Telecommun Syst (2018) 68:277–294
https://doi.org/10.1007/s11235-017-0395-z

An auto-learning approach for network intrusion detection

Ammar Boulaiche[1,2] · Kamel Adi[2]

Published online: 27 September 2017

Abstract In this paper, we propose a novel intrusion detection technique with a fully automatic attack signatures generation capability. The proposed approach exploits a honeypot traffic data analysis to build an attack scenarios database, used to detect potential intrusions. Furthermore, for an effective and efficient intrusion detection mechanism, we introduce several new or adapted algorithms for signature generation, signature comparison, etc. Finally, we use DARPA'99 and UNSW-NB15 traffic to evaluate the proposed approach. The results indicate that the generated attack signatures are of high quality with low rates of false negatives and false positives.

Keywords Intrusion detection · Honeypots · Fuzzy hashing · DARPA'99 dataset · UNSW-NB15 dataset

1 Introduction

During the last decade, the hacker's community has changed substantially, due to the development of the Internet. Indeed, it is now, very easy for anyone to get access to very sophisticated security attack techniques without the need to have a wide or specialized knowledge. Consequently, critical data that are hosted in multitude of computer systems have never been as vulnerable to security attacks and remote intrusions. In order to cope with these threats, many security solutions have been proposed, such as anti-virus software, firewalls, access control systems, intrusion detection systems and encryption techniques. However, all these solutions offer limited protection against all types of attacks, especially zero-day attacks. In this context, the intrusion detection seems to have much more potential for better security systems and several recent research works highlights the growing interest for this technology. Traditional methods for network intrusions detection are mainly based on signatures of known attacks. They detect network attacks by comparing patterns of active connections to the attack patterns provided by human experts. Unfortunately, the major drawback for this technique is the offset, often important, between the appearance of a new attack and the updating of the attack signature database. To address this issue many nowadays research activities are directed to the automation of the generation of attack signatures for misuse based intrusion detection systems [8,20,22].

In this paper, we propose a novel approach for automatically generating signatures and attack scenarios for a network intrusion detection system. In this approach, the generation of attack scenarios is based on the inbound and outbound traffic collected from a honeypot system. The suspected traffic captured from the honeypot system is first pre-processed by filtering and formatting the traffic content. Then, a novel hashing technique is used to generate fingerprints (signatures) of the formatted traffic. The generated signatures are then used to construct attack scenarios. To detect intrusions, we use the longest common subsequence (LCS) algorithm to calculate the similarity between the actual analysed traffic and the attack scenarios generated from the honeypot traffic. Finally, to evaluate our approach, we conduct experiments on DARPA'99 and UNSW-NB15 traffic. The experimental results show that the performance of our proposed approach outperforms many other similar approaches proposed in the literature.

The rest of this paper is organized as follows: Sect. 2 presents an overview of the related works. In Sect. 3, we

✉ Ammar Boulaiche
ammar.boulaiche@gmail.com

1 Computer Science Department, University de Bejaia, 06000 Bejaia, Algeria

2 Computer Security Research Laboratory, University Of Quebec in Outaouais, Quebec, Canada

present the architecture of our proposed approach. In Sect. 4, we present the generation of signatures from messages using our novel hashing technique. In Sect. 5, we introduce the longest common subsequence (LCS) algorithm and show how the later is used to calculate similarities between different attack scenarios. Our experimental study and practical results are discussed in Sect. 6. Finally, Sect. 7 concludes this paper.

2 Related works

In the last years, automatic signature generation for intrusion detection systems has been an active subject and a number of techniques have been proposed in the literature. One of the first tentative to automate the generation of attack signatures is described in [10]. The system, called Honeycomb, generates signatures by using a pattern-detection technique and a packet header conformance tests on honeypots traffic. To detect attack patterns in flow content, Honeycomb applies the longest common subsequences algorithm [2] to binary strings corresponding to the exchanged messages. Two years later, Li et al. [11] proposed Hamsa, an automated signature generation system for polymorphic worms. Based on a worm flow classifier, Hamsa separates the captured traffic flows into suspicious one, sent to suspicious traffic pool and normal one saved in a normal traffic reservoir. Part of the normal traffic reservoir is, then, selected to construct a normal traffic pool. Finally, the suspicious and normal traffic pools are given as input to generate a worm signature. Mohammed et al. [12] propose an automated signature generation system called Honeycyber. The proposed system is based on a double honeynet architecture, in which worm's outbound connections made from the first honeynet (a network of honeypots) are redirected to the second honeynet, and those made from de second honeynet are redirected to the first honeynet. From the traffic of these two honeynets, distinct tokens which appear in every worm traffic instance are extracted to generate a worm signature. Griffin et al. [5] propose the "Hancock" system that automatically generates string signatures for the Symantec antivirus software. The generated signature consists of a contiguous byte sequence that can match many variants in a malware family. The system generates and tests a set of signature candidates based on three types of heuristics: probability based and disassembly based heuristics to filter candidate signatures and diversity based heuristics to select good signatures among these candidates. Tahan et al. [20] propose Auto-Sign, an automatic method for extracting unique signatures of malware executables. The method was designed to operate in two phases: setup phase and signature generation phase. In the setup phase, it creates common function library (CFL) and common threat library (CTL). Then in the signature generation phase, good candidate signatures are generated and ranked in order to select the best one. The minimization of false positives rate is provided by disregarding signature candidates which appear in benign executables. Tang et al. [21] propose PolyTree, a new Network-based signature generation system to defend against polymorphic worms. The proposed system is composed of two components: signature tree generator and signature selector. The signature tree generator is mainly designed to incrementally construct a signature tree, whereas, the signature selector is designed to select a set of accurate signatures which are used to detect new worm attacks. Wang et al. [22] propose a novel automatic signature generation approach based on regular expression formalism with a certain subset of standard syntactic rules. The approach involves four procedures: (1) pre-processing, used to extract application session payload, (2) tokenization, used to find common substrings and incorporate position constraints, (3) multiple sequence alignment, used to find common subsequences, and (4) signature construction, used to transform the results into regular expressions. Jain et al. [8] proposed an hybrid approach that combines honeypots with both signature based and anomaly based detection. In this architecture, three levels of defense are described. In the first level, a signature based detection system is used to detect and block known worm attacks. In the second level, an anomaly based detection system is used to detect deviations from the normal behavior, and in the third level, honeypots are used to detect zero day attacks. In this last level, a controller is implemented to redirect traffic among various honeypots deployed in the honeyfarm, and the longest common subsequence algorithm is used to generate attack signatures.

Compared to previous related works, the present work goes beyond the simple detection of malicious code signatures and can also detect malicious behaviours. We do this by automatically generating an attack scenario for each attack type. The advantage of such an approach is its ability to provide a consistent and detailed description of malicious attacks, which makes the network administrator task even easier. Note that the considered attack scenarios are low-level scenarios (packets-level scenarios), and they are different from high-level attack scenarios addressed in the literature. The advantage of such an approach is a finer, description granularity and a more precise detection. Another difference between our scenarios based system and other systems is that, most of the previous approaches focus on worm-related malware attacks, where a signature is extracted from the traffic that the malware creates when it attempts to spread through the network. However, since our approach tries to capture attacker's behaviours through attack scenarios, it can ensure protection against various types of attacks including reconnaissance attacks, denial of service (DoS) attacks, SQL injection attacks, malware attacks, etc. Note that, the proposed architecture favors the detection of zero-day attacks.

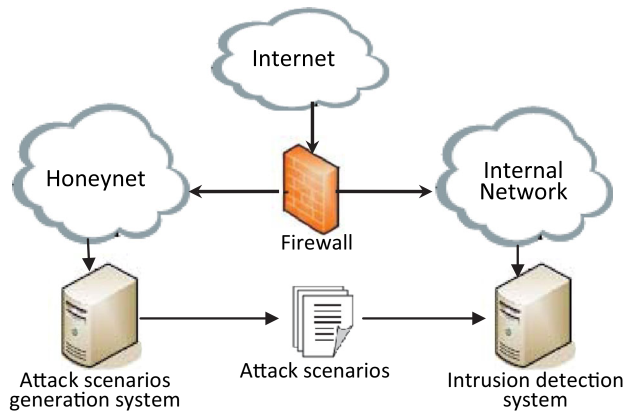
Internet
Honeynet
Firewall
Internal Network
Attack scenarios generation system
Attack scenarios
Intrusion detection system

Fig. 1 Proposed intrusion detection architecture

Indeed, in such an architecture the likelihood that the zero day attack falls soon into the honeypot trap becomes very high, especially when the later is placed in a strategic location and is well configured (simulate enough services while avoiding too much exposure so as not be detected as a honeypot). Now, since all attack scenarios are built from the honeypot traffic, the system has a good chance to be aware of the attacks before they reached production servers. Moreover, in most previous approaches, the generation of signatures is often based on the extraction of tokens that are common to all the flows in the suspicious pool (traffic). However, in these, tokens-based methods, finding good tokens from unknown flows is known to be an NP-hard problem [11], especially when tokens are extracted from noisy suspicious pools which may contain some normal flows. The presence of such noise will increase the complexity of signature generation algorithms and reduce the quality of the generated signatures [11]. To avoid this problem, we propose in this work a novel technique to generate attack signatures based on similarity-preserving hashing approach instead of the tokens extraction technique. We also use the longest common subsequence algorithm to compute the similarity between different attack scenarios.

3 Proposed intrusion detection architecture

The proposed architecture of our intrusion detection system is presented in Fig. 1. This architecture is composed of two main parts; intended to be distributed on two separate domains with different access policies. Indeed, the honeypot part should be implemented in a least restricted domain (i.e.: DMZ domain) to be fully functional. The first part of the architecture is the automatic attack scenarios generation system and the second part is the intrusion detection system.

