國立交通大學

資訊科學與工程研究所

碩 士 論 文

動態惡意程式分析環境中安全及透明的網路流量之重播、重導及轉送

Secure and Transparent Network Traffic Replay, Redirect and Relay in a Dynamic Malware Analysis Environment

研 究 生：施宗筆

指導教授：林盈達 教授

中 華 民 國 一 百 年 六 月

動態惡意程式分析環境中安全及透明的網路流量之重播、重導及轉送

Secure and Transparent Network Traffic Replay, Redirect and Relay in a Dynamic Malware Analysis Environment

研 究 生：施宗筆　　Student：Tzung-Bi Shih

指導教授：林盈達　　Advisor：Ying-Dar Lin

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2011

Hsinchu, Taiwan

中華民國一百年六月

動態惡意程式分析環境中安全及透明的網路流量之重播、重導及轉送

學生：施宗筆　　指導教授：林盈達

國立交通大學資訊科學與工程研究所碩士班

摘　　要

典型的動態分析會搭配封閉的網路環境以避免惡意程式在分析過程攻擊到網際網路上的機器。然而，現今的惡意程式大多需要連線到網際網路以運作。由於連線到網際網路的流量被阻擋，搭配封閉網路的分析環境用途遭受限制。我們提出一個系統，允許動態惡意程式分析環境擁有看似無限制的網際網路存取權，並且透明地將惡意流量導向系統內的誘捕器，同時允許無害的控制流量存取網際網路。在2000 多隻可疑的惡意程式中，我們首先選擇被四套防毒軟體標記的124 隻惡意程式。接著，我們排除那些沒有網路行為或者無法成功連線到它們設計好的機器的惡意程式。最後，我們總共有 12 隻惡意程式樣本。實驗結果顯示，我們的系統可以看到的網路行為平均是封閉網路的 3.35 倍，在分析發送垃圾信件的惡意程式的情況下，我們甚至更勝於開放網路環境。同時，網際網路的安全性也會被改善。

關鍵字： 動態分析、封閉網路、開放網路、導向

Secure and Transparent Network Traffic Replay, Redirect and Relay in a Dynamic Malware Analysis Environment

Student: Tzung-Bi Shih Advisor: Dr. Ying-Dar Lin

Institutes of Computer Science and Engineering
National Chiao Tung University

Abstract

Dynamic analysis is typically performed in a closed network environment to prevent malware under analysis from attacking machines on the Internet. However, many of today's malware require Internet connections to operate. A closed network analysis environment will be of limited use for such malware as Internet bound connections are blocked. We propose a system to allow malware in a dynamic analysis environment to have seemingly unrestricted Internet access. Our system transparently retargets malicious network connections to compatible decoys within our system while allowing Internet access for harmless control traffic in unknown protocols. Among more than 2000 suspicious malwares, we first select 124 malwares that are flagged by all anti-virus scanners from 4 different vendors. Then, we exclude those malwares that exhibit no network activities or cannot connect to their designed machines on the Internet. Finally, we have 12 malware samples. The evaluation result shows that our system can allow the malware to exhibit more network activities than a closed network environment (3.35 times more on average) and even outperform a baseline open network environment for the case of spammer-type malwares. In the meantime, Internet security is significantly improved.

Keywords: Dynamic Analysis, Closed Network, Open Network, Retarget

Contents

Chapter 1 Introduction 1

Chapter 2 Background 3

2.1 Network Traffic of Botnet 3

2.2 Related Works 4

Chapter 3 Network Traffic Replay, Redirect, and Relay in Dynamic Malware Analysis Environment 7

3.1 Approach Overview 7

3.2 Design of Dispatcher 9

3.3 Maintaining Protocol States 13

3.4 Example of Traffic Replay, Redirect, and Relay 16

Chapter 4 Implementation 18

Chapter 5 Experiment Studies 22

5.1 Sample Selection and Experiment Environment 22

5.2 Effectiveness of Transparent Network Environment 24

5.3 Effectiveness of Secure Network Environment 28

5.4 Case Study: A normal case 30

5.5 Case Study: An unexpected case 31

Chapter 6 Conclusions and Future Works 34

References 36

List of Figures

Figure 1: Botnet operations....3
Figure 2: An overview of our approach....8
Figure 3: An overview of the dispatcher....10
Figure 4: Pseudo code of selected stateful modules....15
Figure 5: An example of traffic replay, redirect, and relay....16
Figure 6: System Implementation....18
Figure 7: SMB packet format....20
Figure 8: A SMB logon failure process....21
Figure 9: Basic experiment environment....24
Figure 10: Additional experiment environment....30
Figure 11: DNS queries for spam e-mails....30
Figure 12: A SMTP session for a spam e-mail....31
Figure 13: Snort issues an alert for e-mail content....31
Figure 14: Partial content of the spam e-mail....31
Figure 15: An unexpected case....32
Figure 16: Transfer the malware binary via SMB....33
Figure 17: Using ‘at’ scheduler....33

List of Tables

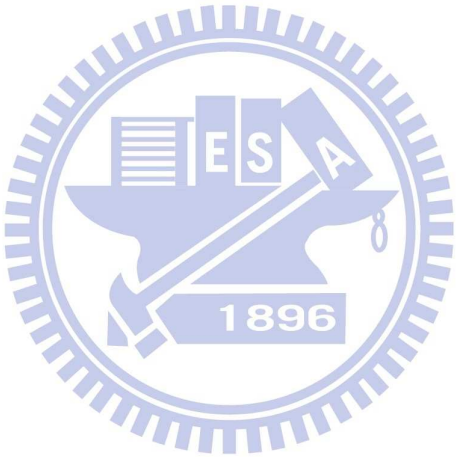
Table 1: Selected samples 23

Table 2: Network activities by malware without C&C 25

Table 3: Number of packets by m10.exe, m11.exe, and m12.exe 25

Table 4: Network activities by malware with C&C 27

Table 5: Number of spam e-mails 28



Chapter 1 Introduction

Malware has been a major threat to computer security for years [1-3]. Defense against malware typically involves a three-stage process: the analysis of unknown malware, producing signature, and the detection of known malware [4]. The analysis of unknown malware is usually the most difficult part in the process of malware defense. In fact, both the production of signature and the detection of known malware depend on a successful analysis first. An obvious reason behind the difficulty of malware analysis is that malware has to be elusive by nature. A malware can be obfuscated [5], can be designed to subvert an analysis tool [6-9], can be purposely made to stay dormant until the "D-Day" has come [4], and etc. All the above can make the analysis of malware a really challenging task.

Environment of Dynamic Malware Analysis

Existing malware analysis procedure often involves a key technique called dynamic analysis [10-12], which basically is to execute an unknown malware in a closed environment (i.e. a sandbox) and observe its behavior during the runtime [6]. Ideally, the unknown malware should exhibit all its malicious behavior in the closed analysis environment exactly as it does in the real world. Unfortunately, this is not always the case. For example, the malware may detect the sandbox environment and refrain from showing its true behavior [6-9]. It may be designed to act only at a specific time (i.e. a logic bomb) [4]. When it comes to bot [13-14], a type of malware that takes commands from a controller on the Internet and usually attacks specific targets again on the Internet, the closed analysis environment often fails to capture the bot's full behavior as network traffic with the outside world (including the bot's communication with the controller) is totally blocked.

Luckily, some of the issues on the dynamic malware analysis environment are not very difficult to solve. For instance, to deal with a malware that is potentially sandbox-aware, one can try a different sandbox implementation that does not possess the signature the malware is looking for [15]. For logic bombs, one can tweak the machine clock to any dates that might be of interest to the malware [16]. A more difficult issue, as we can see, is when the malware's operation depends on some communication with a controller on the Internet or when the malware is designed to attack a specific target on the Internet. With a completely closed network environment, the malware will simply quit working, and little of its behavior can be observed. On the other hand, if the analysis environment is openly connected to the Internet, there is a great concern that the malware may actually cause havoc to innocent victims on the Internet.

