An Improved Distributed Intrusion Detection Architecture for Cloud Computing

Hamid Reza Ghorbani(✉) and Mahmoud Reza Hashemi

School of Electrical and Computer Engineering,
College of Engineering, University of Tehran, Tehran, Iran
{ghorbani.it, rhashemi}@ut.ac.ir

Abstract. In recent years, cloud computing has provided a framework for dynamic and saleable use of a wide range of services. Despite the advantages of cloud, security is still one of its most challenging issues. Intrusion detection systems, as a common security tool, can be used to increase the level of security in cloud environments. However, some of the inherent features of the cloud, such as being highly distributed, the variety and dynamism of its services, and difference security needs of each user or cloud service has made conventional IDSs inefficient for this environment. In this paper, an efficient architecture for intrusion detection has been proposed for cloud computing. For this purpose, we classify services, in terms of their security requirements, into groups of services with similar security constraints. This way the intrusion detection process can be customized according to the specific attacks that usually target the services of each group. The proposed architecture has been evaluated using Snort and by customizing it for each cloud service security requirement. Simulations indicate that the proposed architecture has been able to decrease the total time of traffic analysis against attacks by 17.5 % on average, while having the same detection rate and not losing the accuracy.

Keywords: Cloud computing · Intrusion detection system · Snort · Distributed intrusion detection

1 Introduction

With the constant growth of internet processing and communication requirements of individuals and organizations, cloud computing (CC) has emerged as an effective approach to address this demands by efficiently exploiting technologies such as virtualization, distributed computing and communication networks. CC provides a scalable infrastructure and can dynamically, effortlessly, and instantly accommodate to any user computing and software need with minimum cost. Due to its inherent advantages such as high performance, availability, mobility, variety of services at affordable prices, pay as you go, improved energy efficiency and ease of use, CC has significantly expanded its market. According to a study by IDC, the number of industries that rely on public cloud service platforms will increase tenfold by 2016 [1].

A CC that is based on the public internet is subject to trust, privacy, and other security concerns [2, 3]. Unlike some distributed environments, data storage and

A. Movaghar et al. (Eds.): CNDS 2013, CCIS 428, pp. 105–116, 2014.
DOI: 10.1007/978-3-319-10903-9_9

processing may happen in different locations in CC, and user usually is not concerned and cannot control it. Scalability, mobility and high diversity in user's needs are other challenges of CC. These characteristics make the cloud environment vulnerable in terms of availability, confidentiality and integrity [4].

Intrusion detection systems (IDSs) are one of the main components of any network security. Network intrusions are series of unauthorized actions that attempt to compromise the confidentiality, integrity or availability of the network and computing resources [8]. As businesses rely on CC increases for a majority of their services, the risks and damages of an intrusion increases as well, so does the importance of efficient security policy. As mentioned above, another inherent feature of CC is the large amount of data generated in this environment, generally due to the large number of users and high volume of transactions between them. This can challenge the performance and efficiency of conventional intrusion detection systems. Furthermore, the distributed nature of cloud and the variety of its users and services is target of a wide range of threats and intrusions. Finally, energy consumption is another concern of cloud providers. Optimizing computational complexity and as a result reducing power requirements by using an efficient and optimized intrusion detection system can help in this regard [5].

In this paper, we propose a distributed IDS architecture for CC (DIDSCC) based on multifarious service security requirements. The proposed architecture categorizes services according to their security aspects, and identifies the threats facing each service. As a result, it can customize the intrusion detection process according to the needs of each category which can lead to a more efficient processing cost. Although, in this paper we focus on signature based IDSs, especially in our simulations, but the proposed architecture can be extended to anomaly based IDSs with minimum effort.

The paper is organized as follows. Related works are introduced in Sect. 2. Section 3 introduces the proposed architecture. Simulation results are presented in Sect. 4. Finally, the paper concludes in Sect. 5 with the concluding remarks and future works.

