강건한 안드로이드 앱을 위한 실행시간 인텐트 스펙 검사 방법

고명필

연세대학교 전산학과
원주 흥업면 연세대길 1
myungpil.ko@yonsei.ac.kr

최광훈

연세대학교 컴퓨터정보통신공학부
원주 흥업면 연세대길 1
kwanghoon.choi@yonsei.ac.kr

창병모

숙명여자대학교 컴퓨터과학과
서울 용산구 청파로 47 길 100
chang@sookmyung.ac.kr

요약: 안드로이드 프로그램은 인텐트를 이용해 액티비티, 서비스, 브로드캐스트 리시버와 같은 컴포넌트를 호출한다. 최근 연구 결과에 따르면 안드로이드 플랫폼은 인텐트의 정보가 부족해 발생하는 문제로 취약점이 생겨 강건성이 떨어진다. 이 논문은 컴포넌트가 기대하는 인텐트 스펙을 직접 기술할 수 있는 문법을 제안하고, 전달된 인텐트로 오류가 발생하기 전에 미리 검사하여 핸들링하는 방법을 제안한다. 실험을 통해 안드로이드 플랫폼에서 인텐트 스펙을 사용하여 잘못된 인텐트 전달로 발생하는 문제를 줄이고, 크지 않은 오버헤드로 사용성이 보장되어 제안한 방법이 유용함을 보인다.

핵심어: 안드로이드, 인텐트 스펙, 문법, 인텐트 기술, 파싱, 오류 핸들링

1. 서론

안드로이드[1]는 모바일 디바이스를 위한 구글의 오픈소스 플랫폼이다. 안드로이드 프로그램은 안드로이드 플랫폼 API 를 사용하는 자바 프로그램으로, 액티비티, 서비스, 브로드캐스트 리시버, 컨텐트 프로바이더 컴포넌트로 구성된다.

인텐트는 안드로이드 플랫폼의 IPC 메시지다. 컴포넌트를 호출할 때, 인텐트를 전달하여 호출된 컴포넌트에 필요한 추가 정보를 제공한다.

최근 연구 결과에 따르면, 잘못된 인텐트(예를 들어, 특정 필드의 부재)를 컴포넌트에게 전달하여 안드로이드 앱이 비정상 종료되는 취약점이 있다고 한다[2]. 컴포넌트 테스트 도구 fuzzer[3, 4]를 사용하여 무작위로 생성한 인텐트를 컴포넌트에게 전달해 총 332 개의 액티비티 중 28(8.7%)개의 액티비티에서 오류가 발생함을 보였다.

이러한 취약점에 대응하기 위해 컴포넌트에 전달된 인텐트가 잘못되었는지를 컴포넌트에서 직접 검사하는 방법을 제공한다.

본 논문에서 기여한 점은 첫째, 컴포넌트가 기대하는 인텐트 스펙을 기술할 수 있는 문법을 제시한다. 둘째, 제안된 문법으로 기술된 인텐트 정보와 전달된 인텐트를 실행시간에 검사한다. 셋째, 잘못된 인텐트를 발견하면 대처하는 방법을 개발자가 지정할 수 있다. 넷째, 실행시간 인텐트 오류 검사로 인해 비정상 종료되는 취약점을 개선하여 앱의 강건성을 높이고, 실행시간과 실행파일 크기에 대한 오버헤드가 낮음을 실험을 통해서 보인다.

본 논문의 구성은 다음과 같다. 2 장에서 관련연구에 대해서 설명하고, 3 장에서는 인텐트 스펙을 제안한다. 4 장에서 실험결과를 설명한 다음, 5 장에서 결론을 맺고 향후 연구를 논의한다.

2. 관련연구

2.1 인텐트 구조 및 예제

인텐트는 다음과 같은 필드로 구성된다.

