J Comput Virol (2010) 6:239–259
DOI 10.1007/s11416-009-0125-5

ORIGINAL PAPER

Using the KBTA method for inferring computer and network security alerts from time-stamped, raw system metrics

Asaf Shabtai · Yuval Fledel · Yuval Elovici · Yuval Shahar

Received: 3 January 2009 / Accepted: 23 June 2009 / Published online: 23 July 2009

Abstract In this study, we propose a new approach for detecting previously unencountered instances of known classes of malicious software based on their temporal behavior. In the proposed approach, time-stamped security data are continuously monitored within the target computer system or network and then processed by the knowledge-based temporal abstraction (KBTA) methodology. Using KBTA, continuously measured data (e.g., the number of running processes) and events (e.g., installation of a software) are integrated with a security-domain, temporal-abstraction knowledge-base (i.e., a security 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 can be monitored to detect suspicious temporal patterns. These patterns are compatible with a set of predefined classes of malware as defined by a security expert employing a set of time and value constraints. The new approach was applied for detecting worm-related malware using two different ontologies. Evaluation results demonstrated the effectiveness of the new approach. The approach can be used for detecting other types of malware by updating the security ontology with new definitions of temporal patterns.

A. Shabtai (✉) · Y. Fledel · Y. Elovici · Y. Shahar
Deutsche Telekom Laboratories at Ben-Gurion University, and Department of Information Systems Engineering, Ben-Gurion University, 84105 Beer Sheba, Israel
e-mail: shabtaia@bgu.ac.il

Y. Fledel
e-mail: fledely@bgu.ac.il

Y. Elovici
e-mail: elovici@bgu.ac.il

Y. Shahar
e-mail: yshahar@bgu.ac.il

Keywords Knowledge-based systems · Temporal-abstraction · Computer and network security · Host-based intrusion detection systems · Intelligent visualization

1 Introduction

Today's computer and telecommunication infrastructure interconnects a heterogeneous collection of routers, servers and end-user devices (i.e., desktops, laptops, cellular phones) operating various kinds of software and services. By its open and distributed nature, such an infrastructure is highly susceptible to malicious attacks. One common way of unleashing such attacks is by employing malware (*mal*icious soft*ware*) such as worms, viruses, Trojans or spyware [1,2] which can cause great damage.

There is a plethora of solutions for tackling malware. Antivirus packages are a common method employed by end-user devices and servers. However, 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 unencountered malware. Since the response time of antivirus vendors may vary between several hours to several days to identify the new malware, generate a signature, and update their clients' signature database, hackers have a substantial window of opportunity [3]. This problematic scenario calls for novel methods to rapidly detect previously unencountered malware (presumably belonging to some known malware class) even before its signature is announced.

The main assumption in this study is that behavioral patterns over time of a potential previously unencountered malware 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 [4,5].

Detecting such malware usually involves processing and inspecting an enormous volume of time-stamped raw measurements generated by many data sources. Inspecting the temporal behavior of the whole network, or of a specific computer, in order to determine their status, also requires context-sensitive interpretation of such accumulated data. This high-level, context-sensitive, knowledge-based abstraction of time-oriented data is referred to as a temporal abstraction of the data [6].

To enable early detection of malware temporal patterns we propose a comprehensive architecture and workflow termed eTIME or Electronic Time-oriented Intelligent Monitoring and Exploration. This detection approach combines automatic monitoring capabilities, and support for manual, interactive inspection and visual exploration of continuously accumulated temporal, raw security data and abstractions. eTIME architecture employs a Knowledge-Based Temporal Abstraction (KBTA) framework for representing the security domain knowledge and deriving temporal abstractions. The KBTA method has been used successfully to support many tasks, especially diagnostic or therapeutic decision support [7] 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 proposed architecture incorporates modules that facilitate: data collection from various data sources; continuous inference of temporal abstractions; monitoring the derived temporal-abstractions; manual, dynamic exploration of the contents of security storage; and a knowledge-acquisition and maintenance tool that enables a security expert to easily add or modify temporal patterns in the knowledge base.

eTIME was integrated and evaluated in two scenarios. In the first scenario eTIME was analyzed data collected from end-user devices to identify potentially infected devices. In the second scenario, eTIME analyzed traffic extracted from an IP-compliant network to identify potentially infected files.

The rest of the paper is organized as follows: In Sect. 2, an overview of related work is presented. Section 3 describes the knowledge-based temporal abstraction method. Next, Sect. 4 introduces the proposed architecture and its components. In Sect. 5 we present results from the evaluation of the proposed framework, while Sect. 6 discusses the advantages of the described architecture, and our current efforts to enhance the architecture.

2 Related work

Enormous efforts are being made to provide effective defense against malware. The academic research community is very much engaged in investigating the problem while commercial bodies are developing products that employ various detection and filtering techniques bundled into anti-virus software and intrusion detection or prevention systems (IDS/IPS).

There are two basic approaches for detecting and filtering malware: signature-based detection (or misuse detection) and anomaly detection [8,9]. With the signature-based approach, a signature of a known malware is synthesized and the anti-virus or IDS signals once the signature is detected. The signature might be in the form of a regular expression representing the malicious software code, or in the form of an attack pattern representing unauthorized behavior exploiting known system or software vulnerabilities. In the case of signature-based detection, an updated repository of all known malware signatures must be maintained and updated whenever a new malware is detected. This approach has several drawbacks: (1) the size of the signature repository increases steadily and the matching process against all potential signatures is too lengthy; (2) When a signature database is large, there is a chance that valid content will be incorrectly matched to a malware signature; (3) Changing a single bit in a malware's code may require a new signature and thus a single malware will produce multiple signatures. Despite these drawbacks, this approach is commonly used due to its simplicity and effectiveness in detecting known attacks with low false-positive rates [10–12].

With the anomaly detection approach [13], the detection system is trained with samples of normal behavior of a device, and the IDS signals once an anomalous behavior is detected. While this approach detects new attacks classified as exhibiting anomalous behavior, it usually suffers from a high false-positive detection rate that a security expert must handle [11,12].

Some signature-based methods attempt to generate more generalized malware signatures by learning and clustering the behavior of malicious and benign software. For example, Moskovitch et al. [14] focuses on the feasibility of accurately detecting unknown worm activity in individual computers using *machine learning* classifiers. Jacob et al. [11] present a survey and taxonomy of a behavioral detection system.

Some academic studies have used temporal features to represent the normal behavior (i.e., temporal behavior) [15] of users, systems or networks. In [4], 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 user behavioral patterns.

Ghosh [16] used anomaly detection to represent a program's normal behavior as a frequency table-storing sequence

of system calls. The work by [17–19] describes attack patterns or normal behavior patterns using Allen's temporal relations [20] such as "before", "after" and "meet" between events (e.g., a login by user x should be followed by a logout).

A *Markov* chain was employed in [5] 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 based on the model of the normal behavior. A low probability indicates a high likelihood of abnormal behavior.

Li et al. [21] employ *association-rules* with time granularity to define a normal behavior within different temporal intervals (e.g., network traffic during weekend vs. weekdays). The *Temporal Logic* approach was used in [22] for detecting polymorphic malicious codes that exploit buffer-overflow vulnerabilities.

Morin and Debar [23] used *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 used for defining temporal patterns as a set of events and time constraints. Several scenarios are provided in order to exemplify the benefits of the method: (1) a buffer overflow false alarms is mitigated by tagging a suspicious shellcode that appears during a normal ftp file transfer; (2) multiple alerts related to the same port scan event are aggregated into a single port scan chronicle; (3) reducing the amount of alerts by aggregating recurrent alerts related to the same attack is exemplified using the definition of the Nimda worm chronicle; and (4) alerts from three different sensors and describing three different events are aggregated into a successful code execution attack on a host. As described below, the KBTA method demonstrated in this research can also assist in reducing the number of alerts by using context-based interpretations.

Other formal languages such as Lambda [24], STATL [25] and P-BEST [26] were also utilized for specifying attack patterns and for alert correlation. In Lambda, attack scenarios are described as steps representing the attacker's actions. STATL is a transition-based language describing attacks as sequences of actions performed by an attacker. In P-BEST, attacks are described using a production rule specification language. Unlike the three previous languages, in chronicle formalism as well as in the proposed KBTA method, time is fundamental and an integral part of the semantics of patterns.

To date, a majority of state-of-the-art commercial malware detection products operate in a signature-based mode. Few of them attempt to complement signature-based detection with anomaly detection techniques. Nevertheless, most of the aforementioned solutions do not consider temporal features as first-class-citizen when analyzing data captured from networks or devices. Furthermore, these solutions usually focus on a specific task or domain, such as analyzing and detecting attacks in network traffic rather than providing an overall solution that supports detection of malware using different types of data sources.

In this paper, we incorporate the KBTA method in order to implement a time-aware malware detection system. Using this system the security expert defines a security ontology using standard terms from the security domain. As part of the ontology, a security expert can define temporal patterns of attack or normal behavior as a set of time and data constraints. Such temporal patterns are fuzzy, behavioral signatures indicating the temporal behavior of an attack versus valid activity. The KBTA inference mechanisms mesh a continuous flow of temporal raw security data with the security ontology to generate instances of higher level temporal abstractions. Such temporal information can be monitored and used for issuing alarms whenever an instance of a temporal pattern indicating an attack is detected. In addition, the new architecture supports visual exploration using temporal queries to drill-down and trace the source of an attack or identify new temporal patterns indicating suspicious behavior. The following section describes the KBTA method.

3 The KBTA method

Temporal-Abstraction (TA), a common and important *Temporal Reasoning* (TR) task 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 [7] is a computational framework for supporting the TA task. The 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 abstraction of meaningful patterns from time-oriented security data).

In general, the KBTA method consists of the input which includes a set of time-stamped parameters (e.g., number of FTP connections at each time-point) and events (e.g., keyboard or mouse activity) which create the necessary interpretive context (e.g., "No User Activity" when the mouse and keyboard are not used). The output comprises 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 h of a high number of FTP connections in which no user activity was detected).

A knowledge engineer defines the domain knowledge (e.g., security ontology) by means of five KBTA entities and the relations between them. Five inference mechanisms

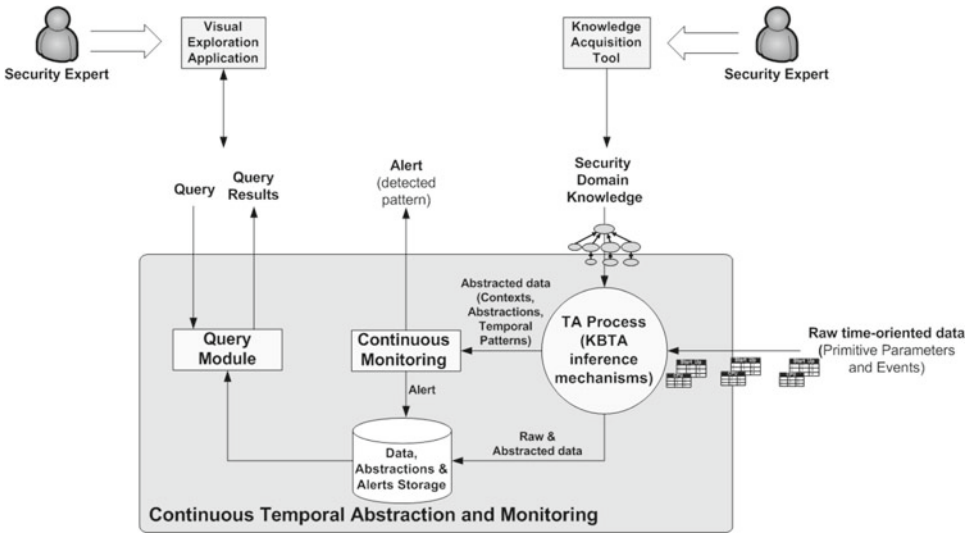

Fig. 1 The KBTA framework

(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 [6]. Figure 1 illustrates the KBTA method. Time-stamped measurements of primitive parameters and time-stamped events, as well as 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.

