Journal of Big Data

RESEARCH Open Access

Big knowledge-based semantic correlation for detecting slow and low-level advanced persistent threats

Amir Mohammadzade Lajevardi[1*] and Morteza Amini[2]

*Correspondence: lajevardi@ce.sharif.edu
[1] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
Full list of author information is available at the end of the article

Abstract

Targeted cyber attacks, which today are known as Advanced Persistent Threats (APTs), use low and slow patterns to bypass intrusion detection and alert correlation systems. Since most of the attack detection approaches use a short time-window, the slow APTs abuse this weakness to escape from the detection systems. In these situations, the intruders increase the time of attacks and move as slowly as possible by some tricks such as using *sleeper* and *wake up* functions and make detection difficult for such detection systems. In addition, low APTs use trusted subjects or agents to conceal any footprint and abnormalities in the victim system by some tricks such as code injection and stealing digital certificates. In this paper, a new solution is proposed for detecting both low and slow APTs. The proposed approach uses low-level interception, knowledge-based system, system ontology, and semantic correlation to detect low-level attacks. Since using semantic-based correlation is not applicable for detecting slow attacks due to its significant processing overhead, we propose a scalable knowledge-based system that uses three different concepts and approaches to reduce the time complexity including (1) flexible sliding window called *Vermiform window* to analyze and correlate system events instead of using fixed-size time-window, (2) effective inference using a scalable inference engine called *SANSA*, and (3) data reduction by ontology-based data abstraction. We can detect the slow APTs whose attack duration is about several months. Evaluation of the proposed approach on a dataset containing many APT scenarios shows 84.21% of sensitivity and 82.16% of specificity.

Keywords: Advanced persistent threat, Big semantic correlation, Ontology, Intrusion detection

Introduction

Since the term "Advanced Persistent Threat" was coined by United States Air Force (USAF) [1–3] in 2006, various general definitions [3–14] are proposed to describe the advanced persistent threat (*APT*) attacks which are often far from real APT scenarios such as Stuxnet, Flame, Project Sauron, Shamoon, and WannaCry [15]. According to our survey on the behavior and anatomy of nearly 70 real APTs, which are reported by Kaspersky Targeted Cyberattacks Logbook [15], these attacks uses low and slow patterns that make them difficult to be detected. **Low** APTs use trusted agents by some

tricks such as code injection to do any malicious activities through a trusted process, and conceal any footprint and abnormalities. To detect such malicious activities it is necessary to use fine-grained event interception in the endpoint systems. Since fine-grained interception leads to huge numbers of events, detection approaches use short time-window to correlate the events. **Slow** APTs use some tricks to increase the time of attacks and escape from short time-window of intrusion detection and alert correlation systems. Hence, security information and event management approaches which collect and correlate the logs and alerts generated by different tools (e.g., antivirus, firewall, UTM, network intrusion detection system, network device, and operating system) are vulnerable against the APT attacks for the following reasons:

- Due to the processing limitation, the generated and correlated logs are mostly coarse-grained. The coarse-grained logs lead to data loss and lack of precise correlation between the low-level operating system events with the network events and rebuilding the attack vectors.
- Due to the processing limitation, the available solutions use short-time windows for correlating the alerts, and hence they are vulnerable to detect slow attacks and long-term attack vectors.

As a result, the purpose of this paper is to solve these problems in practice by proposing a big Knowledge-based semantic correlation engine for detecting slow and low-level APTs, which are the most sophisticated APTs. To this aim, we enhance our previous solution proposed in [16] to detect slow APTs, other than low-level and hybrid APTs. Therefore, the contributions of our proposed approach in this paper are as follows:

- Since, detecting low-level APT attacks needs processing large event logs, and detecting slow APTs makes the processing problem much harder, one of our contributions is to propose an approach to detect both low-level and slow APT attacks.
- Using a long sliding window for detecting the slow APTs. We propose a *Vermiform* sliding window to analyze and correlate system events instead of using a fixed-size time-window.
- Using Scalable Semantic Analytics Stack (*SANSA*) [17] as a big inference engine based on *Spark* for scalable semantic correlation.
- Although SANSA is a good inference engine for processing huge number of events, its processing power is limited. We use *event abstraction* concept to reduce the number of events, to speed up the inference time, and to detect the very slow APTs (whose attack duration is several years instead of several months). By abstracting the old events, we consider them as a history in the detection process instead of being disposed by the movement of the timing window.

The rest of the paper is organized as follows. "Preliminaries" section describes the necessary preliminaries which are used in the paper. The characteristics of APTs, related works, and the formal definition of the problem are described in "Background and problem statement" section. The proposed approach is discussed in "Proposed

approach" section. Evaluation and result analysis are reported in "Evaluation" section. Finally, the paper is concluded in "Conclusion" section.

Preliminaries

In this section, we define the basic terms and concepts that are used in other sections of this paper. The most basic terms of this section are retrieved from our previous work [16], which proposes an approach to detect hybrid and low-level APTs. The summary of the defined symbols in this paper is presented in "List of the symbols used in the paper" section.

Since the proposed approach in this paper employs description logic [18], and Ontology web language- description logic *(OWL-DL)*, we have defined the syntax and semantics of part of description logic for the readers who are not familiar with these concepts in "Syntax and semantics of description logic" section.

Definition 1 An **event** occurs when a subject acts on an object in a specific time or period [16].

More formally, event $e_i \in Event^{\mathcal{I}}$ is defined by a quadruple as follows:

$$\begin{aligned} \forall e_i \in Event^{\mathcal{I}}, e_i = \langle s_i, o_i, a_i, t_i \rangle, \\ & s_i \in Subject^{\mathcal{I}}, \\ & o_i \in Object^{\mathcal{I}}, \\ & a_i \in Action^{\mathcal{I}}, \\ & t_i \in \mathbb{N}, \end{aligned} \tag{1}$$

where s_i is a subject such as a user or a process, o_i is an object such as a socket or file, a_i is an action such as reading (R) or writing (W), and t_i is the timestamp of the event occurrence.

Since in this paper *event* is considered as a concept in system ontology and the languages provided for ontology specification allow only using unary and binary predicates, we should specify the properties of an event e_i with four binary relations as follows [16].

$$e_i = \langle s_i, o_i, a_i, t_i \rangle \Longleftrightarrow \langle e_i, s_i \rangle, \langle e_i, o_i \rangle, \langle e_i, a_i \rangle, \langle e_i, t_i \rangle.$$

In the rest of the paper, for the sake of simplicity, we define the event and its properties as a quadruple.

According to the ontology specified in the section "Semantic correlation", concept *Subject* includes four subject types *Thread*, *Process*, *User*, and *Host* as follows:

$$Subject = Host \sqcup User \sqcup Process \sqcup Thread. \tag{2}$$

Also, function $time : Event^{\mathcal{I}} \longrightarrow \mathbb{N}$ specifies the timestamp of an event, and is defined as follows:

$$\forall\, e_i \in Event^{\mathcal{I}},\ e_i = \langle s_i, o_i, a_i, t_i \rangle \rightarrow time(e_i) = t_i. \tag{3}$$

Similarly functions *subject*, *object*, and *action* specify the subject, the object, and the action of an event respectively as follows:

$$\begin{aligned} &\forall\, e_i \in Event^{\mathcal{I}},\ e_i = \langle s_i, o_i, a_i, t_i \rangle \\ &\quad \rightarrow subject(e_i) = s_i, object(e_i) = o_i, action(e_i) = a_i. \end{aligned} \tag{4}$$

Definition 2 Frame $f : Event \times \mathbb{N} \rightarrow \mathbb{N}$ specifies the number of events in a specific event set which have a specific timestamp.

In other words, $f(E_i, t_i) = n_i$ means, the number of events in E_i where $e_i \in E_i \wedge time(e_i) = t_i$ is equal to n_i.

For example, if $E_i = \{\langle s_1, o_1, a_1, t_1 \rangle, \langle s_2, o_2, a_2, t_2 \rangle, \langle s_3, o_3, a_3, t_1 \rangle\}$ then $f(E_i, t_1) = 2$ and $f(E_i, t_2) = 1$.

Definition 3 **Two events e_i and e_j are related to each other** and denoted by $e_i \sim e_j$ if there are specific relations between their properties and $t_i \leq t_j$ [16]. This relation can be modeled by a directed acyclic graph, which is shown in Fig. 1.

To detect malicious activities, it is necessary to define the system security policy. The security policy is defined as follows.

Definition 4 (*Security policy* [16]) Security policy (*SP*) is defined as $SP \subseteq Subject^{\mathcal{I}} \times Object^{\mathcal{I}} \times Action^{\mathcal{I}}$, which determines the set of all unauthorized events in the system. Any policy rule $p_i = \langle s_i, o_i, a_i \rangle$ in *SP* shows that subject s_i is not authorized to do action a_i on object o_i at any time.

Definition 5 (*Explicit violation* [16]) The occurrence of an event set *ES* ($ES \subseteq Event^{\mathcal{I}}$) in a system causes the **explicit violation of security** with regard to security policy *SP*, if and only if,

$$\begin{aligned} &\exists e_i, p_j, e_i \in ES \wedge p_j \in SP \wedge subject(e_i) = subject(p_j) \wedge \\ &\quad object(e_i) = object(p_j) \wedge action(e_i) = action(p_j). \end{aligned} \tag{5}$$

Definition 6 (*Implicit violation* [16]) The occurrence of an event set *ES* ($ES \subseteq Event^{\mathcal{I}}$) in a system causes the **implicit violation of security** with regard to security policy *SP*, if and only if,

$$\begin{aligned} &\exists e_i, p_j, e_i \in I(ES) \wedge p_j \in SP \wedge subject(e_i) = subject(p_j) \\ &\quad \wedge object(e_i) = object(p_j) \wedge action(e_i) = action(p_j) \wedge \\ &\qquad \nexists e_k, e_k \in ES \wedge subject(e_k) = subject(e_i) \wedge \\ &\quad object(e_k) = object(e_i) \wedge action(e_k) = action(e_i), \end{aligned} \tag{6}$$

where function $I : \mathcal{P}(Event^{\mathcal{I}}) \longrightarrow \mathcal{P}(Event^{\mathcal{I}})$ specifies the set of all events that are occurred implicitly following the execution of another event set. As an example, for

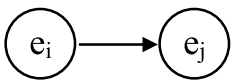

Fig. 1 Event relation $e_i \sim e_j$

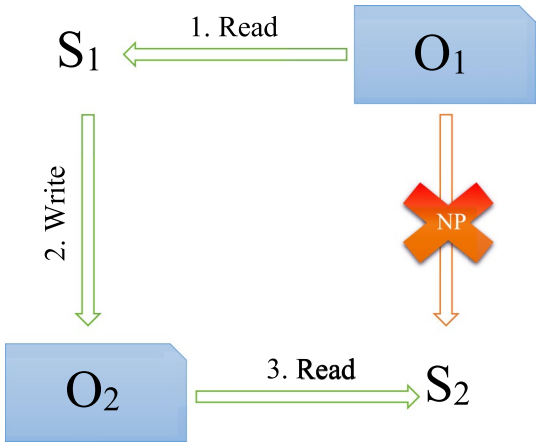

Fig. 2 Schematic description for an implicit violation of security policy example [16]

two untrusted subjects s_1, s_2, if $ES := \{\langle s_1, o_1, r, t_1\rangle, \langle s_1, o_2, W, t_2\rangle, \langle s_2, o_2, r, t_3\rangle\}$ and $SP := \{\langle s_2, o_1, r\rangle\}$ then the violated policy is $I(ES) := \{\langle s_2, o_1, r, t_3\rangle\}$. The schematic description of this example is shown in Figure 2. As shown in this figure, the occurrence of a sequence of events (ES) cause to object o_1 be read by subject s_2 indirectly. In other words, event set ES cause event $e_k = \langle s_2, o_1, r, t_3\rangle$ occur implicitly. The process of indirect access detection is discussed in the proposed approach in "Expanding: Knowledge-based Inference" section.

Definition 7 (*Attack vector*) An attack vector v_i is a set of events ($v_i \subseteq Event^{\mathcal{I}}$) that has the following three characteristics:

- Malicious: An attack vector v_i is malicious if it violates the security policies implicitly or explicitly.
- Minimal: An attack vector v_i is minimal if the exclusion of any event e_i from v_i reduces the maliciousness of v_i [16]. Suppose that function $\zeta : Event^{\mathcal{I}} \rightarrow \mathbb{N}$ shows the value of maliciousness of an event set, then $E_i \subseteq Event^{\mathcal{I}}$ is more malicious than $E_j \subseteq Event^{\mathcal{I}}$, if and only if, $\zeta(E_i) > \zeta(E_j)$ [16]. In other words the minimality means:

$$\forall e_i \in v_i, \zeta(v_i - \{e_i\}) < \zeta(v_i). \tag{7}$$

- Connected: An attack vector v_i is connected, if the relations between all the events of v_i construct a connected directed acyclic graph. In other words:

$$\begin{aligned} &(v_i, \sim) \textit{ is Partial Order} \wedge \nexists v_1, v_2, v_1 \subseteq v_i \wedge \\ &v_2 \subseteq v_i \wedge v_i = [v_1 \cup v_2] \ \wedge \ [v_1 \cap v_2] = \varnothing \wedge \\ &\nexists e_1, e_2, e_1 \in v_1 \wedge e_2 \in v_2 \wedge (e_1 \sim e_2 \vee e_2 \sim e_1). \end{aligned} \tag{8}$$

Set v is defined as the set of all attack vectors.

Background and problem statement

As discussed in the introduction, there are various definitions to describe the APTs. In this section, the characteristics of the APTs and the problem to be solved in this paper are defined.

APT characteristics

According to our survey on the behavior and anatomy of nearly 70 real APTs, which are reported by Kaspersky Targeted Cyberattacks Logbook [15], the APTs can be defined by the following characteristics:

Special-purpose: Since the intruders have sensitive information about the victim's infrastructure, the behaviors of APT attacks are somewhat intelligent. This characteristic means that an APT that is malicious in one infrastructure might be completely benign in another. For example, Stuxnet [19] is an instance of a special-purpose APT, which is malicious after satisfying certain conditions in the victim's infrastructure (e.g., detecting special patterns in centrifuge falls of victim's industrial infrastructure), but it is nearly benign in the system of a normal user.

Slow: Since existing security mechanisms use short time-windows (about a few minutes), some APTs (e.g., ProjectSauron APT [20]) abuse this weakness to bypass the detection methods. In this case, the intruders take advantage of some tricks such as using *wake-up* and *sleep* functions to distribute their attack vectors in several time-windows (about several months). Note that, in real conditions, the attack duration cannot last very long (e.g., several years); because the software migration in the victim's infrastructure can cause the attack to fail.

Low-level: In low-level APTs, the explicit violation of security policy is not probable and the attacker usually violates the security policy implicitly by some methods, including:

- Using trusted events and agents to perform malicious activities: this method takes the advantages of some techniques such as malicious code injection into trusted applications (e.g., Gauss APT [15]), or using stolen digital certificates (e.g., Stuxnet APT [15]), or using genuine recognized removable media to bypass the data loss prevention (DLP) system (e.g., Project Sauron APT [20]), and human errors to infiltrate the victim's system.
- Performing the malicious actions gradually: some APTs (e.g., Carbanak APT [15]), especially the malware that use data exfiltration, steal the sensitive data gradually to hide from intrusion and anomaly detection systems. For example, to exfiltrate 1 GB of data from the victim's system, the malware breaks the data into several tiny parts (e.g., less than 1 MB) and exfiltrates them slowly in several days.

Multi-step: In multi-step APTs (e.g., Flame APT [15]), the attack vector is divided into several steps, and activation of each step depends on the success of the previous steps. In these cases, the main challenge is detecting the relations of the steps and constructing the primary attack vector.

Distribution: In such threats, the intruders distribute the malware attack vectors in several sub-vectors and sub-vectors are executed by different subjects (e.g., different

processes, and in some cases by different hosts). In such cases, communications between malware subjects are established through inter-process communication (IPC). Also, these malwares try to obfuscate the dependencies between the sub-vectors using fake and unrelated events within actual events. The main challenge is to identify the actual events semantically, remove the fake events, and summarize the behavior.

Hybrid: Since most intrusion detection and alert correlation systems do not correlate operating system events with network events, the intruders use a combination of both event types to bypass the detection mechanisms. For example, some APTs (e.g., Stuxnet, Hacking Team RCS, and ProjectSauron APTs [15]), for lateral movement in air-gapped networks, use removable media to spread the malwares from the Internet to local networks.

