(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0087887
(43) 공개일자 2017년07월31일

(51) 국제특허분류(Int. Cl.)
G06F 11/10 (2006.01)
(52) CPC특허분류
G06F 11/1004 (2013.01)
(21) 출원번호 10-2017-7014009
(22) 출원일자(국제) 2015년11월26일
심사청구일자 없음
(85) 번역문제출일자 2017년05월23일
(86) 국제출원번호 PCT/EP2015/077836
(87) 국제공개번호 WO 2016/083541
국제공개일자 2016년06월02일
(30) 우선권주장
14306920.1 2014년11월28일
유럽특허청(EPO)(EP)

(71) 출원인
톰슨 라이센싱
프랑스 92130 이씨레뮬리노 루 잔다르크 1-5
(72) 발명자
살몽-레가뇌 샤를
프랑스 35576 세송 세비네 아브뉘 데 샹 블랑 975 자끄 데 샹 블랑 씨에스 176 16 테크니컬러 알 앤 드 디 프랑스
칼루미 모하메드
프랑스 35576 세송 세비네 아브뉘 데 샹 블랑 975 자끄 데 샹 블랑 씨에스 176 16 테크니컬러 알 앤 드 디 프랑스
(74) 대리인
특허법인코리아나

전체 청구항 수 : 총 14 항

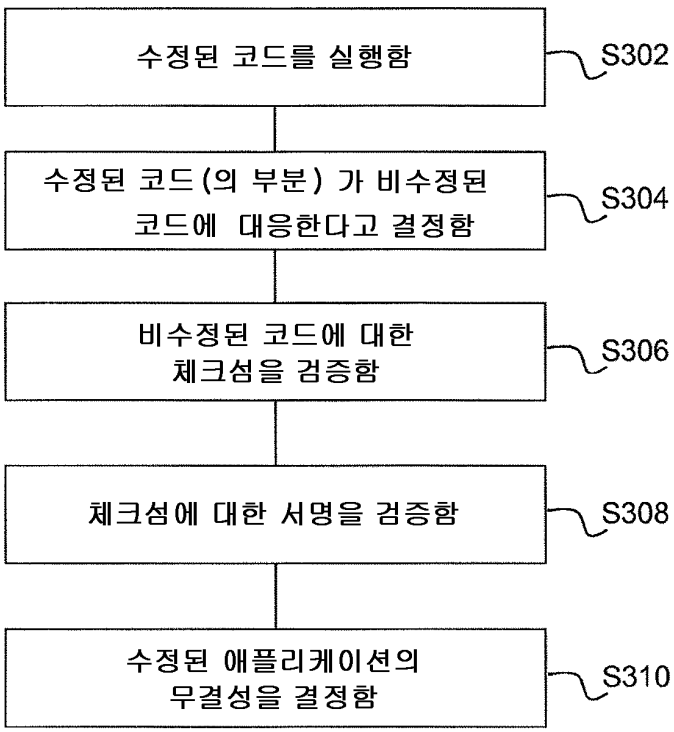
(54) 발명의 명칭 **애플리케이션 무결성의 검증을 제공하는 방법 및 디바이스**

(57) 요 약

비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 실행 (S302) 동안에, 디바이스 (110)는, 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하고 (S304), 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬과 비수정된 애플리케이션에 대한 저장된 체크섬이 매칭하는지의

(뒷면에 계속)

대 표 도 - 도3



여부를 결정하기 위해, 비수정된 애플리케이션에 대한 저장된 체크섬과 비교 (S306) 되는, 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬을 생성하고, 수정된 애플리케이션이 비수정된 애플리케이션에 대응하는 코드에 대응하는 경우에 그리고 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬이 비수정된 애플리케이션에 대한 저장된 체크섬에 매칭하는 경우에 수정된 애플리케이션의 무결성이 성공적으로 검증되었다고 결정한다 (S310). 이 솔루션은, 설치 동안의 DEX 가, 인증된 체크섬이 존재하지 않는 ELF 파일들로 컴파일된 ODEX 또는 OAT 로 최적화되기 때문에 Android OS 를 이용한 디바이스들에 대해 특히 적합하다.

명 세 서

청구범위

청구항 1

초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스 (110) 로서,

상기 디바이스는:

상기 수정된 애플리케이션, 및 상기 초기 애플리케이션의 코드에 대한 저장된 체크섬을 저장하도록 구성되는 메모리 (112); 및

프로세싱 유닛 (111) 을 포함하고,

상기 프로세싱 유닛 (111) 은 상기 수정된 애플리케이션의 실행 동안:

- 상기 초기 애플리케이션의 코드가 상기 수정된 애플리케이션의 코드에 대응한다고 결정하고;

- 생성된 체크섬을 획득하기 위해 상기 초기 애플리케이션의 코드에 대한 체크섬을 생성하고;

- 상기 초기 애플리케이션의 코드에 대한 상기 저장된 체크섬 및 상기 생성된 체크섬을, 이들 체크섬들이 매칭하는지의 여부를 결정하기 위해 비교하고; 그리고

- 상기 수정된 애플리케이션의 코드가 상기 초기 애플리케이션의 코드에 대응하는 경우에 그리고 상기 생성된 체크섬이 상기 초기 애플리케이션의 코드에 대한 상기 저장된 체크섬에 매칭하는 경우에 상기 수정된 애플리케이션의 무결성이 성공적으로 검증되었다고 결정하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 2

