www.ietdl.org

Published in IET Information Security
Received on 26th May 2011
Revised on 22nd May 2012
doi: 10.1049/iet-ifs.2011.0154

ISSN 1751-8709

Agent-based honeynet framework for protecting servers in campus networks

I.S. Kim M.H. Kim

School of Computer Science and Engineering, Soongsil University, 369 Sangdo-Ro, Dongjak-Gu, Seoul 156-743, Korea
E-mail: iksuplorer@gmail.com

Abstract: Intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) that use signatures cannot protect servers from new types of internet worms. Therefore it is important to collect information about new attacks because the detection rules employed by IDSs and IPSs are formulated using this information. Honeypots are valuable security resources that act as baits for attackers. They can monitor intrusions by being probed, attacked or compromised and can detect zero-day attacks and provide researchers intending to improve security with information about the attacks. However, it is almost impossible to immediately generate detection rules from the information collected by honeypots. This study presents an agent-based honeynet framework for protecting servers in a campus network. In this framework, agents remove malicious processes and executable files on servers infected by zero-day attacks as soon as the honeynet detects them. The proposed framework provides a novel defense mechanism that protects servers from new types of internet worms effectively, without the use of signatures.

1 Introduction

With the increasing demand for computer security, there is an increasing need to develop computer security systems that provide adequate protection to all users. In order to improve computer security, security systems such as intrusion detection systems (IDSs) and intrusion prevention systems (IPSs) have been developed. Most security systems use signatures to detect and prevent attacks. Consequently, it is very important for these systems to collect information regarding new attacks as soon as possible, so as to promptly obtain accurate signatures of these attacks.

Honeypots are security resources deployed with the intention of allowing them to be compromised [1]. They are specifically configured to collect information about the strategies and tools of intruders. This information is analysed with the purpose of updating security systems so that the systems can respond to new attacks. Honeypots are also used to decrease the probability of intruders attacking production servers. The probability of luring intruders increases with the number of honeypots deployed. However, an increase in the number of deployed honeypots must be accompanied by a corresponding increase in the number of required internet protocol (IP) addresses and computer hardware. For this reason, previous studies have addressed honeyfarms that regard interactions with unused IP addresses as suspect interactions [2–4]. However, the results of these studies cannot be adopted as suitable solutions for class C networks with an insufficient number of unused IP addresses.

This paper presents an agent-based honeynet framework for protecting servers in a campus network. The proposed framework consists of DecoyPorts [5], through which the probability of intrusion detection is increased, whereas the probability of intruders attacking production servers is decreased. A DecoyPort is a port designed to lure intruders and is not used for services. Therefore every interaction with a DecoyPort is considered to be a suspect interaction. In this framework, agents create DecoyPorts on clients in order to attract intruders. When intruders are granted access to the DecoyPorts, agents redirect them to the honeynet. As the honeynet detects zero-day attacks, agents remove malicious processes and executable files on infected servers. The proposed framework provides a novel defense mechanism that protects servers from new types of internet worms effectively, without the use of signatures.

The rest of this paper is organised as follows. Section 2 presents the background of IDSs and previous studies in the domain of decoy technology. Section 3 describes the proposed framework and Section 4 presents an evaluation of the performance of the proposed framework. Section 5 concludes this paper.

2 Related work

2.1 Intrusion detection systems

During server intrusions, attackers exploit the vulnerabilities in the server programs. The attackers also write self-propagating programs, known as internet worms, which are capable of compromising many servers in a short period of time.

A general method for detecting such attacks is to analyse log files periodically; however, it is impossible to detect the attacks in real time. For this reason, many studies have been conducted on IDSs to facilitate intrusion detection [6–8].

Snort is an open-source network-based IDS designed by Snort team [6]. Through protocol analysis and content searching and matching, Snort can detect attacks such as denial of service, buffer overflow, CGI attacks and stealth port scans. When Snort detects incoming malicious packets, it raises an alert. However, it is almost impossible to generate detection rules against zero-day attacks promptly. As a result, anomaly-based IDSs have been developed, which establish a baseline of normal usage patterns and regard anomalous behaviour as attacks. Yudong *et al.* proposed a detection technique that scrutinises the nature of internet worms to cope with zero-day attacks [7]. Self-replication is a common feature of most internet worms. The technique proposed in [7] detects the self-replication attempts of internet worms during run time by using a system service dispatch table. This technique is useful before the security patch or signature is created. However, it cannot respond to malicious activities, such as accessing and modifying the file system, prior to the self-replication activities of the worm.

2.2 Decoy systems

Honeypots are valuable security resources providing bait to attract attackers. Once attackers are lured, honeypots are in a position to monitor intrusions by allowing themselves to be probed, attacked or compromised [1]. Honeypots are not production systems, and so any suspicious intrusions can be monitored separately from the main system. Monitoring of this activity provides security researchers with information about the motives and tactics of attackers [9–12].

Honeypots can be classified according to their level of interaction with attackers. Low-interaction honeypots simply emulate services such as telnet, FTP and HTTP. Attackers are limited to interacting with these decoy services. Low-interaction honeypots capture known attacks, rather than unknown ones. An example of a low-interaction honeypot is the honeyd that creates virtual hosts on a network [13]. Honeyd enables a single host to claim multiple unused IP addresses in a network. Every interaction with the unused IP addresses is considered suspect. High-interaction honeypots are systems with full-blown operating systems and applications, capable of collecting large amounts of information about unknown attacks. A honeynet such as GEN II is an example of a high-interaction honeypot [14, 15].

One disadvantage of honeypot systems such as honeyd and GEN II are that they can capture only those intruders who interact with the honeypots. The greater the number of honeypots deployed, the greater the probability of identifying, capturing and tracking attackers' activities [16–18]. However, an increase in the number of deployed honeypots results in a consequent increase in the manpower required for honeypot-related administration.

Spitzner introduced the concept of honeyfarms to consolidate a large number of honeypots distributed across the world [2]. When attackers interact with unused, predetermined IP addresses in these networks, redirectors transport them to the honeyfarm. The honeyfarm concept simplifies the analysis and maintenance of honeypots. Jiang *et al.* [3] implemented a honeyfarm that hosts and manages a large number of high-interaction honeypots in a network. This system supports client-side honeypots as well as server-side honeypots. Server-side honeypots such as honeyd and GEN II wait for attackers' intrusions. On the other hand, client-side honeypots such as HoneyC [19] and Capture-HPC [20] are run within client machines, actively interact with servers on the internet and check unauthorised changes in the registry, file system and process structure of the client machines.

