Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.4, Aug. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.4.657

머신러닝 기반의 자동화된 소스 싱크 분류 및 하이브리드 분석을 통한 개인정보 유출 탐지 방법*

심 현 석,[1†] 정 수 환[2‡]
[1,2]숭실대학교(대학원생, 교수)

Machine Learning Based Automated Source, Sink Categorization for Hybrid Approach of Privacy Leak Detection*

Hyunseok Shim,[1†] Souhwan Jung[2‡]
[1,2]Soongsil University(Graduate student, Professor)

요 약

안드로이드 프레임워크는 단 한번의 권한 허용을 통해 앱이 사용자의 정보를 자유롭게 이용할 수 있으며, 유출되는 데이터가 개인정보임을 식별하기 어렵다는 문제가 있다. 따라서 본 논문에서는 어플리케이션을 통해 유출되는 데이터를 분석하여, 해당 데이터가 실제로 개인정보에 해당하는 것인지를 파악하는 기준을 제시한다. 이를 위해 우리는 제어 흐름 그래프를 기반으로 소스와 싱크를 추출하며, 소스에서 싱크까지의 흐름이 존재하는 경우 사용자의 개인정보를 유출하는지 확인한다. 이 과정에서 우리는 구글에서 제공하는 위험한 권한 정보를 기준으로 개인정보와 직결되는 소스와 싱크를 선별하며, 동적분석 툴을 통해 각 API에 대한 정보를 후킹한다. 후킹되는 데이터를 통해 사용자는 해당 어플리케이션이 실제로 개인정보를 유출한다면 어떤 개인정보를 유출하는지 여부를 파악할 수 있다. 우리는 툴을 최신 버전의 API에 적용하기 위해 머신러닝을 통해 최신 버전의 안드로이드의 소스와 싱크를 분류하였으며, 이를 통해 86%의 정확도로 최신 배포 버전인 9.0 안드로이드의 API를 분류하였다. 또한 툴은 2,802개의 APK를 통해 평가되었으며, 개인정보를 유출하는 850개의 APK를 탐지하였다.

ABSTRACT

The Android framework allows apps to take full advantage of personal information through granting single permission, and does not determine whether the data being leaked is actual personal information. To solve these problems, we propose a tool with static/dynamic analysis. The tool analyzes the Source and Sink used by the target app, to provide users with information on what personal information it used. To achieve this, we extracted the Source and Sink through Control Flow Graph and make sure that it leaks the user's privacy when there is a Source-to-Sink flow. We also used the sensitive permission information provided by Google to obtain information from the sensitive API corresponding to Source and Sink. Finally, our dynamic analysis tool runs the app and hooks information from each sensitive API. In the hooked data, we got information about whether user's personal information is leaked through this app, and delivered to user. In this process, an automated Source/Sink classification model was applied to collect latest Source/Sink information, and the we categorized latest release version of Android(9.0) with 88.5% accuracy. We evaluated our tool on 2,802 APKs, and found 850 APKs that leak personal information.

Keywords: Android application, Flow analysis, Malware detection, Privacy leak

Received(05. 15. 2020), Modified(07. 07. 2020), Accepted(07. 07. 2020)

* 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2019-0-00477, 가상화된 신뢰실행환경을 이용한 안드로이드 보안 프레임워크 기술 개발)

† 주저자, ant_tree@naver.com
‡ 교신저자, souhwanj@ssu.ac.kr(Corresponding author)

I. 서 론

스마트폰에서 이용되는 운영체제, 특히 안드로이드 운영체제의 시장 점유율은 2019년 기준 76.08%에 달한다 [1] . 안드로이드 마켓에서는 매일6,000개가 넘는 앱을 출시하고 있으며, 사용자들은 평균적으로 일일 10개[2]의 앱을 사용하고 있다. 안드로이드 앱은 헬스부터, 비즈니스, 금융에 이르기까지 거의 모든 영역에서 사용자와 밀접한 관계를 가지고 있으며[3], 사용자들은 일 평균 2.3시간을 앱 사용에 소모한다[4].

이러한 안드로이드 앱은 의도적으로, 혹은 의도치 않게 사용자의 개인정보에 접근하며 이를 유출시킨다. 사용자는 이러한 개인정보가 얼마나 민감한 것들인지에 대한 자각 없이 개인정보의 유출을 허용하는데, 유출되는 개인정보는 사용자의 휴대폰 번호부터, 개인적인 문자메시지까지 이른다. 안드로이드는 이러한 유출을 방지하기 위해 권한 프레임워크를 사용한다. 권한은 앱을 시작할 때, 혹은 앱을 구동하는 동안에 요청되지만, 그러한 요청은 단 한번만 이루어진다. 따라서 앱은 단 한번의 권한 허용을 통해 앱은 사용자의 정보를 마음껏 이용할 수 있다. 또한 사용자에게 노출되는 권한 요청은 앱이 어떤 정보를 이용하는지에 대해 자세한 정보를 제공하지 않는다. 이러한 문제를 해결하기 위해 우리는 논문과 같은 툴을 제안한다.

