Defeating Honeypots: Network Issues, Part 1

by Laurent Oudot *and* Thorsten Holz

last updated September 28, 2004

0. Abstract

To delude attackers and improve security within large computer networks, security researchers and engineers deploy honeypots. As this growing activity becomes a new trend in the whitehat community, the blackhats study how to defeat these same security tools. Though not everyone agrees on the power of honeypots, they are effective and are being deployed as tools -- and blackhats are already working to find ways to exploit and avoid them. The cyber battle continues.

SecurityFocus™ Jobs
-Click for more info

The purpose of this paper is to explain how attackers typically behave when they attempt to identify and defeat honeypots. This is not an exhaustive description of all the tools and methods that are publicly known (or unknown), but this article will help security teams who would like to setup or harden their own lines of deception-based defense. After some theoretical considerations, we will discuss some technical examples to emphasize our explanations. This two-part paper will focus on network issues. Further papers will move to the system world and the application layer.

1. Theory

This article discusses actions lauched by attackers remotely, far away from a honeypot, as well as local actions launched on a compromised honeypot using the network layer. Beyond the scope of this article, if you're interesting in learning more technical issues from the underground regarding techniques used to defeat honeypots, you should definitely come to the next PacSec meeting in Tokyo, organized by Dragos Ruiu. [ref 0]

1.1 Remote actions

Before attackers obtain access to a honeypot, they will sometimes try to detect if there is something suspicious waiting for them. They might not want to attack a computer being used to trap them, and they might not want to be monitored because this could reveal their identity, their methods and their tools. For example, by using a 0-day exploit against a honeypot that records everything (captures network traffic, low level system activity, and more), an attacker would probably lose the valued secrecy of her techniques.

Most of the time, the remote actions used by attackers are very easy to understand. They simply try to interact with the honeypot and look at the results. This game of stimuli might exist at many layers of the OSI model, depending of the kind of honeypot targeted. Usually, for a TCP/IP based honeypot, an aggressor will initiate a network dialog and look at the results at layer 3 and 4, or even at layer 2 if she acts near or on the network segment of his target. Sometimes, layer 7 (application layer) might also be used by a remote aggressor.

1.2 Local actions

After they get access to a honeypot, for example through a shell or through some custom shellcode, attackers might still use the network layer in order to determine if they have compromised a honeypot instead of a real machine. By doing this, they might reveal their techniques used to fingerprint a honeypot through the network layer. By then, defenders who operate the honeypot will already have a record of the attacker's malicious activity as a kind of burglar alarm.

1.3 Cloaking issues

To quote Lance Spitzner, founder of the Honeynet Project, "*a honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.*" So, for example, if a potential aggressor is able to determine that a computer is a honeypot without being detected, one might believe it to be a problem. But most of the time, honeypots record almost everything so it is difficult for an attacker to be totally stealthy. By "defeating a honeypot", this paper refers to defeating the role of a honeypot, or in other words we will mainly focus on attacks that are used to detect a honeypot and/or disable parts of it (such as disabling its ability to record all activities). Note that some whitehats use honeypots as burglar alarms, so these honeypots being fingerprinted or disabled are not an inherent problem, provided the operator is indeed notified that that a malicious blackhat hacker came in.

1.4 Breaking the Matrix

Even if a honeypot is not a real computer resource on a production network and it is just

sitting there, waiting to be attacked, there are ways to determine its role. Such an activity is called fingerprinting. If you want to understand how attackers succeed in fingerprinting attempts against a honeypot, just ask yourself: what is the difference between a honeypot architecture and a real architecture?

Though this might look very easy and simple on the surface, this is the key question to consider when thinking about cloaking and honeypots. Attackers will try to evaluate if this small world attained by them is a real or fake one. Remember the Matrix movie? This is the same sort of problem, determining reality. Depending of the level of interaction and type of honeypot, the fingerprinting methods used may need to be different.

