(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2019-0038542
(43) 공개일자 2019년04월08일

(51) 국제특허분류(Int. Cl.)
G06F 21/56 (2013.01) *G06F 12/1009* (2016.01)
G06F 12/12 (2016.01) *G06F 21/53* (2013.01)
(52) CPC특허분류
G06F 21/562 (2013.01)
G06F 12/1009 (2013.01)
(21) 출원번호 10-2019-7002550
(22) 출원일자(국제) 2017년06월30일
심사청구일자 없음
(85) 번역문제출일자 2019년01월25일
(86) 국제출원번호 PCT/US2017/040492
(87) 국제공개번호 WO 2018/022255
국제공개일자 2018년02월01일
(30) 우선권주장
62/368,223 2016년07월29일 미국(US)
15/245,037 2016년08월23일 미국(US)

(71) 출원인
퀄컴 인코포레이티드
미국 92121-1714 캘리포니아주 샌 디에고 모어하우스 드라이브 5775
(72) 발명자
데 수브라토 쿠마르
미국 92121 캘리포니아주 샌디에고 모어하우스 드라이브 5775
(74) 대리인
특허법인코리아나

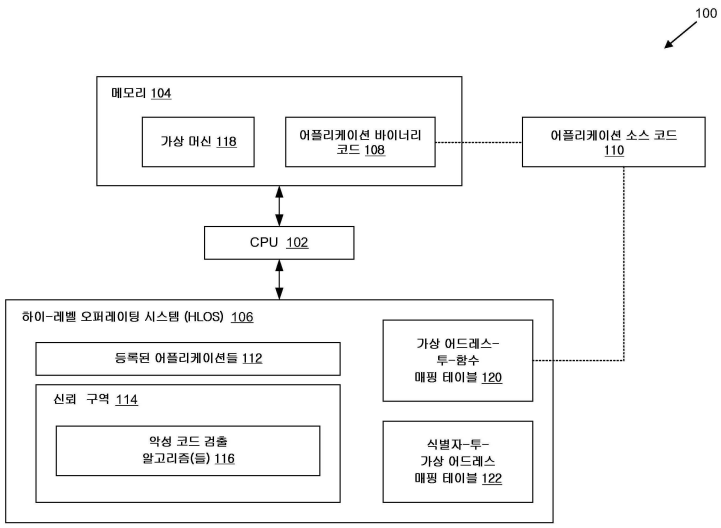
전체 청구항 수 : 총 30 항

(54) 발명의 명칭 **가상 어드레스 매핑을 사용한 타겟 어플리케이션 기능성의 커널 기반 검출**

(57) 요 약

컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템들, 방법들, 및 컴퓨터 프로그램들이 개시된다. 하나의 방법은, 컴퓨팅 디바이스 상의 보안 메모리에, 어플리케이션에 대한 가상 어드레스 매핑 테이블을 저장하는 단계를 포함한다. 가상 어드레스 매핑 테이블은 대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리 코드에서의 복수의 가상 어드레스들을 포함한다. 어플리케이션은 하이-레벨 오퍼레이팅 시스템 (HLOS) 에 등록된다. 어플리케이션 바이너리 코드의 실행 동안, HLOS 는 타겟 어플리케이션 기능성들에 대응하는 가상 어드레스들 중 하나 이상이 가상 어드레스 매핑 테이블에 기초하여 실행되는 경우를 검출한다.

대 표 도



(52) CPC특허분류
G06F 12/12 (2013.01)
G06F 21/53 (2013.01)
G06F 21/566 (2013.01)
G06F 2212/1052 (2013.01)
G06F 2212/65 (2013.01)
G06F 2221/033 (2013.01)
G06F 2221/034 (2013.01)

명 세 서

청구범위

청구항 1

컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법으로서,

컴퓨팅 디바이스 상의 보안 메모리에, 어플리케이션에 대한 가상 어드레스 매핑 테이블을 저장하는 단계로서, 상기 가상 어드레스 매핑 테이블은 대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리 코드에서의 복수의 가상 어드레스들을 포함하는, 상기 가상 어드레스 매핑 테이블들을 저장하는 단계;

상기 어플리케이션을 하이-레벨 오퍼레이팅 시스템 (HLOS) 에 등록하는 단계; 및

상기 어플리케이션 바이너리 코드의 실행 동안, 상기 HLOS 가 상기 타겟 어플리케이션 기능성들에 대응하는 상기 가상 어드레스들 중 하나 이상이 상기 가상 어드레스 매핑 테이블에 기초하여 실행되는 경우를 검출하는 단계를 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 2

제 1 항에 있어서,

상기 보안 메모리는 상기 HLOS 에서의 신뢰 구역에 상주하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 3

제 1 항에 있어서,

상기 어플리케이션 바이너리 코드가 업데이트될 때 상기 타겟 어플리케이션 기능성들에 대해 개정된 가상 어드레스들로 상기 가상 어드레스 매핑 테이블을 업데이트하는 단계를 더 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 4

제 1 항에 있어서,

상기 어플리케이션의 실행과 연관된 하나 이상의 예외들 또는 거동들을 검출하도록 구성된 예외 핸들링 모듈에 상기 가상 어드레스들로부터 검출된 바와 같은 실행된 상기 타겟 어플리케이션 기능성들을 제공하는 단계를 더 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 5

제 4 항에 있어서,

상기 예외 핸들링 모듈은 악성 코드 검출 알고리즘을 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 6

제 1 항에 있어서,

상기 어플리케이션은 보안 웹 어플리케이션 및 웹 브라우저 중 하나를 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 7

제 1 항에 있어서,

상기 어플리케이션 바이너리 코드는 네이티브 바이너리 코드로서 실행되는, 컴퓨팅 디바이스 상에서 실행하는

어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 8

제 1 항에 있어서,

상기 어플리케이션 바이너리 코드는 연관된 가상 머신을 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 방법.

청구항 9

컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템으로서,

컴퓨팅 디바이스 상의 보안 메모리에, 어플리케이션에 대한 가상 어드레스 매핑 테이블을 저장하는 수단으로서, 상기 가상 어드레스 매핑 테이블은 대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리에서의 복수의 가상 어드레스들을 포함하는, 상기 가상 어드레스 매핑 테이블들을 저장하는 수단;

상기 어플리케이션을 하이-레벨 오퍼레이팅 시스템 (HLOS) 에 등록하는 수단; 및

상기 어플리케이션 바이너리 코드의 실행 동안, 상기 타겟 어플리케이션 기능성들에 대응하는 상기 가상 어드레스들 중 하나 이상이 상기 가상 어드레스 매핑 테이블에 기초하여 실행되는 경우를 검출하는 수단을 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 10

제 9 항에 있어서,

상기 보안 메모리는 상기 HLOS 에서의 신뢰 구역에 상주하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 11

제 9 항에 있어서,

상기 어플리케이션 바이너리 코드가 업데이트될 때 상기 타겟 어플리케이션 기능성들에 대해 개정된 가상 어드레스들로 상기 가상 어드레스 매핑 테이블을 업데이트하는 수단을 더 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 12

제 9 항에 있어서,

상기 어플리케이션의 실행과 연관된 하나 이상의 예외들을 검출하도록 구성된 예외 핸들링 모듈에 상기 가상 어드레스들로부터 검출된 바와 같은 실행된 상기 타겟 어플리케이션 기능성들을 제공하는 수단을 더 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 13

제 12 항에 있어서,

상기 예외 핸들링 모듈은 악성 코드 검출 알고리즘을 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 14

제 9 항에 있어서,

상기 어플리케이션은 보안 웹 어플리케이션 및 웹 브라우저 중 하나를 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 15

제 9 항에 있어서,

상기 어플리케이션 바이너리 코드는 네이티브 바이너리 코드로서 실행되는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 16

제 9 항에 있어서,

상기 어플리케이션 바이너리 코드는 연관된 가상 머신을 포함하는, 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 17

메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램으로서,

방법이,

컴퓨팅 디바이스 상의 보안 메모리에, 어플리케이션에 대한 가상 어드레스 매핑 테이블을 저장하는 단계로서, 상기 가상 어드레스 매핑 테이블은 대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리에서의 복수의 가상 어드레스들을 포함하는, 상기 가상 어드레스 매핑 테이블들을 저장하는 단계;

상기 어플리케이션을 하이-레벨 오퍼레이팅 시스템 (HLOS) 에 등록하는 단계; 및

상기 어플리케이션 바이너리 코드의 실행 동안, 상기 HLOS 가 상기 타겟 어플리케이션 기능성들에 대응하는 상기 가상 어드레스들 중 하나 이상이 상기 가상 어드레스 매핑 테이블에 기초하여 실행되는 경우를 검출하는 단계를 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 18

제 17 항에 있어서,

상기 보안 메모리는 상기 HLOS 에서의 신뢰 구역에 상주하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 19

제 17 항에 있어서,

상기 방법이,

상기 어플리케이션 바이너리 코드가 업데이트될 때 상기 타겟 어플리케이션 기능성들에 대해 개정된 가상 어드레스들로 상기 가상 어드레스 매핑 테이블을 업데이트하는 단계를 더 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 20

제 17 항에 있어서,

상기 방법이,

상기 어플리케이션의 실행과 연관된 하나 이상의 예외들 또는 거동들을 검출하도록 구성된 예외 핸들링 모듈에 상기 가상 어드레스들로부터 검출된 바와 같은 실행된 상기 타겟 어플리케이션 기능성들을 제공하는 단계를 더 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 21

제 20 항에 있어서,

상기 예외 핸들링 모듈은 악성 코드 검출 알고리즘을 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실

행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 22

제 17 항에 있어서,

상기 어플리케이션은 보안 웹 어플리케이션 및 웹 브라우저 중 하나를 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 23

제 17 항에 있어서,

상기 어플리케이션 바이너리 코드는 네이티브 바이너리 코드로서 실행되는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 24

제 17 항에 있어서,

상기 어플리케이션 바이너리 코드는 연관된 가상 머신을 포함하는, 메모리에 수록되고 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위해 프로세서에 의해 실행가능한 컴퓨터 프로그램.

청구항 25

실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템으로서,

어플리케이션 바이너리 코드를 실행하도록 구성된 프로세싱 디바이스; 및

대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리에서의 복수의 가상 어드레스들을 포함하는 가상 어드레스 매핑 테이블을 포함하는 하이-레벨 오퍼레이팅 시스템 (HLOS) 을 포함하고,

상기 HLOS 는 상기 타겟 어플리케이션 기능성들에 대응하는 상기 가상 어드레스들 중 하나 이상이 실행되는 경우를 검출하도록 구성되는, 실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 26

제 25 항에 있어서,

상기 보안 메모리는 상기 HLOS 에서의 신뢰 구역을 포함하는, 실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 27

제 25 항에 있어서,

상기 HLOS 는 상기 가상 어드레스들로부터 검출된 바와 같은 실행된 상기 타겟 어플리케이션 기능성들을 수신하고 상기 어플리케이션의 실행과 연관된 하나 이상의 예외들을 검출하도록 구성된 예외 핸들링 모듈을 더 포함하는, 실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 28

제 27 항에 있어서,

상기 예외 핸들링 모듈은 악성 코드 검출 알고리즘을 포함하는, 실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

청구항 29

제 25 항에 있어서,

상기 어플리케이션은 보안 웹 어플리케이션 및 웹 브라우저 중 하나를 포함하는, 실행 어플리케이션의 하이-레

벨 기능성을 검출하기 위한 시스템.

청구항 30

제 25항에 있어서,

상기 어플리케이션 바이너리 코드는 연관된 가상 머신을 포함하는, 실행 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템.

발명의 설명

기 술 분 야

[0001] 관련 출원에 대한 우선권 주장 및 상호 참조

[0002] 본 출원은 2016 년 7 월 29 일 출원된 명칭이 "KERNEL-BASED DETECTION OF TARGET APPLICATION FUNCTIONALITY USING VIRTUAL ADDRESS MAPPING" 인 미국 가출원 제 62/368,223 호에 대해 35 U.S.C. 119(e) 하에서 우선권을 주장하며, 그 전체 내용은 참조로서 본 명세서에 통합된다.

배 경 기 술

[0003] 시스템 또는 플랫폼 계층에서 어떠한 주목할만한 활동도 나타내지 않고 따라서 어플리케이션 실행의 유용한 기능성 (functionaliy) 및 거동 정보를 검출하는데 어떠한 기회도 제공하지 않는 하드웨어 플랫폼 상에서 작동하는 다양한 하이-레벨 어플리케이션들이 있다.

[0004] 일반적인 예는 하이-레벨 웹 브라우저 시스템 및 플랫폼 레벨에서 어떠한 표시적 트레이스도 남기지 않는 디바이스 상의 실행 동안의 보안 악용 (exploit) 들 (예를 들어, 크로스 사이트 스크립팅) 과 절충될 시의 하이-레벨 브라우저 어플리케이션이다. 시스템 라이브러리들, 플랫폼, SOC 하드웨어를 조사하거나 디바이스 레벨 활동들을 주시함으로써 이러한 활동이 하이-레벨 어플리케이션 상에서 발생하고 있는 것을 결정하는 방법은 없다.

[0005] 따라서, 디바이스 상에서 작동하는 다양한 제 3 자 어플리케이션들 상에서 우수한 플랫폼 레벨 제어를 갖고 이러한 실행의 하이-레벨 어플리케이션들의 기능성 및 거동 활동들의 일부를 검출하기 위해, 플랫폼의 HLOS 또는 커널이 이해할 수 있는 형태로 하이-레벨 어플리케이션 기능성들 및 거동을 표현하고 통신하는 것을 가능하게 하는 메커니즘을 개발할 필요가 있다. 이것은 플랫폼이 실행 어플리케이션의 거동에 대해 우수한 이해를 가질 수 있도록 하고 플랫폼이 실행 어플리케이션들의 다양한 상이한 상황들을 핸들링하기 위한 결정들 및 액션들을 취할 수 있도록 할 것이다.

[0006] 일 예로서 제 3 자 웹 브라우저 어플리케이션 상의 웹 보안 악용을 방지하기 위한 플랫폼 레벨 결정이 이 정보를 사용하여 취해질 수 있다. 예시의 사용들의 다른 영역들은, 어플리케이션의 특정 기능성 또는 거동 성질이 본 개시물에서의 메커니즘을 사용하여 HLOS 또는 커널 계층에서 일단 검출되면, 플랫폼이 다양한 SOC 컴포넌트들 (DDR, 버스, CPU, 캐시들) 의 주파수를 증가/감소시키는 것과 같은 결정들을 취하거나 고 또는 저 전력 모드를 고용한다는 것이다. 일반적으로, 이러한 개시물에 의하면, 플랫폼은 어플리케이션에 의해 실행되고 있는 기능성을 검출하고 인식함으로써 디바이스 상에서 실행하는 다양한 제 3 자 어플리케이션들에 대해 다양한 제어들을 행하기 위한 기회를 얻는다. 이것은 SOC 및 플랫폼 벤더들이 플랫폼이 달리 어떻게 할 수 없는 다양한 제 3 자 어플리케이션들에 대한 플랫폼 레벨로부터 우수한 솔루션을 제공할 수 있도록 한다.

발명의 내용

해결하려는 과제

과제의 해결 수단

[0007] 컴퓨팅 디바이스 상에서 실행하는 어플리케이션의 하이-레벨 기능성을 검출하기 위한 시스템들, 방법들 및 컴퓨터 프로그램들이 개시된다. 하나의 방법은, 컴퓨팅 디바이스 상의 보안 메모리에, 어플리케이션에 대한 가상 어드레스 매핑 테이블을 저장하는 단계를 포함한다. 가상 어드레스 매핑 테이블은 대응하는 타겟 어플리

케이션 기능성들에 매핑된 어플리케이션 바이너리 코드에서의 복수의 가상 어드레스들을 포함한다. 어플리케이션은 하이-레벨 오퍼레이팅 시스템 (HLOS) 에 등록된다. 어플리케이션 바이너리 코드의 실행 동안, HLOS 는 타겟 어플리케이션 기능성들에 대응하는 가상 어드레스들 중 하나 이상이 가상 어드레스 매핑 테이블에 기초하여 실행되는 경우를 검출한다.

[0008] 다른 실시형태는 프로세싱 디바이스 및 하이-레벨 오퍼레이팅 시스템 (HLOS) 을 포함하는 시스템이다. 프로세싱 디바이스는 어플리케이션 바이너리 코드를 실행하도록 구성된다. HLOS 는 대응하는 타겟 어플리케이션 기능성들에 매핑된 어플리케이션 바이너리에서의 복수의 가상 어드레스들을 포함하는 가상 어드레스 매핑 테이블을 포함한다. HLOS 는 타겟 어플리케이션 기능성들에 대응하는 가상 어드레스들의 하나 이상이 실행되는 경우를 검출하도록 구성된다.

도면의 간단한 설명

[0009] 도면들에서, 같은 참조 번호들은 달리 나타내지 않으면 다양한 뷰들 전체에 걸쳐 같은 부분들을 지칭한다. "102A" 또는 "102B" 와 같은 문자 지정들을 갖는 참조 번호들에 대해, 문자 지정들은 동일한 도면에 존재하는 2 개의 같은 부분들 또는 엘리먼트들을 구별할 수도 있다. 모든 도면들에서 동일한 참조 번호를 갖는 모든 부분들을 포함하도록 참조 번호가 의도되는 경우 참조 번호들에 대한 문자 지정들은 생략될 수도 있다.

도 1 은 보안 메모리에서 가상 어드레스 매핑을 사용하여 타겟 어플리케이션 기능성을 검출하기 위한 시스템의 실시형태의 블록 다이어그램이다.

도 2 는 대응하는 어플리케이션 바이너리 코드로의 타겟 어플리케이션 기능성의 예시적인 매핑을 도시한다.

도 3 은 가상 어드레스-투-함수 매핑 테이블 (virtual address-to-function mapping tagle; VAFMT) 의 예시적인 실시형태를 도시한다.

도 4 는 도 1 의 시스템에서 악성 코드 활동을 검출하기 위한 방법의 실시형태를 도시하는 플로우챠트이다.

도 5 는 가상 머신 코드 공간의 경계들을 동적으로 식별하기 위해 사용된 VAFMT 의 다른 실시형태를 도시한다.

도 6 은 VAFMT 와의 조합으로 사용된 식별자-투-가상 매핑 테이블 (IVAMT) 의 실시형태를 도시한다.

도 7 은 가비지 수집 프로세스와 관련하여 사용된 VM 코드 공간의 일부를 나타낸다.

도 8 은 가상 머신을 포함한 어플리케이션 바이너리의 실행 동안 가비지 수집 활동의 실행을 검출하는데 사용되는 VAFMT 에서의 관심의 기능적 (functional) 포인트들에 대한 가상 어드레스들 및 도 1 의 가상 머신에서의 가비지 수집 함수에 대한 관심의 예시적인 포인트들을 나타낸다. 도 9 는 가상 머신 히프 (heap) 에 대해 외부/내부 경계들에 대한 가상 어드레스들의 예시적인 매핑을 도시한다.

도 10 은 가상 머신 실시형태에 있어서 도 1 의 시스템에서 악성 코드 활동을 검출하기 위한 방법의 실시형태를 도시하는 플로우챠트이다.

도 11 은 버퍼에 할당된 오브젝트들의 멤버들/필드들의 값들 및 특정 데이터 구조 유형들의 오브젝트들을 포함하는 동적으로 할당된 버퍼들의 가상 어드레스들을 결정하는데 사용되는 특정 버퍼 할당기 함수들에 대한 가상 어드레스들을 포함하는 VAFMT 의 실시형태를 도시한다.

도 12 는 어플리케이션 바이너리 코드의 업데이트된 버전을 수신하는 것에 응답하여 VAMFT 를 자동으로 업데이트하기 위한 시스템의 실시형태를 도시하는 결합된 블록/플로우 다이어그램이다.

도 13 은 업데이트된 가상 어드레스들 및 메타데이터로 도 12 의 VAFMT 를 도시한다.

도 14 는 의사 바이너리 코드 템플릿으로의 도 12 의 VAFMT 에서의 관심의 기능적 포인트들의 예시적인 매핑을 도시한다.

도 15 는 어플리케이션 바이너리 코드의 업데이트된 버전에서 매칭된 영역으로의 도 14 의 의사 바이너리 코드 템플릿의 예시적인 매칭을 도시한다.

도 16 은 어플리케이션 바이너리 코드의 업데이트된 버전을 수신하는 것에 응답하여 VAFMT 를 업데이트하기 위한 방법의 실시형태를 도시하는 플로우챠트이다.

발명을 실시하기 위한 구체적인 내용

[0010] 본 명세서에서 단어 "예시적인" 은 "예, 예증, 또는 예시로서 작용하는" 을 의미하도록 사용된다. 본 명세서에서 "예시적인" 것으로 기재된 임의의 양태가 반드시 다른 양태들보다 선호되거나 유리한 것으로 해석되지 않아야 한다.

