Computer Networks 129 (2017) 261–272

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

MARS: From traffic containment to network reconfiguration in malware-analysis systems

João Marcelo Ceron[a,*], Cíntia Borges Margi[a], Lisandro Zambenedetti Granville[b]

[a] University of São Paulo, SP, Brazil
[b] Federal University of Rio Grande do Sul, RS, Brazil

ARTICLE INFO

Article history:
Received 29 December 2016
Revised 9 October 2017
Accepted 13 October 2017
Available online 16 October 2017

Keywords:
Network reconfiguration
Malware analysis
Malware Containment

ABSTRACT

Malware analysis systems are essential to characterize malware behavior and to improve defense mechanisms. In dynamic malware analysis, the actions performed by malware in a sandbox are highly dependent on the interactions with other hosts and services. However, the current solutions superficially deal with the network environment that surrounds the sandbox, exposing limitations to traffic containment and network resources reconfiguration. We have already shown how Software-Defined Networking (SDN) could enable network access policies changes and thus exposing distinct malware actions. In this paper, we investigate the malware analysis process by considering the entire analysis environment, including a sandbox and other components that comprise it. We developed a fully-automated malware analysis solution that uses network layer as a tool to reconfigure the analysis environment. In that way, it is possible to implement per-flow containment rules, dynamic resources configuration, and to manipulate network traffic to impersonate services. Our experiments show that it is feasible to identify behavioral deviations in different analysis scenarios and reveal many more malware behaviors than those revealed by the state-of-the-art analysis systems.

1. Introduction

Malicious software, often referred as malware, pose a major threat to computer systems. Malware can damage computer systems and perform a myriad of harmful actions in the affected systems. To protect systems from those threats, it is important to develop tools that can characterize malware behaviors and develop defense mechanisms. Antivirus and Intrusion Detection Systems, for example, need to understand the malware behaviors to develop signatures and then defend systems from such software. In this context, malware analysis tools provide a valuable function by enabling to investigate the malware actions.

The substantial volume of malware daily deployed encourages the use of automated malware analysis techniques such as the dynamic malware analysis [1]. Dynamic analysis typically consists of executing a malware sample in a *sandbox, i.e.* instrumented system, and check the malware performed actions. Based on the observed behavior, an analyst can identify the threat severity and develop appropriate countermeasures.

Malware analysis solutions must continually improve their mechanisms to identify actions performed by malware and to provide details about its behavioral profile. A class of malware can inhibit its malicious actions when it detects an instrumented analysis system [2,3]. In that way, by remaining undetectable to analysis systems, malware can execute in systems and perform harmful actions without being identified by security tools.

Analysis results provided by a sandbox are also dependent on the environment surrounding it, including the interaction with other hosts and services. This is especially true in modern malware. A class of malware not only inspects the sandbox itself but also the environment surrounding it. Malware, such as Agobot, SDbot, Kolab, and Mirai, can modify their actions when an analysis system is detected [1,4]. Other malware suspend their execution when they do not have Internet connectivity or availability of certain network services [5]. The advanced persistent threat *Red October*, for example, fully deploys its capabilities exclusively in embassy and government environments [6].

The literature presents many solutions related to dynamic malware analysis. Most of these solutions focus on improving the system in which the malware is executed, neglecting the network components that surrounds the sandbox [7]. Furthermore, current malware analysis solutions pose limitations when faced with modern malware that relies on network resources. In this context,

* Corresponding author.
E-mail addresses: ceron@usp.br (J.M. Ceron), cintia@usp.br (C.B. Margi), granville@inf.ufrgs.br (L.Z. Granville).

https://doi.org/10.1016/j.comnet.2017.10.003
1389-1286/

taking the network layer as part of the malware analysis process can address current solution limitations by improving the malware analysis process. In the network layer, it is possible to observe all the connections established by the malware targeting the Internet and determine services directed to the analysis environment. Acting upon those connections enables the implementation of fine-grained traffic containment rules to avoid malicious activities to abuse other systems. Additionally, the network layer can also be used to modify the analysis scenario by incorporating new elements (*e.g.*, network services) that may reveal unseen malicious behavior from malware that rely on network characteristics to trigger actions.

We here address the issues related to investigate network sensitive modern malware using MARS – MAlwaRe Analysis Architecture based on SDN. As described in our previous work [8], Software-Defined Networking (SDN) [9] is a feasible approach to integrating the network layer as part of malware analysis process. In our previous study [8], we show how it is possible to use different network access profiles to reveal new malware behaviors. In this paper, we focus on the network layer as a tool to reconfigure the malware analysis environment. More precisely, we introduce new capabilities to the malware analysis process by enabling the possibility to reconfigure the analysis environment dynamically based on network connections established by malware. Hence, our solution can inspect malware connections and, based on traffic characterization, perform actions such as forwarding network flows, rewriting DNS queries packets, changing network topology, changing hosts IP addresses, or introducing vulnerable services to the environment to trigger new malicious behaviors.

We implemented a prototype as a proof-of-concept and demonstrate, by experimentation, that network manipulation and malware analysis environment reconfiguration can activate malicious actions that would not happen in other in other malware analysis solutions. As such, the main contributions of this paper are:

- A malware analysis solution that integrates the network layer as part of the malware analysis process by managing the malware connections and by controlling the analysis environment;
- The analysis environment can modify itself during the malware analysis based on network traffic patterns performed by the analyzed malware;
- A system where it is possible to execute a malware in the same environment infrastructure using different environment configurations. This allows identifying, in an automated way, malware behaviors caused by modifications in the environment configuration setup.

The rest of this paper is structured as follows. Section 2 reviews related work, while Section 3 describes our malware analysis system architecture. Section 4 presents the system evaluation and Section 5 depicts the results. We conclude the paper in Section 6, where we present final considerations and future research directions.

2. Related work

Several approaches have been used for understanding the *modus operandi* of suspicious binary files [1]. Especially, the dynamic analysis approach has proven to be an effective way to investigate malware, widely used by the security community to determine malware intents. The most popular approach to implementing dynamic analysis consists in executing a malware sample in a sandbox to monitor the actions performed by that sample [5,10,11]. Sandboxes employ several technologies to monitor malware actions and, as a result, to produce reports of activities. Such reports are essential to identify malware behaviors and enhance the detection mechanisms from the security tools [2,12].

In dynamic malware analysis context, we observe many solutions that focus on improving the quality of hosts-based information collected in the sandbox, besides avoiding sandbox evasion techniques [2,3,12–15]. A set of malware incorporates anti-sandboxing techniques to prevent the malicious code from exposing its activities in the instrumented system. These techniques aim to recognize system aspects, such as hardware particularities, machine configuration files, and execution flow to characterize an analysis environment. MARS architecture defines the use of a sandbox, but anti-sandboxing techniques are not part of our research scope. However, our solution handles sandbox external aspects, such as the connections performed by the malware and other elements surrounding the sandbox.

