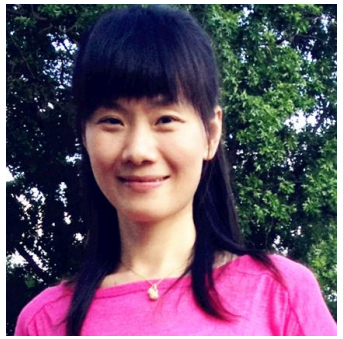
HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network

Shifu Hou[1], Yanfang Ye[1]✉, Yangqiu Song[2], Melih Abdulhayoglu[3]

1. Department of CSEE, West Virginia University, WV, USA
2. Department of CSE, HKUST, Hong Kong, China
3. Comodo Security Solutions, Inc., Clifton, NJ, USA

KDD2017



Yanfang (Fanny) Ye, Ph.D.

Assistant Professor

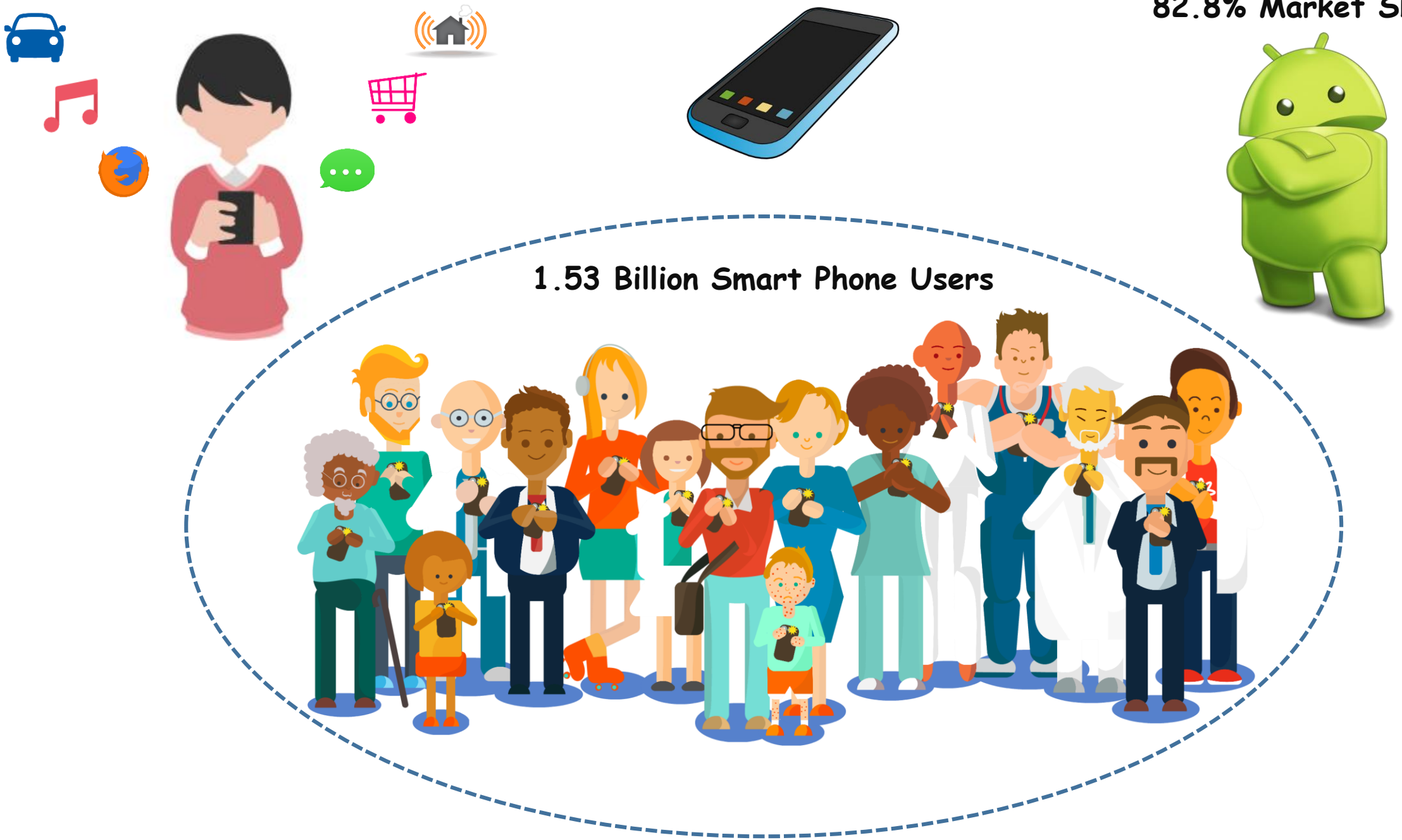
Lane Dept. of CSEE, West Virginia University

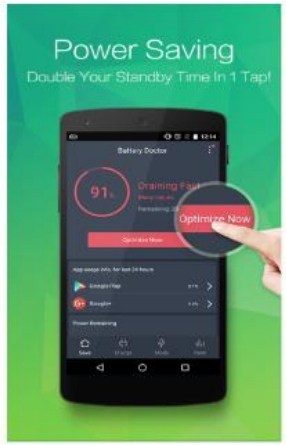
http://community.wvu.edu/~yaye/

- 2014-Now, **Assistant Professor** @ **Lane Department of CSEE, WVU**, U.S.A
- 2010-2013, **Principal Scientist** @ **Comodo Security Solutions, Inc.**, U.S.A. and China
- 2008-2010, **Deputy Director R&D** @ **Kingsoft Internet Security Corporation**, China

My research interests lie in **data mining, cybersecurity, and smart devices**. In recent years, I have **proposed and developed cloud-based approaches to Internet security**, especially malware detection and phishing fraud detection, **using data mining and machine learning techniques** (ACM CSUR 2017, KAIS 2017, IEEE TNNLS 2016, ACM TIST 2015, IEEE TSMC 2012, 2010, ACM SIGKDD 2017, 2011-2009, 2007, …), and have been awarded 3 patents in the area of malware detection and categorization. My work has been built into solutions for popular commercial products including Comodo Internet Security and Kingsoft Anti-virus that serve millions of customers worldwide. I recently received the **NSF SaTC Award** (2016-2019) and WVU Statler **New Researcher of the Year Award** (2016-2017).

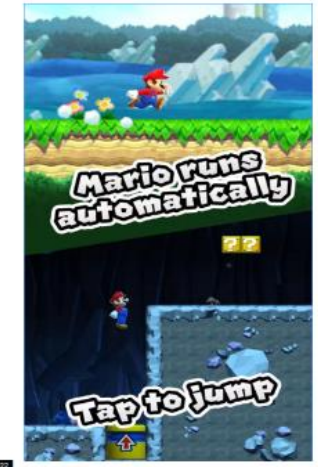
Introduction


82.8% Market Share
1.53 Billion Smart Phone Users


Power Saving
Double Your Standby Time In 1 Tap!
Optimize Now


BRING FRIENDS
TO BATTLE


LIVE
Amanda Carter Wow that looks so fun! I need to be there next time.
Colby Harris Which field is this?
Andrea Elizabeth Love love this so much
George Van de Vel Where are you?!
SHARE
Write a comment...


Mario runs automatically
Tap to jump


IISuperwomanII
HOME
VIDEOS
PLAYLISTS
COMMUNITY
IISuperwomanII
How Single Girls Discuss Relationships

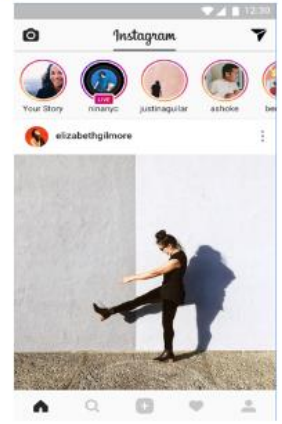

Express yourself with free stickers
Brandi
Happy Birthday!


