



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2017-0101693
(43) 공개일자 2017년09월06일

(51) 국제특허분류(Int. Cl.)

G06F 11/36 (2006.01)

(52) CPC특허분류

G06F 11/3604 (2013.01)

G06F 11/3624 (2013.01)

(21) 출원번호 10-2016-0024577

(22) 출원일자 2016년02월29일

심사청구일자 2016년02월29일

(71) 출원인

충남대학교산학협력단

대전광역시 유성구 대학로 99 (궁동, 충남대학교)

(72) 발명자

류재철

대전광역시 유성구 어은로 57, 132동 801호

허혜지

대전광역시 유성구 문화원로37번길 48, 208호

김상후

대전광역시 유성구 도안동로 523, 215동 1102호

(74) 대리인

홍성욱, 심경식

전체 청구항 수 : 총 19 항

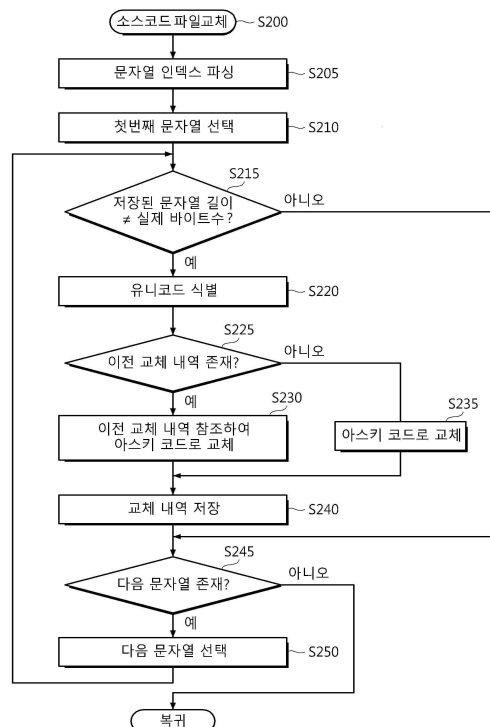
(54) 발명의 명칭 안드로이드 앱 분석 장치 및 그 방법과, 이를 실행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록매체

(57) 요약

안드로이드 앱 분석 장치를 제공한다. 본 발명의 안드로이드 앱 분석 장치는 분석할 안드로이드 앱 파일의 압축을 해제하여 패키징된 다수의 파일들을 분리하거나, 분리된 다수의 파일들을 다시 패키징하여 압축하는 압축 처리부; 상기 분리된 파일들 중 소스 코드를 포함하는 제1 파일을 분석하고, 상기 제1 파일에 포함된 유니코드를

(뒷면에 계속)

대 표 도 - 도3



모두 아스키코드로 교체하여 제2 파일을 생성하고, 상기 앱 파일 내 소스 코드 파일을 상기 제2 파일로 교체한 후 상기 제2 파일을 판독 가능한 형태의 제1 코드로 변환하거나, 상기 제1 코드를 기계어 형태의 제2 코드로 변환하는 코드 처리부; 상기 압축 처리부에서 분리된 다수의 파일들 중 소스코드가 포함된 제1 파일을 추출한 후, 이를 상기 코드 처리부로 전달하여 상기 제1 코드로 변환되도록하고, 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정한 후, 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환한 후 안드로이드 앱 파일로 다시 패키징하도록 상기 코드 처리부 및 상기 압축 처리부를 제어하는 제어부를 포함한다.

(52) CPC특허분류

G06F 11/366 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호 1711032669

부처명 미래창조과학부

연구관리전문기관 정보통신기술진흥센터

연구사업명 정보통신기술인력양성

연구과제명 2015년 SW중심대학(ICT/SW창의연구과정)_충남대

기 여 율 1/1

주관기관 충남대학교 산학협력단

연구기간 2015.10.01 ~ 2016.02.29

명세서

청구범위

청구항 1

안드로이드 앱 분석 장치에 있어서,

분석할 안드로이드 앱 파일의 압축을 해제하여 패키징된 다수의 파일들을 분리하거나, 분리된 다수의 파일들을 다시 패키징하여 압축하는 압축 처리부;

상기 분리된 파일들 중 소스 코드를 포함하는 제1 파일을 분석하고, 상기 제1 파일에 포함된 유니코드를 모두 아스키코드로 교체하여 제2 파일을 생성하고, 상기 앱 파일 내 소스코드 파일을 상기 제2 파일로 교체한 후 상기 제2 파일을 판독 가능한 형태의 제1 코드로 변환하거나, 상기 제1 코드를 기계어 형태의 제2 코드로 변환하는 코드 처리부;

상기 압축 처리부에서 분리된 다수의 파일들 중 소스코드가 포함된 제1 파일을 추출한 후, 이를 상기 코드 처리부로 전달하여 상기 제1 코드로 변환되도록하고, 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정한 후, 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환한 후 안드로이드 앱 파일로 다시 패키징하도록 상기 코드 처리부 및 상기 압축 처리부를 제어하는 제어부를 포함하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 2

제1항에 있어서, 상기 제어부는

상기 압축 처리부에서 다시 패키징된 안드로이드 앱 파일에 개발자 서명을 하여 디버깅앱을 생성한 후 상기 디버깅용 앱을 실행시키는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 3

제1항에 있어서, 상기 안드로이드 앱 분석 장치는

패키징된 다수의 파일들이 압축된 형태로 구현된 안드로이드 앱 파일들을 저장하는 저장부;

상기 제1 코드에 대한 정적 분석이 가능하도록 상기 제1 코드를 표시하는 디스플레이부; 및

상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력을 받아들이는 인터페이스부를 더 포함하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 4

제1항에 있어서, 상기 코드 처리부는

상기 제1 파일에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별하고, 상기 유니코드를 포함하는 문자열을 한 바이트씩 차례로 검사하여 유니코드가 검출되면 그 유니코드를 임의의 아스키코드로 교체하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 5

제4항에 있어서, 상기 코드 처리부는

상기 교체 내역을 별도로 저장하고, 상기 교체 내역을 참조하여 동일한 유니코드 문자열에 대해서는 동일한 아

스키코드 문자열로 교체하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 6

제4항에 있어서, 상기 코드 처리부는

상기 제1 파일에 포함된 모든 문자열들 각각에 대하여 문자열의 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 다른 경우 해당 문자열을 유니코드가 포함된 문자열로 식별하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 7

제6항에 있어서, 상기 코드 처리부는

상기 유니코드를 임의의 아스키코드로 교체한 후 해당 문자열의 헤더 부분에 저장된 문자열 길이를 실제 바이트 수로 변경하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 8

제4항에 있어서, 상기 코드 처리부는

서로 다른 유니코드 각각에 대하여 서로 다른 아스키코드로 교체하는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 9

제4항에 있어서, 상기 코드 처리부는

상기 유니코드가 한글 유니코드 범위에 속하는 경우 해당 바이트를 아스키코드로 변경하지 않는 것을 특징으로 하는 안드로이드 앱 분석 장치.