The automatic attack scenarios generation part is designed to be executed beside a honeypot system; its main task is to sniff the inbound and outbound honeypot traffic and build attack scenarios. These scenarios are then exported, in real

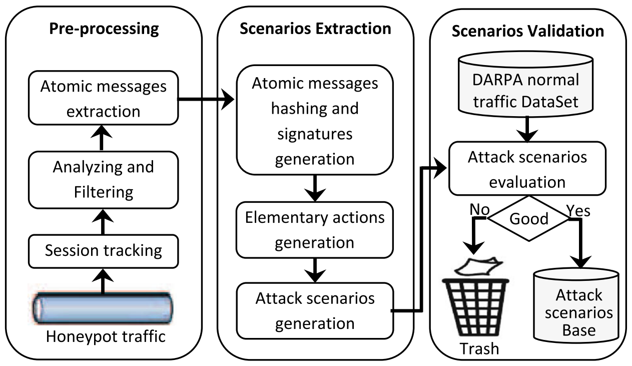
Pre-processing
Atomic messages extraction
Analyzing and Filtering
Session tracking
Honeypot traffic
Scenarios Extraction
Atomic messages hashing and signatures generation
Elementary actions generation
Attack scenarios generation
Scenarios Validation
DARPA normal traffic DataSet
Attack scenarios evaluation
No
Good
Yes
Trash
Attack scenarios Base

Fig. 2 Architecture of the attack scenarios generation part

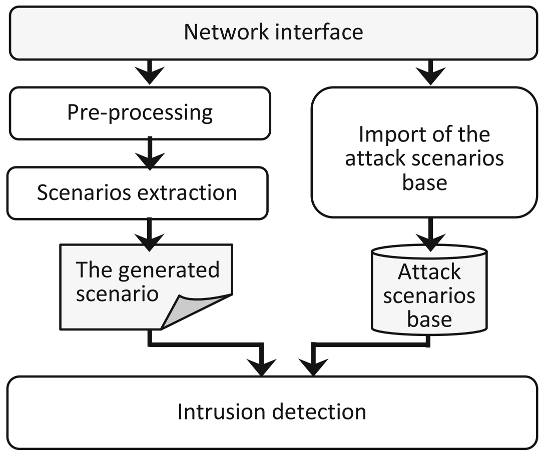

Fig. 3 Architecture of the intrusion detection part

time, towards the intrusion detection system. The architectures of the automatic attack scenarios generation part and that of the intrusion detection part are depicted in Figs. 2 and 3 respectively. There is four principal components: pre-processing module, scenarios extraction module, scenarios validation module and intrusion detection module.

3.1 Pre-processing module

The pre-processing module is mainly designed to work in the same way as in the TCP/IP protocol stack. It captures packets from a honeypot network, and tracks them into sessions (bidirectional flows of packets). Each packet is identified by a 5-tuple (source IP address, destination IP address, source port number, destination port number, IP protocol number). The module has, also, the task to analyse and filter the traffic by eliminating unnecessary packets such as network flow control packets. Indeed, high-level protocol messages can be transmitted in one or more TCP segments. Therefore, the pre-processing module must reassemble all these TCP segments to rebuild the original messages as sent by the high-level protocol. The obtained messages are called atomic messages.

3.2 Scenarios extraction module

The scenarios extraction module is used to build elementary actions and attack scenarios, which are saved into two separate files, namely *Elementary Action File* and *Attack Scenario File*. In our approach, an attack scenario is built as an ordered sequence of elementary actions exchanged during a communication session. An elementary action represents a set of information and features that characterize an atomic message sent from/to an attacker. In this paper, we define an elementary action by four features or fields, namely the action code, the action description, the action actor and the action signature. The action code (ActCode) is an incremental numeric or alphanumeric string that uniquely identifies each elementary action. So, if an elementary action is captured, the scenarios extraction module will first check whether it is already saved in the *Elementary Action File*. If so, the corresponding action code is returned; otherwise, a new action code is incrementally generated, assigned to the new elementary action and the whole is stored in the Elementary Action File. The action description (ActDesc) provides information and features on an elementary action. Its value is composed of two parts separated by the ':' character. The first part contains the last protocol used in the action (ARP, RARP, ICMP, TCP or UDP), whereas the second part contains a short description for the operation performed by the elementary action, its value is automatically set by analyzing and interpreting some specific fields of the last protocol header of the packets composing the elementary action. The protocol header fields interpreted here can be the operation field for the ARP header (Request or Reply), the type and code fields for the ICMP header (Echo Reply, Destination Unreachable, etc.), the flags field for the TCP header (SYN, ACK, PSH, SYN + ACK, ACK + PSH, etc.), etc. The action actor (ActActor) determines the actor which has performed the elementary action: attacker or victim. Finally, the action signature (ActSig) contains a signature (hash) of the atomic Message. Our technique, based on hashing, allows us to overcame the problem of storage space while offering a highly efficient message similarity detection. A detailed description of this novel technique is given in Sect. 4. An example of an elementary action and an attack scenario generated by this module is shown in Figs. 4 and 5.

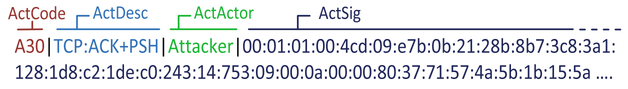
ActCode
ActDesc
ActActor
ActSig
A30|TCP:ACK+PSH|Attacker|00:01:01:00:4cd:09:e7b:0b:21:28b:8b7:3c8:3a1:
128:1d8:c2:1de:c0:243:14:753:09:00:0a:00:00:80:37:71:57:4a:5b:1b:15:5a ....

Fig. 4 Example of an elementary action

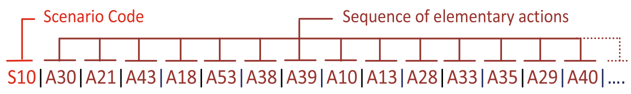
Scenario Code
Sequence of elementary actions
S10|A30|A21|A43|A18|A53|A38|A39|A10|A13|A28|A33|A35|A29|A40|....

Fig. 5 Example of an attack scenario

3.3 Scenarios validation module

Honeypots [19] are servers with no production value, so legitimate users have nothing to do with them. Consequently, each attempt to contact these systems via the network is considered suspect by default. However, this does not mean that all interactions with honeypots are true attacks. Indeed, normal traffic is sometimes redirected towards honeypots by mistake, or by the normal interaction of some network protocols, such as the NetBIOS protocols which periodically send packets to discover their network neighborhood. In fact, the normal traffic redirected towards honeypots by mistake is very rare in practice; their impact on the quality of the generated attack scenarios will, thus, be negligible. However, for the traffic corresponding to the normal interaction of legitimate network protocols, such as NetBIOS protocol, its appearance in the honeypot systems is frequent. This could negatively affect the quality of the generated attack scenarios and then should be filtered. Fortunately, this kind of normal traffic is very common and regularly received by regular machines. As a result, the corresponding attack scenario will necessarily generate a tremendous volume of false alarms, contrary to those corresponding to real attacks. It is therefore very easy to distinguish between malicious and legitimate traffic of honeypot systems. This can be done by detecting the presence of the generated attack scenarios on the traffic of a safe network (free of attacks) for a sufficient period of time, then just count the number of alarms generated to determine whether the scenario corresponds to malicious or legitimate traffic. Hence, to prevent the generation of false attack scenarios, a scenarios validation module is implemented. In this module, a newly generated scenario is first tested on a normal traffic dataset that must contain sufficient amount of normal traffic, in our case it contains more than 100,000 normal communications (redundant traffic must not be removed from this normal dataset). If the test is positive (the scenario is found in this traffic), this corresponds to an IDS false positive. Then, if the number of false positives generated exceeds a certain threshold value, this scenario is ignored; otherwise it is saved in the attack's scenarios base. Furthermore, in some cases an attack scenario can be close enough to some frequent normal scenarios to trigger often the IDS on normal traffic, generating, then, a lot of false alarms. In this case, a good security strategy is to ignore this attack scenario instead of managing a considerable amount of false alarms. Accordingly, our validation module can help to eliminate the attack scenarios corresponding to real attacks that generate a lot of false alarms. This can happen because the detection of intrusion activities in the proposed approach is based on a partial matching between the scenario corresponding to the captured traffic and those of the malicious activities saved in the attack scenarios base. A generated attack scenario may thus wrongly detect all "very" similar normal activities as attacks. Hence, in our validation module when a generated scenario is positively detected on the normal traffic base, then it either corresponds to a normal scenario or an attack scenario very close to frequent normal traffic. In the both cases the scenario should be ignored.

3.4 Intrusion detection module

The intrusion detection module is used to detect intrusions by using the LCS algorithm to compute the similarity between the generated scenario and those of the attack scenarios base. A detailed description of the functioning principle of this module is given in Sect. 5.

4 Generation of elementary action signature

In this section, we introduce a new technique for building compact signatures from brut atomic messages produced by the analysed traffic. Note that, the use of brut atomic messages to build attack scenarios is ineffective in practice. This is mainly due to the required memory space and the inefficiency induced in the attack scenarios comparison process. To implement our solution, we use a hashing technique. However, due to properties of traditional hashing techniques, in which a single bit modification will significantly alter its cryptographic hash, these techniques can not be used to handle similarity between attack scenarios. A potential solution is, then, to apply a hashing function that preserves similarity.

Over the last years, many similarity preserving hash functions have been proposed. The most widely known are the Context-Triggered Piecewise Hashing (CTPH) [9] and the Similarity Digests Hashing (SDHash) [16,17], which are implemented in the tools *sdeep* and *sdhash* respectively. In the CTPH technique, the input data is divided into many parts based on a pseudo random function, and a hash of each piece is produced and finally concatenated to produce the fingerprint. In SDHash technique, the improbable byte sequences are statically identified, afterward, the hash of each sequence is produced and inserted into a bloom filter. Unfortunately, these techniques are primarily designed for large messages (10,000 bytes and above), therefore, if we use them with small message sizes like those of our atomic messages (97% are below 10,000 bytes) the quality of the produced hashs would be significantly reduced. This is, mainly due to the fact that the feature selection for the hash algorithm is done at the bit level, and this selection can not be realized on smaller messages since it contains fewer bytes. In other words, small messages do not contain enough information at the bit level to produce good quality hash. To solve this problem, we propose a novel approach based on the crossword frequencies of the input data.

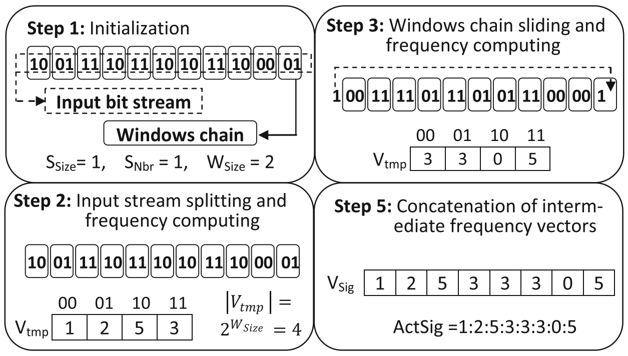

Fig. 6 Example of our signature generation

The basic idea of our similarity preserving hash technique is to generate a vector of crossword frequency for each input data and instead of comparing original input data, we only need to compare their hashes by computing their vector distances. The generation of crossword frequency vector is based on a sliding windows chain technique, in which the input bit stream is split into a windows chain as shown in Fig. 6. Now, and before going into further details, let us first define some notations.

