Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.4, Aug. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.4.617

안드로이드 모바일 단말기를 위한 효율적인 악성앱 감지법*

이 혜 림,† 장 수 희, 윤 지 원‡
고려대학교 정보보호대학원

Efficient Malware Detector for Android Devices*

Hye Lim Lee,† Soohee Jang, Ji Won Yoon‡
Graduate School of Information Security, Korea University

요 약

스마트폰 사용이 급증하였고 스마트폰에 탑재되는 OS 중 안드로이드가 차지하는 비중이 가장 높아졌다. 그러나 오픈소스로 제공되는 안드로이드의 특성이 악의적인 사용자들에게 유용하게 사용되어 스마트폰 사용자들의 프라이버시를 위협하고 있다. 이 논문에서 우리는 안드로이드 앱에서 요구하는 권한 정보를 사용하여 효율적인 악성앱 감지법을 제안한다. 이를 위하여 주성분 분석과 kNN 분류자를 사용하였으며, 새로운 앱들의 특성들을 분류자에 실시간으로 반영하기 위한 incremental kNN 분류자를 제안한다. 또한 이 분류자들의 정확률을 측정하기 위하여 k-묶음 교차 검증법을 사용하였다. 실험에 사용된 실제 악성앱 샘플을 얻기 위하여 Contagio에 요청하였으며 이를 이용하여 분류자의 정확률을 측정하였다.

ABSTRACT

Smart phone usage has increased exponentially and open source based Android OS occupy significant market share. However, various malicious applications that use the characteristic of Android threaten users. In this paper, we construct an efficient malicious application detector by using the principle component analysis and the incremental k nearest neighbor algorithm, which consider an required permission, of Android applications. The cross validation is exploited in order to find a critical parameter of the algorithm. For the performance evaluation of our approach, we simulate a real data set of Contagio Mobile.

Keywords: Android Security; Malware Detection; Principal Component Analysis; kNN-Classification

I. 서 론

스마트폰이 등장한 이후로 스마트폰 시장이 빠르게 증가하고 있다. 가트너는 2013년 2분기에 스마트폰 판매량이 46.5 % 까지 증가하여 최초로 피처폰의 판매량을 넘어섰다고 보고했다[1]. 이처럼 스마트폰의 수요가 증가하면서 그 기능과 성능을 위한 기술들도 빠르게 성장해왔다. 과거에 전화 통화 또는 문자 메시지를 주고받는 것이 주요 기능이었던 휴대폰이 이제는 뉴스와 교통 정보를 제공하고, 연락처와 일정을 관리해주며 금융거래 기능까지 제공해 주게 되었다. 이로 인해 스마트폰에 사용자의 개인정보가 더 많이 담기게 되었고 스마트폰에 대한 사용자의 의존도 또한 높아졌다.

스마트폰에 사용되는 플랫폼에는 Android, iOS, Windows, Blackberry 등이 있다. 그 중에서 가장 많이 판매되고 사용되어지고 있는 플랫폼은 Android이다. Android의 가장 큰 장점은 오픈 소스라는 점이다. 앱 개발자들은 그들이 개발한 앱을 마켓에 쉽게

접수일(2014년 3월 17일), 수정일(2014년 7월 3일), 게재확정일(2014년 8월 6일)
* 본 연구는 미래창조과학부에 의해 마련된 한국연구재단의 신진연구자 지원 사업(NRF-2013R1A1A1012797)에 의해 지원되었으며 부분적으로 고려대학교 신임교원에 대한 특별연구비에 의해 지원되었습니다.
† 주저자, dream8933@naver.com
‡ 교신저자, jiwon_yoon@korea.ac.kr (Corresponding author)

올릴 수 있고 올리는 과정에서 제약을 거의 받지 않는다. 앱을 올릴 수 있는 마켓도 다양하며 마켓에 올라온 다른 개발자의 앱을 다운받아 간단한 역공학을 거치면 소스코드를 보는 것도 가능하다. 하지만 안드로이드는 이러한 장점들로 인해 악의적인 사용자들의 타겟이 되고 있으며 악성코드를 유포하는 도구로 사용되고 있다[2][3]. 스마트폰에 자신의 개인정보를 의존하고 있는 사용자에게는 치명적인 위협이 되었고 안드로이드 플랫폼에 대한 보안 대책이 시급해졌다. 이를 위한 많은 연구들이 진행 중이다. 어플리케이션을 분석하는 데에는 정적분석과 동적분석으로 나누어진다.