- action : 기능을 구분
- data : action 에 필요한 데이터의 위치
- category : action 에 세부 정보
- type : data 의 타입
- component : 대상 컴포넌트
- extras : 추가 제공하는 정보의 묶음
- flags : 플랫폼의 컴포넌트 스택을 조정

예를 들어, 그림 1 은 전달받은 인텐트 정보를 이용해서 제목과 내용을 화면에 보여주는 간단한 액티비티이다. 이 액티비티에 인텐트 정보는 action 이 *INSERT* 또는 *EDIT*, data 가 *Uri* 값, Extras 로 *title, context* 키의 값으로 *String* 타입을 기대한다. 이 예제는 인텐트 정보의 부재를 이용한 악의적인 공격받을 수 있다. 첫째로 action 이 *EDIT* 일 때, 인텐트의 정보 중 data 를 *null* 로 전달하면 *NullPointerException* 이 발생한다. 둘째로 action 이 *EDIT* 일 때, 인텐트의 정보 중 extras 인 *title* 과 *context* 를 *null* 값으로 전달하면 *NullPointerException* 이 발생한다. 마지막으로, action 이 *EDIT* 나 *INSERT* 가 아닌 경우 *uri, title, context* 가 모두 *null* 이므로 *NullPointerException* 발생한다. 안드로이드 플랫폼은 컴포넌트를 호출하는 인텐트로 예제와 같이 취약점이 발생한다.

```
public class Note extends Activity {
  String title, context;
  Uri uri;
  void onCreate(Bundle savedInstanceState){
    Intent intent = getIntent();
    String action = intent.getAction();
    if (Intent.ACTION_EDIT.equals(action)) {
      uri = intent.getData();
      title = intent.getStringExtra("title");
      context = intent.getStringExtra("context");
    } else if (Intent.ACTION_INSERT.equals(action) {
      uri = null; title = "new"; context = "memo"; }
    // Display uri, title, context
  }
  // Skip
}
```

그림 1 인텐트를 사용하는 액티비티

2.2 취약한 인텐트에 관한 연구

안드로이드 앱에서 인텐트는 컴파일할 때 검사하지 못해 취약하다. 악의적인 안드로이드 앱은 다른 앱의 컴포넌트에게 잘못된 인텐트를 전달해 공격한다. Intent fuzzer[3]를 확장한 실험[4]은 인텐트로 인해 컴포넌트가 취약하다는 것을 실험으로 보인다. 이 실험은 랜덤과 일부 정보만 포함한 인텐트를 컴포넌트에게 전달하여 예외가 처리되는지 확인한다. 실험의 결과로 332 개의 액티비티 중 29(8.7%)가 액티비티가 *NullPointerException*, *ClassNotFoundException*, *IllegalArgumentException* 의 오류가 발생해 비정상적으로 종료되거나 OS 가 재부팅까지 되기도한다.

정적 분석 툴 ComDroid[5]은 안드로이드 앱의 바이너리 코드를 실행하지 않고 분석하여 잠재적인 취약점을 발견한다.

정적 분석과 랜덤 인텐트 생성의 아이디어를 조합하여 안드로이드 프로그램을 테스트하는 방법도 제안되었다[4]. 정적 분석을 통해 기대하는 인텐트의 구조를 파악하고, 이 구조에 따라 인텐트를 만들어 대상 컴포넌트를 호출한다. 분석 툴은 개발자가 작성한 코드에서 커버되는 범위와 커버되지 않아 발생하는 문제를 모니터링하는 기능을 제공한다.

본 논문에서는 제안한, 컴포넌트에서 기대하는 인텐트 스펙을 개발자가 직접 기술하여 실행시간에 컴포넌트로 전달된 인텐트가 잘못된 것인지 검사하는 방법은 아직 다른 연구에서 시도된 적이 없다.

3. 인텐트 스펙

인텐트 스펙은 컴포넌트가 기대하는 기술하고 전달된 인텐트와 검사하며 잘못된 인텐트를 직접 핸들링하는 방법이다.

그림 2 는 그림 1 의 액티비티 컴포넌트에 제안한 인텐트 스펙을 사용하여 구현한 예이다. 인텐트 스펙은 *assertOnIntent(검사할 인텐트, 인텐트 스펙, 핸들러)*로 기술한다. Java 언어[2]의 assert 문을 확장했다.

```
public class Note extends Activity {
  ...
  void onCreate(Bundle savedInstanceState){
   ---------------------- 추가된 영역 시작
    new AssertOnIntent().assertOnIntent(
    //검사할 인텐트
    getIntent(),
    //인텐트 스펙
    "{ act = android.intent.action.EDIT        //action
       dat = non-null                          //data
       cmp = kr.ac.yonsei.mobilesw/.Note       //component
       [ title = String, context = String ] }  //extras
    ||
       { act = android.intent.action.INSERT    //action
       cmp = kr.ac.yonsei.mobilesw/.Note }",   //component
     //핸들러
      new MalformedIntentHandler(){
       public void handle
        (Intent intent, MalformedIntentException m)
            {  switch(m.getNumber())
                {  case 102: //data miss matching
                   getIntent().setData(//Default Uri);
                   break;
                   default: //finish; break; } } });
   ---------------------- 추가된 영역 끝
  //아래는 그림1 의 onCreate 메서드와 동일한 코드
    Intent intent = getIntent();
    …
  }
  // Skip
}
```

