Communications Research Centre Canada
An Agency of Industry Canada
Centre de recherches sur les communications Canada
Un organisme d'Industrie Canada

# Last Minute Traffic Forwarding for Malware Analysis in a Honeynet

**Mathieu Couture, Frédéric Massicotte & Daniel Rea**

CRC Technical Note No. CRC-TN-2010-001
Ottawa, June 11th 2010





Canada
CRC

# Contents


**1 Introduction** **3**

**2 Related Work** **4**

**3 Problem Description** **6**

**4 Configuration Language** **8**

**5 Implementation** **10**

**6 Case Study** **11**

6.1 Total Size of Generated Raw Data . . . . . 13

6.2 TCP Traffic on the LAN Network . . . . . 13

6.3 TCP Traffic on the SIM-INTERNET Network . . . . . 15

6.4 DNS Requests . . . . . 16

6.5 IRC Traffic . . . . . 16

6.6 Intrusion Detection System Alarms . . . . . 17

6.6.1 Snort Alarms . . . . . 18

6.6.2 Emerging Threats Alarms . . . . . 19

6.6.3 Conclusion . . . . . 20

**7 Conclusion** **20**

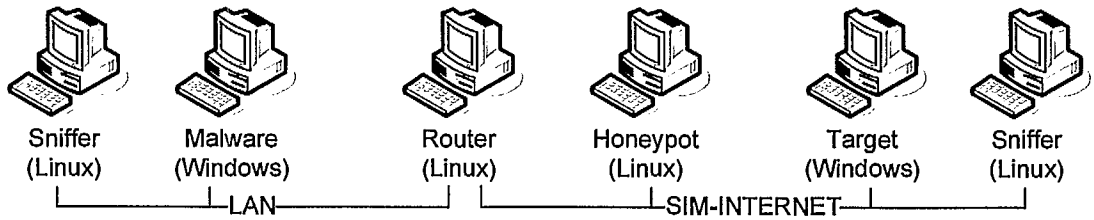


Figure 1: Simple isolated network topology.

# 1 Introduction

Over the last years, efforts have been made to develop malware sample gathering techniques. The samples potentially contain valuable information about the worldwide malware activity. For example, the samples may help in determining what are the most popular infection vectors, what malicious actions are generally being performed, what do the control mechanisms look like in the case of botnets and what are the IP addresses/domain names being targeted. This information may help answering, among others, the following questions: What are the trends in Internet threats? Who is behind them? What parts of the world seem to be responsible for (or targeted by) these threats? What can we do to detect and/or mitigate them?

Extracting this information from the malware samples is not a trivial process. Generally, malware analysis techniques are grouped into two categories: the *dynamic* category, where the samples are executed in a simulated, emulated, virtual or real environment, and the *static* category, where the samples are not executed. The work presented here falls into the dynamic category.

In previous work [4], we presented a virtual framework that allows to study network-related aspects of malware in an isolated environment. It is clear that, technically speaking, malware execution in a network that is not connected to the Internet will most likely produce less information than if an Internet connection was allowed. However, for ethical, political, or legal reasons many organizations choose not to use the Internet or would only resolve to do so in case of extreme necessity. Therefore, it is desirable to develop techniques that allow as much information as possible to be extracted from malware without providing it with Internet access.

Prior to execution, the IP addresses a given malware sample may attempt to contact are unknown. In [4], the strategy we developed to address this problem was to reduce the Internet IP address space to the IP addresses of a small number of hosts that are in an isolated network. To implement this strategy, we used the Destination Network Address Translation (DNAT) functionality of iptables[1] to decide to which host each packet would be forwarded based on the destination port (TCP or UDP). Basically, we forwarded DNS traffic to a Linux host running a custom DNS emulation script and forwarded all the remaining traffic to a Window host in order to study worm

[1] `http://www.netfilter.org/`

propagation mechanisms (see Figure 1). Two of our conclusions were: (1) The generated traffic traces, from which the desired information is extracted in a post-processing phase, contained a great amount of redundant connections, mainly due to worms attempting to contact a high number of IP addresses in order to propagate; and (2) When using such a simplistic packet forwarding strategy, connections being attempted on ports that are closed on the Windows host are not being established, thereby leading to a potential loss of valuable information.

Some of these ports correspond to standard protocols such as SMTP and a simple upgrade of the Linux host (i.e., add more emulation scripts) may solve the problem. However, some connections are also attempted on non-standard ports. In this case, an arbitrary decision has to be made. Depending on the expected payload, one can decide to either forward these connections to a given host by leaving the port unchanged or to a particular port of a particular host that is known to be open. For example, if one believes that port 6767 will be used for IRC, then port 6767 should be forwarded to port 6667 (the defacto IRC port) of a host running an IRC server on that port. However, there are worms that propagate through a backdoor that is open by some exploit payload. For example, that is the case of Blaster, that opens a backdoor on port 4444. In that case, port 4444 should be forwarded to the Windows host.

Since the configuration of the DNAT device is made prior to executing malware, a wrong choice of DNAT rule may lead to some potential information not being revealed and the only way around is to execute malware several times, testing a different DNAT strategy at each time. This process can be costly in time, especially when physical hosts are being used to perform the investigation.

In this paper, we address the problem of forwarding TCP/IP traffic in an isolated network that is used to observe malware execution. More specifically, we present a strategy that allows the decision of which port of which host will handle a particular connection to be postponed until the very moment that it is being established. We call this strategy *Last Minute DNAT* (LM-DNAT). The contributions we make are the following: (1) We identify the features that an LM-DNAT module should have. (2) We propose a configuration language that enables those features. (3) We describe an implementation that allows LM-DNAT to be performed using a Linux host. (4) We present a case study performed using a corpus of 25118 malware samples. These contributions are respectively presented in Section 3, Section 4, Section 5 and Section 6. We conclude in Section 7. We first present an overview of related work.