A hybrid honeypot framework proposed by Artail *et al.* [4] also considers interactions with unused IP addresses as suspect. This framework improves the currently deployed IDSs for protecting networks from intruders. The framework design is structured around the deployment of honeyds using available unused IP addresses. The traffic directed towards honeyds is redirected to high-interaction honeypots. As Jiang *et al.* [3] and Artail *et al.* [4] employ unused IP addresses to identify intruders, their performance depends on the number of available unused IP addresses. Accordingly, they are not suitable for class C networks with an insufficient number of free IP addresses.

For this reason, methods using unoccupied port space have been introduced [5, 21, 22]. Honey@home [21] forwards traffic to unused IP addresses or unused ports to a honeyfarm and sends the answers provided by the honeypots back to the attacker. The Honey@home architecture relies on communities of regular users installing a lightweight honeypot that monitors unused addresses and ports. However, Honey@home does not deal with intrusion protection. DecoyPort [5] is similar to Honey@home but deals with intrusion protection. The attackers that have accessed DecoyPorts are denied access to service ports by the DROP reaction of IPTables. The DROP reaction would be harmful to users of computers infected by internet worms, as they would lose access to services.

There have been several works on the decoy files for detecting malicious insider activities [23–25]. Bowen *et al.* [23] proposed a trap-based defense mechanism, which automatically generates and stores decoy documents on a computer to be protected. The decoy documents are files intended to deceive attackers. The decoy documents contain bogus credentials and stealthy beacons. The stealthy beacon sends a signal to a centralised server when an intruder uses or flows out the decoy documents. Although the decoy files cannot promptly detect external attacks from internet worms, if the decoy files are used in IDSs, internet worm attacks can be detected as soon as possible.

In this paper, the proposed framework protects servers from new internet worms effectively. Unlike the framework used in existing honeypot-related systems, this framework removes malicious processes and executable files on servers infected by zero-day attacks without using signatures. Most of all, the proposed framework can be adopted as suitable solutions for class C networks with an insufficient number of unused IP addresses.

3 Agent-based honeynet framework

3.1 Agent-based honeynet deployment

In network environments with strict security policies, a firewall allows incoming network requests into public servers that host HTTP and FTP services, but it denies requests into other hosts. These environments have the advantage of higher network security, but private servers are not allowed to be built into the network. In network environments with loose security policies, such as campus networks, a firewall denies access only to the ports of protected servers. Therefore graduate students and professors can build their own servers that host, for example, personal websites and remote desktops.

The objective of the agent-based honeynet framework is to decrease the probability of intruders attacking servers in a campus network and to effectively protect the servers. Fig. 1 shows an example of network deployment using the proposed framework. The framework consists of a manager server, a honeywall [26], a honeynet and agents. The agent is a program installed on a client and a server machine, which creates DecoyPorts designed to lure internet worms. Therefore every interaction with a DecoyPort can be considered a suspect interaction. The agent also removes malicious processes and executable files created by internet worms, using information from the manager server.

The DecoyPorts and general ports are represented by dark and white circles, respectively. When intruders interact with DecoyPorts, they are redirected to honeypots, as shown in Fig. 1. The number of DecoyPorts opened by agents is the same as the number of ports opened by honeypots. When an intruder sends a request to port 21, 23, 25 or 80 belonging to the client with the IP address 192.168.0.60, the request is redirected to the honeypot with the IP address 192.168.0.10 or 192.168.0.11. Clients are appropriate for the creation of DecoyPorts because they usually do not require significant computing resources and rarely use service ports for server programs.

The manager server automatically customises agents according to the environment of the honeynet and clients. If the agents needlessly open many DecoyPorts, system and network overheads are inevitable. For this reason, the manager server ensures that the number of DecoyPorts created by the agents is the same as the number of ports opened by the honeypots. The honeywall separates the honeynet from production systems in the internal and external networks and prevents intruders from using the honeynet to attack other non-honeynet systems. Activities are logged by the honeywall using Snort.

3.2 Functional components

The functional components of the proposed framework are shown in Fig. 2. The Hnet.conf file in the manager server includes the IP addresses and open port numbers of honeypots. As soon as an agent is activated, the agent initialiser sends a query to the manager and receives the IP address and open port numbers of a honeypot from the agent manager. The DecoyPort generator then creates DecoyPorts for luring and redirecting intruders by calling the socket() function. As DecoyPorts are not used for services but for luring attacks, the recv() function strips off the header of packets received from a honeypot or an attacker, and the send() function adds a new header to the packets, as shown in Fig. 3. The regenerated packet is sent to the honeypot or attacker. When an attacker is redirected to a honeypot by the DecoyPort, if the attacker executes ifconfig or netstat, the honeypot can easily be fingerprinted. The alter_data() function alters the data included in packets to prevent the attacker from recognising the presence of the honeypot. The usleep() function suspends the transmission of packets for a specified number of microseconds and the limit_session() function limits the number of concurrent attacker connections. A user can specify the number of

