Shark: Spy Honeypot with Advanced Redirection Kit

Ion Alberdi Éric Alata Philippe Owezarski Vincent Nicomette Mohamed Kaâniche
LAAS-CNRS
7 avenue du Colonel Roche - 31077 Toulouse Cedex 4
{ialberdi,ealata,owe,nicomett,kaaniche}@laas.fr

Abstract

Botnets represent a big threat for the Quality of Internet Services, as they are the source of many malicious activities (DoS attacks, Spam, secret violation, etc). Our work focuses on trying to detect, observe and disable botnets by infiltrating them. For this purpose, this study focuses on discovering the behavior of malware which drives the bots. The paper presents the design of an experimental platform which takes advantage of the honeypot technology for downloading malware, and especially, serves as an infrastructure allowing the observation and the analysis of the collected malware behaviors, which is the contribution of this paper. For this purpose, we are designing SHARK, a Spy Honeypot with Advanced Redirection Kit. The redirection kit aims at coping with some liability constraints which forbid one to send out malicious traffic which could pollute or corrupt any other machines. This paper presents first results on the gathered malware observation and analysis, and some conclusions on botnet behaviors, thus demonstrating the interest and pertinence of this platform.

keywords: honeypots, botnet, malware, worms.

I. Introduction

Since the Internet is becoming a multi-services network for different kinds of applications and media with different requirements, Distributed Denial of Service Attacks (DDoS) introduce a threat of significant importance. These kinds of attacks are very frequent. The most famous example of DDoS attack is 5Bcertainly the one launched on October 17th 2002 against the 13 root DNS servers. The attack managed to bring down 7 of them. A significant part of Internet hosts were then unable to take advantage of naming service for a long duration[1]. More generally, [10] counted the occurrence of at least 12,000 Denial of Service (DoS) attacks against 5,000 different hosts during a three weeks period in 2001. According to statistics reported in [13], from January to June 2006 an average of 66,000 active bots launching 6,110 DoS attacks per day was observerd. This study, as well as [5], also points out the role of some machines (also called bots or zombies) corrupted by blackhats in this new kind of threat. It appears that some blackhats are controlling several thousands of machines which can be used jointly for sending DDoS attacks. Corrupting new machines is then of high importance for blackhats to gain more power. If such activity is so widespread, it is probably because botnet renting, for sending spam or DDoS for instance, can be a very profitable activity. [9] evaluated that botnet owners have made more profit than the whole antivirus software industry in 2005. In addition, corrupting a significant number of machines, can be sometimes as feasible as corrupting a single one. Nowadays, the uniformity of software binaries installed on a large number of interconnected computers produces a dramatic weakness. Indeed, a bug on such software that can lead to arbitrary code execution, is exploitable on all the instances of this software, and therefore, on all the reachable computers on which this software is installed.

The threats that such botnets represent against the Quality of Internet Services have to be addressed and ideally removed. Thus, protecting networks against DDoS attacks is one of the strategic problems to be solved. Up to now, usual protection approaches rely on the use of Intrusion Detection Systems (IDS). The drawback with such IDS is their postactive or reactive nature. They can work only when attacks occur [14]. Complementary solutions for protecting networks against botnets and DDoS attacks must be as proactive as possible, i.e., annihilate these threats

[1]A similar attack took place on February 6th 2007. Fortunately, it was not as successful.

as soon as possible. The ideal solution would consist in developing software applications without exploitable bug, or at least making people apply corrective patches frequently. Nevertheless, past experiences demonstrated that those considerations are not realistic. Because of current Internet users habits that result in:

- widely using the same operating system and software binaries,
- neglecting the process of applying security patches,

we do think it would be utopian to try to fight against botnet creation. Our work then focuses on trying to detect, observe and disable botnets by infiltrating them. We then expect to find a way for disabling DDoS attacks before they occur (and then propose a new proactive defense approach). This paper presents our first work towards this long term research objective.