정적분석은 파일의 metadata와 세부적인 코드 자체를 분석하는 방법이다. 동적분석은 에뮬레이터나 장비를 사용하여 파일을 실행시켜보아 그 행위나 동작을 관찰하는 방법이다. Suleiman[4]은 안드로이드 앱으로부터 apk 파일을 추출한 뒤 분석을 위해 역공학을 수행했다. 또한 베이지안 분류를 사용하여 그 정확도를 측정하였다. Anubis는 안드로이드 어플리케이션을 포함한 악성 코드 파일의 분석 보고서를 제공해주는 툴이다[5]. 이 툴은 정적분석과 동적분석을 모두 사용하였으며 파일들의 특성과 정보를 잘 나타내고 있지만 8MB 이하의 파일들만을 업로드 할 수 있는 제약이 있다. 앱을 분석하기 위해서는 앱을 다운받아야 하며 정적분석을 위해서는 앱으로부터 apk 파일을 추출하는 과정이 필요하다. 또한 분석을 위해서는 많은 시간이 소요된다. 이러한 방법이 악성앱의 행동을 관찰하고 그 특징을 파악하는 데에는 도움이 되지만 매일 봇물처럼 쏟아지는 앱들 가운데에서 악성앱을 예방하기에는 많은 제약이 따른다.

이 논문에서는 앱을 다운받기 전의 정보를 사용하여 악성앱을 감지하는 방법을 소개한다. 우리의 실험결과는 87 %의 정확도를 가지고 있다. 기존의 연구에 비해 높은 수치는 아니지만 가볍고 빠르기 때문에 사용자의 스마트폰에 장착하여 사용자들이 다운받으려고 하는 어플리케이션에 대해 위험 여부를 알려주어 악성앱으로부터의 피해를 예방하는 역할로는 적합하다.

이 논문의 구성은 다음과 같다. 2장에서는 안드로이드의 보안 체계에 대해서 설명하였고 3장에서는 특성을 추출하기 위한 도구와 분류를 위한 도구들이 소개되었다. 4장에서는 실험에 대해 설명하였고, 5장에서 실험에 대한 결과를 논의하였다. 이어서 향후연구와 더불어 6장에서 결론을 맺었다.

II. 안드로이드 보안 체계

2.1 권한

권한은 서드 파티 어플리케이션(Third Party Application)에 대한 안드로이드 고유의 보안 체계이다. 만약 어떤 앱이 개발될 때 기본적으로 제공되는 자원 외에 추가적인 기능이나 사용자의 프라이버시를 침해할 수 있는 데이터를 사용해야 한다면 사용자에게 필요한 권한들을 명시하고 해당 권한들을 취득해야 한다.

안드로이드 OS에는 API level 18을 기준으로 134개의 권한들이 존재한다[7]. 사용자가 스마트폰으로 해당 앱을 다운받을 때, 앱이 필요로 하는 권한들이 하나의 창에 모두 요약되어 보여지며, 요약된 각각의 권한을 터치하면 그 권한이 사용자 스마트폰 내의 어떠한 기능이나 정보로의 접근을 가능하게 하는지 좀 더 자세히 알 수 있다. 권한들에 대한 수락은 앱이 필요로 하는 모든 권한에 대한 일괄적인 동의 혹은 거절의 형태로 이루어진다. 하지만 이러한 형태로 명시되는 권한에 대해 자세히 읽어보는 사용자가 매우 적다. Adrienne는 안드로이드 마켓에서 얻은 940개의 샘플 앱들 중 1/3 정도가 과도한 권한을 요청하고 있으며, 안드로이드에서 제공하는 개발자 문서에서 설명하는 권한에 대한 정보는 부족하고 잘못된 정보를 포함하고 있다고 말했다[8].

사용자에게 권한을 요구하는 단계는 앱이 스마트폰으로 다운로드 되기 바로 직전에 이루어진다. 어떠한 앱이라도 사용자들의 스마트폰 내의 개인적인 정보나 추가적인 기능을 사용하기 위해서는 사용자로부터 권한을 취득해야 한다. 즉 권한을 보면 개발자가 앱을 통해 어느 정보나 기능을 사용하려고 하는지 알 수 있다.

