Honeypots

Mathias Gibbens Harsha vardhan Rajendran

1 Introduction

The primary goal of computer security is to defend computers against attacks launched by malicious users. The are a number of ways in which researchers and developers can work to protect the software that they write. Some are proactive, like code reviews and regression testing, while others are reactive, like the pwn2own contest where new vulnerabilities are used to exploit browsers. Some tools can take on aspects of both; one class of these tools are honeypots.

A honeypot is a computer which has been configured to some extent to seem normal to an attacker, but actually logs and observes what the attacker does. Thanks to these modifications, accurate information about various types of attacks can be recorded. The term *honeypot* was first presented by Lance Spitzner in 1999 in a paper titled *To Build a Honeypot* [20].

Honeypots are unique because they allow a security researcher to see and record what actions a malicious user takes on a compromised computer without necessarily interfering or revealing to the attacker that they are being monitored. Because of this invisibility, valuable intelligence can be gathered about the actual strategies of an attacker. A honeypot can be configured to be either proactive or reactive to attacks, depending on the needs of the person who set it up.

Figure 1 illustrates the role of honeypots in a typical network set up. To an outside attacker (Evan and Eve), the honeypots are indistinguishable from the actual production servers. Thus, they will not behave any differently when attacking them. However, the network security team can monitor the honeypots for recorded attacks and later analyze them. The honeypots can also help to absorb attacks directed against the real servers.

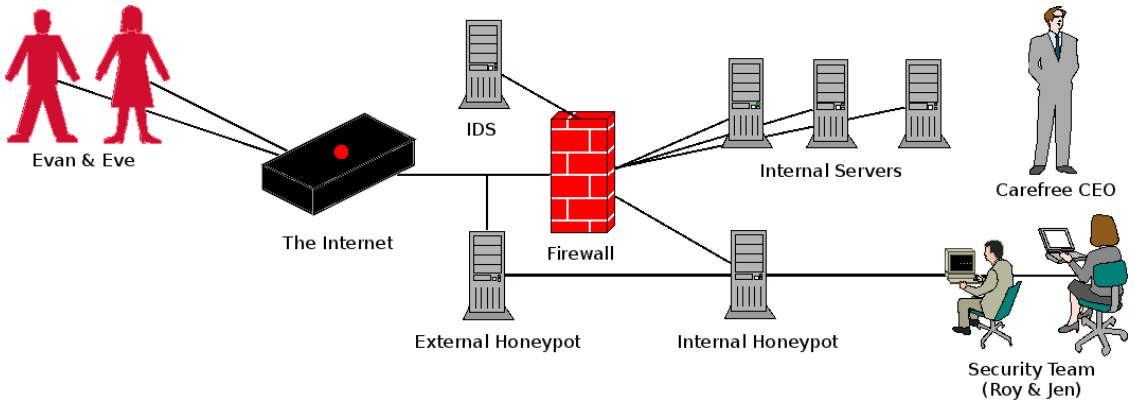

Figure 1: The key characters in our drama.

1.1 History

The term *honeypot* was inspired by actual real-life honeypots. Since such a pot contains something desirable (the honey) to someone (a child or a nest of ants, for example), it could be used to lure them out and then observe them. The same is true for a computer honeypot: a tempting target is presented to an attacker, who then comes out and performs his attacks. Figure 2 shows some of the major honeypot development milestones.

While Spitzner was the first to introduce the term *honeypot*, the concept of trying to figure out how attacks are performed has been around since at least the mid-1980's [19]. An example from the early days of Internet history was recounted by Bill Cheswick regarding his time at AT&T Bell Laboratories in January of 1991 [4]:

> On Sunday evening, January 20, I was riveted to CNN like most people. A CNN bureau chief in Jerusalem was casting about for a gas mask. I was quite annoyed when my terminal announced a security event:
>
> `22:33 finger attempt on berferd`
>
> A couple of minutes later someone used the debug command to submit commands to be executed as root – he wanted our mailer to change our password file!