그림 2 제안한 방법을 추가한 액티비티

그림 2 에서 사용한 인텐트 스펙은 ||로 묶어진 두 가지 유형의 인텐트를 나타낸다. 이 인텐트의 action 은 *EDIT* 이거나 *INSERT* 이다. action 이 *EDIT* 인 경우 data 가 *null* 이 아니고, component 는 *kr.ac.yonsei.mobilesw/.Note* 이며, extras 로 *title* 과 *context* 에 값이 *String* 타입의 문자열이어야 한다. Action 이 *INSERT* 인 경우 component 가 *kr.ac.yonsei.mobilesw/.Note* 이어야 한다.

컴포넌트가 전달받은 인텐트를 인텐트 스펙과 비교하면 세 가지 경우가 발생할 수 있다. 인텐트 스펙에 나열된 인텐트 모양 중 하나와 정확하게 일치하면 받은 인텐트를 정상이라 판단한다. 스펙에 기술된 것보다 더 많은 필드를 포함하고 있는 경우에도(예를 들어, action 이 *EDIT* 인 경우 extras 에 *date* 키와 값을 추가로 포함) 일종의 서브타이핑으로 간주하여 정상이라 판단한다. 오직 더 적은 필드를 가진 경우

에만 비정상으로 판단한다.

컴포넌트가 받은 인텐트가 비정상적이라 판단하는 경우 직접 비정상 인텐트를 개발자가 원하는 방식으로 핸들링할 수 있다. 인텐트 스펙과 매칭되지 않을 때 오류 메시지와 오류 코드를 알리는 *Malformed IntentException* 클래스를 제공한다. 예를 들어, 그림 2 에서 오류 코드 *102* 는 인텐트 스펙에 기술된 data 가 *non-null* 이지만 받은 인텐트의 data 는 *null* 로 매칭에 실패했다는 뜻이다. 이 경우 개발자는 전달받은 인텐트의 data 에 적절한 값을 설정하여 정상적으로 프로그램 실행이 가능하도록 핸들러를 만들 수 있다.

인텐트 스펙에서 사용하는 예외코드는, 인텐트 스펙의 구문이 틀린 경우, action, data, category, type, component, extras, flag 에서 스펙과 다른 경우 및 기타 34 가지로 분류한다.

```
INTENT          ::= { FIELDS } | | INTENT
                  | { FIELDS }
FIELDS          ::= ACTION FIELDS
                  | CATEGORY FIELDS
                  | DATA FIELDS
                  | TYPE FIELDS
                  | COMPONENT FIELDS
                  | EXTRA FIELDS
                  | FLAG FIELDS
                  | ε
ACTION          ::= act=ID
CATEGORY        ::= cat=[ID CATEGORYSUB]
CATEGORYSUB ::= , ID CATEGORYSUB
                  | ε
DATA            ::= dat=non-null
TYPE            ::= typ=non-null
COMPONENT   ::= cmp= ID / (ID | .ID)+
EXTRA           ::= [ID = ID ARR EXTRASUB ]
EXTRASUB        ::= , ID=ID ARR EXTRASUB
                  | ε
FLAG            ::= flg
ID          ::= LETTER (A - Z | a - z | 0 - 9 | _ | .)*
LETTER      ::= (A - Z | a - z)+
ARR         ::= ([])*
```

그림 3 인텐트 스펙을 기술하는 문법

그림 3 은 인텐트 스펙을 서술하기 위한 문법이다. 기술하는 방식은 아래와 같다.