우리는 제어 흐름 그래프를 통해 앱에서 사용되는 소스와 싱크를 추출하며, 더불어 소스가 싱크로 연결되는지 확인한다. 연결된 흐름에서 위험한 API가 이용된다면 이는 사용자의 데이터를 유출하는 것으로 판단하며, 동적 분석을 통해 유출되는 정보에 대한 내용을 함께 분석한다. 또한, 분석의 결과로 앱 내에서 이용되는 민감한 API들을 사용자에게 친숙한 언어로 해석하여 사용자에게 알려준다.

한편, 안드로이드는 빠른 주기로 지속적인 업데이트가 이루어져 왔으며, 매 업데이트마다 API에 다양한 변화가 이루어져 왔다. 안드로이드 API에서 소스와 싱크를 분류하는 과정은 이러한 안드로이드 버전에 의존한다. 기존에는 안드로이드 API 마다의 특징에 따른 자동분류가 가능한 모델이 존재하였으나 최신 API에 대한 분류와 검증이 필요하다.

논문에서 기여한 바는 다음과 같다.

- 상위 버전의 안드로이드 API들에 대해 소스 및 싱크 를 분류하는 방법을 구현하고, 평가하였다.
- 제안된 툴을 이용하여 각 소스와 싱크 중 어떤 API들이 위험한 것인지 판단하는 방법을 고안하였다.
- 정적/동적 분석을 통해 개인정보가 유출되는 근거와, 그 근거에 대해 실제로 데이터가 빠져나가는 것을 확인하였다.
- 각 소스와 싱크를 통해 실제로 어떤 개인정보가 유출되는지 사용자 친화적인 언어로 변환하는 작업을 수행하였다.

논문은 다음 장부터 배경지식을 설명하며, 3장에서 기존의 관련 연구들에 대해 설명할 것이다. 이후 4장에서 논문에서의 툴을 구현하는 방법을 소개하며, 5장에서 툴에 대한 평가가 이루어진다. 논문은 6장에서 한계점에 대해 설명하고, 마지막으로 7장에서 결론을 통해 끝맺는다.

II. 배경지식

2.1 안드로이드 API

안드로이드 플랫폼에서는 SDK를 통해 개발된 사용자 어플리케이션의 버전과 시스템 버전 등의 코드를 API 레벨을 통해 관리하며 2008년에 출시된 안드로이드 버전 1부터 현재까지 총 29개의 API레벨이 개발, 배포되었다. 안드로이드는 빠른 개발, 배포 흐름을 보이고 있으며, API 버전의 변화에 따른 보안정책의 변경과 API의 추가, 변경, 삭제는 매 버전마다 발생한다.

이러한 안드로이드 버전에 따른 API 변경은 안드로이드 SDK의 *api-versions.xml* 파일에 기록되며, 따라서 해당 파일의 정보를 이용하여 변화 내용을 확인할 수 있다. Table 1은 API 레벨의 변화에 따른 클래스 및 메서드 수의 변화를 나타내고 있다. 이는 전체 API 개수인 39,268개에 대하여 API 레벨 14부터의 변화만 다룬 것으로, 평균 192개의 클래스와 533개의 메서드가 변화되었다. 따라서 하위 버전을 기준으로 상위 버전 API를 분석하는 경우에는 새로 추가된 API에 대해 분석이 불가능하며, 최신 버전의 API를 대상으로 하는 것도 업데이트 중에 더 이상 유지보수가 이루어지지 않거나 삭제된 API를 포함하기 때문에 부적절하다. 따라서 특정 버전을 기준으로 모든 안드로이드 API를 분석하는 것은 올바르지 않으며, 각 버전에 맞는 API 정보를 통해 분석하는 것이 올바르다.

Table 1. Added, deprecated and removed APIs for each version.

SDK Version Code	Class		Method	
	Since	Deprecated + removed	Since	Deprecated + removed
I	131	6	294 29	426
J	183	80	671	256
K	132	2	301	28
L	265	403	920	156
M	152	371	508	132
N	357	40	835	55
O	376	16	816	114
P	196	99	485	138
Q(29)	196	65	814	1546
Total	1988	1082	5673	2851

2.2 소스 & 싱크 (Source & Sink)

소스와 싱크는 taint 분석에서 처음 등장한 개념으로, 정보 흐름 분석 영역에서 꾸준히 다루어 온 개념이다. 둘은 각각 데이터 흐름에 따라 정의되며, 소스는 시스템 객체 등에서 데이터를 읽어오는 지점, 싱크는 데이터를 쓰는 지점으로 정의할 수 있다. 예를 들어 소스는 안드로이드에서 정의하는 *getter*와 같은 API에 해당하며, 싱크는 *setter*에 해당한다. **SuSi**[6]에서는 안드로이드에서 사용되는 소스와 싱크에 대해 다음과 같이 정의하였다.

> *Source : Sources are calls into resource methods returning non-constant values into the application code.*
> *Sink : Sinks are calls into resource methods accepting at least one non-constant data value from the application code as parameter, if and only if a new value is written or an existing one is overwritten on the resource.*

정리하자면, 소스에서 싱크로 이어지는 흐름은 잠재적인 정보 유출을 가지고 있다. 여러 연구에서 논의된 것처럼[7], [8], 개인정보 유출은 결국 이러한 소스에서 싱크로 이어지는 흐름을 찾는 것으로 결정된다.

