Architecture of a Platform for Malware Analysis and Confinement

Gilles Berger-Sabbatel, Maciej Korczyński, Andrzej Duda
Grenoble Informatics Laboratory, France
Email: {gberger, korczynski, duda}@imag.fr

***Abstract*—In this paper, we describe an architecture of a platform for studying botnets, finding adequate analysis methods, monitoring the activity of botnets, and finding efficient countermeasures or confinement methods. The platform is composed of a filtering and monitoring gateway, low and high interaction honeypots, and a system for malware analysis. We present all the elements, discuss their roles and functionnalities, and report on already developed tools.**

I. Introduction

A botnet is a network of zombie computers compromised by some malware (virus, worm). Botnets are coordinated by a botmaster through a command and control channel (C&C) to which the malware connects to get instructions. A botmaster can use botnets to perform malicious activities. Most of control protocols are based on the IRC protocol [1], but other protocols such as HTTP may be used and a new trend in controlling botnets is to use peer to peer (P2P) based C&C channels. Malicious activities performed by botnets are numerous: a botnet can scan the Internet to discover vulnerable hosts and eventually disseminate the malware, send spam messages, perform denial of service attacks, steal personal data and many others. They may generate considerable financial revenue through fund extortion, fraud on credit cards, sending spam, or selling counterfeit products (including medicines).

A typical botnet comprises several thousands hosts and there are several thousands known botnets. The traffic generated by botnets represents a significant part of the total Internet traffic. Botnets are currently a major cause of nuisance and threat on the Internet : they can potentially disrupt almost any computer-based service as long as it is connected to the Internet, they are the support of criminal activities, and the source of significant underground economy [2], [3].

Another problem that we want to tackle is detecting and confining distributed denial of service attacks (DDoS) and portscans. Once a virus takes control over a victim host, it can launch DDoS attacks by sending large amounts of packets to target hosts so that they cannot provide service to legitimate users.

Our goal is to study botnets, find adequate analysis methods, monitor the activity of botnets, and find efficient countermeasures or confinement methods. This kind of studies requires setting up a platform for capturing botnets (or malwares) and observing their behaviour, particularly in respect to their interactions with the Internet. While doing that, we have to avoid the risk of participating in malicous activities (propagation of worms, DDoS attacks, spam transmission), or at least minimise the risk and the consequences of such activities.

In this paper, we present the architecture of the platform designed for this study, discuss their roles and functionnalities, and report on already developed tools.

II. Related work

Several approaches are possible to study botnets.

One of them is to passively monitor connection attempts on unused network address ranges to get a statistical view of malware activity (telescopes) : in almost all cases, they are the result of malicious activities even though they can also come from misconfigured host or routers. To get a significant view of malware activity, a telescope should collect connection attempts in a quite large address space, possibly distributed over several sites. Kumar et al used telecopes for the analysis of the propagation of an internet worm [4]. Anyway, a telescope only gives an external view of botnets and does not provide a real knowledge of complex internal botnet behaviour.

Solutions to monitor botnets may be based on the monitoring of DNS lookups [5]. The problem with such solutions in a research context is that they require the cooperation of administrators of DNS at a rather high level. In fact, it would be problematic for most research teams to get such a cooperation and experiment on a highly sensitive component of the internet architecture.

Another approach is to capture malware and study their behaviour. Honeypot are a popular mean used to capture malwares propagated on the internet. A first approach was to setup vulnerable servers to observe the behaviour of intruders [6], but now, the trend is rather to use honeypots to capture internet worms.

The analysis of captured malware may be conducted by the way of reverse engineering, emulation, trace of execution, or automatic code analysis. In [7], Trinius et al propose an abstract instruction set for the representation of the significant activities of a malware. Such representation may be derived from a trace of system calls and allows a higher level of analysis. However, a number of malwares are able to detect the fact that their execution is traced and work around it.

Another way would be to let the malware execute freely in a controled environment. In [8], Alata et al proposed a method based on connection redirection to monitor internet attacks [8].

In [9], Rajab et al present a platform for the study of botnets in a confined environment.

