行政院國家科學委員會專題研究計畫　成果報告

整合格網與入侵偵測

計畫類別：個別型計畫
計畫編號：NSC94-2213-E-029-028-
執行期間：94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：東海大學資訊工程與科學系

計畫主持人：呂芳懌

計畫參與人員：李明昌　胡凱崴　李志揚　李國財

報告類型：精簡報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 95 年 10 月 5 日

行政院國家科學委員會專題研究計畫成果報告

整合網格與入侵偵測

Integrating GRID and Intrusion Detection

計畫編號：NSC 94－2213－E－029－028－

執行期限：九十四年八月一日至九十五年七月三十一日

主持人：呂芳懌　副教授　東海大學資訊工程與科學系

共同主持人：　無

計畫參與人員：李明昌 胡凱崴 李志揚　李國財

東海大學資訊工程與科學系

Abstract

In this article, we propose a fault-tolerant and grid-based IDS, named Fault-tolerant Grid Intrusion Detection System (FGIDS) which exploits grid's dynamic and abundant computing resources to detect malicious behaviors from a massive amount of network packets. In FGIDS, a detector can dynamically leave or join FGIDS anytime. A newly joined one is tested to obtain its key features' performance curves, which are used to balance detection workload among detectors. When a detector due to some reasons can not continue its detection leaving an unfinished task, FGIDS allocates another available one to take over. Therefore the drawbacks ordinary security systems have experienced can be then eliminated.

Keywords: DoS/DDoS, FGIDS, intrusion detection system, detection resource, fault-tolerant

2. Introduction

Recently, network security has significantly attracted researchers' attentions since hackers worldwide have tried to damage network systems anytime. This occurrence has seriously interfered our daily activities which often heavily relies on networks and Internet techniques. Among network attacks, denial-of-service (DoS) and distributed denial-of-service (DDoS) are the most serious and destructive ones. Network administrators often deploy intrusion detection systems (IDSs) to protect their systems.

Although, some traditional IDSs' detection accuracies are high in offline tests, when facing enormous traffic, some of them lose most of their detection capability or even crash since tremendous amounts of packets often markedly slow down detection speed, consequently disabling their detection power.

In this article, we propose a grid-based IDS, named Fault-tolerant Grid Intrusion Detection System (FGIDS), which detect attacks and manage dynamic components on a grid environment. Basically, a detection resource may newly join FGIDS, crash or overload anytime due to some management or maintenance reasons. Besides, FGIDS provides temporary backup mechanism, when the resource can not continue its detection, other resources can take over the unfinished tasks.

The rest of this article is organized as follows. Section 2 briefly describes grid computing, IDSs and fault-tolerant system. Section 3 presents FGIDS framework and detection algorithms. Experimental results are shown in section 4. Section 5 concludes this article.

3. Related Work

Grid, a dynamic and distributed virtual organization over LANs or WANs, aggregates distributed resources and technologies to process difficult and complex tasks and solve large-scale and complicated problems [1] aiming to enable dynamic selection, sharing, and aggregation of distributed autonomous resources not only based on processor availability, memory capacity, execution performance and data access cost of these computing resources, but also relied on an organization's specific task and/or burst of processing requirements.

Leu et al. [2] developed a performance-based grid intrusion detection system (PGIDS) which exploited grid's abundant computing resources to detect logical, DoS and DDoS attacks real-time so that the drawbacks that traditional IDSs suffer were then eliminated. However, PGIDS is performed on a static environment. Its detection flexibility is limited.

Jin et al. [3] proposed a fault-tolerant mechanism to handle three aspects of grid faults, including instance fault, node crash and agent crash based on Java exception-handling, Java thread state capturing technique and

mobile agent technology. If grid nodes failed, necessary software components were transferred to other locations through mobile agent or other mechanisms. Applications were then reconfigured there to provide uninterrupted services. Part of system-level components that failed was also corrected by this mechanism. Finally, grid node that crashed was dynamically deleted. User tasks specified with fault-tolerant sign would be processed correctly.

4. System Framework

The Internet comprises many autonomous network management units, e.g., enterprise's intranets and campus networks. In our proposed scheme, each network management unit employs a FGIDS as its security system. An overview of FGIDS is shown in Figure 1. A switch that has mirror port is placed on the link connecting a subnet to router to collect the subnet's inbound packets. Traffic coming from the mirror port is conducted to detect attacks [2, 4].

FGIDS, as shown in Figure 2, consists of dispatchers, scheduler, detectors, backup broker and block list database. Dispatcher receives packets from a switch's mirror port and predicts possible traffic volume for the next second. A detector detects DoS, DDoS and logical attacks, and gathers its own feature information, such as CPU level, available CPU capability, memory size, available memory capacity, detection capability which represents the average packets the detector is able to detect in one second, number of undetected packets, etc. Some of the features are dynamic, e.g., available CPU capability, available memory capacity, number of undetected packets and detection capability, that change with system load, whereas others are static.