Input data is processed in the bit stream level, it is denoted by D_{IN} and its size is denoted by $|D_{IN}|$. For the sliding windows chain technique, the size of the windows is a constant for all input data and it is denoted by W_{Size}; whereas, their number, denoted by W_{Nbr}, is different for different input data, depending on the size of data: $W_{Nbr} = |D_{IN}| / W_{Size}$. Where "/" represents the integer division operator. Furthermore, for each hash generation step, the windows chain is shifted by n bits; this number of bits is called sliding step size, denoted by S_{Size}. The number of shifting (sliding number) is equal to the number of sliding steps, and is denoted by S_{Nbr}. For each sliding step, an intermediate frequency vector, denoted by V_{tmp}, is created. At the last step, the previous intermediate frequency vectors are concatenated to produce a final frequency vector denoted by V_{Sig}. In the frequency vectors, indexes represent the windows values, whereas, the vector cells content represents the frequencies of windows values. The size of an intermediate vector should be large enough to contain all possible values that a window can contain, i.e.: $|V_{tmp}| = 2^{(W_{Size})}$.

Hereafter, we present the algorithm implementing the similarity preserving hash technique (FSCHash), which involves five steps:

Step 1. Initialization:
$S_{Size} \leftarrow$ sliding step size,
$S_{Nbr} \leftarrow$ sliding number,
$W_{Size} \leftarrow$ window size,
Note: $S_{Nbr} \times S_{Size}$ must always be lower than W_{Size}.

Step 2. The input bit stream (input data) is split into a windows chain, then the frequencies of words (bits sequences) appearing in the windows chain are calculated and stored in the intermediate frequency vector V_{tmp}.

Step 3. The windows chain is shifted by S_{Size} bits, and the frequencies of the new windows values are calculated and stored in a new intermediate frequency vector V_{tmp}.

Step 4. Repeat step 3 until the sliding number (S_{Nbr}) is completed.

Step 5. Concatenate all intermediate frequency vectors to construct a final frequency vector V_{Sig}.

Now, due to the vectorial structure of the produced hashing, several techniques can be used to calculate similarity between two signatures of elementary actions, such as Minkowski Distance, Manhattan Distance, Mahalanobis Distance, etc. The most widely used is the euclidean distance (Eq. 1).

$$d(S1, S2) = \sqrt{\sum_{i=1}^{n} (S1_i - S2_i)^2} \tag{1}$$

However in the present work, we choose to use the Bhattacharya distance (Eq. 2) because it gives better practical results in our analysed dataset.

$$d(S1, S2) = \sum_{i=1}^{n} \sqrt{(S1_i \times S2_i)} \tag{2}$$

5 Sequence-based similarity measurement

In practice, a perfect matching between two scenarios of the same attack is rare. This is due to the network characteristics and the behavior of attackers that, often, introduce cosmetic changes on their attack scenarios to avoid their detection. However, in most cases, a partial matching could be sufficient to detect similarity between two attack scenarios.

A variety of different techniques have been introduced to measure partial matching between two given sequences. In our approach, we use Longest Common Subsequences (LCS) [2,6] technique which measures the longest subsequence of symbols that appears in both input sequences. The LCS algorithm was designed primarily to compute similarities between strings [4,7], but today it is one of the most well-known and the most commonly used solution for many problems of different areas, such as pattern recognition, data mining, file comparison, biological sequence comparisons, etc.

Definition 1 Given a sequence S, any sequence obtained by deleting some of the characters from S is said to be a subsequence of S.

Definition 2 The longest common subsequence (LCS) of two sequences $X = \langle x_1, x_2, \ldots, x_N\rangle$ and $Y = \langle y_1, y_2, \ldots, y_M\rangle$ is a sequence $Z = \langle z_1, z_2, \ldots, z_K\rangle$ such that Z is a subsequence of X and Y and if T is a subsequence of X and Y then $|Z| > |T|$

For example, let $X = \langle abcdgh\rangle$ and $Y = \langle aedfhr\rangle$, then $Z = \langle acg\rangle$ is a subsequence of X, $Z = \langle ad\rangle$ is a common subsequence of X and Y and $Z = \langle adh\rangle$ is a longest common subsequence of X and Y.

To compute the LCS between two attack scenarios $S_1 = \langle x_1, x_2, \ldots, x_n\rangle$ and $S_2 = \langle y_1, y_2, \ldots, y_m\rangle$, we use an algorithm based on the basic dynamic programming [3,7]. S_1 and S_2 are mapped to a matrix L of $n+1$ rows and $m+1$ columns.

$$L = \begin{pmatrix} e_{1,1} & \cdots & e_{1,m+1} \\ \vdots & \ddots & \vdots \\ e_{n+1,1} & \cdots & e_{n+1,m+1} \end{pmatrix} \tag{3}$$

where, rows 2 to $n+1$ correspond to the elements of attack scenario S_1, while columns 2 to $m+1$ correspond to the elements of attack scenario S_2. The first row and column, with index 1, do not correspond to any specific element and they are initialized to 0. Entry $L[i, j]$ represents the LCS between the first i characters of S_1 and the first j characters of S_2, and can be incrementally computed from $L[i-1, j]$, $L[i-1, j-1]$ and $L[i, j-1]$ using the following recursive equation:

$$L[i, j] = \begin{cases} 0 & \text{if } i=1 \text{ or } j=1 \\ L[i-1, j-1]+1 & \text{if } x_i = y_j \text{ and } i, j > 1 \\ max\{L[i-1, j], L[i, j-1]\} & \textit{otherwise.} \end{cases} \tag{4}$$

The last element $L[n+1, m+1] = e_{n+1,m+1}$ holds the length of the longest common subsequence between S_1 and S_2, which is denoted by $\left|LCS(S_1, S_2)\right|$.

For example, if we have two attack scenarios $S_1 =< A, C, B, D, A, D, B, C >$ and $S_2 =< A, D, C, B, A, B, D >$, by running this recursive equation, the visualization of the LCS computing process is shown in Table 1 and Fig. 7.

In the the present work, we use normalized similarity values given by the formula (5).

$$Sim(S_1, S_2) = 2 \times \frac{\left|LCS(S_1, S_2)\right|}{\left|S_1\right| + \left|S_2\right|} \tag{5}$$

6 System implementation and performance analysis

We used a C# language to implement the software prototype, the later is composed by six main modules. Three of them

Table 1 Evaluation process for the longest common subsequence between S_1 and S_2

		A	C	B	D	A	D	B	C
	0	0	0	0	0	0	0	0	0
A	0	↖1	←1	←1	←1	↖1	←1	←1	←1
D	0	↑1	↑1	↑1	↖2	←2	↖2	←2	←2
C	0	↑1	↖2	←2	↑2	↑2	↑2	↑2	↖3
B	0	↑1	↑2	↖3	←3	←3	←3	↖3	↑3
A	0	↖1	↑2	↑3	↑3	↖4	←4	←4	←4
B	0	↑1	↑2	↖3	↑3	↑4	↑4	↖5	←5
D	0	↑1	↑2	↑3	↖4	↑4	↖5	↑5	↑5

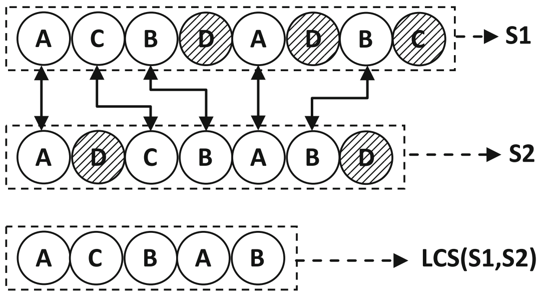

Fig. 7 Example of calculating the LCS of two attack scenarios

form the attack signature generation part. The three others form the detection part. The performance analysis has been performed with a 3.06 GHz Intel Core i3 CPU and 4 GB RAM computer, running Windows 7 Professional operating system.

We conduct three experiments for the evaluation of our proposed model. In the first experiment, we demonstrate the efficiency of our similarity preserving hash technique, proposed in Sect. 4. In the second experiment, we investigate the practical quality of the, automatically generated, attack scenarios by testing their ability to detect attacks on two different datasets. While the third experiment is mainly devoted to evaluate the practical performance of the generated scenarios in terms of their matching time.

6.1 Data for the experiment

We used the raw network traffic of two different datasets, namely, the DARPA'99 dataset and the UNSW-NB15 dataset.

6.1.1 DARPA'99 dataset

This first dataset was developed by the MIT Lincoln laboratory and Air Force Research Laboratory [1]. It contains training data formed by a 3 weeks (weeks 1–3) of captured network traffic in the form of TCP dump and audit data containing approximately 5 million connections. The first and third week of the training dataset are free of attacks, whereas the second week includes 43 instances of 18 labeled attacks and each instance is, in general, formed by more than one connection. The DARPA'99 dataset has, also, a separate testing data. It includes 2 weeks of captured traffic (weeks 4 and 5), with approximately 2 million connections of normal/malicious traffic. The malicious traffic includes 201 instances of 58 attack types, where 40 types are new attack types that don't exist in the training dataset. In addition to tcpdump and audit data, DARPA'99 dataset provides some additional information about the captured attacks: attack flow identifiers, date of the attacks, the time of the arrival of the first packet in the flow, attacks durations, source port, destination port, source IP address, destination IP address, etc. Attacks provided in this dataset belong into one of the following five categories: denial of service (dos), remote to local (r2l), user to root (u2r), probe and data compromise. The dos attack is an unauthorized attempt to disrupt the normal functioning of a victim host or network (e.g., mail bomb attacks); The r2l attack is an unauthorized access from a remote machine (e.g., ssh trojan attacks); The u2r attack is an unauthorized access to local superuser privileges by a local unprivileged user (e.g., buffer overflow attacks); The probe attack is an unauthorized host and port scans to gather information or to find known vulnerabilities (e.g., port scanning attacks). Finally, data attack is an unauthorized access or modification of data on local or remote host (e.g., frame spoofer attacks).

We used the DARPA'99 dataset to evaluate the proposed approach, because it is well-studied, well-documented and publicly available trace in the area of intrusion detection, and is still the most widely used dataset for testing intrusion detection systems. Especially for signature based intrusion detection, where a whole set of attack and normal raw network traffic is available with detailed information about each attack traffic (source and destination IP addresses, source and destination ports, attack starting time, attack duration, etc.). Unfortunately, this is not the case for most of recent datasets despite the importance of such information to extract the attack traffic content and generate the appropriate signatures. However, in spite of these incentives, DARPA dataset is decade-old and is therefore not enough to evaluate recent intrusion detection approaches. For this reason, the proposed approach is also evaluated using UNSW-NB15 dataset.