III. 분류 모델

3.1 특성 추출: 주성분 분석

샘플로 수집된 앱들은 각각의 특성들의 존재 유무에 따라 특성 벡터가 얻어진다. 초기에 얻어지는 특성 벡터 R의 형태는 다음과 같다.

$$R=(r_1, r_2, \ldots, r_n),$$
$$where\ \ r_i=\begin{cases}1 \text{ if } r_i \textit{ is requested}\\ 0 \textit{ otherwise}\end{cases} \quad (1)$$

앱으로부터 얻어진 특성 벡터의 좀 더 효율적인 사

용을 위하여 주성분 분석을 통해 새로운 특성 벡터를 추출하였다[9]. 주성분 분석은 고차원 공간에 놓여있는 데이터들의 정보를 더 잘 나타낼 수 있는 축을 찾아 공간의 차원을 줄여보고자 하는 시도라고 할 수 있다. 다시 말해 주성분 분석을 통해 우리는 훨씬 적은 차원의 공간을 사용하여 거의 모든 원본 데이터의 정보를 나타낼 수 있다. 주성분 분석의 과정은 다음과 같다.

$\boldsymbol{R}$이 n개의 앱에 대한 특성 벡터 $R_x, (x=1,2,...,n)$로 구성된 행렬 데이터라고 할 때, 먼저 주어진 데이터의 평균값 m과 공분산 행렬 $Cov(R_x, R_y)$를 구한다. 다음으로 이에 대한 고유 벡터와 고유 값을 찾는다. 고유 값의 크기 순서대로 상위 h개의 고유 값에 대한 고유 벡터 $E=[e^1, e^2, ..., e^h]$ 를 얻는다. 이를 사용해 다음과 같은 과정으로 새로운 특성 벡터 $\boldsymbol{R}^{\text{new}}$를 추출해 낼 수 있다.

$$\boldsymbol{R} \approx m + EE^T(\boldsymbol{R} - m) \approx EE^T(\boldsymbol{R}) = \boldsymbol{R}^{\text{new}} \quad (2)$$

3.2 k-Nearest Neighborhood 분류자

모든 앱은 정상(ben) 혹은 악성(mal) 클래스로 분류된다. 어떤 앱의 특성 벡터 R_x에 대해서 R_x의 클래스를 다음과 같이 나타낼 수 있다.

$$C(R_x) \in \{ben, mal\} \quad (3)$$

kNN 분류자에서 k개의 가까운 이웃을 뽑을 때 기준이 되는 거리는 두 특성 벡터의 차의 norm값으로 한다. 즉 두 특성 벡터 $R_x=[x_1, x_2, ..., x_k]$ 와 $R_y=[y_1, y_2, ..., y_k]$ 에 대해서 두 벡터의 거리는 다음과 같이 정의한다.

$$\begin{aligned} dist(R_x, R_y) &= norm(R_x - R_y) \\ &= \|R_x - R_y\| \\ &= \sqrt{(x_1-y_1)^2 + ... + (x_k-y_k)^2} \end{aligned} \quad (4)$$

새로운 앱이 발견되었을 때 주성분 분석으로 구해진 그 앱의 특성 벡터를 R_x라 하고, $R_{nr_k(x)}$는 R_x와 가장 가까운 K개의 특성 벡터들을 가지는 집합이다. 분류자 $C(R_x)$는 Fig. 1과 같이 정의된다.

```
input: R_x,  x ∈ {1, 2, ..., n}
if N(C(R_nr_k(x)) = ben) ≥ N(C(R_nr_k(x)) = mal) then
    C(R_x) = ben
else
    C(R_x) = mal
end if
```

Fig.1. The feature vector classifier

$N(C(R_{nr_k(x)}) = ben)$은 $C(R_{nr_k(x)}) = ben$ 인 R_x, $(x=1, 2, ..., h)$의 개수이고, $N(C(R_{nr_k(x)}) = mal)$은 $C(R_{nr_k(x)}) = mal$ 인 R_x의 개수이다.

3.3 Incremental kNN 알고리즘

안드로이드 앱 마켓에서는 하루에도 많은 양의 앱들이 등록되며 이 가운데 존재하는 악성앱을 감지하기 위해 많은 연구가 진행되고 있다. 악성앱 감지를 위해 정상앱으로부터 구별되는 악성앱의 특성들을 사용하게 된다. 하지만 악성앱들은 이들을 감지하는 장치들을 우회하기 위해 그 특성들을 계속적으로 바꾸며 정상앱 또한 그 특성의 패턴들이 시간의 흐름에 따라 달라질 수 있다. 이에 대하여 앱 내의 악성 여부를 판별하는 감지장치 또한 변화되는 특성들을 민감하게 반영할 필요가 있으며 앱이 들어올 때마다 매번 모든 샘플에 대한 training을 다시 수행하는 것은 비효율적이기 때문에 실시간으로 작은 연산만을 통해 새로운 샘플들의 특성을 반영할 수 있는 방법이 필요했다. 전통적인 kNN 분류자는 일괄적인 처리 시스템에 기반하므로 시간에 따라 변화하는 앱들의 특성을 반영하기 어렵다. 이를 보안하기 위해서 incremental kNN을 적용하였다[10].

