XDroid: An Android Permission Control Using Hidden Markov Chain and Online Learning

Bahman Rashidi Carol Fung
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
{rashidib, cfung}@vcu.edu

***Abstract*—Android devices provide opportunities for users to install third-party applications through various online markets. This brings security and privacy concerns to the users since third-party applications may pose serious threats. The exponential growth and diversity of these applications render conventional defenses ineffective, thus, Android smartphones often remain unprotected from novel malware. In this work, we present XDroid, an Android app and resource risk assessment framework using hidden Markov model. In this framework, we first map the applications' behaviors into an observation set, and we introduce a novel approach to attach timestamp to some observations to improve the accuracy of the model. We show that our HMM can be utilized to generates risk alerts to users when suspicious behaviors are found. Furthermore, an online learning model is introduced to enable the integration of the input from users and provide adaptive risk assessment to meet user's preferences. We evaluate our model through a set of experiments on a benchmark malware dataset DREBIN. Our experimental results demonstrate that the proposed model can assess malicious apps risk-levels with high accuracy. It also provide adaptive risk assessment based on input from users.**

I. Introduction

The proliferation of mobile apps is the primary driving force for the rapid growth of the number of the smartphone users, which is expected to double in the next 4 years from 2.7 billion in 2015 to 6 billion in 2020 [15]. On the other hand, the number of mobile apps has been growing exponentially in the past few years. According to the report by Android Google Play Store, the number of apps in the store has reached 1.8 billion in 2015, surpassing its major competitor Apple App Store [23]. As the number of smartphone apps increases, the privacy and security problem of users has become a serious concern, especially when the smartphones are used for sensitive tasks or business purposes. A malicious third-party app can not only steal private information, such as contact lists, text messages, and GPS locations from the user, but it can also cause financial loss of the user by making secretive premium-rate phone calls and text messages [21].

Malicious apps pose a severe threat to Android users. Studies show that a significant percentage ($> 70\%$) of smart phone apps request additional permissions beyond their actual need [1], [10]. For example, a puzzle game app may request for SMS and phone call permissions. In current Android architecture, users decide whether to give out the permissions to apps. However, this architecture has been proven ineffective to protect users since users tend to rush through their decisions and grant all permissions to the apps they desire. An effective malicious app detection can protect users from privacy breaching.

Malware detection techniques can be divided into static and dynamic approaches, where the former focuses on installation-time analysis of apps and the later investigates apps at runtime [7]. One main drawback of static analysis is that it does not detect vulnerabilities introduced at runtime [8]. The dynamic analysis identifies vulnerabilities at runtime and allows the analysis of applications without actual code. It also identifies vulnerabilities that might have been false negatives in the static code analysis [4]. In this paper, we study Android apps' behaviors as they run on the device, and propose a dynamic analysis method based on Hidden Markov Models (HMM) [14]. HMM can be used to model the runtime behaviors of an app, including malicious and normal ones. Existing work [5] utilizing HMM for malicious apps detection suffers from low detection accuracy. One major reason is that their work only considers the apps' intents for observations. In our work, we consider other inputs such as API calls, time and sensitive permission requests to build a comprehensive HMM. We discovered that the introduction of the time feature significantly improved the detection accuracy of the model.

In our approach, we first log the behaviors of apps through an instrumentation tool, named *DroidCat*, developed by our team. Then a filtering and parsing method is employed to synthesize and organize the captured behaviors. We train and test the HMM model using a set of known malicious apps and normal apps. Our experimental results demonstrate that our proposed model achieves high accuracy in terms of detecting malicious apps. The major contributions of this paper include: 1) An instrumentation tool that facilitates app behavior logging to generate high quality dataset for analysis. 2) A comprehensive time-aware Android app behavior analysis, which is based on the apps' intents and actions, as well as extra features that further improves detection accuracy. 3) A trained Hidden Markov Model is used to decide whether an app is malicious or not based on their behaviors. 4) A dynamic model that can be updated in real time to adapt to users' preferences.

To the best of our knowledge, this is the first time that a HMM online learning model is used on malicious app control in smartphone security.

The rest of the paper is organized as follows: Section II describes the system design; Section III describes our proposed model, capturing apps' logs, training and testing the model and on-line parameter updating strategy; we present our evaluation results and the impact of the risk computing parameters in Section IV; related work overview is in Section V; Discussion is in Section VI, and finally section VII concludes this paper.

II. System Design

The ultimate goal of XDroid is to monitor the behavior of probated apps and generate alerts to users when suspicious app behaviors are detected. Fig. 1 shows an architecture design of XDroid. The system contains components on the server side and the mobile device side. Each XDroid device contains an *Interaction Portal* and an *Activity Logger*. The interaction portal provides interface for users to interact with the device. The activity logger is used to monitor the activities of the probated apps. The server side components include *Risk Assessment*, *User Profiling*, and *Alert Customization*. In the rest of this section, we describe the key features of the server.

