



Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Anomaly detection for smartphone data streams



Yisroel Mirsky*, Asaf Shabtai, Bracha Shapira, Yuval Elovici, Lior Rokach

Ben-Gurion University of the Negev, Beer Sheva, Department of Information Systems Engineering, Israel

ARTICLE INFO

Article history:

Received 2 December 2015

Received in revised form 9 June 2016

Accepted 21 July 2016

Available online 10 August 2016

Keywords:

Smartphone security

Data streams

Anomaly detection

Contexts

Continuous authentication

ABSTRACT

Smartphones centralize a great deal of users' private information and are thus a primary target for cyber-attack. The main goal of the attacker is to try to access and exfiltrate the private information stored in the smartphone without detection. In situations where explicit information is lacking, these attackers can still be detected in an automated way by analyzing data streams (continuously sampled information such as an application's CPU consumption, accelerometer readings, etc.). When clustered, anomaly detection techniques may be applied to the data stream in order to detect attacks in progress. In this paper we utilize an algorithm called pcStream that is well suited for detecting clusters in real world data streams and propose extensions to the pcStream algorithm designed to detect point, contextual, and collective anomalies. We provide a comprehensive evaluation that addresses mobile security issues on a unique dataset collected from 30 volunteers over eight months. Our evaluations show that the pcStream extensions can be used to effectively detect data leakage (point anomalies) and malicious activities (contextual anomalies) associated with malicious applications. Moreover, the algorithm can be used to detect when a device is being used by an unauthorized user (collective anomaly) within approximately 30 s with 1 false positive every two days.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

In 2016, over two billion people will have a smartphone as a part of their daily lives [1]. Smartphones provide a means of communication, as well as a central location to store and organize information, a quality which makes the popular devices enticing targets for attackers interested in stealing private information [2]. One way to protect a smartphone is to perform anomaly detection [3]. In this approach, the normal behavior of an internal or external actor is modeled so that malicious activities can be detected as anomalies, behaviors which do not fit the norm. Subsequently the detected malicious activity can be blocked, thereby protecting the user. For example, take the malicious behavior of sending SMSs to premium numbers (monetary theft). In this case, explicit information such as the message's textual content or the destination number could be used to directly determine whether the SMS is anomalous. However, in some cases explicit information is unavailable or insufficient, making it a challenge to detect the SMS as malicious. For instance, if the SMS contains legitimate text (stolen from the user's outbox), or if there is no complete list of premium numbers to blacklist.

In cases where explicit information is lacking, many times contextual information, often in the form of a data streams, is available. Contextual information is the additional information that assists in clarifying a particular event or behavior [4]. In this paper, we refer to data streams as unbounded sequences of measurements sampled continuously from a particular source. Modern smartphones are equipped with a wide range of sources which can be sampled in order to generate a

* Corresponding author.

E-mail addresses: yisroel@post.bgu.ac.il (Y. Mirsky), shabtaia@bgu.ac.il (A. Shabtai), bshapira@bgu.ac.il (B. Shapira), elovici@bgu.ac.il (Y. Elovici), liorrk@post.bgu.ac.il (L. Rokach).

<http://dx.doi.org/10.1016/j.pmcj.2016.07.006>

1574-1192/© 2016 Elsevier B.V. All rights reserved.

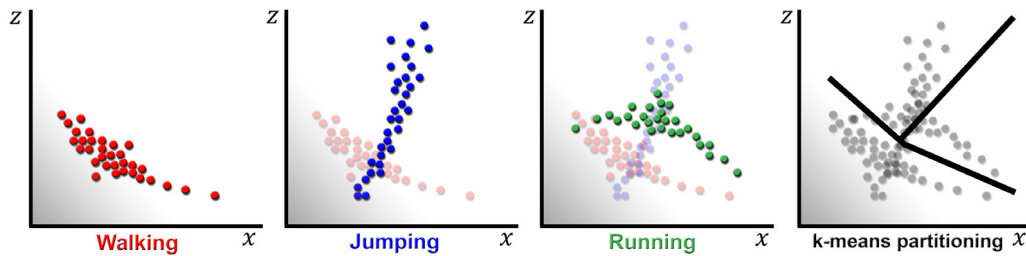


Fig. 1. An illustration of the overlap of different distributions found in data, received along two of the axes from a smartphone's accelerometer sensor. Here the ground truth is activity recognition.

data stream rich in contextual information. For instance, Android smartphones allow all applications to receive information about the phone's other applications (including statistics related to the CPU, memory usage, and system priorities) through the Linux virtual/proc/folder [5]. Furthermore, information on device motion and device status is available as well. When sampled, these data streams can be used to capture the actor's contexts in order to detect possible anomalies. Returning to our example, let us assume that the device's motion sensors are sampled when SMSs are sent. In this scenario, by utilizing the contextual information from the sensor's data stream, it becomes easier to differentiate between a physical human (external actor) sending an SMS and some automated malicious code (internal actor) which is sending the SMS.

Here are additional examples of situations in which explicit information for the detection of anomalies is lacking but contextual data streams are available:

- **Outbound encrypted transmissions:** typically a malware (such as a bot) sends data back to its command and control server (C&C) over an encrypted channel. Therefore, semantic analysis or other explicit information about the data in motion is not obtainable. However, the context of the encrypted transmission, along with other details about transmissions captures information useful in determining a transmission's legitimacy [6].
- **Activities of applications in the Android OS platform:** Applications running in Android are sandboxed in separate Dalvik virtual machines (DVM) [7]. At a basic level, this prevents applications from accessing other applications' data and resources without explicit user permissions [8]. In order to gain full access, a device must be rooted, giving all applications access to privileged commands within Android's subsystems, a security risk in itself. Since devices are shipped unrooted by default, antivirus applications available through the Android marketplace are highly limited in the dynamic analysis (online scanning) they can perform in order to detect anomalies. However, without root privileges applications can obtain contextual information by sampling other applications' statistics (e.g., CPU utilization, memory usage, etc.)
- **High level inference from low level trust-zones:** Understanding what applications are doing from a low level trust zone (e.g., hypervisor) is difficult, because kernel (as well as DVM) information is not directly accessible. For instance, it is not clear what application is sending data or which process is currently in the foreground legitimately using the CPU. However, the motion data and screen on/off data are available from the hypervisor. Using this data, it may be possible to detect illegitimate transmissions as they occur.

In order to utilize the contextual information in a data stream, one needs to *mine* the stream for the hidden contexts. The hidden contexts found in a real world data stream can exhibit behaviors in the form of correlated distributions (clusters) [9]. These clusters of observations are referred to in literature as the *concepts* or *contexts* captured by the data [9–12]. By modeling these contexts, it is possible to detect anomalies in an unsupervised manner [13,14]. However, the detection of anomalies in data streams is a challenging process. This is because data streams are unbounded in length and involve recurring concepts as well as concept drifts [15]. These properties make it difficult to distinguish between a previously seen concept which is now changing and a new concept (an anomaly) which has not been seen before. Current stream clustering algorithms can detect and track concept drifts [16,17]. However, (1) they were not specifically designed to detect various types of anomalies found in data streams, and (2) they are not able to distinguish between clusters which overlap in geometric space. The reason they cannot differentiate between overlapping clusters is because these algorithms seek to form geometric partitions of the feature space, and therefore do not respect the ground truth. To illustrate this issue, Fig. 1 plots the temporal concepts found while performing activity recognition using a smartphone's accelerometer. Here any partition of the feature space into three clusters will not respect the ground truth (that the clusters formed from each activity overlap in geometric space).

In this paper we propose a solution for the detection of various types of anomalies in data streams which have overlapping clusters. In a previous work we proposed pcStream: a stream clustering algorithm used to dynamically detect and manage temporal contexts [18]. The name "pcStream" is based on the **p**roduct of the **p**roducts of the distributions in the data stream which are used to dynamically detect and compare the underlying contexts. One of the advantages of pcStream is that it can detect and model overlapping concepts. Detection and management is accomplished by taking into account the temporal relation of the stream's observations (i.e., temporal contexts). This is the main reason for pcStream's ability to outperform state-of-the-art stream clustering algorithms in detecting contexts in real world data streams (the reader is invited to view our original paper for the analysis [18]). Moreover, pcStream tracks concept drifts, keeping the captured contexts relevant and up to date.

In this work, we propose three extensions to the pcStream algorithm. Each extension designed to detect a different type of anomaly found in data streams generated by smartphones. The general approach is to train pcStream on a data stream which captures the normal contexts (including those which overlap). We can then use the trained model to detect point, contextual, or collective anomalies using the respective extension. The concept for deploying these extensions is to have a security agent running on the device. By sampling the relevant numeric features (e.g., an application's CPU usage), the agent observes the actor for unexpected behaviors which it can either block or inform the user about—thereby providing implicit smartphone security.

Therefore the contributions of this paper are as follows:

- (1) Introduction of a single algorithm, *pcStream*, for detecting **point**, **contextual**, and **collective** anomalies found in temporal data streams, in particular, data streams whose concepts overlap each other in geometric space (i.e., feature space). We make the source code for this algorithm available online, with versions written in R, Matlab, Python, and PySpark (for Hadoop clusters).¹
- (2) Evaluation of pcStream in the realm of smartphone security. We explore examples of how pcStream can be used as a security solution for addressing current smartphone security threats, specifically, the **detection of data leakage**, the **detection of active malware** in dynamic analysis, and the provision of **continuous user authentication**, all achieved by analyzing data streams.

To evaluate pcStream as an anomaly detection tool, we collected a dataset consisting of eight months of sensor data from 31 volunteers all of whom used Samsung Galaxy S5 smartphones. In order to detect short-term anomalies (such as a malicious transmission or a device theft), we sampled the devices' sensors at a high temporal resolution. Other existing datasets do not have this level of resolution for a long period of time. This is most likely because many sensors have significant power consumption. To overcome this challenge, we provided the volunteers with battery cases that nearly triple the battery life of the device (ensuring a total battery life of 9–10 h on a full charge).

The remainder of this paper is structured as follows. Section 2, reviews the pcStream algorithm. Section 3 presents the different types of anomalies addressed in this paper, and proposes the anomaly detection extensions to the pcStream algorithm. Section 4 presents the dataset used, evaluation setup, and evaluation results. Section 6 reviews related works. Section 5 provides a discussion on related issues. Finally, Section 7 provides a conclusion, and proposes future work.

2. Review of the pcStream algorithm

In this section, we briefly review the basic pcStream algorithm as presented in [18].

2.1. Notations and definitions

Definition 1. Let a *context space* be defined as the geometrical space \mathbb{R}^n , where n is the number of attributes which define the stream. For instance, one dimension may be the y -axis readings of a smartphone's accelerometer, while another may be the beats per second (bps) of the smartphone's user. This definition is similar to Context Space Theory (CST), formally proposed in [19].

Definition 2. Let a *stream* S be defined as an unbounded sequence of data objects having the form of points in \mathbb{R}^n , and let $x_i \equiv [x_{1,i}, x_{2,i}, \dots, x_{n,i}]$ be the i th point in the sequence. S can also be viewed as a matrix having n columns and an unbounded number of rows, where row i represents the values sampled at time tick i . We use the notation t to denote the current time tick and the notation x_t to refer to the most recent point received from S . Let $f_{a,S}$ be the arrival rate of the row vectors in S measured in Hz.

Definition 3. Let c be a *context* (i.e., concept) defined as a cluster of sequential points having a correlated distribution in \mathbb{R}^n , in which S exists within for at least t_{\min} time ticks at a time. The distribution of c is generally stationary, but it may change gradually over time as it is subjected to concept drift. For instance, with the accelerometer data of a user's smartphone, the *context* which captures the action of jumping may change as the user gets older or sicker. We use the notation c_t to refer to the current context of S .

Definition 4. We define a *contextual stream* to be a *stream* that captures the temporal contexts of a real world entity. More formally, S is a *contextual stream* if S travels among a finite number of distinct contexts, staying at each for at least t_{\min} time ticks per visit. The property of revisiting certain distributions is known as a reoccurring drift or reoccurring concepts [15].

Let \mathbf{C} be the finite collection of known contexts found in S , such that $c_i \in \mathbf{C}$ is the i th discovered *context*. Let $|\mathbf{C}|$ denote the number of known contexts.

It is important to note that \mathbf{C} does not necessarily form a distinct partition of \mathbb{R}^n . As mentioned earlier, the distributions of *contexts* may overlap each other. Therefore it is possible that two identical points x_a and x_b belong to two distinctly different contexts c_i and c_j .

¹ <https://github.com/ymirsky/pcStream>.

Definition 5. We define a *context category* as all contexts from a *contextual stream* that have the same t_{\min} , rate of concept drift and distinction between their distributions.

2.2. The context model

Since we define *contexts* as correlated distributions in \mathbb{R}^n , we model the *contexts* using principal component analysis (PCA) [20]. PCA captures the relationship of the correlation between the dimensions of a collection of observations stored in the $m \times n$ matrix X , where m is the number of observations. The result of performing PCA on X is two $n \times n$ matrices; the diagonal matrix V (the Eigenvalues) and the orthonormal matrix P (the Eigenvectors, a.k.a. *principal components*). The Eigenvectors p_1, p_2, \dots, p_n form a basis in \mathbb{R}^n centered on X and oriented according to the correlation of X . The Eigenvalues $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$ are the variances of the data in the direction of their respective Eigenvectors. The Eigenvalues of V are sorted from highest to lowest variance and the respective Eigenvectors in P are ordered accordingly. In other words, from the mean of the collection X , p_1 is the direction of highest variance in the data (with σ_1^2). We define the contribution of component p_i as the percent of total variance it describes for the collection X , such that $cont_X(p_i) = \frac{\sigma_i^2}{\sum_{j=1}^n \sigma_j^2}$.

PCA has been widely used to reduce the dimensionality of a dataset while preserving the information it holds [20]. This is accomplished by projecting observations on to the top *principal components* (PCs). Typically, most of a collection's variance is captured by just a few PCs. By retaining only these PCs, we effectively summarize the distribution and focus our future calculations on the dimensions of interest. Let ρ be the target percent of variance to be retained. We define $k \in \mathbb{N}$ to be the fewest, most influential PCs in which their cumulative sum of contributions surpasses ρ . Stated otherwise as $\text{argmin}_k \{ \sum_{i=1}^k cont_X(p_i) \geq \rho \}$. We denote the k associated with context c_i as k_{c_i} .