제 1 항에 있어서,

상기 메모리 (112) 는 비수정된 애플리케이션에 대한 상기 저장된 체크섬에 대한 서명 및 서명한 인증서를 저장하도록 구성되고, 상기 프로세싱 유닛은 상기 서명한 인증서를 이용하여 상기 서명의 유효성을 검증하고, 그리고 상기 서명이 성공적으로 검증된 경우에 상기 수정된 애플리케이션의 무결성이 또한 성공적으로 검증되었다고 결정하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 3

제 1 항에 있어서,

상기 프로세서는 비수정된 애플리케이션에 대응하는 코드를 획득하여, 상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하기 위해 상기 수정된 코드 상에서 역수정을 수행하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 4

제 1 항에 있어서,

비수정된 애플리케이션에 대응하는 코드는 상기 비수정된 애플리케이션이고, 상기 메모리는 또한 상기 비수정된 애플리케이션을 저장하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 5

제 4 항에 있어서,

상기 프로세서는 상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하기 위해, 수정된 코드와 상기 비수정된 애플리케이션에 대응하는 코드 사이의 임의의 차이들이 상기 수정 동안에 획득된 적법적 변환들 (legitimate transformations) 에 대응하는지의 여부를 결정하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 6

제 4 항에 있어서,

상기 프로세서는 제 2 수정된 코드를 획득하기 위해 상기 비수정된 애플리케이션에 대응하는 코드 상에 상기 수정을 수행하고, 상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하기 위해, 수정된 상기 코드와 상기 제 2 수정된 코드를 비교하도록 구성되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 7

제 1 항에 있어서,

비수정된 애플리케이션은 해석된 코드로서 구현되고, 상기 수정된 애플리케이션은 최적화된 해석된 코드로서 또는 네이티브 코드로서 구현되는, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 8

제 1 항에 있어서,

상기 디바이스는 스마트폰 또는 테블릿인, 초기 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스.

청구항 9

비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법으로서,

상기 방법은 상기 수정된 애플리케이션의 실행 (S302) 동안에 디바이스 (110) 에서:

- 상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하는 단계 (S304);

상기 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬을 생성하는 단계;

- 상기 비수정된 애플리케이션에 대응하는 코드에 대한 상기 체크섬과 상기 비수정된 애플리케이션에 대한 저장된 체크섬을, 이들 체크섬들이 매칭하는지의 여부를 결정하기 위해 비교하는 단계 (S306); 및

상기 수정된 애플리케이션이 상기 비수정된 애플리케이션에 대응하는 코드에 대응하는 경우에 그리고 상기 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬이 상기 비수정된 애플리케이션에 대한 상기 저장된 체크섬에 매칭하는 경우에 상기 수정된 애플리케이션의 무결성이 성공적으로 검증되었다고 결정하는 단계 (S310) 를 포함하는, 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법.

청구항 10

제 9 항에 있어서,

서명한 인증서를 이용하여 서명의 유효성을 검증하는 단계, 및 상기 서명이 성공적으로 검증된 경우에 상기 수정된 애플리케이션의 무결성이 또한 성공적으로 검증되었다고 결정하는 단계를 더 포함하는, 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법.

청구항 11

제 9 항에 있어서,

상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하는 단계는, 상기 비수정된 애플리케이션에 대응하는 코드를 획득하기 위해 상기 수정된 코드 상에 역 수정을 수행하는 단계

를 포함하는, 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법.

청구항 12

제 9 항에 있어서,

상기 비수정된 애플리케이션에 대응하는 코드가 또한 상기 수정된 애플리케이션에 대응한다고 결정하는 단계는, 상기 수정된 코드와, 상기 비수정된 애플리케이션에 대응하는 코드 사이의 임의의 차이들이 상기 수정 동안에 획득된 적법적 변환들에 대응하는지의 여부를 결정하는 단계를 포함하는, 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법.

청구항 13

제 9 항에 있어서,

상기 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하는 단계는, 제 2 수정된 코드를 획득하기 위해 상기 비수정된 애플리케이션에 대응하는 코드 상에 상기 수정을 수행하는 단계, 및 수정된 상기 코드와 상기 제 2 수정된 코드를 비교하는 단계를 포함하는, 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하는 방법.

청구항 14

프로세서 (110) 에 의해 실행될 때 상기 프로세서로 하여금 제 9 항에 기재된 방법을 수행하게 하는 명령들을 포함하는 컴퓨터 실행가능 프로그램 (220).

발명의 설명

기 술 분 야

[0001] 본 개시는 일반적으로 컴퓨터 시스템들에 관한 것이고, 보다 구체적으로, 이러한 시스템들에서의 소프트웨어 코드의 무결성에 관한 것이다.

배 경 기 술

[0002] 본 섹션은 아래 설명되고/되거나 청구된 본 개시의 여러 양태들에 관련될 수도 있는 당해 기술의 여러 양태들을 독자에게 도입하도록 의도된다. 본 설명은 본 개시의 여러 양태들의 보다 나은 이해를 용이하게 하기 위해 백그라운드 정보를 독자에게 제공하는데 있어 도움이 될 것으로 믿는다. 따라서, 이들 명세서들은 이러한 견지에서 읽혀져야 하며 종래 기술의 인정으로서 간주되지 않음을 이해하여야 한다.