6.1.2 UNSW-NB15 dataset

The UNSW-NB15 is a very recent dataset which is created by the cyber security research group at the Australian Centre for Cyber Security (ACCS) [13]. It contains about 100 GB of raw network traffic, captured during two simulation periods of 16 and 15 h and proceeded on January 22, 2015 and February 17, 2015 respectively. The raw data includes both real modern normal activities and synthetic contemporary attack behaviors. Synthetic contemporary attacks provided in this dataset can be classified into one of the following nine categories: *Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic, Reconnaissance, Shellcode* and *Worms* [14]. The *Fuzzers* attack is an unauthorized attempt to discover security loopholes in the victim system by feeding it with a massive inputting of random data. The *Analysis* attack is an unauthorized attempt to penetrate the web applications via ports (e.g. port scans), emails (e.g. spam) and web scripts (e.g. HTML files). The *Backdoor* attack is an unauthorized access from a remote machine. The *DoS* attack is an unauthorized attempt to disrupt the normal functioning of a victim host or a network. The *Exploit* attack is a sequence of instructions that takes advantage of a glitch, bug or vulnerability on the victim system. The *Generic* attack is an attempt to cause a collision on the victim system without respecting the configuration of the block-cipher. The *Reconnaissance* attack is an unauthorized attempt to gather information about a victim system to evade its security controls. The *Shellcode* attack is an unauthorized attempt to penetrate a slight piece of code starting from a shell to control the compromised machine. Finally, the *Worms* attack is an attack in which the attacker replicates itself to spread on other computers.

6.2 First experiment

The aim of this first experiment is to verify the efficiency of our similarity preserving hash technique, in which the age of the used attack data is less important, so only one dataset is used here. In this evaluation, we performed the following steps:

1. We produced a dataset containing original atomic messages (that we call *OAMD*) extracted from the DARPA'99 test dataset attacks. This dataset contains about 100,250 atomic messages.
2. We generated four other datasets called Changed Atomic Message Datasets ($CAMD_i, i = 1, \dots 4$). The $CAMD_1$ ($CAMD_2$, $CAMD_3$, $CAMD_4$ respectively) is generated by randomly modifying 5% (20, 50, and 80% respectively) of each atomic message in the original *OAMD*.
3. For each of $CAMD_i, i = 1 \dots 4$, we compared the signature of each modified atomic message with the signature of the corresponding original atomic message. The generation and the comparison of these signatures are performed by using our FSCHash technique (as explained in Sect. 4) with the following parameters: W_{Size} = 8 (window size), S_{Nbr}= 1 (sliding number), S_{Size} = 4 (sliding step size) and Bhattacharya Distance to calculate similarity between two signatures. By gradually increasing the similarity threshold, we calculated the number of modified atomic messages which were correctly associated to their original atomic messages, i.e.: when the similarity value with the associated original atomic messages exceeds the similarity threshold value. Results are shown in Fig. 8. For example, in this figure, the comparison of the $CAMD_2$ and the *OAMD* with a threshold of 0.85 (pink color) show that 79% of the atomic messages are correctly assigned.

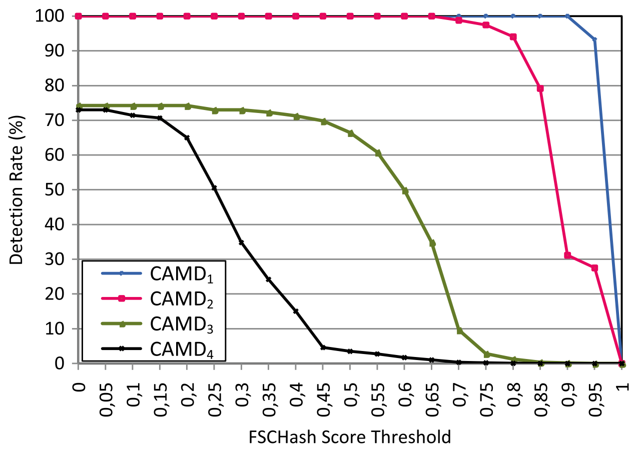

Fig. 8 Detection rates for our FSCHash technique

As shown in Fig. 8, we can clearly see the efficiency of the proposed technique when the change rate of the atomic messages is set to 5% ($CAMD_1$) or 20% ($CAMD_2$). In both cases, the detection rate is near-perfect (nearly 100%) when the detection threshold is 0.95 and 0.80 for $CAMD_1$ and $CAMD_2$ respectively. The detection rate falls to near zero percent when the threshold is more than 0.95 and 0.80 respectively. As for the 50 and 80% change rates corresponding to $CAMD_3$ and $CAMD_4$ respectively, the detection rate is relatively stable around 0.72% when the similarity threshold is less than 0.5 and 0.2 for the $CAMD_3$ and $CAMD_4$ respectively.

However, the behavior on $CAMD_3$ and $CAMD_4$ does not affect the efficiency of our technique. In practice, two signatures are generally qualified as similar when the difference rate between their content is up to 20%. FSCHash demonstrates a good detection rate in these cases. From these results, we can easily deduce that the best value of the similarity threshold is 0.8. With this value, the probability of reporting that two signatures are different, while their difference of content is up to 20%, should be near zero.

The same experiment was repeated over the same *OAMD* and $CAMD_i, i = 1, \dots 4$, using CTPHash and SDHash techniques. The results depicted in Figs. 9 and 10 respectively show the drawbacks of these techniques for network traffic data that have the particularity to range from small to medium size.

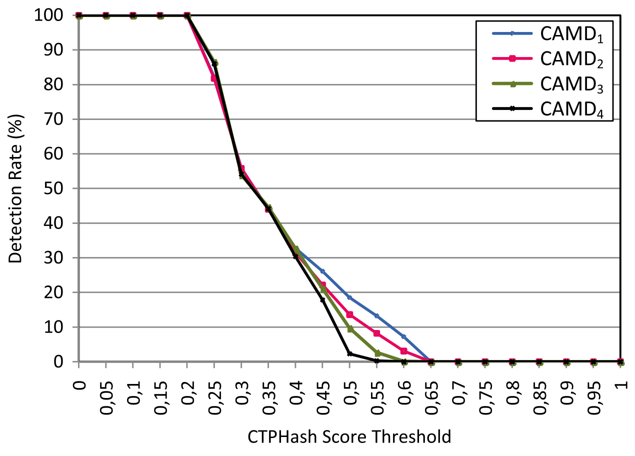

Fig. 9 Detection rates for CTPHash technique (*ssdeep* tool)

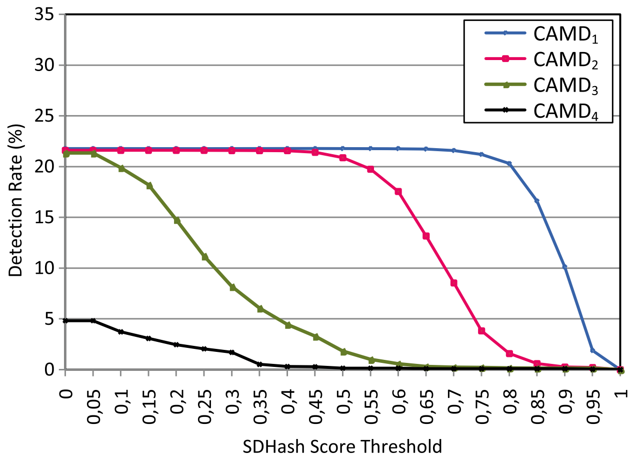

Fig. 10 Detection rates for SDHash technique

Figure 9 clearly shows that the CTPHash technique does not make distinction between *CAMD*$_1$ to *CAMD*$_4$ in terms of detection rate. Likewise, the maximum detection rate of SDHash technique is less than 22% regardless of the change rate applied to datasets as shown in Fig. 10.

In order to evaluate the time cost of our similarity preserving hash technique (FSCHash), we calculate the time required for signature generation by using eight atomic messages of different sizes ranging from 100 to 500,000 bytes. Knowing that all the generated signatures have the same size, we then calculate the time needed for pairwise signature comparison. The same experiment is conducted using the CTPHash and SDHash approaches.

Figure 11 shows the superiority of the FSCHash technique over the two other techniques in terms of signature generation time cost as long as the atomic message size is up to 100,000 bytes. Fortunately, atomic messages having a size over 100,000 bytes are rarely met in practice. From the 100,250 atomic messages of DARPA'99 testing attacks,

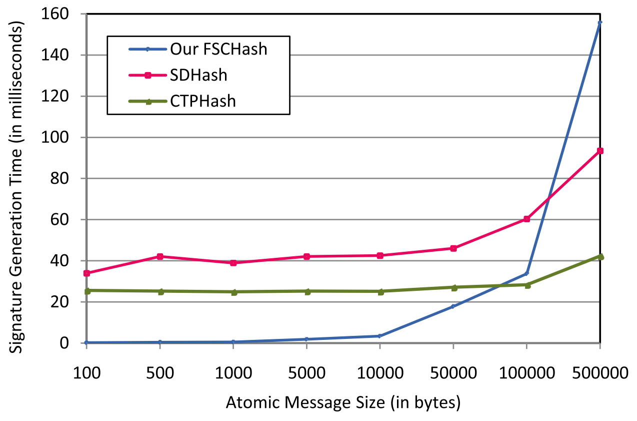

Fig. 11 Signatures generation time

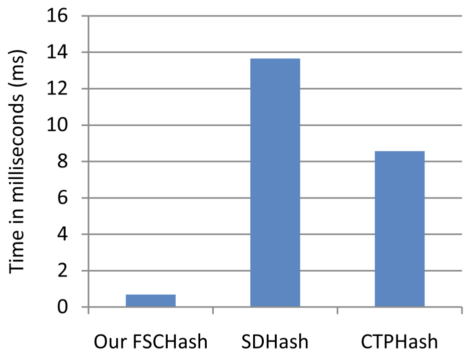

Fig. 12 Signatures comparison time

there is only 17 atomic messages with sizes over than 100,000 bytes.

Likewise, the proposed technique outperforms their counterparts in terms of signature comparison time cost as clearly shown in Fig. 12. Indeed, with the FSCHash technique, the average time is about 0.6 ms which is much better than that required when using the two other techniques (more than 8 ms). This can be explained by the efficiency of the vector-based distance used in our technique over the string-based distance used in the two other techniques.