Cheswick was able to see the commands this remote attacker was issuing, and then fed back modified responses to him. After a day or so, Cheswick and some colleagues worked on setting up a chroot environment for their attacker to play in:

> I wanted to watch the cracker's keystrokes, to trace him, learn his techniques, and warn his victims. The best solution was to lure him to a sacrificial machine and tap the connection. ... We constructed such a chroot "Jail" (or "roach motel") and rigged up logged connections to it through our firewall machine. ... A little later Berferd [the attacker] discovered the Jail and rattled around in it. He looked for a number of programs that we later learned contained his favorite security holes. To us the Jail was not very convincing, but Berferd seemed to shrug it off as part of the strangeness of our gateway. Berferd spent a lot of time in our Jail.

The attacker was observed for several months, until Cheswick finally shut it down. During that same time, this attacker was also active in attacking several other computer networks, and some of the administrators coordinated their information in an attempt to identify the source of the attacks. At the end, all that was known for sure was that the attack was coming from Sweden, and that the attacker had a knack for subverting the system he was on. You most certainly did not want to give an account to this attacker!

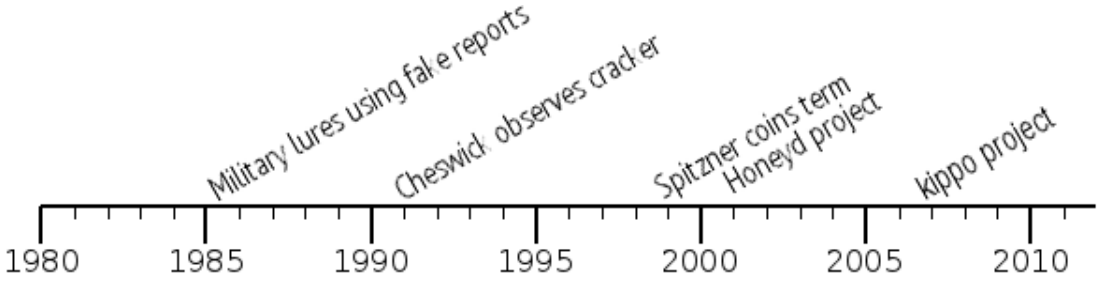

Figure 2: Honeypot development milestones.

1.2 Classification

There are several possible ways to classify honeypots. Some of the more popular are by the level of interaction available to the attacker, the type of data collected, and the type of system configuration [2, 19, 27].

1.2.1 Level of interaction

The most common type of classification is based on the level of interaction which is provided to the malicious user by the honeypot. The more interactive an environment presented, the closer the honeypot becomes to the actual targets of attack, and then potentially more accurate information can be gathered. The downside is that the more realistic honeypots present greater challenges to configure and setup. There are three levels [2]:

1. **Low-interaction** One or more simple services are made available which log all communication attempts to specific services, like a web or SSH server. Usually these are just simple daemons which provide the person who configured them a passive way to monitor attack attempts. The host operating system is not vulnerable to attacks, so low-interaction honeypots are fairly safe to run, but are also unable to be used where a more complex, interactive environment is needed, like a SMTP server.

2. **Medium-interaction** Medium level honeypots begin to emulate collections of software to present a more convincing front to the attacker, but still shield the host operating system. Emulating a collection of software can become quite complex, since the emulated programs should respond the same way as their real counterparts, but must not have the same security issues, otherwise the emulation will break. Further, there are more points of attack for the malicious user, so the chance of system compromise is raised.

3. **High-interaction** Finally there are high-interaction honeypots. The full host operating system is presented to the attacker, along with actual instances of programs, instead of their emulations. The usual goal of high-interaction honeypots is for the attacker to gain root access on the machine, and then see what he does. Because of this goal, this level of honeypot has the highest risk, but also the highest potential for collecting information. Honeypots at this level need constant supervision, since the attacker will actually control it, and could try to use it as a jumping-off point for further attacks.