We model the i th discovered *context* as the tuple $c_i \equiv \langle M_i, \mu_i, A_i \rangle$, where M_i is a $m \times n$ matrix consisting of the last m observations assigned to c_i , μ_i is the mean of the observations in M_i , and $A_i = [p_1\sigma_1, p_2\sigma_2, \dots, p_{k_{c_i}}\sigma_{k_{c_i}}]$ is a $n \times k_{c_i}$ transformation matrix.

A_i is essentially a truncated version of P with its Eigenvectors scaled to their standard deviations (SD). A_i can be calculated by first performing PCA on M_i , to get P_i and V_i , and then calculating $A_i = Q_i \Lambda_i$ where Λ_i is a diagonal matrix of the top k_{c_i} largest SDs (obtained from V_i), and Q_i is the column-wise truncation of P_i so that it only includes the first k_{c_i} columns. The significance of the transformation matrix A_i will be detailed later in the paper.

Matrix M_i acts as a windowed memory for c_i by discarding the m th oldest observation when a new one is added. Windowing over a stream is an implicit method for dealing with concept drift [17,21].

2.3. Similarity scores

When a new observation x_t arrives, we must determine to what degree it belongs to each known context $c_i \in \mathbf{C}$. As mentioned earlier, the clusters that form contexts overlap in the feature space, and the point x_t can belong to multiple contexts at once. Therefore, we must compute the similarity score of the point in question with respect to each known context.

The point's statistical similarity to a distribution is calculated as the Mahalanobis distance using only the top k PCs of that distribution. Equivocally, they produce this score by first zero-meaning the point to that distribution, then transforming it onto the distribution's top k PCs, and finally by computing the resulting point's magnitude. More formally, the similarity of point x to the distribution of c_i is $d_{c_i}(x) = \|(x - \mu_i)A_i\|$.

Intuitively, the contours of performing Mahalanobis distance can be seen as ellipsoid shapes extending from the distribution according to the variance in each direction. From the perspective of the top k_{c_i} and k_{c_j} normalized PCs of each respective context, their similarity scores differ (even though the Euclidean distances between x and both μ_i and μ_j are equivalent). Furthermore, it is possible that the $k_{c_i} > k_{c_j}$ depends on ρ and the correlation of the distribution.

Let $\phi \in [0, \infty)$ be defined as the similarity threshold. We say that a point x is not similar to *context* c_i if $d_{c_i}(x) > \phi$. Let $d_{\mathbf{C}}(x) = [d_{c_1}(x), d_{c_2}(x), \dots, d_{c_{|\mathbf{C}|}}(x)]$ be defined as the score vector (membership degrees) of x to each of the models in \mathbf{C} . We say that x does not fit any known context if all elements in $d_{\mathbf{C}}(x)$ are greater than ϕ . Moreover, we say that x is most similar to *context* c_i if the smallest element in $d_{\mathbf{C}}(x)$ is $d_{c_i}(x)$.

2.4. Detection of new contexts

A critical function of the pcStream algorithm is to detect when a previously unseen context has appeared. From Definition 3, contexts are assumed to have rather stationary distributions. Therefore, a new context is detected when the data distribution of S no longer fits the contexts in \mathbf{C} for a consistent t_{\min} time ticks. At this point, we say that these t_{\min} observations constitute a new context which are then modeled for future use.

To track this behavior, we introduce a new concept called a "drift buffer". Let the drift buffer be called D and have a length of t_{\min} . Should D ever be filled continuously without any intermittent assignments to other contexts (i.e., should $D = \{x_{t-t_{\min}-1}, \dots, x_t\}$), then we have detected a new context (see Definition 3). Therefore, we create a new context model with the content of D (emptying D), and set this context as current context c_t . However, in the case of a partial drift (i.e., D

did not get filled, yet x_t fits some context in \mathbf{C} , we assume that S experiences a wider boundary of c_t ; therefore we empty D into c_t .

2.5. The pcStream algorithm

The basic approach of the core pcStream algorithm is to follow the *stream*'s data distribution. Membership scores are available anytime by calculating the statistical similarities between a point and each known context. As long as the arriving points stay within the distribution of a known context, we assign them to that context. The moment the *stream*'s distribution does not fit any known context, we define a new one. Each of the concepts has a window of memory to allow for concept drifts. Should the allocated memory space be filled, then one of two methods for merging context models is performed. Point anomalies are detected as short-term drifts away from all known contexts. Finally, different context categories are detected by adjusting the algorithm's parameters accordingly. The parameters for pcStream are: the sensitivity threshold ϕ , the context drift size t_{\min} , the model memory size m , and the percent of variance to retain in projections ρ . The pseudo-code for pcStream can be found in Algorithm 1.

In lines 1–3, pcStream is initialized by creating the initial collection \mathbf{C} with context c_1 , and then by setting the current context (c_t) accordingly. The function *init*(S, t_{\min}, m, ρ) runs the function *CreateModel*(X, m, ρ) on the first t_{\min} points of S . The function *CreateModel*(X, m, ρ) returns a new *context model* c by using the collection of observations X and target total variance retention percentage ρ . Remember that the memory of a *context model* M is a window (FIFO buffer) with a maximum length of m (forgetting the oldest observations). Optionally, an initial set of models for \mathbf{C} can be made from a set of observations pre-classified as known contexts of S (e.g. a collection of points that captures running and another a collection of points that captures walking). From this point on, pcStream enters its running state (lines 4–5).

In lines 4.1–4.3, point x_c arrives and x_c 's similarity score is calculated for all known contexts in \mathbf{C} . Stored in i is the index to the model in \mathbf{C} to whom x_c is most similar. Reminder, the index of \mathbf{C} is chronological by order of discovery.

In line 4.4, we determine whether x_c fits any of the contexts in \mathbf{C} . If it does, then we proceed to lines 4.4.1–4.4.3 where we update the model of best fit (c_i) with instance x_c , and update c_t accordingly. At this point, if there are any instances in D , then they are emptied into c_t as well (line 4.4.1). This is because an interruption while continuously filling D with outliers means that the outliers seen until now are not part of a new (unseen) context, but rather a new boundary for the current context which the stream is experiencing. The function *UpdateModel*(c_i, X) re-computes the tuple c_i from \mathbf{C} after adding the observation(s) X to the FIFO memory M_i .

If the check on line 4.4 indicates that x_t does not fit any context in \mathbf{C} , then we add x_t to the drift buffer D , and subsequently check if D is full. If D has reached capacity (t_{\min}) then an unseen *context* has been discovered. In this case, D is then emptied and formed into a new *context model* (c), which is added to \mathbf{C} and set as c_t . The function *AddModel*(c, \mathbf{C}) adds c to \mathbf{C} as $c_{|\mathbf{C}|+1}$. If the additional model is too large for the memory space allocated to pcStream, the function *merge*(c_i, c_j) is used to free one space for c (in \mathbf{C}) by merging the average oldest context model c_i with its nearest context model c_j based on the Euclidean distance between centroids. There are two methods for performing this merge: context freshness (modeling a context from the m most recent observations between M_i and M_j) and context preservation (modeling a context from the interleave between the top $m/2$ observations of M_i and M_j) [18].

<p>Online Algorithm 1: pcStream $\{S\}$ Input Parameters $\{\phi, t_{\min}, m, \rho\}$ Anytime Outputs: $\{c_t, d_c(\tilde{x}_t)\}$</p> <ol style="list-style-type: none"> 1. $\mathbf{C} \leftarrow \text{init}(S, t_{\min}, m, \rho)$ 2. $c_t \leftarrow c_1$ 3. $D \leftarrow \emptyset$ 4. <i>loop</i> <ol style="list-style-type: none"> 4.1. $\tilde{x}_c \leftarrow \text{next}(S)$ 4.2. $\text{scores} \leftarrow d_c(\tilde{x}_c)$ 4.3. $i \leftarrow \text{IndxMin}(\text{scores})$ 	<ol style="list-style-type: none"> 4.4. <i>if</i> $\text{scores}(i) < \phi$ <ol style="list-style-type: none"> 4.4.1. <i>UpdateModel</i>($c_i, \text{Dump}(D)$) 4.4.2. <i>UpdateModel</i>(c_i, \tilde{x}_c) 4.4.3. $c_t \leftarrow c_i$ 4.5. <i>else</i> <ol style="list-style-type: none"> 4.5.1. <i>Insert</i>(\tilde{x}_c, D) 4.5.2. <i>if</i> $\text{length}(D) == t_{\min}$ <ol style="list-style-type: none"> 4.5.2.1. $c \leftarrow \text{CreateModel}(\text{Dump}(D), m, \rho)$ 4.5.2.2. <i>AddModel</i>(c, \mathbf{C}) 4.5.2.3. $c_t \leftarrow c$ 5. <i>end loop</i>
---	--

2.6. The detection of different context categories

Each selection of the parameters ϕ , t_{\min} , and m changes the type of contexts pcStream focuses on in S . In other words, these parameters cause pcStream to seek out all contexts belonging to a single *context category*, where ϕ is the degree of distinction between contexts, and m is the rate of concept drift (see Definition 5).

Consequently, a *small* ϕ will cause pcStream to detect indistinct contexts (i.e., small nuances), while a *large* ϕ will cause pcStream to detect contexts which are more unique. Similarly, a *small* t_{\min} will cause pcStream to detect short term-contexts as opposed to more long-term ones, and a *small* m , will cause pcStream to focus on sudden concept drifts as opposed to more gradual drifts.

Intuitively, multiple *context categories* are in a data stream at any given time. For example, at a given moment, a smartphone's accelerometer can capture the context and determine whether a user is "awake or asleep", "running or walking", and "running to catch the bus, or for sports". Therefore, if one is interested in different context categories at the same time, multiple instances of pcStream should be run in parallel with the respective settings.

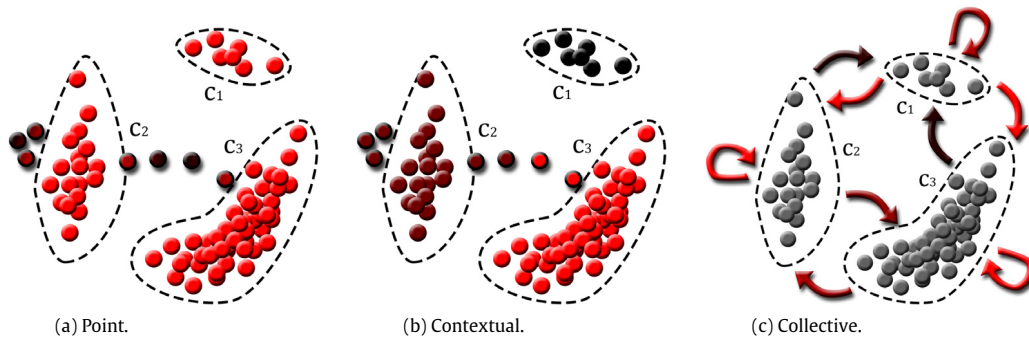


Fig. 2. An illustration of how pcStream views the different types of anomalies. Darker shades of red indicate a higher degree of abnormality. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

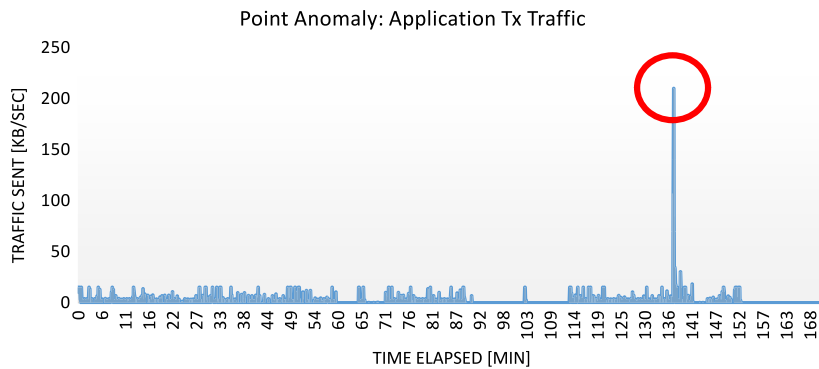


Fig. 3. An illustration of a point anomaly found in a data stream.

3. pcStream anomaly detection extensions

In general, there are three types of anomalies: point anomalies, contextual anomalies, and collective anomalies [13]. In this section, we present three extensions to pcStream which enable it to detect the three anomaly types. In Section 4, we evaluate these added capabilities as implicit smartphone security solutions. Fig. 2 is an illustration referenced throughout this section that visualizes how pcStream views each type of anomaly.

3.1. Point anomaly detection

Point anomalies (otherwise known as outliers) are individual observations which are considered anomalous with respect to the data (i.e., the target behavior). As an example, let us assume we are attempting to detect unexpected outbound transmissions from an application, such as the game, Angry Birds, on Android. After observing the application's outbound traffic rates over a long period of time, we see that the application never has a transmission at a rate of 15 kB per second. Therefore any observation that is significantly above this norm, is considered anomalous.

Fig. 3 illustrates this example. Note that although in this example $x \in \mathbb{R}$, point anomalies also apply to data streams where $x \in \mathbb{R}^n$.

In pcStream, point anomalies are observations which do not fit any known contexts and are not considered part of a new context. In other words, a point anomaly is viewed as those observations which are outliers with respect to the known distributions. This idea reflects the Soft Independent Modelling by Class Analogy method (SIMCA) [22] on which pcStream's clustering algorithm is based. In SIMCA, an outlier is an observation whose Mahalanobis distance is too far from all known classes.

When using a pcStream collection \mathbf{C} to detect anomalies (i.e., we are not updating any of the models) then every single observation that is above the sensitivity threshold ϕ is a point anomaly. However, if we are actively updating \mathbf{C} , then all points which are above the ϕ are not immediately designated as point anomalies. The reason is because we consider a consistent sequence of t_{\min} observations above the threshold as an unseen context that should be modeled. However, the moment we know that these observations do not belong to a new context (i.e., their consistency was broken—line 4.4.1 of Algorithm 1), then we know that they all must be point anomalies. Fig. 2(a) illustrates partial drifts where darker shades represent larger $d_{\mathbf{C}}(x)$ values. The observations outside the dotted lines are the observations we are testing with respect to contexts c_{1-3} .