It is important to note that most APTs have only some of the six mentioned characteristics (especially low and slow features) and a few sophisticated APTs (e.g., Project Sauron APT [20]) have all of the six characteristics.

Related works

In recent years, several methods have been proposed to detect APT attacks (e.g., see [12–14, 21–38]) that suffer from lack of detecting low APTs, lack of evaluation against the well-known APTs such as Stuxnet and Flame, lack of detecting the hybrid APTs, and mostly lack of using dependable theoretical foundations. Nevertheless, recently a few good works have been done as follows.

Brogi et al. [39] proposed an APT detector called TerminAPTor, which tracks the information flows between the operating system processes. This approach intercepts events of a network system for two months and collects 3.2 billion events and 7.4 attacks per day. The main drawbacks of this approach are the lack of evaluation by some well-known APTs and the existence of high false positive alerts.

Ghafir et al. [40] proposed a machine learning-based system called MLAPT, which can detect APTs in real-time in three phases. In the first phase, the system analyzes the network traffic and generates some alerts based on some malicious patterns. In the next phase, the generated alerts of the first phase are correlated, and in the third phase, a machine-learning-based prediction is used for APT prediction. Again this approach has not been evaluated against the well-known APTs and it cannot detect the hybrid APTs.

In [16], a general approach is proposed to detect multi-step, hybrid, and low-level APTs. This approach is based on a knowledge-based system, i.e., the ontology of the operating system and network entities, low-level interception, and inference over the security policies and event relationships. The correlation between the operating system and network events in this approach is done based on the semantic relations of the entities, which are defined in the system ontology. In this approach, malicious behaviors and implicit violation of security policies are detected by deduction based on the existing knowledge of the occurred events and various relations between the entities of the machines and network. The main drawback of this approach is its weakness in detecting slow APTs. Since this approach is based on event correlation (instead of alert correlation) and uses ontology and inference engine, suffers from high processing overhead.

However, we believe this approach is the best available solution for detecting low-level and hybrid APTs.

Mohamed et al. [38] proposed an approach based on adversarial tactics techniques and a common knowledge matrix for detecting advanced persistent attacks. This approach focuses on detecting APT attacks in their first steps of malicious activities and they managed to reduce the detection time of the attack from several months to several minutes.

Problem statement

As discussed in the introduction, attack vectors of APTs have several characteristics such as special purpose, low-level, hybrid, multi-step, slow, or distributed. Since all these characteristics are not necessarily held in one APT, we just focus on the two most important characteristics, which make detection more difficult; low-level and slow APTs. Therefore, the problem is finding an approach $\varphi : \mathcal{P}(Event^{\mathcal{I}}) \rightarrow \mathcal{P}(v)$ to detect attack vectors such as v_i from a set of intercepted events such as $ES \subseteq E$, in which the following conditions are held:

1 v_i is low-level: Since APTs try to hide their malicious activities, explicit violation of security policy is not common in such attacks. For this reason, APTs violate the security policy implicitly and at least one violation of security policy occurs by attack vector v_i. In other words [16]:

$$\begin{aligned} &\exists e_i, p_j,\ e_i \in I(v_i) \wedge p_j \in SP \wedge object(e_i) = object(p_j) \\ &\quad \wedge subject(e_i) = subject(p_j) \wedge action(e_i) = action(p_j) \\ &\quad \wedge\ \nexists e_k, e_k \in v_i \wedge subject(e_k) = subject(e_i) \wedge \\ &\quad object(e_k) = object(e_i) \wedge action(e_k) = action(e_i)). \end{aligned} \tag{9}$$

2 v_i is slow: As mentioned in "Introduction" section, APT attacks infiltrate/exfiltrate data to/from the target system slowly in order to hide their malicious behavior. Our research shows that each APT attack lasts from several days [10, 41] to several months [4, 10]. However, the attack duration cannot last very long (e.g., several years); because the software migration or upgrade in the victim's infrastructure can cause the attack to fail.

3 Complexity of φ should be acceptable: Since, detecting low-level APT attacks require processing large event logs and detecting slow APTs makes the processing problem much harder, proposing an approach to detect both low-level and slow APT attacks in an acceptable time is another main challenge.

Proposed approach

As we mentioned in previous sections, since the most sophisticated APT attacks are low-level and slow, and these two characteristics make detection difficult for intrusion detection and alert correlation systems, the purpose of this paper is to detect this type of APT attack. In our approach, we enhance our previous solution proposed in [16] to detect slow APT attacks other than the low-level ones. Our approach takes the advantages of *event correlation* (instead of *alert correlation*) and using the ontology of operating

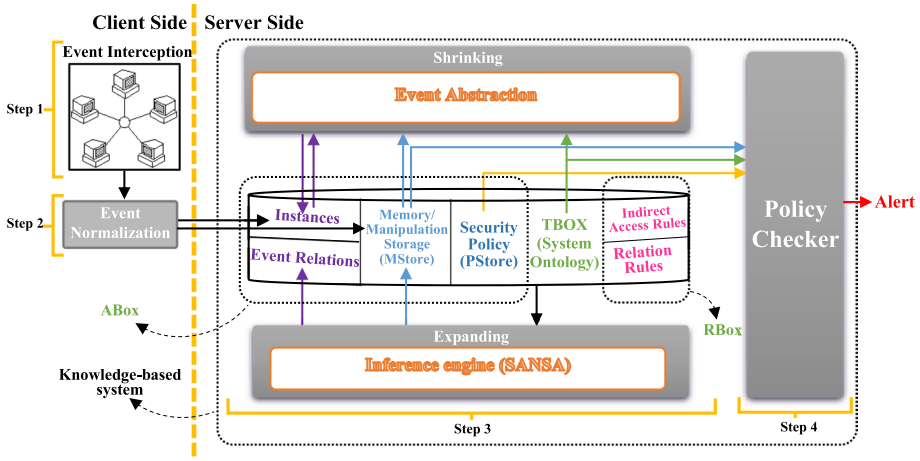

Fig. 3 Architecture of the proposed approach

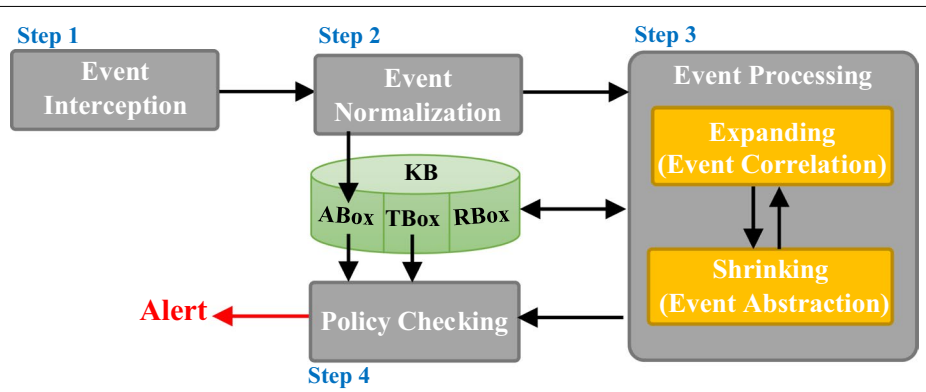

Fig. 4 The detection process based on the proposed approach

system and network entities, which are specified in this section ("Semantic correlation" section). Since the number of events and event relations significantly increases during the time, using semantic correlation leads to massive processing overhead for detecting slow attacks. The other purpose of this paper is to solve this problem.

The architecture of the proposed approach and the process of detecting malicious attack vectors like v_i are shown in Figs. 3 and 4, respectively. In our approach, on the client side, the operating system and network events are deeply intercepted, normalized, and sent to the server side. Afterward, in step 3, on the server side, we can detect the low-level attacks by using *ABox*, *TBox*, *RBox*, and an inference engine. Since the number of intercepted events is very big in slow APTs, we cannot use *Protégé-OWL* [42] as used in [16] for processing and inference. To overcome this problem, we use a scalable inference engine called *SANSA* [17], which can analyze a big size of *ABox* and *TBox* using *Spark*. Although *SANSA* is a good inference engine for processing big-size events, its processing power is limited. Therefore, in step 3, we use *Event Abstraction* concept to reduce the number of events, speed up the inference time, and detect very slow APTs (whose attack duration is more than one year).

In the *Event Abstraction* process, the old events are considered as an abstracted history instead of being completely disposed of. Finally, in step 4, we can detect the violation of security policy based on the inferred data in the previous steps and the high-level user-defined security policies.

The components and concepts that are used in the proposed architecture are explained in the rest of this section. Since our approach uses semantic correlation to detect low-level APTs, at first we explain the semantic correlation concepts and its limitation.

Semantic correlation

The main concepts of semantic correlation for detecting APT attacks which are retrieved from [16] are as follows:

1 *Knowledge Base* or *KB*: In semantic correlation for detecting APT attacks, we employ a knowledge base consisting of the following three boxes:

- *TBox*: This box defines the system ontology and the relations between the system entities. For example, the ontology of Windows operating system is shown in Fig. 5. As shown in this figure, class *Object* consists of three subclasses *KernelObj*, *UserObj*, and *GDIObj*. For another example, subject *Thread* is a part of subject *Process*.

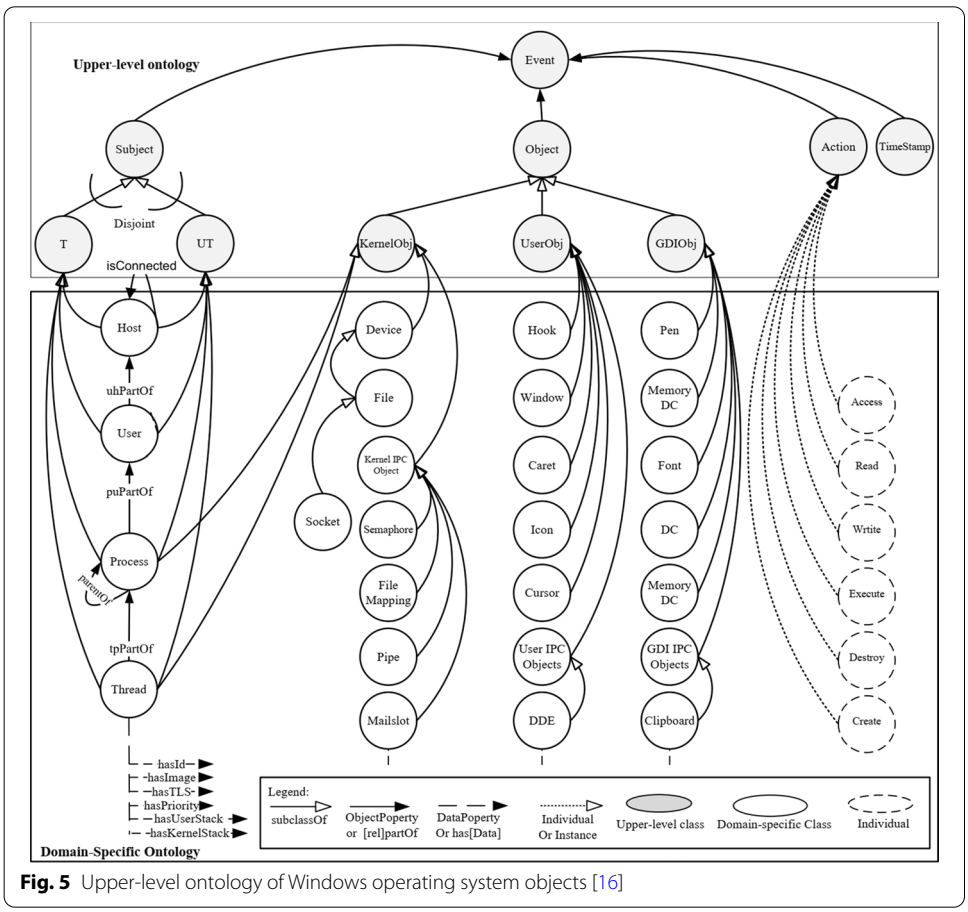

Fig. 5 Upper-level ontology of Windows operating system objects [16]

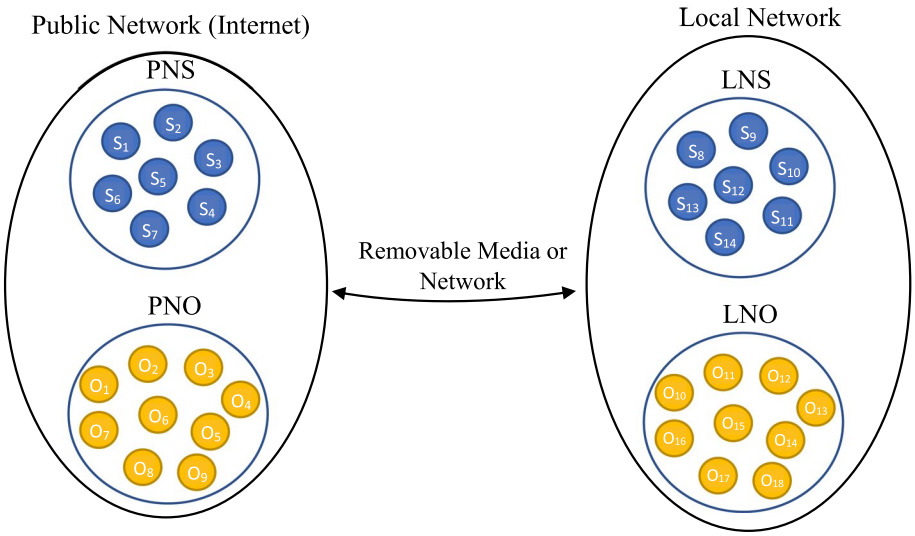

Fig. 6 Data exfiltration from local network to the public network [16]

- *ABox*: This box consists of four sub-boxes as follows:
  Instances or Individual Storage: All instances of the system ontology are stored in *Individual Storage*. For example, all intercepted events, subject instances, and object instances are stored in this box. In other words, Process p_i and Object o_j are samples of instances, which are stored in *Individual Storage*.
  Memory/Manipulation Storage (MStore): This sub-box uses two functions: Memory or *me* and Manipulation or *ma*. Function $me : Subject^{\mathcal{I}} \longrightarrow \mathcal{P}(Object^{\mathcal{I}})$ determines the objects that are read explicitly or implicitly by a specific subject. Function $ma : Subject^{\mathcal{I}} \longrightarrow \mathcal{P}(Object^{\mathcal{I}})$ determines the objects that are written explicitly or implicitly or deleted by a specific subject. For example $o_i \in me(s_i)$ means subject s_i has read object o_i, or $o_j \in ma(s_i)$ means object o_j has been written by subject s_i. These two functions are defined for detecting the violation of confidentiality and integrity, respectively.
  Security Policy (PStore): The security policy, which is defined in "Preliminaries", is stored in *PStore*. It is necessary to note that by using the ontology, we can define high-level and more abstract security policies and then infer the low-level security policies. The general format of security policy is shown in Algorithm 1.

$$ContextualCondition \longrightarrow Predicate$$

Where

$ContextualCondition := C(x) \mid R(x,y) \mid ContextualCondition \wedge ContextualCondition$
$Predicate := x \star MSet$
$\star := \in \mid \notin$
$MSet := me(x) \mid ma(x)$
$x, y :=$ [individual] | [variable]
$C :=$ [concept]
$R :=$ [relation]

Algorithm 1: General format of high-level security policy [16]

For example, in Fig. 2, the main security policy is $o_1 \notin me(s_2)$, which means subject s_2 should not

read object o_1. For another example, consider Fig. 6, if no data would be extracted from local network to the Internet, then the security policy can be defined as $LNO(o_i) \wedge PNS(s_j) \longrightarrow o_i \notin me(s_j)$, where:

$$LNO \sqsubseteq Object \text{ and } PNS \sqsubseteq Subject$$

In this scenario, data can be exfiltrated using different approaches (e.g., through network buffer or USB drive or CD-ROM). Since we use system ontology, it is not necessary to define several security policies, because all data transmission devices (e.g., network buffer or USB drive or CD-ROM) are a type of *PNO* objects. For more details about security policy, readers are referred to [16].

Event Relations: Two events can be related to each other by the relations between their subjects, objects, or actions. For example, relation $e_i \sim e_j$, which is defined for two events e_i and e_j, is stored in this sub-box. This sub-box contains the events that are related to each other based on some relation rules. The relation rules are described in the rest of this section in *RBox* subsection.

