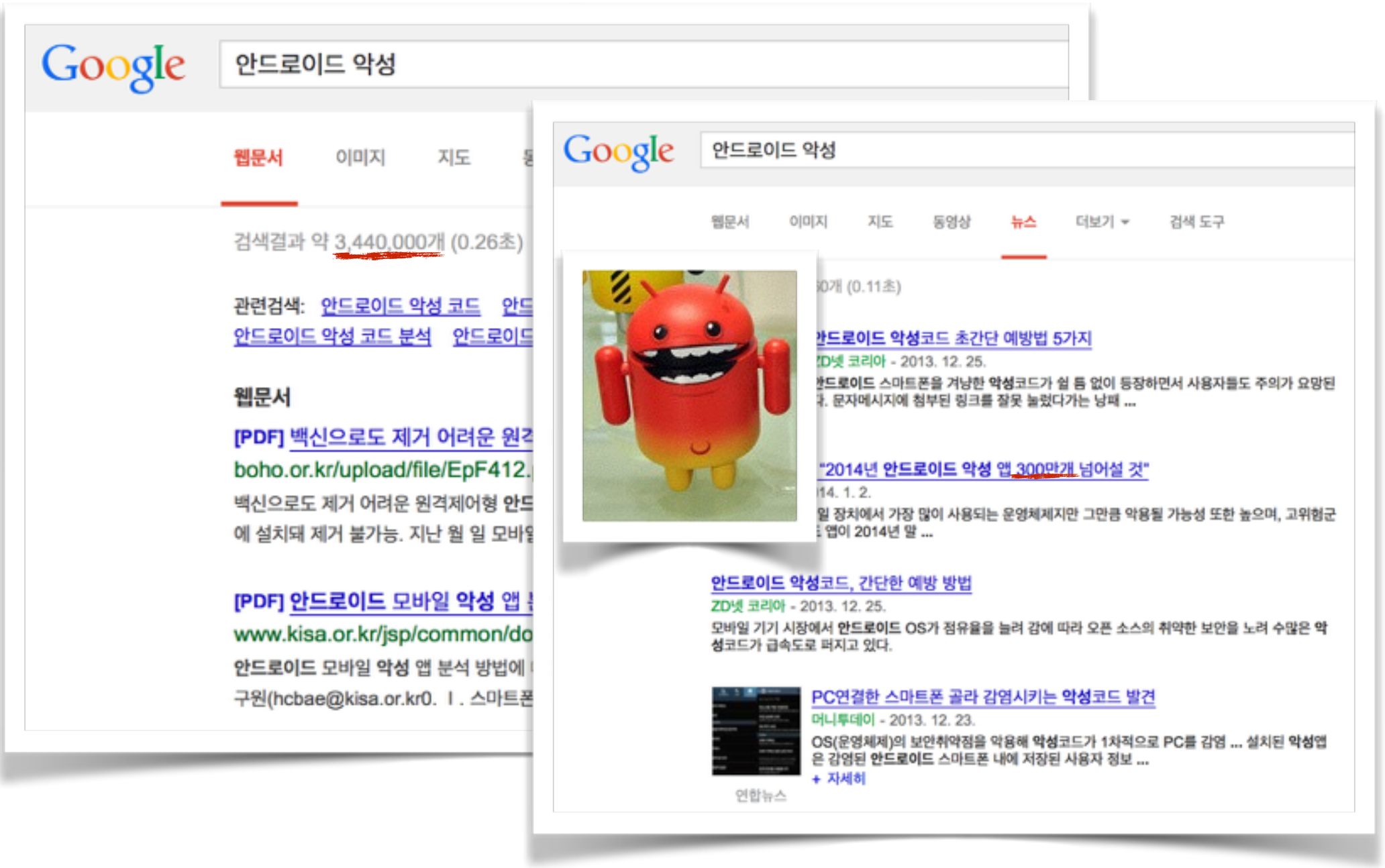
ScanDal: 안드로이드 앱 개인정보 누출 정적분석기

윤용호, 김진영, 이광근
서울대학교 프로그래밍연구실
ROSAEC Workshop
2014.01.15 @ 파주 지지향

요약