1.2.2 Type of data collection

A second way of classifying honeypots is by looking at what type of data is collected concerning an attack [19]. A honeypot can be set up to detect and record one or more types of data: events (things that happen which change something in the honeypot), attacks (attempts by a malicious user to exploit a vulnerability), and intrusions (successful attacks that penetrate the honeypot). All three types of data can provide valuable information about the malicious user, and most honeypots can provide some information from each group.

1.2.3 System configuration

Honeypots can work either alone or as a group. A group of honeypots is commonly referred to as a *honeyfarm*. Individual honeypots may be simpler to set up, but they are inherently less powerful and more prone to unintended failure than honeyfarms because they lack the load balancing abilities and redundancy of a group of cooperating servers.

1.3 Basic uses

Honeypots are typically used in one of two main fashions: as part of an organization's computer network monitoring and defense, and by security researchers who are trying to keep up with the activities of blackhats.

Production environment Honeypots deployed in a production environment serve to alert administrators to potential attacks in real time. Because of the advanced level of logging and information that is available on a honeypot, better defenses against the attacks may be able to be devised for implementation on the real servers. Production honeypots tend to be reactive in nature.

Research environment In a research environment, security analysts are trying to figure out what the next generation of attacks by malicious users will be. These honeypots can be quite dynamic, as they are adjusted and tweaked to lure attackers and respond to new attack strategies. Often a research honeypot is actively monitored by a person in real time.

2 Deception techniques using honeypots

Honeypots have shown a lot of promise in the field of digital security. A part from behaving as a system with no production value, honeypots can also be put to other ingenious uses besides just luring attackers. A few novel deception techniques are explored in this section.

2.1 Honeypots as mobile code throttlers

Viruses and worms, also referred to as *mobile code* since they can often spread with little need for manual assistance, are quite prevalent these days in cyberspace. Almost all of the virus scanners are signature based, that is, they scan for particular file signatures and report them. A new approach to prevent virus spread has been proposed, using honeypots. The approach is based on the fact that an infected machine makes more connections than regular ones. Also, it must be noted that this approach in no way prevents a system from getting infected, but it can stop the spread of the virus. In the process, we sacrifice a few machines for the common good. This approach depends on the network behavior of mobile code as opposed to signature based systems. More specifically it depends on the mobile code attempting to leave the system. Having a network of honeypots to throttle mobile code has a performance incentive and offers better security.

Working model Whenever a TCP connection is established, a three-way hand shake is first performed. In a nutshell, system A sends a SYN packet to system B. System B replies back with a SYN/ACK packet. System A again sends an ACK packet to system B. The idea is to count and limit the SYN packets sent during an infection which stops the spread of the virus.

This system works on the observation that under normal usage the majority of connections are made to recently connected destination addresses and no more than one connection per second is made to a target not recently connected. Therefore whenever a request for a connection goes out, it is checked for "newness" by comparing it with a set of recently visited hosts. This set is called the *working set* and can have up to 5 addresses. If the destination address is in the working set, the connection is immediately allowed. If the destination address is new, but the working set is not full, then this address is pushed into the working set and the connection allowed. Otherwise, the destination address is put in the delay queue. Once every second, the delay queue is processed and the SYN packet at the head of the queue is sent out. Also, this destination address is added

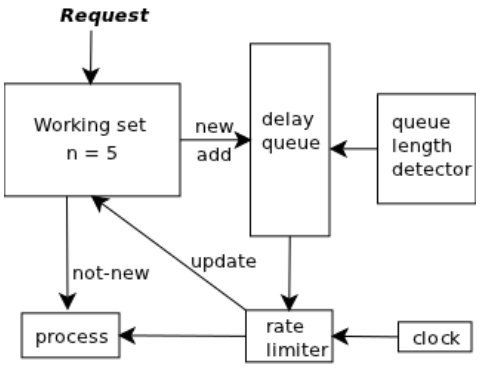

Figure 3: Control flow for mobile code throttler.

