Date of acceptance Grade

Instructor

Honeypots: attacks expected

Visa Hankala

Helsinki 2011-02-13

Seminar report

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Visa Hankala

Työn nimi — Arbetets titel — Title
Honeypots: attacks expected

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminar report	2011-02-13	10 pages

Tiivistelmä — Referat — Abstract

A honeypot is a resource which is intended for being attacked or used without authorization. It does not have any direct production value itself. However, a honeypot's real value lies in the information it collects. For instance, honeypots can be used for detecting malicious activity in networks and studying attackers' techniques and tools. A typical honeypot is a network host running a few services, but there are other types as well.

Honeypots are very varied. They are used both for research and production. In addition, honeypots can differ in terms of allowed interactivity. A honeypot can be implemented using a real computer or it can be simulated in software. Alternatively a honeypot is just a piece of data whose usage is closely monitored. Honeypots have a multitude of applications. Examples of these are securing networks and countering malware.

ACM Computing Classification System (CCS):
C.2.3 [Network Operations]: Network monitoring
K.6.5 [Security and Protection]

Avainsanat — Nyckelord — Keywords
honeyclient, honeypot, honeytoken, network decoy

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Computer security is a constantly changing field. New methods for making attacks are discovered all the time. Because of this, a merely defensive approach to security is rather ineffective, although it might be good in repelling the old and well-known attacks. Proactive defences can work even against unknown security threats. Honeypots are one form of these defences.

A *honeypot* is a resource which is intended for being attacked or used without authorization [Spi02]. In general, a honeypot does not have production value. Hence any attempt to use it ought to be suspicious. A honeypot can be a closely-monitored machine or a computer network or a specialised piece of software. In addition, there are honeypots which are not distinct computer systems at all. Honeypots are very versatile and flexible security tools. They can be used both in security research and in production systems.

The concept of honeypots is relatively old. First public accounts of honeypots are from the late 1980's and early 1990's, and one of the first publicly available honeypot tools, *The Deception Toolkit* [Coh98], was released in 1997 [Spi02]. Commercial honeypot products appeared in the late 1990's. The concept seems to have become a topic of rather active research in the 2000's. Early honeypots functioned mostly on the network level. Since then attackers have begun to target the application level more often, and hence recent honeypots are more application specific, too.

Due to the nature of honeypots, there are some legal issues concerning the topic. Examples of these are privacy and liability [MoA07]. By the law, the honeypot's owner might have only limited ability to monitor attacker's communication with the honeypot. Secondly, a compromised honeypot can be used in further attacks, which might cause lawsuits against the honeypot's owner. However, legal issues are not discussed further in this text.

The rest of this text is organized as follows: Section 2 discusses honeypots in general. Implementation of honeypots and issues related to it are outlined in section 3. Section 4 presents some applications of honeypots. Section 5 concludes the discussion.

2 Honeypots in general

The idea of honeypots is very general. Essentially, a honeypot is something which seems worth attacking. There is a variety of possible applications that in turn have differing

technical requirements. Traditionally a honeypot is a network-connected computer running some services. However, there are alternative forms of honeypots as well.

Two general types of honeypots are *production honeypots* and *research honeypots* [Spi02]. Production honeypots help to make actual production systems more secure. Typical uses of production honeypots are, for instance, monitoring of computer networks and prevention of spam. Since a honeypot is just a decoy and does not have production value in itself, any attempt to use it is essentially questionable [Pro04]. Research honeypots are used for learning various details about attackers. Examples of these are how computer systems are actually attacked against and how the systems are exploited. In addition, a research honeypot can give a view to some of attackers' tools such as rootkits.

Besides of classifying honeypots according to their use, honeypots can be categorized based on the level of allowed interaction. These levels are *low-interaction*, *medium-interaction* and *high-interaction* [Spi02]. Low-interaction honeypots have rather restricted functionality. Essentially, a low-interaction honeypot can simulate some simplified services, perhaps including a network stack. An attacker is not allowed to alter the simulated service, however. A medium-interaction honeypot carries the simulation somewhat further, allowing more involved attacks. For example, a medium-interaction HTTP honeypot might pretend to be exploitable via some specific vulnerabilities. A high-interaction honeypot allows the greatest degree of interactivity. Basically it consist of a real operating system with real application software.