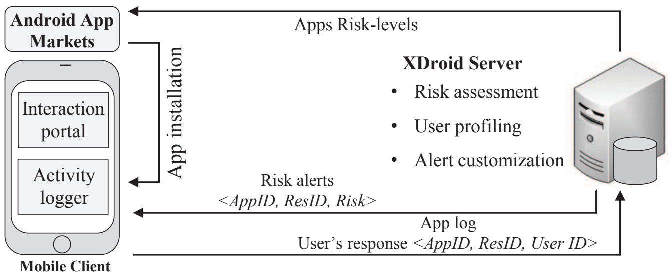


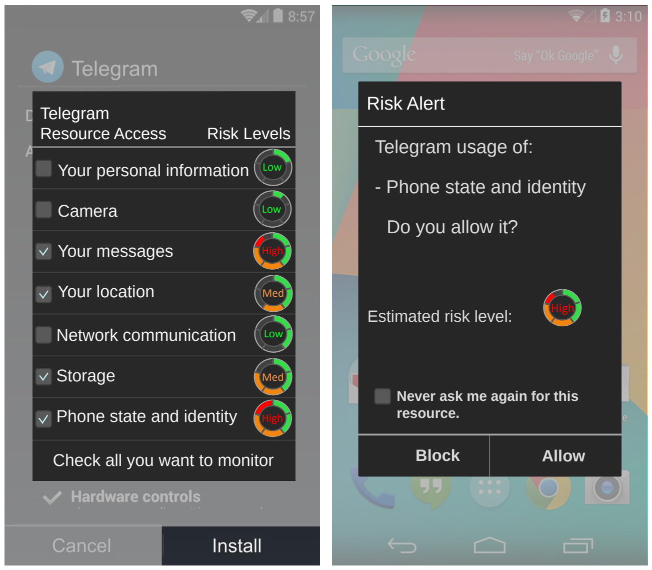
Fig. 1. XDroid system overview

1) Interaction Portal: The interaction portal is to facilitate the interaction between users and devices. For example, when a user installs "Telegram" (a popular messaging application) under probation mode, a set of requested resources are displayed along with their estimated risk-levels (Fig. 2(a)). Users can check resources they want to monitor. If a resource is monitored and suspicious activities related to it are detected, the user is informed through a dialog box (Fig. 2(b)). Users can decide whether to block the resource access or allow it based on the estimated risk suggested by XDroid.

2) Risk Assessment: The purpose of the risk assessment is to provide a quantitative estimation on how likely a resource access from an app causes damage to users. For example, a SMS access from a puzzle game app may be malicious and XDroid can pop up a reminder for users to block it.

To assess the risk-level of resource accesses, the activities of the apps are monitored by activity loggers and the data is sent to the server for analysis. Our risk assessment mechanism uses a Hidden Markov Model (HMM) to analyze the behavior sequences (Section III) and provides users with a risk-level of involved resource accesses.

3) User Profiling: We are aware that users may have different tolerance level on various resources. For example, user A is very concerned with leaking his/her location to a third party, while user B does not care about it. To provide


Telegram
Resource Access Risk Levels
☐ Your personal information Low
☐ Camera Low
✓ Your messages High
✓ Your location Med
☐ Network communication Low
✓ Storage Med
✓ Phone state and identity High
Check all you want to monitor

Risk Alert
Telegram usage of:
- Phone state and identity
Do you allow it?
Estimated risk level: High
☐ Never ask me again for this resource.

Fig. 2. User Interfaces: (a) illustrates the risk computed risk-levels for app's requested resources; (b) shows a popup notifying user the risk-level of resource at runtime

customized risk estimation, we build user profiles. We assign each new user with an initial tolerance model and update the user profile after receiving the users' permission control decisions. For example, if a user agrees GPS access every time is it requested, the tolerance level for the user on GPS access is high.

4) Alert Customization: The purpose of XDroid user profiling is to provide customized risk assessment to users. Upon installing a new app, XDroid adjust the risk-level by integrating the tolerance level of each user. For example, user C has much higher tolerance to GPS than user D. When a new app is installed and a GPS request raises, XDroid shows a lower risk-level to user C than to user D.

III. Hidden Markov Model for Risk Assessment

In this work, we use Hidden Markov Model (HMM) for Android malicious apps risk assessment. We transform the app resource risk computation problem into a HMM problem with two states, *malicious* and *normal*, and we map the app behaviors into the HMM observations. To train the HMM model, we capture the behaviors from both malicious and normal apps' and use them to generate an initial trained HMM for risk estimation. In this section we first present our HMM model and then explain how we use it for customized permission risk-level estimation.

A. Hidden Markov Model

A Hidden Markov Model (HMM) is a statistical Markov model which is widely used in science, engineering and many other areas (speech recognition, optical character recognition, machine translation, bioinformatics, computer vision, finance and economics, and in social science) [14]. Markov Models provide powerful abstraction for data expressed in time series, but fail to provide reasoning about a series of states given some observations related to those states. A HHM can be used to solve the above problem. In this paper we model the Android behavior sequence into a HMM, and compute the risk-levels of given resource usage.

The Hidden Markov Model is a variant of a *finite state machine* which can be represented by a set of hidden states $\mathcal{Q} = \{q_1, q_2, ..., q_{|Q|}\}$, a set of observations $\mathcal{O} = \{o_1, o_2, ..., o_{|O|}\}$, a set of transition probabilities $\mathcal{A} = \{\alpha_{ij} = P(q_j^{t+1}|q_i^t)\}$, and a series of output (emission) probabilities $\mathcal{B} = \{b_{ik} = P(o_k|q_i)\}$. Among the above notations, $P(a|b)$ is the conditional probability of event a given event b; $t = 1, ..., T$ is the time; q_i^t is the event that the state is q_i at time t. In other words, α_{ij} is the probability that the next state is q_j given that the current state is q_i; b_{ik} is the probability that the output is o_k given that the current state is q_i. The initial state probabilities are denoted by $\Pi = \{\pi_i = P(q_i^1)|\forall 1 \leq i \leq |Q|\}$, which is the initial probability of all states at time 1 (initial time). In HMM the current state is not observable. However, each state produces an output with a certain probability (denoted by $\mathcal{B}$). An *HMM* can also be represented using a compact triple ($\lambda = (\mathcal{A}, \mathcal{B}, \Pi)$) [20]. Table I shows the notations and preliminaries.

TABLE I
NOTATIONS

Notation	Description
$\mathcal{Q}$	$\{q_i\}, i = 1 \cdots, N$: Set of n hidden states .
$\mathcal{A}$	$\mathcal{A} = \{\alpha_{ij} = P(q_j^t)\}$ Transition probabilities
$\mathcal{O}$	$\mathcal{O} = \{o_k\}, k = 1, \cdots, M$: Observations (symbols)
$\mathcal{B}$	$\mathcal{B} = \{b_{ik} = P(o_k|q_i)\}$: Emission probabilities
Π	$\Pi = \{\pi_i = P(q_i^1)|\forall 1 \leq i \leq |Q|\}$ Initial state probabilities.
O^t	$O^t \in \mathcal{O}$: Observation at time t.
Q^t	$Q^t \in \mathcal{Q}$: State at time t.

Fig. 3 illustrates the HMM for Android app behavior modeling. The HMM consists of three states: *Start*, *Normal* (0) and *Malicious* (1). The set of observations are defined using apps' behaviors during runtime (see Section III-D). The HMM parameters are: initial state probabilities $\Pi = [\pi_1, \pi_2]$, state transition probabilities $\mathcal{A} = [\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11}]$, Malicious state emission probabilities $\mathcal{B}_M = [b_{11}, b_{12}, \cdots, b_{1N}]$ and Normal state emission probabilities $\mathcal{B}_N = [b_{01}, b_{02}, \cdots, b_{0N}]$.