Scheduler calculates feature scores for each detector as its current capability. Backup broker backups network traffic for future re-detection when a detector is unable to continue its detection. Block list database is a database holding definite intrusion information.

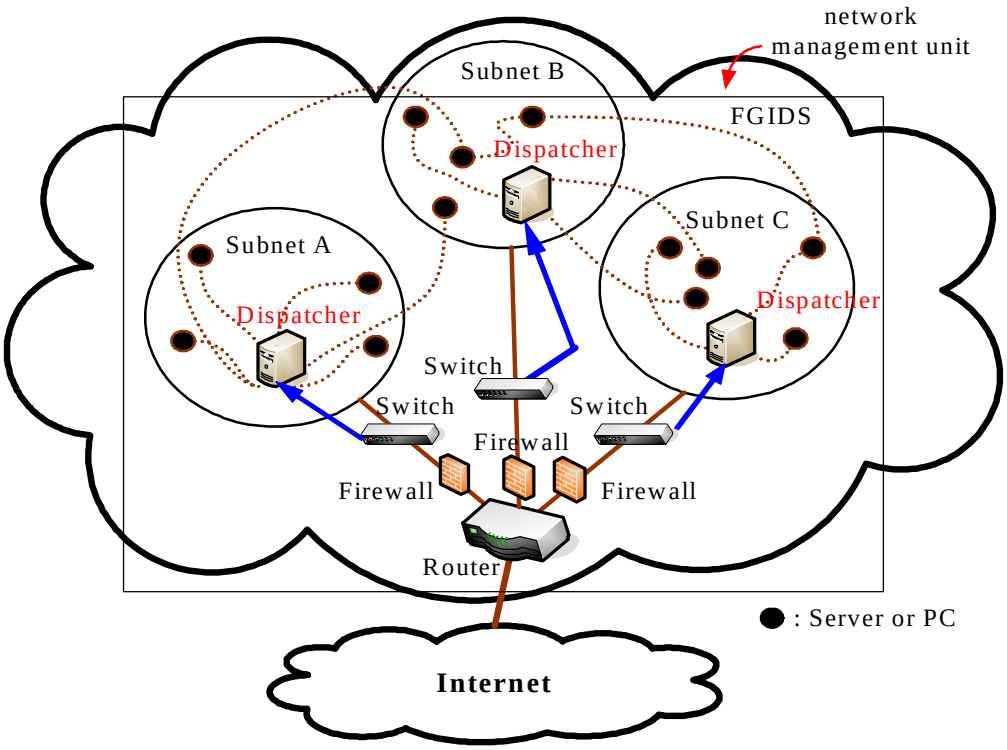

Figure 1 Overview of FGIDS [2, 4]

