

# Enhancing Collaborative Intrusion Detection Methods Using a Kademlia Overlay Network

Zoltán Czirkos and Gábor Hosszú PhD

Department of Electron Devices, Budapest University of Technology and Economics,  
Magyar tudósok körútja 2, Building Q, section B, 3rd floor  
Budapest, H-1117, Hungary  
czirkos, hosszu@eet.bme.hu

**Abstract.** The two important problems of collaborative intrusion detection are aggregation and correlation of intrusion events. The enormous amount of data generated by detection probes requires significant network and computational capacity to be processed. In this article we show that a distributed hash table based approach can reduce both network and computational load of intrusion detection, while providing almost the same accuracy of detection as centralized solutions. The efficiency of data storage can be improved by selecting Kademlia as the underlying overlay network topology, as its routing can easily adapt to the dynamic properties of such an application.

**Keywords:** peer-to-peer; intrusion detection system; collaborative intrusion detection; attack correlation; attack aggregation

## 1 Introduction

In the early days of the Internet, all communication was built upon trust among users. However, with e-commerce emerging and the number of hosts connected to the network increasing to tens and hundreds of millions, serious security concerns came into prominence: sensitive information stored on-line makes hosts a target for a wide range of attacks.

Attacks, being both manually and automatically controlled, have become ever more sophisticated, originating from multiple entities or targetting multiple hosts at the same time. Various worm programs replicate themselves to spread malicious code to vulnerable systems, or scan network nodes to find vulnerabilities. Others compromise hosts of home users to build botnets, to deliver spam e-mail to addresses collected from the web or other infected users' mail applications or to carry out distributed denial of service attacks. These are usually referred to as large-scale coordinated attacks.

Sophisticated attacks are generally problematic to detect as the pieces of evidence are spread across multiple hosts. To recognize such attacks, one has to *aggregate* (collect) the evidence as well as *correlate* (analyze) the pieces collected. In other words, an intrusion detection system must process the evidence from multiple detector *probes* located at different hosts, maybe even on different subnetworks. This poses several problems to solve:

- heterogenous format of probes' output,
- enormous quantities of – often useless – detection data,
- yet inadequate data for decision making,
- communication problem between probes and correlators.

In this paper we present a distributed intrusion detection system, which organizes its participants to a *peer-to-peer* (P2P) based *distributed hash table* (DHT) using the Kademlia topology [5]. This *overlay network* is used to store intrusion detection data with balancing the load of storage of intrusion data and correlation of events amongst its nodes. The network traffic generated by the system is considered, and the implications of using different DHT topologies are discussed.

The rest of this paper is organized as follows. In Section 2, we first review existing research of distributed intrusion detection systems. We present the architecture of our distributed intrusion detection system based on the Kademlia DHT in Section 3. The results of the intrusion detection method and statistics of detection are highlighted in Section 4. Research is concluded in Section 5.

## 2 Related Work

Attackers, having different goals, use various approaches for intrusion of computer network systems. These leave different tracks and evidences, called the *manifestation of attacks* [12], and require different methods to detect. In the following, the terms below are used for discussion [6]:

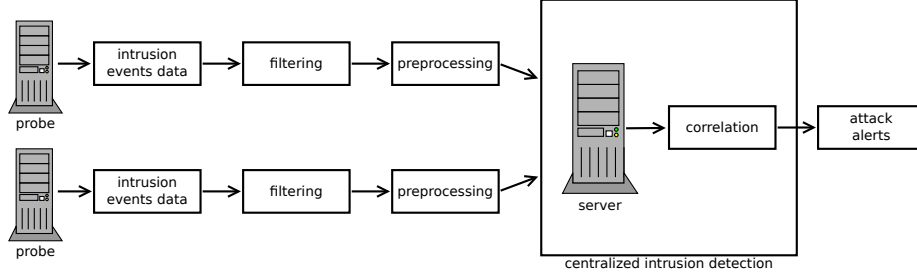
**Attacks** Real intrusion attempts, which are used to gain access to a host or disturb its correct functioning.

**Events** Primary intrusion detection data generated by probes. These events are not necessarily attacks by themselves, but can be part of a complex attack scenario.