III. 기존 연구의 문제점

3.1 상위 버전 API의 소스 싱크 분류 문제

SuSi는 안드로이드 API에 대해 소스와 싱크 함수를 정의하고 머신러닝을 이용해 자동화 된 분류 기술을 제안한 첫 연구이다. 논문에서는 779개의 API를 랜덤하게 선택하여 성능을 평가하였으며, 4.2 버전까지의 전체 API 12,600개에 대해 적용되었다. 해당 툴은 Github를 통해 공개되어 있는데, 실제로 툴을 실행해본 결과 논문에서 제시한 평균 92%의 정확도보다 5% 낮은 87%의 정확도를 확인할 수 있었다. 또한 해당 툴에서 사용된 훈련 데이터는 4.2 버전을 위한 것으로, 상위 버전에서 사용할 수 없다.

이후 발표된 **SWAN**[9]에서는 기존 **SuSi**에서 분류하던 소스 및 싱크로는 취약점 분석의 한계가 있어 Sanitizers, Authentication을 추가한 SRM(Security- Relevant Methods) List를 제안하였다. 그 중 Sanitizer는 취약점을 완화 할 수 있는 기능이 있는 함수로써 사용 될 수 있기 때문에 SRM 함수 목록에 포함된다. 반면 **SWAN**에서는 소스와 싱크에 대해 50개씩의 데이터만을 사용하여 학습하였기 때문에 성능이 제대로 파악되지 않는다는 문제가 있다.

3.2 개인정보 유출 패턴 식별 문제

안드로이드는 사용자의 개인정보와 직결된 데이터를 다루는 API를 포함하여 수 많은 API를 SDK의 형태로 제공한다. 반면 모든 API가 개인정보를 이용하지는 않으며, 개인정보를 이용하는 API들의 경우에도 어플리케이션 내부에서만 사용되며 외부로 유출되지 않는 경우가 존재한다. 따라서 이러한 API들 중 개인정보를 이용하는 API를 선별하는 것과, 그러한 API가 어떠한 개인정보를 유출하는지 파악하는 과정이 이루어져야 한다.

SDK를 통해 개발자에게 제공되며, 어플리케이션으로 구현되어 사용자 레벨에서 사용되는 API들은 Fig.1과 같이 나타난다. 해당 API는 SIPManager를 통해 특정 프로파일의 세션을 강제로 활성화하여, 사용자의 개인정보를 외부로 유출할 수 있는 소스의 역할을 수행한다. 이러한 API는 개발자에게는 친숙한 SIP 등의 기능을 가지고 있지만,

```
1  SipManager manager = SipManager.newInstance(this);
2  SipProfile.Builder builder;
3  builder = new SipProfile.Builder(username,domain);
4  builder.setPassword(password);
5  SipProfile me = builder.build();
   ...
6  manager.open(me.getUriString()); //Open SIP session
```

Fig. 1. Code fragment for SIPManager session open that acts as source.

사용자들은 이러한 SIPManager의 기능을 정확하게 파악하기 어렵다. 따라서 이러한 개인정보 유출 사례를 파악하여 사용자에게 알려주기 위해서는, 정확한 API 분류가 필요하며, 각 유출과 관련된 API에 대해 가능한 자세한 정보의 전달이 필요하다.

IV. 제안 구조 및 방법

4.1 제안 구조

우리가 논문에서 제안하는 툴은 전처리 단계와 분석 단계로 구성되어 있다. 전처리 단계는 기본적으로

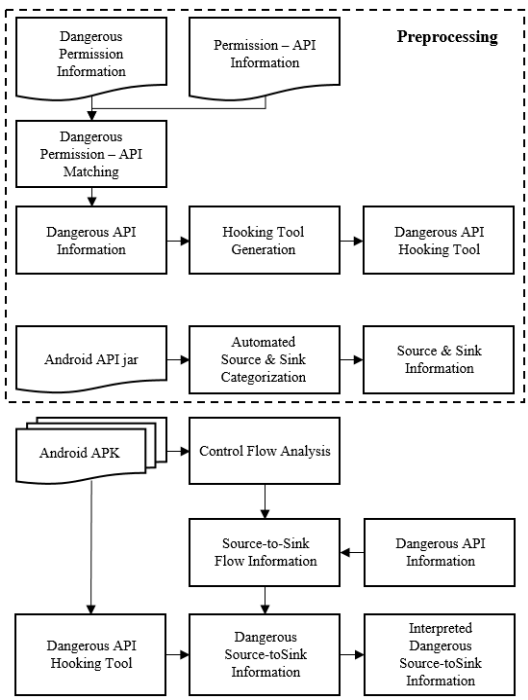


Fig. 2. Overall architecture (analysis pipeline)

분석할 API 리스트를 얻기 위해 데이터셋을 추출하고 파싱하는 것으로 이루어진다. Fig. 2에서 볼 수 있듯이 전처리 단계의 결과인 *Dangerous API Information*과 *Dangerous API Hooking Tool*은 분석 단계에서 사용된다.