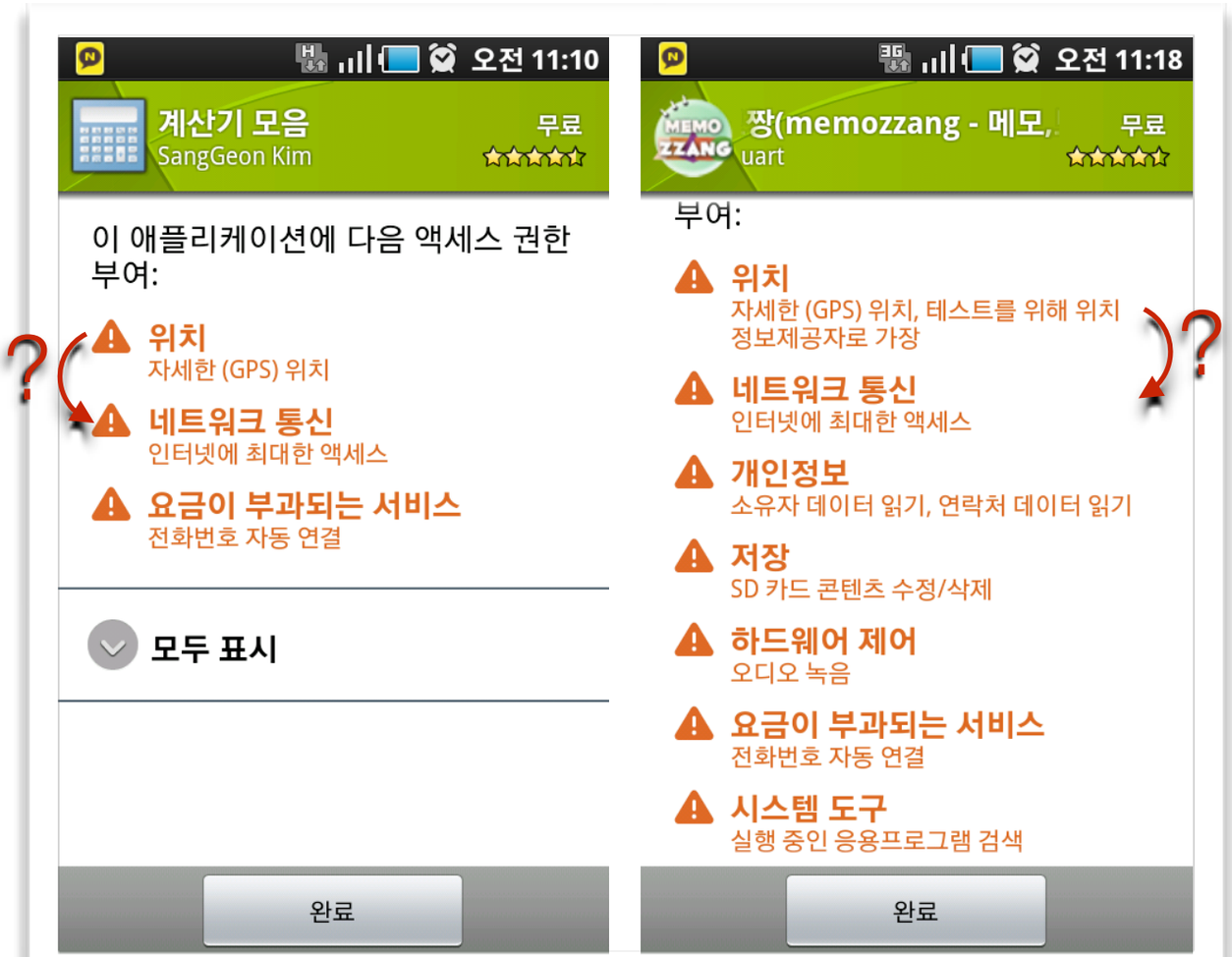
- 문제 상황 : 악성앱, 특히 개인정보 누출
- 해결책 : ScanDal
- 개발중에 부딪힌 각종 암(癌)초들
- 결론

악성앱이 미쳐 날뛰고 있습니다



왜 그리 심각해요?

