The Journal of Systems and Software 83 (2010) 1524–1537

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method

Asaf Shabtai *, Uri Kanonov, Yuval Elovici

Deutsche Telekom Laboratories at Ben-Gurion University, Department of Information Systems Engineering, Ben-Gurion University, Be'er Sheva 84105, Israel

ARTICLE INFO

Article history:
Received 11 September 2009
Received in revised form 22 February 2010
Accepted 18 March 2010
Available online 27 March 2010

Keywords:
Intrusion detection
Mobile devices
Temporal reasoning
Knowledge-based systems
Malware
Android

ABSTRACT

In this paper, a new approach for detecting previously unencountered malware targeting mobile device is proposed. In the proposed approach, time-stamped security data is continuously monitored within the target mobile device (i.e., smartphones, PDAs) and then processed by the knowledge-based temporal abstraction (KBTA) methodology. Using KBTA, continuously measured data (e.g., the number of sent SMSs) and events (e.g., software installation) are integrated with a mobile device security domain knowledge-base (i.e., an ontology for abstracting meaningful patterns from raw, time-oriented security data), to create higher level, time-oriented concepts and patterns, also known as temporal abstractions. Automatically-generated temporal abstractions are then monitored to detect suspicious temporal patterns and to issue an alert. These patterns are compatible with a set of predefined classes of malware as defined by a security expert (or the owner) employing a set of time and value constraints. The goal is to identify malicious behavior that other defensive technologies (e.g., antivirus or firewall) failed to detect. Since the abstraction derivation process is complex, the KBTA method was adapted for mobile devices that are limited in resources (i.e., CPU, memory, battery). To evaluate the proposed modified KBTA method a lightweight host-based intrusion detection system (HIDS), combined with central management capabilities for Android-based mobile phones, was developed. Evaluation results demonstrated the effectiveness of the new approach in detecting malicious applications on mobile devices (detection rate above 94% in most scenarios) and the feasibility of running such a system on mobile devices (CPU consumption was 3% on average).

1. Introduction

Personal digital assistants (PDAs), mobile phones and, most recently, smartphones have evolved from simple mobile phones into sophisticated yet compact minicomputers that can connect to a wide spectrum of networks, including the Internet and corporate intranets. In addition to enabling users to access and browse the Internet, these devices can receive and send emails, SMSs, and MMSs, connect to other devices for exchanging information/synchronizing and activating various applications. Designed as open, programmable, networked devices, smartphones are no longer immune to malicious attacks and are susceptible to various new threats such as viruses, Trojan horses, and worms, all of which are well-known from the desktop computer platforms. An infected smartphone can inflict severe damages to both the user and the cellular service provider. Malware can partially or fully disable a smartphone; cause unwanted billing; steal private information (possibly by phishing and social engineering methods); or infect every name in a user's phonebook (Piercy, 2004). Attack vectors of malware propagating into smartphones include cellular networks, Bluetooth, Internet (via Wi-Fi, GPRS/EDGE or 3G network access), USB, and other peripherals (Cheng et al., 2007).

The challenges for smartphone security are becoming very similar to those that personal computers encounter (Muthukuma et al., 2008) and common desktop-security solutions are downsized to mobile devices. As a case in point, Botha et al. (2009) analyzed common desktop-security solutions and evaluated their applicability to mobile devices. However, some of the desktop solutions (i.e., antivirus software) are inadequate for use on smartphones since they consume too much CPU and memory and might rapidly drain the power source. In addition, since most antivirus software detection capabilities depend on the existence of an updated malware signature repository, antivirus users are not protected whenever an attacker spreads a previously never encountered malware. Since it may take antivirus vendors several hours to several days to identify the new malware, generate a signature, and to update their clients' signature database, hackers have a substantial window of opportunity (Dikinson, 2005). Also

---

* Corresponding author.

E-mail addresses: shabtaia@bgu.ac.il (A. Shabtai), kanonov@bgu.ac.il (U. Kanonov), elovici@bgu.ac.il (Y. Elovici).

0164-1212/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.03.046

some malware, targeted to a specific and small number of mobile devices (e.g., to extract confidential information or to track the owner's location), may take time before they are noticed.

In light of smartphone vulnerability and the variety of smart mobile devices and operating systems, a more generic approach is required to rapidly detect malware that has never been encountered (presumably belonging to some known malware class) even before its signature is announced and while it is being installed or activated on a device.

In this research, an innovative host-based intrusion detection system (HIDS) for detecting malware on mobile devices is suggested. Detection is accomplished by continuously monitoring a mobile device in order to identify malicious temporal behavior. The framework relies on a lightweight agent (in terms of CPU, memory and battery consumption) that continuously samples various features on a device, analyzes collected data and infers the state of the device. The basic premise on which the framework rests is that behavioral patterns over time, of a potential, previously unencountered malware, presumably belonging to some known malware class, are an important factor in facilitating accurate detection. Thus, inspecting behavioral changes in monitored devices or network traffic over time can help in detecting malware even when its signature is unknown. This can be achieved by providing a high level definition of a malware class to detect malware instances, some of which are unknown, with a relatively small number of temporal patterns as defined by a security expert. The pattern definitions represent malware class behavior and not signatures of specific malware instances. For example, it would be desirable that the system alert the user to any application causing a combination of high CPU usage, high system context switch state and an increasing garbage collection trend which might indicate a denial-of-service (DoS).

Detection focuses on the behavioral patterns that the user or security expert describes. Typical examples might include high network activity after SD-card access or too many SMS messages sent to addresses that are not in the contact list in a predefined period of time.

Inspecting the temporal behavior of a specific mobile device in order to determine its status also requires context-sensitive interpretation of accumulated data. This high level, context-sensitive, knowledge-based abstraction of time-oriented data is referred to as a temporal abstraction of the data (Shahar, 1997).

For detecting malware patterns the knowledge-based temporal abstraction (KBTA) method is employed for representing the mobile device's security domain knowledge and deriving temporal abstractions. The KBTA method has been used successfully to support many tasks, especially diagnostic or therapeutic decision-support (Shahar and Musen, 1996) in the medical domain. In the security domain, a network, computer or any other device can be regarded as a patient and by inspecting temporal data related to a device or a network, the security expert can infer whether it is infected or not. The KBTA method was proposed and evaluated in Shabtai et al. (2009) for detecting malware on personal computers or within network traffic. However, since mobile devices are limited in regard to the CPU and battery resources, some modification and customization of the KBTA method was required. It was done by reducing its functionalities and adapting its ontology for incremental computation.

The proposed method was evaluated on Android-based mobile devices. Android,[1] Google's new framework for mobile devices, is among the most significant smartphone operating systems today. Android was chosen due to the ease it displays in implementing and deploying applications and its openness in extracting many useful raw parameters and events from its framework.

The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 describes the knowledge-based temporal abstraction method. Next, Section 4 introduces the proposed intrusion detection system for the Android architecture and its components. In Section 5, the KBTA implementation on Android and the applied modifications are described in order to adapt the method to run on mobile devices. In Section 6 the results from evaluating the proposed approach is presented while Section 7 discusses the advantages of the described architecture, and potential future research.

2. Related work

The overview of related academic literature indicates that most extant research on protecting mobile communication devices has focused on applying and evaluating host-based intrusion detection systems (HIDS). These systems, using anomaly- or rule-based methods, extract and analyze (either locally or by a remote server) a set of features indicating the state of the device. Several systems are reviewed in this section and summarized in Table 1.

Moreau et al. (1997) utilized artificial neural networks (ANNs) to detect anomalous behavior indicating a fraudulent use of the operator services (e.g., registration with a false identity and using the phone to high tariff destinations). The detection was based on 16 features representing mean and standard deviation of the total duration and number of long- and short-term national and international calls.

The intrusion detection architecture for mobile networks (IDAMN) system (Samfat and Molva, 1997) uses both rule-based and anomaly detection methods. IDAMN offers three levels of detection: location-based detection (a user is active in two different locations at the same time); traffic anomaly detection (an area having normally low network activity experiencing high network activity); and detecting anomalous behavior of individual mobile phone users. In order to detect anomalous behavior, a profile is generated by monitoring the user's telephone activity (e.g., call duration, inactivity time between calls, number of handovers performed). In addition, the user's location in the network (roaming) is monitored by generating a state machine with the probability of moving from one location (cell) to another. Preliminary evaluation shows that IDAMN can raise an alarm in less than one second from the time the intrusion occurs and that it has an intrusion detection rate higher than 70% and a false alarm rate lower than 5%.

Yap and Ewe (2005) employ a rule-based behavior checker solution that can detect malicious activities in the system. The authors present a proof-of-concept scenario using a Nokia mobile phone running a Symbian OS. In the demonstration, a behavioral detector detects a simulated Trojan horse attempting to use the message server component, without authorization, to create an SMS message.

Cheng et al. (2007) present *SmartSiren*, a collaborative, proxy-based virus detection system for smartphones. Single-device and system-wide abnormal behaviors are detected by joint analysis of communication activity of monitored smartphones. The SmartSiren architecture consists of a back-end proxy that interacts with lightweight agents on the protected devices. The agents merely collect information and relay it to the proxy which performs the analysis and sends out the alerts. SmartSiren was evaluated by simulating a virus outbreak based on a three-week SMS trace collected from a national cellular service provider in India. The results show that without SmartSiren, virus-generated messages accounted for 68% of the messages that were sent. With SmartSiren in place, the total number of message sent was reduced to about 28%.

---

[1] http://www.android.com/.

Table 1
Academic research on protection of mobile devices.