III. Architecture

The platform aims at providing tools able to fight botnets, mitigate their nuisance, eventually disrupt them, and identify botmasters. The tools will provide the following functionalities:

- monitoring communications on C&C channel to detect harmful activities,
- monitoring traffic for detecting and confining DDoS attacks,
- detecting zombie computers on the network,
- filtering botnet-related traffic out of networks,
- confining infected parts of the network and limiting the propagation of the malware,
- finding methods for disabling botnets.

In order to achieve these objectives, first we have to gather a better knowledge of botnets. We choose to setup a honeypot to capture malware, and study their behaviour, but we have to do it with care, because malware can be dangerous and may become a threat to our production networks. We also have to take into account legal et ethical issues [10].

The overall architecture of the platform is presented in Figure 1. It is a class C subnetwork with a gateway providing connection to the Internet. A router connects the production network and enforces filtering rules: all incoming communications from the Internet to this network will be allowed, while outgoing communications could be restricted and monitored according to given needs. Communications with the production network should be strictly limited to only allow services such as HTTP access from the platform to a local server to allow software updates and ssh connections from a few machines to allow remote control of the platform.

The platform contains several machines, possibly virtual ones. The main functions are the following: a low interaction and high interaction honeypots, and an environment for malware analysis.

A. Filtering and monitoring gateway

The gateway is a carefully secured server or a workstation under Linux with two Ethernet interfaces. Its functions are the following:

- allow observation and control of the traffic between the platform and the Internet. In this way the control can be enforced in a more flexible and smart way than what can be done on a standard production router,
- provide a secure environnement to work on the platform,
- protect the Internet and the production network against attacks that may leak from the platform (e.g. DDoS attacks or portscans),
- protect the platform against some types of attacks from the Internet,
- provide services to the platform: ssh relay, file server, HTTP proxy for software updates.

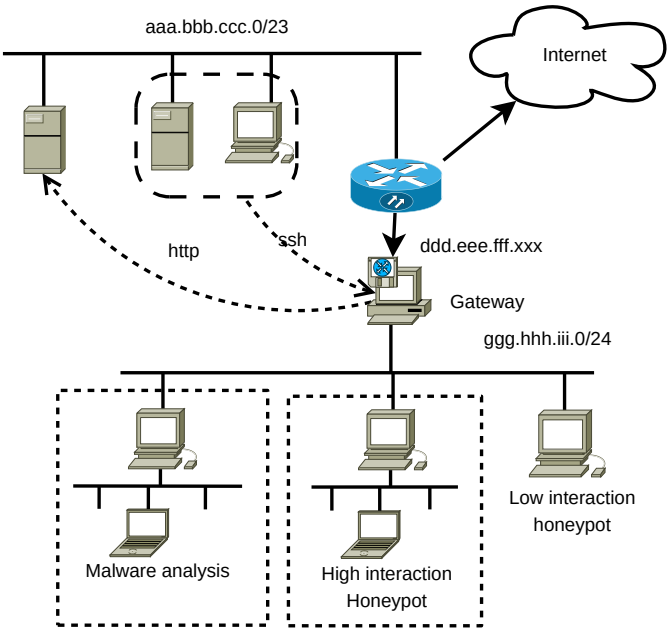

Fig. 1. Architecture of the platform

While the gateway relays traffic between the Internet and the platform, it should not have itself any direct communication withq hosts on the Internet, so that strict filtering rules should be enforced by the router and every outgoing connection attempt should be considered as a sign of a compromised platform.

The gateway can also run a detection method of DDoS attacks and portscans [11]. We have proposed such a method and the proposed platform is a suitable environment for experimenting with the traffic generated by botnets. We discuss more this aspect in Section VI.

B. Low interaction honeypots

Low interaction honeypots are programs that emulate services with known vulnerabilities and let malware succeed in its attack. The result is that malware is downloaded on the honeypot. Such honeypots raise no security problems, as malware is not executed. Malware can be sorted to eliminate duplicates and malware executable is directly recovered. On the other hand, low interaction honeypots only work with malware that exploits known and emulated vulnerabilities. Furthermore, some malware can detect such honeypots and undertake preventive actions.