6.3 Second experiment

6.3.1 Data preparation

As stated above, two datasets were used to evaluate the efficiency of the generated attack scenarios, which are DRPPA'99 dataset and UNSW-NB15 dataset. In both datasets, the raw network traffic composing them is first preprocessed in order to extract the three data subsets required for such evaluation. The three subsets concerned here are: Training Attack Subset (TrAS) that is mainly used to generate the attack scenarios base, Testing Attack Subset (TsAS) that is mainly used to test the generated attack scenarios base in terms of its detection rate (Sensitivity), i.e: the ability of the generated attack scenarios to detect real attacks, and Testing Normal Subset (TsNS) that is mainly used to test the generated attack scenarios base in terms of its specificity, i.e: the ability of the generated attack scenarios base to avoid identifying normal traffic as attack. To extract these three subsets, we performed the following data preprocessing on the two datasets.

a. Data preprocessing on the DARPPA'99 dataset For the DARPA'99 dataset, the testing normal subset TsNS (noted here DARPA-TsNS) is generated by randomly selecting about 110,000 connections from the first and the third weeks of the training data (these 2 weeks are attack-free). Whilst TrAS and TsAS (noted here DARPA-TrAS and DARPA-TsAS respectively) are extracted from the attack traffic of the DARPA'99 testing data (weeks 4 and 5) by the following two steps:

Step 1. All attack connections of the DARPA'99 testing data (weeks 4 and 5) are first extracted by exploiting the additional information provided with the raw tcpdump files of the dataset. The later include details about the location of the attack instances in the data of weeks 4 and 5.

Step 2. The extracted attack connections are randomly divided into training connections (DARPA-TrAS) and testing connections (DARPA-TsAS) so that the number of the training connections is always lower than or equal to that of the testing ones for all types of attack.

In this dataset, the two subsets (DARPA-TrAS and DARPA-TsAS) are generated from the DARPA testing data and not from the DARPA training data for two reasons: first, the additional information provided for the DARPA training subset (week-2) does not contain sufficient details to distinguish between the flows of different attacks unlike the DARPA testing data that provide more details. For example, the DARPA training data does not contain any information about the attack source IP address, the attack duration, the attacked port, etc. Thus, it is very difficult for us to extract and dissociate attack packets from the compressed tcpdump files, especially for overlapping attacks. The second reason is that our proposed system is a misuse-based intrusion detection system with an auto-learning ability. In such a system, the performance evaluation must demonstrate the efficiency of the generated signatures for various attack types. In this case, it is better to rely on the DARPA testing dataset that contains more attack types (58 attack types) than the DARPA training dataset (18 attack types). Moreover, all attacks of the DARPA training dataset are included in the DARPA testing dataset. Therefore, the use of the attacks extracted from the 2 weeks of the DARPA testing data is good enough to evaluate the performance of our proposed model.

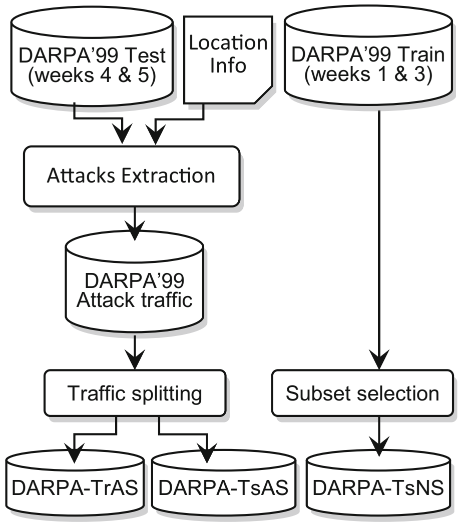

Fig. 13 DARPA-TrAS, DARPA-TsAS and DARPA-TsNS extraction process

The overall data preprocessing on the DARPA dataset is illustrated in Fig. 13. In the Table 2 we describe different attack types and their corresponding occurrence number in these two subsets (DARPA-TrAS and DARPA-TsAS).

b. Data preprocessing on the UNSW-NB15 dataset As stated above, the UNSW-NB15 dataset contains about 100 GB of raw data. It is nearly five times larger than that of the DARPA'99 dataset, which contains a total of about 18 GB of raw data. The use of all the raw data of the UNSW-NB15 dataset is thus a very time and resource consuming task. To prevent this, only 10% of the raw UNSW-NB15 dataset (about 10GB) were used in this study. This part of data is taken from the first ten raw pcap files of the first simulation period that was proceeded on January 22, 2015.

Unlike the DARPA'99 dataset, the UNSW-NB15 dataset has provided no further information about where the attack instances are located in the raw pcap files. Unfortunately, such information is necessary to separate the normal raw data from the abnormal one and then to extract and generate the attack scenarios for the different attack types. To overcome this problem, we analyzed the tcpdump files of the dataset using SNORT [15,18] intrusion detection system and extracted the required information from the alerts log file generated by this later. As shown in Fig. 14, a Snort alert can be represented by ten features: generation time (1), signature ID (2), description (3), classification (4), priority (5), packet type (6), source IP address (7), source port number (8), destination IP address (9) and destination port number (10). From these features, we focused on the first and the last four features that are mainly used to identify the location of every

Table 2 Connection distribution of different attacks in our DARPA-TrAS and DARPA-TsAS subsets

Attacks	Attack connections distribution			Attacks	Attack connections distribution		
	#Train	#Test	#Total		#Train	#Test	#Total
smurf	2	2	4	sendmail	2	2	4
land	1	1	2	sshtrojan	2	2	4
mailbomb	40	1881	1921	xsnoop	1	2	3
processtable	20	662	682	guesstelnet	10	57	67
crashiis	1	2	3	guessftp	10	70	80
warezmaster	1	1	2	ftpwrite	4	4	8
dosnuke	3	4	7	httptunnel	10	15	25
sshprocesstable	20	481	501	phf	5	6	11
pod	2	2	4	sqlattack	1	2	3
warezclient	3	3	6	netcat	10	18	28
apache2	40	1703	1743	imap	1	1	2
syslogd	2	2	4	ppmarcro	10	22	32
neptune	40	72,040	72,080	ncftp	10	37	47
selfping	1	1	2	named	3	3	6
back	20	144	164	guest	10	17	27
ps	7	7	14	snmpget	20	263	283
yaga	9	9	18	netbus	5	5	10
loadmodule	2	2	4	xlock	2	3	5
sechole	9	9	18	guesspop	10	20	30
ffbconfig	1	1	2	dict	10	76	86
casesen	10	14	24	portsweep	20	263	283
eject	5	6	11	satan	40	11,303	11,343
perl	1	2	3	ntinfoscan	10	43	53
fdformat	4	5	9	ipsweep	40	4469	4509
xterm	5	6	11	ls	1	1	2
secret	6	7	13	queso	10	11	21
framespoofer	3	4	7	resetscan	1	1	2

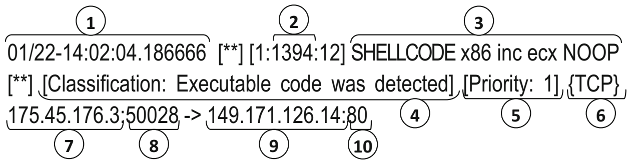
```
01/22-14:02:04.186666 [**] [1:1394:12] SHELLCODE x86 inc ecx NOOP
[**] [Classification: Executable code was detected] [Priority: 1] {TCP}
175.45.176.3:50028 -> 149.171.126.14:80
```

Fig. 14 Example of a Snort alert

attack packets, and the second and the third features that are used to give a unique name for the concerned attack. The later is given by concatenating the first word of the description feature (3) and the signature ID (2) that will be enclosed in brackets (see attack names in Table 3). For instance, the name given for the attack corresponding to the alert of the Fig. 14 is: *SHELLCODE* (1394).

Once the required information about the attack location on the UNSW-NB15 dataset is extracted, the three substs TrAS, TsAS and TsNS (noted here UNSW-TrAS, UNSW-TsAS and UNSW-TsNS respectively) are generated as follow:

Step 1. The normal and malicious data are first separated into two different datasets, namely normal dataset and malicious dataset, using the information generated from the Snort alerts log file.

Step 2. UNSW-TsNS is generated by randomly selecting about 20,000 connections from the normal dataset.

Step 3. The connections of the malicious dataset are randomly divided into training connections (UNSW-TrAS) and testing connections (UNSW-TsAS) so that the number of the training connections is always lower than or equal to that of the testing ones for all types of attack.

The overall data preprocessing on the UNSW-NB15 dataset is illustrated in Fig. 15. Furthermore, we describe in Table 3 different attack types[1] and their corresponding

[1] For more detailed information on any of the attacks listed in the Table 3, just enter the signature ID (the number in brackets of the attack name) in the search area of the web site: https://www.snort.org/search?query=.

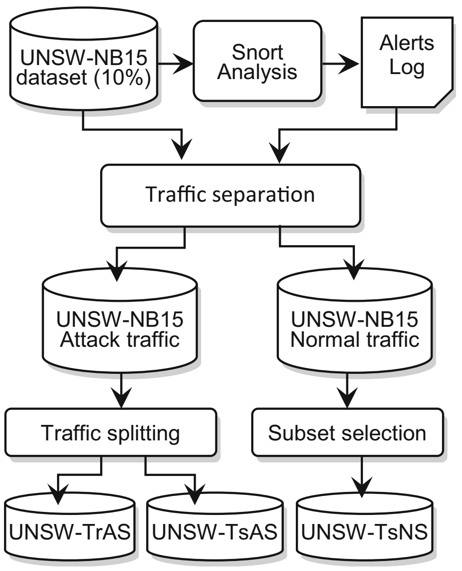

Fig. 15 UNSW-TrAS, UNSW-TsAS and UNSW-TsNS extraction process

occurrence number in these two subsets (UNSW-TrAS and UNSW-TsAS).

6.3.2 *Performance metrics*

The efficiency of an intrusion detection system is often evaluated by computing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), with TP being the number of attacks correctly detected as attacks, TN the number of normal traffic correctly judged as normal traffic, FP the number of normal traffic falsely detected as attacks, and FN the number of attacks wrongly judged as normal traffic. In fact, a good intrusion detection system must minimize both FP and FN together while maximizing TP and TN rates. A high FP rate will seriously affect the credibility of the proposed system while a high FN rate will leave the system vulnerable to intrusions. However, testing the efficiency of an intrusion detection system by using these metrics only is not sufficient and many other metrics are also, often, required, such as accuracy, detection rate, false alarm rate, precision, recall, F-Score, etc. Therefore, the proposed approach is evaluated in terms of true positive rate (also known as the detection rate or the sensitivity or the recall), true negative rate (also known as the specificity), false positive rate, false negative rate, accuracy, precision and f-score, which are mainly computed on the basis of the four previous metrics (TP, TN, FP and FN) as follows:

$$True\ positive\ rate = Recall = \frac{TP}{TP + FN} \times 100\% \tag{6}$$

$$True\ negative\ rate = \frac{TN}{TN + FP} \times 100\% \tag{7}$$

$$False\ positive\ rate = \frac{FP}{TN + FP} \times 100\% \tag{8}$$

$$False\ negative\ rate = \frac{FN}{TP + FN} \times 100\% \tag{9}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{10}$$

$$Precision = \frac{TP}{TP + FP} \times 100\% \tag{11}$$

$$F - Score = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times (Precision + Recall)} \times 100\% \tag{12}$$

β is the relative importance of precision versus recall, it is usually set to 1.