Typical honeypots are network hosts running some services. However, a honeypot does not need to be either a service or a computer. The essence of honeypots can be extended further. Examples of this are *honeyclients* [Wan05] and *honeytokens* [Spi03]. A honeyclient is a piece of client software which is designed to be exploitable. Unlike service-like honeypots that are passively waiting for connections, a honeyclient initiates connections. This allows examining malicious services, such as Web pages exploited with malware. A honeytoken is a special entity, such as a fake database record or a fake credit-card number. The entity is monitored closely, and any attempt to use it is recorded. Honeytokens can be used for detecting data leaks, for example.

Honeypots have some advantages over other security tools. Among those are relatively small data sets and modest demand for resources and discovery of new tools and tactics [MoA07]. Since a honeypot should not have any actual production value and it captures only requests destined to it, generated data sets ought to be rather small and the demand for resources is low. A honeypot can capture new tools and tactics as it can record every interaction.

Among the disadvantages of honeypots are limited vision, possibility of detection and risk of takeover [MoA07]. A honeypot captures only requests destined to it, and hence it can not monitor activity in other parts of a system. An attacker might be able to detect a honeypot, which makes it possible to avoid it. Mistakes in a honeypot's implementation might allow an attacker to take over the honeypot. Then, the honeypot could be used for, for instance, to attack other parts of the system.

3 Implementation

A honeypot's implementation has essentially two objectives. The first one is to seem plausible to attackers. The honeypot should look like it had some real value. Besides, it should not be easy to detect it. The other objective is to collect information from the honeypot. Without this the honeypot is more or less useless.

Honeypots can be divided into *physical honeypots* and *virtual honeypots* according to their implementation. Physical honeypots are covered in subsection 3.1, whereas subsection 3.2 discusses virtual honeypots. As a case study, *Honeyd* [Pro04] by Niels Provos is presented in subsection 3.3. Honeyd is a software framework for implementing virtual honeypots and virtual networks of honeypots.

3.1 Physical honeypots

A *physical honeypot* is a real computer with a complete software stack. The computer is connected into a network and has a dedicated network address. A physical honeypot is presumably the most plausible honeypot as almost everything is authentic and the environment does not have special restrictions. This allows practically the same level of interactivity as a real production system. However, outbound network connections are typically restricted and carefully monitored so that the honeypot can not be used to launch further attacks.

Physical honeypots are relatively expensive and very time-consuming to maintain [Spi02]. Firstly, the honeypot requires a dedicated physical machine. A complete network of physical honeypots requires networking equipment, too. Secondly, monitoring and analysis are somewhat difficult. Monitoring probes have to be hidden so that an attacker can not detect them. Besides, a successful exploit can affect almost the whole software stack. Lastly, physical honeypots entail relatively high risk since there is real possibility for a takeover.

Although the concept of physical honeypots is straightforward, actually running one is presumably too complicated or expensive for most cases. Perhaps in a research setting the versatility of a physical system would be necessary. When the honeypot received only a little traffic or the task at hand did not demand for a complete software stack, a lighter approach to honeypot implementation would suit better.

3.2 Virtual honeypots

A *virtual honeypot* simulates the honeypot system in software. This has various advantages over a physical honeypot. A virtual honeypot is easier and safer to operate since only the necessary functionality needs to be implemented [BKH06]. In addition, simulation allows implementing even complex networks of honeypots with relatively few resources [Pro04].

Because a physical honeypot runs a real operating system, it is almost always a high-interaction honeypot. Virtual honeypots are more varied in terms of interactivity. A very simple low-interaction honeypot could consist of just a dummy service. A more complicated honeypot could implement a virtual network stack and allow running multiple services. A high-interaction honeypot could be implemented with a virtual machine and a real operating system.

Virtual honeypots tend to be easier to monitor than physical honeypots. A virtual honeypot can be designed from the start to log every interaction. Although a honeypot based on a virtual machine is rather similar to its physical counterpart, the virtual machine itself can enforce monitoring. This allows capturing information even of attempts to exploit the actual operating system.