[0011] 본 기재에서, 용어 "어플리케이션" 은 또한 오브젝트 코드, 스크립트들, 바이트 코드, 마크업 (markup) 언어 파일들, 및 패치 (patch) 들과 같은 실행가능한 콘텐트를 갖는 파일들을 포함할 수도 있다. 부가적으로, 본 명세서에 언급된 "어플리케이션" 은 또한, 개방되어야 할 수도 있는 문서들 또는 액세스되어야 하는 다른 데이터 파일들과 같은, 본질적으로 실행가능하지 않은 파일들을 포함할 수도 있다.

[0012] 용어 "콘텐트" 는 또한, 오브젝트 코드, 스크립트들, 바이트 코드, 마크업 언어 파일들, 및 패치들과 같은 실행가능한 콘텐트를 갖는 파일들을 포함할 수도 있다. 부가적으로, 본 명세서에 언급된 "콘텐트" 는 또한, 개방되어야 할 수도 있는 문서들 또는 액세스되어야 하는 다른 데이터 파일들과 같은, 본질적으로 실행가능하지 않은 파일들을 포함할 수도 있다.

[0013] 본 기재에서 사용된 바와 같이, 용어들 "컴포넌트", "데이터베이스", "모듈", "시스템" 등은 컴퓨터 관련 엔티티, 하드웨어, 펌웨어, 하드웨어 및 소프트웨어의 조합, 소프트웨어, 또는 실행에서의 소프트웨어 중 어느 하나를 지칭하도록 의도된다. 예를 들어, 컴포넌트는 프로세서 상에서 작동하는 프로세스, 프로세서, 오브젝트, 실행가능물, 실행의 스레드 (thread), 프로그램, 및/또는 컴퓨터일 수도 있지만 이에 제한되지 않는다. 예시로서, 컴퓨팅 디바이스 상에서 작동하는 어플리케이션 및 그 컴퓨팅 디바이스는 양자 모두 컴포넌트일 수도 있다. 하나 이상의 컴포넌트들은 실행의 스레드 및/또는 프로세스 내에 상주할 수도 있고, 컴포넌트는 하나의 컴퓨터 상에서 국부화되고 및/또는 2 이상의 컴포넌트들 사이에서 분산될 수도 있다. 또한, 이러한 컴포넌트들은 다양한 데이터 구조들이 저장된 다양한 컴퓨터 판독가능 매체들로부터 실행할 수도 있다. 컴포넌트들은 로컬 및/또는 원격 프로세스들에 의해, 예컨대 하나 이상의 데이 m터 패킷들을 갖는 신호 (예를 들어, 그 신호에 의해 다른 시스템들과 인터넷과 같은 네트워크를 통해 및/또는 로컬 시스템, 분산된 시스템에서 다른 컴포넌트와 상호작용하는 하나의 컴포넌트로부터의 데이터) 에 따라 통신할 수도 있다.

[0014] 도 1 은 커널 또는 오퍼레이팅 시스템 (O/S) 계층으로부터 어플리케이션 바이너리의 원하는 또는 타겟 하이-레벨 기능성들을 검출하기 위한 시스템 (100) 의 실시형태를 도시한다. 도 1 의 실시형태에 도시된 바와 같이, 시스템 (100) 은 프로세싱 디바이스 (예를 들어, 중앙 프로세싱 유닛 (CPU)(102)), 메모리 (104), 및 하이-레벨 오퍼레이팅 시스템 (HLOS)(106) 을 포함한다. 메모리 (104) 는 CPU (102) 에 의해 실행될 수도 있는 하나 이상의 어플리케이션들을 저장한다. 메모리 (104) 는 컴퓨팅 디바이스 상에 설치된 어플리케이션(들) 과 연관된 참조 어플리케이션 소스 코드에 대응하는 어플리케이션 바이너리 코드 (108) 를 저장할 수도 있다. 이와 관련하여, 시스템 (100) 은, 예를 들어 개인용 컴퓨터, 랩탑 컴퓨터, 워크스테이션, 서버, 또는 휴대용 컴퓨팅 디바이스 (PCD), 예컨대 셀룰러 전화기, 스마트 폰, 휴대용 디지털 보조기 (PDA), 휴대용 게임 콘솔, 내비게이션 디바이스, 테블릿 컴퓨터, 웨어러블 디바이스 (예를 들어, 스마트 워치), 또는 다른 배터리-전력공급형 휴대용 디바이스를 포함한, 임의의 바람직한 컴퓨팅 디바이스 또는 시스템에서 구현될 수도 있다.

[0015] 실시형태에서, 커널 또는 O/S 계층은 하이-레벨 오퍼레이팅 시스템 (HLOS)(106) 을 포함한다. 도 1 에 도시된 바와 같이, HLOS (106) 는 등록된 어플리케이션들의 리스트 (112), 보안 메모리 (예를 들어, 신뢰 구역 (114)), 및 각각의 등록된 어플리케이션의 어플리케이션 바이너리 코드 (108) 를 위해 특별히 구성된 가상 어드레스 매핑 테이블(들) 을 포함한다. 등록된 어플리케이션들 (112) 의 리스트는 보안 제어 및/또는 지원을 위해 HLOS (106) 에 등록되었던 시스템 (100) 상에 설치된 어플리케이션들을 식별한다. 예를 들어, 어플리케이션 (예를 들어, 웹 어플리케이션, 브라우저 어플리케이션 등) 의 어플리케이션 바이너리 코드 (108) 는 HLOS (106) 에 등록되고 리스트 (112) 에서 식별될 수도 있다. 종래에 알려진 바와 같이, 신뢰 구역 (114) 은 메모리에 로딩되고 및/또는 실행되는 코드 및/또는 데이터가 보안, 기밀성, 무결성 등과 관련하여 보호되는 것을 보장하도록 구성된 보안 메모리 또는 영역을 포함한다. 등록된 어플리케이션(들) 에 대한 어플리케이션 바이너리 코드 (108) 는, 미리결정된 가상 어드레스 포인트들의 실행을 추적함으로써 원하는 또는 타겟 하이-레벨 어플리케이션 기능성들을 식별하기 위해 신뢰 구역 (114) 에서 HLOS (106) 및/또는 알고리즘에 의해 사용되는, 하나 이상의 가상 어드레스 매핑 테이블(들) 을 가질 수도 있다.

[0016] 시스템 (100) 은 커널 계층에서 하이-레벨 어플리케이션 기능성들을 추적하고 검출하는 것이 유리한 다양한 어플리케이션 도메인들에 적용될 수도 있다는 것을 알아야 한다. 예를 들어, 하나의 예시적인 실시형태에서, 커널은 다양한 시스템 온 칩 (SoC) 컴포넌트들 (예를 들어, 중앙 프로세싱 유닛 (CPU), 캐시(들), 이중 데이터 레이트 (DDR) 메모리, 하나 이상의 버스들 등) 의 주파수를 증가 및/또는 감소시키는 것과 같은 결정들을 제어

하거나, 고 및/또는 저 전력 모드들을 설정하고 실행 어플리케이션들의 특정 기능성 또는 거동 성질의 검출에 응답하여 특정 하드웨어 피처들을 인에이블/디스에이블할 수도 있다. 이러한 방식으로, HLOS (106) 및 커널은 어플리케이션에 의해 실행되고 있는 기능성을 검출 및 인식함으로써 디바이스 상에서 실행하는 다양한 제 3 자 어플리케이션들에 대해 다양한 제어들을 구현하는 기회를 갖는다. 이것은 SoC 및 플랫폼 벤더들이 플랫폼이 달리 어떻게 할 수 없는 다양한 제 3 자 어플리케이션들에 대해 플랫폼/HLOS/커널 레벨로부터 개선된 솔루션들을 제공하도록 할 수도 있다는 것을 알아야 한다.

[0017] 예시적인 어플리케이션 도메인에서, 시스템 (100) 은 악성 공격들 또는 웹 어플리케이션들, 웹 브라우저들, 자바스크립트 (JavaScript) 코드 등의 다른 악용에 대한 실시간 보안 보호를 제공할 수도 있다. 종래에 알려진 바와 같이, 자바스크립트는 많은 웹사이트들 및 웹 어플리케이션들에서 사용되는 프로그래밍 언어이며, 자마스크립트 기반 공격들은 사이버 보안에 대한 상위 위협들 중 하나이다. 점점 더 많은 웹 활동이 데스크탑 컴퓨터들로부터 모바일로 시프트됨에 따라, 자바스크립트 공격들이 휴대용 컴퓨팅 디바이스들 상에서 주요 위협이 되고 있다.

[0018] 가장 악성인 자바스크립트 공격들은 악용들에 대해 웹 표준들 및 사양들의 제약들 및 자바스크립트 언어의 특징들을 활용한다. 악성 자바스크립트를 통한 웹 기반 악용들의 일반적인 예들은 다음을 포함한다: 크로스-사이트 스크립팅 (cross-site scripting)(즉, XSS/CSS), 크로스-사이트 요청 위조 (cross-site request forgery)(즉, CSRF/XSRF), 드라이브-바이 다운로드들, 사용자 인터넷 하이재킹 (hijacking), 클릭재킹 (clickjacking), DDoS (distributed Denial of Service), 자바스크립트 스테가노그라피 (steganography) 및 다양한 형태의 오퍼스케이트된 (obfuscated) 자바스크립트. 악성 거동들을 검출하기 위한 시도 시 하이-레벨 웹 거동 및 기능성 지식이 필요하기 때문에, 현재 웹 및 자바스크립트 보안 솔루션들은 통상적으로 브라우저 소프트웨어 아키텍처 내에 구축된다.

[0019] 하지만, HLOS, 커널 및 디바이스 플랫폼 내의 내장 웹 보안 메커니즘은, 웹/자바스크립트-기반 악용들이 플랫폼 활동 (예를 들어, 시스템 호출들, 디바이스 사용량 등) 에 대해 가시적인 표시를 갖지 않을 수도 있기 때문에 제한된다. 많은 웹/자바스크립트-기반 공격들은 표면상 대면하고 있고 사용자의 온라인 어셋 (asset) 들, 활동, 아이덴티티 등만을 절충한다. 즉, 가시적인 활동 패턴들은 단지 웹 브라우저/어플리케이션 소프트웨어 내에서만 검출될 수도 있고 이로써 웹 악용들에 대한 대부분의 보안 메커니즘들은 거의 항상 웹 브라우저 어플리케이션 내에 구축된다.

[0020] 이와 관련하여, 시스템 (100) 에서의 어플리케이션 바이너리 코드 (108) 의 예시적인 실시형태들은 웹 어플리케이션들, 브라우저 어플리케이션들 또는 HLOS (106) 가 미리결정된 가상 어드레스 포인트들을 추적함으로써 하이-레벨 어플리케이션 기능성들을 검출하는 다른 어플리케이션들을 포함할 수도 있다. 도 1 에 추가로 도시된 바와 같이, 시스템 (100) 은 신뢰 구역 (114) 에 상주하는 하나 이상의 악성 코드 검출 알고리즘들 (116) 을 더 포함할 수도 있다. 악성 코드 검출 알고리즘들 (116) 은 가상 어드레스 포인트들의 실행에 관련된 데이터 및 가상 어드레스 매핑 테이블들에서 식별되는 바와 같은 그 연관된 기능적 의미들을 수신할 수도 있다. 이러한 데이터에 기초하여, 알고리즘(들)(116) 은 예를 들어, 악성 코드 및 거동, 악성 자바스크립트 코드 및 실행 등을 검출하고, 보안 위협을 해결하거나 또는 그렇지 않으면 악성 공격을 저지하기 위한 적절한 방법들을 개시할 수도 있다. 실시형태에서, 보안 위협이 검출될 때, 시스템 (100) 은 자동으로 그 위협을 해결하거나 사용자에게 적절한 액션(들) 을 프롬프트할 수도 있다.

[0021] 도 1 의 실시형태에 도시된 바와 같이, HLOS (106) 에 의해 사용된 가상 어드레스 매핑 테이블들은 가상 어드레스-투-함수 매핑 테이블 (120) 및 식별자-투-가상 어드레스 매핑 테이블 (122) 를 포함할 수도 있다. HLOS (106) 및 매핑 테이블들 (120 및 122) 은 시스템 (100) 이 실행 어플리케이션 바이너리 코드 (108) 로부터 원하는 또는 타겟 하이-레벨 기능성 정보를 결정할 수도 있는 통합된 플랫폼 메커니즘을 포함한다. 하이-레벨 기능성 정보는 악성 거동을 검출하기 위해 신뢰 구역 (114) 에서 구현되는 알고리즘(들) 및/또는 모델(들)(예를 들어, 악성 코드 검출 알고리즘(들)(116)) 에 의해 사용될 수도 있다.

[0022] 하기에서 더 상세하게 기재되는 바와 같이, 시스템 (100) 은 어플리케이션 바이너리 코드 (108) 를 실행하기 위해 2 개의 상이한 실행 모델들을 지원할 수도 있다. 제 1 실행 모델은 (예를 들어, C/C++ 코드로부터의) 네이티브 바이너리 실행을 수반한다. 제 2 실행 모델은 관리된 런타임 실행 (예를 들어, 가상 머신 (118) 에 의한 실행) 을 수반한다. 실시형태에서, 가상 머신 (118) 은 자바스크립트 소스들로부터 해석된 코드 또는 동적 JIT (just-in-time) 를 실행할 수도 있다. 관리된 런타임 실행 실시형태에서, 가상 머신 (118) 은 바이너리 코드 (108) 내에서 가상 머신 (118) 이 작동하는 바이너리 코드 (108) 의 부분을 포함할 수도 있다.

하지만, 다른 실시형태들에서, 별도의 VM 및 바이너리 작업부하들이 있을 수도 있음을 알아야 한다.

[0023] 네이티브 바이너리 실행 모델의 예시적인 실시형태가 도 2 내지 도 4 에 도시된다. 네이티브 바이너리 실행을 위해, 등록된 어플리케이션들 (112) 의 리스트에서의 각각의 어플리케이션은 HLOS (106) 에 의해 유지되는 대응 VAFMT (120) 를 갖는다. VAFMT (120) 는 신뢰 구역 (114) 에 상주할 수도 있다. VAFMT (120) 는 연관된 하이-레벨 기능성과 매핑되는 관심의 상이한 가상 어드레스들을 포함한다. 실시형태에서, 각각의 연관된 하이-레벨 기능성은 알고리즘(들)(116) 이 이해하는 매크로 이름으로서 나타낼 수도 있다. 하지만, 예를 들어 특정 가상 어드레스에서 검출된 활동이 직접 알고리즘(들)(116) 에서 트리거링되어야 하는 기능성에 대응하도록 알고리즘 (116) 에서의 함수들 또는 함수 이름들에 대한 포인터들을 포함한, 연관된 하이-레벨 기능성을 나타내기 위한 다른 메커니즘들이 구현될 수도 있음을 알아야 한다. 바이너리 이미지에서의 특정 어플리케이션 함수들 (및 함수들 내의 특정 포인트들) 의 가상 어드레스들은 "관심의 포인트들" 로서 지칭될 수도 있다. 실시형태에서, 관심의 가상 어드레스 포인트들은 예를 들어, 민감한 소스들/싱크 루틴들, 위험한 웹 어플리케이션 인터페이스 (API) 들, 특정 웹 기능성, 버퍼들의 시작/종료, 또는 공격자가 악용할 수도 있는 임의의 다른 오브젝트들 또는 알려진 웹/자바스크립트 공격들의 검출 및 분석을 위한 다른 적절한 정보의 시작 지점 또는 종료 지점 내의 포인트들, 또는 그 사이의 다수의 특정 포인트들을 포함할 수도 있다. 다른 실시형태들에서, 관심의 가상 어드레스 포인트들은 자바스크립트 인터프리터, JIT (just-in-time) 컴파일러, 또는 런타임 환경의 구현 (자바스크립트 소스 코드, 바이트코드/JIT코드 등을 저장하는 가상 머신 히프에 대한 할당/할당해제 함수들) 에서의 포인트들을 포함할 수도 있다.

[0024] 도 2 및 도 3 은 VAFMT (120) 의 예시적인 실시형태를 도시한다. 도 2 는 어플리케이션 바이너리 코드 (108) 내의 대응하는 가상 어드레스 포인트들로의 어플리케이션 소스 코드 (110) 내의 소정의 원하는 또는 타겟 기능적 포인트들의 논리적 매핑 (200) 을 도시한다. 도 2 및 도 3 에서, 가상 어드레스들은 나타나 있지만 바이너리 오브젝트 코드는 나타나 있지 않다. 이 실시형태에서, 어플리케이션 소스 코드 (110) 는 "documentWrite" 함수에 대해 C++ 코드를 포함한다. 소스 코드에서의 포인트 (201) 는 바이너리 코드에서의 가상 어드레스 (202) 에 매핑된다. 소스 코드에서의 포인트 (203) 는 바이너리 코드에서의 가상 어드레스 (204) 에 매핑된다. 소스 코드의 포인트 (205) 는 바이너리 코드에서의 가상 어드레스 (206) 에 매핑된다.

도 3 은 가상 어드레스에서의 코드가 나타내는 개개의 기능성 의미들로의 VAFMT 에서의 칼럼 (302) 아래에 있는 바이너리 코드 (202, 204, 및 206) 에서의 그러한 가상 어드레스들의 논리적 매핑 (300) 을 도시한다. 도 3 에 도시된 바와 같이, VAFMT (120) 는 관심의 기능적 포인트 (칼럼 (304) 의 대응 디스크립션을 갖는 복수의 가상 어드레스들 (칼럼 (302)) 을 포함할 수도 있다. 바이너리 코드 포인트에 대해 202 로 나타낸 가상 어드레스 (0x3273fa94) 는 EVAL_FUNCTION 에 대응하는 기능적 포인트에 매핑된다. 바이너리 코드 포인트에 대해 204 로 나타낸 가상 어드레스 (0x3473fac8) 는 DOCUMENT_WRITE_FUNCTION_START 를 나타내는 관심의 기능적 포인트에 대응한다. 바이너리 코드에서 206 으로 나타낸 가상 어드레스 (0x3473fad4) 는 DOCUMENT_WRITE_1 를 의미하는 매크로를 갖는 기능적 포인트에 매핑된다.

[0025] 도 11 은 특정 데이터 구조 유형들 (예를 들어, 클래스, 구조, 공용체) 의 오브젝트들을 포함하는 동적으로 할당된 버퍼들의 시작 및 종료의 가상 어드레스들을 결정하는데 사용될 수도 있는 특정 버퍼 할당기 함수들에 대해 가상 어드레스들을 갖는 커스텀 (custom) 가상 어드레스 테이블을 포함하는 VAFMT (120) 의 실시형태를 도시한다. 버퍼에 할당된 오브젝트들의 멤버들/필드들의 값들은, 관심의 포인트인 특정 필드/멤버에 대해 테이블에서 또한 유지될 수도 있는, 오프셋 및 길이 필드들을 사용하여 결정될 수도 있다. 버퍼 할당 함수들의 가상 어드레스들은, 예를 들어 할당기 함수들의 가상 어드레스들에 의해 커버되는 영역으로부터 시스템 메모리 할당기 함수들의 실행을 추적함으로써, 할당된 버퍼의 사이즈 및 어드레스들을 검출하는데 사용될 수도 있다.

버퍼 시작 및 종료 가상 어드레스들이 일단 알려지면, 오프셋 및 길이 필드들은 특정 데이터 구조 유형에 대한 오브젝트들의 특정 멤버/필드의 값을 결정하는데 사용될 수도 있다.

[0026] 도 1 에서 파선으로 도시된 바와 같이, 어플리케이션 소스 코드 (110) 가 시스템 (100) 에 저장될 필요는 없다. 오히려, 이것은 오프 라인 또는 오프 디바이스에 위치되고 참조 또는 개방 소스 코드로서 이용가능할 수도 있다. 특정 버전의 참조 소스 코드가 브라우저 또는 웹 어플리케이션들의 실제 상업용 바이너리에서 관심의 가상 어드레스들을 결정하기 위한 참조 및 지침으로서 사용될 수 있다. 동등한 바이너리가 개방 소스 프로젝트의 매칭 코드 개정/버전으로부터 컴파일될 수도 있다. 컴파일된 바이너리는 그 버전/개정에 기초하는 어플리케이션 바이너리의 함수들/포인트들 및 원하는 또는 타겟 가상 어드레스들을 검출하는데 참조로서 사용될 수도 있다. 유사한 컴파일러 및 링커 옵션들이 사용될 수도 있다. 또한, 어플리케이션 코드에 있어서 다양한 포인트들에서의 브레이크포인트들은 가상 어드레스들 및 그 기능성 매핑 포인트들의 결정을 위해 사용될

수 있다. 바이너리 코드 인식 및 유사성 추출 방법들은 개방 소스 프로젝트에 대해 알려진 컴파일된 함수들로부터 참조 바이너리를 사용함으로써 주어진 어플리케이션 바이너리에서의 기능성들을 식별하는데 활용될 수도 있다. 약간 수정된 버전들을 갖는 바이너리들 (또는 알려진 참조 개방 소스 프로젝트들과 일부 소스 코드 차이들을 갖는 소스 베이스로부터 유래하는 바이너리들) 에 대해, 테스트 코드들이 중요한 웹 함수들 및 API들을 호출하는 태스크 코드들이 기입될 수도 있다. 다양한 태스크 경우들로부터의 가상 어드레스 액세스 시퀀스들은 타겟 가상 어드레스 포인트들의 세트로 수렴하는데 사용될 수도 있다. 어플리케이션 바이너리 코드로부터 기능성을 추출하는데 다른 메커니즘이 사용될 수도 있다는 것을 알아야 한다.