[0003] 여러 이유로, 프로세싱 디바이스들이 도용되지 않은 소프트웨어를 실행하는 것을 보장하는 것이 종종 바람직하다. 이를 위해, 상이한 기법들이 도용 공격들에 대항하여 소프트웨어 이미지를 보호하는데 이용될 수 있다.

대부분의 일반 기법은 서명 또는 체크섬을 코드 세그먼트들 상에서 연산한 다음, 나중 단계에서 서명 또는 체크섬을 검증하는 것이다. 암호화 서명의 생성이 개인 키 및 대응하는 공개 키에 대한 서명의 검증을 요구하는 반면, 체크섬들은 일반적으로 어떠한 시크릿 없이 연산되어 검증된다.

[0004] 체크섬 기반 보호의 일 예는 윈도우즈 오퍼레이팅 시스템에 이용되는 포터블 실행가능 (Portable Executable; PE) 포맷에 대한 CRC32 이다. PE 헤더는 대응하는 코드 섹션의 체크섬을 제공하는 CRC32 필드를 포함한다.

보호를 성공적으로 우회시키기 위해, 공격자는 먼저 코드 섹션을 수정한 다음, 수정된 코드 섹션 상에서 연산된 새로운 값으로 오리지널 체크섬을 대체시킨다. 이 유형의 공격은 공격자가 수정된 코드 섹션들의 체크섬들을 업데이트하기 위해 어떠한 시크릿도 필요로 하지 않기 때문에 가능해진다.

[0005] 체크섬들의 취약성을 고려하여, 암호화 서명들은 선호되는 솔루션이다. 서명의 생성은 코드 배포 전에 수행되고, 개인 (및 이에 따라 시크릿) 키를 이용한다. 연관된 공개 키는 코드에 첨부되고 이후에, 코드의 설치 시 또는 런타임 시에 코드 무결성을 체크하는데 이용된다. 공격자는 여전히 코드를 수정할 수 있지만, 코드에 대한 정확한 서명이 공개 키 없이 생성될 수 없기 때문에 공격은 실패한다.

[0006] 네이티브 코드에서 전달 및 실행되는 애플리케이션들의 무결성을 체크하기 위한 많은 솔루션들, 이를 테면, Arxan (GuardIT™), Metaforic (Metafortress™) 등에 의해 제공되는 솔루션들이 존재한다. 네이티브 코드

는 프로세서에 의해 직접 실행가능한 어셈블러 명령들의 세트이다. 명령들의 세트는 설치 후에 변하지 않으며, 이는 프로그램 무결성 값이 설치 전과 설치 후에 동일하다는 것 (즉, 시간에 따라 일정하게 유지되는 것)을 의미한다. 이 경우에, 서명은 사전에 생성되어 애플리케이션 패키지로 전달될 수 있다.

[0007] 한편, 해석된 코드 - 이를 테면, Java, Android DEX 코드로 기록된 코드 - 의 형태로 분배된 애플리케이션들은 애플리케이션이 실행되기 전에 인터프리터를 통과해야 하는 중간 명령들을 포함한다. 네이티브 코드와 달리, 해석된 코드는 최적화 목적을 위하여 설치시간 후에 수정될 수 있다. 코드 변경은 일반적으로 타겟 플랫폼 상에 매우 의존적이며 따라서 반드시 예측가능한 것은 아니다. 코드가 수정되면, 해석된 코드 상에서 생성된 서명은 런타임에서 동작으로 코드 무결성 및 인증성을 체크하는데 이용될 수 없다.

[0008] 이전에 언급된 Android 오퍼레이팅 시스템 상에 애플리케이션 소프트웨어를 분배 및 설치하기 위해, APK - Android Application PacKage - 이라 불리는 파일 포맷이 이용된다. APK 파일을 만들기 위해, Android 에 대한 프로그램은 먼저 중간 언어로 컴파일된 다음, 그 부분들이 압축된 아카이브 파일 (ZIP 포맷) 으로 패키징된다. 아카이브 파일은 단일의 DEX (Dalvik EXecutable code) 파일, 여러 리소스들 (예를 들어, 이미지 파일들), 및 APK 파일의 매니페스트에 전체 프로그램 코드를 포함한다. 아카이브 파일은 2 개의 추가 파일들: CERT.SF 및 CERT.RSA 를 포함한다. CERT.SF 는 모든 다른 아카이브 파일들의 암호 해시들을 포함하고; CERT.RSA 는 서명 검증을 위해 이용되는 공개 키를 포함한다. CERT.SF 만이 RSA 개인 키로 서명된다. CERT.SF 에 대한 RSA 서명은 설치 동안에 APK 파일의 전체 컨텐츠의 검증을 실행한다. 실제로, CERT.SF 파일에 언급된 모든 파일들은 CERT.SF 가 그들의 해시들을 포함하기 때문에 간접적으로 서명된다. 파일 다이제스트가 CERT.SF 파일에서의 해시와 매칭하지 않음을 소프트웨어가 검출하기 때문에, 설치전에 임의의 파일을 변경하는 것은 에러를 야기한다. 대안으로서, (이미 기술된 체크섬 기반 검증에 대항하는 공격에서와 같이) CERT.SF 파일 내에서 암호 해시 값을 수정하는 것은 서명 검증 동안에 에러를 초래할 것이다.