To: Chief Financial Officer
From: Security help desk
Subject: Access to financial database
Sir,
The security team has updated your access to the company's financial records. Your new login and password to the system can be found below. If you need any help or assistance, do not hesitate to contact us.
https://finances.ourcompany.com
login: cfo
password: H0n3yt0k3n

Security Help Desk

Figure 4: An example of a Honeytoken.

to the working set. In essence, connections to the five most recently used machines are immediate and new connections are delayed for 1 second. This interaction is diagrammed in Figure 3. Now, during an infection the delay queue length grows to very large values. Thus, the rate delimiter detects this deviation from the normal and throttles the rate of new connections, in turn affecting the spread of viruses. New connections are not processed and the system administrator is alerted.

This idea has it own merits and demerits. Merits include easy deployment, no legal hassles, and no dependence on virus signatures (so no updates, etc.). Demerits include a denial of service (DoS) by the system when new connections are dropped when the delay queue length crosses the threshold value. Also, this system is based on the assumption that all systems connect to recently visited hosts. This may not always be true, as in the case of an email server.

2.2 Honeytokens (cost-effective honeypots)

The greatest misconception about honeypots is that they have to be a computer. To re-iterate the definition of a honeypot is "an information system resource whose value lies in the unauthorized or illicit use of that resource." So, in essence a Honeytoken is a Honeypot which is not a computer, but some digital entity like a credit card number, Microsoft Word file, database entry, bogus login, etc. whose value lies in the unauthorized use of that resource. No one should be interacting with a honeytoken. Anyone who does is, in all probability, an attacker. Consider the following example of a honeytoken.

Assume that there's a hospital with a large patient record database going into tens of thousands of entries. Assume that there exist authorized users numbering in the thousands. Maintaining who is authorized to perform what actions is complex and might end up in giving out many false alarms. To avert such disasters, a bogus record for a celebrity or other popular figure is created. This is a honeytoken and has no value. Hence it must not be accessed by anyone. When a hacker is scanning the database for interesting information, this record will stand out. Therefore the attacker would naturally access it which should trigger an alarm.

Working model Like every other security solution present, this one is useless when used alone. Its value lies in combining it with other solutions. Honeytokens only detect illicit activity and therefore need to be combined with another solution to actually trace the attacker. Also, honeytokens can play a major role in detecting internal attackers in an organization. Consider the following example.

Assume that there exists a software firm: Reynholm Industries. The security team had a

suspicion that the mail interaction among the executives is being illegally monitored. They send out a mail like the one shown in Figure 4. An attacker who comes across this mail will naturally log into that domain with the username and password provided. What the attacker does not know is that the domain is a honeypot. Upon logging in, his movements will be monitored and he might eventually get caught.

In essence honeytokens are lightweight honeypots which are cost effective, simple to deploy and are highly effective.

3 Honeypot Implementations

When a programmer wishes to implement a honeypot, there are two main considerations: the needed complexity to be convincing and how to hide the fact that it is a honeypot from the attacker. Several honeypot projects exist, both commercial and free; [2] describes several in detail. We will highlight two popular open source honeypots: Honeyd and kippo.

3.1 Honeyd

Honeyd [17] is a low-interaction virtual honeypot that simulates virtual computer systems at the network level. It deceives the attacker by simulating the network stack of various operating systems, thus making him believe that he is interacting with a real system over the network.

Honeyd simulates only the network stack rather than the entire operating system. This ensures that even if Honeyd is compromised, the attacker cannot do much damage. Honeyd can be combined with a virtual machine (VM) to simulate multiple operating systems. To add to the realism, Honeyd also simulates arbitrary network topologies, which will further convince the hacker that he is on a real system.