위에서 언급했듯이, 우리는 전처리 단계에서 기존 데이터셋의 추출, 파싱 및 매칭으로부터 분석 데이터를 생성한다. 먼저 그림의 *Permission-API Information*과 *Dangerous Permission Information* 데이터셋의 매칭을 통해 *Dangerous API Information* 데이터셋을 생성한다. 이 생성된 데이터셋은 API 정보로, 데이터셋에 기록된 모든 API는 동적 분석의 API 후킹 대상에 해당한다. 분석 단계에서는 APK에 대해 *Control Flow Analysis* 및 *Dangerous API Hooking*의 두 가지 분석 프로세스를 수행한다. 전체 분석과정은 정적 분석과 동적 분석으로 나누어 결과를 어플리케이션에 전송한다. 분석 어플리케이션은 정적 및 동적 결과를 파싱하여 사용자에게 보고한다.

4.2 소스 및 싱크 분류

소스와 싱크의 데이터셋 확보를 위해 오픈소스로 공개된 **SuSi**와 **SWAN**을 활용하였다. 클래스 내부의 필드 값을 리턴한 경우, taint 분석을 위한 feature가 올바르게 설정되지 않는 경우가 발견되었다. 기존의 taint 된 변수 처리 방법을 확인한 결과, 오른쪽에 위치한 변수가 왼쪽에 있는 변수로 할당되는 경우에 taint 되었다고 간주하며, 해당 값이 파라미터로 전달받은 값인지만 확인하였다. 반면, taint 분석에서 파라미터가 전파되는 과정을 정확하게 파악하기 위해서는 실제로 해당 값이 사용되었는지 확인해야 한다. 이를 위해서 한 번에 한 라인에서의 데이터만 확인하는 것이 아닌, 함수 내에서 전체 할당되는 변수의 셋을 확인해야 한다.

추가된 API의 특징을 분류하고 함수 자체의 구현을 분석한 결과를 살펴보면 예를 들어, “get”의 특성을 가질 수 있는 함수들을 확인할 수 있다. “create”와 “load” 등 값을 받아 올 수 있도록 API들이 추가되었으며, 싱크의 특징이 될 수 있는 내용들 또한 함께 추가됨을 확인할 수 있었다. 따라서 상위 버전 API의 구조적인 특징을 적용하기 위해 **SWAN**에서 제공되는 feature 중, 상위 버전의 내용을 학습 할

수 있도록 feature를 선별하여 적용하였다. 선별 기준으로는 feature 중 Sanitizer, Authentication 함수만을 위해 사용되는 feature들을 제외하였으며, 소스와 싱크를 분류하는데 기여하는 모든 feature가 추가되었다. 따라서 기존의 **SuSi**에서 사용된 144개의 feature와 선별되어 추가한 50개의 feature를 포함하여 총 194개의 feature가 사용되었다. 추가된 feature는 Table 2를 통해 나타내었으며, 각각은 메서드가 구조체인지 여부를 참 혹은 거짓의 boolean 타입으로 반환하는 MethodIsConstructor feature를 제외하고 모두 특정 키워드를 포함하는지 여부로 이루어진다.

4.3 제어 흐름 그래프

2.2에서 에서 언급한대로, 개인정보 유출을 찾는 것은 바꿔 말하면 소스에서 싱크로 이어지는 흐름을 찾는 것과 같다. 소스는 이전 장에서에서 소개한대로 특정 데이터를 가져오는 지점이며, 싱크는 특정 데이터를 유출하는 지점이기 때문이다. 물론 이러한 데이터가 모두 개인정보라고 할 수는 없다. 따라서 논문에서는 제어 흐름 분석을 통해 소스에서 싱크로 이어지는 흐름을 찾은 뒤, 이후 이러한 흐름이 개인정보를 유출하는지 확인한다.

우리는 제안된 툴에서 소스에서 싱크로의 흐름을 찾기 위해 **FlowDroid**[10]를 활용하였다. **FlowDroid**는 데이터 흐름 분석과 제어 흐름 분석의 자동화를 위한 툴로, taint 분석을 통해 소스에서 싱크로 이어지는 흐름을 추적할 수 있다. 우리는 **FlowDroid**를 통해 소스에서 싱크로의 흐름을 찾고, 그 결과를 xml의 형태로 저장하였다. 저장된 소스에서 싱크로의 흐름은 개인정보를 포함한 데이터 유출의 가능성이 있는 것으로 파악하여, 각 민감한 API 데이터셋과 매칭하여 분석을 수행한다.

4.4. 민감한 API

4.4.1 권한 기반 민감한 API 선정

안드로이드 프레임워크는 단편화된 특징으로 인해, 비슷하거나 같은 기능을 수행하더라도 그 메서드와 클래스명이 다르다[11]. 개발자와 분석가들은 이 때문에 API의 시그니처만을 통해 어떤 데이터를 이용하는지 파악하는 것이 어렵다. 안드로이드 프레임

Table 2. Additional selected features. Features are selected from SWAN, which are source/sink related.

Feature group	Feature
MethodInvocationName	escap, replac, strip, match, encod, regex, check, verif, authori, authen, login, logout, security, credential, bind, connect, get, read, decod, unescap, load, request, creat, output, writ, set, updat, send, handl, put, log, run, execut, dump, print, pars, makedb, execute, saniti
MethodIsConstructor	True/False
ReturnTypeContains-NameFeature	Request Builder
SourceToReturnFeature	get read decode unescape load request create

워크는 권한을 기반으로 API의 사용을 제한하지만, API 마다 어떤 권한을 이용하는지에 대한 문서화된 정보를 제공하지 않는다[12]. 따라서 구글은 Table 3처럼 위험한 권한에 대해 문서화 된 정보[13]를 제공하지만, 어떤 API 사용이 위험한지에 대한 정보를 제공하지 않는다.