- RBox: This box consists of two sub-boxes as follows:
  Relation Rules: As mentioned before, two events can be related to each other based on their subjects, objects, or actions. All types of relations rules are described in Table 1. For example, as shown in this table, relation $e_i \overset{wr}{\sim} e_j$ means $a_i = W$ or Write and $a_j = R$ or Read.
  According to this table, we can define approximately 500 relation rules for event correlation (precisely $(3+1) \times 4 \times (6 \times 6)$ rules which some are meaningless). For example, relation rule $e_i \overset{tewrip}{\sim} e_j$ is equal to three event relations $e_i \overset{te}{\sim} e_j$, $e_i \overset{wr}{\sim} e_j$, and $e_i \overset{ip}{\sim} e_j$.
  Indirect Access Rules: Some event relations result in indirect change to the value of *me* and *ma* of subjects (e.g., as shown in Fig. 2). The related rules, which are used to detect indirect changes to the value of *me* and *ma*, are defined as *Indirect Access Rules*. These rules, which can be used for detecting low-level APTs, are defined in "Expanding: knowledge-based inference" section.

2 *Inference Engine*: Inference engine is a component of semantic correlation that uses the information and rules in the knowledge base to infer the event relations and calculates the *me* and *ma* for each subject. The low performance of inference engines is

Table 1 All types of event relations [16]

Relation type	#	Event relation	Meaning
Object	1	$e_i \overset{te}{\sim} e_j$	$(t_i < t_j) \wedge (o_i = o_j)$
	2	$e_i \overset{ti}{\sim} e_j$	$t_i < t_j) \wedge \left((o_i \neq o_j) \wedge (o_i \xrightarrow{partOf} o_j) \right)$
	3	$e_i \overset{pi}{\sim} e_j$	$(t_i < t_j) \wedge \left((o_i \neq o_j) \wedge (o_j \xrightarrow{partOf} o_i) \right)$
Subject	4	$e_i \overset{it}{\sim} e_j$	$(t_i < t_j) \wedge (s_i = s_j) \wedge Thread(s_i)$
	5	$e_i \overset{ip}{\sim} e_j$	$(t_i < t_j) \wedge Thread(s_i) \wedge Thread(s_j) \wedge \exists s_k, (Process(s_k) \wedge s_i \xrightarrow{partOf} s_k \wedge s_j \xrightarrow{partOf} s_k)$
	6	$e_i \overset{ih}{\sim} e_j$	$(t_i < t_j) \wedge Thread(s_i) \wedge Thread(s_j) \wedge \exists s_k, s_m, (Process(s_k) \wedge Process(s_m) \wedge s_i \xrightarrow{partOf} s_k \wedge s_j \xrightarrow{partOf} s_m \wedge s_k \neq s_m) \wedge \exists s_t, (Host(s_t) \wedge s_k \xrightarrow{partOf} s_t \wedge s_m \xrightarrow{partOf} s_t)$
	7	$e_i \overset{bh}{\sim} e_j$	$(t_i < t_j) \wedge Thread(s_i) \wedge Thread(s_j) \wedge \exists s_k, s_m, (Process(s_k) \wedge Process(s_m) \wedge s_i \xrightarrow{partOf} s_k \wedge s_j \xrightarrow{partOf} s_m \wedge s_k \neq s_m) \wedge \exists s_t, s_p, (Host(s_t) \wedge Host(s_p) \wedge s_k \xrightarrow{partOf} s_t \wedge s_m \xrightarrow{partOf} s_p \wedge s_t \neq s_p)$
Action	8	$e_i \overset{XY}{\sim} e_j$	$(t_i < t_j) \wedge (a_i = X) \wedge (a_i = Y)$

a considerable limitation in this approach. In the approach proposed in [16], *Protégé-OWL* [42] is used as an inference engine to perform reasoning based on Description Logic. The processing power of *Protégé-OWL* is limited to a knowledge base with several million frames. Hence *Protégé-OWL* is not a proper inference engine to detect slow APTs.

3 *Policy Checker*: In the final step, according to the *me* and *ma* functions and the security policy, which is stored in *PStore* and defined based on the system ontology, the system detects the violations of the security policy. More details of *Policy Checker* is explained in "Step 3: big event set processing" section.

For a better understanding, the steps of semantic correlation for detecting the violation of security policy in the case of Fig. 2 are shown in Table 2. As shown in this table, there is a user-defined security policy for the system. Security policy $o_1 \notin me(s_2)$ means subject s_2 is not permitted to read object o_1. According to this table, the intercepted events are $e_1 = \langle s_1, o_1, R, t_1 \rangle$, $e_2 = \langle s_1, o_2, W, t_2 \rangle$, $e_3 = \langle s_2, o_2, R, t_3 \rangle$. In the next step, sets *me* and *ma* can explicitly be calculated through the intercepted events as follows: $me(s_1) = \{o_1\}, ma(s_1) = \{o_2\}, me(s_2) = \{o_2\}, ma(s_2) = \varnothing$. Afterwards, according to the relation rule $time(e_i) < time(e_j) \wedge object(e_i) = object(e_j) \wedge action(e_i) = W \wedge action(e_j) = R \Longrightarrow e_i \overset{tewr}{\sim} e_j$ two events e_2 and e_3 are correlated. In the next step, since e_2 and e_3 are correlated to each other, one indirect access rule is fired and set *me* for subject s_2 should be updated implicitly. Hence, $me(s_2) = \{o_2\} \cup \mathbf{\{o_1\}}$ and this means the security policy is violated implicitly and a low-level attack is occurred.

Table 2 Steps of policy violation detection for file removal example according to Figure 2 [16]

Step	Component	Sample (based on Fig. 2).
Step 1	Event Interception	$e_1 = \langle s_1, o_1, R, t_1 \rangle$, $e_2 = \langle s_1, o_2, W, t_2 \rangle$, $e_3 = \langle s_2, o_2, R, t_3 \rangle$
Step 2	Event Normalization	Using OWL-DL to define the events and initiate the *me* and *ma*.
	Memory/Manipulation Storage (MStore)	$me(s_1) = \{o_1\}, ma(s_1) = \{o_2\}, me(s_2) = \{o_2\}, ma(s_2) = \varnothing$
Step 3	Individual Storage	$Event(e_1), Subject(q_1), Object(o_1), \ldots$
	System Ontology (TBox)	$\sqsubseteq$, $partOf$, $parentOf$, .
	Relation Rules (RBox)	$(time(e_i) < time(e_j) \wedge object(e_i) = object(e_j) \wedge action(e_i) = W \wedge action(e_j) = R) \Longrightarrow e_i \overset{tewr}{\sim} e_j$
	Inference Engine	$e_2 \overset{tewr}{\sim} e_3$
Step 4	ABox and MStore	$e_2 \overset{tewr}{\sim} e_3, me(s_1) = \{o_1\}, ma(s_1) = \{o_2\}, me(s_2) = \{o_2\}, ma(s_2) = \varnothing$
	System Ontology (TBox)	$\sqsubseteq$, $partOf$, $parentOf$, .
	Indirect Access Rules (RBox)	$\forall e_i, e_j, (Event(e_i) \wedge Event(e_j) \wedge e_i \overset{tewr}{\sim} e_j \Rightarrow me(subject(e_j)) = me(subject(e_j)) \cup me(subject(e_i)))$
	Inference Engine	$\mathbf{me(s_2) = me(s_2) \cup me(s_1)}$
Step 5	Memory/Manipulation Storage (MStore)	$me(s_1) = \{o_1\}$, $ma(s_1) = \{o_2\}, me(s_2) = \{o_2\} \cup \mathbf{\{o_1\}}$
	Security Policy Store (PStore)	$o_1 \notin me(s_2)$
	Policy Checker	**Alert**

Limitation of semantic correlation

Semantic correlation is an approach proposed in [16] for detecting multi-step, hybrid, and low APTs. This approach uses description logic, and OWL for correlating the events based on their relations and inferring the violation of security policies to detect the APTs.

The main challenge in using the semantic correlation is the size of the knowledge base that affects the reasoning time. To the best of our knowledge, semantic correlation is a good idea to detect low-level and slow APTs, if we overcome the processing overhead. In the semantic correlation, the reasoning time depends on the size of the knowledge base including the following components:

- *RBox*: Big size of *RBox* can strongly increase the reasoning time. Since in the field of APT detection, the number of rules in *RBox* is limited (in maximum 500 rules as discussed in *Relation rules* subsection), *RBox* does not have a great impact on the reasoning time.
- *TBox*: Since we use *TBox* for storing the ontology of the Windows operating system and the size of the Windows ontology is not very large, *TBox* does not make a challenge for reasoning time.
- *ABox*: Any instances of *OWL* classes and relations (e.g., events, subjects, objects, actions, and relations) are stored in *ABox*. Since the number of collected events (and consequently the number of event relations) grows rapidly during the time, the size of *ABox* increases significantly by event interception and event correlation as well. This problem is more evident for slow APTs. In this paper, we propose an approach for overcoming this problem.

In the rest of this section, we explain four detection steps, which are shown in Figs. 3 and 4.

Step 1: event interception

Since APT attacks are low-level and violate the security policy implicitly, we should intercept all system events deeply. Hence, in our approach, we intercept both network and operating system events in different layers. The required layers of event interception for detecting low-level APTs are shown in Fig. 7. As shown in this figure, we intercept both network and operating system events in different ways. For intercepting network events, we use switch port mirroring (port spanning) to intercept MAC Address, source IP, destination IP, source port, destination port, and timestamp of each flow. For intercepting operating system events, we intercept system calls in user mode, kernel mode, and hypervisor of the virtual machine. Intercepting in user mode is done by code injection and kernel mode is done by hooking system service dispatch (SSDT) and Shadow SSDT tables, IRP hooking, and call back functions. The event interception in the virtual machine layer is done by intel virtualization technology. Afterward, each intercepted system call is normalized to the quadruple format, which consists subject, object, action, and timestamp of the event. The first attribute is one of the *host_id*, *user_id*, *process_id*, or *thread_id*. The second one is the handle of the object, which has a unique identifier. The third attribute is the action of the event, and finally, the fourth one is the timestamp

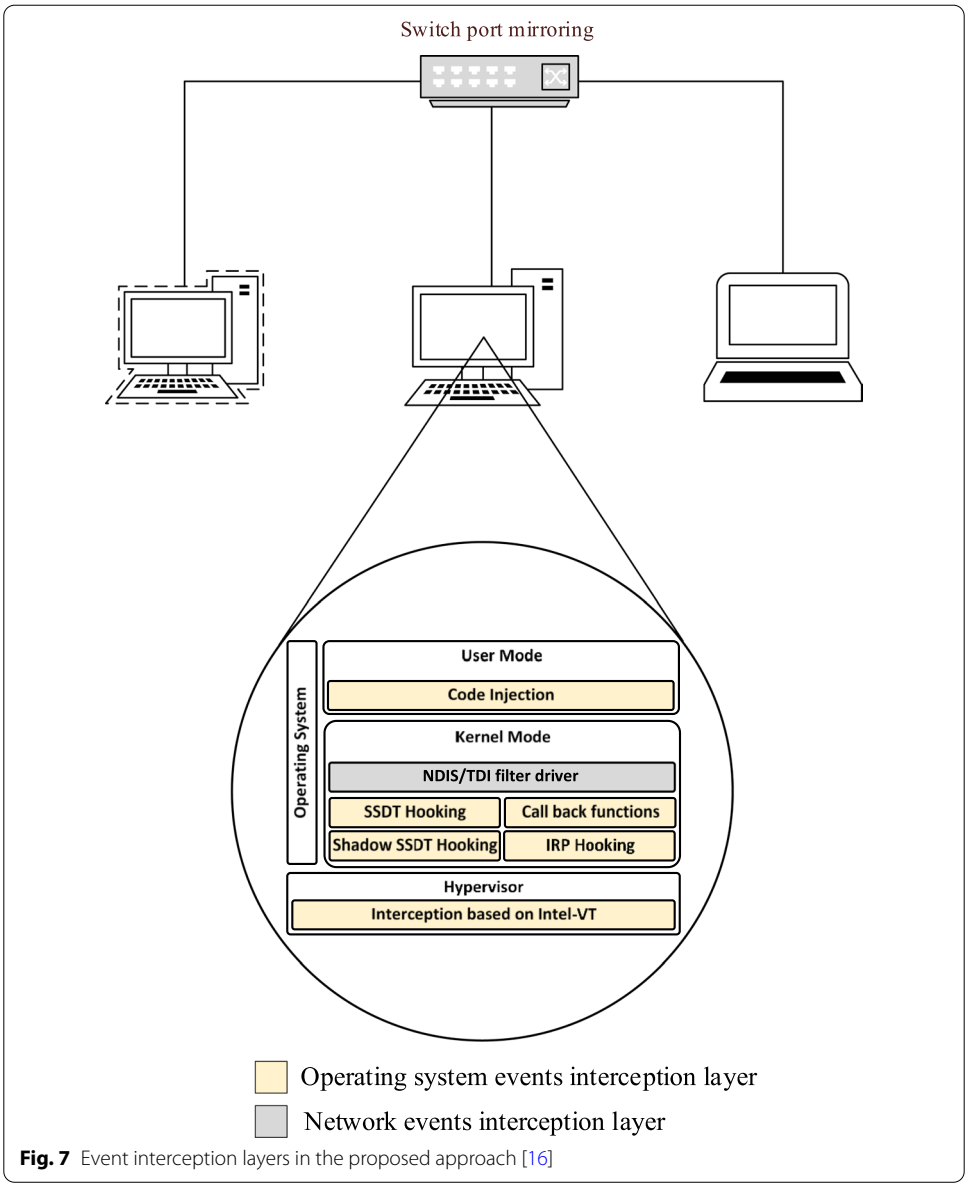

Fig. 7 Event interception layers in the proposed approach [16]

of the system call. Since in the proposed approach we track the information flow, we use network events just for determining which host is connected to another host (by *isConnected* relation) on a specific port at a specific timestamp. Also, since we intercept operating system events at the host level, we determine which specific process is bound to a specific port for receiving/sending data from/to another host. That way, we can specify the information flow between two processes in two different hosts. For example, consider if Process p_1 in host h_1 sends a packet to process p_2 in host h_2, then we can track the information flow through the following events:

1 $e_i = \langle p_1, PortA, W, t_1 \rangle$
2 $isConnected(h_1, h_2, PortA, PortB, t_2)$
3 $e_j = \langle p_2, PortB, R, t_3 \rangle$

Step 2: event normalization

After the interception of events, we should use a uniform format for describing and storing the event logs. Intrusion detection message exchange format (IDMEF) [43] is a standard format for storing and representing event logs in intrusion detection and alert correlation systems; however, since we use semantic correlation, we should store event logs in a standard format which is understandable by the inference engine. Hence we employ *OWL-DL* [44] for describing and storing event logs.

Another role of *Event Normalization* is to specifying the explicit read or written objects by the subjects and initiating two sets *me* and *ma*. In other words, we use two rules in the *Event Normalization* step for initiating two sets *me* and *ma* as follows [16]:

1) $Event(e_i) \wedge (action(e_i) = R) \longrightarrow me(subject(e_i)) := me(subject(e_i)) \cup \{object(e_i)\}$
2) $Event(e_i) \wedge (action(e_i) = W) \longrightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{object(e_i)\}$.

For example, in implicit file read example, according to Table 2, three intercepted events $e_1 = \langle s_1, o_1, R, t_1 \rangle$, $e_2 = \langle s_1, o_2, W, t_2 \rangle$, $e_3 = \langle s_2, o_2, R, t_3 \rangle$ lead to initiating sets *me* and *ma* for each subject as $me(s_1) = \{o_1\}$, $ma(s_1) = \{o_2\}$, $me(s_2) = \{o_2\}$, and $ma(s_2) = \varnothing$.

Step 3: big event set processing

After the events are intercepted and normalized, it is time to process the event logs, and detect attack vectors like v_i on the server side. As mentioned in "Semantic correlation" section, since detecting slow APT attacks depends on analyzing a big number of events, the described approach in [16] is not applicable and adopting a scalable knowledge-based system is essential. To solve this problem, in this paper, we propose a scalable knowledge-based system which is described in "Big event knowledge-based processing" section.