[0027] 도 4 는 네이티브 바이너리 실행 모델에서 악성 코드 활동을 검출하기 위한 방법의 실시형태를 도시하는 플로우 챠트이다. 블록 (402) 에서, VAFMT (120) 가 어플리케이션에 대해 생성된다. 상술한 바와 같이, VAFMT (120) 는 대응하는 하이-레벨 어플리케이션 기능성에 매핑되는 복수의 관심의 가상 어드레스들을 포함한다. 블록 (404) 에서, 어플리케이션은 예를 들어, 휴대용 컴퓨팅 디바이스와 같은, 컴퓨팅 디바이스 상에 설치될 수도 있다. 블록 (406) 에서, 어플리케이션은 HLOS (106) 에 의해 제공된 보안 지원을 위해 등록될 수도 있다 (예를 들어, 등록된 어플리케이션들 (112)). 블록 (408) 에서, 어플리케이션이 론칭될 수도 있고, 이에 대응하여, CPU (102) 가 어플리케이션 바이너리 코드 (108) 를 실행할 수도 있다. 등록된 어플리케이션 (112) 이 작동할 때, HLOS (106) 는 어플리케이션의 작동 프로세스들을 인터셉트할 수도 있다 (블록 (410)). 블록 (412) 에서, HLOS (106) 는 대응 VAFMT (120) 를 사용하여 관심의 기능적 포인트들을 이들이 실행될 때 검출하고 기록할 수도 있다. 블록 (414) 에서, 기록된 포인트들은 악성 공격들을 검출하고 해결하기 위해 악성 코드 검출 알고리즘(들)(116) 에 제공될 수도 있다. 악성 코드 검출 알고리즘(들)(116) 은 시그니처 기반 알고리즘(들), 패턴 매칭 알고리즘들을 포함하거나 머신 학습, 또는 다른 기법들을 채용할 수도 있다. 이러한 방식으로, 악성 코드 검출 알고리즘(들)(116) 은 VAFMT (120) 를 사용하여 그것이 입력으로서 수신하는 가상 어드레스들의 의미를 제공할 수도 있다.

[0028] VAFMT (120) 가 HLOS (106) 의 제어 하에 있기 때문에, HLOS (106) 에 의해 수행되는 어플리케이션 바이너리 코드 (108) 의 가상 어드레스들의 임의의 변환/랜덤화 (예를 들어, 어드레스 공간 레이아웃 랜덤화 (ASRL)) 가 VAFMT (120) 에서의 가상 어드레스들에 적용되어 실행 어플리케이션의 유효 가상 어드레스와 동기된 상태로 이들을 유지할 수도 있다. 실시형태에서, VAFMT (120) 로 어플리케이션 실행 및 자바스크립트 코드로부터 모아진 정보는, 악성 코드 검출 알고리즘 (116) 에 피드될 수 있는 하이-레벨 웹/자바스크립트 기능성 정보를 제공할 수도 있다. 임의의 악성 거동을 검출하면 (블록 (416)), HLOS (106) 는 어플리케이션/렌더러/자바스크립트 프로세스를 일시 정지하고 사용자에 대해 대화 박스를 개방하여, 잠재적 위험에 관하여 경고하고 사용자에게 진행을 위한 명령들을 요구할 수도 있다. 사용자가 여전히 진행하기를 원하는 경우, 브라우저 프로세스가 HLOS (106) 에 의해 재개될 수도 있다. 사용자가 진행하기를 원하지 않는 경우, HLOS (106) 는 사용자에게 탭을 폐쇄하거나 일부 다른 웹사이트로 내비게티이팅할 것을 요구할 수도 있고, 또는 HLOS (106) 가 그 실행 인스턴스 (브라우저 탭) 에 대한 프로세스를 종료할 수도 있다.

[0029] VAFMT (120) 는 어플리케이션 바이너리 코드 (110) 버전이 변화할 때 예를 들어, 오버-디-에어 (over-the-air; OTA) 업데이트들을 통해 업데이트될 수도 있다. 이러한 업데이트들은 HLOS (106) 가 임의의 등록된 어플리케이션들 (112) 에 대해 업데이트된 바이너리들로 준비하는 것을 보장한다. 업데이트된 바이너리들은 관심의 동일한 포인트들에 대해 새로운 가상 어드레스들을 산출할 수도 있다.

[0030] HLOS (106) 및 매핑 테이블들 (120 및 122) 은 또한, 예를 들어 가상 머신 (118)(도 1) 을 수반하는 관리된 런타임 실행 모델을 지원하도록 구성될 수도 있다는 것을 알아야 한다. 이와 관련하여, 상술한 통합된 플랫폼 메커니즘(들) 은 시스템 (100) 이 실행 어플리케이션 바이너리 코드 (108) 로부터 원하는 또는 타겟 하이-레벨 기능성 정보를 결정하는 것을 가능하게 한다. 관리된 런타임 실행 모델의 예시적인 실시형태가 도 5 내지 도 10 에 도시된다.

[0031] 관리된 런타임 또는 가상 머신 실행을 수반하는 실시형태들에서, 자바스크립트 소스들에 대한 자바스크립트 소스들 및/또는 바이트코드/JIT (just-in-time) 바이너리는 다른 테이블 (예를 들어, 식별자-투-어드레스 매핑 테이블 (IVAMT)(122)) 의 도움으로 가상 머신 (VM) 히프의 상이한 부분들로부터 판독될 수도 있다. IVAMT (122) 는 VM 히프의 중요한 경계들에 대해 가상 메모리 어드레스들을 포함한다. 또한, 어플리케이션 바이너리 (108) 또는 가상 머신 (118) 의 다양한 기능적 포인트들에 대한 가상 어드레스들이 유지될 수 있는 엔트리들의 다른 유형들을 포함할 수도 있다. IVAMT (122) 는 일반적으로 어플리케이션 실행 동안 동적으로 업데이트되고 및/또는 결정될 수도 있는 특정 기능적 포인트들에 대한 가상 어드레스들을 위해 사용될 수도 있다. 이와 관련하여, IVAMT (122) 는 기능적 포인트를 가상 어드레스에 매핑할 수도 있다. 다른 한편으로,

VAFMT (120) 는 정적으로 정의된 가상 어드레스를 기능성 의미에 매핑할 수도 있다. 이에 따라, VAFMT (120) 는 어플리케이션 실행 동안 변화하지 않을 수도 있지만, 예를 들어 컴퓨팅 디바이스에 대한 오버-디-에어 (OTA) 업데이트들에 의해 업데이트될 수도 있다. 다른 여러 가지 종류의 테이블들이 VAFMT (120) 및 IVAMT (122) 와 연관될 수도 있다는 것을 또한 알아야 한다. 여러 가지 종류의 테이블들은 가상 어드레스들이 아닌 그들의 파라미터 값들 또는 설정들에 매핑된 다양한 매크로 또는 파라미터 이름들을 포함할 수도 있다.

[0032] 도 9 의 실시형태에서, 가상 메모리 어드레스들 (901) 은 예시적인 VM 히프 구조 (900) 의 다양한 외부 및/또는 내부 경계들에 대해 식별된다. 도 9 에 도시된 바와 같이, VM 히프 구조 (900) 는, 예를 들어 필드로부터 (912), 필드로 (914), 코드 필드 (902), 맵 필드 (904), 큰 오브젝트 필드 (906), 구 (old) 데이터 필드 (908), 및 구 포인터 필드들 (910) 을 포함한, 다양한 내부 및/또는 외부 경계들을 식별하는 복수의 데이터 필드들을 포함할 수도 있다. VM 히프는 네이티브 시스템 히프에 할당되는 VM 관리된 메모리 영역이다. 종래에 알려진 바와 같이, VM 히프에서, VM 은 예를 들어, 메모리 관리, 코드 (예를 들어, 자바스크립트 소스) 의 할당 및 할당해제, 바이트코드, 중간 코드, JITed 바이너리, 실행 동안 생성된 오브젝트들, 및 프로그램 (예를 들어, 자바스크립트 프로그램) 의 실행을 위해 사용된 모든 다른 연관된 하우스키핑 정보 및 내부 데이터 구조들의 추상화를 수행한다. 도 9 에 추가로 도시된 바와 같이, VM 히프 영역은 VM 이 저장하는 것들의 유형에 의존하여 다양한 서브 영역들 (예를 들어, 910, 908, 906, 904, 902, 912, 및 914) 를 포함할 수도 있다. 서브 영역들 (912 및 914) 은 처음으로 생성된 오브젝트들을 포함하는데 사용될 수도 있고 임의의 가비지 수집 활동은 서브 영역들 (912 내지 914) 로부터 라이브 오브젝트들을 스왑하며 그 역 또한 마찬가지이다. 실시형태에서, 서브 영역 (902) 은 자바스크립트 소스, 바이트코드들, 중간 코드들, 및 JITed 바이너리/어셈블리 코드들을 저장하는데 사용될 수도 있다. 서브 영역 (904) 은 프로그램 (예를 들어, 자바스크립트 프로그램) 의 실행 동안 VM 에 의해 생성된 오브젝트들과 연관된 소정의 중간 데이터 구조들을 유지하는데 사용될 수도 있다. 서브 영역 (906) 은 미리결정된 사이즈 (예를 들어, 1 MB) 보다 큰 임의의 종류의 아이템 (코드, 오브젝트) 를 유지하는데 사용될 수도 있다. 서브 영역들 (908 및 910) 은 상수 값들로 오브젝트들을 포커싱하는 서브 영역 (908) 및 다른 오브젝트들을 가리키는 오브젝트들 상에서 포커싱하는 서브 영역 (910) 으로 가비지 수집의 다수의 사이클들을 존속시킨 데이터 및 오브젝트들을 유지할 수도 있다.

[0033] 동작에서, HLOS (106) 는 VM 히프에 대해 메모리 할당들이 변화함에 따라 IVAMT (122) 에서의 가상 메모리 어드레스들 (901) 을 식별하고 동적으로 업데이트할 수도 있다. 자바스크립트 가상 머신 (118) 은 함수가 활성일 때까지 소스들을 히프로 유지한다는 것을 알아야 한다. 관리된 런타임 또는 가상 머신 실행 모델은 VM 히프로부터 자비스크립트 소스들 및/또는 바이트코드/JIT 코드를 식별하는 것을 수반할 수도 있다. 자바스크립트 소스들을 홀딩하는 VM 히프 오브젝트들은 임의의 새로운 기입들에 대해 추적될 수도 있고, 가상 머신 (118) 에 의해 수신된 새로운 자바스크립트 소스들이 식별될 수도 있다. 식별된 자바스크립트 소스들은, 자바스크립트 코드로부터 다양한 피처들을 추출하고 임의의 악성 거동을 검출하기 위해 이들을 사용하는, 신뢰 구역 (114) 에서 알고리즘(들)(116) 에 제공될 수도 있다. 자바스크립트 코드로부터 추출된 피처들의 예들은 다음의 또는 다른 피처들을 포함한다: 문서 오브젝트 모델 (DOM) 수정 및 민감한 함수들; 다수의 평가들; 다수의 문자열들; 스크립트 길이; 문자열 수정 함수(들); 난독화 해제 (de-obfuscation) 를 위한 "내장들 (built-ins)" 등. 신뢰 구역 (115) 은 임의의 악성 활동을 결정하기 위해 악성 코드 검출 알고리즘 (116) 에 추출된 피처들을 피드할 수도 있다.

[0034] 소정의 실시형태들에서, JIT 바이너리/바이트코드들만이 이용가능할 때, 이들로부터 피처들이 추출되고 그 후 악성 코드 검출 알고리즘 (116) 에 전송될 수도 있다. 예를 들어, HLOS (106) 는 하이-레벨 자바스크립트 아티팩트들을 나타내는 바이트코드/JIT 코드 시퀀스들의 라이브러리를 유지할 수도 있다. 이러한 아티팩트들과 VM 코드 공간에서의 자바스크립트 함수들로부터의 바이트코드/JIT 코드 스트림의 임의의 매치들은 악성 특징들의 결정을 위해 악성 코드 검출 알고리즘 (116) 에 기록되고 패스될 수도 있다.

[0035] 도 5 및 도 6 은 관리된 런타임 또는 가상 머신 실행 동안 사용된 IVAMT (122) 및 VAFMT (120) 의 예시적인 실시형태를 도시한다. 도 5 은 대응하는 어플리케이션 바이너리 코드 (108) 로의 VM 코드 공간의 할당에 관련된 타겟 기능성의 논리적 매핑 (500) 을 도시한다. 이 실시형태에서, 어플리케이션 소스 코드 (110) 는 "AllocateVMCodeSpace" 함수에 대한 코드를 포함한다. 도 5 에 도시된 바와 같이, 소스 코드 (110) 에서의 제 1 포인트는 바이너리 코드 (108) 에서의 가상 어드레스들 (502) 에 매핑될 수도 있다. 소스 코드 (110) 에서의 제 2 포인트는 바이너리 코드 (108) 에서의 가상 어드레스 (504) 에 매핑될 수도 있다. 예시의 구현에서, 함수 AllocateVMCodeSpace 는, 실행 동안 VM 이 현재 VM 히프 코드 공간 (902) 에 많은 공간이 있지 않은 것이 결정되고 그것이 실행하여야 하는 새로운 자바스크립트 소스 코드를 얻을 때 호출될 수도 있다. 이

러한 함수는 새로운 자바스크립트 코드의 사이즈를 취하고 VM 히프 코드 공간이 사이즈에서 증가되어야 하는 양을 결정할 수도 있어서 VM 이 자바스크립트 소스, 연관된 바이트코드 또는 중간 코드 및/또는 JITed 바이너리를 저장할 수 있다. 결정된 사이즈에 기초하여, AllocateVMCodeSpace 함수는 mmap(), malloc(), calloc(), 또는 realloc() 와 같은 시스템 할당기 함수들을 사용하여 네이티브 플랫폼의 히프에서 VM 히프 코드 공간의 할당된 공간을 증가시킬 수도 있다. mmap() 함수는 메모리 내의 파일 디스크립터에 의해 특정된 다른 오브젝트로부터의 오프셋에서, 바람직하게 어드레스 시작에서 시작하는 바이트들의 시퀀스를 매핑하는 POSIX 준수 유닉스 시스템 호출 (compliant Unix system call) 이다. mmap() 함수는 오브젝트가 매핑되는 실제 장소를 리턴한다. Malloc(), realloc(), calloc() 및 free() 는 C/C++ 프로그래밍 언어에서 동적 메모리 할당에 대해 수동 메모리 관리를 수행하기 위해 C 표준 라이브러리에 함수들의 그룹을 포함한다. 바이너리 코드 (108) 에서의 관심의 포인트들에 대한 가상 어드레스들 (502 및 504) 은 VAFMT (120) 에서의 칼럼 (302) 에 직접 배치될 수도 있다. 가상 어드레스들에 의해 나타낸 관심들의 상이한 포인트들의 기능성 의미들은 VAFMT (120) 의 칼럼 (304) 에서 매크로 이름들로 리스트될 수도 있다. 검출 알고리즘(들)(116)(도 1) 은 VAFMT (120) 의 칼럼 (304) 에서 매크로들에 의해 나타낸 기능성을 명확히 이해할 수도 있다. VAMFT (120) 에서 특정 로우에 대한 (칼럼 (304) 에서의) 매크로 이름은 프로세서 (예를 들어, CPU (102)) 가 (칼럼 (302) 에서의) 그 가상 어드레스 포인트에서 어플리케이션의 바이너리 명령을 실행할 때 실행되고 있는 기능성을 명백하게 식별할 수도 있다. 이러한 방식으로, 관심의 포인트들에 대한 가상 어드레스들의 실행 통계, 카운트들 및 프로파일을 아는 것에 의해, 검출 알고리즘(들)(116) 은 하이-레벨 어플리케이션 바이너리에 의해 실행되고 있는 기능성을 충분히 이해한다. 매핑은 가상 어드레스 (302) 와 매크로 (304) 로 나타내고 프로세싱 또는 검출을 수행하는 검출 알고리즘(들)(116) 에 의해 이해되는 기능적 의미 사이에 직접 있을 수도 있으며, 이로써 관심의 그 가상 어드레스 포인트에서 실제 바이너리 명령을 알아야 할 필요가 없다는 것을 알아야 한다.

[0036] 가상 어드레스들 및 매크로 의미들로 표현된 관심의 포인트들은 오프라인 결정될 수도 있고 그 후 특정 어플리케이션 바이너리에 대해 VAFMT (120) 에 파퓰레이트될 수도 있다. 많은 유형의 어플리케이션들이 가용 매칭 참조 소스 코드를 가질 수도 있다. 예를 들어, 매칭 참조 소스 코드는 파퓰러 개방 소스 프로젝트들 (예를 들어, blink/Chromium 기반 브라우저들, Webkit 기반 브라우저들, Android 플랫폼들에서의 다양한 가상 머신들, 예컨대 Dalvik, ART, RenderScript) 로부터 개발된 일반적으로 가용 어플리케이션들에 대해 이용가능할 수도 있다. 가용 매칭 참조 소스 코드를 갖는 어플리케이션들에 대해, 다양한 오프라인 메커니즘이 그러한 관심의 포인트들에 대한 소스 코드에서 대응 표현/스테이트먼트에 대한 상업용 어플리케이션 바이너리에서의 관심의 포인트들에 대해 가상 어드레스를 결정하는데 사용될 수도 있다.

[0037] 관심의 포인트들에 대한 가상 어드레스들의 오프라인 결정을 위한 예시적인 실시형태가 기재될 것이다. 관심의 기능성들을 구현하는 소스 코드 (110) 에서의 소정의 중요하고 유용한 함수들은 매칭 참조 소스 코드에서 식별될 수도 있다. 소스 코드 (110) 내의 다양한 포인트들은 특정 고유 기능성을 함께 나타내게 되는 포인트들의 고유 세트를 형성하기 위해 수동으로 결정될 수도 있다. 이것은 기능성에 대해 완전한 소스 코드 (110) 의 전반적인 기능성을 고유하게 나타내는 소스 코드 (110) 내의 샘플 포인트들의 세트와 동등할 수도 있다는 것을 알아야 한다. 소스 코드 (110) 는 실제 상업용 제 3 자 어플리케이션과 동등한 참조 어플리케이션에 컴파일되고, 어셈블리되며 링크될 수도 있다. 양자의 바이너리들 (참조 및 상업용 제 3 자) 는 동일한 소스 코드 (110) 로부터 유래하고 유사한 구축 기법들 (예를 들어, 컴파일, 어셈블, 링크) 및 툴체인들을 사용할 수도 있다. 종래에 알려진 바와 같이, 개방 소스 어플리케이션들은 가용 GCC 또는 LLVM 툴체인들을 자유롭게 사용할 수도 있다. 컴파일러, 어셈블러, 및 링커 툴들은 참조 바이너리 어플리케이션을 생성하기 위해 사용될 수도 있고 소스 코드에서의 중요한 포인트들에 대응하는 가상 어드레스 포인트들이 주목될 수도 있다.

관심의 포인트들에 대한 가상 어드레스들은 바이너리 어플리케이션이 구축 (컴파일, 어셈블, 링크) 되는 소스 코드 (110) 에서 관심의 포인트들의 직접 매핑을 포함할 수도 있기 때문에, 참조 바이너리가 상업용 제 3 자 바이너리에서 관심의 가상 어드레스 포인트들을 식별하기 위해 상업용 바이너리와 비교하도록 오프라인 사용될 수도 있다. 추가로 다른 오프라인 또는 다른 기법들이 상업용 제 3 자 바이너리에서 관심의 포인트들에 대해 가상 어드레스를 결정하는데 사용될 수도 있음을 알아야 한다. 실시형태에서, 도 2 는 소스 코드 (110) 에서 관심의 상이한 포인트들 (201, 203, 205) 이 바이너리 (108) 에서 대응하는 가상 어드레스들 (202, 204, 206) 에 어떻게 직접 매핑될 수도 있는지를 나타낸다.