[0009] DEX 파일 헤더는 또한 DEX 파일의 컨텐츠에 대한 글로벌 체크섬을 포함한다. 애플리케이션의 첫번째 실행시, Android 시스템은 단지 실행전 시간에만 DEX 해석된 바이트 코드를 ODEX (Optimized DEX) 로 불리는 최적화된 머신 명령들 시퀀스로 수정하는 옵티마이저를 이용한다. 옵티마이저는 또한 체크섬을 업데이트한다. ODEX 파일은 그 후, 장래의 이용을 위해 Android 파일 시스템 내에서의 특정 레포지토리에 저장된다. ODEX 파일은 그 후, 애플리케이션 소프트웨어에 대하여 레퍼런스로 되며, 이것이 존재할 때 오리지널 DEX 파일이 더 이상 이용되지 않는다.

[0010] 런타임에서, 시스템은 ODEX 체크섬을 이용하여 애플리케이션에 대한 무결성을 검증할 수도 있다. 그러나, 체크섬 검증은 실행 성능에 대해 무시할 수 없는 영향을 주기 때문에, 이 옵션은 Android 오퍼레이팅 시스템에서 디폴트로 설정되지 않고, 그리고 ODEX 를 실행하는데 이용되는 Dalvik 머신은 항상 ODEX 체크섬들을 체크하는 것은 아니다.

[0011] Android 버전 5.0 이상은 Dalvik 머신을 대체하는 Android Runtime (ART) 을 도입하였다. 애플리케이션은 여전히 DEX 코드로 배치되지만, 설치시에, DEX 코드는 어헤드-오브-타임 컴파일화 (ahead-of-time compilation; (AOT) 피처를 이용하여 네이티브 코드로 컴파일된다. DEX 파일 상에서의 AOT 컴파일화는 바이너리 실행가능 링크가능 포맷 (Executable Linkable Format; ELF) 파일의 결과를 가져온다. 이 후, 애플리케이션의 DEX 코드는 한번 컴파일된 다음, ELF 코드는 애플리케이션이 실행될 때마다 나중에 론칭된다. ART 가 네이티브 코드 (ELF 코드) 를 직접 실행하기 때문에, 이는 애플리케이션들의 더 고속의 실행을 가져오고 전체적인 전력 소모를 개선시킨다.

[0012] 따라서, 이는 Android 시스템에서, APK 서명이 설치시에만 검증된다고 이해될 수 있다. 또한, APK 는 중앙 인증기관에 의해서만 서명되었을 때에도, 사용자가 신뢰되지 않은 소스들로부터 오는 애플리케이션의 설치를 허용하면, Android 디바이스 상에 설치될 수 있다. 그 후, 애플리케이션 개발자들은 임의의 신뢰된 인증기관에 링크되지 않는 그들 자체의 자체-서명된 인증서들을 이용한다. 그 경우, 도용된 애플리케이션들은 그 소유자가 모르는 사이에, Android 디바이스 상에 임의의 해커에 의해 재서명 및 재설치될 수 있다.

[0013] 이미 언급된 바와 같이, Android 애플리케이션들은 인터프리터 포터블 포맷 (DEX) 을 이용한다. 이 포터블 포맷은 상이한 아키텍처들 및 특징들: ARM, x86, MIPS, Little/Big Endian 등으로 큰 세트의 디바이스들 상에서 실행할 수 있다. 성능을 향상시키기 위해, DEX 코드는 타겟 디바이스에 대해 최적화된 ODEX 또는 ELF 바이너리를 생성하기 위해 설치 시간에 또는 애플리케이션의 첫번째 사용시에 수정된다. 최적화 또는 OAT 컴파일화 동안에, 명령들이 다른 것들로 대체될 수 있고, 명령들의 정렬이 변경될 수도 있고, 바이트 순서가 교체될 수 있는 등의 여러 것들이 코드에서 수정될 수 있다.

[0014] 최적화 및 OAT 컴파일화는 그 후 보안 문제가 발생한다. DEX 파일의 서명이 여전히 CERT.SF 및 CERT.RSA 를 이용하여 검증될 수 있지만, ODEX 및 ELF 파일들에 대해서는 그렇지 않은데, 그 이유는 이들이 수정되었고 이들의 무결성이 오리지널 DEX 서명에 대해 더 이상 링크되지 않기 때문이다. 즉, 무결성 및 인증성은 설치시에만 검증될 수 있고, 공격자는 ODEX 및 ELF 코드를 수정가능하고 그에 따라 헤더에서의 결과적인 체크섬을 업데이트가능하기 때문에 실행시에는 검증될 수 없다.

[0015] 따라서, 시스템은 적어도 2 개 클래스의 공격들: 원격 공격 및 루트 공격에 취약하다. 원격 공격에서, 다운로딩된 악성 애플리케이션은 자신의 특권을 상승시키고 시스템 허가들을 얻는다. 그 후, 악성 애플리케이션은 내부 저장부의 캐시 레포지토리 상에 저장된 ODEX 및 ELF 파일들을 조작할 수도 있다. 루트 공격에서, 공격자는 예를 들어, 디바이스 세션을 잠그지 않고 소유자가 부재될 때 디바이스에 액세스하는 것에 의해 또는 디바이스를 훔치는 것에 의해 Android 디바이스를 획득한다. 공격자는 USB 링크를 통하여 디바이스의 내부 저장부로부터 설치된 애플리케이션을 취출하고, 애플리케이션을 수정한 다음, 내부 저장부 상에 다시 그 수정된 애플리케이션을 푸시한다. 이후의 공격이 성공적이기 위해, 디바이스는 "루트되어야" 한다 (즉, "루트 액세스"는 디바이스의 Android 시스템의 제어를 취할 것을 필요로 한다).