Several types of large-scale attacks can only be detected by *collecting* and *correlating* events from a number of detector probes. The collection of evidence has to be extended to primary events as well. To achieve this, various collaborative intrusion detection systems (CIDS) have been proposed, for which an extensive overview can be found in [24].

### 2.1 Centralized Approaches

The earliest collaborative detection systems were centralized. The advantage of centralization is that a single server receives and processes all data that can be gathered, e.g. it has all the information necessary to recognize the attack. However, the approach has two disadvantages: scalability and having a single point of failure (SPOF). The high amounts of data to be collected and correlated cannot be handled for large networks by a single server. Moreover, this correlation server is a possible target for shutting down the entire intrusion detection system.



**Fig. 1.** Intrusion detection with centralized collection and correlation of data from various probes. Every piece of information is sent to a server which handles the correlation.

The Internet Storm Center *DShield* project collects firewall and intrusion detection logs, either automatically or by manual upload from users. The log files are then analyzed centrally. The *NSTAT* system [11] is more advanced in the sense that it is automatic and real-time. In this system the detection data is filtered and preprocessed before being sent to a central server for correlation, as seen on Figure 1. The order of events are then analyzed using using a state transition mechanism with predefined scenarios.

Correlation can be carried out by using various methods. SPICE [20] and CIDS [23] group events by their common attributes. The LAMBDA [3] system aims to fit detected events, which are described using a common language, into pre-defined and known scenarios. The JIGSAW system [19] maps prerequisites and consequences of events in order to find out their purposes.

## 2.2 Hierarchical Collaborative Intrusion Detection

The *DOMINO* system can be used to detect worm activity. It is built on an unstructured P2P network with participants grouped into three levels of hierarchy [22]. The nodes on the lowest level generate statistics only hourly or daily, so they induce small network traffic.

The *PROMIS* protection system (and its predecessor, Netbiotic) uses the JXTA framework to build a partially decentralized overlay network to share intrusion detection data [21]. The nodes of this system generate information for other participants about the number and frequency of detected events. This information is used to fine-tune the security settings of the operating system and the web browser of the participants. While this method creates some level of general protection against worms, it also decreases the usability of the systems.

The *Indra* system is built on the assumption that attackers will try to compromise several hosts by exploiting the same vulnerability [9]. If any of these attempts get detected by any participant of the Indra network, others can be alerted. Participants of this system can therefore enhance their protection against known attackers, rather than developing general protection.

### 2.3 Intrusion Detection Based on P2P Networks

The scalability and SPOF problems of centralized solutions can be solved by using structured P2P application level networks [2]. These enable one to reduce network load compared to the hierarchical networks presented above.

The CIDS system [23] uses the Chord overlay network [17] for implementing a publish-subscribe style application. Nodes of this system store the attacker IP addresses in a list, and they subscribe in the network for events of these addresses. If the number of subscribers to a given IP address reaches a predefined threshold, they are alerted of the possible danger. The Chord network ensures that the messages will be evenly distributed among the participants. However, the instability of Chord under churn [14], and IP address blacklist being the only correlation method used in CIDS does not allow its wide deployment.

The *BotSpot* system [13] uses data in NetFlow format to discover traffic patterns generated by botnets [8]. By dropping specific IP addresses from the data analyzed, anonymity can be ensured for its users. *BotSpot* focuses on botnet traffic only, and it cannot be used to detect other types of attack manifestations.

## 3 The Komondor System Architecture

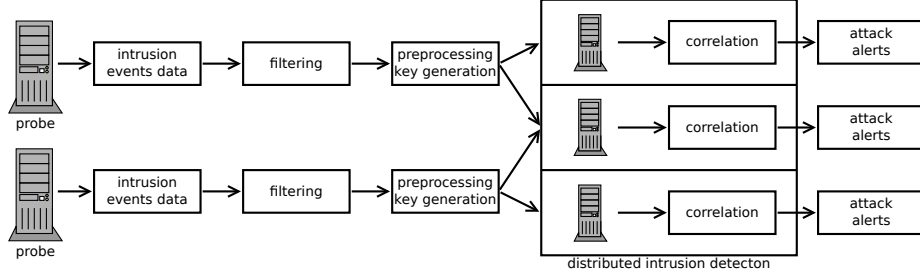
In this section we present our novel intrusion detection system named *Komondor*. The most important notion of this system is that it uses the DHT named Kademlia [15] to store intrusion data and to disseminate information about detected intrusions. The purpose of the overlay is the efficient detection and quick dissemination of alerts: when the analyzing of the collected events indicates the fact of an attack, the Komondor nodes start an alert procedure notifying other nodes of the possible danger.