따라서 우리는 **Axplorer**[14]에서 연구한 결과를 이용하기로 한다. **Axplorer**는 안드로이드 API와 권한 간 매칭을 수행한 프로젝트로, 각 SDK 레벨에 따른 권한과 그 권한을 이용하는 API를 정리하였다. 따라서 우리는 구글에서 공개한 위험한 권한과, 그 권한을 이용하는 API의 목록을 매칭하여 선별을 수행하였다. 우리는 정리된 결과와 더불어, TelephonyManager 클래스에서 누락된 2개의 API를 추가하였다. 해당 API는 구글에서 제공하는 위험한 권한에 의존하지는 않지만, 대부분의 API가 위험한 권한에 의존하는 TelephonyManager 클래

Table 3. Dangerous permissions provided by Google.

CALENDAR	READ_CALENDAR
	WRITE_CALENDAR
CALL_LOG	READ_CALL_LOG
	WRITE_CALL_LOG
	PROCESS_OUTGOING_CALLS
CAMERA	CAMERA
CONTACTS	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
MICROPHONE	REACORD_AUDIO
PHONE	READ_PHONE_STATE
	READ_PHONE_NUMBERS
	CALL_PHONE
	ANSWER_PHONE_CALLS
	ADD_VOICEMAIL
	USE_SIP
SENSORS	BODE_SENSORS
SMS	SEND_SMS
	RECEIVE_SMS
	READ_SMS
	RECEIVE_WAP_PUSH
	RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE
	WRITE_EXTERNAL_STORAGE

스 내에 위치한다. 최종적으로는 선별된 2개의 API를 추가하여 총 68개의 API 리스트를 확보하였다. 확보된 리스트는 json의 형태로 저장되며, 각 SDK 버전에 혹은 용도에 따라 파라미터 등이 다른 경우에는 메서드 이름이 같더라도 다른 API로 간주하였다.

4.4.2 API 정보의 사용자 친화적 변환

우리는 권한에 기반한 위험한 API들의 목록을 확보하였으며, 우리의 궁극적인 목표는 이러한 API를 통해 어떤 데이터가 유출되는지 보고하는 것이다. 사용자에게 데이터 유출에 대해 보고하는 것은 개발자 친화적인 용어가 아닌 사용자 친화적인 언어를 사용해야 한다. 따라서 이 과정에서 API를 사용자 친화적 언어로 변환하는 것이 필요하다.

따라서 우리는 API들에 대해 수동으로 분석하여 어떠한 데이터가 유출되는지 확인한다. 이 과정에서 필요에 따라 각 메서드에서 다른 메서드로 통하는 흐름 또한 분석한다. 분석은 각 API들에 대해 Google의 문서와 AOSP의 소스코드[15]를 이용하여 수동으로 수행하였다.

분석 과정에서 일부 리스너 등록, 추가와 같은 악성행위와 연관성이 적은 API를 제외하였다. 이러한 API는 앱 내에서 다른 API를 호출할 때 해당 API의 호출을 보조하는 역할을 한다. 따라서 주된 기능을 수행하는 API에 집중하여 데이터를 유출하는 행위를 분석하였다.

4.5 동적 분석

어플리케이션은 개발 단계에서 테스트를 위해 삽입된, 실제로는 동작하지 하지 않는 코드가 존재하거나, 혹은 종종 동적 로딩 등의 방법을 통해 정적 분석 환경에서는 발견되지 않는 코드를 포함하고 있다. 따라서 이러한 코드의 실제 수행 가능 여부를 파악하기 위해 동적 실행을 통한 확인 과정이 필요하므로, 정적 분석과 더불어 동적 분석을 추가하여 하이브리드 분석을 수행한다. 그 중 동적 분석 과정을 위해 **Xposed**[16]를 이용하여 써드파티 앱에서의 데이터를 후킹한다. **Xposed**를 이용할 때 메서드의 후킹은 해당 메서드의 클래스, 파라미터 값이 정확히 일치해야 가능하다. Fig. 3는 이러한 과정을 나타내는

```
1  new HookUtil().createInstance(SipManager.class)
2    .addClassLoader(loadPackageParam.classLoader)
3    .minimum(9) //minimum SDK definition
4    .hook("isOpened", String.class) //method name
5    .setOnHookListener((param, methodName) -> {
6      String localProfileUri = param.args[0].toString();
7      String returnValue = (param.getResult() == null)
8      ? ""
9      : param.getResult().toString(); //parameter hook
10     Map<String, String> content = new
HashMap<>(); //content as key-value formed data
       …
11     content.put("localProfileUri", localProfileUri);
12     content.put("returnValue", returnValue);
13     new Util().log(methodName, content);
14   });
```

Fig. 3. Dynamic hooking code fragment for API parameter/return value logging.

데, *content*는 각 파라미터로 전달된 값과 리턴 값을 담게 된다. 따라서 우리는 Google Developer에 공개된 정보를 이용하여 이러한 기능을 구현한다. 또한 후킹되는 메서드의 파라미터 중 *Callback* 혹은 *Listener*, *Handler*와 같은 정보는 그 자체만으로는 의미를 갖지 않으며, 내부의 데이터는 리턴으로 전달되므로, 후킹 대상에서 제외하였다.