[0016] 따라서, Android 애플리케이션 무결성에서의 신뢰성은 애플리케이션의 라이프 사이클 동안에 깨질 수 있다. Android 시스템 상에 설치된 것을 신뢰할 수 있지만, 실행중인 것은 반드시 신뢰할 수 있는 것은 아니다.

발명의 내용

해결하려는 과제

[0017] 해석된 코드 애플리케이션들의 무결성 및 인증성에 관련된 문제들의 적어도 일부분을 극복하는 솔루션을 갖는 것이 바람직함을 이해할 것이다. 본 개시는 이러한 솔루션을 제공한다.

과제의 해결 수단

[0018] 제 1 양태에서, 본 개시는 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 디바이스에 대하여 교시된다. 디바이스는 수정된 애플리케이션, 및 비수정된 애플리케이션에 대한 저장된 체크섬을 저장하도록 구성되는 메모리, 및 프로세싱 유닛을 포함하고, 프로세싱 유닛은 수정된 애플리케이션의 실행 동안: 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하고, 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬을 생성하고, 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬과 비수정된 애플리케이션에 대한 저장된 체크섬을, 이들 체크섬들이 매칭하는지의 여부를 결정하기 위해 비교하고, 그리고 수정된 애플리케이션이 비수정된 애플리케이션에 대응하는 코드에 대응하는 경우에 그리고 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬이 비수정된 애플리케이션에 대한 저장된 체크섬에 매칭하는 경우에 수정된 애플리케이션의 무결성이 성공적으로 검증되었다고 결정하도록 구성된다.

[0019] 제 1 양태의 여러 실시형태들은 다음을 포함한다:

[0020] 메모리는 비수정된 애플리케이션에 대한 저장된 체크섬에 대한 서명 및 서명한 인증서를 저장하도록 구성되고, 프로세싱 유닛은 서명한 인증서를 이용하여 서명의 유효성을 검증하고, 그리고 서명이 성공적으로 검증된 경우에 수정된 애플리케이션의 무결성이 또한 성공적으로 검증되었다고 결정하도록 구성된다.

[0021] 프로세서는 비수정된 애플리케이션에 대응하는 코드를 획득하여, 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하기 위해 수정된 코드 상에서 역수정을 수행하도록 구성된다.

[0022] 비수정된 애플리케이션에 대응하는 코드는 비수정된 애플리케이션이고, 메모리는 또한 비수정된 애플리케이션을 저장하도록 구성된다. 프로세서는 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하기 위해, 수정된 코드와 비수정된 애플리케이션에 대응하는 코드 사이의 임의의 차이들이 수정 동안에 획득된 적법적 변환들 (legitimate transformations) 에 대응하는지의 여부를 결정하도록 구성되는 것이 유리하다. 대안적으로, 프로세서는 제 2 수정된 코드를 획득하기 위해 비수정된 애플리케이션에 대응하는 코드 상에 수정을 수행하고, 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하기 위해 수정된 코드와 제 2 수정된 코드를 비교하도록 구성되는 것이 유리하다.

[0023] 비수정된 애플리케이션은 해석된 코드로서 구현되고, 수정된 애플리케이션은 최적화된 해석된 코드로서 또는 네이티브 코드로서 구현된다.

[0024] 디바이스는 스마트폰 또는 테블릿이다.

[0025] 제 2 양태에서, 본 개시는 비수정된 애플리케이션의 수정에 의해 획득되었던 수정된 애플리케이션의 무결성을 결정하기 위한 방법에 대하여 교시된다. 수정된 애플리케이션의 실행 동안에, 디바이스는, 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하고, 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬을 생성하고, 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬과 비수정된 애플리케이션에 대한 저장된 체크섬을, 이들 체크섬들이 매칭하는지의 여부를 결정하기 위해 비교하고, 그리고 수정된 애플리케이션이 비수정된 애플리케이션에 대응하는 코드에 대응하는 경우에 그리고 비수정된 애플리케이션에 대응하는 코드에 대한 체크섬이 비수정된 애플리케이션에 대한 저장된 체크섬에 매칭하는 경우에 수정된 애플리케이션의 무결성이 성공적으로 검증되었다고 결정한다.

[0026] 제 2 양태의 여러 실시형태들은 다음을 포함한다:

[0027] 본 방법은 서명한 인증서를 이용하여 서명의 유효성을 검증하는 단계, 및 서명이 성공적으로 검증된 경우에 수정된 애플리케이션의 무결성이 또한 성공적으로 검증되었다고 결정하는 단계를 더 포함한다.

[0028] 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하는 것은, 비수정된 애플리케이션에 대응하는 코드를 획득하기 위해 수정된 코드 상에 역수정을 수행하는 것을 포함한다.

[0029] 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하는 것은, 수정된 코드와, 비수정된 애플리케이션에 대응하는 코드 사이의 임의의 차이들이 수정 동안에 획득된 적법적 변환들에 대응하는지의 여부를 결정하는 것을 포함한다.

[0030] 비수정된 애플리케이션에 대응하는 코드가 또한 수정된 애플리케이션에 대응한다고 결정하는 것은, 제 2 수정된 코드를 획득하기 위해 비수정된 애플리케이션에 대응하는 코드 상에 수정을 수행하고, 수정된 코드와 제 2 수정된 코드를 비교하는 것을 포함한다.