$g_{x,t,c}=(R_x, 1, c)$ 은 시간 t에서 주성분 분석을 통해 주어진 새로운 특성 벡터 R_x와 R_x가 속하는 클래스 c, $(c \in \{ben, mal\})$, 그리고 1은 특성 벡터 R_x의 무게치 $w_{x,t}$에 대한 초기값을 나타낸다. 시간 t에서 악성앱과 정상앱 모델은 $g_{x,t,c}$의 튜플들의 집합으로 정의한다.

$$G_{t,c} = \{g_{1,t,c}, g_{2,t,c}, ..., g_{i,t,c}\} \quad (5)$$

R_x가 testing 결과 잘 분류되었다면, R_x에서 가장 가까운 l개의 특성 벡터 중 클래스 c에 속하는 특성 벡터의 무게치를 W_{cor}에 따라 수정한다. R_x가 testing 결과 잘못 분류되었다면, R_x에서 가장 가까

운 l개의 특성 벡터 중 c에 속하지 않는 특성 벡터의 무게치를 W_{err}에 따라 수정한다. Fig.2에서 incremental kNN에 대한 알고리즘을 보여준다. 무게치를 수정하기 위해 사용하는 W_{cor}와 W_{err}를 다음과 같이 정의한다.

$$W_{cor}(w_{x,t}) = -\frac{(w_{x,t}-2)^2}{2}+2$$
$$W_{err}(w_{x,t}) = \frac{w_{x,t}^2}{2} \quad (6)$$

Fig.2의 알고리즘을 실행할 때 무게치들이 극단적인 값으로 가는 것을 방지하기 위해 각 클래스 내의 특성 벡터들의 무게치들의 합이 비슷하게 유지되도록 한다. 이를 위해 우리는 몇 가지 장치를 정의한다. $\sum_w G_{t,c}$는 시간 t에서 클래스 c 내의 특성 벡터들의 무게치들의 합이다. ($\alpha \in (0,1)$)

$$\sum_w G_{t,c} > \frac{1}{2}(\sum_w G_{t,ben} + \sum_w G_{t,mal}) \times (1+\alpha) \quad (7)$$

$$\sum_w G_{t,c} < \frac{1}{2}(\sum_w G_{t,ben} + \sum_w G_{t,mal}) \times (1-\alpha) \quad (8)$$

만일 어떤 앱의 특성 벡터 R_i가 잘 분류되었다면

Algorithm 1 Incremental KNN

```
add new (R_x, 1, c) to G_{t,c}
if R_x is correctly classified then
    find the l NN of R_x
    for each found NN i do
        if c = c_i then
            w_{i,t+1} = W_cor(w_{i,t})
        end if
    end for
else
    find the l NN of R_x
    for each found NN i do
        if c = c_i then
            w_{i,t+1} = W_err(w_{i,t})
        end if
    end for
    for each (R_m, w_{m,t}, c) in G_{t,c} do
        if w_{m,t} < κ then
            remove (R_m, w_{m,t}, c) from G_{t,c}
        end if
    end for
end if
```

Fig. 2. The Algorithm of incremental kNN classifier

알고리즘은 수식 (7)이 만족될 때 멈춘다. 또한 R_i가 잘못 분류되었다면 알고리즘은 수식 (8)이 만족될 때 멈추게 된다.

3.4 정확도 측정치

분류된 앱의 정확도를 측정하기 위해서 다음과 같은 측정치를 사용하였다. $N_{(b,b)}$ 는 잘 분류된 정상앱의 개수이고 $N_{(b,m)}$는 잘못 분류된 정상앱의 개수이다. $N_{(m,b)}$는 잘 분류된 악성앱의 개수이고 $N_{(m,m)}$는 잘못 분류된 악성앱의 개수이다. 이 때 우리가 사용할 정확률 P_{Acc}와 에러율 P_{Err}은 다음과 같이 정의한다.

$$P_{Acc} = \frac{N_{(b,b)} + N_{(m,m)}}{N_{(b,b)} + N_{(b,m)} + N_{(m,b)} + N_{(m,m)}} \quad (9)$$

$$P_{Err} = \frac{N_{(b,m)} + N_{(m,b)}}{N_{(b,b)} + N_{(b,m)} + N_{(m,b)} + N_{(m,m)}} \quad (10)$$

IV. 실험

앱이 사용자로부터 권한을 얻고 스마트폰에 다운되기 전에 악성앱을 거르는 과정이 필요하며 이를 통해 많은 보안 이슈들을 사전에 방지할 수 있다. 안드로이드 OS에서 앱을 다운받기 전에 얻을 수 있는 정보만을 가지고 어느 정도의 성능으로 앱 내의 악성 유무를 파악할 수 있는가를 알아보기 위하여 실험하였다.