Step 4: policy checking

In our approach, the event correlation and policy checking are performed every eight hours. In these situations, after the correlation of the events, if the values of *me* or *ma* change, the policy checker is fired and checks the violation of security policy. As discussed in "Preliminaries" section, the security policies are defined in set *SP* and any action performed by the subjects affects the *ma* and *me* sets. Therefore, following the approach is proposed in [16], the violation of security policy can be detected as follows:

- *Confidentially violation*: A subject s_i that reads an object o_i ($o_i \in me(s_i)$) violates the confidentiality, if and only if the security policy does not permit explicitly or implicitly reading object o_i by subject s_i. In other words:

$$Subject(s_i) \wedge Object(o_i) \wedge o_i \in me(s_i) \wedge \\ \langle s_i, o_i, R \rangle \in SP \Rightarrow confidentialityViolation = True$$

- *Integrity violation*: A subject s_i that writes an object o_i ($o_i \in ma(s_i)$) violates the integrity, if and only if the security policy does not permit explicitly or implicitly writing object o_i by subject s_i. In other words:

$$Subject(s_i) \wedge Object(o_i) \wedge o_i \in ma(s_i) \wedge \\ \langle s_i, o_i, W \rangle \in SP \Rightarrow integrityViolation = True.$$

It is necessary to note that the proposed approach has no solution to detect the APT attacks that violate the availability of the system directly such as power save denial of service (PS-DoS) [45] attack.

When we detect a violation of security policy, the following steps can help to detect the origins of infection:

- Determining the events that cause to violate the security policy as the malicious events.
- Tracing back the attack vectors that contain the malicious events.
- Determining the first events of attack vectors.
- The subjects of these events are the origins of APT attacks.

Big event knowledge-based processing

As we described in previous sections, to overcome the complexity of slow APTs detection, we propose a scalable knowledge-based system which uses the three following techniques:

1 A flexible sliding window called *Vermiform window*,
2 A scalable inference engine called *SANSA*,
3 A data summarization process based on the system ontology called *Event Abstraction*.

Details of these techniques are described in the following sections.

Vermiform window

Since in our approach we deal with the big size of events, it is necessary to use a sliding window to prevent data explosion. In similar circumstances, it is prevalent to use a fixed-length (or fixed-size) sliding window. However since the fixed-length sliding window is not supple and flexible, it is not suitable for our approach. Hence, in our approach, we use a variable-length sliding window, which we called *Vermiform* window. This window is just like a worm with a variable length in movement. The movement steps of this sliding window are shown in Fig. 8. As shown in this figure, the Vermiform window has two steps in movement: *Expanding* and *Shrinking*. We map the expanding and shrinking of the Vermiform window to the context of event correlation as follows:

- *Expanding:* In expanding step, the new intercepted events are appended to the window immediately and the events of the window are correlated to each other every eight hours and then the inference engine checks the violation of security

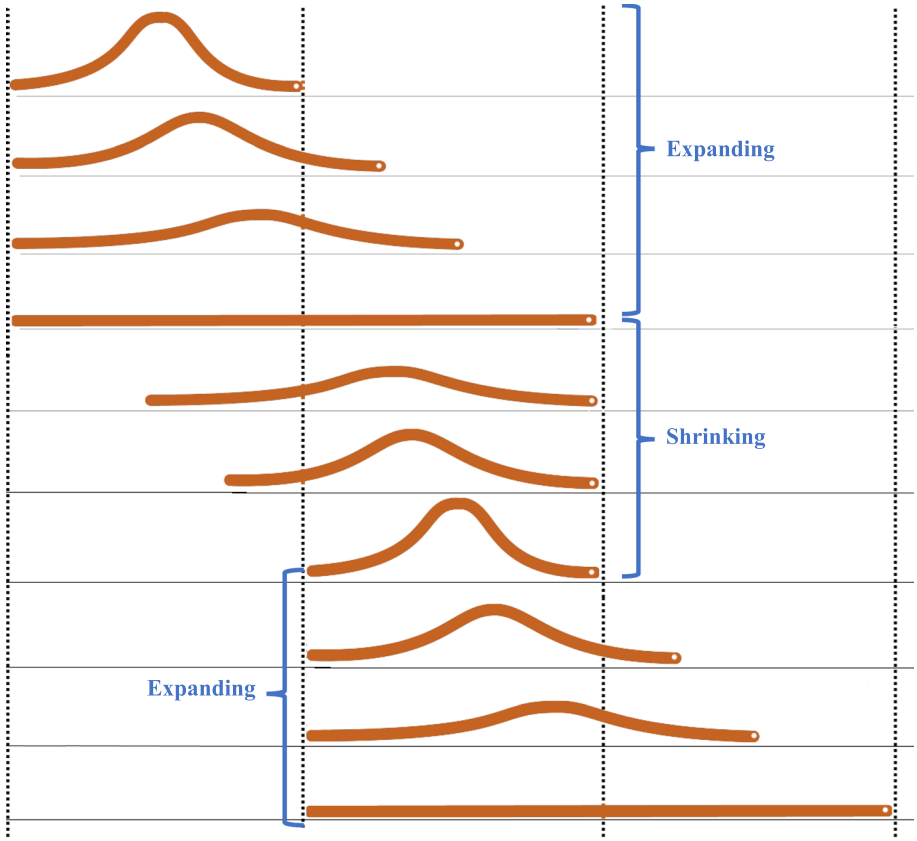

Fig. 8 The movement of the Vermiform window

policy. The process of event correlation in expanding movement is discussed in "Expanding: knowledge-based inference" section.

- *Shrinking:* In some situations, the number of events in the Vermiform window becomes very high and we cannot append any new event to the window. In this state, the prevalent approach is eliminating some of the old events to provide a free space for the new events. Since eliminating the old events reduces the accuracy of the detection approach, we create a history or an abstraction from the old events, instead of eliminating them. The process of event abstraction is discussed in "Shrinking: event abstraction" section. Therefore, the main purpose of shrinking is to reduce the number of events in the sliding window.

Expanding: knowledge-based inference

Since using semantic-based correlation based on *Protégé-OWL* [42] (as an inference engine) is not applicable for detecting slow attacks and leads to huge processing overhead, we should use a scalable platform for processing *OWL* files and reasoning. In our proposed approach, we use *SANSA* [17] as an inference engine. The architecture of *SANSA* is shown in Fig. 9. As shown in this figure, *SANSA* is a scalable inference engine, which takes the advantages of big data processing frameworks such as *Spark*

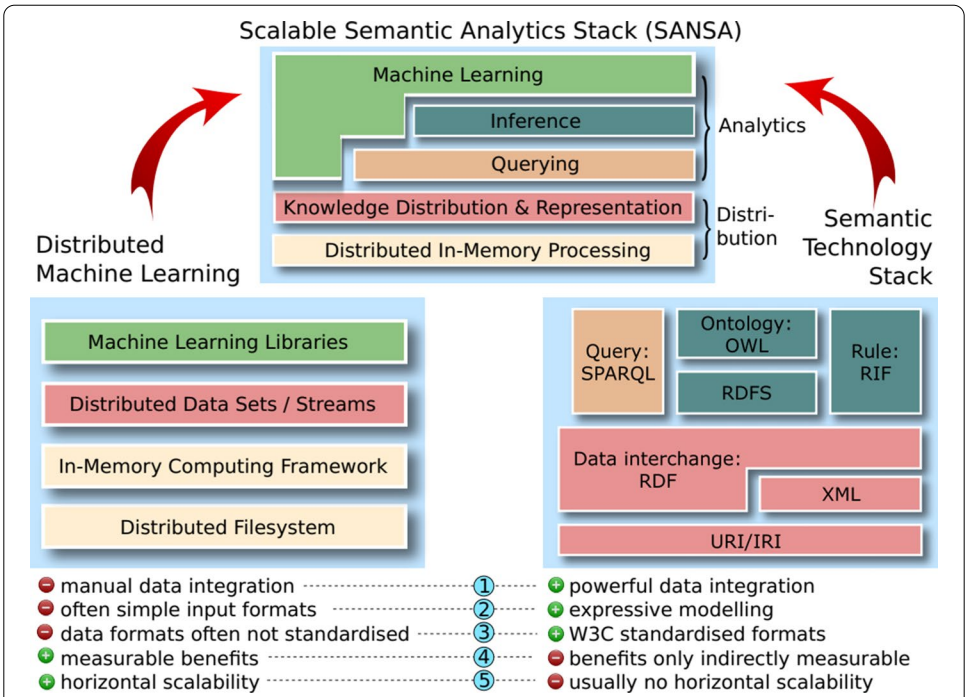

Fig. 9 Architecture of SANSA inference engine [17]

and *Flink* for querying, inferring, and large-scale RDF (Resource Description Framework) data analysis.

Based on our experience, by using *SANSA*, we can simply analyze more than one billion frames in an acceptable time. A drawback of *SANSA* is that this engine does not support semantic web rule language *(SWRL)*, which is used in our proposed approach. For solving this problem, we use *Jena* [46] rule language, which is implemented by *SANSA*, and redefined *RBox* rules (*Relation* rules and *Indirect Access Rules*) using *Jena* [46] rule language.

In the expanding process, the events are appended to Vermiform window every time a new event is generated and intercepted, and the inference engine correlates the events and calculates the implicit actions that cause to change the values of *me* and *ma* sets for each subject based on *ABox*, *TBox*, and *RBox*. After that, the *Policy Checker investigates* the violation of security policy based on the high-level user-defined security policies, and the values of *me* and *ma* sets of subjects. Policy checking is described in "Step 3: big event set processing" section.

The process of calculating the implicit actions is done by using *Indirect Access Rules*, which is placed in *RBox* of the *Knowledge Base*. As mentioned in [16], *Indirect Access Rules* consists of two types of rules which are defined as follows.

- *Transition rules:* These rules specify the circumstances (by the occurrence of a set of events) where some objects are read or written implicitly. In [16], thirteen rules of this type, which implicitly change the values of *me* and *ma* sets, are defined. These rules are shown in Table 3.

Table 3 Patterns of transition rules [16]

#	Rule
1	$Subject(s_i) \wedge Subject(s_j) \wedge (s_i \xrightarrow{partOf} s_j) \Longrightarrow me(s_j) = me(s_j) \cup me(s_i)$
2	$Subject(s_i) \wedge Subject(s_j) \wedge (s_i \xrightarrow{partOf} s_j) \Longrightarrow ma(s_j) = ma(s_j) \cup ma(s_i)$
3	$Process(s_i) \wedge Process(s_j) \wedge (s_i \xrightarrow{parentOf} s_j) \Longrightarrow me(s_j) = me(s_j) \cup me(s_i)$
4	$Process(s_i) \wedge Process(s_j) \wedge (s_i \xrightarrow{parentOf} s_j) \Longrightarrow ma(s_j) = ma(s_j) \cup ma(s_i)$
5	$Event(e_i) \wedge Object(o_j) \wedge (o_j \xrightarrow{partOf} object(e_i)) \wedge (action(e_i) = R) \Longrightarrow me(subject(e_i)) := me(subject(e_i)) \cup \{o_j\}$
6	$Event(e_i) \wedge Object(o_j) \wedge (o_j \xrightarrow{partOf} object(e_i)) \wedge (action(e_i) = W) \Longrightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{o_j\}$
7	$Event(e_i) \wedge Object(o_j) \wedge (o_j \xrightarrow{partOf} object(e_i)) \wedge (action(e_i) = D) \Longrightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{o_j\}$
8	$Event(e_i) \wedge Event(e_j) \wedge action(e_i) = W \wedge action(e_j) = R \wedge object(e_i) = object(e_j) \wedge UT(subject(e_i)) \Rightarrow me(subject(e_j)) = me(subject(e_j)) \cup me(subject(e_i))$
9	$Event(e_i) \wedge Event(e_j) \wedge action(e_i) = W \wedge action(e_j) = R \wedge object(e_i) \xrightarrow{partOf} object(e_j) \wedge UT(subject(e_i)) \Rightarrow me(subject(e_j)) = me(subject(e_j)) \cup me(subject(e_i))$
10	$Event(e_i) \wedge Event(e_j) \wedge UT(subject(e_i)) \wedge (action(e_i) = W) \wedge (object(e_i) = subject(e_j)) \wedge (action(e_j) = W) \Rightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{object(e_j)\}$
11	$Event(e_i) \wedge Event(e_j) \wedge UT(subject(e_i)) \wedge (action(e_i) = W) \wedge (object(e_i) = subject(e_j)) \wedge (action(e_j) = D) \Rightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{object(e_j)\}$
12	$Event(e_i) \wedge Event(e_j) \wedge UT(subject(e_i)) \wedge (action(e_i) = C) \wedge (object(e_i) = subject(e_j)) \wedge (action(e_j) = W) \Rightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{object(e_j)\}$
13	$Event(e_i) \wedge Event(e_j) \wedge UT(subject(e_i)) \wedge (action(e_i) = C) \wedge (object(e_i) = subject(e_j)) \wedge (action(e_j) = D) \Rightarrow ma(subject(e_i)) := ma(subject(e_i)) \cup \{object(e_j)\}$

Table 4 Patterns of untrusted subjects rules [16]

#	Rule	Description
1	$Subject(s_i) \wedge Subject(s_j) \wedge (s_i \xrightarrow{partOf} s_j) \wedge UT(s_i) \Longrightarrow UT(s_j)$	Untrusted subpart
2	$Event(e_i) \wedge Event(e_j) \wedge e_i \overset{tewr}{\sim} e_j \wedge UT(subject(e_i)) \Rightarrow UT(object(e_j)$	Untrusted input
3	$Event(e_i) \wedge Event(e_j) \wedge e_i \overset{tiwr}{\sim} e_j \wedge UT(subject(e_i)) \Rightarrow UT(subject(e_j)$	Untrusted input
4	$Event(e_i) \wedge Event(e_j) \wedge e_i \overset{piwr}{\sim} e_j \wedge UT(subject(e_i)) \Rightarrow UT(subject(e_j)$	Untrusted input
5	$Event(e_i) \wedge UT(subject(e_i)) \wedge (action(e_i) = W) \wedge Subject(object(e_i)) \Rightarrow UT(object(e_i)$	Injection

- *Untrusted subjects rules:* These rules specify the circumstances where the nature of a trusted subject change to an untrusted one. In [16] five rules of this type are defined. These rules are shown in Table 4.

Since we used a sliding window, it is necessary to specify its length in expanding step. The minimum and maximum length of the sliding window in the expanding step depends on the minimum duration of the prevalent APT attacks, and the maximum processing power of the inference engine respectively. Therefore, our sliding window restrictions in expanding are time and size dependent. These restrictions are shown in Fig. 10 and are described in the following.