## 2 Related Work

Dynamic analysis of malware presents a security risk since it involves the execution of software that is designed to perform undesirable actions. The efforts that have been deployed to circumvent that problem led to the establishment of a research field known as *Sandboxing*. A *sandbox* is a software

tool that allows the safe monitoring of the execution of malware or other kinds of programs. Well-known sandboxes include Norman Sandbox [7], CWSandbox [10], and Anubis [3]. These systems mainly aim at describing what malware does *inside* the host.

When focusing on what happens *outside* the host (i.e., on the network) a honeypot may be used to handle some of the generated traffic. A *honeypot*, as defined by Spitzner [9], is "*a security resource whose value lies in being probed, attacked or compromised*". Honeypots, according to Bailey et al. [2], can be classified according to their breadth and depth, where the breadth refers to geographic aspects and the depth refers to the level of interaction. Honeypots with high depth are also called *high-interaction* honeypots. Typically, a high-interaction honeypot is deployed using a computer with a real operating system installation that is made available on the Internet for hackers and/or malware to attack. Attacks can thus be observed with a high degree of realism. However, high-interaction honeypots have to be closely monitored in order to prevent them from being used to cause further damage. Also, the operating system has to be re-installed or reverted to some original state from time to time. At the other end of the spectrum, *low-interaction* honeypots like Honeyd [8] present attackers and malware with minimal network functionalities such as the network stack. In between, *medium-interaction* honeypots such as Nepenthes [1] and SGNET [6], aim to offer a compromise by emulating only the parts of the application layer that are necessary for malware to conduct its malicious actions. This strategy allows live malware propagating on the Internet to be studied without worrying too much about the host being infected or having to revert the system to an original state every now and then. A computer network composed of more than one honeypot is called a *honeynet*.

In this paper, we study the problem of forwarding IP traffic to a honeynet for the purpose of analyzing malware samples in an isolated environment. The general idea is to favor high-interaction honeypots for most traffic, thus offering a high degree of realism, while using medium or low-interaction honeypots to simulate network protocols that are used by malware or to handle traffic on ports that are not open on the high-interaction honeypot. We discuss the problem in the context of an isolated environment, but we believe that the concepts we develop could also be used in a semi-isolated environment where an Internet connection would be used to allow connection to command and control (C&C) servers, test Internet connectivity, or download additional code. Our main contribution is a mechanism called *Last Minute Destination Network Address Translation* (LM-DNAT), that tests whether the destination port of the connection is open *at the time the connection is being established.* This mechanism allows decisions, about which of the honeynet hosts should handle a particular connection or group of connections, to be made at the last minute.

# 3 Problem Description

Figure 1 (page 3) shows a simple network topology that can be used when studying malware from a network point of view. It basically consists in two local area networks. One that we call LAN, where we find a Windows host called Malware on which the malware is being introduced and executed. The other network, that we call SIM-INTERNET, aims to provide malware with as many resources as possible in order for it to perform as many *undesirable* actions as possible (in our case, we actually *desire* malware to perform these actions). In the setup that we used for our study, it just consists of three hosts: one called Sniffer, which is there to record the generated traffic, one called Target, that is used to study the propagation mechanisms, and one called Honeypot, that provides or emulates useful network services such as a DNS server, an IRC server, an SMTP server, a netcat[2] server, etc.

The Router host forwards packets between the LAN and SIM-INTERNET networks. The challenge in this research is about finding the *best* way of forwarding packets, in terms of the amount of information that can be extracted from the analysis of the traffic traces that are being recorded by the two Sniffer hosts during malware execution. Forwarding packets is done using a *forwarding module*, a piece of software that receives packets on one interface and sends them on another after relabeling of the IP addresses and TCP or UDP ports. A well-known example of a forwarding module is Netfilter[3], which is often configured through iptables. In previous experiments [4], we found that the functionalities of iptables were useful although insufficient. For example, iptables lacks the possibility of testing whether a given port is open before forwarding the packets of a connection.

In this section, we give a list of requirements that we believe should be taken into account when designing a forwarding module for purposes of malware analysis. The list we provide is not meant to be exhaustive, it is simply a list of features that, based on our experience, we found were desirable.

Upon execution of a given malware sample, some IP traffic going to a priori unknown IP addresses might be generated. Since these addresses are unknown, one cannot pre-configure a network that contains all the right IP addresses. For that reason, Destination Network Address Translation (DNAT) is the de facto mechanism to be used when analyzing malware in an isolated network. While Network Address Translation (NAT) allows multiple hosts to use the same IP address, DNAT allows multiple IP addresses to be forwarded to the same host. This functionality, built into iptables, allows one to statically decide, prior to executing malware, which host will receive which traffic. The forwarding rules can be expressed in terms of any combination of source/destination addresses and ports. Any port forwarding module should be able to do at least as much as iptables

[2] http://en.wikipedia.org/wiki/Netcat

[3] http://www.netfilter.org/

in terms of DNAT. In fact, as we show in Section 5, it is sufficient to write a script that dynamically reconfigures Netfilter through iptables.

**Requirement 1 (DNAT)** *A forwarding module should support translation of destination IP addresses and TCP/UDP ports.*

To decide to which port of which host a given connection should be forwarded is a relatively simple matter when the destination port is known to be open on one of the hosts that is on the SIM-INTERNET network. For example, all NetBIOS traffic naturally goes to the Target host and all DNS traffic goes to the Honeypot host. However, when a connection request is being issued to a port that is not known to be open, a decision has to be made depending on what it is assumed to mean. For example, if that connection is being established after the payload of an exploit previously sent to the Target host opened a socket on that port, then the connection should be forwarded to the Target host. If the exploit failed and the port is not open, then one might consider to forward it to a netcat port in order to receive the first few bytes of data (if there is any data). It could also be the case that the connection simply carries IRC traffic, in which case it should be forwarded to an IRC server. At the very least, we should be able to test whether a given port is listening before forwarding any traffic to it.