Receiving network data Honeyd in essence replies to the network packets whose destination is any of the simulated honeypots. The network can be configured in several ways. For example by using proxy ARPs (answering the ARP queries of an IP address not in the current network) or network tunnels (a connection across networks). Let A be the IP address of the router and B be the IP address of the Honeyd host. Assume the case in which all IP addresses of the virtual honeypots lie in the local network. When the attacker sends a packet to a specific VM, the router scans its routing table for the forwarding address of the VM. Then it does one of three things: drops packet, forwards packet to another router or sends it directly to the destination. To direct the traffic for a VM to B, the best method is to configure routing entries for all the VMs that point to B. Then the router will directly forward the packets to the Honeyd host. If no route has been configured, then the router uses ARP to determine the MAC address of the honeypot. Normally the ARP requests would fail as all the instances are virtual, but the Honeyd host B has been configured to reply to the ARP requests for the virtual machines, using its own MAC address. Hence the router can directly send the virtual machines' packets to B's MAC address.

Honeyd comprises several components (shown in Figure 5): a configuration database, a central packet dispatcher, protocol handlers and a personality engine. Incoming packets first go through the central packet dispatcher. It is aware of three protocols, TCP, UDP and ICMP. Other packets are discarded. The dispatcher queries the configuration corresponding to the destination address. Then it passes the packet to the protocol-specific handler. On receiving a TCP or UDP packet, the handler helps establish connections to various services. The framework checks if a specific packet is part of an already started service application. If so, all packets are redirected to the

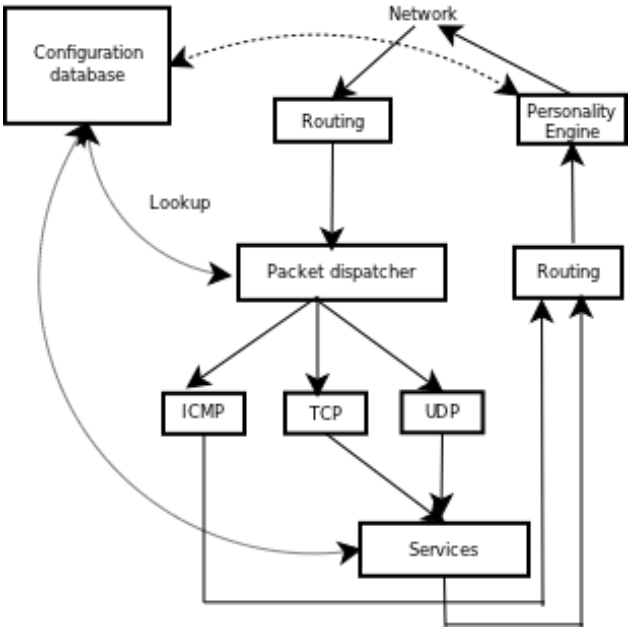

Figure 5: Honeyd architecture.

service, otherwise a new service is started. The handler also helps in redirection of connections; for example, redirecting a DNS request to an actual DNS server. Then the packet is sent to the personality engine which manipulates the packet content to make it look like it has originated from the network stack.

Personality engine The main concern that has to be dealt with while modeling virtual honeypots is that whenever an attacker runs a fingerprinting tool, they shouldn't expose the honeypot as any different from the original system, lest the attacker detects the ruse. This is done by mimicking the personality/behavior of the underlying Operating System. Honeyd simulates the corresponding network stack for various kinds of operating systems. It makes the necessary changes in the headers of every packet that leaves the system so they match the characteristics of the underlying operating system. Honeyd uses N-Map fingerprint information to modify the network stack.

Logging The framework ensures that services report data to be logged using stderr. It then uses syslog to store/maintain that information on the system.

3.2 Service-specific honeypots

In addition to honeypots like Honeyd, there are simpler honeypot options which are easier to set up and monitor. Usually these are low- or medium-interaction stand alone honeypots. A computer running one of these simpler honeypots usually runs it as a service on a specific port and isolates the underlying operating system and file systems. Due to this isolation, more than one honeypot may be running on a single system at the same time, or the computer may be used for additional purposes.