실험을 위해 사용된 앱의 apk(Android application package) 파일은 'Contagio mobile'이라는 사이트에 요청하여 그 샘플을 얻었다[11]. 이 샘플 중에는 유명 커피전문점이나 게임을 사칭하여 SMS trojan 방식으로 해당 스마트폰에 다운받도록 유도하는 악성앱과 해당 스마트폰의 루팅권한을 강제 획득하는 DroidKungFu의 다양한 버전들이 포함되어 있다. 악성앱에서 권한에 대한 정보를 추출하기 위해 Anubis 툴을 이용하였다. 정상앱에 대한 정보는 안드로이드 기반 스마트폰에서 제공하는 '구글 플레이'에서 얻었으며 정상앱 내의 악의적인 요소가 없음을 좀 더 명확하게 하기 위하여 구글, 다음, 네이버, Adobe, 삼성 등 알려진 회사가 출처인 앱으로 선정하였다. 이렇게 악성앱 164개, 정상앱 113개를 수집하였다.

4.1 특성 추출

앱의 특성들을 효율적으로 적용하기 위하여 새로운 특성들을 추출하는 과정을 거쳤다. 수집된 앱들은 각각의 권한들을 요구하는가의 여부에 따라 0 또는 1로 구성된 1*134 벡터를 특성 벡터로 가지게 된다. 이러한 얻어진 특성 벡터들에 주성분 분석을 사용하여 새로운 특성 백터들을 얻었다.

주성분 분석을 통해서 얻어낸 새로운 특성 벡터들에 대해서 이것들이 가지고 있는 정보의 양과 누적 정보량을 Table1에서 보여준다. 정보의 양을 많이 가지고 있는 순서대로 주성분 1부터 주성분 40까지 나열하였다. 나열된 주성분들 중 주성분 1이 가지고 있는 정보량은 17.52 % 로 가장 많고 주성분 40이 가지고 있는 정보량은 0.41 %로 가장 적다. 또한 누적 정보량을 보면 상위 20개의 주성분을 사용하였을 경우 전체 정보의 80.49 %를 포함하게 되고, 상위 40개의 주성분을 사용하였을 경우 전체정보의 95.25 %를 포함하게 됨을 알 수 있다. 즉, 주성분 분석을 사용하였을 경우 적은 양의 성분을 가지고 많은 정보들을 나타낼 수 있어 매우 효율적이다.

Table 1. Percentages for the amount of information(Info) and the accumulated percentages for the amount of information(Cum) depending on the number of principal components(PC).

PC	Info (%)	Cum (%)	PC	Info (%)	Cum (%)
1	17.52	17.52	21	1.32	81.81
2	9.86	27.38	22	1.18	82.99
3	6.94	34.32	23	1.13	84.12
4	6.00	40.32	24	1.02	85.14
5	4.95	45.27	25	0.97	86.11
6	4.38	49.65	26	0.91	87.02
7	3.39	53.04	27	0.85	87.87
8	3.16	56.20	28	0.77	88.64
9	3.01	59.21	29	0.73	89.37
10	2.55	61.76	30	0.69	90.06
11	2.44	64.20	31	0.65	90.71
12	2.30	66.50	32	0.61	91.32
13	2.13	68.63	33	0.58	91.90
14	2.08	70.71	34	0.57	92.47
15	1.93	72.64	35	0.54	93.01
16	1.81	74.45	36	0.52	93.53
17	1.69	76.14	37	0.47	94.00
18	1.53	77.67	38	0.43	94.43
19	1.47	79.14	39	0.41	94.84
20	1.35	80.49	40	0.41	95.25

4.2 Training

kNN 분류자에서는 샘플앱들 중 100개의 정상앱과 100개의 악성앱을 추려내어 kNN 분류자를 실험하기 위하여 사용되었다. Incremental kNN 분류자에서 샘플 앱들 중 200개는 training과 testing을 위해 또한 남은 앱들은 알고리즘을 러닝하여 무게값을 수정하는데 사용하였다. training과 testing을 위해서 10-fold cross validation 을 사용하였다. 또한 kNN과 incremental kNN 분류자의 정확률은 3.5절에서 제시한 수식 (9), (10)을 사용하여 측정하였다.