Symbols
Gym level 2
TrainerNIA
CHARIZARD
CP 885


COLOR SWITCH
GAME MODES


Simple. Personal.
Real time messaging.
WhatsApp
CHATS
Whitmans Chat
Stewart Family
Alice Whitman
Jack Whitman
Lunch Group
Jane Pearson


ANGRY BIRDS
#1 APP OF ALL TIME


Instagram
Your Story
elizabethgilmore


Get the very latest on the things you care about most.
Lindsey
GoldenStateWarriors

Google Play Opera Mobile Store Wandoujia

WARNING

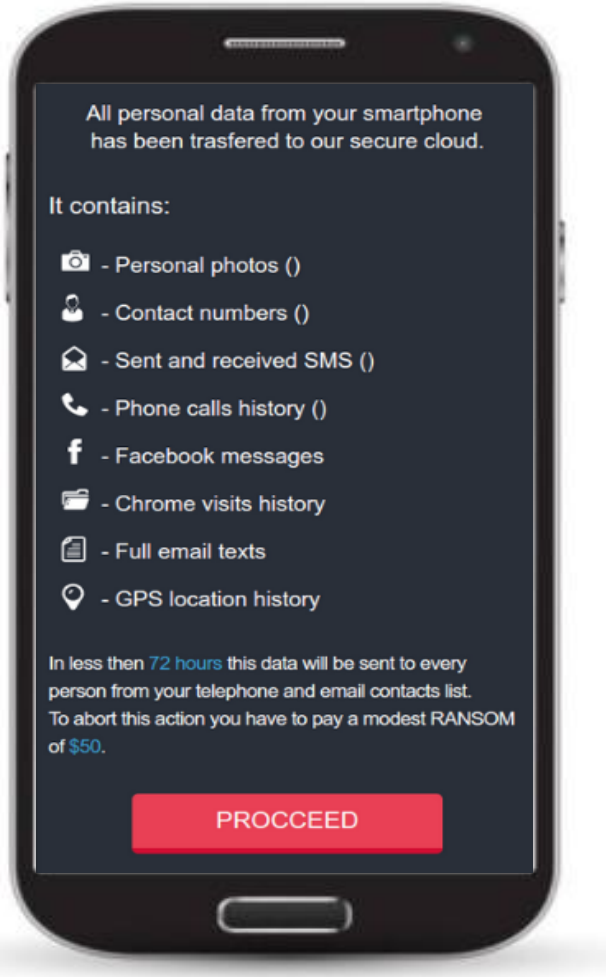
DOWNLOAD

Steal money

Send SMS message

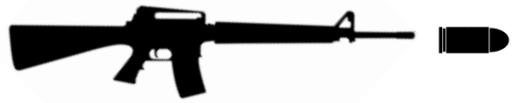
Push advertisement

Download unwanted app

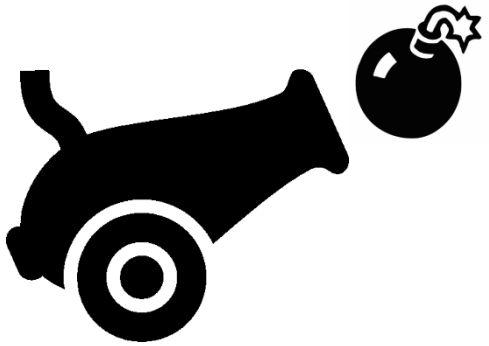


Ransomware:

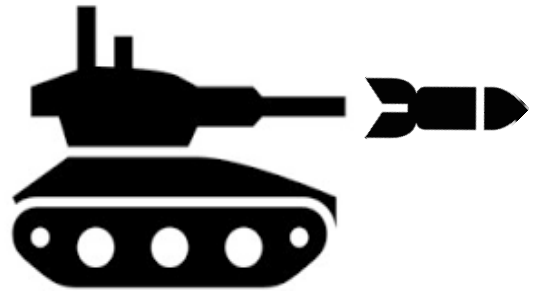
A type of malicious software that blocks access to the victim's data or threatens to publish or delete it until a ransom is paid.



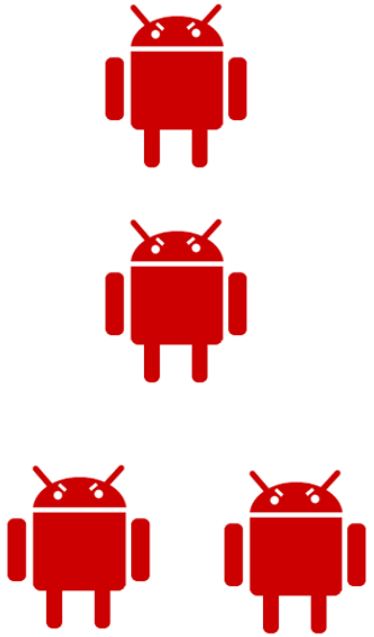
Signature-based Defenders



Machine Learning-based Defenders



HinDroid



HinDroid

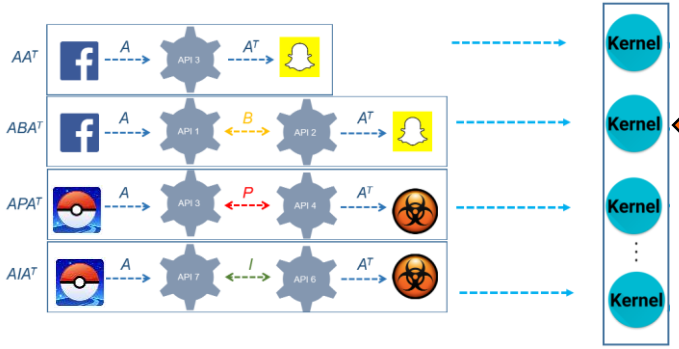
It's a more resilient system that helps protect smart phone users against Android malware attacks and novel threats.

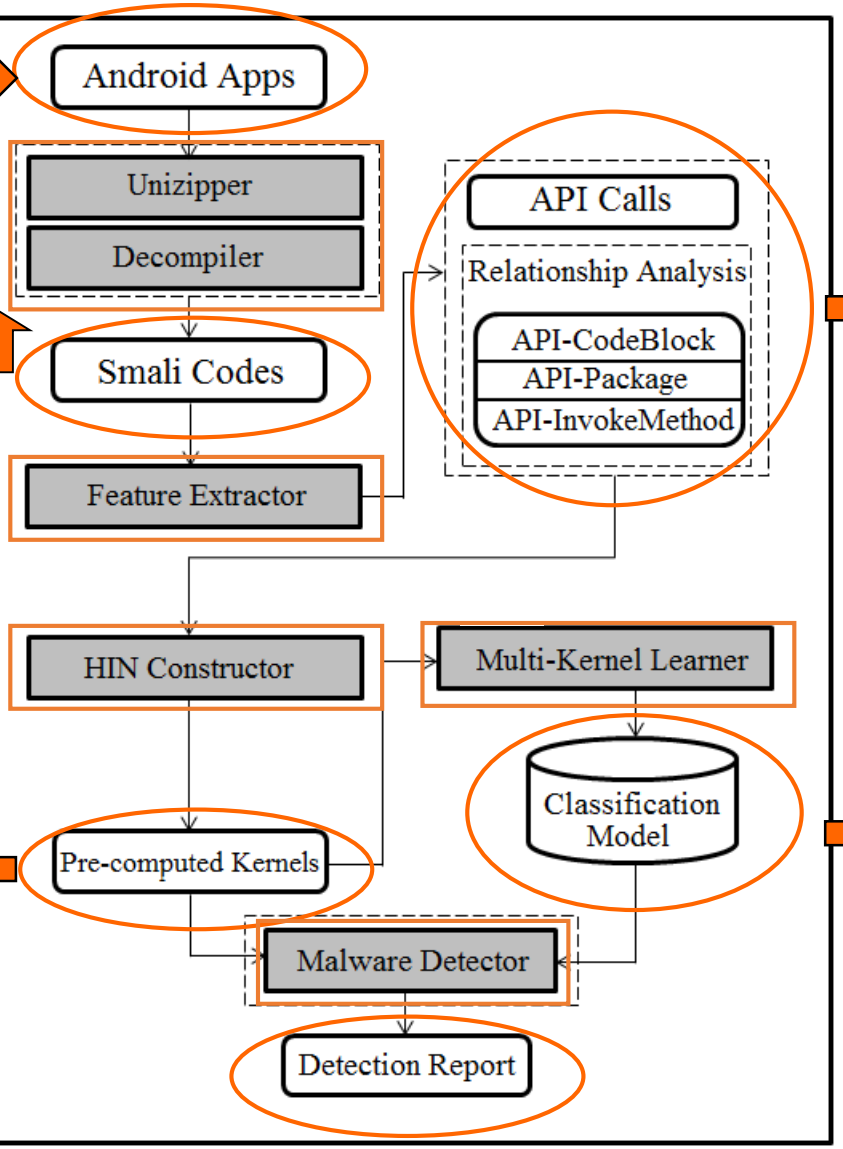
HinDroid System Architecture

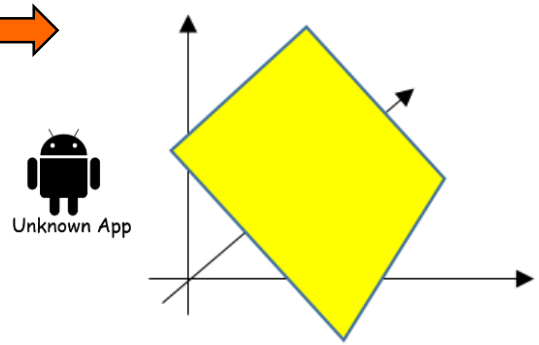
Android app is compiled and packaged in a single archive file (.apk) that includes the app **code** (**.dex** file), resources, assets, and manifest file.