[0038] 도 6 은 도 5 의 VAFMT (120) 와 예시적인 IVAMT (122) 사이의 논리적 매핑 (600) 을 도시한다. VAFMT (120) 는 실행이 관심의 것이고 추적되고 있는 바이너리 어플리케이션에서의 관심의 고정되고 알려진 포인트들의 가상 어드레스들을 포함한다. 이러한 가상 어드레스들은 바이너리 어플리케이션이 변화할 때마다 업데이트될 수도 있다. IVAMT (122) 는 동적일 수도 있고 동적 아이템들 (예를 들어, 런타임 버퍼 시작 또는 종료

포인트들) 의 가상 어드레스들을 나타낼 수도 있는, 바이너리 어플리케이션이 실행할 때 생성되고 업데이트되는 특정 포인트들의 가상 어드레스들을 포함한다. VAFMT (120) 의 좌측 칼럼 (302) 은 가상 어드레스들을 포함하고, 우측 칼럼 (304) 은 그 가상 어드레스 포인트에서 바이너리 코드 (108) 에 존재하는 기능성 디스크립션을 표시할 수도 있다. 이러한 방식으로, VAMFT (120) 는 가상 어드레스를 기능성 의미들에 매핑한다. 일반적으로, IVAMT (122) 는 반대를 포함한다. 이 경우, 기능적 의미 또는 매크로 이름은 알려져 있고, 시스템은 기능적 의미 또는 매크로 이름 (604) 이 바이너리 어플리케이션의 실행 인스턴스에서 구현되거나 이용가능한 가상 어드레스 (602) 를 결정한다. IVAMT (122) 에서의 가상 어드레스들은 런타임에서 결정되는 동적 값들을 포함할 수도 있다. 동적으로 할당된 버퍼 (또는 가상 머신 히프 또는 그 서브 공간들) 의 시작 및 종료가 결정되는 경우에 대해, 동적 버퍼/히프-공간 할당을 행하고 있는 바이너리 어플리케이션에서의 함수들 내에서 관심의 포인트들에 대한 가상 어드레스들은 VAFMT (120) 로부터 획득될 수도 있다. 이러한 함수들의 실행은 VAMFT (120) 에서 가상 어드레스들의 실행을 검출함으로써 결정될 수도 있다. 게다가, 버퍼/히프-공간 할당의 시작/종료 가상 어드레스들은 이러한 함수들로부터 호출된 시스템 메모리 할당 함수들을 검출함으로써 결정될 수도 있다. 버퍼/히프-공간 할당들의 이러한 결정된 시작/종료 가상 어드레스들은 IVAMT (122) 에서 업데이트될 수도 있다.

[0039] 도 7 은 VM 히프 코드 공간에 대한 가비지 수집의 영향 및 가상 머신 (118) 의 가비지 수집 활동의 존재에서 자바스크립트 소스들이 어떻게 일관되게 결정되는지를 나타낸다. 가비지 수집은, 새로운 오브젝트들 및 데드 (dead)(즉, 사용하지 못하는) 오브젝트들의 할당해제가 런타임 또는 가상 머신 (118) 에 의해 명시적으로 핸들링될 수도 있기 때문에 관리된 런타임 또는 가상 머신의 통합 활동인 것임을 알아야 한다. 관리된 VM 히프로부터 데드 (사용되지 않는) 오브젝트들을 회수하는 (reclaiming) 활동은 가비지 수집으로서 지칭된다. 이와 관련하여, 불필요한 스크립트 오브젝트들 또는 다른 오브젝트들이 회수될 때, VM 히프는 재조직화될 수도 있고 기존 객체들은 새로운 오브젝트 할당들을 위한 공간을 만들기 위해 그 주위로 이동되고 압축될 수도 있다.

도 7 은 VM 히프 코드 공간 (704a) 에 대한 그러한 가비지 수집 활동의 영향을 나타낸다. VM 히프 공간 (704a) 은 자바스트립트 오브젝트들 (JS1, JS2, JS3, 및 JS4) 를 포함한다. 가비지 수집 이벤트 후, 이들은 가비지 수집기에 의해 불필요하거나 데드로서 검출되었던 자바스크립트 오브젝트 (JS3) 의 제거로 압축되고, 이에 따라 VM 히프 코드 공간 (704b) 로부터 회수 (삭제) 될 수도 있다. 하지만, VM 히프에서의 오브젝트들의 임의의 이러한 이동 (예를 들어, 제거, 압축 등) 은 자바스크립트 오브젝트가 어디에 상주하는지를 결정하는 가상 어드레스 시작 및 종료 위치들을 변화시킨다. 예시적인 방법에서, 가상 어드레스들은 매 가비지 수집 활동 후 히프 내의 다양한 공간들 (도 9) 및 VM 히프에 대해 도 5 및 도 6 에 도시된 가상 어드레스 결정 메커니즘을 재실행하는 것에 의해 변화될 수도 있으며, 이로써 가비지 수집 동안 스크립트 오브젝트가 이동되었으면 가상 어드레스들을 새로운 값들로 업데이트한다. 도 8 에 도시된 바와 같이, 커널은 가비지 수집 동안 발생하는 오브젝트 이동들 및 이들이 이동하는 거리를 계속 추적할 수도 있다. 오브젝트들이 이동한 어드레스 오프셋을 계속 추적함으로써, VM 히프 코드 공간에서의 자바스크립트 오브젝트의 시작 및 종료에 대한 가상 어드레스 값들이 업데이트될 수도 있다. 유사한 방식으로, VM 히프의 다양한 코드 공간들에 대해 IVAMT (122) 에서의 가상 어드레스는 도 9 에 도시된 VM 히프의 다양한 서브 공간들의 할당들/할당해제들/이동들을 추적함으로써 업데이트될 수도 있다.

[0040] 도 10 은 관리된 런타임 또는 가상 머신 실행 모델에서 악성 코드 활동을 검출하기 위한 방법 (1000) 의 실시형태를 도시하는 플로우챠트이다. 도 10 에서 블록들 (1002, 1004, 1006, 1008, 및 1010) 에 나타낸 단계들 또는 기능성은 일반적으로 도 4 의 방법과 관련하여 위에 기재된 블록들 (402, 404, 406, 408 및 410) 에 대응할 수도 있음을 알아야 한다. 블록 (1012) 에서, 방법 (1000) 은 실행될 때 VM 히프 할당기/할당해제기 함수들에 대해 관심의 가상 어드레스들의 포인트들을 검출한다. 블록 (1014) 에 도시된 바와 같이, 실행이 VM 히프 할당기/할당해제기 함수 내측에 있는 것으로 검출될 때, 방법 (1000) 은 커널의 시스템 할당기/할당해제기 함수로의 진입 VM 을 검출하고 시스템 메모리 할당/할당해제를 기록할 수도 있다. 이에 기초하여, 방법 (1000) 은 VM 의 히프의 시작/종료 가상 어드레스들을 산출하고 결정할 수도 있다. VM 히프에 대한 특정 할당 영역 (예를 들어, 코드 공간, 큰 오브젝트 공간 등) 에 대해 유사한 메커니즘(들)을 구현함으로써, VM 히프 내의 특정 서브 영역들 (예를 들어, 코드 공간, 큰 오브젝트 공간 등) 에 대한 시작/종료 가상 어드레스들이 결정될 수도 있다. 블록 (1016) 에 도시된 바와 같이, 블록 (1014) 에서, 자바스크립트 소스 코드 오브젝트들을 저장하는데 사용된 VM 히프 공간이 일단 결정되면, 방법 (1000) 은 VM 히프 내의 자바스크립트 오브젝트의 시작을 결정하기 위해 (바이너리로) 스크립트 오브젝트 헤더 시그니처/패턴을 사용할 수도 있다. 자바스크립트 오브젝트의 길이는 헤더로부터 추출될 수도 있고 전체 자바스크립트 소스 코드를 추출하는데 사용될 수도 있다. 블록 (1018) 에 도시된 바와 같이, 자바스크립트 소스 코드는 예를 들어, 악성 거동을 검출하기 위해

검출 알고리즘(들)(116) 에 의해 사용된 관심의 특정 피처들을 추출하는데 사용될 수도 있다. 블록 (1020) 에서, 자바스크립트 코드의 악성 거동은 예를 들어, 블록 (1018) 에서 자바스크립트 소스로부터 추출된 피처들에 기초하여 결정될 수도 있다.

[0041] 위에 언급된 바와 같이, VAFMT (120) 은 초기에 오프 라인 방식으로 구성되고 컴퓨팅 시스템 (100)(도 1) 에 제공될 수도 있다. 일 실시형태에서, 어플리케이션 바이너리 코드 (108) 의 새로운 버전이 컴퓨팅 시스템 (100) 에 이용가능하게 될 때, VAFMT (120) 는 유사하게 오프라인 방식으로 업데이트되고 예를 들어, 통신 네트워크 ("오버-디-에어 (OTA) 업데이트" 로 지칭됨) 를 통해, 컴퓨팅 시스템 (100) 에 제공될 수도 있다. 이러한 방식으로 VAFMT (120) 를 업데이트하는 것은 빈번하게 업데이트되는 바이너리 어플리케이션들에 대해 단점일 수도 있다. 어플리케이션 바이너리 코드 (108) 의 업데이트된 버전에서의 바이너리 코드의 상대적으로 큰 부분은 변화되지 않은 채로 유지될 수도 있음을 알아야 한다. VAFMT (120) 에서 식별된 관심의 기능적 포인트들 (304) 은 버전-투-버전으로부터 변경되지 않을 수도 있는 바이너리 코드 및/또는 어플리케이션 바이너리 코드 (108) 의 상대적으로 제한된 부분을 포함할 수도 있다.

[0042] 예를 들어, 컴파일러 동작들 및/또는 설정들은 자주 변경되지 않을 수도 있고 바이너리 코드에서의 다양한 모듈들은 모듈들 사이에서 유사한 또는 미리결정된 오프셋들을 유지할 수도 있다. 도 12 내지 도 16 은 어플리케이션 바이너리 코드 (108) 의 새로운 또는 업데이트된 버전이 설치될 때 VAFMT (120) 에서 가상 어드레스들을 자동으로 업데이트하기 위해 컴퓨팅 시스템 (100) 에서 구현될 수도 있는 다양한 메커니즘들을 도시한다.

[0043] 이러한 메커니즘들은 어플리케이션들의 다양한 유형들 및/또는 사용 경우들에 대해 VAFMT (120) 의 OTA 업데이트들을 위한 필요성을 감소시킬 수도 있다는 것을 알아야 한다. 예를 들어, 웹 보안 어플리케이션들의 맥락에서, 이러한 메커니즘들은 동일한 원래 코드베이스에 기초하는 웹 브라우저 어플리케이션들에 대한 많은 가장 빈번한 업데이트들에 대해 OTA 업데이트들에 대한 필요성을 없앨 수도 있다. 기존 웹 브라우저 어플리케이션들은 주간 또는 월간 단위로 바이너리 어플리케이션 코드를 업데이트할 수도 있다. 새로운 바이너리 버전을 위한 가상 어드레스들은 관심의 기능적 포인트들 (304) 에 관련된 특정 모듈들에 대해 소스 코드가 변경되지 않을 때에도 변화할 수도 있다. 이 경우, 가상 어드레스들은 관심의 기능적 포인트들 (304) 이외의 어플리케이션의 부분들에서의 소스 코드 변화들 또는 어플리케이션의 다른 부분들에 액세스된 변수 유형들 및 데이터 구조 유형들 (예를 들어, C++ 클래스들, C-구조들, 공용체들 등) 에서의 변화들이 있는 경우 변화할 수도 있다. 또한, 컴파일러, 어셈블러, 및 링커 옵션들에서의 소정 종류의 변화들은 어플리케이션의 다른 부분들에서 가상 변화들을 초래할 수도 있다.

[0044] 도 12 는 어플리케이션 바이너리 코드 (108) 의 새로운 또는 업데이트된 버전이 설치될 때 VAFMT (120) 를 자동으로 업데이트하기 위해 컴퓨팅 시스템 (100) 에서 구현될 수도 있는 예시적인 메커니즘들의 실시형태를 도시한다. 도 12 에 도시된 바와 같이, VAFMT (120) 는 메타데이터 (1200) 및 하나 이상의 의사 바이너리 코드 템플릿들 (1202) 로 보충될 수도 있다. 하기에서 더 상세하게 기재되는 바와 같이, 메타데이터 (1200) 및 의사 바이너리 코드 템플릿들 (1202) 은 HLOS (106) 가 어플리케이션 바이너리 코드 (108) 가 새로운 버전으로 업데이트될 때 관심의 기능적 포인트들에 대해 새로운 가상 어드레스들 (302) 을 결정하는 것을 가능하게 할 수도 있다.

[0045] 의사 바이너리 코드 템플릿 (1202) 은 로컬 변수들에 대한 의사 레지스터들 및 메모리에서의 저장 위치들에 대한 기호 표현을 사용하여 동작 스테이트먼트들의 시퀀스를 포함하는 것을 알아야 한다. 의사 바이너리 코드 템플릿 (1202) 은 그 목적을 표시하는 의사 레지스터들의 다양한 카테고리들을 사용할 수도 있다. 일 실시형태에서, ArgumentReg# 는 서브루틴들에 인수 (argument) 들을 패스하는 의사 레지스터들을 나타낼 수도 있다. ReturnReg 는 서브루틴 호출로부터 다시 리턴하는 경우 리턴 어드레스를 포함할 수도 있다. ProgCounter 는 프로세서의 프로그램 카운터에 의해 포인트되는 현재 어드레스를 포함할 수도 있다. ReturnValueReg# 는 서브루틴 호출들로부터 값들을 호출기 코드에게 다시 리턴하는데 사용된 레지스터들을 나타낼 수도 있다. 동작들은 변수들 또는 저장 위치들일 수 있는 입력들 및 출력들로 프로세서에서의 어셈블리 동작들의 면밀한 표현들을 포함할 수도 있다. 예를 들어, AddWord 변수는 4-바이트 또는 1-워드 사이즈의 피연산자의 가산 동작을 나타낼 수도 있다. LoadWord 변수는 미리결정된 사이즈 (예를 들어, 4 바이트 또는 1 워드) 인 메모리로부터의 값을 로딩하는 것을 표시할 수도 있다. LoadByte 변수는 미리결정된 사이즈 (예를 들어, 1 바이트) 인 메모리로부터의 값을 로딩하는 것을 표시할 수도 있다. branchEQ 는 이전 비교 동작이 피연산자들의 동일성을 초래하는 경우 피연산자로서 제공된 타겟에 분기하는 조건부 분기를 포함할 수도 있다. 어드레싱 모드들 또는 어드레스 산출은 로드 또는 저장 동작들로부터 분리될 수도 있다. 일 실시형태에서, 오프셋 및 베이스 레지스터에 의한 로드 동작은 2 개의 동작들로 분할될 수도 있다: 의사 레지스터에 일정 오프셋

값을 가산함으로써 최종 어드레스를 산출하는 가산 동작 다음, 산출된 최종 어드레스를 포함하는 의사 레지스터를 사용하는 실제 로드 동작. 이것은 다양한 형태들의 어드레싱 모드들이 업데이트된 어플리케이션 바이너리에 의해 사용될 수 있기 때문에 가장 일반적인 형태로 표현을 유지하도록 행해질 수도 있다. 상수들인 동작 인수들은 상수들의 유효 범위를 인코딩하는데 필요한 비트들의 수로 나타낼 수도 있다.

[0046] 예를 들어, 상수 "Const8bits" 는 피연산자가 8 비트로 인코딩될 수 있는 임의의 유효 값인 것을 표시하는 동작을 위한 피연산자로서 사용되고, 이에 따라 허용된 값들의 유효 동적 범위를 결정할 수도 있다. 일부 피연산자들은 하드-코딩된 상수들 (예를 들어, 값 '8' 을 표시하는 "#8") 일 수도 있다. 직접 분기 동작의 피연산자들은 현재 프로그램 카운터 (예를 들어, ("ProgCounter + #Const20bits" 또는 "ProgCounter + #12")) 로부터의 오프셋으로서 나타낼 수도 있다. 의사 바이너리 코드 템플릿 (1202) 은 이들 또는 다른 동작 스테이트먼트들을 사용하여 관심의 기능성을 구현할 수도 있다. 동작 스테이트먼트들은 예를 들어, 매칭 기능성 또는 모듈을 통해 정확한 기능성을 구현하는 새로운 업데이트된 바이너리에서의 영역을 식별하는데 사용될 수도 있다. 매칭 모듈은 어플리케이션의 실제 바이너리 및 의사 바이너리 코드 템플릿 (1202) 의 표현 및 포맷 양자 모두를 이해하도록 구성되는 것을 알아야 한다. 매칭 모듈은 매칭을 검출하기 위해 동작들의 윈도우 내에서 동작 별 비교를 수행하거나, 비교를 위해 제어-데이터-플로우 및 제어-데이터-플로우 영역 내의 동작들을 사용할 수도 있다.

[0047] 다양한 매칭 기법들이 사용될 수도 있다. 의사 바이너리 코드 템플릿 (1202) 에서의 동작 스테이트먼트들은 정적 단일 할당 (Static Single Assignment; SSA) 표현을 사용할 수도 있으며, 여기서 특정 의사 레지스터 변수는 한번만 할당됨으로써, 동작 스테이트먼트들 사이에서 진정한 의존성을 드러낸다. SSA 표현은 어플리케이션의 업데이트된 바이너리에서 기능성 영역의 개선된 매칭을 가능하게 할 수도 있다. 용어 "의사" 는 표현이 바이너리 실행가능물이 아니고 프로세서의 실제 어셈블리 명령들, 레지스터들 및 어드레싱 모드들을 사용하지 않으며 바이너리 코드로 어셈블리되지 않는 사실을 지칭한다. 의사 바이너리 코드 템플릿 (1202) 는 어플리케이션의 업데이트된 바이너리에서 관심의 기능성을 검출하기 위해 매칭 모듈이 템플릿 패턴 및 가이드라인으로서 사용하는 기능성 참조를 제공한다. 의사 바이너리 코드 템플릿 (1202) 의 실제 포맷 및 표현은 구현에 의존하고 다양한 다른 대안들이 사용될 수 있음을 알아야 한다. 다른 실시형태들에서, 일부 구현들은 바이너리 어플리케이션이 실행하는 CPU (102) 에 대해 어셈블리 표현을 리셈블하는 실제 어셈블리 명령 표현 또는 표현(들) 을 사용할 수도 있다.

[0048] 상술한 바와 같이, HLOS (106) 는 등록된 어플리케이션들의 리스트 (112) 를 유지할 수도 있다. 각각의 등록된 어플리케이션에 대해, HLOS (106) 는 관심의 기능적 포인트들 (304) 에 대해 가상 어드레스들 (302) 을 포함하는 테이블들 (예를 들어, VAFMT (120), IVAMT (122)) 을 유지한다. 도 12 에 도시된 바와 같이, VAFMT (120) 에서의 하나 이상의 가상 어드레스들은 의사 바이너리 코드 템플릿 (1202) 과 연관될 수도 있다. 도 12 의 실시형태에서, 의사 바이너리 코드 템플릿 (1202) 은 고유 기능성 (documentWrite 함수) 를 나타내는 관심의 기능적 포인트들 (304) 의 특정 세트에 대해 가상 어드레스들 (302) 의 세트와 연관된다. 의사 바이너리 코드 템플릿 (1202) 은 documentWrite 함수를 커버하는 바이너리 코드와 일반적으로 동등한 의사 코드 명령(들) 을 포함한다. 일 실시형태에서, 의사 바이너리 코드 템플릿 (1202) 은 프로세서 명령 세트 아키텍처 (ISA) 를 사용하지 않을 수도 있고 실제 바이너리 코드로 어셈블링될 필요가 없을 수도 있다. 의사 바이너리 코드 템플릿 (1202) 은 어셈블리 동작들과 유사한 동작 스테이트먼트들을 사용하고 스토리지들에 대해 의사 레지스터들 및 기호 참조들을 사용할 수도 있다. 이러한 동작 스테이트먼트들의 시퀀스의 사용을 통해, 의사 바이너리 코드 템플릿 (1202) 은, 어플리케이션의 실제 바이너리에서 구현되는 관심의 기능성 (예를 들어, documentWrite 함수) 과 동일하거나 동등한 것을 나타내는 관심의 기능성 (예를 들어, 위의 예에서의 "documentWrite" 함수의 기능성) 을 구현할 수도 있다. 컴퓨팅 시스템 (100) 은 임의의 수의 의사 바이너리 코드 템플릿들 (1202) 을 포함할 수도 있다는 것을 알아야 한다. 상이한 의사 바이너리 코드 템플릿들 (1202) 의 수는, VAFMT (120) 에서 캡처된 모든 상이한 기능성들이, 관심의 기능적 포인트들의 상이한 세트들을 통해, 새로운 어플리케이션 바이너리 코드가 설치될 때 그것이 커버하는 함수 포인트들에 대한 가상 어드레스들을 업데이트하기 위해 사용되는 적어도 하나의 대표적인 의사 바이너리 코드 템플릿 (1202) 을 갖는 것일 수도 있다.