**Requirement 2 (Test)** *A forwarding module should support TCP/UDP port testing.*

Coming back to the case where the payload of an exploit may or may not succeed in opening a port on the Target host, one might prefer to let the malware know that the port is actually still closed. For all we know, the malware could also simply be scanning the Target host in order to determine what is the best thing to do next. Therefore, once a given IP address has been mapped to one of the hosts in the SIM-INTERNET network (through the forwarding of a connection), one should be able to decide whether or not subsequent connections made to that address on other ports should also be forwarded to that same host. We call this concept *vertical impersonation.* Similarly, if we want a particular host of the honeynet to impersonate all hosts offering a particular service, then we talk about *horizontal impersonation.* More generally, one should be able to decide which parts of the 5-tuple (protocol and source/destination addresses and ports) identifying a flow should be used in order to forward traffic, which makes $2^5$ different types of impersonation, of which vertical and horizontal impersonation are just two specific cases.

**Requirement 3 (Impersonation)** *A forwarding module should support the specification of any type of impersonation.*

The analysis of worms that are trying to massively propagate by infecting as many hosts as possible may become difficult in a network where all traffic is in fact handled by just two hosts.

Therefore, it is useful to be able to specify some thresholds regarding how many connections are handled by a given host, how many hosts a given host is allowed to impersonate, how many connections a given port is allowed to receive, etc.

**Requirement 4 (Thresholds)** *A forwarding module should support the specification of thresholds about how many different addresses/ports a given address/port is being forwarded to.*

A last requirement, that we did not address in this work but that we still think would be useful for malware analysis, is payload inspection. For example, IRC connections are always initiated by the client (i.e., the first packet containing payload is sent by the client). We could take advantage of that in order to validate the forwarding of a given connection to an IRC server. In that case, we would have to postpone the decision of which port and host to forward the connection to until reception of that first packet carrying payload. This becomes tricky in cases where the application layer connection is initiated by the server, in which case the client simply waits for the server to send some payload once the TCP handshake has been completed. We decided to leave this issue for future work.

**Requirement 5 (Payload Inspection)** *A forwarding module should support the postponement of address/port translation until the first packet containing payload is being sent.*

# 4 Configuration Language

We developed a configuration language to specify how traffic should be forwarded. An LM-DNAT configuration consists in a list of rules that we call *LM-DNAT rules* in order to distinguish them from *DNAT rules*, which are written in the iptables configuration language. Although we did not build our configuration language as an extension of the iptables configuration language, we believe that it would be possible to do it if one wanted to implement LM-DNAT within iptables (rather than as a wrapper of iptables, which is what we did).

As with iptables, the configuration file is read from top to bottom and the first LM-DNAT rule that *matches* is the one that applies. Rule matching involves two criteria: (1) the arriving packet must match some filter expressed in terms of addresses and ports (as with iptables), and (2) the port to which the packet is to be forwarded must be validated (i.e., the rule does not match if the port to which the packet is to be forwarded is not open). If no rule matches, the packet is dropped.

Consider, for example, the configuration shown in Figure 2 (page 9). The first rule means that if a TCP connection whose destination port is 80 is being established, then it should be forwarded to port 80 of 10.10.10.1. However, if port 80 of 10.10.10.1 was not open, then the second rule would be tested. The second rule requires the destination port to be 25, so it does not apply. The third rule says that any remaining TCP session (regardless of the destination port) should be forwarded

```
10.10.10.1 proto=tcp dport=80
10.10.10.2 proto=tcp dport=25
10.10.10.3:30 proto=tcp
```

Figure 2: LM-DNAT configuration example.

```
10.10.10.4 proto=udp dport=53 sport=0 dst=0 src=0
10.10.10.5 proto=udp dport=53 sport=0 dst=0 src=0
10.10.10.6 proto=udp dport=53 sport=0 dst=0 src=0
```

Figure 3: Horizontal impersonation example.

```
10.10.10.7 proto=0 sport=0 dport=0 src=0 matchNum=1
10.10.10.8 proto=0 sport=0 dport=0 src=0 matchNum=1
10.10.10.9 proto=0 sport=0 dport=0 src=0 matchNum=1
```

Figure 4: One-to-one impersonation example.

to port 30 of 10.10.10.3. Again, the session is only forwarded if port 30 of 10.10.10.3 is open. Otherwise, the SYN packet is simply dropped.

This simple configuration language already meets Requirement 1 and Requirement 2. In some applications, it might be useful that the matching of an LM-DNAT rule implies the forwarding of more than just one session. For example, one might need to specify that once a session with a given destination address has been forwarded to a particular host, then all subsequent sessions must also be forwarded to that same host. Also, we could imagine a situation where all sessions with the same destination port should be forwarded to the same host. For those reasons, we allow a rule to ignore any of the five terms of the 5-tuple identifying a flow. This is simply done by writing `src=0`, `dst=0`, `sport=0`, `dport=0` or `proto=0`. For example, consider the configuration file shown in Figure 3. It means that all DNS traffic (UDP port 53) should be forwarded to the first host that is found to be running a DNS server among 10.10.10.4, 10.10.10.5 and 10.10.10.6 in that specific order. Once a DNS server has been found, all other DNS requests, regardless of their source and destination address and source port, will be forwarded to that host, even if the port eventually became closed. This is how we addressed Requirement 3.