- 안드로이드 OS 보안 모델의 부실함
  - 퍼미션 기반, 접근 여부만 알 수 있고 흐름은 알 수 없음
  - 앱 설치 시점에 허가하면 프리패스



ScanDal

- 안드로이드 앱의 Dalvik 바이트코드를
- 정적분석(Scan) 하여
- 개인정보 누출을 검출

개인정보 누출은

- 개인정보의 소스 API에서 리턴된 정보가
  - 위치정보, 전화번호, 기기 고유번호, etc.
- 정보를 내보내는 싱크 API에 인자로 전달될 때
  - 네트워크, SMS, 파일, etc.
- 중간에 여러 연산을 거칠 수 있음
  - 문자열 붙이기, 자르기, etc.

성능 : 갤럭시노트2 기본앱

- 기본앱 265개 중 11개 앱에서 누출 검출
- 13개 중 2개는 선택적 문맥 구분 분석으로 허위 경보 제거

앱	dex(KB)	분석 시간	메모리	누출
~~TstoreSideLoading~~	~~127~~	4	~~92~~	(제거)
~~V3Mobile~~	~~328~~	~~5~~	~~97~~	(제거)
FmmDS	255	5	114	기기번호
OMPDL	460	7	141	전화번호
AllSharePlay	770	15	275	파일
SPPPushClient	1015	20	165	기기번호
SyncmlDS	520	23	228	기기번호
11st	937	42	161	기기번호
GoogleTTS	1596	54	203	파일
TMap30	5827	1198	1469	전화번호,위치
GMS_Maps	5276	1369	1256	위치
SecMmsKor	4248	1541	1463	전화번호
AndroidTStore	3534	2510	7883	전화번호,기기번호

성능 : Google Play 마켓앱

- 안드로이드 공식 마켓 앱 250개 중 21개 앱에서 누출 검출

성능 : DroidBench*의 코너 케이스

- DroidBench : EC SPRIDE의 안드로이드 앱 개인정보 누출 분석 코너케이스 벤치마크

- 61개의 벤치마크 중
  - 허위 경보 7건
  - 놓치는 경보 9건(대부분 우리의 분석 범위 밖)

- DroidBench 제작 팀의 FlowDroid**에 비해 좋은 결과

* DroidBench : http://sseblog.ec-spride.de/tools/droidbench/
** FlowDroid : http://www.bodden.de/pubs/TUD-CS-2013-0113.pdf

안드로이드 앱
정적 분석의 어려움

- 분석 속도 및 정확도 문제
- API 라이브러리
- 가상 호출(+ API)
- 앱 실행모델(= API?)
- 리플렉션(= API?)
- 알람 분류의 어려움

분석 속도 향상

- 필요한 정보를 필요한 곳에 보내기
- String을 객체 대신 값으로 취급하기
  - final & imutable이기 때문에
- 복잡한 파티션 인덱스를 정수로 맵핑하기