Listing 1: An example of smali code

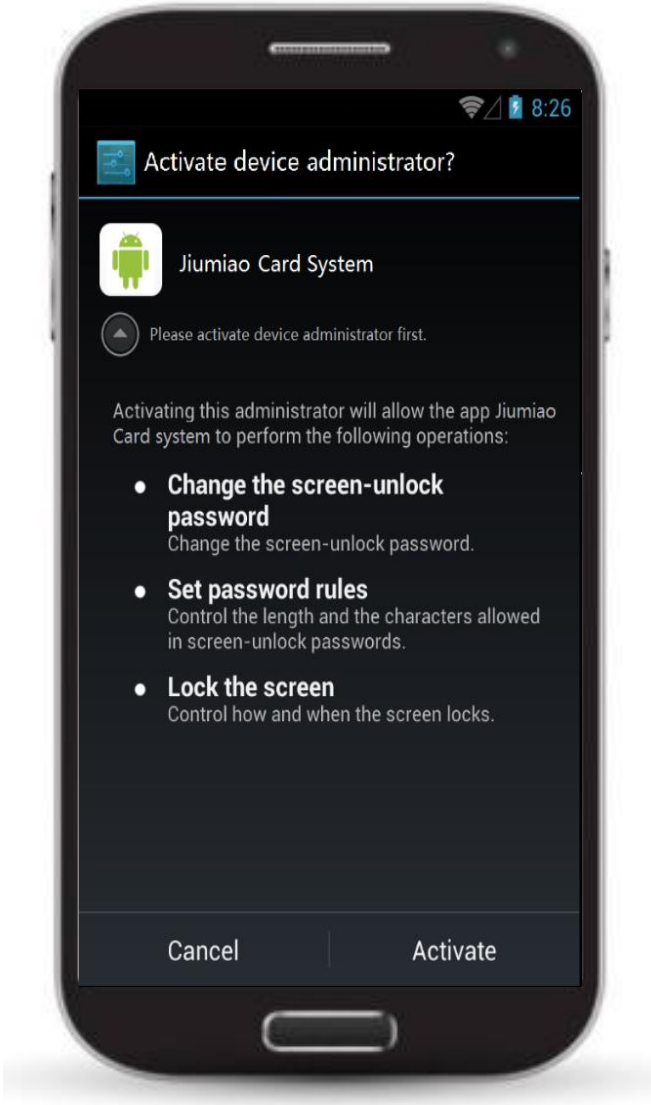
```
1  .method protected
2  loadLibs(Landroid/content/Context;)V
3  .locals 4
4  :try_start_0
5  new-instance v0, Ljava/io/BufferedReader;
6  new-instance v1, Ljava/io/InputStreamReader;
7  invoke-static {}, Ljava/lang/Runtime;->getRuntime()Ljava/lang/Runtime;
8  move-result-object v2
9  const-string v3, "getprop_ro.product.cpu.abi"
10 invoke-virtual {v2, v3}, Ljava/lang/Runtime;->exec(Ljava/lang/String;)
       Ljava/lang/Process;
11 move-result-object v2
12 invoke-virtual {v2}, Ljava/lang/Process;->getInputStream()Ljava/io/InputStream
13 .....
14 .end method
```







Feature Extraction



e.g., **ransomware "Locker.apk"**

(MD5: f836f5c6267f13bf9f6109a6b8d79175)

Feature Extractor

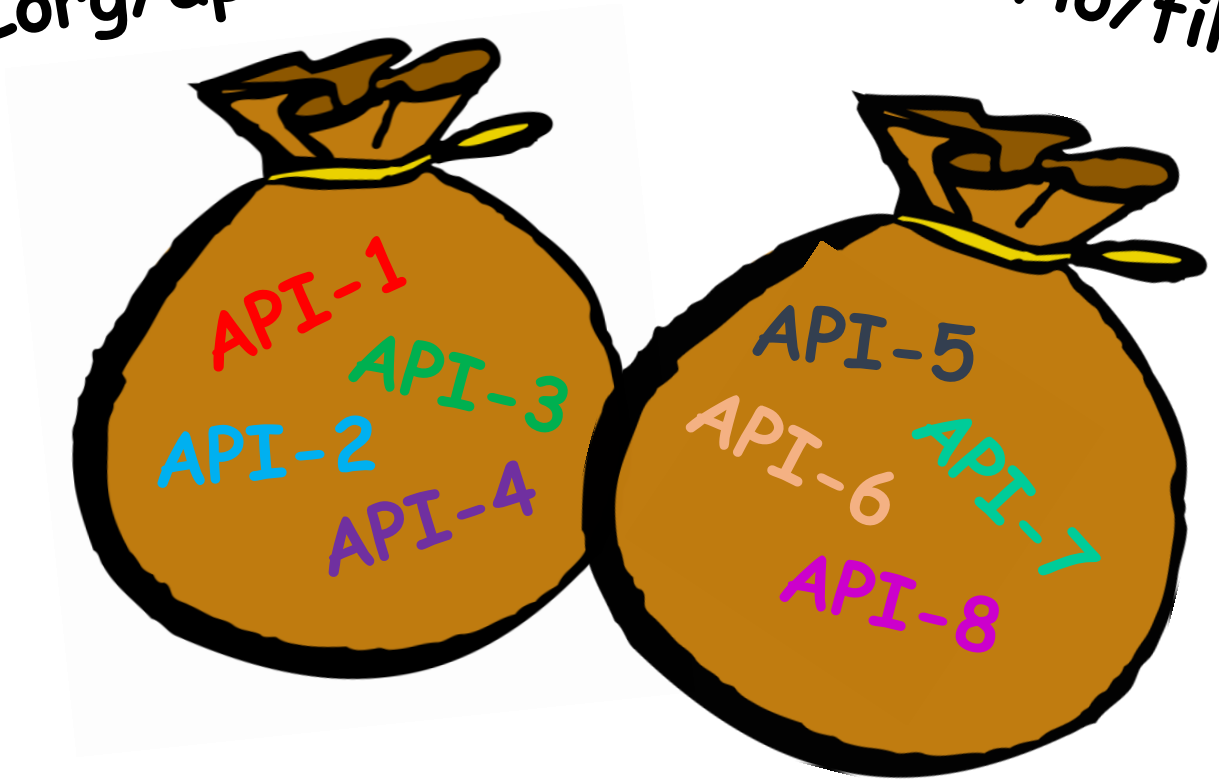
Smali Code:

APIs

```
.method protected
loadLibs(Landroid/content/Context;)V
.locals 4
:try_start_0
new-instance v0, Ljava/io/BufferedReader;
new-instance v1, Ljava/io/InputStreamReader;
invoke-static {}, Ljava/lang/Runtime;->getRuntime()Ljava/lang/Runtime;
move-result-object v2
const-string v3, "getprop ro.product.cpu.abi"
invoke-virtual {v2, v3}, Ljava/lang/Runtime;->exec(Ljava/lang/String;)
    Ljava/lang/Process;
move-result-object v2
invoke-virtual {v2}, Ljava/lang/Process;->getInputStream()Ljava/io/InputStrea
.....
.end method
```