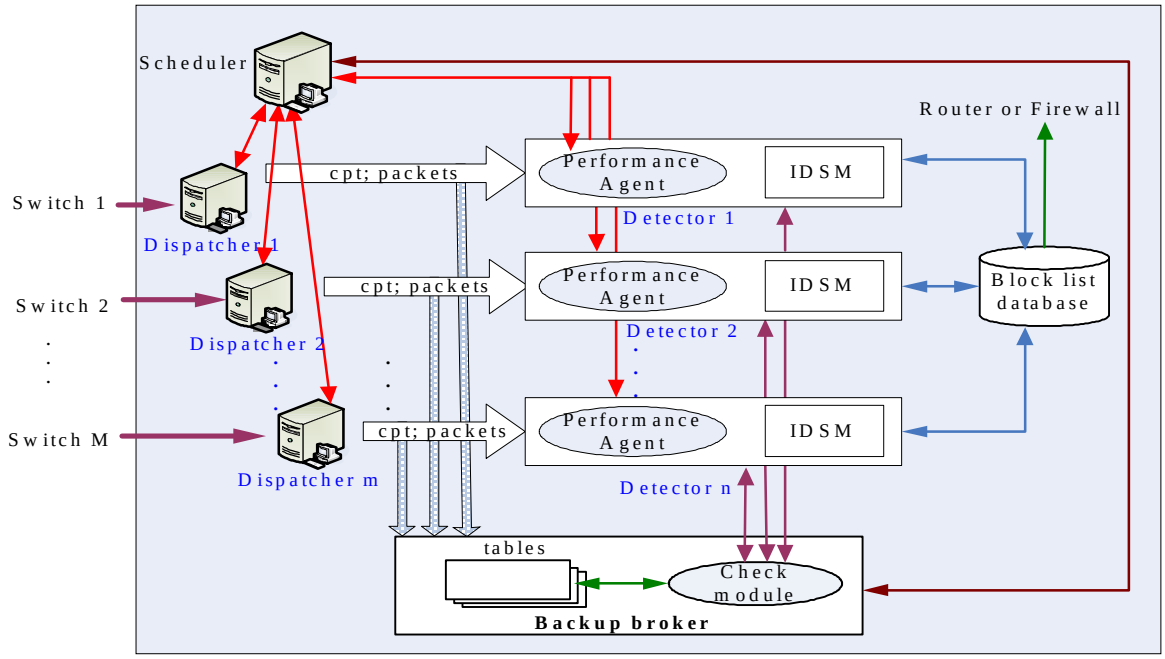

Figure 2 The FGIDS architecture (IDSM: Intrusion Detection sub-system module)

4.1 Dispatcher

A dispatcher, e.g., dispatcher i, utilizes tcpdump to gather packets and computes an exponential average at time *t* to predict traffic volume for *t+1*, denoted by $TV(i)_{t+1}$. $TV(i)_{t+1} = ai \cdot AT(i)_t + (1-ai) \cdot TV(i)_t, i = 1,2,...,m$ *(1)* where αi is the weight of the dispatcher i, $AT(i)_t$ and $TV(i)_t$ the numbers of packets that dispatcher i gathers and predicts respectively at time t and m is the number of dispatchers. Namely, prediction is distributively performed. After prediction, dispatcher i sends $TV(i)_{t+1}$ to scheduler.

Packets gathered by a dispatcher within one second are considered as a detection unit [4] in which every 200 packets are treated as a detection subunit.

4.2 Scheduler

Scheduler sorts $TV(i)_{t+1}$, $i=1,2,...,m$ received and allocates proper detectors to dispatchers. Generally, a larger $TV(i)_{t+1}$ consumes much more detection time; hence, a detector with better current capability will be allocated to detect a heavier traffic to balance detectors' load so as to shorten complete detection and response time.

Scheduler periodically queries detectors to see if they are still alive and able to work properly or not. If yes, it further interrogates detectors' feature status and refers to score tables [2, 4] to calculate feature scores. Tables 1 and 2 represent the score tables used in this research for detection capability and number of undetected packets, respectively. The CPU-level and memory-size score tables are generated by the same way.

A score book, a table comprising features and TSC (total score), is created for each detector. TSC_i, TSC of detector i, is calculated by $\sum_{j=1}^{p} (SC_{ij} \cdot W_j)$ where p is the number of features and SC_{ij} and W_j current score and weight of feature j respectively. In this research p=4, including cpu capability, memory capacity, detection capability and number of undetected packets whose weights are 1.0, 0.5, 1.0, and 0.5, respectively according to our experiment and previous observation. A detector with higher TSC should be one performing better in detection, and thus is allocated with a higher probability.

Table 1 The detection-capability score table

Number of packets detected in one second		Scores
Number of non-attack packets, X	Number of attack packets, A	
$X \geq 7{,}000$	$A \geq 1{,}400$	5
$6{,}000 \leq X < 7{,}000$	$1{,}200 \leq A < 1{,}400$	4
$5{,}000 \leq X < 6{,}000$	$1{,}000 \leq A < 1{,}200$	3
$4{,}000 \leq X < 5{,}000$	$800 \leq A < 1{,}000$	2
$X < 4{,}000$	$A < 800$	1

Table 2 The number-of-undetected-packets score table

Number of undetected packets, X	Scores
$X \geq 16{,}000$	**1**
$12{,}000 \leq X < 16{,}000$	**2**
$8{,}000 \leq X < 12{,}000$	**3**
$4{,}000 \leq X < 8{,}000$	**4**
$X < 4{,}000$	**5**

4.3 Detector

A detector consists of performance agent and IDSM (intrusion detection subsystem module). The former requests detector's current feature status. The latter, which is in charge of packet preprocessing, detection, intrusion notification, consists of head processor, logical detector, flood detector, heap tables and intruder analyzer [4].

Header processor retrieves serveal fields from an IP packet header and passes the field values to logical detector and flood detector to respectively detect logical and DoS/DDoS attacks. A heap table, recording packet statistics from which intruder analyzer can identify intruder[4]. A heap table, created for a host, e.g., $host_j$, in underlying subnet, has seven attributes, including SIP, protocol, type, count, size, temporary_count and temporary_size respectively representing source IP addresses of those packets sent to $host_j$, communication protocol, packet type (e.g., syn or icmp request), packet count, accumulated packet size of a source IP, temporary packet count and temporary accumulated packet size.

