



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2012-0006334
(43) 공개일자 2012년01월18일

(51) Int. Cl.

G06F 9/32 (2006.01) G06F 21/22 (2006.01)
G06F 21/24 (2006.01)

(21) 출원번호 10-2010-0067009

(22) 출원일자 2010년07월12일

심사청구일자 2010년07월12일

(71) 출원인

고려대학교 산학협력단

서울 성북구 안암동5가 1

홍익대학교 산학협력단

서울특별시 마포구 와우산로 94 (상수동)

(72) 발명자

이경호

경기도 남양주시 호평로 149, 1908동 703호 (호평동, 호평마을중흥에스-클래스로하스)

표창우

서울특별시 강남구 압구정로33길 70, 56동 203호 (압구정동, 현대아파트56동)

(74) 대리인

박준석

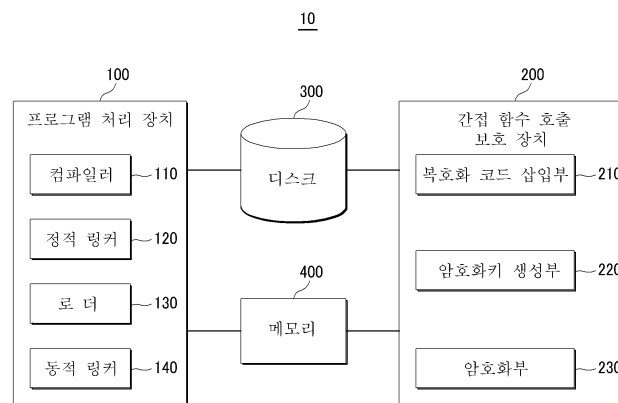
전체 청구항 수 : 총 12 항

(54) 프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 방법 및 보호 장치

(57) 요약

프로그램 카운터 인코딩(program counter encoding)을 이용하여 공유 라이브러리 함수에 대한 간접 함수 호출(indirect function call)을 보호하는 방법 및 장치가 제공된다. 간접 함수 보호 방법은 라이브러리 함수(library function)가 포함된 오브젝트 파일(object file)을 연결(linking)하는 경우에 GOT(Global Offset Table) 엔트리에 저장되어 있는 상기 라이브러리 함수의 주소에 대한 복호화 코드를 PLT(Procedure Linkage Table) 엔트리에 삽입하는 단계 및 상기 복호화 코드에 대응되는 암호화 키를 생성하고, 상기 생성된 암호화 키를 이용하여 상기 라이브러리 함수에 대응되는 GOT 엔트리를 암호화하는 단계를 포함한다.

대표도 - 도1



이 발명을 지원한 국가연구개발사업

과제고유번호 KI002090

부처명 한국산업기술평가관리원

연구관리전문기관

연구사업명 산업원천기술개발사업

연구과제명 신뢰성 컴퓨팅(Trustworthy Computing) 기반 기술개발(신뢰성 모델링 프로그램/시스템
행위 검증기술 개발)

기여율

주관기관 한국과학기술원

연구기간 2010년 03월 01일 ~ 2011년 02월 28일

특허청구의 범위

청구항 1

보호 장치가 프로그램 카운터 인코딩(program counter encoding)을 이용하여 간접 함수 호출(indirect function call)을 보호하는 방법에 있어서,

라이브러리 함수(library function)가 포함된 오브젝트 파일(object file)을 연결(linking)하는 경우에 GOT(Global Offset Table) 엔트리에 저장되어 있는 상기 라이브러리 함수의 주소에 대한 복호화 코드를 PLT(Procedure Linkage Table) 엔트리에 삽입하는 단계 및

상기 복호화 코드에 대응되는 암호화 키를 생성하고, 상기 생성된 암호화 키를 이용하여 상기 라이브러리 함수에 대응되는 GOT 엔트리를 암호화하는 단계

를 포함하는 간접 함수 호출 보호 방법.

청구항 2

제 1 항에 있어서,

상기 복호화 코드를 PLT 엔트리에 삽입하는 단계는,

상기 오브젝트 파일을 연결하여 실행 파일을 생성하는 단계,

상기 실행 파일 내에서, 상기 라이브러리 함수로 연결하는 상기 PLT 엔트리의 위치를 결정하는 단계 및

상기 복호화 코드를 상기 결정된 위치의 PLT 엔트리에 삽입하는 단계

를 포함하는 간접 함수 호출 보호 방법.

청구항 3

제 1 항에 있어서,

상기 복호화 코드는 상기 GOT 엔트리에 실행 파일의 적재 직후에 저장된 초기 주소 또는 암호화된 라이브러리 함수의 주소 중 적어도 어느 하나를 복호화하는 것인 간접 함수 호출 보호 방법.

청구항 4

제 1 항에 있어서,

상기 오브젝트 파일을 연결하여 실행 파일을 생성하는 단계를 더 포함하고,

상기 GOT 엔트리를 암호화하는 단계는,

상기 실행 파일을 메모리에 적재(loading)하는 단계,

상기 라이브러리 함수에 대응되는 상기 암호화 키를 생성하는 단계,

상기 라이브러리 함수의 주소의 GOT 엔트리의 위치를 확인하는 단계 및

상기 위치 확인된 GOT 엔트리를 상기 암호화 키를 이용하여 암호화하는 단계

를 포함하는 간접 함수 호출 보호 방법.

청구항 5

제 1 항에 있어서,

상기 GOT 엔트리를 암호화하는 단계는 상기 라이브러리 함수가 최초로 호출되는 경우에만 수행되며, 상기 라이브러리 함수가 재호출되는 경우에는 상기 GOT 엔트리를 암호화하는 단계가 수행되지 않는 것인 간접 함수 호출 보호 방법.