3.1 Ontological entities

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

Primitive parameters are raw measurable data collected from different sensors (e.g., CPU usage, number of running processes, the number of transmitted RST packets). *Events* are raw data representing actions caused by an outside entity, such as opening a browser, change in a registry or a software installation. *Contexts* are the "state of affairs" of a monitored subject (e.g., a computer). Contexts are induced dynamically, usually by the existence of an event; they affect the interpretation of parameters. The same dataset may be interpreted differently within different contexts. For example, uploading files using a FTP connection 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 Trojan horse).

Abstract parameters are derived from one or more parameters (primitive or abstract). Part of the abstract parameter's knowledge is a classification function that maps the values of the "abstracted-from" parameters to the values of the abstracted parameter. For example, "Memory Utilization STATE" is an abstract parameter abstracted from the primitive parameter "Memory Utilization" (the percentage of memory used at each time point). A classification function can map 0–10% memory utilization to a LOW state. A context is required in order to derive an abstract parameter, and within different contexts, the abstract parameter will have different classification functions. Consequently the same input may result in different output values.

There are three types of abstract parameters: State, Gradient, and Rate. These correspond to three types of abstractions, respectively. State abstractions map the values of the "abstracted-from" parameters values to a "state-describing" set of values (e.g., LOW memory utilization). Gradient abstraction determines the direction of the change of values in a measured parameter (e.g., INCREASING number of failed connections). Rate abstraction classifies the amplitude of a rate of change of a selected parameter (e.g., FAST changing number of modified exe files).

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 [27]. 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 13 temporal relations. The following is an example of a

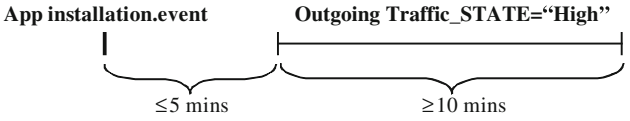

Fig. 2 Example of a pattern with a global constraint (the event is followed by a state abstraction within 5 min) and a local constraint (the state abstraction is "High" for at least 10 min)

temporal pattern: "Application installation on a PC followed within 5 min by HIGH outgoing network traffic for at least 10 min" (Fig. 2). 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 occurring 4 times in 1 week).

Figure 3 shows an example of the temporal abstraction process in the computer-networks security domain. It illustrates the derivation of an IP scan pattern that may indicate that a computer worm in the computer is trying to infect other computers. The input to the KBTA inference mechanisms consists of raw measurements of "System Context Switches" and "TCP Connection Failures" at T_1–T_7, and the "Connection Established" event, which generates the Internet Connection context. Then, within the Internet Connection context, a state of "VERY-HIGH System Context Switches" is interpreted from the "System Context Switches" raw data. An "INCREASING number of TCP Connection Failures" is interpreted from the "TCP Connection Failures" raw data. The IP scan pattern, which is derived from these two contemporaneous abstractions, indicates that the monitored computer is infected with a worm.

The following table summarizes the five KBTA entity types (input types: primitive parameter and event; and output types: context, abstract parameter and pattern) and their semantic relationships. For example, abstract parameter is *abstracted-from* one or more other parameters (primitive or abstract) and can be *abstracted-into* a pattern or other abstract parameter. It can also generate a context (Table 1).

3.2 Inference mechanisms

In order to compute higher level abstractions from a given raw data repository, KBTA uses five inference mechanisms: Temporal Context Formation, Contemporaneous Abstraction, Temporal Inference, Temporal Interpolation and Temporal Pattern Matching [6].

The *Temporal Context-Formation* mechanism creates interpretation-context intervals that enable the TA mechanisms to create context-specific abstractions. This mechanism requires context-induction knowledge such as what concepts generate a context (including start, end and duration of the context). In the example in Fig. 4, the installation event generates the post-installation context interval. The start time of the context is the end of the installation event and the end of the context is 5 h after the end of the installation event. A context can be created backwards (retrospect). This can be very helpful in tracing the source of an attack. For example, if we know that at some point of time a server was attacked, we can generate an attack context from the time it was discovered and backwards for a predetermined temporal duration. This will involve re-inspecting historical data and might create new abstractions that can potentially reveal the source of the attack. As a case in point, software installed from an external network (which is an event) that did not seem suspicious before, might now look suspicious within the context of the new attack.

The *Contemporaneous Abstraction* mechanism is a mechanism for abstracting higher level abstractions (State, Gradient or Rate) from one or more parameters occurring

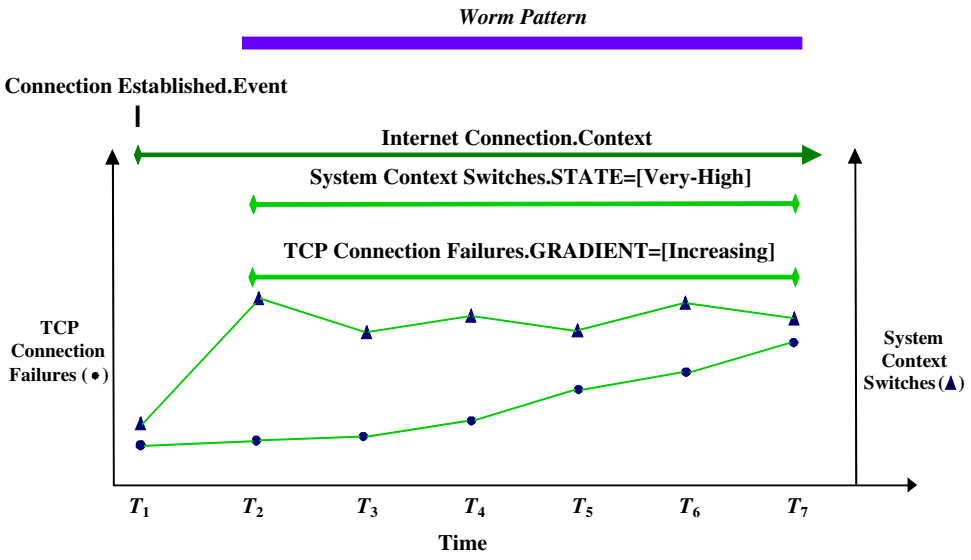

Fig. 3 An example of a worm pattern: raw data is plotted at the bottom. Events and the abstraction computed from the data are plotted as intervals above the data. *Vertical lines* an event, *bullet* number of TCP connection failures, *black triangle* number of system context switches, *dashed lines* a context open interval, *straight lines* an abstraction (derived concept) interval

Table 1 The KBTA entities and their semantic relationships

Entity (is-a)	Children relation	Parent relation	Context relation	Example
Primitive parameter	–	Abstracted-into	Generated-contexts	CPU usage
Abstract parameter	Abstracted-from	Abstracted-into	Generated-contexts	TCP connection failures gradient
Event	Parts	Part-of	Generated-contexts	Installation
Context	Sub-context	Super-context	Generated-from	Post installation context
Pattern	Components	Component-of	Generated-contexts	Worm pattern

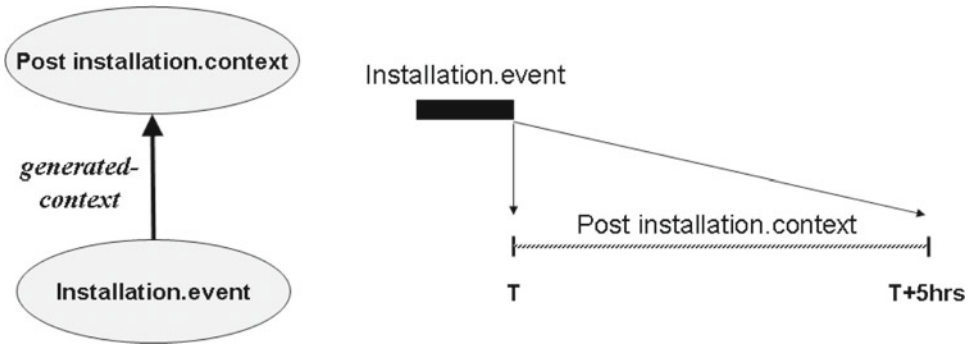

Fig. 4 An example of the Temporal Context Formation mechanism. The closed context interval (post installation) is generated from an event (installation), it starts right after the event finished and lasts for 5 h

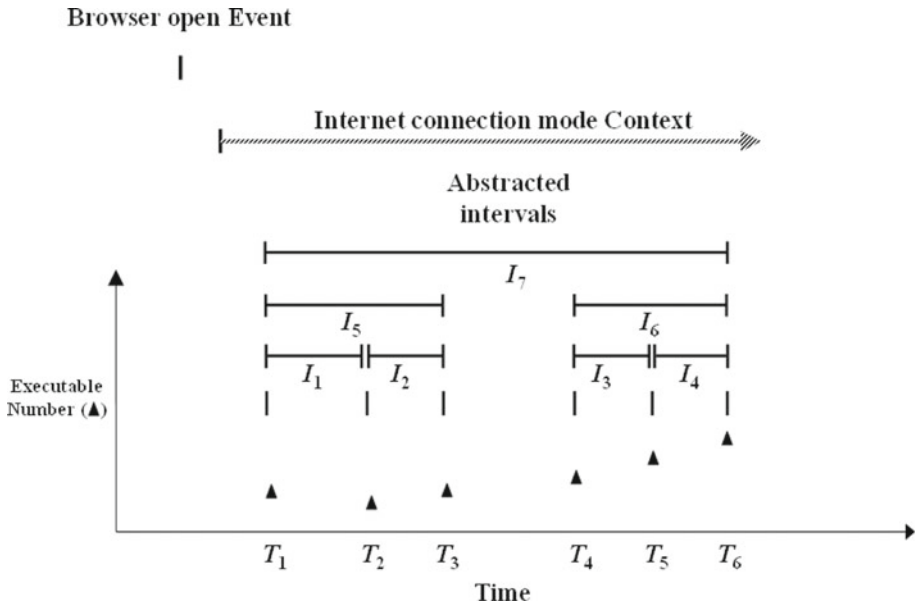

Fig. 5 The number of executable files parameter points (T_1–T_6) are abstracted into abstraction points, over which a High number of executables state abstraction is interpreted, by the contemporaneous abstraction. These point abstractions are first joined into a High number of executable abstraction intervals I_1, I_2, I_3 and I_4 by temporal interpolation. Abstractions I_1 and I_2 are joined by temporal inference into a longer abstraction interval I_5 as are I_3 and I_4 into I_6. I_5 and I_6 are joined into a High number of executable abstraction intervals I_7 by the temporal interpolation

simultaneously. The time interval of the derived fact is the intersection of the base parameter facts; the value is the result of applying a classification function on the base parameter facts. For example, in Fig. 5, the number of executable files parameter time-point measurements is abstracted into the Executable Number STATE abstraction (time-points facts as well).