분석 정확도 향상

- 선택적인 문맥 구분 분석
  - 속도를 희생하여 정확도를 높임
  - 인자에 개인정보가 묻어있을 경우와 아닌 경우로 호출 문맥 구분

- 인코딩 된 라이브러리를 항상 문맥 구분

API 라이브러리

- 정적분석기의 악몽
  - 이것만 없으면 거의 정확한 분석을…

- 요약 실행의미를 인코딩하여 분석기에 내장

- 인코딩 되지 않은 라이브러리를 만나면
  ‘적절하게(?)’ 분석

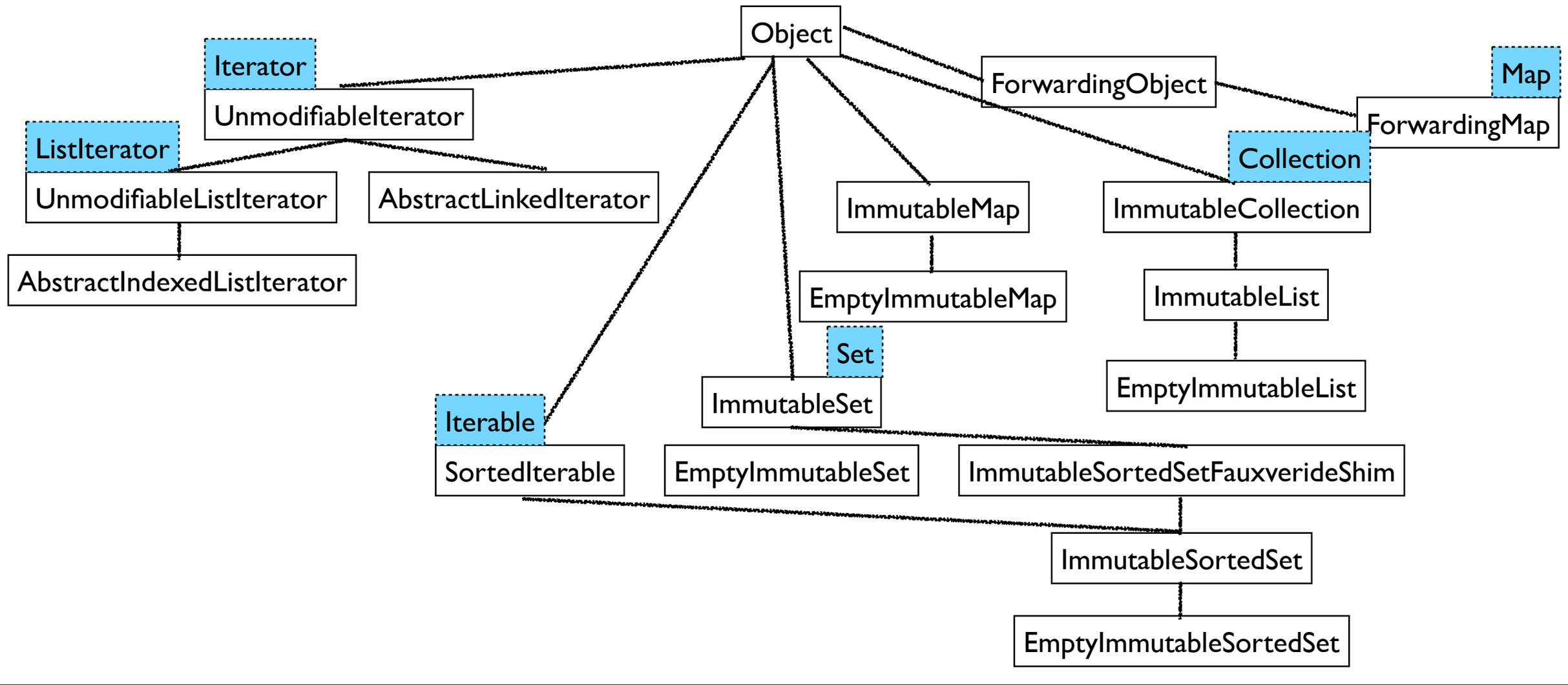
가상호출 + 라이브러리

- 경우에 따라 둘의 시너지가 성능에 엄청난 영향을 미침

앱	dex(KB)	기존(s)	개선 후(s)
SPPPushClient	1015	N/A	20
Books	3741	9385	282
ChatON_MARKET	4110	6550	728
Phonesky	3817	4819	1841
GMS_Maps	5276	N/A	1369

가상호출 + 라이브러리 예

- com.google.common.collect
- 229개 클래스, 29KLOC, 앱 코드 내에 포함됨



가상호출 + 라이브러리 예

- 앱 : GoogleServiceFramework

```
GservicesProvider$OverrideReceiver.onReceive(Intent i) {
  bundle = i.getExtras(); // 리턴 타입 : Bundle
  keys = bundle.keySet(); // 모르는 API, 리턴 타입 : Set

  it = keys.iterator();   // 타겟 : 모든 Set 클래스의 iterator
  while(it.hasNext()) {   // 타겟 : 모든 Iterator 클래스의 hasNext
    s = it.next();        // 타겟 : 모든 Iterator 클래스의 next
    ...
  }
```