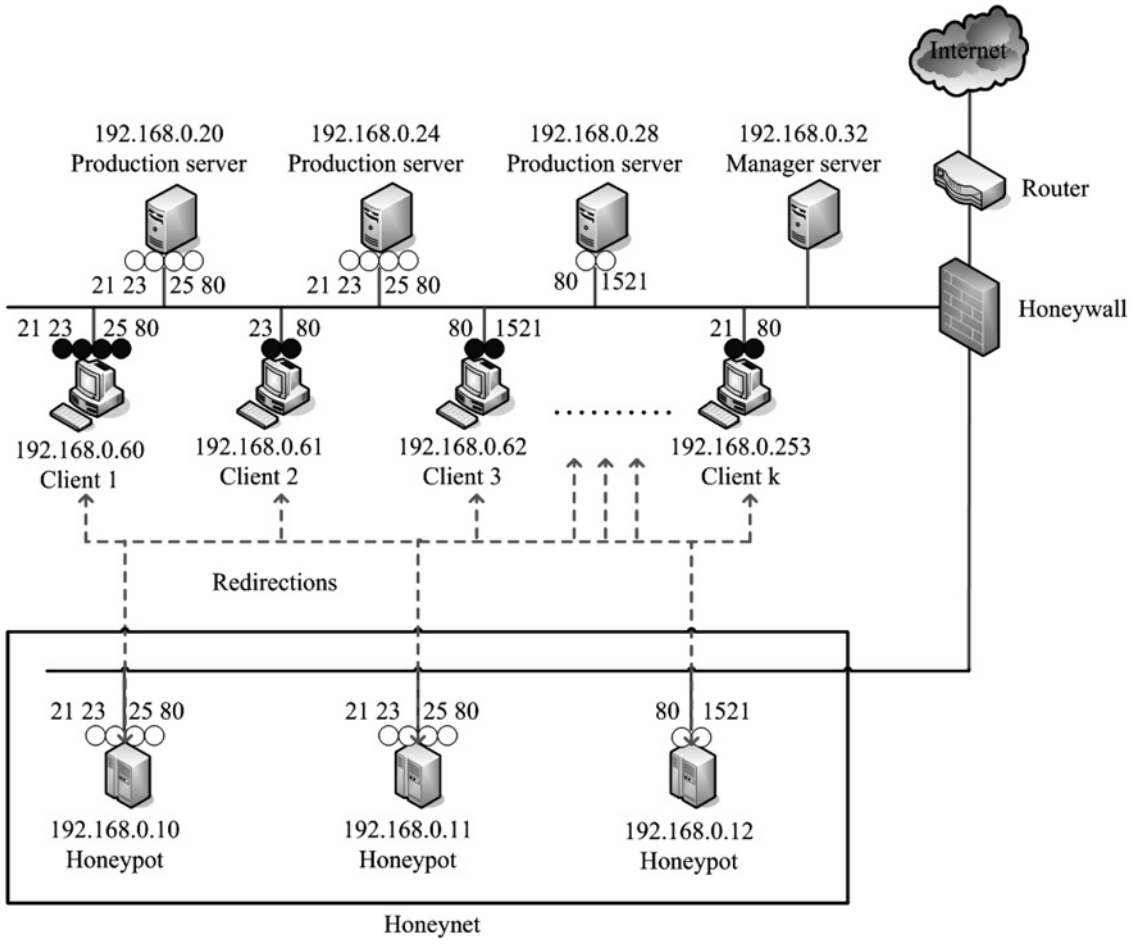

Fig. 1 *Example of network deployment using the proposed framework*

doi: 10.1049/iet-ifs.2011.0154

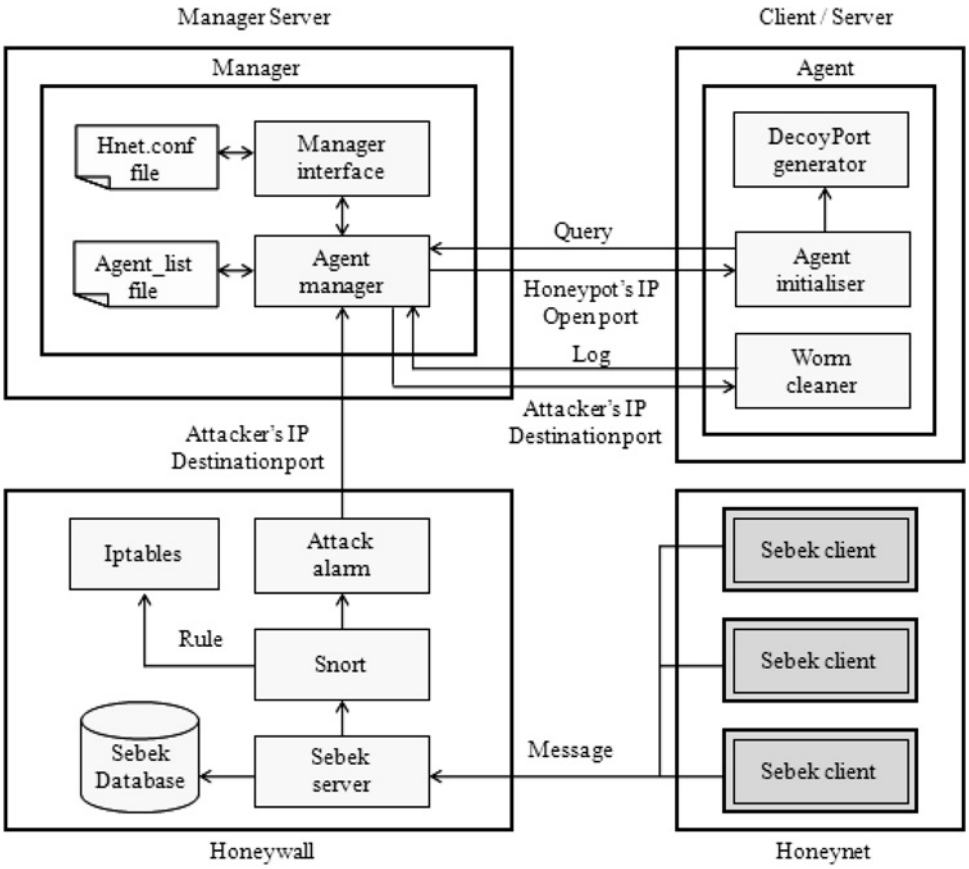

Fig. 2 *Functional components of the proposed framework*

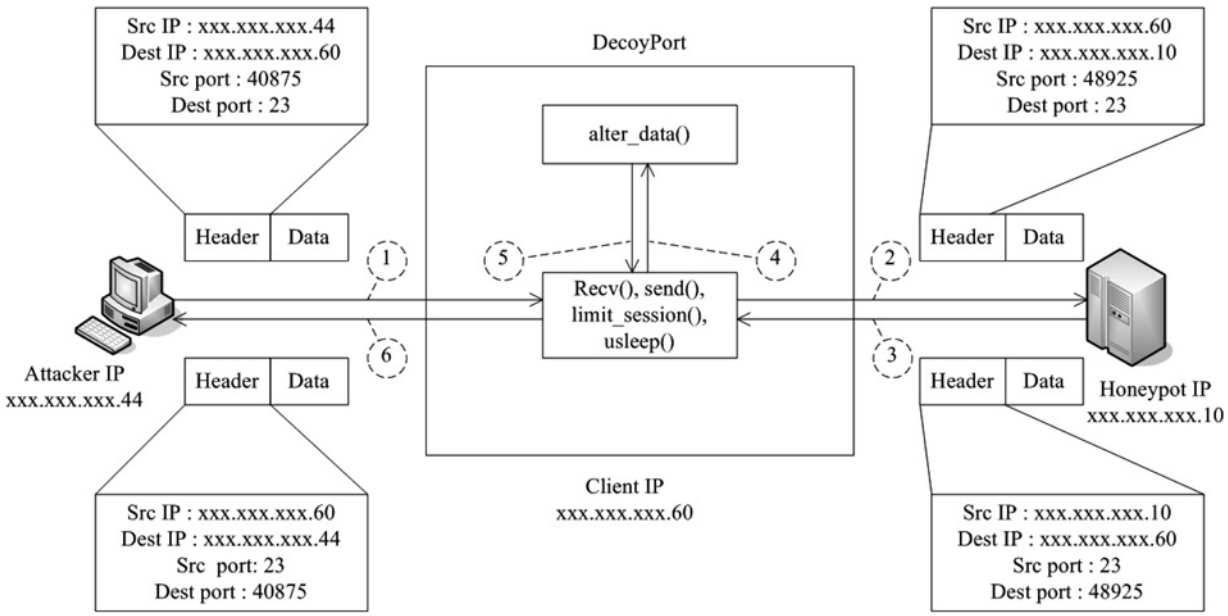

Fig. 3 *Packet manipulation by DecoyPorts*

concurrent attacker connections. This minimises the overheads caused by the denial-of-service. However, the server agents have only a worm cleaner, so DecoyPorts are not created on servers. When the agent is running on a client and a server, the IP addresses of the client and server are added to the Agent_list file, and when the agent is stopped, these IP addresses are removed from the Agent_list file.