Let's now say that we have a collection of targets, and that we want each of them to impersonate at most one Internet host (we call this *one-to-one impersonation*). This is done by simply adding a threshold argument to each LM-DNAT rule. Consider the configuration shown in Figure 4. It makes use of a `matchNum=k` option, that allows a given rule to be matched at most a certain number of times. Once it has been matched `k` times, then it is simply deleted from the rule list and will

not be used to forward packets until the next execution (i.e., until the analysis of another malware sample). In the case shown in Figure 4, the first session whose destination port is found to be open on 10.10.10.7 binds its destination IP address to 10.10.10.7. Any other session whose destination IP address is the same will be forwarded to 10.10.10.7. Moreover, no session with a different IP address will ever be forwarded to 10.10.10.7 since the argument of the `matchNum` option is 1. The next session whose destination port is open on 10.10.10.8 and whose destination IP address is different will be forwarded to 10.10.10.8. This is how we addressed Requirement 4. As we already said, Requirement 5 is left for future work.

# 5 Implementation

Since the configuration language described in Section 4 simply specifies how to dynamically define Destination Network Address Translation (DNAT) rules, which are already well implemented in iptables, all we really had to do was to write a script intercepting packets in Netfilter before forwarding them. We wrote a Python script that makes use of the ipqueue library[4] to get incoming packets from Netfilter, of Scapy[5] to perform the port testing, and of ipqueue again to give the packet back to Netfilter. Figure 5 depicts the overall algorithm implemented by our script. When a packet arrives, Netfilter first checks if a DNAT rule has already been defined for that packet. If yes, then the packet is simply forwarded accordingly. Otherwise, it is passed on to our script, which traverses the list of LM-DNAT rules from top to bottom in order to decide which port of which host the packet should be forwarded to. Tests are performed using a Scapy script. We now described how these tests are performed for TCP, UDP and ICMP packets.

For TCP SYN packets, we simply spoof a TCP SYN packet to the destination address of the LM-DNAT rule using the IP address of the router's interface as source. If a SYN-ACK comes back, then a call to iptables is made to define a new DNAT rule and the original packet is given back to Netfiler, which forwards it according to that new DNAT rule. Subsequent packets on the same session will not be seen by our script. Depending on the elements of the 5-tuple that have been used to create the DNAT rule, some packets belonging to other sessions may also not be seen by our script. Since Scapy bypasses the TCP/IP stack, the router responds to the SYN-ACK that corresponds to the spoofed SYN packet with a RST, which prevents the destination host from being congested with hanging connections. If, following the spoofed SYN, a RST comes back or, after some definable timeout, nothing comes back, then the test fails and the next LM-DNAT rule (if there is one) is tested. If all rules have been tested without any success, then the packet is dropped.

[4] `http://woozle.org/~neale/src/ipqueue/`
[5] `http://www.secdev.org/projects/scapy/`

The strategy for UDP packets is similar, except that the failure of a test is defined by the reception of an ICMP Port Unreachable packet. Since UDP is connectionless we decided to take an optimistic approach to the interpretation of a timeout. If nothing is sent back after some definable timeout, then the test succeeds. If an answer comes back, then the test succeeds as well. For ICMP, only ECHO packets are supported for now. In that case, we perform the test by spoofing an ECHO packet whose destination IP address is the one specified by the LM-DNAT rule and whose source IP address is the one of the router's interface. A success is determined by the reception of a REPLY packet.

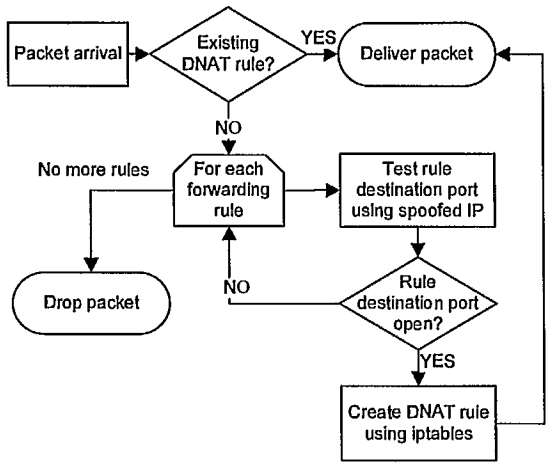


Figure 5: Implementation flow chart.

## 6 Case Study

To evaluate the usefulness of LM-DNAT, we performed a comparative analysis. The questions we were seeking to answer were the following: (1) Does using LM-DNAT to dynamically forward TCP/IP traffic produce more information about network-related actions taken by malware than using just static DNAT rules defined using iptables? and (2) Does using LM-DNAT reduce the amount of redundant information with respect to using only iptables? We measured the performance of each strategy in terms of total amount of generated raw data, successfully established TCP connections, IRC traffic being observed, DNS traffic, and Intrusion Detection System alarms being raised. The choice of our network metrics was mostly motivated by what we thought were standard questions asked by security analysts in terms of network traffic generated by malware.

To perform our study, we used the 25118 malware samples corpus that the Ether team used in a recent study [5]. We ran these samples using the Automated Experimentation System [4], a controlled virtual environment allowing the mass analysis of malware within an isolated network. Each sample was executed twice for a duration of one minute in an isolated network whose topology was the one shown in Figure 1. The sole difference between the two executions resided in the configuration of the Router machine: the first time, we used a static configuration of iptables, and the second time, we used LM-DNAT.

We executed the malware samples on VMware ESX virtual machines. It is likely that this has influenced malware behavior as VMware ESX is known to be detectable by malware. However, our

objective was to compare the amount of information that can be gathered using LM-DNAT with what can be gathered using straight DNAT. Consequently, since we used VMware ESX in both cases, the comparison still holds.

The traffic that was recorded during the execution of malware samples contains a significant amount of background noise due to connection attempts to various update sites. Unless stated otherwise, this traffic has been filtered out to compute the results presented in this section. Specifically, we removed all traffic issued to the following sites: windowsupdate.microsoft.com, wustat.windows.com, time.windows.com and mirrors.fedoraproject.org.