To address the dilemma between an open and a closed network environments for dynamic malware analysis, we propose a framework for adaptive network traffic control. The framework allows the malware to communicate freely with its controller on the Internet, and at the same time, the framework can transparently redirect outgoing malicious traffic to decoys inside our framework. By doing so, we can achieve a good balance between open network and closed network: the malware *will not quit prematurely* due to network inaccessibility, and the *security of the Internet is ensured* as the malicious traffic never leaves the analysis environment in reality.

The rest of the work is organized as follows. Chapter 2 talks about how modern malware leverages on the Internet for propagation and attack, and a survey of related works is also provided. Chapter 3 gives the problem statement and the details on the proposed framework. Chapter 4 describes our implementation details. Chapter 5 presents the experiment results and discussions. Finally, Chapter 6 concludes this work with some words on potential future works.

Chapter 2 Background

In the work, we focus on achieving a good balance between open network (the most transparent) and closed network (the most secure) for a dynamic malware analysis environment. Our system can be generally applied to any kind of malware. However, the benefit is most eminent when it is used on malware that exhibits some sort of network activities (e.g. propagation over the Internet). Among all the different types of malware, bot, a type of malware that takes commands from a controller on the Internet to achieve specific attack goals (e.g. denial-of-service attack to a specific target on the Internet) is the one that involves most network activities. In this chapter, we will give a brief overview of the network activities involved in a bot's operation and mention how existing dynamic analysis systems deal with malware's network activities and their shortcomings.

2.1 Network Traffic of Botnet

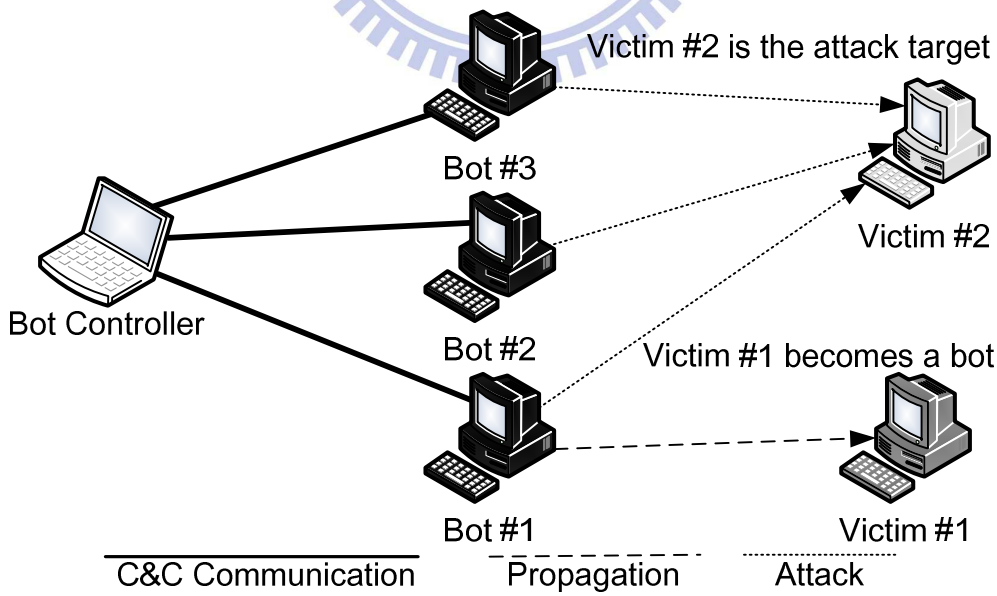


Figure 1: Botnet operations

Typically, bots are designed to function in a collective manner as shown in

Figure 1, where a controller run by a malicious guy commands a herd of bots to carry out attack on a victim (i.e. Victim #2 in Figure 1). The whole system is often referred to as a '*botnet*' [13-14], meaning a network of bots. A bot may attempt to proliferate itself onto other machines over the Internet (Bot #1→Victim #1 in Figure 1). This increases the number of bots in a botnet and can make a subsequent attack more powerful. Note that those machines infected by bots can be the target of attack as well. For instance, part of the purpose of a botnet may be to steal private personal information, such as passwords and credit card numbers, from the infected machines.

As shown in Figure 1, a botnet involves a lot of network activities. The network activities of a botnet can be roughly categorized into three different types: *propagation*, *C&C communication*, and *attack* [13-14, 17]. Propagation is used for increasing the population of bots. C&C communication is for the controller to command the bots and for the bots to send information (e.g. credit card numbers) back to the controller. Finally, attack corresponds to those network traffic generated by the bots for the purpose of attacking the target victim.

2.2 Related Works

Since a lot of malware, especially the bot, involves network activities in their operations. In dynamic analysis, one will have to be able to provide a compatible network environment in order to get the malware run properly. On the other hand, the environment has to be secure so that the malware will not actually cause damages to the Internet. From here, there are two possible approaches for setting up the network environment for dynamic analysis. The first is to allow the malware to have full Internet access (i.e. open network) [11, 18]. This is obviously very dangerous. The second approach (the mainstream approach) is to use a closed network environment, in which the dynamic analysis environment is completely disconnected from the

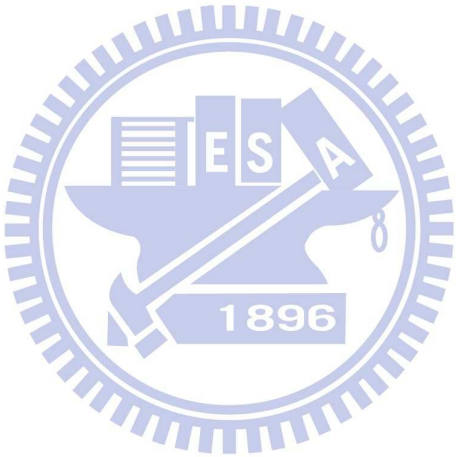
Internet [12, 18-19]. While this approach is very secure, it is not very effective for capturing the full runtime behavior of a malware.

The work that is most closely related to ours is the Honeypot project [20-21]. Although their goals are fundamentally different from ours, their system also involves traffic redirection. However, they did not address the state synchronization issue between the would-be victim and the honeypot that receives the redirected traffic. As a result, if the attack traffic is carried by a stateful connection (e.g. with TCP, or some upper layer stateful protocols), the abrupt redirection will cause the connection to be broken, and the full behavior of the malware is still unknown.

To prevent a malware that had been implanted in a honeynet from leaking out attack traffic to the Internet, the Honeynet project proposed the payload rewriting technique that can nullify any attack effect in Internet-bound attack traffic [22]. Again, the payload rewriting will cause the attack to fail and the full behavior of the malware is left unknown.

The work [23-24] by E. Alata et al. assumes C&C communication can be recognized and transparently relayed to the Internet. They also relay DNS traffic to the Internet and well-known ports of vulnerable services (e.g. TCP port 139 and 445) traffic to honeypots directly. Except them, all the other traffic from the analysis environment is filtered. Because they relay traffic for some specific ports directly, they have no protocol state synchronization issues between the would-be victim and the honeypot that receives the redirected traffic. Their design relies on the fact that bot C&C communication has been based on the well-known IRC protocol for a long time. However, nowadays, we have seen bot C&C communication running customized protocols. Some of them even involve encryption that is almost unbreakable [25-27]. In our system, we follow a different design, in which no assumption is made about C&C communication being recognizable.

The work [28] by G. Berger-Sabbatel et al. also assumes C&C communication can be recognized. They monitor plain text C&C communication for a few weeks, and simulate the C&C server. For cipher text C&C communication, they relay to real C&C servers on the Internet. They did not address the state synchronization issues when redirecting packets in the middle of a connection. However, we can handle the synchronization issues even if the redirection occurs in the middle of a connection.



