J Comput Virol (2011) 7:143–157
DOI 10.1007/s11416-010-0144-2

ORIGINAL PAPER

Set-up and deployment of a high-interaction honeypot: experiment and lessons learned

Vincent Nicomette · Mohamed Kaâniche · Eric Alata · Matthieu Herrb

Received: 24 July 2009 / Accepted: 16 May 2010 / Published online: 27 June 2010

Abstract This paper presents the lessons learned from an empirical analysis of attackers behaviours based on the deployment on the Internet of a high-interaction honeypot for more than 1 year. We focus in particular on the attacks performed via the SSH service and the activities performed by the attackers once they gain access to the system and try to progress in their intrusion. The first part of the paper describes: (a) the global architecture of the honeypot and the mechanisms used to capture the implementation details so that we can observe attackers behaviours and (b) the details of the experiment itself (duration, data captured, overview of the attackers activity). The second part presents the results of the observation of the attackers. It includes: (a) the description of the global attack process, constituted of two main steps, dictionary attacks and intrusions and (b) the detailed analysis of these two main steps.

1 Introduction

In the last decade, malicious activities have proliferated on the Internet and exhibited a dramatic increase in volume and diversity. Such threats include viruses, worms, denial of service attacks, phishing attempts, botnets, etc. [18]. Monitoring malicious threats on the Internet and analyzing how the attackers proceed for exploiting systems' vulnerabilities would provide valuable information to security systems designers to build efficient protection mechanisms taking into account the real threats observed in an operational context. This motivates the need for methods for collecting and monitoring real world data related to malware and attacks and for experimental results based on the analysis of such data.

The security community has set-up several initiatives and data collection mechanisms aimed at fulfilling such objective using different techniques such as:

- the aggregation of logs collected from different sources (intrusion detection systems, firewalls, etc.) [12],
- the monitoring of traffic targeting unassigned IP addresses using the so called darknets [8], network telescopes [19], blackhole monitors [11,5] or background radiation monitors [28],
- the deployment of honeypots, i.e., network resources dedicated to be probed, attacked and compromised, that can be monitored to observe how attackers behave [34].

These techniques are complementary and can be used in combination. In this paper we focus on honeypots and their use for observing attackers behavior.

Honeypots are not supposed to be used for production-related services. Thus, all the traffic observed on the corresponding machines is suspicious and will very likely point to malicious activities. Two types of honeypots can be distinguished depending on the level of interactivity that they offer to the attackers: low and high-interaction honeypots [34]. Low-interaction honeypots do not implement real functional services. They emulate simple services, network stacks or some parts of an operating system and do not offer attackers

V. Nicomette (✉) · M. Kaâniche · E. Alata · M. Herrb
CNRS, LAAS, 7 avenue du colonel Roche,
31077 Toulouse, France
e-mail: nicomett@laas.fr

M. Kaâniche
e-mail: kaaniche@laas.fr

E. Alata
e-mail: ealata@laas.fr

M. Herrb
e-mail: matthieu@laas.fr

V. Nicomette · M. Kaâniche · E. Alata · M. Herrb
Université de Toulouse, UPS, INSA, INP, ISAE, LAAS,
31077 Toulouse, France

the possibility to realize operations. Examples of implementations include `Tiny honeypot` [4], `Nepenthes` [7] and `honeyd` [25]. Thanks to their lightweight implementation and easy administration, data collection infrastructures using low interaction honeypots have been also deployed at a large scale to analyze and provide a global picture of malicious activities observed at different locations of the globe. We can mention for example the leurre.com project [24] that has deployed since 2004 more than fifty honeypot platforms based on `honeyd` covering the five continents.

While low interaction honeypots are useful to provide quantitative statistics about malicious threats and high-level information about attack patterns on the Internet, they are not suitable for monitoring the activity of the attackers who get the control of a target victim machine and try to progress in the intrusion process to get additional privileges. Such objective can be achieved with high-interaction honeypots that offer real services to the attackers to interact with, which makes them more risky than low interaction honeypots. A high-interaction honeypot can correspond to a physical conventional computer system, or consist of virtual machines set up with VMware [38], User-mode Linux UML [9], or Qemu [6]. Virtual implementations offer more flexibility in terms of configuration, deployment and administration. Examples of high-interaction honeypots are presented in [23]. We can mention e.g., Argos [14], Sebek [26], Potemkin [37] and Honeybow [41].

So far, high-interaction honeypots have been mainly used to capture and analyze autonomous propagating malware such as worms, viruses and botnets. On the other hand, little evidence and experimental data have been published about the observation and analysis of non automatic attacks managed by human beings, using high-interaction honeypots. This is mainly due to the difficulty of setting up such experiments. The objective of the research reported in this paper is to contribute to filling this gap.

Contributions: In this paper, we describe the lessons learned from the development and deployment of a high-interaction honeypot aimed at observing the progress of real attack processes and the activities carried out by the attackers in a controlled environment. We are mainly interested in observing and analyzing the activities related to non automatic attacks. In our set-up, we have focused on the monitoring of intrusions requiring the successful connection through the SSH service. Indeed, recent studies of vulnerability trends (see, e.g., the Top20 security risks report published by the SANS Internet Storm Center in 2007 [32]) have shown a significant increase of SSH attacks against Internet services using in particular brute-force attacks. Our honeypot uses Gnu/Linux as a target operating system. Three types of data are recorded by the honeypot: (1) the user passwords and logins tried by the attackers to gain access to the system, (2) the data exchanged within the SSH connections, and (3) the system calls generated by the activity of the attackers.

The proposed honeypot has been deployed on the Internet for more than 1 year (419 days) during which 552,333 SSH connections have been observed. In this paper, we present the methodology that we have developed to process this data and we summarize the main lessons learned. Two main steps of the attack process are investigated: (1) the first one, generally performed by means of automatic tools, concerns brute-force dictionary attacks aimed at gaining access to the system, and (2) the second step concerns the activity carried out by the attackers once they succeed in breaking into the system (i.e., intrusions). As concluded from the analysis, the second step has been performed by human beings.

Preliminary results based on the analysis of a subset of the data were discussed in [2]. However, the analysis made in [2] concerned a shorter period of time (6 months). The results had to be confirmed over a longer period, what is done in this paper. Moreover, the study reported in [2] only focuses on the second step of the attack process and does not include the dictionary attacks. Similar analyses regarding malicious SSH login attacks, are presented in [29] and [33]. They partially confirm some results of this paper. However, the observation period for these studies is much shorter, and the analyses are less complete than the study presented in this paper.

The remainder of this paper is organized as follows. Section 2 discusses the design rationale and the implementation for the deployed high-interaction honeypot. Section 3 gives a high level view of collected data and the observed attack processes. Section 4 is dedicated to the analysis of dictionary attacks and corresponding user logins and passwords tried by the attackers. The intrusions and the corresponding activities performed by the attackers are analyzed in Sect. 5. Finally, Sect. 6 summarizes the main conclusions and future work.

2 Architecture of the honeypot

Our work aims at analyzing the behavior of human being attackers on the Internet. For that purpose, we need to develop an experimental platform allowing the observation and monitoring of the activities carried out by such attackers. In this section, we present the whole architecture and the design choices for our honeypot as well as implementation details.

2.1 Principles

Our goal is to observe and analyze the behavior of human being attackers once they have broken into a computer. For that purpose, we have to answer three questions. How can we attract human being attackers ? How can we collect information about their activities ? How can we control these activities ?

Even if high-interaction honeypots are well suited to observe attackers inside a target system, such tools may capture activities originating from different kinds of attackers: human beings, but also automatic tools. Several papers have analyzed, thanks to honeypots, automatic software such as worms that spread on the Internet. In [10], the authors use honeypots to identify automatic attack patterns of the Internet worms. Honeypots were also used to analyze the behavior of attackers that try to build botnets, as presented in [30] and [40].