The *Temporal Inference* mechanism implements inferences from similar-type propositions that hold for different time intervals according to the following temporal semantic properties: "concatenable", "downward hereditary", "gestalt", "solid" and "forward/backward diffusive". The semantic properties are part of the concept's knowledge. For example, in Fig. 5, by using the "concatenable" property, the temporal inference can join similar-value abstractions of the same parameter that hold for two meeting time intervals (intervals I_1 and I_2 are joined to interval I_5; interval I_3 and I_4 are joined to interval I_6). Another property is "downward-hereditary" which is useful in answering queries. This property states whether one can conclude that if a predicate is true over some time interval then it is true over any contained interval.

The *Temporal Interpolation* mechanism bridges gaps between temporally disjoint point- or interval-based facts of a similar-type (but potentially different values) to create longer intervals (e.g., joining two temporally disjoint high

CPU usage intervals into a longer interval). This mechanism requires temporal semantic knowledge (is the concept concatenable?) and interpolation knowledge that can tell, for example, what is the maximal gap that enables concatenating two point-based facts. Figure 5 demonstrates the joining of point propositions (e.g. joining two time-points facts into I_1) and interval propositions (e.g. joining I_5 and I_6 into I_7).

The *Temporal Pattern-Matching* mechanism, which creates complex temporal pattern intervals, requires pattern classification knowledge about the components of the pattern such as its value and time constraints. Figure 3 presents an example of the pattern-matching process. A worm pattern is created if, during an Internet connection, there is an increase in the number of TCP connection failures together with a high state of system context switches.

All in all, eTIME employs the KBTA method to integrate raw, time-oriented security data from various data sources with knowledge acquired according to the KBTA method for the purpose of deriving meaningful information (i.e., contexts, abstractions and patterns) that can be explored and monitored. The following section describes the eTIME architecture.

4 The eTIME architecture

4.1 Goals

The specific goals of eTIME are to:

(a) Integrate large amounts of security-related data with predefined, context-sensitive, and meaningful patterns of malware.
(b) Implement automated monitoring of continuous, time-oriented data streams by employing real-time, knowledge-based abstractions and creating predefined alerts based on predefined concepts in the knowledge base.
(c) Visually and dynamically explore a security-data repository by means of on-the-fly integration with domain-specific knowledge in order to identify and add new meaningful patterns to the knowledge base.
(d) Implement capabilities for monitoring and querying multiple subjects.
(e) Assist security experts to acquire and maintain temporal-abstraction knowledge.

The expected benefits of the eTIME architecture include: early detection of potentially new malware compatible with predefined temporal patterns; rapid adaptation of the service to new malware through a knowledge acquisition tool; faster identification of new malware patterns by supporting a security officer to visually and interactively explore the contents of a security-data storage; and providing concise, meaningful summaries of large amounts of time-oriented security data in terms familiar to security experts.

4.2 General architecture and modules

eTIME's architecture supports two main modes of operation: (1) an automated, continuous mode for monitoring, recognition and detection of previously unencountered malware instances compatible with known classes of malware; and (2) an interactive, human-mediated mode for dynamically exploring data in a security-related repository in order to identify new temporal patterns that characterize such malware threats.

To facilitate the run-time monitoring and exploration tasks, eTIME has to be initiated. The setup phase involves three essential tasks: (1) a security expert must define the KBTA security ontology; (2) connectivity with the data-sources (static storage and data stream sources) should be established; and (3) a security expert must determine which patterns to monitor and the relevant conditions that should create alerts.

The KBTA-based architecture integrates the following modules and components (Fig. 6):

Data Sources. A set of time-oriented security data sources (these may be static databases or data stream sources).

Knowledge-Bases. A set of one or more knowledge-bases tailored to the process of detecting meaningful temporal patterns of high-level, abstracted concepts (not just raw data).

KB Access Module. An access-point to the temporal-abstraction knowledge-base. This module updates and retrieves elements from the security ontology maintained by the knowledge base.

Temporal-Abstraction Module. A temporal-abstraction, computational process that implements the KBTA's five inference mechanisms and creates abstract temporal patterns such as malware patterns by integrating data and knowledge.

Continuous Monitoring Engine. An automatic mechanism to continuously monitor the output of the temporal-abstraction process and issue alerts whenever a malware pattern is detected.

Data Storage. Stores all raw data records (i.e., primitive parameters and events) and all the temporal abstractions that the temporal-abstraction process produces.

Visual Monitoring Module. A visualization interface that can visually notify a security officer about new alerts issued by the Continuous Monitoring Engine.

Visual Exploration Module. A visualization interface for exploring multiple security-oriented records and their correlations over time. This module also supports an interactive exploratory mode that facilitates identification of new malware.

Query Module. Provides answers to temporal queries submitted by the use of the Visual Exploration Module.

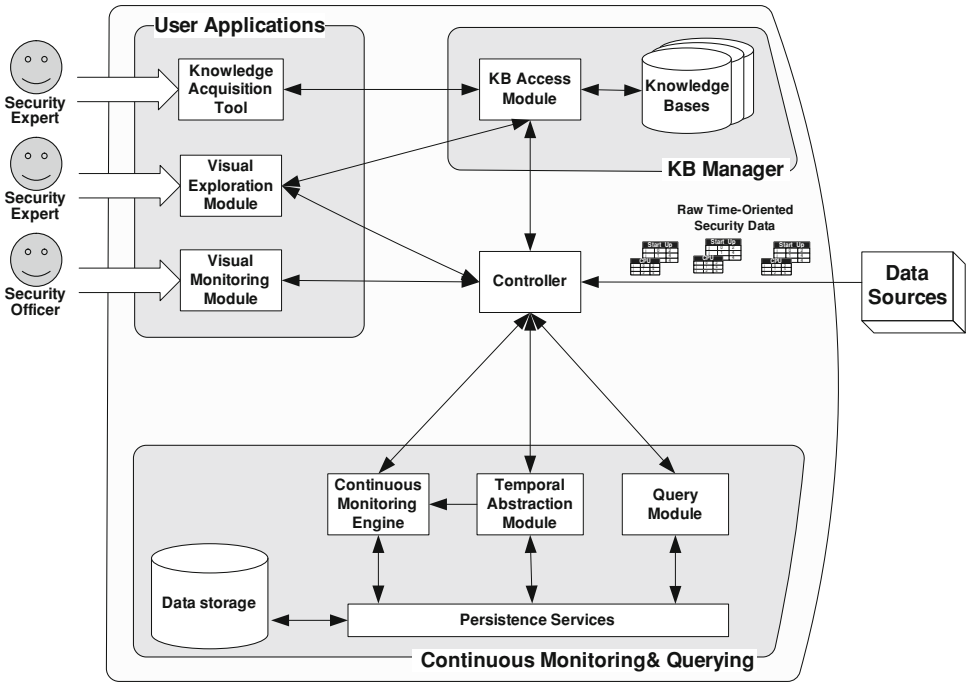

Fig. 6 eTIME general architecture and modules.

Knowledge Acquisition Tool. An application used by a security expert for maintaining the KBTA security ontology (i.e., adding or updating concepts in the ontology).

The aforementioned architecture supports tasks during the setup phase and the run-time as well as in the monitoring and exploration phases. During the setup phase the security expert uses the Knowledge Acquisition Tool to define the security KBTA-based ontology (i.e., all concepts and the relationships between them) which is stored in the Knowledge-Base. The security ontology is then used at run-time by the Temporal-Abstraction Module to continuously create and store abstractions. The output of the Temporal-Abstraction Module is monitored by the Continuous Monitoring Engine. When an important instance of a pre-defined pattern is detected (e.g., worm behavior pattern) an alert is issued [28].

Once an alert has been issued, it is necessary to drill down and explore the data in order to pinpoint the root cause of the alert and to trace the source of the attack for forensic purposes. The exploration task is supported by the Visual Exploration Tool and the Query Module [29].

4.3 Creating and monitoring temporal abstractions

The goal of the Temporal Abstraction Module and the Continuous Monitoring Engine is to provide an integrated environment for continuously abstracting and monitoring temporal security data. Such an environment facilitates the detection of important patterns indicating the existence of a malware and the necessity of notifying the security officer.

Temporal-abstraction monitoring systems usually tend to use a query-response approach in which a query is invoked periodically. For each query, the relevant knowledge and data are first retrieved, and then the abstractions are generated. The query is only executed after these steps have been implemented. This process is computationally intensive since the query re-computes many of the abstractions. Moreover, if a query is not invoked frequently enough, it might not provide effective real-time monitoring.

The periodic querying approach ignores the fact that continuously arriving data usually affects only a few high-level abstractions (temporal redundancy property). This means that only a small portion of the previously created abstractions might be affected by recent raw data. Consequently there is no need to re-compute all the earlier abstractions [30]. The periodic querying approach also ignores the fact that the same data are required for creation of multiple abstractions, which might be used to define an effective way to compute the abstraction.

Based on these observations, it is clear that the periodic query approach is unsuitable for monitoring raw data in the security domain. In the proposed architecture, we implemented an *Incremental Temporal-Abstraction* process, in which the abstraction applies to the newly arrived data by ensuring that previously generated abstractions are updated only when new contradictory data arrived (truth maintenance) and that every generated abstraction is retained until updated or removed by the truth maintenance (persistence maintenance). The incremental approach supports a more efficient monitoring process since most of the abstractions are

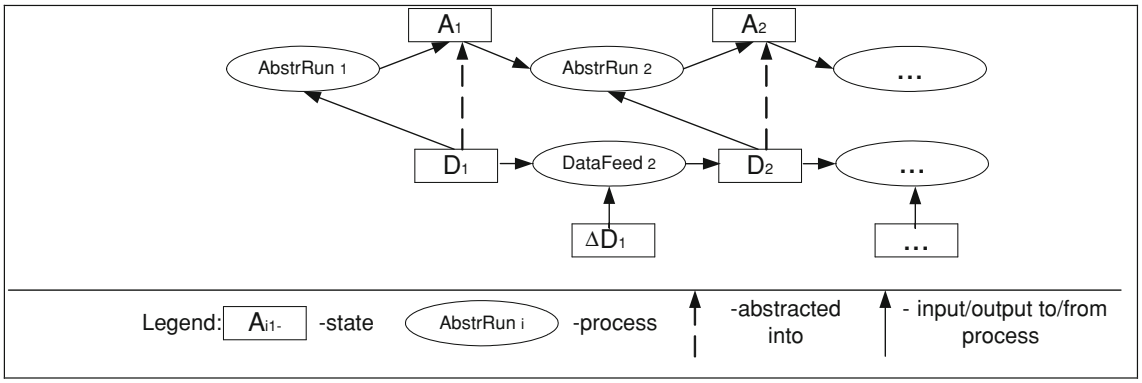

Fig. 7 This schema demonstrates the iterations of the I-KBTA Method. The first iteration of the abstraction process (AbstrRun_1) is initiated by the initial raw data feed D_1. AbstrRun_1 abstracts D_1 into A_1. Consequent iterations of the abstraction process are based on the previously created abstractions