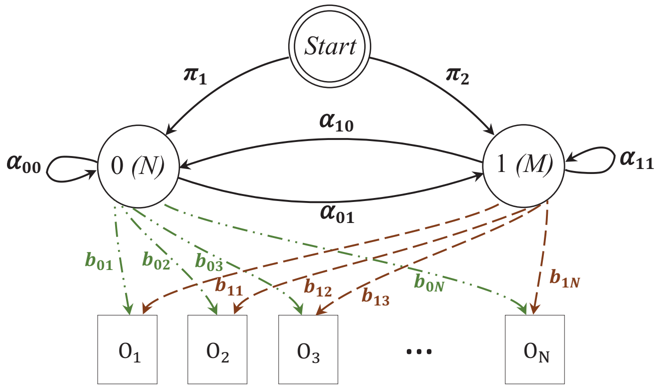


Fig. 3. An overview of the proposed HMM model

However, how to determine the unknown parameters and how to find optimal state sequence of HMM given a sequence of observations. We describe the two problems and their solutions in the next sections.

B. Compute Unknown Parameters

One of the challenge of utilizing the HMMs to model the behavior of Android apps is to define parameters $\mathcal{Q}, \mathcal{A}, \mathcal{O}, \mathcal{B}, \Pi$, and find the most likely set of state transition and output probabilities. We use two states $\mathcal{Q} = \{0, 1\}$ to denote the app is behaving normal (0) or malicious (1). The observation set $\mathcal{O}$ is the set of time-aware system calls expressed using a keywords set (see Section III-D). To define $\mathcal{A}$ and $\mathcal{B}$, we use Baum-Welch algorithm (a.k.a Forward-Backward algorithm) [19]. We call it *HMMs Training*. The algorithm has two main steps, Forward and Backward procedures.

1) Initialization set: $\lambda = (\mathcal{A}, \mathcal{B}, \Pi)$ with random initial conditions. The algorithm updates the parameters of λ iteratively until convergence, following the next procedures.

2) The forward procedure: We define $\alpha_i^t = P(O^1, \cdots, O^t, Q^t = q_i|\lambda)$, which is the probability of seeing the partial sequence $O^1, \cdots, O^t$ and the ending state Q^t at time t is q_i. We can compute α_i^t recursively as follows:

$$\alpha_i^1 = \pi_i b_i(O^1) \tag{1}$$

$$\alpha_j^{t+1} = b_j(O^{t+1}) \sum_{i=1}^{N} \alpha_i^t \alpha_{ij} \tag{2}$$

where $b_i(O^j)$ is the probability that observation O^j at time j given state i.

3) The backward procedure: We define β_i^t to be the probability of the ending partial sequence $O^{t+1}, \cdots, O^T$ given that we started at state q_i, at time T. We can efficiently compute β_i^t as:

$$\beta_i^T = 1 \tag{3}$$

$$\beta_i^t = \sum_{j=1}^{N} \beta_j^{t+1} \alpha_{ij} b_j(O^{t+1}) \tag{4}$$

using α and β, we can compute the following variables:

$$\gamma_i^t \equiv P(Q^t = q_i|O, \lambda) = \frac{\alpha_i^t \beta_i^t}{\sum_{j=1}^{N} \alpha_j^t \beta_j^t} \tag{5}$$

$$\xi_{ij}^t \equiv P(Q^t = q_i, Q^{t+1} = q_j|O, \lambda) = \frac{\alpha_i^t \alpha_{ij} \beta_j^{t+1} b_j(O^{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i^t \alpha_{ij} \beta_j^{t+1} b_j(O^{t+1})} \tag{6}$$

where γ_i^t is the probability that the state at time t is q_i, and ξ_{ij}^t is the probability the state at t is Q_i and the state at $t+1$ is q_j. Let function $\delta(x, y)$ be 1 if $x = y$ and 0 otherwise. With γ and ξ, we can define update rules as follows:

$$\pi_i = \gamma_i^1 \tag{7}$$

$$\alpha_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}^t}{\sum_{t=1}^{T-1} \gamma_i^t} \tag{8}$$

$$b_{ik} = \frac{\sum_{t=1}^{T} \delta(O^t, o_k) \gamma_i^t}{\sum_{t=1}^{T} \gamma_i^t} \tag{9}$$

C. *Finding the Optimal State Sequence*

One of the most common queries of a *HMM* is to find the most likely series of states given an observed series of observations. In our case, we can find the state sequence (e.g., NMNNNNMN...) that most likely happens given the observation sequence. This problem can be solved using Viterbi algorithm [25]. The Viterbi algorithm is similar to the forward procedure except that it only tracks the maximum probability instead of the total probability.

Let δ_i^t be the maximal probability of state sequences of the length t that end in state i and produce the t first observations for the given model. That is,

$$\delta_i^t = max\{P(Q^1,\cdots,Q^{t-1};O^1,\cdots,O^t|Q^t=q_i)\} \tag{10}$$

The two difference between Viterbi algorithm and the Forward algorithm are: (1) it uses maximization in place of summation at the recursion and termination steps, (2) it keeps track of the arguments that maximize δ_i^t for each t and i, storing them in the N by T matrix ψ. This matrix is used to retrieve the optimal state sequence at the backtracking step.

We initial the model as:

$$\delta_i^1 = \pi_i b_i(O^1) \tag{11}$$

$$\psi_i^1 = 0, i = 1,\cdots,N \tag{12}$$

The recursion steps are:

$$\delta_j^t = max_i[\delta_i^{t-1}a_{ij}]b_j(O^t) \tag{13}$$

$$\psi_j^t = argmax_i[\delta_i^{t-1}a_{ij}] \tag{14}$$

Finally the most probable sequence's probability $p(T)$ and the most probable last state $q(T)$ given O are:

$$p(T) = max_i[\delta_i^T] \tag{15}$$

$$q(T) = argmax_i[\delta_i^T] \tag{16}$$

We can have the path (state sequence) through backtracking:

$$q(t) = \psi_{q(t+1)}^{t+1}, t = T-1, T-2,\cdots,1 \tag{17}$$

D. *Observations*

Our vision to specify program behavior is to view the running app as a black-box and focus on its interaction with the Android OS. In this case, a typical interface to monitor is the set of system and API calls that this app invokes during its running time. Every action that involves communication with the apps' resource (e.g., accessing the file system, sending SMS/MMS over the network, accessing the location services, calling Ads APIs or accessing the network) requires the app to launch operating system services or API calls.