We have experimented with two low interaction honeypots:

- *Nepenthes* listens on multiple network ports and emulates known vulnerabilities. When an emulated vulnerability is triggered, it emulates the behaviour of the vulnerable system until downloading the malware. With some effort, it should also be possible to add new attack scenarios from the shellcode.
- *Honeyd* can serve a different purpose than nepenthes—it allows to create a large virtual network of hosts (up to 65535) running services including routers. Currently, it

implements very few services and it is up to the user to implement further services and vulnerabilities with significant effort.

Due to its simplicity of installation and use, we have chosen Nepenthes to collect malware. To catch malware attacks, Nepenthes must run on a network not protected by a firewall. As this is not the case of our production network, Nepenthes has been installed at home on a personnal computer running Linux with an unfiltered connexion to the Internet. In a few weeks of an intermittent operation (a few hours per day on the average), Nepenthes allowed us to capture more than 40 malwares and did not cause any security problems. Nepenthes can run on any machine with a standard installation of Linux without particular requirements in terms of performance.

We are curently switching to Dionaea, a new implementation of a low interaction honeypot intended to become a successor of Nepenthes that tries to make easier to add emulation for new vulnerabilities or services. After one week of continuous execution (still at home), dionaea captured a hundredth of malware, and more than 4500 attacks, most of which on ports 445 and 135.

C. High interaction honeypots

High interaction honeypots are systems with real vulnerabilities often run on virtual machines. They are able to more efficiently capture some malware, but they raise security problems, because they may become infected and pose a threat to other computers on the network. When deployed, they should be protected against already known and unwanted malware. Moreover, malware executables are often hidden in the file system and can be hard to recover. A successful infection must be detected and the system should be reinstalled after getting the malware.

A possible solution to the problem of filtering unwanted malware could be to use some kind of a smart gateway able to detect an infection payload, intercept it (catch the executable), and prevent it to be transfered to the honeypot.

D. Environment for malware analysis

Once malware is obtained, we have to study how it works and what are its interactions with the Internet, particularly its interactions with the C&C channel (Command and Control - IRC, or whatever else). This implies letting the malware run on a machine (real or virtual) and interact with the network, while blocking communications that could be harmful to other Internet hosts (infection, spam sending, DoS attacks).

The environment for malware analysis will include tools to control communications between the infected machine and the Internet, to monitor the activity of C&C channels, and to let the infected machine interact with machines on an isolated simulated network.

IV. Malware analysis

A. Basic tools

Once collected, malware is installed on a Windows XP system (the *victim*), which can run on a real or a virtual machine. Virtualization with hardware assistance is now supported by Linux distributions on most recent processors (KVM) and is able to run Windows. The state of virtual machines can be saved in order to be able to quickly restore a machine in a clean state or to save an infected state for further study. In the case of an installation on a physical machine, a Linux distribution is installed on another disk partition, so that we can keep a copy of the clean Windows installation by copying the Windows partition.

The victim is directly connected to a workstation with two interfaces—the analysis workstation. It is connected to the Internet through the other interface (forwarding between the interfaces should not be activated). The `dnsmasq` software is run to provide a DHCP and DNS service to the victim PC.

We can adopt several solutions to intercept traffic generated by the malware:

- Interception through the `pcap` library. This solution may be used to passively analyse any traffic that passes through the monitoring machine, but it does not allow to implement actions such as selectively forward packets, modify and forward packets, return a fake answer to packets.
- DNS may be tweaked to return the adress of one of our machines for some domain names. Then, packets may be processed at the applications level, forwarded to the real destination, or answered. The drawback is that it can only work as long as we can find a one-to-one mapping between destination adresses and destination ports.
- External adresses may also be redirected to the gateway workstation by an `iptable` rule set by a command such as : "`iptables -i eth0 -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080`". This can be useful for intercepting traffic through an application-specific proxy.
- Traffic may be intercepted at the kernel-level: this solution is much more powerful, but harder to implement. We discuss this issue in Section V.