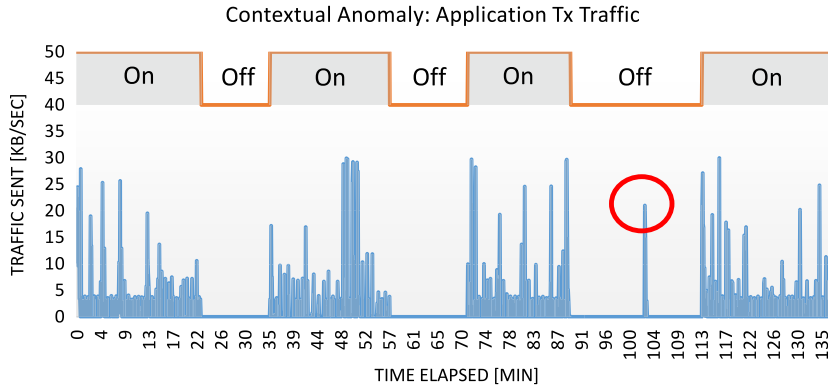


Fig. 4. An illustration of a contextual anomaly found in a data stream.

Naturally, a partial drift can occur when the stream is experiencing a concept drift (a shift in the distribution). Therefore, during the initial training of the algorithm, observations found in partial drifts should not be discarded—but rather included as in the original algorithm.

Using these concepts and a trained pcStream model collection \mathbf{C} , we can measure the degree to which observation x is a point anomaly using the function

$$Anom1(x) = \min(d_{\mathbf{C}}(x)) \quad (1)$$

where $d_{\mathbf{C}}(x)$ is the vector of the Mahalanobis distances from each of the $|\mathbf{C}|$ models in \mathbf{C} with respect to x . In other words, $Anom1$ returns a score indicating the degree x belongs to the collection of all known contexts. Note that $Anom1$ returns increasingly larger values the more x is considered abnormal with respect to the normal behavior captured by \mathbf{C} .

3.2. Contextual anomaly detection

Contextual anomalies are individual observations which are considered anomalous under certain contexts but not under others [23]. Typically, observations analyzed within this category have two types of attributes: contextual and behavioral attributes. Contextual attributes provide added information that help put the behavioral attributes into perspective. Behavioral attributes are the non-contextual characteristics of the instance under analysis. As an example, we shall re-examine the task of detecting unexpected outbound transmissions from the Angry Birds application. Here the outbound data rate is the behavioral attribute. Assume the application has a transmission which is normal with respect to previous transmissions, however, if it occurred while the screen was off, it would be considered anomalous. This is because applications running on Android enter a “Pause” mode once the screen turns off as part of the Android application’s life cycle [24]. Since the Angry Birds application in this example does not run as a service, the screen status feature acts as a contextual attribute for detecting anomalies in the behavior attribute.

Fig. 4 illustrates this example where the data stream consists of two features: the out-bound data rate and the screen status.

In pcStream, we view contextual anomalies as observations which are assigned to rare context models (i.e., models in \mathbf{C} that have had relatively few visits). The intuition behind this is that a pcStream model captures a temporal correlation among the features in the data stream. Therefore, a model that is rarely visited by the stream is by definition a rare situation. In order to detect contextual anomalies, we measure the rarity of the model assigned to x_c in line 4.4.2 of Algorithm 1. Note that observations that do not have an associated context model (point anomalies) are also part of the collection of contextual anomalies. This is because they do not fit any known distribution and are therefore contextual outliers. Fig. 2(b) illustrates what contextual anomalies look like using pcStream. The observations outside the dotted lines are the observations we are testing with respect to contexts c_{1-3} .

In order to keep track of the number of visits each model receives, we extend pcStream in the following way. Let the context model be updated to $c_i \equiv \langle M_i, \mu_i, A_i, v_i \rangle$ where v_i is the number of observations that has been assigned to model c_i . Let r be the rarities (visitation probabilities) of all models, such that $r_i = \frac{v_i}{\sum v_i}$, where $\sum v_i$ equals the number of observations presented by the stream thus far. Using this extension, we can measure the degree to which observation x is a contextual anomaly using the function

$$f(x) = \begin{cases} 0 & \text{if } \min(d_{\mathbf{C}}(x)) > \phi \\ r_{\text{IdxMin}(d_{\mathbf{C}}(x))} & \text{else} \end{cases} \quad (2)$$

where $d_{\mathbf{C}}(x)$ is the vector of the Mahalanobis distances from each of the $|\mathbf{C}|$ models in \mathbf{C} with respect to x , and where $\text{IdxMin}(d_{\mathbf{C}}(x))$ is the index to the model with the smallest score (i.e., the ID of the closest model). The first condition in $Anom2$ means that if x does not fit any context, then x must belong to an unseen context—and therefore is very rare. The

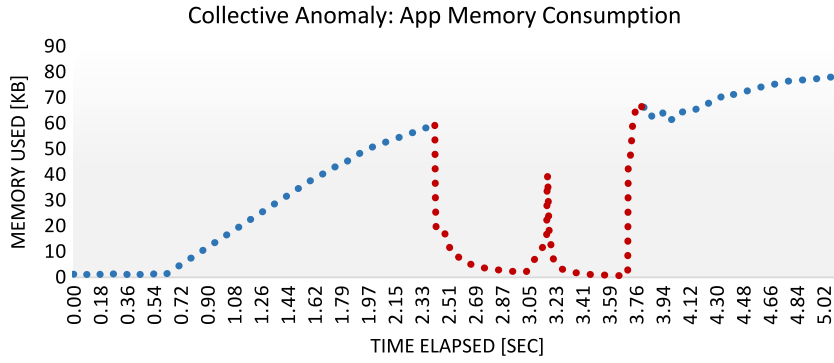


Fig. 5. An illustration of a collective anomaly found in a data stream.

second condition is the rarity of the model to which x belongs. Note that in contrast to function (1), the values returned by function (2) become smaller the more x is considered anomalous.

3.3. Collective anomaly detection

Collective anomalies are a set of related observations which together are anomalous with respect to the dataset, but are not necessarily anomalous individually. These types of anomalies can be found in sequential or time-series type datasets [13]. As an example, consider a data stream created by sampling an application's memory consumption. A single observation from this data stream could belong to an arbitrary activity of the application, and thus examining its value alone is not enough to determine its abnormality. Rather we need to consider the surrounding observations as well. In Fig. 5 we present a simplistic illustration of this example, where the data stream captures the loading of some application. Here, the temporary drop in memory consumption is an abnormal sequence of observations, yet each of the observations themselves is not abnormal on their own (with respect to the entire stream).

In pcStream, a collective anomaly is a rare sequence of transitions between contexts. In order to measure the probability of these transitions, we construct a first order Markov Chain (MC) while training pcStream, where a context is considered a state in the MC. Fig. 2(c) illustrates this concept, where the shade of the transition edges indicates the transition's probability. In the figure transitions to context c_1 are rare and therefore anomalous.

Since we do not know the number of contexts S will exhibit, we model the MC using an extensible Markov Model [25] in the following way. Let $\text{tr}_t \equiv \langle c_{t-1}, c_t \rangle$ be the transition (edge) between a pair of states (contexts) at time tick t (reminder, c_t is the index corresponding to the model assigned to x_t). Let $U_C = [u_{ij}]$ be the transition count matrix where u_{ij} is the number of all transitions which have occurred from model i to j in C . Let $P_C = [\frac{u_{ij}}{v_i}]$ Markov transition probability matrix such that $p(\text{tr}_t) = P_C[c_{t-1}, c_t]$. Note that P_C can be constructed at any time, and that both U_C and P_C are $|C| \times |C|$ matrices.

To detect collective anomalies with pcStream, we maintain the matrix U_C with incremental updates and calculate transition probabilities as they occur: whenever the observation x_t is assigned to a model, we increment the value stored in $U_C[c_{t-1}, c_t]$. When the drift buffer is emptied into the current model c_t (line 4.5.2.1 in Algorithm 1), we first increment $U_C[c_{t-1}, c_t]$ once and then increment $U_C[c_t, c_t]$ with $|D| - 1$. Finally, we output the transition probability for every observation as U_C is updated, using the function

$$\text{Anom3}(\text{tr}_t) = P_C[c_{t-1}, c_t]. \quad (3)$$

Note that Anom3 produces a sequence of first-order transition probabilities $p = \{\dots p_{t-1}, p_t\}$. This can easily be extended to higher orders.

Based on our evaluations (detailed later on), we found that Anom3 can generate many false positives due to low probability transitions between the known contexts. However, we also found that these low probability transitions occurred more frequently in the presence of anomalous segments in the data stream. Therefore, we propose performing smoothing to the sequence p by applying the moving average

$$\text{Anom3}_w(\text{tr}_t) = \frac{1}{w} \sum_{k=0}^{w-1} \text{Anom3}(\text{tr}_{t-k}). \quad (4)$$

4. Evaluation

In this section, we evaluate pcStream as an implicit security solution for the smartphone using the extensions described in Section 3. Specifically, we test whether it is possible to **detect data leakage** (point anomalies), **active malware** (contextual anomalies), and unauthorized users via **continuous authentication** (collective anomalies).

As a comparative baseline in the evaluation of pcStream, we evaluate two other stream clustering algorithms: DBStream [26] and D-Stream [27]. These algorithms have a built-in method for detecting point anomalies, but not contextual or collective anomalies. Therefore, we apply to them the same method proposed for pcStream in Eqs. (2) and (4):

For contextual anomalies, we (1) track the number of visits each cluster receives, (2) divide the counts by the total number of observations produced by the stream, which gives each cluster's rarity, and (3) when a new observation arrives, the degree to which the observation is a contextual anomaly is the rarity of the cluster to which it belongs to. As done with the pcStream version, if the point is an outlier, we return the highest rarity score (zero). For collective anomalies, we (1) model a Markov chain based on the observed transitions between the clusters, and then (2) produce anomaly scores by computing the average probability over a sliding window.

4.1. The dataset

For the purpose of evaluating context-based security solutions for smartphones, we launched a data collection experiment called “SherLock”. The SherLock data collection experiment is an on going (at time of writing) experiment aimed at capturing the long-term behavior of internal and external smartphone actors. What makes this dataset unique is that we are sampling a wide range of software and hardware sensors at a fast sampling rate (“fast” is relative to published datasets discussed later in Section 6). To collect this data, we provided volunteers with new Samsung Galaxy S4 and S5 smartphones. The volunteers were told to use the provided devices as their personal smartphones. The SherLock dataset used in this paper covers eight months of continuous data collection from 31 users (collectively representing approximately 20 years of sensor data). The volunteers participating in the experiment range in age from 12 to 60 years old, and each volunteer has a different level of knowledge, skill, and comfort regarding smartphone use. Since the experiment covers many months, it can be assumed that the users have naturally become adjusted to using their devices, thereby incorporating their personal behaviors and habits into the device's internal and external contexts.

In Table 1 we present the features taken from the SherLock experiment for this paper's evaluations. The reason we selected these features is because they capture both the application's and external user's behaviors in terms of latent contexts. Moreover, other applications can sample these features without any special privileges (verified in Android 6.0), and therefore these features respect the concern of our paper—that data streams are readily available for use in implicitly detecting anomalies where explicit information is not available.

The SherLock collection experiment involved many more sensors than those listed in Table 1. We found that the collection of these sensors depletes the battery of an S5 smartphone in about five hours of regular use. Therefore, in order to collect data at a high temporal resolution while providing a practical battery life, all volunteers were given a battery case for their smartphone which provided the volunteers with an additional 4800 MAh of power to the smartphone's internal 2800 MAh battery (totaling 7600 MAh). We found the power consumption of the sensors to be approximately 5%–7% an hour (measured on a Samsung Galaxy S5 smartphone with all features turned off). Therefore, generating a data stream from the features in Table 1 is a viable solution for the modern smartphone.

4.2. Point anomaly detection

In order to evaluate pcStream as a point anomaly detection algorithm, we consider the following attack scenario: An attacker wishes to steal private or sensitive files from unsuspecting victims. To do so, the attacker performs repackaging and injects malicious code into a *target application*. The *target application* is a common benign application (such as a weather widget). The attacker places the infected application into an unmonitored market (such as an Android black market [28–30]). The victim then downloads the infected application and installs it on his/her device. At some point in time, the malicious code steals a file from the device and transmits it off-site to the attacker.

Our goal is to detect the moment that the malware begins to transmit the file. To achieve this goal, we can train pcStream on a data stream which captures the normal behavior of the *target application*. Assuming the data stream captures the application's behavior across different users, it should be possible to detect an abnormal behavior (such as an unexpected transmission) when the same application exhibits this behavior by a different user.

4.2.1. Evaluation setup—data leakage detection

In our evaluation, we were not able to run actual malware on our volunteers' devices for ethical reasons. Therefore, for these evaluations we performed the following: Let A be an instance of some benign application and let \tilde{A}_m be A infected (repackaged) with the malware m . Let S_A and $S_{\tilde{A}_m}$ be data streams that implicitly capture the behaviors of the respective applications, where an observation \vec{x} of the streams has five numerical features: CPU usage, bytes sent, bytes received, packets sent, and packets received (since the last sample). We define v_m as the average *additional* implicit behavior of \tilde{A}_m with respect to A . v_m is to be calculated by taking the difference between the average values of both streams' features, or formally $v_m = \overline{S_{\tilde{A}_m}} - \overline{S_A}$.