In order to instrument apps to capture the behavior logs, we developed our own tool *DroidCat*. We did not choose the existing instrumentation tools such as Robotium and uiautomator because of their drawbacks. For example, Robotium cannot handle Flash or Web components nor simulate the clicking on soft keyboard, and it is not suitable for multi-process applications tests. Therefore, we decided to develop DroidCat. Fig. 4 illustrates the XDroid system with DroidCat design integrated. The main merit of DroidCat is that it can instrument apps through real human-interaction, so we can get behavior logs which highly assemble real world executing of

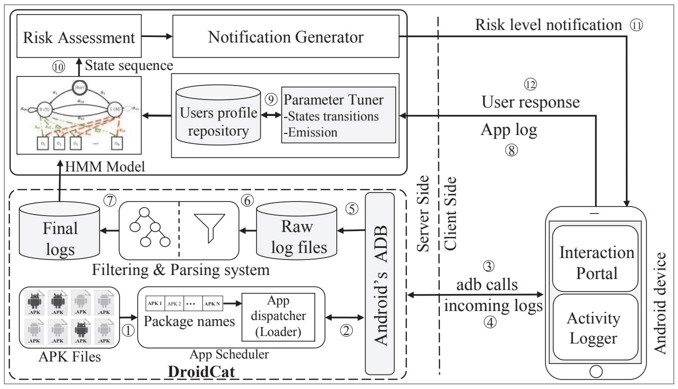


Fig. 4. The architecture of the XDroid system

Android apps. We plan to make the DroidCat's source code free to download online. In the rest of this section we briefly explain the major instrumentation process of DroidCat, which includes the following steps:

1) Extracting packages' names: DroidCat extracts Android packages' names using Android `aapt` tool. This tool is part of the SDK (and build system) and allows us to view, create, and update Zip-compatible archives (`zip`, `jar`, `apk`). This component creates a queue and add the packages' names into it. As you can see, this component is embedded into *App Scheduler*.

2) App dispatcher: DroidCat automatically load Android `apk` files into an Android device, installs and runs them using the Android ADB `logcat` tool and add it to the *App Scheduler*. This component has a timer that lets apps to run for a *specific time period*. It does the same process for all apps.

3) Recording apps' logs: DroidCat records all apps' logs into a text file (raw log files) separately without applying any of `logcat` priority filtering tags (e.g., V, D, I etc.). We use apps' PID as an identifier to capture their logs.

4) Filtering: In this step DroidCat filters the logs and removes unrelevant logs such as the logs related to loading, installing, running and killing process of apps.

5) Parsing: After filtering the log files, DroidCat eliminates unnecessary information and extract important keywords. Each keyword refers to a sensitive resource access request, an API call, or Android action constants. We can also call them *observations*. In our model, we focus not only on the generated Intents by apps but also on API libraries that generate unwanted Ads or cause permission escalation. In total we defined 150 keywords under various categories. Table II lists some selected sample keywords from 6 categories. When analyzing the parsed logs, we noticed that for some resources such as "WiFi", malicious and normal apps have different patterns in the timing of requests during app running. For example, the malicious apps tend to request the WiFi network during the first quarter of their running time period. Because of this, we include the timing of requests or library calls as an additional feature. Among of the 150 keywords, we added the timing feature to 55 of of them. Therefore, we defined 205 *time-dependent* and *time-independent* observations in total.

TABLE II
KEYWORD SAMPLES

Resource	Corresponding Keywords
Ads libraries	'AdMob', 'Ads', 'Wooboo', 'AdsMOGO', ...
Network	'browser', 'http', 'wifi', ...
Messaging	'MMS', 'SMS', 'MmsService', 'getSmsCount', ...
Location services	'GPS', 'ACCESS-COARSE-LOCATION', 'location', ...
File system	'mount', 'unmount', 'Storage', 'Modify' ...
Calling/Contacts	'CallLOG', 'CARRIER', 'INCALL', 'TELECOM', ...

E. Model Training and Testing

After defining observations and parsing logs into sequences of *time-dependent* and *time-independent* observations, we train the HMM model using the Baum-Welch algorithm. In order to do it we need to define the initial state transition probabilities $\mathcal{A}$ and emissions probabilities $\mathcal{B}$. We compute the initial emissions probabilities of each observation based on its frequency of occurrence in all malicious and normal apps' logs. We also initialize all the states' transitions probabilities with fixed value 0.5. After initialization, we apply the Baum-Welch algorithm to find the parameters $\mathcal{A}$ and $\mathcal{B}$ given sequences of observations.

To validate the accuracy of the HMM-based risk computation, we use an app's behavior log as input to the *Viterbi algorithm* to get the most possible state sequence of the app. Fig. 5 visualizes the test process based on the Viterbi algorithm. The Viterbi algorithm uses backtracking method (Eq. 17) to compute the optimal state sequence path given observations, such as (MNN$\cdots$NMM).

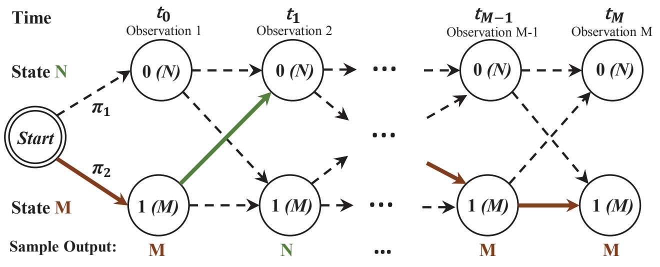


Fig. 5. An illustration of the Viterbi algorithm

F. Malicious Apps Classification

One purpose of the HMM is to classify apps into benign or malicious based on their behavior log. We consider the optimal state sequence to be the mostly likely behavior sequence of an app. Note that a malicious app may behave malicious and normal interchangeably. Our decision model uses a simple threshold θ, which represents the percentage of Ms in the state sequence, to divide malicious apps and benign apps. If the percentage of Ms in an app's state sequence is higher than the threshold θ, then we classify the app to be malicious.