For the DNAT experiment, we used a static configuration of iptables, forwarding SMTP (TCP port 25), DNS (UDP port 53) and HTTP (TCP port 80) to the Honeypot machine. For IRC, we installed an IRC server on the Honeypot machine. We forwarded all TCP traffic going to 20 different TCP ports on which we had observed IRC traffic in past experiments to this IRC server. We forwarded all the other traffic to the Target machine: an unpatched Windows XP Service Pack 2. Note that forwarding all remaining traffic to the Target machine (even the traffic going to ports that are closed at the beginning of the experiment) introduces a bias in the analysis. It supposes that malware will open backdoor ports on the Target machine. The alternative would have been to forward all traffic going to ports that are not known to be open to a netcat listening on a specific port of the Honeypot machine. This would have allowed for all TCP connections to be successfully established and thus the recording of the first few bytes of data being sent at the application level. However, we would not have learnt about new ports being open on the Target machine. Among other things, LM-DNAT allows that compromise to be avoided.

For the LM-DNAT experiment, we used LM-DNAT to dynamically configure Netfilter through iptables as malware was being executed. We used the configuration file shown in Figure 6. All DNS traffic is sent to the Honeypot machine (Rule 1). All HTTP traffic was sent to the Honeypot machine (Rule 2). However, HTTP traffic that was sent to an IP address that was previously impersonated by the Target machine was also sent to the Target machine (because of Rule 4). Rule 3 says that all SMTP traffic that was not already forwarded to the Target machine was to be forwarded to the Honeypot machine. Rule 4 says that we wanted to use a one-to-one impersonation strategy for traffic directed to the Target machine. This means that at most one IP address (`matchNum=1`) was to be mapped to the Target machine and that all traffic directed to that IP was to be forwarded to the Target machine, even the connections that fail. The choice of address was determined by the first connection whose destination port was open on the Target machine. The 20 TCP ports on which we had previously observed IRC traffic were forwarded to the IRC server (Rule 5). This allowed for a setup similar to the DNAT experiment and observation of the difference between IRC traffic handled by a real IRC server and IRC traffic handled by netcat. All other TCP or UDP traffic was forwarded to a netcat and all remaining traffic was sent to the Honeypot machine.

```
   #DNS
1  10.92.36.1 proto=udp dport=53 sport=0 dst=0 src=0
   #port 80 traffic to the HTTP Server
2  10.92.36.1 proto=tcp dport=80
   #SMTP
3  10.92.36.1 proto=tcp dport=25 sport=0 src=0
   #try to route back to the target machine
4  10.92.39.233 proto=0 dport=0 sport=0 src=0 matchNum=1
   #irc traffic uses lots of ports
5  10.92.36.1:6667 proto=tcp dport=6564,6565,6656,6659,6660,6601,6665,6666,6667,6668,
   6669,6714,6868,6968,6969,7000,7001,7007,7045,7475 dst=0 src=0 sport=0
   #send any other traffic from 10.92.64.0/19 to netcat running on port 30 (tcp and udp)
6  10.92.36.1:30 proto=tcp src=10.92.64.0/19
7  10.92.36.1:30 proto=udp src=10.92.64.0/19
   #we'll catch everything else here (icmp)
8  10.92.36.1 src=10.92.64.0/19 dst=0 proto=0
```

Figure 6: LM-DNAT Configuration file used for the case study.

## 6.1 Total Size of Generated Raw Data

The DNAT experiment produced 2.34 GB of raw data (traffic traces and various log files), while the LM-DNAT experiment produced 577 MB on the same malware corpus. The reduction of the amount of generated data was due to the high proportion of connections that were handled by netcat rather than a real daemon (because of the `matchNum=1` option in Rule 4). Since at least one Internet host was completely impersonated by the Target host (because we used a one-to-one impersonation strategy, which can only be made using LM-DNAT), we were still able to gather all the relevant information. The connections to the other Internet hosts were mostly redundant connections initiated by worms trying to infect as many hosts as possible. In the rest of this section, we show that while producing less raw data, the LM-DNAT experiment allowed us to extract an even greater amount of information from the malware samples than the DNAT experiment.

## 6.2 TCP Traffic on the LAN Network

Table 1a provides a summary of the TCP traffic observed on the LAN network. Firstly, more than 60% of the samples did not generate any TCP traffic. There are several possible explanations: the sample was broken, one minute was not long enough, it was not intended to generate traffic, etc, and to investigate these reasons is beyond the scope of this study. Nevertheless, the other samples did generate traffic and that is enough for our case study.

The DNAT experiment generated connection attempts to a greater number of ports (621) than

| | DNAT | LM-DNAT |
|---|---|---|
| samples with TCP traffic other than update sites | 9559 | 9522 |
| ports tentatively used | 621 | 380 |
| ports successfully used | 17 | 377 |
| attempted sessions | 36491 | 19160 |
| established sessions | 22224 | 19073 |

(a) TCP activity on the LAN network.

| dest host | dest port | DNAT | LM-DNAT |
|---|---|---|---|
| Honeypot | 25 | 786 | 775 |
| Honeypot | 30 | 0 | 1973 |
| Honeypot | 80 | 6911 | 6876 |
| Honeypot | 6667 | 292 | 289 |
| Target | 21 | 174 | 161 |
| Target | 23 | 1 | 1 |
| Target | 80 | 0 | 24 |
| Target | 135 | 4 | 3 |
| Target | 139 | 20 | 10 |
| Target | 443 | 68 | 71 |
| Target | 445 | 27 | 11 |
| Target | 1801 | 12 | 12 |

(b) TCP activity on the SIM-INTERNET network.

