(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2015년01월08일
(11) 등록번호 10-1477050
(24) 등록일자 2014년12월22일

(51) 국제특허분류(Int. Cl.)
G06F 21/56 (2013.01) *G06F 9/30* (2006.01)
(21) 출원번호 10-2014-0064560
(22) 출원일자 2014년05월28일
심사청구일자 2014년05월28일
(56) 선행기술조사문헌
KR1020130101956 A
KR1020110046850 A
KR1020130101657 A
KR1020110100923 A

(73) 특허권자
충남대학교산학협력단
대전광역시 유성구 대학로 99 (궁동, 충남대학교)
(72) 발명자
김동우
대전 유성구 문화원로 119, 809호 (봉명동, 봉명시티빌)
정우탁
대전 유성구 대학로 27, 1707호 (봉명동, 유성현대리조텔)
류재철
대전 유성구 어은로 57, 132동 801호 (어은동, 한빛아파트)
(74) 대리인
권혁수, 오세준, 송윤호

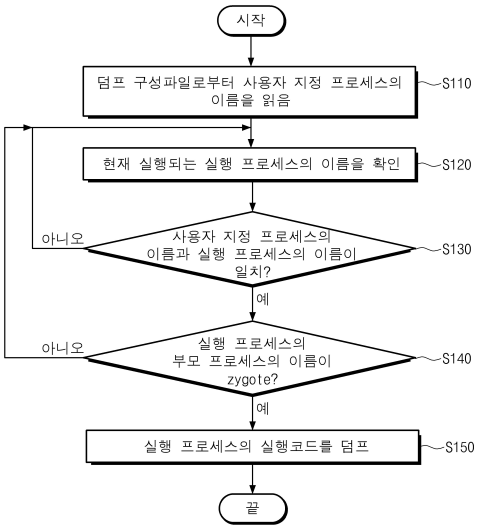
전체 청구항 수 : 총 5 항 심사관 : 구본재

(54) 발명의 명칭 **메모리 덤프 기법을 이용한 어플리케이션의 실행코드 추출 방법**

(57) 요 약

본 발명의 실시 예에 따른 실행코드 추출 방법은, 스토리지 메모리의 덤프 구성파일로부터 사용자 지정 프로세스의 이름을 읽는 단계; 상기 에뮬레이터에서 실행되고 있는 실행 프로세스의 이름을 확인하는 단계; 상기 사용자 지정 프로세스의 이름과 상기 실행 프로세스의 이름을 비교하여 일치 여부를 확인하는 단계; 상기 비교 결과 이름이 일치하는 경우, 상기 실행 프로세스의 부모 프로세스의 이름이 자이고트(zygote) 인지 확인하는 단계; 그리고 상기 확인 결과 상기 부모 프로세스의 이름이 자이고트인 경우, 상기 실행 프로세스의 실행코드를 상기 스토리지 메모리의 설정된 디렉토리에 덤프하는 단계를 포함한다.

대 표 도 - 도2



이 발명을 지원한 국가연구개발사업
과제고유번호 13-912-06-003
부처명 미래창조과학부
연구관리전문기관 한국방송통신전파진흥원
연구사업명 방송통신융합미디어원천기술개발
연구과제명 안드로이드 신규 취약점 탐지를 위한 모바일 소프트웨어 보안 테스팅 도구 개발
기 여 율 1/2
주관기관 충남대학교 산학협력단
연구기간 2013.04.01 ~ 2016.02.28
이 발명을 지원한 국가연구개발사업
과제고유번호 2014-H0301-14-1010 / 1415128814
부처명 미래창조과학부
연구관리전문기관 정보통신산업진흥원
연구사업명 정보통신기술인력양성
연구과제명 클라우드 환경의 스마트 기기와 서비스 보안 기술개발 및 연구 인력양성
기 여 율 1/2
주관기관 숭실대학교산학협력단
연구기간 2013.01.01 ~ 2013.12.31

특허청구의 범위

청구항 1

안드로이드 플랫폼에서 실행되는 에뮬레이터가 실행코드의 복호화 후 워킹 메모리로 로드되는 순간 상기 워킹 메모리를 스토리지 메모리로 덤프하여 실행코드를 추출하는 방법에 있어서,

상기 스토리지 메모리의 덤프 구성파일로부터 사용자 지정 프로세스의 이름을 읽는 단계;

상기 에뮬레이터에서 실행되고 있는 실행 프로세스의 이름을 확인하는 단계;

상기 사용자 지정 프로세스의 이름과 상기 실행 프로세스의 이름을 비교하여 일치 여부를 확인하는 단계;

상기 비교된 이름이 일치하는 경우, 상기 실행 프로세스의 부모 프로세스의 이름이 자이고트(zygote) 인지 확인하는 단계; 그리고

상기 확인 결과 상기 부모 프로세스의 이름이 자이고트 인 경우, 상기 실행 프로세스의 실행코드를 상기 스토리지 메모리의 설정된 디렉토리에 덤프하는 단계를 포함하는 방법.

청구항 2

제 1 항에 있어서,

상기 사용자 지정 프로세스의 이름은 실행코드를 추출하고자 하는 어플리케이션의 패키지 이름인 방법.

청구항 3

제 1 항에 있어서,

상기 실행 프로세스의 이름을 확인하는 단계가,

실행 프로세스의 아이디(PID) 값과 부모 프로세스의 아이디(PPID) 값을 획득하는 단계; 그리고

상기 획득된 프로세스의 아이디(PID) 값 및 상기 획득된 부모 프로세스의 아이디(PPID) 값을 이용하여 파일 시스템에서 상기 실행 프로세스의 이름을 확인하는 단계를 포함하는 방법.