- Intent 는 기대하는 Intent 하나를 {}로 표현한다. 하나 이상의 인텐트를 기대하는 경우, || 를 입력해 추가로 기술한다.
- action 은 값이 문자열이다. 그림 2 에서 *EDIT* 의 문자열은 *android.intent.action.EDIT* 이므로 *act=android.intent.action.EDIT* 로 기술한다.
- category 의 *CATEGORY_BROWSABLE* 실제 값은 *android.intent.category.BROWSABLE* 이므로 *cat=[android.intent.category.BROWSABLE]*로 기술하고, 추가적인 category 가 더 있는 경우 ,로 구분해서 기술한다.
- data 는 일반적으로 *Uri* 값을 입력한다. 만약 data 의 값이 *content://contacts/people/1* 이라면 *dat=non-null* 으로 기술한다.
- type 은 존재하면 *typ=non-null* 으로 기술한다.
- component 는 패키지와 클래스를 /로 구분해서 기술한다. 예를 들어, Note 컴포넌트는 *cmp=kr.ac.yonsei.mobilesw.assertonintent/.Note* 로 기술한다.
- extras 는 키와 값으로 이뤄진다. 예를 들어, *title* 키의 값이 *String* 타입이고 *context* 키의 값이 *String* 타입이라면 extras 는 *[title=String, context=String]*으로 기술한다. 배열은 타입이름과 *[]*로 기술한다. 2 차원 배열에 형태라면 *TypeName[][]*로 기술한다.
- flag 는 값이 있다면 *flg* 로 기술한다.

제안된 인텐트 스펙 구현은 그림 3 의 문법으로 파서를 구현했다. 파서는 인텐트 스펙을 파싱한 결과로 해쉬맵을 낸다. 기대하는 인텐트가 여러 개일 경우, 인텐트 스펙과 전달된 인텐트 매칭을 효율적으로 하기 위해서 해쉬맵을 사용했다. 전달된 인텐트가 인텐트 스펙과 다른 경우 개발자에게 예외 메시지와 코드를 알리는 *MalformedIntentException* 클래스를 만들어 제공하고, 발생한 예외를 개발자가 예외 메시지와 예외 코드를 이용해 직접 제어할 수 있는 방법으로 *MalformedIntentHandler* 를 만들어 제공한다.

제안한 인텐트 스펙을 안드로이드 환경에서 구현한 예제를 다음 사이트에서 확인할 수 있다.

- https://github.com/nzzzn/AssertOnIntent

4. 실험

제안한 방법을 안드로이드 4.4.2 버전의 LG-F220K 폰 환경에서 실험하고 결과를 표로 나타낸다. 실험한 앱 중 BluetoothChat 과 NotesList 는 안드로이드 SDK 에 포함된 샘플이고, Cafe 는 학생이 제작한 앱이다. 그 외 나머지 앱은 안드로이드 프로그래밍 서적에서 참조하였다.

표 1 의 첫 번째 컬럼은 잘못된 인텐트로 인해 안드로이드 앱이 비정상적으로 종료하는 전체 건수 중에 제안한 방법으로 정상적으로 동작한 건수를 나타낸다. NotesList 의 경우 비정상적으로 종료되는 전체 6 건 중 3 건을 방어하여 정상적으로 동작하였다. 방어하지 못한 건에 대해서는 인텐트 스펙을 확장하여 data, type 패턴을 세분화하면 더 많은 오류를 방어할 수 있다.

표 1 의 두 번째 컬럼은 제안한 방법으로 검출할 수 있는 전체 오류의 수이다. 예를 들어, NotesList 의 TitleEditor 액티비티의 경우 action 을 확인하지 않아 잘못된 action 이 전달되더라도 data 의 값만 존재하면 action 은 정확한 것처럼 처리한다. 이때, data 의 값이 유효한지에 따라 정상적으로 동작하거나 앱이 비정상적으로 종료된다. 제안한 방법을 사용하면 전달된 인텐트의 action 이 기대한 action 과 같은지 검사해 잠재적으로 발생할 수 있는 오류를 검출할 수 있다.