If the operator of a honeypot is trying to simulate or emulate an environment, the attacker will probably try to find a path to the truth by looking at specific differences that exist in the compromised envrionment, as compared to a real one. Imagine for example that you deploy a fake proxy service to catch spammers. Consider these questions:

- Are you sure that you will respond to any possible requests the way a real proxy service would do?
- What if the spammer sends unusual or abnormal requests?
- What if the spammer tries to use unimplemented or complex functions that should be found on your service?
- What if the spammer tries to use the proxy to test if it remains functional under heavy load, and for a long time?

As you can see, simulating reality is not so easy. In the above example about spammers, aggressors will probably use remote actions based on layer 7, such as sending bogus requests. By using protocols to their practical limits, attackers will probably find a way to fingerprint the simulation of our fake world.

One solution might be the use of a high interaction honeypot that is based on a real system. From a cloaking point of view, a perfect honeypot could be a "sacrificed" one that has a real system installed on it. But when you setup such a security resource, you want to record as many events as possible. Network captures made on the wire of your honeypot might not be enough. Think about attackers who use encrypted channels. It is widely known that some blackhats enjoy SSH sessions to compromised computers. Because of this, people have begun to manipulate the kernel of the operating system in their need to record low-level system events. Further papers will focus on such system issues.

If you plan to monitor everything on a honeypot and want to be stealthy, how do you export security events to a logging host? After all, you have to store the logging data on a different host because you cannot trust the logged data on a honeypot after it has been compromised. If you are using the network, there might be ways for the attackers to detect it by playing with network tools on your honeypot (e.g. after obtaining a shell).

Another problem with a high interaction honeypot is the activity: as this is not a real computer, what kind of network activity will be seen if the attacker sniffs the traffic? Nothing. And finally, what if the attacker tries to bounce from your honeypot to attack further systems? If you let her jump away, she will be able to attack something else using your IP address as the source of attack, which might be a very big problem from a legal point of view. So usually, this sort of bouncing might be either forbidden or controlled. If it is forbidden, this might look strange to an attacker, like a kind of black hole -- and it will no doubt will raise a flag in the mind of an attacker. If it is controlled, the aggressor might find forbidden requests or limitations, and she might then also conclude that it is a honeypot.

The theory of cloaking and honeypots can be summed up as follows: the closer you are to reality, the more difficult it will be to properly assume data capture and data control, but also the stealthier you will be. That is all part of the game and your choices will depend on the kind of attackers you wish to catch. If you are waiting for script kiddies or unsophisticated attackers, they will probably be blind and will not even notice the honeypot. But if you are waiting for skilled attackers, you should be aware of the methods and tools they use to identify honeypots.

Now let's start with some technical examples.

2. Practical examples

2.1 Tar Pits

A tarpit is a computer entity that will intentionally respond slowly to incoming requests. The goal is to delude clients so that unauthorized or illicit use of a fake service might be logged and slowed down. Note that some purists do not really consider a tarpit to be a honeypot, though it is certainly a fake information system resource that can delay any incoming aggressors. For example, to fight off spammers, some people run tarpits that look like open mail relays, but instead answer very slowly to SMTP commands. These are layer 7 tarpits. Other known tarpits are those that play with the TCP/IP stack in order to hold the incoming client's network socket open while forbidding any traffic over it (layer 4).

The Labrea Tarpit [ref 1] is an excellent example that plays with the TCP/IP stack and has been used to slow down the spread of worms over the Internet, but there are also others such as Honeyd [ref 2] and some native tools in Linux. For example, netfilter/iptables [ref 3] supports a TARPIT target. To achieve this tarpit state, iptables accepts an incoming TCP/IP connection and then immediately switches to a window size of zero. This prohibits the attacker from sending any more data. Any attempt to close the connection is ignored because no data can be sent by the attacker to the target. Therefore the connection remains active. This consumes resources on the attacker's system but not on the Linux server or the firewall running the tarpit. An example iptables rule for TARPIT mode looks like:

```
iptables -A INPUT -p tcp -m tcp -dport 80 -j TARPIT
```

Though tarpits are not built to avoid fingerprinting, this is an interesting technical case to propose for our first example.

For a layer 7 tarpit, by looking purely at the latency from the service, an attacker might guess that she has found a fake system after multiple attempts.

For a layer 4 tar pit like Labrea, the TCP window size is reduced to zero, and the tar pit continues to acknowledge incoming packets. This simple signature will probably alert the attacker.