For this purpose, we propose to use monitoring techniques for observing and analyzing attacks, blackhats and the spread of botnets. Possible monitoring techniques can be classical network passive monitoring technologies, but also, in the case of attacks, more specialized tools as honeypots. In this context, three different techniques can be distinguished for detecting botnets:

1) Observing the malicious traffic sent by bots (floodings, spam, . . .) with the aim of detecting botnets. Up to our knowledge, nobody applied this method yet. This seems still to be an unexplored way.
2) Observing the `Command & Conquer Channel`, (C&C) which is the channel used by the blackhat to control the machines he corrupted. Several projects already applied this approach for the purpose of detecting botnets [1], [8], [12]. Most of them analyze Internet traffic and identify botnets by observing non standard IRC commands, or by measuring the reactivity of IRC clients (which helps to differentiate humans from bots). However, these projects limit their observations to a single kind of C&C channels, i.e. unencrypted IRC. In addition, they assumed that IRC traffic due to botnets is only generated on the standard IRC port number, which is a significant lack as described by [3] for example.
3) Capturing and analyzing the malware which corrupts a machine and makes it a bot. Several proposals have been made according to this approach. They generally take advantage of low and high interaction honeypots, jointly or not, depending on their objectives [7]. For example, [3] proposes to install a high interaction honeypot in a network, and protect this network from a possible propagation by limiting the output bandwidth and filtering connection attempts to the LAN. This system fully simulates a bot behavior. But it presents major drawbacks, the main being its lack of protection against damages that the infected honeypot could cause to the Internet. Indeed, even if LAN protection is guaranteed and high rate DoS attacks cannot be launched from the honeypot, potential malware can still propagate through the Internet from this honeypot. Another methodology presented in [5] aims at behaving as a bot in order to infiltrate a botnet. For this purpose, it starts by downloading the botnet malware, analyzes it and extracts information on the C&C channel. Then it emulates the bot behavior on the channel[2] to gain information. It is of course less accurate than the method described in [3] which proposes to interpret malware commands. But on the other side, it allows an automatic gathering of malware, and ensures not to send out malicious traffic to the Internet.

Our study focuses on discovering the behavior of malware which drives the bots [6]. The paper presents the design of an experimental platform which takes advantage of the honeypot technology for downloading malware [2] , as well, and this is the contribution of this paper, as an infrastructure allowing the observation and the analysis of the collected malware behaviors. For this purpose, we introduce SHARK, a Spy Honeypot with Advanced Redirection Kit. The redirection kit aims at coping with some liability[3] constraints, e.g., in France, that forbid the honeypot owners to send out malicious traffic which could pollute or corrupt any third-party machines[4]. As it is law enforced, SHARK works in a restrictive and controlled environment. It interacts with other machines but must control the traffic sent out, to avoid being suit. This limitation is mitigated in SHARK by the redirection kit which proposes both static and dynamic redirection principles. Section II presents the design of this platform especially detailing the safe simulation environment, and the redirection mechanisms. Section III presents its implementation. The section IV focuses on preliminary results on the gathered malware observation and analysis, and some conclusions on botnet behaviors, thus demonstrating the interest and pertinence of this platform. Finally, section V concludes the paper by describing some limits of the system which will be improved in future work. It also depicts, using few examples, how these results could be exploited for the design of a global proactive protection system for the Internet.

[2]Without simulating its other actions (like propagation attempts). In other words, it only offers a partial level of interaction.

[3]As far as we know, the law seems to be very similar in most of the countries: USA, All western Europe countries, Japan, etc.

[4]The owner of a machine connected to the Internet is responsible of the traffic his machine generates... even the one that is sent out because of the blackhat (who has some control on the machine) requests.

II. Design

Our methodology aims at providing a particular runtime environment that allows the behavior of an attack to be simulated. That simulation environment must prevent the malware from sending out malicious packets to the Internet. This restriction could be discussed: why should we prevent the malware from sending out packets to the Internet if the observation of these packets is fundamental to understand the structure and behavior of botnets ? The French law is clear: this traffic is illegal, and therefore this work aims at obtaining information on botnets without making illegal actions.