When completely receiving a detection subunit, flood detector updates the two fields count and size for the records, that have been modified, with formulas (2) and (3) respectively and finally resets the two fields temporary_count and temporary_size to zeros.

count = count + temporary_count *(2)*

size = size + temporary_size *(3)*

4.3.1 Normal Operation. The algorithm that scheduler uses to allocate detectors to dispatchers is as follows.

Algorithm: Scheduler allocates detectors to dispatchers

Input: $STSC=\{TSC_1, TSC_2,\ldots, TSC_n\}$ and $STV=\{TV(1)_{t+1}, TV(2)_{t+1},\ldots, TV(m)_{t+1}\}$;

Output: Each detection unit is assigned to a detector.

Begin:

1. Let $TSC_j=max(TSC_1, TSC_2,\ldots, TSC_n)$;
2. Let D_i be the Dispatcher with $\max(TV(1)_{t+1}, TV(2)_{t+1},\ldots, TV(m)_{t+1})$;
3. Notifies D_i to transmit its detection unit to detector j and backup broker;
4. Delete $TV(i)_{i+1}$ from TV and reevaluate TSC_j;
5. If (all detection units are assigned to detectors) then stop, otherwise go to step 1;

End.

On the other hand, each time a detection subunit is sent from a dispatcher to a detector and backup broker, a checkpoint, which is a timestamp representing that the last packet of this subunit has been delivered. A detector holds packets with a queue in which it sequentially detects malicious behaviors. After completely collecting a detection subunit, the detector deletes the subunit and transmits the checkpoint C that follows to backup broker which will mark the subunit as a detection commitment. Leu et al. [4] depicted that a DoS/DDoS attack could be detected within one second. Therefore, after finishing detecting a detection unit, the detector deletes the corresponding heap table to release memory space and sends an EOT (End of Traffic) to backup broker.

The performance agent of an detector, e.g., detector k, collects its own detection capability and number of undetected packets once per second and delivers them to scheduler to recalculate TSC_k immediately so as to accurately reflect the detector's current capability, $k = 1,2,3,\ldots,n$ since the size of a detection unit is often not completely equal, only approximates, to the predicted volume, e.g., $TV_{t+1}(l)$, if the detection unit is sent from dispatcher *l*.

4.3.2 A Newly joined Detector. When a node Z joins FGIDS as a detector, scheduler transmits two test files to Z to obtain its detection-capability performance curve. The first one, containing 16,455 attack packets, are 4 MB in length and transmitted to the detector within 3~4 seconds. The second consists of 16,455 unattack packets which are 3.6 MB in size and delivered within 30 seconds. Scheduler connects to Z, once when per 1,000 packets are transmitted, to observe the relationship between its detection demand time and current size of unfinished task or tasks. Figure 3 illustrates an example of a detection-capability performance curve, which can be in turn expressed by

$$\begin{cases} y_1 = b_{1,0} + b_{1,1} \cdot x \\ y_0 = b_{0,0} + b_{0,1} \cdot x \\ y_a = a \cdot y_1 + (1-a) \cdot y_0 \end{cases}$$

where x is the amount of network traffic, y_j the detection time ($j=1,0$ or a), $b_{1,0}$ and $b_{0,0}$ Z's loading coefficients, $b_{1,1}$ and $b_{0,1}$ Z's regression coefficients and a is the probability that the underlying subnet is attacked. Available-CPU-capability performance curve, illustrating the relationship between available CPU capability and number of undetected packets, and available-memory-capacity performance curve, showing the relationship between available memory capacity and number of undetected packets, are generated by the same approach. They are used to judge whether a detector crashes or is in its low performance status or not.