Chapter 3 Network Traffic Replay, Redirect, and Relay in Dynamic Malware Analysis Environment

Modern malware are often tightly coupled with network. They rely on the network to propagate to remote machines. Some of them are even designed to convey attacks over the network. Besides, we have also seen the botnet-type malware to leverage on the network to carry out an organized attack. For dynamic analysis of modern malware to be effective, it is necessary to ensure that the malware has a transparent view to the whole network, including the Internet outside the analysis environment. Otherwise, the malware may not exhibit all its behavior fully if it cannot receive commands from a remote controller or if it cannot reach the IP addresses of victim machines. On the other hand, it is also important to make sure that the malware cannot cause damage to innocent machines and jeopardize the security of the Internet.

In this chapter, we will present a system that is designed to achieve both network transparency and Internet security for dynamic malware analysis. We give our problem statement to clarify the scope of the work and later the details on the proposed system in Section 3.1~3.4.

Problem Statement

Given a set of executable malware and a dynamic malware analysis environment, restructure the environment to improve the network transparency to the malware and the network security to the Internet.

3.1 Approach Overview

As mentioned earlier, the network traffic of a malware may consist of propagation, C&C communication, and attack. On one hand, we would like the traffic

to flow freely, at least from the malware's perspective, so that the most behavior of the malware can be observed during dynamic analysis. On the other hand, we also want to make sure the whole environment is secure so that the malware cannot cause damage to machines on the Internet.

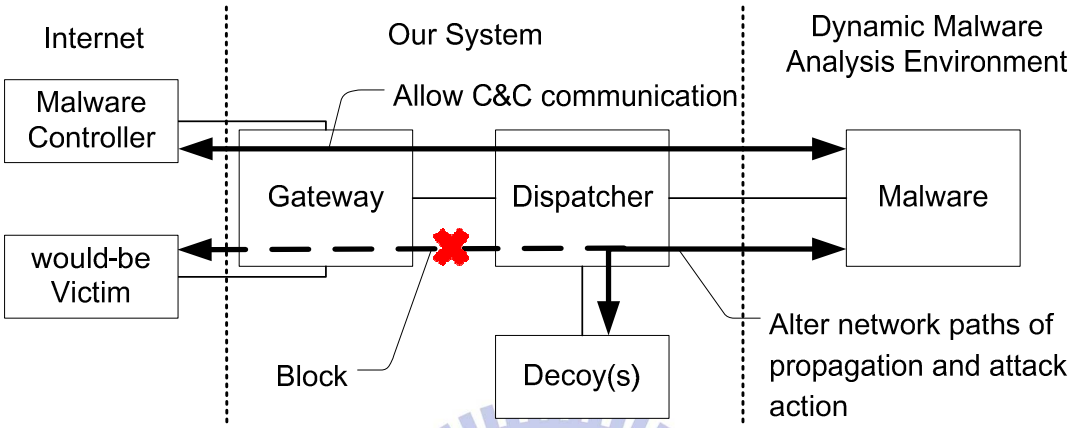


Figure 2: An overview of our approach

An overview of our approach is shown in Figure 2. On the left-hand side of Figure 2 is the Internet, where the malware controllers and would-be victim machines locate. On the right-hand side of Figure 2 is the dynamic malware analysis environment, where the malware runs in a sandbox that is capable of extracting runtime properties such as system calls invoked by the malware. Sitting in the middle is our system, which intercept (and retarget) the network traffic between the analysis environment and the Internet.

In our system, propagation and attack traffic are transparently retargeted to decoys inside our system. Both of the traffic never reaches the Internet. The traffic retargeting is a three phase process, which we will detail in Section 3.2. One important aspect of the traffic retargeting is that it has to be transparent to the malware. The malware can hardly notice the traffic retargeting, so that its full network behavior can be observed. A key characteristic of propagation and attack traffic is that they tend to follow well-known protocols and target well-known services. This

maximizes the malware's propagation and attack capability, a sensible design choice for malware. As a result, one can reasonably assume that both the propagation and attack traffic can be reliably identified so that they can be transparently retargeted. The decoys are machines running well-known vulnerable services that will be subject to attacks by the malware in the analysis environment.

Some malware (notably the bot) require C&C communication with a remote controller on the Internet to operate. As each C&C controller is specifically designed to work with a corresponding malware, it is impractical to assume that one can always find the proper decoy to emulate a controller. Moreover, the C&C communication may rely on non-standard protocols [25-27], which can be difficult to detect for a subsequent retargeting. Due to the above reasons, in our system, we implicitly allow C&C traffic to flow onto the Internet. While this might first sound like a bad idea, we notice that this does constitute much security threat in practice. This is because, C&C communication, by definition, does not carry attack effect in itself. At best, it can be used to extract sensitive information from an infected machine. Fortunately, this is not an issue in our case, since the dynamic malware analysis environment would not contain such sensitive information.

3.2 Design of Dispatcher

Figure 3 shows the design of the dispatcher. The dispatcher runs on a machine with three network interfaces: NIC #1 connects to the dynamic malware analysis environment. NIC #2 connects to the Internet. NIC #3 connects to the decoys. When a packet from the analysis environment reaches NIC #1, it is forwarded to both the coordinator and the IDS (copy to the IDS, and intercept to the coordinator by packet filter). If the IP addresses and port numbers of the packet are blacklisted or if the packet triggers an IDS alert, the coordinator will consider the corresponding

connection as malicious and initiate the traffic retargeting process (detailed below). Otherwise, the packets will be forwarded to Internet via NIC #2. For the other direction, when a packet from the Internet reaches NIC #2, it is forwarded to the IDS and NIC #1 (i.e. the analysis environment). We also forward packets from the Internet to the IDS because the IDS may inspect packets in either flow direction in each connection.

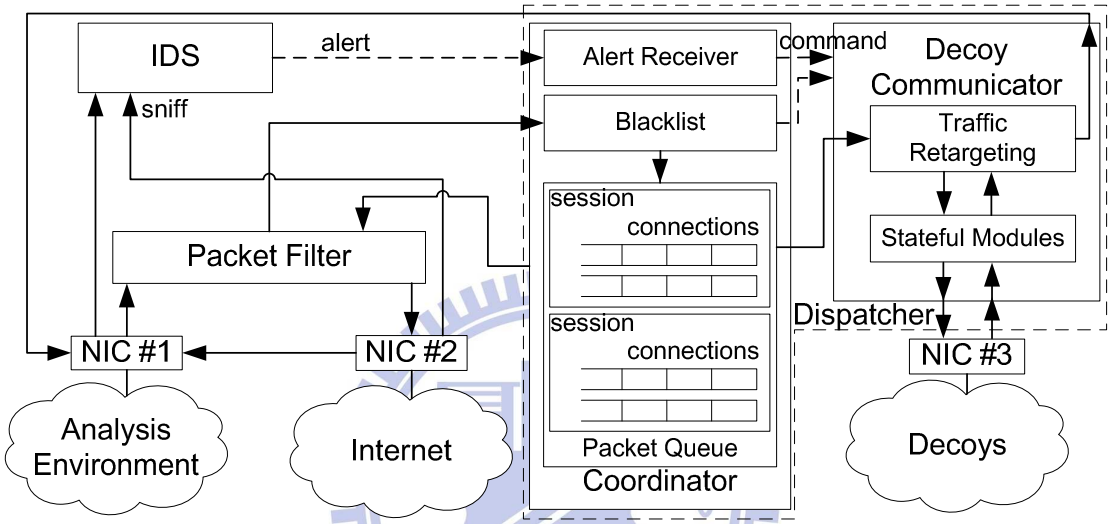


Figure 3: An overview of the dispatcher

We have a packet queue design in the coordinator. The queue stores packets received from the analysis environment. The stored packets may be used in the traffic retargeting process. The design of the queue is a set of ordered lists. Each list (refer as a connection in Figure 3) consists of packets is ordered by the time of the packet arrived the dispatcher. Packets in the same ordered list have identical source and destination port numbers. We group the ordered lists by their source and destination IP addresses (refer as a *session* later and in Figure 3). The group design is archiving by hashing so that we can retrieve a group in the most efficient way. In the traffic retargeting process, the packets in the queue may be removed after used by the decoy communicator (detailed below). In fact, we use the packet queue to be a buffer between the coordinator and the decoy communicator (producer and consumer

model).