Malware actions performed in the sandbox are dependent on the interactions with other computers. To observe interesting behaviors, the sandbox must provide some Internet connectivity. On the other hand, it is also necessary to prevent the sample from attacking external hosts. For example, when a worm is analyzed, its malicious actions could reach multiple systems over the Internet and, without the proper containment policy, harm such other systems.

Network containment applied to malware analysis has been addressed using different approaches [16–18]. In several solutions, network support is implemented using middleboxes, in which containment rules and policies are enforced using external network appliances. Graziano et al. [17], however, propose implementing containment rules by emulating network resources required by malware samples. Instead of allowing the malware to access the Internet, the technique proposed redirects all connections to emulated services that handle deterministic protocols. A different approach is presented in GQ [18], which aims to build per-flow containment policies to enable the samples analyzed to access the Internet in a safe and controlled way. Each network flow is distinguished and properly handled to prevent external attacks. To achieve the proposed containment, GQ architecture relies on external control infrastructure and traditional protocol extensions.

In addition to implementing network containment rules, malware analysis solutions must be flexible to provide a network environment that represents a full access network environment. Other malware suddenly changes its behavior and stops its execution when some restrictions are posed to the environment. More recently, modern malware also observes the network elements surrounding the sandbox to perform malicious activities. Particularly, targeted malware requires specific network conditions to perform their malicious actions. The literature describes malware that checks for specific network services and observes the address space where the sample is running [6].

The network layer has become more relevant in malware analysis processes, once its configuration might expose and trigger new malicious behaviors. However, traditional malware solutions do not properly handle the network layer. Current solutions, as described previously, partially with the network support by implementing containment rules and superficially manipulating the network elements comprising the analysis environment. Differently from other approaches, we propose using Software-Defined Networking (SDN) to implement network support by addressing the network reconfiguration and containment policies. More than using a novel approach to control the malware analysis scenario, we show that new functionalities can be addressed when an integrated solution is introduced. A centralized network control enables to configure the analysis scenario by modifying the network topology, IP address space, available services, and to enforce network containment policies. Thus, malware can be analyzed in distinct network access

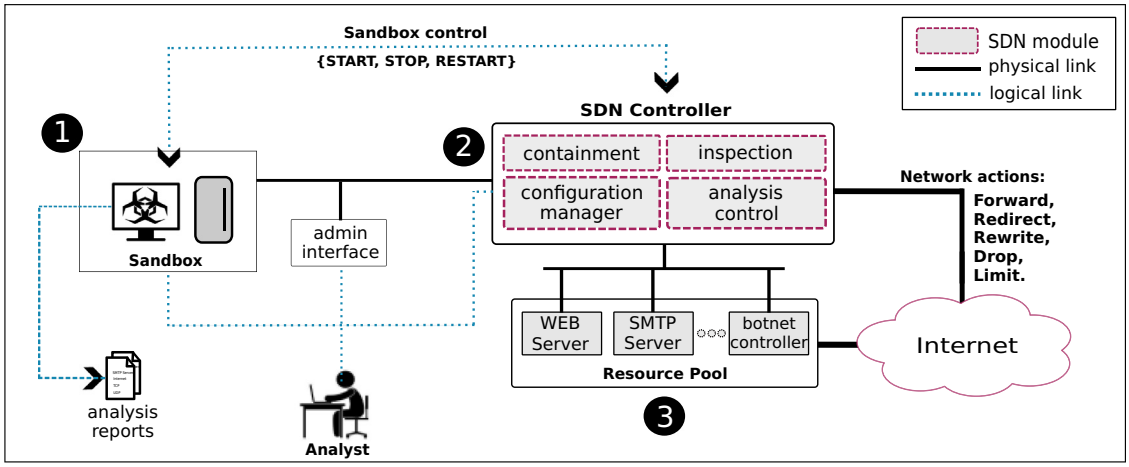

Fig. 1. MARS Architecture: The sandbox (❶) produces reports that describe the actions performed by the malware. The SDN controller (❷) is responsible for managing the malware analysis process and to reconfigure the analysis environment. This analysis environment can assume different network access policies and take advantage of a set of resources – *Resource Pool* ❸ – to compose distinct network scenarios.

policies, which would trigger new malicious actions and to determine the behavioral deviation.

3. Architecture overview

An early version of MARS, our architecture for dynamic malware analysis, was described in our previous paper [8]. However, significant improvements have been integrated to the architecture. Therefore, in this section, we revisit MARS components and its associated execution flow.

MARS architecture integrates the network layer as part of the malware analysis process by managing the malware connections and by controlling the analysis environment. To this end, we designed a modular event-driven architecture where the event defines actions which control the malware analysis process. Those events are flexible and could be customized to fit the requirements described by the security analyst. MARS architecture and its components responsible for handling the events generated are presented in Fig. 1.

Our architecture is essentially designed using three components:

- *Sandbox* - It is responsible for running the binary samples and identifying the actions performed in the system analyzed. Our architecture does not specify any sandbox technology, such as virtual machine or plain physical systems. However, the sandbox must provide mechanisms to support and control the malware analysis process remotely;
- *SDN controller* - It is in charge of managing the malware analysis process and controlling the malware analysis environment;
- *Resource pool* - It is a set of devices and network services that can be used to compose/modify the analysis environment. These elements are controlled by the SDN controller and can be dynamically included in the environment to observe the network traffic characteristics targeting those services.

In addition to the described components, the architecture uses software modules implemented on top of the SDN controller. Those modules effectively interact with the malware analysis process by handling the system configuration. Next, we describe those modules functionalities and their interaction with the analysis process.

3.1. Inspection

The *Inspection Module* is designed to inspect the network traffic looking for predefined patterns and to generate *events* handled by other SDN modules. Those events are generated in consequence of network traffic characteristics, including the number of TCP/UDP sessions, the number of packets, throughput, packet payload, TCP/UDP ports and source/destination addresses. Because of SDN design, the controller can intercept all network connections and examine full stack packet details.

```
1  # traffic redirection and topology reconfiguration
2  if (packet.dst_port == 80 and packet.proto == TCP):
3     server = add_webserver()
4     redirect_traffic(server)

6  # rewrite NXDOMAIN answers
7  if (packet.dns.response.error_code == NXDOMAIN):
8     packet.dns.response.answer.set.new.ip(192.0.2.10)

10 # containment rule
11 if (packet.ip.dst == "www.example.org"):
12    drop(packet)

14 # TCP session limit
15 if (tcp.sessions > 10) and (packet.dst_port == 22)
16    limit(tcp.sessions,dst_port=22)
```

Listing 1. Ruleset implemented by the *Inspection Module* used to detect network signatures and to generate events to other modules.