청구항 4

제 3 항에 있어서,

상기 실행 프로세스의 이름을 확인하는 단계가 상기 파일 시스템의 커맨드라인(cmdline) 파일을 읽어 상기 실행 프로세스의 이름을 확인하는 방법.

청구항 5

제 1 항에 있어서,

덤프된 실행코드를 디컴파일 도구를 이용하여 소스코드로 변환하는 단계를 더 포함하는 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 어플리케이션 분석 방법에 관한 것으로, 좀 더 구체적으로는 메모리 덤프 기법을 이용하여 안드로이드 환경에서 구동하는 어플리케이션의 실행코드를 추출하기 위한 방법에 관한 것이다.

배 경 기 술

[0002] 일반적으로, 안드로이드 환경에서 구동되는 어플리케이션의 경우 패키지 파일 내의 classes.dex 라는 단일의 실행파일을 실행하여 구동된다. 이러한 실행파일은 바이트코드 형태이므로 디컴파일 기법을 이용하여 소스코드로

변환하는 것이 용이하다.

[0003] 이와 같은 소스코드가 노출되는 것을 방지하기 위해 다양한 난독화 기법 및 안티분석 기법들이 연구되고 있다. 하지만 정상적인 어플리케이션 뿐만 아니라 악성 어플리케이션의 경우에도 이러한 난독화 기법 또는 안티분석 기법을 이용하여 소스코드의 노출을 보호하고 있다. 또한, 최근 발견되는 일부 악성 어플리케이션의 경우 실행코드의 암호화 및 안티디버깅, 안티 디컴파일링 기법을 이용하여 기존의 분석도구 및 방법으로는 악성 어플리케이션의 상세분석이 어렵게 하고 있다. 따라서 이러한 악성 어플리케이션에 적용된 보호기법을 우회하여 실행코드를 확보할 수 있는 방법이 요구된다.

발명의 내용

해결하려는 과제

[0004] 상술한 문제점들을 해결하기 위한 본 발명의 목적은 메모리 덤프 기법을 이용하여 로우레벨에서 실행코드를 확보하기 위한 실행코드 추출 방법을 제공하는 것이다.

과제의 해결 수단

[0005] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 실행코드 추출 방법은, 스토리지 메모리의 덤프 구성파일로부터 사용자 지정 프로세스의 이름을 읽는 단계; 상기 에뮬레이터에서 실행되고 있는 실행 프로세스의 이름을 확인하는 단계; 상기 사용자 지정 프로세스의 이름과 상기 실행 프로세스의 이름을 비교하여 일치 여부를 확인하는 단계; 상기 비교 결과 이름이 일치하는 경우, 상기 실행 프로세스의 부모 프로세스의 이름이 자이고트(zygote) 인지 확인하는 단계; 그리고 상기 확인 결과 상기 부모 프로세스의 이름이 자이고트인 경우, 상기 실행 프로세스의 실행코드를 상기 스토리지 메모리의 설정된 디렉토리에 덤프하는 단계를 포함한다.

[0006] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 실행코드 추출 방법은, 상기 사용자 지정 프로세스의 이름은 실행코드를 추출하고자 하는 어플리케이션의 패키지 이름이다.

[0007] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 실행코드 추출 방법은, 상기 실행 프로세스의 이름을 확인하는 단계가, 실행 프로세스 ID(PID)값과 부모 프로세스 ID(PPID)을 획득하는 단계; 그리고 상기 획득된 프로세스 ID값 및 상기 획득된 부모 프로세스 ID값을 이용하여 파일 시스템에서 상기 실행 프로세스의 이름을 확인하는 단계를 포함한다.

[0008] 상기 목적을 달성하기 위한 본 발명의 실시 예에 따른 실행코드 추출 방법은, 상기 실행 프로세스의 이름을 확인하는 단계가 상기 파일 시스템의 커맨드라인(cmdline) 파일을 읽어 상기 실행 프로세스의 이름을 확인한다.

[0009] 상기 목적을 달성하기 위한 본 발명의 실시 예에 다른 실행코드 추출 방법은, 상기 저장된 실행코드를 디컴파일 도구를 이용하여 소스코드로 변환하는 단계를 더 포함한다.

발명의 효과

[0010] 본 발명의 실시 예에 따르면, 암호화 등의 안티분석 기술이 적용된 악성 앱의 실행파일을 워킹 메모리로부터 복호화된 형태로 추출할 수 있다.

[0011] 본 발명의 실시 예에 따르면, 로우레벨에서 악성 앱의 실행파일을 추출할 수 있다.

[0012] 본 발명의 실시 예에 따르면, 악성 앱의 실행파일을 추출하는 시간을 단축할 수 있다.

도면의 간단한 설명

[0013] 도 1은 본 발명에 따른 실행코드 추출 방법을 설명하기 위한 참고 구성도이다.

도 2는 본 발명에 따른 실행코드 추출 방법의 일 실시 예를 보여주는 순서도이다.

도 3은 본 발명에 따른 실행코드 추출 방법의 다른 실시 예를 보여주는 순서도이다.

도 4 내지 6은 본 발명에 따른 실행코드 추출 방법을 설명하기 위한 참고도이다.