You can see that an attacker (10.0.0.2) trying to reach a fake web server, simulated by Labrea in persistent mode (10.0.0.1), in the following recording made with tcpdump:

```
03:26:01.435072 10.0.0.2.1330 > 10.0.0.1.80: S [tcp sum ok]
911245487:911245487(0) win 64240 <mss 1460,nop,nop,sackOK> (DF) (ttl 64, id 6969, len 48)
03:26:01.435635 10.0.0.1.80 > 10.0.0.2.1330: S [tcp sum ok]
3255338435:3255338435(0) ack 911245488 win 3 (ttl 255, id 48138, len 40)
03:26:01.435719 10.0.0.2.1330 > 10.0.0.1.80: . [tcp sum ok]
1:1(0) ack 1 win 64320 (DF) (ttl 128, id 4970, len 40)
03:26:01.435887 10.0.0.2.1330 > 10.0.0.1.80: . [tcp sum ok]
1:4(3) ack 1 win 64320 (DF) (ttl 128, id 4971, len 43)
03:26:01.436224 10.0.0.1.80 > 10.0.0.2.1330: . [tcp sum ok]
1:1(0) ack 4 win 0 (ttl 255, id 44321, len 40)
03:26:03.731433 10.0.0.2.1330 > 10.0.0.1.80: . [tcp sum ok]
4:5(1) ack 1 win 64320 (DF) (ttl 128, id 4973, len 41)
03:26:03.731673 10.0.0.1.80 > 10.0.0.2.1330: . [tcp sum ok]
1:1(0) ack 4 win 0 (ttl 255, id 35598, len 40)
```

By looking at the answers from 10.0.0.1, you will at first notice a window size of 3 and then 0 for the next connection (win 0). You can then understand how an attacker could fingerprint this behavior easily.

2.2 A few words about layer 2

If an attack is launched from the same LAN segment as a honeypot, there might be issues seen at layer 2. This might be important if you want to handle the inherent risks with an intruder who would otherwise succeed in gaining access deeper and deeper into your network infrastructure. It might also be important with a honeypot that would be used to catch malicious internal users.

Labrea also has the capability of answering requests sent to computers that do not exist. By looking at unanswered ARP requests, Labrea might be configured to simulate unused IP addresses, which is very interesting way to fight worms on large networks with thousands of such IP addresses. If an attacker is on the same network segment as Labrea, there is a way to do fingerprinting at layer 2: this daemon always answers with the same unique MAC address 0:0:f:ff:ff:ff, which acts as a kind of black hole, and thus there is an obvious way to detect it. By looking at such ARP responses, the attacker might have such a concern:

```
04:59:00.889458 arp reply 10.0.0.1 (0:0:f:ff:ff:ff) is-at 0:0:f:ff:ff:ff
```

If you want to explore this as an exercise, you can find and change this hard coded value in the sources of Labrea (PacketHandler.c):

```
u_char bogusMAC[6] = {0,0,15,255,255,255};
```

VMWare [ref 4] is a well known commercial software for virtual machines that allows you to launch multiple instances of different operating systems on a single piece of hardware. These operating systems are isolated in secure virtual machines and the VMware virtualization layer maps the physical hardware resources to the virtual machine's resources, so each virtual machine has its own CPU, memory, disks, I/O devices, etc. It only emulates x86 hardware at the moment and it is widely used by honeypot operators because it allows, among other things, an easy deployment of honeypots. Sometimes you can guess that a system is running on top of VMWare by looking at the MAC addresses. It does not mean that this is a honeypot, but this might give pause and some doubts to an aggressor. If you look at the IEEE standards [ref 5], you will find this current range of MAC

aggressor. If you look at the IEEE standards [ref 5], you will find this current range of MAC adresses assigned to VMWare, Inc:

```
00-05-69-xx-xx-xx
00-0C-29-xx-xx-xx
00-50-56-xx-xx-xx
```

So, if you see such a MAC address either by looking at the cached MAC addresses (via arp -a) or by looking at the data related to the interface (Unix: ifconfig or Windows: ipconfig /all), an aggressor might find something interesting.

