REVIEW ARTICLE
doi: 10.2306/scienceasia1513-1874.2013.39S.001

ScienceAsia 39S (2013): **1–5**

A review of dynamic and intelligent honeypots

Wira Zanoramy Ansiry Zakaria*, **Miss Laiha Mat Kiah**

Department of Computer Systems and Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Lembah Pantai, Kuala Lumpur, Malaysia

*Corresponding author, e-mail: wirazanoramy@gmail.com

Received 7 Jan 2013
Accepted 5 Apr 2013

ABSTRACT: A honeypot is a computer resource that is deployed in the network to attract attackers. It is designed to be attacked and misused by them. The functionality of a honeypot depends on its technical configuration made by the system administrators. To properly blend in with the environment, a honeypot must be configured to mimic the production hosts within the network environment which is very dynamic. This paper reviews research on dynamic and intelligent honeypots.

KEYWORDS: intelligent system, case-based reasoning

INTRODUCTION

Nowadays, getting connected to the internet is a very common need. With all sorts of businesses using the internet, this has made the environment a place for cyber-criminals to exercise their illegal activities. Even though a lot of effort has been made to secure the internet from cyber-attacks, loopholes for the attackers still exists. Advanced attackers or worms can often bypass firewalls and intrusion detection systems. Attack events that are happening without the consent of network administrators are a very serious issue.

Because of this, a deception based approach called a 'honeypot' is being used by network administrators to detect attacks and understand their taxonomy. A honeypot is an information system resource that is covertly deployed inside the network and purposely designed to be scanned, attacked, and compromised[1]. The deception and luring concept of a honeypot is not new in the world of IT security – it has already been around for about 20 years. Stoll's "Cuckoo's Egg" and Cheswick's "An Evening with Berferd" explain the basic concepts of honeypots[2,3]. At that time, the term 'honeypot' was not coined yet, but the idea can be clearly seen in their writings. Those in the honeypot community took their contributions as a launch pad for expanding the interest and developments related to honeypot technology.

HONEYPOT BACKGROUND

Definition

A honeypot is a computing resource deployed in the network in which it is meant to be probed, attacked or compromised by attackers[4]. It is an information resource that is designed to be scanned, attacked and compromised[2,5]. By default, a honeypot should have no interaction activities towards it. It is a resource that has no production value[4].

Any interactions detected on a honeypot will be automatically considered as malicious. The value of a honeypot lies in it being probed, scanned, or compromised. The information captured by the honeypot will be used by network administrators to overcome the attacks and to enhance the network security measures.

Honeypots can be built in many forms, either in the form of physical machines, virtual machines (VMs), or emulated virtual hosts. A physical honeypot is a real computing platform that has its own valid IP address. For example, a computer installed with Fedora Linux or Windows 7 with running network services like FTP, Telnet, or SMTP.

A honeypot VM can be built by using virtualization software like VMWare Workstation, Qemu-KVM, Virtualbox, Parallels Desktop, or User Mode Linux. By using either of these tools, honeypot VM can be created with any type of operating system. This software is installed on a computing platform and users can create single or multiple VMs running on the same host platform.

A honeypot can also be built in the form of emulated virtual hosts. By using tools like Honeyd, we can emulate thousands of virtual hosts running different types of operating system on top of a single machine. Each of these virtual hosts is configured with a certain behaviour and personality which defines how the virtual host will respond to attackers' interactions. For example, a virtual host can be programmed to contain a Perl script that is emulating a Sendmail service.

There are two types of honeypots: low-interaction

and high-interaction honeypots[1,6,7]. The use of the word 'interaction' here means the degree of interaction allowed between the honeypots and the attackers. The level of interaction defines how much damage an attacker can do towards a honeypot[6].

Low-interaction honeypots

A low-interaction honeypot has the lowest interaction capability with an attacker and it is also the simplest honeypot to set up[7]. This type of honeypot only provides minimal services and usually it is in the form of a virtual host with emulated services, e.g., a virtual host running an emulated FTP service, built by using Honeyd[7,8]. Honeyd is a popular open source low-interaction honeypot framework that offers a simple way to emulate virtual hosts on a single machine. As per writing this article, the current release version is 1.5c. Honeyd is licensed under GNU General Public License (see www.honeyd.org). The main advantage of a low-interaction honeypot is it has the lowest risk of being taken over by the attacker. This is because it only offers emulated services to the attacker. The attacker's action is limited to what is being offered by the emulation within the virtual host, and they can only scan and connect to the offered ports. Another advantage of a low-interaction honeypot is that it is easy to deploy and maintain. Its disadvantage is that the amount of information that can be collected is minimal.