청구항 10

안드로이드 앱 분석 방법에 있어서,

패키징된 다수의 파일들이 압축된 형태로 구현된 안드로이드 앱 파일로부터 소스코드가 포함된 제1 파일을 추출하는 단계;

상기 제1 파일에 포함된 유니코드를 모두 아스키코드로 교체하여 제2 파일을 생성하고 상기 앱 파일 내 소스코드 파일을 상기 제2 파일로 교체하는 소스코드 파일 교체 단계;

상기 제2 파일을 판독 가능한 형태의 제1 코드로 변환하는 디스어셈블링 단계;

상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정하는 단계;

상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환한 후 안드로이드 앱 파일로 리패키징하는 단계; 및

개발자 서명하여 디버깅용 앱을 생성한 후 상기 디버깅용 앱을 실행하는 동적 분석 단계를 포함하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 11

제10항에 있어서, 상기 소스코드 파일 교체 단계는

상기 제1 파일에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별하는 단계;

상기 문자열 식별 단계에서 유니코드를 포함하는 것으로 식별된 문자열을 한 바이트씩 차례로 검사하여 유니코드를 식별하는 단계; 및

상기 유니코드를 임의의 아스키코드로 교체하는 단계를 포함하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 12

제11항에 있어서, 상기 소스코드 파일 교체 단계는

상기 교체 내역을 별도로 저장하는 단계를 더 포함하고,

상기 아스키코드 교체 단계는

상기 교체 내역을 참조하여, 동일한 유니코드 문자열에 대해서는 동일한 아스키코드 문자열로 교체하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 13

제11항에 있어서, 상기 문자열 식별 단계는

상기 제1 파일에 포함된 문자열 인덱스를 파싱하여 해당 앱에서 사용되는 모든 문자열을 식별하는 단계;

상기 식별된 모든 문자열들 각각에 대하여 문자열의 헤더부분에 저장된 문자열 길이와 실제 바이트 수를 비교하는 단계; 및

상기 비교 결과에 기초하여 상기 모든 문자열들 각각에 대한 유니코드 포함 여부를 결정하는 단계를 포함하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 14

제13항에 있어서, 상기 유니코드 포함 여부 결정 단계는

상기 문자열의 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 서로 다른 경우 해당 문자열에 유니코드가 포함된 것으로 결정하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 15

제11항에 있어서, 상기 유니코드 식별 단계는

상기 검사 결과 해당 바이트의 값이 유니코드 범위에 속하는 경우 해당 바이트를 유니코드로 식별하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 16

제11항에 있어서, 상기 아스키코드 교체 단계는

서로 다른 유니코드 각각에 대하여 서로 다른 아스키코드로 교체하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 17

제11항에 있어서, 상기 아스키코드 교체 단계는

상기 유니코드를 아스키코드로 교체한 후 해당 문자열의 헤더 부분에 저장된 문자열의 길이를 실제 바이트 수로 변경하는 단계를 더 포함하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 18

제11항에 있어서, 상기 아스키코드 교체 단계는

상기 유니코드가 한글 유니코드 범위에 속하는 경우 해당 바이트를 아스키코드로 교체하지 않고 다음 단계로 진행하는 것을 특징으로 하는 안드로이드 앱 분석 방법.

청구항 19

제10항 내지 제18항 중 어느 한 항에 기재된 방법을 수행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록매체.

발명의 설명

기술 분야

[0001] 본 발명은 안드로이드 앱 분석 장치 및 그 방법에 관한 것으로서, 특히, 악의 적으로 생성된 안드로이드 악성 앱을 분석하는 안드로이드 앱 분석 장치 및 그 방법과, 이를 실행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

배경 기술

[0002] 안드로이드(Android)는 휴대폰용 운영체제, 미들웨어, 응용프로그램을 하나로 묶은 소프트웨어 플랫폼으로서, 안드로이드를 기반으로 한 휴대폰용 어플리케이션을 안드로이드 앱(Android App)이라 한다. 이러한 안드로이드 앱(Android App)은 자바 언어로 만들어졌으며, 자바의 코드 특성상 안드로이드 앱은 원본에 가까운 코드 복원이 가능하여 손쉽게 분석이 가능하다. 따라서 앱 제작자들은 안드로이드 앱에 다양한 분석방지기술(예컨대, 코드 난독화, 안티디스어셈블링 등)을 적용하고 있다.

[0003] 그러나 이러한 앱 보호기술 또는 분석방지기술이 악성앱(Malware)에도 적용되기 시작하였고, 이로 인해 악성앱에 대한 신속한 대처가 어려워졌다. 즉, 일반적으로 앱을 분석하는 방법에는 앱의 코드를 보고 앱의 실행 흐름을 분석하는 정적 분석과, 앱의 코드를 직접 실행하며 분석하는 동적 분석이 있는데, 악성앱에 난독화 및 암호화된 코드가 사용된 경우 정적 분석이 어려워지고, 악성앱에 안티디스어셈블링이 적용된 경우 동적 분석이 초기 단계부터 불가능해지는 문제가 있었다. 또한, 이로 인해, 악성앱에 대한 대처가 어려워지는 문제가 있었다.

선행기술문헌

특허문헌

[0004] (특허문헌 0001) 대한민국 공개특허 공개번호 10-2012-96983(악성 프로그램 검출 방법 및 이를 구현하는 휴대 단말기, 삼성전자주식회사, 2012.09.03 공개)

발명의 내용

해결하려는 과제

[0005] 따라서, 본 발명은 유니코드 기반의 안티디스어셈블링을 우회함으로써 안드로이드 악성앱의 동적 분석이 원활하게 이루어지도록 하는 안드로이드 앱 분석 장치 및 그 방법을 제공하고자 한다.