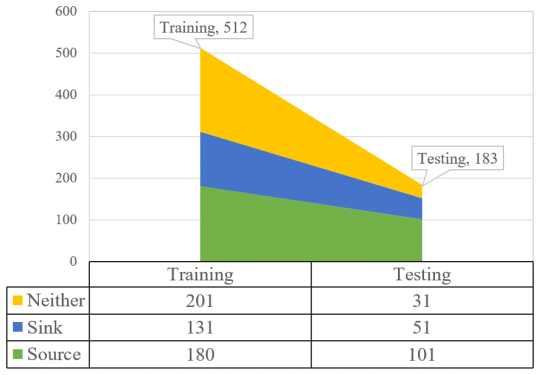
후킹되는 메서드의 파라미터, 리턴 값은 정규식으로 구분할 수 있도록 특별한 구분자(~!13#$)와 같은, 정규식 패턴에 해당하지 않지만 쉽게 구분 가능한 문자열로 연결된다. 이러한 정보는 로그캣을 통해 출력되며, 분석이 종료되면 로그를 수집하여 전달한다.

V. 성능 평가

5.1 소스 및 싱크 분류 성능 평가

논문에서는 전처리 단계에서 소스 및 싱크에 대한 분류를 진행하며, 본 분석 단계에서는 실제 민감한 데이터 유출에 대한 정보를 파악한다. 이 과정에서 기존 **SuSi**에서 제공하는 모델을 통해 소스 및 싱크의 상위 버전 API 적용 가능 여부를 확인하였다. 이 과정에서는 4.2 버전의 API를 통해 학습된 모델을 4.4 버전부터 9.0 버전까지의 API 데이터셋에 적용하여 검증하였다.

그 결과 Table 4와 같은 성능을 확인할 수 있었는데, 소스의 경우 기존의 SuSi 모델과 거의 동일한 성능을 확인하였다. 반면, 싱크의 경우 기존의 **SuSi** 모델을 이용하여 학습한 모델로는 상위 버전의 API를 평가할 수 없는 결과를 확인하였다. 결과가 발생한 원인을 분석해 본 결과, 상위 버전에서 싱크의 역할을 하는 API가 기존의 API와 다른 모습의 양상을 보였다고 예상하였다. 예를 들어 4.2 버전 이하에서 사용되는 싱크의 큰 특징인 "set" String이 start, clear, open, apply등으로 확장



	Training	Testing
Neither	201	31
Sink	131	51
Source	180	101

Fig. 4. Dataset portion for training and testing

된 것을 확인할 수 있었다.

상위 버전의 API에 대한 소스 및 싱크분류를 진행하기 위하여 구성된 데이터셋은 하위 버전의 API를 통해 학습을 진행하며, 상위 버전의 데이터셋에 대해 분류하도록 한다. 이는 여러 버전에 대응 가능한 일반화된 모델을 생성하기 위함이며, 생성된 모델은 상위 버전과 하위 버전의 API에 대해 분류가 가능하도록 한다. 따라서 모델은 SuSi에서 분류를 진행한 4.2 버전부터 7.1 버전의 API를 합하여 훈련되며, 8.1 버전과 9.0 버전에 대하여 평가가 이루어진다. 데이터셋의 분포는 Fig. 4와 같다.

모델의 학습은 SVM과 NaiveBayes, J48 세가지의 알고리즘을 통해 결과를 비교하였으며, 그 결과 NaiveBayes의 F-measure는 평균 0.83, J48은 평균 0.84로, SVM대비 최소 2%가량 낮은 것을 확인할 수 있었다. 따라서 모델은 SVM 알고리즘을 통하여 학습되었으며, feature set은 4.2에서 언급한 것과 같이 194개의 feature로 이루어진다. 결과는 Table 5와 같다.

가장 큰 변화가 나타난 것은 싱크의 분류 결과이다. 싱크의 경우 기존 **SuSi**로 분류 했을 때, 분류 성능이 상당히 낮았으며, 그 결과 최신 버전의 API

Table 4. Result of testing API version 4.4 - 9.0 with original SuSi model.

Class	TP Rate	FP Rate	Precision	F-Measure
Source	0.87	0.10	0.92	0.89
Sink	0.25	0.03	0.79	0.38
Neither	0.89	0.34	0.51	0.65
Weighted Avg	0.71	0.14	0.77	0.69

Table 5. Result of testing API version 8.1 - 9.0 with our approach.

Class	TP Rate	FP Rate	Precision	F-Measure
Source	0.85	0.03	0.98	0.91
Sink	0.84	0.07	0.84	0.84
Neither	0.90	0.11	0.64	0.75
Weighted Avg	0.86	0.05	0.88	0.86

Table 6. Result of static analysis with analysis time, memory consumption.