2 Related Work

In recent years, many researches have addressed intrusion detection, and security in CC. Some of them have worked on improving the efficiency of the IDSs.

When users and their services are known, we can customize the intrusion detection operation accordingly. Some approaches have improved efficiency by categorizing user threats or IDS alerts in terms of their severity. The objective of this categorization is to choose the most appropriate action when an alarm is raised or an attack is detected [5, 9]. Lee et al. [5] have developed a multilevel system event log management and intrusion detection system for cloud computing environment. The authors believe that various risk levels for users based on an abnormal degree specified for each user can improve the overall system performance. This level is represented by a number between one to four, and is measured based on factors such as user's behavior while using a service or the effect of a potential attack performed by this user. A central unit that acts as a decision maker determines the appropriate level of security for each user. The authors have failed to provide any quantitative results, but it seems that the proposed method does not scale well.

Roschke et al. [9] proposed to categorize services into three layers of system, platform and application. Each of these layers has its own characteristics and security concerns. They also use a central control unit to receive information from all IDSs and detect possible correlations between alarms and prevent distributed attacks. This method suffers from a single point of failure.

Some papers have improved efficiency by focusing on specific types of attacks and have proposed application-driven IDSs [8, 11]. Lo et al. [8] have suggested a cooperative framework for network intrusion detection using agents. This article focuses only on Denial of Service attack (DoS). In this framework, several intrusion detection systems have been deployed in a distributed structure. These IDSs share their knowledge bank. The disadvantage of this method is that it has focused on just one particular category of attacks.

Sander et al. [11] introduced a novel type of DDoS attack referred to as Economic Denial of Sustainability (EDoS) in cloud services and introduced a framework for defense against this attack. Due to the importance of full time availability of CC services, authors believe that traditional DoS attacks can be transformed into an Economic DoS attack in CC. This paper is focused on just DoS, as well.

Improving the efficiency of IDS has always been a concern. Some papers in this category have improved conventional IDS efficiency by determining the most important features that can detect intrusion using feature reduction methods [12]. These methods try to reduce the number of attributes and computational complexity while not significantly affecting detection rate [7, 10]. Chebrolu et al. [7] extracted the most effective features using data mining techniques before the intrusion detection process. Although data mining based intrusion detection systems aim to address the issues of processing big data, but some of the mentioned aspects of CC has made these approaches to be less efficient in terms of processing time, performance, and accuracy [21].

To summarize, as it has been stated in [2, 3, 6], most existing methods have either failed to consider the specific requirements of CC in terms of security, or have neglected the diversity in user security requirements. In this paper, a distributed IDS architecture for CC has been proposed that has improved IDS efficiency while maintaining accuracy. The proposed DIDSCC architecture is explained in more details in the next section.

3 Proposed Architecture

In this section, we present the proposed architecture. First, we provide the observations that have led to the new architecture, and then we will describe each of its components.

3.1 Observations

As mentioned before, our first observation indicates that CC environments is faced with a wide range of users, each requesting a different service, and consequently having different security requirements. These requirements not only vary from one user to the next for the same service, but also from one service to another. Applying the same security policy for all users, services and attacks may not be the most efficient approach.

In this paper, our focus is on signature based IDSs. These IDSs analyze each incoming packet against predefined rules. Clearly, some of the rules may be irrelevant to user, service, or attack. In a CC environment, where the number of users, network traffic, and number of services increases, efficient resource utilization is a major concern [13]. Efficient resource utilization, such as processing power, CPU utilization, and memory consumption has a direct impact on detection time and indirectly effect on detection rate [14].

As mentioned in [16], different services have different security concerns and priorities. For each concern, a specific set of attacks has been identified. By properly categorizing services and determining their corresponding set of attacks and rules, we can customize the IDS for each CC service category. Although a service might be more susceptible to a certain attack, and an attack may have more destructive effect on single kind of service, but it still should be protected from the remaining ones with a lower priority or importance. In other words, in a signature based IDS we need to check the rules corresponding to the former attacks first and address the lower priority rules next.

