강건한 안드로이드 앱을 위한
인텐트 스펙의 설계와 구현

연세대학교 대학원

전 산 학 과

고 명 필

강건한 안드로이드 앱을 위한 인텐트 스펙의 설계와 구현

지도교수 최광훈

이 논문을 석사 학위논문으로 제출함

2015년 12월 10일

연세대학교 대학원

전 산 학 과

고 명 필

고명필의 석사 학위논문으로 인준함

심사위원 최 광 훈 인

심사위원 윤 상 균 인

심사위원 임 효 상 인

연세대학교 대학원

2015년 12월 10일

차 례

그림 차례

표 차례

국 문 요 약

강건한 안드로이드 앱을 위한 인텐트 스펙의 설계와 구현

본 논문은 안드로이드에서 인텐트로 발생하는 취약점을 개선하는 인텐트 스펙을 설계하고 그 응용 방법을 제안한다. 안드로이드에서 인텐트는 컴포넌트 간 통신(Inter-Component Communication, ICC)을 하기 위한 메시지이다. 안드로이드 시스템은 인텐트를 전달받아 컴포넌트를 호출하고 컴포넌트가 요구하는 데이터를 전달한다. 안드로이드의 컴포넌트를 함수에 비유하면 인텐트는 함수를 호출할 때 전달되는 인자이다. 안드로이드의 취약점은 화면을 구성하는 액티비티나 백그라운드에서 기능을 제공하는 서비스, 시스템으로 방송되는 메시지를 이용해 기능을 제공하는 브로드캐스트 리시버와 같은 컴포넌트의 요구에 맞지 않는 잘못된 인텐트를 전달하면 어플리케이션이 비정상적으로 종료되거나 안드로이드 OS가 재부팅되는 것이다. 기존의 안드로이드 시스템은 컴파일 시간에 잘못된 인텐트를 검사하지 못해 실행시간에 이 취약점이 드러난다.

본 논문에서는 안드로이드에서 인텐트로 발생하는 취약점을 개선하기 위해 인텐트를 실행시간에 검사하여 방어하는 방법과 취약점 파악을 위해 테스트하는 방법을 제안한다. 기존 연구들의 단점은 컴포넌트가 기대하는 인텐트의 형태를 명확하게 나타낼 수 없다는 것과 테스트하는 인텐트의 형태를 쉽게 변경하기 어려워 다양한 테스트를 위해서 소스코드를 직접 변경해야 한다는 것이다. 이러한 문제점을 개선하기 위해서 본 논문에서는 인텐트 형태를 선언할 수 있는 인텐트 스펙을 제안한다. 인텐트 스펙은 다양한 인텐트의 형태를 쉽게 선언할 수 있고 다양한 인텐트를 생성할 수 있다. 인텐트 스펙을 응용해 인텐트로 발생하는 취약점을 개선하기 위해 네 가지 방법을 제안한다. 첫째, 인텐트 스펙으로 선언된 인텐트와 실행시간에 컴포넌트에 전달된 인텐트를 상호 비교하여 잘못된 인텐트를 직접 방어하는 방법을 제안한다. 둘째, 테스트할 인텐트의 형태를 인텐트 스펙으로 선언하면 선언된 인텐트 스펙을 기반으로 랜덤 인텐트를 생성하여 대상 컴포넌트를 테스트하고 그 결과를 자동으로 파악해주는 방법을 제안한다.

셋째, 소스가 없는 안드로이드 어플리케이션 바이너리 파일로 취약점을 테스트하는 방법이다. 안드로이드 어플리케이션 바이너리 파일에서 인텐트를 추출하여 랜덤 인텐트를 생성한다. 생성된 이 랜덤 인텐트를 이용해 대상 어플리케이션을 테스트하는 방법을 제안한다. 넷째, 안드로이드 스튜디오의 유닛 테스트 도구인 JUnit 테스트를 위한 테스트 케이스를 인텐트 스펙 기반으로 랜덤 생성해주는 자동화 방법을 제안한다.

제안한 방법의 장점은 잘못된 인텐트를 직접 방어하고 비정상 종료를 막아 정상적인 서비스를 제공한다는 것과 인텐트 스펙을 기반으로 인텐트를 자유롭게 생성하여 다양한 방식의 테스트를 진행할 수 있다는 것이다. 실험결과, 제안하는 방법으로 오픈 소스 어플리케이션과 상용 어플리케이션에서 인텐트로 발생하는 취약점을 방어하고 파악할 수 있어 제안한 방법이 유용함을 보였다.

핵심되는 말 : 안드로이드, 인텐트, 인텐트 스펙, 취약점, 테스팅, 강건성

제 1 장

서론

안드로이드[1]는 모바일 폰, TV, 웨어러블 디바이스 등 다양한 스마트 디바이스를 위한 구글의 오픈소스 플랫폼이다. 안드로이드 어플리케이션으로 기업과 사용자를 위한 다양한 서비스가 제공되고 있다. 안드로이드 어플리케이션은 주로 Java[2]로 제작하고 안드로이드 플랫폼 API를 사용한다.

안드로이드 어플리케이션은 사용자 인터페이스를 제공하는 액티비티(Activity) 컴포넌트와 백그라운드에서 기능을 제공하는 서비스(Service) 컴포넌트, 시스템으로 방송되는 방송메시지로 기능을 수행하는 브로드캐스트 리시버(Broadcast Receiver) 컴포넌트, 어플리케이션 간 데이터를 공유하는 컨텐트 프로바이더(Content Provider) 컴포넌트로 구성된다.

안드로이드 플랫폼에서는 인텐트(Intent) 메시지를 이용해 컴포넌트 간 통신(ICC: Inter Component Communication)을 지원한다. 인텐트 메시지는 호출하는 대상 컴포넌트를 지정하고 해당 컴포넌트가 요구하는 데이터를 제공한다. 호출되는 컴포넌트를 함수에 비유하면 인텐트는 함수를 호출하며 전달하는 함수의 인자와 같은 정보이다.

최근 연구 결과에 따르면 안드로이드는 인텐트로 인해 발생하는 취약점이 많다고 한다[3]. 안드로이드 시스템은 컴포넌트에서 요구하는 인텐트 형태와 다르게 누락된 데이터가 있거나 타입이 다른 잘못된 인텐트를 컴포넌트에 전달하면 어플리케이션이 비정상적으로 종료되거나 안드로이드 OS가 재부팅되는 문제가 발생한다. 이러한 문제는 안드로이드 시스템에서 잘못된 인텐트를 컴파일 시간 검출하지 못하기 때문에 발생

한다. 그러므로 실행시간에 잘못된 인텐트가 컴포넌트에 전달되어야 비로소 취약점이 확인된다.

기존 연구들은 이러한 취약점을 방어하고 파악하기 위해 인텐트를 실행시간에 검사하는 방법[4, 5]과 인텐트의 랜덤 생성한 인텐트로 테스트하는 방법[3, 6, 7]을 사용한다. 실행시간에 검사하는 방법은 인텐트의 Extra 필드를 실행시간에 검사해 누락된 값을 기본값으로 대체하는 Dart와 실행시간에 데이터 값이 같은지 비교하는 Jackson 두 가지있다. 테스트 하는 방법은 대상만 지정된 빈 인텐트를 전달하는 Intent Fuzzer와 Intent Fuzzer를 확장한 실험으로 인텐트의 Action, Data, Extra를 랜덤 생성한 인텐트로 테스트하는 jarjarbinks, 정적분석을 통해 Action과 Extra를 파악하고 이 정보를 기반으로 랜덤 생성한 인텐트로 테스트하는 Intent Fuzzer with static analysis 세 가지가 있다.

기존 연구들은 인텐트의 형태를 선언할 수 있는 방법이 없어 인텐트의 일부 필드만을 방어하고, 테스트시 랜덤 생성하는 인텐트를 변경하기 위해 소스코드를 직접 변경해야 하는 단점이 있다.

본 논문에서는 기존 연구들의 문제점을 보안하고 안드로이드에서 인텐트로 발생하는 취약점을 개선하기 위해서 인텐트의 형태를 선언할 수 있는 인텐트 스펙을 제안한다. 또한, 인텐트 스펙을 응용하여 인텐트로 인해 발생하는 취약점을 방어하는 방법 한 가지와 파악하는 세 가지 방법을 제안한다.

첫째, 실행시간 검사방법은 인텐트 스펙으로 컴포넌트가 요구하는 인텐트를 선언하고 선언된 인텐트 스펙과 전달된 인텐트를 실행시간에 검사하여 잘못된 인텐트를 검출한다. 검출된 잘못된 인텐트를 정상 인텐트로 개발자가 복구할 수 있도록 에러 코드와 메시지를 제공한다. 이 방법은 기존 연구에서 일부 필드를 이용해 방어하는 방법과 달리 인텐트 스펙으로 선언한 모든 필드를 검사해 방어한다.

둘째, 유닛 테스트 방법은 테스터가 인텐트 스펙으로 선언한 인텐트를 기반으로 랜덤 생성한 인텐트를 디바이스에 전달하고 디바이스에서 발생하는 취약점을 자동으로 파악하는 자동된 테스트 방법이다. 이 방법은 기존 연구들에서 하지못한 테스트 자동화를 제공하고 인텐트 스펙으로 랜덤 생성하는 인텐트의 형태를 자유롭게 변경할 수

있는 장점이 있다.

셋째, 인텐트 스펙 자동생성 테스트 방법은 안드로이드 설정파일(매니페스트)에 선언되는 인텐트를 이용해 인텐트 스펙을 추출하고 추출된 인텐트 스펙으로 랜덤 생성한 인텐트를 이용해 테스트한다. 안드로이드 설정파일은 프로젝트에도 포함되고 컴파일된 바이너리 파일에도 포함되어 있어 소스코드가 없는 환경에서도 인텐트로 발생하는 취약점을 파악할 수 있다.

넷째, JUnit 테스트 검사 방법은 안드로이드 스튜디오의 JUnit을 이용한 테스트 방법으로 JUnit 테스트 케이스를 인텐트 스펙을 기반으로 자동 생성하는 방법이다. 이 방법은 테스트 케이스를 자동으로 생성해 기존에 테스트 케이스를 자바코드로 직접 작성해야하는 불편함을 제거한다.

본 논문의 구성은 다음과 같다. 2장에서 안드로이드 개요와 관련 연구에 대해서 설명한다. 3장에서는 인텐트 형태를 선언할 수 있는 인텐트 스펙을 제안한다. 4장에서는 인텐트 스펙을 응용한 네 가지 취약점 개선 방안을 제안한다. 네 가지 방법은 실행시간 검사 방법, 인텐트 스펙 기반 랜덤 테스팅 방법, 앱 바이너리에서 인텐트 스펙 자동 생성 방법, 인텐트 스펙 기반 JUnit 테스트 케이스 자동 생성 방법이다. 5장에서는 결론 및 향후 연구를 논의한다.

제 2 장

안드로이드 개요 및 관련 연구

제 1 절 안드로이드 개요

이 절에서는 안드로이드에서 발생하는 취약점에 관해서 살펴보기 위해서 안드로이드 어플리케이션의 구성요소인 컴포넌트와 컴포넌트간 통신 메시지인 인텐트(Intent)에 대해서 살펴본다.

1.1 안드로이드 컴포넌트

안드로이드 어플리케이션은 액티비티(Activity), 서비스(Service), 브로드캐스트 리시버(Broadcast Receiver), 컨텐트 프로바이더(Content provider) 네 가지 컴포넌트로 구성된다. 그림 2.1은 컴포넌트의 관점에서 본 안드로이드 어플리케이션의 구조이다.

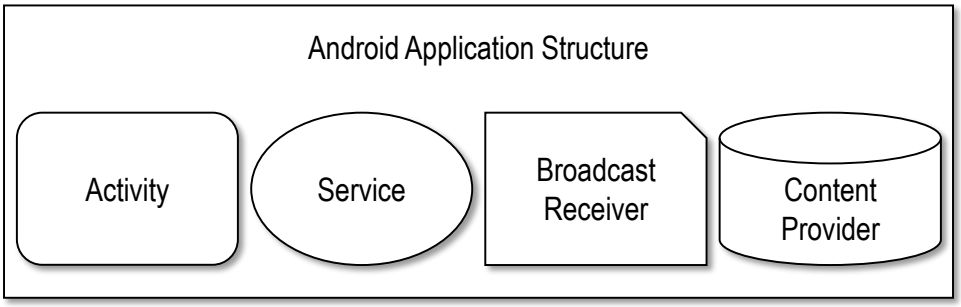

그림 2.1: 컴포넌트 관점으로 본 안드로이드 어플리케이션 구조

액티비티(Activity)는 하나의 화면을 담당한다. 액티비티 내에서 사용자로부터 입력을 받거나 화면에 내용을 출력한다. 서비스(Service)는 백그라운드 작업의 인터페이

스를 담당하는 컴포넌트이다. 액티비티와 달리 사용자 인터페이스가 없고 백그라운드에서 특정한 기능을 수행한다. 브로드캐스트 리시버(Broadcast Receiver)는 시스템 전체에 보내는 메시지를 수신하는 컴포넌트이다. 예를 들어, 시스템 전체로 보내지는 메시지 중 하나인 디바이스의 베터리 정보를 수신하여 일정 이하의 베터리 용량이 되면 사용자에게 알리고 절전상태로 전환하는 기능을 제공한다. 컨텐트 프로바이더(Content provider)는 서로 다른 어플리케이션간 데이터를 공유하기 위해 사용한다. 어플리케이션간에 데이터 공유는 컨텐트 프로바이더의 위치를 전달하고 전달받은 어플리케이션에서 컨텐트 프로바이더의 위치로 접근하여 사용한다.

1.2 인텐트

컴포넌트는 인텐트라는 메시지를 이용해 컴포넌트간 통신을 지원한다. 그림 2.2는 인텐트로 컴포넌트를 호출하고 인텐트가 전달되는 과정이다. 안드로이드 시스템에서는 컴포넌트를 호출하고 호출된 컴포넌트가 요구하는 데이터를 전달하기 위해서 인텐트를 사용한다. 인텐트에 호출할 컴포넌트의 이름과 컴포넌트에게 서비스받을 기능을 지정하고 해당 기능에 필요한 데이터를 추가로 지정한다. 이 인텐트를 안드로이드 시스템에 전달하면 안드로이드 시스템은 해당 컴포넌트를 호출하고 인텐트를 전달해 해당 컴포넌트의 서비스가 실행된다.

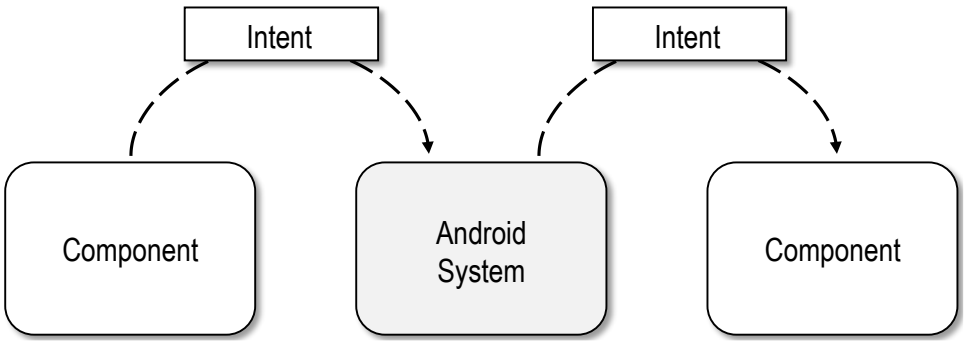

그림 2.2: 인텐트가 전달되는 과정

한 컴포넌트는 다양한 인텐트의 형태를 요구할 수 있다. 일반적으로 수행할 기능을 기술하는 Action 필드에 내용에 따라서 요구하는 데이터가 달리지므로 다른 형태의

인텐트를 요구한다. 인텐트에 지정할 수 있는 필드의 구조는 다음과 같다.

- Action : 컴포넌트가 수행할 기능을 구분
- Data : Action에 필요한 데이터 위치를 URL 형태로 제공
- Category : 실행하는 Action에 대한 추가적인 정보
- Type : Data의 명시적인 타입
- Component : 대상 컴포넌트의 패키지와 클래스 이름
- Extra : 컴포넌트에 제공할 추가적인 정보의 묶음
- Flag : 플랫폼의 컴포넌트 스택을 조정하거나 데이터 권한을 부여

인텐트로 대상 컴포넌트를 호출하는 방법은 명시적으로 컴포넌트의 클래스를 호출하는 명시적인 인텐트(Explicit Intent)와 사용자가 기능을 수행할 수 있는 컴포넌트를 직접 선택해 호출하는 묵시적인 인텐트(Implicit Intent) 두 가지이다. 명시적인 인텐트는 인텐트의 Component 필드에 대상 컴포넌트의 클래스가 명확하게 기술되어 있는 형태로 안드로이드 시스템은 Component 필드를 확인해 해당 컴포넌트를 생성한다. 묵시적인 인텐트는 인텐트 Component 필드에 대상 컴포넌트의 클래스를 기술하지 않고, 기능을 구분하는 Action 필드와 Data 필드, Type 필드를 확인하여 해당 기능을 수행할 수 있는 컴포넌트의 리스트를 화면에 출력한다. 사용자가 해당 기능을 수행할 컴포넌트를 선택하면 안드로이드 시스템에서 선택한 컴포넌트를 생성한다.

묵시적인 인텐트를 전달받는 컴포넌트는 어플리케이션의 설정파일인 안드로이드 매니페스트(AndroidManifest)에 컴포넌트가 제공하는 기능(Action)과 원하는 타입(Type)을 기술해야 한다. 컴포넌트가 요구하는 인텐트의 Action과 Data의 타입, Category 구조를 매니페스트 파일에 선언하면 안드로이드 시스템에서는 선언된 정보를 확인해 해당 인텐트를 처리할 수 있는 컴포넌트의 리스트를 결정한다.

1.3 인텐트로 발생하는 취약점

컴포넌트간 통신을 지원하는 인텐트로 발생하는 취약점을 예를 통해서 설명한다. 인텐트를 사용하는 간단한 예인 그림 2.3은 간단한 메모를 생성하거나 수정하는 액티비티 컴포넌트이다. 이 액티비티는 호출과 함께 인텐트가 전달되면 인텐트의 Action

```
public class Edit extends Activity {
   String title, content;
   void onCreate(Bundle savedInstanceState){
      Intent intent = getIntent();
      String action = intent.getAction();
      if (Intent.ACTION_CREATE.equals(action)) {
         title = "new";
         content = "memo";
      } else if (Intent.ACTION_EDIT.equals(action)) {
         title = intent.getStringExtra("title");
         content = intent.getStringExtra("content");
      }
   // Display title, content
   }
   // Skip
}
```

그림 2.3: 인텐트를 사용하는 취약한 Edit 액티비티

필드를 확인해 메모를 생성하거나 수정하는 기능을 수행하고, 기능을 수행하며 추가 데이터가 필요한 경우는 인텐트 Extra 필드를 사용한다. 이 액티비티는 Action 필드에 기술된 내용에 따라서 다른 인텐트를 요구하므로 컴포넌트가 요구하는 인텐트의 형태는 `CREATE`와 `EDIT` 두 가지이다. 첫째, Action이 `CREATE`인 인텐트이다. 둘째, Action이 `EDIT`이고 Extra로 `title`과 `content`의 값이 `String` 타입인 인텐트이다.

그림 2.3 Edit 컴포넌트의 코드를 살펴보면, 안드로이드 시스템에서 `Edit` 액티비티가 실행되면 자동으로 `OnCreate` 메서드가 실행되고, `getIntent` 메서드로 안드로이드 시스템에서 전달된 인텐트의 정보를 확인한다. 전달된 인텐트에서 `getAction` 메서드로 Action 필드의 정보를 확인하여 Action이 `CREATE`이면 멤버변수인 `title`과 `content`를 초기값으로 설정하여 화면에 생성할 내용을 출력한다. Action이 `EDIT`이면 Extra의 정보에서 키가 `title`인 `String`타입 값을 전달받아 멤버변수 `title`에 할당하고, 키가 `content`인 `String`타입 값을 전달받아 멤버변수 `content`에 할당해 화면에 수정할 내용을 출력한다.

이 액티비티 컴포넌트도 데이터가 누락되거나 데이터의 타입이 다른 잘못된 인텐트를 전달하면 실행시간에 `NullPointerException`이 발생하여 비정상적으로 종료되는 문제가 있다. 이 문제는 첫째, Action이 `CREATE`나 `EDIT`가 아닌 경우 `title`과 `content`의 값이 `null`이므로 비정상 종료된다. 둘째, Action이 `EDIT`인 경우 `title`과 `content`키의 값이 할당되지 않았거나 타입이 다른 경우 비정상 종료된다. 어플리케이션이 잘못된 인텐트를 전달받아 비정상 종료된 화면은 그림 2.4와 같다.

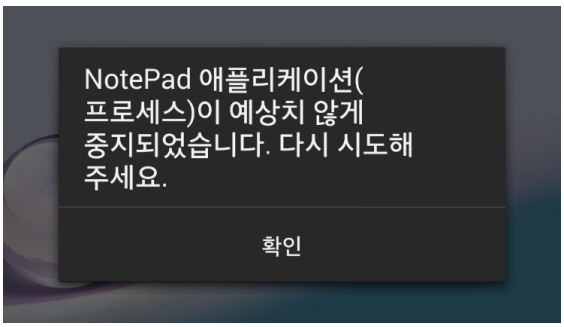

그림 2.4: 잘못된 인텐트로 비정상 종료된 어플리케이션의 예

제 2 절 관련 연구

이 절에서는 인텐트로 발생하는 취약점을 방어하기 위한 실행시간 검사 방법과 취약점을 파악하기 위한 테스팅 방법에 관한 관련연구에 대해서 살펴본다.

2.1 실행시간 검사 관련 연구

실행시간 검사 관련 연구에서는 제안한 실행시간 검사 방법과 유사한 관련 연구에 관해서 설명한다. 관련 연구는 인텐트 Extra에 누락된 데이터를 기본값으로 변경해주는 Dart[4]와 데이터의 값이 같은지 확인하는 Jackson[5] 두 가지 방법이 있다. 두 가지 방식을 간단하게 소개하고 제안한 방법과의 유사점, 차이점을 비교한다.