V. 실험 결과

5.1 주성분 분석

Fig.3은 주성분 분석을 사용해서 새롭게 만들어낸 특성 백터를 사용했을 때의 효율성을 보여준다. 검정색 선은 주성분 분석을 사용하지 않은 특성들 중 40개를 임의로 뽑아 실험을 한 뒤 정확률을 측정한 결과이다. 파란색 선은 주성분 분석을 사용하지 않은 특성들 중 100개를 임의로 뽑아 실험을 한 뒤 정확률을 측정한 결과이다. 빨간색 점선은 주성분 분석을 사용한 특성들 중 상위 40개를 뽑아 실험을 한 뒤 정확률을 측정한 결과이다. 그림에서 보는 바와 같이 검정색선보다 파란색 선이 12% 정도 정확률이 높은 것을 볼 수 있으며, 파란색선과 빨간색 점선의 정확률은 비슷한 것을 볼 수 있다. 즉, 주성분 분석을 사용한 경우

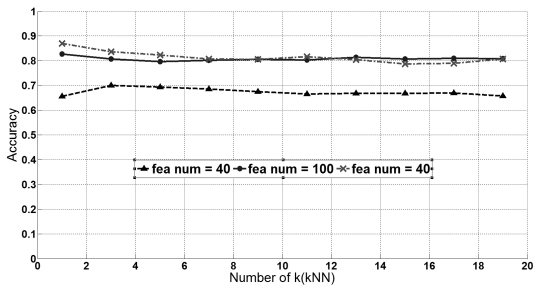

Fig. 3. The figure shows the efficiency when using Principal Component Analysis(PCA) for extracting features. The black line and the blue line are the averages of the accuracy rates using 40 features and 100 features before applying PCA. The dotted red line is the average of the accuracy rate using 40 features that PCA is applied.

훨씬 적은 양의 특성들만으로도 큰 효과를 얻을 수 있다는 것을 볼 수 있다.

5.2 kNN 분류자 실험

Fig.4는 kNN 분류자를 사용했을 때와 베이지안 분류자를 사용했을 때 각각의 정확률과 에러율을 나타내고 있다. 파란선이 나타내는 kNN 분류자에서 우리는 nearest neighbor의 개수 k마다 특성의 개수에 따른 정확률을 측정하여 평균을 내었다. 또한 빨간선이 나타내는 베이지안 분류자는 Suleiman[4]이 제안한 방법을 사용하였다.

kNN 분류자에서 nearest neighbor의 개수 k가 홀수일 때보다 짝수일 때 그 정확도가 떨어지는 것을 볼 수 있다. 이것은 분류의 기준 때문이다. kNN 분류자에서 분류는 Fig.1을 기준으로 하고 있다. k의 개수가 짝수일 때 $N(C(R_i)=c_{정상}) = N(C(R_i)=c_{악성})$ 인 경우가 생기게 되는데 기준에 의하면 다음의 경우는 정상으로 분류된다. 이러한 이유로 k의 개수가 짝수일 때 정확률이 떨어진다.

kNN 분류자를 사용했을 때 80 - 85 %의 정확률을 얻은 반면 베이지안 분류자의 경우 61 %의 정확률을 나타냈다. 그림을 보면 kNN 분류자를 사용했을 때 베이지안 분류자를 사용한 것보다 15 - 25 % 정도 정확률이 높게 나온 것을 볼 수 있다.

5.3 Incremental kNN 실험

Table.2는 kNN 분류자를 사용하였을 때와 incremental kNN 분류자를 사용하였을 때 분류의 정확도를 나타내주고 있다. 어떤 앱의 클래스를 분류하기 위하여 특성 벡터로부터 가장 가까운 k개의 특성 벡터들을 고르게 되는데 여기서 k의 개수와 특성 벡터에 나타나는 특성들 중에서 반영할 특성의 개수 f_n ($f_n \in 5, 10, 15, 20$)을 변경해가면서 정확도를 측정하였다. 반영된 특성들의 개수가 같은 수치 중에서 가장 높은 수치를 굵게 표시해 두었다. 평균이라고 표시된 행에 씌여진 수치들은 각 반영된 특성들의 개수마다 k의 개수 1에서 10개까지의 정확률의 평균이다.