High-interaction honeypots

A high-interaction honeypot is a complex honeypot solution because it offers a real operating system or a suite of real services to attackers[7]. The high-interaction capability enables a lot of information to be collected from the attackers. This information can be used to investigate the attacker's tools, motives, and identity. The ability to gather huge amounts of data is the main advantage of deploying a high-interaction honeypot. As for the disadvantages, setting up this type of honeypot is time-consuming and it is also hard to maintain[1]. Furthermore, this type of honeypot has a higher risk of being hijacked by attackers in order to launch further attacks[9]. A honeynet is an example of high-interaction honeypot. A honeynet is a network of two or more honeypots that work together in deceiving and trapping the attackers.

Other classifications of honeypots

Besides the level of interactions, honeypots can also be categorized based on their form, i.e., whether they are physical and virtual. For example, a physical honeypot can be deployed as a Windows desktop computer with attractive network services such as File Transfer Protocol, Telnet or Simple Mail Transfer Protocol. Since this form of honeypot offers a full suite of an operating system and applications for the attacker to compromise completely, it is usually classified as a high-interaction honeypot[4]. Meanwhile, a virtual honeypot is usually in the form of a VM or emulation.

Honeypots can also be classified based on their usage: production and research honeypots[2]. A production honeypot is easy to build and deploy because it is simple and requires less functionality. It directly adds value to the security of the organization, aids discovering attacks, and helps to mitigate risks. It gathers less information from the attackers than a research honeypot. A research honeypot is far more advanced in its ability to gather information about cyber-criminals. A research honeypot is very capable of analysing the attackers' tracks in order to extract important information about the attackers' identity, their origins, their tactics, and the tools they use to compromise other systems.

According to Refs. 10, 11, a honeypot can also be categorized based on its level of adaptability. From this aspect, honeypots are categorized as dynamic honeypots or static honeypots. A static honeypot is a honeypot that will always maintain the same configuration and exhibits the same appearance to the attackers all the time, regardless of any changes occurring within the network. A dynamic honeypot has the ability to change its appearance automatically based on the attack towards it or the current network landscape around it[2].

CHALLENGES IN DEPLOYING HONEYPOTS

Several issues have been identified in honeypot deployment. The main challenges are in configuring and maintaining the honeypots[11–13]. Since a honeypot is a deception tool which attracts and lures the attackers from attacking the real production hosts, its functionality and effectiveness depend on the technical configuration. Besides that, the honeypot appearance and behaviour also depend on the configuration. Any error in configuring it can lead to missed detection or that the honeypot could fail in trapping the attackers. Advanced attackers could easily detect the presence of the honeypot. Even worse, the attacker could use this poorly configured honeypot to launch further attacks on the network.

Configuration issues include what type of operating system the honeypot will be running, how many TCP and UDP ports to offer, which network services to emulate, which IP address the honeypot will be monitoring, what the behaviour of the honeypot to-

Table 1 Feature comparison of static and dynamic honeypots.

Feature	Static	Dynamic
Personality and behaviour	Fixed	Changes automatically based on condition
Adaptability	None	Adapt and blend with current environment
Human effort	Need to manually re-configure	None
Knowledge	Prior skill and experience	None

wards the attacker is, and whether it should respond to the attacker's interactions or be passive [11, 13, 14].

STATIC HONEYPOTS VERSUS DYNAMIC HONEYPOTS

Basically, honeypots are manually configured by system administrators. The configuration remains static throughout the deployment period, until there is a new configuration from the administrator. A static honeypot's behaviour and IP address are fixed. It is easier to deploy a static honeypot since we do not need to maintain or update its behaviour and configurations [15].

The downside of a statically configured honeypot is that we might miss the attacks (some or all of them) and the existence of the honeypot is more likely to be detected by the attackers. A honeypot that is unable to blend with the current production hosts within the network will be become visible to the attackers. Even worse, the detected static honeypot could be manipulated by the attackers to attack the production host within the network [16].

In contrast to the static honeypot is the dynamic honeypot. A dynamic honeypot is simply a honeypot that has the capability to manage and adapt by itself. At the time of this study, we found that only a few researchers have worked on this idea of dynamic and intelligent honeypot that has the capability to manage and adapt by itself. It is like a fire-and-forget solution, in which the network administrator does not need to monitor or manually configure the honeypot in a timely manner. Since less effort is needed to deploy such a honeypot, the honeypot configuration is less prone to error. The most desirable feature is that this type of honeypot actively adapts to its current network environment. Table 1 compares the features, strengths, and weaknesses of static and dynamic honeypots.