• Dart[4]는 인텐트에서 Extra 필드의 데이터의 누락으로 인해 발생하는 취약점을 방어하는 방법이다. 컴포넌트가 요구하는 Extra 정보 중 일부가 누락된 경우 컴포넌트가 누락된 정보를 리턴받아 사용하면 `NullPointerException`이 발생한다. Dart는

이러한 문제를 해결하기 위해서 실행시간에 전달된 인텐트의 Extra를 검사하고 누락된 Extra를 기본값으로 변경해주는 방식을 사용한다.

```
//Step 1. Default value
@Nullable @InjectExtra String extra1 = "Default Extra";
@Nullable @InjectExtra int extra2 = "123";

class ExampleActivity extends Activity {
  //Step 2. Description
  @InjectExtra("key_1") String extra1;
  @InjectExtra("key_2") int extra2;

  @Override public void onCreate(...) {
    //Step 3. injected
    Dart.inject(this);
    // TODO Use "injected" extras...
  }
}
```

그림 2.5: Dart 사용의 예

그림 2.5는 Dart 사용의 예이다. 기본값을 정의하는 1단계에서 원하는 Extra 기본값을 정의하고, 2단계에서 키와 타입의 값을 기술한다. 마지막으로 3단계에서 `Dart.inject` 메서드를 이용해 전달된 인텐트를 검사하고 만약 누락된 Extra가 있으면 누락된 Extra를 기본값으로 추가한다.

Dart와 제안한 방법은 실행시간에 전달된 인텐트를 검사하여 방어한다는 점이 유사하다. 하지만 Dart에 단점은 Annotation을 이용해 여러 단계를 나눠 정의하므로 복잡하다는 것과 인텐트의 Extra 정보만 기본값으로 변경해주는 제한적인 방어 방법이라는 것이다. 따라서 다른 인텐트의 필드에 문제가 발생한 것은 검출하거나 방어할 수 없다.

• Jackson[5]은 인터넷에서 자료를 주고받을 때 자료의 형태를 텍스트로 표현하는 방식이다. Jackson의 Assert는 실행시간에 데이터의 값을 검사하여 결과를 돌려주는 점에서 제안한 방법과 유사점이 있다. Jackson에서 데이터를 검사하는 방법에 대해서 살펴보고 유사점과 차이점을 설명한다. 그림 2.6은 Jackson의 실행시간 검사방법의 예

이다.

```
String result = "{id:1,stuff:[[1,2],[2,3],[],[3,4]]}";
//1. Result : Pass
JSONAssert.assertEquals(
    "{id:1,stuff:[[1,2],[2,3],[],[3,4]]}", result, true);

//2. Result : Pass
JSONAssert.assertEquals(
    "{id:1,stuff:[[4,3],[3,2],[],[1,2]]}", result, false);
```

그림 2.6: Jackson Assert 사용의 예

Jackson은 데이터 구조와 값을 String 타입으로 표현한다. 1단계 `JSONAssert.assertEquals` 메서드의 첫 번째 인자는 원하는 데이터의 값을 텍스트로 표현하고 두 번째 인자는 비교할 값을 넣는다. 세 번째 인자인 `true`는 첫 번째 인자와 두 번째 인자가 같다면 `true`를 리턴하라는 의미이다. 2단계 `JSONAssert.assertEquals` 메서드는 세 번째 인자가 `false`이므로 첫 번째 인자와 두 번째 인자가 다르면 `true`를 리턴하라는 의미이다. 따라서 첫 번째 인자와 두 번째 인자가 같은 1단계와 첫 번째 인자와 두 번째 인자가 다른 2단계 결과는 모두 `true`이다.

Jackson과 제안한 방법의 유사점은 실행시간에 값을 비교하는 것이다. 하지만 제안한 방법은 실행시간에 전달된 인텐트의 값 하나를 검사하는 것이 아니라 컴포넌트가 요구하는 인텐트들을 선언하고 선언된 인텐트들과 전달된 인텐트의 구조를 검사하여 잘못된 인텐트를 검출하는 점과 잘못된 인텐트를 검출하여 정상적인 인텐트로 복구할 기회를 제공한다는 점에서 상당한 차이가 있다.

2.2 테스팅 관련 연구

테스팅 관련 연구에서는 제안한 테스팅 방법과 유사한 관련 연구에 관해서 설명한다. 인텐트로 발생하는 취약점을 파악하기 위한 기존 연구들은 `Intent Fuzzer`[6], `JarJarBinks`[3], `Intent Fuzzer:Crafting Intents of Death`[7], 정적분석 툴 `ComDroid`[8] 총 네 가지이다. 기존 연구[3, 6, 7]는 랜덤 인텐트로 컴포넌트가 취약

함을 테스트를 통해 보인다. 이 테스트에서는 일부 필드를 랜덤 생성한 인텐트를 컴포넌트에게 전달하는 방식을 사용한다. 정적분석툴[8]은 안드로이드 바이너리를 정적으로 분석해 취약점이 발생할 수 있는 부분을 경고해주는 방식을 사용한다. 기존 연구들은 랜덤한 인텐트를 이용해 테스트하거나 바이너리 코드를 정적으로 분석해 취약함을 확인하는 방법을 사용한다. 이 네 가지 방식을 간단하게 소개하고 제안한 방법과의 유사점, 차이점을 비교한다.

• `Intent Fuzzer`[6]툴은 안드로이드에 설치된 어플리케이션의 컴포넌트에 `Null` 인텐트를 전달하여 어플리케이션이 비정상 종료됨을 보인다. `Intent Fuzzer`툴은 안드로이드 디바이스에서 컴포넌트를 읽어오는 단계와 대상 컴포넌트만 지정한 `Null` 인텐트를 생성하는 단계, 인텐트를 전달하는 단계 총 세 단계로 구성된다. `PackageManager` 클래스의 `getPackageManager` 메서드로 디바이스에 설치된 어플리케이션의 모든 컴포넌트를 읽어온다. 인텐트 생성은 Component 필드에 대상 컴포넌트만 지정한다. 지정되지 않은 다른 필드는 모두 `Null`이다. 생성된 `Null` 인텐트를 안드로이드 시스템에 전달한다. 안드로이드 시스템은 Component 필드를 확인해 대상 컴포넌트를 생성하고 인텐트를 전달한다. 이때, `Null` 인텐트로 취약점이 있는 컴포넌트는 비정상 종료된다.

`Intent Fuzzer`툴은 안드로이드 어플리케이션으로 구현해 테스트를 자동화할 수 없는 제약과 인텐트를 생성 시 다양한 구조를 만들 수 없는 제약, 오류 발생을 판단하기 어려운 제약으로 총 세 가지 제약사항이 있다. 테스트를 자동화할 수 없는 이유는 이전에 실행된 컴포넌트가 다음 실행에 영향을 미쳐 다음 실행에서 다른 결과를 낼 수 있다. 하지만 안드로이드 어플리케이션간에는 강제 종료시킬 방법이 없다. 이러한 문제로 테스터가 이전에 실행된 화면을 직접 종료시키지 않으면 정확한 테스트를 할 수 없어 자동화된 테스트를 진행할 수 없다. 항상 `Null` 인텐트로만 테스트한다는 제약이다. 컴포넌트마다 요구하는 인텐트의 구조나 값이 다르다. 하지만 항상 `Null` 인텐트로 테스트를 진행하므로 다양한 구조를 사용하는 컴포넌트의 경우에는 정확한 테스트를 진행이 어렵다. 오류를 판단하기 어려운 제약이다. 생성한 인텐트를 전달하는 프로그램이므로 정상 실행, 정상 종료, 비정상 종료를 명확하게 판단하기가 불가능하다. 테스

터가 실행되는 화면을 통해서 취약점 발생 여부를 판단해야 한다.

• JarJarBinks[3]툴은 Intent Fuzzer툴을 확장한 테스트 방법으로 안드로이드 어플리케이션 설정 파일인 매니페스트(AndroidManifest)에 선언된 인텐트 정보를 기반으로 생성한 인텐트를 컴포넌트에 전달해 어플리케이션이 비정상 종료됨을 보인다.

JarJarBinks툴은 매니페스트(AndroidManifest)에서 컴포넌트 정보를 추출하는 단계와 Action, Data 조합하고 Extra를 랜덤으로 생성해 인텐트를 만드는 단계, 인텐트를 전달하는 단계 총 세 단계로 구성된다. 안드로이드 어플리케이션의 설정 정보가 기술되는 매니페스트(AndroidManifest)에서 인텐트의 정보를 추출한다. 추출한 Action과 Data를 기반으로 명시적인 인텐트(Explicit Intent)와 묵시적인 인텐트(Implicit Intent)를 생성한다. 명시적 인텐트의 경우 Semi-valid Action and Data와 Blank Action or Data, Random Action or Data, Random Extras 네 가지 모드를 사용한다. Semi-valid Action and Data 모드는 인텐트 필터에서 추출한 Action과 Data로 조합하여 인텐트를 생성한다. Blank Action or Data 모드는 Action이나 Data 중 하나로 인텐트를 생성한다. Random Action or Data 모드는 인텐트 필터에서 추출한 Action이나 Data 중 하나로 인텐트를 생성하고, 다른 하나는 랜덤으로 인텐트를 생성한다. Random Extras 모드는 추출한 Action과 Data와 랜덤으로 생성한 Extra로 인텐트를 만들어 컴포넌트에게 전달한다. 묵시적 인텐트의 경우 Valid Intent와 Semi-valid Intent 두 가지 모드를 사용한다. Valid Intent 모드는 추출한 Action과 Data필드로 인텐트를 생성하고, Semi-valid Intent는 추출한 Action과 Data에 Category를 덧붙여 생성한다. 생성한 인텐트를 안드로이드 시스템에 전달해 컴포넌트를 호출하고 컴포넌트에 인텐트를 전달한다.

JarJarBinks툴은 Action과 Data, 랜덤 생성한 Extra의 값 이용해 Intent Fuzzer 툴보다 다양한 인텐트를 만들 수 있다는 장점이 있지만 랜덤 인텐트 생성 시 인텐트의 모든 필드를 이용하지 않고 3개의 필드만 랜덤하게 생성하는 제약과 테스트를 자동화할 수 없다는 제약, 오류 발생을 판단하기 어려운 제약이 여전히 존재한다.

• Intent Fuzzer with static analysis[7]는 Intent Fuzzer툴을 확장한 방법으로 안드로이드 바이트코드(bytecode)를 정적 분석해 컴포넌트가 요구하는 Extra

의 키와 타입을 파악하고, 인텐트를 생성 시 파악한 Extra의 값을 랜덤으로 변경하여 컴포넌트에 전달해 코드가 커버되는 범위와 디바이스에서 발생하는 로그를 수집한다.

`Intent Fuzzer with static analysis`툴은 Action과 Extra의 정보를 추출하는 단계와 추출한 정보에서 Extra 값을 변경한 랜덤 인텐트를 생성하는 단계, 생성한 인텐트를 전달하는 단계로 총 세 단계로 구성된다. Action과 Extra의 키와 타입을 추출하기 위해 정적 분석 방법을 이용한다. 컴포넌트가 요구하는 인텐트의 Action을 기술할 수 있는 안드로이드 매니페스트(AndroidManifest.xml) 파일에서 컴포넌트가 요구하는 Action을 추출한다. Action이 파악된 컴포넌트의 바이트코드(bytecode)를 분석해 실제 소스에서 사용하는 Extra의 정보를 얻는다. Extra의 키는 문자열 상수이므로 사용하는 문자열이 Extra의 키이다. 키의 타입은 getter 메서드(예를 들어, `getStringExtra`)의 이름을 이용해 타입을 확인한다. 정적으로 분석된 정보인 Action과 Extra를 이용한 랜덤 인텐트를 생성한다. 랜덤 인텐트 생성 시 컴포넌트가 요구하는 Action에 Extra 키와 타입은 그대로 사용하고, Extra의 값을 랜덤으로 생성한다. PC와 안드로이드 디바이스를 연결할 수 있는 ADB[9]로 랜덤 생성한 인텐트를 디바이스에 전달한다. ADB는 PC에서 생성한 인텐트를 안드로이드 시스템에 전달하고 실행 중인 어플리케이션을 종료할 수 있는 기능을 제공한다. 테스트 시 이전 결과에 영향을 받지 않도록 실행 중인 어플리케이션을 종료시켜 실험할 수 있다.

`Intent Fuzzer with static analysis`툴은 PC와 안드로이드 디바이스를 연결하는 ADB를 이용해 자동화하기 쉽다는 장점과 Extra의 키를 추출해 이용하기 때문에 `JarJarBinks`툴보다 정확하게 Extra를 만들 수 있다는 장점이 있다. 하지만 인텐트의 모든 필드를 이용해 인텐트를 생성하지 않고 Action과 Extra만을 이용한다는 제약이 있어 코드 커버리지가 낮고 자동화된 결과를 제공하지 않아 오류 발생을 판단하기 어려운 제약이 있다.

• `ComDroid`[8]툴은 다른 방법들과 달리 실행하지 않고 바이트코드(bytecode)를 정적으로 분석해 인텐트로 발생할 수 있는 취약한 부분을 알린다. 안드로이드 매니페스트 파일(AndroidManifest.xml)에서 어플리케이션이 사용할 수 있는 안드로이드 API의 허가권(Permission)과 다른 어플리케이션이 컴포넌트를 호출할 수 있는지를 확

```
Intent intent = new Intent();

if(...){
   intent.setAction(...);
} else if() {
   intent.setAction(...);
   intent.setData(...);
} else {
   intent.setAction(...);
   intent.setData(...);
   intent.setComponent(...);
}

startActivity(intent);
```

그림 2.7: 인텐트를 생성하고 전달하는 예

인한다. 어플리케이션에서 사용할 수 있는 안드로이드 API 허가권을 검사해 자동으로 부여되는 허가권은 Normal, 어플리케이션 인스톨 시 사용자에게 동의를 받아야 하는 허가권은 Dangerous, 같은 개발자만 접근할 수 있는 어플리케이션은 Signature, 안드로이드 디바이스에 미리 설치된 어플리케이션은 SignatureOrSystem으로 분류한다. 인텐트 생성과 전송하는 부분을 확인해 명시적인 인텐트 사용 여부를 확인한다. 묵시적인 인텐트 형태를 사용하는 경우 악의적인 어플리케이션이 인텐트를 전달받아 정보를 이용할 수 있으므로 취약한 부분으로 판단하고 알린다.

ComDroid툴은 코드 전체를 정적 분석하므로 테스팅 방법보다 코드 커버리지가 높다는 장점이 있다. 하지만 취약하지 않은 코드를 취약한 코드로 알리거나 취약한 코드를 취약하지 않은 코드로 알리는 경우가 있다. 정적 분석에서 잘못된 알림을 하는 경우는 조건문으로 분기가 발생하는 경우이다. 예를 들어, 그림 2.7는 Intent 생성하는 코드의 일부로 ComDroid툴에서 알림을 잘못하는 경우이다. 이 코드는 조건에 따라서 다른 형태의 인텐트를 생성해 startActivity 메서드로 액티비티 컴포넌트를 호출한다. 정적 분석은 코드를 직접 실행하지 않는 방식이므로 실행시간에 어떤 조건에 매칭되는지 판단하기 어렵다. 그러므로 ComDroid툴은 조건으로 판단되는 분기

를 정확하게 따지지 않고 모든 분기의 코드를 합쳐서 실행된다고 해석한다. 첫 번째 분기나 두 번째 분기는 묵시적인 인텐트를 사용하므로 악의적인 컴포넌트가 인텐트를 전달받을 수 있으므로 ComDroid툴 기준으로는 잠재적 취약점을 가진 코드로 알려야 한다. 하지만 ComDroid툴은 모든 분기의 코드를 합쳐서 판단하므로 마지막 분기의 setComponent로 이 코드는 항상 명시적인 인텐트를 만드는 것으로 판단하여 묵시적인 인텐트를 사용하지 않는 경우로 잘못 판단해 취약점이 없는 코드로 알린다. 정적 분석을 통해 실행하더라도 취약점이 발생할 수 있다는 제약점이다. 예를 들어, 안드로이드에 저장된 전화번호에 접근할 수 없는 컴포넌트에 전화번호를 Data로 제공하면 전달받은 컴포넌트는 전화번호에 접근할 수 있는 허가권이 없으므로 실행시간에 에러를 발생시킬 것이다.

기존 연구들은 공통적인 제약사항은 인텐트를 명확하게 선언할 방법이 존재하지 않아 인텐트의 필드 중 일부를 이용해 실행시간에 검사하여 방어하는 것과 필드 중 일부를 이용해 테스트하는 방법을 사용한다는 것이다. 이러한 문제를 해결하기 위해서 인텐트의 형태를 명확하게 선언할 수 있는 방법을 설계하고 선언할 수 있는 방법을 이용해 실행시간 검사 방법과 테스팅 방법을 제안한다.

제 3 장

인텐트 스펙

기존 연구들은 인텐트의 형태를 명확하게 선언할 수 있는 방법이 없다. 이 장에서는 인텐트를 명확하게 선언할 수 있는 방법인 인텐트 스펙을 기반으로 안드로이드에서 인텐트로 발생하는 취약점을 개선한다. 인텐트 스펙은 인텐트의 구조와 정보를 자유롭게 선언하는 방법이다. 기존 안드로이드에서는 인텐트의 형태를 명확하고 쉽게 선언하는 방법이 없다. 안드로이드에서 인텐트를 표현하는 기존 방법은 어플리케이션 설정파일인 안드로이드 매니페스트(AndroidManifest)에서 인텐트 정보의 일부 파악하고 컴포넌트의 소스코드에서 흐름과 분기, 이벤트를 따라가면서 분석해야 하는 어려움이 있다. 인텐트 스펙을 이용하면 다양한 형태의 인텐트를 쉽고 명확하게 표현할 수 있고 다양한 인텐트를 쉽게 생성할 수 있는 장점이 있다. 기존 안드로이드에서 인텐트를 선언하는 방법과 소스코드에서 인텐트를 사용하는 방법에 대해서 살펴보고 인텐트 스펙을 이용해 선언하는 방법에 대해서 설명한다.

```
<activity android:name="EditActivity">
    <intent-filter>
        <action android:name="android.intent.action.CREATE"
           />
        <action android:name="android.intent.action.EDIT"/>
    </intent-filter>
</activity>
```

그림 3.1: 매니페스트에 선언된 인텐트 정보의 예

그림 3.1은 매니페스트에 선언된 인텐트 정보의 예이다. 예에서 EditActivity 컴포넌트가 요구하는 인텐트는 Action이 CREATE나 EDIT이다. 매니페스트에 선언된 인텐트의 정보는 묵시적 인텐트(Implicit Intent)를 전달받기 위해 부가적으로 작성하는 옵션이다. 또한, 매니페스트에 표현할 수 있는 인텐트의 필드는 Action과 Category, Type으로 제한적이다. 그러므로 안드로이드 시스템에서 인텐트 모든 필드를 명확하게 선언할 방법이 없다.

```
Intent intent = getIntent();
String action = intent.getAction();
if (Intent.ACTION_CREATE.equals(action)) {
   title = "new";
   content = "memo";
} else if (Intent.ACTION_EDIT.equals(action)) {
   title = intent.getStringExtra("title");
   content = intent.getStringExtra("content");
}
```

그림 3.2: 소스코드로 표현한 인텐트 형태

그림 3.2는 소스코드에 표현된 인텐트의 예이다. EditActivity 컴포넌트에 전달된 인텐트를 getIntent 메서드를 이용해 리턴받는다. getAction 메서드를 이용해 전달된 인텐트에 포함된 Action을 확인해 ACTION_CREATE, ACTION_EDIT과 비교하고, 만약 ACTION_EDIT이면 getStringExtra 메서드로 String 타입의 title과 content의 값을 전달받는다. 소스코드를 분석해야 컴포넌트가 요구하는 인텐트의 정보를 자세하게 파악할 수 있지만 인텐트를 사용하는 곳이 소스코드 여러부분으로 분포되어 있어 흐름과 분기, 이벤트를 따라가면서 분석해야하므로 어렵다.

기존 안드로이드에서 인텐트를 사용하는 방법은 인텐트 형태가 매니페스트와 소스코드에 분산되어 인텐트의 형태를 명확하게 파악하기 힘들다는 단점이 있다. 특히 소스코드에서 인텐트를 사용하는 곳이 여러 이벤트나 조건에 따라서 분기되어 있다면 사용하는 인텐트의 형태는 더욱 파악하기 힘들다. 이러한 문제점은 컴포넌트가 요구하는 인텐트의 형태를 선언할 방법이 없어 발생한다.

인텐트 스펙을 사용하면 컴포넌트가 요구하는 인텐트의 형태를 자유롭게 선언할 수 있고, 선언된 인텐트 스펙을 응용해 인텐트로 발생하는 취약점을 방어할 수 있다. 4장에서 인텐트 스펙을 응용하여 인텐트로 발생하는 취약점을 개선하는 방법을 알아본다.

제 1 절 인텐트 스펙의 예

이 절에서는 인텐트 스펙을 이용해 인텐트의 형태를 선언하는 방법을 예제를 통해 설명한다. 안드로이드 설정파일인 매니페스트(AndroidManifest)에 선언된 인텐트 형태인 그림 3.1을 인텐트 스펙으로 선언하는 예제를 살펴보고, 소스코드에 표현된 인텐트 형태인 그림 3.2에서 인텐트 스펙으로 선언하는 예제를 살펴본다.