Traffic Retargeting

During traffic retargeting, all the connections bearing the same source and destination IP addresses are considered as a group, and all the connections in the group will be retargeted to the same destination decoy. We name the group of the IP connections as a *session*. Traffic retargeting can be initiated in two situations which are called retargeting decisions. The first situation is when the IP addresses or port numbers of a session are on the blacklist of known malicious connections. All the packets in the session are relayed to the decoy directly (by changing the MAC and IP addresses in the packets). This is similar to the approach used in [23-24].

The other situation of traffic retargeting is when one of the connections in a session is flagged by the IDS as malicious. Unlike the first situation, the retargeting may occur in the middle of a session, when some connections are already closed, and some connections are still ongoing. Both types of connections cannot be simply relayed as this would result in state inconsistencies between a decoy and the running malware. In our system, we have to use a three-phase retargeting process that involves traffic replay, redirect, and relay to properly handle connections at different phases in a session.

The first phase is the replay phase. At the time when retargeting decision occurs, some of the malware's network connections may have been closed. Packets in closed connections may need to be replayed to the decoy, so that the decoy will have its states in consistency with the running malware. We store outgoing packets from the analysis environment in the packet queue (Figure 3). At the time when the retargeting decision occurs, the coordinator will instruct decoy communicator to replay packets corresponding to those closed connections from the packet queue to the decoy through

NIC #3. During the replay, the decoy may generate response packets such as TCP ack. All response packets from the decoy are filtered (they are not forwarded to the malware) since from the malware's perspective of view, these connections have been closed. The replayed packets have the effect of reestablishing the corresponding states in the decoy so that the decoy can mimic the would-be victim and have states that are in consistency with the running malware.

The second phase is the redirect phase. The phase deals with those ongoing connections in a session when retargeting decision occurs. Packets that have been transmitted in each ongoing connection are processed in the same way as in the replay phase. Decoy communicator takes those packets from the packet queue, adapts the packets to the retargeted traffic path with the stateful modules (Section 3.3) and sends the packets to decoy through NIC #3. If there are returning packets from the decoy during the replay, those returning packets are ignored. For subsequent packet transmissions, decoy communicator will again use the stateful modules to adapt the packets to the retargeted traffic path, and then forward the packets to the decoy. The returning packets from the decoy for subsequent packets are forward back to the analysis environment through NIC #1, because from the malware's point of view these connections are still ongoing (the malware receives these packets just like from the would-be victim).

The third phase is the relay phase. The phase changes the traffic path of future connections (connections that occur after retargeting decision) in a session to the decoy. The retargeted traffic in the relay phase only need to have the destination MAC and IP addresses in the packet headers replaced with the ones used by the decoys. These are also handled by the stateful modules.

Through traffic retargeting, our system transparently replay, redirect, and relay the connections in a session that carry malicious traffic. From the malware's point of

view, the connections are still with some victim machines on the Internet, though in reality the underlying traffic has been retargeted to decoys in the secure environment. During the traffic retargeting, there may be states in the upper layer protocols, which require additional processing. This is also taken care of by the stateful modules in our system, which will be detailed in Section 3.3.

In addition to using IDS to flag malicious connections, we also have a blacklist of potentially malicious connections that may not be captured by the IDS. For instance, a stock IDS typically do not have a corresponding signature for detecting e-mail spam traffic. In our system, the blacklist contains a rule to match outbound SMTP traffic, so that the connection will be relayed to the decoy right from the very beginning. We also have rules in the blacklist that matches connections based on IP addresses and port numbers. For instance, if we see an outbound connection to 140.113.40.35 (homepage of NCTU), the connection is much more likely to be part of the propagation or attack traffic instead of part of the C&C traffic.

3.3 Maintaining Protocol States

The fundamental goal of our work is to build a transparent network view for the malware in a secure dynamic analysis environment. In our system, we use traffic replay, redirect, and relay to achieve the aforementioned goal. The approach works well for stateless protocols, where the retargeted traffic will be valid for the decoy as long as the decoy has the corresponding services running on it. For stateful protocols, such as layer 4 TCP, just replaying the captured packets will not work. For TCP, we need to replace the sequence numbers and acknowledge numbers in the TCP packet headers, because the TCP stack on the decoy may choose random sequence numbers different from the ones previously used by the would-be victim machines.

Essentially, when retargeting a connection, the decoy communicator has to

ensure that each packet conforms to the states on both the decoy and the running malware. This is taken care of by the stateful modules in the decoy communicator. Each stateful module is designed to maintain the states for each specific protocol. For instance, we have a MAC (media access control) stateful module to take care of the rewriting of MAC addresses. We also have an IP stateful module to replace IP addresses and recalculate IP checksums. Packets in retargeted traffic will have their destination MAC and IP addresses replaced with the ones of the decoy. At layer 4, we have a TCP stateful module to replace TCP sequence numbers, acknowledge numbers and TCP checksums. For upper layer protocols, we only implement the stateful modules for those protocols relevant to the malware samples used in our experiments. For instance, we have a stateful module for the NETBIOS protocol (layer 5) and a stateful module for SMB protocol (layer 7).

```
01  Maintaining_Protocol_States (packet)
02  {
03      CASE packet.connection.retarget_to OF
04          Replay:
05          Redirect:
06              layer7_stateful_modules (packet)
07              layer5_stateful_modules (packet)
08              layer4_stateful_modules (packet)
09              layer3_stateful_modules (packet)
10              layer2_stateful_modules (packet)
11          Relay:
12              layer3_stateful_modules (packet)
13              layer2_stateful_modules (packet)
14      ENDCASE
15  }
16
17  layer4_stateful_modules (packet)
18  {
19      TCP_stateful_module (packet)
20      UDP_stateful_module (packet)
21  }
22
23  TCP_stateful_module (packet)
24  {
25      If packet is a TCP packet Then
26          If packet is going to the decoy Then
27              mseq = packet.tcp_ack_num
28              mack = packet.tcp_seq_num + packet.tcp_data_len
29              packet.tcp_seq_num = dseq
30              If packet.tcp_ack_flag is set Then
31                  packet.tcp_ack_num = dack
```

```
32                  Endif
33              Else // from the decoy
34                  dseq = packet.tcp_ack_num
35                  dack = packet.tcp_seq_num + packet.tcp_data_len
36                  packet.tcp_seq_num = mseq
37                  If packet.tcp_ack_flag is set Then
38                      packet.tcp_ack_num = mack
39                  Endif
40              Endif
41
42              Fix TCP checksum
43          Endif
44      }
45
46      UDP_stateful_module (packet)
47      {
48      }
```

Figure 4: Pseudo code of selected stateful modules

Figure 4 shows the pseudo code of selected stateful modules. When a packet is sent to the stateful modules through Maintaining_Protocol_States( ), the packet will be dispatched to stateful modules at each protocol layers (Line 03~14). Within each layer, the packet will continue to flow through the relevant stateful modules. For instance, at layer 4, depending on whether the packet is a TCP or UDP packet, the respective stateful module (Line 23 and Line 46) will be invoked to process the packet. Within TCP_stateful_module (in Figure 4), the module checks whether the given packet is a TCP packet, keeps next sequence number and acknowledge number for each direction (Line 27~28 and Line 34~35), and rewrites the corresponding value if needed (Line 29~32 and Line 36~39). During the connection established, the next acknowledge number is calculated by last received sequence number plus last received payload length (Line 28 and Line 35). In Figure 4, we just give a brief introduction to stateful modules. In fact, the sequence number calculation is different when connection establishment and closing. On the other hand, actually, UDP does not have any stateful issues. We just keep the UDP_stateful_module empty.