Our environment has to face two main issues:

- How to obtain malware?
- How to design and implement such a restrictive runtime environment?

A. How to download malware?

One solution for collecting malware consists in voluntarily letting it infect a computer on which a honeypot is running. Two types of malware propagation techniques can be distinguished according to the kind of vulnerabilities they exploit:

- **Client side vulnerabilities:** they target client software and require a human interaction (such as clicking on an infected document attached to an email for example).
- **Server side vulnerabilities:** they target server software and do not require any human interaction.

`nepenthes` [2] is a tool for collecting malware that exploits server side vulnerabilities. We have decided to first analyze malware collected with this software.

B. How to build a restrictive and controlled runtime environment?

The second step of our methodology consists in running collected malware for observing and understanding their behavior. To keep control on such possible harmful executions, some parts of malware interactions have to be simulated. The quality of simulations depends on the level of interaction provided by the environment. By developing their own IRC client to observe a botnet, [5] built a simulation environment with a lower interaction level than the one that is expected if the malware is executed without any control. Nevertheless, even if executing malware without any control provides the best possible level of interaction for observing them, such actions can also generate illegal malicious traffic. We then built our runtime environment with the aim of making it as interactive as possible with respect to all legal constraints.

The execution of such malware may provoke non negligible modifications on the operating system and applications installed. Those modifications could have an impact on the execution of any software running afterwards, and especially some other malware. Then we decided to reinitialize the whole operating system once a malware has been executed.

As already explained, our runtime environment is built with a full control on all communications initiated from the malware. Communications on the C&C channel do not provoke any particular degradation on any information system and thus do not present any risk from our point of view. We therefore decided to let the malware talk through the real C&C channel when it is available, or try to simulate it otherwise[5]. On the other hand, malicious traffic, corresponding to DDOS or spam, represents obvious threats and should therefore be simulated.

The last issue to cope with concerns the identification of the different flows initiated from the malware. We adopted an iterative approach by redirecting the information flows initiated by the malware towards a local machine under our control. This machine simulates the service requested by these flows. These information flows are analyzed in order to decide whether they can be sent to the Internet, redirected or blocked during the next execution of the malware.

1) Static redirection: The static redirection consists in a set of rules controlling the redirection of some connections, that cannot be changed during the same malware execution. One of the most important statically redirected flows are DNS flows. We chose to use our own DNS server and redirect all the DNS requests from the malware to this server. Thanks to that configuration, we can associate to this DNS name the IP address of our choice:

- the real IP address associated to this DNS name, or
- a "fake" IP address, for example a computer in our LAN.

To sum up, our system redirects traffic, first using the DNS protocol. The other redirection techniques arise at levels 3 and 4. For example, all tcp flows towards port 6667 are redirected to the IP ip1, port p1.

Thanks to these redirections, we can analyze the interactions between the malware and computers it tries to connect to. It then helps us to define the execution flow policy for this malware future executions.

But static redirection presents serious drawbacks when considering malware propagation attempts. Malware usually tries to propagate by scanning Internet computers. Scanned addresses cannot be predicted as they can be randomly chosen, or dynamically designated by the botnet administrator. Therefore with static redirections, we can

[5]For example, when the url is outdated or the server unreachable… in the case of an IRC C&C channel.

only enforce redirections based on level 4 port parameter. Then, all the scanning flows can be redirected either to one computer, which would be unrealistic, or to none, which represents the lowest possible interaction level, and therefore provides little information on scanning impact. We therefore implemented a dynamic redirection technique.

2) Dynamic redirection: We have designed and implemented a selective mechanism which allows connections from the honeypot towards Internet to be automatically and dynamically redirected. The goal is to make the attacker have the illusion that he can connect from the honeypot to the Internet whereas, in reality, the connections are simply redirected towards another honeypot. The originality of our method is the dynamicity of this redirection mechanism.

