안드로이드 앱 TUTORIAL

1 ABSTRACT

안드로이드 앱을 정적 분석하여 추출한 여러가지 특성을 기반으로 악성 앱을 탐지하고, 탐지한 악성 앱을 행위에 따라 패밀리를 분류하는 샘플 모델을 소개합니다.

* Tutorial 에서 소개되는 모델은 참고문헌 중 하나인 Andro-AutoPsy 의 작동 원리를 기반으로 하고 있습니다. 악성 행위와 관련된 API, 시스템 명령어 리스트 등에 대한 상세한 내용은 논문 "Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information" 상에서 확인하실 수 있습니다.

** 이번 Data Analysis Challenge 에 제공될 안드로이드 앱은 tutorial 에서 사용한 데이터 셋과 수량 및 파일이 다를 수 있습니다.

2 안드로이드 앱의 특성 추출

정상 앱과 악성 앱을 구별하기 위해서는 앱의 행동 패턴이 반영된 특성을 추출해내어야 합니다. 본 tutorial 에서 소개하는 시스템에서는 다음과 같은 요소를 중점적으로 분석하였습니다.

2.1 인증서의 일련번호 (SERIAL NUMBER OF A CERTIFICATE)

회사에서 내부 시스템에 공격을 시도한 외부 IP 주소 blacklist 를 관리하고 방화벽 정책을 통해 차단시키는 것과 같이, 악성 앱 제작자가 주로 사용하는 인증서 정보를 수집하여 blacklist 를 작성할 수 있습니다. 9,990 개의 악성 앱 데이터 셋에서 사용된 인증서 일련번호를 추출하여 보았을 때 사용된 일련번호는 총 1,834 개이지만, 놀랍게도 그중 겨우 86 개의 일련번호가 70%의 악성 앱에서 사용된 것을 발견하였습니다. 최종적으로는, 전체 인증서 일련번호 리스트 중에서 2 종류 이상의 패밀리에 사용된 일련번호를 blacklist 로 만들어 악성 앱 탐지에 활용하였습니다. 단지 1 개의 악성 앱에서만 발견되는 인증서 일련번호를 제외한 이유는 False Positive 를 증가시킬 수 있기 때문입니다.

2.2 악성 행위와 관련된 API SEQUENCE (MALICIOUS API SEQUENCE)

사용자의 개인정보 수집, 웹사이트 접속, SMS 발신 및 삭제 등 악성 앱에서 많이 사용하는 것으로 알려진 API 및 API sequence 를 분석하였습니다. 예를 들어, getLastKnownLocation(), sendTextMessage(), getUserData() 같은 API 를 고려할 수 있습니다.

2.3 권한 분포 (PERMISSION DISTRIBUTION)

안드로이드 앱 설치 시 요구하는 권한(Permission)의 분포 또한 정상 앱과 악성 앱 간에 차이가 생깁니다. 예를 들어, SEND_SMS 권한은 정상 앱에서는 1~2%의 적은 확률로 사용되지만 악성 앱의 경우 40%가 넘는 확률로 사용됩니다. 따라서 특정 앱이 SEND_SMS 권한을 요구할 경우 악성 앱일 확률이 높습니다. 이와 같이 사용하는 권한의 분포 차이에 따라 정상 또는 악성 여부를 확률적으로 추정하였습니다.

2.4 인텐트 (INTENT)

안드로이드 앱은 작업을 수행하기 위해 activity, service, broadcast receiver, and content provider 라는 구성요소 간에 인텐트라는 메시지를 주고 받습니다. 악성 앱이 SMS 수신 알림을 (안드로이드 폰 상에 기본 설치된 문자 앱을 포함하여) 다른 앱에 전달되는 것을 막아 사용자가 SMS 수신 여부를 알 수 없게 하는 행위를 발견하기 위해 인텐트 정보를 추출하여 이용하였습니다.

2.5 시스템 명령어 (SYSTEM COMMAND)

chmod, killall 등 일반적으로 정상적인 작업에서는 불필요한 시스템 명령어가 코드 상에 포함되어 있다면 악성 행위로 의심할 수 있습니다. 이와 같이 악성 앱에서 주로 발견되는 시스템 명령어를 조사하여 악성 앱 탐지에 활용하였습니다.

2.6 위조된 파일 (FORGED FILE)

앱이 로딩하는 파일의 확장자가 파일 헤더 상의 실제 확장자와는 다르게 설정되어 있다면 악성 행위를 위한 스크립트 등을 정상적인 파일로 보이도록 숨긴 것으로 추측할 수 있습니다. 예를 들어, 헤더 정보를 해석하였을 때는 .elf, .apk, .jar 파일이나 파일 명칭은 .jpg, .gif, .png 와 같은 평범한 그림 파일과 같은 확장자로 설정하는 경우가 있습니다. 이와 같이 파일 위조 여부를 분석하여 악성 앱 탐지에 활용하였습니다.