### 3.1 Distributing Load Among Multiple Correlation Units

The Komondor application level network consists of multiple nodes. All nodes have the responsibility to *collect* and *correlate* intrusion data and to *report attacks* discovered to other nodes of the network. This means that all participants of the Komondor network can serve as intrusion detection units and correlation units as well. When replacing the central correlation server with a number of correlation units, care must be taken to ensure that:

- pieces of intrusion evidence which are correlated must be sent to the same correlation unit, so that it can gather all the information about the attack, and that
- pieces which are part of different ongoing attacks should preferably be sent to different correlation units.

The latter balances load among the units and improves the overall reliability of the system. Komondor achieves the above by *assigning keys to preprocessed intrusion data*, as seen on Figure 2. Keys are used as storage keys in the DHT.



**Fig. 2.** Distributed collection and distributed correlation of intrusion evidence from various probes. The Komondor system assigns keys to pieces of evidence so that data can be stored efficiently in a distributed hash table. By using these keys, computational load of correlating can be distributed among several units.

For different attackers or attack scenarios, a different key is generated, and data is therefore stored at different nodes. Pieces of evidence which might be correlated to each other must be assigned the same key, and by that sent to the same Komondor node for correlation. Note that these pieces are not necessarily detected at the same probe.

The Komondor system has a *middle layer* inserted into the intrusion detection data path shown on Figure 1. The nodes of the DHT act as correlation units, which can implement the same correlation methods as their centralized versions. However, correlation starts as soon as the preprocessing stage, where events are tagged with a key in conformance of the correlation method used, and the key is used for sending the attack report to the correct correlation unit.

The structured overlay therefore has the advantages of distributed and centralized detection systems as well. Event data collected is generally sent to a single collector node only (this would not be possible with an unstructured overlay, as those cannot have global rule to map a key to a node.) Moreover, when Komondor nodes are under multiple but independent attacks, the network and computational load of both aggregation and correlation is distributed among nodes. The system has no single point of failure, as the overlay will reconfigure itself when a node quits or becomes unreachable. The structured overlay can also be used to disseminate other type of information as well, for example the attack alerts which enable node create protection.

### 3.2 Selection of Keys in the Komondor System

The accuracy of detection, also network and computational load balancing depends on the proper selection of keys. If the correct key is failed to be selected at preprocessing stage, pieces of correlated evidence may end up at different correlation units and therefore the attack may remain unnoticed. Detection efficiency can be increased by assigning more keys, and sending evidence to multiple correlation units, should an event be suspected to be a candidate for being part

of different attacks or attack scenarios. However, every subsequent key assigned increases network load as well. To select the correct keys, the inner working of correlation methods has to be known.

For every attack method and scenario, a different key selection mechanisms are feasible. Consider the network scan types, as categorized in [22]:

**Horizontal port scan** Different hosts are scanned by a attackers, but the port number, e.g. the vulnerability searched for, is the same. In this case, a black-list of attackers can be built using the collection and correlation of detected attempts. The key for collection of events should be the identifier of the vulnerability.

**Vertical scan** A single host is under attack. The attack originates from a single host, too. If this is the case, the attacker is known and hosts can protect themselves against it, should it try to attack another host to be protected. The key is the IP address of the attacker.

**Mixed mode scan** Multiple attackers use their network capacity to launch an attack against a single host or a subnetwork. This is the usual scenario for the well known DDoS (distributed denial of service) attacks. The key for the evidence storage in the DHT in this case is the subnetwork address attacked. By analyzing the data collected in this scenario, hosts can automatically detect the fact of the network scale attack, e.g. they can discover that the problem is not only related to a single host but a complete subnetwork or organization.

### 3.3 Kademlia as the DHT Topology of Komondor

The nodes of Komondor create a Kademlia based DHT overlay network. This is the topology, which can adapt its routing tables to the dynamic properties of traffic generated by the intrusion detection module.