[0031] 제 3 양태에서, 본 개시는 프로세서에 의해 실행될 때, 프로세서로 하여금 제 2 양태의 방법을 수행하게 하는 명령들을 포함하는 컴퓨터 실행가능 프로그램에 대해 교시된다.

도면의 간단한 설명

[0032] 본 개시의 선호되는 특징들은 첨부한 도면들을 참조하여 비제한적 예에 의해 이하 설명된다.

도 1 은 본 개시가 구현되는 예시적 시스템을 예시한다.

도 2 는 예시적 시스템의 기능적 양태들을 예시한다.

도 3 은 본 개시의 선호되는 실시형태에 따른 방법의 선호되는 실시형태를 예시한다.

발명을 실시하기 위한 구체적인 내용

[0033] 본 개시에 따르면, ODEX 또는 ELF 파일들의 무결성은, 대응하는 DEX 에 대한 서명을 검증하는 것에 의해, 그리고 ODEX 또는 ELF 파일들이 DEX 에 대응한다고 검증하는 것에 의해 검증된다.

[0034] 도 1 은 본 개시가 구현되는 예시적 시스템을 예시한다. 시스템은 디바이스 (110) 및 애플리케이션 제공자 (애플리케이션 스토어)(120) 를 포함한다. 디바이스 (110) 는 Android OS 상에서 구동하는 임의의 종류의 적절한 디바이스, 이를 테면, 스마트폰 또는 테블릿일 수 있고, 이는 적어도 하나의 하드웨어 프로세싱 유닛 ("프로세서")(111), 메모리 (112), 사용자와 상호작용하기 위한 사용자 인터페이스 (113), 및 접속 (140), 이를 테면, 인터넷을 통하여 애플리케이션 제공자 (120) 와 통신하기 위한 통신 인터페이스 (114) 를 포함한다. 당해 기술 분야의 당업자는 예시된 디바이스가 명료화의 이유로 매우 간략화되어 있으며, 추가로 실제 디바이스들은 전력 공급 장치들, 및 영구 저장 장치와 같은 특징들을 포함하고 있음을 알고 있을 것이다. 애플리케이션 제공자 (120) 는 디바이스 (110) 에 의해 다운로딩될 수 있는 적어도 하나의 애플리케이션 APK 파일 (122) 을 저장하며, APK 파일은 서명인 엔티티에 의해 서명된 APK 인증서를 포함한다.

[0035] 도 2 는 예시적 시스템의 기능적 양태들을 예시한다. 애플리케이션 (220) 은 서명인 엔티티에 의해 서명된 APK 인증서 (222), 애플리케이션 코드 (224)(설치 전 DEX 및 설치 후 ODEX 또는 ELF 파일들), (가능하다면 리스트에서) 적어도 하나의 서명된 DEX 체크섬 (checksum; CS)(226), 및 적어도 APK 인증서를 서명하는데 이용된 키와 상이한 키를 이용하여 서명한 경우 서명 검증 키 (228) 를 포함하는 서명 인증서, 및 소스 획득 모듈 (232)

및 무결성 검증 모듈 (234) 을 포함한 라이브러리 (230) 를 포함한다.

[0036] DEX 체크섬은 유리하게, DEX 의 일부분에 대한 체크섬이며, 그리고 이는 전체 DEX 에 대한 체크섬에 더하여 제공될 수도 있다. DEX 체크섬은 APK 인증서를 서명하였던 서명 키를 이용하여 서명될 수도 있지만, 이는 또한 상이한 키를 이용하여 서명될 수도 있다.

[0037] 애플리케이션은 또한 서명된 체크섬이 계산되었던 DEX 의 카피본을 포함할 수도 있지만, OS 가 ODEX 또는 ELF 파일들을 생성하여 애플리케이션의 설치 후 DEX 코드와 함께 APK 파일의 적어도 부분을 유지하도록 최적화되어 있을 때, 또한 OS 가 이 DEX 의 카피본을 유지하는 것이 가능하다.

[0038] 소스 획득 모듈 (232) 및 무결성 검증 모듈 (234) 은 다른 무엇보다도 서명 검증을 허용하는 확장된 JNI 라이브러리에 액세스하고 애플리케이션과 함께 패키징되는 APK 의 네이티브 라이브러리에 포함된다.

[0039] 소스 획득 모듈 (232) 은 ODEX 또는 ELF 파일들의 적어도 부분을 취하여 이들을 비교하도록 구성된다. 이를 행하는 여러 방법들이 있다.

[0040] 첫번째 방법에서, 소스 획득 모듈 (232) 은 등가의 DEX 코드를 획득하기 위해 ELF 파일들에 역 컴파일화 기능을 적용하거나 또는 ODEX 에 대한 역 최적화 기능을 적용할 수도 있다. DEX 명령들의 유형에 의존하여, 대부분의 ODEX 및 ELF 파일 코드는 가역적이다. 통상적으로, 연산 코드 (opcode) 를 치환만을 하는 DEX 최적화는 DEX 로부터 ODEX 로 또는 그 반대로 쉽게 수행될 수도 있다.