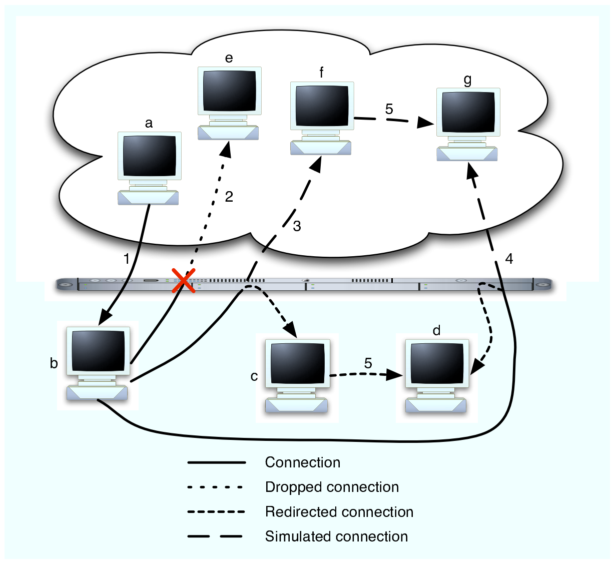

Fig. 1. Example of redirection

figure

Let us take the example presented in Figure 1. An attacker, from Internet host a, breaks into honeypot b (connection 1). From this honeypot, the attacker then tries to break into Internet host e thanks to connection 2. This connection is blocked by our mechanism. The attacker then tries another connection 3 towards Internet host f. This connection is accepted and automatically redirected towards honeypot c. The attacker has the illusion that his connection to f has succeeded whereas it has been redirected to another honeypot. The attacker finally initiates another connection (5) to Internet host g from host f. This connection is also accepted and redirected towards honeypot d.

This mechanism is interesting because it allows the activity of the attacker to be observed on the different hosts that he is supposed to control. In general, a honeypot allows the activity of the attacker to be observed at only one side of the connection. The other connection end is the machine that interacts with the honeypot. For all redirected connections, we can observe an attacker at both connection ends. On the previous example, the attacker is connected from host b to host c and it is possible to observe him on both hosts b and c.

Of course, this mechanism must be as reliable as possible so that we can collect data really representative of the behavior of the attackers. Thus, the implementation of this mechanism must have the following properties:

- it must be adaptable according to the needs of the administrator.
- it must be hidden as much as possible in such a way that the attacker cannot suspect anything strange.
- it must not increase in a visible way the latency of the communications.

III. Implementation

To build our environment, we needed to implement first the malware environment execution, and the static redirection mechanism, and then the dynamic redirection.

A. Malware environment and static redirection

The large majority of interconnected computers use the Windows XP operating system. We therefore chose to use it to create the execution environment. In order to reinitialize our operating system after each malware execution, we chose to use the VmWare x86 architecture virtualization software. This software makes possible to take a snapshot of the state of the virtualized uninfected computer and go back to this state when necessary. Therefore, before executing a malware, we only need to take a snapshot of the uninfected operating system and go back to this snapshot before launching another execution.

We chose the GNU/Linux operating system as host system, essentially because of its firewall *netfilter* and because the VmWare software supports it.

The static redirection has been implemented thanks to the Destination Network Address Translation functionality offered by the GNU/Linux firewall *netfilter*. The selected DNS server is *bind9*. *nc* is used to capture the first payloads sent by a malware after a tcp handshake .

B. Dynamic redirection

The dynamic redirection mechanism has been implemented in the Gnu/Linux operating system. Nevertheless, it has been designed in such a way that it can be easily adapted to other systems.

As illustrated in figure 2 the mechanism includes three components:

- the `redirection module` (inside the kernel) extracts packets.
- the `dialog_handler` decides whether the extracted packets must be redirected or not.
- the `dialog_tracker` is a link between the redirection module and the `dialog_handler`.

The design of the `dialog_handler` and the `dialog_tracker` is simple and does not reveal any particular difficulty. Hence, this paragraph only focuses on the module implemented in the kernel.