Internet worms can compromise many vulnerable servers in a short period of time by sending malicious packets to all the hosts in a network. As Snort uses only the signatures of well-known attacks to detect and prevent attacks, it is unable to provide security for servers against new internet worms. Moreover, as intruders use the encryption channel to attack the honeynet, Snort is not suitable for monitoring their activities. On the other hand, Sebek can monitor

intruders using session encryption [27]. Sebek captures the keystroke data of encrypted sessions from within the honeypot kernel and exports the decrypted data to the honeywall. Furthermore, the Sebek client can detect unknown attacks by monitoring processes created on honeypots.

In the proposed framework, when a new internet worm attacks the honeypots, the Sebek clients will inform the Sebek server about the attack. An attack alarm then sends a message, including the attacker's IP address and the attacked port number, to the agent manager. The agent manager sends this message to all the agents in the Agent_list file, enabling the worm cleaners to search for processes and executable files created by the new internet worm.

A daemon is a process that executes in the background while waiting for connection requests. Most server programs are implemented with a daemon process in order to improve performance. When a daemon process receives the connection request of consumers, it creates a child process that communicates with one consumer. The processes running on an infected server are shown in Fig. 4. Assuming that the daemon is a vulnerable daemon process, when an internet worm succeeds in attacking it, a new child process is created. Thereafter, the internet worm creates a shell process and a malicious process.

The worm cleaner knows the source IP address of the attacker and the port number of the attacked service; therefore it can search for the attacked daemon process with the 'netstat-anp' command, as shown in Fig. 5*a*. Thereafter, the worm cleaner searches for malicious processes with the 'ps-axj' command, as shown in Fig. 5*b*. In this case, PPIDs are the parent process IDs, and SIDs are the session IDs. A session is a collection of one or more process groups. Every other process created from a process has the same SID. Fig. 5*a* exemplifies a scenario where an internet worm has attacked the server through port 23, which is bound to the daemon process whose PID is 19204. The daemon process has created a shell process and the shell process has created the 'malware' process. After the worm cleaner finds all the malicious processes created by the internet worm, it kills them by sending the SIGKILL signal to the kernel. The worm cleaner only removes malicious processes created from a worm. Therefore daemon processes can continue to provide services.

The executable files of the worm can be searched with the 'lsof | grep malware' command. In Fig. 5*c*, '/tmp/malware' is the executable file path of the 'malware' process, and '/lib/ld-2.7.so' and '/lib/libc-2.7.so' are the paths of the system library files used by the 'malware' file. The worm cleaner only deletes the files having the same name as that of the malicious process. Accordingly, system library files can still be used by other legitimate programs.

3.3 Probability of luring attackers

In a network where a redirector and a honeynet are deployed, the probability of luring attackers to DecoyPorts when intruders try to attack server programs is given by the total number of IP addresses allocated to the redirector and honeynet, divided by the number of IP addresses allocated to hosts responding to the attackers. Therefore the probability of luring attackers is as follows

$$P_{\mathrm{LA}} = \frac{\mathrm{NRIP} + \mathrm{NHIP}}{\mathrm{NSIP} + \mathrm{NRIP} + \mathrm{NHIP}} \tag{1}$$

where NRIP denotes the total number of free IP addresses allocated to the redirector. NHIP and NSIP denote the total number of IP addresses allocated to the honeynet and the servers, respectively.

Fig. 6 shows the probability of luring attackers in a class C network. The greater the number of free IP addresses, the higher the probability of luring attackers. When the number

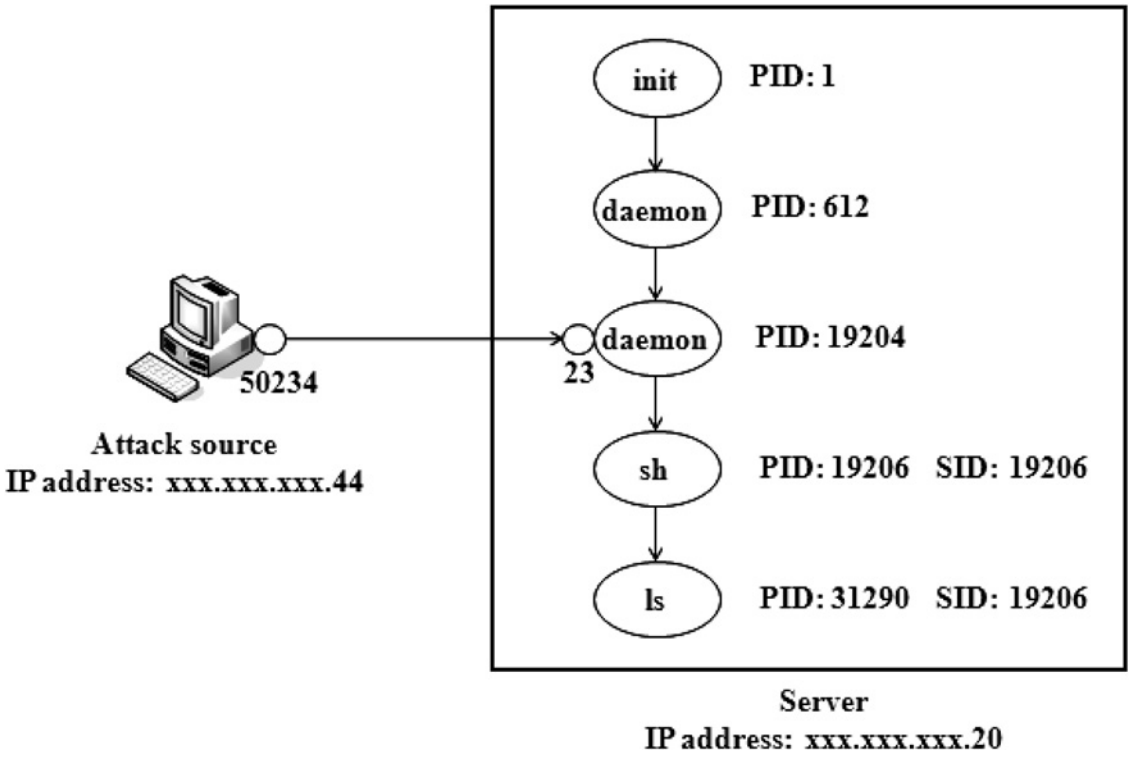

Fig. 4 *Processes on an infected server*

doi: 10.1049/iet-ifs.2011.0154

Local Address	Foreign Address	PID/Program name
xxx.xxx.xxx.20:23	xxx.xxx.xxx.44:50234	19204/daemon

a

PPID	PID	SID	COMMAND
612	19204	612	/usr/sbin/daemon
19204	19206	19206	/bin/sh
19214	19247	19206	./malware

b

COMMAND	PID	NAME
malware	19247	/tmp/malware
malware	19247	/lib/ld-2.7.so
malware	19247	/lib/libc-2.7.so

c

Fig. 5 *Searching for malicious processes and executable files with netstat and ps*

a Searching the attacked process ID with netstat
b Searching malicious process with ps
c Searching malicious file with lsof

of IP addresses used by the redirector and honeynet is low, the probability of luring attackers is also very low, as shown in Fig. 6.