We repeated this process for five different pairs of benign and malicious applications (i.e., $(A_1, m_1), \dots, (A_5, m_5)$). We also extracted the average behaviors from five benign applications (using the same features) in order to provide a wide variety of injected behaviors. To simulate a point anomaly from the malware m_i , in the data stream of a target benign application A' ,

Table 1

Sensor	Features		Sample rate
	Name	Description	
Screen	ON/OFF	A logged event for whenever the device's screen turns on or off.	Per event
	stime utime cstime	Time in clock ticks which this process has been scheduled in kernel mode. Time in clock ticks which this process has been scheduled in user mode. Time in clock ticks which this process's waited-for children have been scheduled in kernel mode.	
Application statistics	cutime	Time in clock ticks which this process's waited-for children have been scheduled in user mode.	0.20 Hz
	CPU usage	An objective measure of CPU usage calculated as the average number of CPU clock-ticks used per second. This measure is computed as $\frac{\Delta cutime + \Delta cstime + \Delta utime}{\Delta WorldTime[s]}$ where Δ refers to the difference between the current and last samples' values.	
	Bytes _{tx}	Number of sent Bytes over the network since last sample.	
	Packets _{tx}	Number of sent Packets over the network since last sample.	
	Bytes _{rx}	Number of received Bytes over the network since last sample.	
	Packets _{rx}	Number of received Packets over the network since last sample.	
	num_threads	Number of threads associated with this process.	
	stime	Time in clock ticks which this process has been scheduled in kernel mode.	
	utime	Time in clock ticks which this process has been scheduled in user mode.	
	cstime	Time in clock ticks which this process's waited-for children have been scheduled in kernel mode.	
	cutime	Time in clock ticks which this process's waited-for children have been scheduled in user mode.	
	dalvikPrivateDirty	The private dirty pages used by dalvik heap.	
	dalvikPss	The proportional set size for dalvik heap.	
	dalvikSharedDirty	The shared dirty pages used by dalvik heap.	
otherPrivateDirty	The private dirty pages used by everything else.		
otherPss	The proportional set size for everything else.		
otherSharedDirty	The shared dirty pages used by everything else.		
Accelerometer	$\bar{a} = (\bar{x}, \bar{y}, \bar{z})$	The mean of the values sampled along each of the 3 accelerometer axis for a duration of 4 s.	0.067 Hz
	$\ \bar{a}\ $	The magnitude of the 3 means, computed as $\sqrt{\bar{x}^2 + \bar{y}^2 + \bar{z}^2}$	
	$FFT_1(x)$	The strongest frequency from the axis "x" taken from 4 s of samples (computed via Fast Fourier Transform).	
Gyroscope	$\ \mathit{FFT}_1(a)\ $	The magnitude of the largest frequencies from all axis, computed as $\sqrt{\mathit{FFT}_1(x)^2 + \mathit{FFT}_1(y)^2 + \mathit{FFT}_1(z)^2}$	
	$\bar{g} = (\bar{x}, \bar{y}, \bar{z})$	The mean of the values sampled along each of the 3 gyroscope axis for a duration of 4 s.	
	$\ \bar{g}\ $	The magnitude of the 3 means, computed as $\sqrt{\bar{x}^2 + \bar{y}^2 + \bar{z}^2}$	

v_{m_i} was added to random observations in the stream $S_{A'}$. A summary of all behaviors extracted with their malware signatures can be found in Table 2.

The *target applications* and the extracted features used in the evaluations are listed in Tables 3 and 4 respectively. In Fig. 6 we present a visual reference regarding where the behaviors lie with respect to the distributions of the *target applications* from Table 3. It is expected that behaviors with a higher or lower CPU/transmission rate than the *target application's* average rates will be easier to detect. However, Fig. 6 shows that the injections fall out across the distributions and not only at the extremes.

We will now explain how the evaluation was performed. For each of 144 different pcStream parameters, we trained a pcStream model over a data stream created from a concatenation of 24 users' usage of a *target application*. We then evaluated each of these pcStream models on each of the remaining seven users' data streams. Each test set was a test user's data injected with one of the malicious behaviors from Table 2. The same datasets were used to evaluate DBStream and D-Stream. Table 5 lists the parameters taken to evaluate the different algorithms.

All data was z-scored using the respective training set, prior to training and testing. For our evaluations we used the area under the receiver operating characteristic (ROC) to measure the algorithm's performance. The ROC curve gives the true positive rate ($TPR = \frac{\#TruePositive}{\#Positive}$) and false positive rate ($FPR = \frac{\#FalsePositive}{\#Negative}$) for every possible decision threshold in a binary classifier's outcome over the supplied dataset. The area under the curve (AUC) is a summary statistic the ROC giving an indication of the classifier's performance. An AUC value of 0.5 indicates a classifier of pure random chance (a 50–50 guess), whereas an AUC of 1.0 indicates a perfect classifier.

4.2.2. Evaluation results—data leakage detection

Table 6 presents a summary of the results in the form of average best AUC across all users. This table provides the expected best performance pcStream can provide at detecting our point anomalies. The best average AUC was found for each *target*

Table 2
The injected behaviors.

Information on the application						The net behaviors that describe the application/Malware				
App name	Ver.	Description	Signature	VirusTotal.com rating	CPU usage	Transmit		Receive		
					Jiffies/s	Bytes/s	Packets/s	Bytes/s	Packets/s	
Malware	AVG antivirus	4.4.2	Virus scanner	SMSsend	1/56	78.15	232	8	21 358	14
	Live locker	2.6	Screen lock	Trojan	25/56	11.12	6	1	0	0
	Bug war	1.1	Game	Trojan	15/56	23.78	3	1	0	0
	Santa ride	1.0.1	Game	Trojan	16/57	1.64	30	2	394	1
	Temple	1.0	Game	Spyware	25/56	9.38	14	1	4	1
Benign	Polaris viewer	6.5.4	Doc. Reader	Clean	0/56	9.99	536	1	295	1
	Video player	2.0.0	Media player	Clean	0/56	9.77	12 372	47	285 982	67
	Memo	2.0.83	Notes	Clean	0/56	2.87	976	2	807	2
	Dropbox	3.0.4.2	Cloud storage	Clean	0/56	1.76	74 914	13	1 924	9
	Google Maps	9.14.0	Navigation	Clean	0/56	6.87	2 824	3	2 853	4

Table 3
The datasets extracted from the *target applications*.

Target application	# Rows in dataset	Collective timespan of dataset	# Rows in train	# Users in train	# Test users	# Injected behaviors	# Test-sets per target App.
Google Maps	256,505	14.8 days	198,454	30			
Video player	257,051	14.9 days	198,991	31			
mvPlayer	199,728	11.6 days	89,818	26			
Memo	256,848	14.9 days	198,803	30			
Dropbox	164,840	9.5 days	120,890	31	7	10	70
AccuWeather	257,051	14.8 days	198,687	31			
Widget							
Digital Clock	143,361	14.9 days	143,361	27			
Widget							
Polaris Doc	130,903	14.8 days	112,021	19			
Viewer5							
WhatsApp	265,606	15 days	207,492	30			

Table 4
Data stream features extracted—per *target application*.

Source	Units	# Features
CPU usage (Jiffies/s)	Jiffies/s	1
$Bytes_{rx}$, $Bytes_{tx}$	bytes	2
$Packets_{rx}$, $Packets_{tx}$	packets	2
dalvikPrivateDirty, dalvikPss, dalvikSharedDirty, otherPrivateDirty, otherPss, otherSharedDirty	kilobytes	6

Table 5
The pcStream, DBStream, and D-Stream parameters selected for the evaluations.

pcStream		DBStream	
Parameter	Value	Parameter	Value
ϕ	0.7–4.2, on 0.1 intervals	Micro cluster radius	1–12 on 0.1 intervals
t_{\min}	3, 4, 5, 6	Sets evaluated: 111	
m	500		
ρ	0.98		
Max model limit (context preservation)	200		
Sets evaluated: 144		D-Stream	
Parameter	Value	Parameter	Value
		Grid size	0.1–10 on 0.05 intervals
		Sets evaluated: 199	

application by performing a grid-search across the ϕ and t_{\min} parameters of pcStream. For each combination of parameters (i.e., point in the grid), we computed average AUC across the *target application*'s test sets. The point in the grid that provided the highest average AUC was considered to be the best.

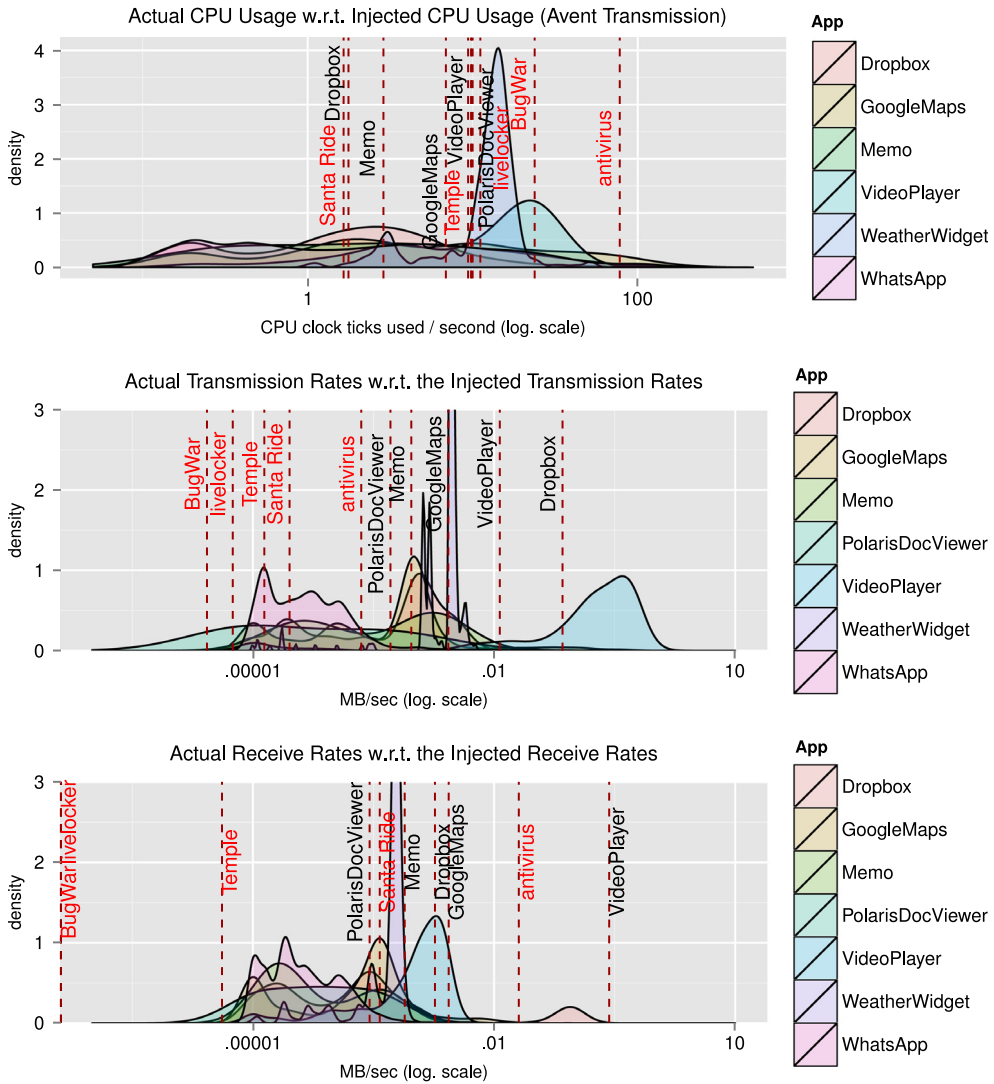


Fig. 6. The three distributions of the target applications' resource utilization. Vertical dashed lines show the values of the injected behaviors (v_m).

To give an idea of the difficulty in determining these parameters (i.e., parameter selection robustness), in Figs. 7 and 8 we plot the *Anom1*'s performance across a grid of the ϕ and t_{\min} parameters. In these figures, brighter colors indicate better performance (i.e., a higher AUC). Fig. 7 shows the robustness in the perspective of the same test user and injected application across the different *target applications*. Fig. 8 shows the averaged AUC of all users and injected behaviors for each of the *target applications*. This plot shows that even when the malicious behavior is unknown, selecting a parameter around $\phi = 4.1$ and $t_{\min} = 5$ provides a good AUC on average (regardless of the user). This generalization is only applicable when the data stream is extracted from the *target application* in the same way as was done in this paper.

From these figures we can see that *Anom1* successfully detects the point anomalies with high accuracy and (a low number of false positives). Each parameter selection (coordinate in the plot) captures a *context category* populated with different clusters found in the test user's data stream. We note that each of the test users and *target applications* have their own unique contexts. pcStream was able to detect these contexts and use them to successfully differentiate between an injected behavior (outlier) and the regular behavior. Fig. 9 shows the performance of pcStream in comparison with DBStream and DStream. We note that pcStream consistently has a better AUC than the other algorithms, regardless of the *target application* or injected behavior.

Lastly, Fig. 10 shows the training times (in minutes) for each of the parameters from Table 5 on the training set from the *target application* WhatsApp. The training was performed on a single logical core on an Intel Xeon E5-2660 v3 processor (a core in a c4.8xlarge virtual machine running in the Amazon EC2 compute cloud). Note that higher ϕ parameters result in a shorter training time. This is because a higher ϕ makes it more difficult for an observation to be placed into the drift buffer D . Therefore general contexts are detected, resulting in fewer models to manage at every update. Since the pcStream model

Table 6
Average best AUCs across all users.