Event data stored in the overlay can generate significant overlay traffic that will load not only detector and collector nodes, but other nodes along the path from the former to the latter one as well. However, if the events are in correlation with the same attack, the key is likely to be the same. The distribution of keys in store messages is therefore highly uneven. By using Kademlia, network traffic can be significantly reduced in this scenario. The reason for this is that any arbitrarily selected node can be inserted to the routing tables of any other Kademlia node while still conforming the rules of the Kademlia protocol. Routing tables of other DHT overlays like CAN or Chord are much more rigid, and therefore the routing algorithm of those cannot optimize store requests with the same key sent in a short time.

To compare the overlay messages generated by intrusion event aggregation, refer to Table 1. In Chord, messages are forwarded by helper nodes in the overlay along the path from the source to the destination of the message. On the other hand, in Kademlia the storing of a  $\langle key; value \rangle$  pair by a node is started by first looking up the IP address of the destination node by successively querying nodes closer to the destination. After finding out its address, data is sent directly from

**Table 1.** Number of messages in structured overlays for intrusion detection

Overlay	Chord	Kademlia
Node lookup	0	$O(\log_2 N)$
First event stored	$O(\log_2 N)$	$O(1 + \log_2 N)$
$n$ events with the same key	$O(n \cdot \log_2 N)$	$O(n + \log_2 N)$
Number of messages per event	$O(n \cdot \log_2 N)/n$	$O(n + \log_2 N)/n$
Number of messages with $n \rightarrow \infty$	$O(\log_2 N)$	$O(1)$

the source and the destination. This also implies that the  $\langle key; value \rangle$  pair to be stored (e.g. the payload of the message) is contained in every message of Chord, and only in the last message of Kademlia.

To store a detection event in Chord, the number of messages generated in the overlay would be in the order of  $\log_2 N$ , where  $N$  is the size of the overlay. For Kademlia, the looking up of the address of the destination also takes  $\log_2 N$  messages. The payload requires one more message (+1). However, if multiple events have to be stored which are detected by the same probe, the lookup procedure can be optimized away, as the key and therefore the collector node is the same, too. For sending data of  $n$  events, the number of messages generated is only  $n + \log_2 N$  for Kademlia and  $n \cdot \log_2 N$  for Chord, which is worse at the factor of  $n$  for the latter one. The limit of messages per event will drop to 1 for Kademlia in this scenario.

### 3.4 Overlay Services Used by Komondor and the Effect of Node Churn

The Komondor system uses no data lookup in the DHT as other applications do, only the data store mechanism is used. Stored events are never looked up, rather the node storing them has to process incoming events to recognize attackers. The collector nodes have the responsibility to start a broadcast if an attack is recognized. The topology of the overlay can be used to send the broadcast message to all nodes in a time frame that is logarithmically proportional to the size of the overlay network [4].

As with other P2P systems, node churn can degrade the performance of the application: detection accuracy might decrease in the event of a collector node disappearing from the overlay network. However, considering the typical key patterns generated by attacks and used in the intrusion detection, the effect of churn can be minimized for the case when a node deliberately and gracefully quits the overlay network, or when a new node appears in the system.

A node intending to quit might store intrusion detection data, which must be transferred to its neighbors before leaving the overlay. As the unhashed keys

of the events are known by this node, it can recalculate the distance of the keys to other known nodes in the NodeID address space, and remap them to other nodes in the system, as described in [10].

The keys remapped might originate from different nodes. Those nodes, having looked up the IP address to the node currently quitting, have cached that information for the lookup request optimization. As the node currently quitting is the closest node to those keys, the source nodes of these events must be notified, in order to inform them that the IP address of the quitting node is to be cleared from the cache. A new lookup request is to be started by them before sending any further data.

The same procedure can be applied to newly joined nodes. As described in [15], new nodes are put into the routing tables of their closest neighbors. These must recalculate the distance of their stored events' keys to the NodeID of the new node. If any of those are closer to the new one, the key must be remapped, and the source node of those events must be notified to send data to the newly joined node rather than the one previously cached for that key.