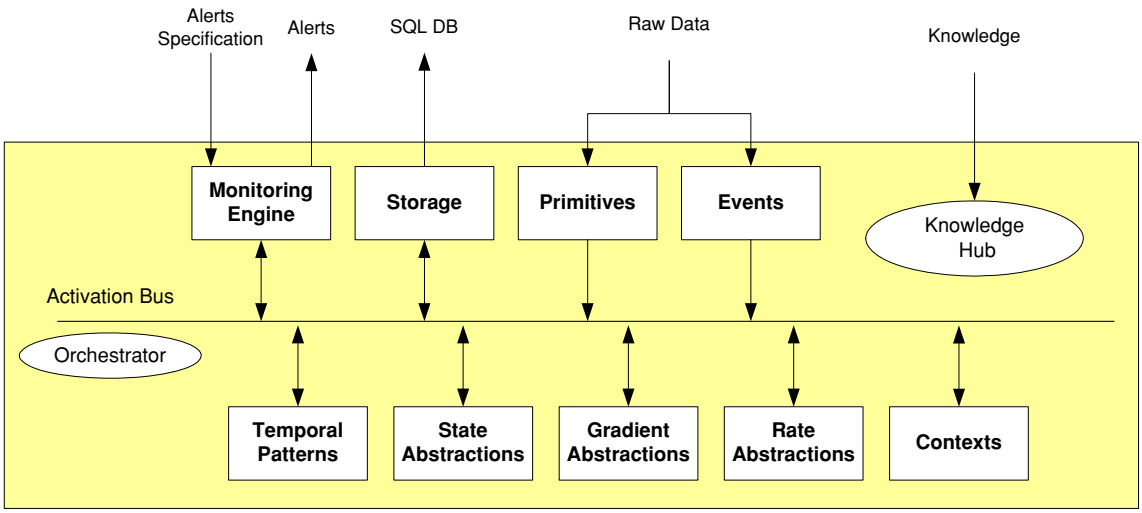

Fig. 8 Internal design of the continuous temporal abstraction and monitoring modules

pre-computed and there is no need to generate abstract concepts on-the-fly.

The incremental temporal abstraction extends the KBTA method to the *Incremental-KBTA* [30] as illustrated in Fig. 7. The first iteration of the abstraction process (AbstrRun_1) is initiated by the initial raw data feed D_1 (i.e., AbstrRun_1 abstracts D_1 into A_1). Consequent iterations of the abstraction process are based on the previously created abstractions; thus, new abstractions are the result of updating abstractions generated in the previous step with new raw data (i.e. AbstrRun_2 abstracts D_2 into A_2 by updating A_1).

I-KBTA supports: (1) incremental computation of temporal abstractions from continuously arriving raw data; (2) adjustability which facilitates extending the computational framework to support new temporal-abstraction tasks not currently defined; and (3) a scalable distributed computing architecture capable of handling a large quantity of data.

Monitoring is applied on temporal abstractions created by the Temporal-Abstraction Module when new raw data is available. The new raw data and abstractions are stored in the data storage, and the new/updated abstractions are sent to the Continuous Monitoring Engine for monitoring. Continuous monitoring specifies monitoring conditions for the attributes of the monitored instances (e.g., value, duration, beginning, ending, type etc.). Monitoring rules can be specified for a group of subjects as well (e.g., more than 20% of infected computers). Alerts are induced by satisfying the monitoring conditions.

As shown in Fig. 8, the computation and monitoring framework was designed to support the specifications above. Each abstraction type (e.g., state, rate, and temporal pattern) is computed (incrementally) by dedicated and distributed computational units that can operate concurrently. The computational units are connected to the bus. Communication between the computational units is managed by an orchestrator while the bus serves as a transport layer. The computational units use the knowledge stored in the knowledge hub, for the abstraction task.

4.4 Query and exploration

The Visual Exploration Module (depicted in Fig. 6) complements the monitoring task by enabling interactive visual exploration of a repository of time-oriented security data. The exploration process is query driven. A user who interactively submits temporal queries can later visualize and explore raw data and abstractions which are received in response.

The derived temporal abstractions can be visualized and explored with a tool especially developed and evaluated for this purpose [31]. Termed VISITORS (Visualization and Exploration of Multiple Time-Oriented Records) the tool can be used by the security expert to submit time-oriented queries based on predefined concepts in the security ontology. The Query Module uses the data storage to answer the query.

VISITORS supports two views of security data. The *individual subject view* is used to explore the data of one subject at a time, such as all worm pattern instances on a specific PC over the previous 10 days. The *aggregated, multiple-subject view* is implemented for various security tasks, such as coping with distributed attacks (i.e., DDoS), when an aggregated, simultaneous view of a group of subjects is more effective than a view of each subject independently. Thus, this view enables visualization and exploration of a group of subjects at various levels of abstraction (i.e., aggregation), by capitalizing on the use of eTIME's temporal abstractions mechanisms. For example, exploration of all PCs on which an installation event has occurred, followed by high outgoing network traffic within five consecutive minutes.

Figures 9, 10 show multiple-subject visualizations. Figure 9 presents a visualization of a single-concept over time. The panel visualizes data of a single raw parameter (CPU usage) over time for a group of computers. The horizontal axis represents the timeline (February 2006) and the vertical axis represents the concept's possible values in terms of percentages. The points' insight and their level of saturation enable the viewer to judge the average amount of items belonging to each area. Minimal values are indicated by the bottom blue line and maximal values of a group by the top red line.

Figure 10 visualizes the distribution of an abstract parameter (TCP Connection Failures STATE) for a group of computers. In this view, the horizontal axis represents the timeline (May 2009) and the vertical axis represents the distribution of the values of the derived parameter. Each value is presented by a color (from bottom to top: "low" blue, "normal" green and "high" red). The user is presented with the proportions of the number of computers in each category for specific time intervals (e.g., on May 3 16:01, 46.67% of the computers had the value "High").

The VISITORS user interface is also based on the security ontology, which improves data exploration and enables navigation of semantically-related raw and abstract concepts. For example, the user can visually explore a virus pattern that the KBTA inference mechanism derived and see all relevant data from which the pattern was derived (assuming, of course, that the virus pattern definition was previously defined by the security expert as a part of the security ontology).

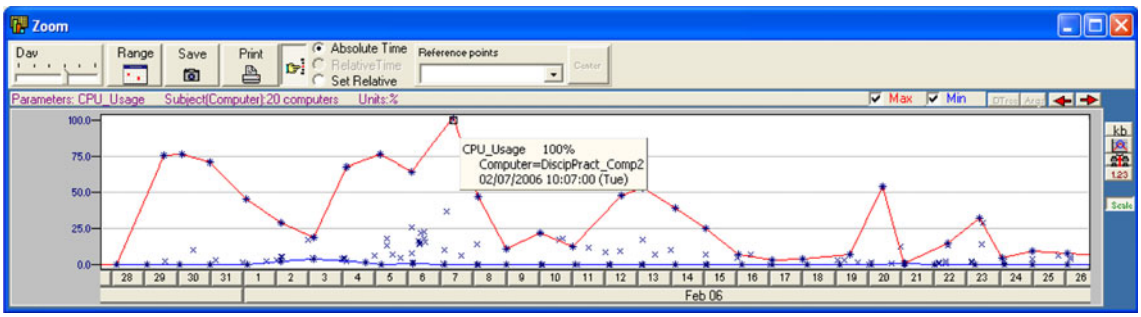

Fig. 9 Visualization of raw parameter for a group of computers

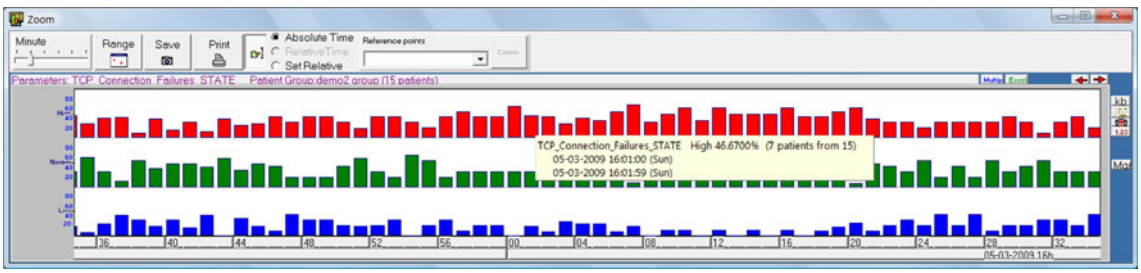

Fig. 10 Visualization of the distribution of abstract parameter

We distinguish between three types of queries. The *Get Subjects* query retrieves the list of subjects from a selected database which satisfies: a set of non-temporal constraints (e.g., IP addresses range); time and value, knowledge-based constraints (define bound constraints on the value, duration and start/end point of a concept as well as definition of time and values pair-wise interrelations between concepts); and statistical constraints (to explore subjects having specific values within a given statistical range of threshold values.) For example a user can construct the following complex query: "Select all the computers and servers in the management department whose CPU usage STATE was abstracted as "Very-High" for more than 80% of the time for at least one hour after Windows patching."

The second type of query is the *Get Temporal Interval*.For example, "Select all time-intervals over which, after an Installation event, a computer was infected by a virus within the subsequent hour." This results in a set of time intervals when certain portions of the subjects have a specific value within a predefined range.

Given a list of subjects and a list of interesting time intervals, the *Get Concept Data* query retrieves the time-oriented data (raw or abstract) of a selected concept. For example, "What was the value of the IP Scan State during January 17–21 for computers #1–#10."

5 eTIME evaluation

To evaluate the proposed eTIME framework we developed an evaluation environment that included: (1) a knowledge acquisition framework consisting of a front-end application used by a domain expert for entering domain knowledge, the knowledge-base and the Knowledge Access Module which is a mediator for accessing the knowledge-base [32]; (2) an Incremental-KBTA framework to provide continuous and incremental computation and monitoring of temporal abstractions; (3) a Visual Exploration tool (VISITORS) [31]; and (4) a Query Module.

eTIME was tested in two scenarios involving two different security ontologies. The first scenario involves collecting various parameters and events from an end-user machine, such as a cellular phone or a PC, and monitoring patterns of malicious behavior. In the second scenario, eTIME was integrated into a network monitoring framework and used parameters and events related to the propagation of files in the network.

5.1 End-user device scenario

eTIME was integrated within a centralized Host-Based Intrusion Detection System (HIDS). Termed NetworkProtect, the system was developed as part of our security research and involves a framework to detect malware on end-user devices by combining distributed monitoring and central analysis capabilities. The framework relies on lightweight agents that continuously sample features from an end-user device and transmit those features to a central server for analysis using various machine learning and temporal reasoning techniques.

The framework comprises three main components: Agent, Server Complex and Control Center (Console). The *agent* constantly operates on end-user devices and logs primitive parameters (e.g., CPU consumption, number of TCP connection failures/sec) and events (e.g., keyboard pressing, email attachment activation, browser opening). The agent must be setup with identification information (in order to be identified by the system) and parameters such as sampling frequency and sampled feature list. Once an agent is activated, following system identification, it continuously samples the various features on the device which are relevant in detecting suspicious behavior.