청구항 6

제 1 항에 있어서,

상기 GOT 엔트리를 암호화하는 단계는 실행 파일이 로딩되어 메모리에 적재되는 경우에 상기 GOT 엔트리의 초기 주소를 암호화하는 것인 간접 함수 호출 보호 방법.

청구항 7

제 1 항에 있어서,

상기 GOT 엔트리를 암호화하는 단계는 상기 라이브러리 함수의 호출과 관계없이 상기 실행 파일이 실행되는 경우에 수행되는 것인 간접 함수 호출 보호 방법.

청구항 8

제 1 항에 있어서,

상기 복호화 코드를 PLT 엔트리에 삽입하는 단계는 정적 링커(static linker)가 동작하는 경우에 수행되는 것인 간접 함수 호출 보호 방법.

청구항 9

제 1 항에 있어서,

상기 GOT 엔트리를 암호화하는 단계는 로더(loader) 및 동적 링커(dynamic linker)가 동작하는 경우에 수행되는 것인 간접 함수 호출 보호 방법.

청구항 10

프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 장치에 있어서,

GOT 엔트리에 저장되어 있는 라이브러리 함수의 주소를 복호화하는 복호화 코드를 PLT 엔트리에 삽입하는 복호화 코드 삽입부,

상기 라이브러리 함수의 주소에 대한 암호화 키를 생성하는 암호화 키 생성부 및

상기 GOT 엔트리에 기록할 상기 라이브러리 함수의 주소를 상기 암호화 키를 이용하여 암호화하는 암호화부를 포함하는 간접 함수 호출 보호 장치.

청구항 11

제 10 항에 있어서,

상기 암호화된 라이브러리 함수의 주소가 상기 대응되는 암호화 키를 사용하여 복호화되지 않고 호출되는 경우에, 상기 라이브러리 함수를 호출한 실행 파일은 실행되지 않고 종료되는 것인 간접 함수 호출 보호 장치.

청구항 12

제 11 항에 있어서,

상기 암호화된 라이브러리 함수가 비정상적인 방법으로 호출된 경우에, 상기 라이브러리 함수가 비정상적으로 호출되었음을 입출력 장치를 통해 표시하는 것인 간접 함수 호출 보호 장치.

명세서

기술 분야

[0001] 본 발명은 프로그램 카운터 인코딩을 이용하여 공유 라이브러리 함수에 대한 간접 함수 호출을 보호하는 방법 및 보호 장치에 관한 것이다.

배경 기술

[0002] 컴퓨터 기술의 급격한 발전으로 인해 현대인들의 삶도 이전보다 편하게 바뀌고 있다. 하지만, 컴퓨터 기술의 발전의 이면에는 이를 악용하여 부당한 이득을 얻으려는 시도도 증가하고 있다. 이러한 예로서, 컴퓨터 바이러스나 웜과 같이 불특정한 다수의 대상을 노리는 공격 유형이 있으며, 특정 시스템을 공격 목표로 하는 공격 유형이 있다.

[0003] 특정 시스템에 대한 공격 유형은, 예전에는 서버에 침투하여 해당 서버에 대한 제어권을 취득하고 필요한 정보를 얻는 유형이 주류였으나, 방화벽과 침입 탐지 시스템 등의 보안 기술의 향상으로 전통적인 공격 방법에도 변화가 생기게 되었다. 이러한 새로운 공격 방법으로는 백 오리피스(back orifice) 등이 있다.

[0004] 새로운 공격 방법은 종래의 백도어 침투 방식 등과는 달리, 에이전트 프로그램을 시스템에 심어놓음으로써 직접 서버에 침투하지 않고도 공격을 수행하거나 원하는 정보를 얻을 수 있다. 또한, 에이전트 프로그램에 감염된 다수의 좀비 시스템을 이용하여 특정 시스템을 공격할 수 있다.

[0005] 특정 시스템의 제어권을 취득하기 위해, 공격자는 운영체제의 취약점을 공략하게 된다. 이러한 취약점으로는 버퍼 오버플로우 취약점(buffer overflow exploit) 및 포맷 스트링 취약점(format string exploit) 등이 있다.

[0006] 포맷 스트링 취약점은 가공된 포맷 스트링을 제공하여 포맷 함수의 작동을 통제하는 것을 의미한다. 포맷 스트링 취약점을 이용하여 시스템의 통제권을 획득하고, 이를 이용하여 다른 프로세스를 죽이거나 감시 프로그램의 작동을 정지시킬 수 있다. 또한, 메모리를 읽어서 정보를 취득하거나 특정 위치의 메모리의 내용을 임의로 변경할 수도 있다.

[0007] 이러한 예로서, 포맷 스트링 취약점을 이용하여 프로그램이 사용할 어떤 함수에 대해 GOT(Global Offset Table) 엔트리를 변경함으로써 시스템의 제어권을 취득할 수 있고, 이를 이용하여 악성 코드가 저장된 주소로 점프(jump)할 수 있다.

[0008] 이러한 공격을 포함하여 아직 알려지지 않은 모든 코드 포인터 공격을 차단하기 위한 종래 기술로서, 실행 파일의 생성 단계에서 함수의 코드 포인터를 암호화하거나, 프로세스 동작 중에 실시간으로 변경 유무를 모니터링하는 방법 등이 논의되었다. 그러나 정적 상태에서 함수의 코드 포인터를 암호화하는 방법은 포인터 등가 분석(pointer equivalence analysis)이 힘들어지는 문제점이 있으며, 실시간으로 변경을 모니터링하는 기술은 모니터링을 위해 별도의 하드웨어나 특수한 아키텍처를 필요로 하거나 또는 시스템에 과도한 부하를 일으키는 문제점이 있었다.

[0009] 따라서, 이러한 함수의 코드 포인터를 조작하여 시스템의 제어권을 취득하는 공격을 효과적으로 방어하면서도 시스템에 부담을 주지 않는 보호 방법 및 보호 장치가 절실히 요구되고 있다.