Listing 1 presents four rules used to characterize the network traffic and to generate events, *i.e.* define configuration actions. The first rule – lines 2, 3, 4 – analyzes HTTP traffic (80/TCP) and redirects it to a local Web server. This redirection allows collecting details about network connections directed to HTTP services. The second rule – lines 7 and 8 – shows an example of data packet manipulation, more precisely, actions associated with the DNS answer rewriting process. The third rule – lines 11 and 12 – exemplifies a rule that blocks connections aimed to a specific target. In the example, connections targeting "www.example.org " are discarded by the controller. The last rule – lines 15 and 16 – shows how to limit the number of simultaneous TCP sessions related to service SSH. This rule, for example, can be used to avoid the analyzed sample to perform brute force attacks.

Our solution associates each triggered event with an action translated into OpenFlow [19] rules and integrated to the SDN con-

```
cd /tmp | cd /var/run;wget http://XXX.XX.1.38/gtop.sh;\
sh gtop.sh;rm -rf gtop.sh;tftp -r tftp1.sh -g ip;\
sh tftp1.sh; tftp ip -c get tftp2.sh; sh tftp2.sh;\
rm -rf tftp.sh tftp2.sh gtop.sh
```

Listing 2. Commands performed by an analyzed malware trying to abuse external systems.

troller. Since network characteristics are matched to predefined patterns, an event is triggered to the appropriate SDN module. That appropriate module handles the events based on their type: *Containment Module* for network containment policies, *Configuration Manager Module* for topology modification and configuration, and *Analysis Control Module* for handling the sandbox running control. Next, we describe these modules that handle events generated by the *Inspection Module*.

3.2. Containment

The *Containment Module* handle events associated with network access policies. Those events describe functionalities related to data packets and network connections, including actions, such as drop, limit, rewrite, redirect, and forward. Naturally, the analyzed malware may interact with another system across the Internet; as a consequence, it is appropriate to implement containment mechanisms to avoid the running malware from abusing third party systems. Despite the risk, most of malware require some degree of communication with external resources to download additional components or to obtain attack instructions. Furthermore, there is malware with aggressive behaviors that try to compromise systems as soon as possible. For example, malware SHA1 hash `d483984a44d0a7d3a9216065d3273a8bdef18f1f` infects IoT devices using default TELNET service password and execute the following commands (Listing 2).

The IP address was sanitized; however, it is possible to observe the command performed by the analyzed malware to an external device. These commands, if successful, actually infect a remote system. Therefore, it is important to implement flexible containment rules that provide instruments to restrict malicious actions. To avoid this particular attack, for example, an analyst could write rules to discard packets with specific payload, such the `wget` command.

A significant advantage when using SDN is the centralized network control. With that control, it is possible to implement containment policies in the core of the network encompassing different aspects, including:

- *Packet payload* - Containment rules can identify strings and particular signatures included in the packet payload. Several rules related to deep packet inspection can affect the controller performance, but this is not a major issue; since the environment is exclusively used for malware analysis, a possible performance impact can be easily detectable, thus not affecting the analysis effectiveness;
- *Network throughput* - To identify the maximum data throughput is a way to evaluate the capabilities of the affected system. Systems connected to high-speed Internet connections are more suitable for particular types of attacks, such as denial of service attacks. Therefore, the configuration flexibility enables the analysis environment to assume distinct throughput limits to map those behaviors;
- *Packet rate* - Similarly to network throughput, it is possible to define a packet rate for the whole malware analysis environment, as well as to configure a packet rate for every element present in the architecture. This configuration is used to avoid attacks using a high rate of small size packets to turn a network service unavailable;
- *Connection limit* - Other attacks have particularities that can easily be blocked by limiting the number of connections performed by the analyzed sample. In attacks associated with vulnerability scanning, for example, to limit the number of connections performed by the malware is an effective way to inhibit those types of attacks;
- *Blacklist* - To block specific destinations is another useful capability in malware analysis systems. Hence, the system can discard packets to predefined destinations and turn targets inaccessible from the analysis environment.

```
1  # containment profile
2  containment_name="001"
3
4  # global limits
5  set limit packet_rate={8000}
6  set limit throughput={1M}
7
8  drop proto tcp from any to 200.160.0.0/20 port = ssh
9  drop proto tcp from any to INTERNET port = smtp
10 drop proto udp from SANDBOX to any limit session 100
```

Listing 3. Containment profile configuration sample in which the environment access policy is defined.

Multiple containment rules can be configured for the analysis system. More specifically, it is possible to configure global access policies affecting all the analysis environment, or applied to a particular set of elements. Listing 3 shows a configuration profile example illustrating containment rules.

The first part – lines 5 and 6 – establishes global rules associated with packet per second rate limit and throughput defined in that environment. Thus, during the analysis elapsed time these parameters must not exceed the limit. Next, lines 8, 9, 10 show rules associated with specific services and elements. This configuration profile defines the initial environment containment policy, although these rules can be modified during the malware execution due to other predefined dynamic rules considering the malware performed actions.

3.3. Configuration manager

This module handles events associated with the network environment configuration. Those events aim to setup the network topology and to modify its configuration as a consequence of predefined network traffic characteristics. The setup configuration is customizable; the SDN controller can control the system by adding new services to the analysis environment, by reconfiguring the running network topology, and by defining different addressing policies.

The analysis environment configuration is specified using a XML-based file in which all the resources available in the analysis environment are defined. Thus, it is possible to create predefined configuration scenarios – template profiles – that could be used in multiple investigations. We designed distinct environment configuration profiles, including: ‘IPv6 only’, ‘academic network’, ‘industrial network’, ‘simple network with few WEB/SMTP servers’, ‘governmental institution’. Yet, an analyst still can create his own templates. Again, the configuration environment is flexible to modify its characteristics during malware execution. Listing 4 shows an example of an environment configuration template.

That configuration profile describes all the resources available, including 3 Web servers, 1 SMTP server, and the sandbox. Despite

```
# configuration profile name
configuration_name = "001"

# initial analysis environment
sandbox = {ip=192.0.2.10}
smtp-sever = {ip=192.0.2.4,status=waiting}
web-server = {ip=192.0.2.1,ip=192.0.2.2,ip=192.0.2.3}
```

Listing 4. Initial setup configuration where the available resource aspects are described.

being an available resource, the SMTP server is not instantiated in the initial setup. The SMTP server is in "waiting mode" and may be inserted into the environment according to the actions performed by the malware.

3.4. Analysis control

In our solution, the analysis process may comprise multiples cycles in which the malware samples are executed using distinct environment configurations. These cycles are managed by the *Analysis Control Module* that handles events requiring the interaction with the sandbox is required.

For that reason, MARS defines a set of predefined actions targeting the sandbox control, which includes: to conclude the analysis, to restart the malware execution, or to revert the sandbox to the initial state for next analysis (see Fig. 2).