The redirection module must extract packets in such a way that they can be redirected or blocked. To do so, our module interacts with the `netfilter` component of the kernel [11]. This component is a firewall which includes five chains. Each chain is used to intercept and possibly modify packets at different stages on their way through the IP stack (`INPUT`, `OUTPUT`, `FORWARD`, `PREROUTING` and `POSTROUTING`). A set of particular functions named `hooks` is associated to each chain. Each hook has a particular role and processes the packets that pass through the chain. For example, the hook `conntrack` updates the state-machine of the connection corresponding to the processed packet. The hooks associated to a chain are ordered by priority. Hence, in a chain, the hook with the highest priority processes packets before all others and the hook with the lowest priority processes packets after all others.

In order to implement our redirection mechanism, we have developed two hooks and inserted them between the hook `conntrack` and the hook `nat_dst` (which changes the destination address of a packet), in the `PREROUTING` chain. Our first hook is in charge of extracting packets and sending them to the `dialog_tracker` in user space, in order to decide whether they have to be redirected or not, whereas our second hook is in charge of tagging the corresponding connections as “redirected” if the decision to redirect them has been taken. We do not systematically redirect all the connections initiated from the honeypot to Internet. Most of them are blocked and only a few of them are redirected.

In fact, thanks to the hook `conntrack` of netfilter (which associates a state-machine per pending connection), the redirection of a whole connection only requires the redirection of the first packet of this connection (the other packets are automatically processed as the first one). Thus, for each connection, our first hook only extracts the first packet and sends it to the `dialog_tracker`[6]. Then, the `dialog_tracker` forwards the packet to the `dialog_handler`, which decides if this packet has to be redirected or blocked. This decision may be evaluated according to different rules such as for example “a connection upon 10 or 100 is redirected, the others are blocked”[7]. When the decision is taken, the `dialog_tracker` informs the kernel module through a `netlink` socket that the corresponding connection has to be tagged as “redirected” and the packet is re-injected into the next hook of the chain, which is our second hook. This second hook is simply in charge of tagging the corresponding connection as “redirected”. The packet is then re-injected in the list of hooks of the corresponding chain. One of them is the hook `nat_dst` which indeed redirects packets of connections tagged as “redirected”, by modifying the destination address of the packet (the destination address is changed to the address of one of our honeypots). So that this modification is correctly made, the `nat_dst` hook must be configured. This configuration is made through a rule inserted thanks to the `iptables` command. For example, the following rule configures this hook to redirect to the machine 192.168.254.3 all packets of a connection tagged as “redirected” with the tag `0x03FEA8C0`:

```
iptables -t nat -A PREROUTING \
    -m connmark --mark 0x03FEA8C0 -j \
    DNAT --to-destination 192.168.254.3
```

There must be as many `iptables` rules as possible redirections, and as many redirections as different tags. In this way, a tagged packet of a connection is redirected according to the tag.

The mechanism presented in this section is summarized in the figure 2. This figure also presents the progress of the first packet of a new connection through our mechanism.

C. Deployed experimental set-up

The global system is depicted in figure 3. Let’s illustrate this system by describing a possible execution scenario. First, all outgoing flows are blocked, except DNS requests that are redirected to our server. We launch the malware execution (M1) and observe that a DNS request is sent to get the IP address of DNSName1,IP1[8]. Let us assume DNSName1 exists[9]. Once the malware gets the IP1 address it sends a TCP SYN packet to the port p1 of IP1. We then configure the *static redirection* of the flow (flow 1 of figure 3) and relaunch the malware. Our computer (M2) will handshake the malware and receive first packets containing payloads with:

[6]We can recognize the first packet because the hook `conntrack` which is just before our first hook tags the first packet of each connection as `NEW`.

[7]Definition of this decision process is part of future work.

[8]All flows generated by the malware are observed by a wireshark (`http://www.wireshark.org`) network sniffer.

[9]If DNSName1 is outdated we change the configuration of our DNS server so that he claims one of our computers to be associated to this url, and relaunch the malware.