In order to observe and monitor human managed manual attacks, we need: (1) to provide to the attackers a vulnerable target operating system, and (2) to ensure that the potential risks that might result from the exploitation of such vulnerabilities are properly controlled and mastered. Also, the observation of the activities related to the attacks requires some modifications of the operating system. For these reasons, we decided to use the Gnu/Linux operating system for our honeypot: we are familiar with this operating system and the availability of the source code is a great advantage. In our experiments, we are more interested in observing and monitoring the activities carried out by the attackers once they succeed in breaking into a system, than in the process that allowed them to break into that system. Accordingly, we have used as a target for attacks an SSH server running on the Gnu/Linux operating system that includes a very common vulnerability to compromise Internet systems: i.e., user accounts with weak passwords. As a consequence, we also limit the intrusions performed by automatic tools.[1] As a matter of fact, it is more difficult for an automatic tool to perform an attack in a coherent way via the SSH service, whereas this is more relevant for manual attacks because such vulnerability offers them the possibility to perform interactive connections on the system.

Authorizing remote accesses to the SSH service only does not particularly betray our honeypot. As a matter of fact, such a configuration is representative of the common configuration of traditional systems connected to the Internet. Thus, it is not specific and should not let the hackers think, at first glance at least, that they are targeting a honeypot.

In order to offer an interesting target to attackers and to allow us to collect detailed information about their behavior, we have set up several honeypots. This is useful in particular to observe potential attempts to perform stepping stone attacks from one honeypot to other honeypots. However, administrating several physical machines is a hard task. The virtualisation technique enables to have only one physical machine to administrate, upon which several virtual operating systems can be hosted. We decided to use this technique and more precisely the `VMware` software [38]. With virtual operating systems, the cloning, the reconfiguration and the modification of the operating system is very simple. Furthermore, if the attacker succeeds in destroying some part of the operating system he has broken into, the recovery procedure is simplified compared to the case of a real operating system.

Our honeypot must be adapted to the observation of manual attacks, while staying as much elusive and transparent as possible, to avoid their detection by the attackers. The modification of the virtual operating system is necessary to collect appropriate information to analyze the behavior of the attackers. We identified three sources of information to capture: (1) the pairs (username and password) tested by the attacker to gain access to the system; (2) the data exchanged inside the SSH connection; and (3) all the system calls generated by the activity of the attacker.[2] In order to capture this information, we made some modifications to the virtual hosts, in the kernel of the operation system. Section 2.2 presents implementation details.

Once the data is captured, it is necessary to archive it for future analyses. Different backup strategies can be considered: (a) through the network or on the honeypot itself, and periodically or not. We chose to backup the data periodically and without the use of the network. We explain the reasons of this choice and the implementation details in Sect. 2.3.

2.2 Modification of the kernel source code

To modify the kernel of a Linux operating system, two main approaches can be distinguished. The first one consists in directly patching the kernel of the operating system and the second one consists in dynamically loading a module in the operating system. The second approach is in general easier to detect by the intruders. Thus, we decided to adopt the first approach, to be as transparent as possible. For the same reason, we chose to develop our own implementation without disclosing the main technical details, instead of using Sebek for example, that is widely known and for which several techniques exist for its detection.

The kernel of each virtual machine was patched at two places. We instrumented: (1) the functions that allow us to record all the keystrokes and characters typed by the intruder while he/she has successfully penetrated the system and used an interactive shell, and (2) the function that allows us to intercept each system call executed by the intruder.[3]

Additionally, in order to capture the usernames and the passwords tested by the attackers, we created a new system

[1] Let us note that some recent worms include password brute force attacks, see e.g., [13].

[2] This system call capture is not efficient if the attacker uses a userland execve [15].

[3] This function was modified in case the analysis of the tty activity would not be sufficient to determine the nature of the attacker behavior (for instance, in the case where an attacker downloads and executes a malicious program that does not print any output information on the tty).

call in the kernel and modified the SSH server accordingly to use this new system call. All this information is logged on a particular region of the kernel memory space of the virtual host. This choice is motivated by the fact that most of the patched code runs in privileged mode (ring 0) and has direct access to kernel memory space.

The first modification of the kernel source code concerns the `tty` driver. This driver controls all the terminals and pseudo-terminals on Linux hosts (and particularly all the characters typed by the user on these terminals). It defines the read and write functions for terminals. The modification of these routines allows us to intercept the characters typed by the attackers as well as all what the attacker sees on the terminal. This modification is realized, for the read operation, by inserting a call to our function `log_tty_read` that stores all the data that are read.

The second modification is done in the `exec` system call, that allows any program to be executed on the system. More precisely, we modified the kernel `do_execve` routine. This routine receives a binary file name, loads it in memory before creating a process associated to this file. By modifying this system call, we can memorize the list of files executed by the attacker.

The last modification of the kernel source code consists in creating a new system call. For that purpose, we added the corresponding routine in the source code and modified the files `syscall_table.S` (table of system calls) and `unistd.h` (list of all system call numbers). Indeed, the SSH server does not run in ring0 and thus cannot directly write in the kernel memory space on the virtual host. It has to use a system call for that.[4] The SSH server uses this new system call to memorize each username/password tested by an attacker.[5]

Let us note that there are many low level system call capture tools, such as `ptrace`, SAL [31] or LinuxBSM. None of these tools was suitable for our requirements. In particular, all these tools log the captured information in the file system, whereas in our case, the captured information is directly stored in the kernel memory (see next section for justification). Furthermore, some of these tools, e.g., `ptrace` are known to be easily detectable by the attackers.

The entry point of our honeypot is the SSH daemon. It is the only service that runs on the honeypot, and, to our knowledge, our SSH version does not contain known vulnerabilities. The attackers may exploit the vulnerabilities of this daemon to gain access to the core of the system. We have analyzed the vulnerabilities published corresponding to the release of the SSH daemon used in our honeypot considering the `openssh` web site and the CVE database. Some of them allow the attacker to bypass the authentication phase. However, even in this case, a pseudo terminal is attached to the attacker and we are able to observe his activity on the honeypot.

2.3 Backup of the collected data

As explained in the previous section, the collected information is stored in the kernel memory address space. This memory region has a fixed size to avoid using dynamic memory allocation, which could produce an overload in the CPU activity. In order to optimize the memory space, the collected data are compressed and encrypted using the LZRW1 algorithm LZRW1[39].[6]

The code that captures these data is part of the kernel running code. As a consequence, storing the captured data in the kernel memory address space is easy to implement and furthermore, is difficult to detect. The detection of the process by the attacker requires that: (1) the attacker gains the root privileges; (2) he identifies the memory region where the data is stored; and (3) he identifies that the LZRW1 algorithm has been used to compress the data.

To backup the data in a database server, the data are not sent through the network, as it is often realized in many high-interaction honeypots (see, for example, [1]). The data are copied into the real host: all the content of the virtual host memory is directly accessible from the real host. The main advantage of this backup strategy is that it is difficult to identify by the attacker. It is a simple copy of data from a memory region to the hard disk of the real host and it is carried out only once a day at a fixed date (this operation lasts less than 1 s). Nevertheless, this strategy could be detected by an attacker who monitors the CPU load. A better, but more complex solution, would be to backup the data when no attacker is logged on the honeypot. However, this would require to interrupt the backup process whenever an attacker logs in during this process.

Figure 1 gives an overview of the honeypot implementation with the data backup mechanisms.

2.4 Deployment and preliminary analyses

The whole platform used for our experiment is composed of three virtual hosts (with Gnu/Linux operating systems) run on top of the VMware software. These virtual hosts are standard machines with usual development tools (text editor, C compiler,...), patched with our kernel modifications. Only incoming connections to the SSH service (port 22) are authorized by a firewall. All outgoing connections are forbidden to prevent the use of our honeypot by the attackers to attack other Internet hosts.

[4] We could also use an existing system call and modify its behavior.

[5] This information is backed-up later on a storage system, see Sect. 2.3.

[6] We did not add any additional mechanism to prevent kernel memory exhaustion, which is still possible, but it never occurred during the whole experimentation.