안드로이드 매니페스트에 표현된 그림 3.1의 인텐트의 형태를 인텐트 스펙으로 선언하는 예를 살펴본다.

```
<activity android:name="EditActivity">
    <intent-filter>
        <action android:name="android.intent.action.CREATE"
           />
        <action android:name="android.intent.action.EDIT"/>
    </intent-filter>
</activity>
```

(a) 안드로이드 설정파인 매니페스트에 선언된 인텐트

```
//Intent Specification
   { act = android.intent.action.CREATE }
|| { act = android.intent.action.EDIT }
```

(b) 인텐트 스펙으로 선언된 인텐트 정보

그림 3.3: 인텐트 스펙으로 표현한 매니페스트의 인텐트 정보

`EditActivity` 컴포넌트는 인텐트 스펙을 이용해 그림 3.3과 같이 표현된다. `EditActivity` 컴포넌트는 Action이 `android.intent.action.CREATE`이거나 `android.intent.action.EDIT`인 두 가지의 인텐트 형태를 요구한다. 인텐트 스

펙에서 {}는 하나의 인텐트를 나타내고 ||는 추가 인텐트가 더 있다는 것을 의미한다. 이 예제에서는 CREATE와 EDIT 두 가지 형태의 인텐트를 요구하므로 {} || {}로 기술된다. Action필드는 인텐트 스펙에서 act로 나타낸다. CREATE의 정의는 android.intent.action.CREATE이므로 {act = android.intent.action.CREATE}로 표현되고 EDIT의 정의는 android.intent.action.EDIT이므로 인텐트 스펙에서 {act=android.intent.action.EDIT}로 표현된다. 전체 인텐트 스펙은 {act = android.intent.action.CREATE} || {act = android.intent.action.EDIT}로 선언한다.

```
Intent intent = getIntent();
String action = intent.getAction();
if (Intent.ACTION_CREATE.equals(action)) {
   title = "new";
   content = "memo";
} else if (Intent.ACTION_EDIT.equals(action)) {
   title = intent.getStringExtra("title");
   content = intent.getStringExtra("content");
}
```

(a) 안드로이드 소스파일에 표현된 인텐트

```
//Intent Specification Without Extra Value
   { act = android.intent.action.CREATE }
|| { act = android.intent.action.EDIT
     [ title = String, content = String ] }
```

(b) 값이 없는 형태의 인텐트 스펙으로 선언된 인텐트

```
//Intent Specification With Extra Value
   { act = android.intent.action.CREATE }
|| { act = android.intent.action.EDIT
     [ title = String "edit", content = String "memo" ] }
```

(c) 값이 있는 형태의 인텐트 스펙으로 선언된 인텐트

그림 3.4: 인텐트 스펙으로 표현한 소스코드의 인텐트 정보

소스코드에 표현된 그림 3.2의 인텐트의 형태를 인텐트 스펙으로 선언하는 예를 살펴본다. `EditActivity` 컴포넌트는 인텐트 스펙을 이용해 그림 3.4와 같이 표현된다. `EditActivity` 컴포넌트의 소스에서는 Action으로 분기하여 메모를 만드는 것과 수정하는 것 두 가지의 기능을 제공한다. 그러므로 두 가지 형태의 인텐트를 전달받아 다른 서비스를 제공하는 것을 코드에서 확인할 수 있다. 두 가지의 인텐트는 인텐트 스펙에서 {} || {}로 선언한다. Action이 `CREATE`이면 `title`과 `content`에 초기값을 설정하므로 추가적인 인텐트 정보를 사용하지 않는다. 인텐트 스펙으로 `act=android.intent.action.CREATE`로 선언된다. Action이 `EDIT`이면 인텐트 스펙으로 `act=android.intent.action.EDIT`이고 `EDIT` Action의 인텐트는 추가적인 Extra를 사용한다. Extra의 타입은 Extra의 값을 얻기 위해 `getStringExtra` 메서드를 이용하므로 String 타입이다. `getStringExtra`에 전달되는 인수가 `title`과 `content`이므로 키는 `title`과 `content`임을 알 수 있다. 인텐트 스펙에서 Extra는 "[Key = Type]"로 표현된다. `EDIT`의 `title` Extra는 인텐트 스펙에서 [title = String]으로 표현할 수 있고, `content` Extra는 인텐트 스펙에서 [content = String]으로 표현할 수 있다. 전체 Extra는 `[title = String, content = String ]`으로 표현된다. 전체 인텐트 스펙은 {`act = android.intent.action.CREATE`} || {`act = android.intent.action.EDIT [title = String, content = String ]`}이다. 4장에서 제안할 인텐트로 발생하는 취약점 파악을 위한 테스팅 프레임워크들은 Extra의 키와 값을 명시하는 방법을 사용한다. 그러므로 값을 포함해 Extra를 선언하는 방법으로 인텐트 스펙에서 "[Key = Type Value]"로 선언한다. 만약 `title`, `content`의 키의 값까지 포함한 인텐트는 {`act = android.intent.action.EDIT [title = String "edit", content = String "memo"]`}로 선언한다. 전체 인텐트 스펙은 {`act = android.intent.action.CREATE`} || {`act = android.intent.action.EDIT [title = String "edit", content = String "memo"]`}로 선언한다.

인텐트의 형태를 자유롭게 선언할 수 있는 인텐트 스펙은 이전에 시도된 적이 없는 최초의 방법이다. 이 방법은 파악하기 어려운 인텐트의 형태를 쉽고 명확하게 선언할

수 있다. 또한, 인텐트 스펙으로 다양한 인탠트를 쉽게 생성할 수 있어 인텐트를 이용하는 다양한 응용프로그램에 활용할 수 있다.

제 2 절 문법

이 절에서는 앞 절에서 인텐트 스펙으로 인텐트를 선언할 때 사용한 인텐스 스펙 문법의 전체를 상세히 살펴본 후, 문법에 맞추어 인텐트 스펙을 기술하는 방법을 설명한다. 인텐트 스펙을 기술할 때 사용하는 전체 문법은 표 3.1과 같다.

표 3.1: 인텐트 스펙 문법

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=IDOROTHER
TYPE	::= typ=IDOROTHER
COMPONENT	::= cmp=COMPTYPE ID/(ID | .ID)
COMPTYPE	::= Activity | Service | BroadcastReceiver | ContentProvider
	| ε – Deprecated, but for compatibility where necessary.
EXTRA	::= [ID=EXTRAVALUE EXTRASUB]
	(Keys are a sequence of alphabets.)
EXTRASUB	::= , ID=EXTRAVALUE EXTRASUB
	| ε
FLAG	::= flg=[ID FLAGSUB]
FLAGSUB	::= , ID FLAGSUB
	| ε

EXTRAVALUE	::= String (“ALL”)?
	| boolean (BOOL)?
	| int (INT)?
	| long (LONG)?
	| float (FLOAT)?
	| uri (IDOROTHER)?
	| component (ID (ID | .ID))?
	| int[] (INTARRAY)?
	| long[] (LONGARRAY)?
	| float[] (FLOATARRAY)?
INTARRAY	::= INT INTARRAYSUB
INTARRAYSUB	::= , INT
	| ε
LONGARRAY	::= LONG LONGARRAYSUB
LONGARRAYSUB	::= , LONG
	| ε
FLOATARRAY	::= FLOAT FLOATARRAYSUB
FLOATARRAYSUB	::= , FLOAT
	| ε
LETTER	::= (A - Z | a - z)+
ID	::= LETTER (A - Z | a - z | 0 - 9 | _ | . | $)*
IDOROTHER	::= LETTER (A - Z | a - z | 0 - 9 | _ | . | / | : | * | ? | !)*
BOOL	::= True | False
INT	::= (-)?(0 - 9)+
LONG	::= (-)?(0 - 9)+
FLOAT	::= (-)?(0 - 9)+.(0 - 9)+

- INTENT는 인텐트 표현한다. 컴포넌트가 요구하는 인텐트 하나는 {}로 선언한다. 컴포넌트가 요구하는 인텐트가 여러 개일 경우는 || {}를 추가로 붙여 {} || {}로 선언한다.
- FIELDS는 인텐트의 필드를 표현한다. FIELDS는 재귀하며 일곱 개의 필드로 구성된 인텐트의 구조에서 필요한 만큼의 필드를 선언한다.
- ACTION은 기능을 표현하기 위해 사용하며 인텐트 스펙에서 act=...로 표현한다. Action의 값은 문자열 형태이다. 만약 Action이 EDIT이면 실제 정의된 상수는 문자열로 android.intent.action.EDIT이므로 act=android.intent.action.EDIT로 선언한다.
- CATEGORY는 Action의 기능을 세부적으로 나누기 위해 사용하고 인텐트 스

펙에서 cat=[...]로 표현한다. Category의 값은 문자열 형태의 배열이므로 Category가 android.intent.category.DEFAULT와 android.intent.category.BROWSABLE이라면 cat=[android.intent.category.DEFAULT, android.intent.category.BROWSABLE]로 선언한다.

- DATA는 데이터의 위치를 기술해 다른 어플리케이션과 데이터를 공유하기 위해 사용한다. 인텐트 스펙에서 dat=...로 표현한다. 다른 어플리케이션과 공유하려는 Data의 위치가 content://contacts/people/1이라면 인텐트 스펙에서는 dat=를 입력하고 나머지 위치를 기술하여 dat=content://contacts/people/1로 선언한다. 또한, DATA 필드는 키워드를 지원한다. non-null 키워드로 dat=non-null은 데이터가 필요하다는 의미이다.
- TYPE은 Data의 타입을 표현하고 인텐트 스펙에서 typ=...로 표현한다. 만약 Data의 타입이 문자열 형태인 text/plain이라면 typ=text/plain으로 선언한다.
- COMPONENT는 컴포넌트의 종류와 패키지 이름, 컴포넌트의 이름을 기술하기 위해서 사용하고 인텐트 스펙에서 cmp=컴포넌트의 종류 패키지이름/컴포넌트 이름으로 표현한다. 만약 컴포넌트의 종류가 Activity이고 패키지의 이름이 kr.ac.yonsei.mobilesw, 컴포넌트의 이름이 EditActivity라면 cmp=Activity kr.ac.yonsei.mobilesw/kr.ac.yonsei.mobilesw.EditActivity나 cmp=Activity kr.ac.yonsei.mobilesw/.EditActivity로 선언한다.
- COMPTYPE은 Intent 필드에는 포함되지 않지만, 옵션으로 컴포넌트의 네 가지 종류를 표현한다. Activity, Service, BroadcastReceiver, ContentProvider인 컴포넌트를 표현하는 옵션이지만 인텐트로 발생하는 취약점을 판단하는 방법의 자동화된 처리를 위해 선언하는 것은 권장한다.
- EXTRA는 컴포넌트가 요구하는 추가정보를 제공하며 인텐트 스펙에서 [key = Type, ...] 또는 key = Type value, ...]로 표현한다. 만약 Extra로 String 타입인 title과 content키의 값을 요구한다면 인텐트 스펙에서 [title = String, content = String]로 기술된다. 추가로 컴포넌트가 요구하는 특정한 값이 있다면 값을 포함해 [title = String “edit”, content = String “memo”]로 선언한다.

- FLAG는 안드로이드의 컴포넌트 스택을 조정하거나 안드로이드에서 접근이 제한된 영역의 권한을 부여하며 인텐트 스펙에서는 flg=[...]로 기술한다. 컴포넌트가 요구하는 flag가 FLAG_ACTIVITY_CLEAR_TOP라면 FLAG_ACTIVITY_CLEAR_TOP의 정의된 값인 67108864를 이용해 flg=[67108864]로 선언한다.
- 나머지 문법은 값을 선언하는 방법이다.

제 3 절 인텐트 스펙과 인텐트 필터 비교

이 절에서는 안드로이드 설정파일인 매니페스트에 인텐트를 선언하는 방법과 본 논문에서 제안한 인텐트 스펙을 비교해 유사점과 차이점에 대해서 살펴본다. 안드로이드 매니페스트 파일에는 컴포넌트가 요구하는 인텐트의 형태를 선언하는 방법으로 인텐트 필터를 제공한다. 인텐트 필터는 인텐트의 형태를 선언하는 방법이라는 점에서 이 논문에서 제안한 인텐트 스펙과 유사한 점이 있지만, 인텐트 필터는 Action과 Data의 타입, Category 필드만 선언하는 제한적인 방법이다.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.
           DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

그림 3.5: 인텐트 필터의 예

그림 3.5는 인텐트 필터로 `ShareActivity` 컴포넌트가 요구하는 인텐트의 형태를 선언한다. 이 예제를 간단히 설명하면 `ShareActivity` 컴포넌트는 인텐트로 `android.intent.action.SEND` Action을 요구하고 Category로 `android.intent.category.DEFAULT`를 요구한다. 원하는 Data의 타입은 `text/plain`이다. 이제 인텐트 스펙으로 `ShareActivity`가 기대하는 인텐트 형태를 선언해 유사점을 확인한다.

```
{ act = android.intent.action.SEND
  cat = android.intent.category.DEFAULT
  dat = non-null }
```

그림 3.6: 인텐트 스펙의 예

그림 3.6은 `ShareActivity` 컴포넌트가 요구하는 인텐트를 인텐트 스펙으로 선언한 예이다. 인텐트 필터와 비슷하게 Action을 `act`로 표현해 `act = android.intent.action.SEND`로 선언하고, Category는 `cat`로 표현해 `cat = android.intent.category.DEFAULT`로 선언한다. 인텐트 필터의 data는 Data 필드의 타입을 의미한다. 인텐트 필터에서 Data의 타입을 `typ`로 표현해 `typ = text/plain`으로 선언한다. 인텐트 필터와 인텐트 스펙의 유사점은 컴포넌트가 기대하는 인텐트의 형태를 선언할 수 있다는 점이다. 하지만 인텐트 필터에서 선언할 수 있는 필드는 7가지 중 Action과 Category, Data의 타입 3가지만 선언할 수 있고, 나머지 Extra 등과 같은 4가지 필드는 선언하는 방법을 제공하지 않아 인텐트의 세부적인 형태를 표현할 방법이 없다는 단점이 있다.

인텐트 스펙은 Action, Data, Category, Type, Component, Extra, Flag 7가지 필드를 모두 선언할 수 있다. 좀 더 확장하여 Data는 실제 데이터 위치를 선언할 수도 있고 non-null 키워드로 데이터가 존재함을 표현할 수도 있다. Extra의 경우에도 Extra에 포함된 키와 타입의 형태로 표현할 수 있고 실제 값이 포함된 키와 타입, 값의 형태로도 표현할 수 있다. 인텐트 스펙은 기존 인텐트 필터와 비교할 때 더 세밀하고 자유롭게 인텐트의 형태를 선언할 수 있는 장점이 있다.

제 4 장

인텐트 스펙 응용

이 장에서는 인텐트 스펙을 기반으로 응용하여 인텐트로 인해 발생하는 취약점을 개선하는 방법에 대해서 알아본다. 1절에서는 인텐트 스펙을 이용해 실행시간에 인텐트를 검사하고 방어하는 방법을 제안한다. 2절에서 인텐트 스펙을 기반으로 랜덤 생성한 인텐트로 유닛 테스팅 하는 방법을 제안한다. 3절에서 소스가 존재하지 않는 안드로이드 바이너리 파일로부터 어플리케이션이 요구하는 인텐트를 추출하고 인텐트 스펙을 자동으로 생성하는 방법을 제안한다. 4절에서 인텐트 스펙을 이용해 JUnit 테스트 케이스를 자동 생성하는 방법을 제안한다.

제안한 응용 방법을 구현한 예제를 다음 사이트에서 확인할 수 있다.

- 1절 : https://github.com/nzzzn/AssertOnIntent
- 2, 3, 4절 : http://mobilesw.yonsei.ac.kr/paper/intent_spec.html

제 1 절 인텐트 스펙 기반 실행시간 인텐트 검사

인텐트 스펙 기반 실행시간 인텐트 검사 방법은 컴포넌트에 전달된 인텐트를 실행시간에 검사하는 적극적인 취약점 방어 방법이다. 안드로이드 시스템은 컴포넌트가 요구하는 인텐트를 확인하지 않고 컴포넌트에 전달하는 문제로 어플리케이션이 비정상 종료되는 취약점이 발생하고 안드로이드 환경의 안전성이 떨어진다. 이러한 문제를 해결하기 위해서 컴포넌트가 요구하는 인텐트 형태를 인텐트 스펙으로 선언하고 실행시간에 전달된 인텐트와 상호 비교하여 잘못된 인텐트를 검출한다. 잘못된 인텐트는

예외가 발생되어 잘못된 인텐트를 정상적인 인텐트로 복구할 기회를 제공한다.

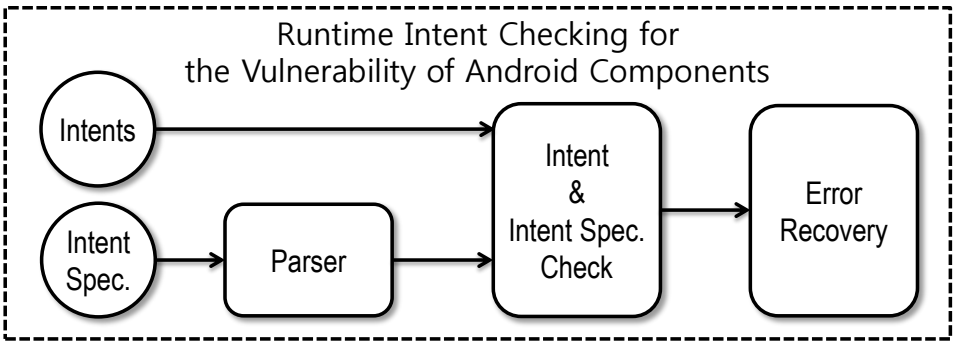

그림 4.1: 인텐트 스펙 기반 실행시간 검사 구조

그림 4.1은 인텐트 스펙 기반 실행시간 인텐트 검사의 구조이다. 컴포넌트는 호출되면서 인텐트를 전달받고, 미리 선언된 인텐트 스펙을 파싱해 컴포넌트가 요구하는 데이터 구조를 만든다. 전달된 인텐트와 컴포넌트가 요구하는 인텐트를 상호 비교해 정상 인텐트와 비정상 인텐트를 구분한다. 비정상으로 판단되는 인텐트가 전달되면 실행시간 검사 방법은 예외를 발생시켜 비정상 인텐트를 정상 인텐트로 복구할 기회를 제공한다. 이때, 비정상 인텐트를 복구할 수 없는 경우 최소 정상적으로 어플리케이션을 종료한다.

1.1 실행시간 검사 방법 적용

실행시간 검사 방법은 그림 4.2와 같이 `assertOnIntent(`검사할 인텐트, 인텐트 스펙, 핸들러`)`로 기술한다. `Java` 언어[2]에서 유효성을 검사하는 `assert`문을 아이디어를 확장한다. 실행시간 검사 방법을 기술하는 곳은 컴포넌트가 시작되는 `onCreate`와 전달된 인텐트를 사용하는 모든 지점이다.

그림 4.3은 인텐트 스펙 기반 실행시간 인텐트 검사방법을 적용한 액티비티 컴포넌트이다. 이 컴포넌트는 메모를 수정하고 추가하는 액티비티로 전달받은 인텐트의 Action과 Extra를 이용해 화면에 메모를 생성하거나 수정할 데이터를 출력한다. 그림 4.3에서 사용한 인텐트 스펙은 ||로 묶어진 두 가지 유형의 인텐트를 나타낸다. 이 인텐트의 Action은 `EDIT`이거나 `CREATE`이다. Action이 `EDIT`인 경우 Component는 `kr.ac.-`

```
new AssertOnIntent().assertOnIntent(
  //검사할 인텐트
  getIntent(),
  //인텐트 스펙
  "{...} || {...}",
  //핸들러
  new MalformedIntentHandler(){...});
```

그림 4.2: 인텐트 스펙 기반 실행시간 인텐트 검사 적용 예

yonsei.mobilesw/.Note이며, Extra로 title과 content에 값이 String 타입의 문자열이어야 한다. Action이 CREATE인 경우 Component가 kr.ac.yonsei.mobilesw/.Note이어야 한다. 만약 실행시간 인텐트 검사 방법에서 잘못된 인텐트를 검출하게 되면 MalformedIntentHandler에서 예외가 발생하고 에러 코드가 넘어온다. 이때, Extra가 잘못되어 109 코드가 넘어오면 인텐트에 Extra의 기본값을 넣어서 정상 인텐트로 복구한다. 이와 다르게 복구할 수 없는 다른 예외가 발생하면 정상적으로 어플리케이션을 종료한다.

1.2 인텐트 검사

인텐트 검사는 실제로 전달된 인텐트와 선언된 인텐트 스펙을 상호 비교하여 컴포넌트 요구에 만족하는 정상 인텐트와 컴포넌트 요구에 만족하지 않는 비정상 인텐트를 판단하여 잘못된 인텐트를 검출한다. 정상으로 판단하는 두 가지는 인텐트 스펙에 나열된 인텐트 형태 중 하나와 정확하게 일치하는 경우와 스펙에 선언된 것보다 더 많은 필드를 포함하여(예를 들어, Action이 EDIT인 경우 Extra에 date키와 값을 추가로 포함) 일종의 서브타이핑으로 간주 되는 경우이다. 비정상으로 판단하는 두 가지는 인텐트 스펙에 선언된 값과 타입이 다른 경우와 더 적은 필드를 가진 경우이다.

표 4.1은 잘못된 인텐트를 검출하면 발생하는 예외메시지이다. MalformedIntentException의 코드와 내용을 34개로 구분해 전달한다. 다음 절에서 개발자는 이 코드와 내용으로 해당 에러를 복구할 수 있는 방어 코드를 작성한다.

```
public class Edit extends Activity {
   String title, content;
   void onCreate(Bundle savedInstanceState){
      //추가된 영역 시작
      new AssertOnIntent().assertOnIntent(
        //검사할 인텐트
        getIntent(),
        //인텐트 스펙
        "{ act = android.intent.action.CREATE   //action
           cmp = kr.ac.yonsei.mobilesw/.Note }  //component
        ||
         { act = android.intent.action.EDIT      //action
           cmp = kr.ac.yonsei.mobilesw/.Note     //component
          [title = String, content = String] }", //extra
        //핸들러
        new MalformedIntentHandler(){
        public void handle
          (Intent intent, MalformedIntentException m)
          { switch(m.getNumber())
             {  case 109: //Extra miss matching
                   getIntent().putExtra(//Default Extra);
                   break;
                default: //finish; break;
          } } });
      //아래는 그림 2.3의 onCreate 메서드와 동일한 코드
      Intent intent = getIntent();
      ...
   }
   // Skip
}
```

그림 4.3: 제안한 방법을 추가한 Note 액티비티

표 4.1: MalformedIntentException 예외 메시지

코드	예외 메시지
0	Couldn't parse Intent.
1	Intent Key is not unique.
2	Duplicated Action field.
3	Duplicated Data field.
4	Duplicated Category field.
5	Duplicated Type field.
6	Duplicated Component field.
7	Duplicated Extra field.
8	Duplicated Flag field.
9	Couldn't parse Action field.
10	Couldn't parse Data field.
11	Couldn't parse Category field.
12	Couldn't parse Type field.
13	Couldn't parse Component field.
14	Couldn't parse Extra field.
15	Couldn't parse Flag field.
16	Couldn't parse after Action field.
17	Couldn't parse after Data field.
18	Couldn't parse after Category field.
19	Couldn't parse after Type field.
20	Couldn't parse after Component field.
21	Couldn't parse after Extra field.
22	Couldn't parse after Flag field.
100	Couldn't find Intent Key on described definition. : .... Actual Intent Key : ....
101	Couldn't match described Action : ....,Actual Action : ....
102	Couldn't match described Data : non-null. Actual Data : null.
103	Couldn't match described number of Category : ...., Actual number of Category : ....
104	Couldn't match described Category : ....,Actual Category : ....
105	Couldn't match described Type : non-null.,Actual Type : null.
106	Couldn't match described Component package : ...., Actual Component package : ....
107	Couldn't match described Component class : ...., Actual Component class : ....
108	Couldn't match described number of Extra : ...., Actual number of Extra : ....
109	Couldn't match described Extra : ....,Actual Extra : ....
110	Couldn't match described Flag : ....,Actual Flag : ....