6.3.3 *Experiment description and results*

We performed two separate experiments to evaluate the efficiency of the generated attack scenarios. In the first experiment, the evaluation process is carried out on the DARPA'99 data subset, where the DARPA'99 Training Attack Subset (DARPA-TrAS) is first used to generate the attack scenarios base. The generated attack scenarios base is then tested over the DARPA'99 Testing Attack Subset (DARPA-TsAS) to compute the number of TP and FN, and over the DARPA'99 Testing Normal Subset (DARPA-NrAS) to compute the number of TN and FP. Once these values are computed, detection rate, specificity, false positive rate, false negative rate, accuracy, precision and F-score are also calculated. This process is repeated for many different values of the LCS similarity threshold (values between 0.8 and 1.0) in order to illustrate the impact of this threshold value on the performance of our proposed system and to define the optimal threshold value which gives the best results. The results of this experiment are summarized in Figs. 18, 20, and 22.

In the second experiment, the same evaluation process is repeated on the UNSW-NB15 subsets (UNSW-TrAS, UNSW-TsAS and UNSW-TsNS). The results of this experiment are summarized in Figs. 19, 21, and 23.

Before analyzing these results, it is very important to note that the attack scenarios that generate too many false positives (more than 5% FP) are automatically deleted by the *Scenarios Validation Module* (see Sect. 3). It should also be noted that the testing normal subsets (DARPA-TsNS and UNSW-TsNS) used in the both experiments to compute the number of TN and FP are the same as those used by the *Scenarios Validation Module* to validate the generated attack scenarios. The attack scenarios deleted by the *Scenarios Validation Module* for different values of LCS similarity threshold in the first and the second experiments are indicated in Figs. 16 and 17 respectively.

Table 3 Connection distribution of different attacks in our UNSW-TrAS and UNSW-TsAS subsets

Attacks	Attack connections distribution			Attacks	Attack connections distribution		
	#Train	#Test	#Total		#Train	#Test	#Total
BACKDOOR (7107)	1	1	2	CHAT (542)	1	2	3
EXPLOIT (14769)	1	2	3	FTP (2338)	2	2	4
FTP (2374)	1	1	2	ICMP (399)	2	3	5
ICMP (427)	1	1	2	ICMP (384)	6	6	12
MISC (13839)	1	1	2	MISC (13269)	1	1	2
NETBIOS (2190)	10	13	23	NETBIOS (15930)	1	2	3
NETBIOS (17639)	4	4	8	NETBIOS (7035)	1	1	2
ORACLE (15255)	1	1	2	P2P (2181)	40	3332	3372
POLICY (3825)	10	74	84	POLICY (17668)	2	3	5
POLICY (2044)	20	173	193	POLICY (560)	3	3	6
POP3 (1866)	1	1	2	RPC (1952)	40	6391	6431
RPC (1262)	2	2	4	RPC (575)	3	3	6
RPC (1263)	3	3	6	RPC (576)	3	3	6
RPC (1264)	3	3	6	RPC (577)	3	3	6
RPC (1747)	3	3	6	RPC (1746)	2	3	5
RPC (1265)	3	3	6	RPC (578)	3	4	7
RPC (2005)	2	3	5	RPC (1280)	1	1	2
RPC (579)	40	6417	6457	RPC (11288)	3	3	6
RPC (1960)	2	2	4	RPC (1959)	3	3	6
RPC (1267)	2	2	4	RPC (2080)	3	3	6
RPC (2079)	3	3	6	RPC (1268)	2	3	5
RPC (581)	3	3	6	RPC (1269)	2	3	5
RPC (582)	3	3	6	RPC (1962)	2	2	4
RPC (1961)	2	3	5	RPC (1270)	3	3	6
RPC (583)	3	3	6	RPC (1271)	3	3	6
RPC (584)	2	3	5	RPC (1733)	2	3	5
RPC (1732)	1	1	2	RPC (1272)	1	1	2
RPC (585)	2	2	4	RPC (1273)	2	3	5
RPC (586)	3	3	6	RPC (12458)	1	2	3
RPC (12626)	2	3	5	RPC (2016)	2	3	5
RPC (587)	3	3	6	RPC (12608)	2	2	4
RPC (1275)	2	3	5	RPC (589)	3	3	6
RPC (1276)	2	3	5	RPC (591)	3	3	6
RPC (1277)	3	4	7	SHELLCODE (12798)	5	5	10
SHELLCODE (12799)	3	3	6	SHELLCODE (12801)	1	2	3
SHELLCODE (12802)	3	4	7	SHELLCODE (10504)	10	35	45
SHELLCODE (1390)	2	3	5	SHELLCODE (1394)	20	747	767
SHELLCODE (17324)	2	2	4	SHELLCODE (648)	10	32	42
SHELLCODE (17325)	1	1	2	SHELLCODE (17322)	7	8	15
SHELLCODE (17323)	2	2	4	SHELLCODE (17344)	2	3	5
SHELLCODE (649)	1	1	2	SHELLCODE (17337)	4	4	8
TFTP (1941)	3	3	6				

As Fig. 16 shows, the number of attack scenarios deleted by the *Scenarios Validation Module* and their false positive rate decrease while the threshold values increase. For example, at a threshold of 0.80, 7 attack scenarios out of 58 (12%) generate more than 5% of false positives, among them five attack scenarios generate more than 50% of false positives.

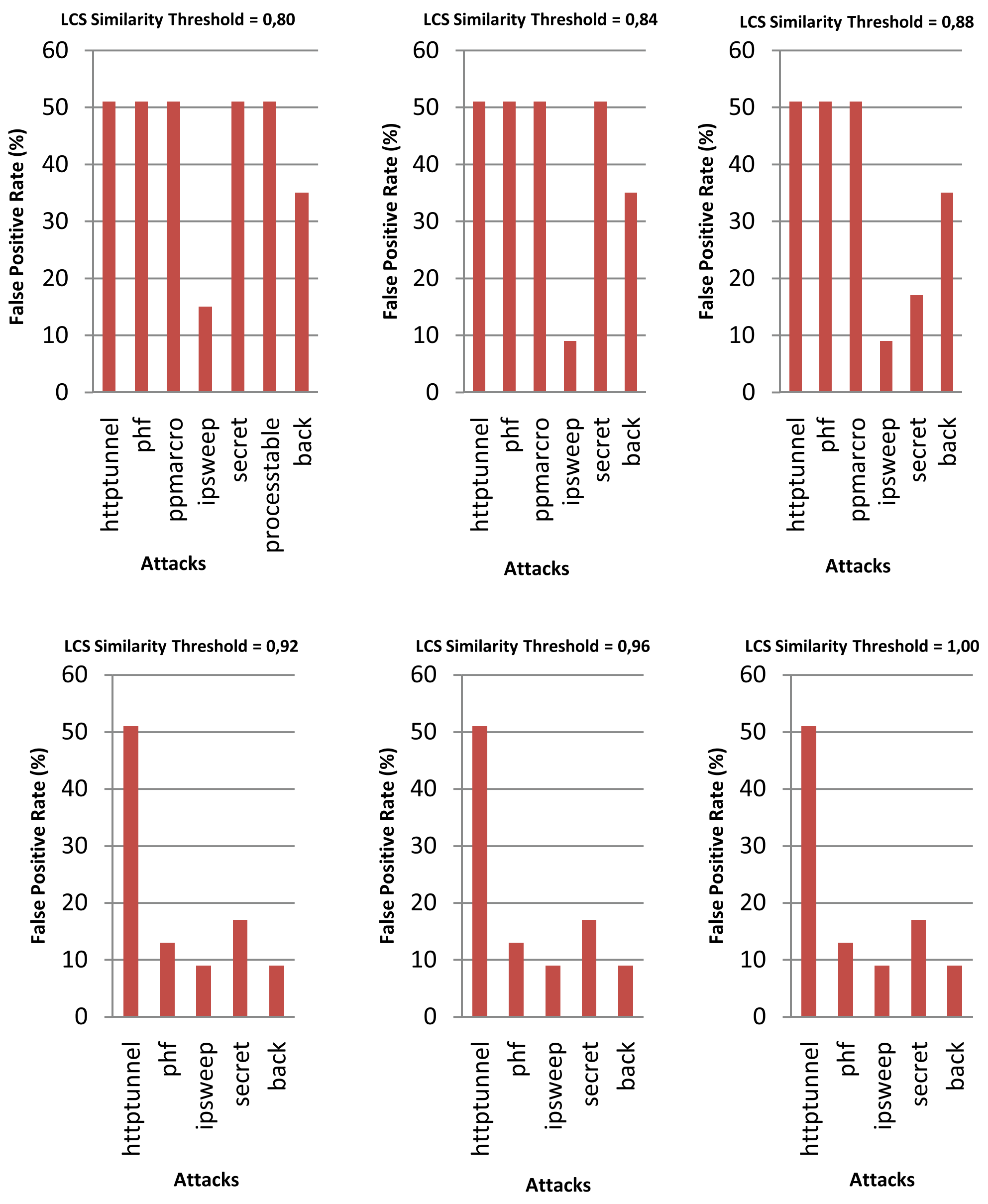

Fig. 16 The attack scenarios deleted by the Scenarios Validation Module during the experiment on the DARPA'99 dataset for different values of LCS similarity threshold

They first decreased to six attack scenarios, from which four scenarios generated more than 50% of false positives at a threshold of 0.84, then they remained at six attack scenarios, from which three generated more than 50% of false positives at threshold 0.88. When the threshold increases to 0.92, only five attack scenarios out of 58 (0.8%) generated more than 5% of false positives. The httptunnel attack still generated more than 50% of false positives while the phf, ipsweep, secret and back attacks generated less than 18% of false positives. The same remark can be made regarding the results shown in Fig. 17.