[0006] 또한, 본 발명은 안드로이드 악성앱에 대하여 신속한 대처를 할 수 있도록 하는 안드로이드 앱 분석 장치 및 그

방법을 제공하고자 한다.

[0007] 또한, 본 발명은 상기 안드로이드 앱 분석 방법을 실행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록 매체를 제공하고자 한다.

과제의 해결 수단

[0008] 상기 목적을 달성하기 위해, 본 발명에서 제공하는 안드로이드 앱 분석 장치는 분석할 안드로이드 앱 파일의 압축을 해제하여 패키징된 다수의 파일들을 분리하거나, 분리된 다수의 파일들을 다시 패키징하여 압축하는 압축 처리부; 상기 분리된 파일들 중 소스 코드를 포함하는 제1 파일을 분석하고, 상기 제1 파일에 포함된 유니코드를 모두 아스키코드로 교체하여 제2 파일을 생성하고, 상기 앱 파일 내 소스코드 파일을 상기 제2 파일로 교체한 후 상기 제2 파일을 판독 가능한 형태의 제1 코드로 변환하거나, 상기 제1 코드를 기계어 형태의 제2 코드로 변환하는 코드 처리부; 상기 압축 처리부에서 분리된 다수의 파일들 중 소스코드가 포함된 제1 파일을 추출한 후, 이를 상기 코드 처리부로 전달하여 상기 제1 코드로 변환되도록하고, 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정한 후, 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환한 후 안드로이드 앱 파일로 다시 패키징하도록 상기 코드 처리부 및 상기 압축 처리부를 제어하는 제어부를 포함한다.

[0009] 바람직하게는, 상기 제어부는 상기 압축 처리부에서 다시 패키징된 안드로이드 앱 파일에 개발자 서명을 하여 디버깅 앱을 생성한 후 상기 디버깅용 앱을 실행시킬 수 있다.

[0010] 바람직하게는, 상기 안드로이드 앱 분석 장치는 패키징된 다수의 파일들이 압축된 형태로 구현된 안드로이드 앱 파일들을 저장하는 저장부; 상기 제1 코드에 대한 정적 분석이 가능하도록 상기 제1 코드를 표시하는 디스플레이부; 및 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력을 받아들이는 인터페이스부를 더 포함할 수 있다.

[0011] 바람직하게는, 상기 코드 처리부는 상기 제1 파일에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별하고, 상기 유니코드를 포함하는 문자열을 한 바이트씩 차례로 검사하여 유니코드가 검출되면 그 유니코드를 임의의 아스키코드로 교체할 수 있다.

[0012] 바람직하게는, 상기 코드 처리부는 상기 교체 내역을 별도로 저장하고, 상기 교체 내역을 참조하여 동일한 유니코드 문자열에 대해서는 동일한 아스키코드 문자열로 교체할 수 있다.

[0013] 바람직하게는, 상기 코드 처리부는 상기 제1 파일에 포함된 모든 문자열들 각각에 대하여 문자열의 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 다른 경우 해당 문자열을 유니코드가 포함된 문자열로 식별할 수 있다.

[0014] 바람직하게는, 상기 코드 처리부는 상기 유니코드를 임의의 아스키코드로 교체한 후 해당 문자열의 헤더 부분에 저장된 문자열 길이를 실제 바이트 수로 변경할 수 있다.

[0015] 바람직하게는, 상기 코드 처리부는 서로 다른 유니코드 각각에 대하여 서로 다른 아스키코드로 교체할 수 있다.

[0016] 바람직하게는, 상기 코드 처리부는 상기 유니코드가 한글 유니코드 범위에 속하는 경우 해당 바이트를 아스키코드로 변경하지 않을 수 있다.

[0017] 한편, 상기 목적을 달성하기 위해, 본 발명에서 제공하는 안드로이드 앱 분석 방법은 패키징된 다수의 파일들이 압축된 형태로 구현된 안드로이드 앱 파일로부터 소스코드가 포함된 제1 파일을 추출하는 단계; 상기 제1 파일에 포함된 유니코드를 모두 아스키코드로 교체하여 제2 파일을 생성하고 앱 파일 내 소스코드 파일을 제2 파일로 교체하는 소스파일 교체단계; 상기 제2 파일을 판독 가능한 형태의 제1 코드로 변환하는 디스어셈블링 단계; 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정하는 단계; 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환한 후 안드로이드 앱 파일로 리패키징하는 단계; 및 개발자 서명하여 디버깅용 앱을 생성한 후 상기 디버깅용 앱을 실행하는 동적 분석 단계를 포함한다.

[0018] 바람직하게는, 상기 소스코드 파일 교체 단계는 상기 제1 파일에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별하는 단계; 상기 문자열 식별 단계에서 유니코드를 포함하는 것으로 식별된 문자열을 한 바이트씩 차례로 검사하여 유니코드를 식별하는 단계; 및 상기 유니코드를 임의의 아스키코드로 교체하는 단계를 포함할 수 있다.

[0019] 바람직하게는, 상기 소스코드 파일 교체 단계는 상기 교체 내역을 별도로 저장하는 단계를 더 포함하고, 상기 아스키코드 교체 단계는 상기 교체 내역을 참조하여, 동일한 유니코드 문자열에 대해서는 동일한 아스키코드 문

자일로 교체할 수 있다.

- [0020] 바람직하게는, 상기 문자열 식별 단계는 상기 제1 파일에 포함된 문자열 인덱스를 파악하여 해당 앱에서 사용되는 모든 문자열을 식별하는 단계; 상기 식별된 모든 문자열들 각각에 대하여 문자열의 헤더부분에 저장된 문자열 길이와 실제 바이트 수를 비교하는 단계; 및 상기 비교 결과에 기초하여 상기 모든 문자열들 각각에 대한 유니코드 포함 여부를 결정하는 단계를 포함할 수 있다.
- [0021] 바람직하게는, 상기 유니코드 포함 여부 결정 단계는 상기 문자열의 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 서로 다른 경우 해당 문자열에 유니코드가 포함된 것으로 결정할 수 있다.
- [0022] 바람직하게는, 상기 유니코드 식별 단계는 상기 검사 결과 해당 바이트의 값이 유니코드 범위에 속하는 경우 해당 바이트를 유니코드로 식별할 수 있다.
- [0023] 바람직하게는, 상기 아스키코드 교체 단계는 서로 다른 유니코드 각각에 대하여 서로 다른 아스키코드로 교체할 수 있다.
- [0024] 바람직하게는, 상기 아스키코드 교체 단계는 상기 유니코드를 아스키코드로 교체한 후 해당 문자열의 헤더 부분에 저장된 문자열의 길이를 실제 바이트 수로 변경하는 단계를 더 포함할 수 있다.
- [0025] 바람직하게는, 상기 아스키코드 교체 단계는 상기 유니코드가 한글 유니코드 범위에 속하는 경우 해당 바이트를 아스키코드로 교체하지 않고 다음 단계로 진행할 수 있다.
- [0026] 또한, 상기 목적을 달성하기 위해, 본 발명에서는 상기 기재된 방법 중 하나를 수행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록 매체를 제공한다.