가상호출 + 라이브러리 예

- 폭발!

```
it = keys.iterator();
```

```
s = it.next();
```

더 큰 문제.
타입 : Object

```
AbstractSet;.iterator
MapMakerInternalMap$EntrySet;.iterator
MapMakerInternalMap$KeySet;.iterator
ImmutableSet;.iterator
SingletonImmutableSet;.iterator
ImmutableSortedSet;.iterator
EmptyImmutableSet;.iterator
EmptyImmutableSortedSet;.iterator
RegularImmutableSortedSet;.iterator
RegularImmutableSortedSet;.iterator
ImmutableSet$ArrayImmutableSet;.iterator
ImmutableSet$TransformedImmutableSet;.iterator
Sets$1;.iterator
Sets$2;.iterator
Sets$3;.iterator
Sets$CartesianSet;.iterator
Sets$PowerSet$1$1;.iterator
Sets$PowerSet;.iterator
Sets$SetFromMap;.iterator
```

가상호출 + 라이브러리 예

- 폭발!

```
AbstractSet;.iterator
MapMakerInternalMap$EntrySet;.iterator
MapMakerInternalMap$KeySet;.iterator
```


```
it = keys
```

```
s = it
```

```
ator
;.iterator
```

```
Sets$3;.iterator
Sets$CartesianSet;.iterator
Sets$PowerSet$1$1;.iterator
Sets$PowerSet;.iterator
Sets$SetFromMap;.iterator
```

더 큰 문제.
타입 : Object

인코딩의 효과

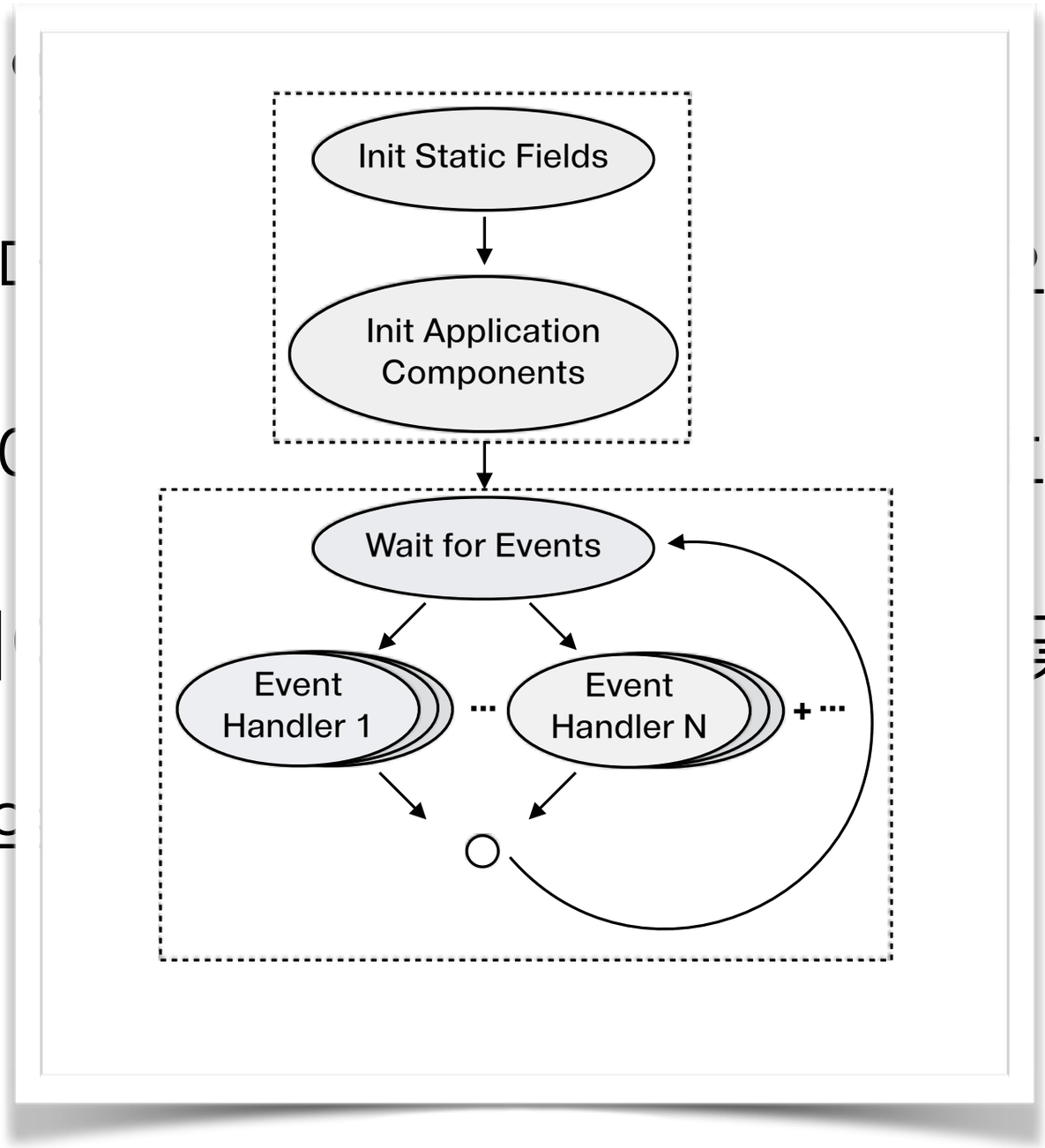
```
GservicesProvider$OverrideReceiver.onReceive(Intent i) {
  bundle = i.getExtras();
  keys = bundle.keySet(); // 이것만 인코딩 해도, 1시간이상 -> 1640초

  it = keys.iterator();
  while(it.hasNext()) {
    s = it.next();
    ...
  }
```