발명을 실시하기 위한 구체적인 내용

[0014] 앞의 일반적인 설명 및 다음의 상세한 설명들은 모두 청구된 발명의 부가적인 설명을 제공하기 위한 예시적인

것이다. 그러므로 본 발명은 여기서 설명되는 실시 예에 한정되지 않고 다른 형태로 구체화될 수도 있다. 여기서 소개되는 실시 예는 개시된 내용이 철저하고 완전해 질 수 있도록 그리고 당업자에게 본 발명의 사상이 충분히 전달될 수 있도록 하기 위해 제공되는 것이다.

[0015] 본 출원에서 사용한 용어는 단지 특정한 실시 예를 설명하기 위해 사용된 것으로, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 출원에서, "포함하다" 또는 "가지다" 등의 용어는 명세서상에 기재된 특징, 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성요소, 부품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0016] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 가진 것으로 해석되어야 하며, 본 출원에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0017] 본 명세서에서, 어떤 부분이 어떤 구성요소를 포함한다고 언급되는 경우에, 이는 그 외의 다른 구성요소를 더 포함할 수도 있다는 것을 의미한다. 또한, 여기에서 설명되고 예시되는 각 실시 예는 그것의 상보적인 실시 예도 포함한다. 이하, 본 발명의 실시 예를 첨부된 도면을 참조하여 상세하게 설명한다.

[0018] 본 발명에 따른 실행코드 추출 방법은 안드로이드 환경에서 구동되는 어플리케이션(이하, '앱')의 실행코드를 추출할 수 있다. 일반적으로 앱이 안드로이드 플랫폼에서 구동되는 경우, 앱의 실행파일이, 예를 들어 램(RAM)과 같은 워킹메모리로 로딩되어 실행된다. 예를 들어 안드로이드 플랫폼 소스 버전 4.0.1의 경우, 실행파일을 워킹메모리로 로드하기 위해 앱은 DexClassLoader API(Application Programming Interface)를 호출한다. DexClassLoader는 최종적으로 달빅가상머신(Dalvik Virtural Machine)의 네이티브 코드 계층에 dvmDexFileOpenFromFd 함수를 호출하게 된다. dvmDexFileOpenFromFd 함수는 실행파일을 네이티브 라이브러리의 복호화 루틴 이후에 워킹메모리로 로드한다.

[0019] 따라서, dvmDexFileOpenFromFd 함수에 덤프코드를 삽입하여 dvmDexFileOpenFromFd 함수가 실행파일을 워킹메모리에 로드한 직후 워킹메모리를 덤프하여 로드된 실행파일을 확보할 수 있다.

[0020] 하지만, 상술한 바와 같이 덤프코드를 이용하여 워킹메모리로 로드되는 모든 실행파일을 추출하는 경우 예외오류(exception error)가 발생할 수 있다. 예를 들면, 덤프코드가 안티분석 루틴을 가지고 있는 자식 프로세스의 메모리로 접근을 시도하는 경우 실행코드를 추출하고자 하는 목표 프로세스가 안티분석 기능에 의해 비정상적으로 종료될 수 있다.

[0021] 따라서, 본 발명에 따른 실행코드 추출 방법은 자식 프로세스가 아닌 앱의 부모 프로세스(parent process)에 의해 워킹 메모리로 로드되는 실행파일만을 추출하여 실행코드를 확보한다.

[0022] 도 1은 본 발명에 따른 실행코드 추출 방법을 설명하기 위한 참고 구성도이다. 도 1을 참조하여 본 발명에 따른 실행코드 추출 방법을 간략히 설명한다. 본 발명에 따른 실행코드 추출 방법은 덤프코드가 삽입된 에뮬레이터(100)에 의해 수행된다. 에뮬레이터(100)가 스토리지 메모리(200)에 저장된 덤프 구성파일(dump.conf)과 파일시스템의 cmdline 파일을 이용하여 앱의 실행코드(code_exe)를 추출한 후, 이를 스토리지 메모리(200)에 저장한다. 저장된 실행코드(code_exe)는 디컴파일러(400)를 통해 디컴파일 되어 소스코드(code_sou)로 변환되어 출력될 수 있다.

[0023] 도 2는 본 발명에 따른 실행코드 추출 방법의 일 실시 예를 도시한 순서도이다. 이하, 도 2를 참조하여 본 발명에 따른 실행코드 추출 방법을 보다 자세히 설명한다.

[0024] 먼저, 본 발명에 따른 실행코드 추출 방법은 덤프코드가 삽입된 에뮬레이터(100)에서 분석대상 앱을 실행키는 것에 의해 수행된다. 에뮬레이터(100)는 달빅가상머신을 포함한다. 에뮬레이터(100)에서 분석대상 앱을 실행시키는 경우, 상술한 바와 같이 분석대상 앱은 실행파일을 워킹 메모리에 로드하기 위해 DexclassLoader API를 호출한다. 호출된 DexclassLoader API는 최종적으로 달빅가상머신에 dvmDexFileOpenFromFd 함수를 호출한다. 호출된 dvmDexFileOpenFromFd 함수가 실행파일을 워킹메모리에 로드한다. 본 발명에 따른 실행코드 추출 방법은 호출된 dvmDexFileOpenFromFd 함수가 복호화된 실행파일을 워킹메모리에 로드한 직후 그 워킹메모리를 덤프한다. 또한, 워킹메모리로 로드되는 프로세스의 실행파일 중 설정된 프로세스 이름을 가지는 프로세스의 실

행파일만을 덤프한다.