A problem with virtual honeypots is how to make them undetectable to the attacker. It is possible, for instance, to remotely figure out or *fingerprint* which operating system a remote computer is running. Fingerprinting is done by tracking specific details of how the remote network stack behaves [Fyo98]. Examples of these are how initial sequence numbers for TCP connections are generated, how the stack responds to specific IP flags, and whether there are any particular mistakes in protocols. Similar variations tend to exist on the level of applications, too. If simulation is used instead of real software, a virtual honeypot has to carefully imitate the behaviour of the chosen operating system and applications so that it will not seem suspicious.

3.3 Honeyd

Honeyd is a framework for creating virtual honeypots. It operates in the network level and can simulate various TCP and UDP services. Thereby Honeyd is a low-interaction honeypot. The framework can simulate both individual network hosts and complete networks. The following treatment in this section is based on Niels Provos' article [Pro04].

The main components of Honeyd are a *packet dispatcher*, *protocol handlers*, a *personality engine* and a *routing component*. A *configuration database* specifies how the other components operate. It describes a virtual network topology and contains a set of *templates*. A template is a specification for a honeypot. Templates are bound to network addresses in order to actually create virtual honeypots.

Incoming packets from the network are received by the packet dispatcher. The packet dispatcher handles TCP, UDP and ICMP protocols. Packets of other protocols are logged and discarded. The dispatcher delivers packets to specific protocol handlers according to the configuration database.

TCP and UDP services are implemented by external applications. Honeyd keeps track of TCP connections and delivers TCP segments for an existing connection to the same handler application. UDP datagrams are passed directly to the corresponding handler. Alternatively, Honeyd can function as a proxy, and mediate TCP connections and UDP datagrams to other network hosts. Honeyd handles ICMP requests by itself. For example, it can send responses to ICMP echo requests. The details of the packet processing, such as generation of ICMP error messages, are controlled by the configuration database.

The personality engine alters outgoing packets in order to mislead fingerprinting tools, such as *Nmap* [Fyo98], about the honeypot's operating system. As the modifications are controlled by the software, a single Honeyd instance can simulate multiple virtual honeypots that seem to run different operating systems. The personality engine uses information from Nmap's fingerprint database among other sources to imitate the desired network stack behaviour. In a way, fingerprinting is done in reverse in the engine.

With the help of the routing component, Honeyd can simulate complete networks of hosts. Incoming packets' route to their destination host is simulated by artificially adding latency and losing packets according to a given likelihood. Besides, Honeyd's routing logic throttles traffic to enforce bandwidth and decrements packets' hop limit value. In order to allow attackers to probe the virtual network, Honeyd generates proper ICMP error messages when a packet's hop limit reaches zero, for example. Similar processing is done for outgoing packets. The overall routing topology is defined in the configuration database.

Operation of a Honeyd installation can be monitored with logfiles. Various events, such as established connections, proxied connections and unknown protocols are written into a logfile. Logs from service applications are collected by Honeyd, as well.

The contents of a configuration database are described in a configuration file. In listing 1, a sample of such a file is shown. The configuration specifies a virtual network with two honeypots. The network is specified on lines 1 and 2. Lines 4–8 and 10–16 define the honeypot templates, whereas lines 18 and 19 bind the templates to specific network addresses, creating the actual honeypots.

In this example, the network's gateway is a honeypot which imitates the network stack of FreeBSD 7.2. By default, the honeypot sends a RST segment to TCP connection attemps. SSH port 22 is open and connections to it are handled by a script. UDP datagrams are responded with the ICMP port unreachable error. The other honeypot's personality is Linux. The honeypot drops incoming TCP and UDP packets by default. However, SMTP port 25 and HTTP port 80 are open and connections to them are handled by scripts. UDP datagrams to port 111 generate an ICMP error message, perhaps giving a signal of a running Sun RPC port mapper service.

```
 1  route entry 10.0.0.1 network 10.0.0.0/16
 2  route 10.0.0.1 link 10.0.1.0/24

 4  create bsdrouter
 5  set bsdrouter personality "FreeBSD 7.2-RELEASE"
 6  set bsdrouter default tcp action reset
 7  set bsdrouter default udp action reset
 8  add bsdrouter tcp port 22 "sh scripts/ssh.sh $ipsrc"

10  create lnxhost
11  set lnxhost personality "Linux 2.6.18"
12  set lnxhost default tcp action block
13  set lnxhost default udp action block
14  add lnxhost tcp port 25 "sh scripts/smtp.sh"
15  add lnxhost tcp port 80 "sh scripts/http.sh"
16  add lnxhost udp port 111 reset

18  bind 10.0.0.1 bsdrouter
19  bind 10.0.1.42 lnxhost
```