1.3 잘못된 인텐트 핸들링

그림 4.4는 잘못된 인텐트로 판별된 인텐트를 MalformedIntentException의 코드를 이용해 핸들링하는 방법을 보인다.

```
//핸들러
new MalformedIntentHandler(){
public void handle
  (Intent intent, MalformedIntentException m)
  { switch(m.getNumber())
     {  case 101: //Action miss matching
           getIntent().setAction
           ("android.intent.action.CREATE");
        case 109: //Extra miss matching
           getIntent().putExtra(//Default Extra);
           break;
        default:
           finish;
           break;
  } } });
```

그림 4.4: 잘못된 인텐트 핸들링

Edit 액티비티 컴포넌트가 사용하는 인텐트의 필드는 Action과 Extra이다. 이 컴포넌트가 요구하는 Action이 아닌 경우와 Extra가 아닌 경우를 복구하는 예제이다.

• MalformedIntentException 코드 101은 전달된 인텐트에 Action이 컴포넌트가 요구하는 Action과 일치하지 않는 경우이다. 전달된 인텐트의 Action을 컴포넌트가 요구하는 기본 Action으로 변경하여 잘못된 인텐트를 복구한다.

• MalformedIntentException 코드 109는 전달된 인텐트에 컴포넌트가 요구한 Action은 정상적으로 존재하지만 추가적으로 필요한 Extra의 정보가 누락된 경우를 나타낸다. 전달된 인텐트의 Extra를 컴포넌트가 요구하는 기본 Extra로 변경하여 잘못된 인텐트를 복구한다.

• 나머지는 복구할 수 없거나 예측할 수 없는 경우이다. 이 경우 default로 컴포넌트를 정상적으로 종료시키는 코드를 이용해 취약점을 방어한다.

1.4 실험결과

인텐트 스펙 기반 실행시간 인텐트 검사 방법을 안드로이드 4.4.2버전의 LG-F220K 폰 환경에서 실험하고 결과를 표로 나타낸다. 실험한 앱 중 `BluetoothChat`과 `NotesList`는 안드로이드 SDK에 포함된 샘플이고, `Cafe`는 학생이 제작한 앱이다. 그 외 나머지 앱은 안드로이드 프로그래밍 서적에서 참조하였다. 실험은 잘못된 인텐트로 발생하는 취약점을 파악하고 제안한 방법으로 방어한 결과를 나타낸다. 마지막으로 제안한 방법을 적용해 발생하는 오버헤드를 측정한다.

표 4.2: 실행시간 인텐트 검사 실험 결과

앱	소스코드에서 검사하지 않는 인텐트 정보	잘못된 인텐트 정보를 검출한 수	비정상 종료 오류의 수	비정상 종료 방어의 수
AndroidSecurity	1	1(100%)	0	0(100%)
BluetoothChat	4	4(100%)	1	1(100%)
Cafe	1	1(100%)	0	0(100%)
ContactsActivity	1	1(100%)	0	0(100%)
NotesList	17	15(88.2%)	6	4(66.7%)
Media_Player	4	4(100%)	0	0(100%)
Earthquake	2	2(100%)	0	0(100%)
합 계	30	28(93.3%)	7	5(71.4%)

표 4.2는 인텐트 스펙 기반 실행시간 인텐트 검사를 오픈 소스 어플리케이션에 적용해 실험한 결과이다. 첫 번째 열은 실험한 안드로이드 앱의 이름이고, 두 번째 열은 각 앱에서 인텐트의 잘못된 정보를 검사하지 않고 사용하는 수이다. 세 번째 열은 제안한 실행시간 인텐트 검사 방법으로 인텐트의 잘못된 정보를 검출한 수이다. 네 번째 열은 검사하지 않는 전체 인텐트 정보에서 비정상 종료된 오류의 수를 의미한다. 마지막으로 다섯 번째 열은 제안한 실행시간 인텐트 검사 방법으로 비정상 종료를 방어한 수이다.

두 번째 열의 소스코드에서 검사하지 않는 인텐트 정보는 잘못된 인텐트를 어플리케이션에 전달하여 비정상 종료되거나 검포넌트가 잘못된 정보를 검사하지 않고 사용한 수를 의미한다. 모든 앱에서 잘못된 인텐트 정보를 검사하지 않아 취약점이 발생할 가능성이 있음을 확인했다. 잘못된 인텐트로 7개의 어플리케이션에서 총 30개의 취약

점을 발생할 수 있다.

세 번째 열은 제안한 실행시간 인텐트 검사 방법으로 검출한 잘못된 인텐트의 수를 의미한다. 두 번째 열에서 사용한 잘못된 인텐트를 제안한 방법을 적용한 어플리케이션에 전달하여 측정한 값이다. 어플리케이션에 잘못된 인텐트가 전달되면 실행시간에 인텐트를 검사하여 잘못된 인텐트를 검출하고 예외를 발생한 경우를 의미한다. 이때, 개발자는 잘못된 인텐트 핸들링을 통해서 정상적인 인텐트로 복구시키거나 정상적으로 프로그램을 종료시킬 기회를 가진다. 총 30개의 잘못된 인텐트 중 28(93.3%)개를 검출할 수 있었다. 검출하지 못한 3개의 인텐트는 Data 필드의 값이 올바르지 않은 위치를 가르치는 경우이다. 제안한 실행시간 검사에서는 Data의 위치가 기술되어 있는지만 판단하여 검출하지 못했지만, 추후에 패턴 매칭이나 실제로 접근 가능한 데이터인지 확인하는 방법을 적용해 개선할 수 있다.

네 번째 열은 검사하지 않고 사용하는 인텐트의 정보 수에서 어플리케이션 잘못된 인텐트로 인해 비정상 종료된 오류의 수를 의미한다. BluetoothChat와 NotesList 두 가지 어플리케이션에서 비정상 종료의 취약점을 검출할 수 있었다. 실험에서 사용한 BluetoothChat의 경우에는 블루투스 장치를 찾았다는 인텐트를 전달하게 되면 전달된 인텐트 정보를 이용해 장치에 접근하기 위한 코드에서 NullPointerException 에러가 발생함을 확인할 수 있었다. 실험에 사용한 NotesList의 경우는 Data 필드가 누락되거나 잘못된 위치를 가르칠 경우 비정상 종료가 됨을 알 수 있었다. Data 필드를 누락시킨 잘못된 인텐트를 전달한 결과 NullPointerException 에러가 발생하여 비정상 종료되었고, Data 필드가 잘못된 데이터의 위치를 가르치는 경우 IllegalArgumentException 에러가 발생하여 어플리케이션이 종료되었다.

다섯 번째 열에서는 비정상 종료되는 오류를 제안한 실행시간 인텐트 검사 방법으로 방어한 수 나타낸다. BluetoothChat과 NotesList 두 가지 어플리케이션에서 모두 비정상 종료를 방어할 수 있었지만 NotesList에서 발생한 에러 중 Data 필드에 잘못된 위치가 기술되어 IllegalArgumentException 에러가 발생하는 형태의 잘못된 인텐트는 해당 위치의 데이터에 존재 여부를 판단하지 못해 방어하지 못했다. 하지만 방어하지 못한 취약점도 접근 가능 여부를 확인하는 방법을 적용해 추후에 해결할 수 있다. 비정

상 종료되는 총 7개의 취약점 중 5(71.4%)개의 취약점을 방어할 수 있었다.

표 4.3: 비정상 종료 에러 수

앱	NullPointer Exception	IllegalArgument Exception
BluetoothChat	1	0
NotesList	4	2
합 계	5	2

표 4.3은 실험에서 발생한 비정상 종료 에러를 나타낸다. BluetoothChat의 경우 잘못된 인텐트로 인해 NullPointerException이 발생했고, NotesList의 경우에는 NullPointerException 4개와 IllegalArgumentException 2개가 발생했다. NullPointerException은 제안한 방법으로 모두 방어할 수 있었지만 IllegalArgumentException 2개의 경우에는 데이터의 존재 여부를 판단하지 못해 방어할 수 없었다.

실험한 표 4.2와 표 4.3을 요약하면, 전체 7개의 어플리케이션에서 모두 인텐트를 검사하지 않고 사용해 취약점이 생길 가능성이 존재했고 2개의 어플리케이션에서는 비정상 종료되는 취약점을 확인했다. 제안한 인텐트 스펙 검사방법으로 잘못된 인텐트 정보를 확인해 30개 중 28(93.3%)개를 방어할 수 있었고 비정상 종료되는 인텐트 7개 중 5(71.4%)개를 방어해 제안한 방법이 유용함을 보였다.

표 4.4: 실행시간 검사 적용에 따른 오버헤드

앱	용량(추가된 byte)	시간(추가된 ms)	추가된 LOC
AndroidSecurity	321,704(+9,099)	421(+10)	2
BluetoothChat	27,545(+10,811)	450(+10)	4
Cafe	1,473,874(+10,057)	431(+9)	3
ContactsActivity	36,116(+10,616)	440(+10)	2
NotesList	60,405(+11,038)	421(+10)	10
Media_Player	154,282(+11,120)	450(+10)	8
Earthquake	51,117(+10,335)	431(+9)	2
평 균	303,577(+10,439)	434(+9.7)	4.4

표 4.4는 어플리케이션에 제안한 실행시간 검사 방법을 적용하여 발생하는 오버헤드를 나타낸다. 첫 번째 열은 앱 이름이고, 두 번째 열은 안드로이드 바이너리 파일의 크기이다. 두 번째 열의 괄호는 제안한 방법을 적용하여 추가된 용량을 나타낸다. 세

번째 열은 어플리케이션을 호출하고 실행되는 데 걸리는 시간인 스타트업 시간이다. 세 번째 열의 괄호는 제안한 방법을 적용하여 발생하는 추가 시간을 의미한다. 네 번째 열은 제안한 방법을 적용하기 위해 추가 작성해야 하는 라인의 수를 의미한다.

두 번째 열은 안드로이드 바이너리 파일의 크기를 나타내고 괄호로 증가한 바이너리 크기를 나타낸다. 어플리케이션에 제안한 실행시간 인텐트 검사방법을 적용하기 위해 추가되는 인텐트 스펙 파서 코드와 잘못된 인텐트를 검사하는 코드, 핸들링하는 코드로 인해 각 앱이 평균 10,439(3.4%)byte가 증가함을 확인할 수 있다. 앱 크기가 작은 앱인 BluetoothChat의 경우에는 10,811byte가 증가하여 39.2%가 증가되기는 했지만 제안한 방법을 적용한 전체 용량은 37Kbyte로 큰 크기는 아니다.

세 번째 열은 어플리케이션을 호출하고 시작되기까지의 시간을 측정한 스타트업 시간을 나타내고, 괄호로 제안한 방법을 적용해 증가한 시간을 나타낸다. 어플리케이션이 시작되면서 선언된 인텐트 스펙을 파싱하고 전달된 인텐트와 검사하는 과정이 추가되어 각 앱이 평균 9.7(2.2%)ms의 시간이 증가한다. 약간의 스타트업 시간이 증가하기는 했지만 디바이스 사용성을 해칠 정도의 큰 시간은 아니다.

네 번째 열은 제안한 실행시간 인텐트 스펙 검사방법을 어플리케이션에 적용하기 위해 추가 작성하는 코드의 라인 수이다. 인텐트를 사용하는 부분이 많을수록, 사용하는 형태가 다양할수록 추가하는 코드의 수는 커지게 된다. 라인 수가 가장 많이 증가한 NotesList의 경우 4개의 액티비티에서 인텐트를 사용하여 10줄의 라인이 증가한다. 각 어플리케이션에 증가하는 평균 라인 수는 4.4줄 정도로 많지 않다.

표 4.4를 요약하면, 제안한 실행시간 인텐트 검사방법을 어플리케이션에 적용하여 발생하는 평균 오버헤드는 용량이 약 10,439(3.4%)byte로 크지 않고, 스타트업 시간은 9.7(2.2%)ms이다. 어플리케이션에 적용하여 증가하는 라인의 수는 어플리케이션이 사용하는 인텐트의 수와 형태에 따라 달라지긴 하지만 평균 4.4줄로 크지 않다. 따라서 실행시간 인텐트 검사방법을 적용해 발생하는 오버헤드는 크지 않아 사용성을 해칠만한 수준의 증가는 아니다.

제 2 절 인텐트 스펙 기반 안드로이드 유닛 테스팅

인텐트 스펙 기반 안드로이드 유닛 테스팅은 선언된 인텐트 스펙을 기반으로 랜덤 생성한 인텐트를 컴포넌트에 전달하여 취약점을 자동으로 판단하는 방법이다. 인텐트 스펙을 기반으로 랜덤 유닛 테스트할 테스트 케이스를 생성하는 것의 장점은 취약점이 자주 발생하는 시나리오나 개발자가 원하는 모든 형태의 테스트 케이스를 쉽고 자유롭게 작성할 수 있다는 것이다.

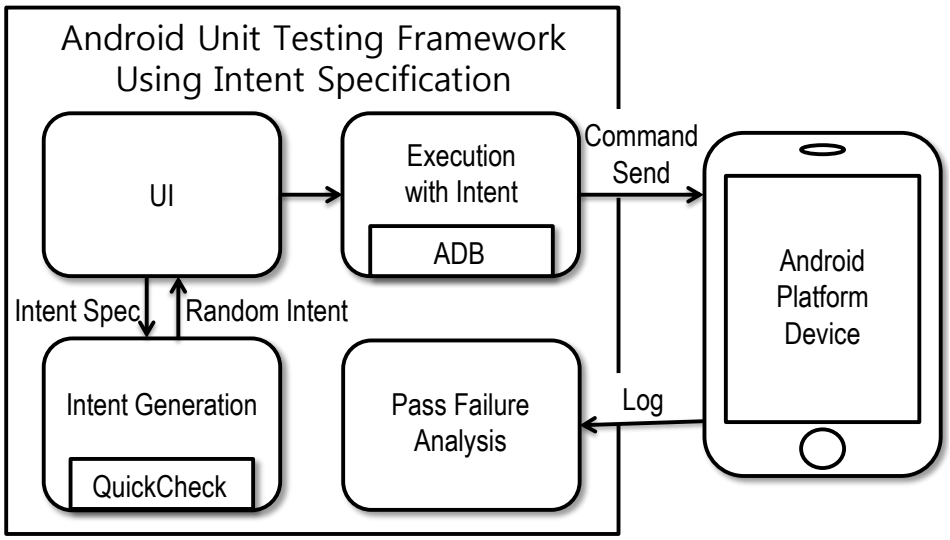

그림 4.5: 인텐트 스펙 기반 안드로이드 유닛 테스팅

그림 4.5은 인텐트 스펙 기반 안드로이드 유닛 테스팅의 구조이다. UI에서 인텐트 스펙을 사용자에게 입력받는다. 인텐트 스펙을 기반으로 데이터구조에 따라 데이터를 랜덤 생성해주는 QuickCheck[10]를 이용해 랜덤 데이터를 생성한다. 랜덤 생성한 인텐트를 PC와 안드로이드 디바이스를 연결하는 ADB[9]를 이용해 디바이스에서 실행한다. 디바이스에서 발생하는 로그를 수집하여 정상실행과 비정상실행을 판단한다. 디바이스로 전달한 인텐트와 인텐트로 발생한 로그, 판단 결과를 엑셀파일 형태로 제공해 추후에 테스터가 오프라인에서 취약점을 확인하는 방법을 제공한다.

그림 4.6은 실제 구현된 인텐트 스펙 기반 안드로이드 유닛 테스팅의 화면이다. 디바이스를 선택하고 생성된 인텐트를 전달한다. 인텐트를 전달받은 컴포넌트 실행되며

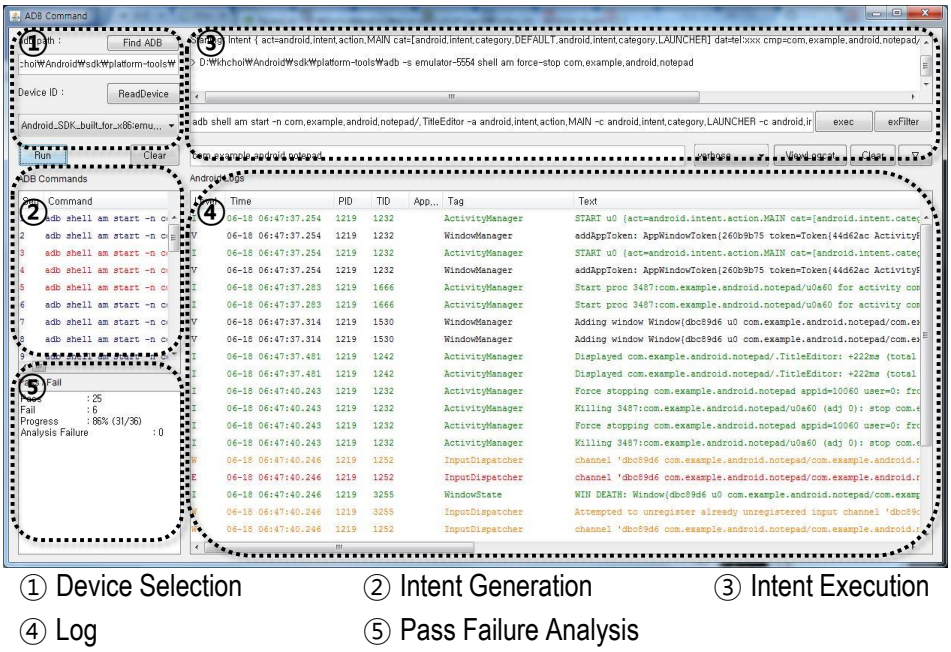

그림 4.6: 인텐트 스펙 기반 안드로이드 유닛 테스팅 UI

디바이스에서 발생한 로그와 정상 실행과 비정상 실행을 판단한 결과를 실시간으로 확인할 수 있다.

2.1 인텐트 스펙 기반 랜덤 인텐트 생성

인텐트 스펙 기반 랜덤 인텐트 생성은 인텐트 스펙으로 다양한 인텐트의 형태를 자유롭게 선언할 수 있는 장점을 이용해 선언된 인텐트 스펙을 기반으로 랜덤 인텐트를 생성한다. 기존 연구들은 인텐트를 생성하는 방법이 제한적이어서 다양한 인텐트를 이용해 테스트하기 위해서는 테스트 프로그램 자체를 수정해야 한다.

```
{ act = android.intent.action.EDIT
[ title = String "new", content = String "content"] }
|| { act = android.intent.action.CREATE }
```

그림 4.7: 컴포넌트가 요구하는 인텐트

그림 4.7는 `Edit` 컴포넌트가 요구하는 인텐트 스펙이다. `Edit` 컴포넌트는 `EDIT`

와 CREATE Action을 요구한다. Action이 EDIT인 경우 Extra의 정보를 추가로 필요로 한다. 그림 4.7과 같은 선언된 인텐트 스펙을 기반으로 다양한 형태의 랜덤 인텐트를 생성할 수 있다.

선언된 인텐트 스펙을 기반으로 랜덤 생성하는 인텐트 모드는 세 가지로 Compatible, Shape-Compatible, Random 이다.

- Compatible : 인텐트 스펙의 필드와 값에 맞춰 생성하되 새로운 필드를 추가한다.
- Shape-Compatible : 인텐트 스펙의 모든 필드는 갖추되 필드 값은 랜덤으로 생성한다. 그리고 새로운 필드도 추가한다.
- Random : 인텐트 스펙과 무관하게 랜덤 생성한 인텐트다.

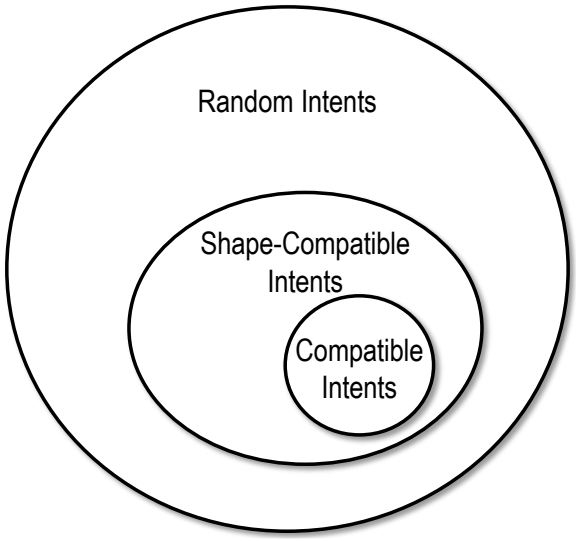

그림 4.8: 인텐트 형태 집합

그림 4.8은 각 모드로 랜덤 생성하는 인텐트의 형태를 집합으로 나타낸다. Compatible 모드는 인텐트 스펙 구조를 그대로 유지하고 새로운 필드만 추가하므로 적은 범위에서 랜덤 생성한다. Shape-Compatible 모드는 구조를 포함하고 값은 랜덤으로 생성하고 새로운 필드로 추가하므로 중간 정도의 범위에서 랜덤 생성한다. Random

모드는 테스트 대상만 지정하고 나머지 필드는 모두 랜덤으로 생성하므로 가장 큰 범위에서 랜덤 생성한다.

- Intent Spec
```
{ act = android.intent.action.EDIT
[ title = String "new", content = String "content"] }
```

- Compatible
```
{ act = android.intent.action.EDIT
[ title = String "new", content = String "content"]
cat = [efXIjn_, TkWShwrDxJrgT] dat = tel:123 }
```

- Shape-Compatible
```
{ act = android.intent.action.DELETE
[ kcIEKke_wek = String "sdkSDFvn238"]
dat = sdfklj324dsf:sdfji23iod }
```

- Random
```
{ act = SDkfiwskWE41sSDF dat = tel:123 }
```

그림 4.9: 각 모드로 생성한 인텐트 스펙의 예

그림 4.9은 각 랜덤 모드로 생성한 인텐트 스펙을 보인다. Compatible 모드에서 생성된 인텐트의 경우 값과 구조의 변경없이 필드가 추가된다. Shape-Compatible 모드에서 생성된 인텐트는 구조는 변경되지 않았지만, 값이 변경되고 데이터가 추가된다. Random 모드에서 생성된 인텐트는 기존 인텐트 스펙과 무관하게 모든 데이터를 랜덤으로 만든다.