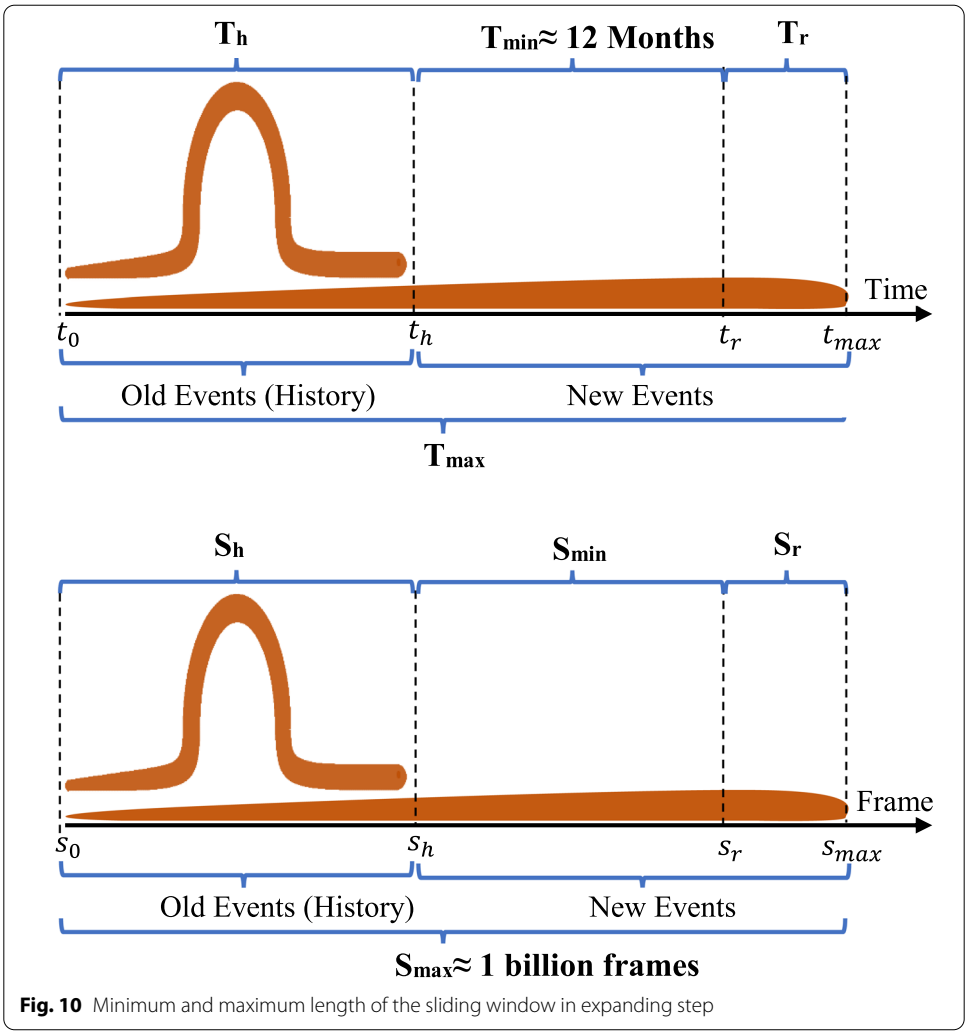

Fig. 10 Minimum and maximum length of the sliding window in expanding step

- T_{min} or minimum time of the Vermiform window in expanding: Since regular APT attacks duration is about several months (maximum one year), we should at least analyze and correlate the events intercepted in this interval. In other words:

$$T_{min} = t_r - t_h \simeq 12\ Months. \tag{10}$$

- S_{max} or maximum size of the Vermiform window in expanding: As mentioned before, the maximum length of the window is when the window is fully expanded. This maximum length depends on the processing power of the inference engine. Since the processing power of our inference engine is limited (approximately one billion frames), the maximum number of frames, which can be in the Vermiform window, is approximately one billion. In other words:

$$S_{max} = s_{max} - s_0 \simeq 1\ billion\ frames. \tag{11}$$

- S_{min} or minimum size of the Vermiform window in expanding: To succeed in discovering the APT attacks, the inference engine must ensure that:

$$S_{max} \geq S_{min}. \tag{12}$$

S_{min} is the number of generated frames from timestamp t_h to t_r. Since this value is time dependent, and differs from one event set to another, and depends on the size of the victim's computer network, the exact value of S_{min} is calculated for each event set such as ES as follows:

$$S_{min}(ES) = \sum_{t=t_h}^{t=t_r} f(ES, t), \tag{13}$$

where f is the frame function (which is defined in "Preliminaries") and ES is the set of all collected events in the expanding step of the Vermiform window.

- T_{max} or maximum time of the Vermiform window in expanding: To succeed in discovering the APT attacks, the inference engine must ensure that:

$$T_{max} \geq T_{min}. \tag{14}$$

T_{max} is the maximum time of the sliding window in expanding. Since this value differs from one event set to another and depends on the size of the victim's computer network, the exact value of T_{max} is calculated for each event set such as ES as below:

$$T_{max} = Max\{time(e_k) \mid e_k \in ES\} - Min\{time(e_k) \mid e_k \in ES\}, \tag{15}$$

where ES is the set of all collected events in the expanding step of the Vermiform window.

The process of appending the new events to the sliding window is paused when the maximum length of Vermiform window becomes equal to S_{max} or approximately one billion frames. In this situation, we should start the shrinking process, which is explained in the next section.

Shrinking: event abstraction

As mentioned before, the number of collected events grows rapidly over time, and the correlation of all collected events is impossible in practice. In this situation, the simplest solution is to eliminate the old events. Since eliminating the old events leads to a reduction in the accuracy rate of the detection, we create a history from the old events (by abstracting the events) instead of eliminating them. The process of event abstraction occurs in the shrinking step.

Before describing our event abstraction approach, it is necessary to specify the length of the sliding window in the shrinking step, which has a direct impact on the event abstraction approach.

Since the events of the previous expanding step should be considered as a history (following the shrinking step) in the current expanding step, the length of the Vermiform window for the history (containing the abstracted events) depends on the free size and time of the sliding window in the current expanding step. These two parameters are shown by T_h and S_h in Fig. 10. Since the history length of the Vermiform window differs from one event set to another and depends on the size of the victim's computer network, the exact value of S_h and T_h are calculated for each event set like ES as follows.

$$0 \leqslant T_h(ES) = t_h - t_0 \leqslant T_{max}(ES) - T_{min} \tag{16}$$

$$\leqslant S_h(ES) = s_h - s_0 \leqslant S_{max} - S_{min}(ES), \tag{17}$$

where ES is the set of all collected events in the current expanding step of the Vermiform window.

After determining the maximum length of the sliding window for abstracted events obtained by the previous shrinking step (i.e., T_h and S_h), it is time to summarize all the events collected in the expending step to a set of abstract events with a maximum length of T_h and S_h.

According to these limitations, abstraction function $\Delta : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$ is defined which maps a set of events such as WE to a set of abstract events such as AE (i.e., $\Delta(WE) = AE$) and the following conditions are held:

- There is no capacity to store more events than those appearing in WE in the current expanding step of the Vermiform window. In other words:

$$S_h(ES) < \sum_{t=0}^{t=\infty} f(WE, t), \tag{18}$$

  where ES is the set of all collected events in the current expanding step of the Vermiform window.
- The size of AE as history in Vermiform window is much less than the maximum size of the current expanding step. In other words:

$$S_h(ES) \geqslant \sum_{t=0}^{t=\infty} f(AE, t). \tag{19}$$

Function Δ considers two main facts for event abstraction as follows:

1 *History size* (S_h): It is obvious that if the size of the Vermiform window in shrinking step becomes less, function Δ should perform more abstraction compared to when the size of the Vermiform window in shrinking (S_h) is big.
2 *Events timestamps*: Since in the process of event correlation, the last events of the sliding window are more valuable than the old events, the function Δ should use less abstraction for the recent events and more abstraction for the old ones.

According to these two facts, we define several abstraction levels, which are used by function Δ, to apply different types of event abstraction based on the size of S_h and events' timestamps. The function Δ employs the system ontology for abstracting the system entities such as objects, subjects, and actions. Using ontology we can replace the lower level (more concrete) entities with the upper-level (more abstract) entities. Function Δ performs the abstraction based on the following entities:

- Abstraction based on actions: To increase the expression power and accuracy, all the system actions were defined by six basic actions (Read, Write, Execute,

Table 5 Abstraction rules of actions

Abstraction rule	Abstraction step	Abstraction condition	Abstraction operation
R_1A	1.1	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge action(e_i) = C$	$e_i = \langle s_i, o_i, W, t_i \rangle$
	1.2	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge action(e_i) = D$	$e_i = \langle s_i, o_i, W, t_i \rangle$
	1.3	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge action(e_i) = A$	$e_i = \langle s_i, o_i, R, t_i \rangle$
	1.4	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge Process(s_i) \wedge Process(o_i) \wedge action(e_i) = E$	Delete e_i and add $s_i \xrightarrow{parentOf} o_i$

Table 6 Abstraction rules of subjects

Abstraction rule	Abstraction condition	Abstraction operation
R_1S	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge Thread(s_i) \wedge \exists s_j, Process(s_j) \wedge s_i \xrightarrow{partOf} s_j$	$e_i = \langle s_j, o_i, a_i, t_i \rangle$
R_2S	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge Process(s_i) \wedge \exists s_j, Process(s_j) \wedge s_i \xrightarrow{parentOf} s_j$	$e_i = \langle s_j, o_i, a_i, t_i \rangle$
R_3S	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge Process(s_i) \wedge \exists s_j, User(s_j) \wedge s_i \xrightarrow{partOf} s_j$	$e_i = \langle s_j, o_i, a_i, t_i \rangle$
R_4S	$\forall e_i = \langle s_i, o_i, a_i, t_i \rangle, Event(e_i) \wedge Process(s_i) \wedge \exists s_j, Host(s_j) \wedge s_i \xrightarrow{partOf} s_j$	$e_i = \langle s_j, o_i, a_i, t_i \rangle$

Table 7 Abstraction rules of objects

Abstraction rule	Abstraction step	Abstraction condition	Abstraction operation
R_1O	1.1	$\forall e_i = \langle s_i, o_i, R, t_i \rangle, Event(e_i) \wedge Object(o_i) \wedge \exists o_j, Object(o_j) \wedge o_i \xrightarrow{partOf} o_j$	$e_i = \langle s_i, partOf(o_j), R, t_i \rangle$
	1.2	$\forall e_i = \langle s_i, o_i, W, t_i \rangle, Event(e_i) \wedge Object(o_i) \wedge \exists o_j, Object(o_j) \wedge o_i \xrightarrow{partOf} o_j$	$e_i = \langle s_i, partOf(o_j), W, t_i \rangle$

Delete, Access, and Create). However, some of these actions (i.e., Create, Delete, and Access) can be replaced by some other actions (i.e., Read and Write). To this aim, we define a set of abstraction rules, which is shown in Table 5. According to these rules, actions Create and Delete are replaced by action Write, action Access is replaced by action Read, and action Execute is eliminated for event correlation; because it does not affect event correlation.

- Abstraction based on subjects: Considering the ontology of the Windows operating system (which is shown in Fig. 5), we can replace the lower level subjects with the upper-level ones. For example, a *Thread* can be abstracted as a *Process*, or a *Process* can be abstracted as a *User* or a *Host*. Hence we define four abstraction rules based on the subjects' relations (which are shown in Table 6). For example, rule R_1S means each event, which is generated by a thread such as s_i, is supposed to be generated by its related process such as s_j.
- Abstraction based on objects: Similar to the subjects, we can replace the lower level objects by the upper-level ones. For example, a *Socket* can be abstracted as a *File* or a *Device*, and a *Device* can be abstracted as a *KernelObj*. Hence, we define an abstraction rule based on objects' relations (which is shown in Table 7) to abstract the events

Table 8 Event abstraction levels

Abstraction level	Abstraction rules	Average event reduction rate	Average impact on detecting very slow attacks	Average impact on detecting slow attacks
L_0	No abstraction	0 %	0 %	0 %
L_1	R_1A	12 %	+2.8 %	0 %
L_2	$L_1 \wedge R_1S$	7 %	+0.12 %	0 %
L_3	$L_2 \wedge R_1O$	11 %	+1.5 %	−0.9 %
L_4	$L_3 \wedge R_2S$	18 %	+3.24 %	0 %
L_5	$L_4 \wedge R_3S$	47 %	−1.11 %	−6.8 %
L_6	$L_5 \wedge R_4S$	51 %	−4.3 %	−12.12

or remove the redundant information. Rule R_1O says if subject s_i wants to read object o_i and this subject had read object o_j before that and o_i is a part of o_j, then this event can be abstracted by another event such as $e_i = \langle s_i, o_j, R, t_i \rangle$ with less details.

By using these six rules, many events are abstracted and replaced by other events. The necessity of these six rules is to reduce the details of each event and increase the abstraction of each event without considering the event relations.

According to Tables 5, 6 and 7, we can combine these rules and define different abstraction levels (e.g., abstraction level $R_1S \wedge R_1O \wedge R_1A$). However, some of them are meaningless or have negative effects on both accuracy of the detection approach and the rate of events reduction. We examined all different abstraction levels by a dataset (which is introduced and discussed in "Evaluation" section) and sift all possible abstraction levels to seven main abstraction levels, which are shown in Table 8. As shown in this table, seven abstraction levels are constructed based on six basic abstraction rules. Also, the first abstraction level (L_0) does not use any abstraction rules and does not have any advantage to reduce the number of events or increasing the accuracy. It is necessary to mention that before applying a new abstraction level, all the redundant events resulting from the previous abstraction level have to be eliminated.

In Table 8, column *Average Event Reduction Rate* specifies the percentage of the events that are reduced by a specific abstraction level using our dataset. Also, column *Average impact on detecting very slow attacks* specifies the average impact of a specific abstraction level on the accuracy of the proposed approach for detecting the attacks that their duration is more than T_{max} (very slow attacks). Symbols '+' and '-' show the increment or decrease of the accuracy of the detection approach. Column *Average impact on detecting slow attacks* specifies the impact of a specific abstraction level on the accuracy of the detection approach for the attacks that their duration is less than T_{max} (slow attacks).

As mentioned before, since in the process of event correlation, the last events of the sliding window are more important than the old events, approach Δ use less abstraction for the recent events and more abstraction for the old events. Therefore, we divide the sliding windows into seven different partitions as shown in Fig. 11. In the partitioning process, the number of events in the seven partitions should be approximately the same. More formally the following condition should be held:

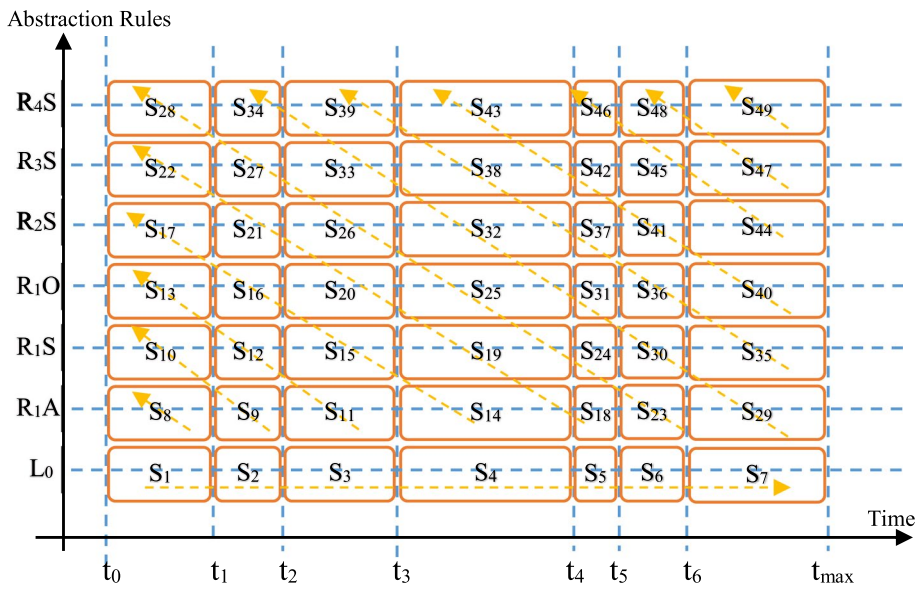

Fig. 11 Using less abstraction for the recent events and more abstraction for the old events

$$\begin{aligned}
&\sum_{t=t_0}^{t=t_1} f(WE,t) \simeq \sum_{t=t_1}^{t=t_2} f(WE,t) \simeq \sum_{t=t_2}^{t=t_3} f(WE,t) \simeq \\
&\sum_{t=t_3}^{t=t_4} f(WE,t) \simeq \sum_{t=t_4}^{t=t_5} f(WE,t) \simeq \\
&\sum_{t=t_5}^{t=t_6} f(WE,t) \simeq \sum_{t=t_6}^{t=t_{max}} f(WE,t).
\end{aligned} \tag{20}$$

As shown in Fig. 11, the process of event abstraction maximally consists of 49 steps. In other words, the process of the event abstraction is started from step S_1 and is continued until the size of the abstracted events become less or equal to S_h, and in the worth case this process is finished in step S_{49}. For example, if the event abstraction process is finished in 19th step (S_{19}), this means the events which their timestamp is t_0 to t_1 are abstracted based on abstraction level L_4, the events which their timestamp is t_1 to t_2 are abstracted based on abstraction level L_3, the events which their timestamp is t_2 to t_3 are abstracted based on abstraction level L_2, the events which their timestamp is t_3 to t_4 are abstracted based on abstraction level L_2, the events which their timestamp is t_4 to t_5 are abstracted based on abstraction level L_1, and the events which their timestamp is t_5 to t_{max} are abstracted based on abstraction level L_0.