Paper	Approach	Detection method	Detects
Moreau et al. (1997)	HIDS	Anomaly detection using ANN	Fraudulent use of the operator services such as high rate calls
Samfat and Molva (1997) (IDAMN)	HIDS, NIDS	Anomaly detection; Rule-based detection	A user is active in two different locations at the same time; traffic anomaly detection; and detecting anomalous behavior of individual mobile phone users based on the telephone activity (such as call duration) and user's location in the network
Yap and Ewe (2005)	HIDS	Signature-based detection	Proof of concept- detects unauthorized attempt to create SMS message
Cheng et al. (2007) (SmartSiren)	HIDS, NIDS	Anomaly detection	Detects anomaly behavior of the device and outbreak of worm-related malware
Schmidt et al. (2009)	HIDS	Anomaly detection	Monitor a smartphone running Symbian operating system and Windows Mobile in order to extract features for anomaly detection. These features are sent to a remote server for further analysis
Bose et al. (2008)	HIDS	Signature-based detection.	Using temporal logic to detect malicious activity over time that matches a set of signatures represented as a sequence of events
Kin et al. (2008)	HIDS	Signature-based detection	Detects, and analyzes previously unknown energy-depletion threats based on a collection of power signatures
Buennemeyer et al. (2008) (B-SIPS)	HIDS, NIDS	Anomaly detection	Detects abnormal current changes and its correlation with network attack
Nash et al. (2005)	HIDS	Statistical method (linear regression)	Detects processes that are likely to be battery-draining attacks
Jacoby and Davis (2004) (B-BID)	HIDS	Signature-based detection	Monitoring power consumption against "normal" power consumption range. Once an anomaly is detected, various system data is matched against known attack signatures
Miettinen et al. (2006)	HIDS, NIDS	Event correlation	Combines both host-based and network-based data collection in order to be able to utilize the strengths of both detection approaches
Hwang et al. (2009)	Authentication	Keystrokes dynamics	Collects 5 features to train and build a classifier capable of detecting impostors. Utilized artificial rhythms and tempo cues to overcome problems resulting from short PIN length

Schmidt et al. (2009) monitored a smartphone running a Symbian OS by extracting features that describe the state of the device and which can be used for anomaly detection. These features were collected by a Symbian monitoring client collected and forwarded to a remote anomaly detection server (RADS). The gathered data was analyzed in order to distinguish between normal and abnormal behavior. The results indicated that most of the top ten applications preferred by mobile phone users affected the monitored features in different ways.

Special effort has been devoted to intrusion detection systems (IDS) that analyze generic battery power consumption patterns to block distributed denial-of-service (DDoS) attacks or to detect malicious activity via power depletion (Racic et al., 2006; Martin et al., 2004). Kim et al. (2008) presents a power-aware, malware-detection framework that monitors, detects, and analyzes previously unknown energy-depletion threats. The framework collects power consumption samples and generates power signatures. These signatures are used for detecting mobile malware by measuring the similarity between power signatures using the χ^2-distance measure. Experimental results on an HP iPAQ device running a Windows Mobile OS, demonstrated a 99% true-positive rate in classifying mobile malware. The battery-sensing intrusion protection system (B-SIPS) Buennemeyer et al., 2008 for mobile computers alerts when abnormal current changes are detected. The B-SIPS correlates host-based anomaly detection with the Snort IDS which provides signature-based detection of attack.

Nash et al. (2005) presented a design for an intrusion detection system that estimated power consumption according to a linear regression model, based on parameters such as CPU load and disk accesses, to determine the amount of energy used on a per process basis and to identify processes that could potentially exhaust batteries.

Jacoby and Davis (2004) presented a host battery-based intrusion detection system (B-BID). The underlying assumption is that monitoring the device's electrical current and evaluating the correlation with known signatures and patterns can facilitate attack detection. The electrical current is measured and once an anomaly is detected, various data (e.g., network activity, CPU usage, process count) is collected and matched against known attack signatures. This data can also be forwarded to the network administrator for further analysis.

Miettinen et al. (2006) claim that host-based approaches are required, since network-based monitoring alone is not sufficient to counter the future threats. They adopt a hybrid network/host-based approach. A correlation engine on the back-end server filters the received alarms according to correlation rules in its knowledge-base and forwards the correlation results to a security monitoring GUI to be analyzed by security administrators. Hwang et al. (2009) evaluated the effectiveness of keystroke dynamics-based authentication (KDA) on mobile devices. Their empirical evaluation focused on short PIN numbers (four digits). The proposed method yielded a 4% misclassification rate.

The above solutions disregard temporal features as first-class-citizens when analyzing data captured from networks or devices. They do not consider the behavioral changes along time but only analyze snapshots of data representing the system's state.

Some studies have focused on temporal features to represent the normal temporal behavior of the user, the system or the network. Bose et al. (2008) propose an interesting behavioral detection framework, based on temporal logic, for detecting mobile worms and trojan horse activity over time. A malware behavior is represented by describing the temporal ordering of an application's actions which may reveal malicious intent even when each action alone may appear harmless. A database of malicious behavior signatures was generated by studying more than 25 distinct families of Symbian OS malware. Next, a two-stage mapping technique constructs these signatures in run-time from the monitored system events and API calls in the Symbian OS. The system differentiates the malicious behavior of malware from the behavior of benign applications by training a classifier based on support vector machines (SVM). The evaluation of both simulated and real-world malware samples indicated that behavioral detection can identify current mobile viruses and worms with more than 96% accuracy. The temporal logic approach was used in Talbi et al. (2008) for detecting polymorphic malicious codes that exploit buffer-overflow vulnerabilities.

Other studies have explored the detection of malicious or abnormal temporal behavior (Lane and Brodley, 1999) on personal computers or servers. In (Seleznyo and Mazhelis, 2002), the user's normal temporal behavior is extracted as a temporal-probabilistic tree in which the nodes correspond to actions (such as reading an e-mail) and the edges correspond to the transition from one action to the next one. Temporal features such as action duration and temporal relations between actions are stored in the temporal-probabilistic tree nodes and edges. Each branch of the tree represents one or more possible behavioral patterns of a user.

Ghosh et al. (1999) utilized anomaly detection to represent a program's normal behavior as a frequency table-storing sequence of system calls. Naldurg et al. (2004), Ning et al. (2001), and Kohout et al. (2002) describe attack patterns or normal behavior patterns based on Allen's temporal relations (Allen, 1983) such as "before", "after" and "meet" between events (e.g., a login by user *u* should be followed by a logout). Ye (2000) employed a Markov chain to model normal temporal behavior of a computer as a sequence of computer-related actions. Observed behavior is updated with a probability that supports the normal behavior. A low probability indicates a high likelihood of abnormal behavior. Li et al. (2002) employ association-rules with time granularity to define a normal behavior pattern within different temporal intervals (e.g., network traffic during weekend versus weekdays).

Morin and Debar (2003) used the chronicle formalism to reduce the number of alerts raised by multiple sensors (i.e., intrusion detection systems and logging tools) such as Snort and Syslog and to improve the quality of the alerts by reducing false alarms. Chronicle formalism, which is based on reified temporal logic, was applied to define temporal patterns as a set of events and time constraints. Several scenarios are provided in order to exemplify the benefits of the method. These temporal-based studies perceive anomalous or malicious behavior by identifying sequences of events. However, as will be described below, the KBTA method demonstrated in this paper also considers the duration as well as the values of events and parameter. This makes it possible to detect changes over time (trends). Furthermore, since the KBTA method assumes that the interpretation of various events and parameters is dependent on the current state of the system (i.e., context), the derivation of the temporal behavior is context-based, resulting in a more accurate depiction of the reality. In addition, the aforementioned solutions usually focus on a specific task or domain (e.g., analyzing and detecting attacks based on phone call records or battery consumption) rather than providing an overall solution that supports detection of malware using different types of data sources.

Fuzzy cognitive map (FCMs) (Aguilar, 2005; Stach et al., 2005) were also incorporated into intrusion detection systems. In (Siraj et al., 2004), FCMs are used for defining alerts based on concepts (i.e., information provided by various sensors) and causal relationships of these concepts and the alerts. FCMs differ from the KBTA method in several points. First, in the semantic links between concepts, FCMs support only a causal link between two concepts which indicates the degree of causality (i.e., the effect of the cause) of one concept from another. The KBTA method supports other semantic relationships between concepts, including the abstracted-from relation (indicating which concepts are used for the abstraction of a concept); the necessary-context relation (indicating which contexts are required for the abstraction of a concept); and the components relation (indicating the concepts that are required for pattern derivation). Second, time is not handled as an inherent factor in the definition of a FCM. Thus, it does not support the definition of the duration of a concept and the time interval between concepts. Third, in addition to pattern matching that is supported by both, the KBTA and FCM, the KBTA also supports the abstraction of high level concepts from raw data (such as state, rate and trend). These high level concepts can later be used to define temporal patterns.

3. The knowledge-based temporal abstraction method

Temporal abstraction (TA) is a common and important temporal-reasoning (TR) task. TA incorporates a computational mechanism that integrates raw, time-stamped data and knowledge to extract and summarize meaningful interpretations of the raw, time-stamped data. The knowledge-based temporal abstraction (KBTA) method (Shahar and Musen, 1996) is a computational framework for supporting the TA task. KBTA was proposed for an automated derivation of context-specific interpretations and conclusions (i.e., temporal abstractions), from the raw time-oriented data, by using a domain-specific knowledge-base (e.g., a security ontology specialized for abstracting meaningful patterns from time-oriented security data).

In general, the KBTA method is defined as follows. The input includes a set of time-stamped parameters (e.g., CPU usage) and events (e.g., keyboard activity or touch screen activity) that create the necessary interpretation context (e.g., "no user activity" when the touch screen and keyboard are not used). The output includes a set of interval-based, context-specific parameters at the same or at a higher level of abstraction and their respective values (e.g., a period of 5 min of a high CPU usage while no user activity was detected).

A knowledge engineer defines the domain knowledge (e.g., security ontology) by using five KBTA entities and the relations between them. Five inference mechanisms (temporal context formation, contemporaneous abstraction, temporal inference, temporal interpolation and temporal pattern matching) are then applied in parallel to derive the high level abstractions from the raw data (Shahar, 1997). Fig. 1 presents a description of the KBTA method. Time-stamped measurements of primitive parameters and time-stamped events, as well as the predefined KBTA ontology are the input to KBTA's five inference mechanisms. The outputs of the five inference mechanisms are time-intervals of contexts, abstractions and patterns that can be automatically monitored and stored for later inspection and exploration.