The stateful modules are used to handle the stateful issues between the decoy and the would-be victim. Because in the redirect and relay phase, returning packets from

the decoy may forward back to the running malware, the decoy communicator applies layer 2, 3, 4, 5, and 7 stateful modules for the redirect phase and layer 2 and 3 for the relay phase. However, in the replay phase, since the connections have been finished, we simply ignore the returning packets from the decoy, so that the decoy communicator does not apply any stateful modules for it.

3.4 Example of Traffic Replay, Redirect, and Relay

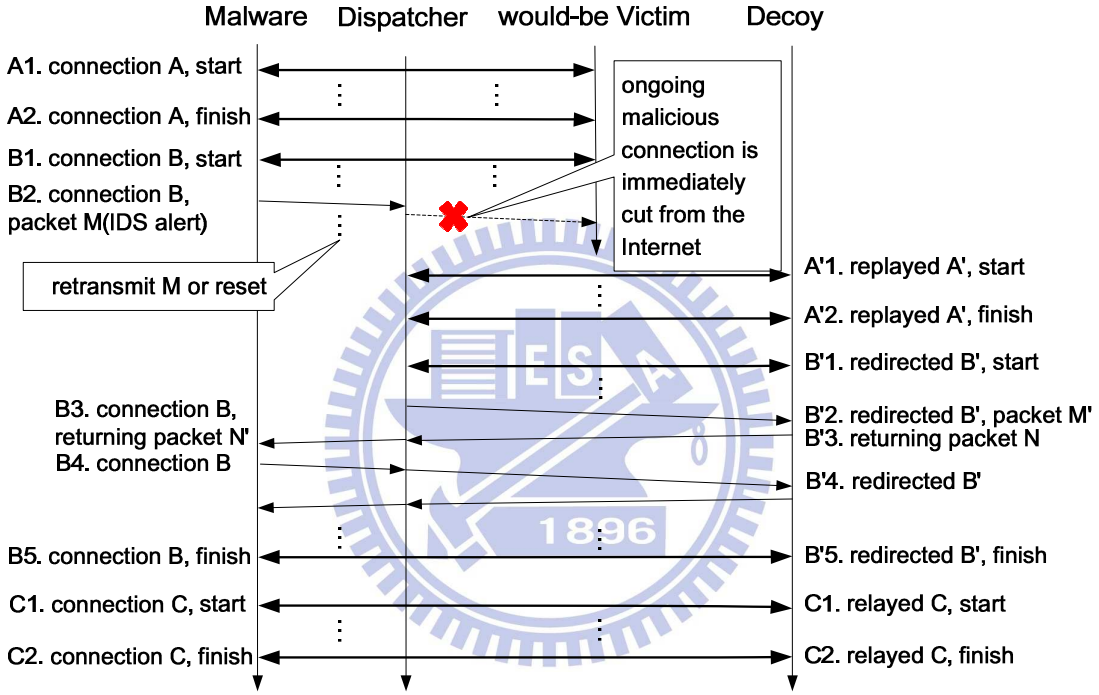


Figure 5: An example of traffic replay, redirect, and relay

Figure 5 shows an example of the network traffic retargeting, which consists of the three-phase process: replay phase, redirect phase, and relay phase, for a malicious session.

First, the malware in the analysis environment makes some network connection (connection A) with some would-be victim machine on the Internet. The dispatcher forwards the packets of connection A in both directions and also keeps a copy of the forwarded packets in the packet queue. Later, the malware makes another connection (connection B) with the would-be victim. The dispatcher again forwards and keeps a

copy of the packets in connection B. Now, assume that in the middle of connection B, the malware transmits out packet M, which contains malicious exploits that trigger an IDS alert. At this time, the dispatcher will consider the corresponding session (based on source and destination IP addresses) as malicious and begin the traffic retargeting. We use A' and B' instead of A and B to represent the replayed and redirected connections because the replayed (or redirected) packets will have to have different headers from the original ones such as TCP sequence numbers or MAC addresses.

Connection A, which has finished before the retargeting decision (i.e. IDS alerts), is replayed to the decoy as connection A'. Connection B, which is still ongoing, has to be redirected. For those packets in connection B transmitted before packet M, they are replayed as B'1 to the decoy. During the replay of B'1, the decoy may generate some corresponding response packets such as TCP ACK. We ignore these response packets from the decoy, because from the malware's point of view, the response packets had been received (from the would-be victim). Subsequent packets (include M) in connection B are relayed to the decoy (such as B'2, B'4). Note that, for this part of connection B, we need to forward returning packets from the decoy, if any, back to the malware (e.g. B3), or the ongoing connection can get broken prematurely. Connection C is opened after the retargeting decision, so it will be simply relayed in each direction by the dispatcher.

Chapter 4 Implementation

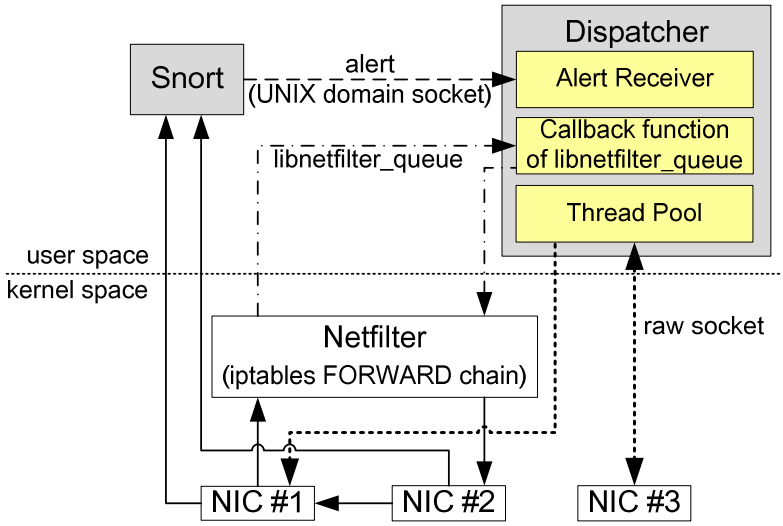


Figure 6: System Implementation

Figure 6 shows the implementation of our system. We implement our system on Linux, and use bridge-util to bridge two network interface cards (NIC #1 and NIC #2). One is connected to the Internet (NIC #2) and another is connected to the analysis environment (NIC #1). Packets received on NIC #2 (i.e. the Internet) can also be seen on NIC #1 (i.e. analysis environment). We use Netfilter [29] to intercept packets from NIC #1 (by setting iptables rules) and use libnetfilter_queue to forward the packets to the dispatcher. Packet flow from the analysis environment has to go through the dispatcher in order to reach the Internet even though NIC #1 and NIC #2 are bridged.

Dispatcher

If the destination IP address and port number of the packet are blacklisted, the callback function of libnetfilter_queue will signal the decoy communicator to carry out the traffic retargeting. Otherwise, the packets will be forwarded to NIC #2. We use Snort [30] as the IDS to detect propagation and attack traffic. Both incoming and outgoing traffic of the analysis environment will be inspected by Snort. In our system, Snort is modified to use UNIX domain socket to communicate with the alert receiver

in the dispatcher. If the alert receiver receives an alert, it will instruct the decoy communicator to initiate the traffic retargeting for the corresponding session.

To decrease the delay and increase efficiency of each component, we use concurrent programming model. Snort in our system is a standalone process while the alert receiver and the callback function of libnetfilter_queue run within two separate threads in the dispatcher process. We use UNIX domain socket for communication between Snort and the dispatcher.