인텐트를 디바이스에 전달 시 ADB(Android Debug Bridge)[9] 명령어 형태를 이용한다. ADB는 PC와 디바이스를 연결할 수 있는 안드로이드 시스템의 유틸리티이다. 테스트 시 생성되는 실제 데이터는 ADB 명령어 형태로 변환되어 전송한다. 그림 4.10는 그림 4.7를 기반으로 각 모드로 생성한 ADB 기반의 인텐트의 모양이다. 랜덤 인텐트를 생성하는 자세한 방법은 부록을 참고한다.

• Compatible

```
adb shell am start -a android.intent.action.EDIT -n kr.ac.
   yonsei.mobilesw/.Edit --es title "new" --es content "
   content"
adb shell am start -a android.intent.action.EDIT -n kr.ac.
   yonsei.mobilesw/.Edit --es title "new" --es content "
   content" -c efXIjn_ -c TkWShwrDxJrgT
adb shell am start -a android.intent.action.EDIT -n kr.ac.
   yonsei.mobilesw/.Edit --es title "new" --es content "
   content" -c cQJ -c WbPlLfMWcbLOyv -c wJ -c iZtMThgjo -c
   LJv -c krLbesebdvOsk -c EFWX -c HkzLUrNEHG -c Wa -c
   wRXAKMiEdosRtH -c fillbuV
```

• Shape-Compatible

```
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -a com.
   android.action.Edit --efa x 935.0,-805.0,-981.0  --eu
   QfJOlXGLrc host -d rfRN
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -t Bwz/rk
   -c android.intent.category.APP_CALENDAR --ei KDLCimrAm 12
    --ei 4i5tbTwHlxH9SDB -12 --eu Lc96XqWot BMsNjUXYc9sitN
   --ef JU2WOu -75.79375 --ela gCX5IBidnUz
   -71,-275,406,901,78,352,384,-293
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -a android
   .intent.action.EDIT --es title "new" --es content "
   content"
```

• Random

```
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -d 4K -c
   KZSYPUX -c NofAVDa_.laYqek -c vsvsYGeEWso -c
   XSJOFnvrsBTjsSf
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -d tel:123
    -a .JkFOurhT_vOeP
adb shell am start -n kr.ac.yonsei.mobilesw/.Edit -d content
   ://com.google.provider.NotePad/notes
```

그림 4.10: 각 모드로 생성한 ADB 명령어 형태의 인텐트

2.2 유닛 테스트 실행

생성한 다양한 인텐트를 디바이스에 전달해 테스트한다. PC에서 랜덤으로 생성한 인텐트를 디바이스에 전달하기 위해 ADB[9]를 이용한다. ADB는 안드로이드 SDK로 PC에서 안드로이드 플랫폼 디바이스에 인텐트를 전달할 수 있는 유틸리티이다.

테스트는 생성된 인텐트를 순차적으로 실행한다. 이때 이전 결과가 다음 실행에 영향을 받지 않도록하기 위해 인텐트당 총 7초의 시간을 부여한다. 인텐트 전달 후 어플리케이션이 시작되도록 5초의 여유시간 후, 어플리케이션을 강제로 종료시키고 2초의 종료 여유시간을 부여한다.

ADB로 전달할 수 있는 Extra의 타입은 null, String, Boolean, Int, Long, Float, URI, Component, Int[], Long[], Float[] 제한된다. ADB를 이용하는 유닛 테스트 방법은 Extra 타입이 제한되는 단점이 있다. 이 단점은 추후 4절에서 설명하는 JUnit 테스트로 해결할 수 있다.

2.3 로그 수집과 테스트 결과 판단

유닛 테스트를 진행하면서 디바이스에서 발생하는 로그를 수집하고 이 로그를 자동으로 분석해 테스트의 결과를 알린다. 수집한 로그는 테스트 대상이 되는 컴포넌트의 패키지를 추적하여 분류한다. 분류된 로그를 실시간으로 분석해 정상 실행의 형식으로 로그가 출력되면 정상 실행, 강제 종료된 형식으로 로그가 출력되면 비정상 종료로 구분하여 자동 판단한다.

- Activity
  - 정상 실행 : Displayed, Finishing, Force removing
  - 비정상 종료 : Force finishing
- BroadcastReceiver
  - 정상 실행 : none
  - 비정상 종료 : has died

- Service
  - 정상 실행 : VM exiting with result code
  - 비정상 종료 : Shutting down VM

로그를 판단하는 기준은 특정 로그의 패턴이 수집된 로그에서 발생하는지 확인해 판단한다. 액티비티에서 발생한 로그 중에서 Displayed, Finishing, Force removing이 포함된 로그가 발생하면 정상 실행으로 판단한다. Force finishing이 포함된 로그가 발생하면 비정상 종료로 판단한다. 브로드캐스트 리시버에서 발생한 로그 중에서 has died가 포함된 로그가 발생하면 비정상 종료로 판단하고 has died가 포함된 로그가 발생하지 않으면 정상 실행으로 판단한다. 서비스에서 발생한 로그 중에서 VM exiting with result code가 포함된 로그가 발생하면 정상 실행으로 판단하고 Shutting down VM이 포함된 로그가 발생하면 비정상 종료로 판단한다. 로그를 이용해서 결과를 판단하는 자세한 방법은 부록을 참고한다.

그림 4.11은 테스트를 마치고 저장된 Excel 파일의 화면이다. 자동화된 테스트를 마치면 디바이스로 전달한 인텐트 메시지와 대상 컴포넌트의 로그, 판단 기준이 된 로그, 최종 결과를 Excel 형식의 파일로 저장한다. 테스터는 테스트 종료 후 취약점으로 비정상 종료된 어플리케이션의 로그를 오프라인에서 확인하여 취약점을 파악할 수 있다.

2.4 실험결과

제안한 방법을 안드로이드 4.4.2버전의 LG-F220K폰 환경에서 실험하고 결과를 표로 나타낸다. 실험한 앱 중 `BluetoothChat`과 `NotesList`는 안드로이드 SDK에 포함된 샘플이고, `Cafe`는 학생이 제작한 앱이다. 그 외 나머지 앱은 안드로이드 프로그래밍 서적에서 참조하였다. 실험은 유닛 테스트로 취약점을 파악하고 어떤 유형의 에러들이 발생하는지 살펴본다. 마지막 실험에서 컴포넌트가 요구하는 인텐트와 구조가 같아 정상 인텐트로 판단되는 인텐트와 구조가 달라 비정상 인텐트로 판단되는 인텐트를 이용해 정상 실행과 비정상 종료되는 경우를 살펴본다.

표 4.5는 인텐트 스펙 기반 안드로이드 유닛 테스팅을 오픈 소스 어플리케이션으로

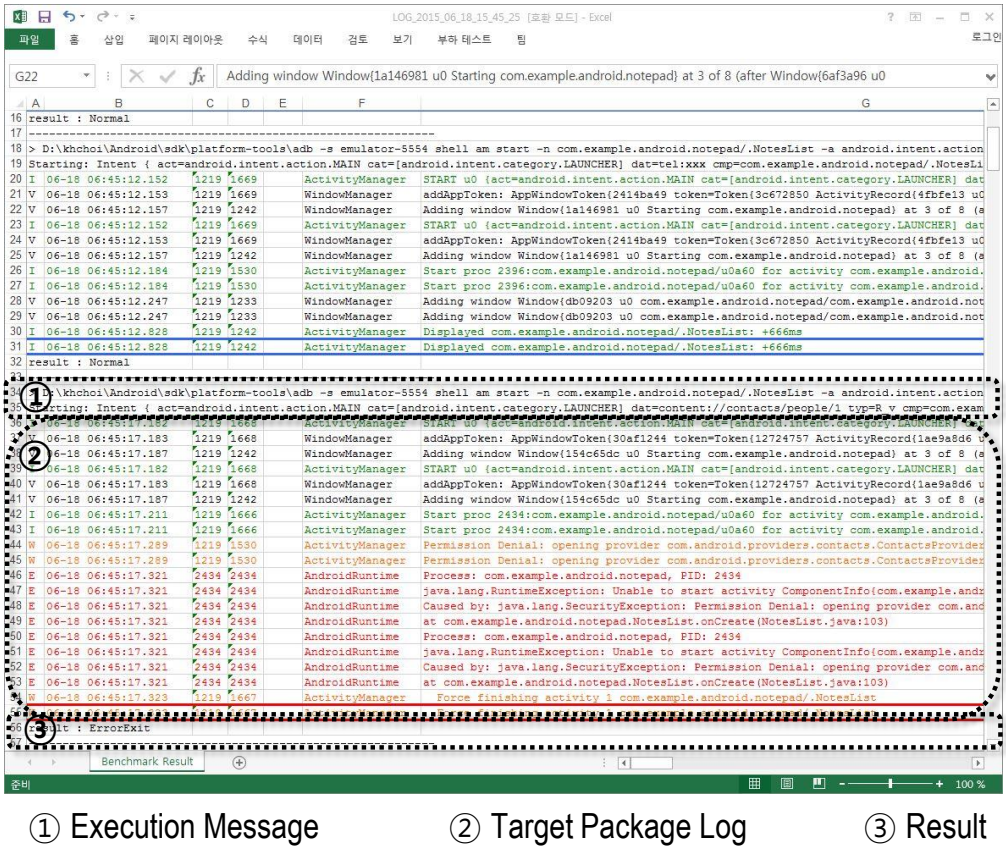

그림 4.11: 출력된 결과 Excel 파일

테스트한 실험 결과이다. 첫 번째 열은 실험한 안드로이드 앱의 이름이고, 두 번째 열은 각 어플리케이션이 요구하는 인텐트를 Action으로 구분해서 나타낸 수이다. 세 번째 열은 요구하는 인텐트 수로 랜덤 생성한 인텐트의 수이고 랜덤 생성 시 Compatible, Shape-Compatible, Random 모드로 25개씩 생성했다. 생성시 랜덤으로 생성한 인텐트의 개수가 많을수록 더 많은 취약점을 발견할 수 있지만 테스트하는 시간이 증가하기 때문에 실험에서는 취약점을 판단할 수 있는 정도로 25개씩 생성했다. 네 번째 열은 정상 실행된 인텐트의 수이고, 다섯 번째 열은 비정상 종료된 인텐트의 수이다.

두 번째 열의 요구하는 인텐트 수는 각 어플리케이션이 요구하는 인텐트의 형태를 분석해서 파악했다. 실험에 사용한 어플리케이션은 대부분 간단한 어플리케이션으로 사용하는 인텐트의 수가 많지 않다.

표 4.5: 유닛 테스팅 실험결과

앱	Action으로 구분한 인텐트 수	테스트한 랜덤 인텐트 수	정상 실행	비정상 종료
AndroidSecurity	1	75	75(100%)	0(0%)
BluetoothChat	1	75	75(100%)	0(0%)
Cafe	1	75	75(100%)	0(0%)
ContantsActivity	1	75	75(100%)	0(0%)
NotesList	6	450	427(94.9%)	23(5.1%)
Media_Player	6	450	436(96.9%)	14(3.1%)
Earthquake	4	300	300(100%)	0(0%)
합 계	20	1,500	1,463(97.5%)	37(2.5%)

세 번째 열의 테스트한 랜덤 인텐트 수는 각 어플리케이션에서 요구하는 인텐트를 이용해 인텐트 스펙 기반으로 랜덤 생성했다. 생성 시 인텐트에 랜덤 생성한 새로운 필드를 추가하는 Compatible 모드로 25개, 요구하는 인텐트에서 랜덤 생성 값과 새로운 필드를 추가하는 Shape-Compatible 모드로 25개, 모든 필드와 값을 랜덤하게 생성하는 Random 모드로 25개씩 생성했다. 요구하는 인텐트 1개당 75개의 랜덤 인텐트를 생성하고 생성된 랜덤 인텐트를 이용해 제안한 유닛 테스팅을 진행한다.

네 번째 열은 랜덤 생성한 인텐트를 어플리케이션에 전달해 정상 실행된 수를 의미한다. 랜덤 생성한 총 1500개의 인텐트로 테스트한 결과 1463(97.5%)개의 인텐트가 정상적으로 실행됐다. 랜덤 생성한 인텐트가 모두 정상 실행된 어플리케이션은 7개 중 5(71.4%)개이다.

다섯 번째 열은 랜덤 생성한 인텐트를 어플리케이션에 전달해 비정상 종료된 수를 의미한다. 랜덤 생성한 총 1500개의 인텐트에서 비정상 종료된 인텐트는 37(2.5%)이다. 제안한 유닛 테스팅 방법으로 실험한 결과 NotesList와 Media_Player 2개의 어플리케이션에서 비정상 종료되는 취약점을 발견했다. 실험에서 NotesList에서 발생한 취약점은 누락된 데이터로 발생하는 NullPointerException 에러가 5개, 허가되지 않은 데이터에 접근하여 발생하는 SecurityException 에러가 5개, 잘못된 URL 데이터에 접근하여 발생하는 IllegalArgumentException 에러가 13개이다. 실험에서 Media_Player은 누락된 데이터로 발생하는 NullPointerException 에러만 14개 발생했다.

표 4.6: 비정상 종료 에러

앱	NullPointer Exception	Security Exception	IllegalArgument Exception
NotesList	5	5	13
Media_Player	14	0	0
합 계	19	5	13

표 4.6은 랜덤 생성한 인텐트로 비정상 종료된 에러의 수를 나타낸다. 실험에서 비정상 종료된 NotesList와 Media_Player는 모두 NullPointerException 에러가 발생하고, 발생한 개수가 19(51.4%)개로 가장 높은 비율을 차지한다. NullPointerException 에러는 전달된 인텐트에서 누락된 정보가 존재하면 발생하기 때문에 비정상 종료된 많은 어플리케이션에서 확인되는 취약점이다. NotesList에서 5(13.5%)개가 발생한 SecurityException 에러는 권한이 없는 데이터에 접근하는 경우 발생한다. 예를 들어, NotesList는 전화번호에 접근할 수 있는 권한이 없어 전화번호 데이터를 인텐트로 전달받아 접근을 시도하면 발생한다. 마지막으로 NotesList에서 5(13.5%)개가 발생한 IllegalArgumentException 에러는 잘못된 데이터의 위치를 전달받아 접근을 시도하면 발생한다.

표 4.5와 표 4.6을 요약하면, 제안한 유닛 테스팅 방법으로 전체 7개의 어플리케이션 중 2(28.5%)개의 어플리케이션에서 취약점을 파악할 수 있었다. 랜덤 생성한 1500개의 인텐트로 제안한 유닛 테스팅 방법을 이용해 실험한 결과 1463(97.5%)개의 인텐트가 정상 실행되고 37(2.5%)개의 인텐트에서 취약점이 발생하여 NullPointerException, SecurityException, IllegalArgumentException 에러가 발생했다. 이 실험을 통해서 제안한 인텐트 스펙 기반 안드로이드 유닛 테스팅 방법으로 인텐트로 발생하는 취약점을 파악할 수 있음을 확인했다.

1절에서 제안한 실행시간 인텐트 검사 방법 표 4.2의 실험결과와 이 절에서 제안한 유닛 테스팅 방법 표 4.5의 실험결과를 비교해보면 BluetoothChat과 Media_Player이 서로 다른 결과로 나타난다. BluetoothChat의 경우 잘못된 인텐트를 전달하고 버튼을 눌러 이벤트가 발생해야 취약점을 확인할 수 있지만, 유닛 테스팅의 경우 인텐트를

전달해 발생하는 취약점으로만 판단하므로 유닛 테스팅 방법으로는 파악되지 않았다. Media_Player의 경우는 1절에서 소스를 분석해 작성한 잘못된 인텐트로는 취약점을 파악할 수 없었지만 이 절에서 제안한 방법인 유닛 테스팅 방법으로 취약점을 추가 파악할 수 있었다.

표 4.7: 유닛 테스트 추가 분석 실험결과

앱	인텐트 스펙에 만족하는 인텐트		인텐트 스펙에 만족하지 않는 인텐트	
	정상 실행	비정상 종료	정상 실행	비정상 종료
AndroidSecurity	33	0	42	0
BluetoothChat	35	0	40	0
Cafe	32	0	43	0
ContantsActivity	37	0	38	0
NotesList	193	13	234	10
Media_Player	135	0	301	14
Earthquake	211	0	89	0
합계	676	13	787	24

표 4.7은 1절에서 설명한 인텐트 스펙 기반 실행시간 인텐트 검사 방법과 이 절에서 제안하는 인텐트 스펙 기반 안드로이드 유닛 테스팅 방법을 동시에 적용한 실험이다. 실행시간 인텐트 검사 방법은 컴포넌트가 요구하는 인텐트와 전달된 인텐트를 검사하여 컴포넌트가 요구하는 인텐트와 요구하지 않는 인텐트를 알리는 기능을 제공한다. 이 방법을 이용해 인텐트 스펙에 만족하는 인텐트와 인텐트 스펙에 만족하지 않는 인텐트로 구분하고 구분한 각 인텐트로 정상 실행되는 경우와 비정상 종료되는 경우를 파악했다. 만족하는 인텐트로 판단된 경우는 컴포넌트가 요구하는 인텐트와 일치하는 인텐트이거나 더 많은 정보를 담고 있는 인텐트이다. 만족하지 않는 구조의 인텐트는 컴포넌트가 요구하는 인텐트와 일치하지 않거나 더 적은 정보를 담고 있는 인텐트이다.

두 번째 열에서 인텐트 스펙에 만족하는 인텐트는 실행시간 인텐트 검사 방법을 통과한 인텐트이다. 실험한 NotesList는 인텐트 스펙을 통과하고 NullPointerException 에러 3개, SecurityException 에러 5개, IllegalArgumentException 에러 5개가 발생해 비정상 종료되는 경우가 존재한다. 이 에러들은 NotesList에서 Data 필드를 사용하며 발생한다. 예를 들어, Data 필드에 e4S/2bdzIT9와 같은 값을 담은 인텐트를 NotesList에

전달하면 e4S/2bdzIT9의 위치에는 데이터가 존재하지 않으므로 NullPointerException 에러가 발생한다. Data필드에 content://contacts/people/1과 같은 값을 담은 인텐트를 NotesList에 전달하면 접근 권한이 없는 NotesList는 content://contacts/people/1로 접근해 SecurityException 에러가 발생한다. 마지막으로 Data 필드에 tel:123과 같은 값을 담을 인텐트를 NotesList에 전달하면 잘못된 URL로 인식해 IllegalArgumentException 에러를 발생시킨다.

세 번째 열에서 인텐트 스펙에 만족하지 않는 인텐트는 실행시간 인텐트 검사 방법을 통과하지 못한 인텐트이다. 인텐트 스펙에 만족하지 않는 인텐트들에서 발생한 취약점은 1절에서 제안한 인텐트 스펙 기반 실행시간 인텐트 검사 방법을 적용하면 모두 검출하여 방어할 수 있는 인텐트이다.

실험결과, 제안한 인텐트 스펙 기반 안드로이드 유닛 테스팅 방법으로 취약점을 쉽게 파악할 수 있어 유용함을 보였다. 또한, 1절에서 제안한 인텐트 스펙 기반 실행시간 검사 방법을 추가로 적용해 취약점을 좀 더 세부적으로 파악할 수 있음을 보였다.

제 3 절 인텐트 스펙 자동 생성 테스팅 방법

앞서 제안한 1절의 인텐트 스펙 기반 실행시간 인텐트 검사 방법과 2절의 인텐트 스펙 기반 안드로이드 유닛 테스팅 방법은 테스트할 어플리케이션의 소스코드가 있는 상태에서 인텐트로 인한 취약점을 개선하는 방법이었다. 이 절에서는 안드로이드 설정파일인 매니페스트 파일의 인텐트 필터를 참조하여 인텐트를 자동생성하고 생성된 인텐트 스펙을 기반으로 테스팅하는 방법이다. 안드로이드 매니페스트 파일은 안드로이드 프로젝트 포함되고 컴파일된 안드로이드 바이너리 APK(Android application package)에도 포함되는 파일이므로 소스코드 존재 유무와 상관없이 테스트를 진행할 수 있다. 그러므로 앱 스토어와 같이 어플리케이션이 지속해서 등록되는 환경에서 인텐트로 발생하는 취약점을 파악하는 자동화된 방법으로 제공할 수 있다.

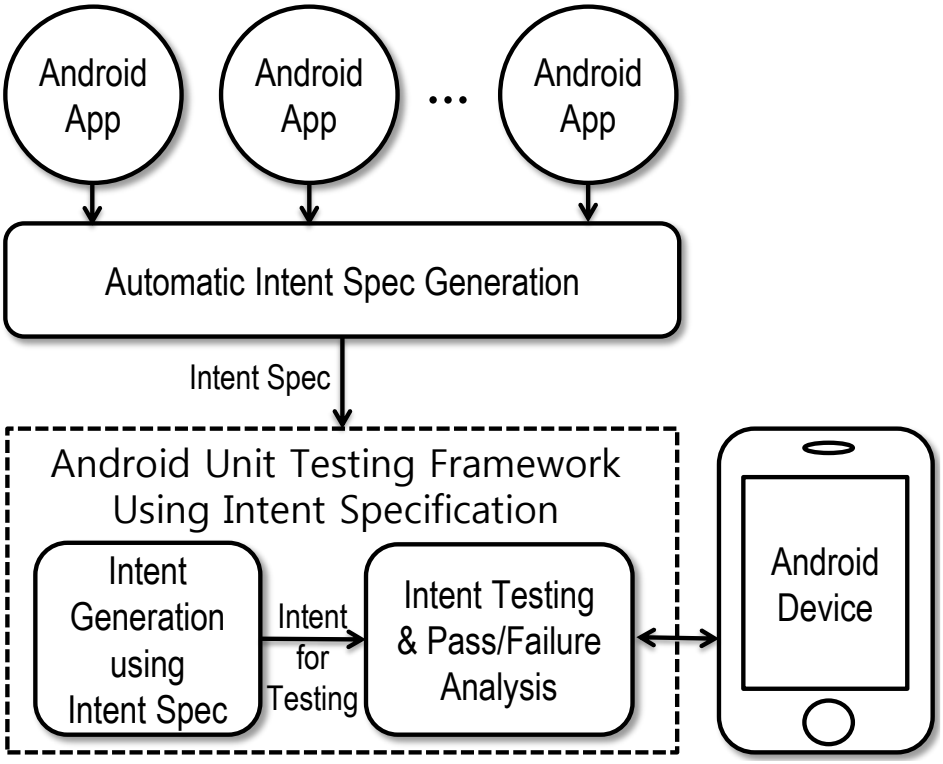

그림 4.12: 인텐트 스펙 자동 생성의 구조

그림 4.12는 인텐트 스펙 자동생성의 구조이다. 첫 번째 단계에서 안드로이드 어플

리케이션이나 안드로이드 매니페스트 파일을 입력으로 받는다. 두 번째 단계에서 입력된 안드로이드 어플리케이션에서 안드로이드 설정 파일이 있는 매니페스트 파일을 추출한다. 추출한 매니페스트 파일에서 컴포넌트가 외부로 노출한 인텐트를 이용해 인텐트 스펙을 생성한다. 어플리케이션에 따라서 차이가 있지만, 실험에 사용하기 위해 안드로이드 마켓에서 다운로드한 상위권 어플리케이션에서 추출된 인텐트는 평균 55.5개이다. 미자막 단계에서는 2절에서 설명한 인텐트 스펙 기반 안드로이드 유닛 테스팅 방법을 그대로 이용해서 테스트를 진행하고 결과를 돌려받는다.