발명의 내용

해결하려는 과제

[0010] 본 발명의 일 실시예는 GOT 엔트리에 저장된 라이브러리 함수의 주소를 암호화(encoding)하여 비정상적인 방법을 이용하여 GOT 엔트리를 번조한 후 해당 함수를 간접 함수 호출하는 공격을 차단함으로써 공격자가 시스템의

제어권을 취득하는 것을 방지할 수 있는 간접 함수 호출의 보호 방법 및 장치를 제공한다.

[0011] 또한, 본 발명의 일 실시예는 전체 라이브러리 함수의 주소를 암호화하지 않고도 실제로 호출되는 라이브러리 함수의 주소를 암호화하여 보호할 수 있는 간접 함수 호출의 보호 방법 및 장치를 제공한다.

[0012] 또한, 본 발명의 일 실시예는 라이브러리 함수가 실제로 호출되는 경우에 해당 주소만을 복호화(decoding)하여, 시스템에 미치는 부담을 현저히 감소시킨 간접 함수 호출의 보호 방법 및 장치를 제공한다.

과제의 해결 수단

[0013] 상술한 기술적 과제를 달성하기 위한 기술적 수단으로서, 본 발명의 제 1 측면에 따른 프로그램 카운터 인코딩을 이용한 간접 함수 호출의 보호 방법은, 라이브러리 함수(library function)가 포함된 오브젝트 파일(object file)을 연결(linking)하는 경우에 GOT 엔트리에 저장되어 있는 상기 라이브러리 함수의 주소에 대한 복호화 코드를 PLT(Procedure Linkage Table) 엔트리에 삽입하는 단계 및 상기 복호화 코드에 대응되는 암호화 키를 생성하고, 상기 생성된 암호화 키를 이용하여 상기 라이브러리 함수에 대응되는 GOT 엔트리를 암호화하는 단계를 포함한다.

[0014] 또한, 본 발명의 제 2 측면에 따른 프로그램 카운터 인코딩을 이용한 간접 함수 호출의 보호 장치는, GOT 엔트리에 저장되어 있는 라이브러리 함수의 주소를 복호화하는 복호화 코드를 PLT 엔트리에 삽입하는 복호화 코드 삽입부, 상기 라이브러리 함수의 주소에 대한 암호화 키를 생성하는 암호화 키 생성부 및 상기 GOT 엔트리에 기록할 상기 라이브러리 함수의 주소를 상기 암호화 키를 이용하여 암호화하는 암호화부를 포함한다.

발명의 효과

[0015] 전술한 본 발명의 과제 해결 수단에 의하면, GOT 엔트리에 저장된 라이브러리 함수의 주소를 암호화하여 비정상적인 방법을 이용한 간접 함수 호출의 실행을 제한함으로써, 부정확한 공격 시도를 차단할 수 있다.

[0016] 또한, 전술한 본 발명의 과제 해결 수단에 의하면, 실제로 호출되는 라이브러리 함수의 주소만을 복호화함으로써, 간접 함수 호출의 보호 방법에 의한 서버의 부하를 최소화할 수 있다.

도면의 간단한 설명

[0017] 도 1은 본 발명의 일 실시예에 따른, 프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 시스템의 세부 구성도이다.

도 2는 본 발명의 일 실시예에 따른, 프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 시스템의 처리 진행도이다.

도 3은 본 발명의 일 실시예에 따른, 복호화 코드 삽입 단계의 세부 흐름도이다.

도 4는 본 발명의 일 실시예에 따른, GOT 엔트리의 암호화 단계의 세부 흐름도이다.

도 5는 본 발명의 일 실시예에 따른, 실행 파일이 실행되는 과정에서의 메모리의 상태를 설명하기 위한 도면이다.

도 6은 본 발명의 일 실시예에 따른, 실행 파일이 실행되는 과정을 설명하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0018] 아래에서는 첨부한 도면을 참조하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 본 발명의 실시예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.

[0019] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미한다.

[0020] 한편, 명세서 전체에서 기재된 암호화/복호화 과정은 특정한 암호화/복호화 알고리즘에 한정되지 않으며, 본 발

명은 기존의 암호화/복호화를 수행하는 어떠한 알고리즘에 의해서도 구현될 수 있으며, 장치 구현될 암호화/복호화 알고리즘에 의해서도 구현될 수 있다.