The second observation indicates that a simple but widely-applicable security model is the CIA triad; standing for Confidentiality, Integrity and Availability. Confidentiality refers to limiting information access and disclosure to authorized users, Integrity refers to preventing unauthorized modification of data and systems, and Availability refers, to the availability of information resources. Almost all network attacks can be traced to one or several of CIA elements [17, 18]. Hence in this paper we only consider these three security concerns.

In the next section we use the these two observations and present a new distributed IDS architecture to address them.

3.2 Proposed Approach

In this paper, an improved distributed IDS architecture has been proposed. In this architecture the cloud is segmented into three server groups. According to the second observation of the previous section, these three groups correspond to Confidentiality, Integrity, and Availability attacks, respectively.

For each cloud customer, according to its requested service and the attacks that this service is subject to, it will be hosted in one of these three server groups. The relation between services and attacks can be determined and announced by the CC provider or be left to the customer to select.

For example, consider a customer who wants to host its Personal Health Record service on a cloud provider. On a high level view, this service consists of two main component; namely, a portal for the introduction and delivery of services and a data storage system for patients' information. The main concern of the former is availability (e.g. being robust against DoS attacks), while the storage component should guarantee data integrity by protecting customer information and securing access to them. For some of the information, confidentiality is critical. In our scheme, the cloud provider will host each of these components in a different server group.

Each server group has its own IDS which have been customized according to the attacks that are determined in the CC policy for each of the confidentiality, integrity, and availability attacks, respectively. Although each IDS is customized for one set of attack, in order to be safe against the remaining attacks, their rules can be included selectively but with a lower priority. The rules used in each IDS can be customized based on the functionality of that IDS. The objective of this customization is to improve its performance and efficiency in resource utilization. For each customized IDS we shall improve the quality of its rules based on a functional knowledge of the IDS (and threats against it), and by tuning its rules accordingly. For example, rules that are not properly tuned can cause an IDS to waste time by redundantly checking them due to an improper sequence of rule checking. Using tuning approach, when designing IDS rules, leads to a significant improvement in performance when checking real-time traffic. The high level view of the proposed architecture has been illustrated in Fig. 1.

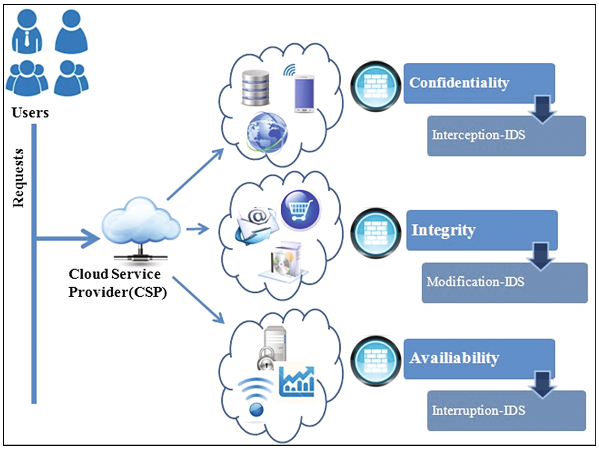
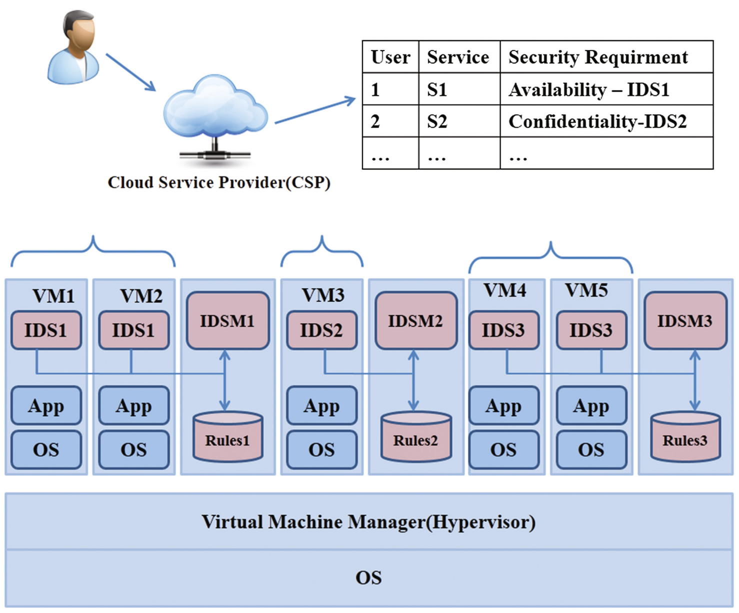