Package1:
Lorg/apache/http

Package2:
Ljava/io/file

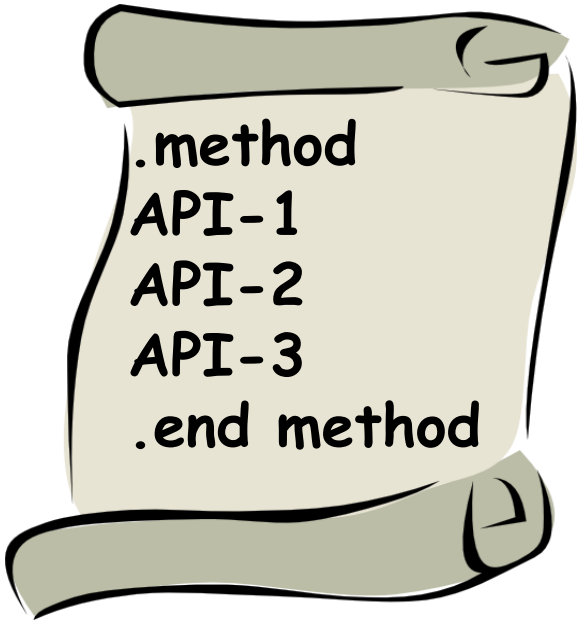


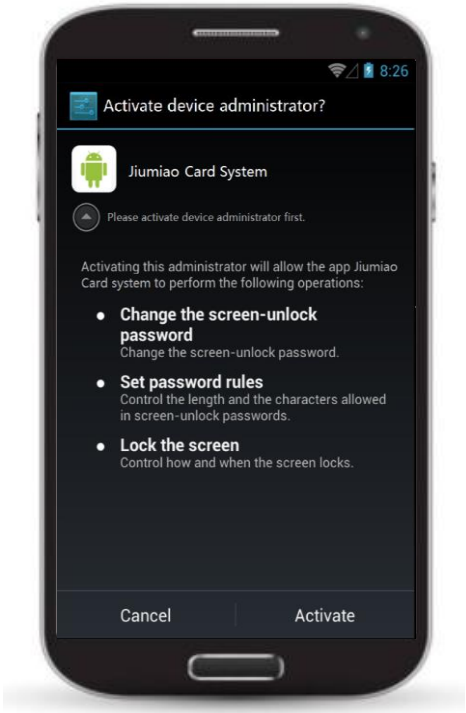
API-1: Lorg/apache/http->HttpConnection()
API-2: Lorg/apache/http->HttpMessage()
API-3: Lorg/apache/http->HttpRequest()
API-4: Lorg/apache/http->HttpResponse()

API-5: Ljava/io/file->getPath()
API-6: Ljava/io/file->setReadOnly()
API-7: Ljava/io/file->canRead()
API-8: Ljava/io/file->delete()

- To represent such kind of relationship, we generate the **API-Package** matrix **P** where each element $\mathrm{P}_{ij} = p_{ij} \in \{0, 1\}$ denotes if a pair of API calls are with the same package name.

CodeBlock





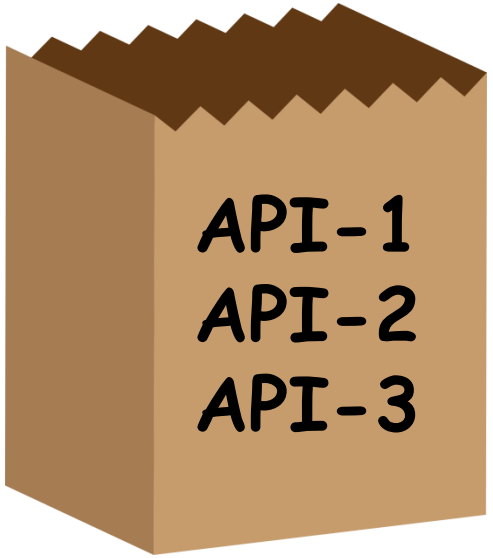
For example, for the ransomeware "Locker.apk" *(MD5: f836f5c6267f13bf9f6109a6b8d79175)*, the **API calls** of

- Ljava/io/FileOutputStream -> write
- Ljava/io/IOException -> printStackTrace
- Ljava/lang/System -> load

together in the method of "**loadLibs**" in the converted smali code indicate this ransomware intends to write malicious code into system kernel.

- To represent such kind of relationship, we generate the **API-CodeBlock** matrix **B** where each element $\mathrm{B}_{ij} = b_{ij} \in \{0, 1\}$ denotes if a pair of API calls belong to the same codeblock.

InvokeMethod




```
.method protected
loadLibs(Landroid/content/Context;)V
.locals 4
:try_start_0
new-instance v0, Ljava/io/BufferedReader;
new-instance v1, Ljava/io/InputStreamReader;
invoke-static {}, Ljava/lang/Runtime;->getRuntime()Ljava/lang/Runtime;
move-result-object v2
const-string v3, "getprop_ro.product.cpu.abi"
invoke-virtual {v2, v3}, Ljava/lang/Runtime;->exec(Ljava/lang/String;)
    Ljava/lang/Process;
move-result-object v2
invoke-virtual {v2}, Ljava/lang/Process;->getInputStream()Ljava/io/InputStre
.....
.end method
```

In the smali code, there are five different **methods to invoke an API call**:

1. invoke-static: invokes a static method with parameters;
2. invoke-virtual: invokes a virtual method with parameters;
3. invoke-direct: invokes a method with parameters without the virtual method resolution;
4. invoke-super: invokes the virtual method of the immediate parent class;
5. invoke-interface: invokes an interface method.

API calls use the same invoke method

Words have the same part of speech

like

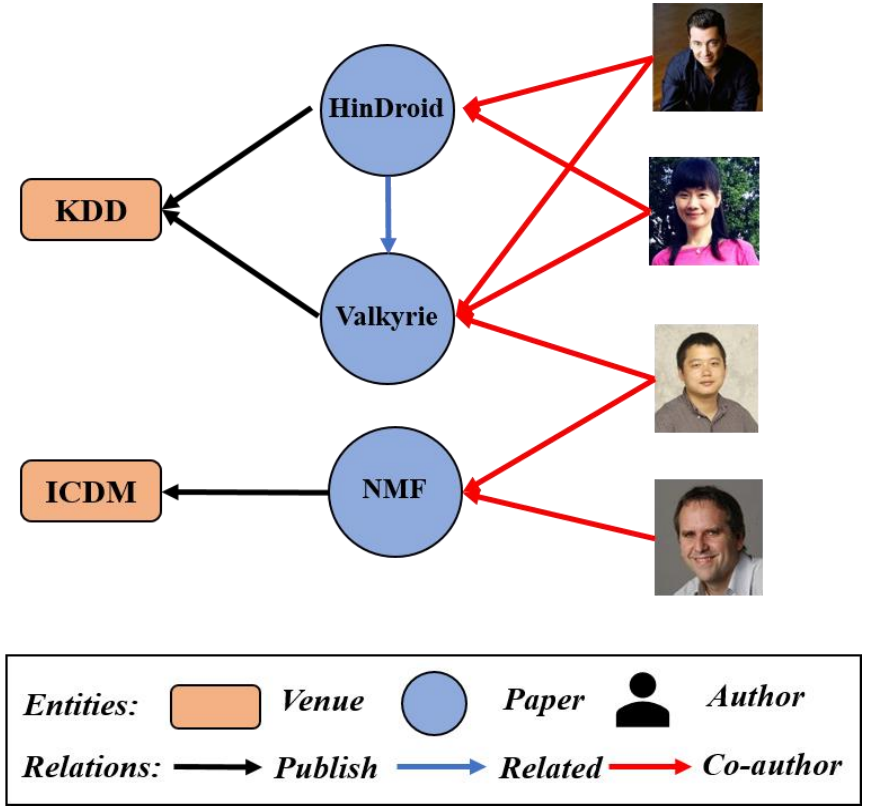
- To represent such kind of relationship, we generate the **API-InvokeMethod** matrix **I** where each element $I_{ij} = i_{ij} \in \{0, 1\}$ denotes if a pair of API calls use the same invoke method.

Heterogeneous Information Network (HIN)

HIN is capable to be composed of different types of entities and relations.