The mentioned event abstraction process reduces the number of objects, subjects, events, event relations, and consequently reduces the processing overhead. Although this abstraction approach reduces the detection accuracy in some situations, it is still a better solution for detecting slow and very slow attacks, in comparison with the other solutions, which eliminate the old events. For a better understanding, we consider the implicit file removal example, which is shown in Fig. 12 and described in Table 9. As

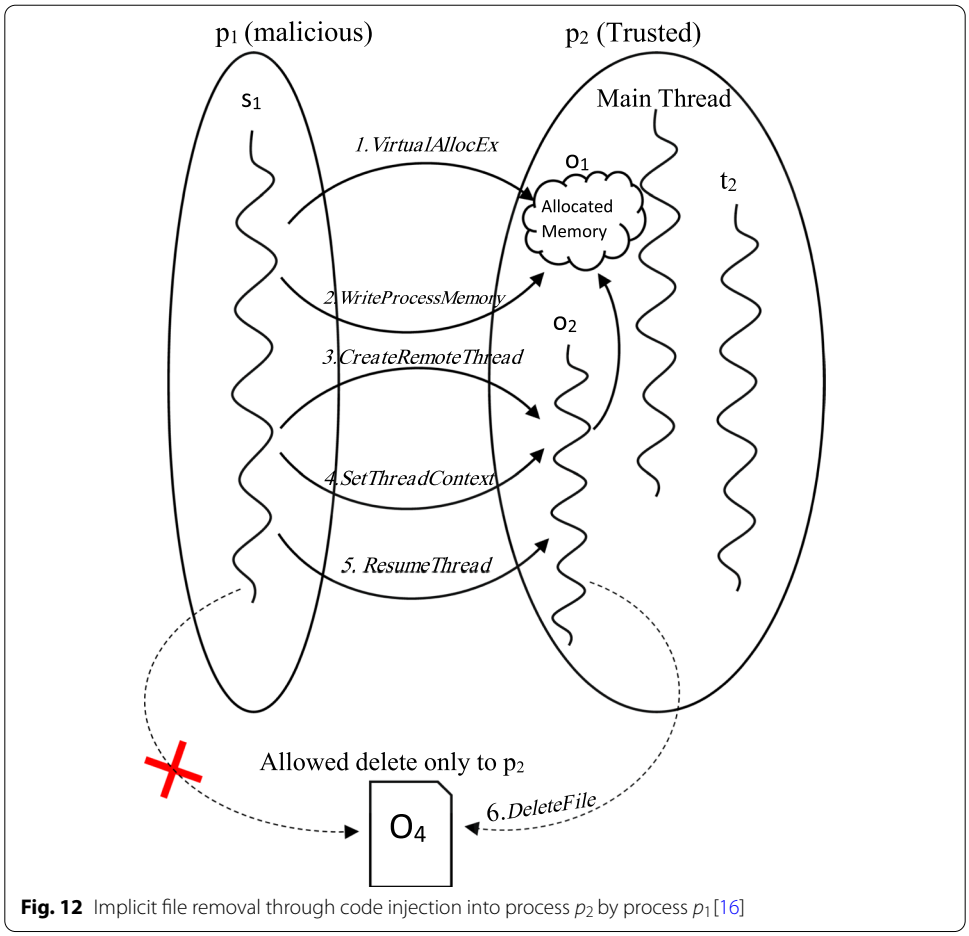

Fig. 12 Implicit file removal through code injection into process p_2 by process p_1 [16]

Table 9 Implicit file removal through code injection into process p_2 by process p_1 [16]

#	System calls	Description
1	*VirtualAllocEx*	$e_1 = \langle s_1, o_1, a_1, t_1 \rangle, a_1 = C, Thread(s_1), s_1 \xrightarrow{partOf} p_1, Memory(o_1), o_1 \xrightarrow{partOf} p_2$
2	*WriteProcessMemory*	$e_2 = \langle s_1, o_1, a_2, t_2 \rangle, a_2 = W, t_2 = t_1 + \epsilon$
3	*CreateRemoteThread*	$e_3 = \langle s_1, o_2, a_3, t_3 \rangle, a_3 = C, Thread(o_2), o_2 \xrightarrow{partOf} p_2, o_1 \xrightarrow{partOf} o_2, t_3 = t_2 + \epsilon$
4	*SetThreadContext*	$e_4 = \langle s_1, o_3, a_4, t_4 \rangle, a_4 = W, Context(o_3), o_3 \xrightarrow{partOf} o_2, t_4 = t_3 + \epsilon,$
5	*ResumeThread*	$e_5 = \langle s_1, o_2, a_5, t_5 \rangle, a_5 = E, t_5 = t_4 + \epsilon$
6	*DeleteFile*	$e_6 = \langle o_2, o_4, a_6, t_6 \rangle, a_6 = D, File(o_4), t_6 = t_5 + \epsilon$

Table 10 An example of events abstraction process

#	L_0	$L_1(R_1A)$	$L_2(L_1 \wedge R_1S)$	$L_3(L_2 \wedge R_1O)$	End of abstraction
1	$e_1 = \langle s_1, o_1, C, t_1 \rangle$	$e_1 = \langle s_1, o_1, \boldsymbol{W}, t_1 \rangle$	**(*Redundant*)**	–	–
2	$e_2 = \langle s_1, o_1, W, t_2 \rangle$	$e_2 = \langle s_1, o_1, W, t_2 \rangle$	$e_2 = \langle \boldsymbol{p_1}, o_1, W, t_2 \rangle$	$e_2 = \langle p_1, \boldsymbol{p_2}, W, t_2 \rangle$	**(*Redundant*)**
3	$e_3 = \langle s_1, o_2, C, t_3 \rangle$	$e_3 = \langle s_1, o_2, \boldsymbol{W}, t_3 \rangle$	$e_3 = \langle \boldsymbol{p_1}, o_2, W, t_3 \rangle$	$e_3 = \langle p_1, \boldsymbol{p_2}, W, t_3 \rangle$	**(*Redundant*)**
4	$e_4 = \langle s_1, o_3, W, t_4 \rangle$	$e_4 = \langle s_1, o_3, W, t_4 \rangle$	$e_4 = \langle \boldsymbol{p_1}, o_3, W, t_4 \rangle$	$e_4 = \langle p_1, \boldsymbol{p_2}, W, t_4 \rangle$	$e_4 = \langle p_1, p_2, W, t_4 \rangle$
5	$e_5 = \langle s_1, o_2, E, t_5 \rangle$	**(*Deleted*)**	–	–	–
6	$e_6 = \langle o_2, o_4, D, t_6 \rangle$	$e_6 = \langle o_2, o_4, \boldsymbol{W}, t_6 \rangle$	$e_6 = \langle \boldsymbol{p_2}, o_4, W, t_6 \rangle$	$e_6 = \langle p_2, \boldsymbol{o_4}, W, t_6 \rangle$	$e_6 = \langle p_2, o_4, W, t_6 \rangle$

shown in the figure and table, an implicit file removal through code injection into process p_2 by process p_1 has occurred. The process of events abstraction for this example is shown in Table 10. As shown in this table, we use four levels of abstraction for this example and abstraction levels L_4 to L_6 have no impact on reducing the number of events. Therefore, the seven events of this example can be abstracted into two events i.e., $e_4 = \langle p_1, p_2, W, t_4 \rangle$ and $e_6 = \langle p_2, o_4, W, t_6 \rangle$.

Other restrictions

If the rate of the event generation in a computer network is low and the size of the abstracted events is small, then the maximum time of the window can be increased. In such a situation, we can append more new events to the window and consider more events than the ones exist in the 12 months. These extra events are determined by S_r and T_r parameters which are determined using the following equations. Since S_r and T_r are dependent to $S_h(ES)$ and $T_h(ES)$ respectively and as mentioned before these two values are depended on specific event set ES, hence S_r and T_r are depended on ES. It is obvious that the mentioned situation appears when the S_r or T_r are greater than zero by the following equations.

$$S_r(ES) = S_{max} - S_{min}(ES) - S_h(ES) \tag{21}$$

$$T_r(ES) = T_{max}(ES) - T_{min} - T_h(ES). \tag{22}$$

Evaluation

In this section, we describe the dataset that is used for evaluating the proposed approach and the experimental results obtained from the evaluation.

Dataset

In recent years, several different dataset are proposed (e.g., Darpa98 [47], Darpa99 [48], Darpa2000 [49], KddCup [50], ISCX2012 [51], Defcon2015 [52], LBNL2016 [53], CIC-IDS2017 [54], IDS2018 [55], DDoS2019 [56], Darknet2020 [57]) for evaluating the intrusion detection and alert correlation systems (e.g., see [58]). Unfortunately, these available datasets have some drawbacks as follows, which make them inappropriate to be used for evaluating APT detection approaches.

- These datasets contain regular and simple attacks whereas APT attacks have many complex behaviors.
- These datasets do not contain any hybrid, low-level, and slow attacks, which are prevalent in APT attacks.
- These datasets do not contain any host-based event logs and mostly contain network-based logs and attacks, which are sufficient for APT detection.
- Attacks duration in these datasets are maximally limited to several weeks, which is not proper for evaluating the slow attacks.

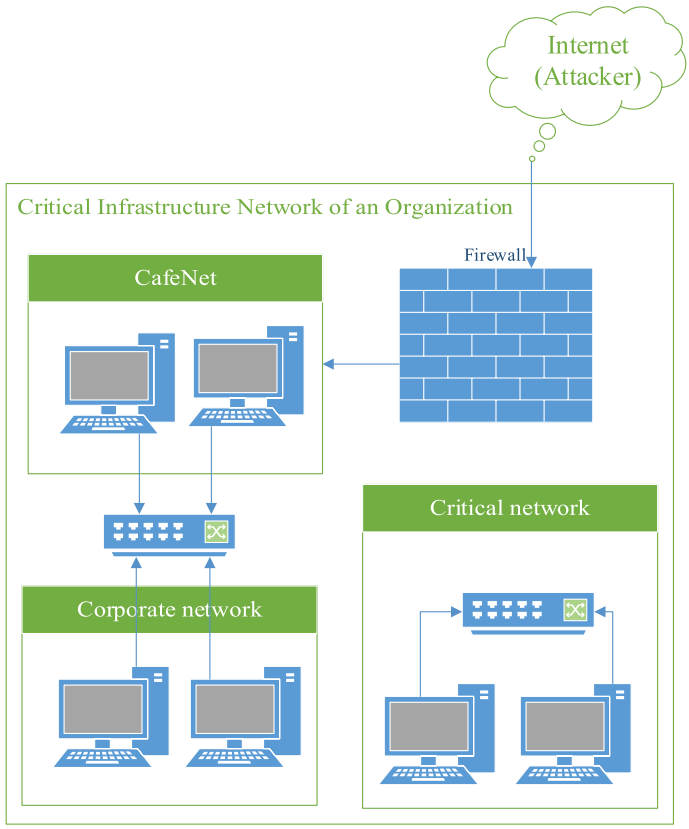

Fig. 13 The architecture of our test network

- The volume of these datasets is limited to several gigabytes, which is not proper for evaluating the scalability of detection approaches.

Due to the mentioned problems, we generated a new evaluation dataset, which has the following characteristics and is useful for evaluating our approach. This dataset is available online at [59].

- The architecture of the test bed that is used for creating this dataset is shown in Fig. 13.

  As shown in this figure, the test network contains four sub-networks. The first subnet is the Internet, which is the invasive way to the organization network. The second subnet is a *CafeNet*, which is connected to the Internet. The third one is *Corporate network*, which contains local services of an organization and is connected to *CafeNet*, and the fourth subnet is *Critical network*, which is an air-gapped network and isolated from the other networks.
- This dataset contains nine APT scenarios, which are shown in Table 11. These scenarios are the abstracted scenarios of some well-known APT samples, which are reported in [15]. Some APT scenarios of this table were implemented based on the available source codes, reversed codes, and published vulnerabilities of these

Table 11 APT scenarios in generated dataset

#	Adapted from	Attack type			Propagation channels	Purpose of function	Attack duration	Special feature
		Multi-step	Low-level	Slow				
1	Project Sauron APT [20]	✓	✓	✓	Internet and USB device	Sabotage	7 months	Bypass air-gapped network
2	Flame APT [62]	✓	✓	✓ (very slow)	Internet	Data theft	11 months	Low-level data exfiltration
3	Shamoon [63] and StoneDrill [64] APT	✓	✓	✓	Internet and LAN spreading	Data wiping	5 month	Code injection
4	WannaCry APT [65]	✓	✓	–	Internet	Ransomware	1 day	Exploits
5	Cloud Atlas [66] and Red October[67] APT	✓	✓	✓(very slow)	Internet and USB device	Data theft	15 months	Exploits, social engineering
6	Cloud Atlas [66] and Red October [67] APT	✓	✓	–	Internet and USB device	Data theft	1 month	Exploits, social engineering
7	Poseidon APT [68]	✓	✓	–	Internet	Remote control	1 month	Backdoor and code injection
8	Dark hotel [69]	✓	✓	✓ (very slow)	Internet	Surveillance	14 months	Stolen digital certificates
9	Dark hotel [69]	✓	✓	✓	Internet	Surveillance	9 months	Stolen digital certificates

malwares on the Internet. Then, all APTs were run in the testbed. Some were run concurrently, and some, with overlapping scenarios, were run asynchronously.

- The behaviors of malwares are intercepted in the operating system in two different ways. The kernel events are intercepted by implementing a Windows driver for hooking and a mini-filter driver for using call-back functions. The user events are intercepted by Easy hook library [60] through the code injection. The interception in hypervisor level is implemented by customizing a version of Ether [61] on Xen hypervisor. Also, the network events are collected by switch port mirroring.
- The volume of the dataset is approximately 2 Terabytes.
- The dataset contains low-level, and slow attacks.
- The dataset contains both the network and host event logs.
- The total number of intercepted events in the test network is about 1.646 billion events.
- We use seven hosts (one of which belonged to the attacker) for the simulation and supposed one user per host.
- Different attacks with different duration times were considered. We deployed one short attack with one-day duration, two almost slow attacks with one-month dura-

tion, four slow attacks with five, seven, and eleven months duration, and two very slow attacks with fourteen and fifteen months duration.

- The simulated network contained 110 benign processes and 9 attack vectors. Each attack vector contained several sub-processes. Our dataset contains the operating system and network event logs for all processes.
- The normal behaviors were generated by the real users in CafeNet and Corporate networks. Since the actions in Critical networks do not have many dependencies on the users, the normal behaviors of such networks were simulated by some softwares, which were running without interacting with users.

Experimental results

After deploying the reconstructed attacks in our test network and creating the dataset, we evaluated our proposed approach using the generated dataset. To this aim:

- We used the ontology that was proposed in [16] and implemented by *WinDbg* tool [70] and Microsoft MSDN library [71].
- We use *OWL-DL* language [44] for specification of the system ontology and *Jena* language [72] for specification of user-defined inference rules. We employed *SANSA* for loading and saving OWL files, querying, and reasoning based on Description Logic.
- The processing time of the proposed approach for detecting hybrid and low-level APTs in the test networks was 6.1 hours.

The experimental results of evaluating our proposed approach are shown in Table 12. As shown in this table, there are 110 benign processes and 9 malicious or attack vectors performed by different processes. Based on the evaluation results, the *accuracy* rate is 89.07%. However, since the events which are generated by APT attacks are very rare, the two classes of events (i.e., APT and Benign) are imbalanced, and using *accuracy* and *precision* as the evaluation criteria is not reliable. Hence in these cases using other criteria such as *sensitivity* and *specificity* are more valuable and informative.

Specificity or *true negative rate*, determines the ability of our approach to detect benign samples. In other words, specificity measures the rate of the detected benign samples that are truly benign. Sensitivity or *true positive rate* or *recall* determines the ability of our approach in detecting the APT samples. In other words, sensitivity measures the rate of the detected malicious samples that are truly APTs.