## 4 Results and Discussion

In this section we present statistics of intrusion attempts detected using the implemented Komondor system. The statistics are evaluated to show, which types of attacks this system can be used to detect.

### 4.1 Network and Computational Load Balancing

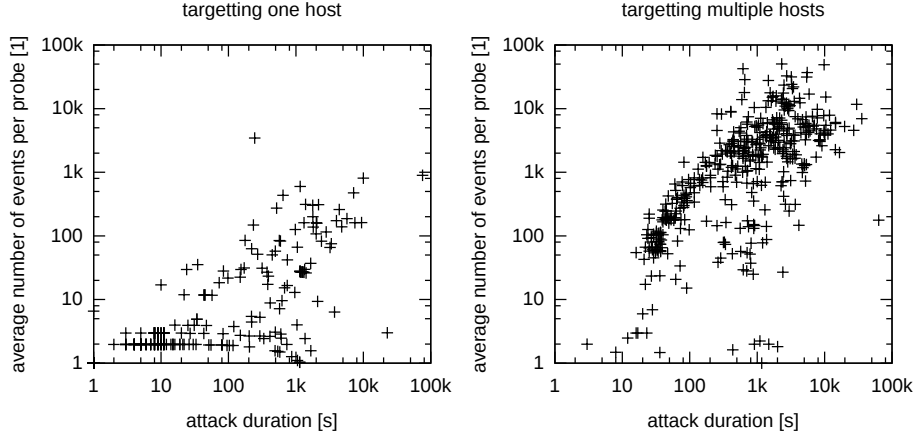
The Komondor reference implementation used a similarity based correlation method, using the *source IP address* of offending packages as a key. The primary events were detected by three probes on participating hosts:

- the open-source *Snort* intrusion detection system [1],
- a built-in module of Komondor which examined the operating system log files,
- the packet log of the firewalls of the systems.

We selected common event types from the Snort database and also tagged events with a severity level in the interval  $[0; 10]$ . Intrusion alert was triggered when the sum of scores reached the predefined threshold level 10. (An attack of severity level 10 immediately triggers a response alert.) Receiving the alert instructed the participant nodes to automatically block the attacker through their own *firewalls*, which were configured to *log dropped packets* as well. This method enabled us to determine the efficiency and reliability for known attack types presented here, and also to estimate the ratio of attacks, which were continued by the attackers even after the Komondor system having noticed their activity.

Figure 3 shows invalid passwords detected for SSH login attempts on various hosts. Every dot on the graph is an individual attack, e.g. it represents one or more event that originated from a single IP address. The  $y$  axis shows the





**Fig. 3.** Number of invalid password events detected for various attacks ( $y$  axis) plotted by the duration of the attack ( $x$  axis). The left hand side figure shows the attacks which were only detected by one probe, whereas the attacks shown in right hand side figure were detected by multiple probes.

number of events or the number of invalid passwords detected. The duration of an attack is the time interval between the first and the last event detected, and seen on the  $x$  axis. Several attackers were detected by multiple Komondor probes, because the SSH worm that intended to gain access to the subnetwork tried to login all hosts it found. Attacks which were only detected by one probe are shown in the left hand side graph, and attacks detected by multiple probes in the right hand side graph. Attacks detected by multiple probes usually suggest automatic worm programs using dictionary attacks [18].

This experience suggests that distributed intrusion detection can benefit from the advantages of DHTs. Namely, the attackers could be detected by several probes at the same time. As multiple hosts were attacked, by recognizing an attacker at any node of the Komondor network, several hosts can be protected at the same time. Moreover, as attack evidence came from multiple probes, one attack is likely to be associated to thousands or tens of thousands of events, which must be stored and processed in the distributed detection system. This type of load can be handled by a DHT fairly well.

#### 4.2 Network Cost of Intrusion Detection

Intrusion attempts issued against a host generate network traffic, regardless their success or failure. For example, for an SSH login attempt, the victim of the attack has to answer the request: a TCP connection is negotiated, encryption keys are exchanged and so on. When collaborative intrusion detection is used, sharing intrusion data will also increase the unnecessary network traffic, making

legitimate connections less responsive. Therefore, the network traffic induced by the intrusion detection system should be kept down to a minimum.

