IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

Detecting new generations of threats using attribute-based attack graphs

ISSN 1751-8709
Received on 1st August 2018
Revised 19th November 2018
Accepted on 30th January 2019
E-First on 20th February 2019
doi: 10.1049/iet-ifs.2018.5409
www.ietdl.org

Mehran Alidoost Nia[1] ✉, Behnam Bahrak[1], Mehdi Kargahi[1], Benjamin Fabian[2]

[1]School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
[2]Information Systems, Hochschule für Telekommunikation Leipzig, Leipzig, Germany
✉ *E-mail: alidoostnia@ut.ac.ir*

Abstract: In recent years, the increase in cyber threats has raised many concerns about security and privacy in the digital world. However, new attack methods are often limited to a few core techniques. Here, in order to detect new threat patterns, the authors use an attack graph structure to model unprecedented network traffic. This graph for the unknown attack is matched to a pre-known threat database, which contains attack graphs related to each known threat. The main challenge is to associate unknown traffics to a family of known threats. For this, the authors utilise random walks and pattern theorem. The authors utilise the pattern theorem and apply it to a set of proposed algorithms for detecting new generations of malicious traffics. Under the assumption of having a proper threat database, the authors argue that for each unknown threat, which belongs to a family of threats, it is possible to find at least one matching pattern with high matching rate and sensitivity.

1 Introduction

In recent years, computer systems were susceptible and vulnerable to cyber attacks. A statistical analysis of vulnerability databases performed by Palo Alto Networks reveals that the number of core techniques used by attackers is limited to several core techniques [1]. Compared to the thousands of vulnerabilities recorded in the National Vulnerability Database (NVD), the number of unprecedented attack techniques is limited to a dozen [2]. According to NVD results, in 2016, ∼6000 types of threats were recorded. Reviewing similar results for recent years reveals that these threats are mostly categorised as new branches of classic threat families. These new threats are either implemented using a new technique or exploit a vulnerability through previously known attack methods in the system. It is important to detect the behaviour of new intrusion methods to come up with corresponding countermeasures. For this, it is necessary to review intrinsic vulnerability features and the related threats associated with each of them. In the area of network security, it is necessary to deal with new generations of threats and to frequently update protection techniques in security devices. One of the outstanding examples in the field of network security is the Heartbleed vulnerability that was caused by a bug in the OpenSSL library [2]. This weakness made it easy for intruders to eavesdrop on the encrypted data exchange. However, this bug existed in OpenSSL since 2012; it was not discovered until 2014.

The main question is whether we could detect threats using such vulnerability from behavioural analysis. Network-traffic analysis is shown to be an effective method for detecting unprecedented types of threats [3]. A challenge with detecting these threats is their unknown nature that makes it difficult to detect malicious behaviour without any prior knowledge.

In this research, we aim to determine to what extent we can predict the behaviour of a new class of threats using the known malicious behaviour of previously detected ones. If all of the threats were pre-known, established traffic engineering and analysis methods would be viable to detect malicious behaviour of an adversary [3]. In this situation, our threat database would be clairvoyant, and matching each newly sampled traffic to a record from the database would be simple. If the malicious traffic is identified, monitoring tools such as firewalls can block it, even though adversaries have certain techniques to hide malicious traffic from the external observer, which is done by making fundamental changes in network flow patterns [4].

However, we have no clairvoyant threat database at hand. When the security device faces a new traffic sample, it is even-handed to consider it as either trusted one or malicious traffic. The main goal of traffic analysis is to match network flows with pre-known traffic samples using a pessimistic approach. If the matching process fails to decide on the entry, we should not trust the incoming traffic, since trust in an unidentified traffic may lead to security defects; therefore, we must come up with a pragmatic solution for matching unknown types of network traffic. Addressing this challenge is the motivation of this research work.

Our proposed method is based on attack graphs. Attack graphs are a logical description elicited from data related to the threat. In fact, an attack graph represents a set of paths that the adversary can adopt to penetrate a host or network [5]. The main application of attack graphs can be seen in digital forensics where forensic examiners employ attack graphs to reconstruct a cyber-crime scene [6]. The attack graph is related to the target application, which means that an attack graph of the same threat may be different in various network topologies. Therefore, the first step towards proposing a solution is to standardise the required specifications of the environment.

Based on this suggested solution, at first, we generate attack graphs corresponding to each threat. If the sampled traffic belongs to neither trusted patterns nor malicious ones, we must construct a new attack graph based on that traffic. Therefore, we need a similarity measurement for evaluating the similarity of the traffic to the pre-known threats [7]. It is significant how we define the similarity measurement. In fact, it is introduced by reviewing the set of similar subgraphs [8]. In this research, we employ a customised type of random walk, which is applied to attack graphs with higher dimensions [9]. To the best of our knowledge, this is the first research work that uses attack graphs for online traffic analysis. Another novelty of this work is to apply strong randomness features of random walk to measuring the similarity of attack graphs. The main issue is how we apply the random walk to the structure of attack graphs. Considering the structure of attack graphs, we need to come up with a customised algorithm for matching them accurately. We represent implementation details and proposed algorithms for matching problems in attack graphs.

Concerning our key contributions, we present a number of achievements in this work:

- We put forward traffic-engineering methods to detect unprecedented network threats.
- We argue that the presented method is applicable to online detection facilities in order to decide about unknown threats in real time.
- We use logical attack graphs to reduce the size of computations in the presented method. This could help security facilities like intrusion detection systems (IDSs) to come up with quick solutions for online decisions.
- We apply the random walk to achieve high-precision detection rates. The results show that the sensitivity of the method is up to 98%. This assures high-level security against new generations of threats.
- We base on strong mathematical foundations such as the pattern theorem, which enhances the credibility of the proposed system.

The rest of the paper is structured as follows. In Section 2, preliminaries are reviewed, including prerequisites, definitions, formulas, applications, and previous work. In Section 3, related work is discussed. Section 4 reveals the main problem and the idea of the paper. In Section 5, the proposed algorithms and solution approach are presented. Section 6 is dedicated to the experimental result and comparison to previous outstanding research. Finally, the paper concludes in Section 7.

2 Preliminaries

We start with explaining how an attack graph is generated. As mentioned, an attack graph is a precise path describing how the attack is executed. In this research, the main purpose is to utilise the power of matching in graphs in order to detect threats. However, the generation of attack graphs is not the only challenge, but paying attention on how they are organised is also important. A logical representation of attack paths indicates the behaviour of the adversary [10]. In this research, the main purpose is to utilise the power of matching in order to detect threats in unknown traffics. Afterwards, we provide a review on traffic-engineering methods, random walk, and pattern theorem.

2.1 Attack graph

From a mathematical point of view, attack graphs are directed graphs, but under some condition, we can also represent them as undirected [6]. Each vertex of the graph shows an exploit. Each vertex has a pre-condition and post-condition, which represent prerequisites before and after an exploit happens. An edge connects two vertices if the source's post-condition satisfies part of the destination's pre-condition.

For designing an attack graph, many tool sets have been proposed. In this research, we employ MulVal, which is an open-source tool-set for generating attack graphs based on formal definitions 1–3 [11, 12]. This tool has been designed based on Prolog and automatically detects vulnerabilities existing in an enterprise network [13].

Definition 1: A formal attack model is a finite automaton in the form of $M = (S, \tau, s_0)$, where S is the set of states, $\tau \subseteq S \times S$ is the transition relation, and $s_0 \in S$ is the initial state. State space S represents a triple set of agents denoted by I in the form of $I = \{E, D, T\}$, in which E is the adversary, D represents the defender, and T indicates the system that is under attack. In this definition, each agent is indicated by $i \in I$ and has its own set of states denoted by S_i.

Definition 2: A finite execution of a specific attack model $M = (S, \tau, s_0)$ is a sequence of possible states in the form of $\alpha = s_0 s_1 \ldots s_n$, where $0 \leq i \leq n$, $(s_i, s_{i+1}) \in \tau$. An infinite execution of an attack model is also represented as $\beta = s_0 s_1 \ldots s_n \ldots$, where $i \geq 0$, $(s_i, s_{i+1}) \in \tau$.

Definition 3: Considering Definition 2 and an attack graph with model M, a series of security properties P is denoted as the set $L(M)/P$ in which L is the labelling function. It emphasises that the attack graph encompasses a set of failed executions of model M under security properties P.