Some basic tools are used for the analysis:

- *Relay*: it is a home-made tool that runs on the analysis workstation to analyse the interaction of malware with some external systems such as its C&C IRC channel without letting it really connect to the Internet. It takes a list of addresses and port numbers in the form address:port as arguments. This means that we must first identify relevant ports and addresses. Then, it listens to TCP connections to given port numbers. When a connection is accepted, it establishes a connection to the same port at the corresponding address, forwards and records every segment received on one side to the other side. Hence, a malware can normaly interact with its C&C channel, while harmful trafic remains confined to the private network. Optionally, packets can be forwarded with a small delay to let the operator some time to see what happens and possibly stop forwarding.
- *Relay-http*: an HTTP proxy such as `squid` is quite

complex to configure, does not allow us to monitor trafic, and it is quite tricky to recover transmitted data. Hence, we have implemented `relay-http`, a very basic HTTP relay that does exactly what we need.
- *Fakemta*: it is used to mimic a mail transfert agent (MTA). It listens to port 10025 by default, answers SMTP requests, accepts emails, stores them in a file, and obviously does not forward them to an actual MTA. All SMTP traffic coming from the victim should be redirected to port 10025 of the analysis workstation with an `iptables` rule.

B. Methodology

The first malwares that we have captured were manually analysed with the following methodology:

- Unless it is already done, we try to identify the malware using various anti-virus tools: we use `clamav` under Linux, but other Windows anti-virus tools may also be used. The malware can also be submitted to Norman Sandbox (http://www.norman.com/microsites/nsic/Submit/en: this can provide useful information, such as Internet accesses done by the malware).
- Connect a clean victim PC (no malware installed) to the analysis workstation.
- Make sure that the analysis workstation is correctly configured.
- Start `wireshark` on the analysis workstation and capture packets on the second interface (the private network).
- Boot the victim PC. Observe the generated traffic. Depending upon the network configuration, there should be a fair amount of traffic during several seconds. Wait until there is no more traffic for a few seconds.
- Run the malware on the victim PC, while observing the traffic captured by `wireshark` and record the events that would likely denote attemps to connect to the C&C channel:
  - DNS requests,
  - attempts to connect to unknown ports,
  - scanning of unknown ports.
- According to the previous observations, setup `dnsmasq` to return the address of the analysis workstation for adresses we found interesting, start *relay* to intercept connection attempts, and restart the victim PC to observe packets exchanges, and further network traffic.
- If the victim attempts to connect to an SMTP server, use `fakemta` to intercept emails. It it tries to connect to HTTP servers, use *relay-http*.
- If the victim tries to connect to unknown ports other than their C&C channel, use *relay* with a delay of a few seconds, observe the traffic with wireshark, and be ready to stop the transmission if it appears harmfull (simply kill relay).

C. Case study

The first malware that we studied was labeled "Trojan.SdBot-4763" by `clamav`. Once started, the malware tries to connect to `botz.noretards.com`, port 65146 that appears to be an IRC server. This address corresponds to a pool of several machines most probably compromised ones. Figure 2 plots the number of IP addresses corresponding to `botz.noretards.com` day by day on a period of 250 days while Figure 3 presents the cumulative distribution function of the number of days each machine remains in the pool. As we can see, the number of machines vary, but is most often around 9. A majority of machines stays in the pool less than 20 days, but there is a long tail of machines that stay more than 40 days. All this seems to indicate that the botnet was actively maintained during this period, as machine recovered by their administrators were quickly replaced.

