Setup Honeypot: Team Contagiarium

Mark Vijfvinkel, Aram Verstegen, Joost Kremers & Bart Lutgens

November 1, 2011

1 Introduction

In this document we explain how we have set up the honeypot. We have decided to split up our monitoring time into two parts. In the first part we configure the virtual machine as a system of security 'traps', also called a low-interaction (LI) honeypot, and monitor activity for one month. Then we will reconfigure the machine to a high-interaction (HI) honeypot, exposing real, exploitable vulnerabilities that can lead to code execution on our machine and monitor much more intently for another month.

After these periods we will analyze the results of both setups and compare the differences in the results we have captured. The advantage of a LI honeypot is that the machine can only be compromised to a certain extent and can not really be used for any malicious activity. The disadvantage is that it is not possible to capture all the activity an attacker might perform, like after escalating privileges - which should not be possible. The advantage of a HI honeypot is that you can capture all the activity, but that the system might also be completely compromised and do damage to other systems. That is why we need to be much more vigilant in monitoring our honeypot in the HI phase.

2 Setup: Monitoring Machine

In this section we describe the setup of the monitoring machine.

2.1 Syslog-ng

By default, `syslog` uses UDP to transmit its log messages to a remote server. Relying on UDP is not desirable, because everything is sent in plaintext and delivery of UDP messages is not guaranteed. To allow for confidential and reliable transmission of the syslog messages from the one machine to another, one needs to use TCP connections in syslog, and provide some way of encapsulating the socket used in a cryptographic layer.

We have decided to set up a reverse SSH tunnel from the monitoring machine to the honeypot machine, as described in the article "Remote logging with SSH and syslog-ng". [HP] The example command establishes an SSH connection the honeypot, setting options to forego a login shell or anything a normal user would desire, and binds the

local socket 9514 on the honeypot machine to tunnel its connections to the same port on our monitoring machine.

We have configured a locally installed the `syslog-ng` package for our user on the monitoring machine, have configured it to listen to local TCP connections on port 9514 and will thereby receive syslog messages transmitted through this port. We are aware there is still the possibility of other users on the monitoring machine poisoning our logs, but we feel we can rule out foul play by our peers in this assignment.

To allow for secure SSH access, a public keypair was generated on the util machine and the associated public key was added to the honeypot's `authorized_keys` file.

We have put the command to create the SSH tunnel in a script to allow the connection to be re-esthablished quickly and easily if the SSH session is terminated for some reason. A severance of the connection would lead us to assume the worst initially. If re-establishing this connection fails after any attackers have gained access (which should never happen if we monitor adequately), we will have to assume the machine has been compromised beyond our control and will request a shutdown of the VM, if no network downtime was otherwise announced.

2.2 Monitoring

After collecting our syslog information, we want to somehow be alerted of interesting information automatically. Some seemingly appropriate projects are available under an open source license, and we have not yet decided which one we want to use. A customizable solution is absolutely necessary owing to the wide range of incoming information and the modifications made to a few of the installed software packages specifically for logging to syslog. Next to any off-the-shelf solution, some more custom scripting seems inevitable here.

3 Setup: Honeypot

In this section we describe the setup of the honeypot.

3.1 Rsyslog

On the honeypot, we have modified the default `rsyslog` configuration to also send syslog messages to the (now bound) local port 9514, completing the chain to the syslog instance on our monitoring machine. The modified configuration is now stored in its own file in the syslog.d directory, which is a bit obvious to anyone taking an interest. We plan to move this small configuration over to one of the default configuration files to avoid raising suspicion outright. Note that the chosen port for remote syslog is very similar to the default syslog port 514, and this might give us away. We plan to change it to a more nondescript port number.

3.2 SSHFS mount

Not all services support logging to syslog, and for some it is quite infeasible. We will create a read-only `sshfs` mount from our monitoring machine that we will use to write our logfiles for various services to. We will use interactive login to mount this, as allowing the honeypot back into our monitoring machine with an SSH keypair just seems dangerous.

3.3 scanlogd

To detect port scans we have installed the service `scanlogd`. Per detected scan an entry is written to syslog.

3.4 iptables

To log all incoming and outgoing connection attempts, we have added the following rules to the iptables configuration. This will also log to syslog.

```
iptables -I INPUT -m state --state NEW -j LOG --log-prefix "New Connection: "
iptables -I OUTPUT -m state --state NEW -j LOG --log-prefix "New Connection: "
```

We are also going to filter outgoing SMTP, SIP and NetBIOS traffic.