특성의 개수가 5개, 10개, 15개, 20개일 때, 평균을 나타내고 있는 행을 비교해보면 incremental kNN 분류자를 사용했을 때의 정확률이 기존 kNN 분류자를 사용했을 때의 정확률보다 정확률이 높다. 또한 반영된 특성의 개수가 10개 이상인 경우 각 열에서 k의 개수가 1일 때의 정확도가 가장 높다. 이를 보아 특성의 개수가 10개 이상인 경우 1개의 가장 가까

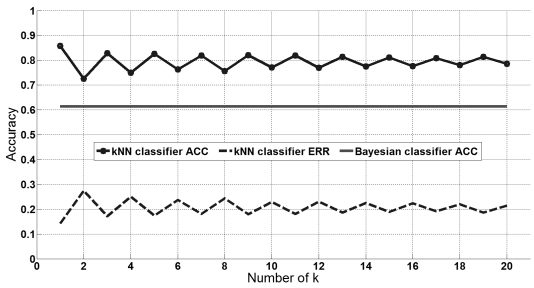

Fig. 4. Comparing the accuracy rate of using the Bayesian classifier and the kNN classifier. The blue line and blue dotted line are the averages accuracy rate and error rate using the kNN classifier and the red line is the average accuracy rate using the Bayesian classifier.

Table 2. Comparing the accuracy rate of using kNN classifier and incremental kNN classifier. The bold ones are greatest value in each number of features. ($f_n \in \{5, 10, 15, 20\}$)

k_NN	f_n			
	5	5(inc)	10	10(inc)
k=1	0.83	0.84	**0.85**	**0.85**
k=2	0.71	0.75	0.72	0.76
k=3	0.80	0.80	0.81	0.71
k=4	0.74	0.78	0.73	0.75
k=5	**0.85**	**0.86**	0.80	0.81
k=6	0.79	0.82	0.74	0.77
k=7	**0.85**	0.85	0.79	0.80
k=8	0.80	0.83	0.74	0.78
k=9	0.84	0.84	0.82	0.82
k=10	0.79	0.81	0.77	0.80
avg.	0.80	0.82	0.78	0.79
k_NN	f_n			
	15	15(inc)	20	20(inc)
k=1	**0.85**	**0.85**	**0.85**	**0.85**
k=2	0.72	0.77	0.74	0.77
k=3	0.84	0.84	0.83	0.84
k=4	0.73	0.78	0.76	0.79
k=5	0.80	0.84	0.83	0.83
k=6	0.74	0.80	0.76	0.80
k=7	0.79	0.81	0.82	0.81
k=8	0.74	0.79	0.76	0.79
k=9	0.82	0.81	0.83	0.83
k=10	0.77	0.79	0.78	0.80
avg.	0.78	0.81	0.80	0.81

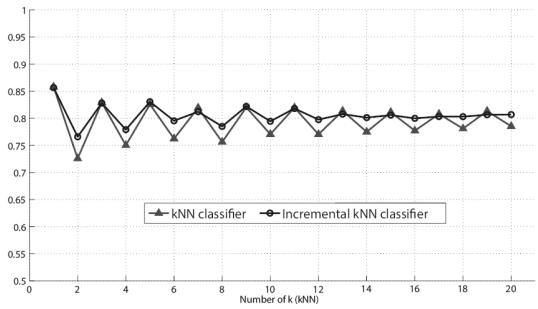

Fig. 5. Averages accuracy rates using the kNN classifier and the incremental kNN classifier. The red line is the accuracy using the kNN classifier and the blue line is the accuracy using the incremental kNN classifier.

운 이웃 벡터만을 사용하는 걸로도 충분한 정확률을 나타낸다는 것을 알 수 있다.

Fig.5는 incremental kNN 분류자와 kNN 분류자를 적용했을 때 정확률의 평균을 보여준다. 파란색선은 incremental kNN 분류자를 사용하였을 때의 정확률을 나타내며 빨간색선은 kNN 분류자를 사용했을 때의 정확률을 나타낸다. 그림에서 incremental kNN을 사용하였을 때 전반적으로 정확률이 높아졌으나 k의 개수가 홀수일 때에는 오히려 정확률이 줄어들기도 하였다. 하지만 새롭게 들어오는 앱들의 특성을 적용하여 그 이 후의 앱을 분류할 때 그 정확도에 변화가 있었으며 전반적으로 향상되어 긍정적인 효과를 얻었다.

VI. 결론 및 향후 연구 계획

이 논문에서의 목적은 안드로이드 OS 기반의 스마트폰에서 앱을 다운받기 전에 해당 앱에 악의적인 목적이 있는지에 대한 여부를 감지하는 것이었다. 사용자가 앱을 다운받기 바로 직전에 이루어지는 권한에 대한 수락 단계는 앱 개발자의 입장에서 앱이 사용자의 개인 정보나 기기 자원들의 추가적인 사용을 위해 필수적으로 통과해야 하는 과정이다. 이렇게 요구된 모든 권한에 대해 사용자로부터 수락을 받으면 앱 다운로드가 가능하다. 즉 앱이 사용자에게 요구하는 권한을 보면 해당 앱이 스마트폰에 다운로드 되었을 때 취할 행동에 대해 짐작이 가능하다.