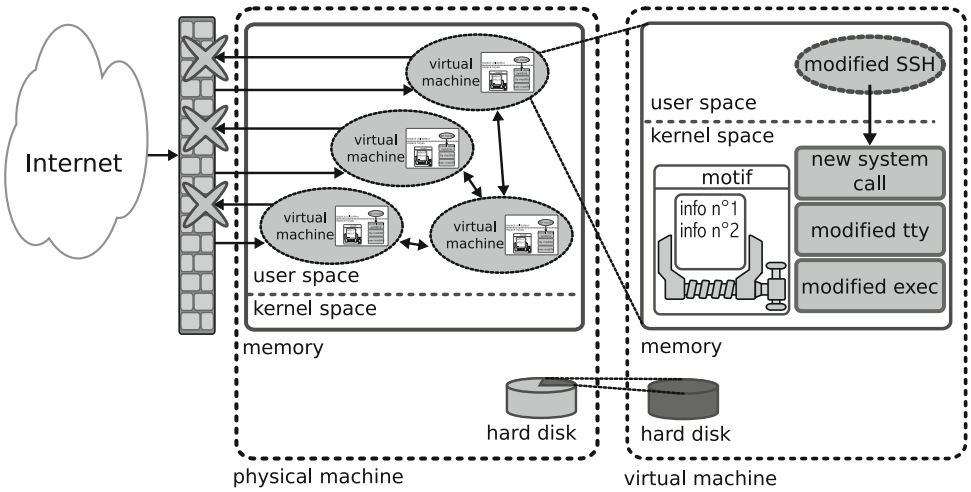

Fig. 1 Implementation of the high-interaction honeypot

A connection to the SSH service is the observation unit supplied by the honeypot. Each connection is characterized by five attributes: the date of the connection, the IP address of the origin of the connection, the username and password tested, a boolean indicating if the password is correct and a boolean indicating if this connection contains interactive commands.

Our experiment was carried out in two steps. We started by deploying the honeypot with the SSH service up but without any user accounts. During this first month of the experiment, we obviously observed only failed SSH connections. However, we used these connections to have an idea of the most tested pairs (username/password). Then, we created 17 user accounts. All these user accounts were created with valid passwords, and more precisely, the login was identical to the password. Furthermore, no fake activities were created for these accounts.

This platform has been deployed on the Internet and has been running for 419 days (from 5 January 2006 to 20 March 2007) during which 650 IP addresses have tried to contact the SSH port of the honeypots, for a total number of 552,333 connections. For each of these connections, a pair (username/password) has been supplied by the attacker. Some pairs were used several times. We counted 98,340 different pairs. A detailed analysis of these pairs and the way they are used by attackers is presented in Sect. 4.

Most of the SSH connections submitted to the honeypot were unsuccessful. Considering the successful connections observed for each user account created in the experiment, as illustrated in Fig. 2, we can distinguish two main stages corresponding to two different types of activities. The first stage identified by τ_1 corresponds to the period between the time when the account was created and the time when the first successful connection for this account was observed. The second stage, identified by τ_2, corresponds to the period between the first successful connection for this account and

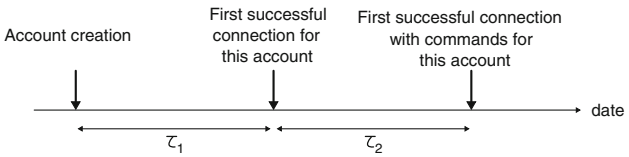

Fig. 2 Relationship between τ_1 and τ_2

the first successful connection including commands typed by the attacker recorded on the honeypot.

In our experiment, we did not observe attack activities including commands that were not preceded by an authentication phase (see discussion at end of Sect. 2.2).

Table 1 presents the durations corresponding to τ_1 and τ_2, that have been observed for each user account. It is interesting to note that τ_2 is never null (except for one account). This clearly indicates that there are two steps in the intrusion process (we will deeply analyze this process in Sect. 3). The first step consists in guessing the correct password for a particular account, password that is not used immediately. The second step, later, consists in using this password to perform an interactive connection to the honeypot with commands.

3 The attack process

This section is dedicated to the precise analysis of the intrusion processes observed on our honeypot. We first introduce the concept of attack session and we describe how we classified them in order to analyze the corresponding attack process. Then, we characterize in more detail, in Sects. 4 and 5, the two main steps of the attack process.

3.1 Identification of attack sessions

Among the SSH connections, some include commands. Let us call these connections *action connections*. Others, that

Table 1 Durations τ_1 and τ_2 observed for each user account

User account	Duration between creation and first successful connection (days)	Duration between first successful connection with commands
UA1	1 day	4 days
UA2	half a day	4 min
UA3	15 days	1 day
UA4	5 days	10 days
UA5	5 days	Null
UA6	1 day	4 days
UA7	5 days	8 days
UA8	1 day	9 days
UA9	1 day	12 days
UA10	3 days	2 min
UA11	7 days	4 days
UA12	1 day	8 days
UA13	5 days	17 days
UA14	5 days	13 days
UA15	9 days	7 days
UA16	1 day	14 days
UA17	1 day	12 days

constitute the majority, do not include commands. Let us call them *authentication connections*.

These authentication connections may of course be due to mistakes, such as a user who does not correctly enter the IP address of the host he wants to connect to. He tries to connect to the bad host, two or three times, realizes his mistake and gives up. However, most probably these connections correspond to brute-force dictionary attacks. A dictionary attack is a succession of authentication connections in a short time, aimed at guessing valid user accounts on the target host. Most of these connections fail, the successful ones correspond to the guessing of valid accounts. Clearly, it is very likely that the discovered accounts will serve for future attacks using action connections including interactive commands. In order to distinguish the different dictionary attacks, it is necessary to specify the minimum number of authentication connections that constitute a dictionary attack. This choice is empirical: we used a sufficiently high value (9) to distinguish real dictionary attacks from mistakes.

In order to have better insights into the attack processes observed on our honeypot platform, and analyze the activities performed by the intruders, we need to organize the large number of connections that we have recorded into attack sessions. A *session* is a sequence of connections in a short time, realized from the same IP address targeting the same IP address (i.e., one virtual machine). A session represents the behavior of an attacker on the honeypot. For example, if an attacker realizes a sequence of connections in a short time

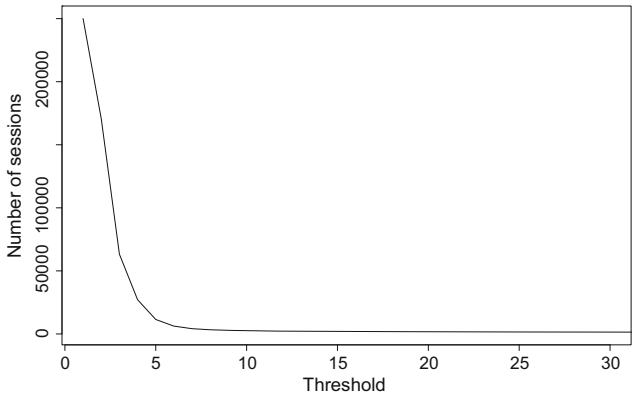

Fig. 3 Evolution of number of sessions according to threshold

the day D, then does not make any connection for 2 days and then performs again a sequence of connections in a short time the day $D+3$, then, we identify two sessions of the same attacker. In some contexts, the IP address of the attacking machine might change due to DHCP lease renewal. If such a change occurs during an attack, then we should observe on the honeypot two contiguous sessions, from two different addresses. Such a situation is not very likely. It did not occur during the whole period of our experiment.

As detailed in [2] (PhD thesis), the identification of the attack sessions can be done based on a sliding window clustering algorithm, considering a timing threshold that defines the maximum duration separating the reception at the honeypot of two consecutive packets belonging to the same attack session. The definition of suitable values for this threshold is generally based on heuristics. Figure 3 plots the number of sessions obtained from the grouping of SSH connection for different values of the timing threshold (in seconds). It can be seen that the number of sessions does not change significantly when the threshold exceeds 16 s, when compared to lower threshold values. In this case, 1,940 attack sessions are obtained from the grouping of the 552,333 SSH connections recorded at the honeypot. The choice of the value of 16 s for the threshold is empirical. However, we noticed that the choice of a higher threshold does not have a significant impact on the classification of sessions and the analysis of attackers activities depicted in the following sections.

3.2 Categorization of sessions

To classify the sessions, we defined three categories: the first one gathers attack sessions that include some action connections corresponding to the execution of commands on the honeypot. These sessions are called *intrusions*. The second category gathers sessions that are composed of dictionary attacks. The third category gathers the remaining sessions. This classification is depicted in Fig. 4.