- [0021] 이하, 본 명세서의 설명의 편의를 위해, 약어 및 용어에 대해 미리 정의한다.
- [0022] “소스 파일(source file)”은 프로그래머가 특정 언어의 문법으로 작성하는 프로그램 파일이다. 소스 파일은 예를 들어, 텍스트 파일의 형태로 작성되어 있다.
- [0023] “컴파일러(compiler)”는 소스 파일을 컴퓨터가 이해할 수 있는 이진 파일(binary file)로 번역하는 프로그램이다.
- [0024] “오브젝트 파일(object file)”은 소스 파일을 이진 코드(binary code) 형태의 기계어 프로그램으로 번역한 파일이다. 오브젝트 파일은 각각의 소스 파일마다 생성된다.
- [0025] “링커(linker)”는 오브젝트 파일을 연결하여 하나의 실행 파일을 만드는 프로그램이다. 링커는 다른 오브젝트 파일이 정의하는 변수 또는 함수의 주소를 재배치(relocate) 하는 과정을 수행한다. 후술할 동적 링커와 구별하기 위해, 명세서 전체에서 링커를 “정적 링커(static linker)”로 표현하기로 한다.
- [0026] “실행 파일(execution file)”은 링커에 의해 하나 이상의 오브젝트 파일을 연결한 이진 파일이며, 메모리에 적재하여 실행할 수 있는 형태의 파일이다.
- [0027] “라이브러리(library)”는 링커에 의해 하나 이상의 오브젝트 파일을 연결하여 생성된 이진 파일이며, 여러 실행 파일에서 공통으로 사용되는 함수를 담고 있다. 정적 라이브러리와 동적 라이브러리를 모두 포함한다. 본 명세서 전체에서 라이브러리와 실행 파일을 통틀어서 “실행 파일”로 표현될 수 있다.
- [0028] “라이브러리 함수(library function)”는 라이브러리에 포함되어 있는 함수를 말하며, 프로세스의 수행 도중에 호출되어 실행된다.
- [0029] “로더(loader)”는 실행 파일의 내용을 사용하여 메모리에 적재하여 프로세스 이미지를 구성하고, 적재된 프로세스 이미지를 이용하여 가상 메모리 공간을 구성한다.
- [0030] “동적 링커(dynamic linker)”는 실행 파일과 동적 연결되는 공유 라이브러리를 실행 시간에 연결하는 프로그램이다.
- [0031] “PLT”는 프로시저 연결 테이블(Procedure Linkage Table)의 약어이다. PLT는 간접 분기 명령(indirect branch instructions)을 이용해서 공유 라이브러리 내의 라이브러리 함수에게 제어권을 전달하는 스텐브 함수(stub function)들을 모아 놓은 테이블이다.
- [0032] “GOT”는 전역 오프셋 테이블(Global Offset Table)의 약어이다. GOT는 PLT 엔트리가 참조하는 함수의 코드 포인터들을 저장하고 있다.
- [0033] “재배치(relocation)”란 전역 심볼(global symbols)의 값(value) 또는 주소(address) 등을 정의(define)하는 동작이다.
- [0034] “암호화 재배치(encoded relocation)”란 포인터 변수에 할당(assign)된 값 또는 주소만을 조정하는 대신, 정적 및 동적 링커의 기능을 확장하여 재배치 수행시에 값 또는 주소 등을 인코딩하는 것이다. 암호화 재배치는 링킹 시간 또는 실행 시간에 수행될 수 있다.
- [0035] “프로그램 카운터 인코딩(program counter encoding)”이란 프로그램에서 사용되는 제어 데이터(PC-bound data)를 암호화/복호화 과정을 통해 보호하는 프로토콜(protocol)이다. 프로그램 카운터 인코딩은 제어 데이터가 변수에 저장될 때 암호화하고, 그 변수(PC-bound variables)를 참조할 때 복호화할 수 있다. 제어 데이터에는 프로시저 반환 주소(procedure return address), setjmp()/longjmp()와 같은 비로컬 점프(non-local jumps)가 정의하고 사용하는 주소, 프로시저 링크(procedure link)나 라이브러리 점프 테이블(library jump table)의 동적 함수 포인터(dynamic function pointer)등이 포함될 수 있다.
- [0036] “PC 인코딩(PC-encoding)”이란 프로그램 카운터 인코딩의 약어이다.
- [0037] 이하, 첨부된 도면을 참고하여 본 발명을 상세히 설명하기로 한다. 도 1은 본 발명의 일 실시예에 따른, 프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 시스템의 세부 구성도이고, 도 2는 본 발명의 일 실시예에 따른, 프로그램 카운터 인코딩을 이용한 간접 함수 호출 보호 시스템의 처리 진행도이다.