Those actions are essentially convenient in cases in which the analysis environment is reconfigured. The analyst can restart the malware analysis process to investigate behaviors triggered as a consequence of environment modification. If a novel activity is presented, it is feasible to repeat that process until achieving the stop condition (a specific behavior found, network threshold, number of rounds).

The communication between the controller and the sandbox is implemented using network protocol messages. We specified a communication protocol – depicted in the Fig. 3 – to manage the sandbox analysis process. The protocol message format is described as follows:

- *Sandbox ID* - It is an alphanumeric code used to identify the running sandbox. This is useful to discriminate the sandbox in situations in which more than one sandbox is in execution in logically separated scenarios;
- *Instruction Code* - This code defines an action that should be executed by the controller in the analysis environment. These actions include environment configuration operations and notification messages. The defined codes and respective actions are described in Table 1;

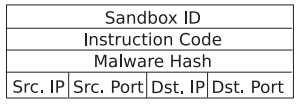

Fig. 3. Custom protocol designed to implement the communication between the sandbox and the SDN controller.

Table 1
Instruction codes used in control messages.

Code	Description	Target
000	Sandbox has started the analysis	controller
001	Sandbox has finished the analysis	controller
010	Sandbox must start	sandbox
011	Sandbox must stop	sandbox
100	Sandbox must restart	sandbox

- *Malware Hash* - It is a distinct value used to identify the malware being analyzed. Hence, the controller can identify when an analysis starts/ends and its respective network accesses;
- *Src. IP* and *Dst. IP* - It defines the instruction packet source IP address and packet destination IP address respectively;
- *Src. Port* and *Dst. Port* - It defines the instruction packet source port and destination port.

The available messages related to the *Instruction Code* field are presented in Table 1.

Several messages can be used to modify the analysis execution process. These instructions – Table 1 – represent a set of commands associated to the controller and sandbox communication. For instance, the instruction code 000 represents a notification message used to notify the SDN controller that the analysis has started. The controller thus knows that a malware sample is executing and can handle the specific packets. Conversely, instruction code 100 tells the sandbox to stop the malware execution and start it again to proceed a new analysis.

The instruction messages described in this section are essential to control the malware analysis process. The analysis process can therefore be instrumented to run a sample in multiple configuration environments and to detect behavioral deviations.

3.5. Automation

The ability to analyze a malware sample using automated tools is an essential feature provided by the dynamic malware analysis approach. However, in traditional solutions, the analysis automation is not extensible to the network environment surrounding the sandbox. In our solution, we extend this automation to the network layer by using configuration profiles. Hence, a sample can be analyzed in different environments without demanding human intervention to configure the network layer elements.

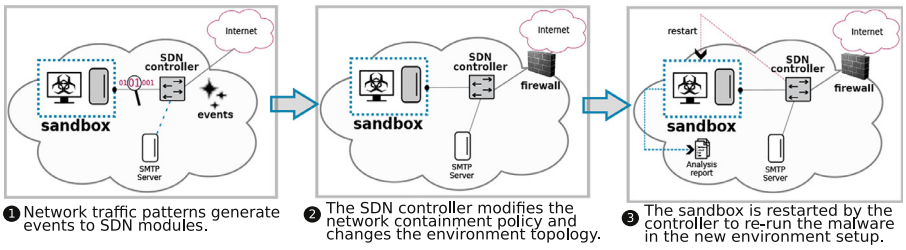

Fig. 2. Malware analysis process - The malware executed in the sandbox (❶) performs network actions that can reach the Internet or other systems present in the analysis environment. All the network actions are inspected by the SDN controller that, based on predefined patterns, generates events to the implemented SDN modules. Those events are handled by the modules and can modify the analysis environment by redirecting specific network flows to internal devices (❷). The controller can enforce the environment access policy by managing the network flows (packet-rate, throughput, blacklist), as well as to interact with the sandbox to stop the malware analysis or to reinitiate the analysis using a new setup configuration (❸).

```
sample=2aa58c26deac88c63b544318da7ab08e2a5fff97,cycle=3,sequence=cycle1,cycle2,cycle3

containment-profile="001"
environment-configuration-profile="002"
inspection-profile="001"
platform = windows
label = cycle1
runtime=5

containment-profile="002"
environment-configuration-profile="002"
inspection-profile="001"
platform = windows
label = cycle2
runtime=5

containment-profile="003"
environment-configuration-profile="001"
inspection-profile="001"
platform = windows
label = cycle3
runtime=5
```

Fig. 4. Configuration file that describes the malware analysis process. This particular process is composed of three cycles using distinct setup configurations.

In MARS, the whole malware analysis process can be configured by using an XML description file. The later allow defining the environment characteristics and other attributes associated with the malware execution. Fig. 4 illustrates a malware analysis process composed of multiple sandbox cycles. The description file indicates the malware analysis process associated with the sample SHA1 hash *2aa58c26deac88c63b544318da7ab08e2a5fff97*. In particular, the example describes a process composed of three cycles and their respective setup configuration.

We discuss the configuration attributes used to specify that analysis process as follows.

- *containment-profile* - It is a template file that describes the access policy profile to the running analysis environment;
- *environment-configuration-profile* - This directive refers to a configuration file in which all the available resources are described, including their configuration and attributes;
- *inspection-profile* - It refers to a file that describes the network events that must be observed by the controller to issue events to reconfigure the analysis environment;
- *runtime* - The number of minutes malware should be executed in the sandbox;
- *platform* - It describes the platform in which the malware sample is executed (sandbox platform);
- *label* - It is a configuration file identification label.

This particular analysis process (Fig. 4) produces three report files describing the malware behavior in each environment configuration. These reports provide the resources to detect the malicious behavioral deviation in the different analysis environments and, based on that, it is possible to tune the analysis process by modifying the setup configuration.

4. Methodology

This section presents our methodology to evaluate MARS. The goal is to determine the improvements provided by our solution to trigger new malware actions in the dynamic analysis process. To this end, a feasible approach is to execute a malware sample in different malware analysis solutions and identify the actions performed. Consequently, we can observe the MARS efficacy regarding revealing unseen malware actions.

To the best of our knowledge, there is not a well-established methodology and metrics to evaluate in the context of malware actions in malware analysis solutions. Several studies evaluate a subset of characteristics to determine the malware solution effectiveness. In Pytrigger [3], authors evaluate the malware activity by observing the file system access and system configuration key modifications performed by the malware in the sandbox. Lindorfer et.al [16] describe a technique based on system calls for detecting malware samples exhibiting different behavior across different sandboxes. Those evaluation techniques, focusing on system calls, present limitations to observe the malware behaviors. For example, system calls related to inter-process communications and asynchronous calls are overlooked by those techniques. On the other hand, there are methodologies that consider the connections established by malware to detect behavioral deviation. For example, in GQ [18], the behavioral malware difference is determined by the number of connections performed by the malware.