Table 1

| | DNAT port 6667 | DNAT | LM-DNAT |
|---|---|---|---|
| samples doing IRC | 224 | 571 | 1017 |
| IRC servers | 149 | 289 | 400 |
| IRC ports | 1 | 12 | 127 |
| IRC channels | 145 | 166 | 170 |
| samples doing a JOIN | 201 | 265 | 278 |
| IRC nicks | 75 | 157 | 244 |
| samples doing a NICK | 59 | 74 | 159 |
| IRC users | 223 | 6410 | 2457 |
| samples doing a USER | 209 | 473 | 741 |

(a) IRC analysis summary.

| | DNAT | LM-DNAT | Total |
|---|---|---|---|
| Snort | | | |
| signatures | 56 | 57 | 59 |
| samples | 682 | 643 | 721 |
| Emerging T. | | | |
| signatures | 290 | 303 | 311 |
| samples | 4938 | 5610 | 5724 |

(b) IDS alarms summary.

Table 2

the LM-DNAT experiment (380). However, this is meaningless since a small number of samples exhibited a scan behavior that slightly differed between the two experiments. For example, a single sample respectively used 255 and 16 ports in the DNAT and LM-DNAT experiments. Without this sample, the number of ports having been used is respectively reduced to 376 and 365.

We also observe that the number of sessions being successfully established during the DNAT experiment is lower than during the LM-DNAT experiment. However, there are also fewer connection attempts. This is most likely due to the fact that our implementation introduces a delay in

the establishment of a connection and that consequently, fewer connection attempts can be made within one minute.

### 6.3 TCP Traffic on the SIM-INTERNET Network

Table 1b shows the number of samples successfully establishing a TCP connection for each dataset, per redirected destination host and port, on the SIM-INTERNET Network. The main observation to be made is that, over both experiments, no TCP connection was successfully established on a port that was not already open on the Target machine. However, this does not mean that no malware sample attempted to open a backdoor port on the Target machine.

For example, the corpus we used contained 5 copies of diverse variants of the Allaple worm, which uses an exploit against vulnerability MS06-040[6] to open TCP port 9988 on the target machine. The exploit simply did not work against the Target machine. Upon analysis of the LM-DNAT traffic traces, we learned that there is enough intelligence in that worm to not attempt a connection to port 9988 if the NetBIOS session did not complete normally, but not enough to determine if the exploit was successful without attempting a connection. Indeed, as specified by Rule 4 of the configuration shown in Figure 6, the Target machine was only allowed to impersonate one Internet host, all other NetBIOS connections were handled by the netcat server of the Honeypot machine. A connection attempt to port 9988 was only made to the host that was impersonated by the Target machine. Since Rule 4 also specifies that all TCP connections made to this impersonated host should be forwarded to the Target machine (because of the `proto=0`, `dport=0`, `sport=0`, and `src=0` options), the connection was simply reset and no payload could be observed. Had we not used the impersonation options, the connection to port 9988 would have been handled by the netcat server of the Honeypot host.

We observe that although Rule 2 is before Rule 4, some HTTP traffic (port 80) has been forwarded to the Target rather than the Honeypot. We identified two reasons for this. First, the timeout we had set for the reception of a SYN-ACK was 0.2 seconds, after which the port was qualified as closed by LM-DNAT. It turned out that this timeout was not always long enough, an observation that we have to consider in the future. Second, and that is a direct consequence of the impersonation features of LM-DNAT, there are some malware samples that connect to a NetBIOS port (TCP 139 and 445) of seemingly randomly selected hosts and then perform some HTTP requests on the same hosts. That was the case, for example, of a specific sample identified as Unixon by most anti-virus products. The IDS alarms that were triggered, however, were too generic for us to provide an explanation for that behavior. Also, at the time we are writing this paper, there seem to be very few online analysis reports about Unixon.

The LM-DNAT experiment generated significantly less NetBIOS traffic (TCP ports 139 and

[6] http://www.microsoft.com/technet/security/bulletin/ms06-040.mspx

445) than the DNAT experiment, a fact that does not seem to be incidental since we re-performed the LM-DNAT experiment for the 27 specific samples having generated traffic during the DNAT experiment three times with similar results. Although we are able to explain a few cases, at the time we are writing this paper, we still have no general explanation that accounts for all cases. In particular, there were ten samples for which a NetBIOS connection was not even attempted.

### 6.4 DNS Requests

The DNAT dataset (the data generated during the DNAT experiment) contains 6655 different queries done by 8975 samples, whereas the DNAT-WT dataset contains 5933 done by 8935 samples. Globally, a total number of 9040 samples issued 7921 DNS queries in either of the datasets. This means that, roughly, the same samples generated a significant number of different queries in both datasets. In particular, 1988 queries were observed only in DNAT and 1266 queries were observed only in DNAT-WT. Most of these differences seem to be explainable by the presence of fast-flux botnets in the corpus. For example, respectively 1119 (1113) of the unique 1988 (1266) queries in the DNAT (DNAT-WT) dataset are of the form XXXXX.nb.host-domain-lookup.com. The higher number of unique queries in the DNAT dataset is mainly attributable to two specific samples, which respectively generated 368 and 568 queries in the DNAT dataset and only generated 73 and 21 queries in the DNAT-WT dataset. Finally, 105 samples only generated queries in DNAT while 65 only generated queries in DNAT-WT. Consequently, the DNAT-WT strategy does not seem to present any notable advantage over the DNAT strategy with respect to DNS traffic.

### 6.5 IRC Traffic

The Internet Relay Chat (IRC) protocol has been used by botnets for several years to allow communication between the bot herder and the infected hosts constituting the botnet. Often, bot herders setup an IRC server that runs on an arbitrary port, 6667 being one of the most commonly used. Our experiments showed that many botnets do not wait for any specific authentication from the server before to send login information such as user names and nick names. In fact, they do not seem to expect anything at all from the server once the TCP handshake is established. Therefore, we could deduce IRC information not only from the traffic that we forwarded to the IRC server that was running on the Honeypot machine (recall that we forwarded 20 different TCP ports to port 6667 of the Honeypot machine), but also from some traffic that was forwarded to the netcat port of the Honeypot.