- [0038] 도 1에 도시된 바와 같이, 본 발명의 일 실시예에 따른 간접 함수 호출 보호 시스템(10)은 프로그램 처리 장치(100), 간접 함수 호출 보호 장치(200), 디스크(300) 및 메모리(400)를 포함한다. 프로그램 처리 장치(100)는 컴파일러(110), 정적 링커(120), 로더(130) 및 동적 링커(140)를 포함할 수 있다. 간접 함수 호출 보호 장치(200)는 복호화 코드 삽입부(210), 암호화 키 생성부(220) 및 암호화부(230)를 포함할 수 있다.
- [0039] 컴파일러(110)는 하나 이상의 소스 파일(500)을 컴퓨터가 이해할 수 있는 오브젝트 파일(600)로 번역한다. 예를 들어, 소스 파일(500)의 개수만큼의 오브젝트 파일(600)이 생성될 수 있다. 오브젝트 파일(600)에는 실행 파일에서 필요한 함수들이 포함되어 있을 수 있다.
- [0040] 정적 링커(120)는 하나 이상의 오브젝트 파일(600)을 연결하여 하나의 실행 파일(700)을 생성한다. 오브젝트 파일(600)에는 실행 파일(700)의 수행시에 필요한 함수들이 포함되어 있으며, 정적 링커(120)는 실행 파일(700)에 필요한 오브젝트 파일(600)들을 연결하여 실행 파일(700)을 생성한다. 정적 링커(120)는 후술할 복호화 코드 발생기(210)와 연동하여 PLT 엔트리에 복호화 코드를 삽입할 수 있다.
- [0041] 로더(130)는 실행 파일(700)의 수행시에 해당 실행 파일(700)을 메모리(400)에 적재하여 프로세스 이미지(800)를 구성한다.
- [0042] 동적 링커(140)는 실행 파일(700)과 동적 연결되는 공유 라이브러리를 실행 시간에 실행 파일의 프로세스 이미지(800)에 연결한다. 동적 링커(140)는 후술할 암호화 키 생성부(220) 및 암호화부(230)와 연동하여 라이브러리 함수의 주소를 암호화하여 GOT 엔트리에 기록할 수 있다. 암호화에 필요한 키도 이때 생성될 수 있다.
- [0043] 복호화 코드 삽입부(210)는 실행 파일(700)의 PLT 엔트리에 복호화 코드를 삽입한다. 복호화 코드 삽입부(210)는 PLT 엔트리에서 참조하는 GOT 엔트리를 복호화하는 코드를 PLT 엔트리에 삽입한다. 복호화 코드 삽입부(210)는 정적 링커(120)에 결합되거나 정적 링커(120)의 기능을 확장하는 형태로 구현될 수 있다. 복호화 코드의 삽입은 정적 링커(120)가 오브젝트 파일(600)을 연결하여 실행 파일(700)을 생성하는 작업과 동시에 수행될 수 있다. 암호화 키 생성부(220)는 라이브러리 함수 주소를 암호화하기 위한 암호화 키를 생성한다. 암호화 키는 후술할 암호화부(230)가 라이브러리 함수의 주소를 암호화하는 과정에서 이용된다. 암호화 키는 라이브러리별로 또는 함수별로 생성되며, 하나의 라이브러리 또는 함수에 대해 한 개의 암호화 키가 생성될 수 있다.
- [0044] 한편, 하나의 라이브러리 또는 함수에 대해 복수 개의 암호화 키를 이용하여 암호화함으로써, 더욱 강력한 방어 가 될 수 있게 할 수 있다. 예를 들어, 복수 개의 암호화 키를 이용하여 라이브러리 함수의 주소의 암호화를 복수 회 수행한 경우에, 적절한 암호화 키를 이용하지 않고 라이브러리 함수의 주소를 복호화하면, 해당 암호키로 암호화된 라이브러리 함수의 주소만 정상적으로 복호화되지 않게 된다. 이를 이용하여 관리자는 비정상적인 공격 시도가 있었음을 탐지할 수 있다.
- [0045] 암호화 키 생성부(220)는 동적 링커(140)에 결합되거나 동적 링커(140)의 기능을 확장하는 형태로 구현될 수 있다.
- [0046] 암호화부(230)는 GOT 엔트리에 저장되는 라이브러리 함수의 주소를 암호화한다. 동적 링커(140)가 실행 파일(700)과 공유 라이브러리를 연결할 때, GOT 엔트리에 라이브러리 함수의 주소를 기록한다. 암호화부(230)는 동적 링커(140)가 GOT 엔트리에 기록하는 라이브러리 함수의 주소를 생성된 암호화 키를 이용하여 암호화한다. 암호화부(230)는 동적 링커(140)에 결합되거나 동적 링커(140)의 기능을 확장하는 형태로 구현될 수 있다. 디스크(300)는 소스 파일(500), 오브젝트 파일(600) 및 생성된 실행 파일(700)이 저장되는 곳이다.
- [0047] 메모리(400)는 실행 파일(700) 및 공유 라이브러리 등이 적재되어 실행되는 곳이다. 실행 파일(700)이 실행되는 경우에, 실행 파일(700)과 공유 라이브러리는 로더(130)에 의해 메모리 상에 적재되어 프로세스 이미지(800)가 구성되며, 메모리에 적재된 각각의 프로세스 이미지(800)는 동적 링커(140)에 의해 연결될 수 있다. 메모리는 운영체제(operating system)의 커널(kernel)에 의해 가상 메모리의 형태로 사용될 수 있다. 본 명세서의 전체에 걸쳐서 가상 메모리와 물리적 메모리를 구분하지 않고 "메모리"로 표시하기로 한다.
- [0048] 도 2에 도시된 바와 같이, 프로그램 카운터 인코딩, 예를 들어, 암호화 재배치를 이용하여 간접 함수 호출을 보호하는 방법은, 라이브러리 함수가 포함된 오브젝트 파일(600)을 연결(linking)하는 경우에 GOT 엔트리에 저장되어 있는 상기 라이브러리 함수의 주소에 대한 복호화 코드를 PLT 엔트리에 삽입하는 단계 및 복호화 코드에 대응되는 암호화 키를 생성하고, 생성된 암호화 키를 이용하여 라이브러리 함수에 대응되는 GOT 엔트리를 암호화하는 단계를 포함한다. 또한, 상기 복호화 코드는 실행 파일의 적재 직후에 상기 GOT 엔트리에 저장된 초기 주소를 복호화할 수 있다.

