난독화를 이겨내는
안드로이드 앱 정적 분석을 향해

김진영
서울대학교 ROPAS

ROSAEC 워크샵
2014년 7월 29일

안드로이드 앱 정적분석

- ScanDal
  - (지금은) 개인정보 누출 분석
  - 요약해석 기반
  - 이따가 마일스톤 (윤용호)

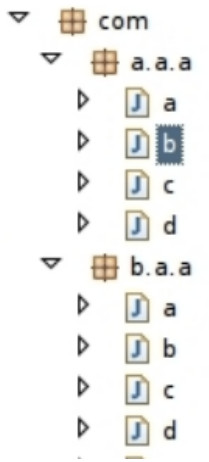
걸림돌과 대응들

- 안드로이드 실행주기와 리스너
  - 앱의 실행모델
- 빈번한 가상 호출
  - 클래스 분석
- 리플렉션
  - 문자열 접두사 분석
- 분석비용
  - Localization

새로운 걸림돌: 난독화

- 왜 쓰나?
  - 가독성 저하
  - 역공학 무력화
  - **하는 일 숨기기**
- 악성 앱에서 특히 매우 빈번

가독성 저하: 이름 바꾸기




```
1 public class a {
2  private String a( String ab )
3  { ... }
4  public void b( String ac )
5  { ... }
6 }
```

- 가장 흔히 쓰임
- 분석에는 큰 영향 없음

역공학 무력화:
바이트코드 수준 난독화

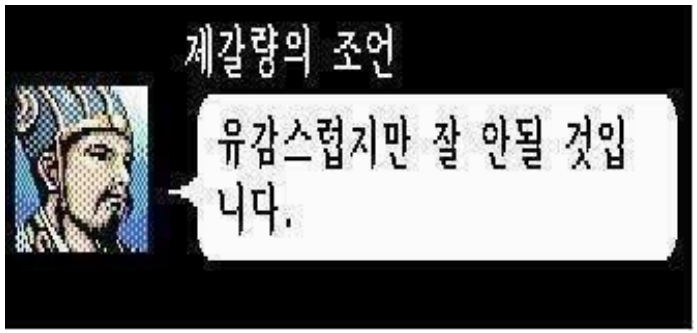
```
    8 : v0=0
    9 : v2=94
   11 : if v0<=v2 then jump 16
   13 : jump 37891 (* 존재하지 않는 주소로 점프 *)
   16 : ...
        ...
37889 : v4=1
37890 : v2[v3]=v4
37892 : v2=v10.JPFreq
```

- 실행흐름이 도달하지 않는 지점에 적법하지 않은 명령어
- Dalvik 수준에서 분석

하는 일 숨기기: 문자열 숨기기

- 숨기고 싶은 문자열을
- 여러가지 방법을 동원해 숨겨서
- 문자열을 통해 악성행동 달성

- 셋 다 잘 알아야 하지만
  - 현재 ScanDal로 정확한 정적분석이 어려움



숨기는 값

- URL (악성 서버)
- 접근하는 데이터 (변수, 리소스 이름)
- Class와 Method의 이름
- 등

어떻게 숨기나

- 문자열 테이블
  - String 배열, Map 등 collection 사용
- 인코딩-디코딩, 암호화-복호화 사용
  - Base64, DES, AES 라이브러리 사용
  - 키도 앱 어딘가에 숨겨져 있음
- 스트링 처리 루틴 내장

어떤 행동을 하나

- Reflection
  - 문자열을 알아야 어떤 메소드가 실행되고 어떤 값을 접근하는지 알 수 있음
- Class, method들을 문자열 Map으로 관리
  - Map의 문자열 키와 값을 알아야 함

예: SmsBoxer

- 코드

```
..
jdField_b_String = ... (“L1RodW1icy5kYg==”)
..
```

- Base64 디코딩을 거치면

```
“/Thumbs.db”
```

예: Geinimi

- 코드
  - 개인정보를 저장하는 Map의 key와 value를 DES 암호/복호화

```
c = Cipher.getInstance(“DES”)
...
return Cipher.doFinal([])
```

예: PJapps

- 코드

```
...
... = ... (“alfo3gsa3nfdsrfo3isd21d8a8”)
...
```

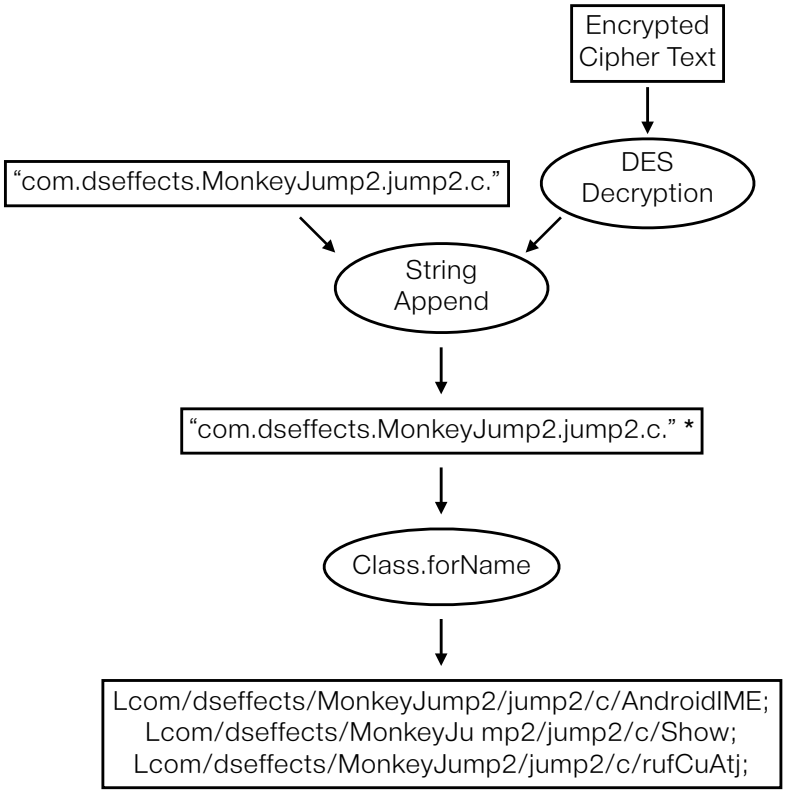
- 내장 루틴을 통해 짝수번째만 추림

```
“logandroid188” // alfo3gsa3nfdsrfo3isd21d8a8
```

- URL: logandroid188.com

예: MonkeyJump2

- Reflection으로 생성할 class의 이름을 DES로 복호화



지금 하는 일

- 문자열 난독화의 냄새를 맡는 분석기 제작중
  - 어떤 지점에서
  - 분석에 어떤 영향을 주고 있나
- 결과는:“A지점에서 생성된 문자열이 B지점에서 분석에 ~~한 걸림돌”

앞으로 할 일

- ScanDal이 난독화에 얼마나 지고 있는지 파악
- 난독화를 이기는 ScanDal 분석기 만들기
  - 필요한 지점의 문자열만 잘 분석하게

감사합니다

?