Logged data are sent via a secure channel to the *server complex*. The server complex runs several processing and detection services that process raw data sampled from the agents, and determine the infection state of an agent. These detection services include classification algorithms (i.e., artificial neural networks, Bayesian networks, and decision trees) as well as the eTIME framework. An evaluation of malicious behavior that was detected using machine learning techniques within the NetworkProtect framework is presented in [33,34].

The *Control Center* (Fig. 11) provides an updated overview of each monitored device by presenting information about particular devices, including their current and historic infection states, as well as aggregated statistics for multiple devices.

The integration of the eTIME framework within the NetworkProtect system is depicted in Fig. 12. The agents that are installed on devices such as PCs, laptops, servers and smartphones, communicate with the External Server which handles registration and logging requests as well as provides detection services. Samples of raw data extracted by the agents are also forwarded by the External Server to eTIME's Temporal Abstraction Computation and Monitoring server which invokes the Incremental-KBTA process. It continuously receives new data records and employs the security ontology (provided by the knowledge-base) in order to create or update the temporal abstractions. Updates or newly created abstractions are then stored in the data storage. Issued alerts detected by the Temporal Abstraction server are sent to the Internal Server which handles all the connections with the NetworkProtect Consoles (Fig. 11).

For our evaluation, we used an agent that was developed for PCs running Windows XP and which monitors every t seconds (according to the agent's settings) various features the Windows Performance application monitors.

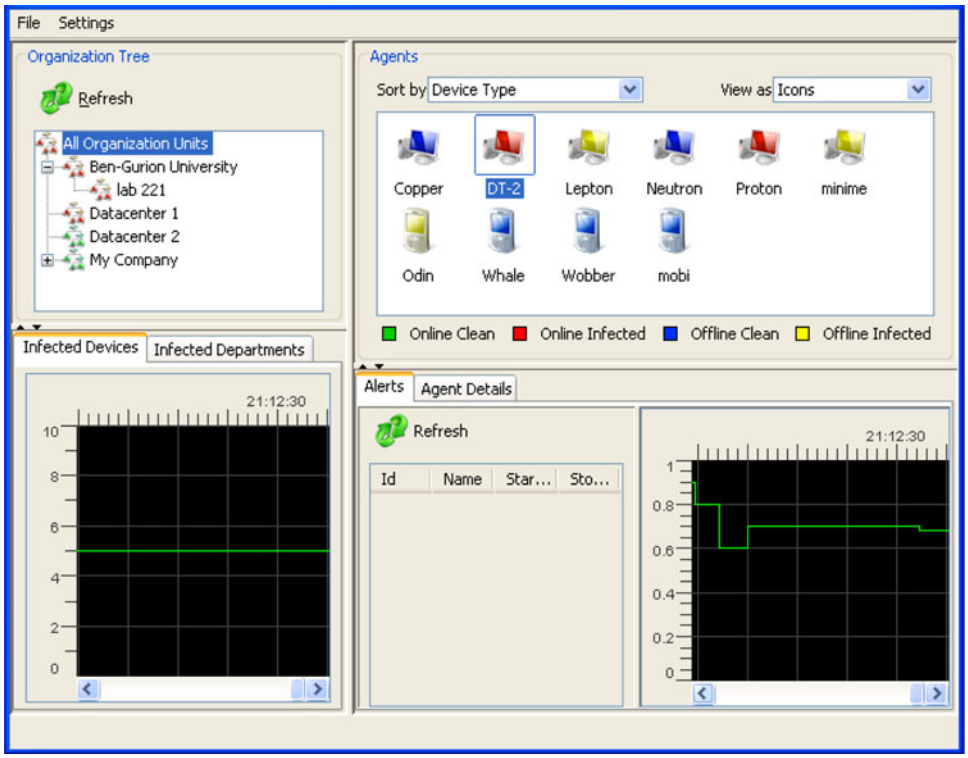

Fig. 11 The NetworkProtect Console for monitoring end-user devices

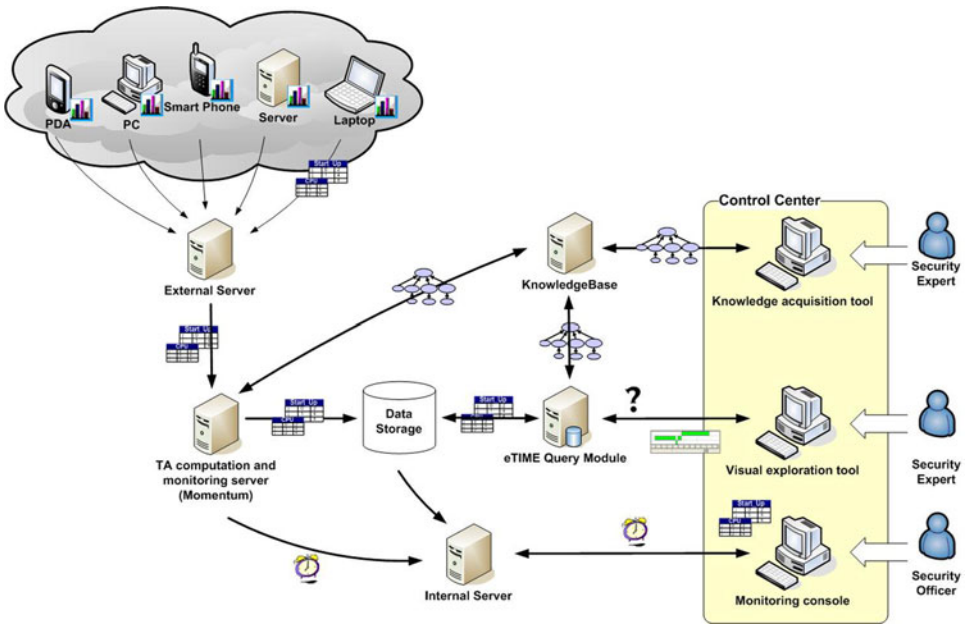

Fig. 12 The integration of eTIME with the NetworkProtect Host-based Intrusion Detection System

The agent monitored more than 100 raw parameters and events, including some that were used to define basic abstractions and complex patterns. The features appeared within the categories: *Internet Control Message Protocol*(*ICMP*), *Internet Protocol* (*IP*), *Memory*, *Network Interface*, *Physical Disk*, *Processes*, *Processor*, *System*, *Transport Control Protocol* (*TCP*), *Threads* and *User Datagram Protocol* (*UDP*). We installed the agent on different interconnected computers, both real and virtual, which were monitored, and later deliberately infected with worms.

The computer network environment we created consisted of seven computers containing heterogenic hardware and a server simulating the Internet. We used two types of computer configurations, both operated using MS Windows XP. The two PC configurations were: Pentium III 800 MHz CPU, bus speed 133 MHz and memory 512 Mb, and Pentium IV 3 GHz CPU, bus speed 800 MHz and a memory of 1 GB. We also connected to this network environment two powerful VMWare servers to create six virtual PCs running with Windows XP, two web servers and two DHCP servers.

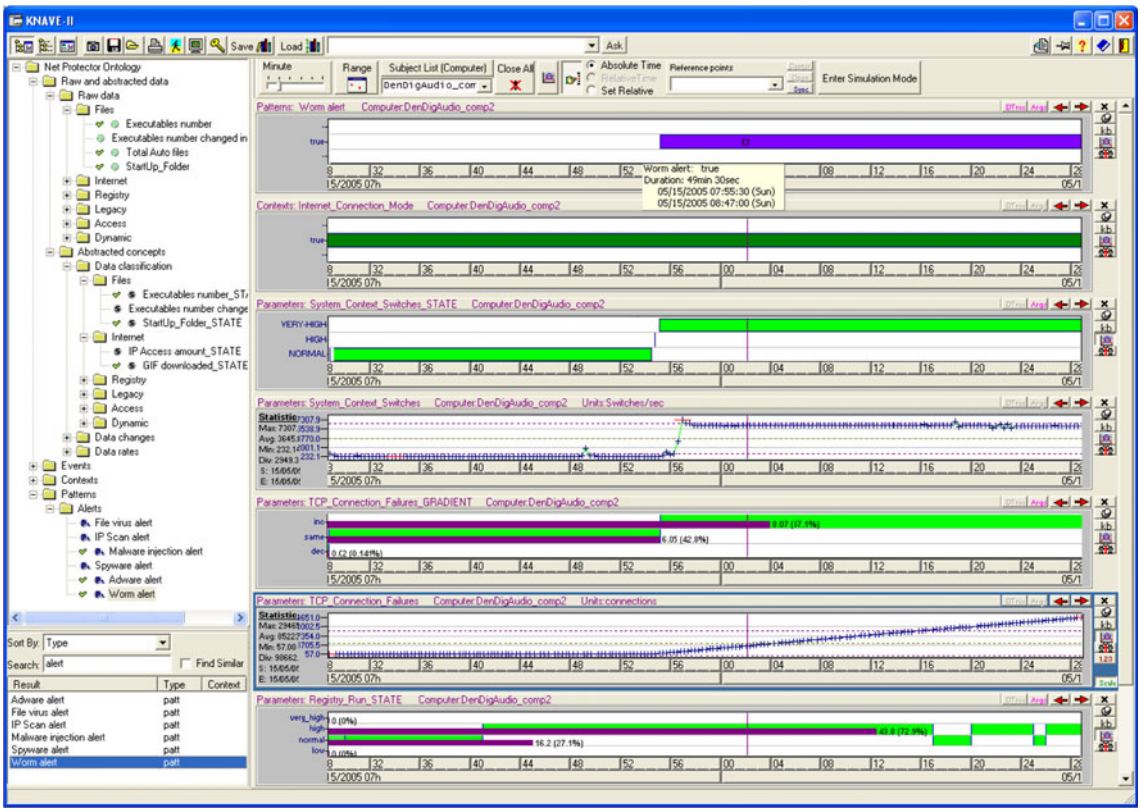

Fig. 13 Exploring the worm pattern data of a single computer in the VISITORS system

We injected worms into the network environment, and then monitored various computer features in each of the infected and uninfected computers. Using virtual PCs enabled us to manually infect a computer and rapidly revert it to its clean state. The real computers were used to validate that the behavior of a malware inside a virtual computer was similar to that on a real one.

The first step in the evaluation was to define and load the security knowledge base according to the KBTA ontology. For the knowledge acquisition task, we used the knowledge acquisition tool. The ontology was stored in the knowledge-base server (Fig. 12). Examples of some of the concepts in the ontology are:

- CPU-usage (primitive parameter measured in % and which can be any value from 0 to 100)
- Software installation (event)
- Software installation context generated by a software installation event; starts when the installation is complete and ends after 5 h
- The TCP connection failures Gradient; derived from the primitive parameter TCP connection failures within the context of the Internet connection mode; it requires additional information such as the mapping function and persistence table.

We activated six different worms: DebormY, DoomjuiceB, PadobotKorgoX, SasserD, DaberA and SlackorA (see Appendix A) on different computers. Using the exploration tool and by examining the measured parameters, we were able to identify a temporal pattern that indicated the existence of a worm. This pattern was previously presented in Fig. 3. By defining this simple worm pattern we were able to detect, at a very early stage, the activation of a worm on a PC running on Windows XP.