The behaviour and personality of a static honeypot remain the same no matter what changes occur within the network environment. The honeypot does not have the capability to nicely blend with the situation of its network environment. The honeypot behaviour remains static regardless of any changes that happened on the production hosts. Because of this, the honeypot is less attractive to the attackers. It also might lose valuable information from the attackers, such as their tactics and tricks.

RELATED WORK ON DYNAMIC AND INTELLIGENT HONEYPOTS

A dynamic honeypot has the ability to learn about the network and then deploy honeypots to appropriately blend in with the current situation inside the network. After the deployment, the dynamic honeypots will continuously monitor the network for any changes and then it will update the current honeypots configurations based on the changes. For example, if a network has all Windows-based hosts, the dynamic honeypots will autonomously deploy Windows honeypots. If a Linux host is being added to the network, automatically Linux honeypots will be deployed. It is like some network monitoring software that can be simply installed in a server, continuously gathers information about available production hosts, and it deploys honeypots with suitable configurations and behaviour.

Basically, a dynamic honeypot consists of three main modules: information gathering, configuration engine, and honeypot deployment. The function for information gathering module is to collect information from production hosts. For instance, the information collected is the operating system type, version, patch version, open ports and uptime. Later, this information will be used by the configuration engine module in order to build a honeypot personality, complete with its settings and behaviour. The honeypot deployment module will use this configured personality to deploy honeypots in the network.

The idea of dynamic and intelligent honeypot has triggered many researchers from around the world to take up the challenge to develop dynamic honeypot systems. For instance, some work has been done by using available open source tools [8, 10, 14, 17, 18].

Liu et al designed a dynamic honeypot system with a different approach [8]. Instead of using either an active or passive fingerprint technique, they used both. They applied an active fingerprint technique with a

combination of passive techniques. This is to ensure that the remote OS detection result is as accurate as possible.

Liu et al and Kuwatly et al approached this problem by using operating system (OS) fingerprint tools and a low-interaction honeypot framework[8,14]. The tools that they used are almost the same. Nmap, p0f and Snort are used as the OS fingerprint tools. In a computer network, we can use this OS fingerprint tool in order to determine the personality of the remote host. Information such as its OS type, version, patch number, list of open ports, host uptime, and host location can be gathered from such tools.

The open source honeypot framework, Honeyd, is used to deploy a low-interaction honeypot. The OS fingerprint tool gathers information from production hosts and the information will be used to build a honeypot configuration script which deploys virtual honeypots in the network. The personality, behaviour, and location of this virtual honeypot are determined by the honeypot configuration script.

Leita et al have developed an add-on tool for Honeyd called ScriptGen[17]. It is designed to be completely autonomous and allows the emulation of any protocol of any kind without any knowledge about it. This tool has the ability to automatically generate Honeyd configuration scripts. They analysed the quality of the generated scripts by launching known attacks on the machines that run the generated scripts. Besides that, they also deployed the same machine on the internet by putting it next to a high-interaction honeypot for about two months. Their approach in doing this is much more complicated because it involves the skills and knowledge of building state machines.

Hecker et al built a Honeyd Configuration Manager in Perl[18]. This module actively scans the network using the active OS fingerprinting tool Nmap. Even though this approach does disturb the network with bandwidth consumption, Nmap can detect the state of ports from a remote host faster than the passive OS fingerprinting tool. The module triggers Nmap to use SYN stealth scan, RPC scan, UDP scan, and OS detection towards the remote hosts. The IP address assignment is not done automatically. In their approach, the administrators still need to configure the generated Honeyd script.

There is some research that implemented the idea of a catering honeypots framework[10]. It has the capability of managing the personality of each honeypot by offering the services that the attackers are currently aiming for. The honeypot framework constantly monitors the network traffic, identifies services that are currently attractive to the attackers, and deploys honeypots that run such services. These services are meant to be the bait to attract the attackers to interact with the honeypots. During the real implementation of BAIT-TRAP, they managed to capture a number of trendy attack incidents.

There are researchers who implemented artificial intelligence (AI) techniques to build dynamic and adaptive honeypots. For example, Chowdhary et al approached this problem with a modification of data-mining, which they called service mining[9]. By using this approach, the honeypots are able to learn about the behaviour of the running services inside the network. The behaviour is then extracted from the interactions that are being produced by the services. Later, an emulated version of the service is created based on the learned behaviour. These emulated services behave like the real services and are run on a honeypot to distract the attackers from attacking the real services.