The KBTA ontology comprises five ontological entities that define the domain ontology which is related to a subject in the specific domain (e.g., smartphone). The five ontological entities are: primitive parameters, abstract parameters, contexts, events, and patterns.

Primitive parameters are raw measurable data collected from different sensors. In general, primitive parameters may be either numeric (e.g., CPU usage level, number of transmitted packets via Wi-Fi) or symbolic (e.g., the name of the operating system).

Events are raw data and represent actions caused by the system or the user, such as starting an application, touching the screen or installing an application. Events can be described by numeric or symbolic attributes. For example, potential attributes for an application start-up are the name of the application (symbolic) and the process-id running it (numeric).

Contexts are the "state of affairs" of a monitored subject (e.g., a mobile device). Contexts are induced dynamically, sometimes by the existence of a primitive or abstract parameter with a certain value, most commonly, however, by the existence of an event. Since the contexts affect the interpretation of parameters, the same dataset may be interpreted differently within different contexts. For example, high CPU usage on a mobile device may be interpreted as normal within the "user activity" context and abnormal within the "no user activity" context (which might indicate the existence of a resource draining malware).

Abstract parameters are derived from one or more parameters (primitive or abstract). Part of the knowledge inherent in the abstract parameter is a classification function that maps the values of the "abstracted-from" parameters to the values of the abstracted parameter. For example, "CPU Usage STATE" is an abstract param-

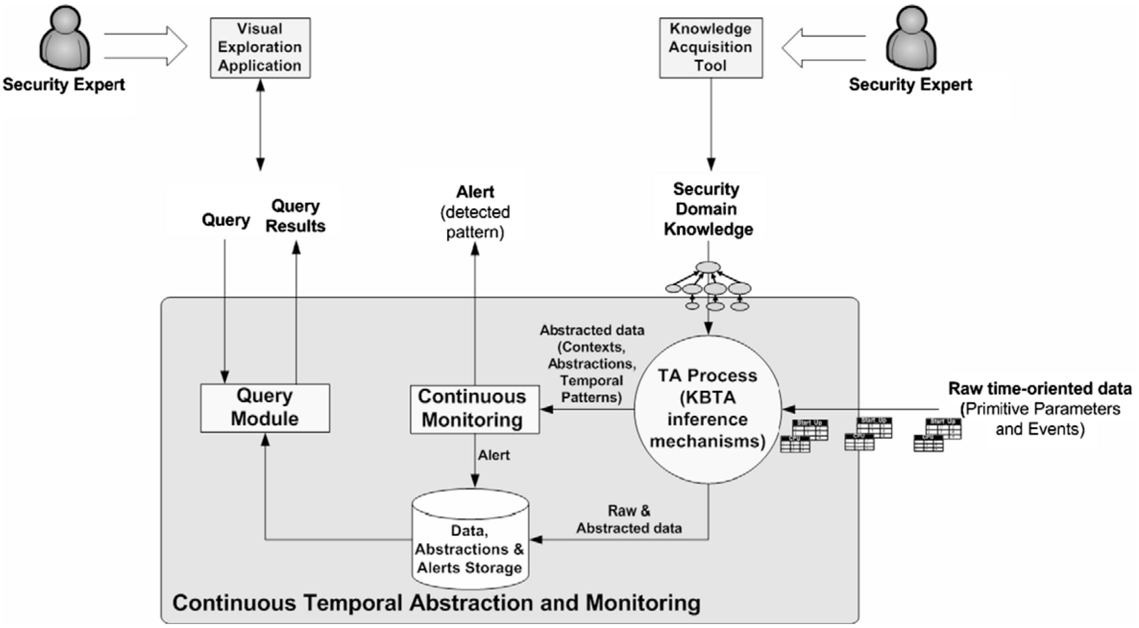
Security Expert
Visual Exploration Application
Knowledge Acquisition Tool
Security Expert
Query
Query Results
Alert (detected pattern)
Security Domain Knowledge
Abstracted data (Contexts, Abstractions, Temporal Patterns)
Query Module
Continuous Monitoring
TA Process (KBTA inference mechanisms)
Raw time-oriented data (Primitive Parameters and Events)
Alert
Raw & Abstracted data
Data, Abstractions & Alerts Storage
Continuous Temporal Abstraction and Monitoring

Fig. 1. The KBTA general framework.

eter that is abstracted-from the primitive parameter "CPU Usage" (the percentage of time the CPU wasn't idle). A classification function can map 0–10% CPU Usage to a LOW state.

A context is required in order to derive an abstract parameter; within different contexts, the abstract parameter will have different classification functions. Thus the same input may result in different output values. There are three types of abstract parameters: state, gradient (or trend), and rate. These correspond to three types of abstractions, respectively. State abstraction maps the values of the abstracted-from parameter values to a state-describing set of values (e.g., LOW CPU usage level). Gradient (or trend) abstraction determines the direction of the change of values in a measured parameter (e.g., INCREASING number of pictures taken by the camera). Rate abstraction classifies the amplitude of a rate of change of a selected parameter (e.g., FAST changing number of pictures taken by the camera).

The objective is to derive, for each abstraction, the longest possible time interval from the raw data with the same value. A persistence function, which is also part of the abstract parameter knowledge, determines the maximal gap between two time-intervals that enables their concatenation into a longer interval.

Patterns are a complex set of value and time constraints defined over a set of parameters (primitive and abstract), events, and contexts. There are two types of constraints: local and global (Chakravarty and Shahar, 2000). A local constraint is defined for one concept over one time interval, for example, HIGH CPU usage state for more than 10 min. A global constraint defines the pair-wise temporal relation between two intervals based on Allen's 13 temporal relations (Allen, 1983). A typical example is "HIGH number of pictures taken followed within 2 minutes by HIGH outgoing network traffic for at least 5 min".

In addition, there are two types of patterns: linear and repeating. A linear pattern occurs only once. A repeating pattern is a linear pattern that occurred two or more times (for example, the above pattern occurred four times in one week).

Fig. 2 shows an example of the temporal abstraction process. The figure illustrates the derivation of a SD-card information leakage pattern that may indicate that a trojan in the mobile device is

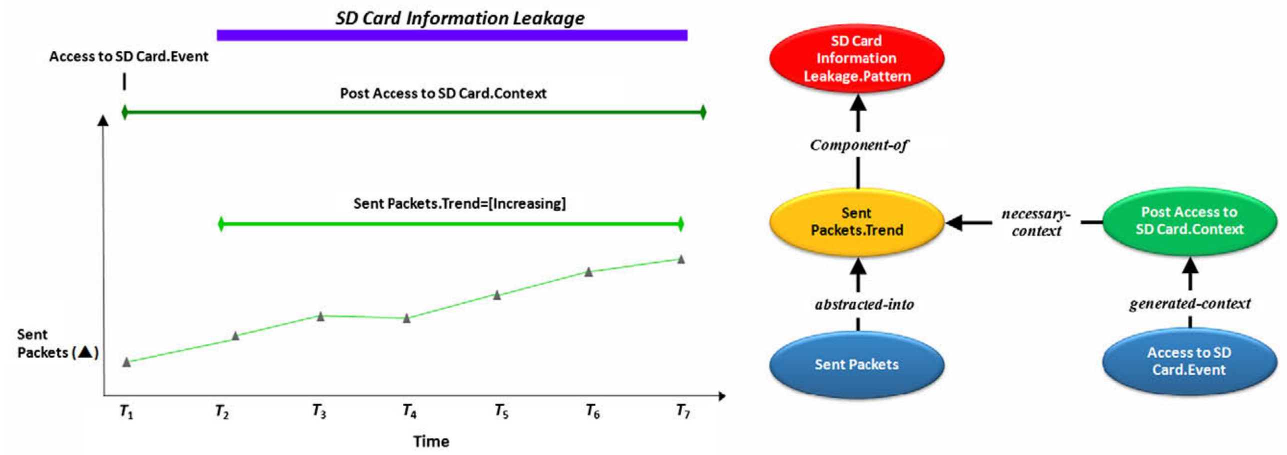

Fig. 2. An example of SD-card information leakage pattern. Raw data is plotted at the bottom. Events and the abstraction computed from the data are plotted as intervals above the data. | = an event; ▲ = sent packets via Wi-Fi; |- - -| = a context open interval; |—| = an abstraction (derived concept) interval.

transmitting private data stored on the SD-card to a remote server. The input to the KBTA inference mechanisms consists of raw measurements of "Sent packets" at T_1–T_7, and the "Access to SD-card" event, which generates the Post access to SD-card context. Then, within the Post access to SD-card context, an INCREASING sent packet trend is interpreted from the raw data of the "Sent packets". The SD-card information leakage pattern is derived from this trend.

Two methods are applicable for defining the domain knowledge and more specifically mobile phone security ontology (specializing in detecting malware). The first method involves the security expert using a dedicated knowledge acquisition tool to define new patterns based on new classes of malware that were identified by the security community. The second method involves applying temporal data mining process, as suggested in Moskovitch and Shahar (2009), on data collected from devices infected by a new class of malware. The temporal data mining process may reveal new temporal patterns that the KBTA method can employ in order to detect the new malware in run-time.

4. Intrusion detection systems for android-based devices

Google's Android, a comprehensive software framework targeted towards smart mobile devices, includes an operating system, a middleware and a set of key applications. Android has emerged as an open-source and open platform that provides mobile devices with APIs for most of their software and hardware components. Specifically, it allows third-party developers to develop their own applications. The applications are written in the Java programming language based on the APIs provided by the Android Software Development Kit (SDK). Although not that common, developers can also develop and modify kernel-based functionalities for smartphone platforms. To evaluate the proposed KBTA method, a lightweight host-based intrusion detection system (in terms of CPU, memory and battery consumption) for Android was developed.