Figure 13 depicts the detection of a worm (DabberA) using the exploration interface. The worm was detected (the first graph from the top) by a simple pattern in which, within the Internet-Connection-Mode context, the amount of TCP connection failures increased (a gradient abstraction shown at the second graph from the top). At the same time there was a "Very-High" state of System context switches (a state abstraction shown in the fourth graph from the top). The third and fifth graphs from the top of Fig. 13 depict raw data of TCP connection failures (number of times TCP connections have failed) and the System context switches (number of switches from one thread to another made by the processor per second) measured on the computer. The "TCP connection failure Gradient" abstraction and the "System context switches State" abstraction are the result of the temporal abstraction mechanisms which derived these abstractions from the collected

raw data based on functional classification knowledge, temporal semantic knowledge and temporal interpolation knowledge. The results are time intervals in which the raw data was classified into "Increasing", "Same" or "Decreasing" in the case of the gradient abstraction and "Normal", "High" and "Very-High" in the case of the state abstraction. Zooming into one of the samples opens a tool-tip with additional information. One can observe that the worm was activated at 07:55:30 as the TCP connection failures immediately starts to increase; a few seconds later, the number of system contexts switches becomes very high.

Following is the worm pattern definition in CAPSUL language [27]:

Linear Pattern: Worm behavior pattern
- **Context**: Internet connection mode
- **Linear Components**:
  - **Parameter Component**:
    - TCP connection failure GRADIENT
    - **Abstracted From**:
      - TCP connection failure
    - **Local Constraints**:
      - value = INCREASING
      - duration > 2min
  - **Parameter Component**:
    - System context switches STATE
    - **Abstracted From**:
      - System context switches
    - **Local Constraints**:
      - value = VERY-HIGH
      - duration > 2min
- **Global Constraints**:
  - **Quantitative Gap Constraint**:
    - System context switches STATE ≤ 1min
      AFTER TCP connection failure GRADIENT
  - **Output Value of Pattern**:
    - Value Function: value = Worm Behavior

In order to demonstrate that a single worm pattern can detect different threat instances, we show the results of monitoring the behavior of a PC that was infected with the DabberA (Fig. 14b) and Sasser.C (Fig. 14c). Activating these two worms in our evaluation environment resulted in the derivation of a worm pattern. This required fine tuning of the "TCP Connection Failure Gradient" and the "System Context Switches State" abstractions' mapping functions. It can be seen that the DabberA is more aggressive than Sasser.C; immediately after the activation of the DabberA worm, an increasing "TCP Connection Failure Gradient" and "High System Context Switched State" were derived concurrently (Fig. 14b).

In order to check for false alarms (i.e., detecting the worm pattern in benign activity) we monitored a clean (uninfected) computer while performing various background activities which should affect the "TCP Connection Failures" and the "System Context Switches" parameters. These activities include browsing the Internet and downloading files, using MS messenger and Word and Excel applications. Figure 14a shows the raw and abstract data derived from the uninfected computer for a 15-min period. Since the "TCP Connection Failure Gradient" that was derived was the "Same" (i.e., no significant change), no worm pattern was derived. We simulated the same activities while activating the DabberA and Sasser.C worms and the results indicated that the background activity did not affect the derivation of the worm pattern.

Following is another example of a temporal pattern definition using the KBTA ontology. A malware injection pattern refers to suspicious behavior which is associated with the installation of unwanted, potentially malicious, software on a computer. The pattern is composed of three components: "Software Installation Event", "Auto files number Gradient" and "Executable files number Gradient". Within the Internet connection mode context (e.g., browsing the Internet), if there an increasing amount of Auto files (i.e., the sum of the number of files in the registry_Run and registry_Shell, which are executed on restart by the operating system and the number of files in the StartUp folder), and concurrently there is an increase in the amount of Executable files, without any observed installation process, then the malware injection pattern is invoked.

Linear Pattern: Malware injection pattern
- **Context**: Internet connection mode
- **Linear Components**:
  - **Parameter Component**:
    - Auto files number GRADIENT
    - **Abstracted From**:
      - Auto files number
    - **Local Constraints**:
      - value = INCREASING
      - duration > 1hour
  - **Parameter Component**:
    - Executable files number GRADIENT
    - **Abstracted From**:
      - Executable files number
    - **Local Constraints**:
      - value = INCREASING
      - duration > 1hour
  - **Event Component**:
    - Software installation
    - Local Constraints:
      - NOT EXIST
- Global **Constraints**:
  - **Quantitative Gap Constraint**:
    - Auto files number GRADIENT DURING
      Executable files number GRADIENT
- **Output Value of Pattern**:
  - Value Function: value = Malware Injection

The port scanning pattern indicates that an attacker is possibly using the TCP/IP protocol behavior for revealing open ports on the machine. The pattern is invoked whenever there is an increasing number of "sent RST packets" within the Internet connection mode context.

Linear Pattern: Port scanning pattern
Context: Internet connection mode
Linear Components:
Parameter Component:
IP RST packets GRADIENT
Abstracted From:
IP RST packets
Local Constraints:
value = INCREASING
duration > 5min
Output Value of Pattern:
Value Function: value = Port Scanning

5.2 Monitoring files in network traffic

eTIME was incorporated into a system that monitors parameters and events related to the appearances of files in a network's strategic locations and attempts to detect new instances of malware compatible with a set of predefined malware classes (i.e., in this version of the security ontology, the monitored subjects are files).

Each file extracted from the network's traffic (by a sniffing device) is forwarded to a main processing module with additional meta-data such as the number of times the files were captured in the past, the specific location, the source and destination IP addresses, source and destination ports, etc. These features along with additional information such as the number of times the file has been captured before in the whole network (by all the deployed sniffing devices), and the number of distinct sniffers which captured the file before, are sent to eTIME. eTIME's Incremental Temporal Abstraction Module applies, the pre-defined knowledge on this time-stamped raw data to create alerts whenever a suspicious pattern is identified.

A preliminary ontology included the following patterns: fast spreading worm, DDoS attack, slow spreading worm (possibly infection with zombies) and local scanning worm. For example, the fast spreading worm pattern is extracted when a high number of appearances of a file are detected and the rate of appearance increases very-fast along with a high number of unique destination IPs, where "high number of appearances" and "very-fast increase of appearances" are both high-level concepts from the computer-network's security domain. Following is the fast spreading worm pattern definition in CAPSUL language:

Linear Pattern: Fast spreading Worm
Context: Network capture
Linear Components:
Parameter Component:
Number of appearances STATE
Abstracted From:
Number of appearances
Local Constraints:
value = HIGH
duration > 2min
Linear Components:
Parameter Component:
Number of appearances GRADIENT
Abstracted From:
Number of appearances
Local Constraints:
value = INCREASING
Linear Components:
Parameter Component:
Number of appearances RATE
Abstracted From:
Number of appearances
Local Constraints:
value = VERY-FAST
Linear Components:
Parameter Component:
Number of unique destination IPs STATE
Abstracted From:
Number of unique destination IPs
Local Constraints:
value = HIGH
Global Constraints:
Quantitative Gap Constraint:
Number of appearances STATE DURING Number of appearances GRADIENT
Number of appearances STATE DURING Number of appearances RATE
Number of appearances STATE DURING Number of unique destination IPs STATE
Output Value of Pattern:
Value Function: value = Fast Propagation

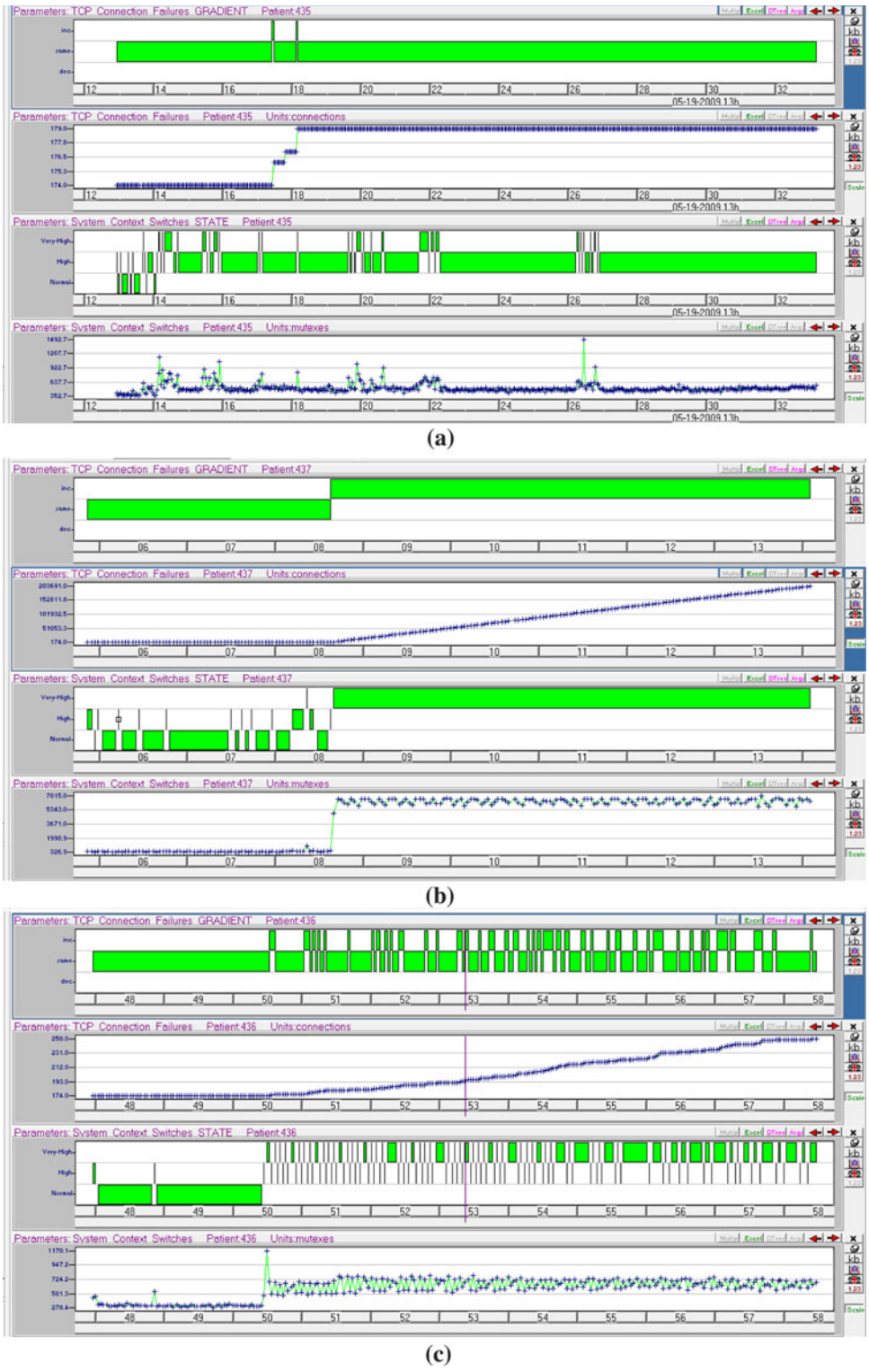

Fig. 14 Exploring the raw and abstract data of uninfected and infected computers, while background user activity is performed, in the VISITORS system. **a** Monitoring an uninfected computer **b** Monitoring a computer infected with DabberA worm. **c** Monitoring a computer infected with Sasser.C worm

In this scenario in order to simulate Internet traffic in a large network with multiple sniffing devices, we integrated the simulation tool[1] presented in [35,36] with the eTIME system. This simulation tool (depicted in Fig. 15) was proposed as an innovative decision support system (DSS) for placement of Intrusion Detection and Prevention Systems in large-scale communication networks. This tool is intended to support network security personnel in optimizing the

[1] http://dtlabs.bgu.ac.il/edare23.