With regard to the efficiency of the remaining attack scenarios (those that generate less than 5% of FP), Figs. 18 and 19 show clearly that the true positive and the true negative rates remain always higher than 95% whatever the value of the LCS similarity threshold. However, the true positive rate decreases inversely in proportion to the threshold values, while the true negative increases proportionately with the increase of the threshold values. This is due to the fact that whenever the similarity threshold value is smaller, the number of similar scenarios become high. On the other hand, for the testing attack subsets (DARPA-TsAS and UNSW-TsAS), whenever the number of similar scenarios is higher, the number of true positives become high, while, for the testing normal subsets (DARPA-TsNS and UNSW-TsNS), whenever the number of similar scenarios is higher, the number

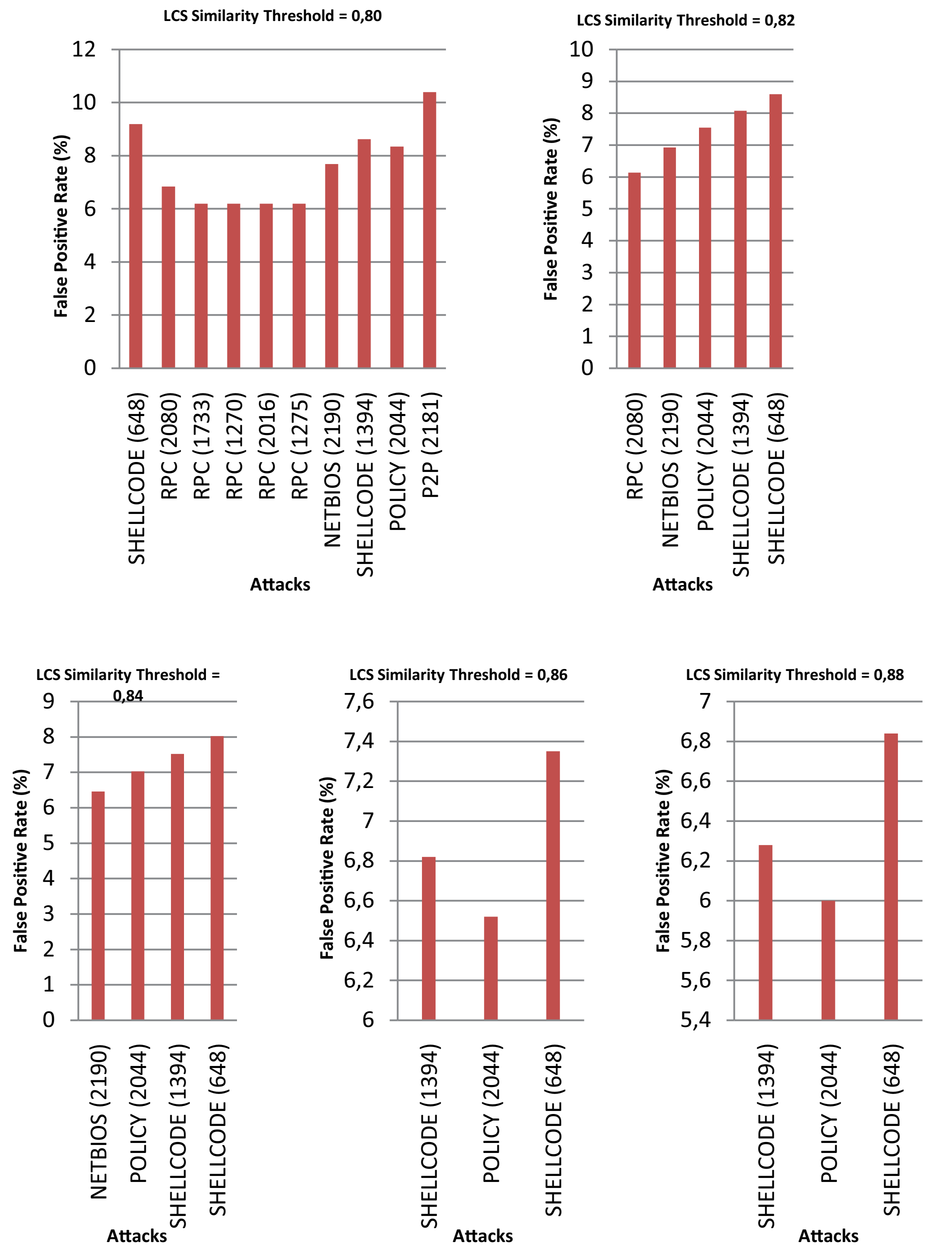

Fig. 17 The attack scenarios deleted by the Scenarios Validation Module during the experiment on the UNSW-NB15 dataset for different values of LCS similarity threshold

of true negatives become small. Consequently, the number of true positives is high at a small LCS similarity threshold and small at a high LCS similarity threshold, whereas the number of true negatives is small at a small LCS similarity threshold and high at a high LCS similarity threshold. The same explanation may be given for the results of Figs. 20 and 21 because the false negative rate is the inverse of the true positive rate, and the false positive rate is the inverse of the true negative rate.

Figures 18 and 19 show some abnormal decrease in the true negative rate when the LCS similarity threshold is on the increase, as at the thresholds of 0.84 and 0.92 in Fig. 18 and the thresholds of 0.84 and 0.86 in Fig. 19. This is due to two reasons: first, when increasing the LCS similarity threshold, the size of the attack scenarios base should be increased, and many new scenarios will be inserted. These new scenarios may generate some new false positives that may decrease the true negative rate. The second reason is that before the threshold of 0.84 in the first experiment (experiment on the DARPA'99 dataset), the number of attacks filtered by the *Scenarios Validation Module* was 7, and then at the threshold 0.84, this number fell to 6, and the processtable attack is no

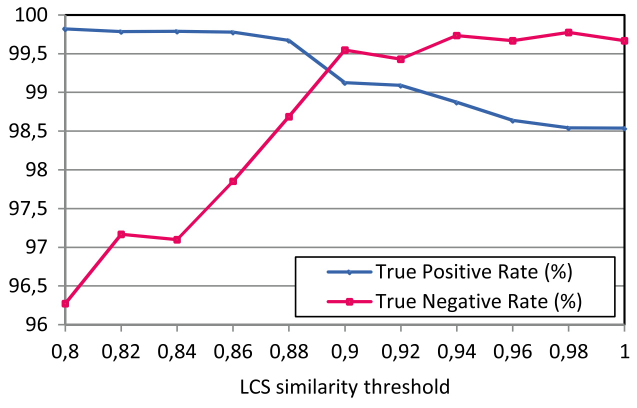

Fig. 18 True positive rate versus true negative rate for the experiment on the DARPA'99 dataset

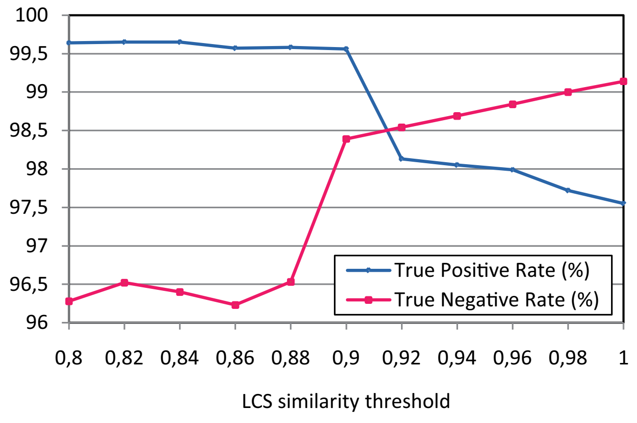

Fig. 19 True positive rate versus true negative rate for the experiment on the UNSW-NB15 dataset

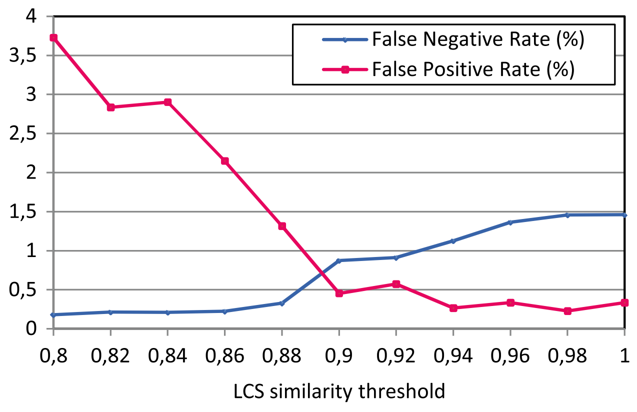

Fig. 20 False positive rate versus false negative rate for the experiment on the DARPA'99 dataset

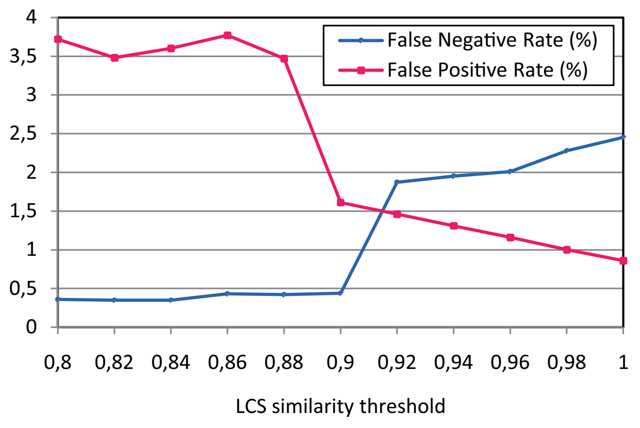

Fig. 21 False positive rate versus false negative rate for the experiment on the UNSW-NB15 dataset

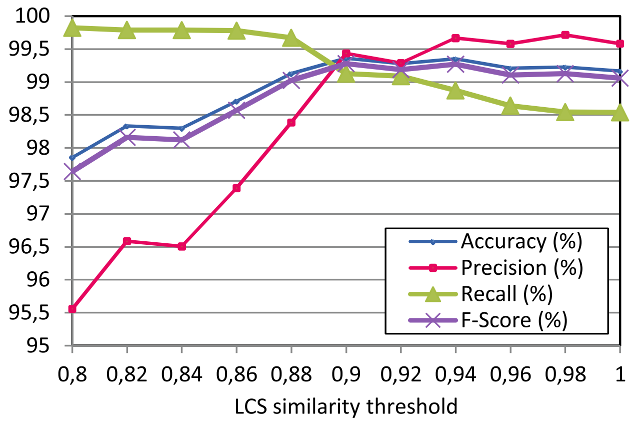

Fig. 22 Accuracy, precision, recall and f-score values for the experiment on the DARPA'99 dataset

longer filtered (see Fig. 16). The scenarios of the last attack may generate some new false positives that may decrease the true negative rate. The same holds for the threshold of 0.92, where the ppmarcro attack is no longer filtered. The same observation and explanation applies for Fig. 19.