앱 실행 모델

- 안드로이드 앱에는 main 함수가 없음
  - 프로그래머는 각종 구성 요소들을 조립식으로 구현
  - OS가 알아서 정해진 순서대로 메소드를 호출
- 유저 인터페이스를 위해 온갖 리스너가 등록 및 호출됨
- 우리는 : 번역 단계에서 main 함수를 씌워줌

앱 실행 모델

- 안드로이드
  - 프로그래
  - OS가 알
- 유저 인터페
- 우리는 : 번



로 구현

출

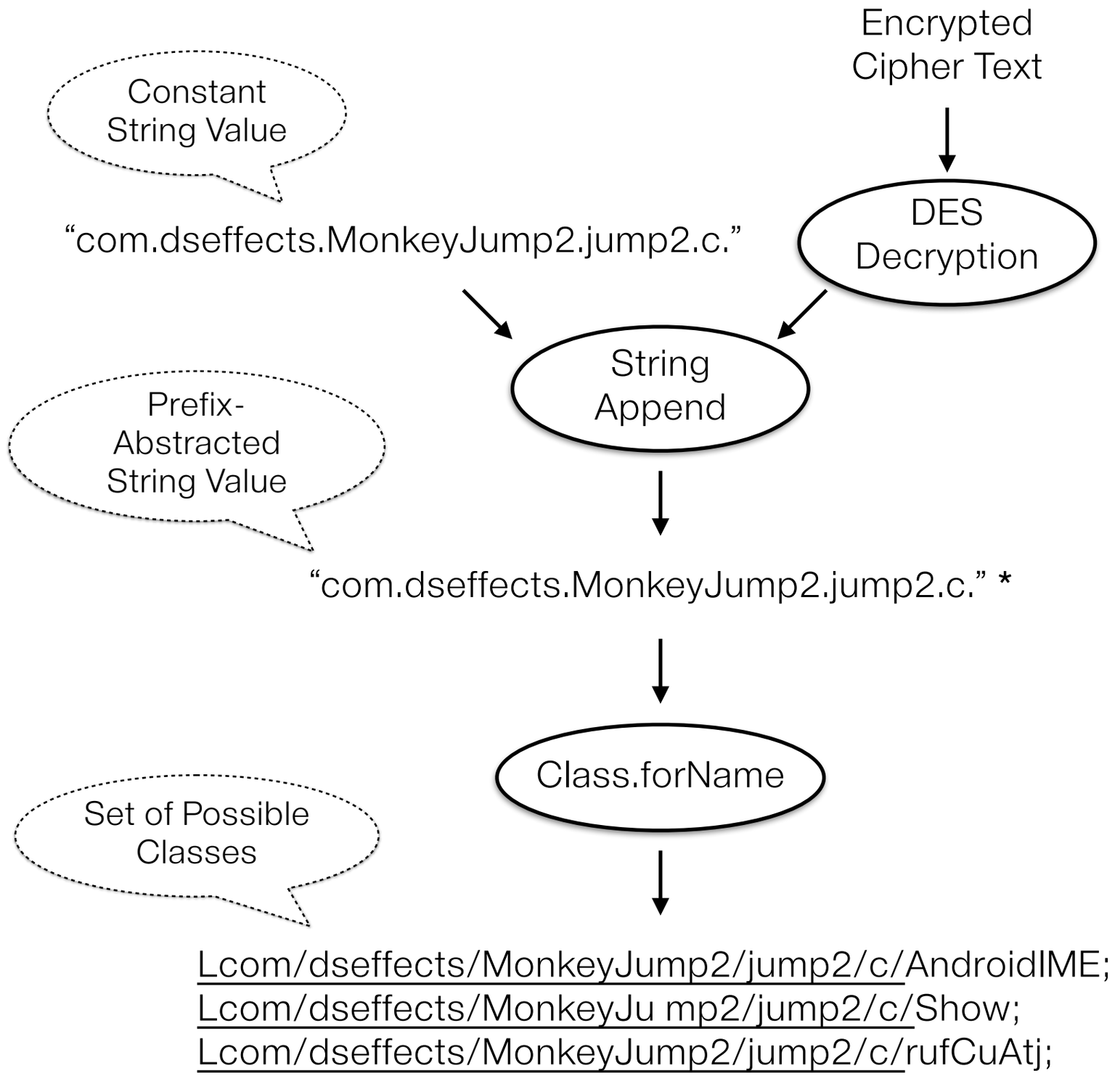
및 호출됨

리플렉션

- java.lang.reflect 패키지의 라이브러리
- 문자열로 클래스, 메소드, 필드 등에 접근

- 일반앱 : 구버전 API들에 접근하기 위해 주로 쓰임
  - 구글이 추천함, 문자열 상수를 주로 이용 -> 쉬움
- 악성앱 : 악성행동을 숨기기 위해 의도적으로 쓰임
  - 복잡한 문자열 연산을 거칠 수 있음

악성앱의 리플렉션 분석 예



알람 분류는요…?

- 가장 많이 받던 질문 :
  - “false alarm ratio가 얼마나 돼요?”
-
  -

알람 분류는요…?

- 가장 많이 받던 질문 :
  - “false alarm ratio가 얼마나 돼요?”
- 대답 :
  - “알람 분류가 힘들어서 잘 모르겠습니다…”

알람을 경로와 함께

- 알람 : 소스와 싱크의 쌍으로만