표 1 실험 결과

앱	방어한 개수 (총 오류 수)	잠재적으로 검출 가능한 오류 수
AndroidSecurity	0(0)	1
BluetoothChat	1(1)	3
Cafe	0(0)	1
ContactsActivity	0(0)	1
MigrateClinic	0(0)	1
NotesList	3(6)	11
Media_Player	0(0)	4
Earthquake	0(0)	2

실험 결과 잘못된 인텐트로 비정상적으로 종료되는 전체 7 건 중 제안한 방법으로 잘못된 인텐트를 방어해 정상적으로 동작한 건수는 4(57.1%)건이다.

표 2 제안한 방법 적용에 따른 오버헤드

앱	용량(추가된 byte)	시간(추가된 ms)
AndroidSecurity	321,704(+9,099)	421(+10)
BluetoothChat	27,545(+10,811)	450(+10)
Cafe	1,473,874(+10,057)	431(+9)
ContactsActivity	36,116(+10,616)	440(+10)
MigrateClinic	53,999(+10,839)	1402(+60)
NotesList	60,405(+11,038)	421(+10)
Media_Player	154,282(+11,120)	450(+10)
Earthquake	51,117(+10,335)	431(+9)

표 2 는 제안한 방법을 적용함에 따라 발생하는 오버헤드를 보인다. 용량은 원본 실행파일(apk)의 용량을 byte 로 표시하고 제안한 방법을 추가함에 따라 증가하는 용량을 표시했다. 시간은 컴포넌트의 스타트업 시간이다. 인텐트로 컴포넌트를 호출해 화면이 출력된 시간을 ms 로 표시하고 괄호는 제안한 방법으로 문제를 검출하기 위해서 추가되는 시간을 표시했다.

제안한 방법을 적용하여 apk 파일에 추가되는 용량의 평균은 약 10,489byte 이고, 13.75%이다. 원본 apk 파일의 크기가 작은 BluetoothChat 의 경우 28.19%가 증가했고, 원본 apk 파일의 크기가 큰 Cafe 의 경우는 0.68%가 증가했다. 제안한 방법을 사용하여 증가한 용량은 앱을 스마트폰에 설치할 때 큰 차이를 느낄만한 수준의 증가는 아니다.

제안한 방법을 적용하여 추가되는 스타트업 시간은 평균 약 16ms 이고, 2.49%이다. 스마트폰에 설치하여 사용할 때 증가하는 스타트업 시간의 차이는 미미해 사용자가 느낄만한 정도의 시간은 아니다.

5. 결론 및 향후 연구

잘못된 인텐트를 컴포넌트에 전달해 비정상적으로 앱이 종료되는 문제를 인텐트 스펙을 기술하여 해결하는 방법을 제안하였다. 이 방법을 사용하면 안드로이드 앱의 강건성이 높아짐을 실험을 통해 보였다.

향후 연구로는 인텐트 스펙을 기술할 때 세분화된 패턴을 적용하여 더 넓은 범위의 잘못된 인텐트를 검출할 수 있도록 한다. 좀 더 나아가 자바 어노테이션[2]을 이용해 인텐트 스펙을 선언하고 컴파일 타임에 이것을 미리 파싱할 수 있다. 미리 파싱하면 컴포넌트를 실행할 때 발생하는 파싱 시간과 파서의 코드를 제거하여 시간과 크기를 최적화할 수 있다.

참고문헌

[1] Android API, http://developer.android.com, 2014

[2] Amiya K, Maji, Fahad A. Arshad, Saurabh Bagchi, and Jan S. Rellermeyer, “An empirical study of the robustness of Inter-component Communication in Android”, In proceedings of the 2012 42nd Annual IEEE/IFIP International comference on Dependable Systems and Networks (DSN) (DSN’12), IEEE Computer society, Washington, DC, USA, 1-12, 2012

[3] Intent Fuzzer, https://www.isecpartners.com-/tools/mobile-security/intent-fuzzer.aspx, iSEC Partners, 2009.

[4] Raimondas Sasnauskas and John Regehr, Intent Fuzzer: crafting intents of death, In Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and software and System Performance Testing, Debugging, and Analytics (PERTEA)

[5] Erika Chin, Adrienne porter Felt, Kate Greenwood, and David Wagner, “Analyzing inter-application communication in android”, In Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services(MobySys11), pages 239-252, ACM, New York, Ny, Usa, 2011

[6] James G, Bill J, Guy S, Gilad B, Alex B. 2014. The Java™ Language Specification (Java SE 8 Edition). Oracle.