Fig. 1. An overview of the proposed IDS architecture

In the proposed architecture for each server in a server group, one instance of the customized IDS is activated on each virtual machine (VM) that corresponds to a customer cloud service. In addition, each server has an intrusion detection system manager (IDSM). The main duty of this component is to provide flexibility. For instance it can upgrade the rule set of all the IDSs on this server, it can coordinate the alerts among them to address distributed attacks to the server, it can also coordinate with the other IDSMs on other servers in the same group or servers in other groups. The detailed view of the proposed architecture is illustrated in Fig. 2.



User	Service	Security Requirment
1	S1	Availability – IDS1
2	S2	Confidentiality-IDS2
…	…	…

Fig. 2. Detailed architecture of a cloud service provider

4 Evaluation Results

In order to evaluate the proposed architecture, in this paper we have used Snort [14] as our IDS. By customizing it for each security requirement and using the customized version on each IDS, we have compared our architecture to a conventional non customized IDS structure.

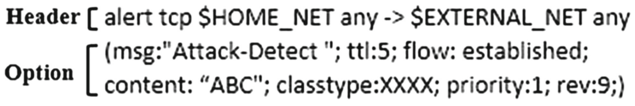
In Subsect. 4.1 we briefly introduce Snort. In Subsect. 4.2 we introduce the used dataset and the corresponding platform configurations. Finally, we provide the simulations results in Subsect. 4.3.

4.1 Background

Snort. Snort is an open source signature based sniffer and network intrusion detections system (NIDS) utilizing a rule-driven language. It captures packets and checks their content with its predefined set of patterns and signatures. The detection engine of Snort allows registering, alerting and responding to any known attack. Snort utilizes descriptive rules to determine what traffic it should monitor and a modular detection engine to pinpoint real time attacks. When an attack is identified, Snort can take a variety of actions to alert the system administrator. Each rule has parameters that can be tuned. We use this capability to customize and configure it in order to fit our architecture.

Snort Rules. Each rule in Snort indicates intent, impact and target of an attack. A rule is what tells the engine where and what to look for in network traffic and what needs to be done if an attack has been detected. A rule in Snort consists of a header and a body (option). The header defines the traffic source. Rule headers consist of six elements: rule action, protocol type, source IP address, source port.

A rule body has over thirty optional parameters. For any Snort rule to trigger an event, all its optional rule parameters must match. It is basically a message to Snort to inspect the packets for the values that we specified in this section and determine whether it is malicious or not. This option allows rules to be classified by type of attack, and have a default priority value associated with that type. By using the two rule section values, Snort makes a rule tree from the loaded rule database, and processes each receiving packet in this tree based on packet parameter values. Figure 3 presents an example of a Snort rule.



```
Header [ alert tcp $HOME_NET any -> $EXTERNAL_NET any
       [ (msg:"Attack-Detect "; ttl:5; flow: established;
Option [ content: "ABC"; classtype:XXXX; priority:1; rev:9;)
```