In order to reduce the rapid decrease in P_{LA}, agents using IP addresses of clients to lure attackers can be installed on client systems. In this case, the total number of IP addresses allocated to agents is given by the following equation

$$\text{NAIP} = \text{NCIP} \times \frac{\text{PA}}{100} \tag{2}$$

where NCIP and PA denote the total number of IP addresses allocated to clients and the percentage of agents to be installed on the client systems, respectively.

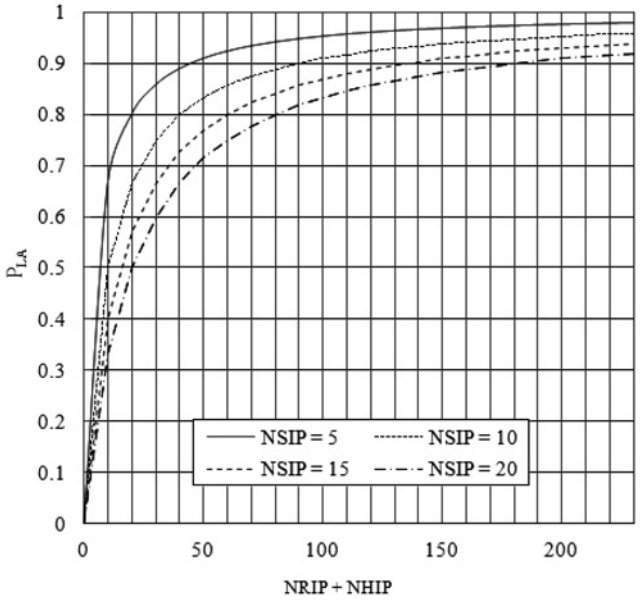

Fig. 6 *Probability of luring attackers in class C network*

Graph shows the probability of luring attackers using a redirector and honeynet

On client machines running server programs, agents open only a few DecoyPorts. Therefore PA is calculated as follows

$$\text{PA} = \sum_i \frac{\text{NDP}_i}{\text{NHP}_i \times \text{NA}_i} \tag{3}$$

where NDP_i denotes the total number of DecoyPorts redirecting attacks to the ith honeypot. NHP_i and NA_i denote the total number of ports opened by the ith honeypot and agents redirecting attacks to the ith honeypot, respectively.

Finally, the probability of luring attackers is as follows

$$P_{\text{LA}} = \frac{\text{NRIP} + \text{NHIP} + \text{NAIP}}{\text{NSIP} + \text{NRIP} + \text{NHIP} + \text{NAIP}} \tag{4}$$

4 Case study

4.1 Test beds and attack scenario

After attackers probe the available services on a server, they exploit server program vulnerabilities. It is impossible for us to predict which servers would be attacked. For this reason, the OPNET simulator was used to evaluate the proposed framework. The OPNET simulator contains powerful tools for creating and testing network environments [28].

Table 1 shows the test beds used for the evaluation of the proposed framework. T1 is similar to the network at the System Software Laboratory of Soongsil University. The number of servers and honeypots in the network is 17 and 3, respectively. A network where only a redirector is deployed is denoted by (R), whereas a network where both a redirector and agents are deployed is represented by (RA). The KrCERT/CC recently announced that 45.4% of the clients used free security programs, and so it was assumed that 45.4% of the laboratory members such as graduate students had installed agents to protect their computers. For the evaluation of the proposed framework, we

Table 1 Test beds for evaluation

Test beds	Clients	Agents	Free IP addresses
*T*1(R)	62	0	160
*T*1(RA)	62	28	160
*T*2(R)	92	0	130
*T*2(RA)	92	42	130
*T*3(R)	122	0	100
*T*3(RA)	122	55	100
*T*4(R)	152	0	70
*T*4(RA)	152	69	70
*T*5(R)	182	0	40
*T*5(RA)	182	83	40
*T*6(R)	212	0	10
*T*6(RA)	212	96	10

implemented a vulnerable service program and an internet worm sample A similar to the Linux.Slapper worm which sweeps predetermined IP addresses. The worm sends the exploit code to 234 IP addresses in class C networks. In addition, we implemented a worm sample B such as Blaster worms which sweeps sequential IP addresses, and a worm sample C such as W32/Lovsan worm which sweeps random IP addresses. If the attack succeeds, malicious processes and executable files are created on the servers.

The attack scenario is as follows. An attacker finds vulnerability in a service program and tries to intrude into the server. After the attacker successfully intrudes, an internet worm is implemented, compromising many servers in a short period of time. The worm then sends malicious packets to the class C network and the attacker can break into vulnerable servers.

4.2 Evaluation

The graph in Fig. 7*a* shows the results obtained from (4) and the graph in Fig. 7*b* shows the simulation results with a worm sample A. For the DecoyPorts that are opened on agents, it was assumed that the number of DecoyPorts opened by each agent was the same as the number of ports opened by honeypots. As seen from Fig. 7, the two results are almost the same. In the network where only a redirector is deployed, the probability of luring attackers decreases exponentially with a decrease in free IP addresses. However, the probability of luring attackers decreases only slightly in the network where both a redirector and agents are deployed.

In considering a real situation in which System Software Laboratory members have built some personal servers on their own computers, Fig. 8 shows the simulation results with a worm sample A. As only a few members have built their own servers that host remote desktops and personal websites, the simulation results in Figs. 7 and 8 are almost same.