By considering Definition 3, we face a state-space explosion. Consequently, the construction process of attack graphs involves a serious challenge. For dealing with the mentioned issue, we have to reduce the state-space. An attack-graph generator must employ approximation techniques to simplify the output graph while preserving its main features.

2.2 Traffic analysis

Traffic analysis is one of the basic approaches to detect network-level threats. Not only it is useful to determine new threats, but it also helps to detect certain behaviour of a specific application according to network-level transactions. To the best of our knowledge, no perfect approach with 100% correctness has so far been proposed for detecting new threats. They work best in a specific environment based on an accurate threat database, which is ideally filled by pre-known records.

Traffic analysis is useful for security enhancement in an enterprise network; however, it would endanger privacy and security of users' applications [4]. Some methods are able to detect encrypted network flows using traffic patterns. The hidden Markov model is one of the outstanding examples that is used to detect network-level threats [3]. In this context, three features of data existing in sent packets are considered for traffic analysis. These are the direction, time, and size of the packets. Depending on the application, however, it is possible to perform the entire analysis without some of them [3]. In applications such as Tor, which uses multi-layered encrypted packets, direction is not accessible [4]. To perform an accurate analysis, we need to access at least two of the mentioned features. For generating an attack graph, we employ these three features in the form of (d, t, s).

Another property in traffic analysis methods is to consider different security levels. A level states the destructive power of each attack and their side effects on the system's functionalities [14]. We have employed a set of levels as a coefficient in the threat database, which is a metric to show the importance of the same threat. For example, in matching unprecedented traffics using pre-known threats, it would be useful to adjust the weight assigned to each threat and to receive results that are certainly more accurate.

In our proposed method, MulVal gets the provided information from a set of network flows using the structure described in this section. The input for MulVal is the same network flows that are not recognised by the firewall/IDS. So, it is able to generate a graph that represents the penetration point of the flow. The access points to the local network are analysed and an attack graph is generated. The strong point of MulVal is the speed of graph generation and the reduction in the graph size. By applying a set of logical methods, it can manage to generate a lightweight and reduced attack graph that describes the penetration paths very well. In generating the final attack graphs, some conditions must be considered. Pre-conditions and post-conditions are two types of the conditions that refer to a set of penetration/access points in the graph. The conditions must be fulfilled before/after another penetration/access point. This depends on the structure of the network and the behaviour of the adversary.

2.3 Random walk

Consider a salesperson who is walking through a specific path on the x-axes. He starts the walk from an initial position while he is able to walk in either $-x$ or $+x$ directions. In each step, he selects one of his neighbours randomly with equal probability. When he walks in two directions, it means that the walker has only two alternatives in each step. He can move either forward or backward. According to random selections in each step, after walking a number of steps, we say that the set of steps constructs a random walk.

Definition 4: Suppose that $\boldsymbol{X}_k$ be a sequence of random independent vectors. Therefore, $\boldsymbol{X}_k$ can take values $\pm e_i$, $i = 1,2,\ldots,d$

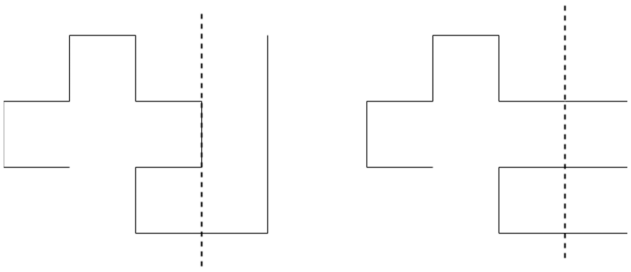

Fig. 1 *Comparing two different SARWs with similar patterns*

with probability $1/2d$, where d is the number of dimensions. The main question is to compute the position of the walker after m steps of random walk. Simply, we need to compute the output value of the below equation

$$R_m = \sum_{k=1}^{m} X_k \tag{1}$$

We call this set of random steps that construct a random path a simple random walk (SRW) [15]. The main application of SRW is observed in simulation of behaviour in random phenomena. The outstanding question of Polya divides random walks into two different categories, transitive and recursive [9]. A random walk is called recursive if we are sure that the walker will revisit the previously visited positions. According to Polya's theorem, a random walk with $d \leq 2$ is recursive. It states that we definitely face the recursion problem in two-dimensional environments [16]. Equation (2) states the same fact using Markov property

$$\mathbb{P}^x\{R_n = 0 \quad \text{for some } n \geq 1\} = 1 \tag{2}$$

According to Markovian properties, if $p \in P'_d$ under $d \leq 2$, so p would be recursive. On the other hand, if we have $p \in P^x_d$ (P in higher dimensions) and $d \geq 3$, therefore, p is transitive [16]. Equation (3) presents the same condition, which is called the escape probability of q

$$q = \mathbb{P}\{R_n \neq 0 \quad \text{for all } n \geq 1\} \tag{3}$$

We aim to employ higher orders of random walks to our research. Considering the recursion issue, each type of random walk may cause some intersections. To come up with a similarity measurement in attack graphs, we need the same property in a way to avoid revisiting the nodes through a random walk on vertices of an attack graph. To deal with this issue, we introduce self-avoiding random walk (SARW) [17]. Next, we explain how it works.

2.4 Self-avoiding random walk

From a mathematical point of view, the computations based on the paths related to an SARW are more complicated.

Definition 5: An SARW with length n in the space $\mathbb{Z}^d$ is a sequence ω_i defined as $\omega = (\omega_0, \omega_1, \ldots, \omega_n)$, where $\omega_0 = 0$, $\omega_i \in \mathbb{Z}^d$, and $\|\omega_j - \omega_{j-1}\| = 1$ for each $j = 1,\ldots,n$ under the condition $\omega_i \neq \omega_j$ where $i \neq j$.

C_n is the distance of the random walker after n steps in the SARW, and it is computed with respect to the limitations imposed by Definition 5. According to (4), C_n is limited by two values derived from d through n steps

$$d^n \leq C_n \leq 2d(2d-1)^n \tag{4}$$

However, the precise computation of C_n is complicated; we can estimate it via (5). This equation is derived from [18] where the square distance from the initial point is computed. We have imposed a further limitation in which the ω parameters are considered as a vector that belongs to the same lattice [19]. We can perceive the reason behind this limitation after applying pattern theorem

$$|\omega(n)|_n^2 = \frac{1}{C_n} \sum_{\omega:|\omega|=n} |\omega(n)|^2 \tag{5}$$

Discussion about SARW is extensive; however, we only talk about a series of selected topics related to this research scope. It is obvious that imposing new limitations on a random walk's behaviour is similar to making some changes in the probability distribution function. We are changing the random behaviour in order to make it closer to the reality of a system's behaviour. In graphs with large sizes like real attack graphs, we must deal with a greater number of selections. We cannot give up the selection process and assign similar probabilities to the alternatives. Respecting these limitations, we will be able to utilise the pattern theorem in practice.

2.5 Pattern theorem

Simply, a pattern is a small part of an SARW. According to Kesten's pattern theorem, if a given pattern is able to happen several times in an assuming SARW, that pattern must also happen aN times on the N steps of the same SARW, where $a > 0$ [19]. This introduces a new measurement for determining the probability of repeating a part of SARW in a part of other SARWs.

Statistical analysis of each pattern would be different according to the target application. When it comes to SARW, many interesting statistics exist that base on repeating of the same pattern in the beginning part of an assumed SARW. We refer to the ergodic feature as one of the most significant analytical properties [17]. It states that the system's behaviour remains constant in the space over time. As shown in Fig. 1, two random walks are illustrated, which have similar patterns in early steps. In the following, we give a precise definition of a pattern [19].

Definition 6: We say that a pattern in the form of $P = (p(0), \ldots, p(n))$, happens in the jth step of an SARW with specification of $\omega = (\omega(0), \ldots, \omega(N))$ if a vector $\boldsymbol{v}$ exists in the space Z^d where for each $k = 0,\ldots,n$; $\omega(j+k) = p(k) + \boldsymbol{v}$ and $\boldsymbol{v}$ is computed by $\omega(j) - p(0)$.