Table 1: Description of each matrix

G	Element	Description
A	a_{ij}	If app_i contains API_j, then $a_{ij} = 1$; otherwise, $a_{ij} = 0$.
B	b_{ij}	If API_i and API_j co-exist in the same code block, then $b_{ij} = 1$; otherwise, $b_{ij} = 0$.
P	p_{ij}	If API_i and API_j are with the same package name, then $p_{ij} = 1$; otherwise, $p_{ij} = 0$.
I	i_{ij}	If API_i and API_j use the same invoke method, then $i_{ij} = 1$; otherwise, $i_{ij} = 0$.

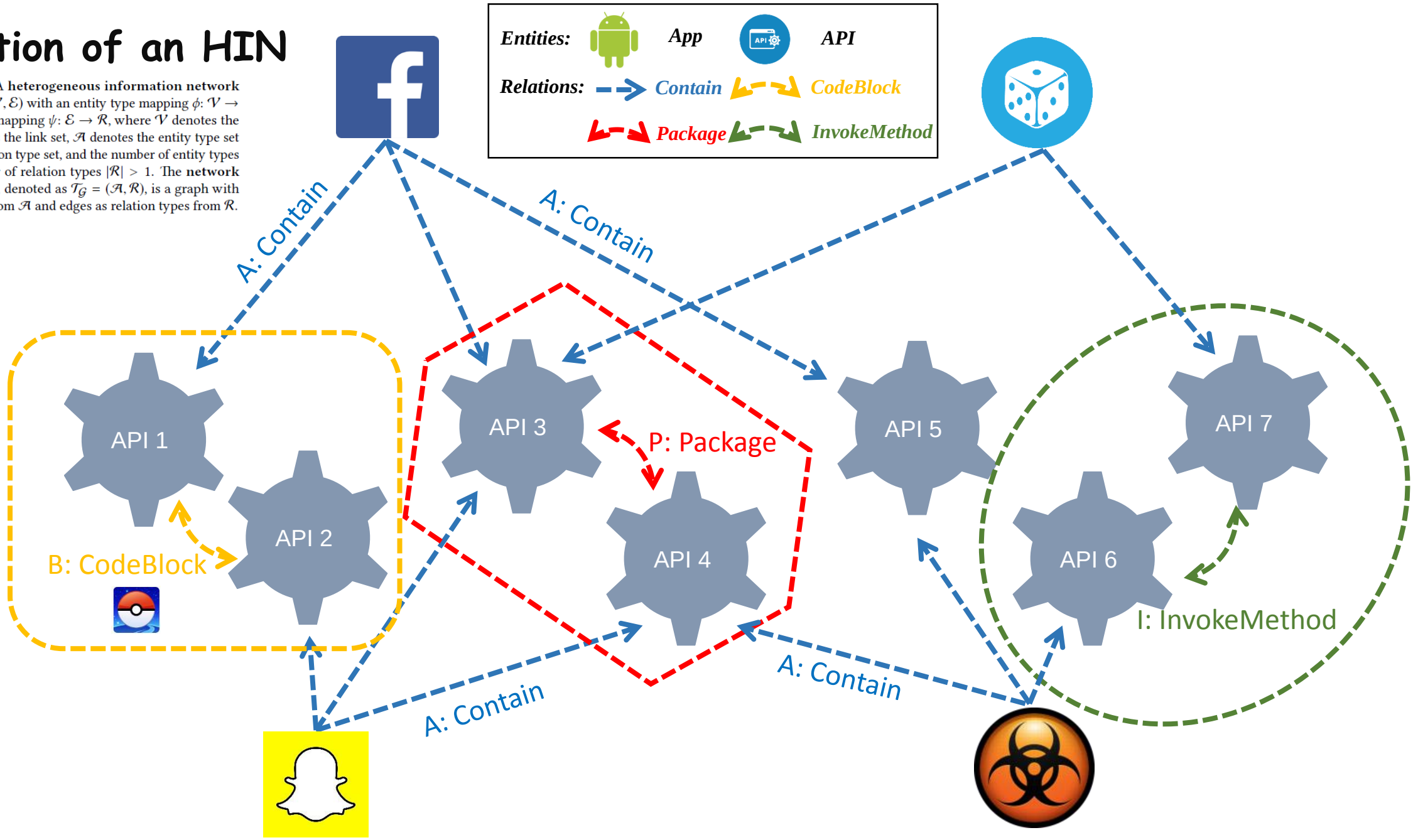


DBLP Bibliographic Network

HIN Construction and Multi-Kernel Learning

Illustration of an HIN

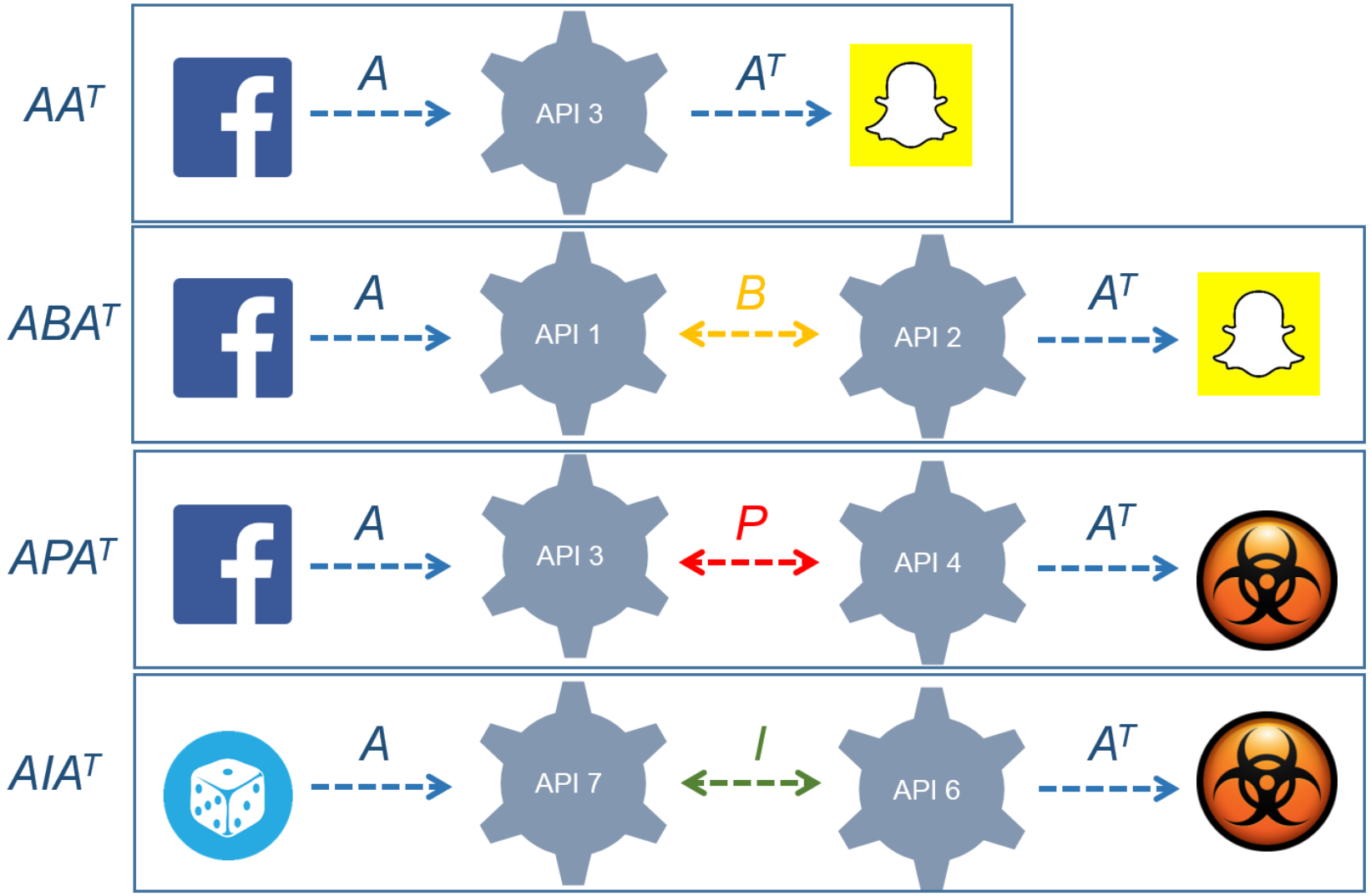
Definition 3.1. [18] A **heterogeneous information network** (HIN) is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an entity type mapping $\phi: \mathcal{V} \rightarrow \mathcal{A}$ and a relation type mapping $\psi: \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{V}$ denotes the entity set and $\mathcal{E}$ denotes the link set, $\mathcal{A}$ denotes the entity type set and $\mathcal{R}$ denotes the relation type set, and the number of entity types $|\mathcal{A}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The **network schema** for network G, denoted as $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, is a graph with nodes as entity types from $\mathcal{A}$ and edges as relation types from $\mathcal{R}$.