The basis of the intrusion detection process consists of monitoring, collecting and optionally pre-processing various parameters and events. Parameters can be pre-processed by aggregating or normalizing according to time-intervals. Various detection units then analyze the features that have been sent to them following collection and pre-processing.

The primary detection unit, presented in this paper, is KBTA-based. However multiple processors can be attached to the agent, each employing its own method of detecting malicious behavior and outputting a threat assessment accordingly. Each time interval all of the pending threat assessments are weighted to produce a single consistent alert. The weighting process, which is per threat type, i.e., virus threat assessments are weighted separately from worm threat assessments, also includes a smoothing phase (combining the generated alert with the past history of alerts) in order to avoid instantaneous false alarms.

After the weighting phase, a proper notification is displayed to the user. Moreover, the alert is matched against a set of automatic or manual actions that can be taken to mitigate the threat. Among the automatic actions are uninstalling an application without any user interaction, killing of a process, disconnecting all radios, encrypting data, changing firewall policy and more. A manual action may involve uninstalling an application with user interaction.

The components of the agent can be clustered into four main groups (Fig. 3): feature extraction, main service, processors and the graphical user interface. The *Feature Extractors* communicate with various components of the Android framework, including the Linux kernel and the application framework layer in order to collect raw features. The *Feature Manager* triggers the feature extractors and the requests for features every predefined time interval. In addition, the feature manager may apply some pre-processing on the raw features that are collected by the feature extractors. A *Processor* is an analysis and detection unit. It is provided as a pluggable, separate component that can be seamlessly installed and uninstalled. It receives feature vectors from the *Agent Service*, analyzes them and outputs threat assessments to the *threat weighting unit* (*TWU*). Each processor may expose an advanced configuration screen. Processors may be rule-based, knowledge-based or classifiers or anomaly detectors based on machine-learning methods. The TWU receives the analysis results from all active processors and applies an ensemble algorithm (such as majority voting, distribution summation etc.) in order to derive a final and more accurate decision regarding the device infection level. The *Alert Manager* receives the final ranking as produced by the TWU. It then can apply some smoothing function in order to provide a more persistent alert and to avoid instantaneous false alarms. Examples of such functions may be moving average and leaky-bucket. The smoothed infection level is then compared with predefined minimum and maximum thresholds.

The *Agent Service* is the most important component in the intrusion detection system since it synchronizes the feature collection, detection and alert process. The *Agent Service* manages the detection flow by requesting new samples of features, sending the new information to the processors and receiving the final result from the *Alert Manager*. The *Loggers* provide logging options for

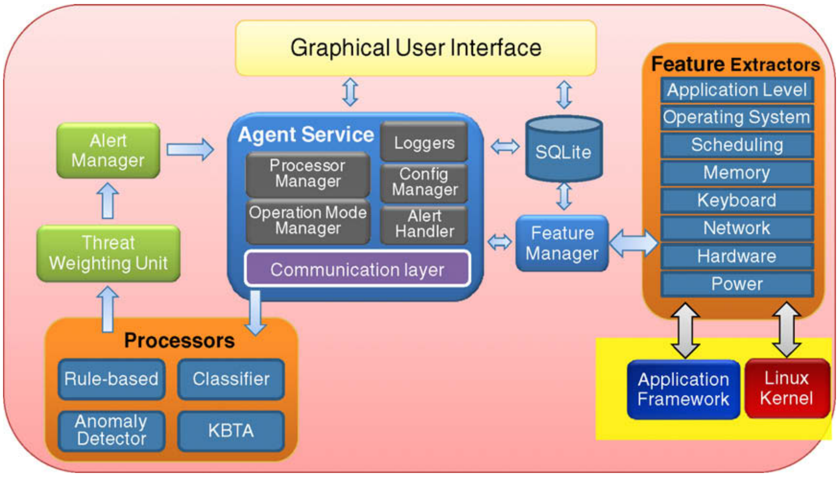

Fig. 3. The Android HIDS architecture.

debugging and for evaluating detection algorithms. The *Config Manager* manages the configuration of the agent (such as active processors, active feature extractors, alert threshold, active loggers, sampling time interval, operation modes configuration). The *Alert Handler* triggers an action as a result of a dispatched alert (e.g., visual alert in the notification bar; uninstalling an application; sending notification via SMS or email; locking the device and preventing further usage; disconnecting any communication channels). The *Processor Manager* registers/unregisters processors and activates/deactivates processors. The *Operation Mode Manager* changes the agent from one operation mode to another based on the configuration. This will activate/deactivate processors and feature extractors. The change from one operation mode to another (i.e., from full security mode to normal mode) is triggered as a result of changes in resource levels (battery, CPU, network).

The last component is the GUI which provides the user with the means to configure agent parameters, activate/deactivate (for experimental usage only) visual alerts and visual exploration of collected data. The agent also provides a communication layer which allows it to register to a remote management system and update that system with the device's current status and with additional information to allow a security officer to explore the data when an alert is raised.

In the next section, the modified incremental Knowledge-Based Temporal Abstraction process which was integrated as a processor in the Android HIDS is described.

5. The modified incremental KBTA process

Since the KBTA process is complex and computationally expensive, in this study it was modified and adapted for resource-limited mobile devices. As its basic input, the processor receives various raw features/data (i.e., name, value, time). The features are received in temporal order, each predefined time interval; this is the basis of the incremental computation process. The abstraction process is incremental in the sense that each time new data arrives the abstraction unit creates new abstractions or updates existing abstractions, and thus it is not necessary to recalculate abstractions derived from previous data. Such a form of computation provides rapid calculation and thus allows real-time monitoring of derived abstractions and patterns.

After receiving primitive parameters and events, the processor creates new contexts, ends existing contexts, creates or updates states and creates or updates trends. These actions are carried out in a loop which may last as long as any of the new or existing elements are being created or modified, respectively. This process is finite and it converges fairly quickly given a proper knowledge-base. For the remainder of this section, references to "new" elements includes both new and existing elements that have been modified (i.e., had their value or duration changed).

The pseudo-code in Fig. 4 describes the modified KBTA process. The computation loop starts upon receiving new primitives. The first step, context destruction, is fairly straightforward. While iterating each existing context, the mechanism checks if there are any new/modified elements that meet the destruction conditions of the context. If such an element exists, the destroying element trims the context. For example, an existing context, such as the USB connection, is destroyed by a USB disconnection event.

The creation of new contexts follows the destruction of the existing ones. In order to create a context, an element matching any of its inductions must be located. If such an element exists, a context is created (as specified in the knowledge-base). An assumption made here is that the process is not retrospective and contexts starting/ending in the past cannot be created. The basis for this assumption is that data comes sequentially to the KBTA processor due to the nature of the HIDS.

The next stage in the process is the creation of abstractions. The classification function of trend and state is as in the ordinary KBTA process. However the persistence function, which was noted in Section 3, could be simplified due to the fact that data arrives in constant time intervals. A primary manifestation of the incremental nature of the algorithm is that upon creation of abstractions by the classification function, the persistence function immediately concatenates these abstractions (if possible) into longer intervals.

Once the process of determining the existing contexts and abstractions stabilizes, the creation of patterns starts. The modified KBTA process only employs two of Allen's 13 temporal relations:

```
Modified Incremental Knowledge-Based Temporal Abstraction

Initialization:
    Let newPrimitives be a collection of new primitive parameters
    Let newEvents be a collection of new events
    Let ElementsCollection all instances that are within the time window t and are stored
        by the KBTA detection unit

Incremental_Abstraction(newPrimitives, newEvents)
    Add newPrimitives to ElementsCollection;
    Add newEvents to ElementsCollection;
    ModifiedElements ← true;
    While (ModifiedElements) do
        DeriveContexts();
        DeriveStateAbstractions();
        DeriveTrendAbstractions();
    DeriveTemporalPatterns();
```

Fig. 4. Pseudo-code describing the abstraction process.

before and overlapping. Combined with a set of properties on the gap between the elements, the length of the overlap and the distance of the overlap from the beginning of the first element, these two relations are able to imitate all of Allen's temporal relations. The patterns are also created incrementally, that is, in each iteration (receiving new data), partial patterns consisting of the elements matching the pattern's local and global constraints are saved. A complete pattern is obtained once the elements enabling the completion of a partial pattern are found. After the creation of linear patterns, the creation of repeating patterns ensues.

An important design decision taken in the modified Incremental KBTA process is that elements (both raw and abstracted data) are only kept in memory for a limited amount of time in order not to exhaust the memory of the mobile device; they are not stored in a local database. This timeframe is configurable and chosen to allow complex pattern computation; thus partial patterns and the elements comprising them may be kept as long as they do not violate the local and pair-wise constraints associated with the pattern.

Theorem 1. *The computational complexity of the modified and incremental KBTA-based detection unit is constant.*

Proof. Let,

k – the number of concepts in the ontology
t – the time window for removing existing elements

The context abstraction destroys and generates contexts. The first action involves checking for all the latest context instances in the element collection for possible destroying elements; the latter traverses all of the contexts defined in the ontology and seeks the generating elements. Since for either operation there are at the most k contexts and k possible destroying/generating elements, the context abstraction is bounded by $O(k^2)$.

The state and trend abstractions are performed by checking for each state/trend in the ontology and for the possibility of abstraction from among all the latest instances in the element collection. Only new elements can be abstracted and among the new elements there is at most one for each concept. Thus the state and trend abstractions are bounded by $O(k^2)$.

In each iteration of the whole loop, at least one new element is created (otherwise the whole loop will be terminated). Each element can only be created once during the invocation of "Incremental_Abstraction". Thus the whole loop will be executed at the most k times, which results in the upper bound of $O(k^3)$.

The last step, when no new elements are being created, is pattern derivation. In the time interval (window) t, each concept in the ontology has at most t instances. While each pattern has at most k elements, there are at most t^k partial patterns. Since there are at most k patterns in the ontology, the pattern derivation is bounded by $O(t^k \cdot k)$.