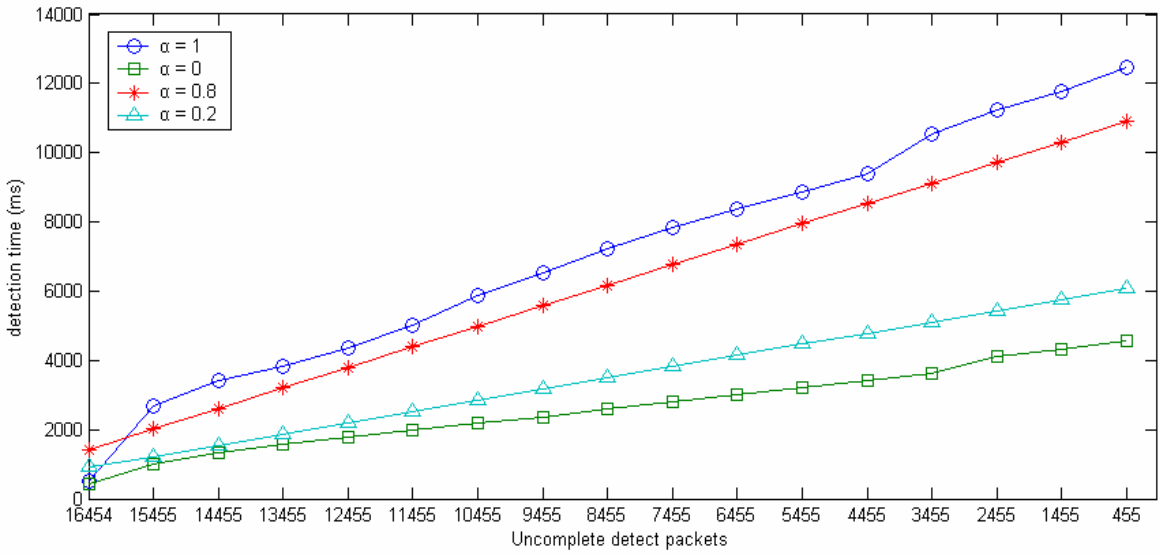

Figure 3 A detector's detection-capability performance curve (a : the probability the underlying subnet to be attacked)

4.3.3 A detector Crashes or is under low performance. Before crash, a detector, e.g., detector i:

(1) sends a message "before crash" to scheduler with exception handling;
(2) stops its detection;
(3) deletes temporary_count and temporary_size fields from all heap tables; /* due to an unfinished detection subunit*/
(4) transmits heap tables to backup broker;

Scheduler then:

(1) notifies detector i's dispatcher, e.g., dispatcher k, stop sending packets to detector i;
(2) recalculates TSC_l, $l=1,2,3,\ldots,n$;
(3) allocates a detector that has the highest TSC, e.g., detector j, to dispatcher k;
(4) sends a message (T-H, k, i, j) to backup broker where T-H stands for transmit heap tables;
(5) notify dispatcher k to transmit new arrival packets to detector j.

Meanwhile, packets keep flowing to backup broker from dispatcher k.

In addition, when detector i's available CPU capability or available memory capacity is below its threshold, detector i's performance agent actively sends "may crash" to scheduler. Moreover, if scheduler finds that detector i has crashed, it does the same but in step (4) message (T, i, k, j) instead of (T-H, i, k, j) is sent to backup broker since heap tables are lost.

4.4 Backup Broker

Backup broker backups traffic and records traffic information in backup table, which is a table consists of two fields, checkpoint and detection subunit. The table name, i.e., dispatcher's ID, is then inserted into index_table as an index. When receiving:

(1) a checkpoint cpt_g from a detector, e.g., detector i, that serves dispatcher k, backup broker marks cpt_g in backup table k with a "*" indicating the detection subunit has been completely detected.
(2) EOT from detector i, backup broker deletes backup table k.

(3) heap tables Hs from detector i, it save Hs.
(4) a message (T-H, k, i, j) from scheduler, it transmits the following items to detector j.
  (A) uncommitted packets and their checkpoints recorded in backup table k;
  (B) undetected packets including those of unfinished and undetected task or tasks also in backup table k;
  (C) detector i's heap tables Hs;
(5) a message (T, k, i, j), it unmarks all marked checkpoints and delivers all the packets backuped in table k, including detected and undetected packets and checkpoints of underlying detection unit, e.g., detection unit p and undetected task or tasks, to detector j which will generate new heap tables to re-calculate detector i's packet statistics.

4.5 Block List Database

Block list database lists the definite intrusion information, including the time an attack is discovered, intruder IP, victim IP, attack protocol and attack type. All detectors can access block list database to avoid having anyone repeatedly detecting packets issued by a known intruder so as to hugely save detection resources. Router or firewall can accordingly follow the block list to discard packets[2, 4].

5. Experimental Results

The test-bed of our experiments comprises resources of Tunghai University. We use two dispatchers to gather traffic from two subnets and deploy eight detectors as shown in Table 3 to detect attacks. All detectors run some version of Fedora operating system and globus toolkit 3.2. Detection programs are written with Java language and connected to MySQL.

Table 3 The detectors' specifications

Attributes / Node	Processor	Frequency (MHz)	Memory (MB)
Detector 1	Pentium IV	1,816.739	256
Detector 2	Pentium IV	1,514.253	256
Detector 3	Pentium III	797.712	256
Detector 4	Pentium III	598.664	256
Detector 5	Pentium IV	1,816.609	512
Detector 6	Pentium IV	1,816.809	1024
Detector 7	Pentium III	599.231	512
Detector 8	Pentium III	597.448	256

We tested the following security systems: Snort, Kaspersky Anti-Hacker 1.5 (Kaspersky in short), McAfee VirusScan Home Edition 7.0 (McAfee VirusScan in short), Panda Titanium Antivirus 2005 (Panda Titanium in short) and FGIDS.