[0025] 분석대상 앱이 에뮬레이터(100)에서 실행되는 경우, 에뮬레이터(100)는 스토리지 메모리(200)에 저장된 덤프 구성파일(dump.conf)로부터 사용자 지정 프로세스 이름을 읽어들인다(S110). 사용자 지정 프로세스 이름은 워킹메모리로 로드되는 실행파일 중 덤프를 원하는 실행파일만을 덤프하기 위한 것이다. 사용자 지정 프로세스 이름은 분석대상 앱의 패키지 이름일 수 있다. 사용자 지정 프로세스 이름은 예를 들어, 앱의 AndroidManifest.xml 파일을 통해 획득될 수 있다. 앱의 프로세스의 이름은 이에 한정되지 않고 다양한 방법으로 획득될 수 있다. 획득된 앱의 패키지 이름은 스토리지 메모리의 덤프 구성파일(dump.conf)에 저장될 수 있다.

[0026] 다음으로, 에뮬레이터(100)는 에뮬레이터(100)에서 실행되고 있는 실행 프로세스의 이름을 확인한다(S120). 에뮬레이터(100)에서 앱이 구동되는 경우, dvmDexFileOpenFromFd 함수에 의해 시스템 제공 기본 실행파일을 포함한 다수의 실행파일이 워킹메모리로 로드된다. 에뮬레이터(100)는 dvmDexFileOpenFromFd 함수에 의해 워킹메모리로 로드되는 프로세스의 이름을 확인한다. 프로세스의 이름은 로드되는 프로세스의 ID(PID) 값을 이용하여 확인할 수 있다.

[0027] 실행 프로세스의 이름을 확인한 후, 에뮬레이터(100)는 사용자 지정 프로세스의 이름과 실행 프로세스의 이름이 일치하는지를 비교한다(S130). 비교 결과 일치하지 않는 경우, 에뮬레이터(100)는 dvmDexFileOpenFromFd 함수가 워킹메모리로 로드하는 다른 프로세스의 이름을 확인한다.

[0028] 비교 결과 이름이 일치하는 경우, 에뮬레이터(100)는 실행 프로세스의 부모 프로세스의 이름이 'zygote' 인지 확인한다(S140). 'zygote' 프로세스는 앱의 구동 초기에 실행되는 프로세스로서 실행되는 앱에 필요한 프레임워크에 포함된 클래스와 플랫폼 자원을 담아 필요한 클래스를 로딩하고 메인 프로세스를 실행한다. 'zygote' 프로세스로 실행된 앱들의 부모 프로세스 ID(PPID) 값은 'zygote' 프로세스의 ID(PID)와 같다.

[0029] 비교 결과 실행 프로세스의 부모 프로세스의 이름이 'zygote' 인 경우, 에뮬레이터는 워킹메모리로 로드된 해당 실행 프로세스의 실행코드를 덤프한다(S150).

[0030] 도 3은 본 발명에 따른 실행코드 추출 방법의 다른 실시 예를 보여주는 순서도이다. 도 3을 참조하면, 본 발명에 따른 실행코드 추출 방법의 다른 실시 예를 보다 자세히 설명한다. 도 3의 S210, S240, S250, 및 S260은 각각 도 1의 S110, S130, S140, 및 S150과 동일하므로 그 설명을 생략한다.

[0031] 에뮬레이터(100)는 dvmDexFileOpenFromFd 함수를 호출하여 실행코드를 워킹메모리로 로드하는 실행 프로세스의 ID(PID)값과 부모 프로세스의 ID(PPID) 값을 획득한다(S220). 프로세스 ID(PID) 값과 부모 프로세스 ID(PPID) 값은 예를 들어, 'get pid()' 및 'get ppid()' 함수를 통해 획득될 수 있다.

[0032] 다음으로, 에뮬레이터(100)는 획득된 프로세스 ID(PID)값 및 부모 프로세스 ID(PPID)값을 이용하여 파일시스템(300)으로부터 실행 프로세스의 이름을 확인한다(S230). 에뮬레이터(100)는 파일 시스템(300)의 cmdline 파일을 읽어 실행 프로세스의 이름을 확인할 수 있다.

[0033] 에뮬레이터(100)는 S260 단계에서 덤프한 실행코드를 저장할 수 있다(S270). 덤프한 실행코드는 스토리지 메모리(200)의 지정된 디렉토리에 저장될 수 있다.

[0034] 에뮬레이터(100)는 스토리지 메모리(200)에 저장된 실행코드를 디컴파일러(400)를 통해 디컴파일하여 소스코드로 변환할 수 있다. 저장된 실행코드가 DEX 포맷인 경우 예를 들어 Dex2Jar, JD-GUI 등의 도구를 이용하여 디컴파일될 수 있다. 저장된 실행코드가 ODEX(optimized DEX)인 경우 Baksmali, Smali 등의 도구를 이용하여 DEX 포맷으로 변환한 후 상술한 디컴파일 과정을 거쳐 소스코드로 변환될 수 있다.

[0035] 도 4는 본 발명에 따른 실행코드 추출 방법을 이용하여 샘플 앱에서 추출한 실행코드를 예시적으로 보여준다. 도 4를 참조하면, 총 3개(①,②,③)의 실행파일이 추출되었다. 추출된 실행파일을 분석해 보면, 첫 번째 실행파일(①)은 'android.test.runner.jar'에 있는 디폴트 실행파일이다. 두 번째 실행파일(②)은 샘플 앱의 디폴트 실행파일이다. 세 번째 실행파일(③)은 네이티브 라이브러리에 의해 복호화된 후 동적으로 로드된 외부 실행파일임을 확인할 수 있다.