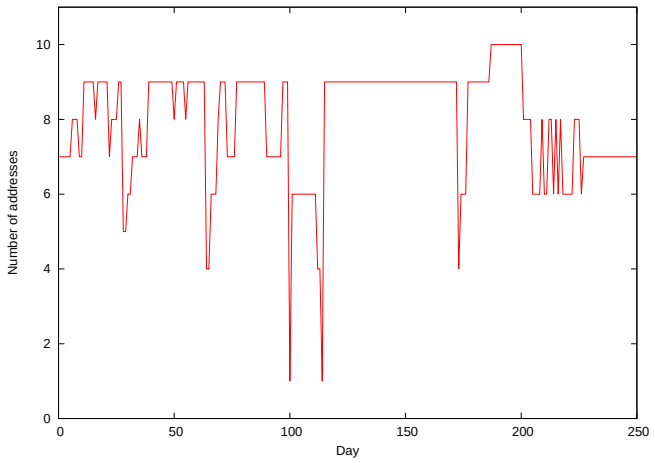

Fig. 2. Number of machines resolving to `botz.noretards.com`

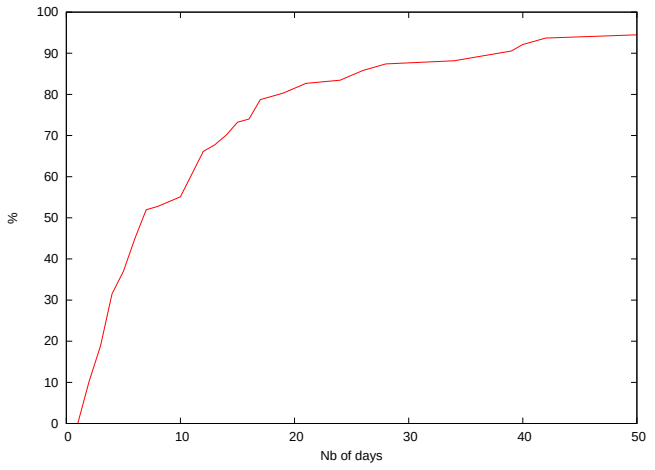

Fig. 3. Cumulative distribution function of the number of days each machine remains in the pool of compromised machines

Once connected to its IRC server, the malware enters random nicknames and user names, and connects to a channel

with a password hard wired in the program. Then, it gets the command ":!root.mass -a -r -s" from the C&C channel and begins to scan ports 135 and 445 with adresses randomly generated in the adress range xxx.0.0.0/8 (where xxx is the first byte of the IP address of the victim).

We did not observe any other malicious activities of this malware. We can make a few hypothesis to explain this apparent lack of valuable activity for such an actively maintained botnet :

- it may by a spyware that does not find any interesting data to send,
- we may have missed interesting activities as we did not observe its behaviour long enough,
- it may expect some events (such as infecting other hosts) before being ordered to do other activities.

Obviously, we cannot afford neither to let the malware unattended, nor to spend a long time observing it, nor letting it infect a third party computers. So, on one hand, we need a means to monitor the activity of the C&C channel without the need of running the actual malware, and on the other hand, we need a means to let the malware infect other hosts—actually another honeypot.

V. MWNA : MALWARE NETWORK ANALYSER

The methodology presented in Section IV-B is cumbersome:

- We have to manually set up multiple tools. In most cases, this implies to reboot the victim as soon as we identify interesting transmission attempts and decide what to do with them.
- We have to redirect connection attempts on a case by case basis using either DNS redefinition (reconfiguring and restarting `dnsmasq`), or `iptables` rules.
- we have to identify interesting traffic among a lot of other regular communications.
- when the malware begins to scan the network, `wireshark` becomes almost unusable unless we filter out scanning activity, which can only be done after observing a significant number of attempts.

Furthermore, it is not possible to observe the malware activity for a long period of time.

Hence, we need a tool to securely support the analysis of the malware network activity.

To achieve this, we have started to develop `mwna` (MalWare Network Analyser). It is based on the Linux packet filter mechanism (cf. Figure 4) that allows to intercept and process packets passing through several hooks in the processing of packets flowing through the kernel.

Intercepted packets can be handled by programs running in the user space (not in the kernel) by the way of `iptable` rules set by the following commands, where `a.b.c.d` is the IP address of the victim:

```
iptables -t mangle -A PREROUTING -s a.b.c.d -j NFQUEUE
iptables -t mangle -A POSTROUTING -d a.b.c.d -j NFQUEUE
```

These commands append two rules to the mangle table. The first is applied at the prerouting hook to packets coming from the victim, and the second is applied to the postrouting hook to packets outgoing to the victim. For both rules, packets are sent to queue 0 of the packet filter, which passes them to `mwna`. After processing, packets can continue their way in the network stack or they can be dropped. They can also be modified, which means that they can be redirected.

