A virtual high-interaction Honeypot

Guillaume Chazarain, Benoît Vallette d'Osia, Nicolas Nobelis, Karima Boudaoud
I3S Laboratory, University of Nice Sophia Antipolis, France
{*chazarai,vallette,nobelis,karima*}*@essi.fr*

Abstract

In order to protect oneself from crackers, it is advisable to know their behavior. Instead of trying to make the machines completely hardened, the principle of the honeypot[13] is to discover interesting information from the attacks thanks to machines voluntarily configured to be cracked. It is then possible to use part of this information to better protect the production machines. Our honeypot is described as one with high interactions because it leaves the cracker completely free and no assumption are made on its actions.

Keywords

High interaction Honeypot, Virtualization, User Mode Linux, Security, Attacker behavior

1. Introduction

Tel est pris qui croyait prendre.

—Jean de La Fontaine[2]

The development of computer science and networks made it possible to put whole data on machines, and to exchange them through means of communication. The possibilities offered by these tools are exploited for productive aims but can also be exploited by badly-disposed people. Thus, there is a great need for security which grows with the network, a security which must adapt to the various attacks.

Many security tools are available to the users and administrators to face these problems. The purpose of all these tools is to protect the machines against attacks. Among these tools, there are firewalls, antivirus, and intrusion detection systems (IDS) that analyze events to find out if the computer is attacked or not.

Honeypots are a very original approach to computer security. A honeypot is a decoy, a seemingly badly protected machine, but which actually monitors the users and their behaviors. Contrary to the usual way, it is wished that malicious people use the honeypot. The honeypot is a resource that does not have a production value, it does not have an authorized activity, any usage is obviously malicious.

The honeypots are classified in two categories[8]:

- Low interaction: Fake services. This type of honeypot is characterized by the emulation of services and applies a strong, customized, monitoring to these services. They are relatively simple to make and deploy, and they represent very little risk because the cracker does not have access to the complete system. The emulation has nevertheless a very negative effect as it makes the honeypot easily detectable as such by the cracker.
- High interaction: Complete access. The cracker enters on the machine and thus does not have limits, this gives many useful information that make it possible to let the cracker shows unknown techniques. On the other hand it causes significant risks for the

honeypot since the cracker controls it, but also some risks for other machines as the cracker has an access to the network. It's a part of the project to limit what is available to him. Our honeypot is of this type.

The honeypot is on a dedicated machine with the aim of being crackerd. To gain in dynamism and especially to make the honeypot simpler, we use in our project a virtual honeypot: i.e. a system in the system. The benefits of this virtualization are multiple. The installation of the system, just like its reinstallation amounts to loading an image-disk. The IDS can store its data on the computer, but separated from the virtual honeypot, thus out of reach for the attacker. The deployment of a honeypot comes down to launching its execution on a host machine, which permits for example to have several honeypots on the same machine.

The goal of the honeypot is to retrieve information from the attackers without those realizing they are supervised. In the case of a virtual honeypot, it is necessary to mask to the attackers of the honeypot that they work on a virtual machine, by maintaining good performances, by offering credible information on the hardware and the system, while preserving hidden some resources reserved to the host system.

The paper is organized as follows: the second part gives a description of our honeypot in four parts: hardening, monitoring, installing and testing. The third part looks at related works and the fourth part concludes the paper.

2. Our Honeypot

Our project thus proposes to address fundamental points of the honeypots which are security and monitoring. Moreover we add what appears to us to be crucial for any software, an installation and an easy and complete configuration. Obviously we let us benefit from that to update the honeypot software, that is to say a complete system.

Our project must profit from the recent evolutions, with regard to the base of the honeypot, like the kernel, and the services, but also so that the whole system is as standard as possible to the eyes of the attacker. For the kernel we use User-Mode-Linux (UML)[4], a modified version of the Linux kernel designed to run on top of a standard Linux kernel. The main points are:

Installation of a Fedora Core distribution to take into account its current success as well as its specific qualities.

Usage of the Linux Kernel 2.6 because it is from now on ready for production but also because it integrates the UML basis. The principal advantages are at the performance level, such as for example with the support of NPTL*. One can also notices improvements in the build system, invisible to the user but appreciable at the time of the development.

In order to isolate the system image-disk from the user modifications we use the COW* semantic, provided by the UML, which we will detail here. The system image is in a read-only file, whereas the modifications are in a separate COW file. As this file contains only the differences between the original image-disk and that seen by the cracker, it should be very light.

*Native POSIX Thread Library
*Copy On Write

With this system one can preserve the traces of several intrusions without sacrificing precious disk space.