One of the most commonly attacked class of services today is remote administration, like RDP/VNC for Windows servers, and SSH for Linux servers. Because of the access granted by a compromised remote connection, they present a tempting target for an attacker. Due to the

popularity of attack, it is common to see SSH honeypots. One particular SSH honeypot implementation is called kippo [23], which mimics an actual SSH server and virtual file system using Python. It can log both connection attempts, and if the attacker is allowed access, all commands and outputs of the emulated shell are saved for playback and analysis at a later date. Because of its limited focus, kippo is easy to setup and use, but does have some limitations to what it can mimic.

Several other services can be honeypotted, such as a web server, FTP server, or DNS server. However, the idea for all these simpler honeypots remains the same: observe and capture information about the attacker.

4 Honeypot deployment strategies

After implementation and testing, the honeypot needs to be deployed into an existing network. Depending on the need, honeypots can be deployed using various strategies and in various positions on a network (Figures 1 & 6). Some honeypot deployment techniques may involve a variety of legal issues.

4.1 Sacrificial lamb

This is the simplest of all deployment techniques. As the name suggests, the honeypot just sits on the network, waiting to be pounced upon by an attacker. That is, you give the attacker what he wants and keep him busy, all the while trying to track him down. The honeypot as always has no production value, but helps buy time for the administrators to act upon. Examples of honeypots which can be deployed using the Sacrificial lamb strategy include Deception tool kit (DTK) and Specter.

4.2 Deception ports on production systems

This deployment strategy involves mimicking various services on the different ports of the system. For example HTTP would be mimicked on port 80, SMTP on port 25, etc. The idea here is quite similar to the one in the Sacrificial lamb technique. That is, you deceive the attacker in such a way that he is stuck-up trying to solve the deception, while measures like trace-back, forensics, etc. can be taken. This kind of deployment is the most common and has the least liability. Examples of honeypots which can be deployed using this strategy include Honeyd and Specter.

4.3 Proximity decoys

The main idea behind this deployment strategy is to maintain the honeypot close to the production server. If you ensure that the honeypot is in the same subnet that the main server is in, then it would technically imply that it is still on the same network. Therefore, there will not be any legal hassles, as you are entitled to use any techniques to protect your assets as long as it is within your network. Also, it will be easier to re-route any attacks aimed at the production servers or to trap them. Examples include deployment of honeypots using VMware and Usermode Linux (UML).

4.4 Redirection shield

Honeypots deployed using this strategy mainly use port re-direction or traffic re-routing. It is believed that this strategy is the most promising one of all, as it has the most commercial value. Just like Software as a Service, honeypots can be used as a service and not just a device. The

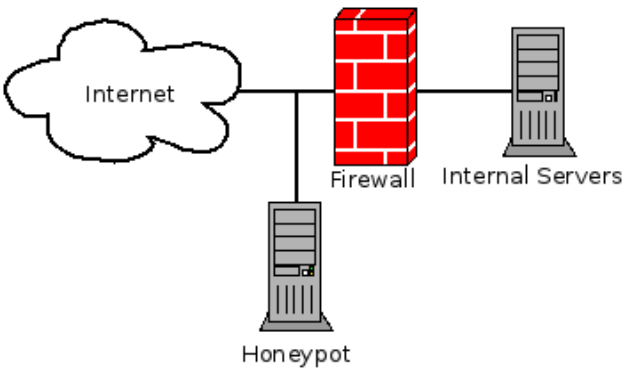

Figure 6: A example of an exposed honeypot.

strategy mainly involves installing re-routing switches at the client sites. Now the honeypots can sit anywhere in the world, and receive all the traffic flowing to the servers and track hackers.

4.5 Minefield

Honeypots deployed using the Minefield strategy are placed at the perimeter. Just as the name suggests, they explode upon contact (usually by an attacker). To ensure complete security, the intrusion detection systems (IDS) and vulnerability scanners are also placed in the network. But rather than monitoring the production servers, these IDSs and vulnerability scanners use the contents of the honeypots to sound alarms. So, the probability of producing false alarms is highly reduced. In essence these Honeypots act as a third layer of defense and they trap, deceive, trace or tarpit attackers. Examples include LaBrea and Honeyd both when used in Stealth mode and Mantrap.