Some attackers try to reach remote NetBIOS services in order to launch Windows specific attacks. Honeypots builders dream of catching 0-day exploits against a patched system, but using the Windows integrated firewall might stop most attackers. That's why they often open the related Windows ports (NetBIOS ports, including 135, 137-139 and 445 TCP/UDP), waiting for an intruder. But what if an attacker interacts with the NetBIOS service? She will be able to get the MAC address and guess that a system is in fact a VMWare guest (Unix: nmblookup or Windows: nbtstat -A @IP). Some could argue that it is possible to change the MAC address in the configuration of VMWare, but still only some addresses might be accepted: VMWare's MAC addresses are beginning with 00:50:56 (e.g. ethernet0.address = 00:50:56:XX:YY:ZZ).

There are also other points of interests for attackers that would like to fingerprint a VMWare owing to MAC addresses. For example, when the VMWare ESX server automatically generates MAC addresses like 00:05:69:XX:YY:ZZ, it usually means that the IP address of this server is like A.B.C.D where XX is the hexadecimal of C, and YY is the hexadecimal of D. This might reveal the use of NAT before the VMWare box (different external address).

Honeyd [ref 2] is a powerful open source honeypot daemon written by Niels Provos. In the past, most people have used Honeyd with another tool, arpd. This one answered ARP requests in order to redirect needed traffic to Honeyd. Some people thought that this could create a stealth problem because there would be multiple IP address with the same MAC address (but this can also happen on a layer 2 bridge). If you use a recent version, Honeyd now allows you to specify a MAC address for each virtual computer without being limited to just one. Simply add this line for a created template, by choosing the MAC address for your simulated systems:

```
set template ethernet "<vendor|mac address>"
```

This might be better than using the arpd daemon and gives a great opportunity for stealth at layer 2. Maximillian Dornseiff has also outlined some possibilities for using honeyd without arpd. [ref 6]

User-Mode Linux (UML), a free software under the GPL, is another tool to create virtual machines. [ref 7] It virtualises Linux itself so that you can run an entire Linux environment in user-space and it allows you to run multiple instances of Linux at the same time and on a single piece of hardware. Dedicated to Linux, it looks similar to the commercial solution VMware. That's why so many people use it to build honeypots. From a layer 2 point of view, there is a powerful option to fix the MAC address of the UML guests by appending some parameters while launching it:

```
eth0=tuntap,,xx:xx:xx:xx:xx:xx,@IP (where xx:xx:xx:xx:xx:xx is the MAC address
and @IP is the IP address).
```

3. Concluding Part One

It is a difficult problem to deploy honeypots that cannot be detected by hackers. We must remember that honeypot technology is only effective if an attacker does not know she is attacking a "trap" instead of a real system. Therefore, it is critically important for security professionals who deploy honeypots to be aware of the methods blackhat hackers use to identify them.

3.1 Coming up

This introduced the issues at hand and then discussed issues with tarpits and virtual machines, primarily at layer 2 of the OSI model. Next time we'll continue the discussion with many more practical examples of detecting honeypots, including Sebek-based honeypots, snort_inline, Fake AP, and Bait and Switch honeypots. Stay tuned.

4. References

[**ref 0**, PacSec in Tokyo [http://pacsec.jp/], organized by Dragos Ruiu, and attend the talk "Countering Attack Deception Techniques" from Oudot Laurent.

[**ref 1**, Labrea Tarpit, by Tom Liston: http://labrea.sourceforge.net/]

[**ref 2**, Honeyd project, by Niels Provos: http://honeyd.org/]

[**ref 3**, netfilter/iptables: http://www.netfilter.org/]

[**ref 3**, netfilter/iptables: http://www.netfilter.org/]

[**ref 4**, VMWare: http://www.vmware.com/]

[**ref 5**, IEEE standards: http://standards.ieee.org/regauth/oui/oui.txt]

[**ref 6**, Maximillian Dornseif's discussion on using honeyd without arpd. http://blogs.23.nu/antlab/stories/4485/]

[**ref 7**, Know Your Enemy: Learning with User-Mode Linux http://www.honeynet.org/papers/uml]

View more articles by Laurent Oudot on SecurityFocus.