Thus the modified incremental abstraction process is bounded by $O(t^k \cdot k + k^3)$, where k, the number of concepts in the ontology, and t, the time window, are constants. This concludes the proof. □

Next comes the monitoring phase, which depends on the output of the KBTA process. The knowledge-base includes: a description of the high level elements (states, trends, linear and repeating patterns) that represent malicious behavior along with value and/or duration constraints on the elements; a base severity level (in %); and a recommended mitigation strategy for the agent (e.g., killing malicious process, removing application). Threat assessments are generated when suitable elements (fitting the constraints) are computed and the severity level of the generated threat assessment is calculated according to Eq. (1) where *ActualDuration* is the duration of the element that triggered the generation of the threat assessment; *MinimalDuration* is the minimal duration of the element needed to produce the threat assessment; and the *BaseSeverity* is the severity level attached to the monitored element with the minimal duration.

$$\frac{ActualDuration}{MinimalDuration} \times BaseSeverity \qquad (1)$$

6. Evaluation

This section presents the evaluation of the proposed KBTA-based detection framework. In Section 6.1 five Android malicious applications, which were developed and used for the evaluation, are presented. Section 6.2 describes the KBTA ontology for detecting malware on mobile devices that was defined for the evaluation. Section 6.3 presents the results of the experiments.

6.1. Description of malicious applications

There are several types of malware threats that target mobile devices. This research focuses on attacks against the phones themselves and not the service provider's infrastructure. Four classes of attacks on a mobile device were identified (Botha et al., 2009): unsolicited information, theft-of-service, information theft and DoS. Since no malicious applications are yet available for Android, five malicious applications, that perform DoS and information theft attacks, were developed for evaluating the agent and the KBTA processor. DoS attacks against mobile phones can be categorized into the two types. The first attempts to flood and overwhelm a device by issuing fraudulent service requests; the second tries to drain the power resources of the device. Information theft attacks against mobile devices can be classified as targeting either transient information or static information. Transient information includes a mobile device's location, power usage patterns, and other temporal data that a mobile device typically does not record. Static information refers to any information that a mobile device intentionally and persistently maintains/sends on behalf of its owner, i.e., device identification data, contact information, phone numbers, programs and media files. Attackers may attempt to steal both types of information if they perceive it as valuable. Following is a description of the malware that were developed.

6.1.1. Schedule SMS and lunar lander

The first malware, which belongs to the class of information theft, includes two Android applications, Schedule SMS and Lunar Lander, exploiting the Shared-User-ID feature. In Android each application requests a set of permissions that is granted at installation time. The Shared-User-ID feature enables multiple applications to share their permission sets, provided they are all signed with the same key and explicitly request the sharing. It is noteworthy that the sharing is done behind the scenes without informing the user or asking for approval, resulting in implicit granting of permissions. The first application is Schedule SMS, a truly benign application that sends delayed SMS messages to people from a contact list, for which the application requests necessary permissions. The second application, Lunar Lander, is a seemingly benign game that requests no permissions. Nevertheless, once both applications are installed and the Lunar Lander obtains an ability to read the contacts and send SMS messages, it exhibits a Trojan horse-like behavior leaking all contact names and phone numbers through SMS messages to a predefined number. This resembles RedBrowser, a Trojan horse masquerading as a browser that infects mobile phones running J2ME by obtaining and exploiting SMS permissions.

6.1.2. HTTP upload

The second malicious information theft application that was developed, HTTP Upload, also steals information from the device.

It exploits the fact that access to the SD-card does not require any permission. Therefore, all applications can read and write to/from the SD-card. The application requires only the Internet permission and in the background it accesses the SD-card, steals its contents and sends it through the Internet to a predefined address.

6.1.3. Snake

The third and last information theft application masquerades as a Snake game and misuses the devices camera to spy on the unsuspecting user. The Snake game requests Internet permission for uploading top scores. While depicting the game on the screen, the application unobtrusively takes pictures and sends them to a remote server.

6.1.4. Tip calculator

The Tip Calculator is a malicious application that unobtrusively performs a DoS attack. When a user clicks the ‘‘calculate” button to calculate the tip, the application starts a background service that waits for some time and then launches several hundreds of CPU-consuming threads. The effect of this attack is an almost absolute paralysis of the device. The system becomes very unresponsive and the only effective choice is to shutdown the device (which also takes some time). An interesting observation is that the Android system often kills a CPU-consuming service but always keeps on re-launching it a few seconds later. Particularly disturbing is the fact that the malware is able to achieve such an effect without requesting any permission.

6.1.5. Malware injection

A second DoS malware is a virus consisting of PC and Android components. The first component infects the PC and upon detecting a USB connection of an Android-based device, it installs and executes the Android component. The installation occurs without alerting the user which makes it possible to silently install the Android virus that is automatically granted permission to process and block outgoing phone calls.

6.2. Defining a mobile security ontology

The intrusion detection agent monitors every *t* seconds (according to the agent’s settings) more than 160 raw parameters and events (i.e., features), including some that defined basic abstractions and complex patterns. The features appeared within the categories: Applications (e.g., installation, removal, activation); Binder (inter-process communication); CPU load; Phone calls; Hardware; Messaging; Operating System; Permissions; Power; Memory; Keyboard; Scheduler; System configuration; Network; Touch-screen and more. The IDS was installed on five Android devices which were monitored and later deliberately infected with the malwares.

The first step in the evaluation was to define and load the security knowledge-base according to the KBTA ontology. The ontology was defined in XML format. Examples of basic concepts from the mobile security ontology are presented in Table 2. In addition to the pattern in Figs. 2 and 5 present four patterns and their derivation trees as defined in the mobile phone security ontology.

The pattern shown in Fig. 2 illustrates the derivation of a SD-card information leakage pattern that possibly indicates that a Trojan in the mobile device is transmitting private data stored on the SD-card to a remote server. The pattern emerges when a ‘‘Post access to SD-card” context is generated by an ‘‘access to SD-card” event. Then, within the ‘‘Post access to SD-card” context, an INCREASING ‘‘sent packet trend” is interpreted from the ‘‘Sent packets” raw data.

The camera abuse pattern (Fig. 5) is derived from the ‘‘Pictures Taken/min” state and ‘‘Pictures Taken/min” trend when induced within the context of non-system applications with ‘‘Camera permission exists” (i.e., there is a non-framework application, that was installed by the device owner or as a result of a malicious act, and that application has permission to use the camera). The pattern will be generated when a ‘‘High” and ‘‘Increasing” number of pictures are taken. It is noteworthy that the interpretation of the state and trend of the number of taken pictures that generated the pattern is specific to the encompassing context. Had only system applications with ‘‘camera permissions” existed, the interpretation would be different and the pattern would not have been generated.

The malware injection pattern (Fig. 5) depicts a situation where an application is installed on the mobile device while connected to a PC via a USB cable when there is no user activity. First, the personal desktop computer is infected by an Android virus (without the user knowing about it). This virus carries an Android malware payload. When an Android phone is connect to the PC via USB cable, the Android virus identifies the connected phone and executes the installation of the malware payload on the phone. Following the installation, the Android virus executes the malware which disables any outgoing phone calls. This process occurs without user interaction and without any indication of the installation or the execution of the application (see Fig. 6). The ‘‘user activity” state is derived from two ‘‘primitives”: average screen pressure and average dwell time, each characterizing the usage of the touch screen and the keyboard, respectively.

The SMS abuse pattern (Fig. 5) is generated when no user activity exists and at the same time there is an increase in the number of ‘‘outgoing SMS/min”. The derivation of the latter is specific to

Table 2
Examples of concepts from the mobile security ontology.

Concept name	Concept type	Description
CPU usage	Primitive	Measured in% and can be any value from 0 to 100
Garbage collections	Primitive	Measured in number of garbage collections/min and can be any non-negative value
Access to SD-card	Event	An application accessing to the SD-card (e.g., opening/reading a file, listing a directory)
Connecting to USB	Event	Phone is connected to a PC via USB cable
Disconnecting from USB	Event	The USB cable is disconnected from the PC
Post access to SD-card	Context	Generated by an access to SD-card event; starts at the time of the access to the SD-card and ends 30 seconds afterwards
Connected to USB	Context	Generated by connecting to USB event; starts with the occurrence of connecting to USB event and ends when a disconnecting from USB occurs
Sent packets trend. Post access to SD-card	Trend	Derived from the primitive parameter sent packets within the context of post access to SD-card; mapped to the following optional symbolic values: Increasing, same, decreasing
User activity state	State	Derived from the primitive parameters avg screen pressure and avg keyboard dwell time; mapped to the following optional symbolic values: activity, no activity
Context switches state. Post application-run	State	Derived from the primitive parameter context switches within the context of post application-run; mapped to the following optional symbolic values: normal, high