Fig. 3. A sample Snort rule

This rule means that if there is an established TCP stream from HOME_NET to EXTERNAL_NET that contains the string "ABC" in its payload with a ttl[1] value equal to "5", Snort should generate an alert and log a file entitled "Attack-Detect". Rules can be labeled with classifications and priority numbers to be used for grouping and distinguishing them. The rule classification, specified by the "*classtype*" keyword, determines the type of malicious traffic that the rule is designed to detect. The Priority field has two meanings: precedency of the rule in processing and analyzing traffics (rules with higher precedence or lower priority value are processed earlier) and specifies a threshold for IDS alarm generation. It should be noted that Snort has set the default priority value for each type of attack according to its destructive effect. These classifications and priorities can adjusted and we are going to use this feature in our approach for improving the quality of rules for each service type.

Snort Detection Engine. The detection engine is the most important part of Snort. Its responsibility is to detect if any intrusion activity exists in a packet using Snort rules. The computational load or speed of the detection engine depends on parameters such as number of rules, network load and the processing power of the machine that is executing Snort. One important property of Snort is its ability to search a data pattern inside a packet, with "*content*" keyword. For rules that have this property, *content* string pattern matching is one the most computationally expensive Snort processes. Hence, *content* keywords should be selectively and carefully used in IDS rules [20].

[1] Time to live.

The signature matching used in Snort IDS rule tree consists of two main parts: (i) packet classification, which involves examining the values of packet header fields, and (ii) deep packet inspection, in which the packet option is matched against a set of predefined patterns. The combination of header and payload search determines whether a packet is an intrusion attempt or not. According to [19], these two are the most computationally consuming parts of Snort.

Through improving IDS rules quality, by employing methods likes tuning classifications and priorities of rules, we can distinguish between high and low-risk alerts. These features are very useful when we want to differentiate high-risk alerts or want to pay attention to them first. In other words, a parameterized search will use a customized search order based on each parameter's relative ability to terminate a useless search quickly or limit the search space efficiently.

4.2 Implementation

In order to evaluate the DIDSCC architecture we used Snort (version 2.9.4.6) installed on CentOS Linux platform and used the DARPA99 dataset [15]. The DARPA dataset has five weeks of network traffic, with a list of when and where the attacks have occurred. Weeks 1–3 are for training, weeks 4–5 are for testing. The attacks in this dataset can be categorized into Probe, Denial of Service (DoS), Remote to Local (R2L) and User to Remote (U2R) attacks. In this simulation we consider R2L to exploit Confidentiality, U2R and Probe to exploit Integrity and DoS to exploit Availability.

We considered two scenarios for the evaluations and for each scenario we have three users that were assigned to three VMs and three IDSs that run in parallel. In the first scenario, based on the CIA security requirements the Snort rules have been categorized and then customized. The selected rules are presented in Table 1. In the second scenario, in all of the virtual machines the same Snort rules are deployed. In other words, rule sets and other Snort configurations are the same in all IDSs in this latter scenario.

The rule customization that is used to improve rule quality in the first scenario consists of the following two parts: (i) in each IDS group, their selected rules have the maximum priority and other rules have the normal priority. (ii) as mentioned before, Snort arranges rules using the header field and according to the *"content*". For rules with *content* a pattern matching algorithm is used to select rules according to the *content* keyword. The optional *fast-pattern* property of the *content* keyword can be used to speed up the *content* string matching process by guiding Snort detection engine to avoid unnecessary matches. If a packet does not match the *content* keyword that has the *fast-pattern* property, we know that at least one of the rule options will not match, and an alert will never be generated. Hence, additional searches for the remaining parameters are no longer necessary and can be avoided, resulting in an improved performance [14].

Each VM has a 2.2 GHz CPU, 2 GB RAM and 50 GB Hard Disk. The virtual machines run on a Xen hypervisor. All evaluations for both scenarios are performed with the same settings and conditions.