The decoy communicator maintains a pool of threads, each of which handles a replayed, redirected, relayed connection. For traffic replay, since the malware no longer cares about the connections (they had been closed), the decoy communicator just copies payloads from the stored packets in the packet queue and uses standard socket (TCP or UDP) to regenerate the packets for the replay. Notably, we should fake the malware's IP address as the source IP address in these connections, because we should establish some states for the running malware. By setting the IP address of NIC #3 to the malware's IP address, we can use standard TCP or UDP socket to bind on it, and fake the connections (seems from the malware) through the socket. For traffic redirect and relay, we use raw socket to create the corresponding packets, so that fields such as MAC address, IP address, and TCP acknowledge number can be properly set by the respective stateful modules.

Stateful Modules

The decoy communicator includes stateful modules for upper layer protocols that are used by the malware samples we used for the experiments. The protocols include Server Message Block (SMB) [31] and NT LAN Manager Security Support Provider (NTLMSSP) [31]. The SMB stateful module replaces the tree id, process id, user id, and multiplex id fields in a SMB packet [31] (Figure 7) during the redirect phase. The

NTLMSSP stateful module is used to fix states during a SMB logon process (Figure 8), which relies on the NTLMSSP challenge-response authentication.

8	16	24	32 bits
Command	RCLS	Reserved	ERR
ERR	REB/FLG	Reserved	
Reserved			
Reserved			
Reserved			
Tree ID		Process ID	
User ID		Multiplex ID	
WCT	VWV		
BCC		BUF	

Figure 7: SMB packet format

As an example to show how a stateful module works, let's consider that a malware is attempting a SMB logon to a would-be victim machine through brute-force password guessing. Assume that before the malware succeeds with its password guessing, the IDS generates an alert due to too many SMB logon failures (such as Figure 8). The dispatcher will now retarget connections in the corresponding session. Essentially, we want to redirect the SMB logon connection to the decoy, which include three transmitted packets (1), (3), and (5) in the packet queue. The dispatcher will first replay packet (1) to the decoy, which works perfectly. Then, it will replay (3) to the decoy, and the decoy will reply with a NTLM_CHALLENGE_MESSAGE (like packet (4)) that contains a challenge value. If we continue to replay packet (5), the logon process will fail, as the response in packet (5) corresponds to the original challenge in packet (4) from the would-be victim and does not match the challenge generated by the decoy. As a result, the NTLMSSP [32] stateful module needs to calculate a new response value for the challenge from the decoy so that the redirected logon process can continue to proceed.

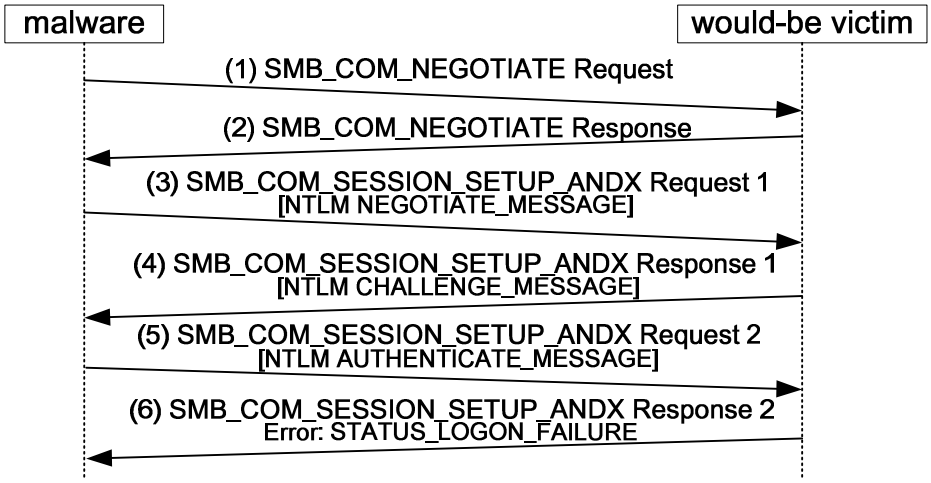


Figure 8: A SMB logon failure process

Decoy

The implementation of the decoy is a virtual machine loaded with operating system or network services corresponding to the well-known protocols that will be subject to the attacks from the malware in the analysis environment. For example, we implement a SMTP decoy (with standard socket) to emulate a victim SMTP server for the spammer-type malware, which we use for evaluation in Section 5.2. The SMTP decoy accepts any SMTP request, but never sends the e-mails in actually.

Chapter 5 Experiment Studies

We evaluate our system design with 12 real-world malware samples. In Section 5.1, we describe the selection criteria for the malware samples and the experiment environment. In Section 5.2~5.3, we evaluate the effectiveness of our system by comparing the dynamic analysis results from three different environments: our secure and transparent network environment, a closed network environment, and an open network environment. In Section 5.4, we give a case study to show our system's operations. In Section 5.5, we give another case study on those unexpected results observed from the experiment.

5.1 Sample Selection and Experiment Environment

We collect more than 2000 suspicious malware samples from different sources including P2P file sharing, e-mail attachments, phishing websites, and honeypots running with Nepenthes [33]. First, we scan the suspicious malwares with anti-virus software from four different vendors and keep only those flagged by all of the four scanners. This results in 124 malwares. Next, we execute the remaining malware samples in an open network environment for 2 minutes and observe if they exhibit any network activities. We exclude those samples that exhibit no network activities and also exclude those whose network connection attempt does not get through (presumably, the remote server is not operational). In the end, we have a total of 12 malware samples. The 12 malwares are further separated into two groups: *malware without C&C*, and *malware with C&C*.

The first group, malware without C&C, consists of worms and e-mail spammers. Worms in the first group (m10.exe, m11.exe, and m12.exe in Table 1) propagate by brute-force password guessing on SMB logon over NETBIOS. After a successful

logon, the worm binary is copied to the target machine and gets executed. The first group also includes a spammer (m7.exe) which attaches a copy of the spammer binary "Worm/NetSky.P" to the e-mail content.

The second group consists of malware with C&C (also known as bots). In this group, we have spammers (m8.exe and m9.exe) whose spam e-mail content and target recipient lists can be updated from a C&C server. We also have malwares (m1.exe, m2.exe, m3.exe, m4.exe, m5.exe, and m6.exe) that await commands from the C&C server to carry out propagate or attack actions. For example, we observe that they receive propagation-related commands, and scan machines randomly and propagate via vulnerabilities of NETBIOS.

The selected samples are summarized in Table 1 and the discovery time is based on [34].

Table 1: Selected samples

Type	Malware	Scan Result	Discovered	Activities
Malware Without C&C	m7.exe	Email-Worm.Win32.NetSky.q	Mar 24 2004 09:02 GMT	"Worm/NetSky.P" attachment
	m10.exe	Worm.Win32.Fujack.aa	Jul 02 2007 14:18 GMT	SMB password guessing
	m11.exe	Worm.Win32.Fujack.aa	Jul 02 2007 14:18 GMT	
	m12.exe	Worm.Win32.Viking.n	Aug 03 2006 22:09 GMT	
Malware With C&C	m1.exe	Trojan.Win32.Scar.bqfv	Feb 25 2010 16:09 GMT	SMB password guessing NETBIOS buffer overflow attempts
	m2.exe	Packed.Win32.Black.d Backdoor.Win32.Rbot.gen	Aug 06 2004 12:02 GMT	
	m3.exe	Trojan-PSW.Win32.Dybalom.bu	Aug 15 2009 09:06 GMT	
	m4.exe	P2P-Worm.Win32.Palevo.vyc	Mar 05 2010 12:11 GMT	
	m5.exe	Trojan-PSW.Win32.Dybalom.bu	Aug 15 2009 09:06 GMT	
	m6.exe	Trojan-PSW.Win32.Dybalom.bu	Aug 15 2009 09:06 GMT	
	m8.exe	Virus.Win32.Tenga.a	Jul 22 2005 17:11 GMT	Get e-mail content and recipient lists from the C&C
	m9.exe	Trojan-PSW.Win32.LdPinch.gqo	Feb 13 2009 15:42 GMT	

We set up an experiment environment following the architecture in Figure 2 which is shown in Figure 9. For each experiment, a malware is executed for 10 minutes. We use TCPDUMP [35] to record three types of traffic: the traffic that interacts with the analysis environment (A in Figure 9), the traffic that reaches the Internet (B in Figure 9), and the traffic that retargets to the decoys (C in Figure 9). We use the recorded traffic to evaluate both the improvement on network transparency

and security as provided by our proposed system.