따라서 이 논문에서는 권한을 중심으로 특성 벡터를 추출한 뒤 kNN을 이용하여 분류하는 실험을 하였다. 가장 정확률이 높은 결과는 87 % 정도이다. kNN 분류자는 가볍고 빠르기 때문에 사용자들의 스마트폰에 탑재하여 다운받으려는 어플리케이션에 대한 위험이나 안전의 정도를 알려주기에 적합하다. 기존의 방식은 사용자가 앱을 다운받기 전 하나의 창에 권한의 목록이 제시되는 방식이지만 이에 더하여 해당 앱의 위험성의 정도를 알려준다면 사용자는 위험을 보다 잘 인식할 수 있게되며 좀 더 강화된 안드로이드 보안 체계를 기대해 볼 수 있다. 또한 incremental kNN 분류자의 사용으로 새롭게 등장하는 앱에 대해 그 특성을 분류자에 쉽게 반영하여 변화하는 악성앱에 대해서도 대비할 수 있다.

앱 내의 악성의 여부를 감지하는 감지기는 크게 두 부분에 탑재할 수 있다. 한 부분은 안드로이드 기반의 스마트폰 내부이고 한 부분은 안드로이드 앱들이 올라오는 마켓이다. kNN 기반의 감지기는 분류 정확률이 상대적으로 상당히 높지는 않지만 빠르고 가볍기 때문에 스마트폰 내부에 탑재하는 것이 적합하다. 향후 연구에서는 kNN 기반 감지기에서 정확률과 효율성을 높이기 위한 노력을 하려고 한다. 악의적인 성향을 가졌다는 것이 확실한 앱과 악의적인 성향을 갖지 않았다는 것이 확실한 앱, 또한 그 기준이 애매한 앱으로 분류하여 그 기준이 애매한 앱에 대해서 한 번 더 분석 과정을 거치도록 하는 체계를 생각해 보았다. 이를 위해서 그 세 분류에 대한 기준치와 추가적으로 첨가될 분석 방법에 대해 연구할 것이다.

References

[1] "Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time," Gartner, August, 2014

[2] "Mobile threat report q4 2012," F-Secure, 2012

[3] "NQ mobile 2012 security report," NQ mobile, 2012

[4] S. Y. Yerima, S. Sezer, G. McWilliams and I. Muttik, "A new android malware detection approach using bayesian classification," Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on. IEEE, pp.121-128, 2013.

[5] C. Kolbitsch, Anubis: Analyzing un-

known binaries. URL {http://anubis.iseclab.org}
〔6〕 A. Reina, A. Fattori and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," EuroSec, April. 2013.
〔7〕 Android, Android developers. URL {http://developer.android.com}
〔8〕 A. P. Felt, E. Chin, S. Hanna, D. song and D. Wagner. "Android permissions demystified," Proceedings of the 18th ACM conference on Computer and communications security, ACM, pp. 627-638, Oct, 2011.
〔9〕 D. Barber, "Machine learning a probabilistic approach," 2006.
〔10〕 K. Forter, S. Monteleone, A. Calatroni, D. Roggen and G. Troster, "Incremental kNN classifier exploiting correct-error teacher for activity recognition," Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on. IEEE, pp. 445-450, Dec. 2010.
〔11〕 Mila, Contagio mobile. URL {http://contagiominidump.blogspot.kr}

〈저자소개〉

이 혜 림 (HyeLim Lee) 학생회원
2012년 8월: 고려대학교 정보수학과 졸업
2013년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 정보보호, 안드로이드 보안, 데이터마이닝

장 수 희 (SooHee Jang) 학생회원
2012년 8월: 가톨릭대학교 경제학과 졸업
2013년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 정보보호, 모바일 보안, 신호처리, 금융보안

윤 지 원 (JiWon Yoon) 정회원
2003년 2월: 성균관대학교 정보공학사 졸업
2005년 2월: University of Edinburgh, 정보학과 석사 졸업
2008년 11월: University of Cambridge 전자공학과 박사 졸업
2008년 2월~2009년 5월: University of Oxford, 로봇연구소 박사후과정
2009년 5월~2011년 5월: University of Dublin 통계학과 연구원 및 강사
2011년 7월~2012년 8월: IBM 연구소 정규 연구원
2012년 9월~현재: 고려대학교 정보보호대학원 조교수
〈관심분야〉 신호정보처리, 응용통계, 도감청 탐지기술