G. Permission Risk Assessment

Another purpose of HMM is to compute the risk of the resource access from the apps. In this subsection we explain the process of the permission risk assessment and explain how we can perform user profiling and customized alerts.

1) Resource risk assessment: To assess the risk of each requested resource from an app, we keep tracking the observation logs and users' decisions on permission requests (allow or block). An online HMM learning technique is used to update the HMM parameter sets for each app. Note that the initial HMM parameters for each app are learned from the base training dataset and the HMM will be refined further through integrating inputs from users.

Let S be the optimal state sequence path given observations. We estimate the risk-level of a permission as follows:

$$R = \frac{\sum_{i=1}^{|S|} \delta(S_i, M)}{|S|} \tag{18}$$

where $\sum_{i=1}^{|S|} \delta(S_i, M)$ is the number of malicious state in the sequence, and $|S|$ is the length of the sequence.

2) User profiling: We record all users' profiles into the repository on the serve. Each user profile includes a customized HMM parameter tuner, users' past responses to apps' permission requests, apps' behavior logs reported by the user's device, and the risk-levels for all apps and their corresponding resources.

3) Activity logger: The XDroid client contains an Activity logger to capture the apps' logs and report them to the XDroid server. One way to implement this is to modify the `ContextImpl.java` class of the context component of the Android. This class is called whenever an application seeks to use some resources that are not hardware related. On the server side DroidCat is used to filter and parse the logs (Fig. 4), which will be the input source for the HMM model.

4) Customized alert generator: This component is responsible for generating customized alerts and send them to each user. Alerts are displayed though popups (Fig. 2(b)). Note that the risk-level of the same permission request will be tuned based on each user's past tolerance level to that type of requests. For example, a GPS call may be acceptable for one user but a big concern for the other.

H. Parameter updating through online learning

In order to integrate users' responses (preferences) to the generated risk alerts, we use an online learning technique. This way we can customize the risk computation for each user. The online learning techniques for HMM parameters can optimize the log-likelihood function. These techniques are derived from another method which uses batch input for computation. However the key difference of online learning is that the HMM parameter updating are based on the currently presented subsequence (observations from apps) of observations without iterations [14], [13]. Fig. 6 illustrates the learning scenario where data blocks (apps observations) are used to update the model in an incremental fashion. The HMM model is updated periodically after a certain amount of apps' logs are collected. Let $D_1, D_2, \cdots, D_n$ be the blocks of training data available to the model at time $t_1, t_2, \cdots, t_n$. The update process starts with an initial risk value m_0 which constitutes the prior knowledge of the domain through training process with existing datasets. During the incremental training, m_0 is updated to m_1 with input D_1, and so on. In our model, a data block contains a sequence of observations related to a resource with which the risk-level is computed. Therefore, the updating process tunes the corresponding emissions probabilities based on the users responses to risk alerts.

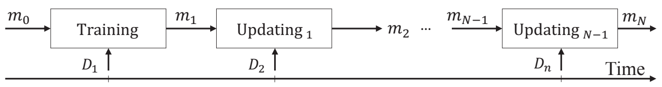


Fig. 6. Training ans updating process using an on-line technique

The on-line learning technique proposed by Mizuno et al. [17] is based on the Baum-Welch algorithm. It applies a decayed accumulation of the state densities and a direct update of the model's parameters after each subsequence of observations. Starting with an initial model λ_0, the conditional state densities are iteratively computed after processing each subsequence (r) of observations of length T by:

$$\sum_{t=1}^{T-1} \xi_{r+1}^t(i,j) = (1-\eta_r) \sum_{t=1}^{T-1} \xi_r^t(i,j) + \eta_r \sum_{t=1}^{T-1} \xi_{r+1}^t(i,j) \quad (19)$$

$$\sum_{t=1}^{T} \gamma_{r+1}^t(j)\delta(O^t, o_k) = (1-\eta_r) \sum_{t=1}^{T} \gamma_r^t(j)\delta(O^t, o_k) + \eta_r \sum_{t=1}^{T} \gamma_{r+1}^t(j)\delta(O^t, o_k) \quad (20)$$

where the model parameters are directly updated using equations (8,9). The learning rate (forgetting factor) η_k is expressed in polynomial form $\eta_r = (\frac{1}{t})$. The η_k parameter be initialized manually or automatically (*time dependent*). Time-dependent means that the impact on the learning rate will discount through time. Using this parameter we can control the impact of the former models on the updating process.

IV. Evaluation

In this section we present our experiments to evaluate the proposed model. We first explain the experimental setup and then results on the performance of the HMM based risk assessment model. First, we validate the accuracy of the model in terms of recognizing malicious apps from normal apps; second, we carried out a few experiments to evaluate the computed risk-levels and the parameters impact.

A. Experiment Setup

In this subsection we describe the experiments environment, hardware, software and the dataset that we used to train the model and test it in terms of accuracy.

a) Hardware: To collect the app behaviors we used 5 LG Nexus 4 devices equipped with Android OS version 4.3. We chose Android 4.3 version because all apps in our datasets are compatible with this version. We also configured the devices and turned on all sensitive resources (services) such as WiFi, Bluetooth and GPS. We run our DroidCat on a 64-bit Windows machine with 3.30GHz Intel Xenon, 16G RAM.

b) Software: Our experiment environment is MATLAB 2015 running on a same machine. We implemented the Baum-Welch algorithm with default tolerance 1e-4 and the Viterbi algorithm to train and test the model. The Baum-Welch algorithm specifies the tolerance used for testing convergence of the iterative estimation process and controls how many steps the algorithm executes before the function returns an answer.

c) Datasets: To have an effective HMM risk assessment model, we need to train the model with sufficient behavior logs from both Android malicious apps and normal apps. We obtained our malicious apps set from the Computer Security Group of University of Gttingen, which was collected under the Drebin project [3]. The dataset contains 5560 malicious apps from 179 different malware families. We selected 700 apps from this dataset so that we have multiple apps from all the malware families. In addition to the 700 malicious apps, we collected 700 benign apps from various categories of Android apps. We randomly selected 500 malicious and 500 benign apps from both datasets (malicious and benign) and use them as *training sets* for the model, and the remaining 200 apps from each dataset are used as *test set* to test the performance of the model.