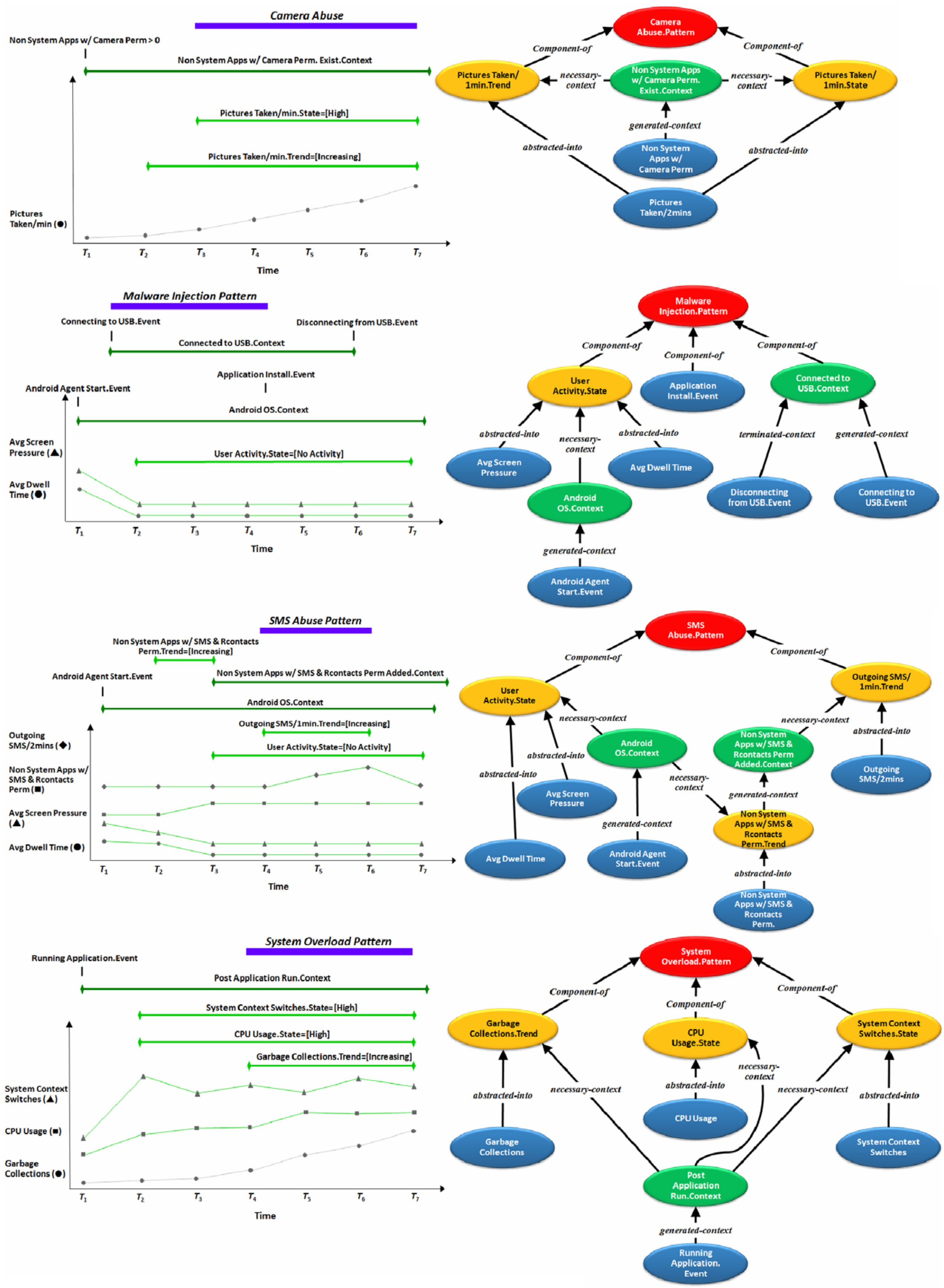
Camera Abuse
Non System Apps w/ Camera Perm > 0
Non System Apps w/ Camera Perm. Exist.Context
Pictures Taken/min.State=[High]
Pictures Taken/min.Trend=[Increasing]
Pictures Taken/min (●)
T_1 T_2 T_3 T_4 T_5 T_6 T_7
Time
Camera Abuse.Pattern
Component-of
Component-of
Pictures Taken/ 1min.Trend
necessary-context
Non System Apps w/ Camera Perm. Exist.Context
necessary-context
Pictures Taken/ 1min.State
generated-context
abstracted-into
abstracted-into
Non System Apps w/ Camera Perm
Pictures Taken/2mins
Malware Injection Pattern
Connecting to USB.Event
Disconnecting from USB.Event
Connected to USB.Context
Application Install.Event
Android Agent Start.Event
Android OS.Context
Avg Screen Pressure (▲)
User Activity.State=[No Activity]
Avg Dwell Time (●)
T_1 T_2 T_3 T_4 T_5 T_6 T_7
Time
Malware Injection.Pattern
Component-of
Component-of
Component-of
User Activity.State
Application Install.Event
Connected to USB.Context
abstracted-into
necessary-context
abstracted-into
terminated-context
generated-context
Avg Screen Pressure
Avg Dwell Time
Disconnecting from USB.Event
Connecting to USB.Event
Android OS.Context
generated-context
Android Agent Start.Event
SMS Abuse Pattern
Non System Apps w/ SMS & Rcontacts Perm.Trend=[Increasing]
Android Agent Start.Event
Non System Apps w/ SMS & Rcontacts Perm Added.Context
Android OS.Context
Outgoing SMS/1min.Trend=[Increasing]
Outgoing SMS/2mins (◆)
User Activity.State=[No Activity]
Non System Apps w/ SMS & Rcontacts Perm (■)
Avg Screen Pressure (▲)
Avg Dwell Time (●)
T_1 T_2 T_3 T_4 T_5 T_6 T_7
Time
SMS Abuse.Pattern
Component-of
Component-of
User Activity.State
Outgoing SMS/ 1min.Trend
necessary-context
necessary-context
Android OS.Context
Non System Apps w/ SMS & Rcontacts Perm Added.Context
abstracted-into
abstracted-into
abstracted-into
Avg Screen Pressure
necessary-context
Outgoing SMS/2mins
generated-context
generated-context
Non System Apps w/ SMS & Rcontacts Perm.Trend
Avg Dwell Time
Android Agent Start.Event
abstracted-into
Non System Apps w/ SMS & Rcontacts Perm.
System Overload Pattern
Running Application.Event
Post Application Run.Context
System Context Switches.State=[High]
CPU Usage.State=[High]
Garbage Collections.Trend=[Increasing]
System Context Switches (▲)
CPU Usage (■)
Garbage Collections (●)
T_1 T_2 T_3 T_4 T_5 T_6 T_7
Time
System Overload.Pattern
Component-of
Component-of
Component-of
Garbage Collections.Trend
CPU Usage.State
System Context Switches.State
necessary-context
abstracted-into
necessary-context
abstracted-into
necessary-context
abstracted-into
Garbage Collections
CPU Usage
System Context Switches
Post Application Run.Context
generated-context
Running Application. Event

Fig. 5. Definition of mobile malware temporal patterns.

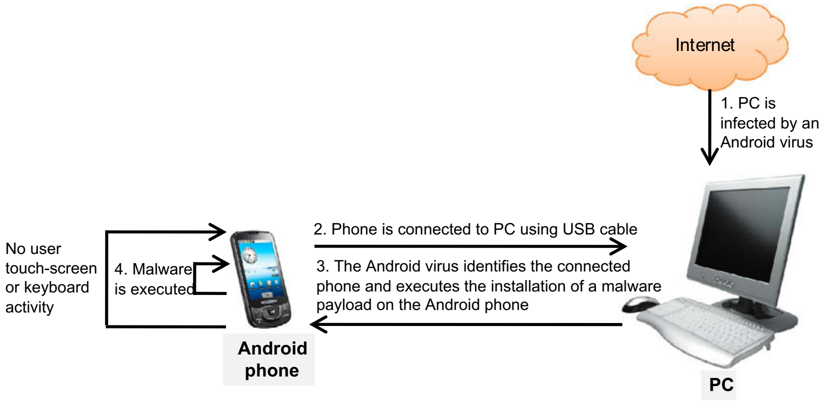
Internet
1. PC is infected by an Android virus
PC
2. Phone is connected to PC using USB cable
3. The Android virus identifies the connected phone and executes the installation of a malware payload on the Android phone
Android phone
No user touch-screen or keyboard activity
4. Malware is executed

Fig. 6. Malware injection process.

the context indicating that non-framework applications have the ability to both read the contact list and send SMS messages.

The system overload pattern (Fig. 5) detects resource consuming applications. It is derived from a concurrent presence of an increasing trend of garbage collections and high states of context switches and CPU usage. The three abstractions are induced within the post-application-run context (which is generated by an application-run event) and indicate a strain on the memory, scheduler and CPU, all caused by a running application.

In addition to a verbal description of the patterns, a more formal definition can be obtained by using CAPSUL language (Chakravarty and Shahar, 2000). The following example defines the malware injection pattern:

Linear Pattern: Malware Injection Pattern
  Context: Connected to USB
  Linear Components:
    Parameter Component:
      User Activity STATE
      Abstracted From:
        Average Screen Pressure
        Average Dwell Time
      Local Constraints:
        value = No_Activity
        duration > 2min
  Linear Components:
    Parameter Component:
      Application Installation EVENT
      Local Constraints:
        value = True
  Global Constraints:
    Quantitative Gap Constraint:
      User Activity STATE DURING Application Installation EVENT
  Output Value of Pattern:
    Value Function: value = Malware Injection

An additional capability of the KBTA-based detection unit is to reduce false alarms. Let us assume that the IDS applies anomaly detection to the user's behavior (e.g., biometric parameters such as keyboard and touch screen activity, or sequences and duration of applications that the user usually activates). When the anomaly level is above a pre-set threshold, the system assumes that the device is not being used by the legitimate owner and can initiate an action such as locking the device, encrypting all the data, and asking for additional identification in order to use the device again. Anomaly detectors are prone to false alarms and an unjustified action, such as device locking, may be annoying to the device owner.

Fig. 7 provides an example of using the KBTA for reducing false alarms. In this example, the anomaly level that is produced by the anomaly detector is input to the KBTA mechanism as primitive parameter. The ontology provides the knowledge for deriving the anomaly level state which is interpreted differently within various contexts, such as the one induced by the occurrence of a call (or sending of an SMS) to a number not from the contact list.

6.3. Evaluation results

In order to evaluate the effectiveness of the method, the defined ontology and the Android implementation, the KBTA-based IDS was activated for one week on five Android devices belonging to different owners. In the course of the week, the devices were used normally. The goals of evaluation were to:

- check the ability of the KBTA-based IDS to detect the five malicious applications
- test the sensitivity of the IDS to the sampling time interval
- identify false alarms
- measure CPU usage while the IDS is activated on the device

Thus, the main goal of the evaluation was not only to evaluate the performance of the method in terms of standard measures such as true-positive and false positive but also to understand the sensitivity of these measures to the sampling time interval. This should be investigated since smartphones have limited resources. Moreover, the KBTA method's inference mechanisms are sensitive to the sampling interval (e.g., the persistency computation and the trend abstraction). Thus, a larger sampling time interval will consume less CPU and battery but there is the possibility of missing a pattern. Consequently, based on the nature of the malware used for the evaluation, with most exhibiting abrupt and short activity, the following sampling time interval were chosen: 2, 5, 8 and 15 s.