For our case study, in order to determine which TCP sessions contained IRC traffic, we used the following criteria:

1. The payload of the session contains one of the words: JOIN, USER or NICK.

2. The tshark[7] command line utility properly decodes the session as IRC traffic (i.e., at least one of the three above operations is properly decoded).

We most likely generated a few false positives and false negatives, but accurate IRC traffic detection is beyond the scope of this study.

Table 2a provides a summary of the IRC information that was deduced from the analysis of the traffic traces generated during each experiment. The second and third columns provide statistics derived from the DNAT and LM-DNAT experiment, while the first column provide statistics derived from the DNAT experiment when looking only at port 6667. In other words, it is an estimate of the statistics we had obtained if, rather than forwarding 20 different ports to the IRC server, we only forwarded port 6667.

Generally, we can conclude by looking at these three columns that having some a priori knowledge of which ports might be used for IRC is useful, but that making sure that all TCP connections are being handled allows even more information to be gathered. We observe that a greater number of unique IRC users were identified during the DNAT experiment (6410) than during the LM-DNAT experiment (2457). However, four samples generated IRC traffic containing 5339 and 1725 different users respectively during the DNAT and the LM-DNAT experiment. Without those four samples, 472 different users were observed during the DNAT experiment, while 733 different users were observed during the LM-DNAT experiment, which means about one user per sample doing a user operation. Several users were observed during the execution of more than one sample.

### 6.6 Intrusion Detection System Alarms

We analyzed the generated traffic traces using the Snort[8] signatures released on February 17 2010 and the Emerging Threats[9] signatures released on April 17 2010. We used the Emerging Threat signatures because they are specially written for malware detection. Table 2b shows a summary of the analysis results. As expected, the number of signatures and the number of samples having generated an alarm was higher during the LM-DNAT experiment. It is more interesting to look at the number of *exclusive* signatures and samples, i.e. those that generated alarms in only one of the two experiments. In Table 3 (page 18), we provide a list of the signatures that were triggered on a number of exclusive samples greater than 20 for at least one of the two experiments. Table 3 also gives the total number of samples having triggered the signature, i.e. the number of exclusive samples for each signature plus the number of samples having triggered the signatures during both experiments.

[7] `http://www.wireshark.org/`
[8] `http://www.snort.org/`
[9] `http://www.emergingthreats.net/`

| Sig. ID | Message | DNAT only | LM-DNAT only | Total |
|---|---|---|---|---|
| Snort | | | | |
| 491 | INFO FTP Bad login | 42 | 2 | 158 |
| 1394 | SHELLCODE x86 inc ecx NOOP | 26 | 27 | 77 |
| Emerging Threats IRC | | | | |
| 2000345 | Nick change on non-std port | 4 | 282 | 344 |
| 2000347 | Private message on non-std port | 0 | 28 | 59 |
| Emerging Threats TROJAN | | | | |
| 2008021 | Turkojan C&C Initial Checkin (ams) | 0 | 113 | 113 |
| 2008026 | Turkojan C&C Keepalive (BAGLANTI) | 0 | 111 | 111 |
| 2002974 | Hupigon Possible Control Connection | 0 | 261 | 261 |
| 2002975 | Hupigon INFECTION - Reporting Host Type | 0 | 261 | 261 |
| 2008042 | Hupigon CnC Data Post (variant abb) | 0 | 263 | 263 |
| 2003555 | Bandook v1.35 Initial Connection and Report | 0 | 24 | 24 |
| 2007823 | Banker.OT Checkin | 11 | 27 | 203 |
| Emerging Threats Others | | | | |
| 2009885 | SCAN Unusually Fast 404 Error Messages | 34 | 44 | 296 |
| 2003303 | POLICY FTP Login Attempt | 42 | 2 | 164 |

Table 3: Number of exclusive samples per signature, per experiment, and total number of samples per signature, across both experiments (limited to signatures with number of exclusive samples in one of the two experiments being greater than 20).

### 6.6.1 Snort Alarms

The first Snort signature, related to FTP, was triggered on more samples during the DNAT experiment than during the LM-DNAT experiment, which was to be expected given that, as shown in Table 1b, there were more successful FTP connections during the DNAT experiment than during the LM-DNAT experiment. The second Snort signature, related to shellcode, has mostly been triggered over several different ports and there does not seem to be any general explanation for the 26 and 27 samples having exclusively triggered that signature during the DNAT and LM-DNAT experiments respectively. Generally, the statistics we obtained using Snort signatures are insufficient to conclude that the LM-DNAT strategy generated traffic that contained significantly more or less information. However, there were also very few samples that generated Snort alarms. For that reason, we also analyzed the traffic traces using the Emerging Threats signatures.

### 6.6.2 Emerging Threats Alarms

The first two Emerging Threats signatures are related to IRC, which is not surprising (recall Section 6.5). Only four samples have triggered signature 2000345 exclusively during the DNAT experiment against 282 during the LM-DNAT experiment. All the samples having triggered signature 2000347 during the DNAT experiment also triggered it during the LM-DNAT experiment. Therefore, LM-DNAT outperforms DNAT with respect to these two signatures.