[0036] 도 5는 도 4에서 추출한 세 번째 실행파일을 디컴파일한 소스코드를 보여준다. 도 4에서 표시된 영역(①)을 참조하면, 추출된 실행파일은 개인의 은행정보를 가로채기 위한 악성앱의 실행파일임을 확인할 수 있다.

[0037] 도 6은 도 5에서 변환된 소스코드의 일부를 보여준다. 도 6에서 표시된 영역(①)은 악성앱에 의해 획득된 개인의 은행정보가 전송되는 서버의 주소를 보여준다. 따라서, 본 발명에 따른 실행코드 추출 방법을 이용하는

경우, 악성 앱의 종류뿐만 아니라 추출된 개인정보 등이 전송되는 서버의 주소까지 알 수 있기 때문에 악성 앱에 대한 구체적인 대응을 할 수 있다.

[0038] 한편, 본 발명의 범위 또는 기술적 사상을 벗어나지 않고 본 발명의 구조가 다양하게 수정되거나 변경될 수 있음은 이 분야에 숙련된 자들에게 자명하다. 상술한 내용을 고려하여 볼 때, 만약 본 발명의 수정 및 변경이 아래의 청구항들 및 동등물의 범주 내에 속한다면, 본 발명이 이 발명의 변경 및 수정을 포함하는 것으로 여겨진다.