- [0049] 이하, 첨부된 도 3, 도 4를 참조하여, 복호화 코드를 PLT 엔트리에 삽입하는 단계 및 GOT 엔트리를 암호화하는 단계를 상세히 설명하기로 한다.
- [0050] 도 3은 본 발명의 일 실시예에 따른, 복호화 코드 삽입 단계의 세부 흐름도이다.
- [0051] 도 3에 도시된 바와 같이, 복호화 코드를 삽입하는 과정은 정적 링커(120)가 동작하는 경우에 함께 수행된다. 컴파일러(110)에 의해 생성된 오브젝트 파일(600)은 정적 링커(120)에 의해 연결된다(S310). 정적 링커(120)는 전역 심벌의 주소를 조정하는 재배치 과정을 통해 오브젝트 파일(600)들을 연결하여 하나의 실행 파일(700)로 만들 수 있다.
- [0052] 오브젝트 파일(500)은 실행 파일(700)의 메인 함수를 포함하는 것일 수 있고, 메인 함수에서 호출되는 라이브러리 함수를 포함하는 것일 수도 있다.
- [0053] 오브젝트 파일(500)이 연결되는 과정에서, 복호화 코드 삽입부(210)는 공유 라이브러리 함수로 연결하는 PLT의 실행 파일(700) 내의 위치를 파악한다(S320). 복호화 코드 삽입부(210)는 GOT에 저장되어 있는 암호화된 공유 라이브러리 주소에 대한 복호화 코드를 상기 파악된 위치의 PLT 엔트리에 삽입한다(S330). 실행 파일(700)의 PLT 엔트리에 복호화 코드가 삽입되면 실행 파일(700)은 완성되며, 생성된 실행 파일(700)은 디스크(300)에 저장된다(S340).
- [0054] 도 4는 본 발명의 일 실시예에 따른, GOT 엔트리의 암호화 단계의 세부 흐름도이다.
- [0055] 도 4에 도시된 바와 같이, GOT 엔트리의 암호화 단계는 로더(130) 및 동적 링커(140)의 동작 과정에서 함께 수행된다. 실행 파일(700)이 실행되기 위해, 로더(130)는 실행 파일(700)을 메모리(400)에 적재하고, 메모리(400) 상에 프로세스 이미지(800)를 구성한다(S410). 이때, 실행 파일(700)의 실행시에 필요한 공유 라이브러리들도 로더(130)에 의해 함께 메모리(400)에 적재되어 프로세스 이미지(800)를 구성하게 된다.
- [0056] 암호화 키 생성부(220)는 실행 파일(700)의 실행에 필요한 라이브러리 함수의 주소를 암호화하기 위해 필요한 암호화 키를 생성한다(S420). 예를 들어, 암호화 키는 라이브러리 당 또는 함수 당 한 개씩 대응될 수 있고, 보안 강화를 위해 복수 개의 암호화 키를 대응시키는 것도 가능하다.
- [0057] 암호화부(230)는 암호화할 라이브러리 함수에 대응되는 GOT의 함수 주소 엔트리 위치를 파악한다(S430). GOT 엔트리의 위치가 파악되면, 암호화부(230)는 해당 GOT 엔트리의 코드 포인터를 상기 생성된 암호화 키를 이용하여 암호화하여, 프로세스 가상 메모리 형상을 완성한다(S440).
- [0058] 상기 GOT 엔트리를 암호화하는 단계는 상기 라이브러리 함수가 최초로 호출되는 경우에만 수행되고, 상기 라이브러리 함수가 재호출되는 경우에는 상기 GOT 엔트리를 암호화하는 단계가 수행되지 않을 수 있다. 다만, 본 발명의 다른 실시예에서는, 라이브러리 함수의 실제 호출 여부와 무관하게, 실행 파일이 실행되는 경우에 연관된 라이브러리 함수 전체에 대해 GOT 엔트리를 암호화하는 단계를 수행하도록 구성할 수도 있다. 한편, 실행 파일이 로딩되어 메모리에 적재된 직후에 GOT 엔트리의 초기 주소를 암호화하도록 구성될 수 있다.
- [0059] 도 5는 본 발명의 일 실시예에 따른, 실행 파일이 실행되는 과정에서의 메모리의 상태를 설명하기 위한 도면이다.
- [0060] 도 5에 도시된 바와 같이, 실행 파일의 복사본(810) 및 공유 라이브러리(820)가 메모리(400) 상에 적재되어 프로세스 이미지(800)를 구성한다. 프로세스 이미지(800)는 실행 코드, 데이터 영역, PLT(812) 및 GOT(813)를 포함할 수 있다.
- [0061] 동적 링커(140)는 실행 파일의 복사본(810)을 메모리(400)에 적재하여 프로세스 이미지(800)를 구성한다. 또한 동적 링커(140)는 관련된 공유 라이브러리(820)를 프로세스 이미지(800)에 추가 연결하고, 실행 파일의 복사본(810)과 공유 라이브러리(820)의 재배치를 수행할 수 있다. 암호화 키 생성부(220)와 암호화부(230)는 동적 링커(140)와 연동하여 암호화 대상이 되는 GOT(813)의 엔트리를 암호화하여 프로세스 이미지(800)를 완성할 수 있다.
- [0062] 실행 파일의 복사본(810)의 실행 코드에서 공유 라이브러리 함수 f를 호출하는 경우를 예를 들어 설명한다. 공유 라이브러리(820) 내의 함수 f를 호출하는 코드(811)가 실행되면 PLT(812)의 함수 f에 대응하는 엔트리로 점프한다(S510).
- [0063] 함수 f의 PLT 엔트리인 PLT_f 는 다수의 명령(instructions)으로 이루어져 있으며, GOT[f]를 참조하는 간접 분기

(indirect branch)를 포함하고 있다. 또한, PLT_i 에는 $GOT[f]$ 의 코드 포인터를 복호화하는 복호화 코드가 삽입되어 있을 수 있다.

[0064] 복호화 코드를 실행하여 $GOT[f]$ 의 코드 포인터를 복호화한다(S520). 복호화된 $GOT[f]$ 의 코드 포인터를 이용하여 간접 분기하면(S530), 함수 f 가 최초로 호출된 경우에는 분기 다음 명령인 push 명령으로 이동하게 된다(S540). push 명령이 수행되면, 동적 링커(140)에게 $GOT[f]$ 의 재배치 작업에 필요한 정보가 있는 곳을 스택을 통해 알려준다. 이 과정에서 공유 라이브러리(820) 내의 함수 f 의 주소를 확인한다(S550). 그리고, PLT_0 로 분기하여 암호화 등의 나머지 과정을 수행하게 된다(S560). 실행 파일의 실행 과정에 대해서는 도 6에서 상세히 후술하기로 한다.

[0065] 도 6은 본 발명의 일 실시예에 따른, 실행 파일이 실행되는 과정을 설명하기 위한 도면이다.

[0066] 도 6에 도시된 바와 같이, 실행 파일(700)에서 공유 라이브러리 내의 함수 f 가 호출되면(S610), PLT 에 있는 함수 f 에 대응하는 스텐브 함수인 PLT_i 로 제어권이 넘어간다(S620). PLT_i 가 참조하는 GOT 엔트리인 $GOT[f]$ 는 PLT_i 의 push 명령의 암호화된 주소로 초기화되어 있는 상태이다. 암호화된 $GOT[f]$ 를 복호화하여 주소값을 구한다(S630).