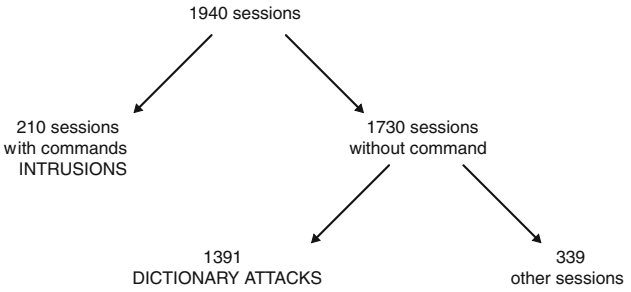

Fig. 4 Classification of sessions

As illustrated in Fig. 4, 210 intrusions were identified from the initial set of 1,940 sessions. The analysis of the remaining 1,730 sessions that did not include the execution of commands, led to the identification of 1391 dictionary attacks. The remaining sessions (339) correspond to sessions that we were not able to classify.

3.3 The main steps of the attack process

Considering the set of dictionary attacks and intrusions, we made further analyses in order to identify relationships between these different types of attacks so that we can better understand the global attack process. We have investigated in particular the IP addresses of the attacking machine as well as the starting date of the attacks.

First of all, we observed that a dictionary attack that enables to discover a user account *always* precedes an intrusion realized on the host with this pair. We also compared the set of IP addresses of the machines that perform dictionary attacks and the set of IP addresses of the machines that perform intrusions: *the intersection is empty*.

Furthermore, the IP addresses used to perform dictionary attacks correspond to machines that could not be located precisely (the RIPE database for example identifies the corresponding country as "EU # Country is really world wide"). They generally belong to class C addresses, whereas the IP addresses used to perform intrusions are mostly located in the same European country, and belong to the same class A network.

This means that the machines that perform dictionary attacks are very likely specialized for this kind of attacks, and they are not used to perform intrusions. This fact is all the more interesting that it was verified during a long period (419 days).

We also compared the IP addresses that performed the dictionary attacks and the intrusions to the IP addresses observed on the low interaction honeypots deployed in the context of the Leurre.com project [22]. *None* of these addresses were observed on the low interaction honeypots. This result confirms the specialization of the machines that only perform one type of attack. Furthermore, as shown in [21], attacks are not necessarily performed blindly on any IP address on the Internet. A preliminary scan step of the Internet, in order to identify the available services running on the accessible machines may be carried out before the dictionary attack. However, the port scanning step is not systematic in any attack process. This has been observed for example in the experimental study presented in [27] in which more than 50% of the observed attacks were not preceded by a port scan.

In the following, we focus on the analysis of the two main steps of the global attack process observed on our honeypot:

1. the dictionary attacks aimed at guessing valid user accounts, and
2. the intrusion process itself, that consists in executing commands on the compromised host.

4 The dictionary attack step

Our main goal in this section is to make a more in-depth analysis of this step and particularly to analyze the different dictionaries that are used in the dictionary attacks, in order to identify some properties of these dictionaries. We try to answer questions such as: are there as many dictionaries as dictionary attacks or are there very few dictionaries that are shared by the attackers community?

4.1 Overview of the observed dictionary attacks

When observing a dictionary attack, we have no means to know if the attacker has completely executed his attack or not. Thus, it is likely that the set of recorded pairs (username/password) is only a part of the dictionary of the attacker. Several dictionary attack tools generate a set of pairs using concatenation of words retrieved from a single list. So, we call *vocabulary* associated to a dictionary attack the set of username and password used during this attack. Formally, a vocabulary $\mathcal{V}(e)$ associated to a dictionary attack e is defined as follows:

$$\mathcal{V}(e) = \{username(t), password(t)/t \in e\} \quad (1)$$

where t is a SSH connection belonging to the dictionary attack e and $username(t)$ and $password(t)$ form the pair used during this connection.

Two vocabularies whose intersection is empty were probably generated from different dictionaries. Two vocabularies whose intersection is important were probably generated from the same dictionaries.

Tables 2 and 3, respectively, present statistics on the vocabulary sizes of the dictionary attacks observed on the honeypot as well as the durations of the corresponding attacks (in seconds). The size of the vocabularies varies between 2

Table 2 Statistics on dictionary attacks durations

Feature	Value (s)
Minimum	1
Quantile 25%	77
Median	313
Quantile 75%	781
Maximum	34,969
Average	701

Table 3 Statistics on vocabulary sizes

Feature	Value
Minimum	2
Quantile 25%	20
Median	123
Quantile 75%	308
Maximum	11,182
Average	289

and 11,182 words. The average and the third quantile are similar. This means that the dictionary attacks observed generally use less than 289 words. Let us note that the first quantile is relatively small (20). The duration of the dictionary attacks are directly correlated to the size of the corresponding dictionaries. As shown in Table 3, the majority of the attacks have a duration between 1 and 13 min. Nevertheless, some attacks were very long, maximum being 9 h.

Some vocabularies share words with other vocabularies and some do not. As a result, the intersection between all the vocabularies is empty. To identify the similarities or dissimilarities between two vocabularies, we propose, in the following section, the definition of a distance between vocabularies and a clustering algorithm based on this distance.

4.2 Distance between two vocabularies

The analysis of the similarities between vocabularies is very close to the analysis of the similarities between two texts. An example of such analysis can be found in [17], in which the author introduces an intertextual distance that takes into account texts with different sizes. Let us define two texts, $\mathcal{A}$ and $\mathcal{B}$ such that the size of text $\mathcal{B}$ is much more higher than the size of text $\mathcal{A}$. If this difference is not taken into account, the distance may be too important. In order to solve this problem, the frequencies of the words in text $\mathcal{B}$ are weighted by the quotient of size of text $\mathcal{A}$ and size of text $\mathcal{B}$.

We propose a distance similar to the intertextual distance. However, we must adapt it so that we can take into account repetitions. In a novel, these repetitions are frequent and common. Regarding dictionary attacks, very few vocabularies include repetitions. As a consequence, the frequency of the words of the vocabulary is very often equal to 1. Thus, we have adapted the intertextual distance so that the repetitions of the words does not have a dominant weight in the definition of the distance.

$$D_{(\mathcal{A},\mathcal{B})} = (1-s)\cdot(1-s') \tag{2}$$

$$(1-s) = \frac{\sum_{m\in\mathcal{M}}(F(m,\mathcal{A}) - F(m,\mathcal{B}))}{\sum_{m\in\mathcal{A}} F(m,\mathcal{A})} \quad (1-s')$$

$$= \frac{\sum_{m\in\mathcal{N}}(F(m,\mathcal{B}) - F(m,\mathcal{A}))}{\sum_{m\in\mathcal{B}} F(m,\mathcal{B})}$$

$$\mathcal{M} = \{m \in \mathcal{A} / F(m,\mathcal{A}) \geq F(m,\mathcal{B})\}$$

$$\mathcal{N} = \{m \in \mathcal{B} / F(m,\mathcal{B}) > F(m,\mathcal{A})\}.$$

In this definition, $F(i,\mathcal{A})$ is the frequency of word i in text $\mathcal{A}$. So, s (respectively, s') represents the proportion of the words of $\mathcal{A}$ (respectively, $\mathcal{B}$) shared with $\mathcal{B}$ (respectively, $\mathcal{A}$). This distance is defined in the interval [0; 1]. If one of the vocabularies is included in the other, it is equal to 0. If the intersection between the two vocabularies is empty, it is 1.

Let us note $s_{max} = \max(s, s')$ and $s_{min} = \min(s, s')$. The distance becomes: $D_{(\mathcal{A},\mathcal{B})} = (1 - s_{max}) \cdot (1 - s_{min})$.

s_{min} is the proportion of the words of the biggest vocabulary shared with the smallest. When s_{min} tends towards 1, then the big vocabulary tends to be identical to the small vocabulary. s_{min} measures the *similarity* between the two vocabularies. In the same way, s_{max} is the proportion of words of the smallest vocabulary shared with the biggest. When s_{max} tends towards 1, the small vocabulary tends to be covered by the big vocabulary. s_{max} measures the *coverage* of the smallest vocabulary by the biggest one.