5 Pros/Cons

Honeypots are a powerful tool that can be used to monitor attacks. However, they are not without fault.

Pros Chief of the advantages of honeypots is that they shield actual production servers from direct attack by emulating or visualizing services. If an attack would have penetrated a weakness, only the decoy was impacted, instead of the real server. Additionally, since honeypots can maintain logs of attacks, information can be gathered about the tactics and software used by an attacker once the system has been compromised. This lets security researchers devise defenses to be implemented on the real servers to foil hacking attempts. Finally, honeypots do not have to contain any actual data of value. If data is leaked during an attack, the administrator of the honeypot can be confident that no sensitive information was compromised.

Cons In spite of their usefulness, honeypots do have disadvantages. First, they are at best a copy of the real target. If the copy is not good enough, the information recorded or means of compromise may not closely match an actual server's response. Second, the emulation done by a honeypot must not have the same weaknesses, known or unknown, otherwise the attacker will be able to compromise the honeypot itself. Thirdly, using and deploying a honeypot takes time in addition to the time needed for setting up and maintaining the real server. Administrative staff

may not have the needed time to do this, which would prevent them from using honeypots as part of their security defenses.

6 Real life results

Honeypots are used every day as part of comprehensive security configurations to detect, deflect, and prevent security breaches. The type of deployment strategy used directs what role the honeypot plays in the system. An example of honeypots used in a minefield configuration was described by a person in Utah State University's IT department [12]. In addition to their normal SSH servers, several SSH honeypots are deployed on servers which do not have any entries in DNS pointing to them. Thus, any SSH attempts will most likely be coming from port scanners. Credentials from the attacker are collected and the offending IP address is banned. Additionally, since only automated attacks should be seen on these honeypots, USU is able to work with ISPs who own the IP address block containing the attacker's source IP to help remove malware from infected computer systems.

7 Improvements

Like any piece of software designed to target and reveal information about malicious users, there is a constant battle between the researchers trying to outsmart the attackers and vice versa. If a certain identifying characteristic of a honeypot is discovered, attackers can look for that, and if it is found, they know they are on a fake system. Developers then need to find a way to mask the identifier to make the honeypot even more similar to the targeted system or service. Additionally, honeypot implementations may contain undesirable bugs or security flaws which could open up the honeypot for use maliciously itself. Lastly, as new versions of servers and software are published, naturally hackers will target them. Honeypots need to be updated regularly to be able to continue providing a tempting target.

Most of the honeypots which are in use today concentrate on emulating parts of a server. However, there are other places in a networked environment where the idea of using a honeypot could give security professionals an advantage in watching attacks. One example is managed networking equipment (switches, routers, etc). If these devices are not properly configured or are running outdated firmware, an attacker may be able to intercept or disrupt network traffic. A honeypot representing itself as a network device could help detect how such an attack was performed.

8 Conclusion

Hopefully through this short introduction to honeypots you have been able to see the potential that they bring to the field of computer security. Using them allows researchers and security professionals to monitor malicious attackers without their knowledge, and hopefully learn about their techniques and exploits in real time. Also a few honeypot deployment schemes have been discussed, which we hope has given the reader a fair idea of how honeypots are actually used in the real world. But one important point to be noted is that usage of only honeypots to secure a system is not recommended. You have to have multiple security tiers to successfully thwart attacks. Therefore combining honeypots with other security solutions is necessary.

References

[1] Cláudia J. Barenco Abbas, L. Javier García Villalba, and Victoria López López. Implementation and attacks analysis of a honeypot. In *Proceedings of the 2007 international conference on Computational science and Its applications - Volume Part II*, ICCSA'07, pages 489 – 502, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] Reto Baumann and Christian Plattner. Honeypots, 2002.