발명의 효과

- [0027] 본 발명의 안드로이드 앱 분석 장치 및 방법은 안드로이드 앱의 소스 코드에 악의적으로 포함된 모든 유니코드들을 아스키코드로 교체함으로써, 안티디스어셈블링을 우회할 수 있다. 즉, 본 발명은 안드로이드 앱의 동적 분석을 위한 디버깅앱 생성시 유니코드 때문에 발생하는 오류를 제거함으로써, 안드로이드 앱(특히 악성앱)의 동적 분석이 원활하게 이루어지도록 한다. 따라서, 본 발명은 안드로이드 악성앱에 대하여 신속하게 대처할 수 있는 장점이 있다.

도면의 간단한 설명

- [0028] 도 1은 본 발명의 일 실시 예에 따른 안드로이드 앱 분석 장치에 대한 개략적인 블록도이다.
- 도 2는 본 발명의 일 실시 예에 따른 안드로이드 앱 분석 방법에 대한 개략적인 처리 흐름도이다.
- 도 3은 도 2에 예시된 디스어셈블링 과정에 대한 상세한 처리과정을 설명하기 위한 순서도이다.
- 도 4는 유니코드 기반 안티디스어셈블링으로 인해 디스어셈블링 에러가 발생한 예를 도시한 도면이다.
- 도 5는 도 4에 예시된 오류를 발생시키도록 유니코드를 포함하여 작성된 실행코드의 예를 도시한 도면이다.
- 도 6은 안드로이드 앱에 포함된 실행코드 및 문자열의 구조를 설명하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

- [0029] 아래에서는 첨부한 도면을 참고로 하여 본 발명의 실시 예에 대하여 설명하되, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 본 발명을 용이하게 실시할 수 있도록 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시 예에 한정되지 않는다. 한편 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다. 또한 상세한 설명을 생략하여도 본 기술 분야의 당업자가 쉽게 이해할 수 있는 부분의 설명은 생략하였다.
- [0030] 명세서 및 청구범위 전체에서, 어떤 부분이 어떤 구성 요소를 포함한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성 요소를 제외하는 것이 아니라 다른 구성 요소를 더 포함할 수 있는 것을 의미한다.
- [0031] 도 1은 본 발명의 일 실시 예에 따른 안드로이드 앱 분석 장치에 대한 개략적인 블록도이다. 도 1을 참조하면, 본 발명의 일 실시 예에 따른 안드로이드 앱분석 장치(100)는 인터페이스부(110), 저장부(120), 제어부(130),

디스플레이부(140), 압축 처리부(150), 코드 처리부(160)를 포함한다.