	Injected behavior									
	Antivirus	Bug war	Live locker	Santa ride	Temple	Dropbox	Google Maps	Memo	Polaris DocViewer	
Dropbox	1.0000	0.9996	0.9986	0.9950	0.9985	0.9995	0.9999	0.9971	0.9985	
Google Maps	1.0000	0.9997	0.9996	0.9970	0.9996	0.9999	0.9999	0.9996	0.9997	
Polaris viewer	1.0000	0.9908	0.9820	0.9611	0.9808	0.9914	0.9910	0.9779	0.9842	
Memo	1.0000	1.0000	1.0000	0.9998	1.0000	1.0000	1.0000	1.0000	1.0000	
Video player	1.0000	0.9989	0.9959	0.9689	0.9945	0.9895	0.9880	0.9783	0.9854	
Weather widget	1.0000	1.0000	0.9996	0.9989	0.9995	1.0000	1.0000	0.9958	0.9974	
Clock widget	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
WhatsApp	0.9999	0.9985	0.9899	0.9648	0.9881	0.9994	0.9840	0.9433	0.9708	
mvPlayer	1.0000	0.9997	0.9999	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
Dropbox	1.0000	0.9996	0.9986	0.9950	0.9985	0.9995	0.9999	0.9971	0.9985	

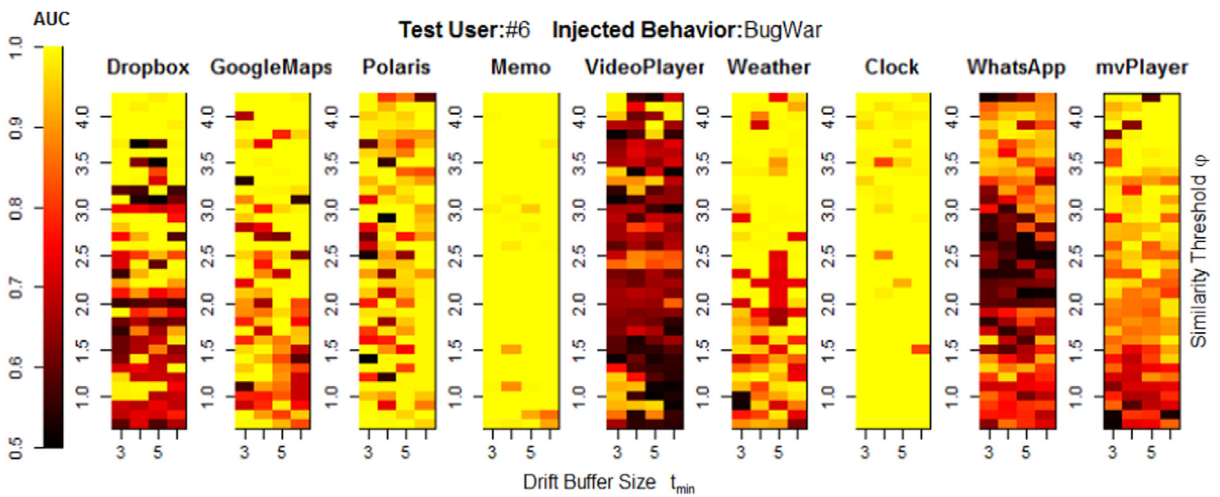


Fig. 7. A plot of *Anom1*'s pcStream parameter selection robustness across different *target applications* (with the same injected behavior and test user). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

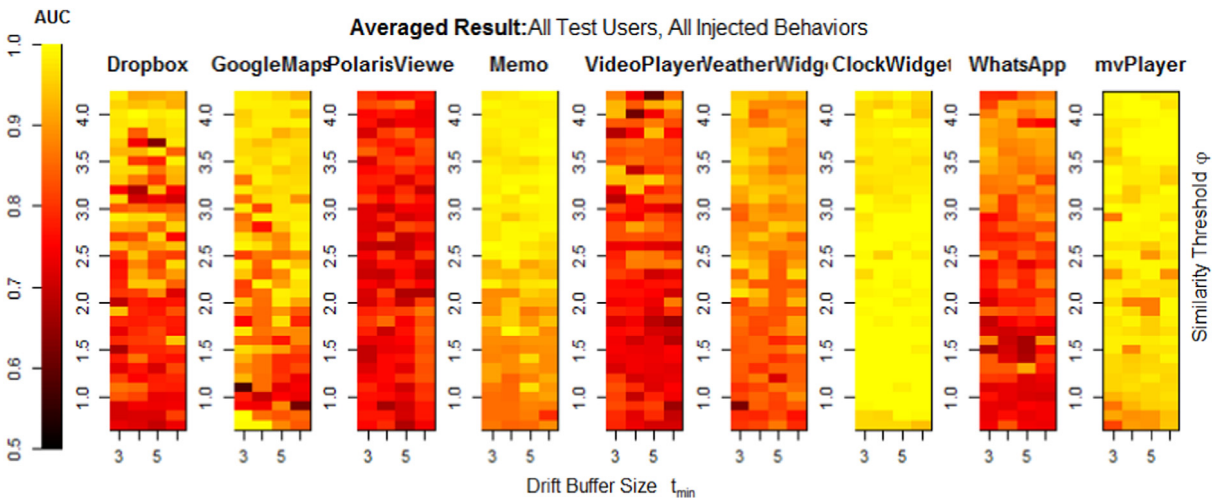


Fig. 8. Heat plots visualizing pcStream's performance over different parameter settings. The colors represent the obtained AUC score averaged across all test users and all injected behaviors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

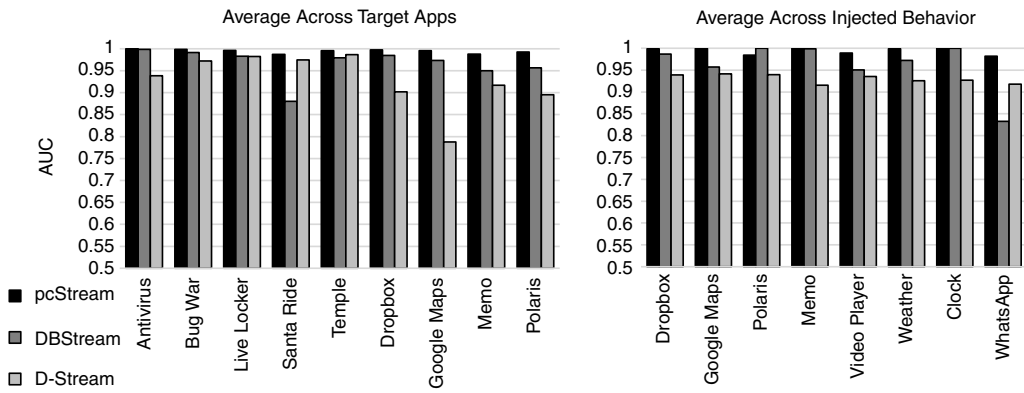


Fig. 9. A comparison of stream's point anomaly detection performance with respect to the *target applications* (left) and *injected behaviors* (right).

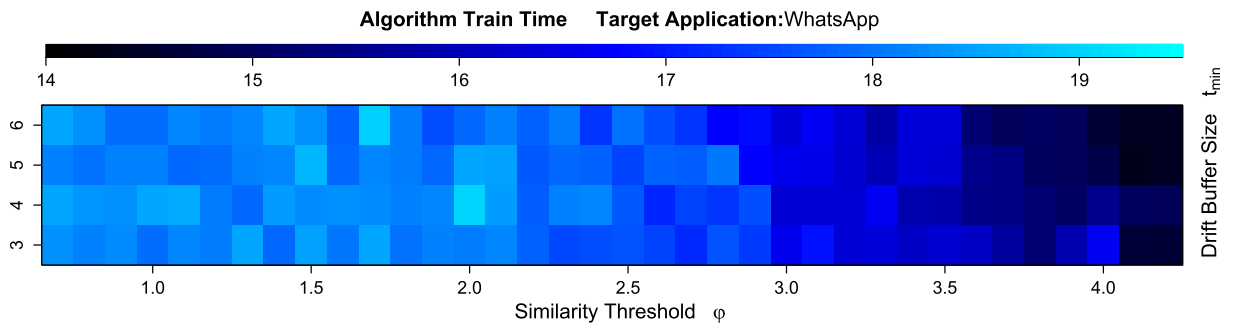


Fig. 10. The pcStream elapsed train times for different parameters over the WhatsApp dataset.

in this case is a global (not specific to any one user), the algorithm can be trained off-site (not on the smartphone itself). It took 15.3 min to train a model for WhatsApp using the best parameter—approximately 4.4 ms per observation.

4.3. Contextual anomaly detection

For the evaluation of pcStream as a contextual anomaly detection algorithm, we generalize the attack scenario from Section 4.2.1. The scenario was that the user has downloaded and installed an infected application from an unmonitored Android marketplace. In this section, we consider that the attacker's goal is to execute arbitrary code multiple times throughout the day. For instance, the attacker wishes to spy on the user's location or collect relevant information on the device for targeted adware purposes [2]. To maintain a constant presence, the attacker has the malware (inside the *target application*) perform its malicious acts even while the user is not interacting with the device.

Observing *behavioral attributes* does not always provide enough information to detect anomalies. In these cases, *contextual attribute(s)* are required to detect abnormalities in the *behavioral attributes*. In the current attack scenario, we use the *CPU-usage* and *num_threads* of the *target application* as the *behavioral attributes* in order to detect anomalies caused by malicious code. The reason we chose this set as the behavioral features is because (1) the features are correlated thereby capturing the application's behavior, and (2) while the malicious behavior is active these features can exhibit the *same expected behavior* as seen during normal usage. As the *contextual attribute* we consider the device's motion in order to distinguish between legitimate and illegitimate behaviors exhibited by the *behavioral attributes*.

As a more concrete example, assume that the *target application* is a navigation application which is infected with spyware that tracks the user's location (even while the application is not in the foreground). Here, without explicit information it is difficult to determine whether the application's observed behaviors are warranted. However, if we know that the clean version of *target application* only exhibits these behaviors while the device is in the user's hands, then all other scenarios indicate a contextual anomaly.

Therefore, our goal is to detect malicious behaviors as they occur outside their expected contexts.

4.3.1. Evaluation setup—malware detection

We will now go into detail regarding how the evaluation was performed. Similar to our assumptions described in Section 4.2.1, we assume that we have access to the data streams from multiple users running the same *target application*. We also assume we are able to group these users according to their device model (so as to insure that the accelerometer readings have a similar bias). Using these assumptions, for each *target application* we created one training set consisting

Table 7

Target train and test data stream sizes.

Target application	# Rows in train	Collective timespan of train (h)	# Rows in test	Collective timespan of test (h)	# Injections
Dropbox	74,032	103	33,984	47	39
Google Maps	203,901	283	33,982	47	205
Memo	211,638	294	30,688	43	113
Video player	237,815	330	33,977	47	13
Weather widget	237,817	330	34,001	47	17
Whatsapp	237,988	330	34,001	47	3242

Table 8

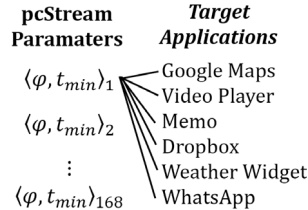
The pcStream, DBStream, and D-Stream parameters selected for the evaluations.

pcStream		DBStream	
Parameter	Value	Parameter	Value
φ	1–5.1, on 0.1 intervals	Micro cluster radius	0.2–30 on 0.1 intervals
t_{\min}	3, 4, 5, 6	Sets evaluated: 299	
m	500	D-Stream	
ρ	0.98	Parameter	Value
Max model limit (context preservation)	2000	Grid size	0.1–10 on 0.01 intervals
Sets evaluated: 168		Sets evaluated: 991	

Table 9

Data stream features extracted per application.

Source	Units	# Feat.	Type
CPU usage	Jiffies/s	1	Behavioral
Num_threads	–	1	Behavioral
$\ \bar{a}\ $	m/s ²	1	Contextual

**Fig. 11.** A tree summarizing the experiments performed in this section.

of seven users' data streams joined together sequentially. Each *target application* was then accompanied by a test set (the remaining users' data stream).

To create the contextual anomalies, we randomly selected observations from the test user's data stream and replaced the values of $\|\bar{a}\|$ with those from another context. However, changing the value of $\|\bar{a}\|$ based on observations of $\|\bar{a}\|$ would be biased. Therefore to change the context implicitly, we randomly selected half of the observations which were sampled while the screen was off. We then changed their $\|\bar{a}\|$ values to the values of $\|\bar{a}\|$ found in random observations when the screen was on.

In summary, the simulated anomalies (*injections*) are instances where the CPU Usage and num_threads appear normal, but the motion of the device indicates otherwise.

The *target applications*, along with details relating to their number of *injections*, are presented in Table 7. The features used to make the data streams are listed in Table 9. The pcStream parameter used across all of the experiments is presented in Table 8. For the reader's convenience, a visual summary of the evaluation can be found in Fig. 11. We chose the applications listed in Table 7 as the target applications, because they were the most common among all users who had the same device (in this case a Galaxy S5).

4.3.2. Evaluation results—malware detection

The performance of function (2) across the different pcStream parameters is presented in Fig. 12. It is interesting to see how the messaging application *WhatsApp* performed poorly in comparison to the other applications. This is because the training sets are made up of the data stream of multiple users. The contexts (i.e., device motion) under which each of these users uses WhatsApp are highly personal to each user's behaviors and habits. In contrast, the contexts in which each of the

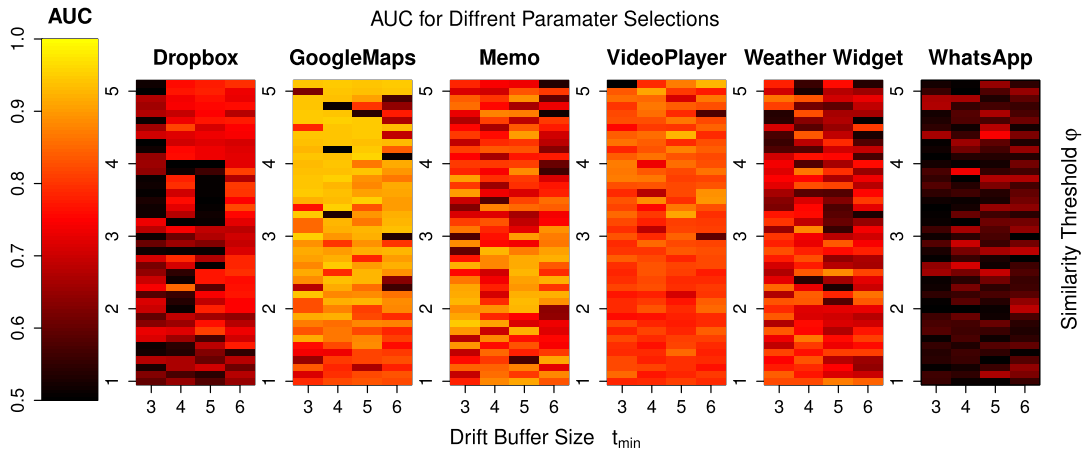


Fig. 12. The performance of *Anom2* in detecting the contextual anomalies over different pcStream parameters.