Meta-path Generation

Definition 3.2. [19] A **meta-path** $\mathcal{P}$ is a path defined on the graph of network schema $\mathcal{T}_G = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \dots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is the length of $\mathcal{P}$.

Definition 3.3. [19] Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its network schema $\mathcal{T}_\mathcal{G}$, a **commuting matrix** $M_\mathcal{P}$ for a meta-path $\mathcal{P} = (A_1 - A_2 - \dots - A_{L+1})$ is defined as $M_\mathcal{P} = G_{A_1A_2}G_{A_2A_3} \dots G_{A_LA_{L+1}}$, where $G_{A_iA_j}$ is the adjacency matrix between types A_i and A_j. $M_\mathcal{P}(i, j)$ represents the number of path instances between entities $x_i \in A_1$ and $y_j \in A_{L+1}$ under the meta-path $\mathcal{P}$.



Multi-kernel Learning

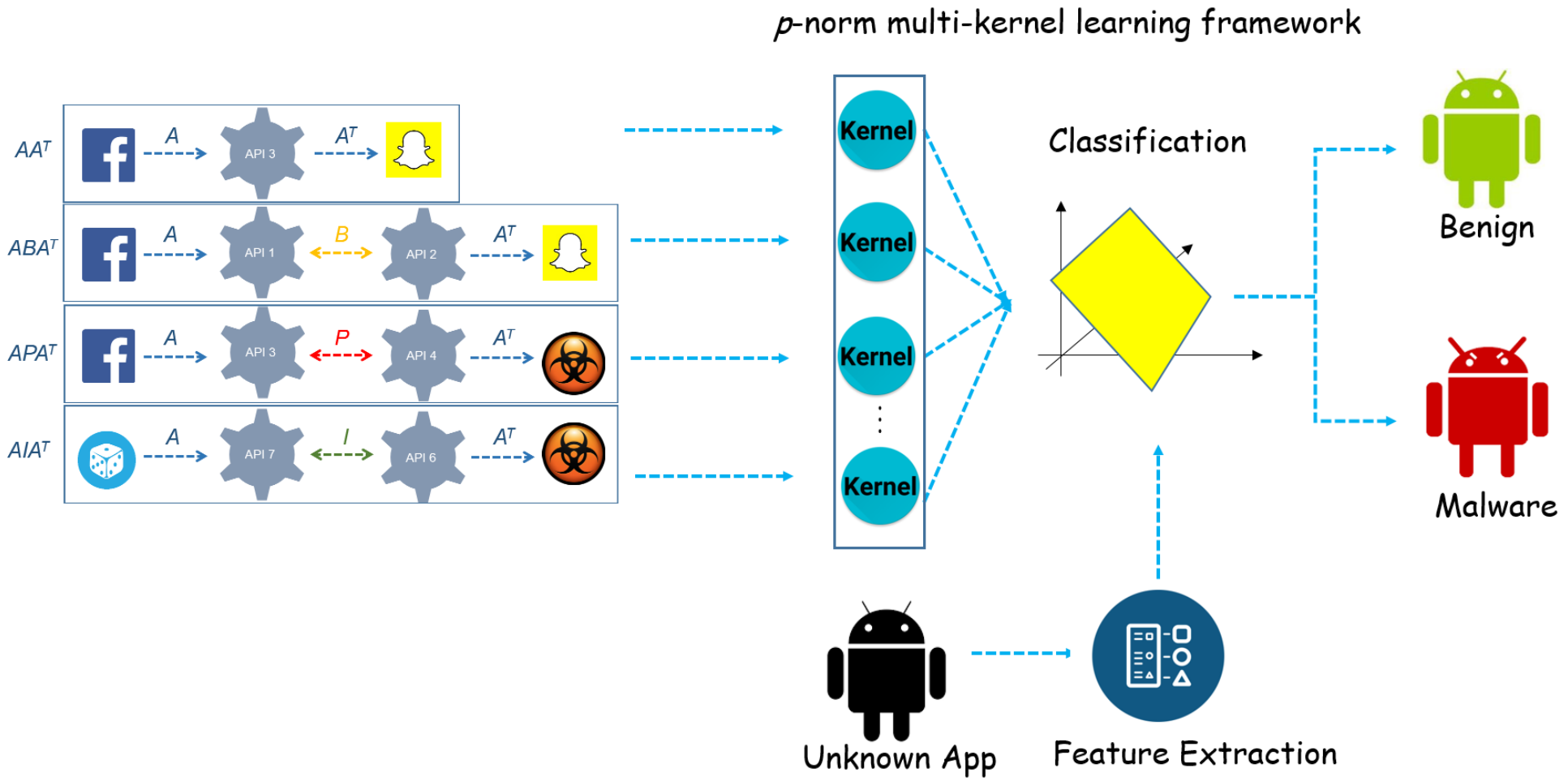
Suppose we have K meta-paths $\mathcal{P}_k, k = 1, \ldots, K$. We can compute the corresponding commuting matrices $M_{\mathcal{P}_k}, k = 1, \ldots, K$, where $M_{\mathcal{P}_k}$ is regarded as a kernel. If the commuting matrix is not a kernel (not positive semi-definite, PSD), we simply use the trick to remove the negative eigenvalues of the commuting matrix. Following [10, 16, 23], we use the linear combination of kernels to form a new kernel:

$$M = \sum_{k}^{K} \beta_k M_{\mathcal{P}_k}, \tag{1}$$

where the weights $\beta_k \geq 0$ and satisfy $\sum_{k=1}^{K} \beta_k = 1$.

To learn the weight of each meta-path, we assume we have a set of labeled data $\{x_i, y_i\}_{i=1}^{N}$, where x_i is the app (here we can regard x_i as an ID), and $y_i \in \{+1, -1\}$ is the label. Then we use the p-norm multi-kernel learning framework [23] with following objective function to learn the parameters:

$$\min_{w>0, \xi_i \geq 0, \beta_k \geq 0} \frac{1}{2} \sum_{k} ||w_k||^2 / \beta_k + C \sum_{i} \xi_i + \frac{\lambda}{2} \left(\sum_{k} \beta_k^p \right)^{2/p},$$
$$s.t. \quad y_i \left(\sum_{k} w_k^T \phi_k(x_i) + b \right) \geq 1 - \xi_i, \tag{2}$$



Experimental Results and Analysis

Experimental Setup

- Data Collection: we obtain two datasets from Comodo Cloud Security Center.

  1. The first sample set includes daily collected Android apps (through January 30, 2017 to February 5, 2017), which contains 1,834 training Android apps (920 of them are benign apps, while the other 914 apps are malware including the families of Lotoor, RevMob, Fakegupdt, and GhostPush, etc), and 500 testing samples (with the analysis by the anti-malware experts of Comodo Security Lab, 198 of them are labeled as benign and 302 of them are malicious). (**E1-E3**)
  2. The second dataset has larger sample collection containing 30,000 Android apps obtained within one month (Januray 2017), half of which are benign apps and the half are malicious apps. (**E4**)

- Performance Measures:

Table 2: Performance indices of Android malware detection

Indices	Description
TP	# of apps correctly classified as malicious
TN	# of apps correctly classified as benign
FP	# of apps mistakenly classified as malicious
FN	# of apps mistakenly classified as benign
$Precision$	$TP/(TP + FP)$
$Recall$	$TP/(TP + FN)$
ACC	$(TP + TN)/(TP + TN + FP + FN)$
$F1$	$2 * Precision * Recall/(Precision + Recall)$