Table 12 Results of evaluation

Actual class\Predicted class	APT	Benign	Total	Recognition(%)
APT	8(TP)	1 (FN)	9(P)	88.88 (Sensitivity= $\frac{TP}{P}$)
Benign	12(FP)	98(TN)	110(N)	89.09 (Specificity= $\frac{TN}{N}$)
Total	20	99	119	89.07(Accuracy= $\frac{TP+TN}{P+N}$)
Other metrics(%)	40.00 (Precision= $\frac{TP}{TP+FP}$)	88.88 (Recall= $\frac{TP}{TP+FN}$)	55.17 (F-Measure)	10.92 (Error-rate= $\frac{FP+FN}{P+N}$)

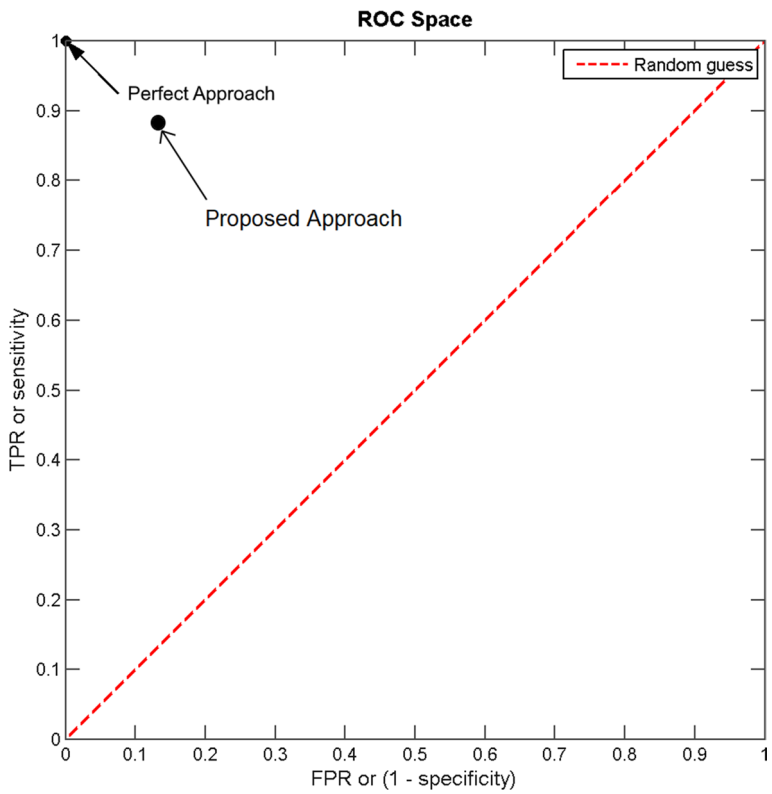

Fig. 14 The coordinate of our approach in ROC space

Table 13 Results of evaluation per each APT sample of Table 11

APT sample	Number of APT events (P)	Number of other events (N)	TPR (%)	TNR (%)	Accuracy (%)	Precision (%)	Detection result
1	9.1 million	1.637 billion	86.29	90.14	90.08	4.59	APT
2	13.7 million	1.6322 billion	83.17	87.58	87.50	5.24	APT
3	6.3 million	1.6397 billion	94.9	89.94	89.91	3.47	APT
4	73 thousand	1.64527 billion	95.06	97.48	97.39	0.16	APT
5	26 million	1.62 billion	98.32	77.97	78.28	6.67	APT
6	1.6 million	1.6444 billion	81.02	95.1	94.85	1.50	APT
7	2.1 million	1.6439 billion	91.73	90.38	90.33	1.19	APT
8	27.7 million	1.6183 billion	60.33	89.87	89.35	8.70	Benign
9	10.1 million	1.6359 billion	94.13	97.80	97.17	16.96	APT

Table 14 Results of comparing two APT detection approaches

Method	Accuracy	Sensitivity	Specificity	Precision
Lajevardi & Amini [16]	77.31	44.44	80.00	15.38
Our proposed approach	89.07	88.88	89.09	40.00

According to the values that are shown in Table 12, we can draw the receiver operating characteristic (ROC) curve for our approach. As shown in Fig. 14, the coordinate of our approach in the ROC curve is (0.8888, 0.1091). This point in the ROC curve means any other solution for detecting APT attacks, which are evaluated based on our proposed dataset, should try to increase the sensitivity while the specificity is fixed or become more. In other words, any new solution should try to be placed near point (0,1) in the ROC space.

Also, the experimental results of our approach per each APT scenario, which was described in Table 11, are shown in Table 13. As shown in this table, since our dataset is unbalanced and the number of malicious events versus the benign ones is very small, the values of accuracy and precision are unrealistic and useless.

To evaluate more accurately, we reevaluate and compared the proposed approach in [16] with the proposed approach in this paper using our new generated dataset, which contains slow attacks. As shown in Table 14, since the approach proposed in [16] can just detect low and hybrid attacks and cannot detect the slow ones, the sensitivity of the prior approach [16] is too low (44.44 %) in comparison with the sensitivity of our proposed approach (88.88 %) using this new dataset.

Discussion

APT detection is a new challenge in the field of computer security and the problem of detecting low and slow APTs is a very new challenge in the field of malware analysis, hence the number of publications about detecting APT attacks is rare. Since we cannot find any other approaches focusing on low-level and slow APTs and to the best of our knowledge, our proposed approach is the first solution for detecting this type of APTs, we cannot use quantitative comparison between our approach and the other correlation approaches. However, since the related works use alert correlation methods for detecting APT attacks, we do a qualitative comparison (shown in Table 15) between our approach and the other correlation approaches. As shown in this table, the main drawback of all previous works, even our previous work [16], is the lack of detecting slow attacks. Our approach can detect multi-step, hybrid, low-level, and slow attacks.

As mentioned before, the average batch processing time in our proposed approach to detect the slow attacks was about 6.1 hours. This time depends on several parameters as follows, and can be improved by the following considerations.

- Correlation algorithms: Since the solution for detecting low-level APTs is based on reasoning, the processing time depends on the reasoning algorithm. We use *OWL-DL* (like a solution in [93]) for reasoning by *SANSA*. In this situation, the complexity class of reasoning is ExpTime-Complete. Since *SANSA* inference engine uses *OWL-Horst* [94] forward chaining inference for reasoning and we do not use all features of basic description logic (Attributive Language with Complements), we can use *OWL-Horst* instead of *OWL-DL* to reduce the reasoning time. In this situation the complexity class of reasoning is Nondeterministic Polynomial Complete (NP-Complete) and in a nontrivial case, it is Polynomial [94].
- SANSA framework: To detect slow APTs, we encounter with analyzing a big number of frames (about several billion frames). Since we use SANSA as an inference engine,

Table 15 A subjective comparison between the proposed approach and other correlation methods

Method	Attack detection method			Attack type		
	Correlation type	Alert causal analysis	Hybrid correlation[a]	Multi-step attack detection	Low-level attack detection	Slow attack detection
Debar and Wespi [73]	Alert Correlation	–	–	–	–	–
Valeur et al. [74]	Alert Correlation	–	–	✓	–	–
Wang and Chiou [75]	Alert Correlation	✓	–	–	–	–
Valdes and Skinner [76]	Alert Correlation	–	–	✓	–	–
Julisch 2001 [77]	Alert Correlation	✓	–	–	–	–
Julisch 2003 [78]	Alert Correlation	✓	–	–	–	–
Al-Mamory and Zhang [79]	Alert Correlation	✓	–	✓	–	–
Peng et al. [80]	Alert Correlation	✓	–	–	–	–
Qin and Lee [81]	Alert Correlation	✓	–	✓	–	–
Goldman et al. [82]	Alert Correlation	✓	–	✓	–	–
Viinikka et al. [83]	Alert Correlation	–	–	–	–	–
Treinen and Thurimella [84]	Alert Correlation	✓	–	✓	–	–
Ourston et al. [41]	Alert Correlation	–	–	✓	–	–
Ren et al. [85]	Alert Correlation	✓	–	✓	–	–
Zhitang et al. [86]	Alert Correlation	–	–	✓	–	–
Ma et al. [87]	Alert Correlation	–	–	✓	–	–
Zhitang et al. [88]	Alert Correlation	–	–	✓	–	–
Farhadi et al. [89]	Alert Correlation	✓	–	✓	–	–
Manganiello et al. [90]	Alert Correlation	✓	–	✓	–	–
Soleimani and Ghorbani [91]	Alert Correlation	–	–	✓	–	–
Ramaki et al. [92]	Alert Correlation	✓	–	✓	–	–
Ghafir et al. [40]	Alert Correlation	–	–	✓	–	–
Lajevardi and Amini [16]	Event Correlation	✓	✓	✓	✓	–
Mohamed and Belaton [38]	Alert Correlation	✓	–	✓	–	–
Our proposed approach	Event Correlation	✓	✓	✓	✓	✓

[a] Hybrid correlation means correlating operating system events with network events

the processing time is highly dependent on the processing power of SANSA. Our experience shows SANSA can analyze several million frames in some seconds.

- Processing infrastructure: Since SANSA uses *Spark* technology to process the big size of data, the infrastructure that is used by *Spark* has a key role in the processing time. We deployed our approach on a cluster with 8 computing nodes and 80 cores, 1 Terabyte of RAM, and an NFS (Network File System) server node with 10 Terabyte capacity. The operating system was Rocks cluster 7.

Conclusion

Targeted cyberattacks, which are known as APTs, have some characteristics such as low-level, slow, multi-step, distributed, and hybrid. Since the most complex APT attacks are low-level and slow, in this paper, we focus on this type of APT attack. Our approach uses a scalable knowledge-based system, and semantic correlation following the enhanced version of the approach we proposed in [16] for detecting the low-level as well as slow APTs. In our approach, we use a sliding window called *Vermiform* window. This window has two steps or phases in its movement: *expanding* and *shrinking*. In *expanding*, we use a scalable inference engine called *SANSA* for correlating a big size of events based on big data frameworks such as Spark. In some situation that the APT attack duration last a very long time or the number of intercepted events is very much, we use the shrinking process. In shrinking, the events are abstracted and reduced and in fact, we create an abstract history from the old events. This solution is different from the regular approaches which eliminate the old events completely.

To evaluate the proposed approach, we use a dataset that contains seven implemented low-level and slow APT attacks. The proposed approach shows 88.88% of sensitivity and 89.09% of specificity.

To the best of our knowledge, our approach is the first solution for detecting slow APTs which whose duration is about several months. However, there are still many opportunities for innovations in the domain of APT attacks. We believe that the following works could continue and complete our research on detecting APT attacks with more accuracy:

- One of the main challenges in the field of APTs is attack prediction. Our approach in this paper cannot predict the APT attacks before the malware fulfills its malicious activity. Using machine learning algorithms, we can predict malicious activities before an APT completes its malicious activities.
- The proposed approach has no solution to detect the APT attacks that violate the availability of the system and propose an approach for solving this weakness can be considered as future work.
- The proposed approach has no solution to verify the generated alerts. Alert verification can help us to reduce the number of false alerts.
- Another future work is to improve and release our dataset to be used by other APT detection approaches to have a better and precise evaluation result.
- Proposing a stream-based approach for detecting the APT attacks that are not slow is another research topic, which could be considered in the future.

It is worthwhile to note that the proposed approach can be applied on different network structures such as IT networks, smart grids, microgrids [95], and even IoT networks.

Abbreviations

The list of abbreviations and the symbols used in the paper are listed in Table 16 in "List of the symbols used in the paper" section.

Appendix

List of the symbols used in the paper

The symbols, which are defined and used in this paper, are listed in Table 16.

Table 16 List of the symbols used in the paper

Symbol	Category	Description
Event	Class	Event type (of all possible events)
Subject	Class	Subject type (of all possible subjects)
Object	Class	Object type (of all possible objects)
Action	Class	Action type (of all possible actions)
$subject : Event^{\mathcal{I}} \longrightarrow Subject^{\mathcal{I}}$	Function	Determines the subject of an event
$object : Event^{\mathcal{I}} \longrightarrow Object^{\mathcal{I}}$	Function	Determines the object of an event
$action : Event^{\mathcal{I}} \longrightarrow \{R, W\}$	Function	Determines the action of an event
$time : Event^{\mathcal{I}} \longrightarrow \mathbb{N}$	Function	Determines the timestamp of an event
$\sim$	Relation	Event relation
$ES \subseteq Event^{\mathcal{I}}$	Set	Suspicious event set
$SP \subseteq Event^{\mathcal{I}}$	Set	Set of all unauthorized events
$I : Event^{\mathcal{I}} \longrightarrow Event^{\mathcal{I}}$	Function	Effect function
$\sqsubseteq$		Subsumption
$\xrightarrow{partOf}$	Relation	Part of relation
W	Individual	Abbr. of Write
R	Individual	Abbr. of Read
ν_i	Set	Attack vector
ν	Set	Set of all attack vectors
$f : Event \times \mathbb{N} \rightarrow \mathbb{N}$	Function	Specifies the number of events in a specific event set which have a specific timestamp
$me : Subject^{\mathcal{I}} \longrightarrow \mathcal{P}(Object^{\mathcal{I}})$	Function	Determines the memory of a specific subject
$ma : Subject^{\mathcal{I}} \longrightarrow \mathcal{P}(Object^{\mathcal{I}})$	Function	Determines the objects that are changed by a specific subject
$\Delta : \mathcal{P}(E) \rightarrow \mathcal{P}(E)$	Function	Abstracting function
WE	Set	Set of events
AE	Set	Set of abstracted events
APT		Abbr. of Advanced Persistent Threat
SWRL		Abbr. of Semantic Web Rule Language
IPC		Abbr. of Inter-Process Communication
DLP		Abbr. of Data Loss Prevention
OWL		Abbr. of Ontology Web Language
DL		Abbr. of Description Logic
IDMEF		Abbr. of Intrusion Detection Message Exchange Format
SANSA		Abbr. of Scalable Semantic Analytics Stack
ROC		Abbr. of Receiver Operating Characteristic
SSDT		S ystem service dispatch table

Syntax and semantics of description logic

Part of syntax and semantics of description logic (DL), which is used in our research, are shown in Table 17.

Table 17 Syntax and semantics of description logic [18]

Syntax	Description	Semantics
C, D	Concepts or Classes	$C^{\mathcal{I}}, D^{\mathcal{I}} \subseteq \Theta$
R, S	Relations or Properties	$R^{\mathcal{I}}, S^{\mathcal{I}} \subseteq \Theta \times \Theta$
$C \sqsubseteq D$	Class subsumption	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$R \sqsubseteq S$	Property subsumption	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
$C \sqcup D$	Union of classes	$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$C \sqcap D$	Intersection of classes	$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\neg C$	Complement	$(\neg C)^{\mathcal{I}} = \Theta - C^{\mathcal{I}}$
$\exists R.C$	Existential quantification	$(\exists R.C)^{\mathcal{I}} = \{a \in \Theta \mid \exists b, \langle a, b\rangle \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
$\forall R.C$	Value restriction	$(\forall R.C)^{\mathcal{I}} = \{a \in \Theta \mid \forall b, \langle a, b\rangle \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
$\top$	Top concept or Thing	$\top^{\mathcal{I}} = \Theta$
$\bot$	Bottom concept or Nothing	$\bot^{\mathcal{I}} = \varnothing$

- The model is $\mathcal{M} = \langle \Theta, .^{\mathcal{I}} \rangle$, where

Θ is the domain of objects, and

$\mathcal{I}$ is an interpretation function

Authors' information

Amir M. Lajevardi received the Ph.D. degrees in the computer software engineering Sharif University ofTechnology, Tehran, Iran, and now he is a postdoctoral research associate in the Department of ComputerEngineering at the Sharif University of Technology. His research interests are intrusion detection systems,malware detection, alert correlation, and operating system security. Morteza Amini is currently an associate professor at the Department of Computer Engineering, SharifUniversity of Technology, Tehran, Iran and he is also one of the directors of Data and Network Security Lab(DNSL) in this department. His research interests include database security, access control, intrusion detectionsystems, and formal methods in information security.

Acknowledgements

This article is supported and funded by a research grant from the Iran National Science Foundation (INSF) and Iran's National Elite Foundation with the grant number 99020686.

Author's contributions

AML research principal in this work as well as the technical issues. MA advise all process for this work. All authors read and approved the final manuscript.

Funding

This article is supported and funded by a research grant from the Iran National Science Foundation (INSF) and Iran's National Elite Foundation with the grant number 99020686.