Fig. 9 shows the probability of the first intruder attacks being detected by the honeynet. Although agents may not be the optimum solution, they generally increase the detection probability. Consequently, the proposed framework can protect servers from new internet worms effectively. Finally, we run an experiment with worm samples A, B and C. Fig. 10 shows the probability of luring internet worms. As seen from Fig. 10, simulation result for several types of worms was more similar to that from (4), when compared with that from a sample worm A. Consequently, the probability of luring worms can be

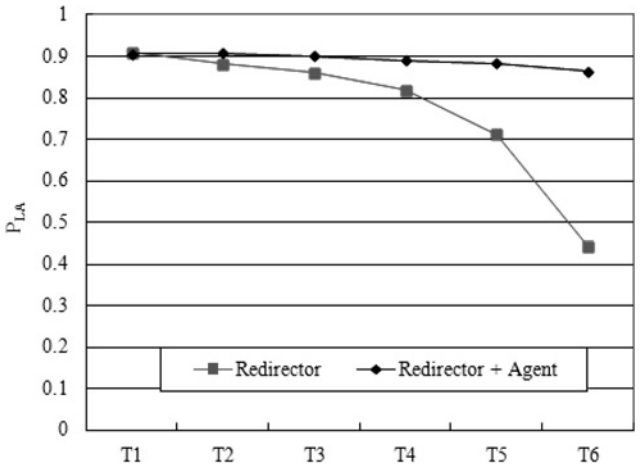

Fig. 8 *Simulation result for the network where the clients running personal services are deployed*

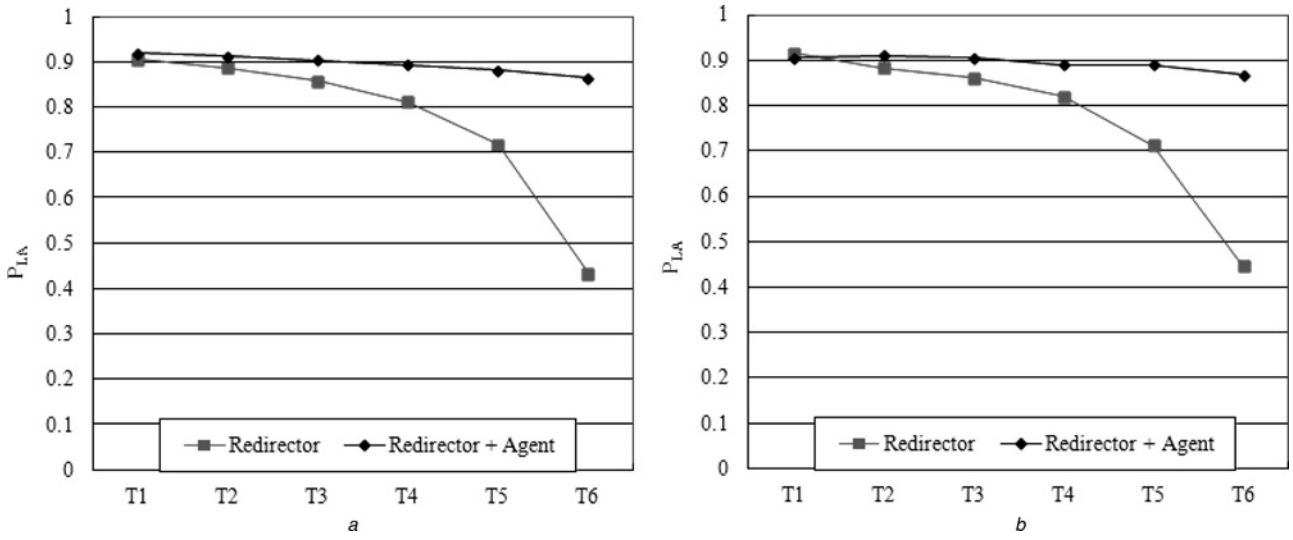

Fig. 7 *Probability of luring attackers*

a Result from (4)
b Result from simulation

doi: 10.1049/iet-ifs.2011.0154

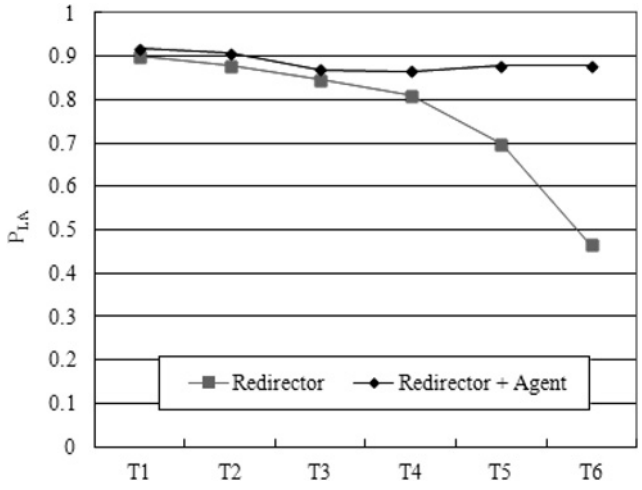

Fig. 9 *Probability of first attacks on honeypots*

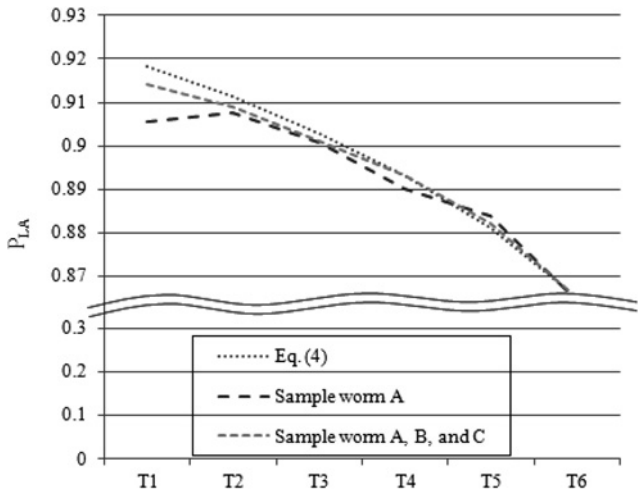

Fig. 10 *Simulation result for several types of worms*

much more similar to the result obtained from (4) when worms sweep all IP addresses or totally random IP addresses in a class C network.

Fig. 11 shows the results of simulations based on PA. In T6, when PA = 10, P_{LA} is 54% higher than when PA = 0. Moreover, P_{LA} is always higher than 80% when PA is higher than 30. When agents are deployed to all clients, the probability of luring attackers is not lowered, even though the number of free IP addresses decreases. Consequently, the proposed framework is very effective when clients are forced to install agent software.

As an agent uses the resource of a client, the number of packets forwarded by DecoyPorts should be restricted to minimise the client's system and network loads. However, if the number of forwarded packets is too small, agents may be identified by attackers. Considering the current computer and network conditions, the response time for requests should not exceed 3 s [26, 27]. In considering this response time, we accessed the main website in our campus network via an agent and measured the time it took to download the index file, as shown in Fig. 12. When an agent was placed inside the network that had a honeynet and the input value of the usleep() function did not exceed 100 000, the index file could be downloaded within 3 s. However, when an agent was placed outside the network that had a honeynet and the input value of the usleep() function exceeded 20 000, the index file could not be downloaded within 3 s. In this case, if the input value is set to 10 000, the index file could be downloaded within 3 s, but a heavy network load would be experienced by the client's system.