Table 2 shows measured traffic rates for SSH worm attacks, and the response traffic of Komondor hosts. For the tests, we used *SSH-2.0-OpenSSH\_5.5* on the server side, *SSH-2.0-OpenSSH\_5.8p1* on the client side, and *WireShark 1.6.2* packet capture software for packet inspection and statistics generation. The SSH server software was configured to abort connection on every third invalid login attempt, which means that attackers must reconnect the server on every third tested password. As Table 2 shows, every tested password costs the attacker 1518 bytes of traffic on the wire (i.e. including packet headers), and it costs the victim 1140 bytes.

The traffic generated by Komondor will also load the victim’s network. The estimations here are calculated for an overlay of  $2^{10} = 1024$  nodes. The Kademlia version of Komondor will initiate a node lookup on the first attack event detected. After the lookup is completed, each stored attempt only cost the overlay 103 bytes of traffic in our measurements. This optimization could not be achieved using Chord or CAN or other overlay topologies with rigid routing networks. Our estimation is that storing an attack event would cost the overlay 1030 bytes (on each event), if we would have used Chord for storage. The optimization of lookup requests greatly reduces the traffic generated by the overlay. Actual packet sizes, of course, would depend on the exact size of the overlay, as well as the exact type of attack information to be stored.

**Table 2.** Analysis of network traffic cost of SSH password attacks, for attackers, victims and the Komondor intrusion detection network. The table includes TCP control packets, as well as the count of header bytes. The bottom rows show the traffic overhead generated by intrusion detection, compared to the SSH server responses.

Traffic source	Packets	Bytes
Attacker	7	1518
Victim	6	1140
Komondor lookup (Kademlia)	20	2020
Komondor data (Kademlia)	1	103
Komondor data (Chord, model based estimation)	10	1200
Overhead (Kademlia)	20%	10%
Estimated overhead (Chord, model based)	166%	90%

### 4.3 Efficiency of Detection and Protection

Table 3 shows various attack types and the efficiency for the Komondor system regarding protection. The types listed are as follows:

- php-my-admin** This shows the activity of a worm scanning for open MYSQL administration web interfaces.
- cyberkit-ping** ICMP packets generated by the CyberKit Windows software [7].
- sql-overflow** Infection attempt by the Slammer worm [16].
- sshd-\*** Various login attempt failures detected by the OpenSSH software: broken connections (port scans), failed passwords for existing users and login attempts with invalid user names.
- vsftpd-fail-login** Invalid login attempts on the FTP servers.

The *protection* column shows the number of attacks for each type, for which the attack continued after it was blocked on the firewall, and the activity of the attacker was detected by another Komondor node of the same subnetwork. For these attacks, the collaborative intrusion detection can greatly enhance the protection of hosts.

The usefulness of the collaborative detection provided by the Komondor system varies with the approach used by the attackers or worm software in question. The Slammer worm randomly generates IP addresses to be attacked, and therefore it is highly unlikely that the same worm instance will attack two nodes of a Komondor network in a short timeframe. On the other hand, a port scan can usually be detected by many hosts on the same subnetwork using this method.

**Table 3.** Number of all attacks and attacks for which protection could be built by Komondor, for each attack types.

Type of attack	Attacks	Protection	Ratio
<b>php-my-admin</b>	107	71	66%
<b>cyberkit-ping</b>	546	515	94%
<b>sql-overflow</b>	4355	15	0%
<b>sshd-conn-lost</b>	490	321	65%
<b>sshd-failed-password</b>	546	219	40%
<b>sshd-invalid-user</b>	51	47	92%
<b>vsftpd-fail-login</b>	46	2	4%

## 5 Conclusions

Attacks on the Internet are a constantly growing problem. To detect sophisticated attacks promptly and correctly, intrusion data must be collected and analyzed automatically. In this article we have presented the Komondor intrusion detection system, which enables current attack correlation methods to be

upgraded to work in a distributed fashion, thereby improving their efficiency in case of for large-scale deployment. We achieved this by inserting a middle layer into the intrusion detection datapath, in which a key is attached to detected events. This key is then used to send the events for correlating to several correlation units that are organized as a DHT. This mechanism can be used to reduce network and computational load and increase reliability of the system, while still retaining the advantages of centralized approaches of intrusion detection.