- [0032] 인터페이스부(110)는 외부 장치 및 사용자와의 인터페이스를 제공한다. 예를 들어, 인터페이스부(110)는 분석하고자 하는 앱 선택 정보 및 선택된 앱의 정적 분석 결과에 따른 외부 입력 정보(예컨대, 코드 수정 정보 등)를 입력받아 제어부(130)로 전달할 수 있다.
- [0033] 저장부(120)는 안드로이드 앱 분석 장치(100)를 동작시키기 위해 필요한 각종 정보 및 데이터들을 저장한다. 예를 들어, 저장부(120)는 패키징된 다수의 파일들이 압축된 형태로 구현된 안드로이드 앱 파일(예컨대, APK 파일 등)들을 저장할 수 있다.
- [0034] 제어부(130)는 기 저장된 동작 프로그램 및 인터페이스부(110)를 통해 입력되는 입력 정보 등에 기초하여 안드로이드 앱 분석 장치(100)의 동작을 제어한다.
- [0035] 디스플레이부(140)는 안드로이드 앱 분석 장치(100)의 동작 정보 등을 표시한다. 예를 들어, 디스플레이부(140)는 안드로이드 앱 분석 과정 중 안드로이드 앱 파일에 포함된 소스 코드가 판독 가능한 형태의 코드로 변환된 경우 이를 표시하여, 사용자가 정적 분석을 실시할 수 있도록 한다.
- [0036] 압축 처리부(150)는 다수의 파일들을 압축 형태로 패키징하거나, 압축 파일을 해제하여 패키징된 파일들을 분리한다. 예를 들어, 압축 처리부(150)는 분석할 안드로이드 앱 파일의 압축을 해제하여 패키징된 다수의 파일들을 분리하거나, 분리된 후 수정된 파일들을 다시 패키징하여 압축할 수 있다.
- [0037] 코드 처리부(160)는 기계어로 표현된 파일을 판독 가능한 형태의 코드로 변환하거나, 판독 가능한 형태의 코드를 기계어로 변환한다. 예를 들어, 코드 처리부(160)는 압축 처리부(150)에서 분리된 다수의 파일들 중 소스코드가 포함된 제1 파일(예컨대, classes.dex)을 추출한 후, 이를 판독 가능한 형태의 제1 코드(예컨대, smali 코드로) 변환할 수 있다. 이 때, 코드 처리부(150)는 상기 제1 파일을 상기 제1 코드로 변환하기 전에, 상기 제1 파일을 분석하여 상기 제1 파일에 유니코드(unicode)가 포함된 경우, 그 유니코드를 모두 아스키코드(ASCII code)로 교체하여 제2 파일을 생성하는 과정을 더 포함하여야 한다. 이는 상기 제1 파일에 유니코드가 포함된 경우, 해당 앱 파일을 동적 분석하기 위해 디버깅 가능한 상태로 변환하는 과정에서 오류가 발생하기 때문이다. 도 4에 이러한 예를 도시하였다. 도 4는 유니코드 기반 안티디스어셈블링으로 인해 디스어셈블링 에러가 발생한 예를 도시한 도면으로서, 도 4를 참조하면, apktool을 이용하여 sample.apk를 디스어셈블하는 경우 A, B와 같은 오류 메시지가 나타남과 동시에 디스어셈블링 에러가 발생함을 알 수 있다. 한편, 도 5는 도 4에 예시된 오류를 발생시키도록 유니코드를 포함하여 작성된 실행코드의 예를 나타내고 있다.
- [0038] 이와 같은 오류를 방지하기 위해, 코드 처리부(160)는 먼저, 상기 제1 파일에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별한다. 즉, 코드 처리부(160)는 상기 제1 파일에 포함된 모든 문자열들 각각에 대하여 문자열의 헤더 부분에 저장된 문자열 길이와 실제 바이트 수를 비교하고, 그 수가 서로 다른 경우 해당 문자열을 유니코드가 포함된 문자열로 식별한다. 이는 유니코드의 경우 한 글자당 2~3 바이트를 차지하므로 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 달라지기 때문이다.
- [0039] 도 6은 안드로이드 앱에 포함된 실행코드 및 문자열의 구조를 설명하기 위한 도면으로서, 도 6의 (a)를 참조하면, 안드로이드 앱(.apk)(200)에 포함된 classes.dex(210)는 실행 코드를 포함하는 파일로서, 헤더영역(Header(211)), 각종 인덱스 영역(String_ids(212), Type_ids(213), Proto_ids(214), Fields_ids(215), Methods_ids(216)), 클래스 정의 영역(Classes_defs(217)), 데이터 영역(Data(218))을 포함하고, 문자열 구조는 도 6의 (b)에 예시된 바와 같다. 도 6의 (b)를 참조하면, 샘플 앱의 classes.dex(210)에는 45479개의 문자열들이 포함되며, String_ids(212)에는 45479개의 어드레스 정보가 포함된다. 한편, 상기 어드레스 정보에 의하면, 상기 45479개의 문자열들은 데이터 영역(Data(218))의 주소 0x001E0EEA(281)에는 길이가 8인 문자열 (76 6S 72 73 69 6F 6E 3A)이 저장되고, 주소 0x001E0EFE(282)에는 길이가 5인 문자열 (4C 6F 2F E3 83 AE 3B)이 저장됨을 알 수 있다. 즉, 주소 0x001E0EEA(281)에는 헤더 부분에 저장된 문자열 길이(8)와 동일한 바이트수의 문자열 (76 6S 72 73 69 6F 6E 3A)이 저장되었지만, 주소 0x001E0EFE(282)에는 헤더 부분에 저장된 문자열 길이(5)와 다른 바이트 수의 문자열(4C 6F 2F E3 83 AE 3B)이 저장되었음을 알 수 있다. 따라서 주소 0x001E0EEA(281)에 저장된 문자열(76 6S 72 73 69 6F 6E 3A)은 유니코드를 포함하지 않지만, 0x001E0EFE(282)에 저장된 문자열(4C 6F 2F E3 83 AE 3B)은 유니코드를 포함한다는 것을 알 수 있다.
- [0040] 이러한 특성에 의해, 코드 처리부(160)는 각 문자열에 유니코드가 포함되었는지 여부를 결정하고, 상기과 같이 유니코드가 포함된 문자열이 식별되면, 상기 유니코드를 포함하는 문자열을 한 바이트씩 차례로 검사하여 유니코드 여부를 결정한다. 즉, 코드 처리부(160)는 해당 바이트가 유니코드 범위에 속하는 경우 이를 유니코드로

결정한다. 도 6의 (b)를 참조하면, 코드 처리부(160)는 0x001E0EFE(282)에 저장된 문자열(4C 6F 2F E3 83 AE 3B)을 구성하는 문자들을 한 바이트씩 차례로 검사하여(4C -> 6F -> 2F -> E3 -> 83 -> AE -> 3B), 해당 바이트의 값이 유니코드 범위에 속하는 지 여부를 결정한다.

[0041] 그 결과 유니코드가 검출되면 코드 처리부(160)는 그 유니코드를 모두 임의의 아스키코드로 교체한다. 이 때, 코드 처리부(160)는 상기 교체 내역을 별도로 저장하고, 상기 교체 내역을 참조하여 동일한 유니코드 문자열에 대해서는 동일한 아스키코드 문자열로 교체하는 것이 바람직하다. 이는 각 유니코드 문자열을 모두 유니크한 아스키 코드 문자열로 교체할 경우 클래스명이 달라지므로 결국 메소드 참조시 오류를 유발하게 되기 때문에 이를 예방하기 위함이다. 예를 들어, 동일 클래스명이 서로 다른 클래스로 식별되지 않도록 하기 위함이다.

[0042] 또한, 코드 처리부(160)는 검출된 모든 유니코드를 서로 다른 아스키 코드로 교체한다. 이는 서로 중복되지 않도록 하기 위함이다. 예를 들어, 유니코드의 검출 순서에 따라, 알파벳 A부터 Z까지 순차적으로 선택하여 교체하거나, 그 역순으로 선택하여 교체할 수 있다. 이와 같이, 유니코드를 임의의 아스키코드로 교체한 후, 코드 처리부(160)는 해당 문자열의 헤더 부분에 저장된 문자열 길이를 실제 바이트 수로 변경한다. 도 6의 (b)를 예에서, 0x001E0EFE(282)에 저장된 문자열(4C 6F 2F E3 83 AE 3B)에 포함된 유니코드를 모두 아스키 코드로 교체한 후, 코드 처리부(160)는 0x001E0EFE(282)의 헤더 부분에 저장된 5를 7로 변경한다.

[0043] 이는 상기 두 값이 다름으로 인하여 디스어셈블링시 발생할 수 있는 오류를 예방하기 위함이다.

[0044] 이 때, 코드 처리부(160)는 상기 유니코드가 한글 유니코드 범위(0xeab080~0xed9fbf)에 속하는 경우 해당 바이트를 아스키코드로 변경하지 않는다. 이는 유니코드 중 한글은 클래스나 메소드 등의 이름으로 사용되지 않고, 앱 사용 안내 및 오류를 나타내는 메시지로 사용되기 때문이다. 즉, 모든 유니코드를 아스키 코드로 변경하는 경우 메시지 또한 의미없는 아스키 코드로 변경되어 오히려 분석에 필요한 주요 정보를 손실하게 되는 결과를 초래하기 때문이다.