Table 1. Selected Snort rules

IDS type	Snort rule files
Availability	dos.rules, ddos.rules, dns.rules, icmp.rules, icmp-info.rules, local.rules, netbios.rules, other-ids.rules, snmp.rules, web-cgi.rules, web-client.rules, web-iis.rules, web-misc.rules, bad-traffic.rules, attack-responses.rules, experimental.rules
Confidentiality	backdoor.rules, exploit.rules, finger.rules,ftp.rules, tftp.rules, icmp.rules, misc.rules,imap.rules, local.rules,netbios.rules, other-ids.rules, pop3.rules, rservices.rules, smtp.rules, snmp.rules, telnet.rules, web-iis.rules, web-misc.rules, web-php.rules, x11.rules,, attack-responses.rules, experimental.rules
Integrity	bad-traffic.rules, exploit.rules, misc.rules, info.rules, local.rules, mysql.rules, nntp.rules.oracle.rules, sql.rules, other-ids.rules,rpc.rules, scan,rules, shellcode.rules, web-attacks.rules, web-cgi.rules, web-coldfusion.rules, web-frontpage.rules, web-iis.rules, web-misc.rules, web-php.rules, attack-responses.rules, experimental.rules

4.3 Simulation Results

Our performance criteria consist of processing time and accuracy rate. Processing time is the required time for Snort to detect intrusions based on its preconfigured rules. This time is reported by Snort. Accuracy rate is the total percentage of detected intrusions per category. We have tested the two mentioned scenarios ten times and reported the average results in Tables 2 and 3.

Table 2. Processing time

Scenario/Time	t(s)
Scenario1- IDS 1 (Confidentiality)	228
Scenario1- IDS 2 (Integrity)	219
Scenario1- IDS 3 (Availability)	207
Scenario 2	264

Table 3. Detection rate

Scenario/Requirement	C	I	A
Scenario1- IDS 1 (Confidentiality)	84 %	56 %	82 %
Scenario1- IDS 2 (Integrity)	65 %	72 %	79 %
Scenario1- IDS 3 (Availability)	77 %	60 %	90 %
Scenario 2	88 %	76 %	94 %

As we can see above, in the first scenario with customized IDSs, the processing time has decreased by 17.5 % on average in comparison to the second scenario. This is an indication of a more efficient use of processing resources. Clearly, this customization enables our cloud provider to handle more traffic loads with the same resources.

Table 4 compares several general aspects of our proposed architecture and other approaches. In this table, "Optimized processing" property refers to performing intrusion detection processing efficiently and based on each service or security needs. "Adaptation to any customer requirement" refers to the ability to create an adjustable mapping between each service and its corresponding security requirement. "The Overall protection" implies that in addition to the ability to focus on high-priority security needs in each situation, other security requirements are also considered. "Parallelism" refers to the ability to perform the detection process in parallel. "The structure of IDS" represents how the system works in distributed or centralized mode (Fig. 2). In distributed mode, IDSs perform the intrusion detection operation individually while in other modes only one central IDS carries out the intrusion detection operation. "Scalability" means the IDS can scale as network traffic increases.

Table 4. Comparison of DIDSCC with few existing IDSs for CC

Property/System	DIDSCC	[7]	[8]	[9]
Optimized processing	✓			✓
Adaptation to any customer requirement	✓			✓
Overall Protection	✓	✓		✓
Parallelism	✓		✓	
Distributed(D)/Centralized(C) Intrusion detection	C + D	C	C + D	C
Scalability	✓		✓	

5 Conclusion and Future Work

Cloud computing has provided a framework for dynamic and saleable use of a wide range of services that can provide an infrastructure, a platform or software. Despite the advantages of cloud computing, security is a major challenge for cloud computing. In this paper, an efficient architecture for intrusion detection has been proposed for CC. The proposed distributed intrusion detection system for cloud computing (DIDSCC) has been evaluated using Snort and by optimally customizing it for each cloud service security requirement. Results from simulation, using Snort as our IDS, show that the proposed architecture is able to reduce processing time by 17.5 % on average. Also this customization enables our cloud provider to handle more traffic loads with the same resources.