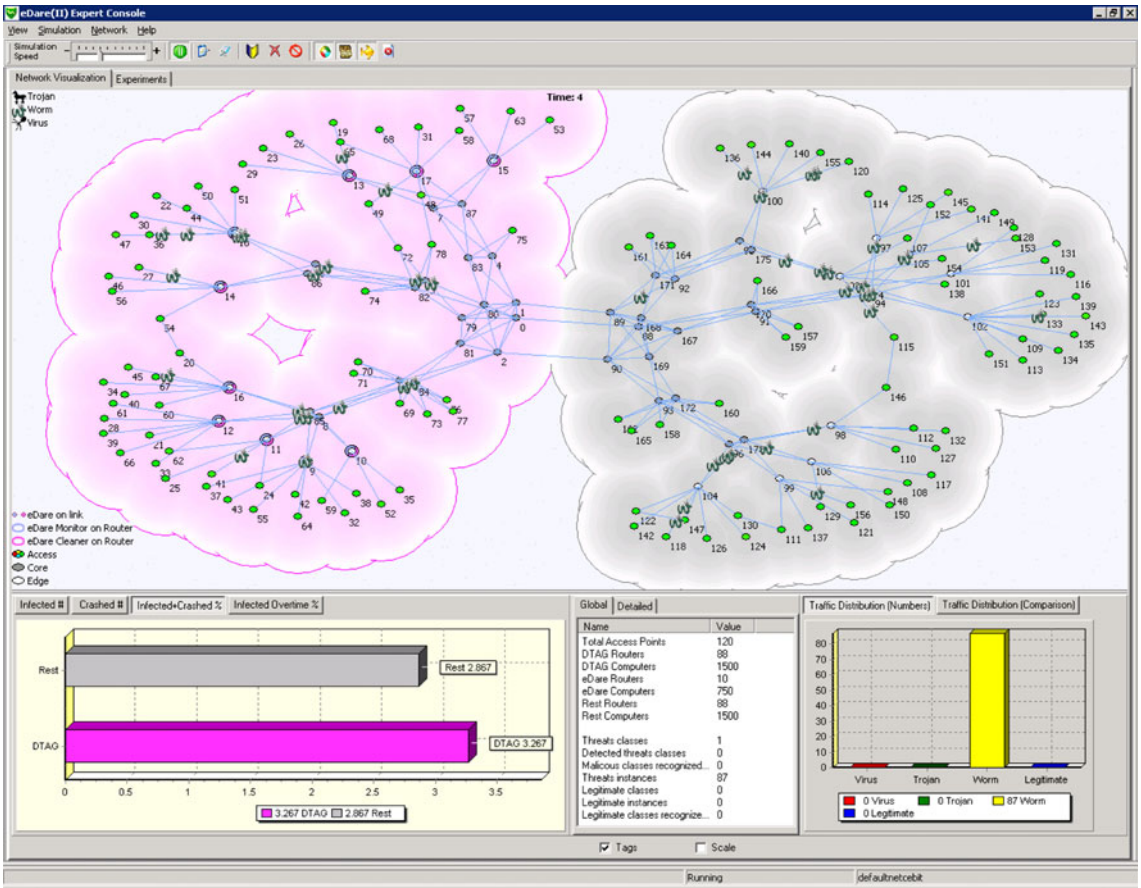

Fig. 15 The network simulation tool for simulating the propagation of various malware or benign files in the network as well as deployed sniffing devices

placement and configuration of malware filtering and monitoring devices within network service provider (NSP) infrastructures and enterprise communication networks. The simulation tool meshes innovative and state-of-the-art elements from graph theory, epidemic modeling, and network simulation in order to define the communication patterns induced by network users (thereby establishing a virtual overlay network). Parallel attack models enable a user to define various interdependent network attacks such as: Internet worms, Trojan horses, DoS attacks, and others. The simulation tool incorporates a set of deployment strategies (employing graph-theoretic centrality measures) in order to intelligently place filtering and monitoring devices. The tool's dedicated network simulator evaluates the various deployments.

The simulation tool can be used to: (1) define networks of any size and type; (2) simulate sniffing devices deployed in the network; and (3) simulate network traffic that may contain both malicious and benign files. When a file passes through a simulated sniffing device, it is sent to a central processing system.

To simulate the propagation of a malware, we used a randomly generated network which contained two NSPs. Every NSP was composed of three POPs (points of presence) with a total of 88 core routers and 60 access routers (20 of which represent corporate customers connected directly to the core) and total of 8,000 machines. Figure 15 presents a screenshot of the network. Since we assumed that only one NSP is protecting its customers (the left NSP in Fig. 15), we deployed 10 monitoring devices to capture files in network traffic for forwarding to the central analysis system.

We simulated the propagation models of: the Slammer worm (Internet worm) using the parameters from [37]; the "Love Letter" (eMail worm) using the parameters from [38] and of the P2P worm using the parameters from [39]. Due to space constraints, we provide only the results of simulating an Internet worm pattern. For simulating an Internet worm (e.g., the Slammer worm) we configured the following parameters:

- scan rate—the number of probes per simulation time unit
- address space—the optional address space
- vulnerable population size.

In Table 2 we present the number of infected computers after 10 simulation time units and the number of worm

Table 2 The infection rate of an Internet worm using the simulation tool

Scan rate (probes/time unit)	5	10
Vulnerable population size (%)		
15	Infected: 95 machines (4%)	Infected: 285 machines (12%)
	Seen by sniffers: 420 instances	Seen by sniffers: 1,200 instances
37	Infected: 420 machines (7%)	Infected: 2,900 machines (49%)
	Seen by sniffers: 1,200 instances	Seen by sniffers: 8,000 instances

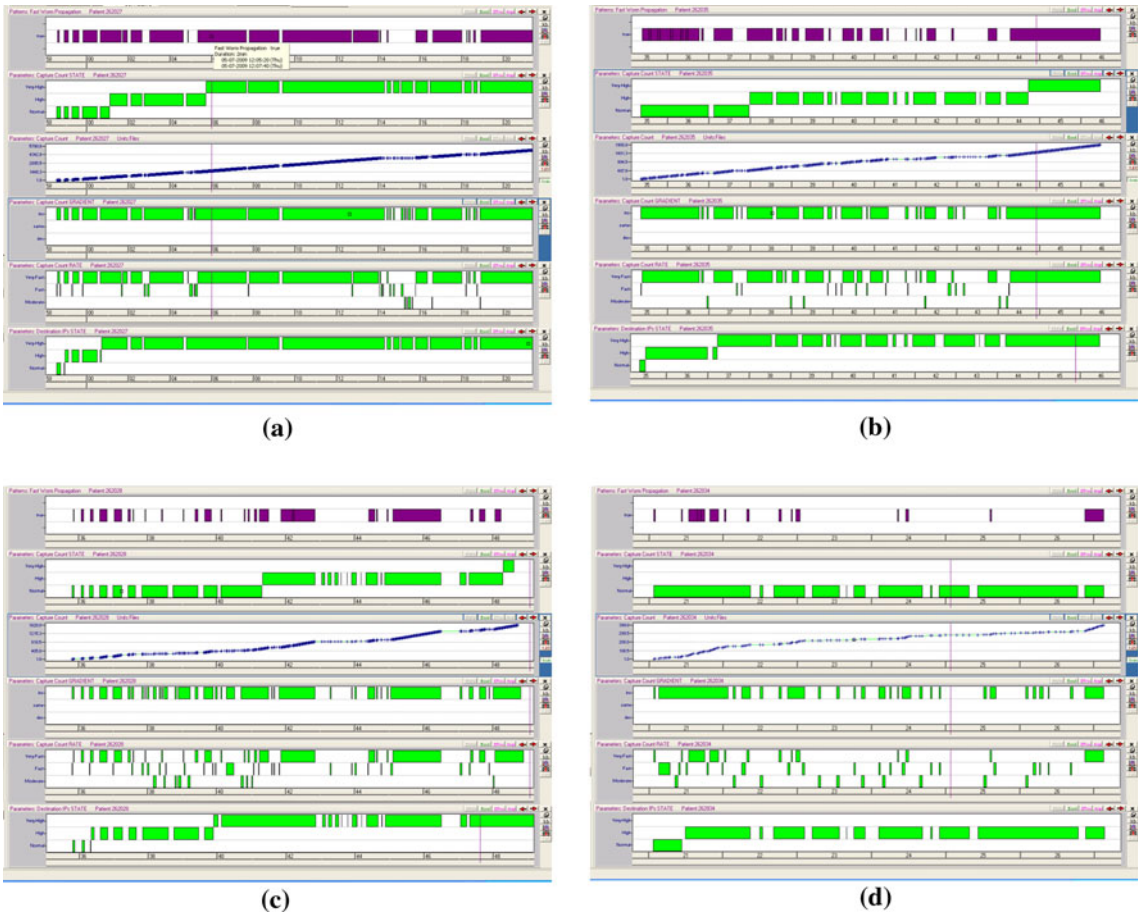

Fig. 16 Exploring the data of four worm scenarios using the query and exploration tool. (**a**) Vulnerable population 37%, scan rate 10 probes; (**b**) vulnerable population 37%, scan rate 5 probes; (**c**) vulnerable population 15%, scan rate 10 probes; (**d**) vulnerable population 15%, scan rate 5 probes

instances that were captured by the 10 sniffing devices deployed on the protected NSP. The scan rate was set at 5 probes or 10 probes, and the vulnerable population size at 15 or 37%. The scanning address space was set at 2^{16}.

Figure 16 depicts the data of a single malicious file propagating in the simulated network for each of the four scenarios presented in Table 2. The data (raw and abstract) is used to derive the worm spreading pattern. The first graph from the top of each scenario depicts the fast worm spreading pattern. The number of appearances (detected by the 10 sniffing devices) is presented in the 3rd graph from the top; the gradient and rate of the number of appearances are shown in the 4th and 5th graphs respectively. The Destination IP gradient is shown in the last graph from the top.

These graphs show the influence of the scan rate and vulnerable population size parameters in deriving the fast-spreading worm pattern. We can see that the vulnerable population size parameter has a higher impact on the pattern detection.

6 Summary and conclusions

This paper presents the KBTA method as a new approach for detecting malware by analyzing temporal security data from

several sources on multiple subjects. The data is analyzed in order to identify new malware compatible with predefined temporal patterns specified at a high level of abstraction by a security expert. The eTIME framework, that implements the KBTA method, is capable of automatically and continuously creating new abstractions from a continuous flow of raw data. These abstractions are automatically monitored to alert the appropriate user (or another process) whenever a suspicious pattern indicate the possible presence of a malware.