To evaluate an agent, one was installed on a Core Duo 1.66 GHz computer with 1 GB RAM and 100 Mbps network speed. The agent had the following DecoyPorts open: 20, 21, 22, 23, 25, 80, 110, 443 and 515. Our measurements have shown that the agent requires less than 1.0% of the CPU in a whole month when the input value of the usleep() function and the data size are set to 100 000 and 1024 bytes, respectively. However, the CPU use went up to 4.3% when the input value of the usleep() function was set to 10 000. In summary, the proposed framework was found to be suitable when agents were placed inside the network containing a honeynet and the input value of the usleep() function was set to 100 000. In order to measure the time it takes for a worm cleaner to search malicious processes and files on a server, one was installed on a Dual core 3 GHz computer with 4 GB RAM. In our measurement, the worm cleaner on average took less than 250 ms to search malicious processes and files.

The manager program generates logs when a new internet worm succeeds in attacking servers, as shown in Fig. 13. The logs include IP addresses of a victim server and attacker, and the vulnerable service port number. An administrator can manage servers that should be protected through the server list box and manage honeypots through

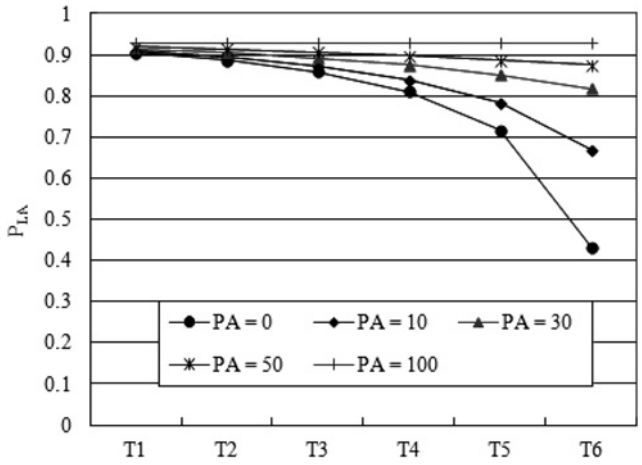

Fig. 11 *Results of PA-based simulations*

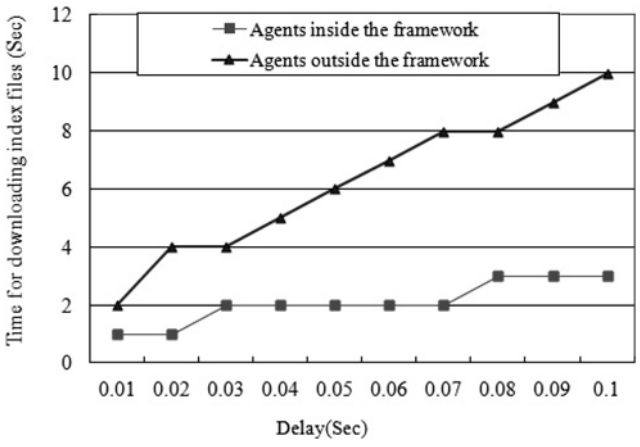

Fig. 12 *Time taken for downloading index files*

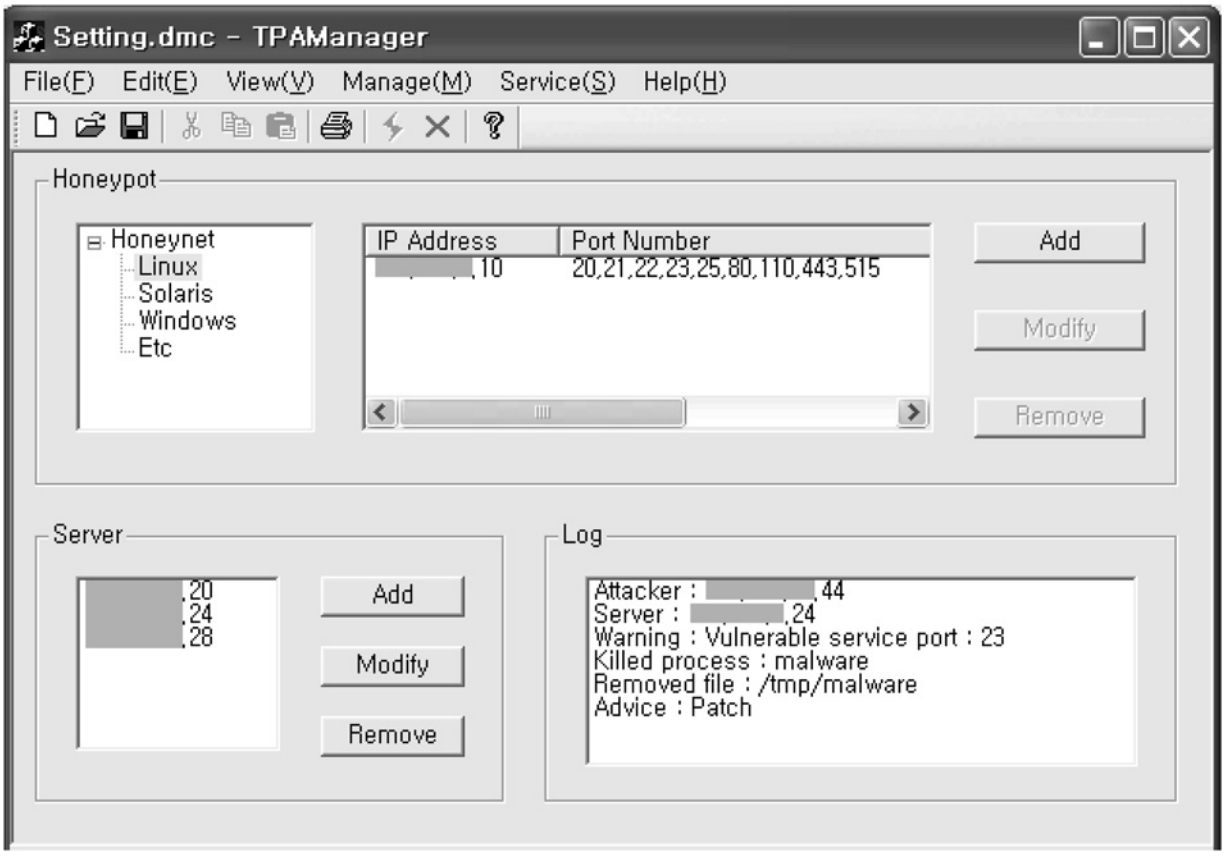

Fig. 13 *Manager program interface*

the honeypot tree view. In the list view, the IP address field and the port number indicate the honeypot's IP address and the DecoyPorts, respectively.