[0049] 일 실시형태에서, 의사 바이너리 코드 템플릿 (1202) 는 타겟 어셈블리 명령(들) 의 일반적인 형태, 하나 이상의 의사 레지스터들, 및 메모리에서 특정 참조 포인트들을 나타내는 일반적인 베이스 (예를 들어, 글로벌 히프 또는 스택, 심볼/변수 이름) 로부터의 메모리 오프셋들을 포함할 수도 있다. 메타데이터 (1200) 는 일반적으로, 예를 들어 바이트 오프셋을 사용한 가상-어드레스 프리 표현을 포함한다. 가상 어드레스

(0x3473fac8) 에 대한 메타데이터 (1200) 는 바이트 오프셋 (BASE2 = BASE0 + 74709704) 을 포함한다. 가상 어드레스 (0x3473fad4) 에 대한 메타데이터 (1200) 는 바이트 오프셋 (BASE2 + 12) 을 포함한다. 가상 어드레스 (0x3473fae8) 에 대한 메타데이터 (1200) 는 바이트 오프셋 (BASE2 + 32) 을 포함한다. 이러한 메타데이터는 "document_write" 기능성을 고유하게 나타내는 관심의 3 개의 가상 어드레스 포인트들의 세트에 대응하는 고유 세트를 형성할 수도 있다는 것을 알아야 한다.

[0050] 의사 바이너리 코드 템플릿들 (1202) 은 초기에 오프라인 방식으로 생성되고, 컴퓨팅 시스템 (100) 에 제공되며, 디바이스의 보안 스토리지에 저장될 수도 있다. 의사 바이너리 코드 템플릿들 (1202) 은 관심의 기능적 포인트들 (304) 에 의해 커버된 영역에서 예를 들어, 코드 및/또는 데이터 구조들에서의 주목할만한 변화가 있을 때에만 업데이트되어야 할 수도 있음을 알아야 한다. 이러한 유형의 변화들은 상대적으로 빈번하지 않을 수도 있다 (예들 들어, 6 개월에 한번). 이러한 또는 다른 유형의 업데이트들은 OTA 업데이트를 통해 구현될 수도 있다. 이것은 주간/월간 단위로부터의 가상 어드레스들의 OTA 업데이트들을 단지 6 개월에 한번 의사 바이너리 코드 템플릿들 (1202) 의 OTA 업데이트들을 행하는 것으로의 상당한 감소를 가능하게 할 수도 있다.

[0051] 기존 등록된 어플리케이션에 대한 새로운 바이너리 버전의 업데이트 또는 재설치가 검출될 수도 있다. 이에 대응하여, 메타데이터 (1200) 및 의사 바이너리 코드 템플릿 (1202) 은 VAFMT (120) 를 자동으로 업데이트하는데 사용될 수도 있다. 도 12 에 도시된 바와 같이, 의사 바이너리 코드 템플릿들 (1202) 은 새로운 어플리케이션에서 바이너리 코드의 영역 (1206) 을 패턴 매칭하는데 사용될 수도 있으며, 여기서 의사 바이너리 코드 템플릿들 (1202) 로 나타낸 관심의 기능적 포인트들 (304)(및 이에 따른 이러한 특정 의사 바이너리 코드 템플릿이 나타내는 관심의 가상 어드레스 포인트들) 이 위치된다. 메타데이터 (1200) 는 어플리케이션 바이너리 코드 (108) 의 업데이트된 버전 (1204) 에서 탐색될 영역 (1206) 을 포커싱하는데 사용될 수도 있다. 초기 시도들은 고유 기능성을 위한 관심의 기능적 포인트들 (304) 에 대해 원래 베이스 (BASE0) 로부터의 상대적 OFFSET 을 사용함으로써 포커싱된 영역 (1206)(예를 들어, 베이스, BASE2 전후에 미리결정된 퍼센티지) 상에서 탐색하도록 이루어질 수도 있다. 많은 유형의 빈번한 업데이트들에서 이러한 상대적 오프셋들은 가까이에 있다는 것을 알아야 한다. 도 12 에 추가로 도시된 바와 같이, 매칭이 검출될 때, 새로운 가상 어드레스들은 새로운 바이너리로부터 획득될 수도 있고, VAFMT (120) 은 새로운 가상 어드레스들을 반영하기 위해 업데이트될 수도 있다. 관심의 하나 이상의 기능적 포인트들 (304) 이 새로운 바이너리에서의 매칭을 산출하지 못하는 경우, 컴퓨팅 시스템 (100) 은 OTA 업데이트를 개시할 수도 있고, 또는 다른 실시형태에서, 특정 기능성의 중요성에 기초하여 VAFMT (120) 로부터 연관된 가상 어드레스들 및 관심의 특정 기능성을 삭제할 수도 있다.

[0052] 도 13 은 업데이트된 가상 어드레스들 (그레이 아웃 상자들로 나타냄) 로 도 12 로부터의 VAFTM (120) 을 나타낸다. 관심의 DOCUMENT_WRITE_FUNCTION_START 포인트 (304) 에 대응하는 가상 어드레스 (302) 는 새로운 가상 어드레스 (0x3133b61c) 로 업데이트되었다. 관심의 DOCUMENT_WRITE_1 포인트 (304) 에 대응하는 가상 어드레스 (302) 는 새로운 가상 어드레스 (0x3133b62c) 로 업데이트되었다. 관심의 DOCUMENT_WRITE_2 포인트 (304) 에 대응하는 가상 어드레스 (302) 는 새로운 가상 어드레스 (0x3133b62c) 로 업데이트되었다. 도 12 에 추가로 도시된 바와 같이, 가상 어드레스에 대응하는 메타데이터 (1200) 가 또한 업데이트될 수도 있다.

도 13 에 도시된 바와 같이, 새로운 가상 어드레스 (0x3133b61c) 에 대한 메타데이터 (1200) 는 "BASE2 = BASE0 + 74709000" 로 업데이트되었다. 이것은 어플리케이션의 업데이트된 바이너리에서 2 개의 관심의 기능성들 사이에서 (즉, "KERNEL_ALLOCATOR_FUNCTION" 와 "DOCUMENT_WRITE_FUNCTION" 사이에서) 약간의 상대적인 포지션 변화가 있었음을 도시한다. 변화는 상대적으로 작을 수도 있다. 예를 들어, 변화는 이들 사이의 원래의 총 거리 74709704 바이트에서 704 바이트의 감소일 수도 있다. 이에 따라, 2 개의 관심의 기능성들 사이에서 베이스 오프셋 메타데이터 (즉, 74709704 바이트) 가 탐색 영역을 좁히는 것에 의해 효과적인 매칭을 허용하기 전후에 약간의 허용오차로 탐색이 포커싱되었다. 새로운 가상 어드레스 (0x3133b62c) 에 대한 메타데이터 (1200) 는 BASE2 + 16 으로 업데이트되었다. 새로운 가상 어드레스 (0x3133b640) 에 대한 메타데이터 (1200) 는 BASE2 + 36 로 업데이트되었다.

[0053] 도 14 및 도 15 는 DOCUMENT_WRITE 함수에 관련된 관심의 기능적 포인트들 (304) 의 세트와 연관된 의사 바이너리 코드 템플릿 (1202) 의 예시적인 실시형태를 도시한다. 관심의 기능적 포인트들 (304) 의 세트는 DOCUMENT_WRITE_FUNCTION_START 모듈, DOCUMENT_WRITE_1 모듈, 및 DOCUMENT_WRITE_2 모듈을 포함한다. 도 14 에 도시된 바와 같이, 세트에서의 관심의 기능적 포인트들 (304) 의 각각은 의사 바이너리 코드 템플릿 (1202) 내에서 "관심의 의사 바이너리 명령 포인트들" 을 형성하는 특정 의사 코드 명령들과 직접 연관된다.

의사 바이너리 코드 템플릿 (1202) 내의 이러한 "관심의 의사 바이너리 명령 포인트들" 은 "관심의 의사 바이

너리 포인트들" 과 직접 매칭되는 업데이트된 어플리케이션 바이너리에서의 특정 바이너리 명령들에 의존하여 어플리케이션 바이너리의 업데이트된 버전의 관심의 새로운 가상 어드레스 포인트들과의 현재 VAFMT (120) 에서의 관심의 가상 어드레스 포인트들의 일 대 일 매핑을 포함한다. 도 14 에 도시된 바와 같이, DOCUMENT_WRITE_FUNCTION_START 모듈은 제 1 의 2 개의 호출기 저장된 의사 레지스터들 (CallSave0, CallSave1) 및 리턴 레지스터 (ReturnReg) 를 저장하는 "푸시" 동작과 연관된다. 다음으로 후속 LoadWord 동작에 의해 필요한 어드레스를 컴퓨팅하는 AddWord 동작이 후속한다. AddWord 동작은 프로그램 카운터로 8 비트로 고정하여야 하는 상수 값을 가산하고 그 결과를 의사 레지스터 (reg0) 에 저장한다. 후속 LoadWord 동작은 값을 로드할 어드레스로서 reg0 에서의 어드레스를 직접 사용한다. 어플리케이션에 대한 실제 바이너리에서, 8 비트 상수를 갖는 AddWord 는 어드레싱 모드의 부분으로서 LoadWord 명령에 직접 포함될 수 있다. 'Const8bits' 는 옵션이 8 비트로 고정하는 임의의 상수 값을 갖도록 한다. 로딩된 값은 의사 레지스터 (reg1) 에서 유지되고 의사 레지스터 (reg2) 에서의 값을 로딩하는 제 2 LoadWord 동작에 대해 어드레스로서 사용된다. DOCUMENT_WRITE_FUNCTION_START 로 나타낸 관심의 기능적 포인트에 대해, "푸시" 동작은 이러한 의사 바이너리 코드 템플릿 (1202) 에서 "관심의 의사 바이너리 명령 포인트" 이다.

[0054] DOCUMENT_WRITE_1 모듈은 의사 레지스터 (reg0) 에서 유지되고 의사 레지스터 (reg1) 에 저장되는 값의 16 비트에 의한 로컬-시프트-좌측 동작과 연관된다. 이것은 그 후 상수 값 '4' 로 가산되고 그 후 값이 의사 레지스터 (reg3) 에 로딩되는 어드레스로서 사용되는 의사 레지스터 (reg2) 에 저장된다. 실제 바이너리 로드 명령에 대해, 어드레싱 모드는 직접 상수 값 4 로 가산을 수행할 수 있고, 이로써 AddWord 및 LoadWord 는 단일 로드 명령으로 나타낼 수 있음을 유의해야 한다. reg3 에서의 값은 또한, 비트 값이 제 1 인수로서 호출된 루틴에 패스하는데 사용되는 제 1 인수 레지스터 'ArgumentReg0' 로 로딩되는 어드레스인 의사 레지스터 (reg4) 에서 최종 어드레스를 생성하기 위해 프로그램 카운터 값 (PC) 에 가산된다. 그 후, 20 비트로 고정할 수 있는 값인 오프셋에 있는 어드레스에 직접 분기가 있다. 하지만, 직접 분기 명령 전에, 직접 분기가 어플리케이션의 상이한 부분에 대한 제어를 취한 후 (ReturnReg 를 적절히 설정하는 것에 의해) 리턴하기 위해 어드레스를 저장하는 AddWord 명령이 있다. "논리적-시프트-좌측" 동작은 DOCUMENT_WRITE_1 로 나타낸 관심의 기능적 포인트에 대해 이러한 의사 바이너리 코드 템플릿 (1202) 에서의 "관심의 의사 바이너리 명령 포인트" 이다.

[0055] DOCUMENT_WRITE_2 모듈은 프로그램 카운터에 의해 8 비트로 고정할 수도 있는 상수 값을 가산하고 그 결과를 의사 레지스터 (reg0) 에서 유지하는 AddWord 동작과 연관된다. 그 후 의사 레지스터 (reg0) 는 값이 의사 레지스터 (reg2) 에 로딩되는 어드레스로서 사용된다. 다음으로 프로그램 카운터의 카운터 값 및 의사 레지스터 (reg2) 를 가산하고 그 결과를 의사 레지스터 (reg1) 에 유지하는 다른 AddWord 동작이 후속한다. 의사 레지스터 (reg1) 는 그 후 직접 분기 명령을 통해 후속 서브루틴 호출에 값을 패스하는데 사용되는 ArgumentReg0 에서 값이 로딩되는 어드레스로서 사용된다. 실제 바이너리 로드 명령에 대해, 어드레싱 모드는 상수 값에 의해 가산을 직접 수행할 수 있고 이로써 AddWord 및 LoadWord 는 어플리케이션의 실제 바이너리에서의 단일 로드 명령으로 나타낼 수 있음을 유의해야 한다. LoadWord 동작 후, 20 비트로 고정할 수 있는 값인 오프셋에 있는 어드레스에 대해 직접 분기가 있다. 하지만, 직접 분기 명령 전에, 직접 분기가 어플리케이션의 상이한 부분에 대해 제어를 취한 후로 (ReturnReg 을 적절히 설정하는 것에 의해) 리턴하기 위해 어드레스를 저장하는 AddWord 명령이 있다. 서브루틴에 대한 호출 다음에 의사 바이너리 코드 템플릿 (1202) 내의 근방의 위치들에 대한 2 세트의 비교 및 분기가 후속한다. 양자의 비교들은 제 1 서브루틴 리턴 값 레지스터 (ReturnValueReg0) 상에서 행해져서 서브루틴에 의해 리턴된 특정 값들 ('0' 및 '1') 을 체크하며 국부적으로 BranchEQ 및 BranchNE 동작들을 각각 사용하여 분기들을 행하는 리턴된 값에 기초한다. 분기 타겟 어드레스들은 현재 프로그램 카운터 값으로부터 상수 오프셋으로서 제공된다. 프로그램 카운터로 Const8bits 피연산자를 가산하는 AddWord 동작은 DOCUMENT_WRITE_2 로 나타낸 관심의 기능적 포인트에 대해 이러한 의사 바이너리 코드 템플릿 (1202) 에서의 "관심의 의사 바이너리 명령 포인트" 이다. 어플리케이션의 실제 바이너리는 ("ldr r1, [pc,#80] 로서) 단일의 실제 명령으로의 의사 바이너리 코드 템플릿 매치에서의 LoadWord 동작과 함께 이러한 어드레스 산출 동작 (AddWord) 을 가질 수 있고, 이 경우 "관심의 의사 바이너리 명령 포인트" 가 전체 또는 서브 부분으로서 매칭하는 실제 바이너리 명령이, 어플리케이션의 바이너리의 새로운 버전에서 업데이트된 가상 어드레스를 결정하는 명령이 되는 것을 유의해야 한다.

[0056] 도 15 는 어플리케이션 바이너리 코드 (108) 의 업데이트된 버전 (1204) 의 매칭된 영역 (1206) 에서 동등한 대응 바이너리 코드로의 의사 바이너리 코드 템플릿 (1202) 에서의 의사 코드 명령들의 각각의 매칭을 도시한다.

동작에 있어서, 의사 바이너리 코드 템플릿 (1202) 이 영역 (1206) 과 매칭할 때, 관심의 기능적 포인트 (304) 와 매칭하는 바이너리 코드에서의 대응 명령들의 가상 어드레스들이 새로운 가상 어드레스들이 되고

VAFMT (120) 에서 업데이트된다. 새로운 베이스 및 오프셋들은 새로운 가상 어드레스들에 기초하여 컴퓨팅될 수도 있고, 메타데이터 (1200) 는 업데이트될 수도 있다.

[0057] 도 16 은 어플리케이션 바이너리 코드 (108) 의 새로운 또는 업데이트된 버전이 설치될 때 VAFMT (120) 를 자동으로 업데이트하기 위해 컴퓨팅 시스템 (100) 에서 구현되는 방법 (1600) 의 일 실시형태를 도시한다. 블록 (1602) 에서, HLOS (106) 에 등록된 어플리케이션에 대한 가상 어드레스 매핑 테이블 (120) 은 상술한 바와 같이, 컴퓨팅 시스템 (100) 에 저장될 수도 있다. VAFMT (120) 는 HLOS (106) 에서의 보안 메모리에 저장될 수도 있다. 도 12 에 도시된 바와 같이, VAFMT (120) 는 등록된 어플리케이션에 대해 어플리케이션 바이너리 코드 (108) 에서의 대응 타겟 어플리케이션 기능성들 (관심의 기능적 포인트들 (304)) 에 매핑된 복수의 가상 어드레스들 (302) 의 세트를 포함할 수도 있다. 어플리케이션 바이너리 코드 (108) 의 업데이트된 버전 (1204) 을 수신하는 것에 응답하여 (결정 블록 (1604)), 가상 어드레스 매핑 테이블 (120) 에서의 복수의 가상 어드레스들의 세트들 (302) 중 하나 이상과 연관된 대응하는 의사 바이너리 코드 템플릿들 (1202) 이 결정될 수도 있다 (블록 (1606)). 위에 언급된 바와 같이, 일 실시형태에서, 의사 바이너리 코드 템플릿들 (1202) 은 초기 VAFMT (120) 와 함께 시스템 (100) 에 대한 오버-디-에어 (OTA) 업데이트들을 통해, 또는 시스템 (100) 상의 코드/데이터를 다운로딩 및 설치하는 임의의 다른 수단에 의해 초기에 취득될 수도 있다. 이러한 의사 바이너리 코드 템플릿들 (1202) 및 VAFMT (120) 의 양자 모두는 HLOS (106) 및 커널에 의해 액세스 가능한 위치들에서 시스템 (100) 에 저장될 수도 있다. 실제 저장 위치는 구현에 의존한다. 보안 메모리 구성들 또는 보안 보호의 다양한 레벨들은 저장 위치들에 대해 고려될 수 있고 구현 선택에 의존한다. 의사 바이너리 코드 템플릿들 (1202) 은, 예를 들어 기존 템플릿들 중 하나 이상이 어플리케이션의 업데이트된 바이너리에서 임의의 매치를 찾을 수 없을 때 업데이트될 수도 있다. 미스매치들은 상술한 변화들의 다른 종류들, 또는 관심의 영역들에 있어서 어플리케이션 코드에서의 큰 스케일 변화에 기인하여 발생할 수도 있다. 이러한 상황들 동안, 업데이트된 의사 바이너리 코드 템플릿들 (1202) 및 업데이트된 VAFMT (120) 는 OTA 다운로드되고 시스템 (100) 에 설치될 수도 있다. 결정 블록 (1608) 에서, 의사 바이너리 코드 템플릿 (1202) 은 어플리케이션 바이너리 코드 (108) 의 업데이트된 버전 (1204) 을 탐색하고 의사 코드 명령(들) 을 등가의 바이너리 명령들에 매칭하는데 사용된다. 매치들이 발견될 때, 블록 (1610) 에서, 바이너리 명령들에 대응하는 새로운 가상 어드레스들이 결정된다. 블록 (1612) 에서, 가상 어드레스 매핑 테이블 (120) 은 새로운 가상 어드레스들 및 대응하는 업데이트된 베이스/오프셋 메타데이터 (1200) 로 업데이트될 수도 있다.

[0058] 도 16 에 도시된 바와 같이, 블록들 (1606, 1608, 1610, 및 1612) 은, 모든 의사 바이너리 코드 템플릿들 (1202) 이 매칭되고 VAFMT (120) 에서의 모든 가상 어드레스들이 업데이트될 때까지 모든 상이한 의사 바이너리 코드 템플릿들 (1202) 에 대해 반복될 수도 있다. 결정 블록 (1611) 에서, 방법 (1600) 은 모든 의사 바이너리 코드 템플릿들 (1202) 이 프로세싱되었는지를 결정할 수도 있다. "예" 이면, 방법 (1600) 은 블록 (1613) 에서 종료할 수도 있다. "아니오" 이면, 블록 (1606) 에서 새로운 의사 바이너리 코드 템플릿 (1202) 이 선택될 수도 있다. 결정 블록 (1608) 에서, 매칭 바이너리 시퀀스들이 특정 의사 바이너리 코드 템플릿 (1202) 에 대해 어플리케이션의 업데이트된 바이너리에서 식별됨에 따라, 방법 (1600) 은 매칭을 위해 다음 의사 바이너리 코드 템플릿 (1202) 에 대해 반복할 수도 있다. 일부 반복에서, 어플리케이션의 업데이트된 바이너리에서의 의사 바이너리 코드 템플릿 (1202) 에 대해 매치가 없는 경우, 의사 바이너리 코드 템플릿 (1202) 으로 나타낸, 관심의 기능성이 VAFMT (120)(결정 블록 (1607)) 으로부터 삭제될 수 있는지가 먼저 결정된다. 삭제될 수 있으면 (기능성의 중요성이 낮은 것을 포함한, 상이한 이유들에 기인할 수도 있음), 이러한 관심의 기능성에 대해 관심 엔트리들의 모든 가상 어드레스 포인트가 VAFMT (120) 로부터 삭제될 수도 있고 (블록 (1606)), 반복은 다음 의사 바이너리 코드 템플릿 (1202) 에 대한 매치를 탐색하기 위해 블록 (1606) 으로 계속된다. 하지만, 기능성 (및 이에 따른 의사 바이너리 코드 템플릿 (1202)) 이 중요하고 삭제되지 않아야 하는 경우 (블록 (1609)), 자동 업데이트 메커니즘이 실패하며, 이 경우 가상 어드레스들 및/또는 의사 바이너리 코드 템플릿들 (1202) 에 대한 완전한 오버-디-에어 (OTA) 업데이트가 수행될 수도 있다. 이것은 어플리케이션의 업데이트된 바이너리에서의 급격한 변화/수정이 있는 경우 존재할 수도 있다 (예를 들어, 적은 빈도로, 6 개월에 한번 발생함).