3.1 인텐트 스펙 자동 생성 방법

이 절에서는 안드로이드 어플리케이션 바이너리 파일인 APK를 이용해 인텐트를 추출하는 방법에 관해서 설명한다. 안드로이드 어플리케이션 파일 APK는 ZIP 형식으로 압축된 파일로 ZIP 압축 해제 방식으로 해제할 수 있다. 압축을 풀면 어플리케이션의 바이트코드와 함께 컴파일된 안드로이드 매니페스트 파일이 포함된다. 이 안드로이드 매니페스트 파일에는 컴포넌트가 요구하는 인텐트의 형태가 선언되어 있어 이 정보를 이용해 컴포넌트가 요구하는 인텐트를 생성한다. 안드로이드 매니페스트에 선언되는 인텐트의 형태는 Action과 Data의 타입, Category만 표현할 수 있는 제약사항이 있어 실제로 컴포넌트가 요구하는 인텐트의 형태와는 차이가 있다. 이러한 인텐트 필터의 제약사항을 보안하기 위해서 랜덤으로 생성된 인텐트의 필드를 추가하고 수정하는 방식을 사용한다.

```
<intent-filter>
  <action name="android.intent.action.EDIT" />
  <category name="android.intent.category.DEFAULT" />
</intent-filter>
<intent-filter>
  <action name="android.intent.action.CREATE" />
  <category name="android.intent.category.DEFAULT" />
</intent-filter>
```

그림 4.13: Edit 액티비티의 인텐트 필터

그림 4.13는 안드로이드 매니페스트에 선언된 인텐트 정보이다. 이 정보를 이용해 인텐트 필터의 정보를 인텐트 스펙으로 변환하는 방법을 알아본다. 첫번째 인텐트 필터에는 Action으로 `android.intent.action.EDIT`이고, Category로 `android.intent.category.DEFAULT`를 요구한다. DEFAULT Category는 category 필드의 기본값이다. 두번째 인텐트 필터의 Action은 `android.intent.action.CREATE`이고, Category는 `android.intent.category.DEFAULT`이다. 이 인텐트 필터에서 표현된 인텐트 정보는 해당 컴포넌트가 어떤 데이터를 수정하고 생성한다는 것이다.

```
{ act = android.intent.action.EDIT
  cat=[android.intent.category.DEFAULT] }
||
{ act = android.intent.action.CREATE
  cat=[android.intent.category.DEFAULT] }
```

그림 4.14: 인텐트 필터로 자동 생성한 인텐트 스펙

그림 4.14은 그림 4.13의 인텐트 필터로부터 자동으로 생성한 인텐트 스펙이다. 인텐트 필터에 인텐트 정보를 인텐트 스펙으로 변환하는 방법은 인텐트 필터에 `action name`는 인텐트 스펙으로 `act`로 변환하고 해당 값은 그대로 사용한다. 인텐트 필터에 `category name`은 인텐스 스펙으로 `cat`로 변환하고, 해당 값은 그래도 사용한다. 해당 컴포넌트는 `EDIT`와 `CREATE` 두 가지의 인텐트를 기대하므로 인텐트 스펙에서는 두 가지의 인텐트를 표현한다는 의미를 {act=... cat=...} || {act=... cat=...}으로 표현된다. 이러한 원리를 이용해 안드로이드 APK 파일의 매니페스트 파일에 선언된 인텐트 정보를 추출하여 인텐트 스펙으로 변환한다.

자동으로 작성된 인텐트 스펙을 이용해 제2절에서 설명한 인텐트 스펙 기반 안드로이드 유닛 테스팅 프레임워크로 테스트한다. 이 절에서 제안하는 인텐트 스펙 자동생성 방법의 장점은 인텐트 스펙을 APK 파일로부터 추출하므로 소스코드가 없는 상태에서도 테스트할 수 있고, 이 모든 과정을 자동으로 진행할 수 있어 대량의 안드로이드 어플리케이션 등록되는 환경에서 인텐트로 발생하는 취약점을 쉽게 검사 할 수 있다.

그림 4.15은 안드로이드 APK파일을 입력으로 받아 매니페스트에서 추출한 인텐트

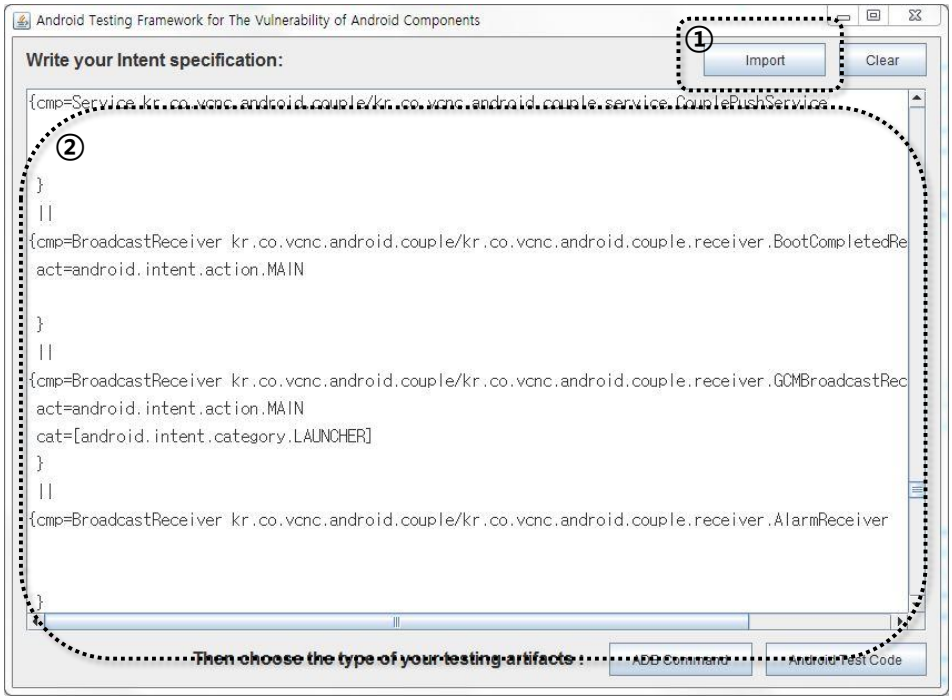

그림 4.15: 안드로이드 APK를 입력받아 인텐트 스펙을 생성한 화면

스펙의 화면이다. Input 영역에 있는 Import 버튼으로 APK를 지정하면, Intent Spec 영역에 안드로이드 매니페스트에 선언된 인텐트 정보를 이용해 인텐트 스펙을 생성해 출력한다.

3.2 실험결과

인텐트 스펙 자동생성 방법을 안드로이드 4.4.2 버전의 LG-F220K 환경에서 실험하고 결과를 표로 나타낸다. 이 방법은 소소코드가 없이 사용할 수 있는 방법으로 테스트한 어플리케이션은 안드로이드 마켓에서 상위권에 있는 안드로이드 어플리케이션 20개를 선정하여 실험하였다. 실험은 안드로이드 APK 파일에서 추출한 인텐트를 이용해 랜덤 생성한 인텐트로 각 어플리케이션을 제안한 방법으로 테스트하여 취약점을 파악하고 어떤 유형의 에러가 발생했는지 살펴본다. 마지막 실험에서 자주 발생하는 취약점을 확인하고 어떤 이유로 취약점이 발생하는지 살펴본다.

표 4.8: 인텐트 스펙을 자동 생성해 테스트한 실험결과

앱	인텐트 필터에서 action으로 구분해 추출한 인텐트 스펙의 수	랜덤 생성한 인텐트 수	정상 실행	비정상 종료
비트윈	38	2,280	2,226(97.6%)	54(2.4%)
CGV	121	7,260	6,989(96.3%)	271(3.7%)
GS SHOP	37	2,220	2,160(97.3%)	60(2.7%)
네이버	108	6,480	6,400(98.8%)	80(1.2%)
라인	41	2,460	2,417(98.3%)	43(1.7%)
멜론	108	6,480	6,438(99.4%)	42(0.6%)
모두의마블	8	480	455(94.8%)	25(5.2%)
문리더	6	360	360(100%)	0(0%)
아프리카티비	17	1,020	966(94.7%)	54(5.3%)
연세앱	7	420	420(100%)	0(0%)
우리은행	12	720	705(97.9%)	15(2.1%)
인스타그램	20	1,200	1,200(100%)	0(0%)
직방	14	840	813(96.8%)	27(3.2%)
카메라365	28	1,680	1,628(96.9%)	52(3.1%)
카카오스토리	53	3,180	3,082(96.9%)	98(3.1%)
카카오톡	91	5,460	5,276(96.6%)	184(3.4%)
쿠차	96	5,760	5,731(99.5%)	29(0.5%)
클린마스터	76	4,560	4,560(100%)	0(0%)
페이스북	153	9,180	9,180(100%)	0(0%)
후후	72	4,320	3,991(92.4%)	329(7.6%)
합 계	1,106	66,360	64,997(97.9%)	1,363(2.1%)

표 4.8은 인텐트 스펙 자동생성을 안드로이드 마켓의 상위권에 있는 어플리케이션 이 20개에 실험한 결과이다. 첫 번째 열은 실험한 안드로이드 앱의 이름이고, 두 번째 열은 제안한 방법을 이용해 APK 파일로부터 추출한 인텐트 스펙의 수이다. 인텐트의 action 필드에 따라서 요구하는 인텐트의 형태가 달라지므로 인텐트 스펙은 action으로 구분해 추출한다. 세 번째 열은 추출한 인텐트를 이용해 랜덤 생성한 인텐트의 수로 랜덤 생성한 인텐트를 이용해 테스트한다. 네 번째 열은 랜덤 인텐트로 테스트해 정상 실행으로 판단된 인텐트의 수이다. 다섯 번째 열은 랜덤 인텐트로 테스트해 비정상 종료된 인텐트의 수이다.

두 번째 열의 APK로 추출한 인텐트의 수는 APK 파일로부터 추출한 인텐트 스펙의

수이다. 어플리케이션에 따라서 사용하는 인텐트의 개수는 다르지만 다른 어플리케이션과 연동이 많은 인기 어플리케이션이 사용하는 인텐트의 수도 많음을 알 수 있다. 이 실험에서 추출한 인텐트의 수는 1,106개이고, 어플리케이션이 평균 55.3개의 인텐트를 사용한다.

세 번째 열의 랜덤 생성한 인텐트 수는 APK로 추출한 인텐트를 기반으로 랜덤 생성한 인텐트 수이다. 생성 시 인텐트에 랜덤 생성한 새로운 필드를 추가하는 Compatible 모드로 20개, 요구하는 인텐트에서 랜덤 생성 값과 새로운 필드를 추가하는 Shape-Compatible 모드로 20개, 모든 필드와 값을 랜덤하게 생성하는 Random 모드로 20개씩 생성했다. 요구하는 인텐트 1개당 60개의 랜덤 인텐트를 생성하고 생성된 랜덤 인텐트를 이용해 제안한 테스팅을 진행한다. 총 테스트한 인텐트는 66,360개로 각 어플리케이션에 테스트한 평균 인텐트 수는 3,318개이다.

네 번째 열의 정상 실행은 랜덤 생성한 인텐트로 테스트하여 정상 실행으로 판단된 인텐트의 수를 나타낸다. 랜덤 생성한 총 66,360개의 인텐트로 테스트한 결과 64,997(97.9%)개의 인텐트가 정상적으로 실행됐다. 랜덤 생성한 인텐트가 모두 정상 실행된 어플리케이션은 20개 중 5(25%)개로 많은 수의 어플리케이션이 인텐트로 인한 취약점이 발생함을 알 수 있다.

다섯 번째 열의 비정상 종료는 랜덤 생성한 인텐트를 어플리케이션에 전달해 비정상 종료된 수를 의미한다. 랜덤 생성한 총 66,360개의 인텐트에서 비정상 종료된 인텐트는 1,363(2.1%)개이다. 테스트한 20개의 어플리케이션 중에서 15(75%)개의 어플리케이션이 제안한 벙법으로 테스트해 취약점이 발생한다. 어플리케이션이 비정상 종료되는 취약점은 표 4.9에서 확인할 수 있다.

표 4.9는 랜덤 생성한 인텐트로 인해 각 어플리케이션이 비정상 종료된 에러의 수를 나타낸다. 실험에서 11개의 어플리케이션에서 비정상 종료되는 NullPointerException 에러가 발생하고, 발생된 개수가 1,084(78.5%)개로 가장 높은 비유을 차지한다. NullPointerException 에러는 인텐트에서 누락된 정보가 존재하면 발생하기 때문에 랜덤으로 생성한 인텐트에 컴포넌트가 요구하는 정보가 부족해 취약점이 발생한다. 실험에서 5개의 어플리케이션에서 비정상 종료되는 UnsupportedOperationException 에러

표 4.9: 어플리케이션별 비정상 종료 에러

앱	NullPointer Exception	UnsupportedOperation Exception	에러메시지 불명
비트윈	54	0	0
CGV	269	0	2
GS SHOP	60	0	0
네이버	22	0	58
라인	43	0	0
멜론	0	42	0
모두의마블	8	17	0
아프리카티비	47	7	0
우리은행	0	15	0
직방	27	0	0
카메라365	0	0	52
카카오스토리	98	0	0
카카오톡	128	31	25
쿠차	0	0	29
후후	328	0	1
합 계	1,084(79.5%)	112(8.2%)	167(12.3%)

가 발생하고, 발생된 개수는 112(8.2%)개이다. UnsupportedOperationException 에러는 인텐트의 정보에 URL 형태의 정보가 넘어가지만, 해당 URL이 부정확한 경우에 발생하므로 다른 어플리케이션과 연동하여 URL 형태를 사용 경우에 취약점이 발생해 비정상 종료된다. 마지막으로 6개의 어플리케이션에서 비정상 종료된 167(12.3%)개의 에러메시지 불명은 어플리케이션이 비정상 종료된 로그가 발생했지만 정확한 에러 메시지는 파악할 수 없던 경우이다. 소스코드가 존재하지 않으므로 자세한 에러의 유형은 파악하지 못했다. 하지만 제안한 테스팅 방법은 어떤 인텐트에서 취약점이 발생하는지 판단할 수 있는 Excel 파일이 결과로 제공되므로 소스코드에 접근할 수 있는 개발자는 취약점을 파악 가능하다.

표 4.8과 4.9의 실험결과를 요약하면, 제안한 인텐트 스펙 자동생성 방법으로 전체 20개의 어플리케이션 중 15(75%)개의 어플리케이션에서 취약점을 파악할 수 있었다. APK 파일에서 추출한 인텐트를 이용해 랜덤 생성한 66,360개의 인텐트로 실험한 결과 64,997(97.9%)개의 인텐트가 정상 실행되고 1,363(2.1%)개의 인텐트가 취약점이 발생

하여 NullPointerException, UnsupportedOperationException, 에러메시지 불명으로 비정상 종료되었다. 이 실험을 통해서 제안한 인텐트 스펙 자동생성 방법으로 인텐트로 발생하는 취약점을 소스코드 없이 파악할 수 있음을 확인했다.

제 4 절 인텐트 스펙 기반 JUnit 테스트 케이스 자동생성

인텐트 스펙 기반 JUnit 테스트 케이스 자동생성은 인텐트 스펙을 기반으로 랜덤 JUnit 테스트 케이스를 생성하여 테스트하는 방법이다. 안드로이드 개발 도구에서 JUnit 테스트 프레임워크[11]를 확장 지원한다. 개발자가 원하는 테스트 케이스를 작성하고 실행하면 해당 테스트를 진행하고 그 결과를 알린다. 실패한 테스트의 경우 그 로그를 확인해 디버깅하거나 수정할 수 있는 기능을 제공한다.

본 논문에서 제안한 기존 테스팅 방법은 PC와 디바이스를 연결하는 ADB를 이용했다. ADB는 소스코드 없이 테스트가 가능하다는 장점이 있지만, Extra 필드의 타입이 null, String, Boolean, Int, Long, Float, URI, Component, Int[], Long[], Float[]로 제한되고 외부에서 호출할 수 있는 노출된 컴포넌트만 테스트할 수 있다. 이 절에서 제안하는 JUnit 테스트 방법은 소스코드가 있는 상태에서만 테스트가 가능한 방법이지만 Extra 필드의 타입이 제한이 없고 내부에서만 사용하는 컴포넌트도 테스트 가능하다. 또한, JUnit 테스트는 테스트 속도가 빠르다.

JUnit 테스트 프레임워크를 사용하기 힘든 단점은 테스트 케이스를 개발자가 직접 작성해야 한다는 것이다. 테스트를 위해서 많은 테스트 케이스를 작성해야 하는 어려움이 있다. 또한, 테스트할 대상의 코드가 변경되면 테스트 케이스도 같이 변경해야 한다.

이러한 문제를 해결하는 방법으로 인텐트 스펙으로부터 JUnit 기반 테스트 케이스를 자동 생성하는 방법을 제안한다. 이 방법은 안드로이드 바이너리의 인텐트 정보와 선언한 인텐트 스펙을 기반으로 JUnit 테스트 케이스를 자동으로 생성한다. 개발자는 테스트 케이스를 작성할 필요 없이 인텐트 스펙이나 안드로이드 바이너리 파일을 이용해서 JUnit 테스트에 필요한 테스트 케이스를 생성할 수 있어 테스트 케이스를 작성해야 하는 어려움을 해결할 수 있다.

그림 4.16은 인텐트 스펙 기반 JUnit 테스트 케이스 자동생성의 구조이다. 테스트 케이스를 작성하기 위해서 안드로이드 바이너리나 인텐트 스펙을 직접 입력한다. 이때, 안드로이드 바이너리 파일은 3절의 방법과 동일하게 인텐트 필터를 이용해서 인텐트를

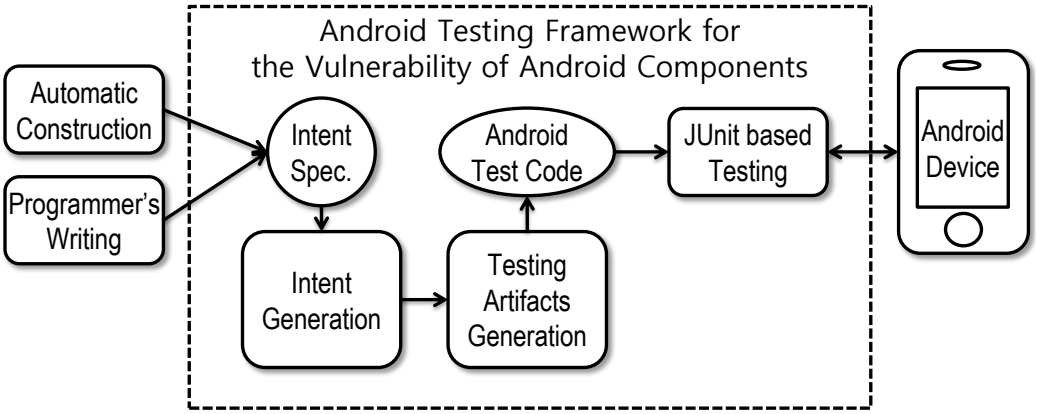

그림 4.16: 인텐트 스펙 기반 JUnit 테스트 케이스 자동생성

추출하고 인텐트 스펙을 생성한다. 인텐트 생성 단계에서는 인텐트 스펙을 이용해 랜덤한 인텐트를 생성한다. 생성된 인텐트를 이용해 JUnit에서 사용되는 테스트 케이스를 생성하여 JUnit 기반으로 테스팅한다.

4.1 JUnit 테스트 케이스 자동 생성 방법

JUnit 테스트 케이스 자동 생성 방법은 인텐트 스펙을 기반으로 JUnit 테스크 케이스를 자동으로 생성하는 방법이다. 이 방밥은 JUnit 테스트에서 테스트 케이스를 작성해야되는 단점을 보완하고 JUnit 테스트를 쉽게 자동화할 수 있다. JUnit 테스트 케이스를 자동으로 생성하는 방법은 두 가지 입력을 지원한다. 안드로이드 바이너리에서 인텐트 스펙을 자동 추출하는 방법은 사용자가 테스트 케이스를 작성할 필요가 없다는 장점이 있고 개발자가 직접 인텐트 스펙을 작성하는 방법은 취약점이 쉽게 발생하는 부분을 인텐트 스펙으로 작성하면 자동으로 JUnit 테스크 케이스를 쉽게 생성할 수 있는 장점이 있다.

```
   { act = android.intent.action.CREATE }
|| { act = android.intent.action.EDIT
     [ title = String,
       content = String ] }
```

그림 4.17: 인텐트 스펙

그림 4.17는 Note 액티비티 컴포넌트가 요구하는 인텐트의 모양이다. 액션은 CREATE나 EDIT을 기대한다. EDIT Action은 Extra로 title과 content 키로 String 타입 값을 요구한다. 이 인텐트 스펙을 이용해서 랜덤 인텐트를 생성한다.

```
{ act = android.intent.action.EDIT
 [ title = String "my title",
   content = String "my content" ]
 dat = qoFXwARtpfV_LNN }
```

그림 4.18: 랜덤 생성된 인텐트 스펙

그림 4.18은 그림 4.17를 이용해서 랜덤 생성한 예이다. 인텐트의 값과 구조를 변경하지 않고 새로운 필드만 추가하는 Compatible모드로 생성한 예로 기존 인텐트에 랜덤으로 생성된 Data 필드만 추가됐다.

```
public class NoteTest_0001
          extends ActivityUnitTestCase<Note>
{
  public NoteTest_0001()
  { super(Note.class); }
  public void testcase1() {
    Intent intent = new Intent();
    intent.setAction("android.intent.action.EDIT");
    intent.putExtra("title", "my title");
    intent.putExtra("content", "my content");

    try{
      intent.setData(Uri.parse("qoFXwARtpfV_LNN"));
    }
    catch(Throwable t) { }
    startActivity(intent, null, null);
  }
}
```

그림 4.19: 인텐트 스펙으로 생성된 JUnit 코드

그림 4.19은 JUnit 테스트 케이스로 생성한 코드이다. 생성된 랜덤 인텐트로 JU-

nit에서 사용하는 테스트 케이스를 개수와 상관없이 자동으로 생성할 수 있다. 또한, ADB를 이용한 테스트 방법은 Extra로 전달할 수 있는 타입이 String, int, Long 등과 같은 기본 타입으로 제한되지만 JUnit 테스트는 Extra의 타입이 제한되지 않는 장점이 있다. 이 JUnit 코드는 새로운 Intent를 생성하고 `setAction` 메서드로 `android.intent.action.EDIT`을 지정한다. `EDIT` Action은 Extra를 요구하므로 `putExtra` 메서드로 `title`과 `content`키로 `String` 타입의 값을 저장한다. 랜덤 생성된 Data는 `setData` 메서드로 값을 지정한다. 마지막으로 `startActivity`로 인텐트를 실행한다.