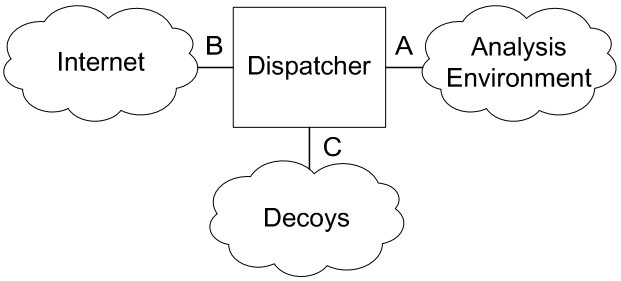


Figure 9: Basic experiment environment

5.2 Effectiveness of Transparent Network Environment

Our system gives 3.35 times more packets than closed network

By using our system, we can see 3.35 times more packets than using a closed network environment on average. Table 2 summaries the observed network activities from running malware without C&C in a closed network environment (no connection with Internet) and in our secure and transparent network environment. In the closed network environment, the majority of the packets are just TCP SYN as connections to the outside world are blocked. On the other hand, we can see a lot more network activities in our system. For instance, we observe that m7.exe attempts to initiate SMTP connections for sending spam e-mails. The malware m10.exe generates a lot more port 139 and 445 traffic (369199 packets) compared with the result from the closed network environment (707 packets). Because some of the traffic from m10.exe is flagged by the IDS ("NETBIOS SMB-DS repeated logon failure" alert message), we retarget the network paths as described in Section 3.2. As a result, the propagation activities of m10.exe can be completely observed (m11.exe and m12.exe are similar to m10.exe) and the spam e-mail content of m7.exe can also be completely observed if we provide a proper SMTP decoy (details in Section 5.3 and Section 5.4).

In this group, some attack traffic (e.g. NETBIOS attack traffic from for m10.exe, m11.exe, and m12.exe) are LAN-based and do not require Internet access. The kind

of traffic can be observed in a closed network environment if a vulnerable server is present in the analysis environment. It is quite often seen that the same malware also involve Internet traffic. For instance, m10.exe, m11.exe, and m12.exe all make HTTP connections to advertising sites on the Internet. If we execute them in closed network environment, only a few TCP SYN packets for HTTP connections are observed. (Note: in our system, the HTTP traffic is allowed because they are deemed harmless as the IDS generates no alarms on them).

Table 2: Network activities by malware without C&C (closed network vs. our system)

Malware	Closed Network	Our system
m7.exe	No response for DNS MX record	9 spam e-mail attempts
m10.exe	362 TCP port 139 SYN packets 345 TCP port 445 SYN packets	369199 packets for TCP port 139 and 445 HTTP GET advertising HTML files
m11.exe	407 TCP port 139 SYN packets 388 TCP port 445 SYN packets	23161 packets for TCP port 139 and 445 HTTP GET advertising HTML files
m12.exe	Probe machines by ICMP echo request	Probe machines by ICMP echo request 60285 packets for TCP port 139 and 445 HTTP GET advertising HTML files

We notice that if we provide a vulnerable SMB in the closed network environment (Figure 10), the traffic of m10.exe, m11.exe, and m12.exe will not only contain TCP SYN packets. The running malware will inject into the vulnerable SMB server and generate lots of traffic.

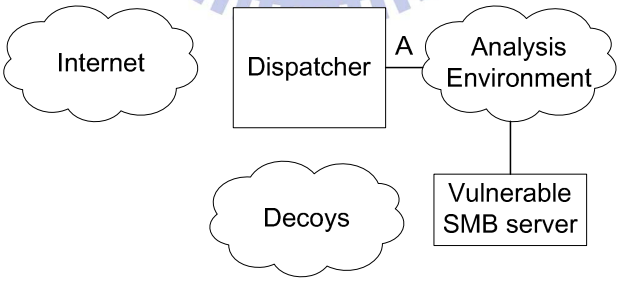


Figure 10: Additional experiment environment with closed network

Table 3: Number of packets by m10.exe, m11.exe, and m12.exe (closed network vs. our system)

Malware	Closed Network	Our System
m10.exe	2750	10915
m11.exe	8261	44692
m12.exe	1018	21376

Table 3 shows the results from doing this experiment. Notably, the numbers are different from Table 2 because we redo the experiment. We get 2.13 times more

packets than closed network on average.

The results for malware with C&C are shown in Table 4. Again, we can only see those packets corresponding to unsuccessful connection attempts in the closed network environment. With our system, we can observe a lot more network activities. Malwares in this category heavily depend on interaction with the C&C server to operate. For instance, m4.exe needs to connect to an IRC server on port 47221 and waits for commands. During the experiment period, m4.exe received a HTTP downloading command that gets the malware "TR/Kazy.15451.21". Malware m4.exe also received a propagation-related command (".asc" command in Table 4 for m4.exe) and scanned machines for port 445 in order to initiate propagation actions. Unfortunately, the machines targeted by m4.exe were not running during the experiment period, thus the propagation traffic did not trigger the IDS alert. Similar to m4.exe, m3.exe received a propagation-related command (".advscan" command in Table 4 for m3.exe), and scanned machines for port 445. At this time, the IDS issued an alert for "NETBIOS DCERPC NCACN-IP-TCP srvsvc NetrPathCanonicalize overflow attempt". Then, we retargeted the network paths of m3.exe as described in Section 3.2. Malware m8.exe and m9.exe are spammers which target Yahoo e-mail service. They connect to a C&C server on port 80, and download some sentences (e-mail subjects) and e-mail addresses (recipients). After finishing the downloading, they start to send spam e-mails. These spam e-mails will not trigger the IDS alert which are actually sent to the victim recipients. Luckily, they are deemed harmless that we discuss in Section 5.3.

Consequently, if we execute malwares of this group in a closed network environment, we may get few network activities due to the Internet inaccessible. Without a C&C server, the malware will not know how to take actions. However, in our system, since we allow the C&C traffic to access the Internet, the malware can get

the commands.

Table 4: Network activities by malware with C&C (closed network vs. our system)

Malware	Closed Network	Our system
m1.exe	No response for DNS A query No response for TCP SYN	TCP C&C connection (60.165.98.198:8680)
m2.exe	No response for DNS A query	TCP C&C connection (70.107.249.167:6668) TCP SYN flooding at port 139 after receiving "xvvv asn1smbnt 100 0 0 -b -r -s" command
m3.exe **m5.exe** **m6.exe**	No response for DNS A query	TCP C&C connection (74.117.174.122:16667) TCP SYN flooding at port 445 after receiving ".advscan asn445 100 5 0 -b -r -s" command FTP connection with non-standard port
m4.exe	No response for DNS A query	TCP C&C connection (46.161.29.202:47221) HTTP GET "TR/Kazy.15451.21" after receiving ".asc -S -s|.http http://black-cash.com/rep.exe|.asc exp_all 10 0 0 -b -s|.asc exp_all 20 0 0 -b -r -e –s" command HTTP GET status report from other bots in the C&C channel TCP SYN flooding at port 445 after receiving command
m8.exe	No response for DNS MX query TCP SYN flooding at port 139	TCP C&C connection (208.77.45.146:80) TCP SYN flooding at port 139 34 spam e-mails
m9.exe	No response for DNS MX query	TCP C&C connection (208.77.45.146:80) 179 spam e-mails

Our system can be more transparent than open network

We notice that in a few situations, with proper decoys, our system can outperform an open network environment in terms of network transparency.