In modern malware, different actions are triggered based on the execution context, making it necessary to observe multiple execution parameters to identify behavioral variance. Therefore, using a methodology based on individualized parameters could introduce imprecision to the evaluation. For example, a set of malware tries to connect to dynamic IP addresses – P2P administration controller, *fast-flux* domains, and CDNs. As a consequence, each analysis cycle generates distinct network events that do not represent a behavioral deviation.

In this work, we argue that an evaluation metric must examine multiple parameters to more precisely map the actions performed by the malware. In the absence of a common metric to compare malware analysis solutions, we discuss a methodology that examines a subset of parameters provided by sandbox reports. The parameters are:

- *IP Conversations* - This parameter define a network traffic conversation between two peers.;
- *IP Addresses* - Provide all IP addresses observed in the malware analysis environment, including the addresses requested by the analyzed malware sample;
- *Behavioral Sandbox Signature* - These signatures are provided by the sandbox and describe actions based on file system and registry indication. Signatures typically represent generic actions, such as “HTTP traffic detected”, “Malware tries to connect to an IRC Server”, “Malware tries to access Tor Network”, and other signatures that correspond to a predefined behavioral pattern;
- *TCP/UDP Ports* - Provide information related to all TCP/UDP ports requested by the malware sample in the analysis environment;
- *Bytes Transmitted* - Quantify the number of conversations that transmitted more than 1000 bytes (inbound and outbound). The number of bytes transmitted in a conversation could reveal prominent attacks and actions performed by the malware. This threshold (1000 bytes) was defined based on the analysis reports. Conversations with less than 1000 bytes tend to be less significant regarding revealing malicious behaviors;
- *Conversation Time Elapsed* - Describes the number of network conversations lasting over to 2 s. Likewise *Bytes Transmitted*, the conversation duration reveals valuable characteristics of the sample analyzed. In association with another parameter, it is possible to identify specific malware behaviors such as network scan (multiple conversations targeting distinct port in a short time window). Again, a manual investigation allows defining the threshold (2 s) that disclose more significant connections in terms of revealing malicious behavior.

The heterogeneous nature of the parameters aims to reflect the particularities of each analysis and, more consistently, compare results from the different analysis. In addition to the defined parameters, the results produced by our solution should ide-

Table 2
Comparison parameters extracted from the malware `TR/Dropper.Gen` when analyzed in *Reference System* and MARS.

Solution	Conversation	IP	Signature	Duration	Bytes	TCP	UDP
Reference	11	6	2	1	1	0	3
MARS	12	8	5	4	2	0	3

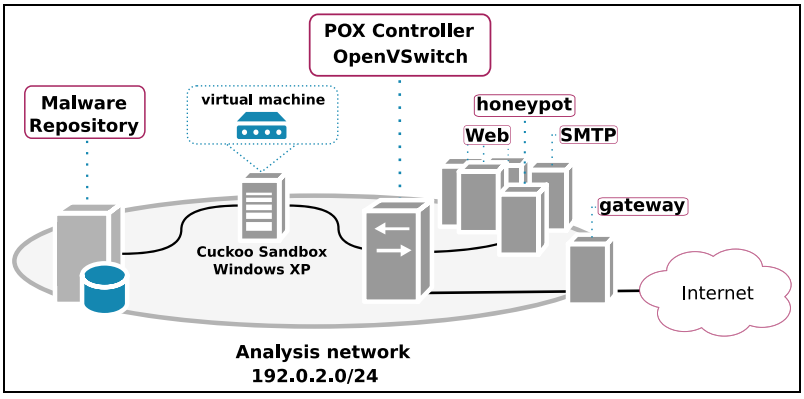

Fig. 5. Malware analysis environment setup.

ally be confronted with other architectures for comparison purposes. However, solutions related to our architecture, as described in Section 2, are not publicly available and cannot be used for comparison. Thus, our methodology uses a similar approach to that proposed by Balzarotti [7], in which a reference system is implemented to draw a baseline for the analysis results. For this reason, our solution is based on the approach of comparing the execution of the malware in a reference system. As a result, each malware sample is analyzed in both systems (MARS and Reference) producing a set of behavioral reports from where the parameters of our methodology are extracted.

An example that illustrates a comparison line is presented in Table 2. The parameters provided by `TR/Dropper.Gen` are timidly contrasting. However, in this comparison, MARS triggered more events related to the following parameters: *IP, Signature, Duration, Bytes*. In numbers, the sample analyzed in MARS revealed: 12 conversations; 8 distinct IP addresses; 5 sandbox signatures; 4 conversations with more than 2 s; 2 conversations with more than 1000 bytes transmitted each; and, 3 distinct UDP ports. When the number of distinct *Conversation* is observed individually, the *Reference System* shows more events than MARS. However, when other parameters are observed it is possible to have a more precise view of the analysis characteristics.

Finally, our methodology provides the set of parameters to depict the solution effectiveness. Based on these parameters, an analyst can analyze reports and determine what solution has triggered more events. The next section describes our experimental analysis based on the parameters presented in this section.

5. Case study

We conducted a set of experiments involving a number of distinct malware samples. The objective is to detect behavioral deviations triggered by our solution. The experiments were carried out using the same sandbox tool and identical malware database, but using different environment configuration in each experiment, as described as follows.

5.1. Malware samples

We used a heterogeneous malware database to conduct the experiments. Inspired by the experiments performed by Balzarotti [16], we defined a subset of 100 malware collected from public sources, such as VirusShare.[1] The samples were selected observing a set of requirements to avoid malware analogous behaviors, including: (i) unique ClamAV antivirus signature family [20]; (ii) unique malware hash, and (iii) fuzzy hash de-duplication [21]. Furthermore, to increase the diversity of binaries the database was composed of 50% of malware classified as “APT” by VirusShare, and 50% of samples classified as “non-APT”. Thus, we could identify whether our solution is more efficient when analyzing malware that considers the environment that surrounds it. A complete list of analyzed malware and respective signatures are available in the project website.[2]

5.2. MARS experimental environment

Fig. 5 depicts our experimental malware analysis environment. MARS prototype was implemented using OpenVSwitch and the POX controller [22,23]. The experimental environment also has a set of resources that can be dynamically integrated into the analysis environment in which the available resources comprise:

- Web server running a default configuration of Nginx [24] version 1.6.2, running on a Linux Debian 8 machine;
- Web server running Wordpress on the top of Apache server version 2.4.23;
- SMTP server is a Linux Debian 8 machine running Postfix (version 3.0.2) customized to accept mail message, but without forwarding them;
- Honeyd is a honeypot [25] server that runs arbitrary services to collect information about attacks. In this environment, the honeypot emulates the following services: TELNET (23/TCP), FTP (21/TCP), VPN (5900/TCP), IRC (6667/TCP).