DoS/DDoS attacks can be classified into two types: bandwidth consumption, e.g., udp flood, and resource consumption, e.g., icmp flood. Intrusion tools were invoked to launch DoS/DDoS bandwidth and resource consumption attacks. Table 4 lists the details where "ANP/sec" in the field packet intensity, standing for the average number of launched packets per second, represents different attack intensities. Each of udp, tcp syn and icmp floods was performed separately ten times, and each time lasted ten seconds in order to effectively and consistently compare the systems mentioned.

Table 4 Information about attacks

Attack type		Attack period	Packet intensity (ANP/sec)
Resource consumption attack	tcp syn flood	10 seconds	9,160
	icmp flood		705
Bandwidth consumption attack	upd flood		704

Table 5 lists the detection results of a resource consumption attack of tcp syn flood at 9,160 ANP/sec. Let k ($k \leq 8$) be the number of detectors really deployed to detect attacks. The items considered included: (1) average response time (ART) of the k detectors where response time is the time period from the beginning of the attack to the time the attack is discovered; (2) average blocking response time (ABRT) of the k detectors where blocking response time is the time period from the moment the attack is discovered to the time point an attack packet is firstly blocked; (3) average turnaround time (ATT) of the k detectors where turnaround time is the

period from the beginning of the attack to the time point detection ends; (4) average waiting time (AWT) of the k detectors where waiting time is the period from when a packet arrives at a detectors to the time point the packet begins to be detected. An SD appearing after ART, ABRT, ATT and AWT in Table 5 respectively represents standard deviation of ARTs, ABRTs, ATTs and AWTs obtained from the ten-fold experiments. ARTs generated by FGIDS is the shortest. Also, FGIDS detected an attack immediately, i.e., AWT=0. However, except FGIDS, all other systems concerned do not support measuring mechanisms to evaluate ABRT. Kaspersky, McAfee VirusScan, Panda Titanium and Snort systems do not provide measuring mechanisms to evaluate AWT.

Table 5 Detection results of tcp-syn-flood resource consumption attack at 9,160 ANP/sec

Statistics / Secu. systems	ART/SD (sec.)	ABRT/SD (sec.)	ATT/SD (sec.)	AWT/SD (sec.)	Nodes deployed
Kaspersky	10/0	--	10/0	--	Detector 1
McAfee VirusScan	2/0	--	10/0	--	Detector 1
Panda Titanium	2/0	--	10/0	--	Detector 1
Snort	6/0	--	13/0.03	--	Detector 1
FGIDS	0.74/0.08	0.18/0.08	10/0	0/0	Detector 1~8

Table 6 Detection results of icmp-flood resource consumption attack at 705 ANP/sec

Statistics / Systems	ART/SD (sec.)	ABRT/SD (sec.)	ATT/SD (sec.)	AWT/SD (sec.)	Nodes deployed
Kaspersky	10/0	--	10/0	--	Detector 1
McAfee VirusScan	2/0	--	10/0	--	Detector 1
Panda Titanium	2/0	--	10/0	--	Detector 1
Snort	8/0	--	15/0.02	--	Detector 1
FGIDS	0.61/0.08	0.05/0.03	10/0	0/0	Detector 1~8

Table 6 lists the detection results of a resource consumption attack of icmp flood at 705 ANP/sec. FGIDS only took 0.61 seconds to detect an attack.

The results of bandwidth consumption attack by issuing udp flood 704 ANP/sec is shown in Tables 7. Now, we can conclude that FGIDS performed the best in detecting the two types of consumption attacks.

Table 7 Detection results of udp-flood bandwidth consumption attack at 704 ANP/sec

Statistics / Secu. systems	ART/SD (sec.)	ABRT/SD (sec.)	ATT/SD (sec.)	AWT/SD (sec.)	Nodes deployed
Kaspersky	10/0	--	10/0	--	Detector 1
McAfee VirusScan	2/0	--	10/0	--	Detector 1
Panda Titanium	2/0	--	10/0	--	Detector 1
Snort	5/0	--	13/0.01	--	Detector 1
FGIDS	1.77/0.09	0.04/0.02	10/0	0/0	Detector 1~8

Logical attacks are attacks that intruders send irrational packets to crash or destroy a victim (maybe a host, router or network) or to make the victim work improperly by exploiting its vulnerabilities. To evaluate FGIDS's detection accuracy, the normal amount of traffic and attack traffic each were issued 1000 times. Each time traffic also lasted 10 seconds. Among the 1000 attacks, 500 were DoS/DDoS resource consumption attacks and the remaining 500 DoS/DDoS bandwidth consumption attacks. The attack intensities range from 500 to 10,000 packets/sec.