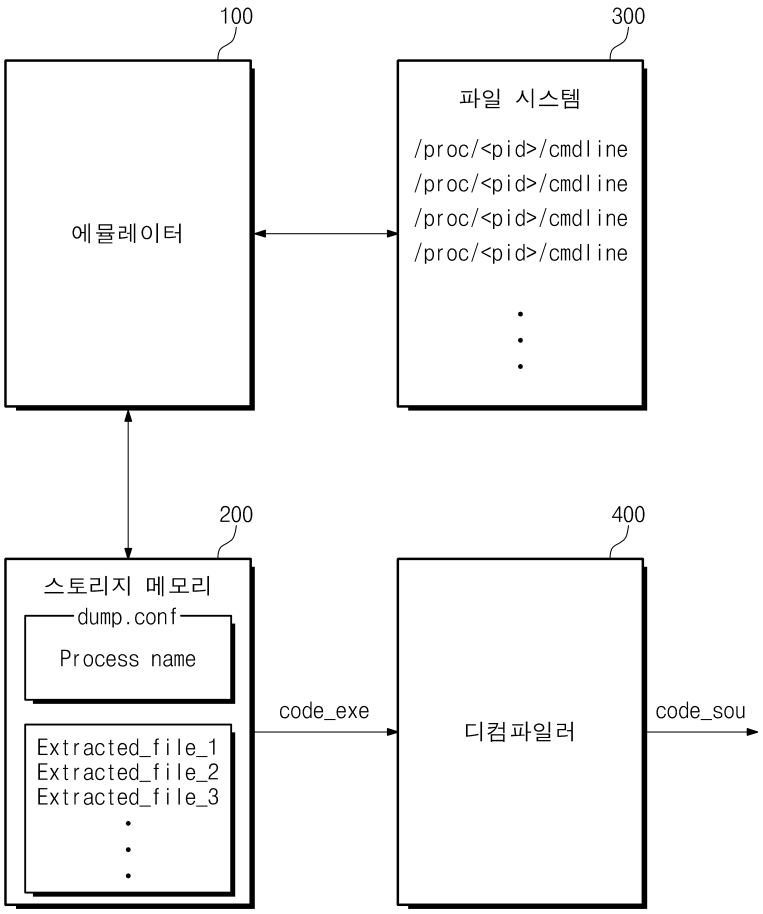
부호의 설명

[0039] 100: 에뮬레이터 200: 스토리지 메모리

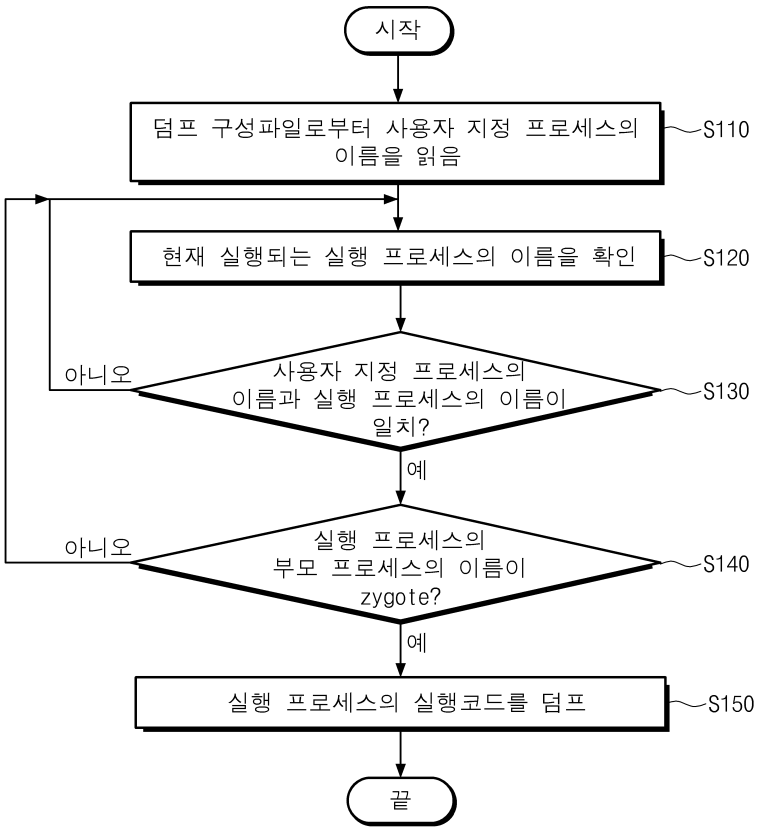
300: 파일시스템 400: 디컴파일러

도면

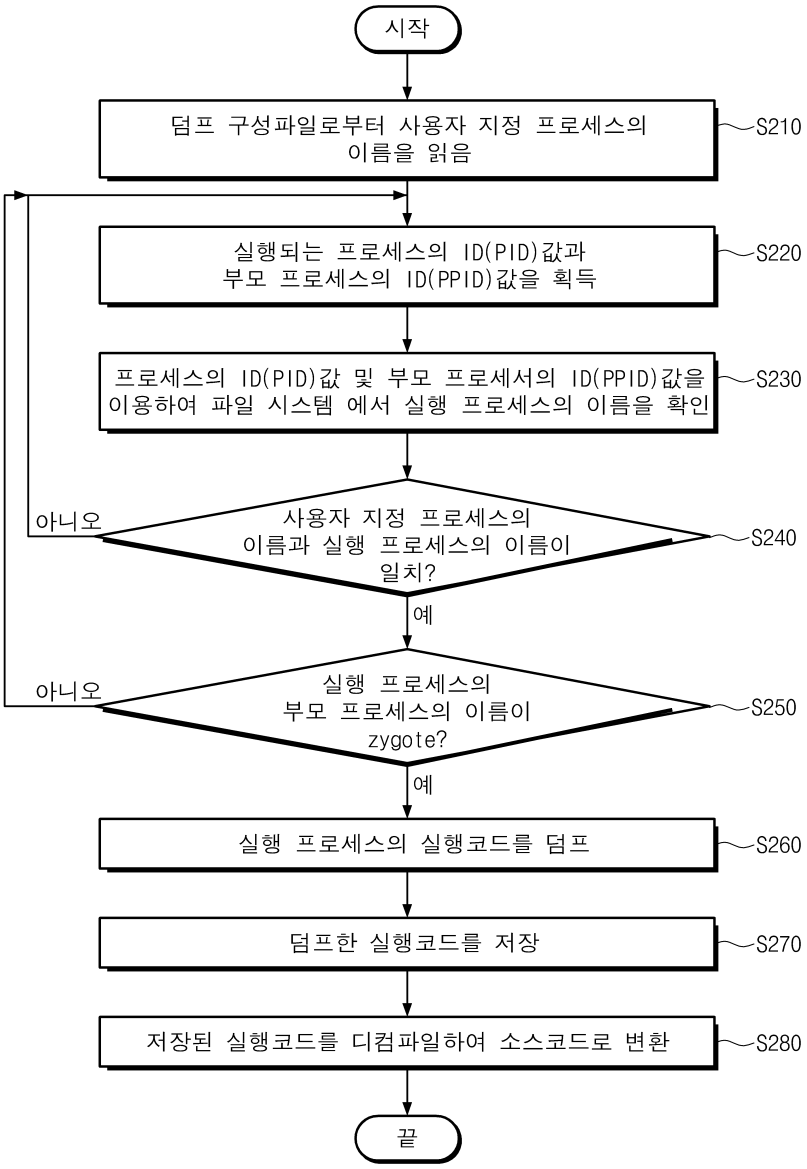
도면1



도면2


시작
덤프 구성파일로부터 사용자 지정 프로세스의 이름을 읽음
S110
현재 실행되는 실행 프로세스의 이름을 확인
S120
사용자 지정 프로세스의 이름과 실행 프로세스의 이름이 일치?
S130
아니오
예
실행 프로세스의 부모 프로세스의 이름이 zygote?
S140
아니오
예
실행 프로세스의 실행코드를 덤프
S150
끝