[0045] 한편, 코드 처리부(150)는 상기 제1 코드를 기계어 형태의 제2 코드로 변환할 수도 있다. 예를 들어, 코드 처리부(150)는 상기 제1 코드에 대한 정적 분석 결과로 입력된 외부 입력에 기초하여 상기 제1 코드가 수정된 경우, 이를 다시 기계어 형태의 제2 코드로 변환할 수 있다.

[0046] 이를 위해, 제어부(130)는 압축 처리부(150)에서 분리된 다수의 파일들 중 소스 코드가 포함된 제1 파일을 추출한 후, 이를 코드 처리부(160)로 전달하여 제1 코드로 변환되도록 제어한다. 또한, 제어부(130)는 상기 제1 코드에 대한 정적 분석 결과로 인터페이스(110)를 통해 입력 신호가 전달되면, 그 신호에 기초하여 상기 제1 코드를 수정한다. 그리고, 제어부(130)는 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환하도록 코드 처리부(160)를 제어하고, 상기 제2 코드를 안드로이드 앱 파일로 다시 패키징하도록 압축 처리부(150)를 제어한다.

[0047] 또한, 제어부(130)는 압축 처리부(150)에서 다시 패키징된 안드로이드 앱 파일에 개발자 서명(임시 서명)을 하여 디버깅용 앱을 생성한 후, 동적 분석을 위해 상기 디버깅용 앱을 실행시킨다.

[0048] 도 2는 본 발명의 일 실시 예에 따른 안드로이드 앱 분석 방법에 대한 개략적인 처리 흐름도이다. 도 1 및 도 2를 참조하면, 본 발명의 일 실시 예에 따른 안드로이드 앱 분석 방법은 다음과 같다.

[0049] 먼저, 단계 S100에서는, 분석대상 앱 파일로부터 소스코드가 포함된 제1 파일을 추출한다. 이를 위해, 제어부(130)가 저장부(120)에서 분석 대상 앱 파일(예컨대, sample.APK)을 선택하여 압축 처리부(150)로 전달하면, 압축 처리부(150)는 상기 분석 대상 앱 파일의 압축을 해제하여 다수의 파일들을 분리하고, 제어부(130)가 상기 분리된 파일들 중 소스코드가 포함된 제1 파일(예컨대, classes.dex)을 추출하는 일련의 과정을 수행하는 것이 바람직하다.

[0050] 단계 S200에서는, 코드 처리부(160)가 상기 제1 파일에 포함된 유니코드를 모두 아스키코드로 교체하여 제2 파일을 생성하고 상기 앱 파일 내 소스코드 파일을 상기 제2 파일로 교체한다. 이 때, 단계 S200에 대한 보다 상세한 처리 과정의 예가 도 3에 예시되어 있으므로, 단계 S200에 대한 보다 상세한 설명은 도 3을 참조하여 설명할 것이다.

[0051] 단계 S300에서는, 코드 처리부(160)가 상기 제2 파일을 판독 가능한 형태의 제1 코드(예컨대, smali 코드)로 변환하는 디스어셈블링을 수행한다.

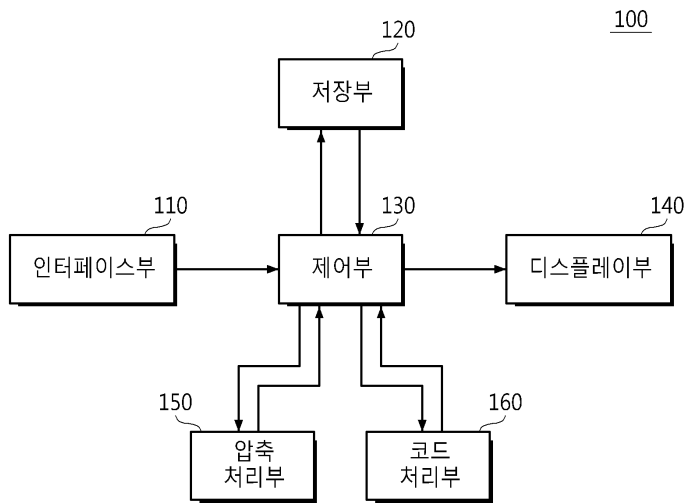
[0052] 단계 S400에서는, 제어부(130)가 상기 제1 코드에 대한 정적 분석 결과로 입력되는 외부 입력에 기초하여 상기 제1 코드를 수정한다. 예를 들어, 악성 코드의 동작 여부를 모니터링하기 위한 소스 코드의 삽입을 요청하는 등의 외부 입력이 인터페이스부(110)를 통해 전달되면, 제어부(130)는 그 외부 입력에 기초하여 상기 제1 코드를

수정한다.