[0041] 두번째 방법에서, 소스 획득 모듈 (232) 은 (예를 들어, APK 파일로부터의) 오리지널 DEX 코드를 취출하고 이것을 ODEX 또는 ELF 파일과 비교하여, 2 개 사이의 차이가 최적화로 인하여 적법적 변환들에 대응하는지를 결정한다. 그러한 경우에는, 이는 ODEX 또는 ELF 파일들이 오리지널 DEX 에 대응하는 것으로 결정된다. DEX 체크섬은 오리지널 DEX 코드로부터 생성된다.

[0042] 세번째 방법에서, 소스 획득 모듈 (232) 은 (예를 들어, APK 파일로부터의) 오리지널 DEX 코드를 취출하고 최적화를 수행하여, 생성된 ODEX 또는 OAT 컴파일화를 획득하여 저장된 ODEX 또는 ELF 파일들에 비교되는 ELF 파일들을 획득하여 이들이 일치하는지의 여부를 결정한다. DEX 체크섬은 오리지널 DEX 코드로부터 생성된다.

[0043] 3 개의 방법들에서, ODEX 또는 ELF 파일들이 저장된 DEX 에 대응하거나 또는 생성된 DEX 에 대응하는 것으로 결정된다. 따라서, 이들 DEX들의 각각은 ODEX 또는 ELF 파일들에 대응하고, 그리고 ODEX 또는 ELF 파일들을 생성하는데 이용되었던 DEX 에 대응한다.

[0044] 현재 ODEX 또는 ELF 파일들이 서명된 DEX 에 대응하는 DEX 로부터 생성되었다고 소스 획득 모듈 (232) 이 결정하였다면, 무결성 검증 모듈 (234) 은 현재 DEX 체크섬 및 서명을 검증할 수 있다.

[0045] 첫번째 방법에 대해, 소스 획득 모듈 (232) 은 생성된 DEX 로부터 현재 DEX 체크섬을 연산하고 이를 서명된 DEX 체크섬 (226) 과 비교한다. 매칭은 ODEX 또는 ELF 파일들이, 획득된 DEX 로부터 획득되었다는 것을 표시한다. 두번째 및 세번째 방법에 대해, 소스 획득 모듈 (232) 은 오리지널 DEX 로부터 현재 DEX 체크섬을 연산한다. 네번째 방법에 대해, 현재 DEX 체크섬은 오리지널의 최적화되지 않은 DEX 코드로부터 연산된다.

[0046] 무결성 검증 모듈 (234) 은 서명한 인증서 (또는 동일한 키가 이용되었다면 APK 인증서) 로부터 공개 검증 키 (228) 를 취출하도록 구성된다. 이는 또한 DEX 에 대한 서명을 검증하기 위해 그리고 검증 키 (228) 가 취출되었던 인증서의 검증을 검증하기 위해 구성된다.

[0047] 모든 검증들이 성공적인 경우, ODEX 또는 ELF 파일들이 검증된 것으로 고려된다. 적절한 대책들이 다른 경우들에서 취해질 수도 있음을 알고 있을 것이다.

[0048] 도 3 은 선호되는 실시형태에 따른 방법의 플로우차트를 예시한다.

[0049] 단계 S302 에서, 디바이스 (110) 는 ODEX 또는 ELF 파일들, 즉, 서명에 이용가능한 DEX, 즉, 비수정된 코드의 수정에 의해 획득된 수정된 코드를 실행한다.

[0050] 단계 S304 에서, 디바이스 (110) 는 ODEX 또는 ELF 파일들의 적어도 부분이 DEX 에 대응한다고 결정한다. DEX 에 대응하는 코드는 DEX 자체일 수도 있지만, 이는 또한 ODEX 또는 ELF 파일들을 획득하는데 이용된 DEX 의 카피본일 수도 있다. 결정은 본원에 설명된 방법들 중 어느 것을 이용하여 수행될 수도 있다.

[0051] 디바이스 (110) 가 ODEX 또는 ELF 파일들이 DEX 에 대응한다고 결정하는 경우, 단계 S306 에서 DEX 체크섬이 검증된다.

[0052] DEX 체크섬이 성공적으로 검증되는 경우, 단계 S308 에서 디바이스 (110) 가 DEX 체크섬에 대한 서명을 검증한다.

[0053] 단계 S310 에서, ODEX 또는 ELF 파일들이 DEX 에 대응하고 DEX 에 대한 체크섬이 검증되는 경우에 이것이 행해지기 때문에 서명의 긍정적 검증의 경우 ODEX 또는 ELF 파일들의 무결성이 검증되었던 것으로 결정된다.

[0054] 단계들 304, 306 및 308 이 임의의 순서로 수행될 수 있음을 주지해야 한다. 예를 들어, 먼저, DEX 체크섬에 대한 서명이 검증되고 (단계 S308) 그리고 나서 DEX 체크섬이 검증되고 (단계 S306) 마지막으로, ODEX 또는 ELF 파일들 및 DEX 가 매칭하는지의 여부가 결정된다 (단계 S304). 이들 단계들 중 적어도 일부가 또한 동시에 수행될 수도 있다.

[0055] 무결성은 애플리케이션의 실행 동안에 복수회 체크될 수도 있다.

[0056] 솔루션은 현재 배치된 Android 시스템들에 대한 어떠한 수정도 필요하지 않음을 주지한다.