도면3

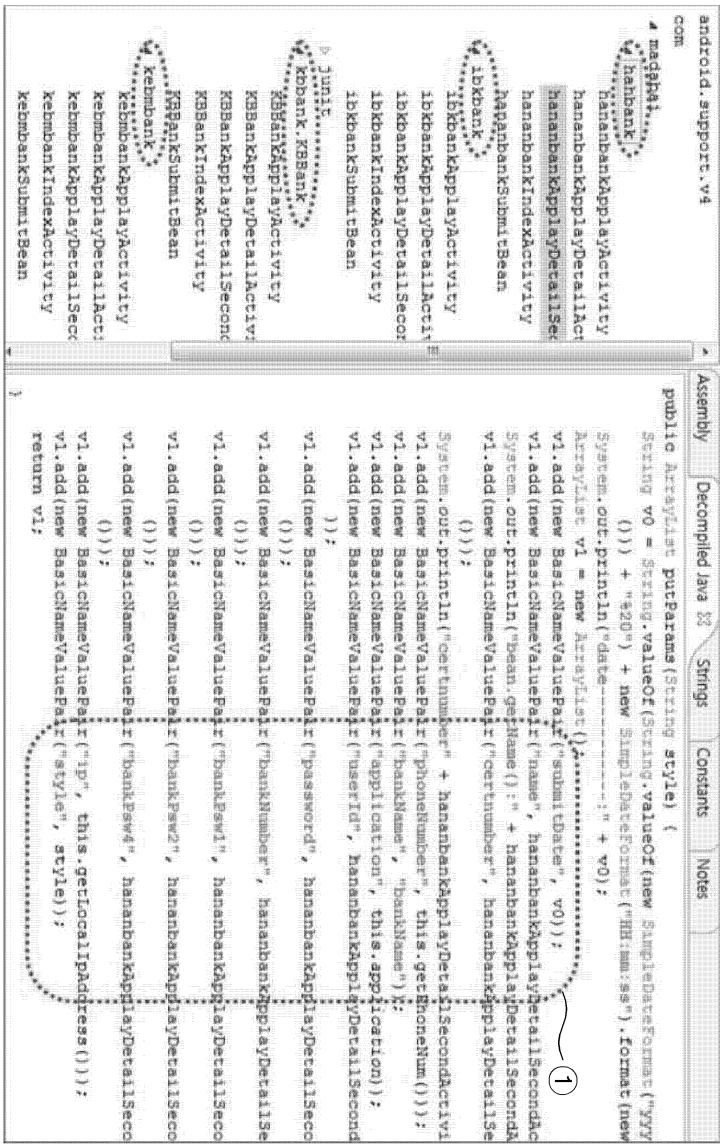

시작
덤프 구성파일로부터 사용자 지정 프로세스의 이름을 읽음
S210
실행되는 프로세스의 ID(PID)값과 부모 프로세스의 ID(PPID)값을 획득
S220
프로세스의 ID(PID)값 및 부모 프로세서의 ID(PPID)값을 이용하여 파일 시스템 에서 실행 프로세스의 이름을 확인
S230
사용자 지정 프로세스의 이름과 실행 프로세스의 이름이 일치?
S240
아니오
예
실행 프로세스의 부모 프로세스의 이름이 zygote?
S250
아니오
예
실행 프로세스의 실행코드를 덤프
S260
덤프한 실행코드를 저장
S270
저장된 실행코드를 디컴파일하여 소스코드로 변환
S280
끝