E1: Detection Performance Evaluation of the Proposed Method

Table 3: Detection performance evaluation

PID	Method	F1	β	ACC	TP	FP	TN	FN
1	AA^T	0.9529	0.1069	94.40%	283	19	189	19
2	ABA^T	0.9581	0.0900	95.00%	286	9	189	16
3	APA^T	0.9495	0.0858	94.20%	273	0	198	29
4	AIA^T	0.9183	0.0623	90.40%	270	16	182	32
5	$ABPB^TA^T$	0.9479	0.0670	94.00%	273	1	197	29
6	$APBP^TA^T$	0.9502	0.0565	94.20%	277	4	194	25
7	$ABIB^TA^T$	0.8683	0.0639	84.60%	254	29	169	48
8	$AIBI^TA^T$	0.8722	0.0639	85.00%	256	29	169	46
9	$APIP^TA^T$	0.8373	0.0445	81.20%	242	34	164	60
10	$AIPI^TA^T$	0.8761	0.0572	86.60%	237	2	196	65
11	$ABPIP^TB^TA^T$	0.9184	0.0616	90.80%	259	3	195	43
12	$APBIB^TP^TA^T$	0.8597	0.0617	84.60%	236	11	187	66
13	$ABIPI^TB^TA^T$	0.9284	0.0426	91.80%	266	5	193	36
14	$AIBPB^TI^TA^T$	0.8237	0.0426	82.60%	218	3	195	84
15	$AIPBP^TI^TA^T$	0.8597	0.0469	81.60%	215	5	193	87
16	$APIBI^TP^TA^T$	0.8597	0.0458	84.60%	236	11	187	66
17	Combined-kernel (5)	0.9214	—	91.20%	258	0	198	44
18	Combined-kernel (16)	0.9740	—	96.80%	300	14	184	2
19	Multi-kernel (5)	0.9834	—	98.00%	297	5	193	5
20	Multi-kernel (16)	**0.9884**	—	**98.60%**	299	4	194	3

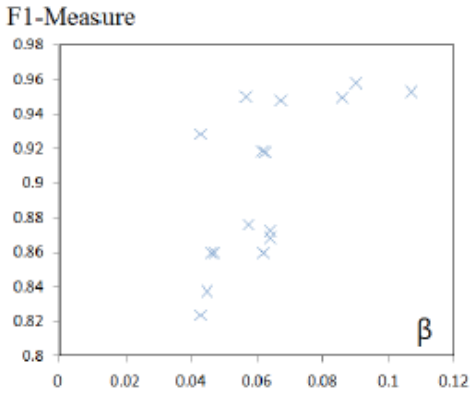


Figure 3: β_k and F1 correlation.

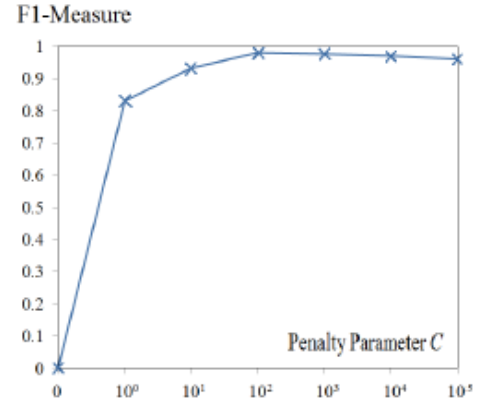


Figure 4: Parameter sensitivity evaluation.

Remark: In Figure 3, β_k is the parameter learned by multi-kernel learning, shown in Eq. (1). F1 is the actual performance of SVM using each meta-path as kernel.

- **Combined-kernel**: We rank each meta-path using its Laplacian score. The order of the ranking is: *PID12 -> PID16 -> **PID6** -> **PID3** -> **PID5** -> PID11 -> PID9 -> PID2 -> PID8 -> PID7 -> PID13 -> PID14 -> PID15 -> PID10 -> PID4 -> PID1.*

E2: Comparisons of HinDroid and other Alternative Detection Methods

Table 4: Comparisons between HinDroid and alternative detection methods. "Original" means all the algorithms use original app features (i.e., API calls) as input. "Augmented" means that, we simply put all HIN-related entities and relations as features for different algorithms to learn.

Original	F1	AUC	ACC	TP	FP	TN	FN
ANN-1	0.9173	0.9023	90.20%	272	19	179	30
NB-1	0.8514	0.8511	83.60%	235	15	183	67
DT-1	0.9202	0.9005	90.40%	277	23	175	25
SVM-1	0.9529	0.9458	94.40%	283	9	189	19
Augmented	F1	AUC	ACC	TP	FP	TN	FN
ANN-2	0.9409	0.9316	93.00%	279	12	186	23
NB-2	0.9025	0.8891	88.60%	264	19	179	38
DT-2	0.9539	0.9397	94.40%	290	16	182	12
SVM-2	0.9590	0.9537	95.20%	281	7	191	17
HinDroid	**0.9884**	**0.9849**	**98.60%**	299	4	194	3

For ANN, we use 3 hidden layers (500 neurons in each hidden layer) and train the network using back propagation. The learning rate is set to 0.3 and the momentum is set as 0.5. For SVM, we use Lit SVM in our experiment and the penalty is empirically set to be 1,000.

- To check whether the overall improvement is significant, we also run 30 random trials of training and testing examples to compare HinDroid and SVM with feature engineering, and the probability associated with a paired **t-Test** with a two-tailed distribution is 1.62×10^{-13}.
- This shows that HinDroid is significantly better than the best baseline method we compared. The reason behind this is that, in HinDroid we use more expressive representation for the data, and build the connection between the higher-level semantics of the data and the final results.
- The experiment results also demonstrates that using HinDroid can reduce the work of feature engineering, and significantly improve the Android malware detection performance.

E3: Comparisons of HinDroid and other Commercial Mobile Security Products

Table 5: Comparisons with other mobile security products

Family	Sample #	Norton	Lookout	CM	***HinDroid***
Lotoor	78	75	74	76	78
RevMob	52	46	50	48	52
Malapp	33	29	32	30	33
Fakebank	31	29	30	29	30
Generisk	29	29	29	29	29
GhostPush	19	15	16	18	18
Fakegupdt	16	15	14	14	16
Danpay	21	19	20	20	21
HideIcon	12	11	9	8	12
Idownloader	11	10	9	9	10
Total	302	278	283	281	299
DetectionRate	–	92.05%	93.71%	93.05%	**99.01%**

For the comparisons, we use all the latest versions of the mobile security products (i.e., Clean Master (CM): 2.08, Lookout: 10.9-7f33b3e, and Norton: 3.17.0.3205)

- The success of HinDroid may lie in its novel higher-level semantic feature representations as well as the multi-kernel learning based on the constructed HIN in feature engineering.
- Besides, HinDroid also has high detection efficiency: the prediction of an Andriod app is around ***3-5 seconds*** on average, including the feature extraction.

E4: Evaluations Based on Larger and Real Sample Collection from Industry

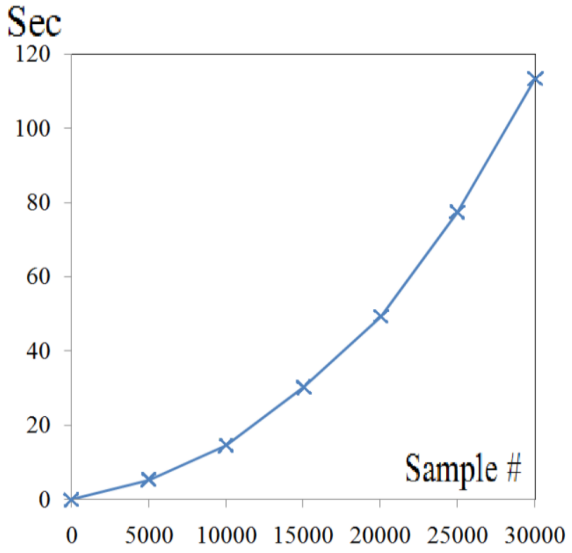


Figure 6: Scalability evaluation of *HinDroid*

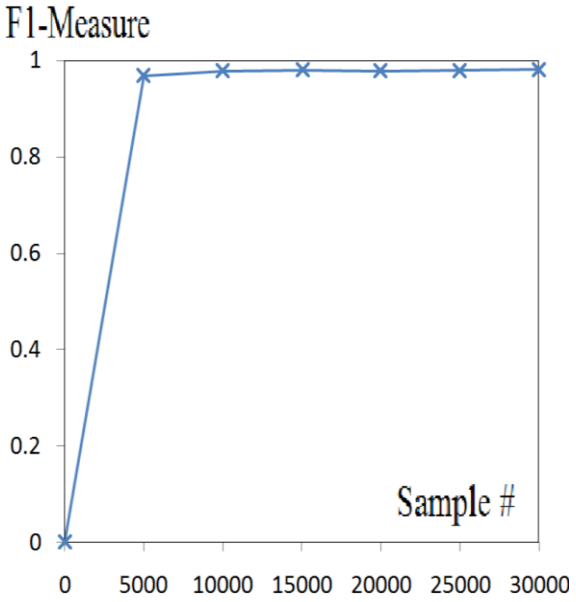


Figure 7: Comparisons when training data sizes vary

- Based on a real and larger data collection from Comodo Cloud Security Center (i.e., 30,000 Android apps obtained within one month (Januray 2017), half of which are benign apps and the half are malicious apps), Figure 5 shows the overall and zoomed-in receiver operating characteristic (ROC) curves for this experiment based on the ten-fold cross validations. From Figure 5, we can see that HinDroid achieves an impressive 0.9833 average TP rate at the 0.0087 average FP rate while labeling the newly collected Android apps.