In terms of behavior logging, we set the timer in the DroidCat's App Scheduler to $2-5$ minutes per app and it took around 79 hours to capture all logs through human interactions with the 1400 apps.

B. Model Accuracy and Reliability

In the first experiment, we study the accuracy of the model. After training the HMM, we measure the true positive rate (TP) and false positive rate (FP) regarding the computed risk-levels' accuracy. TP refers to probability that malicious apps risk-levels are computed correctly by the model, whereas FP is the probability that a benign app risk-level is falsely computed. Note that true negative rate (TN) and false negative rates (FN) can be derived from TP and FP. We start with the risk threshold from 0 and increase it by 0.05 each round till it reaches 1. Fig. 7(a) shows that TP and FP drop when the risk threshold increases. The FP and FP drop rate increase drastically when the threshold passes 0.5 and 0.6, respectively. From Fig. 7(b) we can see that the TP and TN cross at around 90% accuracy when the threshold is around 0.7.

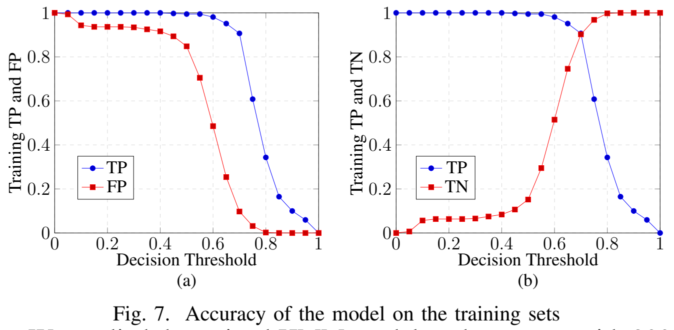


Fig. 7. Accuracy of the model on the training sets

We applied the trained HMM model to the test set with 200 malicious apps and 200 benign apps, and show the results in Fig. 8. As we can see, the accuracy is slightly lower than the ones on the training set. From this experiment, we can see that our model achieves high accuracy to compute the malicious apps' risk-levels. When the risk threshold θ is 0.7, the risk assessment system achieves 88% accuracy on TP and TN on the training set and 80% TP and TN on the test set. In the merged results presented in Fig. 9, we can see that the false positive rate is higher in the test set than the training set.

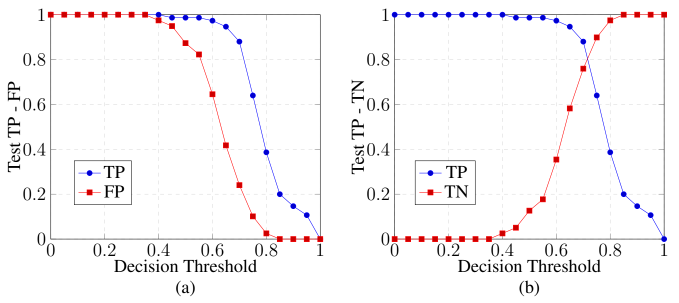


Fig. 8. Accuracy of the model on the test sets

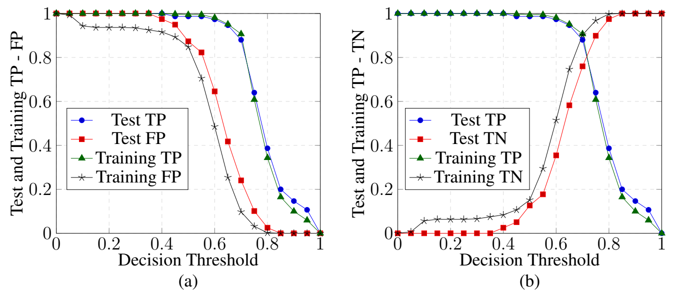


Fig. 9. Accuracy of the model on both test and training sets

Considering that 0.7 is an ideal risk threshold, we carried out another experiment to study the influence of training dataset size on the TP and TN. In this experiment, we use cross validation by splitting the training data into different sizes. We train the HMM model with one set and test the result using the other. The size of the training set start from 100 and increases by 100 each round. Fig. 10 shows that the TP and TN increase with the size of the training dataset. We also see that when the training datasets reaches 800 the accuracy of risk computation does not get better by increasing the training dataset, which means the training dataset of 800 (400 malicious apps and 400 benign apps) is sufficient.

Table III presents the performance of the model on the test dataset in terms of Recall, Precision, F-Measure and Accuracy for all ten training dataset sizes. We can see that all performance indicators increase with the training dataset size until it reaches 800.

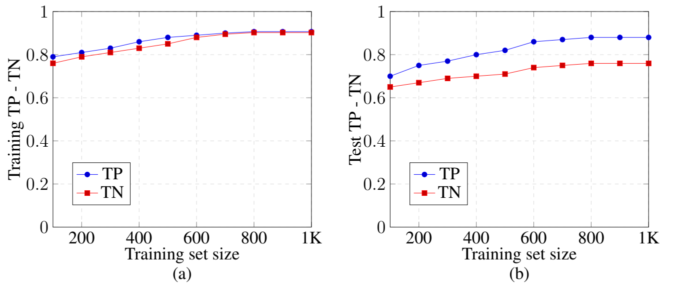


Fig. 10. Accuracy of the model on both test and training sets with different set sizes

TABLE III
PERFORMANCE MEASUREMENT - RECALL (RC), PRECISION (PR), F-MEASURE (F), ACCURACY (AC)

Size	TP	TN	FP	FN	Rc	Pr	F	Ac
100	0.70	0.65	0.35	0.30	0.70	0.67	0.68	0.68
200	0.75	0.67	0.33	0.25	0.75	0.69	0.72	0.71
300	0.77	0.69	0.31	0.23	0.77	0.71	0.74	0.73
400	0.80	0.70	0.30	0.20	0.80	0.73	0.76	0.75
500	0.82	0.71	0.29	0.18	0.82	0.74	0.78	0.77
600	0.86	0.74	0.26	0.14	0.86	0.77	0.81	0.80
700	0.87	0.75	0.25	0.13	0.87	0.78	0.82	0.81
800	0.88	0.76	0.24	0.12	0.88	0.79	0.83	0.82
900	0.88	0.76	0.24	0.12	0.88	0.79	0.83	0.82
1000	0.88	0.76	0.24	0.12	0.88	0.79	0.83	0.82

C. Risk Evaluation

In this subsection we study the risk-levels for apps and their resources through experiments. We also discuss the risk computation and the impact of users' preferences on the computed risk-levels. We use cross validation to evaluate the impact of the parameter η_k (forgetting factor) on the estimated risk-levels for apps' resources access. The η_k rate for the first three experiments is set using the *time-variant* polynomial form.