3 안드로이드 악성 앱 탐지

3.1 탐지 알고리즘

챕터 2 에서 설명한 인증서 일련번호, 악성 API sequence, 앱이 요청하는 권한 등의 요소들을 apk 파일에서 추출하여, 다음과 같은 알고리즘을 통해 안드로이드 앱의 악성 여부를 탐지하는 모델을 설계하였습니다.

Algorithm 1: Malware detection algorithm

Input : Application = $\{A_1, A_2, ..., A_n\}$
Output : {malicious, benign}

Initialization:
B = $\{SN_1, SN_2, ..., SN_m\}$ ▷ Serial number blacklist
SN_{A_k} ▷ Serial number of Application k
API_{A_k} ▷ API of Application k
MAPI = $\{MAPI_1, MAPI_2, ..., MAPI_k\}$ ▷ Malicious API dictionary
for $i \leftarrow 1$ **to** n **do**
  if $SN_{A_i} \in B$ *AND* $API_{A_i} \in MAPI$ **then**
    $A_i \leftarrow$ malicious;
  else
    if $API_{A_i} \notin MAPI$ **then** $A_i \leftarrow$ benign;
    else if ∃ *system commands in* A_i *AND* ∃ *forged files in* A_i **then**
      $A_i \leftarrow$ *malicious*;
    end
    else if ∃ *Code to conceal received SMS notification in* A_i **then**
      $A_i \leftarrow$ malicious;
    end
    else if ∃ *Behavior pattern of smishing in* A_i **then**
      $A_i \leftarrow$ malicious
    end
    else if *Likelihood ratio of requested critical permission* $> T_L$ *AND*
    Likelihood ratio of API-related critical permission $> T_L$ *AND*
    Calibrator to reduce false positives **then**
      $A_i \leftarrow$ malicious;
    end
    else
      $A_i \leftarrow$ benign;
    end
  end
end

[그림 1] 안드로이드 악성 앱 탐지 알고리즘

3.2 탐지 결과

A 에서 제시한 알고리즘을 Python 으로 구현하여 테스트를 진행하였습니다. 테스트에 사용된 데이터 셋은 총 119,183 개의 안드로이드 앱(정상 앱: 109,193 개 / 악성 앱: 9,990 개)으로 구성되어 있습니다.

실험결과는 다음 표와 같습니다.

Category		Actual Class	
		Malware	Benign
Estimated Class	Malware	9,865	535
	Benign	125	108,658

- ✓ Accuracy: 99.45%

(Accuracy 산출 방법은 tutorial 의 6.Evaluation Standard 를 참고하시기 바랍니다.)

4 안드로이드 악성 앱 패밀리 분류

4.1 분류 알고리즘

챕터 3.2 에서 Malware 로 탐지한 앱을 대상으로 악성 행위의 특성에 따라 패밀리를 분류하였습니다. 분류 알고리즘에서 사용된 주요 요소는 앱에서 추출한 악성 행위와 관련된 API sequence, 시스템 명령어, 권한 요청 정보가 있습니다. 각 요소에 대해 아래 표와 같은 유사도 기법을 적용하였으며, 각 유사도 값의 평균을 구하여 점수가 높은 패밀리에 분류되도록 설계하였습니다. 기존 패밀리에 새로운 앱이 추가되면, 해당 군집을 대표하는 프로파일에 새로 추가된 앱의 정보를 합집합 형태로 업데이트 합니다. 만약 유사도 점수가 0.7 보다 작을 경우, 기존 패밀리에 포함시키지 않고 새로운 패밀리를 생성하도록 설정합니다.

Behavior factor	Contents	Similarity metric	Update
Malicious API sequence	Information retrieving-related API (System info., Personal info.), Transmission-related API (Data network, SMS, Call), Dynamic loading-related API	Needleman-Wunsch algorithm [0, 1]	X
Usage of system commands for leveraging forged files	Privilege escalation commands, File/Directory ownership commands, Execution commands, Process commands	Jaccard coefficient [0, 1]	O
Usage of critical permission	Requested critical permissions, API-related critical permissions	Levenshtein distance [0, 1]	O

[표 1] 분류에 사용한 behavior factor 별 유사도 기법 및 업데이트 여부

4.2 분류 결과

악성 앱 탐지 알고리즘을 테스트할 때 사용한 119,183 개의 안드로이드 앱 데이터 셋으로 분류 알고리즘 또한 테스트하였습니다.

실험결과는 다음과 같습니다.