Dangerous Privacy Leakage Found (static, # per each APK)	Static (per each apk)							Total APK (#)
	Avg. Leakage Founed in FlowDroid(#)	Avg. Analysis time(s)	Avg. Max Memory Consumption (MB)	Avg. Source Used(#)	Avg. Sink Used(#)	Avg. Dangerous Source Used(#)	Avg. Dangerous Sink Used(#)	
0	0.75	1.92	9.05	3.28	11.52	0	0	1751
1-5	14.51	143.40	54.70	20.96	51.88	1.8	0.47	633
6-10	14.60	25.84	48.86	26.49	64.69	6.69	0.73	130
11-20	41.95	64.69	87.67	57.88	175.55	13.46	0.754	61
21-30	65.75	113.0	120.75	95.25	302.83	24.83	0.58	12
31 -	147.0	291.64	203.21	189.71	641.07	47.78	1.07	14

분류는 SuSi로 분류하기 어려웠다. 따라서 상위 버전의 API를 학습 데이터로 사용했으며, 그 경우에 최신 버전의 싱크 특징을 학습하여 분류 성능이 개선된 모습을 볼 수 있었다.

5.2 하이브리드 분석 성능 평가

평가를 위해 안드로이드 악성 프로그램 분석 플랫폼인 AMAaas에서 2,802개의 APK를 수집했다[17]. 2,802개의 APK 중 201개는 FlowDroid로 분석할 수 없으므로 나머지 APK에 대해 테스트를 진행하였다.

Table 6와 같이 1,751개의 APK는 정적 분석에서 민감한 개인정보 유출이 발생하지 않은 것을 확인할 수 있었다. 이러한 경우는 소스에서 싱크까지의 흐름은 존재하지만 민감한 개인정보 유출은 발생하지 않은 경우이다. 평균 4개의 어플리케이션에서 평균 3번의 소스에서 싱크까지의 흐름이 발견되었지만 민감한 개인정보 유출은 확인되지 않았다.

Table 7. Result of dynamic analysis per static analysis result.

Dangerous Privacy Leakage Found (static, # per each APK)	Dynamic (per each apk)		Total APK (#)
	Avg. Actual Leakage Found(#)	Avg. Analysis time(s)	
0	0	4.07	1751
1-5	6.07	4.11	633
6-10	5.16	4.06	130
11-20	7.08	4.08	61
21-30	9.67	4.14	12
31 -	8.0	4.05	14

한번도 민감한 개인정보 유출이 발생하지 않은 그룹을 제외하면 1-5번의 개인정보 유출이 발생한 그룹이 가장 지배적인 결과로 확인되었다. 이 경우 다른 그룹에 비해 평균 분석 시간이 긴 것으로 확인된다. 이는 분석을 위한 시간과 메모리 소비는 소스와 싱크의 개수보다는 APK의 복잡성에 의존하기 때문이다. 따라서 633개의 APK로 구성된 해당 그룹은 2초부터 10분까지의 분석 시간 분포를 가지기 때문

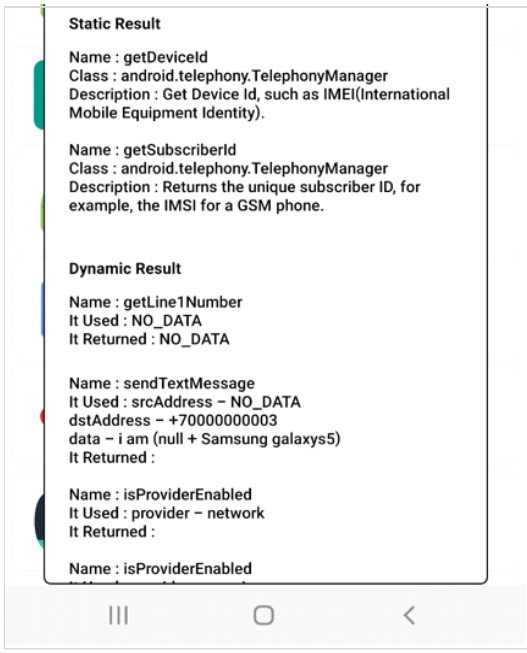


Fig. 5. Real-world application evaluation sample

에 높은 평균값을 가진다. 그 외 6개 이상의 개인정보 유출을 가지는 다른 그룹들은 각 그룹의 개인정보 유출의 증가에 따른 전체 데이터의 점진적인 증가를 보여준다.

Table 7은 동적 분석을 통한 결과를 나타낸다. 동적 분석을 위해서는 monkey 전략을 입력 생성 전략으로 사용하였으며, 각 어플리케이션 당 3분 동안 테스트를 진행하였다. 각 입력 시간이 동일하기 때문에 동적 분석에 소모되는 모든 분석 시간은 서로 거의 같다. 또한 동적 분석에서 발견된 실제 누출의 수는 정적 분석에서 나타난 결과보다 약간 낮으며 이는 모든 실행 경로에 도달할 수 없는 동적 분석의 코드 커버리지 문제점으로 인해 나타난다. 또한, 중복된 메소드 호출은 몇 번이 호출되어도 하나로 계산하는 것도 영향을 미친다.

Fig. 5는 이러한 어플리케이션을 통해 실제로 평가를 진행한 결과로, 특정 API에 대한 설명과, 어떤 값이 유출되는지 정보를 출력한다.