도면4

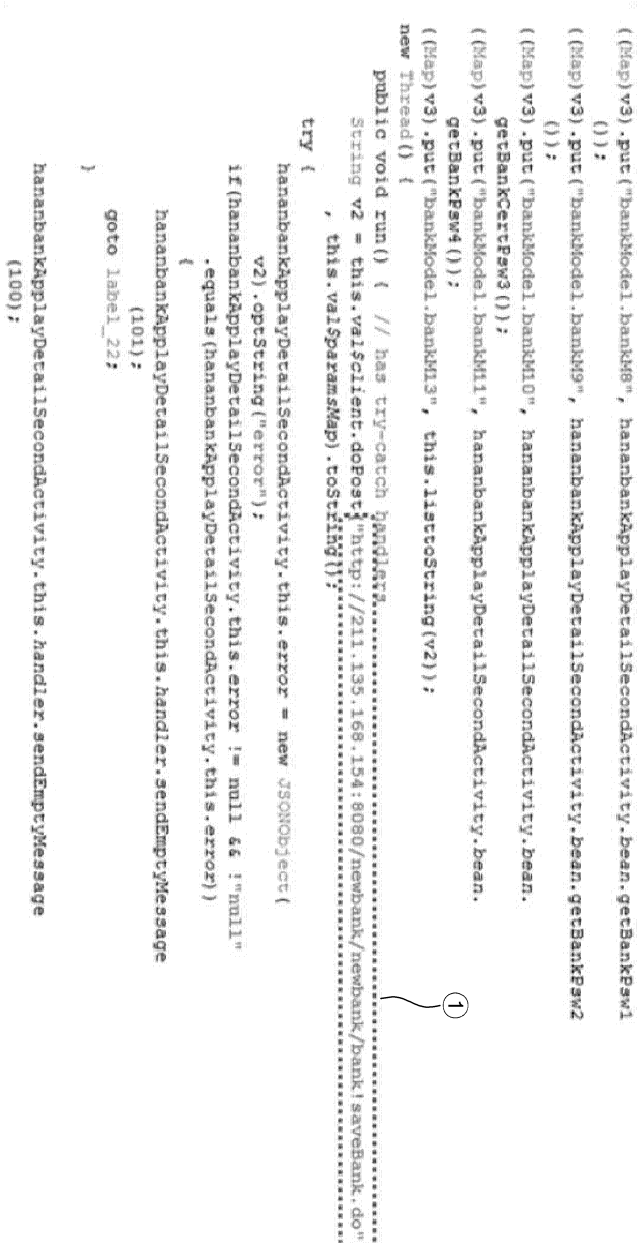


Application	Tag	Text
com.madabai.kim	scott	Couldn't open /system/framework/android.test.runner.odex: No such file or directory
com.madabai.kim	scott	dvmJarFileOpen: Checking cache for /system/framework/android.test.runner.jar (/data/
		.jar@classes.dex)
com.madabai.kim	scott	Current process ID : 724 Parent process ID : 37
com.madabai.kim	scott	Designated Process Name : com.madabai.kim
com.madabai.kim	scott	Current Process Name : com.madabai.kim
com.madabai.kim	scott	Process Name Correct! Memory Dumping : com.madabai.kim
com.madabai.kim	scott	com.madabai.kim is being saved at /sdcard/scott/com.madabai.kim_168528.odex
com.madabai.kim	scott	Memory Addr : 0x47ec3000, Memory Size(bytes) : 168528
com.madabai.kim	scott	+++ DEX 0x1b0f18: allocateAux 2346+352+1458+241 * 4 = 17588 bytes
com.madabai.kim	scott	Successfully opened 'classes.dex' in '/system/framework/android.test.runner.jar'
com.madabai.kim	scott	Opening DEX file '/system/framework/android.test.runner.jar' (Jar)
com.madabai.kim	scott	Couldn't open /data/app/com.madabai.kim-1.odex: No such file or directory
com.madabai.kim	scott	dvmJarFileOpen: Checking cache for /data/app/com.madabai.kim-1.apk (/data/dalvik-cac
com.madabai.kim	scott	Current process ID : 724 Parent process ID : 37
com.madabai.kim	scott	Designated Process Name : com.madabai.kim
com.madabai.kim	scott	Current Process Name : com.madabai.kim
com.madabai.kim	scott	Process Name Correct! Memory Dumping : com.madabai.kim
com.madabai.kim	scott	com.madabai.kim is being saved at /sdcard/scott/com.madabai.kim_20088.odex
com.madabai.kim	scott	Memory Addr : 0x47eed000, Memory Size(bytes) : 20088
com.madabai.kim	scott	+++ DEX 0x1f8cc8: allocateAux 374+73+184+20 * 4 = 2604 bytes
com.madabai.kim	scott	Successfully opened 'classes.dex' in '/data/app/com.madabai.kim-1.apk'
com.madabai.kim	scott	Opening DEX file '/data/app/com.madabai.kim-1.apk' (Jar)
com.madabai.kim	scott	Couldn't open /data/data/com.madabai.kim/.cache/classes.odex: No such file or direct
com.madabai.kim	scott	dvmJarFileOpen: Checking cache for /data/data/com.madabai.kim/.cache/classes.jar (/d
com.madabai.kim	scott	Current process ID : 724 Parent process ID : 37
com.madabai.kim	scott	Designated Process Name : com.madabai.kim
com.madabai.kim	scott	Current Process Name : com.madabai.kim
com.madabai.kim	scott	Process Name Correct! Memory Dumping : com.madabai.kim
com.madabai.kim	scott	com.madabai.kim is being saved at /sdcard/scott/com.madabai.kim_2265048.odex
com.madabai.kim	scott	Memory Addr : 0x4857b000, Memory Size(bytes) : 2265048
com.madabai.kim	scott	+++ DEX 0x199e48: allocateAux 19217+2362+16811+6004 * 4 = 177576 bytes
com.madabai.kim	scott	Successfully opened 'classes.dex' in '/data/data/com.madabai.kim/.cache/classes.jar'
com.madabai.kim	scott	Opening DEX file '/data/data/com.madabai.kim/.cache/classes.jar' (Jar)

도면5


android.support.v4
com
madapa
hahnbank
hananbankApplayActivity
hananbankApplayDetailAct
hananbankApplayDetailSec
hananbankIndexActivity
hananbankSubmitBean
ibkbank
ibkbankApplayActivity
ibkbankApplayDetailActiv
ibkbankApplayDetailSecon
ibkbankIndexActivity
ibkbankSubmitBean
junit
kbbank.KBBank
KBBankApplayActivity
KBBankApplayDetailActivi
KBBankApplayDetailSecond
KBBankIndexActivity
KBBankSubmitBean
kebmbank
kebmbankApplayActivity
kebmbankApplayDetailAct
kebmbankApplayDetailSec
kebmbankIndexActivity
kebmbankSubmitBean
Assembly
Decompiled Java
Strings
Constants
Notes
public ArrayList putParams(String style) {
String v0 = String.valueOf(new SimpleDateFormat("yyy
()) + "%20") + new SimpleDateFormat("HH:mm:ss").format(new
System.out.println("date----------:" + v0);
ArrayList v1 = new ArrayList();
v1.add(new BasicNameValuePair("submitDate", v0));
v1.add(new BasicNameValuePair("name", hananbankApplayDetailSecondAc
System.out.println("bean.getName():" + hananbankApplayDetailSecondA
v1.add(new BasicNameValuePair("certnumber", hananbankApplayDetailSe
());
System.out.println("certnumber" + hananbankApplayDetailSecondActivi
v1.add(new BasicNameValuePair("phoneNumber", this.getPhoneNum());
v1.add(new BasicNameValuePair("bankName", "bankName");
v1.add(new BasicNameValuePair("application", this.application));
v1.add(new BasicNameValuePair("userId", hananbankApplyDetailSecond
));
v1.add(new BasicNameValuePair("password", hananbankApplayDetailSeco
());
v1.add(new BasicNameValuePair("bankNumber", hananbankApplayDetailSe
());
v1.add(new BasicNameValuePair("bankPsw1", hananbankApplayDetailSeco
());
v1.add(new BasicNameValuePair("bankPsw2", hananbankApplayDetailSeco
());
v1.add(new BasicNameValuePair("bankPsw4", hananbankApplayDetailSeco
());
v1.add(new BasicNameValuePair("ip", this.getLocalIpAddress());
v1.add(new BasicNameValuePair("style", style));
return v1;
}
①

도면6



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 제5항

【변경전】

저장된 실행코드

【변경후】

덤프된 실행코드