In the first experiment we compute the average risk-levels for all malicious and normal apps in our datasets. Fig. 11(a) shows the results of the risk computation for two app groups, malicious and normal apps. We can see in the figure that the computed risk-levels for normal apps start from near 0 to 0.8, and risk-levels for malicious apps start from 0.4 to 1. The quantity of the computed risk-levels are presented in the Table IV. We can see that 450 (90%) of the malicious apps have risk-levels higher than 0.7, whereas only 30 (6%) normal apps risks are above this level. The results of this experiment show that risk-level is an effective criteria to separate malicious apps from normal apps.

TABLE IV
RISK-LEVEL DISTRIBUTION

Type	$0-0.1$	$0.1-0.3$	$0.3-0.5$	$0.5-0.7$	$0.7-0.1$
Normal	25	4	97	344	30
Malicious	0	0	2	48	450

In the second experiment, we study the impact of the user responses on the average risk-level of an app. We define a scenario where we target at the risk-level of the network resource of a malicious app. Without any user input, the risk-level of the resource is 0.9. We plot the change curves of the estimated risk-levels of the same resource before and after users chose to "allow" or "block" the resource access. At the beginning there is no log input for the resource so that risk-level is 0. Then we start to feed log files of the malicious app and the risk-level of the app increases drastically to 0.9. At time 0.3 we inject two types of responses to the system and observe its impact to the risk value of the app. Fig. 11(b) illustrates the impact of the user's response to the estimated risk-levels. We can see that if the user's response is "Allow", the model turns to be less conservative and the risk-level of the resource decreases. On the other hand, if user's response is "Block", the risk-level increases (the user is more conservative). When no user response is in place, the risk-level of the app remains at around the same level. From this experiment we can see that the risk assessment model can adapt to the user's responses and provide customized risk-levels.

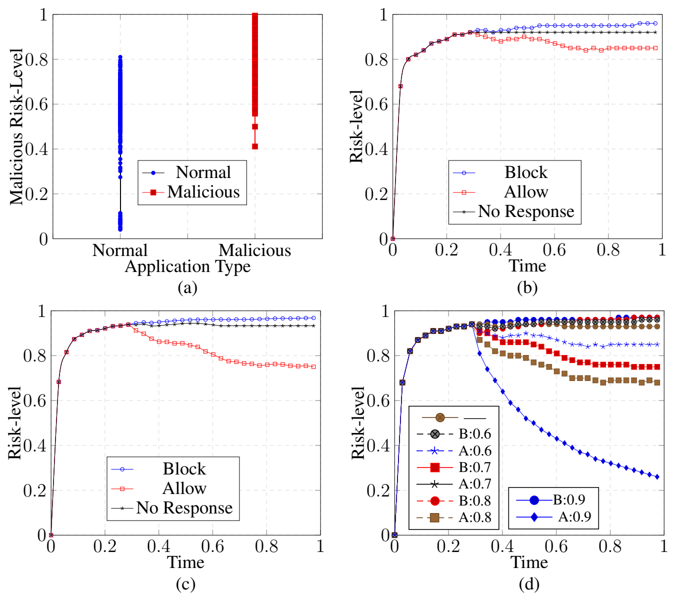


Fig. 11. Applications' computed risk-levels

In the third experiment, we still use the same scenario as the last experiment, but this time we focus on the estimated risk-levels for the network resource only. Fig. 11(c) shows the computed risk-level for both "Allow" and "Block" responses. We can see that this result also shows the influence from the user response. However, the risk-level drops faster in the case that "Allow" is chosen by the user. The impact from user response is higher since it is focused on one resource only.

The last experiment is to study the impact of the learning rate parameter η_k (forgetting factor) on the computed risk-levels by the system. In this experiment we ran the experiment multiple times with different settings for η_k. We let η_k change from 0.6 to 0.9 and observe its impact on the risk values. Fig. 11(d) shows that the higher η_k is, the higher impact user's responses bring to the risk-levels. This is because higher η_k means more emphases on the recent input, which is the user's responses.

V. Related Work

There have been many studies towards the principles and practices to manage resource usage and Android security [16], [18], [6], [11], [12], [22], [9] and privacy protection [2]. Stochastic models have been a powerful method to model security issues or defend against attacks. These models solutions have been widely discussed in literature. As a stochastic process model Markov chains have been extensively used to model security attacks. In the last two years, a few works [5], [24] have been proposed that use the concept of Hidden Markov Models to address the mobile security issues such as users' privacy preserving, malicious apps detection, and targeted malware.

Chen et al. [5] proposed a hidden Markov model to detect Android malicious apps at runtime. This work is the closest work to our proposed model. Their work is based on application intents passing through the Android binder only. The proposed HMM model's emissions (observations) and training process are not described. However, relying on the apps' intents as observations is not sufficient to decide whether the app is malicious or not. For example, they do not take into account the malicious API calls (Ads libraries), time, sensitive permission requests etc. This can be the reason why their detection accuracy is 67%. In contrast, we take all these features into account and achieve much better accuracy.

Suarez-Tangil et al. [24] proposed a stochastic model to address targeted malware. The context where a malicious behavior takes place plays a key role in the proposed work. In order to capture how the users interact with an app or a set of apps, they rely on a discrete-time first-order Markov process.

The major difference between our proposed model and the existing ones is that they do not use a comprehensive app behavior analysis in their model training. The definition and discovery of proper observations, such as apps' intents, API calls, and time-stamp, make the model a unique solution in terms of Android malware detection. Therefore, we design a model that involves the features described above to enhance the accuracy. In addition, our proposed model updates the model's parameters dynamically based on users' preferences througha self-train strategy. To the best of our knowledge, this the first model to help users through risk alerts generated by a well-trained HMM model and gets updated in a real-time manner.

VI. Discussion

While the major challenges for proposed model have been discussed, many other ones still remain. In this section, we discuss some potential issues in the system and our future plan to build and complete such a system.