3.5 Kippo

To capture attackers that try to come in by brute-forcing SSH logins, we are using Kippo. Kippo is a sandbox SSH service, used to log bruteforce attacks and the shell interaction performed by the attacker after gaining access (with a trivial password). We've installed the latest development version to run as a separate user.

To make this as realistic as possible we have added a redirect in the iptables configuration from the real SSH port 22 to port 2222 that Kippo uses by default. The reason for forwarding in iptables was to not give Kippo root rights, which would be very dangerous. Unfortunately the forward broke down around the time the firewall was being reconfigured, for reasons yet unclear . If it can't be resolved, we can circumvent this issue by running Kippo through a copy of the Python interpreter that has the capability to bind privileged ports.

To be able to still log on to the honeypot, we have configured our real SSH service to run on port 9022.

We have patched Kippo to log messages to syslog.

3.6 DenyHosts

If someone finds out our real SSH port changed to 9022, they might try to break in through our real SSH service. To make sure no bruteforce attacks can be done on this port, DenyHosts is installed. This service blocks IP-addresses that try to log in more

than 3 times with the wrong password by adding them to the `hosts.deny` file. The hosts listed there are blacklisted on the machine, but we can edit them if needed. Denyhosts can log to syslog by editing its configuration file.

4 Apache & Nginx

To attract some web traffic, we have installed the Apache web server, the Apache PHP runtime and the `mod-security` module which allows us to spoof its version information, to make it appear old and vulnerable. The Nginx webserver is installed as a reverse proxy in front of Apache, which runs on local port 8080. This provides 2 layers of logging (to files), and Nginx's superior transparent caching mechanism allows us to avoid exhausting our resources with scripting languages, should we ever face an application-layer DoS attack (Nginx appears quite resilient against those).

Nginx has been compiled from source and was patched to send the spoofed Apache version header as well. We will configure both services to log to our `sshfs` mount.

4.1 Google Hack Honeypot

Google Hack Honeypot (GHH) is designed to be used to attract search engine attackers. These attackers will use search queries to find sensitive data, for example password files, that due to some misconfiguration ended up being indexed by a search engine like Google. GHH allows logging of these requests to a MySQL database or Comma Seperated Value (CSV) file.

Since GHH offers multiple search targets, we picked one to start with, namely GGH version 1.2 passlist.txt. We had no particular reason for this, only that it was easy to set up. We plan to install more variants of the GHH to attract more attackers, but before this could happen the VM's were shut down.

We configured GHH to log to in CSV format, since it requires minimal setup and is easily parsed. Then, we patched the GHH code to allow logging to syslog, to maintain a single, consistent point of log collection.

We also hope to detect and capture a bruteforce attack or scan on generic filenames, for example `passwd.txt`. This will probably warn us that an attacker will try to use a dictionary attack.

4.2 Wordpress

To capture more attackers during the HI phase, we plan to use an outdated version of WordPress. Wordpress is software, powered by PHP and MySQL, that can be used to easily create websites or blogs and manage them via the content management system. Exploits are found on a frequent basis for Wordpress and patches are published on a regular basis. We plan to use an older version of Wordpress, for example a version below version 3.0.4, to be sure that we use a version for which public exploits exist.

4.3 Snort

Snort is an open source intrusion detection system (IDS) that uses realtime traffic analysis and packet logging on internet protocol networks. Snort was designed to be used as a network IDS, however we will be running it on our host machine, effectively making it a host-based IDS.

Unfortunately the VM network went down before we were able to complete our Snort configuration. We intend to set up Snort using the community rule-set that will only alert on suspicious network behaviour, but do nothing to prevent it. Included are rules on detecting exploit attempts on various common services like HTTP, FTP, SMTP, and many more.

Considering we are in a VM we might be limited in RAM, which Snort seems to require a lot of. The configured rules will have to be minimal to avoid exhausting resources.

We will configure Snort to send alerts to syslog, and to log its packet captures to the read-only `sshfs` mounted from our monitoring machine. We will install a script like `oinkmaster` to automatically retrieve the latest community Snort rules, with a Snort API key registered specifically for this project.

4.4 Auditing

For the HI phase, we would like to have a way to audit commandline interaction. Various options exist, such as the user-space `auditd` daemon, but to make auditing harder to detect this we feel it would be best to look for a kernel-space solution. An auditing module developed by the Honeynet project called Sebek can provide this, but is unmaintained and outdated. The grsec kernel patches also provide auditing from kernel space, but we were advised against replacing the stock kernel.

We are still investigating the best way to achieve useful auditing. A self-written kernel module that hooks system calls, albeit limited, might suffice to get some much needed auditing information from the honeypot.

References

[HP] Deer Run Associates Hal Pomeranz. Remote logging with ssh and syslog-ng.