[0059] 본 명세서에 기재된 방법 단계들 중 하나 이상은 상술한 모듈들과 같은, 컴퓨터 프로그램 명령들로서 메모리에 저장될 수도 있다는 것을 알아야 한다. 이러한 명령들은 본 명세서에 기재된 방법들을 수행하기 위해 대응하는 모듈과 조합으로 또는 이와 함께 임의의 적절한 프로세서에 의해 실행될 수도 있다.

[0060] 본 명세서에 기재된 프로세스들 또는 프로세스 플로우들에서의 소정의 단계들은 본질적으로 기재된 바와 같이 기능하기 위해 발명에 대해 다른 것들을 선행한다. 하지만, 발명은 그러한 순서 또는 시퀀스가 발명의 기능

성을 변경하지 않는다면 기재된 단계들의 순서에 제한되지 않는다. 즉, 일부 단계들은 발명의 범위 및 사상으로부터 벗어나지 않으면서 다른 단계들 전에, 후에 또는 병렬로 (실질적으로 동시에) 수행될 수도 있다는 것이 인식된다. 일부 경우들에서, 소정의 단계들은 발명으로부터 벗어나지 않으면서 생략될 수도 있고 또는 수행되지 않을 수도 있다. 또한, "이후", "그후", "다음" 등과 같은 단어들은 단계들의 순서를 제한하도록 의도되지 않는다. 이러한 단어들은 단순히 예시적인 방법의 기재를 통해 독자를 안내하는데 사용된다.

[0061] 부가적으로, 프로그래밍에서의 당업자는 예를 들어, 본 명세서에서의 연관된 기재 및 플로우챠트들에 기초하여 어려움 없이 개시된 발명을 구현하도록 적절한 하드웨어 및/또는 회로들을 식별하거나 컴퓨터 코드를 기입할 수 있다.

[0062] 따라서, 상세한 하드웨어 디바이스들 또는 프로그램 코드 명령들의 특정 세트의 개시는 발명을 제작하고 사용하는 방법의 적절한 이해를 위해 필요한 것으로 고려되지 않는다. 청구된 컴퓨터 구현 프로세스들의 발명의 기능성은 위의 기재에서 더 상세하게 그리고 다양한 프로세스 플로우들을 예시할 수도 있는 도면들과 함께 설명된다.

[0063] 하나 이상의 예시적인 양태들에서, 기재된 기능들은 하드웨어, 소프트웨어, 펌웨어, 또는 이들의 임의의 조합으로 구현될 수도 있다. 소프트웨어에서 구현되는 경우, 함수들은 컴퓨터 판독가능 매체 상에서 하나 이상의 명령들 또는 코드로서 저장되거나 송신될 수도 있다. 컴퓨터 판독가능 매체들은 일 장소에서 다른 곳으로 컴퓨터 프로그램의 전송을 용이하게 하는 임의의 매체를 포함하는 통신 매체들 및 컴퓨터 저장 매체들의 양자 모두를 포함한다. 저장 매체들은 컴퓨터에 의해 액세스될 수도 있는 임의의 가용 매체들일 수도 있다. 한정이 아닌 예시로서, 이러한 컴퓨터 판독가능 매체들은 RAM, ROM, EEPROM, NAND 플래시, NOR 플래시, M-RAM, P-RAM, R-RAM, CD-ROM 또는 다른 광학 디스크 저장, 자기 디스크 저장 또는 다른 자기 저장 디바이스들, 또는 컴퓨터에 의해 액세스될 수도 있고 명령들 또는 데이터 구조들의 형태로 원하는 프로그램 코드를 반송하거나 저장하는데 사용될 수도 있는 임의의 다른 매체를 포함할 수도 있다.

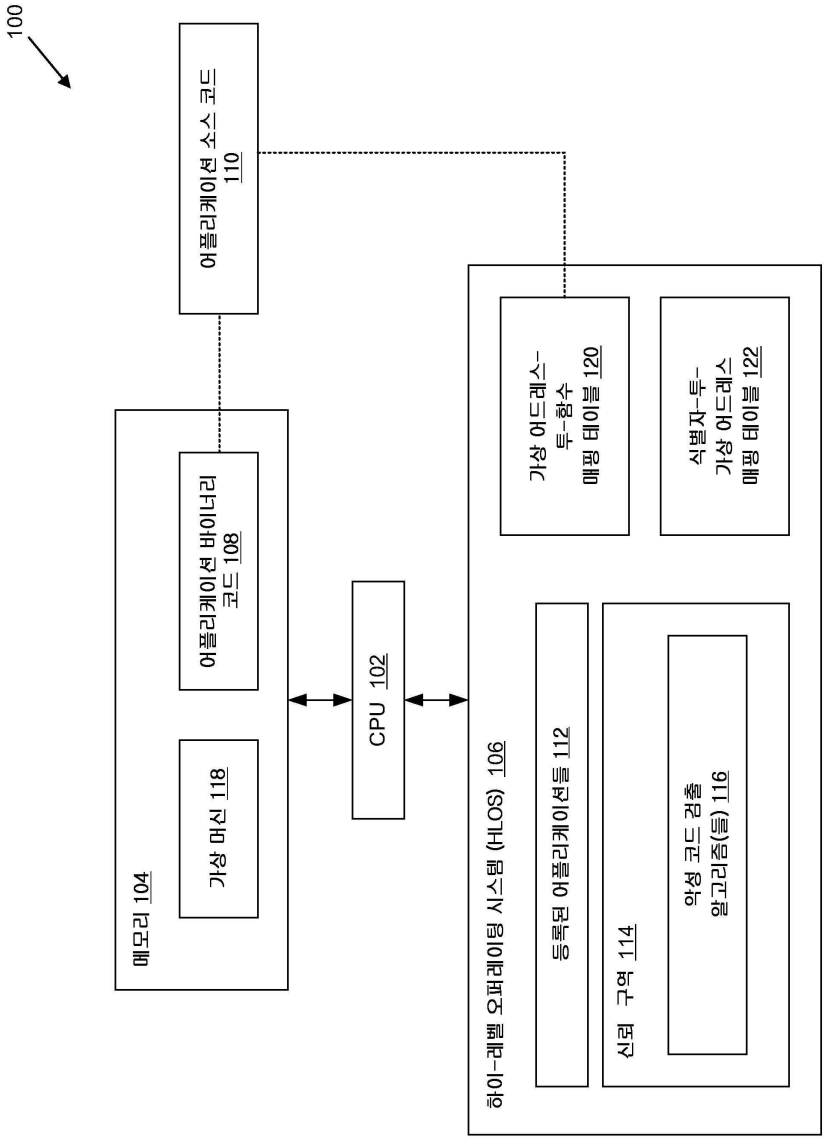
[0064] 또한, 임의의 접속은 적절히 컴퓨터 판독가능 매체로 지칭된다. 예를 들어, 소프트웨어가 동축 케이블, 광섬유 케이블, 연선, 디지털 가입자 라인 ("DSL"), 또는 적외선, 무선, 및 마이크로파와 같은 무선 기술들을 사용하여 웹사이트, 서버, 또는 다른 원격 소스로부터 송신되면, 동축 케이블, 광섬유 케이블, 연선, DSL, 또는 적외선, 무선, 및 마이크로파와 같은 무선 기술들은 매체의 정의 내에 포함된다.

[0065] 본 명세서에서 사용된 바와 같이, 디스크 (disk) 및 디스크 (disc) 는, 컴팩트 디스크 ("CD"), 레이저 디스크, 광학 디스크, 디지털 다기능 디스크 ("DVD"), 플로피 디스크, 및 블루레이 디스크를 포함하며, 여기서 디스크 (disk) 들은 보통 자기적으로 데이터를 재생하는 한편, 디스크 (disc) 들은 레이저를 이용하여 광학적으로 데이터를 재생한다. 상기의 조합들이 또한, 컴퓨터 판독가능 매체들의 범위 내에 포함되어야 한다.

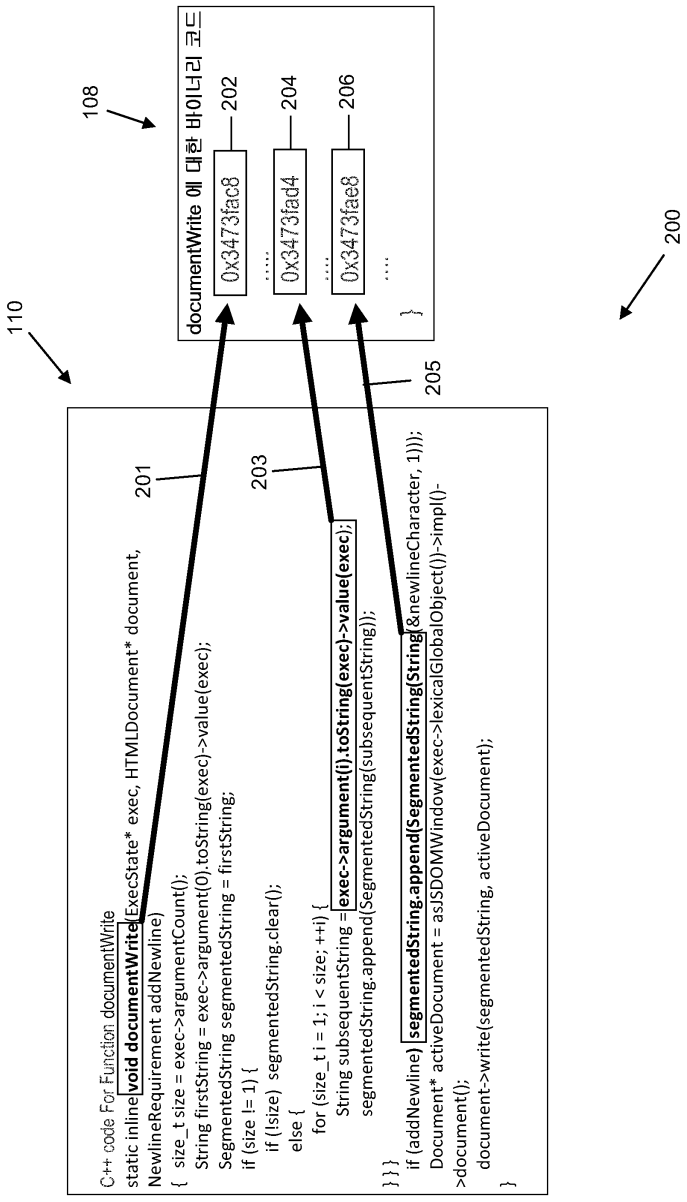
[0066] 대안의 실시형태들은 그 사상 및 범위로부터 벗어나지 않으면서 발명이 속하는 기술 분야의 당업자에게 명백해질 것이다. 따라서, 선택된 양태들이 상세하게 도시되고 기재되지만, 다음의 청구항들에 의해 정의되는 바와 같이, 본 발명의 사상 및 범위로부터 벗어나지 않으면서 다양한 하위 치환들 및 변경들이 이루어질 수도 있다.

도면

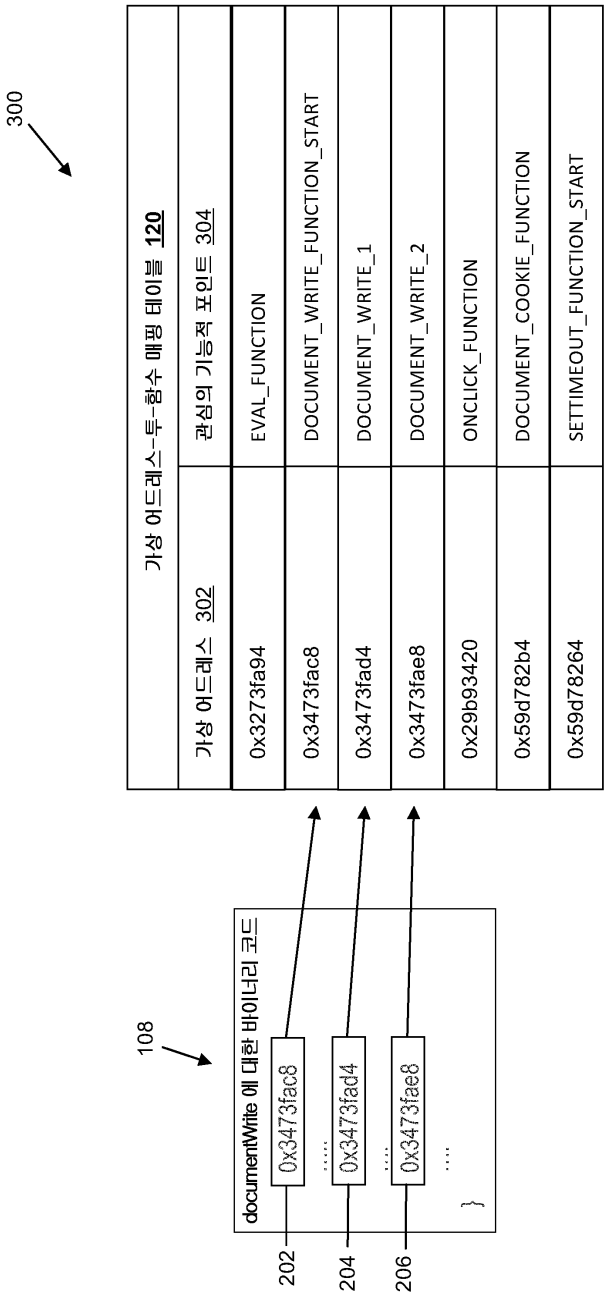
도면1


100
어플리케이션 소스 코드 110
메모리 104
가상 머신 118
어플리케이션 바이너리 코드 108
CPU 102
하이-레벨 오퍼레이팅 시스템 (HLOS) 106
등록된 어플리케이션들 112
신뢰 구역 114
악성 코드 검출 알고리즘(들) 116
가상 어드레스-투-핑수 매핑 테이블 120
식별자-투-가상 어드레스 매핑 테이블 122

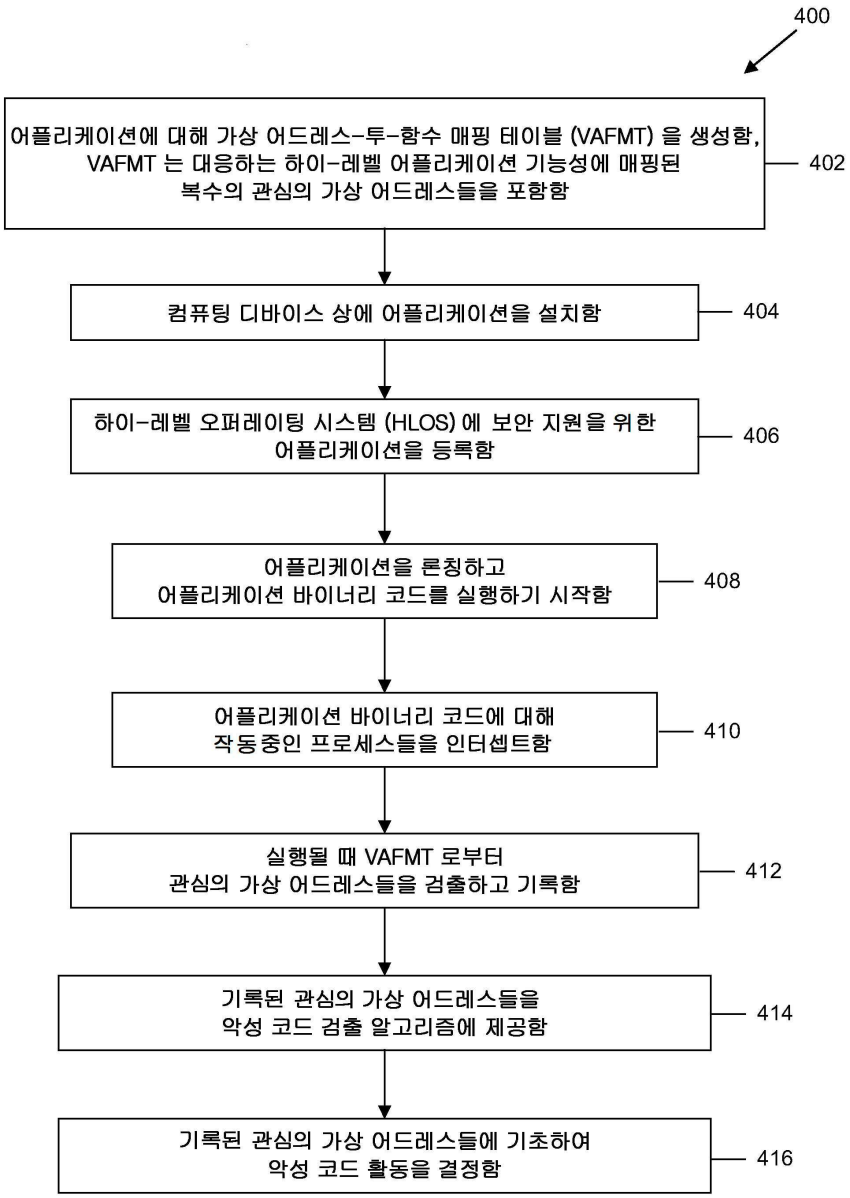
도면2


200
108
documentWrite 에 대한 바이너리 코드
0x3473fac8
0x3473fad4
0x3473fae8
202
204
206
110
201
203
205
C++ code For Function documentWrite
static inline void documentWrite(ExecState* exec, HTMLDocument* document,
NewlineRequirement addNewline)
{ size_t size = exec->argumentCount();
String firstString = exec->argument(0).toString(exec)->value(exec);
SegmentedString segmentedString = firstString;
if (size != 1) {
if (!size) segmentedString.clear();
else {
for (size_t i = 1; i < size; ++i) {
String subsequentString = exec->argument(i).toString(exec)->value(exec);
segmentedString.append(SegmentedString(subsequentString));
} } }
if (addNewline) segmentedString.append(SegmentedString(String(&newlineCharacter, 1)));
Document* activeDocument = asJSDOMWindow(exec->lexicalGlobalObject())->impl()->
>document();
document->write(segmentedString, activeDocument);
}

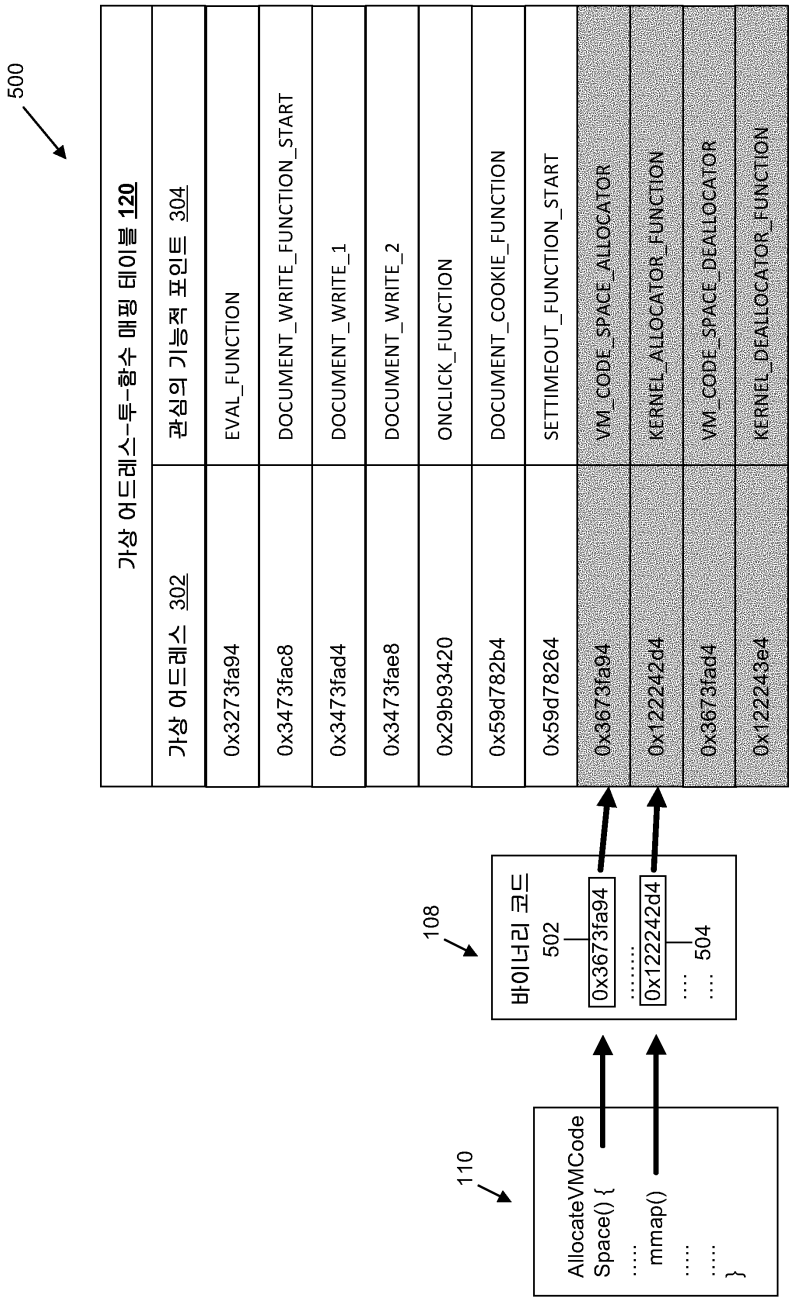
도면3


300
가상 어드레스-투-함수 매핑 테이블 120
가상 어드레스 302
관심의 기능적 포인트 304
0x3273fa94
EVAL_FUNCTION
0x3473fac8
DOCUMENT_WRITE_FUNCTION_START
0x3473fad4
DOCUMENT_WRITE_1
0x3473fae8
DOCUMENT_WRITE_2
0x29b93420
ONCLICK_FUNCTION
0x59d782b4
DOCUMENT_COOKIE_FUNCTION
0x59d78264
SETTIMEOUT_FUNCTION_START
108
documentWrite 에 대한 바이너리 코드
0x3473fac8
0x3473fad4
0x3473fae8
}
202
204
206