[0067] 함수 f 의 첫 호출 이후에 $GOT[f]$ 는 공유 라이브러리 함수 f 의 실제 주소로 암호화 재배치되어 저장된 상태이다. 따라서, 함수 f 가 첫 호출이 아닌 경우에는 공유 라이브러리 함수 f 의 실제 주소로 간접 분기하여 함수 f 를 수행하고, 함수 f 가 처음 호출된 경우에는 상기 복호화한 push 명령의 주소로 간접 분기한다(S640).

[0068] 함수 f 가 처음 호출된 경우(S650), 동적 링커(140)에게 $GOT[f]$ 의 암호화 재배치 작업에 필요한 정보가 있는 곳을 스택을 통해 알려 준다(S660). jump PLT_0 명령에 의해 PLT_0 의 위치로 점프하고, PLT_0 의 명령에 의해 동적 링커(140)가 호출된다(S670).

[0069] 동적 링커(140)는 $GOT[f]$ 의 값이 함수 f 의 주소를 가리키도록 암호화 재배치를 수행한다. 이를 위해, 동적 링커(140)는 공유 라이브러리 함수 f 의 가상 메모리 공간 상의 주소를 확인하고(S680), 암호화 키 생성부(220)는 함수 f 의 가상 메모리 주소를 암호화 하기 위한 암호화 키를 생성하고, 암호화부(230)는 함수 f 의 가상 메모리 주소를 암호화 키를 이용하여 암호화한다(S690). 동적 링커(140)는 암호화된 함수 f 의 주소를 $GOT[f]$ 에 저장하고(S700), 제어권을 함수 f 로 넘긴다(S710). 이후, 함수 f 의 실행이 시작된다(S720).

[0070] 이상과 같은 방법 또는 장치로써 간접 함수 호출을 보호하는 경우에, GOT 엔트리와 같은 제어 데이터를 공격자가 원하는 주소로 변경하더라도 복호화 과정을 거치면 제어 데이터는 예상치 못하는 주소 값으로 변환되며 그 결과 제어 흐름은 공격자가 의도한 주소로 이동하지 못하고, 라이브러리 함수를 호출한 실행 파일은 수행을 멈추고 종료하여 침투 시도를 봉쇄하게 되므로, 부정한 공격으로부터 시스템을 방어할 수 있다. 공격자는 제어 데이터 암호화에 사용한 키 값을 정확히 알아내어, 이를 이용하여 제어 데이터 변수에 원하는 주소를 정확히 암호화하여 저장할 수 있을 때에만 공격에 성공할 수 있다.

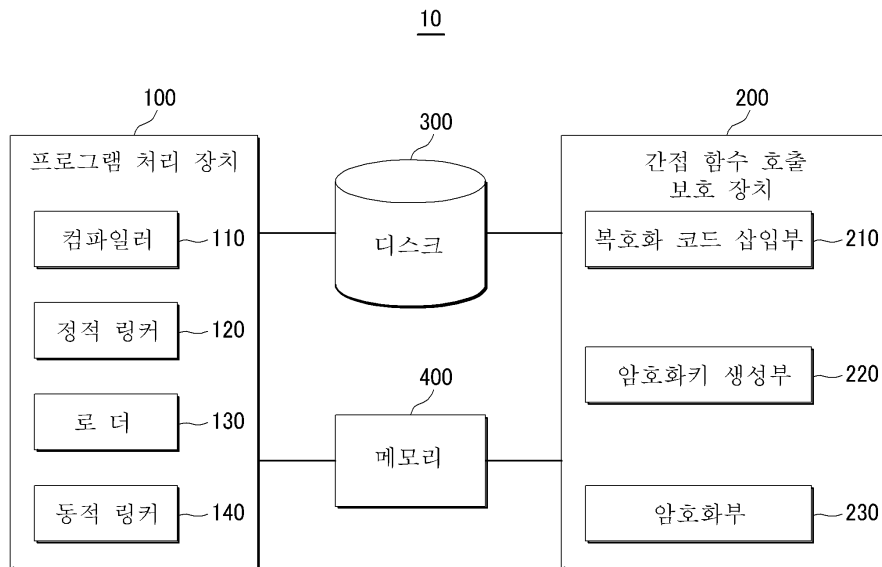
[0071] 또한, 상기 암호화된 라이브러리 함수가 비정상적인 방법으로 호출된 경우에 비정상적 호출의 발생을 입출력 장치를 통해 표시하도록 할 수도 있다.

[0072] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

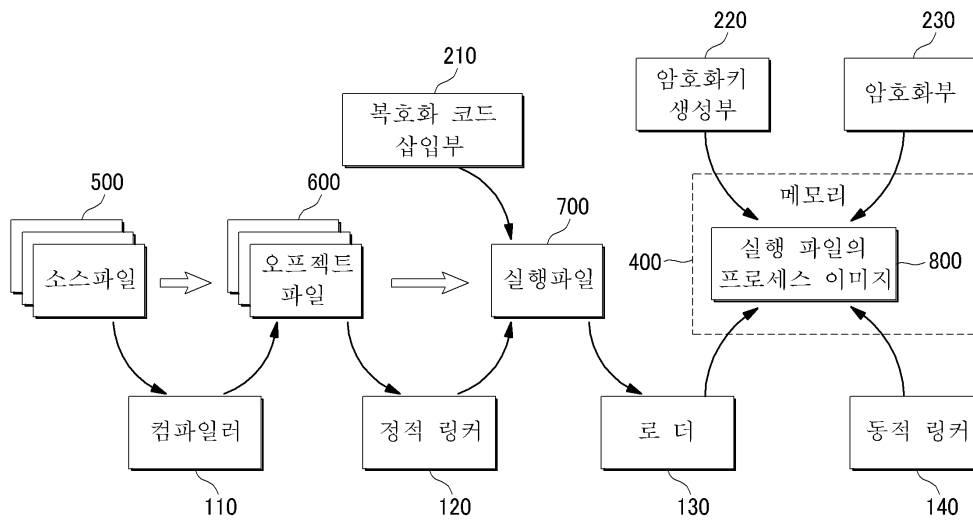
[0073] 본 발명의 범위는 상기 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