Availability of data and materials

The datasets for this study are available on request to the corresponding author.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. [2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

Received: 23 July 2021 Accepted: 23 October 2021

Published online: 27 November 2021

References

1. Jeun, I, Lee, Y, Won D. A practical study on advanced persistent threats. Comput Appl Secur Control Syst Eng. 2012;144–152 (Chap. 21).
2. Zhang, Q, Li, H, Hu, J. A study on security framework against advanced persistent threat. In: Proceedings of 2017 IEEE 7th International Conference on Electronics Information and Emergency Communication, ICEIEC 2017, 2017; pp. 128–131. https://doi.org/10.1109/ICEIEC.2017.8076527
3. Cole E. Advanced persistent threat: understanding the danger and how to protect your organization. 2012. p. 320.
4. Auty M. Anatomy of an advanced persistent threat. Netw Secur. 2015;4(4):13–6.
5. Chen P, Desmet L, Huygens C. A study on advanced persistent threats. In: Conference on Communications and Multimedia Security, 2014; pp. 63–72
6. Ghafir I, Prenosil V. Advanced persistent threat attack detection: an overview. 2014;4(4):1–5.
7. Tankard C. Advanced persistent threats and how to monitor and deter them. Netw Secur. 2011;2011(8):16–9.
8. Thonnard O, Bilge, L O'Gorman, G Kiernan, S Lee, M. Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7462 LNCS, 2012;pp. 64–85. https://doi.org/10.1007/978-3-642-33338-5_4
9. Thomson G. APTs: a poorly understood challenge. Netw Secur. 2011;2011(11):9–11.
10. Brewer R. Advanced persistent threats: minimising the damage. Netw Secur. 2014;2014(4):5–9.
11. Virvilis N, Gritzalis D, Apostolopoulos T. Trusted computing vs. Advanced persistent threats: Can a defender win this game? In: International Conference on Autonomic and Trusted Computing, pp. 2013;396–403
12. Marchetti M, Pierazzi F, Colajanni M, Guido A. Analysis of high volumes of network traffic for advanced persistent threat detection. Comput Netw. 2016;109(2):127–41.
13. Lemay A, Calvet J, Menet F, Fernandez JM. Survey of publicly available reports on advanced persistent threat actors. Comput Secur. 2018;72:26–59.
14. Chen J, Su C, Yeh KH, Yung M. Special issue on advanced persistent threat. Future Gen Comput Syst. 2018;79:243–6.
15. Kaspersky: Targeted cyberattacks logbook. https://apt.securelist.com Accessed 2021-11-27.
16. Lajevardi AM, Amini M. A semantic-based correlation approach for detecting hybrid and low-level APTs. Future Generat Comput Syst. 2019;96:64–88.
17. Lehmann J, Sejdiu G, Bühmann L, Westphal P, Stadler C, Ermilov I, Bin S, Chakraborty N, Saleem M, Ngonga Ngomo AC, Jabeen H. Distributed semantic analytics using the SANSA stack. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10588 LNCS, 2017;pp. 147–155. Springer. https://doi.org/10.1007/978-3-319-68204-4_15
18. Mann CJH. The description logic handbook-theory, implementation and applications. Kybernetes. 2003;32(9/10). https://doi.org/10.1108/k.2003.06732iae.006.
19. Matrosov A, Rodionov E, Harley D, Malcho J. Stuxnet under the microscope. ESET LLC. 2010.
20. Global Research and Analysis Team: The ProjectSauron APT. Kaspersky Lab 2016;02, 1–23.
21. US9628507B2-Advanced persistent threat (APT) detection center-Google Patents. https://patents.google.com/patent/US9628507B2/en. Accessed 2020-02-05.
22. Balduzzi M, Ciangaglini V, McArdle R. Targeted attacks detection with SPuNge. In: International Conference on Privacy, Security and Trust, 2013;pp. 185–194
23. Liu ST, Chen YM, Lin SJ. A novel search engine to uncover potential victims for APT investigations. Lecture Notes in Computer Science. 2013;405–416 (Chap. 34).
24. Quader F, Janeja V, Stauffer J. Persistent threat pattern discovery. In: IEEE International Conference on Intelligence and Security Informatics, 2015;pp. 179–181.
25. Zhao G, Xu K, Xu L, Wu B. Detecting APT malware infections based on malicious DNS and traffic analysis. IEEE Access. 2015;3:1132–42.
26. Niu W, Zhan X, Li K, Yang G, Chen R. Modeling attack process of advanced persistent threat. In: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, 2016;pp. 383–391.
27. Friedberg I, Skopik F, Settanni G, Fiedler R. Combating advanced persistent threats: From network event correlation to incident detection. Comput Secur. 2015;48:35–57.
28. Bhatt P, Yano E, Gustavsson P. Towards a framework to detect multi-stage advanced persistent threats attacks. In: International Symposium on Service Oriented System Engineering, 2014;pp. 390–395.
29. Haopu, Y. Method for behavior-prediction of APT attack based on dynamic Bayesian game. In: Proceedings of 2016 IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2016, 2016;pp. 177–182. https://doi.org/10.1109/ICCCBDA.2016.7529554
30. Giura, P, Wang, W. A context-based detection framework for advanced persistent threats. In: Proceedings of the 2012 ASE International Conference on Cyber Security, CyberSecurity 2012, 2012;pp. 69–74. https://doi.org/10.1109/CyberSecurity.2012.16.
31. Moon D, Im H, Kim I, Park JH. DTB-IDS: an intrusion detection system based on decision tree using behavior analysis for preventing APT attacks. J Supercomput. 2017;73(7):2881–95.
32. Das A, Shen MY, Shashanka M, Wang J. Detection of exfiltration and tunneling over DNS. In: Proceedings - 16th IEEE International Conference on Machine Learning and Applications, ICMLA 2017, vol. 2017-Decem, 2017;pp. 737–742. https://doi.org/10.1109/ICMLA.2017.00-71.
33. Debatty T, Mees W, Gilon T. Graph-based APT detection. In: 2018 International Conference on Military Communications and Information Systems, ICMCIS 2018, 2018;pp. 1–8. https://doi.org/10.1109/ICMCIS.2018.8398708.
34. Joloudari JH, Haderbadi M, Mashmool A, Ghasemigol M, Band SS, Mosavi A. Early detection of the advanced persistent threat attack using performance analysis of deep learning. IEEE Access. 2020;8:186125–37. https://doi.org/10.1109/ACCESS.2020.3029202.
35. Zimba A, Chen H, Wang Z, Chishimba M. Modeling and detection of the multi-stages of Advanced Persistent Threats attacks based on semi-supervised learning and complex networks characteristics. Future Generat Comput Syst. 2020;106:501–17. https://doi.org/10.1016/j.future.2020.01.032.

36. Xiang Z, Guo D, Li Q. Detecting mobile advanced persistent threats based on large-scale DNS logs. Computers and Security. 2020;96. https://doi.org/10.1016/j.cose.2020.101933.
37. Shang L, Guo D, Ji Y, Li Q. Discovering unknown advanced persistent threat using shared features mined by neural networks. Comput Netw. 2021;189:107937. https://doi.org/10.1016/j.comnet.2021.107937.
38. Mohamed N, Belaton B. SBI model for the detection of advanced persistent threat based on strange behavior of using credential dumping technique. IEEE Access. 2021;9:42919–32. https://doi.org/10.1109/ACCESS.2021.3066289.
39. Brogi G, Tong VVT. TerminAPTor: Highlighting advanced persistent threats through information flow tracking. In: International Conference on New Technologies, Mobility and Security, 2016;pp. 1–5.
40. Ghafir I, Hammoudeh M, Prenosil V, Han L, Hegarty R, Rabie K, Aparicio-Navarro FJ. Detection of advanced persistent threat using machine-learning correlation analysis. Future Generat Comput Syst. 2018;89:349–59.
41. Ourston D, Matzner S, Stump W, Hopkins B. Applications of hidden Markov models to detecting multi-stage network attacks. In: Proceedings of Conference on System Sciences, 2003;pp. 1–10.
42. Fensel D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PFOIL. An ontology infrastructure for the semantic web. IEEE Intell Syst. 2001;16(2):38–45.
43. Costa R, Cachulo N, Cortez P. An intelligent alarm management system for large-scale telecommunication companies. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5816 LNAI, 2009;pp. 386–399. https://doi.org/10.1007/978-3-642-04686-5_32.
44. McGuinness DL, Van H, Frank. OWL web ontology language overview. W3C Recommend. 2004;10(10):1–22.
45. Agarwal M, Purwar S, Biswas S, Nandi S. Intrusion detection system for PS-Poll DoS attack in 802.11 networks using real time discrete event system. IEEE/CAA J Automat Sin. 2017;4(4):792–808. https://doi.org/10.1109/JAS.2016.7510178.
46. Reasoners and rule engines: Jena inference support. https://jena.apache.org/documentation/inference/ Accessed 2019-02-12.
47. 1998 DARPA Intrusion Detection Evaluation Data Set. https://www.ll.mit.edu/ideval/data/1998data.html Accessed 2019-04-03.
48. 1999 DARPA Intrusion Detection Evaluation Data Set. https://www.ll.mit.edu/ideval/data/1999data.html Accessed 2021-04-03.
49. 2000 DARPA Intrusion Detection Scenario Specific Data Sets. https://www.ll.mit.edu/ideval/data/2000data.html Accessed 2021-04-03.
50. KDD Cup Archives. http://www.kdd.org/kdd-cup Accessed 2016-04-03.
51. Shiravi A, Shiravi H, Tavallaee M, Ghorbani AA. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput Secur. 2012;31(3):357–74.
52. Defcon. https://www.defcon.org/ Accessed 2021-04-03.
53. LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing/Overview.html Accessed 2019-04-03.
54. Sharafaldin I, Habibi Lashkari A, Ghorbani AA. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In: International Conference on Information Systems Security and Privacy, 2018;pp. 108–116.
55. Sharafaldin, Iman, Lashkari, Arash Habibi, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSp, 2018;pp. 108–116.
56. Sharafaldin I, Lashkari AH, Hakak S, Ghorbani AA. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In: Proceedings - International Carnahan Conference on Security Technology, vol. 2019-Octob 2019. https://doi.org/10.1109/CCST.2019.8888419. https://ieeexplore.ieee.org/abstract/document/8888419/.
57. Habibi Lashkari A, Kaur G, Rahali A. In: DIDarknet: DIDarknet: A contemporary approach to detect and characterize the darknet traffic using deep image learning. In: ACM International Conference Proceeding Series, pp. 1–13. Association for Computing Machinery, (2020). https://doi.org/10.1145/3442520.3442521.
58. Teng S, Wu N, Zhu H, Teng L, Zhang W. SVM-DT-based adaptive and collaborative intrusion detection. IEEE/CAA J Automat Sin. 2018;5(1):108–18. https://doi.org/10.1109/JAS.2017.7510730.
59. APT Test Dataset. http://www.ce.sharif.edu/lajevardi/APTDataset Accessed 2021-05-04.
60. EasyHook. https://easyhook.github.io/ Accessed 2020-03-03.
61. Ether: Malware Analysis via Hardware Virtualization Extensions. http://ether.gtisc.gatech.edu/source.html Accessed 2021-03-03.
62. The Flame: Questions and Answers. https://securelist.com/blog/incidents/34344/the-flame-questions-and-answers-51/ Accessed 2021-02-03.
63. Global Research and Analysis Team (GReAT): Shamoon the Wiper – Copycats at Work. https://securelist.com/shamoon-the-wiper-copycats-at-work/57854/ Accessed 2021-04-04.
64. Global Research and Analysis Team (GReAT): From Shamoon to StoneDrill. https://securelist.com/from-shamoon-to-stonedrill/77725/ Accessed 2021-04-04.
65. Global Research and Analysis Team (GReAT): WannaCry ransomware used in widespread attacks all over the world. https://securelist.com/wannacry-ransomware-used-in-widespread-attacks-all-over-the-world/78351/ Accessed 2021-04-04.
66. Global Research and Analysis Team (GReAT): Cloud Atlas: RedOctober APT is back in style. https://securelist.com/cloud-atlas-redoctober-apt-is-back-in-style/68083/ Accessed 2021-04-04.
67. Global Research and Analysis Team (GReAT): Red October Diplomatic Cyber Attacks Investigation. https://securelist.com/red-october-diplomatic-cyber-attacks-investigation/36740/ Accessed 2021-04-04.
68. Global Research and Analysis Team (GReAT): Poseidon Group: a Targeted Attack Boutique specializing in global cyber-espionage. https://securelist.com/poseidon-group-a-targeted-attack-boutique-specializing-in-global-cyber-espionage/73673/ Accessed 2021-04-04.
69. Global Research and Analysis Team (GReAT): The Darkhotel APT. https://securelist.com/the-darkhotel-apt/66779/ Accessed 2021-04-04.
70. Microsoft WinDbg. https://developer.microsoft.com/en-us/windows/hardware/download-windbg Accessed 2017-04-02.
71. Microsoft MSDN. https://msdn.microsoft.com/library Accessed 2019-04-03.

72. Horrocks I, Patel-schneider PF, Boley H, Tabet S, Grosof B, Dean M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member submission 21. 2004;21(79):1–20.
73. Debar H, Wespi A. Aggregation and Correlation of Intrusion Detection Alerts. In: International Workshop on Recent Advances in Intrusion Detection, 2001;vol. 2212, pp. 85–103.
74. Valeur F, Vigna G, Kruegel C, Kemmerer RA. A comprehensive approach to intrusion detection alert correlation. Dependable and Secure. Computing. 2004;1(3):146–68.
75. Wang C-H, Chiou Y-C. Alert correlation system with automatic extraction of attack strategies by using dynamic feature weights. Comput Commun Eng. 2016;5(1):1–10.
76. Valdes A, Skinner K. Probabilistic alert correlation. In: International Workshop on Recent Advances in Intrusion Detection, 2001;pp. 54–68.
77. Julisch K. Mining alarm clusters to improve alarm handling efficiency. in Proceedings of Annual Computer Security Applications Conference, 2001;12–21.
78. Julisch. Clustering intrusion detection alarms to support root cause analysis. ACM Trans Inform Syst Secur. 2003;6(4):443–71.
79. Al-Mamory SO, Zhang H. IDs alerts correlation using grammar-based approach. Comput Virol. 2009;5(4):271–82.
80. Peng X, Zhang Y, Xiao S, Zheng W, Cui JQ, Chen L, Xiao D. An alert correlation method based on improved cluster algorithm. In: Workshop on Computational Intelligence and Industrial Application, 2008; vol. 1, pp. 342–347.
81. Qin X, Lee W. Attack plan recognition and prediction using causal networks. In: Conference on Computer Security Applications, 2004; pp. 370–379
82. Goldman RP, Heimerdinger W, Harp SA, Geib CW, Thomas V, Carter RL. Information modeling for intrusion report aggregation. In: Proceedings of DARPA Information Survivability Conference and Exposition, 2001;vol. 1, pp. 329–342.
83. Viinikka J, Debar H, Mé L, Séguier R. Time series modeling for IDS alert management. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security, 2006; pp. 102–113.
84. Treinen JJ, Thurimella RA. Framework for the Application of Association Rule Mining in Large Intrusion Detection. In: Workshop on Recent Advances in Intrusion Detection, 2006;pp. 1–18.
85. Ren H, Stakhanova N, Ghorbani AA. An Online Adaptive Approach to Alert Correlation. In: Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2010;pp. 153–172.
86. Zhitang L, Aifang Z, Jie L, Li W. Real-time correlation of network security alerts. In: Proceedings of Conference on e-Business Engineering, 2007;pp. 73–80.
87. Jie M, Li ZT, Li WM. Real-time alert stream clustering and correlation for discovering attack strategies. In: Proceedings of the 5th International Conference on Fuzzy Systems and Knowledge Discovery, 2008;vol. 4, pp. 379–384.
88. Li Z, Zhang A, Li D, Wang L. Discovering novel multistage attack strategies. In: Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2007;vol. 4632 LNAI, pp. 45–56. Springer, https://doi.org/10.1007/978-3-540-73871-8_6.
89. Farhadi H, Amirhaeri M, Khansari M. Alert correlation and prediction using data mining and HMM. ISC J Inform Secur (ISeCure). 2015;3(2):77–101.
90. Manganiello F, Marchetti M, Colajanni M. Multistep attack detection and alert correlation in intrusion detection systems. In: Conference on Information Security and Assurance, vol. 200, 2011;pp. 101–110.
91. Soleimani M, Ghorbani AA. Multi-layer episode filtering for the multi-step attack detection. Comput Commun. 2012;35(11):1368–79.
92. Ramaki AA, Amini M, Ebrahimi Atani R. RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection. Comput Secur. 2015;49:206–19.
93. Westphal P, Fernández JD, Kirrane S, Lehmann JSPIRIT. A semantic transparency and compliance stack. In: CEUR Workshop Proceedings. 2018;2198.
94. Ter Horst HJ. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Web Semant Sci Serv Agents World Wide Web. 2005;3(2–3):79–115.
95. Rana MM, Li L, Su SW. Cyber attack protection and control of microgrids. IEEE/CAA J Automat Sin. 2018;5(2):602–9. https://doi.org/10.1109/JAS.2017.7510655.

Publisher's Note