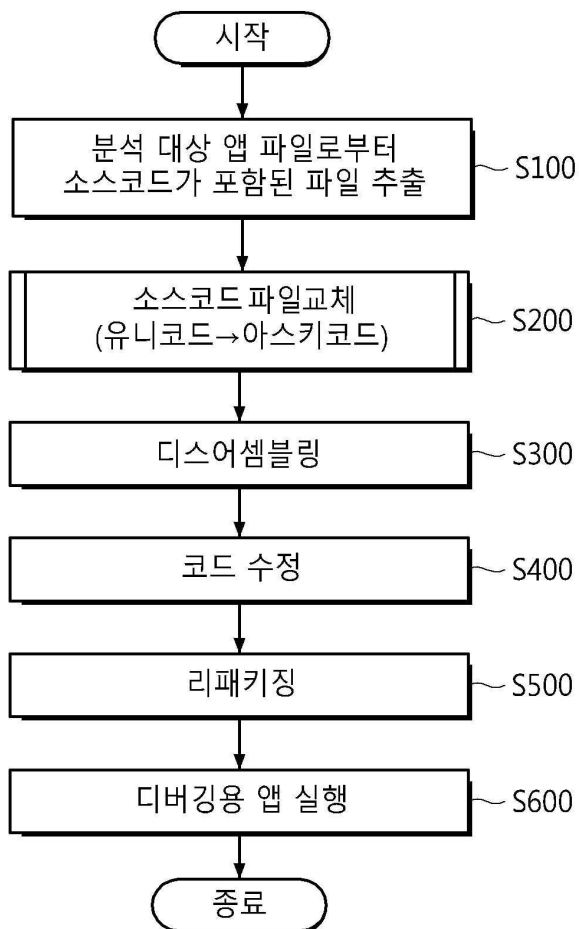
- [0053] 단계 S500에서는, 코드 처리부(160)가 상기 수정된 제1 코드를 기계어 형태의 제2 코드로 변환하고, 압축 처리부(150)가 상기 변환된 제2 코드를 안드로이드 앱 파일로 리패키징한다.
- [0054] 단계 S600에서는, 제어부(130)가 상기 리패키징된 안드로이드 앱 파일에 개발자 서명(임시 서명)을 하여 디버깅용 앱을 생성한 후, 동적 분석을 위해 상기 디버깅용 앱을 실행한다.
- [0055] 도 3은 도 2에 예시된 소스파일 교체 과정(S200)에 대한 상세한 처리과정을 설명하기 위한 순서도이다. 도 1 및 도 3을 참조하면, 먼저, 코드 처리부(160)가 상기 제1 파일(예컨대, classes.dex 파일, 도 6 참조)에 포함된 모든 문자열을 분석하여 유니코드를 포함하는 문자열을 식별한다. 이를 위해, 단계 S205에서는, 코드 처리부(160)가 상기 제1 파일(예컨대, classes.dex 파일, 도 6 참조)에 포함된 문자열 인덱스(String_ids)(도 6의 212)를 파싱한다. 그리고, 단계 S210에서는, 코드 처리부(160)가 첫 번째 문자열을 선택하고, 단계 S215에서는, 상기 선택된 문자열의 헤더부분에 저장된 문자열 길이와 실제 바이트 수를 비교하고, 그 비교 결과에 기초하여 상기 모든 문자열들 각각에 대한 유니코드 포함 여부를 결정한다. 이는 유니코드의 경우 한 글자당 2~3 바이트를 차지하므로 헤더 부분에 저장된 문자열 길이와 실제 바이트 수가 달라지기 때문이다.
- [0056] 상기 비교 결과 상기 선택된 문자열의 헤더부분에 저장된 문자열 길이와 실제 바이트 수가 서로 다른 경우, 단계 S220에서는, 유니코드를 식별한다. 즉, 코드 처리부(160)는 상기 선택된 문자열의 데이터들을 한 바이트씩 차례로 검사하여 유니코드를 식별한다. 즉, 코드 처리부(160)는 해당 바이트가 유니코드 범위에 속하는 경우 이를 유니코드로 결정한다.
- [0057] 단계 S225 내지 S240에서는, 상기 식별된 유니코드를 아스키코드로 교체하고, 그 교체 내역을 저장한다. 즉, 단계 S225에서, 이전 교체 내역이 존재하는 지 여부를 결정하고, 단계 S230에서는, 이전 교체 내역이 존재하는 경우 그 내용을 참조하여 해당 유니코드를 아스키코드로 교체한다. 즉, 동일한 유니코드 문자열에 대해서는 동일한 아스키코드 문자열로 교체한다. 단계 S235에서는, 이전 교체 내역이 존재하지 않는 경우 해당 유니코드를 임의의 아스키코드로 교체한다. 한편, 단계 S230 및 단계 S235에서는, 서로 다른 유니코드 각각에 대하여 서로 다른 아스키코드로 교체하는 것이 바람직하다. 이는 이는 서로 중복되지 않도록 하기 위함이다. 예를 들어, 유니코드의 검출 순서에 따라, 알파벳 A부터 Z까지 순차적으로 선택하여 교체하거나, 그 역순으로 선택하여 교체할 수 있다. 이와 같이, 유니코드를 임의의 아스키코드로 교체한 후, 코드 처리부(160)는 해당 문자열의 헤더 부분에 저장된 문자열 길이를 실제 바이트 수로 변경한다. 도 6의 (b)를 예에서, 0x001E0EFE(282)에 저장된 문자열(4C 6F 2F E3 83 AE 3B)에 포함된 유니코드를 모두 아스키 코드로 교체한 후, 코드 처리부(160)는 0x001E0EFE(282)의 헤더 부분에 저장된 5를 7로 변경한다. 이는 상기 두 값이 다름으로 인하여 디스어셈블링시 발생할 수 있는 오류를 예방하기 위함이다.
- [0058] 만약, 단계 S220에서 식별된 상기 유니코드가 한글 유니코드 범위(0xeab080~0xed9fbf)에 속하는 경우, 코드 처리부(160)는 해당 바이트를 아스키코드로 변경하지 않는다. 이는 유니코드 중 한글은 클래스나 메소드 등의 이름으로 사용되지 않고, 앱 사용 안내 및 오류를 나타내는 메시지로 사용되기 때문이다. 즉, 모든 유니코드를 아스키 코드로 변경하는 경우 메시지 또한 의미없는 아스키 코드로 변경되어 오히려 분석에 필요한 주요 정보를 손실하게 되는 결과를 초래하기 때문이다.
- [0059] 그리고 단계 S240에서는, 상기 교체 내역을 별도로 저장한다. 이는 각 유니코드 문자열을 모두 유니크한 아스키코드 문자열로 교체할 경우 클래스명이 달라지므로 결국 메소드 참조시 오류를 유발하게 되기 때문에 이를 예방하기 위함이다. 예를 들어, 동일 클래스명이 서로 다른 클래스로 식별되지 않도록 하기 위함이다.
- [0060] 단계 S245에서는, 다음 문자열이 존재하는 지를 결정하고, 단계 S250에서는, 다음 문자열이 존재하는 경우 다음 문자열을 선택한 후, 단계 S215 이후의 과정을 수행한다. 다음 문자열이 존재하지 않는 경우, 디스어셈블링 과정(S200)을 종료한다.
- [0061] 상술한 예시적인 시스템에서, 방법들은 일련의 단계 또는 블록으로써 순서도를 기초로 설명되고 있지만, 본 발명은 단계들의 순서에 한정되는 것은 아니며, 어떤 단계는 상술한 바와 다른 단계와 다른 순서로 또는 동시에 발생할 수 있다.
- [0062] 또한, 당업자라면 순서도에 나타난 단계들이 배타적이지 않고, 다른 단계가 포함되거나 순서도의 하나 또는 그 이상의 단계가 본 발명의 범위에 영향을 미치지 않고 삭제될 수 있음을 이해할 수 있을 것이다.