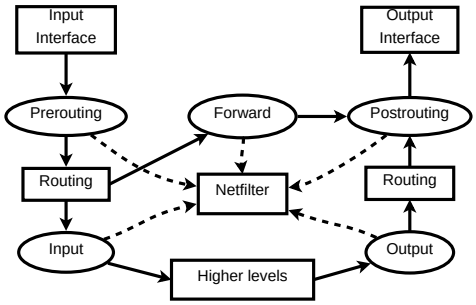

Fig. 4. The packet filter mechanism

The overhead due to the handling of packets through the packet filter is small : if the user program immediately reinject the packets into the kernel (without anay processing), more than 900 Mbit/s can be processed while the normal rate is 934Mbit/s in TCP.

`mwna` gets raw ethernet packets. Hence, a basic analysis of the protocol headers must be done to identify most common protocols.

DNS packets are analysed to maintain a list of known hosts, i.e., hosts whose addresses have been resolved: most regular transmissions would be addressed to a known host, while packets sent to unknown hosts are likely to be parts of a scan.

A list of TCP connections is also built in order to be able to take decisions according to the context.

The program can accept rules to determine what to do with specific packets. Rules can be attached to a specific protocol. A rule contains conditions that must match packets and actions. The rules are put in a configuration file parsed with the `config` library.

Any combination of conditions can be specified. The currently defined and mostly implemented conditions allow to check:

- whether the destination or source IP address has been resolved by DNS (known, unknown).
- the direction of the transmission (in, out).
- whether the packet is internal (to/from the local network) or external (to or from the internet) (int, ext).
- whether the packet is sent to a multicast, a broadcast or a unicast address (multicast, broadcast, unicast).
- the source or destination port (sport/dport),
- the source or destination IP address (ipfrom, ipto).
- time to live of the packet (ttl).

Actions can be the following:

- **set verdict**: specify what to do with the packet (accept, drop, etc.).
- **trace**: display a trace of the packet on the terminal.

- **ctrace**: display a trace of connexion changes on the terminal.
- **interval**: only accept packets if the given time interval from the previous packet is elapsed.
- **redirect**: redirect the packet to a specified address/port.
- **ask**: ask the user what to do.

VI. Detection of DDoS attacks and portscans

The most difficult problem in the detection of DDoS attacks is to distinguish between the malicious behavior and the regular traffic in an early stage of an attack for efficient prevention. The difficulty of solving the problem is even exacerbated by the ever increasing capacity of current networks and important traffic fluctuations. Consequently, we cannot always consider traffic anomalies as a result of potentially malicious activities. Otherwise, such an assumption may lead to an increased *false positive* rate, i.e. we may classify legitimate packets as malicious ones. Moreover, existing anomaly detection methods may fail when applied to the backbone, because traffic characteristics are not necessarily visible at this level of traffic aggregation. For instance, the distribution of source IP addresses may stay approximately the same even during high-volume attacks. This is an important reason for an increased *false negative* rate, i.e. we may classify malicious packets as legitimate.

Complexity of algorithms is another problem faced when designing efficient detection methods. The cost of some methods such as those based on entropy is extremely high, because they need to analyze every packet to be efficient. Sampling methods like Cisco NetFlow [12] only inspect some packets, which if combined with simple and robust detection techniques may significantly reduce the amount of monitored traffic data. The last aspect relates to the detection time—the time to react after an attack detection needs to be sufficiently short to avoid severe damages or rapid proliferation in case of a worm epidemic. This calls for an automated defense method.

We have proposed a detection method of high-volume malicious traffic composed of DDoS attacks and portscans [11]. Our method first analyzes packets during normal network operation to establish baseline parameters and derive thresholds that may reflect different network environments. We extract a number of essential attack features such as concentrated destination and source IP addresses. We examine TCP sessions from a new perspective so that we can easily detect traffic asymmetry in case of DDoS attacks or portscans. Finally, we combine the method with a rate limiting scheme that controls traffic rates. Our method is suitable for sampling so that we can reduce the amount of monitored traffic and still achieve good detection performance.