Listing 1: Sample configuration for Honeyd

4 Applications

In this section a few examples of how to use honeypots are presented. Network decoys used for confusing attackers are discussed in subsection 4.1. In subsection 4.2, a few

methods to prevent spam are covered. These two cases are relatively traditional, whereas the following ones are a little more recent. In subsection 4.3, it is presented how honeypots can automatically collect malware samples. Finally, detection of malicious Web content using honeypots is discussed in subsection 4.4.

4.1 Network decoys

Honeypots are useful for monitoring networks [Pro04]. For monitoring, honeypots are deployed in such parts of a network that are not used for production. When an attacker probes the network, some traffic should eventually hit one of the honeypots. As normal traffic should not arrive at honeypots, warnings are rather reliable. However, honeypots are useless if the attacker is aware of them. Neither can they detect the absence of attacks.

Besides of network monitoring, honeypots can be used for confusing attackers by implementing decoy systems [Pro04]. The attacker might not be able to tell which systems have real value and which do not. Because of this, the attacker may have to work harder and use more time targeting the system. This makes detection easier. Nevertheless, the setup of plausible decoys can be rather tedious, and they involve risk, as well.

4.2 Prevention of spam

Spammers abuse *open mail relays* and *open proxies* to hide their identity [Pro04]. An open mail relay accepts any sender without authentication to send mail further. Open proxies accept any client in the network to make connections through it. Honeypots masquerading as open mail relays or open proxies can be used to capture spam and reveal its sources. Captured spam makes it possible to improve filtering. Knowing a source of spam might allow switching off the spammer from the network.

Alternatively, a honeypot can collect source addresses of attempted mail deliveries. The addresses are temporarily added into the actual mail server's blacklist. This helps to filter out sources that almost certainly try to send spam.

Honeypots seem to have been effective to some extent since spammers have developed methods to detect false open proxies [Kra04]. A simple test is to try to send mail back to itself via the proxy. The proxy is very likely a honeypot if it claims a success, but in reality the message has not come back. The test is relatively simple to counter, however. The honeypot has only to compare the source and destination addresses and let the connection through if they are the same. A more complicated test would place the sender and receiver

on different hosts. In a general setting, this is much more difficult to cope without being detected as the honeypot should not be a real open proxy.

Unfortunately, honeypots are probably less effective against spam sent using botnets than via open mail relays and open proxies. A botnet's controller is presumably carefully hidden and can not be figured out from spam delivery attempts. In addition, blacklisting attempts are not very useful either, since there are so many potential senders.

4.3 Collecting malware

A suitable honeypot can automatically collect samples of malware that spread autonomously. This allows large-scale capture of currently active malware. This in turn allows, for example, research on live data and constant refinement of intrusion detection and anti-virus software [BKH06]. Manual capture of malware would be just too slow.

The objective of a malware-collecting honeypot is essentially to download the actual malware and record the details of that event. *The Nepenthes platform* is a low-interaction honeypot which achieves this in the following way [BKH06]. The platform emulates a set of known vulnerabilities that are remotely exploitable. When a network connection might lead to an exploit, the honeypot captures the connection's payload. It is then analysed whether the payload contains machine executable code or network addresses. If enough information is found, the honeypot downloads the possible malware.

Low-interaction honeypots can, at least in principle, capture only malware that exploit known vulnerabilities since they rely on emulation. More comprehensive capture requires a high-interaction honeypot which runs a real operating system. The Nepenthes platform is able to switch a network connection from an emulated target to a real honeypot if it detects unexpected behaviour [BKH06]. This makes the platform efficient as it has to resort to high interactivity only when necessary.