Assume that R_N be a representative for N step of an SARW with the specified path ω, where $\omega(0) = 0$. For each $k \geq 0$ and pattern P, we introduce $C_N[k, P]$, the number of steps in which P happens at most in k steps, and also $F_N[P]$, a subset of walks where P happens in the 0th step. We call P a proper front pattern if $F_N[P]$ is a non-empty set for each large N[19]. In addition, the same pattern can be a proper internal pattern if for each k, there exists an SARW in which P happens at least in kth different steps. On the other hand, this means that the same extracted pattern must be contained in kth steps to be considered as a proper internal pattern with parameter k.

In this case, P is called a proper internal pattern. Equivalently, for such a pattern, there exists a cube with specification $Q = \{x: 0 \leq x_k \leq b\}$ and an SARW with specification ϕ (e.g. number of steps and their directions) in which if P happens in some steps of ϕ, ϕ includes Q. This means that the final points of ϕ are the corners of Q. In addition, there exists an SARW with specification ω where P happens in more than three steps of ω. With this background, we start defining cube and pattern matching schemas according to Kesten's definitions [19].

Definition 7: A cube Q is a set in the form of (6) where $a_1,\ldots,a_d$ and $b > 0$ are in $\mathbb{Z}$

$$Q = \left\{x \in Z^d: a_i \leq x_i \leq a_i + b \quad \text{for all } i = 1, \ldots, d\right\} \tag{6}$$

Definition 8: Consider $\bar{Q}$ that, according to (7), is two units in each direction greater than Q. The outer boundary of Q is shown as ∂Q in (8) where it is needed to determine a pattern in further steps. It includes the set of points in $\bar{Q}$ that are not in Q

$$\bar{Q} = \left\{ x \in Z^d : a_i - 2 \le x_i \le a_i + b + 2,\ i = 1, \ldots, d \right\} \quad (7)$$

$$\partial Q = \bar{Q} / Q \quad (8)$$

It is proved that the whole random path generated by an SARW exists in $\bar{Q}$ [19]. A strong conclusion from the abovementioned relations is stated in (9). It is the basis for this research, which is called pattern-matching relation where $y(i)$ is a part of ∂Q. On the other hand, this fact indicates that expanding the current boundary with j still includes the same pattern $P(i)$

$$\forall i, j \in \{0, \ldots, k\} \text{ and } y(i) \in \partial Q:\ y(j + i) = P(i) \quad (9)$$

We utilise the patterns in order to work with attribute-based attack graphs. The attributes of an attack graph refer to the properties that are extracted from the graph and are able to transfer to the final patterns in the form of self-avoiding walks. So, the pattern theorem will be applicable when we are able to generate attribute-based graphs.

3 Related work

In previous sections, we discussed the theoretical and technical background needed for diving into the design space. In this section, we are going to have a quick review on outstanding previous research conducted in this area. We start with research work on attack graphs. The first and most important application of attack graph can be seen in digital forensics [6]. In such applications, examiners come up with an evident graph that consists of digital evidences of an attack or a cyber-crime. This graph is compared to the pre-generated attack graphs. Each vertex of the graph represents evidence and the path of attack is reconstructed using the corresponding path in attack graph. As new techniques used by intruders will become more complicated, the analysis of such huge graphs requires new solutions [13]. We use the attack graphs in our work with one fundamental difference. Instead of comparing the graphs directly, we generate patterns from the graphs and match these patterns together.

The generation of attack graphs is an important issue for researchers. The large number of paths and vertices in attack graphs makes it difficult to have an accurate analysis. In addition, the lack of an existing logical framework in generating such graphs exacerbates the situations for conducting a proper analysis [10]. Employing tools that take advantage of having a logical and formal framework, like MulVAL, could help to generate complex attack graphs in polynomial time [11]. Another issue is graphical representation of large attack graphs. In a very useful research conducted in MIT, the required framework for presentation of large attack graphs has been developed [12]. It is helpful for us to use such visual tools to represent the final patterns generated from the graphs, and compare the results visually.

Random walks have many applications in engineering sciences. In computer science, the main application of random walks is observed in modelling of complex networks [20]. One of the other applications of random walks are PCS networks [21]. In such networks, in order to update distance metrics dynamically, random walks are deployed. Another important application of random walk is traffic-engineering [4]. Not only it is used in order to detect cyber-threats, but it is also applied to network-based applications to improve the privacy of users through the Internet [14]. In our previous work that is highly related to the current research, we deployed random walks in order to directly analyse the network flows. However, in the current research work, we aim to use random walks for extracting the patterns from the attack graphs. Then we can match the patterns instead of comparing the graphs. It helps out to remove some limitations of classic approaches that do not have the ability to represent large network structures. In our previous work, we had the same limitations that decrease the scope of analysis.

One of the popular applications of random walk is similarity measurement between two given graphs [9]. Similarity measurement is a technique through which an observer can decide if two or more graphs are similar. The similarity is evaluated based on a set of pre-defined metrics. If we aim to match two graphs, the best idea is to extract some elements in graphs that are in common. The first element that comes to mind is subgraphs. Similarity measurement among subgraphs is the subject of a dedicated research direction [22]. Furthermore, we divide the main problem of similarity measurement of graphs to similarity measurement of subgraphs. Mainly, this type of research is based on distance computation in all maximal common subgraphs. The main concern about this research, comparing to our work, is the ability to analyse large network structures.

Similarity measurement is an important feature in the development of matching techniques. Doing that, random walk has special abilities related to similarity measurement between nodes that in a previous work, it is deployed in recommender systems [23]. In this approach, two random variables are defined on two different graph structures. Based on the maximum degree of the graph, the dimension and other features of random walk are determined. So, as a gaming model, two variables play an interactive game by walking on the graphs. The similarity level is evaluated based on the similarity of the walks on two graphs. Finally, one can decide if two graphs are similar to each other. The approach helps to improve the precision of decision-making by walking through the graph edges. Another application of this measurement can be seen in parsed texts, which is related to natural-language applications [24]. As an application for random walk in machine learning, we refer to natural-language processing. Through a random walk, the walker/agent can learn many aspects that are used for further evaluations. For example, if the agent has the experience of walking on a ring graph, he can learn to distinguish walking on this type of graphs in the near future. This can be considered as a common point between pattern theorem and random walk where we learn from patterns to determine the similar walks in unprecedented graphs. In our work, in each random walk, the walker collects a set of experiences and utilises them in its future walks to discover and learn new patterns

Fault-tolerant graph matching is a fundamental concept in the reliability of the results [8]. To this end, we must ensure that our matching technique is accurate in terms of similarity metrics. The final result of the matching technique should lead to reliable decisions. This feature is costly, and therefore, it requires a precise strategy in security-related applications. Applying each matching technique must ensure the reliability of the result by an acceptable level of accuracy. For example, if the accuracy of the result is evaluated 98%, we should expect 98 accurate matching results among 100 matching operations.

Detecting unprecedented threats is one of the main goals of this research. Considering this feature, we need to enhance accuracy and integrity of the detection process. This is not the first work in this area [3, 14, 25]; however, we propose a new solution to deal with this issue. Academic and business implementations for threat-detection systems are applied in two different ways. The first is online detection of threats that suffers many limitations [26, 27]. It is difficult to apply detection, recognition, and making decision in real time when the traffic is considered as unknown. The second approach is to conduct some parts of the detection process offline. These techniques have some applications in tracking and monitoring. They focus more on prevention instead of detection/recognition. One of the most significant products designed in Palo Alto Company is exploit prevention module (EPM) [1]. Such prevention techniques concentrate more on penetration techniques for exploiting the entire system. They try to secure a specific system using a set of security policies. This approach decreases the ability to detect threats in real time. Therefore, we focus on the first group of techniques that preserve the ability for detecting threats in real time.

4 Problem statement and solution idea

As discussed earlier, the main problem is detecting new generations of cyber-threats. In this section, we provide a detailed overview on the main problem and the solution idea.

