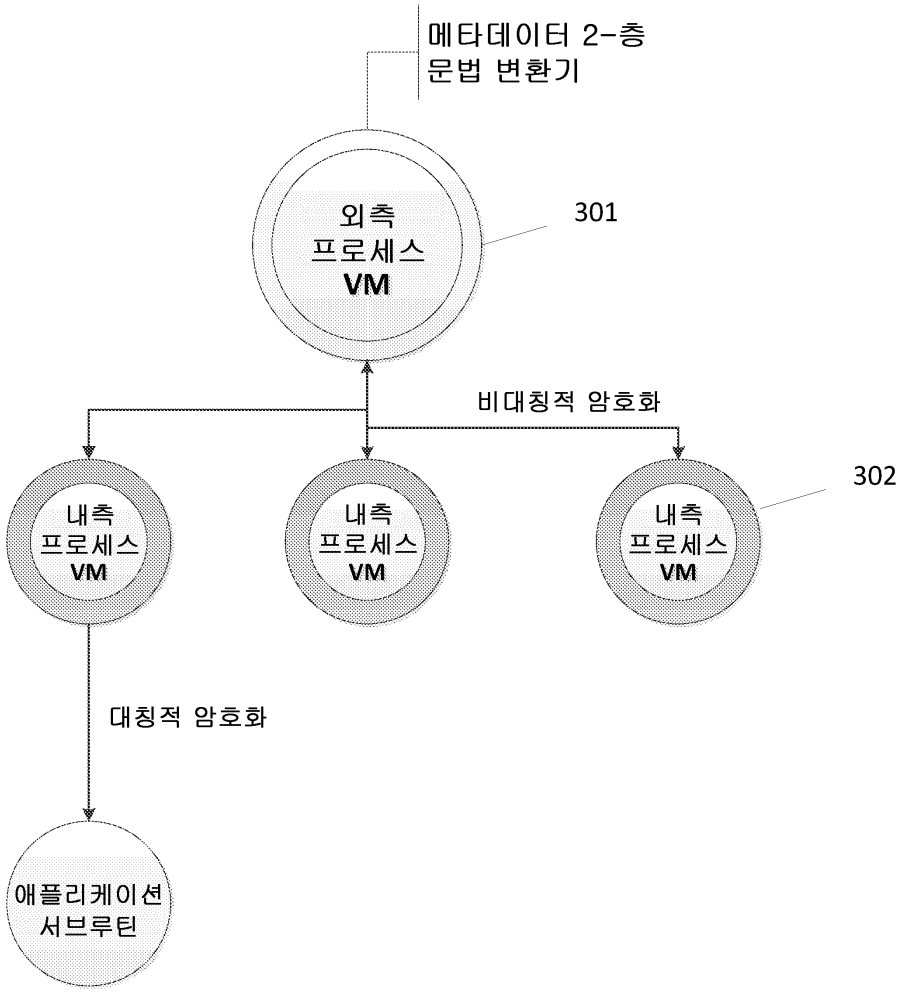
도면1


마스터 프로세서
110
마스터 랜덤화 절차
131
마스터 랜덤화된 변환 테이블
141
하위-프로세서 1
121
하위 랜덤화 절차 1
132
랜덤화된 변환 테이블 1
142
하위-프로세서 2
122
하위 랜덤화 절차 2
133
랜덤화된 변환 테이블 2
143
하위-프로세서 3
123
애플리케이션 1
101
100