5 Conclusion

In this paper, we proposed an agent-based honeynet framework for protecting the servers in a campus network from attacks. Table 2 compares the advantages and disadvantages of existing systems and the proposed framework. The proposed framework can be adopted to solve the problem of honeypots being unable to lure attackers that intrude into active computers. Moreover, agents can remove malicious processes and executable files created on infected servers without using signatures, as soon as the honeynet detects new types of internet worms. An administrator can identify vulnerable service programs using manager program logs.

The probability of servers being attacked can be reduced by deploying more agents. Restricting internet use for clients with no agents installed is an effective method of forcing clients to install agents. The manager program can then allow only clients with installed agents to use the internet. In this method, when a client without an installed agent sends packets to the internet, the manager sends a TCP packet in which the RST flag has been set to the client. As a result, the client cannot use the internet. The agent program is a free worm cleaner from the viewpoint of clients. From the viewpoint of organisations, on the other hand, it is a kind of trap to lure attackers.

Table 2 Comparison between the proposed framework and existing systems

	Advantages	Disadvantages
Snort [6]	detects known worms	does not detect new worms
Yudong *et al.* [8]	detects known and unknown worms	does not detect malicious activities before a worm's self-replication
Jiang *et al.* [3]	decreases the probability of servers being attacked	needs free IP addresses is not suitable for IPv4 class *C* networks with an insufficient number of free IP addresses
Artail *et al.* [4]		
Honey@home [21]	provides large-scale threat monitoring at low cost	requires the client's resources does not protect servers from unknown internet worms
Decoyports [5]	provides large-scale threat monitoring at low cost blocks suspicious users	requires the client's resources blocks users who have computers that are infected by internet worms
proposed framework	detects known and unknown worms removes malicious processes and executable files decreases the probability of servers being attacked	requires the client's resources

IET Inf. Secur., 2012, Vol. 6, Iss. 3, pp. 202–211
doi: 10.1049/iet-ifs.2011.0154

6 References

1 Spitzner, L.: 'Honeypots: tracking hackers' (Addison-Wesley Professional, 2002, 1st edn.)
2 http://www.symantec.com/connect/articles/honeypot-farms, accessed May 2011
3 Jiang, X., Xu, D., Wang, Y.: 'Collapsar: a VM-based honeyfarm and reverse honeyfarm architecture for network attack capture and detention', *J. Parallel Distrib. Comput.*, 2006, **66**, (9), pp. 1165–1180
4 Artail, H., Safa, H., Sraj, M., Kuwatly, I., Al-Masri, Z.: 'A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks', *Comput. Secur.*, 2006, **25**, (4), pp. 274–288
5 Iksu, K., Myungho, K.: 'The decoyport: redirecting hackers to honeypots'. Proc. First Int. Conf. NBiS, Regensburg, Germany, September 2007, pp. 59–68
6 http://www.snort.org, accessed May 2011
7 Yudong, H., Donggue, P., Seungyeop, Y., Hwangbin, Y., Jongsoo, J., Jintae, O.: 'A study of the worm detection method using self-replication', *J. KICS*, 2009, **34**, (6), pp. 169–178
8 Skormin, V., Volynkin, A., Summerville, D., Moronski, J.: 'Prevention of information attacks by run-time detection of self-replication in computer codes', *J. Comput. Secur.*, 2007, **15**, (2), pp. 273–301
9 Balas, E., Viecco, C.: 'Towards a third generation data capture architecture for honeynets'. Proc. Sixth IEEE Information Assurance and Security Workshop, NY, USA, June 2005, pp. 21–28
10 Dagon, D., Qin, X., Gu, G., *et al.*: 'Honeystat: local worm detection using honeypots'. Proc. Seventh Int. Symp. on Recent Advances in Intrusion Detection, Sophia Antipolis, France, September 2004, pp. 39–58
11 Levine, J., Grizzard, J., Owen, H.: 'Using honeynets to protect large enterprise networks', *IEEE Secur. Priv.*, 2004, **2**, (6), pp. 73–75
12 Pouget, F., Dacier, M.: 'Honeypot-based forensics'. Proc. AusCERT Asia Pacific Information Technology Security Conference, Brisbane, Australia, May 2004, pp. 1–15
13 Provos, N.: 'Honeyd – a virtual honeypot daemon'. Proc. Tenth DFN-CERT Workshop, Hamburg, Germany, February 2003, pp. 1–7
14 http://old.honeynet.org/papers/honeynet, accessed May 2011
15 http://old.honeynet.org/papers/gen2, accessed May 2011
16 Hoepers, C., Steding-Jessen, K., Cordeiro, L., Chaves, M.: 'A national early warning capability based on a network of distributed honeypots'. Proc. 17th Annu. FIRST Conf. on Computer Security Incident Handling, Singapore, Singapore, June 2005, pp. 1–4
17 Pouget, F., Dacier, M., Pham, V.H.: 'Leurre.com: on the advantages of deploying a large scale distributed honeypot platform'. Proc. E-Crime and Computer Conference, Monaco, Monaco, March 2005, pp. 1–13
18 Spitzner, L.: 'The honeynet project: trapping the hackers', *IEEE Secur. Priv. Mag.*, 2003, **1**, (2), pp. 15–23
19 https://projects.honeynet.org/honeyc, accessed May 2011
20 https://projects.honeynet.org/capture-hpc, accessed May 2011
21 Antonatos, S., Anagnostakis, K., Markatos, E.: 'Honey@home: a new approach to large-scale threat monitoring'. Proc. Fifth ACM Workshop on Recurring Malcode, Alexandria, USA, November 2007, pp. 38–45
22 John, R., Peter, A., Fabian, E.: 'Vortex: enabling cooperative selective wormholing for network security systems'. Proc. Tenth Int. Conf. RAID, QLD, Australia, September 2007, pp. 317–336
23 Bowen, B., Hershkop, S., Keromytis, A., Stolfo, S.: 'Baiting inside attackers using decoy documents'. Proc. Fifth Int. ICST Conf., Athens, Greece, September 2009, pp. 51–70
24 Bowen, B., Salem, B., Hershkop, S., Keromytis, A., Stolfo, S.: 'Designing host and network sensors to mitigate the insider threat', *IEEE Secur. Priv.*, 2009, **7**, (6), pp. 22–29
25 Salem, B., Stolfo, S.: 'Decoy document deployment for effective masquerade attack detection' (Springer, 2011) (*LNCS*, **6739)**, pp. 35–54
26 http://old.honeynet.org/papers/cdrom, accessed May 2011
27 http://old.honeynet.org/papers/sebek.pdf, accessed May 2011
28 http://www.opnet.com, accessed May 2011