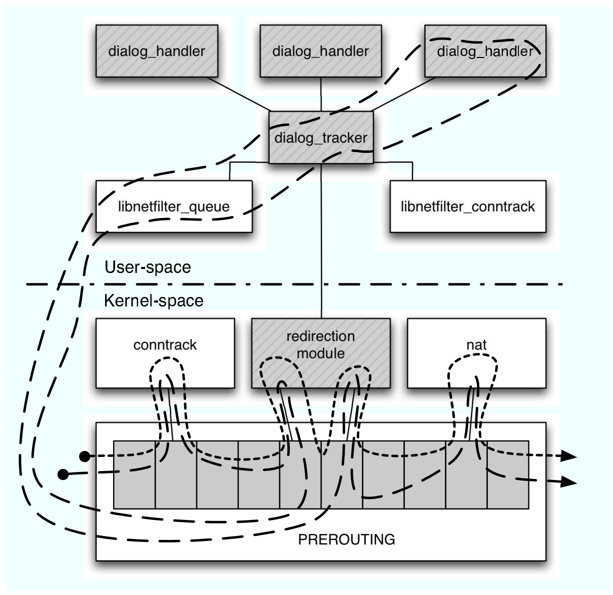

Fig. 2. Mechanism of redirection

figure

```
USER <user1>
NICK <nick1>
```

We conclude that this flow (IP1,p1) is part of a C&C channel and we configure the firewall so that it let this flow access the Internet (flow 2). After communicating with this C&C channel, let us assume that our malware starts sending SYN packets to a range of different IP addresses on port p2. We then assume that this is a propagation attempt, and configure our *dynamic redirection* system to make it redirect at least one of the scanned addresses to one of our honeypots (M3). We then relaunch the malware and observe the propagation simulation success (flow 3).

IV. Experimentation and results

This section is dedicated to the presentation of the preliminary results we obtained from our experiments. Thanks to `nepenthes`, we downloaded 46 malware binaries with different md5 checksums. The analysis of the binaries provided us some interesting results about the C&C channel and the structure of the botnets.

A. C&C channel

All the collected malware that tried to make a connection to a bot used a protocol similar to IRC as the communication channel. We identified 32 different C&C channels. We observed that some malwares with different md5, and even identified as different kind of malwares by

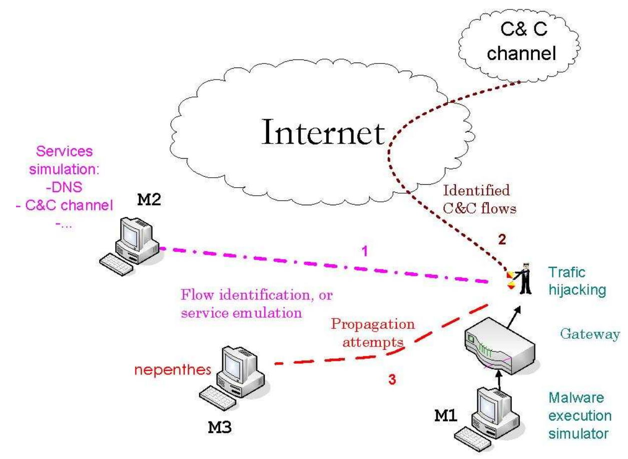
C& C channel
Internet
Services simulation: -DNS - C&C channel -...
M2
1
Identified C&C flows
2
Trafic hijacking
Flow identification, or service emulation
Propagation attempts
Gateway
nepenthes
3
M3
M1
Malware execution simulator

Fig. 3. Global system

figure

an antivirus software, tried to connect to the same channel. We observed C&C channels on TCP ports: 22, 1863, 3000, 3211,
4545, 4747, 5190, 5500, 6667(5 times), 7000(7 times), 8080(twice), 8081, 8585, 8885, 10324, 19555, 21999, 40321(twice), 51555 and 65520.

In [3], authors discovered that the IRC servers could listen for connections on ports greater than 6667. We also noticed that they should also listen on ports lower than 6667 and even less than 1024 (i.e. privileged ports). The conclusion is that IRC servers may be listening on any ports. This result is important in order to calibrate our experiments to detect botnets.