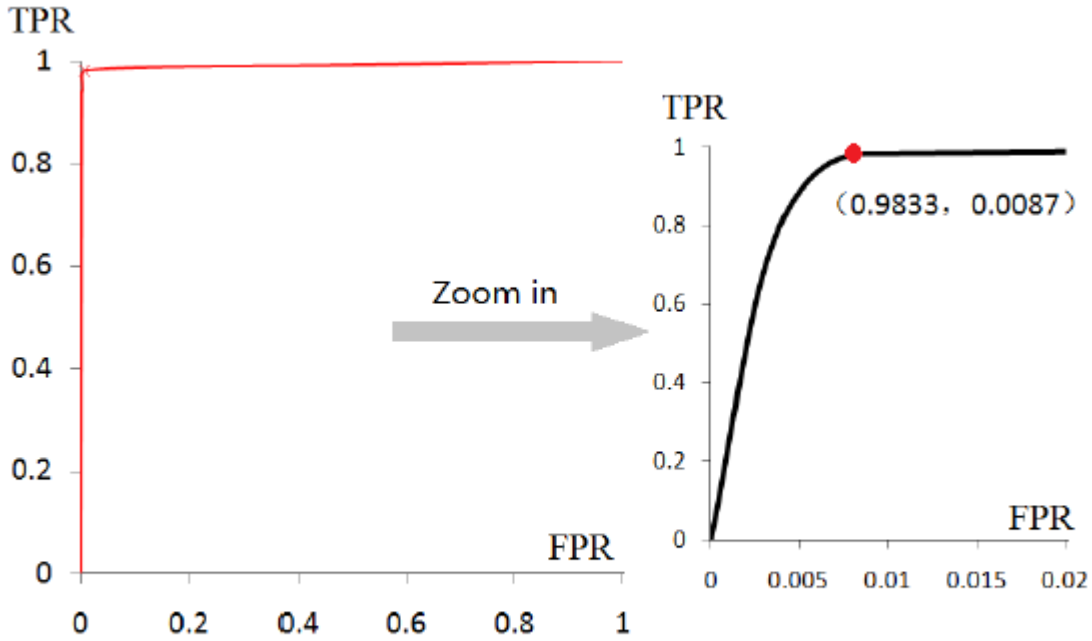


Figure 5: *Left*: ROC curve of HinDroid, *Right*: Zoomed-in.

System Deployment and Operation

- HinDroid has already been incorporated into the scanning tool of Comodo's Mobile Security Product to predict new collected 15,000 unknown apps per day. HinDroid has been deployed and tested based on the real daily sample collection for around half a year (about 2,700,000 Android apps in total have either been trained or tested).
  Note that Android malware techniques are constantly evolving and new malware samples are produced on a daily basis. To account for the temporal trends of Android malware writing, the training sets of our developed system are dynamically changing to include newly collected apps.
- Due to the high detection efficiency and effectiveness, the developed system HinDroid can greatly save human labors and reduce the staff cost: In practice, an anti-malware analyst has to spend at least 8 hours to manually analyze 40 Android apps for malware detection. Using the developed system HinDroid, the analysis of ~15,000 apps can be performed within minutes with multiple servers. This would benefit over 10 million smart phone users of Comodo's Mobile Security product.

Summary of Our Work

Related Work

- Intelligent Android malware detection systems using machine learning and data mining techniques [6–8, 29, 30]:
  - ✓ Classification/clustering based on dynamic analysis: e.g., DroidDolphin [30], Crowdroid [6], CopperDroid [22];
  - ✓ Android malware detection based on static analysis: e.g., DroidMat [29], DroidMiner [32].

  Our work: Different from the existing works [15, 29, 32], after API call extraction, we then further **analyze the relationships between them** (i.e., whether the extracted API calls belong to the same smali code block, are with the same package names, or use the same invoke method). Based on these extracted features, the **Android apps are represented by a structured heterogeneous information network (HIN)**, and a **meta-path based approach is used to link the apps**.

- **HIN** has been applied to scientific publication network analysis [17, 19, 20, 35], public general social media analysis [14, 33, 34], and document analysis based on knowledge graph [24–27]. Different from PathSim [19] and unsupervised meta-path weighting mechanisms [25, 26], in our application, the problem of Android malware detection is considered as a task of classification, thus a better idea is to jointly optimize both the classification boundary and the meta-path weights based on the provided labels (either malicious or benign). To address this challenge, in **our work**, we first use each meta-path to formulate a similarity measure over Android apps, and **aggregate different similarities using multi-kernel learning**.

Our Contributions

In this paper, we develop a system called HinDroid for intelligent Android malware detection, which has the following major traits:

- **Novel structural feature representation:** Instead of using API calls only, we further **analyze the relationships** among them. Based on the extracted features, the **Android apps will be represented by a structural heterogeneous information network (HIN)**, and a **meta-path based approach will be used to link the apps**. In this way, the detection of a malicious Android app is an aggregation of different similarities defined by different meta-paths. This is much more complicated than traditional approaches and is more difficult and costly to be evaded.
- **Multi-kernel learning for HIN:** HIN is a conceptual representation of many other kinds of data, e.g., social networks, scholar networks, knowledge graphs, etc. The similarities defined by different meta-paths can be used to make decisions in an aggregated way. In this paper, we propose a **multi-kernel learning to learn from data to determine the importance of different meta-paths**. This is a very natural way to handle HIN based similarities but to our best knowledge is a first attempt.
- **A practical developed system for real industry application:** We develop a practical system HinDroid for automatic Android malware detection and provide a comprehensive experimental study based on the **real sample collection from Comodo Cloud Security Center**, which demonstrates the effectiveness and efficiency of our developed system. HinDroid has already been incorporated into the scanning tool of Comodo Mobile Security product. The system has been deployed and tested based on the real daily sample collection (over 15,000 Android apps per day) for around half a year (about 2,700,000 Android apps in total).

Acknowledgement: We would also like to thank the anti-malware experts of Comodo Security Lab for the data collection as well as helpful discussions and supports. The work of S. Hou and Y. Ye is partially supported by the U.S. National Science Foundation under grant CNS-1618629 and WVU Senate Grants for Research and Scholarship (R-16-043). The work of Y. Song is supported by China 973 Fundamental R&D Program (No.2014CB340304).

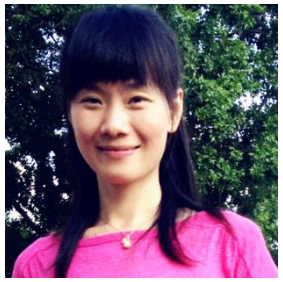
COMODO
Creating Trust Online®

NSF

WVURESEARCH

Thank you

Shifu Hou



Yanfang Ye

Yangqiu Song

Melih Abdulhayoglu

KDD2017

Committees, Reviewers, …

- To Colleagues

I and my lab (http://community.wvu.edu/~yaye/) enjoy finding simple, yet deep and elegant solutions to real-world problems that generate high impact. I very look forwards to the collaborations with **researchers** and **industry partners** in **data science** and **cybersecurity** to make this world better!

- To Perspective Students

I am currently looking for Ph.D. students doing supervised research or independent study with me at LCSEE, WVU. If you are a well motivated and dedicated student pursuing a Ph.D. degree related to the areas of Cybersecurity, Data Mining, and Smart Devices, please feel free to contact me: yanfang.ye@mail.wvu.edu.

Q & A

yanfang.ye@mail.wvu.edu