Table 8 shows the detection accuracy of resource/bandwidth consumption attacks. The error comes from low attack intensities being treated as normal traffic. Table 9 shows the detection accuracies of logical attacks. Experimental data included 1000 packets of different attacks, and 1000 non-attack packets.

Table 8 Detection accuracy of resource/bandwidth consumption attacks

Statistics / Secu. systems	True positive	True negative	False positive	False negative	Detection accuracy
Snort	**89.5%**	**95.2%**	**10.5%**	**4.8%**	**92.35%**
Kaspersky	**99.2%**	**100%**	**0.8%**	**0%**	**99.60%**
McAfee VirusScan	**89.5%**	**100%**	**10.5%**	**0%**	**94.75%**
Panda Titanium	**87.7%**	**100%**	**12.3%**	**0%**	**93.85%**
FGIDS	**99.3%**	**100%**	**0.7%**	**0%**	**99.65%**

Table 9 Detection accuracy of logical attacks

Statistics / Secu. systems	True positive	True negative	False positive	False negative	Detection accuracy
Snort	**95.6%**	**94.4%**	**4.4%**	**5.6%**	**95%**
Kaspersky	**100%**	**97%**	**0%**	**3%**	**98.5%**
McAfee VirusScan	**100%**	**96.5%**	**0%**	**3.5%**	**98.25%**
Panda Titanium	**100%**	**95.8%**	**0%**	**4.2%**	**97.9%**
FGIDS	**100%**	**97%**	**0%**	**3%**	**98.5%**

Different nodes have different crash and low-performance thresholds due to heterogeneous detection environment. Figures 4 and 5 respectively show a detector's available-CPU-capability and available-memory-capacity performance curves in which average available CPU capability and available memory capacity are 47.38% and 67.46%, respectively. Table 10 summarizes the crash and low-performance thresholds for this detector. According to our experience and observation, the available CPU capability's crash threshold and low-performance threshold are obtained by respectively subtracting two standard deviations and one standard deviation from average available CPU capability. Similarly, the available memory capacity crash threshold and low-performance threshold are obtained by subtracting three and two standard deviations from average available memory capacity, respectively.