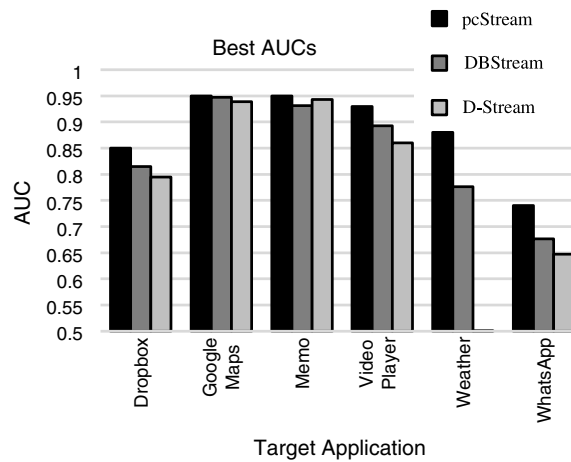


Fig. 13. A performance comparison: highest achieved AUC.

Table 10

The parameters which achieved the best performance.

Target application	Best AUC	Best parameters	
		ϕ	t_{\min}
Dropbox	0.85	2.3	4
Google Maps	0.95	4.1	5
Memo	0.95	1.8	3
Video player	0.93	4.4	5
Weather widget	0.88	2.5	5
Whatsapp	0.74	4.4	5

users uses the other applications is similar across all of the users. Therefore, it is important to make this distinction when applying function (2) to datasets which combine multiple behaviors.

The best parameters and accompanying AUC values for each *target application* are presented in Table 10. In Fig. 13, we compare pcStream’s contextual anomaly detection performance to other algorithms.

In Fig. 14, we present the algorithm’s training time for each of the parameters (measured in minutes), along with the number of contexts found. The *target application* presented is Google Maps. It took 4.8 min (using a c4.8xlarge instance in the Amazon EC2 cloud) to train a model for Google Maps using the best parameter (a rate of approximately 1.4 ms per observation). Similar to the method proposed in Section 4.2.1, the current method involves a global pcStream model (one that is not specific to any one user). Therefore the algorithm can be trained off-site.

Fig. 14 also demonstrates the relationship between the pcStream parameters and the contexts discovered. Specifically, larger ϕ values find more distinct contexts, and a larger t_{\min} finds contexts that are more long-term. Therefore it is expected

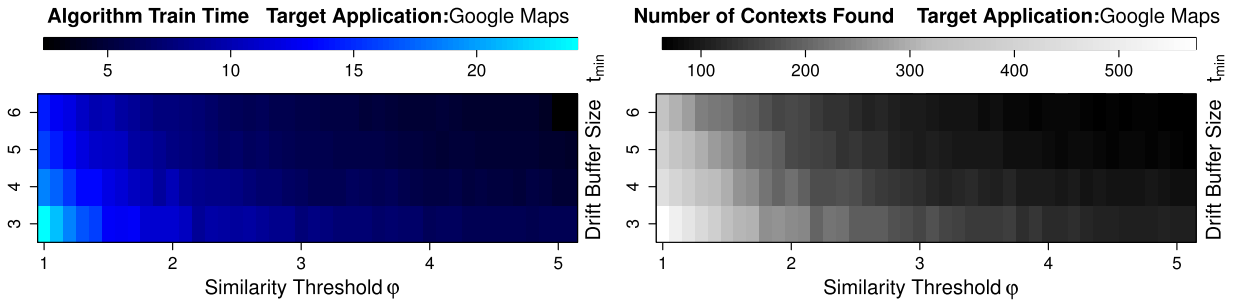


Fig. 14. The train time (left) and number of discovered contexts (right) for each parameter while training on the *target application* Google Maps.

that smaller values will discover more contexts in the data stream, since there are naturally more *nuances* than global behaviors.

It is clear that the training time increases with the number of discovered contexts. This is because we calculate the distance between every new observation and all existing contexts in line 4.2 in Algorithm 1. In the case where the overhead is too high, the number of models can be limited by merging older ones (see Section 2.5).

4.4. Collective anomaly detection

To evaluate pcStream’s ability to detect collective anomalies, we investigate detecting the anomalies caused by the malicious activity of an external actor (a human), as opposed to the previous sections where we considered internal actors (applications and code). We propose the following attack scenario. The attacker is an external actor who does not regularly use the target smartphone. The attacker wants to either steal or access an application on the device. To do so, the attacker waits until the device is left unattended on a table and then picks it up and begins using it. In this scenario we assume that the attacker is familiar with the model of the phone and interface.

It has been shown that every user has a unique way of handling a smartphone, and that the device’s motion can be used as a biometric [31]. Further studies have shown how these biometric signals can be used to provide continuous authentication (i.e., we assume the current user is the authentic user until shown otherwise and then subsequently lock the device). However, state-of-the-art methods rely on the attacker to perform specific activities. For instance, by analyzing the way the user walks (gait analysis) [32,33], touches the screen [34–36], or roams between different locations [37,38], etc. This is problematic, because an attack could be perpetrated without such activities. For instance, access to an application evades gait and location analysis, and device theft evades touch recognition. Moreover, for methods like location and gait-based authentication require the device to travel—incurring significant delays in detection time. With shorter delays, access to an application can be blocked before any damage is done, and the user can be notified of a theft sooner so that the thief can be stopped before leaving the scene.

Our goal is to provide users with a generic solution for continuous authentication that does not rely on the user to perform any particular activity in order to verify his/her authenticity. To accomplish this goal we assume that the motion caused by a user on his/her smartphone is unique to that user. Therefore, we assume that the sequence of contexts observed from the device’s motion is also unique to that individual’s behavior, body form, and habits. Furthermore, since the contexts are correlated distributions, the motions associated with walking or answering a call will not match a different user’s respective distributions. We use function (4) to provide continuous authentication for a *target user*.

4.4.1. Evaluation setup—continuous authentication

In order for the evaluation to be fair, all of the phones must be of the same hardware/model. Our dataset contains a mix of Galaxy S4 and S5 phone series, each with different models such as those with either quad or octa-core CPUs. Therefore, for this evaluation, we only used the data of eight volunteers, all having the exact same model of S5. Each of the eight users was selected to be the *target user* (i.e., the victim) where the seven remaining users formed the test sets (i.e., simulated thieves).

The training sets consisted of 180,000 observations (31 days of data) from the *target user*. Each test set was formed by concatenating the next 20,000+ observations from the *target user*, with 30,000 records from the respective test user (approximately nine days of data altogether). The transition point between the *target user* and test user was selected so that the *target user*’s device was motionless (e.g. on a table) for *at least* eight observations and so the test user’s device was just about to be in motion. These test sets effectively simulate the situation in which *target user*’s device is taken by a foreign user from a stationary location and used by this test user. To better simulate this scenario, observations indicating stationary motion in the test user’s data were removed.

After training on each *target user* with each of the parameters in Table 11, we attempted to detect the test user with $Anom3_w$ using window sizes of 1 through 14. For the reader’s convenience, Fig. 15 shows the evaluation trials performed, and Fig. 16 illustrates how the training and test sets were generated (i.e., the pairings between columns 3 and 4 in Fig. 15). Table 12 lists the features extracted to form the data streams.

Table 11

The pcStream, DBStream, and D-Stream parameters selected for the evaluations.

pcStream		DBStream	
Parameter	Value	Parameter	Value
φ	0.6–1.5 on 0.05 intervals	Micro cluster radius	1–12 on 0.01 intervals
t_{\min}	3–12	Sets evaluated: 1101	
m	5000	D-Stream	
ρ	0.98	Parameter	Value
Max model limit (context preservation)	2000	Grid size	0.1–15 on 0.01 intervals
Sets evaluated: 190		Sets evaluated: 1491	

Table 12

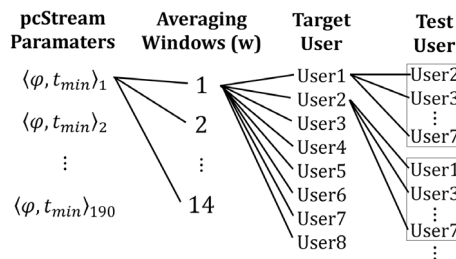
Data stream features extracted per target user.

Source	Units	# Features
$\ \bar{a}\ $	m/s ²	1
$\ FFT_1(a)\ $	Hz	1
$\ \bar{g}\ $	rad/s	1

Table 13

Summary of results.

Target user	FPR	FPs over test set		FPs per day	Detection delay (min)	
	Mean	Mean	Std.	Mean	Mean	Std.
User 1	7.86E–05	1.571	2.225	0.453	1.929	1.179
User 2	4.29E–05	0.857	1.069	0.247	2.071	2.207
User 3	2.29E–04	4.571	9.624	1.317	1.893	1.361
User 4	4.29E–05	0.857	1.574	0.247	1.429	0.965
User 5	0	0	0	0	0.500	0
User 6	4.36E–04	8.714	11.161	2.510	6.893	8.410
User 7	2.81E–03	56.143	101.570	16.169	3.393	2.313
User 8	0	0	0	0	0.500	0

**Fig. 15.** A tree summarizing the experiments performed in this section.

4.4.2. Evaluation results—continuous authentication

Similar to *Anom1* and *Anom2*, a cut-off threshold must be selected such that all values above the threshold indicate an anomaly. In the previous evaluations we used AUC which essentially considers all relevant thresholds. However, in addition to the false positive and true positive rates, there is another dimension that must be considered in continuous authentication: detection delay. For instance, it may be possible to get a very high AUC, but doing so requires a very long averaging window—causing a substantial delay. Such a delay may be more than enough for an attacker to take the targeted information off the phone or get away with the stolen device.

Therefore, in order to automate the process of selecting the best parameters, we used the metric

$$\text{Metric}(FP, DD) = FP + DD \quad (5)$$

where FP is the number of false positives recorded over the dataset (nine days), and DD is the detection delay in minutes. Note, since the ratio of FPs to detection delay is a matter of user preference, further consideration should be given regarding another more appropriate metric than (5). For now, we leave this as a matter for future work.

Using (5) we were able to find the best combination of pcStream parameters for training the algorithm, and setting the window size of *Anom3_w*. Table 13 presents a summary of the best results (using this metric) for each of the test users. It is clear from these results that *Anom3_w* is capable of providing continuous authentication for smartphone users with relatively

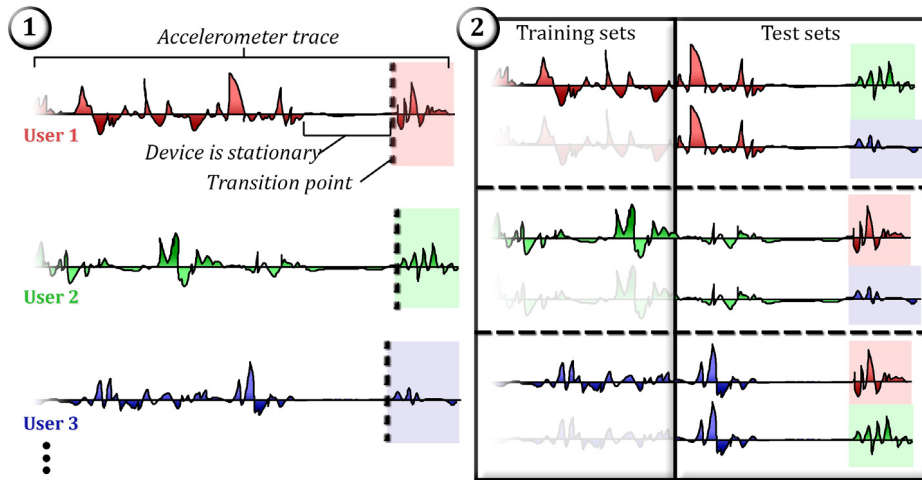


Fig. 16. An illustration of how the training and test sets were made. (1) The division of each user's accelerometer trace, and (2) the creation of the datasets where (rows) are the datasets for each target user, and (columns) are the training and test sets respectively.

Table 14
Performance comparison: average over test users.

	FP per day	Detection delay (min)	
	Mean	Mean	Std.
pcStream	2.618	2.326	2.054
DBStream	44.176	43.357	75.488
D-Stream	28.605	38.674	35.526

few false positives and a rather short detection delay. We found that the reason for the short delay is because the transition between the *target user* and test user causes a very low $Anom3_w$ score. In addition, since the data stream in this evaluation produces observations at 0.2 Hz, a 1 min delay means that the attacker was detected after 4 observations. Therefore, it is conceivable that a faster sample rate will decrease the detection delay further. Table 14 shows that pcStream performs better than the other algorithms in collective anomaly detection. This is likely because pcStream is designed to detect situations (contexts) implicitly from data streams.

There are two reasons why $Anom3_w$ is successful at providing continuous authentication based solely on the motion of the device. Firstly, the same context experienced by different users generates different distributions (e.g. the way a user walks). Secondly, the sequence of contexts a user produces is unique to that user's behavior, subject to concept drifts (e.g. taking the phone out of a pocket to answer a call).

In Fig. 17 we present the distributions of the results for each user. In column 1 of Fig. 17 are the distributions of the results across each user as the *target user*, and column 2 are of each user as the test user (here the number of FPs is across the entire test set—nine days of data). An interesting phenomena we can see from Fig. 17 is that Users 6 and 7 score relatively poorly as *target users* but are quite easy to detect as test users. This is because these users visit rare contexts from time to time and their popular (frequently visited) contexts are concentrated similarly to those of other users, thus they are hard to protect and detect.

Another interesting case can be seen with Users 5 and 8. These users have consistently FPs when selected as the test user, but generate many FPs when they are selected as a test user. The reason for this is because these users have contexts which are common to all other users. However, the other users have many other personal contexts which do not exist by users 5 and 8 respectively. Therefore, it is easy to detect when the stream of User 5 or 8 changes to another user's stream, however, it is challenging to detect when some user's stream changes into the stream of User 5 or 8. In summary, these two users are easy to protect from device theft, but are hard to defend against as thieves.

In general, the results in Fig. 17 show that we can detect an attacker who attempts to steal the *target user's* device reasonably well, even if the attacker's behavior is similar to that of the *target user's*. However, if the *target user* exhibits common contexts which are very similar to the attacker's (as well as others), the target user will suffer a higher FPR.