4.4 Detection of malicious Web content

Vulnerabilities in Web browsers might allow malicious Web pages to install malware into the system. Exploited pages are rather common nowadays, and thus their manual detection and analysis is not practical [WBJ06]. Client honeypots can automate detection at least partially and help out in analysis.

HoneyMonkey is a high-interaction client honeypot for detecting exploits [WBJ06]. The system consists of a set of Windows XP instances with different levels of patches running

in virtual machines. The system is given a list of URLs that a modified Web browser within a virtual machine visits one by one. Between the URL visits, the state of the system, files and registry, is checked. If there were any modifications outside the browser's working area, the URL would be reported as an exploit and marked for further analysis. In that case, the exploited virtual machine instance is discarded and a clean one is started.

5 Conclusion

The essence of honeypots is to be attacked or used without authorization. A honeypot should not have actual production value in itself. The real value lies in the information that is gained from misuse attempts.

Honeypots are very varied both in their field of application and type. Two main application areas are research and production. Research honeypots allow researchers to study and learn techniques used by attackers. As an example, honeypots can collect autonomously spreading malware and rootkits. Production honeypots are used for securing existing systems. Examples of this are network decoys and prevention of spam.

Honeypots can be categorized based on their interactivity. Low-interaction honeypots provide only a relatively simple ground for attacks, whereas high-interaction honeypots consist of real software allowing complex and even unknown attacks and exploits. Medium-interaction honeypots allow richer interaction than their low-interaction counterparts. However, they do not match a complete system. Traditional honeypots imitate servers, but there are alternate forms as well, such as honeyclients and honeytokens.

Two main objectives of a honeypot implementation are plausibility and monitoring. The honeypot has to seem valuable to attackers, but the true nature of the system must remain undetectable. The value of a honeypot lies in the information it captures. Hence monitoring is essential.

A honeypot can be either physical or virtual. A physical honeypot is a real machine with a dedicated network address. Typically physical honeypots run a more-or-less complete software stack. Because of this, they are highly interactive. Virtual honeypots are either specialized virtual machine installations or simulator applications. Simple virtual honeypots have only limited degree of interactivity whereas more complex ones are practically the same as real software from the attacker's point of view. A simulated honeypot has to imitate carefully the behaviour of the target system in order to avoid detection. Honeyd was presented as a case study of honeypot implementation.

References

BKH06 Baecher, P., Koetter, M., Holz, T., Dornseif, M. and Freiling, F., The Nepenthes platform: an efficient approach to collect malware. In *Recent Advances in Intrusion Detection*, Zamboni, D. and Kruegel, C., editors, volume 4219 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pages 165–184.

Coh98 Cohen, F., The Deception Toolkit, 1998. `http://all.net/dtk/dtk.html`. [2011-02-13]

Fyo98 Fyodor, Remote OS detection via TCP/IP Stack FingerPrinting, 1998. `http://www.insecure.org/nmap/nmap-fingerprinting-article.html`. [2011-02-13]

Kra04 Krawetz, N., Anti-honeypot technology. *IEEE Security & Privacy*, 2,1(2004), pages 76–79.

MoA07 Mokube, I. and Adams, M., Honeypots: concepts, approaches and challenges. *Proceedings of the 45th Annual Southeast Regional Conference*. ACM, 2007, pages 321–326.

Pro04 Provos, N., A virtual honeypot framework. *Proceedings of the 13th USENIX Security Symposium*. USENIX, 2004, pages 1–14.

Spi02 Spitzner, L., *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

Spi03 Spitzner, L., Honeytokens: The Other Honeypot, 2003. `http://www.securityfocus.com/infocus/1713`. [2011-02-13]

Wan05 Wang, K., Using honeyclients to detect new attacks, RECON Conference, 2005.

WBJ06 Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S. and King, S. T., Automated web patrol with strider HoneyMonkeys: finding web sites that exploit browser vulnerabilities. *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA*. The Internet Society, 2006.