Previous work has shown that anomalies, which only impact packet counts, are likely to be visible at lower sampling rates, while anomalies that influence flow counts go undetected. We have found that it is important to take into account appropriate metrics and integrate them with an efficient detection method as well as an adequate sampling technique. We have validated our method on traces captured during real network attacks unlike other proposals that used network simulations or experiments on obsolete data sets with outdated background traffic. The prototype implementation effectively detects 99% of DDoS attacks including TCP SYN, ICMP flooding, and scanning activity. In comparison with the Uniform Rate Limiting method [13], our results are much superior with no false positives. We have also shown that three different sampling schemes (deterministic, random 1-out-of-N, and random uniform) result in very good performance despite extremely low rates like 0.0625%, which means analyzing 1 out of 1600 packets on the average.

The method will be deployed on the gateway of our platform so that we will be able to experiment with real botnet traffic and observe the detection performance.

VII. Conclusions

In this paper, we have described an architecture of a platform for studying botnets, finding adequate analysis methods, monitoring the activity of botnets, and finding efficient countermeasures or confinement methods. The platform is composed of a filtering and monitoring gateway, low and high interaction honeypots, and a system for malware analysis. We have presented all the elements, discussed their roles and functionnalities, and reported on already developed tools.

Acknowledgments

This work was partially supported by the EC FP7 project INDECT under contract 218086.

References

[1] J. Zhuge, T. Holz, X. Han, J. Guo, and W. Zou, "Characterizing the irc-based botnet phenomenon," Department for Mathematics and Computer Science, University of Mannheim ; TR-2007-010, Tech. Rep., 2007.

[2] A. Hackworth and N. Ianelli, "Botnets as a vehicle for online crime," *The International Journal of Forensic Computer Science*, vol. 2, no. 1, pp. 19–39, 2007.

[3] J. Franklin, V. Paxson, S. Savage, and A. Perrig, "An inquiry into the nature and causes of the wealth of internet miscreants," in *Conference on Computer and Communications Security*. ACM, 2007, pp. 375 – 388.

[4] A. Kumar, V. Paxson, and N. Weaver, "Exploiting underlying structure for detailed reconstruction of an internet-scale event," in *IN PROC. ACM IMC*, 2005.

[5] D. D. Anirudh Ramachandran, Nick Feamster, "Revealing botnet membership using dnsbl counter-intelligence," in *SRUTI '06: 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, U. Association, Ed., 2006, pp. 49–54.

[6] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, and M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot," in *6th European Dependable Computing Conference (EDCC-6)*, Coimbra (Portugal), 2006, pp. 39–44.

[7] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," University of Mannheim, Tech. Rep. 2009-007, 12 2009.

[8] E. Alata, I. Alberdi, V. Nicomette, P. Owezarski, and M. Kaaniche, "Internet attacks monitoring with dynamic connection redirection mechanisms," *Journal on Internet Computer Virology*, vol. 7, no. 2, 2008.

[9] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 41–52.

[10] A. Burstein, "Conducting cybersecurity research legally and ethically," in *first USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET '08)*. San Francisco: Usenix association, April 2008. [Online]. Available: http://www.truststc.org/pubs/322.html

[11] M. Korczyński, L. Janowski, and A. Duda, "An Accurate Sampling Scheme for Detecting SYN Flooding Attacks and Portscans," in *IEEE ICC 2011*, June 2011.
[12] "Cisco IOS NetFlow," http://www.cisco.com/go/netflow/.
[13] K. Park, D. Seo, J. Yooand, H. Lee, and H. Kim, "Unified Rate Limiting in Broadband Access Networks for Defeating Internet Worms and DDoS Attacks," in *Information Security Practice and Experience*, vol. 4991. Springer Berlin / Heidelberg, October 2008, pp. 176–187.