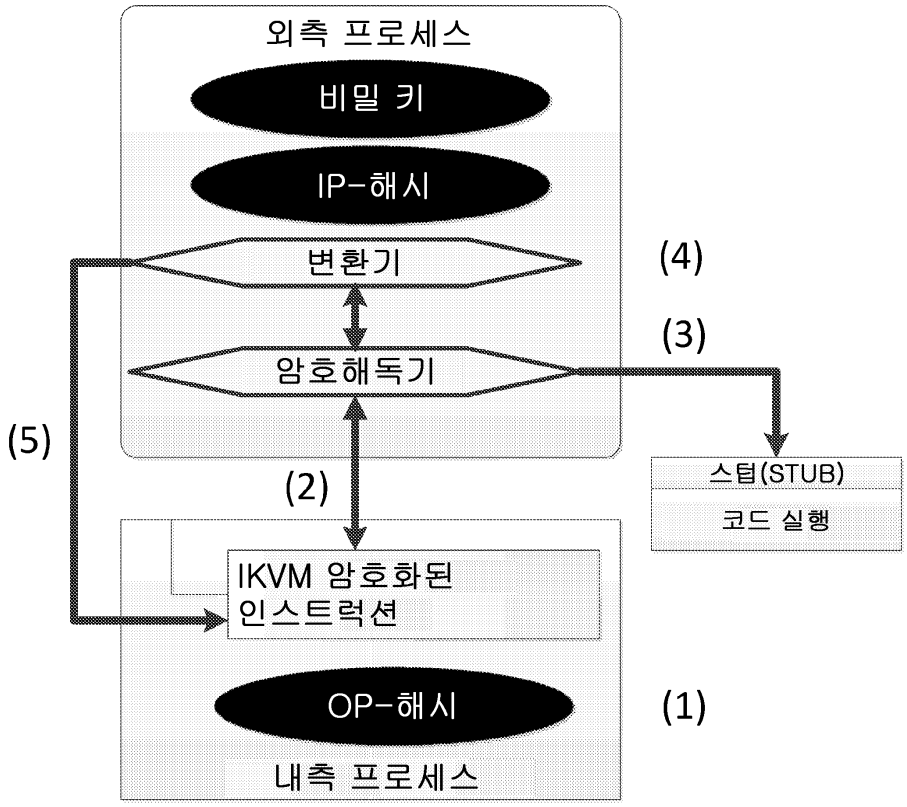
도면2


프로그램 로딩부
210
프로세서
220
바이트코드 파일
230
250
헤더
240
인덱스
특정 키
0
222
1
47
2
9Z
3
55
260

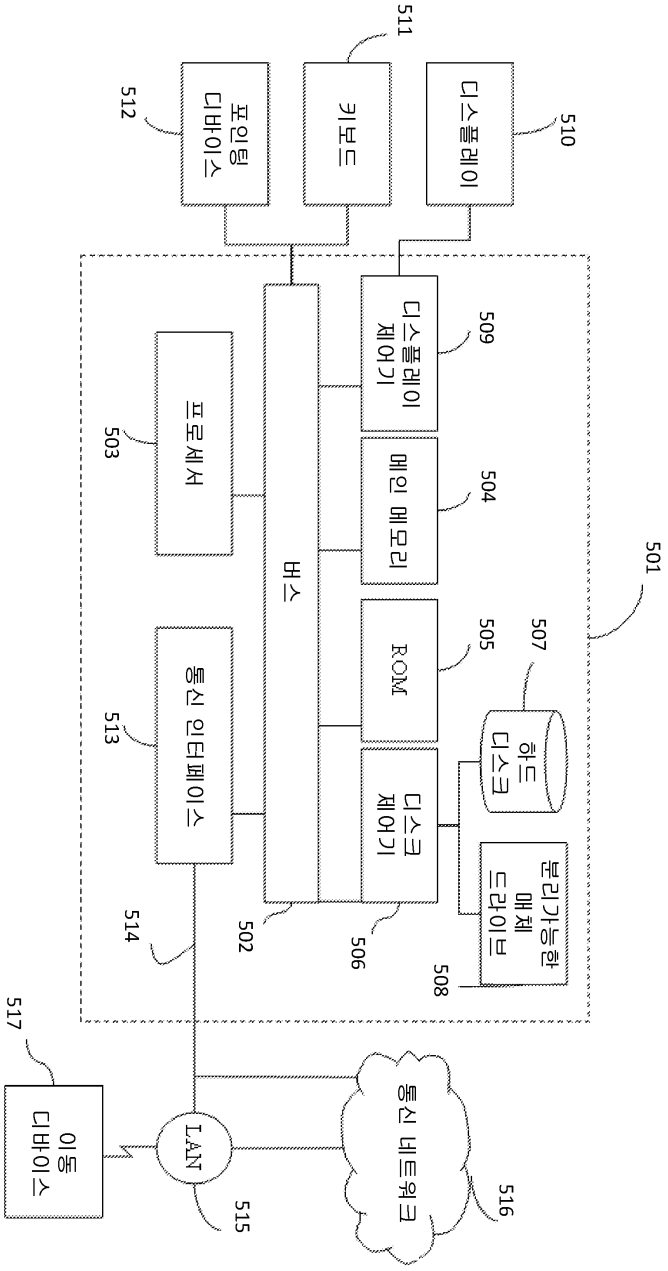
도면3


메타데이터 2-층
문법 변환기
외측
프로세스
VM
301
비대칭적 암호화
내측
프로세스
VM
내측
프로세스
VM
내측
프로세스
VM
302
대칭적 암호화
애플리케이션
서브루틴

도면4


외측 프로세스
비밀 키
IP-해시
변환기
암호해독기
(4)
(3)
(5)
(2)
스텁(STUB)
코드 실행
IKVM 암호화된
인스트럭션
OP-해시
(1)
내측 프로세스

도면5


511
포인팅 디바이스
512
키보드
디스플레이
510
디스플레이 제어기
509
메인 메모리
504
501
프로세서
503
버스
ROM
505
507
통신 인터페이스
513
하드 디스크
디스크 제어기
분리가능한 매체 드라이브
508
502
506
514
517
이동 디바이스
LAN
515
통신 네트워크
516