

Problems and Challenges with Honeybots

Lance Spitzner 2004-01-14

For the past 18 months we have seen a tremendous growth in honeybot technologies. Everything from OpenSource solutions such as Honeyd and Honeynets, to commercial offerings such as KFSensor are commonly available. However, as with any relatively new technology, there are still many challenges and problems. In this paper we take an overview of what several of these problems are, and look at possible approaches on how to solve them. By identifying these problems now, we can hope to make honeybots a stronger technology for the future. The three problems we discuss below are identifying honeybots, exploiting honeybots, and attacker clientele. It is assumed you have already read and understood the concepts previously covered in [Honeybots: Definitions and Values](#).

Identifying honeybots

As we have seen in the past months, there are many types of different honeybots (both low and high interaction) that can achieve many different things (tarpping, detection, countering spam, information gathering, etc). Most of these honeybots share a common trait -- their value diminishes upon detection. Once detected, an attacker now knows which systems to avoid (your honeybot) or potentially even worse, can now feed your honeybot false or bogus information. This is why in most cases you want your honeybot to avoid detection. Some exceptions do exist, such as in deterrence. Organizations may want to be known for using honeybots or have some of them detected, as it could deter attacker from probing their networks. Or in the case of sticky honeybots stopping worms, there is little threat (at least at the moment) of the worm using honeybot detection routines as worms are too busy scanning to care about honeybots.

In most cases you want your honeybots to avoid detection. Honeybots are growing in use and we have already begun to see tools and techniques released to counter and detect them. One of the more unique examples is the commercial tool [Honeybot Hunter](#), used by the Spamming industry to identify honeybots. Here is a tool developed and released for the sole purpose of identifying Spam-catching honeybots. Other tools have been developed to [identify virtual honeybots](#), and papers have been published that [identify potential issues](#) .

So, what can be done? First, realize that no matter what type of honeybot you are dealing with, from the most basic [Back Officer Friendly](#), to the most advanced Honeynet, any honeybot can eventually be detected. While the goal is to have a honeybot that is never detected, if you have

an adversary that has the necessary skills or the proper tools and they are looking for honeypots, then its only a matter of time. In many ways, just like most other technologies such as IDS sensors, honeypots are in an arms race. As new honeypots are released, or newly updated versions appear, attackers can identify ways to detect and identify them. As these new detection methods are developed, counter detection measures can be built into the honeypots. Attackers can then counter these new measures, and the cycle continues. For example, to remotely identify older versions of the Honeyd honeypot, you merely had to send a SYN packet, as the honeypot would respond with a SYN/ACK packet that had no options. However, if you were to use Nmap to profile the same honeypot, then it would respond to SYN packets with options (this has now been corrected in Honeyd ver 0.7a).

As I see it, there are two steps you can take now to address this problem. First, decide at what point does detection diminish the value of your honeypot. If your honeypot can derive value before it is detected, then it has potentially still done its job. For example, lets say you deploy honeypots on your internal network to detect unauthorized activity (such as someone scanning for open file shares). The purpose of your honeypot is to act as a burglar alarm (as Marcus Ranum likes to call honeypots). Let's say that an attacker is on your internal network, and while probing for vulnerable systems he probes (and detects) your honeypot. In this case, even though the honeypot was identified it still has potentially done its job, detecting and alerting you to a threat. Even if it was detected minutes after being probed, the honeypot is letting you know there is a threat on your internal network, and the threat is actively looking for open file shares. For other honeypots, the story is different. For example, if you are using Honeynets to gather information, detection compromises your ability to collect accurate data. In this case, you want your honeypot to go for days, if not weeks or even months, without detection. This can be much harder to achieve. So, the first step is for you to decide just how important detection is to you, and how long your honeypot needs to remain undetected.

If in step one you determine that avoiding detection is important to you, then you want to consider customization. There are many different types of honeypot solutions you can download and work with. Just as you can easily download the solutions, so to can attackers. They can download evaluation copies or source code of anything publicly available, analyze it, and identify signatures. In many ways, its similar to how Fyodor's powerful Nmap can [remotely fingerprint](#) operating systems, as each IP stack has its own unique idiosyncrasies. To counter this, you need to customize your honeypot, change its behavior or appearance so it does not look like every other honeypot on the Internet. The more you modify the default behavior, the potentially more difficult it will become for attackers to identify it. For example, if you are using

the [Honeyd Toolkit for Linux](#), you do not want to use the default honeyd.conf files for production environments. Instead, modify the behavior of the templates to adapt to your environment. More advanced users can potentially modify the source code, so as to change how packets are created. Regardless, attackers may be looking for a specific type of known honeypot behavior. You can help minimize the chance of detection if your honeypot is behaving or reacting in ways attackers do not expect.