Two dictionary attacks using the same dictionary may be stopped by the attackers at different stages of the attack. In this case, the vocabularies observed on the honeypot have different sizes. In this context, it is more appropriate to evaluate if two vocabularies are part of the same dictionary based on the coverage metric. As a consequence, the criterion which we consider to evaluate if two vocabularies are part of the same dictionary is the coverage. Thus, we define a clustering criterion according to the coverage of the smallest vocabulary by the biggest.

4.3 Clusters of vocabularies

The objective is to identify the very similar vocabularies and group them into clusters. Our approach consists in gathering two vocabularies if a big proportion of the words of the smallest vocabulary is shared with the biggest one.

In this study, we gather together the vocabularies such that 75% of words of the smallest one are shared with the biggest. This condition is satisfied if $D_{(\mathcal{A},\mathcal{B})} \leq 0.0625$. By using this

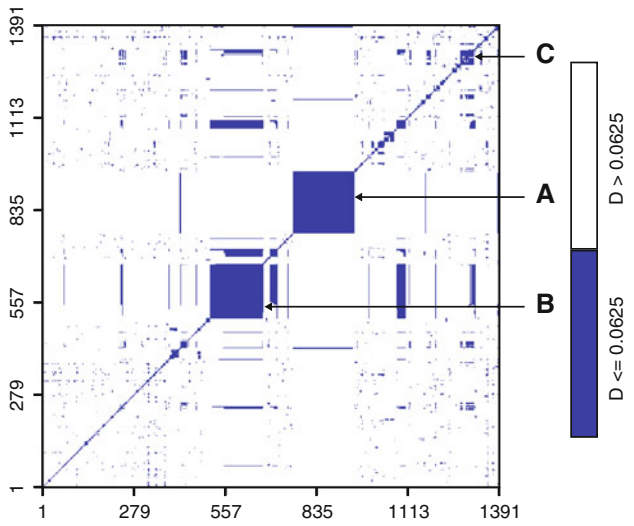

Fig. 5 Distance between vocabularies

criterion, we identified, from the set of 1391 vocabularies, 413 clusters. Only 2.2% of these clusters (i.e., 9 clusters) represent 33% of the observed dictionary attacks. This result lets us conclude that few dictionaries are shared by the attackers on the Internet.

Figure 5 is a matrix presentation of the vocabulary clusters. Each column and each line represents a vocabulary. The intersection of a column, representing vocabulary $\mathcal{A}$, and a line, representing vocabulary $\mathcal{B}$ is either a white or dark point, according to the distance between these two vocabularies. The dark color is used if the distance is lower than our threshold. Columns and lines have been ranked in such a way that similar vocabularies are placed close to each others.

Figure 5 shows two big clusters. These two clusters include more than 20% of the vocabularies. The biggest cluster (called A in the figure) includes 185 vocabularies: 183 vocabularies are exactly identical and made up of 8 words and the 2 others have different sizes: 140 and 611. All the distances in this cluster are 0: the vocabularies of 8 words are included in those of 140 and 611 words.

The other cluster (called B in the figure) is composed of 131 vocabularies: 111 are made up of 147 words and are identical; the other vocabularies have different sizes, between 13 and 623 words.

The third big cluster (called C in the figure) gathering 51 vocabularies is also noteworthy. The sizes of all these vocabularies are different, from 45 to 925 words, the average being 169 words per vocabulary.

We have also analyzed, for these three clusters, the time when the corresponding dictionaries have been used during the experiment. The results are presented in Fig. 6. Let us first note that these dictionaries have been used during a very long period of time. For example, the cluster A is associated to 64 different IP addresses. These addresses have been observed several times on the honeypot, at a regular basis, during the 419 days period of the experiment. A dictionary attack of this cluster has occurred in average every 2 days. It is noteworthy that none of the IP addresses of the cluster A is associated to the two other clusters. On the other hand, 17 among 38 addresses of the cluster C are also associated to the cluster B. Thus, these analyses lead us to conclude that these three clusters identify different communities that have used the same dictionary during the observation period.[7]

In a recent study reported in [20] about passwords and methods used in brute-force SSH attacks, dictionaries of the same size as in our study (including in particular 8 and 147 words) have been also observed. This also seems to confirm the fact that the same attack tools are shared in the community for performing brute-force SSH attacks.

In the next section, we analyze more precisely the dictionaries associated to each cluster.

4.4 Identification of dictionary kernels

Vocabularies inside each cluster are similar. They have probably been generated from the same dictionary or from close dictionaries. Of course, it is not possible to obtain the whole dictionary because some pairs may not have been used by the attacker. The idea is to reconstitute a part of this dictionary that we call *dictionary kernel*.

The dictionary kernel associated to a cluster c, denoted as $\mathcal{D}_d(c)$, is defined as the set of words shared by at least two vocabularies of this cluster. We did not consider the union of all the dictionaries because it would include also the pairs that have been tested only once, which is not significant. On the other hand, we did not consider the intersection of all the dictionaries to avoid the elimination of the pairs that have been tested by several dictionary attacks but not all of them, which would be restrictive.

The example of Fig. 7 presents four vocabularies v1, v2, v3 and v4. The associated dictionary kernel is identified by the grey surface.

We have analyzed the dictionary kernels associated to the 413 clusters, based on the examination of the pairs (username/password) composing these kernels. The username and the password may be words of a spoken language such as French or English but may also be words without any significance or words particularly chosen by the attackers.

Let us first note, that most of the dictionary kernels include many words without any significance, that do not belong to the English or French language: 3/4 of the dictionary kernels hold more than 75% of such words. Also, we have observed that the attackers did not adapt their dictionary to the geographical location of their victim. Indeed, although our

[7] Let us note that communities may refer to really different groups of machines but also different configurations of the same botnet.

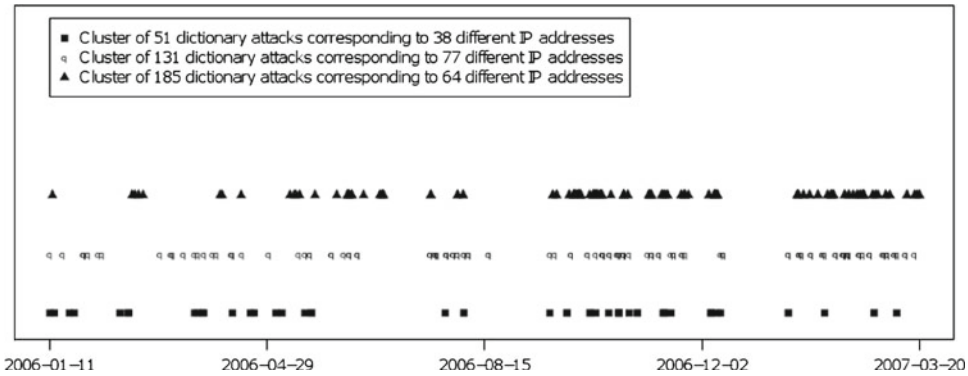

Fig. 6 Evolution of the use of the dictionaries

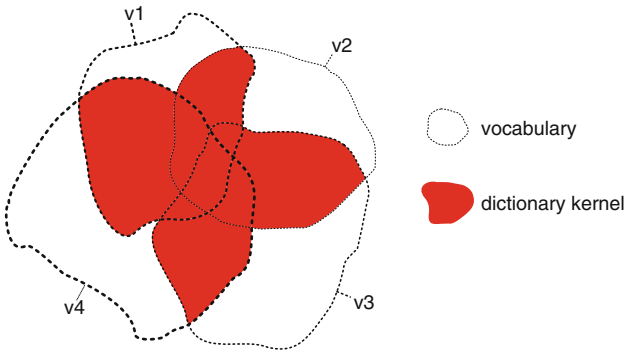

Fig. 7 Example of dictionary kernel for a cluster

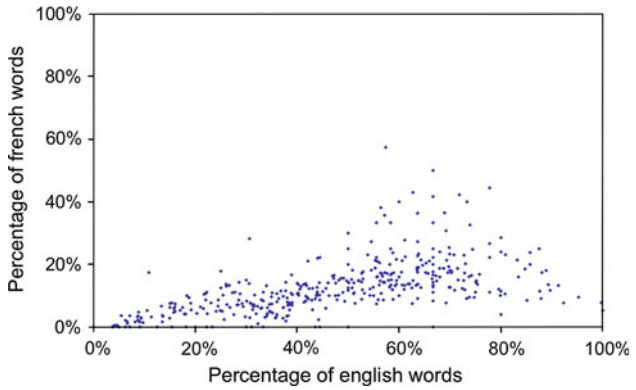

Fig. 8 Proportion of English and French words of dictionary kernels

honeypot was deployed in France, most the words included in the dictionaries are English. This is illustrated in Fig. 8 in which each dictionary is represented as a point. The X coordinate is the percentage of these words that are English words and the Y coordinate is the percentage of these words that are French words (the sum of the percentages may be higher than 100% because a word may be both English and French).