Users' Privacy: The proposed model is a crowdsourcing-based solution which involves collecting apps' logs from users' devices. The system also collects users' responses regarding whether to allow or block the resource access. Data collection may raise privacy concerns. To protect the privacy of users, we design a privacy-aware data collection mechanism that uses *hashing and salting* method to protect the true identity of the users.

User Participation: Similar to any other participatory services, our system should have an incentive model to motivate users to participate. Benefit users getting from the risk alerts might not be enough to justify the inconvenient of reporting their decisions to a central server all the time. Incentivizing users however is out of scope of this paper.

False Responses: One of the main important threats to the system is the injection of false responses to mislead the risk assessment system. We have investigated this potential threat and developed a multi-agent game theory model to study the gain and loss of malicious user and the defense system. We derived a system configuration to discourage rational attackers to launch such attacks. Furthermore, we also consider enhancing the system in which risk-levels are adapted according to each user's different security and privacy needs.

VII. Conclusion

In this paper, we propose XDroid, an Android app resource access risk estimation framework using hidden Markov model. We first define and select features to represent behaviors of Android apps and collect them through our own developed human-oriented instrumentation tool DroidCat. A filtering and parsing method is then employed to synthesis and organize the captured behaviors. We train the model with a proper malicious app dataset using the Baum-Welch algorithm and test it with different test datasets using the Viterbi algorithm. Through our model, we can compute the risk-level of those apps that behave malicious with high level of accuracy. The model informs users the risk-level of their apps in real-time. Our model is able to update the model's parameters dynamically using an on-line algorithm and users' preferences. Our experimental results demonstrate that our proposed model achieve a satisfying accuracy in terms of true positive and false positive rate. Our evaluation results also show that our model can effectively provide customized risk estimations depending on users' preferences. As our future work, we plan to apply more external features to further improve the detection accuracy. We also plan to include the users' expertise level into the risk-level computation process for better accuracy.

References

[1] What is the price of free. http://www.cam.ac.uk/research/news/what-is-the-price-of-free.

[2] Y. Agarwal and M. Hall. Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 97–110, New York, NY, USA, 2013. ACM.

[3] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[4] T. Ball. The concept of dynamic analysis. *SIGSOFT Softw. Eng. Notes*, 24(6):216–234, Oct. 1999.

[5] Y. Chen, M. Ghorbanzadeh, K. Ma, C. Clancy, and R. McGwier. A hidden markov model detection of malicious android applications at runtime. In *Wireless and Optical Communication Conference (WOCC), 2014 23rd*, pages 1–6, May 2014.

[6] J. Crussell, R. Stevens, and H. Chen. MAdFraud: Investigating ad fraud in android applications. In *12th CMSAS*, MobiSys '14, pages 123–134, New York, NY, USA, 2014. ACM.

[7] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan. Android security: A survey of issues, malware penetration, and defenses. *IEEE Communications Surveys Tutorials*, 17(2):998–1022, Secondquarter 2015.

[8] A. Gosain and G. Sharma. A survey of dynamic program analysis techniques and tools. In S. C. Satapathy, B. N. Biswal, S. K. Udgata, and J. Mandal, editors, *Proc. of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA'14), Odisha, India*, volume 327, pages 113–122. Springer International Publishing, November 2015.

[9] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *Proc. of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12), Low Wood Bay, Lake District, UK*, pages 281–294. ACM, June 2012.

[10] S. Gunasekera. *Android Apps Security*. Apress, Berkely, CA, USA, 1st edition, 2012.

[11] Q. Ismail, T. Ahmed, A. Kapadia, and M. K. Reiter. Crowdsourced exploration of security configurations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 467–476, New York, NY, USA, 2015. ACM.

[12] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu. Riskmon: Continuous and automated risk assessment of mobile applications. In *Proc. of the 4th ACM Conference on Data and Application Security and Privacy (CODASPY '14), San Antonio, Texas, USA*, pages 99–110. ACM, March 2014.

[13] W. Khreich, E. Granger, A. Miri, and R. Sabourin. A comparison of techniques for on-line incremental learning of hmm parameters in anomaly detection. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–8, July 2009.

[14] W. Khreich, E. Granger, A. Miri, and R. Sabourin. A survey of techniques for incremental learning of {HMM} parameters. *Information Sciences*, 197:105 – 130, 2012.

[15] I. Lunden. 6.1b smartphone users globally by 2020, overtaking basic fixed phone subscriptions. http://techcrunch.com/2015/06/02/6-1b-smartphone-users-globally-by-2020-overtaking-basic-fixed-phone-subscriptions.

[16] R. Mittal, A. Kansal, and R. Chandra. Empowering developers to estimate app energy consumption. In *18th CMCN*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

[17] J. Mizuno, T. Watanabe, K. Ueki, K. Amano, E. Takimoto, and A. Maruoka. *Discovery Science: Third International Conference, DS 2000 Kyoto, Japan, December 4–6, 2000 Proceedings*, chapter On-line Estimation of Hidden Markov Model Parameters, pages 155–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[18] O. R. E. Pereira and J. J. P. C. Rodrigues. Survey and analysis of current mobile learning applications and technologies. *ACM Comput. Surv.*, 46(2):27:1–27:35, Dec. 2013.

[19] L. Rabiner. First hand: The hidden markov model. *IEEE Global History Network*, pages 4–15, October 2013.

[20] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.

[21] W. Rothman. Smart phone malware: The six worst offenders. http://www.nbcnews.com/tech/mobile/smart-phone-malware-six-worst-offenders-f125248.

[22] G. Russello, A. B. Jimenez, H. Naderi, and W. van der Mark. Firedroid: Hardening security in almost-stock android. In *Proc. of the 29th Annual Computer Security Applications Conference (ACSAC '13), New Orleans, Louisiana, USA*, pages 319–328. ACM, December 2013.

[23] Statista. Number of available applications in the google play store from december 2009 to november 2015. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[24] G. Suarez-Tangil, M. Conti, J. E. Tapiador, and P. Peris-Lopez. *Computer Security - ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, chapter Detecting Targeted Smartphone Malware with Behavior-Triggering Stochastic Models, pages 183–201. Springer International Publishing, Cham, 2014.

[25] A. J. Viterbi. A personal history of the viterbi algorithm. *IEEE Signal Processing Magazine*, 23(4):120–142, July 2006.