Category	**Family**	**Classification Result**	
		False Positives	False Negatives
Malware (9,990)	AdWo (2,807)	556	253
	Boxer (2,138)	190	43
	Dowgin (936)	246	150
	AirPush (495)	343	114
	Gappusin (487)	102	259
	SMStado (288)	37	6
	Counterclank (276)	84	196
	Wapsx (273)	88	103
	OpFake (240)	56	115
	SMSAgent (202)	26	3
	Kuguo (192)	92	85
	Ropin (192)	103	120
	SmsPay (183)	26	56
	Youmi (173)	64	73
	Kmin (119)	11	0
	DroidKungFu (112)	21	27
	Mseg (103)	41	18
	SmsReg (103)	16	18

	FakeBattScar (99)	3	4
	GingerMaster (94)	10	71
	FakeNotify (86)	32	1
	PremiumSMS (63)	4	33
	JiFake (53)	12	19
	Boqx	11	34
	DroidDream (51)	24	41
	Plankton (49)	8	38
	DroidRooter (46)	25	15
	SMSSend (46)	24	15
	Agent (20)	2	13
	Utchi (12)	5	0
Benign (109,193)		125	535
Average		77	79

- ✓ Accuracy: 58.11%

  (Accuracy 산출 방법은 tutorial 의 마지막 챕터를 참고하시기 바랍니다.)

5 EVALUATION

악성 앱 탐지 및 패밀리 분류 모델을 구현한 환경은 다음과 같습니다.

- ✓ Intel(R) Xeon(R) X5660 processor
- ✓ 8 GB RAM
- ✓ 64-bit Microsoft Windows 7 Enterprise operating system

위 환경에서 실험 진행 시 분석 속도는 72 초/MB 로 측정되었습니다. 분석 시간의 대부분은 앱에서 필요한 정보를 추출하는데 쓰였으며, 악성 여부 탐지 및 패밀리 분류에 쓰여진 시간은 앱 당 평균 0.3 초만 소요되었습니다.

챕터 3.2 에서 확인하였듯이 악성 앱의 탐지 정확도는 99.45%로 상당히 높게 나타났습니다. 그에 비해 탐지한 악성 앱의 패밀리를 분류한 결과에 대한 정확도는 58.11%였습니다. 주로 다른

패밀리와 특정 행위가 겹치거나(e.g. GingerMaster 의 ginger-break exploit code) 특징적인 악성 행위가 없을 경우(e.g. Counterclank) 패밀리의 분류 정확도가 많이 낮았습니다.

6 EVALUATION STANDARD

6.1 안드로이드 악성 앱 탐지 결과에 따른 정확도

Category		**Actual Class**	
		Malware	Benign
Estimated Class	Malware	*True Positive*	*False Positive*
	Benign	*False Negative*	*True Negative*

True Positive 는 실제 *Malware(True)*를 *Malware(True)*로 정확하게 예측한 상황을 의미하며,
False Positive 는 실제 *Bengin(False)*을 *Malware(True)*로 예측한 상황을 의미합니다.
False Negative 는 실제 *Malware(True)*를 *Bengin(False)*로 예측한 상황을 의미하며,
True Negative 는 실제 *Benign(False)*를 *Benign(False)*로 예측한 상황을 의미합니다.

탐지 결과가 위 표와 같을 때, 전체 결과 중 정상 앱과 악성 앱을 정확히 탐지한 비율을 측정합니다.

$$\text{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + False\ Positive + False\ Negative + True\ Negative}$$

6.2 안드로이드 악성 앱 패밀리 분류 결과에 따른 정확도

Category		**Number of APPs in each family**	**Estimated Result**	
			Correct	Wrong
Actual Class	Family 1	T_1	A_1	B_1
	Family 2	T_2	A_2	B_2
	…	…	…	…
	Family N	T_N	A_N	B_N

분류 결과가 위 표와 같을 때, 전체 결과 중 악성 앱의 실제 패밀리를 정확히 탐지한 비율을 측정합니다.

$$\text{Accuracy} = \frac{A_1 + A_2 + \cdots + A_N}{T_1 + T_2 + \cdots + T_N}$$

7 REFERENCE

본 tutorial 에서 소개되는 모델은 참고문헌 중 하나인 Andro-AutoPsy 의 작동 원리를 기반으로 하고 있습니다. 악성 행위와 관련된 API, 시스템 명령어 리스트 등에 대한 상세한 내용은 아래 논문에서 참고하시기 바랍니다. 논문 파일은 Data Analysis Challenge 사이트 Challenge - Tutorial 메뉴에서 확인하실 수 있습니다.

- Jae-wook Jang, Hyunjae Kang, Jiyoung Woo, Aziz Mohaisen, and Huy Kang Kim, "Andro-AutoPsy: Anti-malware system based on similarity matching of malware and malware creator-centric information," Digital Investigation, vol. 14, pp. 17–35, 2015.

- 본 데이터 분석 챌린지에서 제공하는 데이터셋을 기반으로 논문을 작성할 경우, 해당 논문을 인용해야합니다.