Table 4 gives the list of the usernames that were tested the most, for all the dictionary attacks. It appears that the `root` account is the most tested account (85,426 tries whatever the passwords, in 419 days). It can be observed that this table includes a lot of accounts created by default (`test, admin, guest, user, mysql, oracle,web, test, ftp`). We can conclude that the dictionaries are built by taking into account the specificities of these particular user accounts. Let us note that similar trends are reported in [29] and [33]. However, these papers do not present a detailed dictionary analysis as presented in this paper.

Table 4 Particular accounts per cluster

Classification	Username	Total number of connections	Number and % of kernels including this username
1	root	85,426	193 (46.7%)
2	admin	15,556	159 (38.5%)
3	test	5,615	201 (48.7%)
4	guest	2,425	118 (28.6%)
5	user	2,165	109 (26.4%)
6	mysql	1,671	118 (28.6%)
7	oracle	1,479	106 (25.7%)
8	ftp	1,283	98 (23.7%)
9	web	1,133	82 (19.9%)

As illustrated in Table 4, we can notice that, even if these accounts are the most tested, they are not tested by a large number of dictionary kernels. For example, the `root` account is only included in 193 among 413 dictionary kernels, for 85,426 connection attempts.

Finally, if we take a closer look at Table 5, we can notice that a large number of username/password pairs tested have identical username and password. Approximately, one third of the dictionary kernels include 75% of such pairs. This result lets us think that the attackers assume that an important proportion of the users on Internet behave in a such way.

4.5 Comparison with John the Ripper

John The Ripper is the most popular password cracking system that is widely available through the Internet. One of the questions that can be raised is: how similar are the dictionary

Table 5 List and frequency of most tested pairs

Classification	Number of connections	Username	Password
1	909	test	test
2	879	admin	admin
3	864	root	root
4	824	guest	guest
5	819	root	123456
6	790	user	user
7	757	root	password
8	706	mysql	mysql
9	676	richard	richard
10	663	oracle	oracle
11	660	sales	sales
12	659	test	123456
13	650	web	web
14	597	ftp	ftp
15	584	michael	michael
16	574	paul	paul

kernels identified in our experiment with the dictionary used by John the Ripper? Figure 9 plots for each dictionary kernel the percentage of words from John the Ripper dictionary[8] that are contained in each kernel.

The Y axis on the left reports the size of each kernel. The dictionary kernels are sorted by increasing order according to the percentage of coverage achieved with respect to John the Ripper. As can be seen, the intersection with John the Ripper is generally low (about 80% of the kernels contain less than 50% of the words used in John the Ripper dictionary). As a conclusion, the dictionaries used by the attackers observed in our experiment look generally different from John the Ripper dictionary. Only a few kernel dictionaries included a high percentage close to 100% of the words used by John the Ripper dictionary.

4.6 Dictionary attacks: conclusion

The main results of this experiment can be summarized as follows:

- Only a few number of dictionaries are shared among the attacker community to execute brute-force attacks.
- These dictionaries are different from the dictionary used in the popular John The Ripper tool, which seems to indicate that the attackers community also possesses its own dictionaries to brute-force passwords.
- The dictionary kernels identified in our study were used all along the experiment period and are thus stable in time.
- The attackers did not take into account the geographical location of their victim (they did not adapt the language used for their logins and passwords according to this location).
- The dictionaries include, for most of the couples (login/password), arbitrary words without significance but the most tested couples correspond to well-known accounts, often created by default.

[8] We used the dictionary available at: http://www.openwall.com/john. This dictionary contains 3,107 words.

5 The intrusion step

A dictionary attack is only a preliminary step that enables an attacker to perform an *intrusion* (a session which includes action connections, see 3.2). Among the 1,940 sessions previously identified, 210 are considered as intrusions. These intrusions are studied in this section.

The number of different IP source addresses corresponding to these 210 intrusions is 57. They targeted 21 different user accounts. Table 6 presents, for each user account, the number of intrusions, addresses and passwords associated. First, it can be noticed that several passwords may be associated to the same account. This is not surprising because one of the first operations performed by the attackers once connected to our honeypot was to change the password. It is interesting to note that none of these IP addresses *was used to perform dictionary attacks*. One possible conclusion is that the attackers on the Internet specialize their machines to some specific attacks. We can even suppose that the attackers are organized in communities in order to perform their actions in concert by using different sets of machines to perform their attacks.

In the following paragraphs, we analyze the behavior of the intruders. First, we characterize the nature of the intruders: humans or automatic tools. Then we give an overview of the different kinds of activities the intruders have carried out on the honeypot. Finally, we discuss their skills.

5.1 Nature of intruders: humans or automatic tools

The intruders may be human beings or automatic tools that only perform simple tasks. We have considered two main criteria to identify activities performed by humans: (1) typos and (2) the mode of data transmission between the user and the honeypot.

Typos constitute a first element that seems to characterize human beings. To correct the typos, the user regularly hits the backspace key (127 ASCII code). Thus, the first possibility to determine the nature of the intruders is to analyze the sequence of keystrokes hit by the attackers and check if the backspace key is included. We assume that it is unlikely that

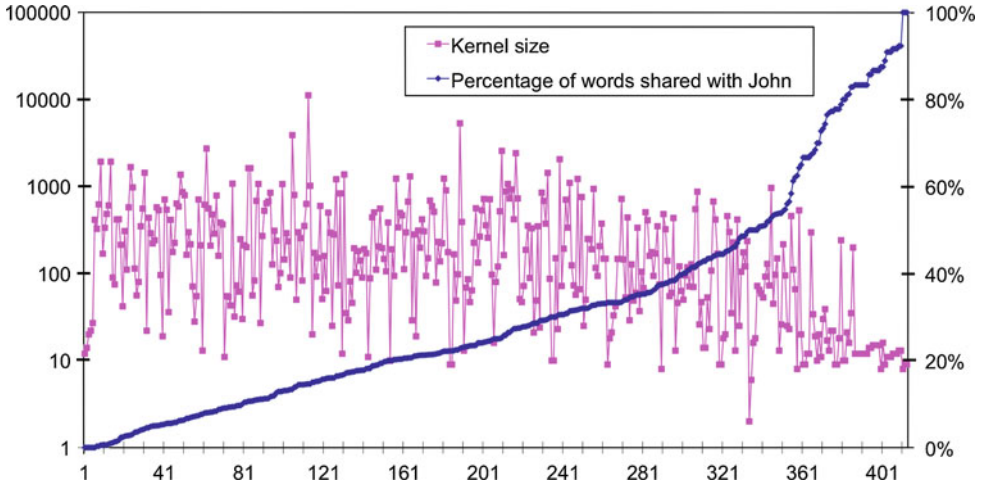

Fig. 9 Percentage of words shared with John the Ripper

Table 6 Number of intrusions per account

Account	Number of intrusions	Number of passwords	Number of addresses
C_1	11	2	6
C_2	3	2	2
C_3	39	2	10
C_4	2	2	1
C_5	3	2	2
C_6	22	2	7
C_7	21	2	2
C_8	32	2	3
C_9	14	2	6
C_{10}	2	1	2
C_{11}	3	2	2
C_{12}	17	3	3
C_{13}	4	2	1
C_{14}	3	2	2
C_{15}	1	1	1
C_{16}	3	2	1
C_{17}	19	1	7
C_{18}	2	2	1
C_{19}	1	1	1
C_{20}	5	1	3
C_{21}	2	1	1

the automatic tools will try to imitate the human behavior by including deliberately such typos.