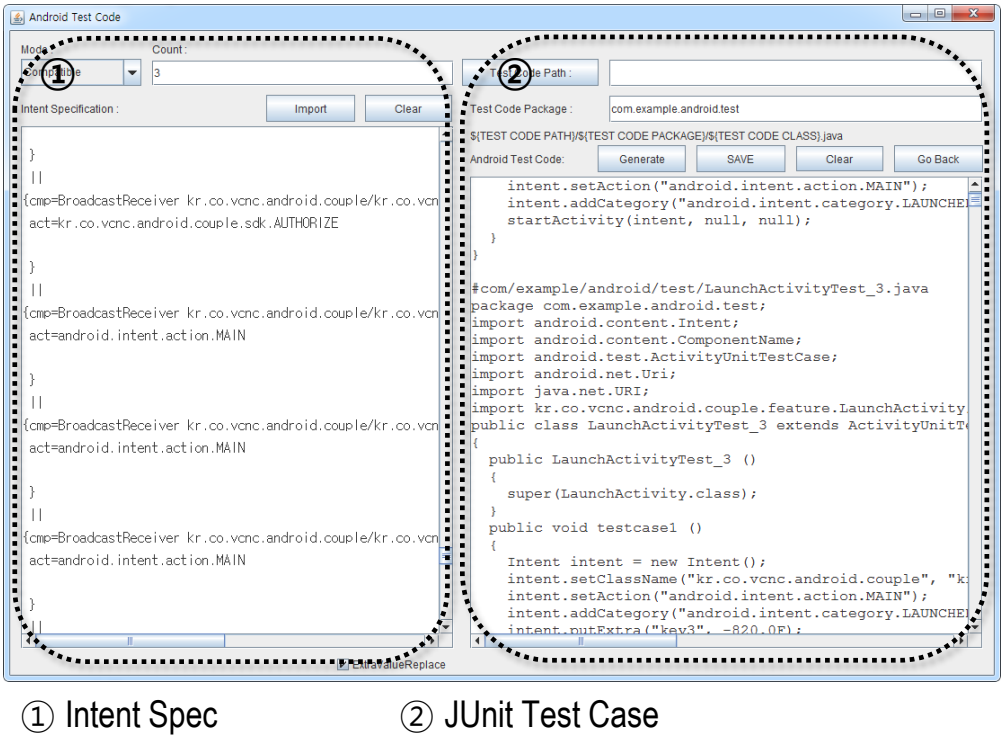

그림 4.20: 인텐트 스펙으로 생성한 JUnit 테스트 케이스

그림 4.20는 안드로이드 바이너리를 입력받아 생성한 인텐트 스펙을 이용해 JUnit 테스트 케이스를 생성하는 화면이다. 좌측은 상용 어플리케이션의 안드로이드 바이너리 파일에서 인텐트 스펙을 생성하고 이 인텐트 스펙을 이용해 랜덤 생성한 화면이다. 경우에 따라서 사용자는 특정한 부분을 위한 인텐트 스펙을 직접 작성하여 해당 부분의

취약점만을 위한 인텐트를 작성할 수 있다. 우측은 랜덤 생성된 인텐트 스펙을 이용해서 JUnit 테스트 케이스를 작성한 화면이다. 인텐트 스펙을 기반으로 JUnit 테스트 프레임워크에서 사용되는 테스트 케이스를 코드로 자동으로 생성해 많은 테스트 케이스를 작성해야 하는 테스트의 어려움을 해결한다.

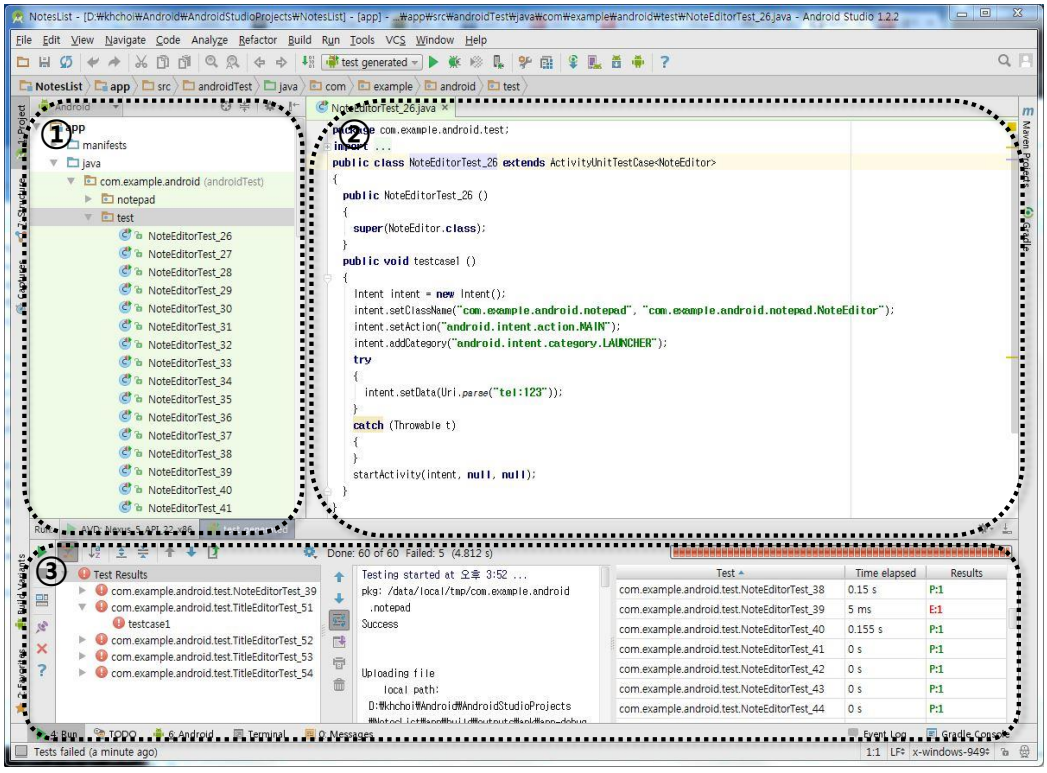

그림 4.21: 안드로이드 스튜디오에서 실행한 JUnit 테스트

그림 4.21은 자동으로 생성된 JUnit 테스트 케이스로 안드로이드 스튜디오에서 테스트하는 화면이다. 안드로이드 스튜디오에서는 JUnit 테스트를 위한 사용자 인터페이스를 제공한다. 테스트 케이스에 자동으로 생성된 테스트 케이스를 등록한다. 생성된 테스트 케이스를 이용해서 JUnit 테스트를 위한 다른 준비 없이 바로 테스트를 진행할 수 있을 정도로 간편하다. 테스트 케이스 코드는 각각의 테스트 케이스를 직접 확인하거나 추후에 취약점이 있는 테스트 케이스를 파악하여 취약점을 수정할 수 있다.

테스트 진행 시 테스트 결과에 성공하지 못한 테스트 케이스가 나타나고 해당 테스트로 발생한 취약점을 쉽게 파악할 수 있다.

제 5 장

결론 및 향후 연구

본 논문에서는 인텐트를 표현하는 방법으로 인텐트 스펙을 제안하고, 인텐트 스펙을 응용하여 인텐트에 의해 발생하는 안드로이드 어플리케이션의 취약점을 방어하는 방법을 제안했다. 본 논문에서 공헌한 점은 다음과 같다.

첫째, 안드로이드에서 인텐트를 표현하는 새로운 방법을 제안했다. 기존 안드로이드에서는 안드로이드 속성을 기술하는 인텐트 필터와 소스코드에 메서드로 인텐트를 표현해 파악하기 힘들다는 단점이 있다. 인텐트 스펙을 이용하면 다양한 형태의 인텐트를 쉽게 선언할 수 있다는 장점이 있다. 또한, 인텐트 스펙을 응용하면 인텐트로 인해 발생하는 취약점을 다양한 형태로 방어할 수 있다.

둘째, 컴포넌트 개발자가 인텐트로 인해 발생하는 취약점을 쉽게 방어할 수 있는 인텐트 스펙 실행시간 검사 방법을 제안했다. 기존 안드로이드 시스템은 잘못된 인텐트를 검사하는 방법을 제공하지 않고 개발자에게 맡겨 많은 앱에서 취약점이 발생한다. 이러한 문제를 해결하기 위해서 실행시간에 컴포넌트가 요구하는 인텐트와 전달된 인텐트가 부합하는 여부를 판단하고 부합하지 않는 잘못된 인텐트를 방어할 방법을 제안해 취약점을 개선했다.

셋째, 인텐트 스펙을 기반으로 다양한 형태의 인텐트를 이용해 안드로이드 어플리케이션의 취약점을 확인하는 유닛 테스팅 방법을 제안했다. 기존 연구들은 일부의 필드로 테스트하거나 소스코드를 분석해 발생할 수 있는 취약점을 판단하는 방법을 사용했다. 본 논문에서 제안한 방법은 인텐트 스펙을 기반으로 인텐트를 생성해 테스트하므로

코드 커버리지의 범위를 조절할 수 있어 취약점이 자주 발생하는 인텐트의 형태나 테스트를 요하는 부분을 집중적으로 테스트할 수 있다는 장점이 있고, 쉽게 다양한 형태를 만들어 낼 수 있다는 장점이 있다.

넷째, 인텐트 스펙을 어플리케이션에서 자동으로 추출하는 방법을 제안했다. 테스팅 자동화 프레임워크는 안드로이드 바이너리로 인텐트 스펙을 생성한다. 생성된 인텐트 스펙을 기반으로 랜덤한 인텐트를 생성하여 안드로이드 어플리케이션을 자동으로 테스트한다. 이 결과로 정상 실행여부와 비정상 실행여부를 결과로 돌려주고 해당 로그를 수집한다. 관리자는 이 내용을 가지고 어플리케이션을 마켓에 등록할 것인지 판단할 수 있고 더 나아가 개발자에게 취약점에 대한 피드백을 제공할 수 있다.

다섯째, 인텐트 스펙을 기반으로 생성한 랜덤 인텐트를 JUnit 테스트 케이스를 생성해주는 방법을 제안했다. 기존 JUnit 테스트 케이스를 작성하려면 개발자가 직접 랜덤하게 코드를 작성해야한다. 이러한 문제점을 해결하기 위해서 다양한 인텐트를 JUnit 기반으로 테스트 케이스를 작성해준다. 이 방법을 이용하면 JUnit 테스트 케이스를 코드를 생성해주므로 인텐트로 발생하는 취약점 판단을 위한 비용을 줄일 수 있다.

이러한 결과로 볼때 인텐트의 다양한 형태를 선언할 수 있는 인텐트 스펙을 기반으로 다양한 응용 프로그램을 만들 수 있다는 점과 인텐트로 발생하는 취약점을 효율적으로 파악할 수 있다는 점을 알 수 있다.

본 논문의 향후 연구는 다음과 같은 세 가지가 필요하다.

첫째, 랜덤 생성하는 인텐트의 형태를 더 정교하게 생성하는 방법이 필요하다. 기존 테스트에서 취약점을 발생시킨 인텐트들의 형태를 피드백으로 받는다. 랜덤 생성시 피드백된 형태의 인텐트의 생성 비율을 높여 테스트에서 취약점 파악의 효율성을 높인다.

둘째, 랜덤 인텐트로 발생하는 취약점을 테스트하는 방법은 같은 에러의 메시지가 자주 발생하여 결과 분석을 어렵게 만든다. 이러한 문제를 개선하기 위해서 동일한 에러 로그를 그룹해 에러 유형별로 확인하는 기법이 필요하다. 이 방법은 결과 분석의 효율성을 높인다.

셋째, 랜덤 인텐트로 취약점을 분석하는 방법은 정적으로 소스코드를 분석하는 방법보다 간단하지만, 소스코드의 모든 부분을 테스트하지 못해 커버리지가 낮은 단점이

있다. 이러한 문제를 개선하기 위해서 정적으로 소스코드를 분석해 판단하는 기법이 필요하다. 이 방법은 정적으로 모든 소스코드를 분석해 취약점이 발생할 수 있는 부분을 자동으로 체크한다.

참고 문헌

[1] "Android API." [Online]. Available: `http://developer.android.com`.

[2] J. Gosling, B. Joy, G. L. Steele Jr, G. Bracha, and A. Buckley, *The Java Language Specification*. Pearson Education, 2014.

[3] A. K. Maji, F. A. Arshad, S. Bagchi, and J. S. Rellermeyer, "An empirical study of the robustness of inter-component communication in android," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pp. 1–12, IEEE, 2012.

[4] "Dart." [Online]. Available: `https://github.com/f2prateek/dart`.

[5] "Jackson." [Online]. Available: `https://github.com/FasterXML/jackson`.

[6] "Intent fuzzer." [Online]. Available: `https://www.isecpartners.com/tools/mobile-security/intent-fuzzer.aspx`.

[7] R. Sasnauskas and J. 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)*, pp. 1–5, ACM, 2014.

[8] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th International*

Conference on Mobile Systems, Applications, and Services, pp. 239–252, ACM, 2011.

[9] "Android debug bridge." [Online]. Available: `https://developer.android.com/tools/help/adb.html`.

[10] K. Claessen and J. Hughes, "Quickcheck: a lightweight tool for random testing of haskell programs," *ACM Sigplan Notices*, vol. 46, no. 4, pp. 53–64, 2011.

[11] "Android unit test." [Online]. Available: `https://developer.android.com/training/testing/unit-testing/local-unit-tests.html`.

[12] "Adb shell arguments." [Online]. Available: `http://developer.android.com/tools/help/shell.html#shellcommands`.

부록

부록에서는 제안한 인텐트 스펙과 취약점을 개선하기 위한 툴을 구현한 내용을 소개한다. 부록은 선언된 인텐트 스펙을 파싱하는 파서와 인텐트 스펙을 기반으로 인텐트 랜덤 생성하는 방법, 툴 자동화를 위한 ADB 사용법, 디바이스의 로그를 패턴 매칭하여 자동으로 결과를 알리는 방법을 설명한다.

제 1 절 인텐트 스펙 파서

인텐트 스펙을 선언하고 선언한 내용을 인텐트의 데이터구조로 파싱하기 위해서 인텐트 스펙 문법에 따라서 인텐트 스펙 파서를 구현했다. 인텐트 스펙 파서는 좌측유도(Leftmost derivation) 방식이고, 안드로이드 폰에서 동작할 수 있도록 Java로 작성된 것과 PC에서 사용할 수 있는 함수언어인 Haskell 두 가지 방식으로 작성되었다.

Haskell에서 인텐트 스펙에서 Action을 파싱하는 예로 간략하게 설명한다. Haskell로 구현된 코드를 설명하는 이유는 안드로이드의 파서를 구현할 때 먼저 Haskell에서 파서를 구현한 후 Haskell 코드를 Java로 재작성하는 방법을 사용했기 때문이다. Haskell은 함수를 인자로 전달할 수 있는 함수형 언어로 복잡하지 않고 직관적이므로 파서를 빠르게 구현있어 실수를 줄일 수 있다. 인텐트 스펙으로 선언된 Action 필드를 파싱하는 코드를 간단하게 살펴보자.

```
action :: Parser Field
action = do symbol "act"
            symbol "="
            act <- idOrDot
            return (Action act)
```

인텐트 스펙이 act = android.intent.action.CREATE 형태로 입력되었다고 가정하자. action 함수는 파싱할 문자열의 첫 번째 토큰이 act인지 확인한다. act기 맞으므로 다음 문자는 = 인지 확인할 것이다. 다음은 android.intent.action.CREATE를 확인할 차례이다. idOrDot 함수는 첫 문자가 알파벳이고 나머지는 알파벳, 숫자, _, ., $인 문자열을 돌려줄 것이므로 android.intent.action.CREATE 형태가 돌아온다. action 함수는 Action이라는 데이터 타입에 android.intent.action.CREATE 문자열을 값으로 저장해 돌려준다.

```
symbol    :: String -> Parser String
symbol xs =  token (string xs)
```

action 함수에서 사용한 symbol 함수는 string 함수로 띄어쓰기나 공백으로 나뉜 문자열을 읽어드려 token을 만들어주는 기능을 제공한다. 만약 symbol "act"를 입력하면 선언된 인텐트 스펙에 파싱할 문자가 "act"인지 확인해 token으로 돌려주거나 failure를 돌려준다.

```
string         :: String -> Parser String
string []      =  return []
string (x:xs) =  do char x
                    string xs
                    return (x:xs)
```

symbol 함수에서 사용한 string 함수는 문자들을 모아 문자열 만들어 돌려주는 함수이다. 만약 입력된 것이 없다면 빈 문자열을 돌려준다. 문자가 있다면 문자를 char로 확인해 만족할 때까지 다시 string 함수를 재귀하여 문자를 모아 리스트 형태로 돌려주는 기능을 제공한다. 함수의 결과는 빈 문자열 또는 문자열이다.

```
char   :: Char -> Parser Char
char x =  sat (== x)
```

string 함수에서 사용한 char 함수는 원하는 문자가 맞는지 sat 함수로 확인해서 돌려주는 기능을 제공한다. 이 함수의 결과는 한 문자가 원하는 문자가 맞다면 그 문자를 돌려줄 것이고 아니면 실패를 알리는 failure를 돌려줄 것이다.

```
sat   :: (Char -> Bool) -> Parser Char
sat p =  do x <- item
```

```
if p x then return x else failure
```

char 함수에서 사용한 sat 함수는 선언된 인텐트 스펙으로부터 한 문자를 읽어온다. 해당 문자가 원하는 문자와 같다면 문자를 돌려주고, 같지 않으면 실패를 알리는 함수이다. 이 함수는 문자와 숫자, 특정 문자인지를 검사하는 기본적인 기능을 제공한다. 이 함수의 결과는 문자를 돌려주거나 실패를 알리는 failure를 돌려준다.

```
idOrDot :: Parser String
idOrDot =  token identOrDot
```

action 함수에서 사용한 idOrDot 함수는 파싱할 문자열을 identOrDot 함수에 전달하여 token을 만들어 돌려주는 기능을 제공한다. 이 함수를 이용해서 첫 문자가 알파벳이고 나머지는 알파벳, 숫자, _, ., $인 문자열을 token으로 만든다. 이 함수까지 작성되면 이제 act = android.intent.action.CREATE 형태의 인텐트 스펙을 파싱할 준비가 된 것이다.

```
identOrDot :: Parser String
identOrDot = do x  <- letter
                xs <- many alphanumOrDot
                return (x:xs)
```

idOrDot 함수에서 사용한 identOrDot 함수는 파싱할 문자열에서 첫 문자가 알파벳이고 나머지는 알파벳이나 문자인 문자열을 만들어서 돌려주는 기능을 제공한다. letter함수를 이용해서 파싱할 첫 문자가 알파벳이면 나머지는 알파벳이나 숫자, _, ., $인 문자열을 만들어 돌려준다. 첫 문자가 알파벳이 아니라면 sat 함수에서 failure를 돌려준다. 예를 들어, a29sdk4$인 문자라면 a29sdk4$를 돌려주고, 29sdk4$이면 첫 문자가 알파벳이 아니므로 failure를 돌려준다.

```
letter :: Parser Char
letter =  sat isAlpha
```

identOrDot 함수에서 사용한 letter 함수는 파싱할 문자열의 시작 문자가 알파벳이 맞는지 isAlpha 함수로 검사하는 기능을 제공한다. 이 함수의 결과는 문자가 알파벳이라면 알파벳을 돌려주고 알파벳이 아니라면 failure를 돌려준다.

```
alphanumOrDot :: Parser Char
```

```
alphanumOrDot = do s <- sat isAlphaNum
                   return s
                +++ do s <- sat (== '_')
                       return s
                +++ do s <- sat (== '.')
                       return s
                +++ do s <- sat (== '$')
                       return s
```

identOrDot 함수에서 사용한 alphanumOrDot 함수는 파싱할 문자가 문자나 숫자가 만족하거나 _, ., $이면 그 값을 돌려주고 아니면 failure를 돌려주는 기능을 제공한다. 이 함수를 이용해서 문자, 숫자, 입력 가능한 특수 문자를 구별할 수 있다. 만약 인텐트 스펙에 선언된 문자에 입력할 수 없는 문자가 입력되었다면 이 함수에서 failure를 돌려줄 것이다.

```
many :: Parser a -> Parser [a]
many p =  many1 p +++ return []

many1 :: Parser a -> Parser [a]
many1 p =  do v  <- p
              vs <- many p
              return (v:vs)
```

identOrDot 함수에서 사용한 many 함수는 원하는 문자인지 검사해서 원하는 문자열을 만들어 주는 기능을 제공한다. 만약 만족하지 않으면 빈 문자열을 돌려주고 만족하면 many1 함수로 만족하지 않을 때까지의 문자를 모아서 돌려준다.

many1 함수는 파싱할 문자열의 첫 번째 문자가 조건에 만족하는지 확인하고 다시 many 함수를 호출한다. many 함수는 다시 many1 함수를 호출할 것이고 만족하면 many 함수와 many1 함수를 서로를 부르는 재귀를 한다. 만약 만족하지 않는 문자가 나오게 되면 many에서 빈 리스트를 돌려주게 되므로 만족한 문자열을 돌려준다.