Fig. 18 plots the anomaly scores calculated by $Anom3_w$ where the *target user* is User 5 and the test user is User 6. The figure shows that User 5 has distinctly different behavior than User 6, and therefore we can detect the moment of the theft quite easily. To visualize the differences between two users' contextual behaviors, we present the data streams and Markov models of Users 1 and 8 (each with their own pcStream model trained with the same parameters). Fig. 19 plots 180,000 observations (31 days) from Users 1 and 8's data streams, where the observation's color indicates the assigned context (with respect to each user's model collection C), and darker colors indicate contexts detected earlier on. Fig. 20 plots the

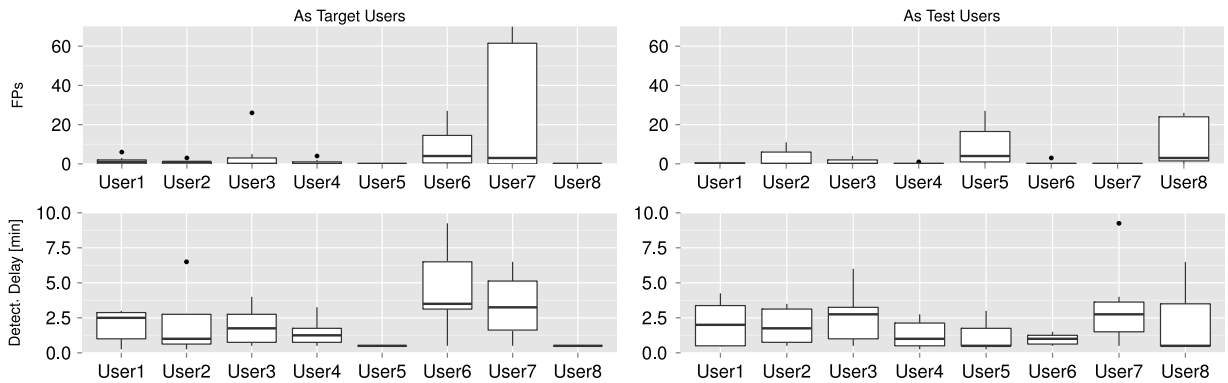


Fig. 17. The distribution of the results across the users as the *target user* (column 1) and as the *test user* (column 2).

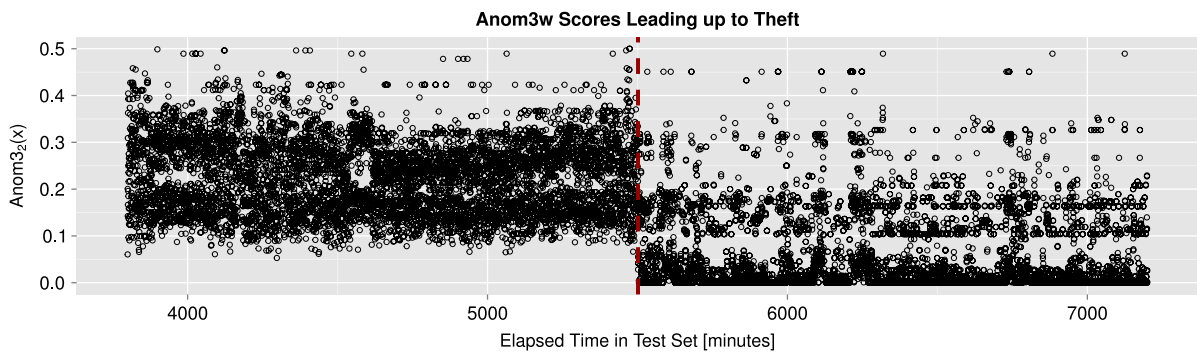


Fig. 18. The $Anom3_w$ scores leading up to the theft (marked in red). *Target user*: 5, *test user*: 6. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Markov models of the users using the graph visualization tool Gephi [39]. There, each vertex is a pcStream context (state in the Markov model), and an edge indicates a recorded transition. The size of a vertex indicates the context's in-degree (popularity), the color of a vertex indicates the context's community using the algorithm from [40], and the color of an edge indicates the transition's probability (darker is more probable).

We note that should an attacker use the device while it is on a table top, it will be unlikely that the proposed approach will successfully detect the attacker. To overcome this weakness, either (1) the user should be unauthenticated if there is no device motion for a period of time, or (2) the overall CPU utilization can be incorporated into the stream so that the context of internal actors will be captured as well.

5. Discussion

Parameter selection. Like many machine learning algorithms, pcStream has multiple parameters, namely t_{\min} and ϕ . Finding the optimum parameters can be challenging, especially since stream datasets can be very long and therefore involve a lengthy training time. In this paper we performed a non-exhaustive gridsearch over these parameters, where each point in the grid is a set of parameters. For larger problems, one may consider using a hyper-parameter selection technique which performs a minimal number of trials. For example, we can begin the search with the default hyper-parameter setting and then select the next point with best-first search and cross-validation evaluation procedure [41]. Another option is to use the design of experiments (DOE) approach, where the parameters are optimized by following a systematic method of learning their relationship to the response feedback [42]. More recently, Bayesian optimization has shown to be effective for finding the optimal hyper-parameter settings. The idea is to construct a probabilistic model and exploits this model for the next setting to evaluate [43].

Parameter tuning is not possible when there is a no feedback, i.e., the lack of a labeled dataset. This commonly occurs in the task of anomaly detection where there is an abundance of normal data, but explicitly labeled anomalies are scarce. For these situations we offer the following guidelines to selecting a decent set of parameters for pcStream. First, one should observe and analyze the stream to determine the interesting types of contexts it exhibits. For t_{\min} , this parameter determines the term-length of the contexts. Therefore, if the shortest duration of an interesting context is 500 observations, then t_{\min} should be about the same value. For ϕ , this parameter controls the distinctiveness of the detected contexts. Therefore, if

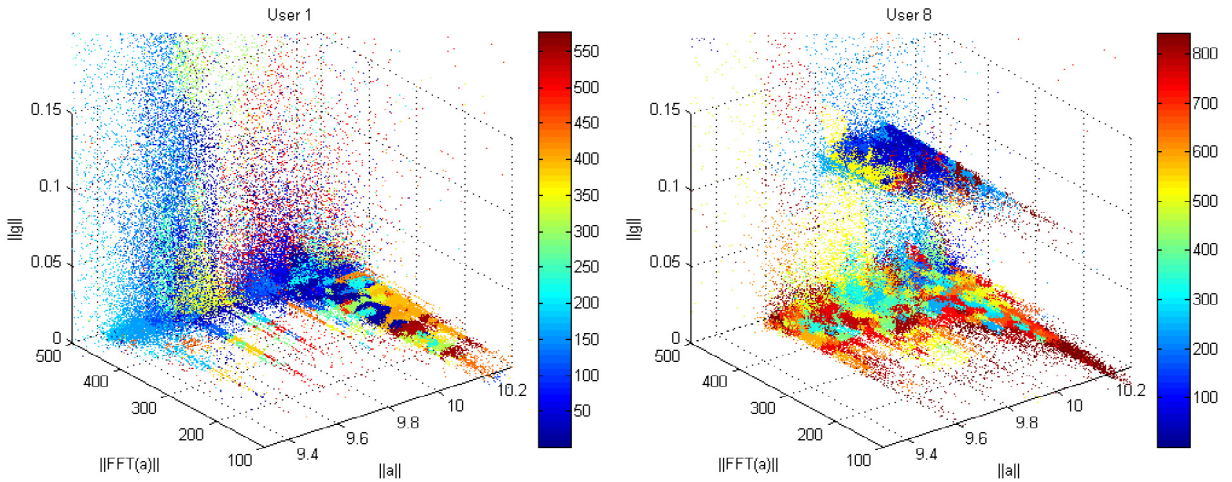


Fig. 19. Users 1 (left) and 8's (right) data streams from one month of collection. Colors indicate the assigned context. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

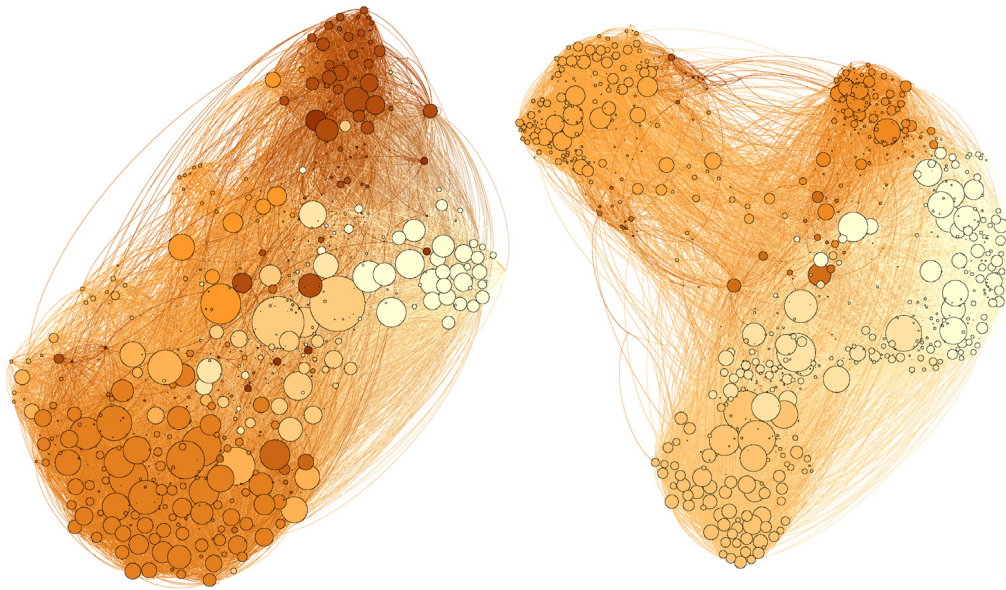


Fig. 20. The Markov chains of Users 1 and 8.

the interesting contexts are more unique behaviors and less like small nuances, then a $\phi < 1$ should be selected. Finally, m considers the duration of the concept drifts of the contexts. If the interesting contexts hardly change, then a large m should be chosen. Otherwise, an m that reflects the length of the drift should be selected.

Algorithm complexity. As mention in [18], pcStream has a complexity of $O(mn^2)$ for processing an observation, where m is the model memory size, and n is the stream's dimensionality. The upper bound depends on m , n and the limit on the number of models. To consider what is practical, one should consider the velocity of the target stream. For the continuous authentication evaluation, it took 5 ms to process each observation. Since the stream only produced one observation per 15 s, the complexity of the algorithm is not an issue. However, as a general guideline, we suggest selecting no more than the 3–10 features that have the lowest self (inter) correlation as the data stream. As future work, we intend on investigating incremental PCA (IPCA) as a means of improving the complexity of pcStream.

Protecting the algorithm. It is important to note two security concerns related to the work presented in this paper. First, in this work some of the training sets are based on a collection of other user's data. Since machine learning algorithms are susceptible to poisoning attacks [44], one who implements this method should consider that an attacker may be present among the training users. To mitigate this issue, we suggest that *leave one out cross validation* be performed across the users in the training set, and that the users who performed poorly be removed. The second concern is that the algorithm is context-based. This means that the algorithm is susceptible to malicious actors who are context-aware, because the algorithm can be

evaded if the malicious activities are performed within the expected contexts. One possible solution is to protect the system with a subsystem that measures the contextual awareness of the relevant actors. This can be accomplished by tracking the actors' (e.g. applications) access to contextual information (such as sensors). Another possible solution is that the subsystem can create "fake contexts" in order to trigger the context-aware attacks outside the normal context. This subsystem could either warn the user or change the sensitivity of the algorithm in an attempt to lower the false positive rate. These concepts have not been verified, and are therefore a subject of future work.

6. Related works

6.1. Related literature

As mentioned, in this study we aim to detect anomalies when explicit information on the application, device, or user behavior is unavailable. In such cases, we opt to analyze sensor data to extract the latent contexts which can be used to detect anomalies.

With the evolution of smartphones, the importance of implementing context-aware security for such devices has been acknowledged. One example includes the CrePE system [45] which introduced a context-based policy enforcement on applications. CrePE provides a fine-grained access control to applications accessing different resources on the device based on the observed context. The context-based policy is defined by the owner of the device, as opposed to the proposed approach in our paper which is data-driven.

Another example is CASA [46] which proposed a dynamic authentication approach. Upon activating the screen, CASA presents the user with an appropriate authentication method (depending on its trust of the user). The trust is determined by examining the context of the user and by using, for simplicity, Nave Bayes as the underlying model. This model assumes a level of conditional independence between each contextual factor. In their evaluation, the authors specifically use the probability of the user's current location as the context. Compared to CASA, the anomaly detection extensions in our paper assume a level of conditional independence between factors. In addition, the pcStream anomaly detection extensions continually provide an analysis of the situation, since each observation in the data stream is classified as normal or anomalous. However, CASA provides such analysis only at the occurrence of specific events, and therefore provides less coverage.

The SenSec system [47] provides continuous authentication for smartphone users based on sensory data collected from the accelerometer, gyro-scope, and magnetometer sensors. SenSec aggregates the sensor information by segmenting the stream into equal sized parts, extracting features from each part, and uses the k-means algorithm to detect concepts. Anomalies are detected by observing the probability of transitions from one cluster to another. Our proposed solution differs from SenSec in the following ways. First, pcStream inspects observations on an individual basis and does not aggregate the information by segmenting the stream into parts. The segmentation process is prone to the loss of information and indications of anomalies. Second, SenSec applies the k-means algorithm to detect the concepts, however, k-means seeks to find a partition of the data in the feature space which is problematic if it is assumed that concepts may overlap (see Fig. 1). Furthermore, k-means is not a streaming algorithm and therefore must store all observations. Not only is this a memory issue, but the increasing size of the collection results in increasing complexity when adding new observations. In contrast, pcStream is specifically designed to handle unbounded data streams as well as handle concept drifts.

Context-based anomaly detection was also explored in previous studies. In [48] a knowledge-based intrusion detection system for Android was presented. Using an expert's knowledge, sensor data is interpreted within the current context in order to detect meaningful patterns that indicate a potential anomaly or threat. This however, requires a human expert to model the threats and contexts. The TCADS framework was also proposed as a general concept of context-based anomaly detection [49]. The framework uses sensor data from the device or external sources (e.g., from an enterprise's IT system such as a network-based intrusion detection system) in order to detect anomalies. However, in this paper only a conceptual description of the framework is presented and no specific algorithm or evaluation is offered. The importance of context-based anomaly detection was also demonstrated by Dixon et al. [38] who analyzed the power consumption within a context. There, contexts are defined as time and location. In this study, our approach is more generic as we derive concepts from a wide range of data streams.