For the other metrics of Figs. 22 and 23, we can see that accuracy, precision and f-score metrics increase proportionately with the increase of the LCS similarity threshold values, while the recall metric decreases inversely in proportion to the threshold values. The figures also show that the four measurements gradually converge so closely that they almost meet at the threshold 0.90 in Fig. 22 and the threshold 0.92 in Fig. 23, and then start diverging again. From this last remark, we can deduce that the optimal threshold value which gives the best results for our proposed system is a value between 0.9 and 0.94. In this threshold range, accuracy, precision, recall and f-score all have a value greater than 97.5%.

6.4 Third experiment

To further investigate the performance of the proposed approach, the matching time was also evaluated. In fact, the matching time is mainly influenced by the size of the detected scenario (the number of elementary actions composing the scenario) and the size of the attack scenarios base (the number of scenarios in the base). For this reason, we considered

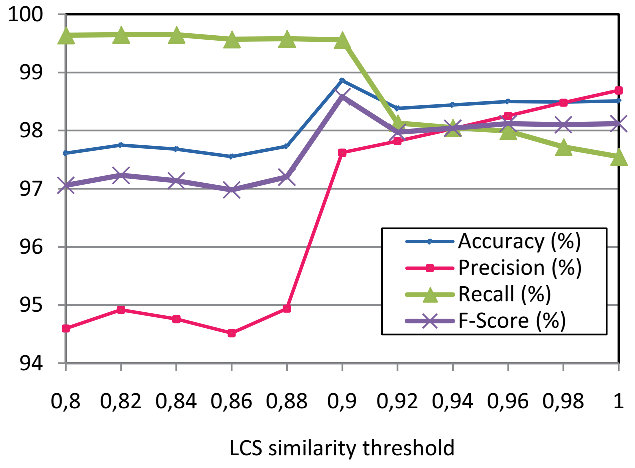

Fig. 23 Accuracy, precision, recall and f-score values for the experiment on the UNSW-NB15 dataset

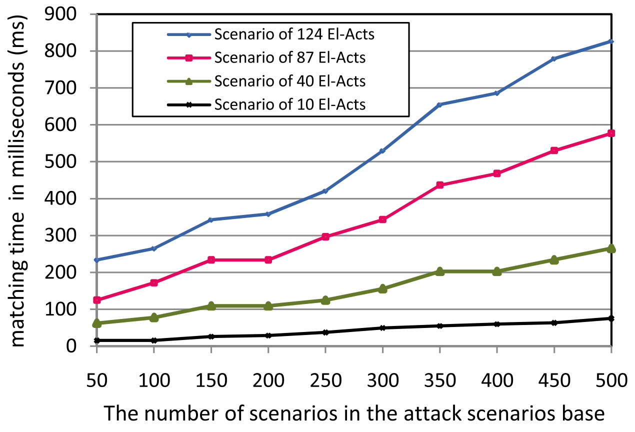

Fig. 24 The matching time for some scenarios of different sizes

four different scenarios consisting of 10, 40, 87 and 124 elementary actions respectively. In order to illustrate the effect of the size of the attack scenarios base on the matching time, we used ten attack scenario bases of different sizes (from 50 to 500). The time spent in the detection engine for each scenario over each attack scenarios base is shown in Fig. 24. Note here that the considered four scenarios do not belong to any of the ten attack scenarios bases. Thus, the computed time is the maximum that we can achieve for each scenario.

As Fig. 24 shows, the matching time for the four scenarios increases linearly with the number of scenarios in the attack scenarios base. We can also see that the increase of the matching time relies mainly on the number of elementary actions in the detected scenario. For example, for the scenario of 124 elementary actions, the matching time increases rapidly with the increase of the scenario base size, while for the scenario of 10 elementary actions, the matching time is almost the same whatever the size of the scenario base is. This high time consumed for matching the scenarios of large sizes is mainly due to the preprocessing task performed to generate the scenario elementary actions (more than 218 packets were preprocessed and concatenated to generate the 124 elementary actions of the first scenario) on the one hand, and to the LCS algorithm complexity which increases linearly with the increase of the scenario sizes on the other hand. Note that, in our experiment, we used the dynamic programming technique to implement the LCS algorithm, in which the running time for two sequences of n and m elements is $O(n \times m)$. Fortunately, practical experience shows that the number of scenarios of more than 50 elementary actions is very limited. For example, in our experiment over the DARPA'99 dataset, more than 97% of the generated scenarios are less than 50 elementary actions. Hence, among the four matching times of Fig. 24, those of 10 and 40 elementary actions are the most important for us. As shown in the figure, the matching time of these two scenarios is significantly low compared with those of 87 and 124 elementary actions.

In summary, with reference to the above results, we can conclude that our proposed intrusion detection system is very efficient and can detect many different attack types with a very high accuracy, precision, and with fewer false alarms, all within a reasonable detection time

7 Conclusion

In this paper, we presented a new effective, pragmatic and efficient intrusion detection system (IDS). Based on an automatic generation of attack scenarios, our IDS has the ability to detect new intrusion attacks. For the present work, we made three significant contributions. The first contribution is the proposition of a cooperative architecture between a honeypot system (the automatic attack scenarios generation part) and an intrusion detection system (intrusion detection part), which allows an automatic update of the attack scenarios database. The second contribution consists of generating an attack scenario for each attack type instead of a single contiguous string signature as is the case for many works focusing on the development of fully automatic IDSs. This approach provides more accurate forecasts than those provided by single signature approaches. The third contribution is the similarity preserving hash technique that we proposed to generate the signatures of each attack scenario. Based on this technique, we operate an effective grouping of atomic messages for signatures generation while preserving the similarity between messages. Furthermore, to detect intrusions, the longest common subsequence (LCS) algorithm have been used to calculate the similarity between the new captured traffic and the attack scenarios generated from the honeypot traffic. Based on LCS similarity, a threshold partial matching is computed instead of an exact matching. This allows the detection of attack variants. The experimental results demonstrate the efficiency of our proposed approach, which detects attacks with very high detection rates and low false positive rates.

References

1. Web Site of MIT Lincoln Laboratory: DARPA intrusion detection data sets. Available on: http://www.ll.mit.edu/ideval/data/. Accessed on March 2016.
2. Aho, A. V., Hirschberg, D. S., & Ullman, J. D. (1976). Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, *23*(1), 1–12.
3. Baker, B. S., & Giancarlo, R. (2002). Sparse dynamic programming for longest common subsequence from fragments. *Journal of Algorithms*, *42*(2), 231–254.
4. Elloumi, M. (1998). Comparison of strings belonging to the same family. *Information Sciences*, *111*(1–4), 49–63.
5. Griffin, K., Schneider, S., Hu, X., & Chiueh, T. (2009). Automatic generation of string signatures for malware detection. In: Recent advances in intrusion detection, 12th international symposium, RAID 2009, Saint-Malo, France, September 23–25, 2009. Proceedings (pp. 101–120).
6. Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *Journal of the ACM*, *24*(4), 664–675.
7. Hsu, W. J., & Du, M. W. (1984). Computing a longest common subsequence for a set of strings. *BIT Numerical Mathematics*, *24*(1), 45–59.
8. Jain, P., & Sardana, A. (2012). Defending against internet worms using honeyfarm. In: Proceedings of the CUBE international information technology conference, CUBE '12, Pune, India (pp. 795–800).
9. Kornblum, J. D. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, *3*(Supplement–1), 91–97.
10. Kreibich, C., & Crowcroft, J. (2004). Honeycomb: Creating intrusion detection signatures using honeypots. *Computer Communication Review*, *34*(1), 51–56.
11. Li, Z., Sanghi, M., Chen, Y., Kao, M., & Chavez, B. (2006). Hamsa: Fast signature generation for zero-day polymorphicworms with provable attack resilience. In: 2006 IEEE symposium on security and privacy (S&P 2006), May 21–24, 2006, Berkeley, CA, USA (pp. 32–47).
12. Mohammed, M. M. Z. E., & Chan, H. A. (2008). Fast automated signature generation for polymorphic worms using double-honeynet. In: 2008 Third international conference on broadband communications, information technology & biomedical applications, BroadCom 2008, November 23–26, 2008, Pretoria, Gauteng, South Africa (pp. 142–147).
13. Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military communications and information systems conference, MilCIS 2015, Canberra, Australia, November 10–12, 2015 (pp. 1–6).
14. Moustafa, N., & Slay, J. (2016). The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective*, *25*(1–3), 18–31.
15. Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In: Proceedings of the 13th conference on systems administration (LISA-99), Seattle, WA, November 7–12, 1999 (pp. 229–238).
16. Roussev, V. (2009). Hashing and data fingerprinting in digital forensics. *IEEE Security & Privacy*, *7*(2), 49–55.
17. Roussev, V. (2010). Data fingerprinting with similarity digests. In: Advances in digital forensics VI: Sixth IFIP WG 11.9 international conference on digital forensics, Hong Kong, China, January 4–6, 2010. Revised selected papers (pp. 207–226).
18. Snort. (2017). The open source network intrusion detection system. Available on: http://www.ll.mit.edu/ideval/data/. Accessed on July.
19. Spitzner, L. (2002). *Honeypots: Tracking hackers*. Boston, MA: Addison-Wesley Longman Publishing Co. Inc.
20. Tahan, G., Glezer, C., Elovici, Y., & Rokach, L. (2010). Auto-sign: An automatic signature generator for high-speed malware filtering devices. *Journal in Computer Virology*, *6*(2), 91–103.
21. Tang, Y., Xiao, B., & Lu, X. (2011). Signature tree generation for polymorphic worms. *IEEE Transactions on Computers*, *60*(4), 565–579.
22. Wang, Y., Xiang, Y., Zhou, W., & Yu, S. (2012). Generating regular expression signatures for network traffic classification in trusted network management. *Journal of Network and Computer Applications*, *35*(3), 992–1000.

Ammar Boulaiche received the Engineering degree in computer science from University of Jijel (Algeria) in 2005 and Magister degree in computer science (Networking and Distributed Systems option) from University of Bejaia (Algeria) in 2008. He is currently a Ph.D. student at University of Bejaia (Algeria). He works as an assistant teacher at the Department of computer science of University of Jijel (Algeria). His research interests include information security, intrusion detection and data mining and machine learning techniques.

Kamel Adi holds a Master degree in theoretical computer science from Pierre et Marie Curie (Paris VI) University and a Ph.D. degree in computer security from Laval University, Quebec, Canada. He is currently a full professor in the Department of Computer Science and Engineering at the University of Quebec in Outaouais, Canada. He is also the director of the Laboratory for Research in Computer Security. His research activities focus on the Development and Application of Formal Methods for Solving Problems related to Computer Security.