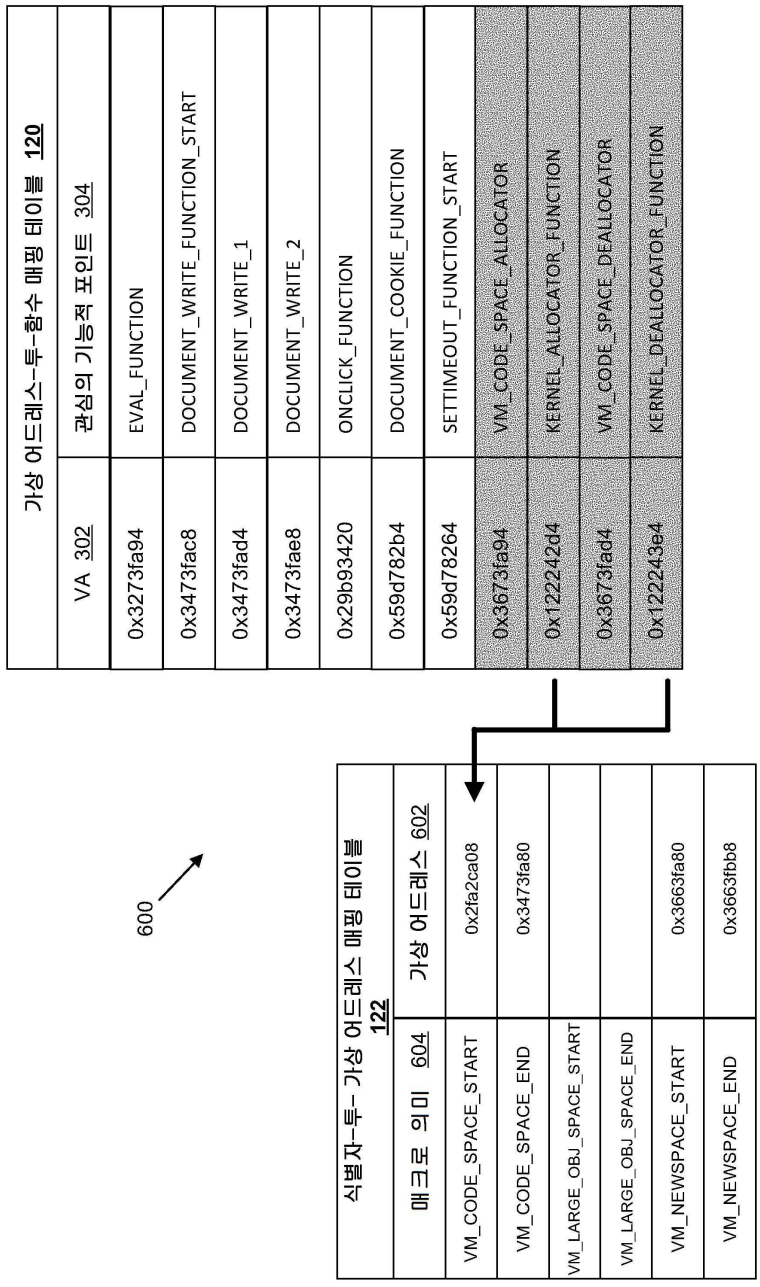
도면4


400
어플리케이션에 대해 가상 어드레스-투-함수 매핑 테이블 (VAFMT) 을 생성함, VAFMT 는 대응하는 하이-레벨 어플리케이션 기능성에 매핑된 복수의 관심의 가상 어드레스들을 포함함
402
컴퓨팅 디바이스 상에 어플리케이션을 설치함
404
하이-레벨 오퍼레이팅 시스템 (HLOS) 에 보안 지원을 위한 어플리케이션을 등록함
406
어플리케이션을 론칭하고 어플리케이션 바이너리 코드를 실행하기 시작함
408
어플리케이션 바이너리 코드에 대해 작동중인 프로세스들을 인터셉트함
410
실행될 때 VAFMT 로부터 관심의 가상 어드레스들을 검출하고 기록함
412
기록된 관심의 가상 어드레스들을 악성 코드 검출 알고리즘에 제공함
414
기록된 관심의 가상 어드레스들에 기초하여 악성 코드 활동을 결정함
416

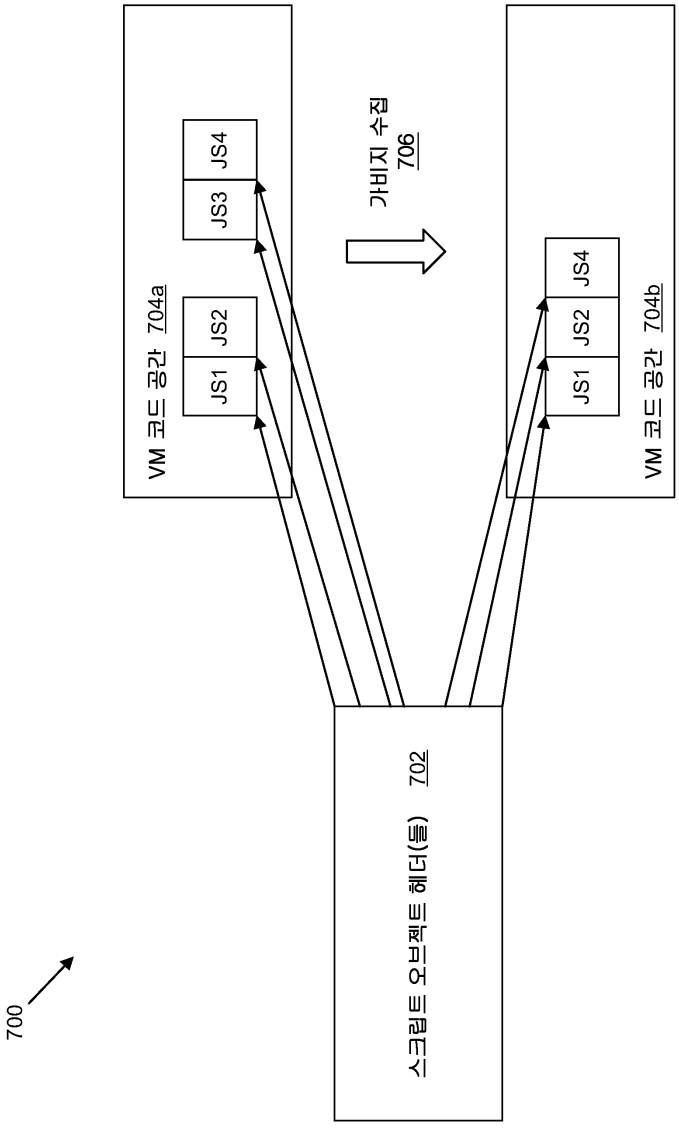
도면5


500
가상 어드레스-투-함수 매핑 테이블 120
가상 어드레스 302
관심의 기능적 포인트 304
0x3273fa94
EVAL_FUNCTION
0x3473fac8
DOCUMENT_WRITE_FUNCTION_START
0x3473fad4
DOCUMENT_WRITE_1
0x3473fae8
DOCUMENT_WRITE_2
0x29b93420
ONCLICK_FUNCTION
0x59d782b4
DOCUMENT_COOKIE_FUNCTION
0x59d78264
SETTIMEOUT_FUNCTION_START
0x3673fa94
VM_CODE_SPACE_ALLOCATOR
0x122242d4
KERNEL_ALLOCATOR_FUNCTION
0x3673fad4
VM_CODE_SPACE_DEALLOCATOR
0x122243e4
KERNEL_DEALLOCATOR_FUNCTION
108
바이너리 코드
502
0x3673fa94
0x122242d4
504
110
AllocateVMCode
Space() {
mmap()
}

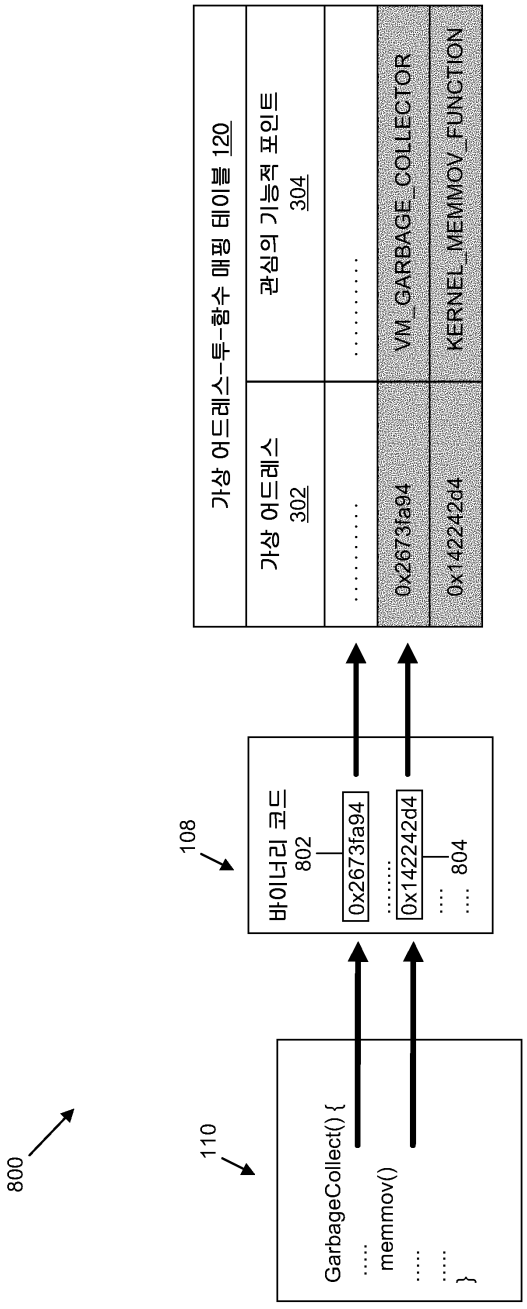
도면6


600
식별자-투- 가상 어드레스 매핑 테이블 122
매크로 의미 604
가상 어드레스 602
VM_CODE_SPACE_START
0x2fa2ca08
VM_CODE_SPACE_END
0x3473fa80
VM_LARGE_OBJ_SPACE_START
VM_LARGE_OBJ_SPACE_END
VM_NEWSPACE_START
0x3663fa80
VM_NEWSPACE_END
0x3663fbb8
가상 어드레스-투-함수 매핑 테이블 120
VA 302
관심의 기능적 포인트 304
0x3273fa94
EVAL_FUNCTION
0x3473fac8
DOCUMENT_WRITE_FUNCTION_START
0x3473fad4
DOCUMENT_WRITE_1
0x3473fae8
DOCUMENT_WRITE_2
0x29b93420
ONCLICK_FUNCTION
0x59d782b4
DOCUMENT_COOKIE_FUNCTION
0x59d78264
SETTIMEOUT_FUNCTION_START
0x3673fa94
VM_CODE_SPACE_ALLOCATOR
0x122242d4
KERNEL_ALLOCATOR_FUNCTION
0x3673fad4
VM_CODE_SPACE_DEALLOCATOR
0x122243e4
KERNEL_DEALLOCATOR_FUNCTION

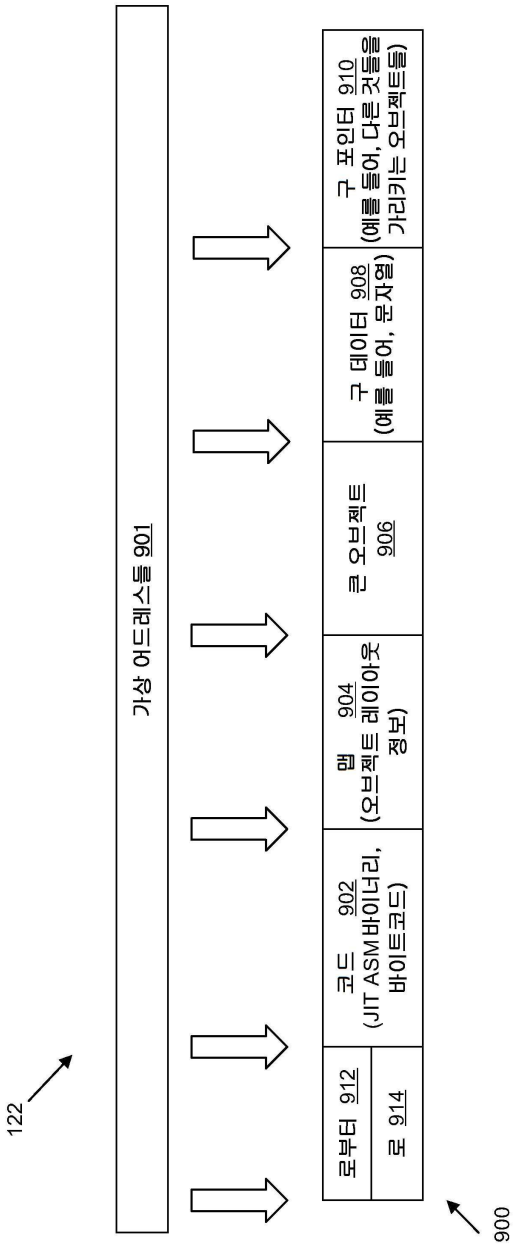
도면7


700
스크립트 오브젝트 헤더(들) 702
VM 코드 공간 704a
JS1
JS2
JS3
JS4
가비지 수집
706
VM 코드 공간 704b
JS1
JS2
JS4

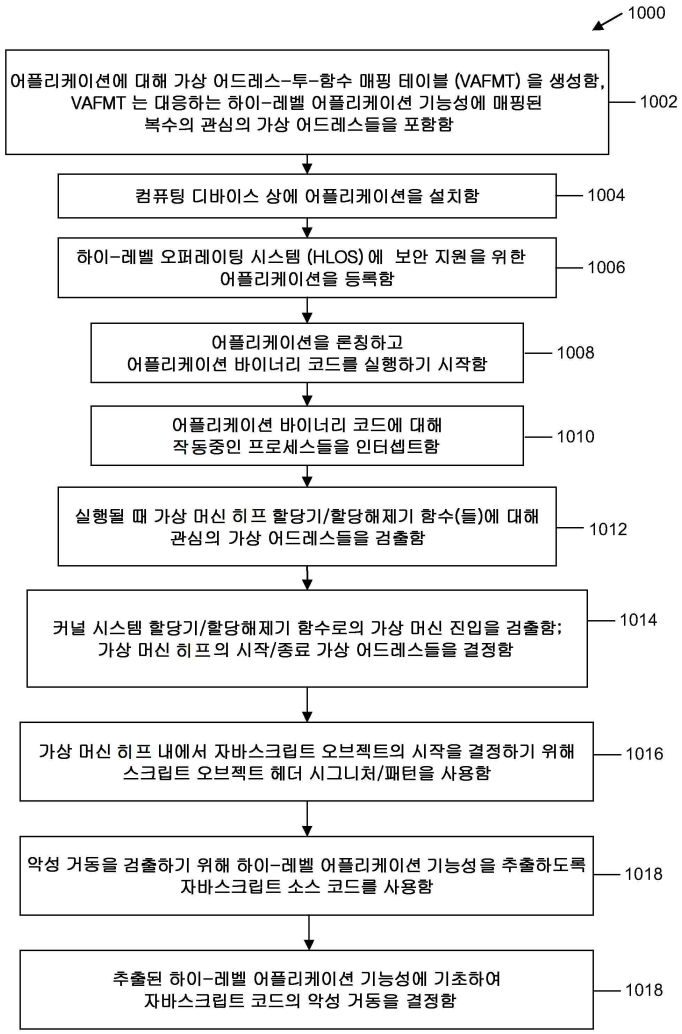
도면8


800
110
GarbageCollect() {
.....
memmov()
.....
.....
}
108
바이너리 코드
802
0x2673fa94
.....
0x142242d4
....
....
804
가상 어드레스-투-함수 매핑 테이블 120
가상 어드레스
302
관심의 기능적 포인트
304
.....
.....
0x2673fa94
VM_GARBAGE_COLLECTOR
0x142242d4
KERNEL_MEMMOV_FUNCTION

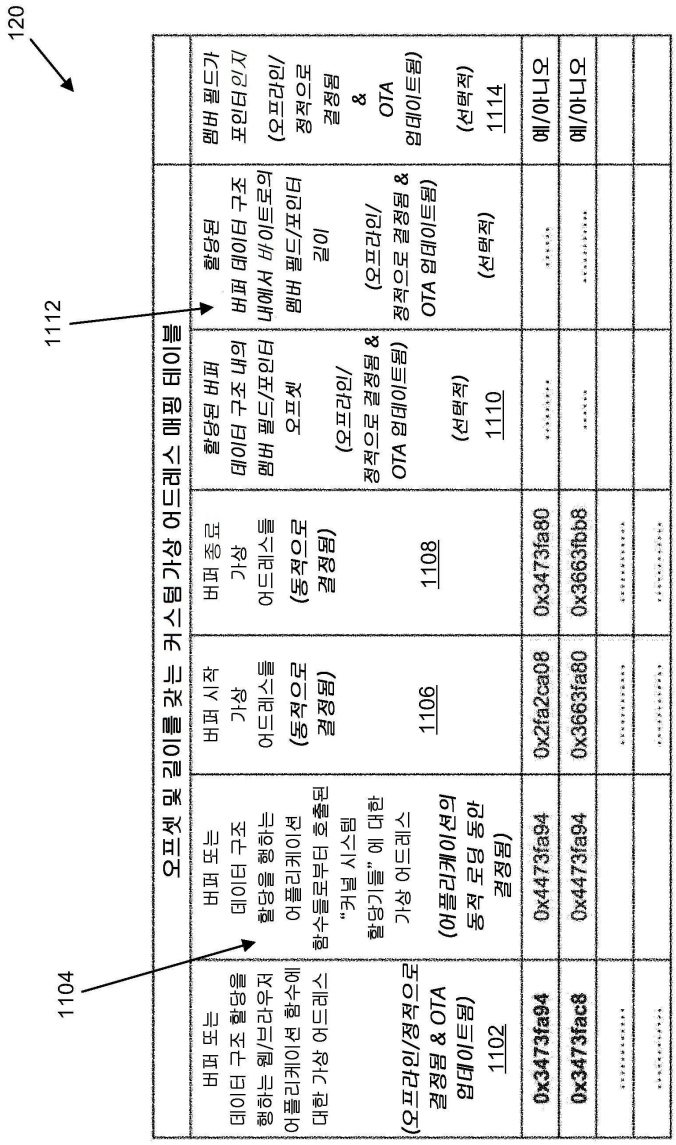
도면9


122
가상 어드레스들 901
로부터 912
로 914
코드 902
(JIT ASM바이너리, 바이트코드)
맵 904
(오브젝트 레이아웃 정보)
큰 오브젝트 906
구 데이터 908
(예를 들어, 문자열)
구 포인터 910
(예를 들어, 다른 것들을 가리키는 오브젝트들)
900