나머지 자세한 소스는 https://github.com/kwanghoon/AndroidAppTester에서 확인할 수 있다.

제 2 절 인텐트 스펙 랜덤 생성

선언된 인텐트 스펙을 이용해서 랜덤한 인텐트를 생성하기 위해서 QuickCheck[10] 라이브러리를 이용한다. QuickCheck 라이브러리는 정의한 데이터 구조에 맞춰 랜덤한 데이터 구조 생성하고 해당 데이터 구조의 값도 랜덤한 형태로 생성하는 기능을 제공한다. 이 절에서는 QuickCheck 라이브러리를 이용해서 랜덤한 인텐트를 생성하는 방법을 간략히 살펴본다.

```
intTypeElements :: Gen Int
intTypeElements = elements [-500..500]
```

우선 int 타입의 값을 랜덤하게 생성해보자. intTypeElements 함수는 integer 타입의 값을 생성하는 기능을 제공한다. elements는 인자로 들어온 리스트에서 한 값을 랜덤하게 뽑아서 돌려준다. 이 예에서는 -500부터 500까지의 값 중 한 값을 랜덤하게 뽑아서 돌려준다.

```
floatTypeElements :: Gen Float
floatTypeElements = elements [-1000.0..1000.0]
```

float 타입의 값을 랜덤하게 생성하는 floatTypeElements 함수도 intTypeElements와 비슷한 형태를 가진다. floatTypeElements 함수는 -1000.0부터 1000.0까지의 한 값을 랜덤하게 뽑아서 돌려준다. 예를 들어, 랜덤하게 생성된 float 타입의 값은 10.2939183가 될 수 있다.

```
actionElements = elements ["com.android.action.Edit",
                           "com.android.action.Add",
                           "com.android.action.Delete"]

actionArbitrary = listOf1 $ elements
              (['a'..'z'] ++ ['A'..'Z'] ++ ['.', '_'])
```

인텐트에서 Action 필드의 값을 만들어 내는 함수를 고려해보자. 인텐트에서 Action은 안드로이드에서 정의한 값을 사용하는 방법과 사용자가 자기 직접 정의한 값을 사용하는 두 가지 방법이 있다. 그러므로 랜덤한 Action 필드의 값은 두 가지 방식을 모두 고려해 생성한다.

actionElements 함수는 안드로이드에 이미 정의된 값을 사용하는 방식에서 랜덤한 Action 필드의 값을 생성하는 방법이다. elements 함수는 “com.android.action.Edit”, “com.android.action.Add”, “com.android.action.Delete” 중에서 한 값을 랜덤하게 선택해 Action 필드의 값으로 사용한다. 랜덤하게 두 번째 값을 선택한다면 Action 필드의 값은 com.android.action.Add이다.

actionArbitrary 함수는 사용자가 정의한 값을 사용하는 방식에서 랜덤한 Action 필드의 값을 생성하는 방법이다. listOf1 함수는 뒤에 따라오는 함수를 여러 번 실행하여 그 값을 모아 리스트를 돌려주는 함수이다. listOf1 함수에 인자로 들어온 elements 함수는 인자로 들어온 리스트에서 한 문자를 랜덤하게 선택해 값을 만들다. 인자로 들어온 리스트는 a-z, A-Z, . _이다. 그러므로 대문자 알파벳과 소문자 알파벳, ., _중에서 값이 랜덤하게 선택됨을 반복하고 listOf1은 그 값을 모아 문자열로 돌려준다. 실제로 랜덤하게 생성된 값은 xaQ, JAnqKrJoyWY, oJhWZ._E 같은 모양이다.

```
instance Arbitrary ExtraType where
  arbitrary = oneof[ liftM StringType stringTypeArbitrary,
                     liftM IntArray intArrayElements,
                     liftM FloatArray floatArrayElements]
```

ExtraType arbitrary 인스턴스는 Extra 필드의 타입과 값을 랜덤하게 생성하는 기능을 제공한다. 이 예에서는 간략하게 String 타입과 int 타입 배열, float 타입 배열의 값을 생성하는 방법을 살펴본다.

```
stringTypeArbitrary = listOf1 $ elements (['a'..'z'] ++ ['A'
   ..'Z'] ++ ['0'..'9'] ++ [' ', '.', '_', '/', '?', ])
```

stringTypeArbitrary 함수는 문자열 값을 랜덤하게 생성하는 기능을 제공한다. elements 함수는 a-z, A-Z, 0-9, 공백, _, , ? 중 한 문자를 선택하고 listOf1 함수는 elements 함수를 랜덤하게 반복시킨 후 문자열 리스트를 만들어준다. 성생된 값은 Wekdx93j23. 129?31S의 형태이다.

```
intTypeElements :: Gen Int
intTypeElements = elements [-500..500]
intArrayElements = listOf1 $ intTypeElements
```

```
floatTypeElements :: Gen Float
floatTypeElements = elements [-1000.0..1000.0]
floatArrayElements = listOf1 $ floatTypeElements
```

intArrayElements 함수는 랜덤하게 int 타입의 값을 만드는 intTypeElements 함수를 호출하고 listOf1으로 랜덤하게 반복한다. 반복해서 얻어진 리스트를 돌려준다.

floatArrayElements 함수도 intTypeElements 함수와 비슷하게 float 타입의 값을 만드는 floatTypeElements 함수를 호출하고 listOf1으로 얻어진 리스트를 돌려준다.

제 3 절 PC와 디바이스를 연결하는 ADB

본 논문에서 제안하는 방법의 큰 장점 중 하나는 테스트부터 결과까지 자동화된 테스트를 지원하는 것이다. 자동화를 위해서는 인텐트를 디바이스에 전달하여 실행하고 실행 중 발생한 로그를 분석해야 한다. 또한, 다음 테스트에 이전 테스트의 결과가 영향을 미치지 않도록 종료할 수 있어야 한다. 본 논문에서는 안드로이드 디바이스를 인텐트로 자동화된 테스트를 하기 위해서 ADB를 사용한다. ADB는 PC와 안드로이드 디바이스를 연결할 수 있는 통로로 인텐트를 PC에서 디바이스로 전달할 수 있고, 디바이스에서 실행되고 있는 어플리케이션을 강제 종료하는 기능이 있다. 또한, 디바이스에서 발생하는 모든 로그를 PC로 수집할 수 있다. 이 절에서는 툴 자동화를 위한 명령어 사용 방법에 대해서 살펴본다.

3.1 ADB를 이용해 PC에서 디바이스로 인텐트를 전달하는 방법

ADB는 커멘드 기반으로 다양한 옵션을 제공하고 있다. 인텐트를 전달하기 위해서는 우선 전달받을 대상 디바이스를 선택해야 한다. 대상 디바이스를 선택하는 방법은 -s 옵션이고 디바이스마다 고유한 시리얼 번호가 있다. 예를 들어, LG-F220K 스마트폰의 시리얼 번호는 5888e6a4이므로 -s 5888e6a4로 기술한다. 다음으로 컴포넌트의 유형과 컴포넌트의 위치를 지정한다. 컴포넌트 유형 선택은 shell 명령어를 사용한다. 액티비티는 shell am start, 브로드캐스트 리시버는 shell am broadcast, 서비스는 shell

am startservice로 기술한다. 대상 컴포넌트 위치는 -n 패키지 이름/대상컴포넌트 이름으로 지정할 수 있다. 예를 들어, 안드로이드 샘플인 NoteEditor의 패키지는 com.example.android.notepad이고, 컴포넌트 클래스는 해당 패키지의 NoteEditor 클래스라 가정하면, -n com.example.android.notepad/.NoteEditor이 된다. 이 명령어들을 종합해보면 adb -s 5888e6a4 shell am start -n com.example.android.notepad/.NoteEditor이다. 이제 인텐트만 기술하면 인텐트를 전달할 수 있다. 예를 들어, 액티비티 컴포넌트 com.example.android.notepad/.NoteEditor는 Action이 android.intent.action.EDIT이고 Data가 content://com.google.provider.NotePad/notes/1인 인텐트를 요구한다고 가정하자. ADB 명령어는 아래와 같다.

```
adb -s 5888e6a4 shell am start -n com.example.android.
   notepad/.NoteEditor -a android.intent.action.EDIT -d
   content://com.google.provider.NotePad/notes/1
```

ADB로 인텐트를 생성하여 전달하는 인텐트 관련 인수[12]는 표 5.1에서 자세히 살펴볼 수 있다.

3.2 디바이스에서 실행되고 있는 어플리케이션을 강제 종료하는 방법

다음 테스트를 위해서 이전에 실행된 어플리케이션을 ADB를 이용해 강제로 종료하는 방법을 살펴본다. 이전에 인텐트로 실행된 어플리케이션을 종료하지 않으면 이전 실행 결과가 다음 테스트의 결과에 영향을 미칠 수 있으므로 강제 종료하는 방법은 자동화를 하기 위해 중요한 기능이다. 강제 종료는 ADB의 shell am force-stop 명령어를 사용한다. 기술하는 방법은 adb shell am force-stop 패키지 이름 형식으로 기술한다. 예를 들어, 안드로이드의 샘플 어플리케이션인 notepad 어플리케이션을 강제 종료하려면 notepad의 패키지 이름인 com.example.android.notepad를 사용해 종료시킨다. notepad를 종료시키는 명령어는 다음과 같다.

```
adb -s 5888e6a4 shell am
     force-stop com.example.android.notepad
```

명령어를 전달받은 디바이스는 강제로 notepad 어플리케이션을 종료시킬 것이다. 본 논문에서 다음 인텐트로 컴포넌트를 테스트하기 위해서 명령어를 전달 후 2초의

표 5.1: ADB 인수표

인텐트 인수	설명
−a (ACTION)	Action을 지정한다. −a android.intent.action.EDIT
−d (DATA_URI)	Data를 URI 형태로 지정한다. −d content://contacts/people/1
−t (MIME_TYPE)	Type을 지정한다. −t image/png
−c (CATEGORY)	Catetory를 지정한다. −c android.intent.category. APP_CONTACTS
−n (COMPONENT)	인텐트를 전달받을 대상 컴포넌트를 지정한다. −n com.example.android. notepad/.NoteEditor
−f (FLAGS)	Flag를 지정한다. −f 268435456
−−esn (EXTRA_KEY)	Extra를 지정한다. 키의 값은 null이다. −−esn key
−e|−−es (EXTRA_KEY) (EXTRA_STRING_VALUE)	Extra로 String을 지정한다. −−es key “value”
−−ez (EXTRA_KEY) (EXTRA_BOOLEAN_VALUE)	Extra로 Bool을 지정한다.
−−ei (EXTRA_KEY) (EXTRA_INT_VALUE)	Extra로 int를 지정한다.
−−el (EXTRA_KEY) (EXTRA_LONG_VALUE)	Extra로 long을 지정한다.
−−ef (EXTRA_KEY) (EXTRA_FLOAT_VALUE)	Extra로 float를 지정한다.
−−eu (EXTRA_KEY) (EXTRA_URI_VALUE)	Extra로 URI를 지정한다.
−−ecn (EXTRA_KEY) (EXTRA_COMPONENT_NAME_VALUE)	Extra로 Component를 지정한다.
−−eia (EXTRA_KEY) (EXTRA_INT_VALUE) [,(EXTRA_INT_VALUE...]	Extra로 int 배열을 지정한다. −−eia key 1,2,3
−−ela (EXTRA_KEY) (EXTRA_LONG_VALUE) [,(EXTRA_LONG_VALUE...]	Extra로 long 배열을 지정한다.
−−efa (EXTRA_KEY) (EXTRA_FLOAT_VALUE) [,(EXTRA_FLOAT_VALUE...]	Extra로 float 배열을 지정한다.

여유 시간을 추가로 기다리게 구현했다.

3.3 디바이스에서 PC로 로그를 수집하는 방법

테스트 시 결과를 판단하려면 디바이스에서 발생하는 로그를 수집해야 한다. 본 논문에서 제안한 테스팅 방법은 테스트마다 자동으로 정상 실행과 정상 종료, 비정상 종료의 결과를 판단해 알린다. 그러므로 테스트 자동화를 위해서는 디바이스에서 발생하는 로그를 수집하는 것이 필요하다. ADB에서 대상 디바이스에서 발생하는 로그를 수집하는 방법과 대상 컴포넌트의 로그를 추적하는 방법에 관해서 설명한다.

다음은 ADB로 대상 디바이스의 로그를 수집하는 logcat 명령어이다.

```
adb -s 5888e6a4 logcat -v threadtime *:V
```

logcat -v는 로그를 수집하는 우선순위를 의미하고 threadtime의 의미는 로그 발생 시간 표시 여부를 의미한다. logcat이 제공하는 로그 수집 모드는 총 5가지로 모든 로그를 확인할 수 있는 Verbose 모드, 디버깅으로 발생하는 로그 이상을 확인할 수 있는 Debug모드, 정보로 표시된 로그 이상을 확인할 수 있는 Info 모드, 경고로 표시된 로그 이상을 확인할 수 있는 Warning모드, 에러로 표시된 로그를 확인할 수 있는 Error 모드이다. 본 논문에서는 디바이스의 모든 로그를 확인하기 위해서 verbose 모드를 사용하고 로그 발생 시간도 같이 수집했다. 다음은 verbose 모드로 수집된 로그의 예이다. 날짜와 시간, pid, uid, 로그의 우선순위, Log 순으로 출력된다.

```
04-21 22:24:31.360   945  1499 I ActivityManager: Start proc
    com.example.android.notepad for activity com.example.
   android.notepad/.NoteEditor: pid=22811 uid=10307 gids
   ={50307}
04-21 22:24:31.911   945   959 I ActivityManager: Displayed
   com.example.android.notepad/.NoteEditor: +561ms
04-21 22:24:31.911   945   959 I ActivityManager: Timeline:
   Activity_windows_visible id: ActivityRecord{4371e660 u0
   com.example.android.notepad/.NoteEditor t1116} time
   :255491635
04-21 22:24:31.961 22811 22811 I ActivityManager: Timeline:
   Activity_idle id: android.os.BinderProxy@42889fb0 time
   :255491687 id:com.example.android.notepad
```

테스트하는 대상 컴포넌트의 모든 로그를 추척하여 수집하기 위해서는 대상 컴포넌트가 인텐트로 실행되며 발생하는 로그를 확인해야 한다. 패키지 이름이나 컴포넌트 이름만으로 로그를 수집하면 대상 컴포넌트에서 발생하지만, 대상 컴포넌트의 이름이 기록 안된 로그는 수집할 수 없다. 패키지 이름과 컴포넌트 이름뿐 아니라 pid를 추적하여 수집해야만 대상 컴포넌트의 전체 로그를 수집할 수 있다. 다음은 인텐트로 컴포넌트가 실행되면 발생한 로그에서 pid를 확인하는 방법의 예이다.

```
04-21 22:24:31.360   945  1499 I ActivityManager: Start proc
    com.example.android.notepad for activity com.example.
   android.notepad/.NoteEditor: pid=22811 uid=10307 gids
   ={50307}
```

안드로이드 샘플인 NoteEditor 컴포넌트가 인텐트를 전달받아 실행되면서 발생한 로그에는 Start proc라는 로그가 발생하는 패턴이 있다. Start proc에는 패키지의 이름과 컴포넌트 이름 pid의 정보가 담겨있다. 이 로그를 이용해 pid의 번호를 저장하고 대상 컴포넌트의 로그를 수집할 때 pid가 22811인 모든 로그를 수집해야 한다. 이 방법을 이용하면 대상 컴포넌트에서 발생하는 모든 로그를 수집할 수 있다.

제 4 절 로그 분석 방법

본 논문에서 제안한 방법은 로그를 실시간으로 분석하여 결과 알린다. 테스트하면서 발생하는 로그를 분석해 정상 실행과 정상 종료, 비정상 종료의 결과를 판단하는 방법에 관해서 설명한다. 결과 판단은 대상 컴포넌트에서 발생하는 로그의 특정 패턴을 이용한다.

- 액티비티 컴포넌트

액티비티에서 발생하는 로그에서 정상 실행을 판단하는 기준은 Displayed 로그의 발생 여부로 판단한다. 예를 들어, 테스트 대상 컴포넌트에서 발생된 로그 중 Displayed 패턴을 찾을 수 있으면 이 결과는 정상 실행으로 판단한다. 다음은 Displayed가 발생한 로그이다.

```
04-21 22:24:31.911   945   959 I ActivityManager: Displayed
   : +561ms
```

액티비티에서 발생하는 로그에서 정상 종료로 판단하는 기준은 Finishing과 Force removing 로그 발생 여부로 판단한다. 예를 들어, finishing은 모든 처리가 끝나 액티비티가 종료될 때 발생한다. 이 경우 서비스가 진행되면서 아무런 문제 없이 모든 서비스를 진행하고 액티비티가 종료되었다는 의미이다. Force removing은 액티비티의 코드에서 정상적으로 종료하는 finish 메서드로 종료되었다는 의미이다. 이 경우도 개발자가 작성한 코드에서 의해서 종료된 것이므로 정상 종료로 판단한다. 다음은 Finishing과 Force removing이 발생한 로그이다.

```
04-21 22:31:06.606   945  1858 W ActivityManager: finishing
   activity

04-21 22:30:46.856   945  1893 W ActivityManager:
   Force removing ActivityRecord{4330bfd0 u0}: app died, no
   saved state
```

액티비티에서 발생하는 로그에서 비정상 종료를 판단하는 기준은 Force finishing 로그의 발생여부로 판단한다. 예를 들어, 테스트하는 액티비티에서 `java.lang.IllegalArgumentException`이 발생하면 안드로이드 시스템은 대상 액티비티를 강제 종료시키고 비정상 종료를 알리는 Force finishing 로그가 발생한다. 이 경우 잘못된 인텐트로 인한 취약점이 발생해서 대상 액티비티가 비정상적으로 종료된 것을 의미한다. 다음은 `java.lang.IllegalArgumentException`의 에러로 Force finishing이 발생한 로그이다.

```
04-21 22:36:01.583   945  1893 E ActivityManagerServiceEx:
   java.lang.RuntimeException: Unable to start activity
   ComponentInfo: java.lang.IllegalArgumentException:
   Unknown URL tel:123

04-21 22:32:51.748   945  1891 W ActivityManager:
     Force finishing activity
```

- 브로드캐스트 리시버 컴포넌트

브로드캐스트 리시버 컴포넌트에서 정상 실행을 판단하는 기준은 비정상으로 판단되는 로그의 발생 여부 따라 달라진다. 브로드캐스트 리시버는 정상적인 인텐트를 전달

받은 때 특정한 로그를 출력하지 않는다. 또한, 브로드캐스트 리시버는 정상 종료하는 메서드가 존재하지 않으므로 정상 종료는 불가능하다. 따라서 비정상 종료 로그인 has died 패턴이 출력되지 않으면 정상 실행으로 판단하고 로그에 has died 패턴이 나타나면 비정상 종료로 판단한다. 다음은 has died가 발생한 로그이다.

```
05-21 20:10:24.440  5070  5070 W System.err: java.lang.
   NullPointerException
05-21 20:10:24.460   940  1480 I ActivityManager: Process
   pid 5070 has died.
```

- 서비스 컴포넌트

서비스 컴포넌트의 경우에도 브로드캐스트 리시버 컴포넌트와 같이 정상 실행에는 특정 패턴의 로그가 발생하지 않는다. 서비스 컴포넌트에서 발생하는 로그로 정상 실행을 판단하는 기준은 정상 종료된 VM exiting with result code 패턴의 로그나 비정상 종료된 Shutting down VM 패턴이 발생하지 않은 경우이다.

서비스 컴포넌트에서 발생하는 로그로 정상 종료로 판단하는 기준은 VM exiting with result code 로그 발생 여부로 판단한다. VM exiting with result code 패턴의 로그는 개발자가 정상적인 메서드를 통해서 종료시 발생한다. 다음은 VM exiting with result code가 발생한 로그이다.

```
10-08 19:22:54.497: I/AndroidRuntime(8584):
   VM exiting with result code 0, cleanup skipped.
```

서비스 컴포넌트에서 발생하는 로그로 비정상 종료로 판단하는 기준은 Shutting down VM 로그 발생 여부로 판단한다. Shutting down VM 패턴의 로그는 잘못된 인텐트를 인해 서비스 컴포넌트가 비정상 종료된 경우이다. 다음은 Shutting down VM가 발생한 로그이다.

```
05-21 19:56:42.078 25570 25570 D AndroidRuntime:
   Shutting down VM
```

ABSTRACT

A Design and Implementation of Intent Specification for Robust Android Applications

Ko, Myung Pil
Dept. of Computer Science
The Graduate School
Yonsei University

In this thesis, we propose Intent specification framework which improves the vulnerability caused by intent in android. Intent is the message for Inter-Component Communication(ICC) between components. An android component delivers the intent to android system, and activates as a target component to deliver the required data by the target component. Comparing the component in android to a function, the intent can be regarded as the function argument. An android application can be stoped abnormally if the malformed intent is delivered, which does not follow what the component expects. This vulnerability is due to the fact that android system cannot decide whether it is a malformed intent. With this vulnerability, android application can be stoped abnormally, or even the android system can get rebooted.

In this thesis, to remedy the vulnerability caused by intent, the following are proposed : one defending method of Android components by checking the intent in runtime, and three testing methods discover out the vulnerability in runtime. Under the previous research work, it is hard to describe the clear shape of intent which the component expects, and it is necessary to change the code for testing due to the trickiness of changing the shape of intent. To solve these problems, we suggest intent

spec, which can describe the shape of intent. The intent spec allows intent shape to describe easily and it is easy to generate intents in various shapes. Applying the approach described above, we propose four different methods to improve the vulnerabilities that might be caused by intent. The first method is a defending method by which the system enables to compare the shape of intent described by intent spec and the intent delivered to the component in runtime so that it can identify the malformed intent and defend the system against it. The second one is a testing method which assists in testing the target component and recognizing the test results automatically by generating random intent corresponding to the description for target intent to be tested. The third one is for testing vulnerability with android application binary file, which doesn't have source code. We build random intent with extracted intent from android application binary file. Using this produced random intent, we suggest the target application testing method. The final one is an automated method that generates the random test cases for JUnit tests based on intent spec, which is unit test tool of Android studio.

The advantages of the proposed methods are that they can defend wrong intent directly and prevent the abnormal termination. Therefore, they can provide normal service and be able to run various ways of test as generating freely the intent which is based on intent spec. The test results demonstrated that the proposed methods are useful for defending and identifying the vulnerabilities caused by the intent in open source applications and commercial applications.

Key words : Android, Intent, Intent Specification, Vulnerable, Testing, Robustness