6.3.1. Detecting malicious software

Each malware was activated 10 times on each of the five devices, with the IDS sampling interval set at 2 s, 5 s, 8 s and 15 s. Table 3 summarizes the results in terms of average detection time and rate (averaged over all activations and all devices). Fig. 8 depicts the detection time for the five malware as a function of the sampling time interval. Similar detection intervals for 2, 5 and 8 s can be observed with a major gap between 8 s and 15 s. Thus, monitoring with a sampling interval of 8 s is preferred since it will cause less of a strain on the system while maintaining reasonable detection times.

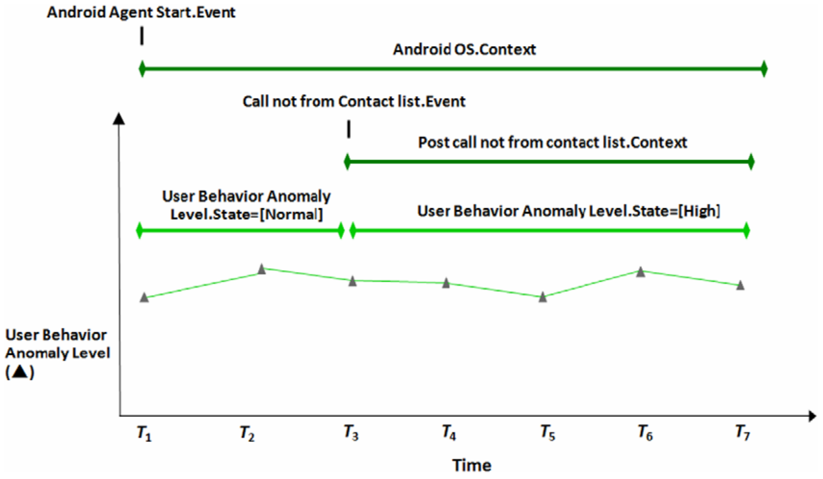
Android Agent Start.Event
Android OS.Context
Call not from Contact list.Event
Post call not from contact list.Context
User Behavior Anomaly Level.State=[Normal]
User Behavior Anomaly Level.State=[High]
User Behavior Anomaly Level (▲)
T_1
T_2
T_3
T_4
T_5
T_6
T_7
Time

Fig. 7. Context-aware interpretation of an anomaly detection alerts.

Table 3
Malware detection results.

Malware	Description	Action when detected	Sampling interval (sec)	Detection time (sec)	Detection rate (%)
Snake	Information theft – while playing a snake game, in the background the application is quietly taking pictures and sending them off to a remote server	Automatically removed the application	2	29	100
			5	33	100
			8	36	100
			15	50	100
Tip calculator	Denial-of-service – launches a background service that in turn spawns hundreds of threads; thus effectively overloading the system	Automatically removed the application	2	32	100
			5	41	100
			8	47	100
			15	72	100
Malware injection	Denial-of-service – A virus on a PC secretly installs a virus on the Android which blocks outgoing phone calls	Offers to uninstall the application	2	14	98
			5	14	96
			8	17	98
			15	27	96
SD-card leakage (HTTP uploader)	Information theft – reads sensitive information stored on the SD-card and other locations and sends it to a remote server	Disabling all communications (Wi-Fi, Cellular)	2	7	100
			5	9	100
			8	16	96
			15	22	94
Lunar Lander & SMS scheduler	Information theft – exploiting the Shared-User-ID and reading contacts and sending them via SMS	Offers to uninstall the applications	2	5	100
			5	8	100
			8	–	0
			15	–	0

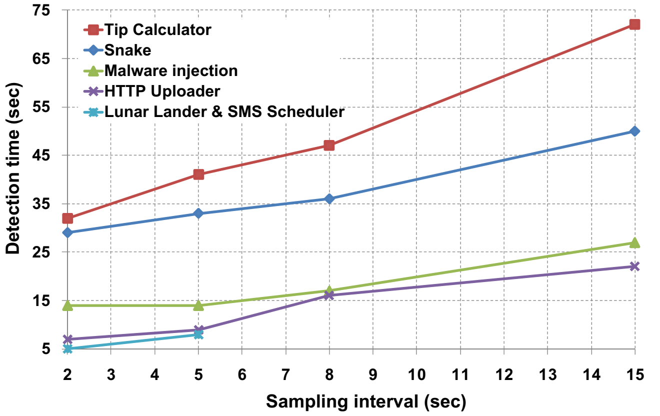

Fig. 8. Presenting the detection time for the five malware as a function of the sampling time interval.

When analyzing the detection rate, the malware injection pattern and the SD-card malware were not detected in several of the activations. The malware injection pattern went undetected due to “accidental” touch screen activity during the installation. This prevented the pattern from being generated since it is necessary that neither touch screen nor keyboard activity occur at the time of the installation (while connected via USB).

The SD-Leakage malware went undetected three times when setting the sampling interval to 15 s and twice when setting the sampling interval at 8 s. The detection failure was due to the fact that latency in the network connection may hinder the derivation of the sent packets trend and result in a different temporal behavior than that expected by the pattern. One of the qualities of the temporal patterns in KBTA is the ability to define behavior in a fuzzy manner that can account for different yet similar temporal sequences. The network latency can thus be accounted for by further “fuzzing” the pattern’s definition.

A SMS abuse pattern was not detected when the sampling time interval was set at 8 and at 15 s. This was because an essential component of the SMS abuse pattern is the increase in the number of non-system applications with the ability to read the contacts and send SMSs, which is detected using a trend abstraction. The direction of the trend (i.e., increasing, decreasing or same) is obtained by calculating the value variation over the time interval and comparing it with the threshold specified in the knowledge-base. However, changing the sampling interval affects the calculated variation which in turn affects the direction of the trend. This prevents the pattern from being generated. To conclude, once the optimal sampling time interval is determined, the ontology can be modified accordingly.

6.3.2. False alarm scenarios

The fuzzy nature of the patterns can be perceived both as an advantage and disadvantage. On the one hand, the fuzzy status offers the ability to capture previously unencountered behavior (e.g., new malware); on the other, the fuzziness may also become a hindrance by causing false alarms. During the activation on the five devices, false alarms were identified in the "Camera abuse" (Snake application), malware injection and SD-card leakage patterns.

A false camera abuse pattern can occur when installing a legitimate third-party (non-system) camera application, and using that application rapidly. To reduce false alarms, the pattern can be modified with additional components. For example, assuming that the malware will attempt to leak the pictures from the device via the network without storing the pictures, a check for lack of access to the SD-card events (for storing pictures) or for abnormal network traffic states and trends can be added.

A false malware injection occurred during the experiments when a user connected his/her Android phone to a PC (via USB cable) and used the "adb install" command to install an application. The adb command is mainly used by developers or expert users. Thus, when such a false alarm occurs, the advanced user should be able to identify whether it is a false alarm caused by intentionally installing the application or the result of a malicious attack.

The SD-card leakage false alarm may occur when an application is accessing the SD-card (e.g., a camera application storing the pictures, looking for ringtones) and at the same time another application is sending data over the Internet (e.g., Android automatic synchronization mechanisms). This scenario can be overcome by adding the ability to link the concepts (access to the SD-card and network traffic state/trend) to the same application and to identify whether the application is a framework application.

6.3.3. Measuring the strain on the CPU caused by the IDS

The CPU usage was recorded while activating the KBTA-based IDS with a sampling interval of 2 s and while normally using the Android device. Fig. 9 depicts the CPU usage measured during the course of one hour. It is evident that for the most part the CPU usage is approximately 3% with peaks of less than 9% while processing data. The rate of extreme peaks can be accounted for by extreme activity of other applications or by the system at the time of the measurement. To conclude, the IDS does not hinder the system nor greatly affect the user experience and can be deemed as unobtrusive and with low resource demands.

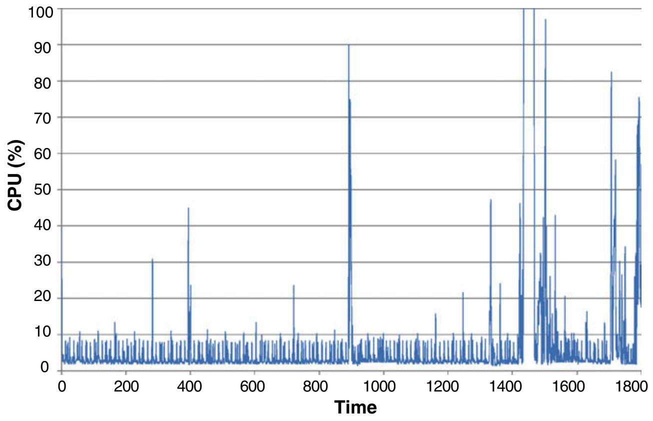

Fig. 9. CPU consumption while activating the KBTA-based IDS.

7. Summary and conclusions

This paper presents a modified, lightweight version of the KBTA method for detecting malware on mobile devices by analyzing temporal security data. The data is analyzed in order to identify new malware compatible with predefined temporal patterns specified at a high level of abstraction. The implementation of the modified, incremental KBTA method on Android is capable of automatically and continuously creating new abstractions from a continuous flow of raw data. These abstractions are automatically monitored to alert the user (or another process) whenever a suspicious pattern emerges, indicating the possible presence of a malware.

The new approach can also quickly adapt to new malware classes by updating the knowledge-base (i.e., security ontology). In addition, the proposed method is platform-independent and the same ontology can fit any smartphone or mobile operating system.