Lastly, in [50] we explored the reductions from different classes of multiplicative path finding problems to suitable classes of additive path finding problems. In order to demonstrate these reductions, we proposed a few variations of *collective anomaly* detection metrics. These metrics search a Markov model for optimum paths (in terms of probability) and use these paths to understand the probability of the current situation. In [50] we used pcStream to create the Markov model in the evaluation. In order to compute the anomaly score for a single observation, one must solve the optimal path problem, which takes $O(|C|k)$ time, where k is the path length considered (typically about 50 transitions). The time complexity of $Anom_w$ is $O(1)$ since it takes the mean of a window of probabilities. As a comparison, $Anom_w$ computes an anomaly score in about 55 μ s, and the metrics in [50] compute scores in about ten seconds with 1000 contexts. Even when sampling at the maximum rate of one sample per ten seconds, the detection delays become unacceptable and the smartphone's processor would always be operating at full capacity (never entering power saving mode). This would deplete the battery in a matter of a few hours. Therefore, the metrics proposed in [50] are not practical for mobile security.

Table 15
Approximate measures of the datasets' temporal resolution.

		SherLock dataset	Device analyzer dataset
Probe frequency	<i>Application statistics</i>	Every 5 s (0.2 Hz)	Every 5 min (0.003 Hz)
	<i>Motion sensors</i>	Every 15 s (0.067 Hz)	
Motion coverage	<i>Probe duration</i>	4 s	1 s
	<i>Relative coverage</i>	26.666%	0.003%

6.2. Related datasets

The Sherlock collection experiment dataset is not the first long-term smartphone sensor dataset collected. In 2005, the MIT Human Dynamics Labs published a reality mining dataset consisting of 100 Nokia 6600 smartphones [51]. The objective of the experiment was to provide social and contextual information on the external actor. The data was collected once every six minutes and did not contain any motion data or application statistics (such as memory or CPU utilization).

In 2013, the University of Cambridge published Device Analyzer; a collection agent for Android phones opens to the public as a free download from the Android marketplace [52]. Device Analyzer collects an extensive amount of information from the host device and uploads it to a central server for research purposes. There are several reasons why we did not use the Device Analyzer dataset for testing pcStream as a smartphone security solution:

- (1) **Specific features:** Device Analyzer's application statistics do not provide the LINUX process state (Sleep, Zombie etc.), number of active threads, and other minor details which are useful for observing an application or service's behavior. Moreover, it has been shown that features based on a signal's frequencies (e.g. using a Fourier transform) improve the task of activity recognition [53] and are therefore useful in the task continuous authentication. This data along with other statistical information (such as the covariance between axes) is not provided in the device Analyzer dataset.
- (2) **Temporal resolution:** Device Analyzer's application statistics are probed *approximately* once every five minutes. This time interval is too long to catch malicious behavior which can happen in a matter of seconds or less. Furthermore, information can be totally lost since a user can open and close an application within that window of time.

Moreover, Device Analyzer's motion sensors are also probed approximately once every 5 min, but for a duration (window) of 1 s (a relative 0.003% coverage). In the application of continuous authentication, an attacker can do a significant amount of damage (in terms of data theft) within this time window, and therefore this dataset is not suitable for our evaluations. Table 15 provides a summarized comparison between the temporal resolutions of the Sherlock dataset (used for this paper) and the Device Analyzer dataset.

7. Conclusion

In mobile security, there are situations where it is not possible to explicitly determine the legitimacy of various actors (internal and external). However, in many of these situations, implicit information in the form of data streams can be collected and used for detecting anomalies. Therefore, in this paper we proposed an extension to the pcStream algorithm, enabling it to detecting point, contextual, and collective anomalies in contextual data streams. To evaluate the algorithm's capability in providing smartphone security, we evaluated these extensions by detecting data leakage, malwares, and device thefts using an eight month dataset collected from 31 volunteers. Although we only evaluated one example security threat for each type of anomaly, the pcStream extensions are general, and can be applied to any case where a contextual data stream captures the behavior of an actor under question.

In the evaluations, the sources sampled in order to generate the data streams are accessible by any application without special privileges (verified on Android 6.0). This demonstrates that mobile security threats can be detected implicitly from data streams, without explicit information. Moreover, this fact makes the examples evaluated in this paper deployable by third parties via application markets. The results showed that pcStream can implicitly detect the three types of anomalies generated by attacks on the smartphones. In the future, we plan on investigating the effects of long-term concept drifts on the quality of the results. We also plan on improving the efficiency of the pcStream algorithm by incorporating iPCA. Lastly, we plan on investigating the use of Kernel PCA which can capture non-linear relations with a cost of an additional computational burden.

Acknowledgment

This research had been funded by the Israeli Ministry of Science, Space and Technology.

References

- [1] eMarketer, 2 billion consumers worldwide to get smart(phones) by 2016, 2015. <http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694> (accessed: 26.10.15).
- [2] A.P. Felt, M. Finifter, E. Chin, S. Hanna, D. Wagner, A survey of mobile malware in the wild, in: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM'11*, ACM, New York, NY, USA, 2011, pp. 3–14. URL <http://doi.acm.org/10.1145/2046614.2046618>.
- [3] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, M. Conti, M. Rajarajan, Android security: a survey of issues, malware penetration, and defenses, *IEEE Commun. Surv. Tutor.* 17 (2) (2015) 998–1022.
- [4] A. Zimmermann, A. Lorenz, R. Oppermann, An operational definition of context, in: *Modeling and Using Context*, Springer, 2007, pp. 558–571.
- [5] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C.A. Gunter, K. Nahrstedt, Identity, location, disease and more: Inferring your secrets from android public resources, in: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM*, 2013, pp. 1017–1028.
- [6] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, Mobile malware detection through analysis of deviations in application network behavior, *Comput. Secur.* 43 (2014) 1–18.
- [7] Android developers glossary, <https://developer.android.com/guide/appendix/glossary.html> (accessed: 26.10.15).
- [8] A. Shabtai, Y. Fiedel, Y. Elovici, Securing android-powered mobile devices using selinux, *IEEE Secur. Privacy* (3) (2009) 36–44.
- [9] M.B. Harries, C. Sammut, K. Horn, Extracting hidden context, *Mach. Learn.* 32 (2) (1998) 101–126.
- [10] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, *Mach. Learn.* 23 (1) (1996) 69–101. URL <http://dx.doi.org/10.1007/BF00116900>.
- [11] G. Widmer, Tracking context changes through meta-learning, *Mach. Learn.* 27 (3) (1997) 259–286. URL <http://dx.doi.org/10.1023/A:1007365809034>.
- [12] J.a.B. Gomes, E. Menasalvas, P.A.C. Sousa, Calds: Context-aware learning from data streams, in: *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques, StreamKDD'10*, ACM, New York, NY, USA, 2010, pp. 16–24. URL <http://doi.acm.org/10.1145/1833280.1833283>.
- [13] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, *ACM Comput. Surv. (CSUR)* 41 (3) (2009) 15.
- [14] Z. He, X. Xu, J.Z. Huang, S. Deng, Mining class outliers: concepts, algorithms and applications in crm, *Expert Syst. Appl.* 27 (4) (2004) 681–697.
- [15] A. Tsybal, The problem of concept drift: definitions and related work, Computer Science Department, Trinity College Dublin 106.
- [16] C.C. Aggarwal, A survey of stream clustering algorithms, 2013.
- [17] J.A. Silva, E.R. Faria, R.C. Barros, E.R. Hruschka, A.C.P.L.F.d. Carvalho, J.a. Gama, Data stream clustering: A survey, *ACM Comput. Surv.* 46 (1) (2013) 13:1–13:31. URL <http://doi.acm.org/10.1145/2522968.2522981>.
- [18] Y. Mirsky, B. Shapira, L. Rokach, Y. Elovici, pstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts, in: *Advances in Knowledge Discovery and Data Mining*, Springer, 2015, pp. 119–133.
- [19] A. Padovitz, S.W. Loke, A. Zaslavsky, Towards a theory of context spaces, in: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, 2004, pp. 38–42. <http://dx.doi.org/10.1109/PERCOMW.2004.1276902>.
- [20] J. Shlens, A tutorial on principal component analysis, arXiv Preprint [arXiv:1404.1100](http://arxiv.org/abs/1404.1100).
- [21] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan, Maintaining variance and k-medians over data stream windows, in: *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'03*, ACM, New York, NY, USA, 2003, pp. 234–243. URL <http://doi.acm.org/10.1145/773153.773176>.
- [22] S. Wold, M. Sjostrom, Simca: a method for analyzing chemical data in terms of similarity and analogy, 1977.
- [23] X. Song, M. Wu, C. Jermaine, S. Ranka, Conditional anomaly detection, *IEEE Trans. Knowl. Data Eng.* 19 (5) (2007) 631–645.
- [24] Pausing and resuming an activity, <http://developer.android.com/training/basics/activity-lifecycle/pausing.html> (accessed: 26.10.15).
- [25] M. Dunham, Y. Meng, J. Huang, Extensible Markov model, in: *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, 2004, pp. 371–374. <http://dx.doi.org/10.1109/ICDM.2004.10067>.
- [26] M. Hahsler, M. Bolaños, Clustering data streams based on shared density between micro-clusters, *IEEE Trans. Knowl. Data Eng.* 28 (6) (2016) 1449–1461.
- [27] L. Tu, Y. Chen, Stream data clustering based on grid density and attraction, *ACM Trans. Knowl. Discov. Data (TKDD)* 3 (3) (2009) 12.
- [28] Android blackmart alpha, <http://www.blackmart.us/> (accessed: 26.10.15).
- [29] Apk black market, <http://www.apkblackmarket.com/> (accessed: 26.10.15).
- [30] Apk download, <http://android.downloadatoz.com/> (accessed: 26.10.15).
- [31] M.O. Derawi, Smartphones and biometrics: Gait and activity recognition.
- [32] C.C. Ho, C. Eswaran, K.-W. Ng, J.-Y. Leow, An unobtrusive android person verification using accelerometer based gait, in: *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia, ACM*, 2012, pp. 271–274.
- [33] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S.-M. Makela, H. Ailisto, Identifying users of portable devices from gait pattern with accelerometers, in: *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP'05). IEEE International Conference on*, Vol. 2, IEEE, 2005, pp. ii/973–ii/976.
- [34] M. Frank, R. Biedert, E.-D. Ma, I. Martinovic, D. Song, Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication, *IEEE Trans. Inf. Forensics Secur.* 8 (1) (2013) 136–148.
- [35] H. Gascon, S. Uellenbeck, C. Wolf, K. Rieck, Continuous authentication on mobile devices by analysis of typing motion behavior, in: *Sicherheit*, 2014, pp. 1–12.
- [36] C. Bo, L. Zhang, X.-Y. Li, Q. Huang, Y. Wang, Silentsense: silent user identification via touch and movement behavioral biometrics, in: *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, ACM*, 2013, pp. 187–190.
- [37] F. Zhang, A. Kondoro, S. Muftic, Location-based authentication and authorization using smart phones, in: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2012, pp. 1285–1292.
- [38] B. Dixon, S. Mishra, J. Pepin, Time and location power based malicious code detection techniques for smartphones, in: *2014 IEEE 13th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2014, pp. 261–268.
- [39] M. Bastian, S. Heymann, M. Jacomy, Gephi: An open source software for exploring and manipulating networks, 2009. <http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154>.
- [40] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, *J. Stat. Mech. Theory Exp.* 2008 (10) (2008) P10008.
- [41] R. Kohavi, G.H. John, Automatic parameter selection by minimizing estimated error, in: *ICML*, 1995, pp. 304–312.
- [42] E. Ridge, D. Kudenko, Tuning an algorithm using design of experiments, in: *Experimental Methods for the Analysis of Optimization Algorithms*, Springer, 2010, pp. 265–286.
- [43] J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, in: *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.
- [44] L. Huang, A.D. Joseph, B. Nelson, B.I. Rubinstein, J. Tygar, Adversarial machine learning, in: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ACM, 2011, pp. 43–58.
- [45] M. Conti, B. Crispo, E. Fernandes, Y. Zhauniarovich, Crêpe: A system for enforcing fine-grained context-related policies on android, *IEEE Trans. Inf. Forensics Secur.* 7 (5) (2012) 1426–1438.
- [46] E. Hayashi, S. Das, S. Amini, J. Hong, I. Oakley, Casa: context-aware scalable authentication, in: *Proceedings of the Ninth Symposium on Usable Privacy and Security*, ACM, 2013, p. 3.
- [47] J. Zhu, P. Wu, X. Wang, J. Zhang, Sensec: Mobile security through passive sensing, in: *2013 International Conference on Computing, Networking and Communications, (ICNC)*, IEEE, 2013, pp. 1128–1133.

- [48] A. Shabtai, U. Kanonov, Y. Elovici, Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method, *J. Syst. Softw.* 83 (8) (2010) 1524–1537.
- [49] I. Bente, B. Hellmann, J. Vieweg, J. Von Helden, G. Dreo, Tcads: Trustworthy, context-related anomaly detection for smartphones, in: 2012 15th International Conference on Network-Based Information Systems, (NBiS), IEEE, 2012, pp. 247–254.
- [50] Y. Mirsky, A. Cohen, R. Stern, A. Felner, L. Rokack, Y. Elovici, B. Shapira, Search problems in the domain of multiplication: Case study on anomaly detection using markov chains, in: Eighth Annual Symposium on Combinatorial Search, 2015.
- [51] N. Eagle, A. Pentland, Reality mining: sensing complex social systems, *Pers. Ubiquitous Comput.* 10 (4) (2006) 255–268.
- [52] D.T. Wagner, A. Rice, A.R. Beresford, Device analyzer: Large-scale mobile data collection, *SIGMETRICS Perform. Eval. Rev.* 41 (4) (2014) 53–56. URL <http://doi.acm.org/10.1145/2627534.2627553>.
- [53] T. Huynh, B. Schiele, Analyzing features for activity recognition, in: Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies, ACM, 2005, pp. 159–163.