The input of the problem is an unprecedented network flow, which is not identified by network security tools like firewall and IDS. A finite sequence of network packets is sampled in the form

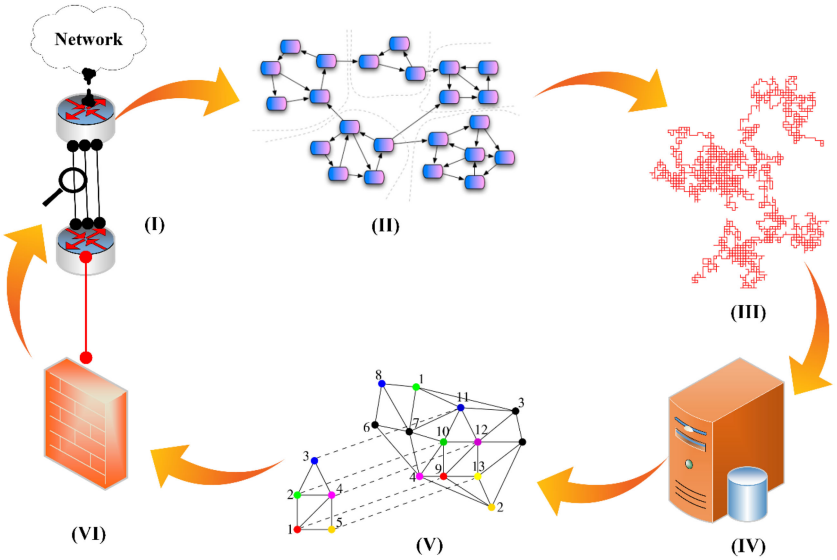

Fig. 2 *Overview of the solution approach; it shows the life cycle of the system using analysis of attack graphs*

of $(d,\ t,\ s)$, where d, t, and s are representatives for direction, time, and size of the packets, respectively. Further information, including encryption status, application-level features, employed protocols, requested ports, and accessed network paths, is attached to the sample. Afterwards, all input/output flows associated with the same path will be continuously analysed in order to model its behaviour in an attack graph.

For each unprecedented network flow, we consider a pessimistic scenario in which all unprecedented flows are assumed malicious until our analysis determines whether it belongs to a threat family or not. Therefore, we can come up with a corresponding attack graph. We assign the flow's attributes to vertices of the attack graph G that must be compared to other pre-known attack graphs existing in our threat database denoted by DB. The quantity of samples in threat database is not the main topic of our study. We aim to propose a systematic approach to cope with the matching problem of unprecedented attack graphs.

The matching process is applied on graphs using the generation of a random path from an SARW. Instead of matching subgraphs, we aim to match elicited patterns π from SARW with patterns existing in our database Π. From now on, pattern theorems on SARWs would help us to address the matching problem in attack graphs. Matching rate MR is the output of this step for each attack graph existing in our threat database. We assign each attack graph in the database a weight according to its destructive effects in the system. Determining this coefficient requires experimental results that are discussed in the next section. So, the problem is summarised to find out the matching rate between a set of pre-known patterns in our database Π and a set of unknown patterns extracted from the given network flow π. The former is computed based on the knowledge in the threat database, and the latter is determined by applying SARW to online traffic. Equation (10) describes the entire problem formally: its output gives the matching rate

$$\forall\, i \in \{1 \ldots n\}\,. \quad \mathrm{MR}_i = \mathrm{Match}_i(\Pi_{\mathrm{DB}},\ \gamma) \tag{10}$$

Here, n shows the number of pattern types that should be compared in the database. One can easily measure the similarity between the patterns according to the output of the matching rate.

The threat database stores an information related to pre-known vulnerabilities that support our pattern matching technique. This includes name and the family of the threat, behavioural coefficient, sample network flow f in the form of $(d_i,\ t_i,\ s_i) \in f$ for all $i \in \mathbb{N}$, generated attack graph G corresponding to f, and finally a set of extracted patterns P in the form of $(p_1,\ p_2, p_3, p_4) \in P$.

Based on Definition 3, the structure of $G(V, E)$ is just like other ordinary graphs with the difference that V is a representative for penetration points into the network and E denotes a set of penetration paths between each two nodes of the graph. Formally, an attack graph is employed to illustrate the behaviour of the threat/adversary. As discussed earlier, the patterns are actually a subset of a walk on the attack graph. So, to represent a pattern, it is necessary to store it by a special data structure. In our model, we store a pattern/walk by a multi-dimensional array. Depending on the graph density and its maximum degree, the dimension is determined. This fact is fully discussed in the next sections.

We give an overview of the solution idea in the form of a set of steps. As shown in Fig. 2, decisions about unknown traffics are made with at most six consequent steps. At first, the firewall tries to detect an incoming network flow, but cannot evaluate it properly. In this part (I), incoming traffic is investigated by monitoring tools, which sample the required data from the incoming flow. In part (II), an attack graph is generated, which helps to provide an accurate evaluation about an unprecedented flow.

Considering Fig. 2, in part (III), we generate a set of random walks from the graph. The main algorithm in this section is SARW, which is fully discussed in the next section. According to part (IV), we need to compare the random walk to the existing ones from our database. If we could not find any 100% match, it is proved that the flow is unprecedented, and, respectively, we need to come up with a pattern-matching technique in part (V). At the end of this life cycle, the final decision is sent to the firewall (or any other security tool of the network).

5 Steps of the solution approach

In this section, we are going to review the implementation process and the proposed algorithms. We can improve the precision of the whole process using proper parameter selection in the network environment. Next, we look at the tools and implementation process.

5.1 Solution approach

To implement the proposed system, we need to integrate a set of tools and modules. We have indicated them in Fig. 3. The implementation step is divided into two independent parts: the first is associated with the attack graph, which is generated using the MulVAL tool set. After generating an attack graph, a set of dedicated modules cooperate in order to final decision about that flow is made.

At first, the *SARW* module converts the graph to the random walk. At the same time, the required patterns are generated using the *PatG* module. As a result, we have a set of generated patterns that must be matched to random walks existing in the database. This work is performed using the module *PatM*. At the end, the final decision is made whether we consider the unprecedented flow

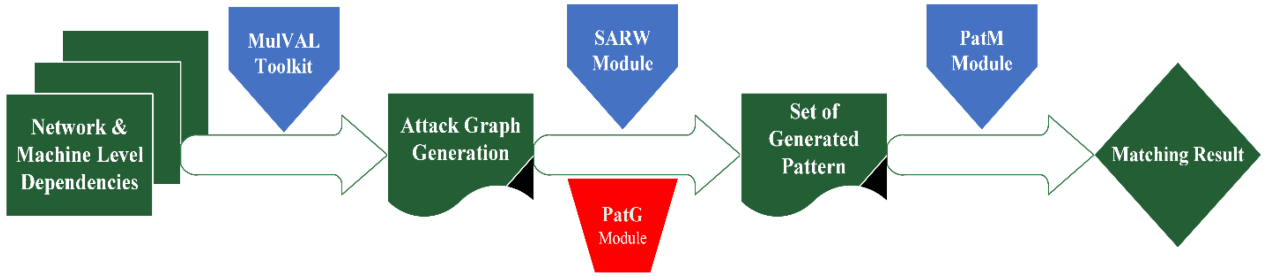

Fig. 3 *Review on pattern generation and matching modules in our proposed system*

INPUT: graph G[][], size n, Δ(G).
OUTPUT: list of random steps as *walk*[][].
1: cur=node(G[0][0]), dim=⌈(Δ(G))/2⌉, i=0
2: *dir*=2*dim
3: initial *pv* as set of previously visited nodes
4: initial *List* as set of next available nodes
5: **while** cur!=Null **do**
6: add set of next available nodes to *List*
7: remove nodes belong to previously visited edges from List
8: next=RandomSelection(List, dir)
9: walk[i][0]=next
10: walk[i][1]=dir(next)
11: add edge(cur, next) to *pv*
12: cur=next
13: i++
14: **end while**
15: **if** *walk* has at least one member **then**
16: return *walk*
17: **else**
18: return 0
19: **end if**

Fig. 4 *Algorithm 1: SARW algorithm for attack graph*

as malicious or not. In the next sections, we discuss the special abilities of each module.

The tool chain that we have presented helps the network security devices to decide about unknown network flows. Pattern matching, which is the basis of this research, is equivalent to matching subgraphs in two given attack graphs. It comes from Definition 6 that a pattern is a subset of an SARW. On the other hand, each set of random steps belongs to a specific attack graph, which reflects a unique behaviour.