[1] http://www.virusshare.com/.
[2] http://mars.botlog.org/.

Those elements are used to compose or to modify the analysis environment topology during the malware execution. Together with the provided services, our experimental setup employs Cuckoo Sandbox [10] with minor modifications to perform the analysis process in a Windows virtual machine. The sandbox source code and configuration used is available on the project Website.[2] Every malware was executed in a *Windows XP Home Edition Service Pack 3*, using Internet Explorer 6.

The element *Malware Repository* stores the malware samples and provides them to the sandbox analysis. To this end, the repository has a dedicated interface to the environment sandbox able to submit samples to the malware analysis. Finally, for convenience, we used a network gateway to provide flexibility to our setup so that we can move the architecture over different network setups and avoid external network address conflict.

5.3. Prototype implementation

As required by the POX controller, all SDN modules used in our solution were implemented in Python. Particularly, we used the common OpenFlow interface provided by POX to achieve the network layer reconfiguration. In this sub-section, we present the main features implemented by our solution to improve the malware analysis process regarding the network containment policy and environment reconfiguration rules.

A. Containment policy rules

The access policy provided to the sandbox can truly affect the malware analysis results. To detect this effect, we implemented a set of predefined containment access policies used in an experimental analysis to carry out our evaluation. Based on that, we defined three containment policies to characterize the network access profile of all elements in the malware analysis environment. The predefined policies are:

- *open* - No network traffic restrictions are imposed *i.e.*, all protocols in any direction (outbound or inbound) are permitted;
- *partial* - No restrictions are posed to the local network, but the access to external services (Internet) is limited. The allowed services are: DNS (53/TCP), DNS (53/UDP), HTTP (80/TCP), DHCP (68/UDP), and IRC (6667/TCP);
- *close* - Local network access is permitted, with no external or Internet network access.

Note that the access policy is defined in the malware analysis process initialization (see Fig. 4), although this policy can be modified. A specific traffic behavior, for example, can modify the access policy and block all the connections to a particular target.

B. Environment reconfiguration rules

The environment reconfiguration enables MARS to modify the setup surrounding the sandbox during the malware analysis. Modifying the environment in which the malware is executed is another way to encourage a malware to reveal unseen actions. While the analysis environment can assume multiple configurations, in our prototype we specified a set of predefined reconfiguration actions available in the experimental scenarios. The predefined reconfiguration actions are structured into three categories:

- *Network service inclusion* - When the controller detects a packet aimed to port 80/TCP, it adds a Web server in the analysis environment by activating the OpenVSwitch port. The idea is to observe attacks towards the Web server or to a specific Web application, such as Wordpress. In particular cases, the Web server included in the environment exposes attack details that are not shown in other analysis setups;

Table 3
Deployed experiments and their respective environment configuration aspects.

	Network Containment Policy Rules (Section 5.3 A)	Environment Reconfiguration Rules (Section 5.3 B)
Experiment I	–	✓
Experiment II	✓	✓

- *Network traffic redirection* - The SDN controller redirects specific predefined requests to the honeypot to collect more details about the attack. When the analyzed malware sends packets aimed to key ports – TELNET (23/TCP), FTP (21/TCP), VPN (5900/TCP), IRC (6667/TCP) –, the controller promptly redirects it to the respective emulated services in our honeypot. The honeypot provides a limited interaction with the attacks but incorporates network services in the environment that can collect sensitive information, such as credentials used by the malware;
- *Packet manipulation* - The controller inspects all DNS requests and rewrites packets whose request cannot be translated to an IP address (`NXDOMAIN`). DNS packets with the error message `NXDOMAIN` are rewritten to a valid entry, pointing to the honeypot IP address. As a consequence, the malware action associated with the initially unavailable addresses can be examined in the honeypot. The connections directed to emulate honeypot services are properly handled and connections targeting non-emulated services are saved for post analysis.

These reconfiguration actions and the access policies described in this section were used to setup the analysis environment used in our experiments. Next, we describe the *Reference System* used to perform our experiments and to support analyses evaluation.

5.4. Reference system environment

The *Reference System* is a malware analysis environment implemented for comparison purposes. Indeed, this environment was deployed using a simplified version of MARS infrastructure, as depicted in Fig. 5. The objective is to map the evaluation parameters in an environment similar to the MARS setup but without the functionality provided by MARS, such as environment reconfiguration and containment access policy. It is thus possible to build a baseline of results and expose the behavioral aspects triggered by MARS.

In the *Reference System* implementation, we used the same sandbox configuration and networking IP addressing setup previously described. The additional components comprising the environment, such the Web Servers, SMTP server, and the honeypot system, were also included in this environment for comparison consistency. However, the POX controller and OpenVSwich were removed from the environment since OpenFlow rules are implemented. As a result, the actions performed by the malware in the sandbox do not suffer intervention from the analysis system. Furthermore, the sandbox has direct communication with the network layer whereby its connections have full access to the Internet and locally available services. In the next section, we present the experiments performed in this *Reference System* and compare them to the MARS solution.

6. Experimental results

The experiments performed in MARS were conducted using the infrastructure depicted in Fig. 5, where each malware sample was executed by the sandbox for 3 min. We designed two experiments to evaluate the malware samples analysis, as summarized in Table 3.

The experiments deployed introduce characteristics to the analysis environment associated with network access policies and dynamic reconfiguration. In Experiment I, MARS is configured to apply the environment reconfiguration ruleset. In Experiment II, rules associated with network containment are combined into environment reconfiguration rules. The malware samples were executed in both environments to determine which setup configuration is more effective to trigger malicious behavior and, more widely, how the environment surrounding the sandbox affects the malware analysis. The experiments are discussed as follows.

6.1. Experiment I

In the first experiment, the objective is to identify the behavioral deviation in malware when rules associated with traffic redirection, packet manipulation, and network service inclusion are applied to the setup. To this end, we used the ruleset described in Section 5.3 to manage the network conversations performed by the malware.

The 100 malware were analyzed producing 200 analysis reports (100 analysis using MARS and 100 analysis using the *Reference System*). An automated process extracts the evaluation parameters and provides the information to study the malware actions. Based on the methodology defined (Section 4), a specialized analyst can identify malicious behavior modifications. In summary, the experiment detected 38 malware – *i.e.* 38% of the samples – exposed novel malicious behavior when executed in MARS.

For example, the MARS analysis associated with the malware SHA1 hash `c4343f6b7d899ceae825af31209985824e97cd2` showed more events when compared to the *Reference System*. In general, all the evaluation parameters presented a sensitive increase, being: 1 new malware behavioral signature; 14 new IP conversations; 7 new connections with more than 1000 bytes transmitted; 13 new conversations with more than 2 s; 189 new IP addresses; and 1 new TCP port. As characterized, our system enabled this malware to reveal actions that were not shown in the *Reference System*.