- 소스에서 싱크까지 값이 어떻게 전달되는지는 안알랴줌
- 경로를 함께 알려주면 도움이 되지 않을까?

알람을 경로와 함께

- 개인정보의 전달 경로 하나를 그래프의 형태로 보여줌
- 실제 실행중에 일어날 수 없는 실행 경로일 경우 다른 경로 탐색 시도 (사람의 입력 필요)
- 사람이 알람분류를 할 때 하는 일과 비슷한 과정
- 개선 작업중

결론

- ScanDal을 만들었고
- 만들다 보니 이런저런 어려움이 있었고
- 이렇게 저렇게 극복했더니
- 쓸만한 툴이 된 것 같다

각종 이미지 출처

- 구글 이미지 검색
    - 교학사 국사교과서류 인용방법

각종 이미지 출처

- https://www.google.co.kr/#newwindow=1&q=안드로이드+악성
- http://loved.pe.kr/entry/soribada-event
- http://thecolorcodedlyrics.wordpress.com/2013/04/29/shinee-why-so-serious-mv-version/
- http://tv.solidworks.co.kr/movie/list.php?id=mem
- http://arch7.net/210
- 김성모, “대털”
- http://mirror.enha.kr/wiki/가로쉬%20헬스크림
- 김성모, “태극기 펄럭이며”
- 우스타 쿄스케, “삐리리 불어봐 재규어”
- 요코야마 미츠테루, “전략삼국지”