5.2 Generation SARW from attack graph

The primary mechanism for utilisation of pattern matching is a random walk. Therefore, it is important to pay special attention to details of the SARW module. Self-avoiding walk has been applied to many research; however, we have customised it according to attack graph model. It depends on the graph size to determine in what dimension the random walk must be implemented. We have to set a threshold for the number of directions in the random walk. It is determined according to the maximum degree of the graph's vertices. In each dimension, we have two selections. This metric is selected by maximum output degree (Δ) of the graph. For example, if $\Delta = 6$, we must set dimensions to three in order to reach six directions in the random walk. We use (11) to calculate the number of dimensions

$$d = \left\lceil \frac{\Delta(G)}{2} \right\rceil \tag{11}$$

Our investigations show that attack graphs generated by the MulVAL tool contain no considerable bottleneck. It is due to the logical operations performed in MulVAL [13]. The maximum number of dimensions used in this work does not exceed five. It has been proved that the pattern theorem is applicable for higher dimensions [19]. Imposing such constraints to the system does not affect the strength of our proposed approach.

As the number of dimensions increases, the process of generating SARW would be simpler. This is due to the reduction in previously visited points in higher dimensions. The more the number of dimensions increases, the more the alternatives rise. Consequently, the number of intersections occurring in a random walk will be decreased [14]. As a result, using higher dimensions will decrease the cost of the entire system.

Algorithm 1 (see Fig. 4) indicates how a random walk is constructed on a given graph. The input of this algorithm is a set of information associated with the attack graph. We need to get adjacency matrix, graph size, and maximum degree of the graph. According to the properties of that graph, it generates the random walk in the form of a two-dimensional array. Lines 1–4 show the initialisation process and auxiliary variables. Lines 5–14 indicate the main functionalities of the algorithm. The random walk starts from an arbitrary vertex of the attack graph. It is the first vertex in the adjacency matrix. At first, a set of accessible nodes is determined as alternatives for the next step. As we have selected self-avoiding walk, it cannot intersect the previously visited nodes.

To this end, among the neighbours, a set of vertices that guarantee this rule is selected and put in *List*. In the next round, it selects from the same list at random, and according to the set of paths *dir*, the next random step would be selected. In each step of the algorithm, the selected node and the direction of the walk are stored. Visited vertices are added to *pv* in order to be imposed as further limitation in the next rounds. The algorithm is executed recursively until we cannot select any vertex according to set of visited nodes. It is not necessary to cover all the edges of the graph because we need subsets of the random walk. According to the pattern theorem, apart from behaviour of a complete graph, we can match a specific part of it to the other graphs.

5.3 Pattern generation from random walk

We need to extract proper patterns from the output of Algorithm 1. We have previously mentioned the module *patG*, which is indicated in Fig. 3. This is the most sensitive part of the research. Generating precise patterns from random walks leads to accurate decisions in the matching process. Algorithm 2 (see Fig. 5) illustrates the criteria required for implementation of the *patG* module.

As shown in Algorithm 2, we search for four different patterns that lead us towards the generation of unique patterns. *P* is the set of the four types of patterns that we have proposed to facilitate the matching process and it is the output of the algorithm. They are proposed based on the uniqueness of random walk and the behaviour of the patterns. Actually, we have discovered these types of patterns by empirical analysis of the threat database. In our database, a set of patterns is generated for each threat family. By analysing these patterns, we found out that even if the number of patterns increases we are still able to categorise them into the same types of patterns. Note that if we plan to generate patterns for more than four dimensions, the analysis process should be applied again, and some new types of patterns may be discovered.

According to our analysis on the threat database, the four types of patterns are categorised as follows, in which each index indicates the pattern's priority in the given SARW.

- Paths that have no two consequent steps in the same *dir* (*P*1).
- Paths where all steps happen in the same *dir* (*P*2).
- Paths that form unclosed squares and rectangles; this means that the beginning and the end of paths happen in the same *dim* with different *dir*, while the steps between them have the same *dir* (*P*3).
- Paths that construct a part of a cube (*P*4).

After generation of a random path, it is possible to select a variety of subpaths as pattern. The four abovementioned cases are selected according to the pattern theorem and the behaviour of the attack graphs. We have tried to select types of patterns that reflect more contrasts in the matching process. For example, *P*1 has the most contrast level and shows tendencies to change the behaviour in attack graphs performed by the adversary. In addition, patterns *P*3 and *P*4 illustrate patterns in two and three dimensions. To match similar behaviours in two or three dimension, we employ geometric matching techniques.

5.4 Pattern matching process

In this section, we are going to match generated patterns to the ones existing in our threat database. To do so, we need to another algorithm, which is illustrated in Fig. 3 as *patM* module. Note that the system covers four types of predefined patterns where each type is only comparable to the same type. The output for matching is shown with a number between zero and one. The output indicates to what extent patterns have similar behaviours. An interesting feature of this approach is that it is independent of the initial point. This means that it is not important where the patterns are selected from the SARW. The only feature that we are interested in is similarity in behaviour.

Algorithm 3 (see Fig. 6) shows the matching of subpatterns associated with attack graphs. To measure the similarity of two attack graphs, we need to compare the similarity between two sets of patterns. We say that two sets of patterns, P and Q, are matched if and only if there exists two patterns $P_N \in P$ and $Q_N \in Q$, where Q_N and P_N are matched together. Note that some graphs may not include a type of patterns. This is due to their behaviour and the topology of attack graph. For example, P_4 happens only in attack graphs with complex structures. In order to observe such patterns, the walk must be generated in at least three dimensions. However, some patterns like P_3 are susceptible to occur in all walks.

Illustrated in Algorithm 3, sets of generated patterns are given to the algorithm as input. In addition, a set of selected patterns is picked from the threat database for which we should test whether they match or not. Matching processes for each type of patterns are performed separately. According to the algorithm, matching more than one pattern is not important to us. On the other hand, matching only one pattern suffices for making any decision. In the context, functions entitled *Matching* are responsible for deciding about the matching process. If we do not reach 100% matching rate, the results are given to another algorithm to estimate the similarity level using a set of thresholds *M*. In this situation, we may force to recalculate the random walk. Algorithm 3 generates the final decision if it reaches one matching for each type of pattern.

As mentioned earlier, the main goal of this project is to detect new generations of attacks. For example, consider that the input attack graph is a representative for a DoS attack. We expect to receive positive result when comparing it to the attack graphs in our database that belong to the family of DoS attacks. The output indicates whether the current traffic has a similar behaviour comparing to the other family members. If the positive result is confirmed, we store it as a new generation of the same family.

According to Algorithm 3, if the matching result is calculated as 100%, we have already detected the unknown traffic. Consider now that the matching rate is lower than 100%. In this situation, we must store it in the database with a probability for further utilisation. Assume that a matching has resulted in vector $\boldsymbol{P} = \{0.6, 0.8, 0.92, 0.73\}$. According to Algorithm 3, it needs further analysis because the rates in the vector are higher than the given threshold (0.5). We have ten different results from this family in the database in the form of P_1, P_2, …, P_{10}. According to (12), we calculate the average amounts. If the result is higher than a given threshold γ, we store it in the database with probability ϕ

$$\forall \boldsymbol{P}_i, \quad 1 \le i \le n \cdot p_{i1}, p_{i2}, p_{i3}, p_{i4} > \gamma \;\Rightarrow\; \phi = \left(\frac{\sum_{i=1}^{n} \boldsymbol{P}_i}{n} \right) \quad (12)$$

The probabilistic coefficient ϕ is calculated by (11) and can be updated in the future. It is not possible to assign it with certainty to a family of threats. However, we have shown that with probability ϕ, its behaviour is similar to the assigned family. We call ϕ the *behavioural coefficient*. Previously known threats have the same coefficient in the database that is set to one. In further comparisons, the newly added records to the database will also participate, but their final effect is multiplied by their behavioural coefficient.