In some cases, the intruder uses very few commands and is very unlikely to make typos. Moreover, a very careful human being may perfectly enter his commands without making typos. In these situations, the analysis of the protocol used to send the data through the network can help us to distinguish between a human being and an automatic tool. Indeed,

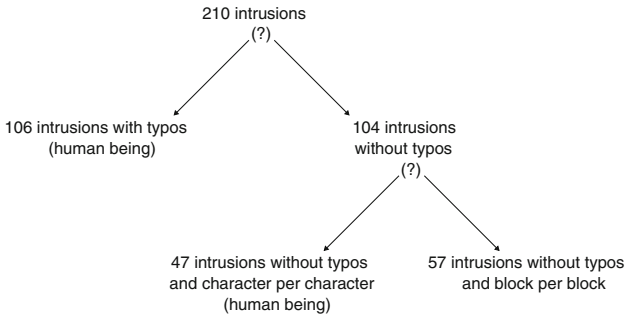

Fig. 10 Nature of intruders

through SSH connections, the transmission of data between the client and the server is asynchronous. Most of the time, the implementations of the SSH clients use the `select()` function to get the key hit by the user. When the user hits a key this function ends and the client sends the character entered to the server. In the case of a copy/paste in the terminal of the user, a buffer including all the data from the copy/paste action is sent through the SSH connection. In our implementation, the data sent through the SSH connection are received by the `tty_read()` function. Thus, we consider that, if this function returns more than one character, the data have been transferred by a copy/paste operation. In summary, we consider that an intrusion without any typos and for which all the data are transferred per blocks is performed by an automatic tool. If an intrusion transfers data character per character and also by blocks, we consider that it is performed by a human being, the transfers per blocks being due to keyboards shortcuts.

By applying these two criteria (typos and data transfer mode), we obtained the classification of intrusions presented in Fig. 10. 106 among 210 intrusions include typos. These 106 intrusions were very likely performed by human beings. Regarding the 104 intrusions that do not include typos, 47 of them have used the character per character data transmission

mode. Thus, according to our second criterion, they have been carried out by human beings. Regarding the other intrusions, the activities carried out do not enable us to conclude on the nature of the intruder. Indeed, they include too few commands (only one in general, such as `w` for example). To conclude, more than 75% of the intrusions were realized by human beings.

5.2 Intruder activities

In this section, we analyze the different commands used by the intruders to identify the different types of activities they have carried out, considering the 210 intrusions.

Let us first notice that all the intruders start their activity by changing the password of the account they have just broken. Changing the password enables them to be the only one who can connect and use this account on the machine. This is surprising at some extent because this modification of password betrays their activity on the honeypot. If a legitimate user regularly uses this account, he will easily detect that he has been attacked.

The second activity performed by the intruders is the downloading of malicious programs from the Internet. The command mostly used to download programs is `wget`. 96 among 210 intrusions used it. Let us recall that the outgoing connections are blocked, so a connection to a remote web server cannot succeed using such a command. However, the intruders can upload their malicious programs from Internet to the honeypot by using another SSH connection thanks to the `sftp` command. Surprisingly, only 30% of the intruders thought about uploading their malicious software through the SSH connection.

Two explanations seem relevant. We can first imagine that the intruders use cookbooks, available on the Internet. These cookbooks propose some attack methods. Most of the time, they use the `wget` command in order to download files and the intruders that use the cookbooks do not necessarily know other methods to download files. Regarding the second explanation, we can imagine that the intruder suspected that our machine is not a "regular" machine when he realized that the `wget` command fails and decided to give up. In fact, the first explanation seems to be the good one. As a matter of fact, we noticed that, the sessions corresponding to the activity of the intruders that give up after the `wget` command fails include a lot of copy/paste operations. This lets us strongly suppose that such intruders use cookbooks. Furthermore, we noticed that these intruders generally come back again to the honeypot several days after their first intrusion and that they try again the same sequence of commands. Some of them came back several times.

When the intruders have succeeded in downloading their malware on the honeypot, they uncompress their files. Approximately 75% of the intruders do not use directories of the broken account for that but rather, standard writable directories of the Unix system, such as `/tmp`, `/var/tmp` or `/dev/shm`. These directories are used and shared by all the users of a Unix system, so it is difficult to identify the malicious activity of the intruders in these directories. In these standard directories, the intruders create their own hidden directories (directories whose name starts with the '.' character) or create the " " directory. This is surprising as, on one hand, by doing so the intruders try to hide their activities; however, on the other hand, by changing the password of the account they broke, they can be easily detected.

We identified three classes of activity of the intruders. The first one is the scanning activity. The intruders did not scan all the ports but specifically the SSH port, in order to find other machines that propose this service. Let us note that the scanning was never realized on the local network. It seems that the intruders were not interested in attacking the local network of our honeypot, probably in order to hide as much as possible their activity. More probably, their objective was to use our honeypot as a stepping stone to attack remote targets. Furthermore, this scanning activity is dedicated to the SSH service, just as if the intruders were specialized in this particular task. The second activity consists in using `irc` clients. These clients are generally used in the deployment of botnets. Thanks to `irc` communications, generally, the bots communicate with some master servers that send propagation or attack orders to the bots. This activity is thus clearly dedicated to the building of botnets in the Internet. Let us note that similar conclusions about attackers behavior have been reported in [33], during a shorter observation period (2 months).

The third activity consists in getting the `root` privileges. Surprisingly, very few intruders (only 3) have tried to get these privileges. Two different malware have been used for that purpose. The first one exploits one vulnerability of the system call `mremap`[36] and one other that affects the manager in charge of the heap of the processes [35]. These exploits could not be successful on our honeypot because of a kernel version incompatibility. The intruder should have realized this incompatibility because he checked the version of the system (`uname -a` command). Nevertheless, the intruder launched the exploit, that failed. The second malware exploits a vulnerability in the `ld` program. This exploit has been tested by 3 intruders, but was not successful.

5.3 Intruders skill and intention

Intruders can be classified into two categories: script kiddies (inexperienced people who probably use malware downloaded on the Internet without necessarily understanding them) and black hats (security experts who are really dangerous and who perfectly understand what they do).

Our experiment lets us think that most of the attackers we observed on the honeypot are script kiddies. Most of them even seem not to be familiar with the Unix access rights (e.g., they try to delete files that they obviously cannot delete, or they try to kill processes for which they don't own the right privileges).

Most of them did not delete the history files (such as `.bash_history`, located in the home-directory) that log the different commands used by the intruder.

Most of the intruders tried to identify the system version, i.e., tried to obtain information regarding the hardware (by means of the `/proc/cpuinfo` file for example). But none of them checked, thanks to well-known techniques, if `VMware` was installed on the system [16] (which could enable the intruder to detect a honeypot).

6 Conclusion and future work

In this paper, we have presented the main lessons learned from the development and deployment of a high-interaction honeypot aimed at the observation and analysis of the activities carried out by the attackers on the Internet, including humans and automatic tools. The results are based on an experiment carried out over a long period of time (419 days), during which we have observed: (1) the various steps that lead an attacker to successfully break into a vulnerable machine, and (2) his behavior once he has managed to take control over the machine. We have considered the case of attacks carried out through the SSH service running on a Linux system. Two classes of attacks have been investigated: (1) brute force dictionary attacks aimed at gaining access at the target system, and (2) intrusions corresponding to interactive attacks performed with successful logins.

The detailed analysis of the observed attacks reveals several interesting facts. Considering the analysis of dictionary attacks, we have identified three main dictionaries shared by the attackers that have been used during a very long period of time, including a large number of username/password pairs that are commonly used in Unix and Windows systems, as well pairs with identical user names and passwords. Furthermore, these dictionaries appear to be different from John the Ripper dictionary tool that is widely available on the Internet. The attackers do not seem to adapt their dictionaries to the geographic location of their victim. A noteworthy observation from our experiment was that the IP addresses that performed the dictionary attacks were completely distinct from those used for the intrusions. This seems to confirm the existence of machines or communities that are dedicated to specific types of attacks.