A longer investigation on the malware showed that its behavior is conditional to a specific FTP server. The malware fully shows its actions when a specific FTP server is available in the analysis environment. However, this particular FTP server was unavailable (`NXDOMAIN`) when the analysis investigation was executed. As consequence, in the *Reference System*, this malware stops its execution since traffic manipulation rules are not present. However, in MARS, because of the DNS packet manipulation and network service inclusion rules, the malware could access an FTP server and expose its actions.

Despite being effective in most of cases, this packet manipulation can present side effects in particular cases, for example, the behavior detected the malware ef40b9699a3b9156219b1b1ae1bb920f61cda37d. This malware tries to access a specific IP address before starting network scanning. However, this IP address was unavailable (`NXDOMAIN`) and MARS forwarded its conversations to the local Web server. The local Web server does not have what the malware expects and the sample does not start the aggressive network scanning as it would do in a regular analysis. Nevertheless, this analysis evidences this malware behavior and could lead an analyst to reconfigure the analysis environment to address this situation.

The features provided by MARS were also effective to characterize the network traffic flows. We found few samples that try to send massive mail messages to distinct mail servers. As designed in this experiment, all connections associated with SMTP service (25/TCP) were redirected to our local mail server. This configuration allowed us to study the SPAM campaigns and collect information from attacks that would not be possible to collect without traffic manipulation.

In general, the environment reconfiguration rules describe an important feature to trigger malware actions. The services requested by the malware are essential to its execution and providing them, even partially, could reveal important malware execution aspects.

Table 4
Events performed by malware *Win.Downloader.81796-1* executed in different environment setups. These events are summarized and normalized to avoid events duplication and then compared to the *Reference System*.

Containment Profile	Conversation	IP	Signature	Duration	Bytes	TCP	UDP
Open	43	19	5	28	34	3	6
Partial	45	21	6	34	32	2	5
Close	47	24	5	32	33	2	6
MARS Unique Events	76	37	6	52	50	3	6
Reference	25	20	5	14	10	2	5

6.2. Experiment II

In our second experiment, we extrapolate the previously experiment by incorporating configuration rules associated with network containment policy. In this experiment, each sample was executed in MARS for three times using distinct containment policies.

In the first execution cycle, the malware is analyzed using the profile “open” equivalent to the setup implemented in *Experiment I*. In the second, the containment profile “partial” is used and, in the third cycle, the profile “close” is used. The three analysis reports provided by MARS executions are summarized and compared to the *Reference System*. To characterize this process, we created comparison tables in which the evaluation aspects for every malware analyzed are summarized. Table 4 exemplifies the aspects for sample SHA1 hash `d20581741e64c8306fa94c7c8605e768a83168d4`, classified by ClamAV antivirus as `Win.Downloader.81796-1`. Other evaluation aspects related to other malware families analyzed can be found on the project Website.[2]

This experiment was more effective than the first experiment to reveal unseen malicious actions. By executing the malware in containment profiles, combined with environment reconfiguration and packet manipulation, we detected a behavioral deviation in 100% of the sample analyzed. An extended analysis of the network conversations reveals how malware are dependent on the Internet. A number of types of malware simply abort or become dormant if they cannot connect to the Internet. Our experiment showed more network events in “partial” and “open” profiles, which permitted access to the Internet is permitted at different levels. The profile “close” showed a decrease in the number of network events. The distinct access profiles showed the malware adaptation scheme: few samples try to access different ports to inspect the available network access policy. For example, malware SHA1 hash `b3423fef635638bb078b01c34166d1e6eb638d36` tries to connect to multiples services across the Internet. However, when one of those services is not accessible because of policy restrictions, the malware uses alternative ports. This malware, when executed in the profile “partial”, modifies its behaviors to adapt to the network access policy. Lastly, we observe that most of malware tries to reach the Internet using TCP ports associated with Web traffic and with proxy services. The most observed TCP ports were: 80/TCP in 99% of the analysis, 443/TCP in 83% of the analysis, and 1053/TCP in 20% of the analysis.

7. Conclusions

In this paper, we propose a solution to improve the dynamic malware analysis process. Since the actions performed by malware in a sandbox are highly dependent on the surrounding environment, it is fundamental to provide flexibility to implement distinct environment setups. We developed a centralized solution using Software-Defined Networking (SDN) that integrates the sandbox operation with the analysis environment components. The malware analysis environment could assume multiple network configurations regarding the topology, network access policy, and services availability. Moreover, the environment can dynamically modify its characteristics based on actions performed by the malware actions. That flexibility allows identifying malware behavioral deviation and reveals unseen actions performed by the malware.

To demonstrate our solution, we analyzed a set of 100 malware samples in MARS in two experiments, using distinct environment configuration. In the first experiment, focusing on environment reconfiguration and traffic manipulation, it was possible to observe new malicious behaviors in 38% of the samples analyzed. In the second experiment, in which distinct access policies were incorporated into the environment, we detected a behavioral deviation in 100% of samples analyzed. In summary, the experiment evaluation lead us to the following conclusions:

- The proposed analysis environment for dynamic malware analysis process affects the quality of the information available in the reports provided by the sandbox. A proper environment setup increases the activities performed by the malware in the sandbox;
- The integration of the environment configuration as part of the malware analysis process enables to deploy a complex analysis environment easily;
- Simple modifications in the environment configuration trigger notable changes in the sample behavior. Reconfiguration rules that include traffic redirection and packets manipulation can be customized to particular malware families improving the results;
- The process of detecting behavioral deviation should observe multiple parameters to avoid impressions;
- In the analyzed set of malware, we identified samples that adapt their behaviors based on access policies. Several malware try to access different service ports when a particular port is unreachable. Particularly, we observed this behavior is more associated to TCP ports than to UDP ports;
- Traffic redirection and DNS packet manipulation were effective tools to trigger unseen behaviors in most of the malware analysis;
- The greater part of malware requires external network resources, such as contacting a service or downloading additional components. In the absence of those resources, the malware aborts it execution or becomes dormant during the sandbox analysis.

More than exposing novel malicious behaviors, MARS provides valuable analysis capabilities unavailable in current tools. MARS presents a set of resources to reconfigure the analysis environment and to automate the analysis process. However, to improve the analysis results it is necessary to tune the environment based on malware characteristics. To trigger inactive behaviors from modern malware such as APT, an analyst should observe the sample particularities to specify a proper analysis environment. Nevertheless, MARS brings this flexibility to the analyst, by providing the support to reconfigure the available resources and to identify the most suitable solution.