INPUT: walk[][], number of steps m, dim.
OUTPUT: set of generated patterns as P={p1, p2, p3, p4}.
1: current direction as cur_dir
2: cur_dir=walk[0][1]
3: traverse walk[][1] for sub-directions while in each step walk[i][1]!=cur_dir
4: add them to p1
5: traverse walk[][1] for sub-directions while in each step walk[i][1]==cur_dir
6: add them to p2
7: traverse walk[][1] for sub-directions which give rectangular patterns
8: add them to p3
9: traverse walk[][1] for sub-directions which give rectangular cube patterns
10: add them to p4
11: **if** *P* has at least one non-empty member **then**
12: return *P*
13: **else**
14: return 0
15: **end if**

Fig. 5 *Algorithm 2: Pattern generation algorithm (patG module)*

INPUT: set of generated patterns as P={p1, p2, p3, p4}, DB pattern as DP, dim.
OUTPUT: Matching result as M[4]
1: **for** all patterns in P and DP **do**
2: M[0]=Match1(p1, dp1)
3: M[1]=Match2(p2, dp2)
4: M[2]=Match3(p3, dp3)
5: **if** *dim.P>2 & dim.DP>2* **then**
6: M[3]=Match4(p4, dp4)
7: **else**
8: **end for**
9: **if** *M has the value 1* **then**
10: return *matched*
11: **else if** all members have a matching value greater than 0.5 **then**
12: return M
13: **else**
14: return 0
15: **end if**

Fig. 6 *Algorithm 3: Pattern-matching algorithm (patM module)*

5.5 Details of the algorithms

At the end of this section, we are going to review all the algorithms via a simple scenario. Assume that a new incoming network flow is not detected by the firewall. However, it is blocked (e.g. by IPS) by applying network policies, and the analysis process is started by sampling the network traffic. So, the traffic is sent to the MulVal analysis tool plus some information on the network structure.

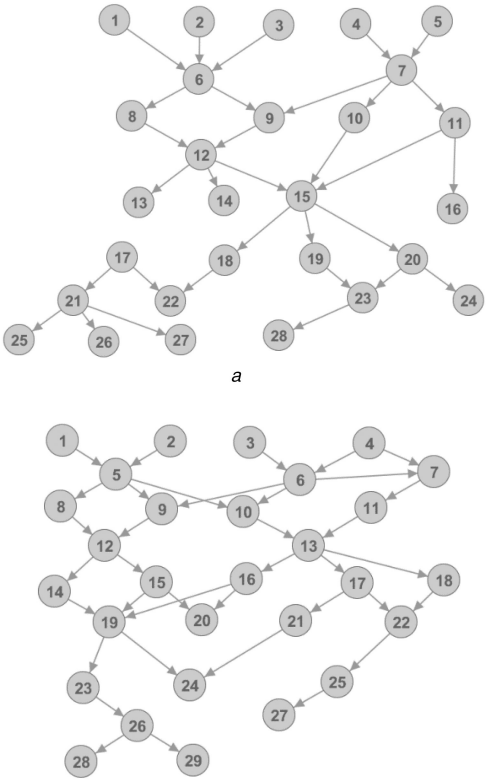

Fig. 7 *Two sample attack graphs for applying the pattern-matching process using the proposed system*
(a) AG2, ***(b)*** AG3

MulVal generates the attack graph corresponding to the same network flow. At the moment, Algorithm 1 is applied to the graph and the random walk is generated. The output of Algorithm 1 is sent to Algorithm 2 in order to generate behavioural patterns. Based on four pre-defined types of patterns, we expect to receive a set of patterns for each type separately. Finally, these patterns are sent to Algorithm 3, and the pattern-matching process is conducted according to the threat database. Based on the matching result, the security mechanism can decide if the flow is trusted or malicious. The final decision depends on the probability ϕ that is calculated according to (12).

6 Experimental results and discussion

In this section, we discuss the details of our implementations and results. We start by a real example and extend the same approach to other experiments.

6.1 Implementation and example

Here, we are going to present a sample of our method and explain it step by step. As shown in Fig. 7, two attack graphs are generated by MulVAL. Note that attack graphs are directed graphs and may contain several *sinks* and *sources*. Apart from the real meaning of each node, we have numbered each vertex sequentially, reflecting the dependencies of network structure on local policies. Since we are interested in modelling the behaviour of the adversary using random walks, we do not need any further information about network architecture. We have generated ten attack graphs labelled by AG1–AG10.

After the extraction of graphs, it is time to determine the data structure. Since we have to execute random walks on attack graphs, we select the adjacency matrix as data structure. While the graphs are directed, the matrix would not be symmetric. Random walks are applied independently from the number of vertices and other parameters of the graph. Therefore, matching graphs with different sizes does not negatively affect our method.

Each pair indicates coordinates in two dimensions. In calculations related to AG2, we faced an exception. Exceptions are

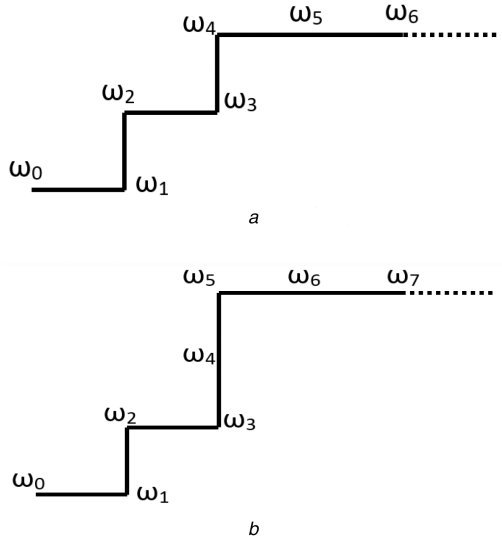

Fig. 8 *Random walk in two dimensions according to generated paths*
(a) SARW for AG2, ***(b)*** SARW for AG3

thrown when self-avoiding rules associated with SARW are violated. In this situation, we must roll back and modify the conflicts. The exceptions do not belong to the calculated path and must be removed at the end. In order to reduce the number of exceptions, we must use higher dimensions of random walk.

In the next step, we generate patterns from random walks. According to Algorithm 2, each type of pattern is detected. The result of this step is sent to Algorithm 3 in order to compute the matching rate. We are seeking for the greatest random subpath that contains four specified subpatterns in the patterns vector. Based on Algorithm 2, vector $\boldsymbol{P}$ is computed for each random walk (illustrated in Fig. 8) as follows:

$$\text{AG2} \Rightarrow \text{PAG2} = \{p_1 \rightarrow \omega_0 - \omega_5,\ p_2 \rightarrow \text{Null},\ p_3 \rightarrow \text{Null},\ p_4 \rightarrow \text{Null}\}$$

$$\text{AG3} \Rightarrow \text{PAG3} = \{p_1 \rightarrow \omega_0 - \omega_4,\ p_2 \rightarrow \text{Null},\ p_3 \rightarrow \text{Null},\ p_4 \rightarrow \text{Null}\}$$

According to the results, we can guess that the two random walks are matched. Since the vectors have some similarities in common, they would be comparable. Then it is time to calculate matching rates. In this example, we assume that AG3 belongs to the threat database, and we should decide about AG2. The length of the paths for AG2 is greater than for AG3. Therefore, AG2 dominates AG3 and we receive 100% matching rate.

We must pay special attention to each subpattern p_1 to p_4. For example, p_1 has intermittent structure. Consequently, if each two random walks are computed in the same space, it is not necessary to compare the complete set of steps. In such matching, we receive 100% matching rate if the given path has a greater number of steps. On the other hand, p_2 illustrates sets of steps in a unique direction. Therefore, a sufficient condition for matching is that two given subpatterns have equivalent length and move in the same direction. However, p_3 and p_4 require a set of more complex computations.