One of the C&C channels obfuscated orders given to the malware. We observed for example this kind of irc log:

```
irc>:hub.59972.com 332 ZtiSDgAy #wish\
irc>:=JiALxH3TzGc+xLRjk65nRq03KD2fcDnt\
irc>LeKyirc>b5x9DTzSyfdxeoa7kH/an9qT8CS\
irc>XkDnXrEbzP2XMJR5l1bb8kvgPvGwquUtUUc6\
irx>cFmzIFDpxT+9GwOPPh6X3KOHVU63dXaiRKKR\
irc>O+w7BvNakyqCaXcIPWvY2+hlHIpDOtJ
```

This order resulted in:

- the download of a keylogger, and
- the scan of the local network, trying to exploit a flaw on the Windows samba service.

Without executing the malware, this kind of order would be difficult to understand.

We also observed, thanks to the simulation of a successful break-in of an attack, that propagation successes are reported on the C&C channel. Such an example follows. After the bot opened a connection to its server and IRC channel, the server returns a propagation order:

```
irc>:$url_irc_server$ 332
irc>BiTch|76914 #bitch :
irc>+advscan MassAsn 50 5 0
irc>201.x.x.x -r
```

The bot then announced on the C&C channel the beginning of the propagation attempt:

```
irc>PRIVMSG #bitch :[SCAN]:
irc>Random Port Scan started on
irc>201.x.x.x:445 with
irc>a delay of 5 seconds for 0
irc>minutes using 50 threads.
```

We simulated the success of this propagation. Then the bot considered that the propagation was a success and this information was sent on the C&C channel:

```
irc>PRIVMSG #bitch :[TFTP]:
irc>File transfer started to IP:
irc>$@simulated_victim$ ($ex_path$).
irc>PRIVMSG #bitch :[TFTP]:
irc> File transfered to IP:
irc>$@simulated_victim$($ex_path$).
```

These experiments showed that the attacker (the human that controls the botnet) can follow the propagation of the malware (he knows when one of his bots succeeds in breaking into a machine, because the attacking bot reports it, and because the attacked machine also announces on the channel when it joins the botnet). This result is important and shows the relevance of our redirection mechanism. We could not have understood these malware propagation features without this mechanism.

This result also puts the emphasis on the fact that using honeypots to infiltrate botnets presents the following limitation: as a honeypot cannot legally break into other machines, the attacker can observe that this bot does not try to propagate itself and can deduce that this bot is a honeypot. Even if the honeypot lies by reporting a false propagation success, the attacker can realize that this is a lie because the designated pseudo victim never tries to join the botnet (these problems are described in details in [4]). However, blackhats need to deal with the fact that IP addresses of corrupted computers can be dynamic, and that such computers are not always powered up and connected to the Internet. Therefore the issue to know whether they use this technique or not is not obvious. The correlation of results obtained by our system with other systems that do not redirect attacks, can be a way to obtain this information.

B. Botnets structure

Our experiments highlighted that the botnets are not static. We observed that one of the malware, when connected to a first C&C channel, received the order to upgrade itself:

```
irc>:STA 332 Bot|2153 #server# :
irc>.dl http://url/malware.exe>
irc> <malware2.exe> 1
```

The first version of this malware had been detected by an up-to-date antivirus. But this antivirus was not able to detect the new version of this malware. We had to wait for a new release of the antivirus for getting an alarm on this new malware form. When running the second version of the malware, we observed that it made a connection to the same IRC server but using a different channel. The IRC server sent the following attack order on the channel:

```
irc>:STA 332 FRA|230440991 #.to.:
irc>.asc dcom135 200 0 0 -r -s
```

We can deduce from these experiments that the deep analysis of the data exchanged on the C&C channel is fundamental to understand the whole structure of a botnet. In order to destroy a botnet, we must not deactivate a C&C channel when it is discovered. It is very important to analyze the data exchanged on this channel, as the `malware2.exe` of the above example. In this particular case, this analysis allowed us to learn the existence of other channels. The best methodology to destroy a botnet is probably to infiltrate it and to be able to get a whole map of its structure.