In MARS designed architecture, the solution is dependent on the action performed by the malware in the sandbox, and this also introduces variables that may interfere with the solution results. As detailed in Section 2, several malware implement procedures and heuristics to detect a sandbox system and to inhibit its actions. When the malware does not properly run in the sandbox, it also affects the MARS effectiveness. Our architecture, however, does not restrain or specify the sandbox technology. Moreover, we support the use of different sandbox tools by enabling the use of the ones that better fit the analysis requirements. Yet, the sandbox is a critical asset and must be properly selected to compose the architecture by observing its limitations and the improvements demanded.

MARS is vulnerable to attacks and threats targeting the SDN paradigm. If the controller is affected by an attack, the system cannot properly handle the malware analysis process. Additionally, the elements that comprise the analysis environment may be attacked by malware or fingerprinted. Likewise, the actions performed by the SDN controller to manipulate packets could be fingerprinted by malware that inspects environment anomalies. Thus, in the future, we should implement mechanisms to ensure system integrity and study techniques to prevent resources fingerprinting. Furthermore, we are working to enhance the dynamic configuration provided by our solution to stimulate new malware behaviors. Therefore, we expect to extensively analyze the malware ecosystem including malware associated with IoT devices and smartphones.

References

[1] L. Mclaughlin, Bot Software Spreads, Causes New Worries, IEEE Educational Activities Department, Piscataway, NJ, USA, 2004.
[2] D. Kirat, G. Vigna, C. Kruegel, Barecloud: bare-metal analysis-based evasive malware detection, in: Proceedings of the 23rd USENIX Conference on Security Symposium (SEC'14). USENIX Association, Berkeley, CA, USA, 2014, pp. 287–301.
[3] D. Fleck, A.G. Tokhtabayev, A. Alarif, A. Stavrou, T. Nykodym, Pytrigger: a system to trigger & extract user-activated malware behavior, in: 2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013, 2013, pp. 92–101.
[4] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, Y. Zhou, Understanding the mirai botnet, in: 26th USENIX Security Symposium (USENIX Security 17), USENIX Association, Vancouver, BC, 2017, pp. 1093–1110.
[5] X. Chen, J. Andersen, Z.M. Mao, M. Bailey, J. Nazario, Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware, in: Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on, IEEE, 2008, pp. 177–186.
[6] N. Virvilis, D. Gritzalis, The big four-what we did wrong in advanced persistent threat detection? in: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, IEEE, 2013, pp. 248–254.
[7] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, G. Vigna, Efficient detection of split personalities in malware., NDSS, Citeseer, 2010.
[8] J. Ceron, C. Margi, L. Granville, Mars: an sdn-based malware analysis solution, in: ISCC 2016 - The twenty first IEEE Symposium on Computers and Communications, IEEE, 2016.
[9] S. Sezer, S. Scott-Hayward, P.-K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao, Are we ready for sdn? Implementation challenges for software-defined networks, 51, IEEE, 2013, pp. 36–43.
[10] D. Oktavianto, I. Muhardianto, Cuckoo Malware Analysis, Packt Publishing Ltd, 2013.
[11] C. Kruegel, E. Kirda, U. Bayer, Ttanalyze: a tool for analyzing malware, in: Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference, 4, 2006.
[12] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, 44, ACM, 2012, p. 6.
[13] C. Willems, T. Holz, F. Freiling, Toward automated dynamic malware analysis using cwsandbox, IEEE Security & Privacy, 2007.
[14] A. Moser, C. Kruegel, E. Kirda, Exploring multiple execution paths for malware analysis, in: Security and Privacy, 2007. SP'07. IEEE Symposium on, IEEE, 2007, pp. 231–245.
[15] H. Yin, Z. Liang, D.H. Song, Identifying and un-derstanding malware hooking behaviors, Proeeedings of the 16th Network and Distributed System Security Symposium (NDSS 2008). San Diego, 2008.
[16] M. Lindorfer, C. Kolbitsch, P.M. Comparetti, Detecting environment-sensitive malware, in: Recent Advances in Intrusion Detection, Springer, 2011, pp. 338–357.
[17] M. Graziano, C. Leita, D. Balzarotti, Towards network containment in malware analysis systems, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACM, 2012, pp. 339–348.

[18] C. Kreibich, N. Weaver, C. Kanich, W. Cui, V. Paxson, Gq: practical containment for measuring modern malware systems, in: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference, ACM, 2011, pp. 397–412.
[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, 38, ACM, 2008, pp. 69–74.
[20] ClamAv Project, ClamAv - The Open Source Anti-Virus Engine, [Accessed 10 September, 2017].
[21] B. Stein, Fuzzy-fingerprints for text-based information retrieval, in: Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), Graz, 2005, pp. 572–579. Journal of Universal Computer Science.
[22] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of sdn/openflow controllers, in: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, ACM, 2013, p. 1.
[23] B. Pfaff, OpenvSwitch - distributed virtual multilayer switch, [Accessed 10 September, 2017] (2015).
[24] Nginx - high performance load balancer, web server, [Accessed 10 September, 2017].
[25] N. Provos, Honeyd-a virtual honeypot daemon, in: 10th DFN-CERT Workshop, Hamburg, Germany, 2, 2003, p. 4.

João Marcelo Ceron is a Ph.D. candidate at the University of Sao Paulo. He holds a master's degree from Federal University of Rio Grande do Sul, Brazil, where he worked with honeypots for botnet detection. He currently works with incident handling at CERT.br/Nic.br, and his research interests include malware analysis, honeypots, and network flows data analysis.

Cíntia Borges Margi obtained her BS (1997) and MSc (2000) in Electrical Engineering at University of São Paulo, her PhD in Computer Engineering at University of California Santa Cruz (2006) , and her Habilitation (Livre Docência) (2015) in Computer Networks from the University of São Paulo. She is Associate Professor at the department of Computer and Digital Systems Engineering (PCS) at Escola Politécnica - Universidade de São Paulo (EPUSP) since 2015, where she started as Assistant Professor in June/2010. Before that she was Assistant Professor at Escola de Artes, Ciências e Humanidades da Universidade de São Paulo (EACH-USP) from February 2007 until June 2010. Her research interests include: wireless sensor networks (protocols, systems, security, energy consumption and management, embedded hardware) and software-defined networking.

Lisandro Zambenedetti Granville is an associate professor at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He received his M.Sc. and Ph.D. degrees, both in computer science, from UFRGS in 1998 and 2001, respectively. He is president of the Brazilian Computer Society (SBC), co-chair of the IRTF's Network Management Research Group (NMRG), chair of IEEE ComSoc's Committee on Network Operations and Management (CNOM), and member of the Brazilian Internet Committee (CGI.br). He has served as a TPC member for many important events in the area of computer networks (e.g.,IM, NOMS, and CNSM) and was TPC co-chair of DSOM 2007 and NOMS 2010.