Considering the analysis of the activities logged during the intrusions, we have showed that these activities have been carried out by human beings. Most of the observed intrusions were only partially automated and carried out by script kiddies. This is very different from what can be observed against other ports, such as 445, 139 and others, where worms have been designed to completely carry out the tasks required for the infection and propagation. The main activities carried out by the attackers consist in (1) executing `irc` clients and (2) scanning the SSH port of other machines on the Internet. These two activities are clearly dedicated to the building of botnets in the Internet.

Of course, our experiment at this stage is not sufficient to derive general conclusions. We would need to deploy the honeypot at other different locations. Also, it would be worth running the same experiment by opening other vulnerabilities into the system and verifying if the identified steps remain the same, if the types of attackers are similar. This is something that we are in the process of assessing.

Acknowledgments This work has been partially supported by the European research project CRUTIAL (IST-027513), the European Network of Excellence ReSIST (IST-026764), and a research action CADHo funded by the French ACI "Securite & Informatique".

References

1. Alberdi, I., Gabès, J., Jamtel, E.L.: UberLogger : un observatoire niveau noyau pour la lutte informatique défensive. In: Symposium sur la Sécurité des Technologies de l'Information et des Communications (2005)
2. Alata, E.: Observation, caractérisation et modélisation de processus d'attaques sur Internet. PhD thesis, LAAS-CNRS, Toulouse (2007)
3. Alata, E., Nicomette, V., Kaâniche, M., Dacier, M., Herrb, M.: Lessons learned from the deployment of a high-interaction honeypot. In: 6th European Dependable Computing Conference (EDCC'06), pp. 39–44, Coimbra (2006)
4. Bakos, G.: Tiny honeypot- resource consumption for the good guys. http://www.alpinista.org/thp
5. Bailey, M., Cooke, E., Jahanian, F., Nazario, J., Watson, D.: The Internet motivon sensor: a distributed blackhole monitoring system. In: 12th Annual Network and Distributed System Security Symposium (NDSS), San Diego (2005)
6. Bellard, F.: Qemu: A fast and portable dynamic translator. In: USENIX Annual Technical Conference, FREE Track, Anaheim, pp. 41–45 (2005)
7. Baecher, P., Koetter, M., Holz, T., Dornseif, M., Freiling, F.: The Nepenthes platform: an efficient approach to collect malware. In: Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection, RAID06, Hamburg, Germany, 20–22 September 2006. Lecture Notes in Computer Science, vol. 4219. Springer, Berlin (2006)
8. Cymru Team: The darknet project. http://www.cymru.com/Darknet/index.html
9. Dike, J.: User Mode Linux, 2nd edn. Prentice Hall, USA (2006)
10. Dressler, F., Jaegers, W., German, R.: Flow-based worm detection using correlated honeypot logs. In: Braun, T., Carle, G., Stiller, B. (eds.) 15. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007), pp. 181–186, Bern (2007). iVDE
11. Stone, R., Song, D., Malan, R.: A snapshot of global Internet worm activity. In: FIRST Conference on Computer Security Incident Handling and Response (2002)

12. Dshield; Cooperative Network Security Community—Internet Security. http://www.dshield.org
13. Secure, F.: Malware information page: Allaple.a (2006). http://www.f-secure.com/v-descs/allaple_a.shtml
14. Bos, H., Portokalidis, G., Slowinska, A.: Argos: An emulator for fingerprinting zero-day attacks. In: ACM SIGOPS, EuroSys (2006)
15. The Grugq. The Design and Implementation of Userland Eexc Bugtraq (2004). http://archive.cert.uni-stuttgart.de/bugtraq/2004/01/msg00002.html
16. Holz, T., Raynal, F.: Detecting honeypots and other suspicious environments. In: 6th IEEE Workshop on Information Assurance and Security (IAW'05), pp. 29–36, West Point (2005)
17. Labbé, C., Labbé, D.: Inter-textual distance and authorship attribution Corneille and Molière. J. Quant. Ling. **8**(3), 213–231 (2001)
18. Ramzan, Z., Jacobsson, M.: Crimeware: Understanding New Attacks and Defenses. Symantec Press, Addison Wesley, Boston (2008)
19. Moore, D.: Network telescopes: observing small or distant security events. In: Proceedings of 11th Usenix Security Symposium, San Francisco (2002)
20. Owens, J., Matthews, J.: A Study of passwords and methods used in Brute-Force ssh attacks. http://people.clarkson.edu/jnm/publications/leet08.pdf
21. Pouget, F., Dacier, M., Pham, V.H.: Understanding Threats: A prerequisite to enhance survivability of computing systems. In: Proceedings of IISW'04, International Infrastructure Survivability Workshop 2004, in conjunction with the 25th IEEE International Real-Time Systems Symposium (RTSS 04) 5–8 December 2004, Lisbonne, Portugal (2004)
22. Pouget, F., Dacier, M., Pham, V.H.: Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In: E-Crime and Computer Conference (Ecce '05), Monaco, mars (2005)
23. Provos, N., Holz, T.: Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Addison Wesley, Reading (2007)
24. Pouget, F.: Distributed system of honeypot sensors: discrimination and correlative analysis of attack processes. PhD thesis, Institut Eurecom (2006)
25. Provos, N.: A virtual honeypot framework. In: 13th Usenix Security Symposium, San Diego (2004)
26. Honeynet Project. Know Your Enemy: Honeywall cdrom roo. (2005). http://www.honeynet.org
27. Panjwani, S., Tan, S., Jarrin, K., Cukier, M.: An experimental evaluation to determine if port scans are precursors to an attack. In: Proceedings of the International Symposium on Dependable Systems and Networks (DSN-2005), pp. 602–611, Yokohama 28 June–1 July, 2005
28. Pang, R., Yegneswaran, V., Barford, P., Paxson, V., Peterson, L.: Characteristics of Internet background radiation. In: Proceedings Of the 4th ACM SIGCOMM Conference on Internet Measurement, pp. 27–40. ACM Press, New York (2004)
29. Ramsbrock, D., Berthier, R., Cukier, M.: Profiling attacker behavior following ssh compromises. In: 37th Annual IEEE/IFIP International Conference on Dep endable Systems and Networks (DSN), pp. 119–124, Edinburg, 25–28 June, 2007
30. Rajad, M.A., Jay, Z., Monrose, F., Terzis, A.: A multifaceted approach to understanding the Botnet phenomenon. In: Internet Measurement Conference, pp. 41–52 (2006)
31. SAL. Secure auditing for linux. http://www.secureaudit.sourceforge.net
32. Sans Institute. 2007 top-20 2007 security risks (2007 annual update). http://www.sans.org/top20/2007
33. Seifert, C.: Analyzing Malicious SSH Login Attempts. SecurityFocus (2006). http://www.securityfocus.com/infocus/1876
34. Spitzner, L.: Honeypots: Tracking Hackers. Addison Wesley, Reading (2002)
35. Us- Cert. Linux kernel do_brk() function contains integer overflow. http://www.kb.cert.org/vuls/id/981222
36. Us- Cert. Linux kernel mremap(2) system call does not propertly check return value from do_munmap() function. www.kb.cert.org/vuls/id/981222
37. Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voekler, G.M., Savage, S.: Scalability, fidelity, and containment in the Potemkin Virtual Honeyfarm. ACM SIGOPS Oper. Syst. Rev. **39**(5), 148–162 (2005)
38. Vmware. http://www.vmware.com
39. Williams, R.: An extremely fast Ziv-Lempel data compression algorithm. In: Data Compression Conference, pp. 362–371 (1991)
40. Zou, C.C., Cunningham, R.: Honeypot-aware advanced Botnet construction and maintenance. In: International Conference on Dependable Systems and Networks (Dsn '06), pp. 199–208 (2006)
41. Zhuge, J., Holz, T., Han, X., Song, C., Zou, W.: Collecting autonomous spreading malware using high-interaction honeypots. In: Proceedings of 9th International Conference on Information and Communications Security (ICICS'07), Zhengzhou, China (2007)