The next two signatures are related to the Turkojan trojan. Turkojan, which was first seen in 2003[10], is a backdoor that reportedly initiates a connection to TCP port 15963[11]. While this was the case for 101 samples, 12 other samples initiated connections to different ports. We also observed that the 113 samples identified as generating Turkojan traffic initiated connections to 93 different DNS names or hardcoded IP addresses, all of which appeared both during the LM-DNAT and DNAT experiments, which makes it unlikely that the samples we tested use fast-flux techniques or that they are polymorphic copies of the exact same threat (i.e., same issuing malicious individuals, same release date, etc). The fact that 12 samples used a port other than 15963 (which is the default), could lead to the belief that the issuing individuals are more experienced or have greater will to be stealthy than the issuers of the 101 other samples. Even though this is all speculation, we believe that this is potentially useful information that we could not deduce from the results of the DNAT experiment. The DNAT experiment only provided us with the destination ports, but since the TCP handshake did not complete, it did not allow us to identify the generated traffic as being Turkojan traffic.

The next three signatures are related to Hupigon, a trojan that was first seen in 2006[12]. Over all sample executions, seven different Hupigon related Emerging Threats signatures were identified across 315 samples. The four signatures not being displayed in Table 3 matched the same number of samples in the DNAT and the LM-DNAT experiments (for a total of 50 alerts). A total of 267 samples were responsible for having triggered the three Hupigon signatures shown in Table 3. None of those 267 samples triggered any of the four signatures not being displayed in Table 3.

During the LM-DNAT experiment, 261 out of these 267 samples generated traffic to port 8000 of 206 different DNS names or hardcoded IP addresses. A particular name was used by 14 samples, which could lead us to believe that these particular samples are polymorphic copies of the same threat, and that they may belong to a common, more recent or at least more sophisticated threat. These facts could have been observed during the DNAT experiment if we had forwarded port 8000 to port 80 or a netcat port of the Honeypot, but again, the whole point of the LM-DNAT experiment is that we cannot think about all these configuration requirements ahead of time. For example,

[10] http://www.symantec.com/security_response/writeup.jsp?docid=2003-032816-3726-99

[11] http://www.threatexpert.com/report.aspx?md5=e3bce7ee0e5fc54767c7b20400c273a3

[12] http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Win32/Hupigon

the six samples not generating traffic on port 8000 generated traffic on port 2007. The captured traffic only contained one session, which triggered signature 2008042. Possibly, these six samples are variants that use a different port for command and control. There is no way we could have known in advance that traffic would be generated on port 2007.

Bandook is a remote administration tool that was first observed in 2005[13]. All the samples that triggered signature 2003555 generated traffic on port 1167, which was redirected to the Target machine in the DNAT experiment. Since that port was closed, traffic generated on that port could not be captured, which explains why alarms were only generated while analyzing traffic traces generated during the LM-DNAT experiment.

Banker.OT is a data-stealing Trojan[14] that was first observed in 2008[15]. The presence of exclusive samples in each dataset seems to be mostly due to the fact that all samples did not generate traffic during both experiments. Having run each sample several times and/or for a longer period of time might have produced more uniform results across both experiments. The same comment applies to signature 2009885. Finally, the same comment that applied to Snort signature 491 applies to signature 2003303.

### 6.6.3 Conclusion

Globally, although the DNAT strategy allowed the identification of behaviors that the LM-DNAT strategy could not, analyzing the LM-DNAT traffic traces with the Emerging Threats signatures provided more information about more samples. Consequently, we believe that the LM-DNAT strategy allows the generation of traffic traces that give a better description of malware behavior.

## 7 Conclusion

In this paper, we introduced LM-DNAT, a tool that helps studying network-related aspects of malware behavior using an isolated honeynet. Our case study showed that using LM-DNAT generates more information about malware than using static DNAT rules. Moreover, by reducing the number of redundant connections, the total amount of generated raw data can be up to four times smaller, thus reducing the amount of data that has to be analyzed in a post-processing phase.

The way we used LM-DNAT in our case study was meant to be both simple and generic in order to facilitate interpretation of the results. In particular, we made little use of low or medium-interaction honeypot technologies. Future work will include integration of more sophisticated honeypot technologies and more diverse high-interaction honeypots. Another aspect that we left aside,

[13] http://en.wikipedia.org/wiki/Bandook

[14] http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=TrojanSpy:Win32/Banker.OT

[15] http://doc.emergingthreats.net/2007823

but plan to investigate further, is traffic forwarding based on application layer payload.

Although our case study showed that LM-DNAT is useful for malware analysis, to determine the best way to use it remains a challenge. We see three inter-related aspects of the problem: (1) find the best forwarding strategy (i.e., the best configuration file), (2) find the best hosts to place in the honeynet, and (3) find the best topology to be used. Finally, all of our study was implicitly performed with the client-server model in mind. It would be interesting to see how the concepts we developed apply to peer-to-peer protocols. Given that those protocols are now used by botnets, we believe that knowing how to study them in a contained environment would be beneficial for many cyber security applications.

## References


[1] Paul Baecher, Markus Koetter, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In *In Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection (RAID*, pages 165–184. Springer, 2006.

[2] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson. The internet motion sensor: A distributed blackhole monitoring system. In *In Proceedings of Network and Distributed System Security Symposium (NDSS 05)*, pages 167–179, 2005.

[3] Ulrich Bayer, Andreas Moser, Christopher Krügel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.

[4] Mathieu Couture and Frédéric Massicotte. Studying malware in an isolated network sandbox. Technical note, Communications Research Centre Canada, September 2009.

[5] Artem Dinaburg, Paul Royal, Monirul I. Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 51–62. ACM, 2008.

[6] Corrado Leita. *SGNET : automated protocol learning for the observation of malicious threats.* PhD thesis, EURECOM, December 2008.

[7] Norman Solutions. Norman sandbox whitepaper. http://download.norman.no/ whitepapers/whitepaper_Norman_SandBox.pdf, 2003.

[8] Niels Provos. A virtual honeypot framework. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.

[9] Lance Spitzner. *Honeypots: Tracking Hackers.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[10] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5(2):32–39, 2007.