VI. 결 론

논문에서는 하이브리드 분석을 통한 개인정보 유출 탐지와, 그에 필요한 소스 및 싱크 분류에 대한 연구를 수행하고 평가하였다. 안드로이드 어플리케이션은 사용 용도에 따라 소스와 싱크의 분포 양상이 많이 달라지기 때문에 실행 환경에서 실제로 개인정보를 유출하는지 여부를 파악할 필요가 있다. 이를 위해 소스와 싱크의 흐름과 관련된 위험한 API 정보에 관한 연구를 실시하고, APK에 대해서 정적/동적으로 분석하여 정보가 유출되는지를 확인했다. 또한, 수집된 2,802개의 APK 중 38%가 실제로 실행 시간에 정보를 유출한다는 것을 증명하였고, 어떤 데이터가 유출되고 있는지를 밝혀냈다.

우리가 제안한 시스템은 본인의 개인정보가 악성 어플리케이션에 의한 탈취를 우려하는 사용자에게 적합하다. 또한 우리의 시스템과 어플리케이션이 안드로이드 플랫폼의 시스템 어플리케이션으로 적용될 수 있다면, 사용자가 구글 플레이 등의 마켓에서 앱을 내려받는 즉시 분석 가능할 것이다.

향후 연구로는 더욱 정밀하게 선별된 API 정보로 평가할 계획이다. 현재는 68개의 위험한 API만 선별하였지만, 개인 정보 유출에 대한 더 많은 결과를 보기 위해 더 많은 API 정보를 수집할 수 있다.

References

[1] Statcounter, "Mobile Operating System Market Share Worldwide" https://gs.statcounter.com/os-market-share/mobile/worldwide, Last Accessed 22 Apr 2020.

[2] Statista, "Average number of new Android app releases per day from 3rd quarter 2016 to 1st quarter 2018" https://www.statista.com/statistics/276703/android-app-releases-worldwide/, Last Accessed 22 Apr 2020.

[3] Techcrunch, S"martphone owners are using 9 apps per day, 30 per month" https://techcrunch.com/2017/05/04/report-smartphone-owners-are-using-9-apps-per-day-30-per-month/, Last Accessed 22 Apr 2020.

[4] Statista, "Market reach of the most popular Android app categories worldwide as of June 2018" https://www.statista.com/statistics/200855/favourite-smartphone-app-categories-by-share-of-smartphone-users/, Last Accessed 22 Apr 2020.

[5] Comscore, "The 2017 U.S. Mobile App Report" https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report?cs_edgescape_cc=KR, Last Accessed 22 Apr 2020.

[6] Siegfried, R., Steven, A., Eric, B., "A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks," in Proc. Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS 2014), Feb. 2014.

[7] Shengqian, Y., Dacong, Y., Haowei, W., Yan, W., Atanas, R., "Static Control-Flow Analysis of User-Driven

Callbacks in Android Applications," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, pp. 89-99, May. 2015.

[8] Noriyuki, S., Tetsuo, K., Katsuhisa, M., "Detecting Invalid Layer Combinations Using Control-Flow Analysis for Android," COP'16 Proceedings of the 8th International Workshop on Context-Oriented Programming, pp. 27-32, July. 2016.

[9] Goran, P., Lisa Nguyen, Q., Eric, B., "Codebase-adaptive detection of security-relevant methods," Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 181-191, July. 2019.

[10] Steven, A., Siegfried, R., Christian, F., Eric, B., Alexandre, B., Jacques, K.,, et al., "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, Edinburgh, pp. 259-269, June. 2014.

[11] Lei Z., Zhemin Y., Yuyu H., Zhenyu Z., Zhiyun Q., Geng H., et al., "Invetter: Locating Insecure Input Validations in Android Services," CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1165-1178, Oct. 2018.

[12] Yan H., Weiqiang K., Deng D., Jun Y., "Method-Level Permission Analysis Based on Static Call Graph of Android Apps," 2018 5th International Conference on Dependable Systems and Their Applications (DSA), pp. 8-14, Sep. 2018.

[13] Google Developer, "Google Developer" https://developer.android.com/?hl=ko, Last Accessed 22 Apr 2020.

[14] Michael B., Sven B., Erik D., Patrick M., Damien O., Sebastian W., "On demystifying the android application framework: re-visiting android permission specification analysis," SEC'16 Proceedings of the 25th USENIX Conference on Security Symposium, Austin, pp. 1101-1118, Aug. 2016.

[15] AndroidXRef, "AndroidXRef" http://androidxref.com/, Last Accessed 22 Aug 2019

[16] Xposed Module Repository, "XPosed" https://dl-xda.xposed.info/framework/, Last Accessed 22 Apr 2020.

[17] Android Malware Analysis as a Service, "AMAaaS" https://AMAaaS.com, Last Accessed 22 Apr 2020.

〈저 자 소 개〉

심 현 석 (Hyunseok Shim) 학생회원
2019년 2월: 숭실대학교 전자정보공학과 졸업
2019년 3월~현재: 숭실대학교 정보통신공학과 석사과정
〈관심분야〉 AI 보안, 모바일 보안, 흐름 분석, 개인정보 보호

정 수 환 (Souhwan Jung) 종신회원
1985년 2월: 서울대학교 전자공학과 졸업
1987년 2월: 서울대학교 전자공학과 석사
1996년 6월: University of Washington 박사
1988년~1991년: 한국통신 전임 연구원
1997년~현재: 숭실대학교 전자정보공학부 교수
〈관심분야〉 AI 보안, 모바일 보안, 클라우드 보안, 네트워크 보안