Wagener et al approached this problem by using reinforcement learning in order to make the honeypot capable of dynamically changing its behaviour based on the attacker's interaction[11]. Apart from luring the attackers towards the honeypot, their implementation also has the ability to insult the attackers with the intention of making the attackers reveal their identity.

FUTURE RESEARCH

A dynamic and intelligent honeypot is such an interesting research problem to work on[19]. At present, only two pieces of research have been done which used an AI approach to build a dynamic honeypot framework. AI can solve problems efficiently in many areas such as medical information systems, economics, real-time control, engineering, prediction systems, and biometrics. AI approaches such as case-based reasoning (CBR), expert system (ES), fuzzy logic, hybrid expert system, intelligent agent and swarm intelligence are good candidates for building intelligent honeypots that can learn and adapt to their environment. For example, by implementing the techniques in CBR, we can build a honeypot that is able to configure and maintain itself through past experiences[19]. By combining ES and fuzzy logic, this hybrid system could be used to formulate the appropriate number of honeypots to deploy and the number of ports to offer.

Acknowledgements: This study is funded by the University of Malaya Research Grant RG056/11ICT.

REFERENCES

1. Mokube I, Adams M (2007) Honeypots: concepts, approaches, and challenges. In: *Proceedings of the 45th Annual Southeast Regional Conference*, New York, pp 321–6.
2. Spitzner L (2003) *History and Definition of Honeypots*, Pearson Education, Boston.
3. Chamotra S, Bhatia JS, Kamal R, Ramani AK (2011) Deployment of a low interaction honeypot in an organizational private network. In: *International Conference on Emerging Trends in Networks and Computer Communications*, pp 130–5.
4. Provos N, Holz T (2008) *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, Addison-Wesley, Boston.
5. Chin WY, Markatos EP, Antonatos S, Ioannidis S (2009) HoneyLab: large-scale honeypot deployment and resource sharing. In: *3rd International Conference on Network and System Security*, pp 381–8.
6. Grimes RA (2005) *An Introduction to Honeypots*, Apress, Berkeley.
7. Nicomette V, Kaâniche M, Alata E, Herrb M (2011) Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. *J Comput Virol* **7**, 143–57.
8. Liu X, Peng L, Li C (2011) The dynamic honeypot design and implementation based on Honeyd. In: Lin S, Huang X (eds) *Advances in Computer Science, Environment, Ecoinformatics and Education*, Springer, pp 93–8.
9. Chowdhary V, Tongaonkar A, Chiueh T (2004) Towards automatic learning of valid services for honeypots. In: *Proceedings of the 1st International Conference on Distributed Computing and Internet Technology*, New York, p 469.
10. Podhradsky AL, Casey C, Ceretti P (2012) The Bluetooth honeypot project. In: *Wireless Telecommunications Symposium*, pp 1–10.
11. Wagener G, State R, Engel T, Dulaunoy A (2011) Adaptive and self-configurable honeypots. In: *IFIP/IEEE International Symposium on Integrated Network Management*, pp 345–52.
12. Hecker C, Hay B (2013) Automated honeynet deployment for dynamic network environment, *46th Hawaii International Conference on System Sciences*, pp 4880–9.
13. Budiarto R, Samsudin A, Heong CW, Noori S (2004) Honeypots: why we need a dynamics honeypots? *International Conference on Information and Communication Technologies: From Theory to Applications*, pp 565–6.
14. Kuwatly I, Sraj M, Masri ZA, Artail H (2004) A Dynamic honeypot design for intrusion detection. In: *Proceedings of the IEEE/ACS International Conference on Pervasive Services*, Washington D.C., pp 95–104.
15. Tian L (2010) Design and implementation of a distributed intelligent network intrusion detection system. In *International Conference on Electrical and Control Engineering*, pp 683–6.
16. Sardana A, Joshi RC (2008) Autonomous dynamic honeypot routing mechanism for mitigating DDoS attacks in DMZ. In: *16th IEEE International Conference on Networks*, pp 1–7.
17. Leita C, Mermoud K, Dacier M (2005) ScriptGen: An automated script generation tool for Honeyd. In: *Proceedings of the 21st Annual Computer Security Applications Conference*, pp 203–14.
18. Hecker C, Nance KL, Hay B (2006) Dynamic honeypot construction. In: *10th Colloquium for Information Systems Security Education*, pp 95–102.
19. Zakaria WZA, Mat Kiah ML (2012) A review on artificial intelligence techniques for developing intelligent honeypot. In: *Proceedings of the 3rd International Conference on Next Generation Information Technology*, Seoul, pp 696–701.