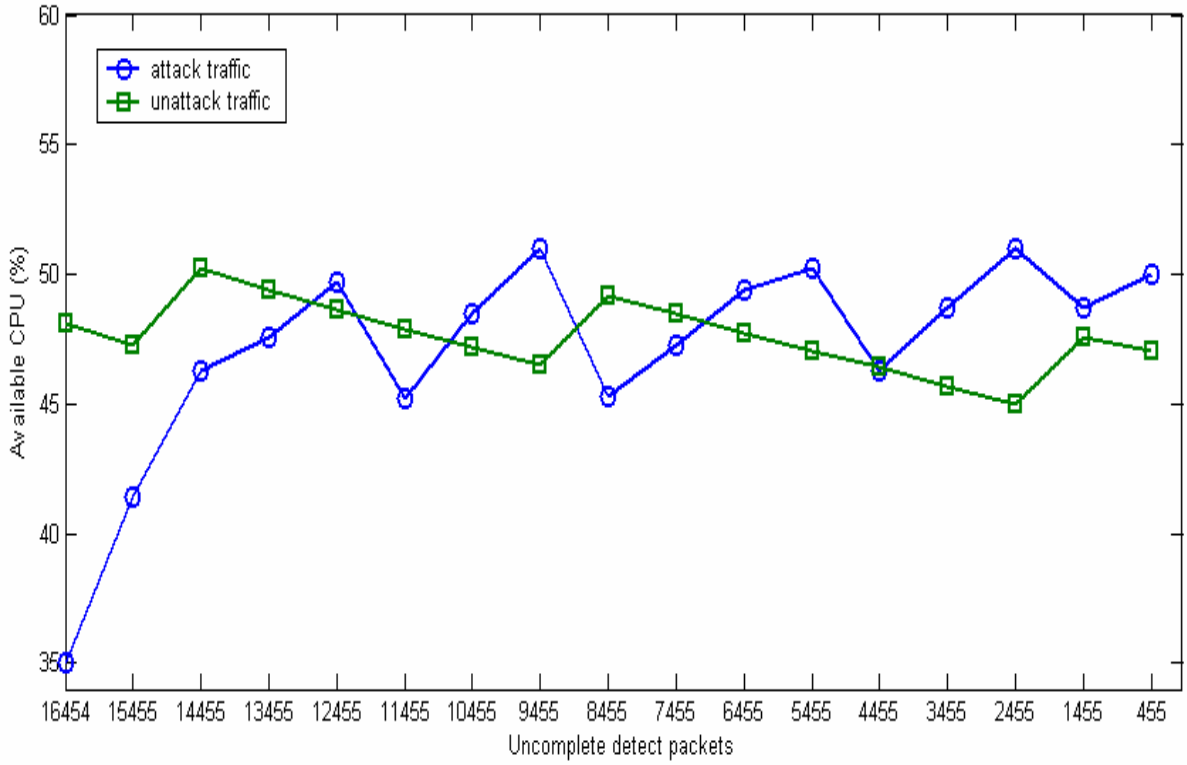

Figure 4 A detector's available-CPU-capability performance curve

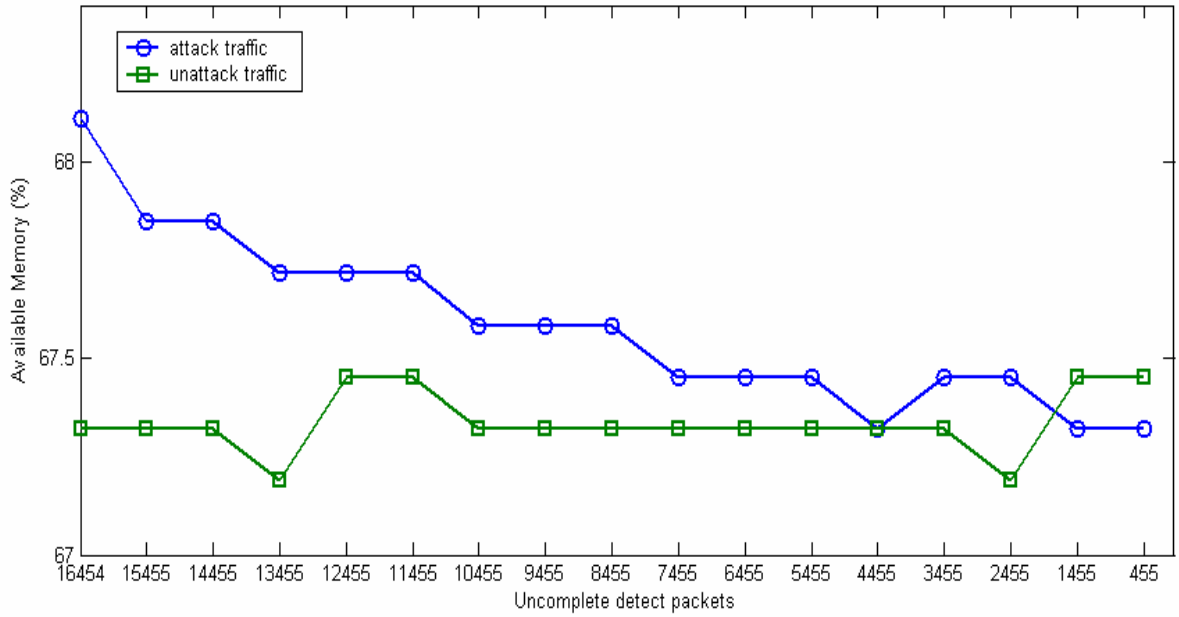

Figure 5 A detector's available-memory-capacity performance curve

Table 10 An example: detector's average performance, crash threshold and low-performance threshold

Feature	Average	Crash-threshold	Low-performance threshold
Available CPU capability	**47.38%**	**17.98%**	**29.74%**
Available memory capacity	**67.46%**	**26.62%**	**36.83%**

6. Conclusion

This article proposes the FGIDS architecture which provides a dynamic detection environment. Therefore, we consider some of detector's probable events, e.g., a detector newly joins a FGIDS or a detector crashes or is overloaded by extra tasks. When FGIDS's detection and analytical performance becomes unbearably low very frequently, we can join some powerful detectors to FGIDS to improve its system performance. If a detector can not continue its detection, scheduler will choose another one to take over its task or tasks so that using backup broker FGIDS can completely detect whole network traffic without losing any logical and/or DoS/DDoS attacks. The occurrence improves drawbacks that a static detection environment suffers.

Our experimental results show that FGIDS can effectively detect logical, resource and bandwidth consumption attacks. Compared to the other security systems tested, FGIDS performs the best and has the highest detection accuracy.

7. References

[1] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann: San Francisco (CA), 1999.

[2] F.Y. Leu, J.C. Lin, M.C. Li and C.T. Yang, "A Performance-Based Grid Intrusion Detection System," *Proc. of IEEE Annual International Computer Software and Applications Conf.*, pp. 525-530, July 2005.

[3] L. Jin, W.Q. Tong, J.Q. Tang and B. Wang, "A Fault-Tolerant Mechanism in Grid," *Proc. of IEEE International Conference on Industrial Informatics*, pp. 457-461, Aug. 2003.

[4] F.Y. Leu, M.C. Li and J.C. Lin, "A Real-time Grid Intrusion Detection System," Submitted to the *IEEE Trans. on Dependable and Secure Computing*.