6.2 Experimental results

In this section, the results regarding to implementation space and conditions of experiments are reviewed. Our method is experimentally validated via a series of attacks belonging to the DDoS family. We have sampled a set of ten types of attack graphs generated from the same attack family. In fact, our simulator generates the network flows corresponding to each DDoS attack. All parts of our experiments are executed on the same network structure, and the results are calculated by simulation technique. The simulation is performed on a dual core computer with 2.53 GHz speed, 4 GB memory, and 100 Mb/s network controller. In addition, we have used a local database system for storage of threat

Table 1 Basic specifications and the output patterns of the experiment for each attack graph

Graph id	n	Δ_{out} (G)	Source	p_1	p_2	p_3	p_4
AG1	25	5	3	3	2	4	0
AG2	28	4	5	5	2	3	0
AG3	29	3	4	5	3	0	0
AG4	24	3	3	5	2	3	0
AG5	23	4	2	3	5	3	0
AG6	21	4	3	2	3	4	0
AG7	27	4	3	3	4	3	0
AG8	20	3	2	3	4	0	0
AG9	26	3	4	3	5	3	0
AG10	22	3	3	3	3	0	0

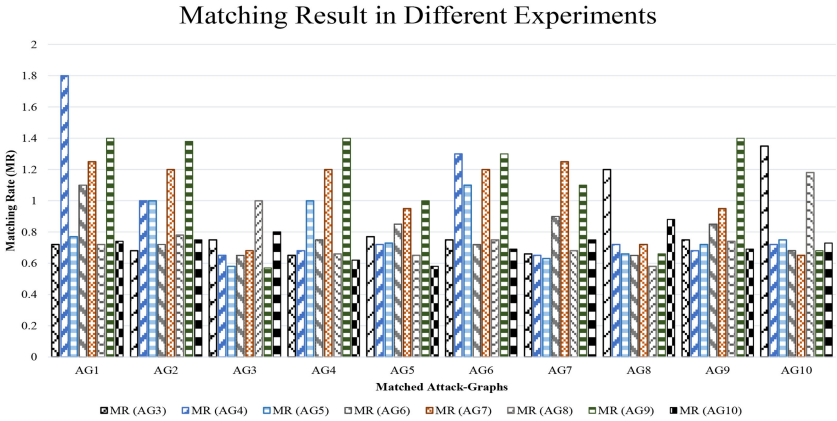

Fig. 9 *Set of matching results for different attack graphs using the proposed system*

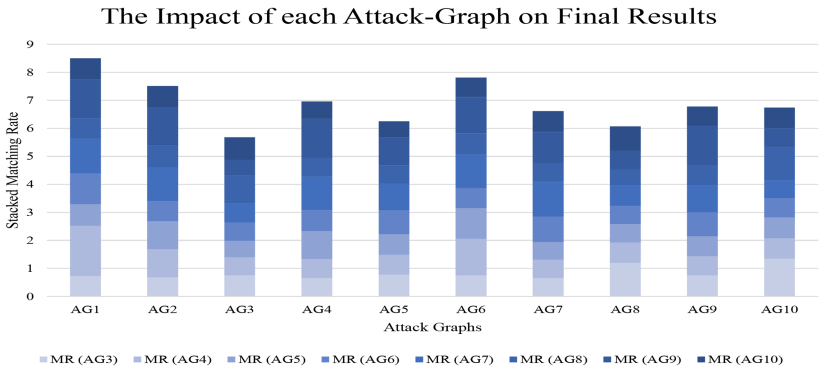

Fig. 10 *Stacked matching rate for each attack graph, showing their impact on the matching process*

patterns/features, and all programs are written in the JAVA programming language.

We execute the random walk on the graphs and build proper outputs according to a set of points. Afterwards, related patterns are generated and the patterns are tested whether they match. In each step, we assume one member in the set of attack graphs as unknown and compare it to the other nine patterns existing in threat database.

Table 1 indicates sets of specifications elicited from the output of the patterns after the execution of algorithms discussed in previous sections. Since we have used MulVAL's logical attack graphs, the size of graphs has been drastically reduced, and consequently, random walks are generated in two dimensions. Therefore, it is meaningful to miss p_4 that belongs to three or higher dimensions.

In this step, we are going to investigate matching results. We are interested in measuring the matching rates between extracted patterns. Fig. 9 indicates the results of a set of experiments performed according to matching algorithms. Based on the results, we achieved at least one case with 100% matching rate in each experiment. Each column in the figure shows how much the same attack graph is matched to the given unknown traffics, indicated by MR.

Achieving success in experiments depends on both the proposed algorithms and completeness of the threat database. For example, in the matching process associated with AG10, the main challenge is that p_3 is not generated. Therefore, it is comparable only to those that contain no p_3 pattern. Since we found two similar configurations among other graphs, we could match it. According to differences in the patterns' distributions and the structures of graphs, some matching rates are calculated above one. This would be normalised to one in implementation steps. Also, the impact of each attack graph in the matching process is shown in Fig. 10. It is a stacked measurement representing the power of each pattern in the results.

In the second experiment, we have compared attack graphs shown in Table 1 (AG1–10) to another family of threats belonging to a set of brute-force attack graphs (BR1–10). In this experiment, each attack graph AG_n is separately compared to all BR_n attack graphs $(0 \leq n \leq 10)$. According to given criteria, an essential condition for matching two attack graphs is to receive at least 80% matching rate. As shown in Fig. 11, in all executed experiments,

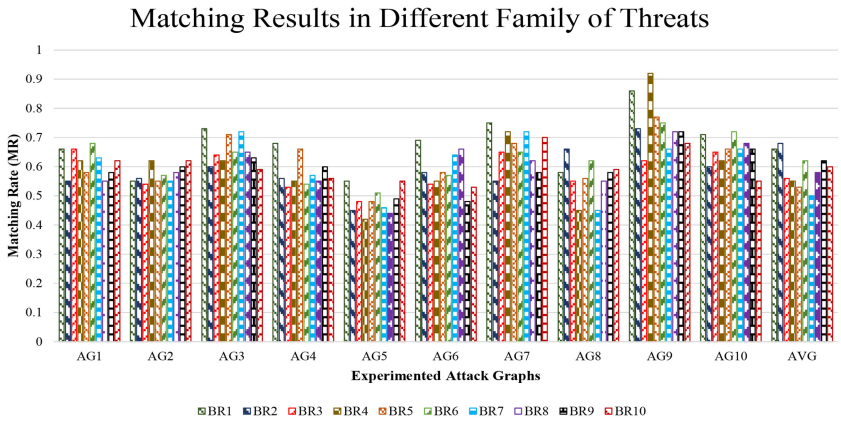

Fig. 11 *Set of matching results for different families of attack graph using the proposed system*

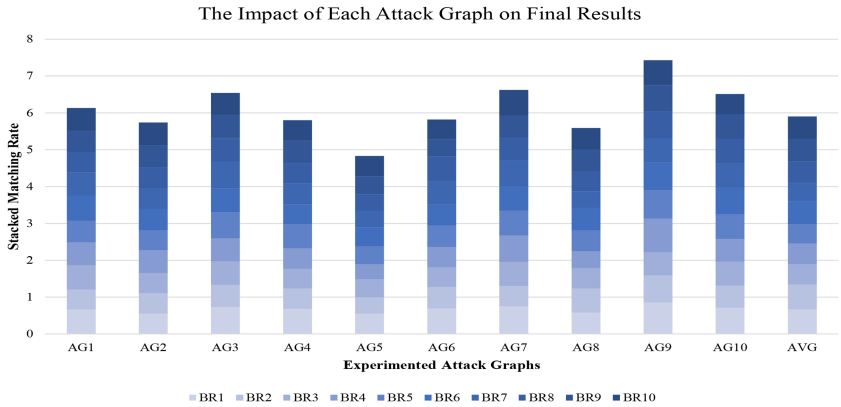

Fig. 12 *Stacked matching rate for each attack graph in the second experiment*

Table 2 Comparing approaches in detecting threats

Techniques	Average True detection rate, %	Detecting unknown threats	Complexity	Sensitivity to attack families	Online/offline
hidden Markov model [3]	80	no	e^n	neutral	online
EPM [1]	90	yes	$O(n)$	exploit	offline
co-clustering [25]	93	no	e^n	neutral	online
RAPS [14]	95	yes	$O(n^4)$	neutral	online/offline
proposed system	100	yes	$O(n^4)$	network attacks	online

we did not receive any 100% matching result. However, we obtain two cases >80% matching result out of hundred tests.

As we have prior knowledge about the existing differences between these two generations of threats, we can compute the false-negative (FN) matching rate as 2%. This means that on average, we may expect receiving two wrong matching results out of each hundred experiments where we compare two different generations of threats. According to the first experiment, the result for FN matching-rate was zero. We can confirm that, in this case, true-negative and true-positive (TP) rates are more significant than FN and false-positive rates because our task is to primarily detect malicious traffics.

In Fig. 12, the stacked impact of matching rates for the second experiment is indicated. Comparing to the first experiment, the second has a higher number of matches. In the second experiment, we have analysed 100 possibilities of potential matching, while in the first experiment, the number of matching tests was 90. Therefore, the overall stacked matching rate of the second experiment is higher than the first one.