The new approach can also quickly adapt to new malware classes through an interactive, knowledge-based visual-exploration tool. This tool enables a security expert to recognize new meaningful temporal patterns, coupled with a knowledge-acquisition tool that enables human experts to modify the malware knowledge base. In addition, the KBTA-based architecture supports acquiring, in a flexible manner, multiple security-related ontologies such as a PC ontology, server ontology, cellular phones/pocket PC ontology, file propagation ontology. Thus, the suggested approach is also useful for detecting malware on a variety of devices and operating systems.

The KBTA method provides concise, meaningful summaries of large amounts of temporal security data in terms familiar to security experts. By supporting automatic monitoring and intelligent, interactive visual exploration, it rapidly identifies new malware patterns. 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.

We implemented a prototype that is capable of creating, monitoring and exploring temporal abstractions and patterns. A central server rather than an end-user device processes the data. Agents installed on end-user devices collected the raw security data. The prototype demonstrated the capability of our method in defining and evaluating new temporal patterns which might indicate the existence of malware such as a worm.

Although we successfully defined additional patterns of other types of malware such as viruses and Trojans we encountered several problems when evaluating the system's effective on these types of malware. First, the patterns that were defined were derived from parameters that are very difficult to extract from the system, such as the number of modified exe/dll files per minute or the number of exe files on the computer. Second, unlike the medical domain, it is very difficult to define abstractions and patterns in the security domain. To solve this difficulty we introduced an automatic temporal data mining (TDM) process on the abstracted data in order to automatically detect new temporal patterns. This approach will assist a security expert in detecting new unknown malware classes and update the security ontology so it can be used for detecting new types of malware.

The automatic mining of patterns can be applied on another version of the agent that was implemented. This type of agent collects measurements and the events of system calls by the Operating System. Since system calls are the primary method for affecting a system, they can be very useful for detecting patterns of unwanted behavior. However, it is a very difficult task for a domain expert to define patterns using these low-level parameters and events and therefore automatic mining of patterns is essential.

In a second scenario, eTIME was tested with an ontology that assists in detecting patterns of worm-related malware propagating in networks.

The eTIME framework 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 identifying time-intervals in which the normal behavior patterns are not derived as proposed in [4]. This scenario was not evaluated. However, as mentioned before, we believe that a temporal data mining process employed on raw and abstracted data will define patterns for both normal and malicious behavior. A parallel research that deals with such Temporal Data Mining framework is being performed and new results will hopefully be presented in the future.

Two additional major advantages of the system that we identified are: (1) the system can assist in summarizing low-level monitored parameters that arrive at high rate by applying simple abstractions (State, Rate, and Gradient) and (2) the system can help in integrating alerts from other sensors as primitive parameters and "smoothing" the alerts by applying the temporal-abstraction process and notifying the user only if the alert instances persist. Meshing alerts from multiple sensors reduces the amounts of false alarms that a system administrator needs to cope with those alerts that persist for a substantial time interval.

Acknowledgments This research is supported by Deutsche Telecom AG.

Appendix A: Worms description

1. W32.Deborm.Y (DebormY)

This worm scans the local network and tries to propagate to other computers on the local network. It attempts to share C$ (C drive) using the accounts of the administrator, owner or guest (it succeeds if a certain account does not have a password).

2. W32.HLLW.Doomjuice.B (DoomJuiceB)

This worm randomly generates IP addresses and attempts to propagate to computers by using the backdoor opened by the worm W32.Mydoom.A@mm. It tries to connect to computers using TCP port 3127. If the connection is established it uses the backdoor to infect the computer. It is programmed to add itself to the registry so that it is loaded on startup.

3. W32.Korgo.X (PadobotKorgoX)

This worm generates random IP addresses and exploits the LSASS Buffer overrun vulnerability using TCP port 445. If it succeeds in taking over a computer, the newly infected computer will send a request for downloading the worm from the infecting computer by using a random TCP port.

4. W32.Sasser.D (Sasser.C)

This worm spreads by generating random IP addresses using 128 threads. The IP addresses are generated so that 48% of them should be close to the current computer by using the current computer's IP and 52% of them are generated completely at random. It connects to the remote computer using TCP port 445. If the connection is established, a remote shell is opened. The remote shell is used to connect to the infected computer's FTP server and transfer the worm.

5. Daber.A (Daber.A)

This worm scans networks for random IP addresses, searching for victim computers that have the ftp component of the Sasser worm installed on port 5554. When the worm finds a suitable victim machine, it sends a vulnerability exploit to it to infect the system. It then launches the command shell on port 8967. It also installs a backdoor on port 9898 to receive external commands.

6. Slackor.A (Slackor.A)

When the Slackor worm is run, it sends a SYN TCP packet to randomly generate IP addresses through port 445 to search for the systems using Server Message Block (SMB). It then attempts to connect to the Windows default shares on these systems by using the username and password pair that it carries. If successful, it tries to copy the worm to the system.

References

1. Kienzle, D.M., Elder, M.C.: Internet WORMS: past, present, and future: recent worms: a survey and trends. In: Proceedings of the ACM Workshop on Rapid Malcode (2003)
2. Heidari, M.: Malicious codes in depth. Security docs. http://www.securitydocs.com/pdf/2742.PDF (2004)
3. Dikinson, J.: The new anti-virus formula. 2005. http://www.ironport.com/pdf/ironport_new_anti-virus_formula.pdf
4. Seleznyov, A., Mazhelis, O.: Learning temporal patterns for anomaly intrusion detection. In: Proceedings of the 17th ACM Symposium on Applied Computing (2002)
5. Ye, N.: A Markov chain of temporal behavior for anomaly detection. In: Workshop on Information Assurance and Security (2000)
6. Shahar, Y.: A framework for knowledge-based temporal abstraction. Artif. Intell. **90**(1–2), 79–133 (1997)
7. Shahar, Y., Musen, M.A.: Knowledge-based temporal abstraction in clinical domains. Artif. Intell. Med. **8**(3), 267–298 (1996)
8. Jones A.K., Sielken R.S.: Computer System Intrusion Detection: A Survey. Technical Report, Computer Science Department, University of Virginia, USA (2000)
9. Axelsson, S.: Intrusion Detection Systems: A Survey and Taxonomy. Technical Report, Department of Computer Engineering, Chalmers University, Sweden (2000)
10. Christodorescu, M., Jha, S.: Testing malware detectors. ACM SIGSOFT Softw. Eng. Notes **29**(4), 34–44 (2004)
11. Jacob, G., Debar, H., Filiol, E.: Behavioral detection of malware: from a survey towards an established taxonomy. J. Comput. Virol. **4**, 251–266 (2008)
12. Idika, N., Mathur, A.P.: A Survey of Malware Detection Techniques. Technical Report, Department of Computer Science, Purdue University, USA (2007)
13. Estevez-Tapiador, J.M. et al.: Anomaly detection methods in wired networks: a survey and taxonomy. Comput. Commun. **27**(16), 1569–1584 (2004)
14. Moskovitch, R., Elovici, Y., Rokach, L.: Detection of unknown computer worms based on behavioral classification of the host. Comput. Stat. Data. Anal. **52**(9), 4544–4566 (2008)
15. Lane, T., Brodley, C.E.: Temporal sequence learning and data reduction for anomaly detection. ACM Trans. Inf. Syst. Secur. **2**(3), 295–331 (1999)
16. Ghosh, A.K., Schwartzbard, A., Schatz, M.: Using program behavior profiles for intrusion detection. In: Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring (1999)
17. Naldurg, P. et al.: A temporal logic based framework for intrusion detection. In: Proceedings of the 24th Formal Techniques for Networked and Distributed Systems International Conference (2004)
18. Ning, P., Jajodia, S., Wang, X.S.: Abstraction-based intrusion detection in distributed environments. ACM Trans. Inf. Syst. Secur. **4**(4), 407–452 (2001)
19. Kohout, L.J., Yasinsac, A., McDuffie, E.: Activity profiles for intrusion detection. In: North American Fuzzy Information Processing Society-Fuzzy Logic and the Internet (2002)
20. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
21. Li, Y. et al.: Enhancing profiles for anomaly detection using time granularities. J Comput. Secur. **10**(1–2), 137–157 (2002)
22. Talbi, M., Mejry, M., Bouhoula, A.: Specification and evaluation of polymorphic shellcode properties using a new temporal logic. J. Comput. Virol. (2008)
23. Morin, B., Debar, H.: Correlation of intrusion symptoms: an application of chronicles. In: Proceedings Recent Advances in Intrusion Detection (RAID) Symposium (2003)
24. Cuppens, F., Miege, A.: Alert correlation in a cooperative intrusion detection framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy (2002)
25. Eckmann, S.T., Vigna, G., Kemmerer, R.A.: STATL: an attack language for state-based intrusion detection. J. Comput. Secur. **10**(1), 71–104 (2002)

26. Lindqvist, U., Porras, P.A.: Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In: Proceedings of the 1999 IEEE Symposium on Security and Privacy (1999)
27. Chakravarty, S., Shahar, Y.: CAPSUL: a constraint-based specification of repeating patterns in time-oriented data. Ann. Math. AI **30**(1–4), 3–22 (2000)
28. Shabtai, A., Shahar, Y., Elovici, Y.: Monitoring for malware using a temporal-abstraction knowledge base. In: Proceedings of the 8th International Symposium on System and Information Security (2006)
29. Shabtai, A., Shahar, Y., Elovici, Y.: Using the knowledge-based temporal-abstraction (KBTA) method for detection of electronic threats. In: Proceedings of the 5th European Conference on Information Warfare and Security (2006)
30. Spokoiny, A., Shahar, Y.: An active database architecture for knowledge-based incremental abstraction of complex concepts from continuously arriving time-oriented raw data. J. Intell. Inf. Syst. **28**(3), 199–231 (2007)
31. Shabtai A., Klimov D., Shahar Y., Elovici Y.: An intelligent, interactive tool for exploration and visualization of time-oriented security data. In: Proceedings of the 3rd International Workshop on Visualization for Computer Security (2006)
32. Shabtai, A., Atlas, M., Shahar, Y., Elovici, Y.: Evaluation of a temporal-abstraction knowledge acquisition tool in the network security domain. In: Proceedings of the 4th International Conference on Knowledge Capture (2007)
33. Stopel, D., Moskovitch, R., Boger, Z., Shahar, Y., Elovici, Y.: Using artificial neural networks to detect unknown computer worms. J. Neural Comput. Appl. (2009)
34. Moskovitch, R., et al.: Host based intrusion detection using machine learning. IEEE Inf. Secur. Inf. (2007)
35. Puzis, R., Tubi, M., Elovici, Y., Glezer, C.: A decision support system for placement of intrusion detection and prevention devices in large-scale networks. Submitted to ACM Transactions on Information and System Security (TISSEC)
36. Tubi, M., Puzis, R., Elovici, Y.: Deployment of DNIDS in social networks. ISI (2007)
37. Moore, D. et al.: Inside the slammer worm. IEEE Secur. Priv. (2003)
38. CERT 2000. Love letter worm. http://www.cert.org/advisories/CA-2000-04.html
39. Thommes, R., Coates, M.: Epidemiological modeling of peer-to-peer viruses and pollution. In: Proceedings of IEEE Infocom (2006)