2.1 Hardening

Security is the topic of our project, it relates to the host system which must be impenetrable, the UML which must prevent the user from reaching certain parts of the host, and their communications which will be hidden.

- Hardening of the UML to even more prevent its finding. In particular a forged `/proc`* filesystem helps us, but also a way to integrate the communications between the honeypot and the host directly in the core of the UML contributes to the hiding. This communication is achieved with an additional system call whose function is to transfer messages from the honeypot to the host. We added a system call because the kernel (the UML in our case) is the only thing we truly control.

2.2 Monitoring

The purpose of the honeypot is to collect information on the attacker, and that must be done with concealment and efficiency. The tools for monitoring must be undetectable since they are unusual, as for the logs, they should not be intercepted by the attacker but their existence will not astonish the attacker. Here are the main aspects of this monitoring:

- Monitoring of the system logs, that is `syslog`, and users of the honeypot. The information provided is interesting but we must keep in mind that the attacker can tamper with these logs.
- Monitoring using security tools for intrusion detection (IDS), such as Prelude[6] or Snort[11].
- Monitoring using network tools capturing network traffic, in particular Ethereal[1].

2.3 Installation

The installation is very simple thanks to generic scripts discovering by themselves some information. The goal is to make the honeypot installable on any Linux system and by any user (with `root` privileges). For that goal, we have a collection of scripts, each one for a task. They are carried out in order, and to add or to remove a task it is enough to add or remove the corresponding script.

2.4 Tests

A testing phase will be carried out to validate the security of our honeypot, the quality of the collected information will help to better tweak it and to obtain the best results. For example, we'll make sure an attacker in the honeypot cannot use Ettercap[7] to see the production network traffic.

Intrusion

An important part of the test is to try to penetrate the honeypot, but unlike in the real life if we success it is a good sign. Nmap[5] will remind us what services are activated and Nessus[3] will probe them. Finally, thanks to lists such as Packet Storm[14] and K-otik[12] we'll make sure exploits are available for the programs we run.

*Virtual filesystem used to gather information about the running system

3. Related work

In the litterature, there are two main works : Honeyd[10] and Honeynet[9].

Honeyd is a daemon that creates virtual hosts that can be configured to expose services, so it belongs in the low interaction category. Compared to our approach, as it is a low interaction honeypot, it cannot be used to analyze the attacker behaviour.

Honeynet is a project aiming at putting several honeypots in a network. Honeynet intends on providing additional information, such as their motives in attacking, how they communicate, when they attack systems and their actions after compromising a system. It is thus a high interaction honeypot.

In the commercial world, there is a high interaction honeypot from Symantec called Symantec Decoy Server. It runs only on Solaris, so we could not test it, and this limits its deployment. This has to be compared to the high availability of Linux systems.

4. Conclusion

This project deals with security in a very unusual way, trying to catch the attacker at its own game, observing it and obtaining information on its behavior. More technically this work requires the use, the configuration, and the customization of a complete system on all levels, from the kernel to the choices of the services, with the network too. At this stage, we have a functional, up-to-date, with installation scripts but not very protected honeypot. The goals are to get at the end of the assigned time a very protected honeypot that one can configure, to be able to test the security of our work, and finally to present the whole in a ready to install package.

References

[1] Gerald Combs. Ethereal. `http://www.ethereal.com`.
[2] Jean de La Fontaine. The rat and the oyster - livre viii - fable 9.
[3] Renaud Deraison. Nessus. `http://www.nessus.org`.
[4] Jeff Dike. A user-mode port of the linux kernel.
[5] Fyodor. Remote os detection via tcp/ip stack fingerprinting.
[6] Vincent Glaume Mathieu Blanc, Laurent Oudot. Prelude hybrid ids.
[7] Alberto Ornaghi and Marco Valleri. Ettercap. `http://ettercap.sf.net`.
[8] Honeynet Project. Know your enemy: Defining virtual honeynets.
[9] The Honeynet Project. Know your enemy, 2nd edition.
[10] Niels Provos. A virtual honeypot framework.
[11] Martin Roesch. Snort - lightweight intrusion detection for networks.
[12] K-OTik Security. `http://www.k-otik.com/exploits`.
[13] Lance Spitzner. To build a honeypot.
[14] Packet Storm Staff. `http://packetstorm.linuxsecurity.com`.

Biography

Guillaume Chazarain and Benoît Vallette d'Osia are Engineering students in Computer Science at ESSI (École Supérieure en Sciences Informatiques) - Sophia Antipolis.