## 6 Acknowledgement

The work reported in the paper has been developed in the framework of the project “Talent care and cultivation in the scientific workshops of BME”. This project is supported by the grant TÁMOP - 4.2.2.B-10/1-2010-0009.

## References

1. Snort – open-source intrusion detection system. <http://www.snort.org/>
2. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)* 36(4), 335–371 (2004)
3. Cuppens, F., Ortalo, R.: LAMBDA: A language to model a database for detection of attacks. In: *Recent advances in intrusion detection*. pp. 197–216. Springer (2000)
4. Czirkos, Z., Hosszú, G.: Peer-to-peer Based Intrusion Detection. *Infocommunications Journal* LXIV(I), 3–10 (2009)
5. Czirkos, Z., Tóth, L.L., Hosszú, G., Kovács, F.: Novel Applications of the Peer-to-Peer Communication Methodology. *Journal on Information Technologies and Communications* E-1(1(5)), 59–70 (2009)
6. Debar, H., Wespi, A.: Aggregation and Correlation of Intrusion-Detection Alerts. In: Lee, W., Mé, L., Wespi, A. (eds.) *Recent Advances in Intrusion Detection*, *Lecture Notes in Computer Science*, vol. 2212, pp. 85–103. Springer Berlin / Heidelberg (2001)
7. Duffield, N., Haffner, P., Krishnamurthy, B., Ringberg, H.: Rule-based anomaly detection on ip flows. In: *INFOCOM 2009*, IEEE. pp. 424–432. IEEE (2009)
8. Grizzard, J.B., Johns, T.: Peer-to-Peer Botnets: Overview and Case Study. In: *In USENIX Workshop on Hot Topics in Understanding Botnets (HotBots’07)* (2007)
9. Janakiraman, R., Waldvogel, M., Zhang, Q.: Indra: A Peer-to-peer Approach to Network Intrusion Detection and Prevention. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on. pp. 226–231. IEEE (2003)
10. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. pp. 654–663. STOC ’97, ACM, New York, NY, USA (1997)
11. Kemmerer, R.: Nstat: A model-based real-time network intrusion detection system. University of California-Santa Barbara Technical Report TRCS97 18 (1997)
12. Kemmerer, R., Vigna, G.: Intrusion detection: a brief history and overview. *Computer* 35(4), 27–30 (2002)

13. Kenyeres, P., Szentgyörgyi, A., Mészáros, T., Fehér, G.: BotSpot: Anonymous and Distributed Malware Detection. *Recent Trends in Wireless and Mobile Networks* pp. 59–70 (2010)
14. Krishnamurthy, S., El-Ansary, S., Aurell, E., Haridi, S.: A statistical theory of chord under churn. *Peer-to-Peer Systems IV* pp. 93–103 (2005)
15. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-peer Information System Based on the XOR Metric (2002)
16. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: Inside the slammer worm. *Security & Privacy, IEEE* 1(4), 33–39 (2003)
17. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31(4), 149–160 (2001)
18. Su, Y., Chen, Y., Chung, G., Wu, B.: Developing a ssh dictionary attack defense system in the multi platform environments through the analyzing log. In: *Internet Technology and Applications, 2010 International Conference on*. pp. 1–4. IEEE (2010)
19. Templeton, S., Levitt, K.: A requires/provides model for computer attacks. In: *Proceedings of the 2000 workshop on New security paradigms*. pp. 31–38. ACM (2001)
20. Valdes, A., Skinner, K.: Probabilistic Alert Correlation. *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* pp. 54–68 (October 2001)
21. Vlachos, V., Spinellis, D.: A PProactive Malware Identification System based on the Computer Hygiene Principles. *Information Management and Computer Security* 15(4), 295–312 (2007)
22. Yegneswaran, V., Barford, P., Jha, S.: Global intrusion detection in the domino overlay system. In: *Proceedings of NDSS*. vol. 2004 (2004)
23. Zhou, C.V., Karunasekera, S., Leckie, C.: A Peer-to-Peer Collaborative Intrusion Detection System. In: *Networks, 2005. 13th IEEE International Conference on*. vol. 1, p. 6. IEEE (2006)
24. Zhou, C., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security* 29(1), 124–140 (2010)