Exploiting honeypots

Anything coded by humans can be compromised. For years this has been true for various applications such as firewalls, web servers, or browsers. Whenever a new application has been released, we can expect bug or vulnerability reports. Honeybots are no different. We have to assume for that every honeypot released, there are known (and unknown) vulnerabilities in those systems. As with any other security technology, steps should be taken to protect against unknown attacks. With low interaction honeypots, the risk is somewhat limited as there are only emulated services for attackers to interact with, they are not given real applications to exploit, nor real operating systems to gain access to. However, we should assume that an attacker can bypass the controlled environments of emulated services, and as such everything should be done to secure the honeypot application. For Win32 low-interaction honeypots (such as [KFSensor](#)), you want to build a secure base OS with the latest patches, disabling all services. Perhaps even install a host based firewall, one that allows inbound connections to any port the honeypot is monitoring but blocks all other inbound connections. Even more importantly, have the firewall block (and alert) any outbound-initiated connections, to help protect against the threat of the honeypot when compromised. For Unix low-interaction honeypots we can take greater measures. Chroot() is one usual way to improve containment against attacked processes on a Unix system. Jail() (under FreeBSD) proposed a real way to restrict what could be seen by processes. Low level kernel patches like [Systrace](#) (Process based Discretionary Access Control) or [Grsecurity](#) (Process based Mandatory Access Control, Address Space Protection, etc) or others such as [SE Linux](#) should be used in low-interaction honeypots to help protect against known and unknown attacks.

For high-interaction honeypots, the problem is more challenging. These solutions provide real operating system and applications for attackers to interact with, as a result they have greater risk. It is expected for attackers to gain privileged control of the honeypots. This means external Data Control measures have to be put in place, such as an IPS (Intrusion Prevention System) or bandwidth limiting. In these cases (such as Honeybots) there are two steps you can

take. First, use several layers of control. This prevents having the risk of a single point of failure. The second is human intervention. High-interaction honeypots should be closely monitored. Any time there is anomalous activity on your honeypot (outbound connections, uploaded files, increased system activity, new processes, system logins, etc) a human should then be monitoring everything that happens on the system. Anytime an attacker's action exceeds your organizations threshold for risk (such as attempting an outbound attack) you can terminate the attacker's connection, drop packets, redirect connections, etc. The advantage to real time monitoring is you can potentially identify activity that automated mechanisms may miss. This also gives you far greater control over what the attacker does, and how your honeypot responds.

Attacker clientele

For me, this has been the toughest nut to crack, in part because it's just not a technical issue. One of the biggest challenges of honeypots is how can they be deployed to detect, identify, and capture the activity of specific threats, both internal and external to a company. Think of deploying honeypots as similar to fishing. Traditionally, most honeypot deployments have not been focused on a specific target, instead they have been common systems deployed on external networks. This is similar to going fishing to any local lake, throwing out a line with an ordinary worm on it, and you're happy with whatever you catch. In most cases, these 'fish' have been attackers that focus on targets of opportunity, probing and breaking into as many systems they can find, often using automated tools. These threats are relatively easy to capture with honeypots, as they are highly active, will attack anything with an IP stack, and most often don't spend the time checking to see if they are interacting with a honeypot.

Honeypots have the potential to capture bigger fish. Organizations may not be concerned about automated or common attacks, they may be more concerned about advanced attackers targeting their critical systems, or employees who are stealing and selling their confidential information. For honeypots to capture such threats, the honeypots have to be tuned for each individual threat, we need the proper bait and location. When you are fishing for 150 pound Tarpons, you don't simply throw a hook and worm in the local pond. Instead, you travel to the Florida Keys during the spring and summer. The same analogy holds true for more advanced attackers. Your honeypots have to be located in the proper location, at the proper time, and with the correct bait. For this, such honeypots have to be customized to your specific threat, a much more difficult job to do. For example, if you are concerned about organized crime breaking into your ecommerce site, throwing a default RedHat 7.3 honeypot on your external

network is most likely not going to capture their activity. If you are out to catch the latest attacks or exploits, you need high value targets, such as a CVS honeypot, that will give attackers a high ROI (Return on Investment) for their new attack. For internal threats, you need honeypots that have value to that insider, such as honeypots that appear to be research and development databases. To go after a specific threat, your honeypot has to be tuned to that individual.

Conclusions

Honeypots have tremendous potential for the security community, and they can accomplish goals few other technologies can. Like any new technology, they have some challenges to overcome. Most likely none of these problems will ever be completely solved or eliminated. However, expect to see in the next 12 to 18 months many new developments that help address these, and other issues.

Author Credit

View [more articles](#) by Lance Spitzner on SecurityFocus.

[Privacy Statement](#)

Copyright 2006, SecurityFocus