In future work, we would try to configure IDSs dynamically according to the needs of services. Due to features of CC environment and diversity in service and security concerns, we can generate and configure rule and signatures based on user/service behavior or requirements dynamically. Also we can determine classes of services other than CIA model or classify security rules based on other criteria. Communication model between IDSMs of different groups and alert/message aggregation of these created models in each or between group can be extended. Also checking extra important factors such as scalability and other performance parameters like false alarm rate can be the next step for improving the proposed architecture.

References

1. TOP 10 PREDICTIONS, IDC Predictions 2013: Competing on the 3rd Platform. http://www.idc.com/research/Predictions13/downloadable/238044.pdf
2. Tanzim Khorshed, Md., Shawkat Ali, A.B.M., Wasimi, S.A.: A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. Future Gener. Comput. Syst. **28**(6), 833–851 (2012)
3. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2011)
4. Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Gener. Comput. Syst. **28**(3), 583–592 (2012)
5. Lee, J.-H., Park, M.-W., Chung, T.-M.: Multi-level intrusion detection system and log management in cloud computing. In: 13th Interntional Conference on Advanced Communication Technology (ICACT), Seoul, pp. 552–555 (2011)
6. Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., Rajarajan, M.: A survey of intrusion detection techniques in Cloud. J. Netw. Comput. Appl. **36**(1), 42–57 (2013)
7. Chebrolu, S., Abraham, A., Thomas, J.P.: Feature deduction and ensemble design of intrusion detection systems. Comput. Secur. **24**(4), 295–300 (2005)
8. Lo, C.-C., Huang, C.-C., Ku, J.:A cooperative intrusion detection system framework for cloud computing networks. In: 39th International Conference on Parallel Processing Workshops (ICPPW), San Diego, vol. 39, pp. 280–284 (2010)
9. Roschke, S., Cheng, F., Meinel, C.: Intrusion detection in the cloud. In: 8th IEEE International Conference on Dependable, Autonomic and Secure Computing, Chengdu, pp. 729–735 (2009)
10. Tsamardinos, I., Aliferis, C.F., Statnikov, A.: Time and sample efficient discovery of Markov blankets and direct causal relations. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 673–678 (2003)
11. Sander, V., Shenai, S.: Economic denial of sustainability (edos) in cloud services using http and xml based ddos attacks. Int. J. Comput. Appl. **41**(20), 11–16 (2012)
12. Nguyen, H.H., Harbi, N., Darmont, J.: An efficient local region and clustering-based ensemble system for intrusion detection. In: 15th Symposium on International Database Engineering & Applications, pp. 185–191 (2011)
13. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**, 50–58 (2010)
14. Snort-Homepage. https://www.snort.org/
15. Darpa 99 Intrusion detection data set. http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1999data.html
16. National Institute of Standards and technology (NIST), Computer Security Devision, Special Publications Series (800 Series). http://csrc.nist.gov/publications/PubsSPs.html

17. Stoneburner, G.: Underlying Technical Models for Information Technology Security. Technical Report. NIST SP 800-33, United States (2001)
18. Greene, S.: Security Policies and Procedures: Principles and Practice. Prentice-Hall Inc., Upper Saddle River (2005)
19. Fisk, M., Varghese, G.: Fast Content-Based Packet Handling for Intrusion Detection. Technical report, University of California at San Diego (2001)
20. Yoshioka, A., Shaikot, S.H., Kim, M.S.: Rule hashing for efficient packet classification in network intrusion detection. In: 17th International Conference on Computer Communications and Networks (ICCCN), US Virgin Island, pp.1–6 (2008)
21. Meenakshi, R.M., Saravanan, E.: A data mining analysis and approach with intrusion detection/prevention with real traffic. In: IJCA Proceedings on EGovernance and Cloud Computing Services, EGOV(4), pp. 13–17 (2012)