6.3 Discussion

The mentioned results have been achieved on a specific sample of real attacks. Repeating such experiments in large scale requires access to larger data sets. We also need to come up with new tool chains for automating attack-graph generation. The strength of this approach is provided by the power of the pattern theorem. For each attack graph on which we are able to execute a random walk, it is possible to apply the proposed approach. For new types of attack graphs that may contain additional features, the proposed algorithms can be customised.

In Table 2, we compare our proposed method to the other research studies in this field. The comparison is based on average true detection rate, the ability to detect unknown threats, the complexity of implementation, sensitivity to a special attack family, and online/offline approach. In comparison to the previous works, our method reaches high precision in detecting unprecedented threats. However, it has costs of higher complexity of the algorithm and a limitation to network-level threats. Here, in terms of complexity, n refers to the number of nodes in the given attack graph.

We should emphasise that the complexity of our method is the total complexity reflected by Algorithms 1–3. We did not discuss the FN rate of the proposed system. This is due to the completeness assumption about the threat database. We assume that the threat database covers all the threats related to the network environment. If the database would be complete, the number of patterns related to the same family of threats is considerable. Therefore, when the FN rate increases, it does not affect the overall decision because we need to match one pattern at a time. This means that we do not care if the FN rate increases; it suffices to achieve at least one accurate pattern matching. To show the strength of our method, we can

compute the sensitivity of the system based on the TP and FN rates. It is computed according to (13), where TP and FN are computed as 100 and 2%, respectively. To this end, the sensitivity of the proposed method would be >0.98%. Note that the average true detection rate that is mentioned in Table 2 actually is the TP rate among the results. The high level of sensitivity shows the validity and strength of our proposed method. In addition, looking at true detection rate indicates the precision and correctness of the method

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (13)$$

7 Conclusion and future work

In this research, we focused on detecting new generations of threats using a pattern-matching approach. The basis for this matching is the utilisation of attack graphs. An attack graph indicates the set of possible paths accessible to the adversary. According to the proposed algorithms, we introduced a new approach for generating random walks on the attack graphs. Based on the pattern theorem, we elicited similar patterns through the graphs. In this research, ten types of attack graphs belonging to the same family of threats were generated. In each of ten separated experiments, we observed at least one case with 100% matching rate. The approach is generalised for other types of threats, and we can consider it as a basic approach to address the detecting of new generations of threats.

The proposed approach is not appropriate for very large graphs and only tested for logical attack graphs. One of the future directions for this research is to extend it to large-scale attack graphs. For instance, DBIR is a good case to experiment with [28]. In addition, working on the other metrics related to the analysis and design process is crucial. This part of research requires creative solutions in order to improve the matching results.

8 References

[1] Goodwin, S.: 'The harsh reality of cyber protection', in '*ISACA online webinar, Palo Alto networks*' (Enterprise Network Security Company, USA, 2015)

[2] National Vulnerability Database: 'CVSS severity distribution over time'. Annual report of NIST, September 2016

[3] Wright, C. V., Monrose, F., Masson, G. M.: 'On inferring application protocol behaviors in encrypted network traffic', *J. Mach. Learn. Res.*, 2006, **7**, pp. 2745–2769

[4] Nia, M. A., Atani, R. E., Ruiz-Martinez, A.: 'Privacy enhancement in anonymous network channels using multimodality injection', in '*Security and communication networks*', vol. **8** (Wiley Online Library, USA, 2015), Issue 16, pp. 2917–2932

[5] Ingols, K., Lippmann, R., Piwowarski, K.: 'Practical attack graph generation for network defense'. 22nd Annual Computer Security Applications Conf. (ACSAC'06), Miami Beach, FL, 2006, pp. 121–130

[6] Liu, C., Singhal, A., Wijesekera, D.: 'Using attack graphs in forensic examinations'. Int. Conf. on Availability, Reliability and Security (ARES), Prague, 2012, pp. 596–603

[7] Jeh, G., Widom, J.: 'Simrank: a measure of structural-context similarity'. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD '02), New York, USA, 2002, pp. 538–543

[8] Danai Koutra, A. R., Parikh, A., Xiang, J.: 'Algorithms for graph similarity and subgraph matching'. Ecological Inference Conf., Harvard University, 2011

[9] Allen, B., Nowak, M.: 'Games on graphs', *EMS Surv. Math. Sci.*, 2014, **1**, (2), pp. 113–151

[10] Ou, X., Boyer, W. F., McQueen, M. A.: 'A scalable approach to attack graph generation'. The 13th ACM Conf. on Computer and Communications Security, 2006, pp. 336–345

[11] Fouss, F., Pirotte, A., Renders, J. M., *et al.*: 'Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation', *IEEE Trans. Knowl. Data Eng.*, 2007, **19**, (3), pp. 355–369

[12] Chu, M., Ingols, K., Lippmann, R., *et al.*: 'Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR'. The Seventh Int. Symp. on Visualization for Cyber Security (VizSec '10), New York, USA, 2010, pp. 22–33

[13] Liu, C.: 'A probabilistic logic programming based model for network forensics'. PhD thesis, Graduate Faculty of George Mason University, 2015

[14] Nia, M. A., Ebrahimi Atani, R., Fabian, B., *et al.*: 'On detecting unidentified network traffic using pattern-based random walk', in '*Security & communication network*', vol. **9** (John Wiley & Sons, USA, 2016), Issue 16, pp. 3509–3526

[15] Lovász, L.: 'Random walks on graphs: a survey', *J. Math. Stud.*, 1993, **2**, pp. 1–46

[16] Kozdron, M.: 'An introduction to random walks from Polya to self-avoidance'. Technical report, Duke University, December 1998

[17] Slade, G., Blath, J., Imkeller, P., *et al.*: 'The self-avoiding walk: a brief survey'. Surveys in Stochastic Processes, The Thirty-third SPA Conf., Berlin, 2009, pp. 181–199

[18] Bauerschmidt, R., Duminil-Copin, H., Goodman, J., *et al.*: 'Lectures on self-avoiding walks', *Clay Math. Proc.*, 2012, **15**, pp. 395–467

[19] Madras, N., Slade, G.: '*The self-avoiding walk*' (Birkhäuser Basel-Springer, Switzerland, 2013, 1st edn.), pp. 229–255

[20] Noh, J. D., Rieger, H.: 'Random walks on complex networks', *Phys. Rev. Lett.*, 2004, **92**, pp. 1–4

[21] Akyildiz, I. F., Lin, Y. B., Lai, W. R., *et al.*: 'A new random walk model for PCS networks', *IEEE J. Sel. Areas Commun.*, 2000, **18**, (7), pp. 1254–1260

[22] Bunke, H., Shearer, K.: 'A graph distance metric based on the maximal common subgraph', *Pattern Recognit. Lett.*, 1998, **19**, (3–4), pp. 255–259

[23] Fouss, F., Pirotte, A., Renders, J. M., *et al.*: 'Random-walk computation of similarities between nodes of a graph, with application to collaborative recommendation', *IEEE Trans. Knowl. Data Eng.*, 2006, **19**, (3), pp. 355–369

[24] Minkov, E., Cohen, W. W.: 'Learning graph walk based similarity measures for parsed text'. The Conf. on Empirical Methods in Natural Language Processing (EMNLP '08), Stroudsburg, PA, USA, 2008, pp. 907–916

[25] Ahmed, M., Mahmood, A. N.: 'Network traffic pattern analysis using improved information theoretic co-clustering based collective anomaly detection'. Int. Conf. on Security and Privacy in Communication Networks, 2015, pp. 204–219

[26] Nia, M. A., Ruiz-Martinez, A.: 'Systematic literature review on the state of the art and future research work in anonymous communications systems', in '*Computers & electrical engineering*' (Elsevier, Netherlands, 2017), pp. 497–520

[27] Sommer, R., Vern, P.: 'Outside the closed world: on using machine learning for network intrusion detection'. IEEE Symp. on Security and Privacy (SP '10), Washington, DC, USA, 2010, pp. 305–316

[28] 'Mitigations aren't effective after the first six (a DBIR attack graph analysis)'. Available at https://securityblog.verizonenterprise.com/?p=7077, accessed August 2017