[3] Jeremy Briffaut, Jean-Francois Lalande, and Christian Toinard. Security and results of a large-scale high-interaction honeypot. *Journal of Computers*, 4(5):395 – 404, 2009.

[4] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *In Proc. Winter USENIX Conference*, pages 163–174, 1992.

[5] Marc Dacier, Fabien Pouget, and Hervé Debar. Honeypots: practical means to validate malicious fault assumptions. In *Dependable Computing, 2004. Proceedings. 10th IEEE Pacific Rim International Symposium on*, pages 383 – 388, march 2004.

[6] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian Grizzard, John Levine, and Henry Owen. Honeystat: Local worm detection using honeypots.

[7] Maximillian Dornseif, Thorsten Holz, and Sven Müller. Honeypots and limitations of deception.

[8] Anand Gupta, S. K. Gupta, Isha Manu Ganesh, Pankhuri Gupta, Vikram Goyal, and Sangeeta Sabharwal. Opaqueness characteristic of a context honeypot system. *Information Security Journal: A Global Perspective*, 19(3):142 – 152, 2010.

[9] Amit Lakhani. Deception techniques using honeypots. Master's thesis, University of London, UK.

[10] John Levine, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. The use of honeynets to detect exploited systems across large enterprise networks. *IEEE Systems Man and Cybernetics SocietyInformation Assurance Workshop 2003*, (June):92–99, 2003.

[11] Mario Marchese, Roberto Surlinelli, and Sandro Zappatore. Monitoring unauthorized internet accesses through a 'honeypot' system. *International Journal of Communication Systems*, 24(1):75 – 93, 2011.

[12] Miles. Ssh honeypots have many benefits. http://ask.slashdot.org/comments.pl?sid=2170514&cid=36188168, Mar 2011.

[13] Iyatiti Mokube and Michele Adams. Honeypots: concepts, approaches, and challenges. In *Proceedings of the 45th annual southeast regional conference*, ACM-SE 45, pages 321–326. ACM, 2007.

[14] Anh-Quynh Nguyen and Yoshiyasu Takefuji. Towards an invisible honeypot monitoring system. In *ACISP*, volume 4058 of *Lecture Notes in Computer Science*, pages 111–122, Melbourne, Australia, 2006. Springer.

[15] Laurent Oudot. Fighting internet worms with honeypots.

[16] Van-Hau Pham and Marc Dacier. Honeypot trace forensics: The observation viewpoint matters. *Future Generation Computer Systems*, 27(5):539 – 546, 2011.

[17] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 1–14, Berkeley, CA, USA, 2004. USENIX Association.

[18] Niels Provos. Development of the honeyd honeypot. http://honeyd.org/, Feb 2012.

[19] Christian Seifert, Ian Welch, and Peter Komisarczuk. Taxonomy of honeypots, 2006.

[20] Lance Spitzner. To build a honeypot. http://www.spitzner.net/honeypot.html, Aug 1999.

[21] Lance Spitzner. Honeypots: catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179, 2003.

[22] Lance Spitzner. *Honeypots: Tracking hackers*. Pearson Education, Inc., 2003.

[23] Upi Tamminen. kippo - ssh honeypot. https://code.google.com/p/kippo/, Feb 2012.

[24] Jamie Twycross and Matthew Williamson. Implementing and testing a virus throttle.

[25] Steve Webb, James Caverlee, and Calton Pu. Social honeypots: Making friends with a spammer near you.

[26] Zaiyao Yi, Liuqing Pan, Xinmei Wang, Chen Huang, and Benxiong Huang. Ip traceback using digital watermark and honeypot. In *Ubiquitous Intelligence and Computing*, volume 5061 of *Lecture Notes in Computer Science*, pages 426–438. Springer Berlin / Heidelberg, 2008.

[27] Feng Zhang, Shijie Zhou, Zhiguang Qin, and Jinde Liu. Honeypot: A supplemented active defense system for network security.