[0057] 본 설명에서, 용어 '체크섬' 은 생성되었던 데이터가 체크섬의 생성 후에 수정되었는지의 여부의 검증을 가능하게 하는 값을 커버하도록 의도된다. 체크섬은 따라서 예를 들어, 또한 해시 값, 주기적 리던던시 체크 (Cyclic Redundancy Check; CRC) 값 또는 다른 종류의 다이제스트일 수도 있고, 체크섬으로부터 코드를 획득하는 것이 연산적으로 실행불가능한 것이 선호된다. 추가로, 단일의 체크섬이 명료화를 위해 이용되었지만, 복수의 체크섬들이 이용될 수도 있고, 여기에서, 체크섬은 코드의 별개의 부분에 대해 생성될 수도 있고 (여기에서 상이한 부분들이 중복할 수도 있음), 코드의 상이한 부분들에 대한 복수의 체크섬들이 비교에 이용된 단일의 글로벌 체크섬을 생성하는데 이용된다. 서명은 임의의 적절한 암호화 서명, 이를 테면, 해시 기반 메시지 인증 코드 (Hash-based Message Authentication Code; HMAC) 또는 예를 들어, RSA 에 기초한 서명, 디지털 서명 알고리즘 (Digital Signature Algorithm; DSA) 또는 타원곡선 디지털 서명 알고리즘 (Elliptic Curve Digital Signature Algorithm; ECDSA) 일 수도 있다.

[0058] 본 솔루션은 루트 공격들 및 원격 공격들 양쪽을 성공적으로 대처할 수 있는 것으로 이해될 것이다.

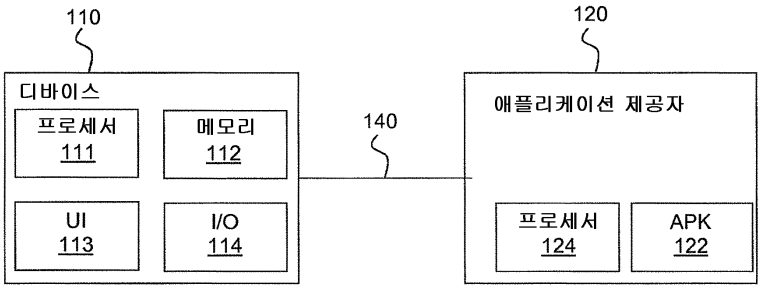
[0059] 본 솔루션이 Android 환경에서 설명되어 있지만, 이는 런타임시 설치된 애플리케이션의 보안 무결성 검증을 실행함이 없이 설치 동안에 코드를 수정하는 다른 오퍼레이팅 시스템들에도 채택될 수 있다.

[0060] 따라서, 본 개시는 Android 디바이스들 상의 애플리케이션들의 런타임 무결성을 가능하게 할 수 있는 솔루션을 제공함을 이해할 것이다.

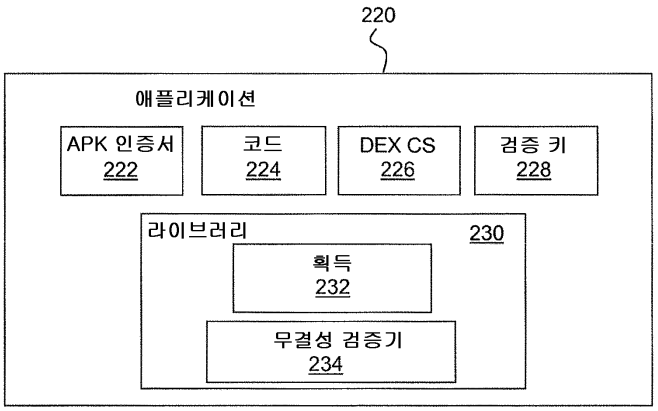
[0061] 명세서 및 (필요에 따라) 청구항들 및 도면들에 개시된 각각의 특징은 임의의 적절한 조합들로 또는 독립적으로 제공될 수도 있다. 하드웨어에서 구현되는 바와 같이 설명된 특징들은 소프트웨어에서 또한 구현될 수도 있고, 그 역도 가능하다. 청구항들에 나타내어진 참조 번호들은 예시적인 목적일 뿐, 청구항들의 범위에 대하여 어떠한 제한 효과도 없다.

도면

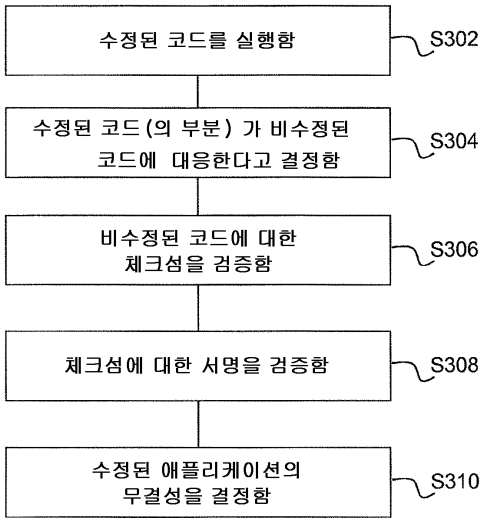
도면1



도면2


220
애플리케이션
APK 인증서
222
코드
224
DEX CS
226
검증 키
228
라이브러리
230
획득
232
무결성 검증기
234

도면3


수정된 코드를 실행함
S302
수정된 코드(의 부분) 가 비수정된 코드에 대응한다고 결정함
S304
비수정된 코드에 대한 체크섬을 검증함
S306
체크섬에 대한 서명을 검증함
S308
수정된 애플리케이션의 무결성을 결정함
S310