V. Conclusion and Future Work

This paper proposes an experimental platform with all its components for simulating the behavior of malware. This platform aims at providing a high interaction level for simulated malware but without sending out malicious traffic to the Internet. It permits us to get information on C&C channel structure and on the general structure of botnets.

These encouraging results motivate us to improve this approach towards three different directions:

- First, run more extensive experiments: the risks related to this kind of experiments forced us initially to monitor their execution in real time. With the confidence we gained with this first use of the platform, we are planning to run extensive malware simulations without watching at them continuously.
- Second, we plan to make flow identification automatic. This task is difficult. Existing solutions in IDS are best effort and do not provide any guarantee on *0day*[10] attacks, or on known attacks but sent in ways IDSes have not predicted.

[10]Tool that exploits a vulnerability for which no patch has been published.

- The third direction consists in improving the quality of infiltration of our honeypots; the goal is to make them hardly detectable by hackers, and then make them keep their capabilities of informing us about botnets. We do think that establishing a network of honeypots collaborating with each other would help to improve their transparency to hackers. We are then working on the (for the moment theoretical) design of collaboration mechanisms inside such a honeypots network, for which we evaluate the performances in terms of transparency for the hackers, as well as quality and quantity of produced information about the botnets and commanders.

Some other improvements of our botnet analysis approach will also be investigated in a short future. However, this botnet infiltration work finally aims at developing, at the Internet scale, a global proactive protection system. We expect that infiltrated honeypots would receive some orders for launching attacks. It would then become easier to block the related attacks, or even better, to block the spreading of the attack orders in the botnet. In the short term, this infiltration work will provide us with a list of corrupted machines for which the risk of being involved in a DoS attack is significantly high. This information is interesting as it could help better managing security policies. For instance, for an IDS, profile based attack detection thresholds can legitimately be reinforced for the machines identified as corrupted. For them, the probability that an anomaly on their outgoing traffic corresponds to an attack (instead of a legitimate increase of traffic) is significantly high.

References

[1] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, and S. Yamaguchi. A proposal of metrics for botnet detection based on its cooperative behavior. In *Worskshop SAINT 2007*. IEEE, 2006.

[2] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The nepenthes platform: An efficient approach to collect malware. In *RAID*, pages 165–184, 2006.

[3] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Sruti'2005*, pages 39–44. USENIX, 2005.

[4] C. C.Zou and R. Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN'06)*, Orlando,FL 32816-2362, 2006.

[5] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. Technical Report AIB-2005-07, RWTH Aachen, 2005.

[6] T. Holz. A short visit to the bot zoo. *IEEE Security & Privacy Magazine*, 3:76–79, May-June 2005.

[7] T. Holz and F. Pouget. A pointillist approach for comparing honeypots. In *Proceedings of the 'Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2005)' Conference, Vienna, 7.-8. July 2005*, volume 3548 of *Lecture Notes in Computer Science*, pages 51–68, Frankfurt am Main, Germany, 2005. Spriger.

[8] J. Kristoff. Botnets. In *32nd Meeting of the North American Network Operators Group*, October 2004.

[9] Y. Mashevsky. Les dessous de l'économie souterraine des codes malicieux : chevaux de troie, virus et malware. *VirusList.com*, 2006.

[10] D. More, G. M.voelker, and S. Savage. Inferring internet denial-of-service activity. In *Proceedings of the 10th USENIX Security Symposium, Washington, D.C, USA*. The USENIX Association, 2001.

[11] D. Napier. IPTables/NetFilter – Linux's next-generation stateful packet filter. *j-SYS-ADMIN*, 10(12):8, 10, 12, 14, 16, Dec. 2001.

[12] S. Racine. Analysis of internet relay chat usage by ddos zombies. Master's thesis, April 2004.

[13] Symantec. Symantec internet security threat report, September 2006. `http://www.symantec.com/enterprise/threatreport/index.jsp`.

[14] G. Zhang and M. Parashar. Cooperative mechanism against ddos attacks. In *International Conference on Security Management (SAM .05), Las Vegas, NV, USA*. The Applied Software System Laboratory, Department of Electrical and Computer Engineering, Rutgers University, CSREA Press, 2005.