도면

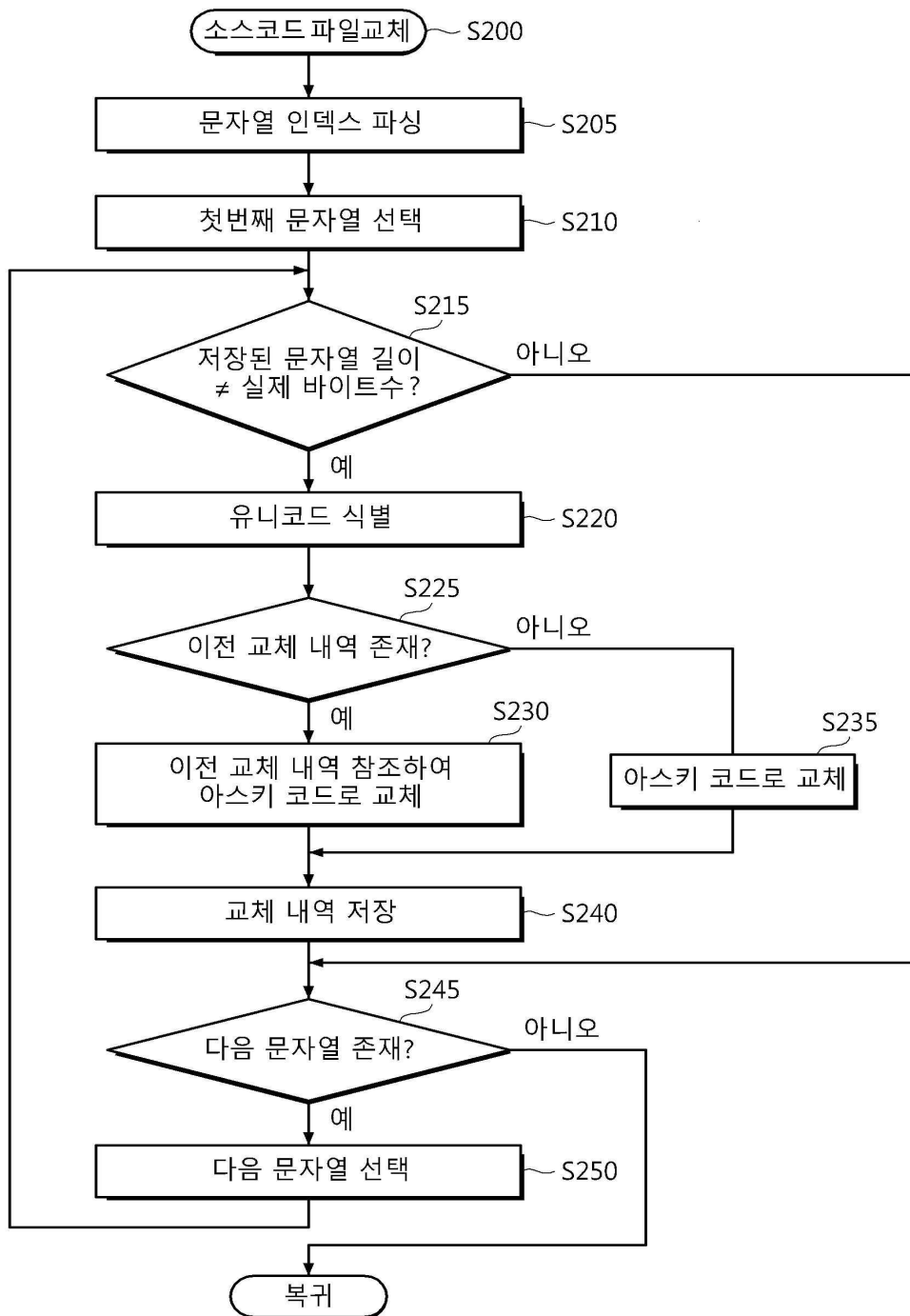
도면1



도면2



도면3

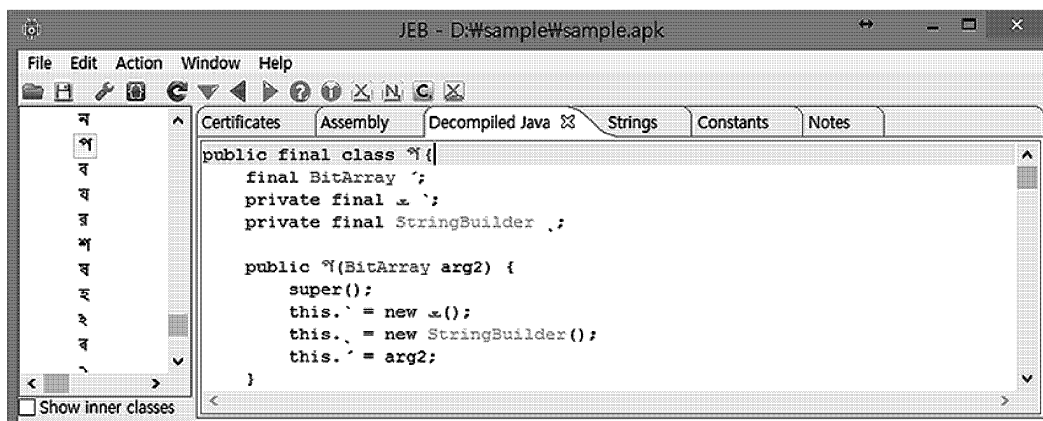


도면4

사용도구 : apktool (apktool d -d sample.apk)

```
I: Baksmaling classes.dex...
Exception in thread "main" brut.androlib.AndrolibException: brut.androlib.AndrolibException: java.nio.charset.MalformedInputException: Input length = 1
    at brut.androlib.Androlib.decodeSourcesSmali<Androlib.java:96>
    at brut.androlib.ApkDecoder.decode<ApkDecoder.java:96>
    at brut.apktool.Main.cmdDecode<Main.java:165>
    at brut.apktool.Main.main<Main.java:81>
Caused by: brut.androlib.AndrolibException: java.nio.charset.MalformedInputException: Input length = 1
    at brut.androlib.src.SmaliDecoder.decode<SmaliDecoder.java:111>
    at brut.androlib.src.SmaliDecoder.decode<SmaliDecoder.java:46>
    at brut.androlib.Androlib.decodeSourcesSmali<Androlib.java:94>
    ... 3 more
```

도면5



도면6