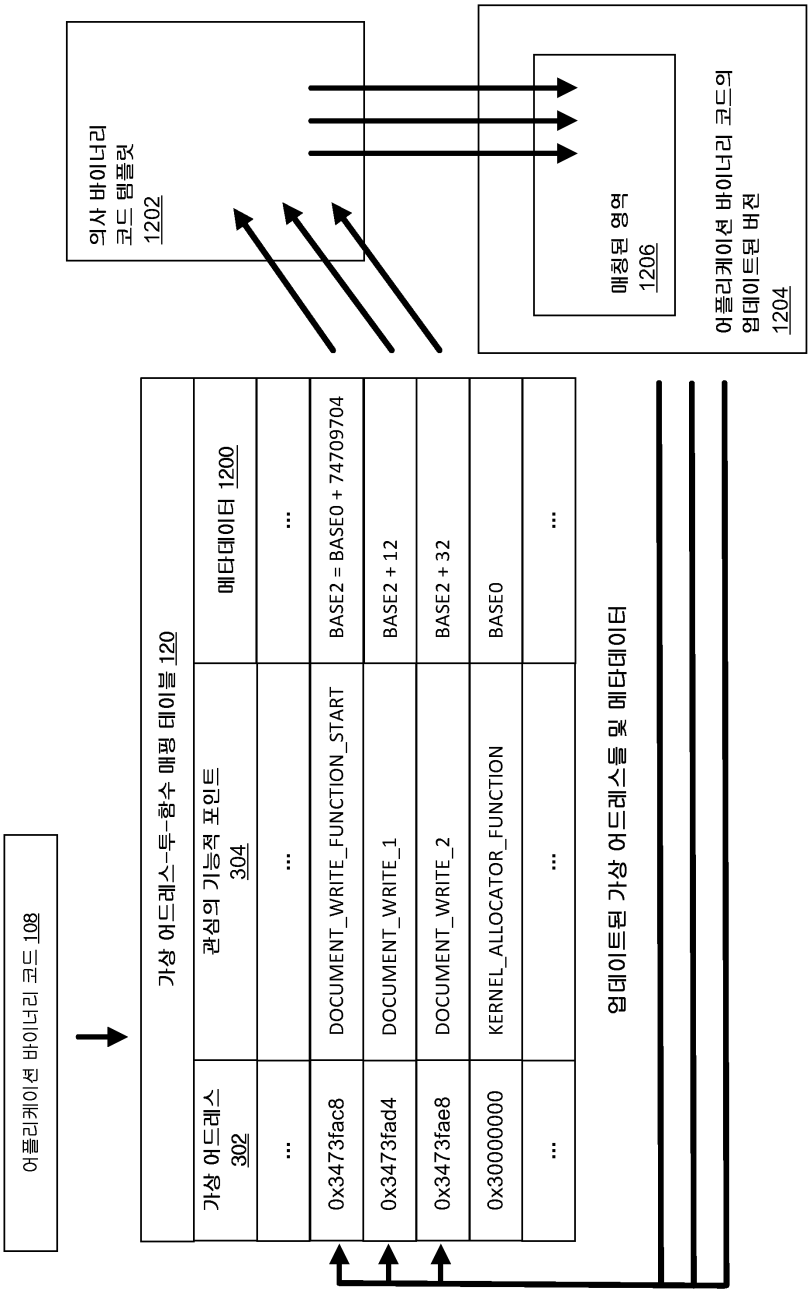
도면10


1000
어플리케이션에 대해 가상 어드레스-투-함수 매핑 테이블 (VAFMT) 을 생성함, VAFMT 는 대응하는 하이-레벨 어플리케이션 기능성에 매핑된 복수의 관심의 가상 어드레스들을 포함함
1002
컴퓨팅 디바이스 상에 어플리케이션을 설치함
1004
하이-레벨 오퍼레이팅 시스템 (HLOS) 에 보안 지원을 위한 어플리케이션을 등록함
1006
어플리케이션을 론칭하고 어플리케이션 바이너리 코드를 실행하기 시작함
1008
어플리케이션 바이너리 코드에 대해 작동중인 프로세스들을 인터셉트함
1010
실행될 때 가상 머신 히프 할당기/할당해제기 함수(들)에 대해 관심의 가상 어드레스들을 검출함
1012
커널 시스템 할당기/할당해제기 함수로의 가상 머신 진입을 검출함; 가상 머신 히프의 시작/종료 가상 어드레스들을 결정함
1014
가상 머신 히프 내에서 자바스크립트 오브젝트의 시작을 결정하기 위해 스크립트 오브젝트 헤더 시그니처/패턴을 사용함
1016
악성 거동을 검출하기 위해 하이-레벨 어플리케이션 기능성을 추출하도록 자바스크립트 소스 코드를 사용함
1018
추출된 하이-레벨 어플리케이션 기능성에 기초하여 자바스크립트 코드의 악성 거동을 결정함
1018

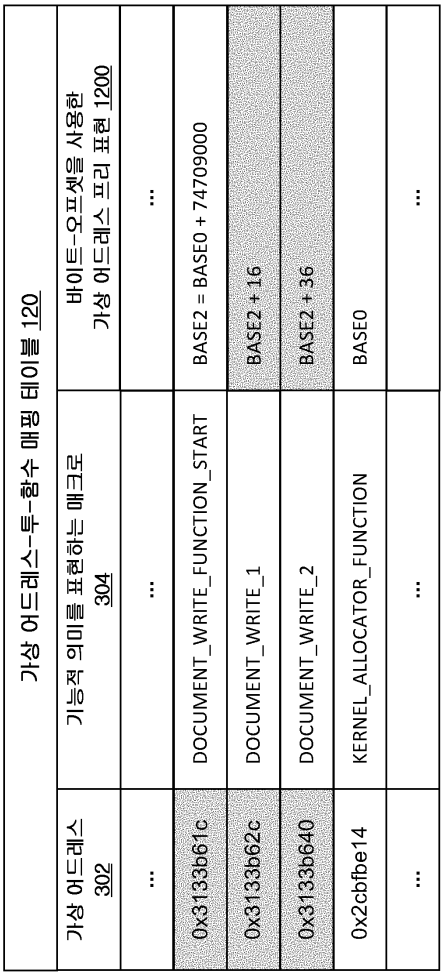
도면11


120
1112
1104
오프셋 및 길이를 갖는 커스텀 가상 어드레스 매핑 테이블
버퍼 또는 데이터 구조 할당을 행하는 웹/브라우저 어플리케이션 함수에 대한 가상 어드레스 (오프라인/정적으로 결정됨 & OTA 업데이트됨) 1102
버퍼 또는 데이터 구조 할당을 행하는 어플리케이션 어플리케이션 함수들로부터 호출된 "커널 시스템 할당기들"에 대한 가상 어드레스 (어플리케이션의 동적 로딩 동안 결정됨)
버퍼 시작 가상 어드레스들 (동적으로 결정됨) 1106
버퍼 종료 가상 어드레스들 (동적으로 결정됨) 1108
할당된 버퍼 데이터 구조 내의 멤버 필드/포인터 오프셋 (오프라인/정적으로 결정됨 & OTA 업데이트됨) (선택적) 1110
할당된 버퍼 데이터 구조 내에서 바이트로의 멤버 필드/포인터 길이 (오프라인/정적으로 결정됨 & OTA 업데이트됨) (선택적) 1114
멤버 필드가 포인터인지 (오프라인/정적으로 결정됨 & OTA 업데이트됨) (선택적)
0x3473fa94
0x4473fa94
0x2fa2ca08
0x3473fa80
예/아니오
0x3473fac8
0x4473fa94
0x3663fa80
0x3663fbb8
예/아니오

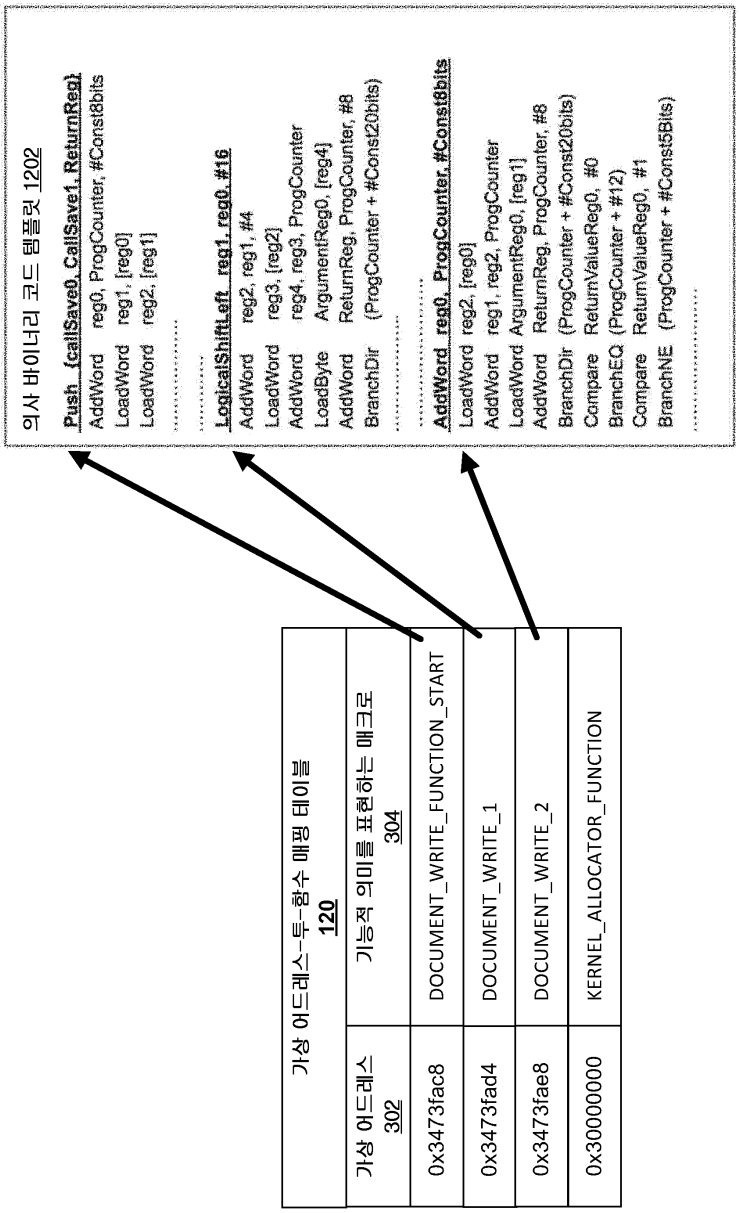
도면12


어플리케이션 바이너리 코드 108
가상 어드레스-투-함수 매핑 테이블 120
가상 어드레스 302
관심의 기능적 포인트 304
메타데이터 1200
0x3473fac8
0x3473fad4
0x3473fae8
0x30000000
DOCUMENT_WRITE_FUNCTION_START
DOCUMENT_WRITE_1
DOCUMENT_WRITE_2
KERNEL_ALLOCATOR_FUNCTION
BASE2 = BASE0 + 74709704
BASE2 + 12
BASE2 + 32
BASE0
업데이트된 가상 어드레스들 및 메타데이터
의사 바이너리 코드 템플릿 1202
어플리케이션 바이너리 코드의 업데이트된 버전 1204
매칭된 영역 1206

도면13

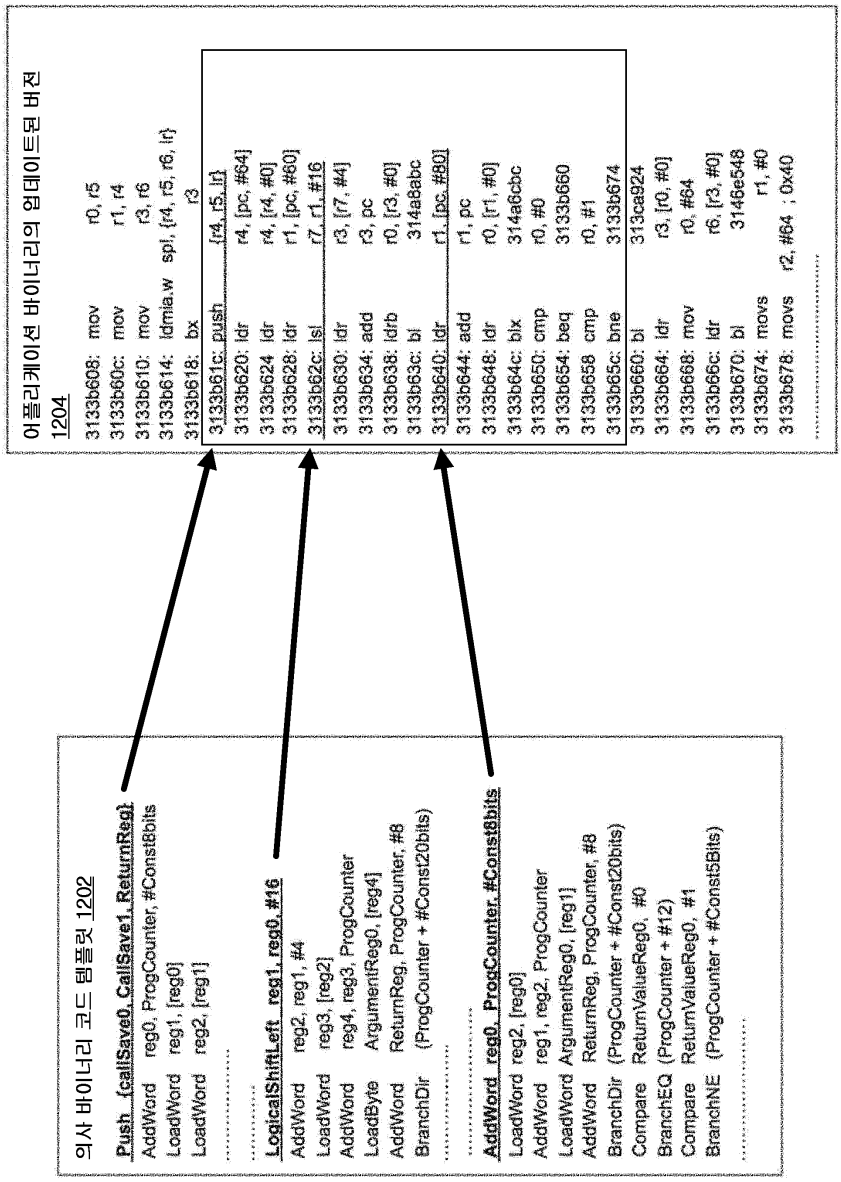

가상 어드레스-투-함수 매핑 테이블 120
가상 어드레스 302
기능적 의미를 표현하는 매크로 304
바이트-오프셋을 사용한 가상 어드레스 프리 표현 1200
...
...
...
0x3133b61c
DOCUMENT_WRITE_FUNCTION_START
BASE2 = BASE0 + 74709000
0x3133b62c
DOCUMENT_WRITE_1
BASE2 + 16
0x3133b640
DOCUMENT_WRITE_2
BASE2 + 36
0x2cbfbe14
KERNEL_ALLOCATOR_FUNCTION
BASE0
...
...
...

도면14

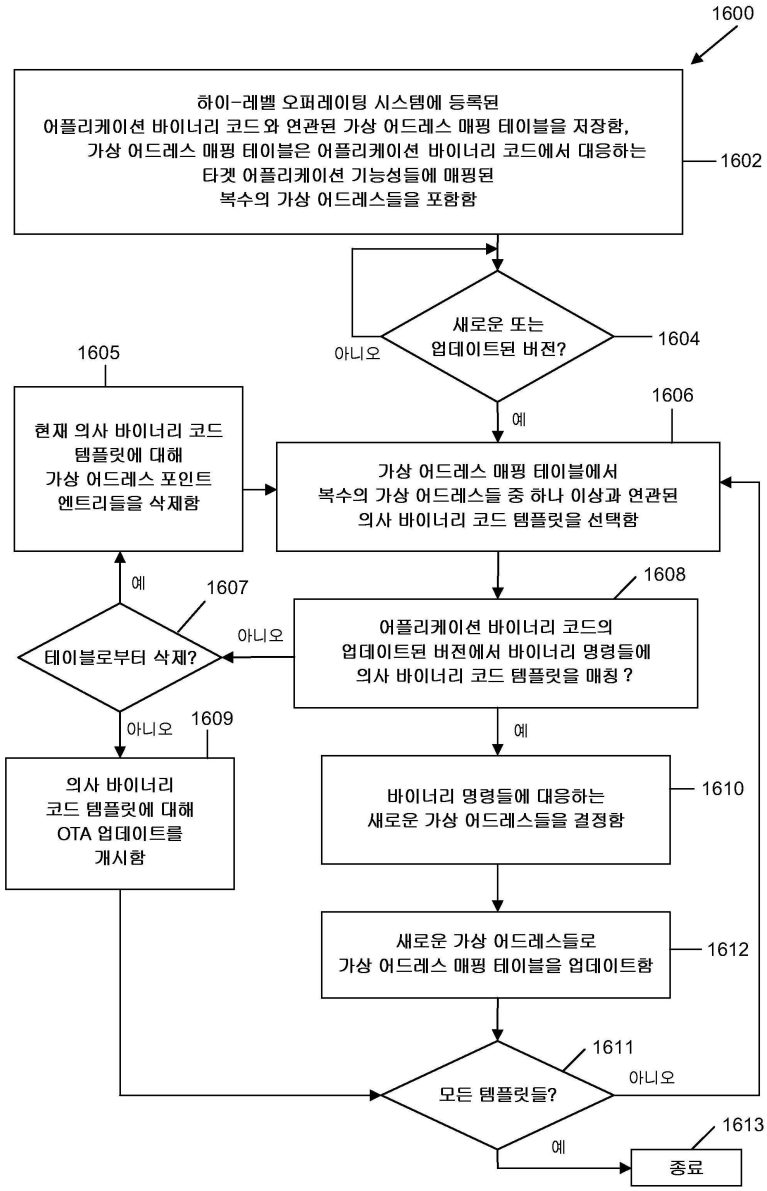


가상 어드레스-투-함수 매핑 테이블 120	
가상 어드레스 302	기능적 의미를 표현하는 매크로 304
0x3473fac8	DOCUMENT_WRITE_FUNCTION_START
0x3473fad4	DOCUMENT_WRITE_1
0x3473fae8	DOCUMENT_WRITE_2
0x30000000	KERNEL_ALLOCATOR_FUNCTION

도면15


의사 바이너리 코드 템플릿 1202
Push {callSave0, CallSave1, ReturnReg}
AddWord reg0, ProgCounter, #Const8bits
LoadWord reg1, [reg0]
LoadWord reg2, [reg1]
.....
.....
LogicalShiftLeft reg1, reg0, #16
AddWord reg2, reg1, #4
LoadWord reg3, [reg2]
AddWord reg4, reg3, ProgCounter
LoadByte ArgumentReg0, [reg4]
AddWord ReturnReg, ProgCounter, #8
BranchDir (ProgCounter + #Const20bits)
.....
.....
AddWord reg0, ProgCounter, #Const8bits
LoadWord reg2, [reg0]
AddWord reg1, reg2, ProgCounter
LoadWord ArgumentReg0, [reg1]
AddWord ReturnReg, ProgCounter, #8
BranchDir (ProgCounter + #Const20bits)
Compare ReturnValueReg0, #0
BranchEQ (ProgCounter + #12)
Compare ReturnValueReg0, #1
BranchNE (ProgCounter + #Const5Bits)
.....
.....
어플리케이션 바이너리의 업데이트된 버전
1204
3133b608: mov r0, r5
3133b60c: mov r1, r4
3133b610: mov r3, r6
3133b614: ldmia.w sp!, {r4, r5, r6, lr}
3133b618: bx r3
3133b61c: push {r4, r5, lr}
3133b620: ldr r4, [pc, #64]
3133b624 ldr r4, [r4, #0]
3133b628: ldr r1, [pc, #60]
3133b62c: lsl r7, r1, #16
3133b630: ldr r3, [r7, #4]
3133b634: add r3, pc
3133b638: ldrb r0, [r3, #0]
3133b63c: bl 314a8abc
3133b640: ldr r1, [pc, #80]
3133b644: add r1, pc
3133b648: ldr r0, [r1, #0]
3133b64c: blx 314a6cbc
3133b650: cmp r0, #0
3133b654: beq 3133b660
3133b658 cmp r0, #1
3133b65c: bne 3133b674
3133b660: bl 313ca924
3133b664: ldr r3, [r0, #0]
3133b668: mov r0, #64
3133b66c: ldr r6, [r3, #0]
3133b670: bl 3146e548
3133b674: movs r1, #0
3133b678: movs r2, #64 ; 0x40
.....

도면16


1600
하이-레벨 오퍼레이팅 시스템에 등록된
어플리케이션 바이너리 코드와 연관된 가상 어드레스 매핑 테이블을 저장함,
가상 어드레스 매핑 테이블은 어플리케이션 바이너리 코드에서 대응하는
타겟 어플리케이션 기능성들에 매핑된
복수의 가상 어드레스들을 포함함
1602
새로운 또는
업데이트된 버전?
1604
아니오
예
1605
현재 의사 바이너리 코드
템플릿에 대해
가상 어드레스 포인트
엔트리들을 삭제함
1606
가상 어드레스 매핑 테이블에서
복수의 가상 어드레스들 중 하나 이상과 연관된
의사 바이너리 코드 템플릿을 선택함
예
1607
테이블로부터 삭제?
아니오
1608
어플리케이션 바이너리 코드의
업데이트된 버전에서 바이너리 명령들에
의사 바이너리 코드 템플릿을 매칭 ?
아니오
1609
예
의사 바이너리
코드 템플릿에 대해
OTA 업데이트를
개시함
바이너리 명령들에 대응하는
새로운 가상 어드레스들을 결정함
1610
새로운 가상 어드레스들로
가상 어드레스 매핑 테이블을 업데이트함
1612
1611
모든 템플릿들?
아니오
예
1613
종료