도면

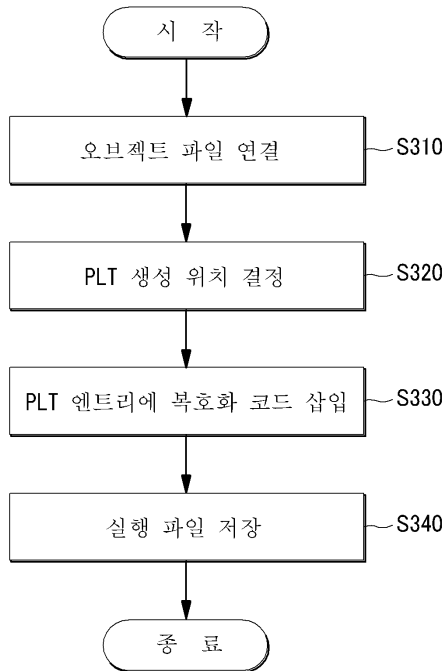
도면1



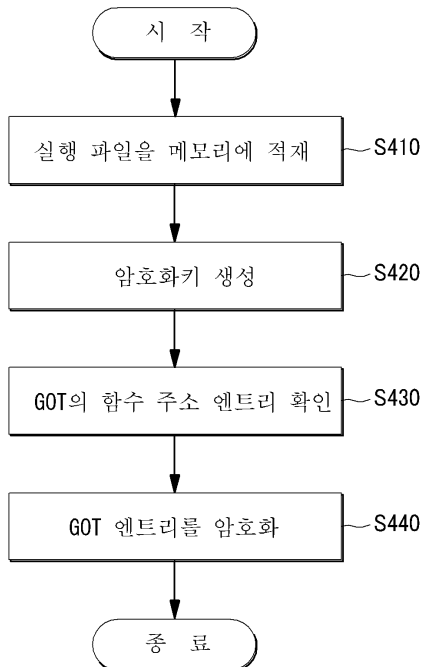
도면2



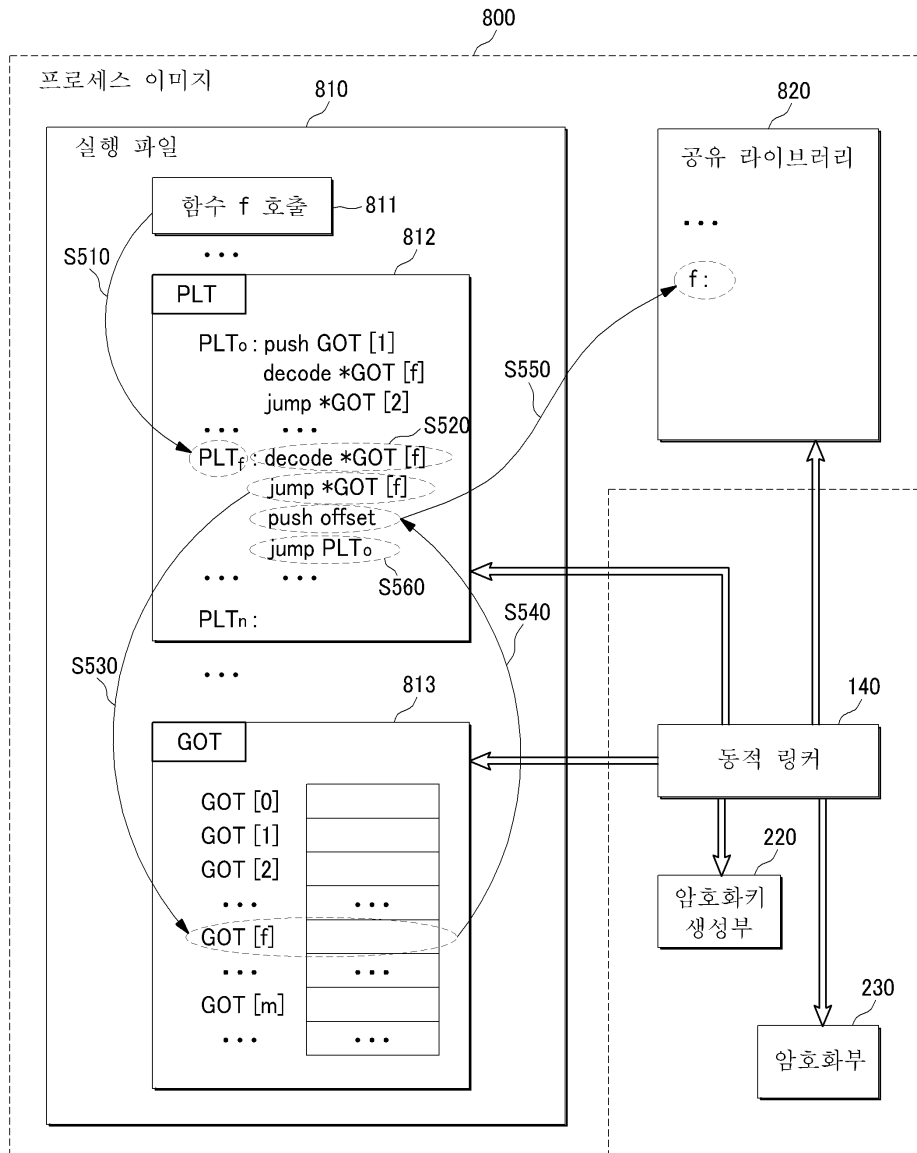
도면3



도면4



도면5



도면6