The KBTA method provides concise, meaningful summaries of large amounts of temporal security data in terms familiar to the user or security experts. In addition, KBTA defines malware patterns in a fuzzy fashion as a set of constraints rather than as a hard-coded signature for each and every known malware. Consequently, it facilitates detection of instances of malware even when they have not been encountered before.

A prototype, capable of creating and monitoring temporal abstractions and patterns on Android mobile phones, was implemented. The prototype demonstrated the capabilities of the method in defining and evaluating new temporal patterns that might indicate the existence of malware. The KBTA method is platform-independent and as such, it can be applied to new domains, without additional coding, by providing domain knowledge and domain raw data to the KBTA framework. The ontology evaluated in this paper is defined using concepts in a high level of abstraction (e.g., no user activity, low memory state etc.) that are common to any smartphone and thus can be applied to any smartphone or mobile operating system (e.g., Symbian, Windows Mobile, and iPhone). The main effort in such case is the extraction of raw data (primitive parameters and events) from the system and the implementation of the remedial actions to be taken when a temporal pattern representing malicious behavior is detected. These two tasks are platform dependent and are more accessible and easy to implement in open-source platforms such as the Android. Thus, given a proper implementation of the detection framework on an alternative smartphone OS, the proposed ontology will be efficient at the same extent.

An additional advantage is that the system can help in integrating alerts from other sensors as primitive parameters and "smoothing" the alerts by applying the temporal abstraction process which facilitates context-based interpretation of the alerts. The user would be notified only if the alert instances persist; thus reducing the amount of false alarms.

The applicability of a KBTA-based intrusion detection system was demonstrated by developing a powerful framework without excessive demands on resources. The intrusion detection framework is also modular and it can be extended with new analysis or detection units (i.e., algorithms), such as using machine-learning anomaly detection and classification algorithms. As a case in point, the experiments in (Shabtai and Elovici, 2010) used the same framework presented here. In the experiments several classification and anomaly detection algorithms (Decision Tree, Naïve Bayes, Bayesian Networks, *k*-Means, Histogram and Logistic Regression) were evaluated for their capability in detecting unknown malicious applications.

The KBTA detection unit potentially supports both misuse detection and anomaly detection. Anomaly detection can be supported by defining temporal patterns of normal behavior (for example, normal behavior of network connections) and by identifying time-intervals in which the normal behavior patterns are not derived as proposed in Seleznyo and Mazhelis (2002). Manual definition of temporal patterns along with the output of a temporal data mining process employed on raw and abstracted data will define patterns for both normal and malicious behavior. The current study is being extended by conducting research that deals with such a temporal data mining framework and new results will hopefully be presented in the future.

Future work will also focus on developing an Android-user interface for defining patterns and the ontology. Finally, the KBTA's pattern matching mechanism can be extended to support the derivation of patterns (optionally assigned with a level of certainty) even if not all of its components exist, resulting in partial patterns that can provide better accuracy and a faster response time.

References

Aguilar, J., 2005. A survey about fuzzy cognitive maps papers. International Journal of Computational Cognition 3 (2), 27–33.

Allen, J.F., 1983. Maintaining knowledge about temporal intervals. Communications of the ACM 26 (11), 832–843.

Bose, A., Hu, X., Shin, K.G., Park, T., 2008. Behavioral detection of malware on mobile handsets. In: Proceeding of the Sixth International Conference on Mobile Systems, Applications, and Services.

Botha, R.A., Furnell, S.M., Clarke, N.L., 2009. From desktop to mobile: examining the security experience. Computer and Security 28, 130–137.

Buennemeyer, T.K. et al., 2008. Mobile device profiling and intrusion detection using smart batteries. International Conference on System Sciences, pp. 296–296.

Chakravarty, S., Shahar, Y., 2000. CAPSUL: A constraint-based specification of repeating patterns in time-oriented data. Annals of Mathematics and AI 30 (1–4), 3–22.

Cheng, J., Wong, S.H., Yang, H., Lu, S., 2007. SmartSiren: virus detection and alert for smartphones. In: Proceedings of the Fifth International Conference on Mobile Systems, Applications and Services.

Dikinson, J., 2005. The new antivirus formula. http://www.ironport.com/pdf/ironport_new_anti-virus_formula.pdf.

Ghosh, A.K., Schwartzbard, A., Schatz, M., 1999. Using program behavior profiles for intrusion detection. In: Proceedings of the First USENIX Workshop on Intrusion Detection and Network Monitoring.

Hwang, S.S., Cho, S., Park, S., 2009. Keystroke dynamics-based authentication for mobile devices. Computer and Security 28, 85–93.

Jacoby, G.A., Davis, N.J., 2004. Battery-based intrusion detection. Global Telecommunications Conference (GLOBECOM'04).

Kim, H., Smith, J., Shin, K.G., 2008. Detecting energy-greedy anomalies and mobile malware variants. In: Proceeding of the Sixth International Conference on Mobile Systems, Applications, and Services.

Kohout, L.J., Yasinsac, A., McDuffie, E., 2002. Activity profiles for intrusion detection. North American Fuzzy Information Processing Society-Fuzzy Logic and the Internet.

Lane, T., Brodley, C.E., 1999. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security 2 (3), 295–331.

Li, Y. et al., 2002. Enhancing profiles for anomaly detection using time granularities. Journal of Computer Security 10 (1–2), 137–157.

Martin, T., Hsiao, M., Ha, D., Krishnaswami, J., 2004. Denial-of-service attacks on battery-powered mobile computers. In: Second IEEE International Conference on Pervasive Computing and Communications, pp. 309–318.

Miettinen, M, Halonen, P., Hätönen, K., 2006. Host-based intrusion detection for advanced mobile devices. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications.

Moreau, Y., Verrelst, H., Vandewalle, J., 1997. Detection of mobile phone fraud using supervised neural networks: a first prototype. In: Proceedings of the Seventh International Conference on Artificial Neural Networks.

Morin, B., Debar, H., 2003. Correlation of intrusion symptoms: an application of chronicles. In: Proceedings Recent Advances in Intrusion Detection (RAID) Symposium.

Moskovitch, R., Shahar, Y., 2009. Medical temporal-knowledge discovery via temporal abstraction. AMIA 2009, San Francisco, USA.

Muthukumaran, D. et al., 2008. Measuring integrity on mobile phone systems. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies.

Naldurg, P. et al., 2004. A temporal logic based framework for intrusion detection. In: Proceedings of the 24th Formal Techniques for Networked and Distributed Systems International Conference.

Nash, D.C. et al., 2005. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. Pervasive Computing and Communications Workshops.

Ning, P., Jajodia, S., Wang, X.S., 2001. Abstraction-based intrusion detection in distributed environments. ACM Transactions on Information and System Security 4 (4), 407–452.

Piercy, M., 2004. Embedded devices next on the virus target list. IEE Electronics Systems and Software 2, December–January, pp. 42–43.

Racic, R., Ma, D., Chen, H., 2006. Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phone's Battery. Securecomm and Workshops, pp. 1–10.

Samfat, D., Molva, R., 1997. IDAMN: an intrusion detection architecture for mobile networks. IEEE Journal on Selected Areas in Communications 15 (7), 1373–1380.

Schmidt, A.D., Peters, F., Lamour, F., Scheel, C., Camtepe, S.A., Albayrak, S., 2009. Monitoring smartphones for anomaly detection. Mobile Networks and Applications (MONET) 14 (1), 92–106.

Seleznyov, A., Mazhelis, O., 2002. Learning temporal patterns for anomaly intrusion detection. In: Proceedings of the 17th ACM Symposium on Applied Computing.

Shabtai, A., Fledel, Y., Elovici, Y., Shahar, Y., 2009. Using the KBTA method for inferring computer and network security alerts from time-stamped, raw system metrics. Journal in Computer Virology 8 (3), 267–298.

Shabtai, A., Elovici, Y., 2010. Applying behavioral detection on android-based devices. In: Proceedings of the 3rd International Conference on Mobile Wireless Middleware, Operating Systems, and Applications (Mobilware 2010), Springer, Chicago, USA.

Shahar, Y., 1997. A framework for knowledge-based temporal abstraction. Artificial Intelligence 90 (1–2), 79–133.

Shahar, Y., Musen, M.A., 1996. Knowledge-based temporal abstraction in clinical domains. Artificial Intelligence in Medicine 8 (3), 267–298.

Siraj, A., Vaughn, R.B., Bridges, S.M., 2004. Intrusion sensor data fusion in an intelligent intrusion detection system architecture. In: Proceedings of the 37th Hawaii International Conference on System Sciences.

Stach, W., Kurgan, L., Pedrycz, W., 2005. A survey of fuzzy cognitive maps learning method. In: Grzegorzewski, P., Krawczak, M., Zadrozny, S. (Eds.), Issues in Soft Computing: Theory and Applications, Exit, pp. 71–84.

Talbi, M., Mejry, M., Bouhoula, A., 2008. Specification and evaluation of polymorphic shellcode properties using a new temporal logic. Journal in Computer Virology.

Yap, T.S., Ewe, H.T., 2005. A mobile phone malicious software detection model with behavior checker. Lecture Notes in Computer Science 3597, 57–65.

Ye, N., 2000. A Markov chain of temporal behavior for anomaly detection. Workshop on Information Assurance and Security.

Asaf Shabtai is a PhD student at Ben-Gurion University of the Negev (BGU), Israel, and an R&D manager at Deutsche Telekom Laboratories. His main areas of interests are computer and network security, data mining, and temporal reasoning. He has an MSc in information systems engineering from BGU.

Uri Kanonov has a BSc in software engineering from Ben-Gurion University of the Negev (BGU), Israel.

Yuval Elovici is the director of Deutsche Telecom Laboratories at Ben-Gurion University of the Negev (BGU), Israel, and is a member of the information systems engineering department. His main areas of interest are computer and network security, information retrieval, and data mining. He has a PhD in information systems from Tel-Aviv University, Israel.