Since the number of all network activities is hard to count, we can focus on different types of activities to calculate the improvement rate (# of activities in our system / # of activities in open network environment).

For NETBIOS-based propagation activities, we may count number of intruded machines. However, it is another concern if we allow the malware to propagate to machines on the Internet. One possible way to calculate the improvement rate is providing enough machines running vulnerable NETBIOS server within a closed network environment (isolated from the Internet). In our experiment, we have no such many machines, thus we do not show the evaluation for this.

For spam e-mail activities, we may count number of e-mails that successfully be

sent out. Table 5 shows improvement rates for each spammer in our experiment. For m7.exe, SMTP servers on the Internet do not relay for the malware anymore, so that the number of e-mail in open network environment is zero. But in our system, the SMTP decoy will not refuse any SMTP request. We can still see 14 spam e-mails in this case. For m8.exe and m9.exe, both of their spam e-mails target the Yahoo e-mail service. We observe that some of spam e-mails of m8.exe and m9.exe are denied due to the anti-spam mechanism from Yahoo [36]. However, in our system, again, the SMTP decoy will not refuse any SMTP request. We can see even more number of spam e-mails than in open network environment. As a result, we get 170.32% improvement rate on average. (Note: in this experiment, we use our system with blacklist in order to relay SMTP traffic directly; the numbers are different from Table 2 and Table 4 since we redo the experiment)

Table 5: Number of spam e-mails (our system vs. open network)

Malware	Our System	Open Network	Improvement Rate
m7.exe	14	0	N/A
m8.exe	117	68	172.06%
m9.exe	118	70	168.57%

5.3 Effectiveness of Secure Network Environment

Internet security is ensured

We notice the Internet security face of the experiments in Section 5.2. Among all the packets in the experiments in Section 5.2, we retarget 80.66% packets on average. The retargeted packets trigger Snort 303 alert messages (from m1.exe, m2.exe, m3.exe, m4.exe, m5.exe, m6.exe, m10.exe, m11.exe, and m12.exe) and contain 222 spam e-mails (from m7.exe, m8.exe, and m9.exe with blacklist). If we execute the malwares in an open network environment, the propagation or attack actions and spam e-mails will flow to the Internet.

The dispatcher finds m10.exe, m11.exe, and m12.exe launching SMB password

guessing attack when the IDS issues “NETBIOS SMB-DS repeated logon failure” alert. The dispatcher will retarget the corresponding session of the malware, so that the would-be victim on the Internet can be secured. However, from the malware’s point of view, the victim still on the Internet (i.e. the decoy). The malware can easily logon to the decoy because the help from stateful modules. After the malware logons successfully, it will transmit itself to the decoy and register to the decoy’s system scheduler for executing. If we do not retarget the password guessing attack, the would-be victim will suffer from the same situation. Thus, it is dangerous to the Internet.

Our system is able to retarget the password guessing attack at an early stage (about 10 attempts, depending on Snort rule setting). If a machine (vulnerable SMB server in Figure 11) on the Internet adopts a quite simple password (e.g. easily guessable after 3 attempts), our system can also protect it. Although the malware logon the vulnerable SMB server successfully, the IDS will issue an alert “NETBIOS SMB-DS ADMIN$ unicode share access” when the malware binary is copying into the SMB server. As a result, the dispatcher will retarget the traffic; the propagation traffic cannot completely reach the would-be victim (i.e. the vulnerable SMB server).

For others malware, the Internet security is also ensured. For instance, m8.exe and m9.exe send spam e-mails. Although the e-mails may annoy users, the mail content actually harmless. Malware m7.exe produces spam e-mails with a malware attachment. However, the SMTP servers targeted by m7.exe refuse to relay e-mails for the malware. In this case, the spam e-mails cannot be sent out, making the Internet more secure. If a SMTP server (SMTP server in Figure 11) on the Internet relay e-mails for m7.exe, the IDS will produce an alert “SHELLCODE x86 inc ecx NOOP” when the IDS inspects the mail attachment (i.e. a malware attachment). The alert will trigger the dispatcher to retarget the traffic. Again, the would-be victim (i.e. the

SMTP server) on the Internet is secured.

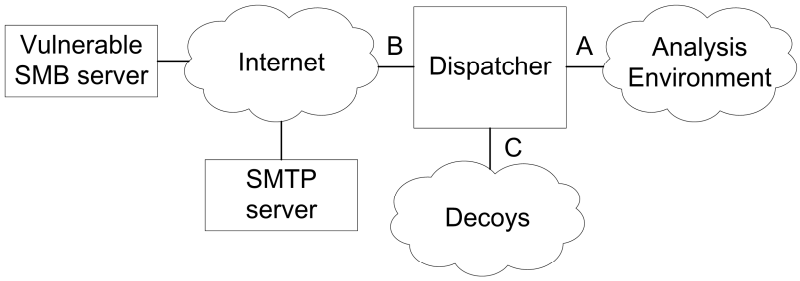


Figure 11: Additional experiment environment

5.4 Case Study: A normal case

In this case study, we select m7.exe to be the malware that runs in the dynamic analysis environment. Notably, m7.exe is a spammer; however, the SMTP servers targeted by m7.exe refuse to relay e-mails for it. We setup a SMTP server on the Internet that relays e-mails for the malware (as shown in Figure 11). This seems very insecure, but actually in our system, we can retarget the malicious traffic of m7.exe and ensure that the spam e-mails never reach the victim's mailbox.

First, the malware tries to query DNS MX records (packet #1 and #4 in Figure 12). Then, the DNS servers will reply with the MX results (packet #2 and #5 in Figure 12) that also include A records in DNS additional records. Actually, we modify %SystemRoot%\system32\drivers\etc\hosts configuration file, so that the malware will connect to our designed machine (i.e. SMTP server in Figure 11) and the A record is unused. Upon receiving the responses, the malware starts to connect to the remote SMTP servers (packet #3 in Figure 12). Note that the malware may simultaneously connect to multiple SMTP servers, in order to increase the efficiency of the spammer (packet #1 and #3 in Figure 12 are for different targets).

```
1 Standard query MX sexnet.com
2 Standard query response MX 10 mailstore1.secureserver.net MX 0 smtp.secureserver.net
3 1035 > 25 [SYN] Seq=0 Win=64860 Len=0 MSS=1410 SACK_PERM=1
4 Standard query MX domain.com
5 Standard query response MX 10 sentry.domainbank.com
```

Figure 12: DNS queries for spam e-mails

```
 9 S: 250-smtp.emu.org | 250-8BITMIME | 250-SIZE 41943040 | 250 PIPELINING
10 C: MAIL FROM:<austria@msdirectservices.com>
11 S: 250 sender <austria@msdirectservices.com> ok
12 C: RCPT TO:<user@domain.com>
13 S: 250 recipient <user@domain.com> ok
14 C: DATA
15 S: 354 go ahead
16 C: From: austria@msdirectservices.com
17 C: DATA fragment, 1444 bytes
18 25 > 1036 [ACK] Seq=196 Ack=1537 Win=8460 Len=0
19 C: DATA fragment, 325 bytes
20 C: DATA fragment, 1410 bytes
21 C: DATA fragment, 1410 bytes
```

Figure 13: A SMTP session for a spam e-mail

Afterwards the malware initiates a SMTP session in order to send spam e-mails (Figure 13). Due to the Snort alert (Figure 14) triggered by the packet #20 in Figure 13, the dispatcher in our system will retarget the whole session. Finally, we can extract the entire content of e-mails (Figure 15) and also ensure Internet security through traffic retargeting.

```
[**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
[Classification: Executable Code was Detected] [Priority: 1]
04/21-22:44:54.368243 140.113.88.187:1036 -> 140.113.88.179:25
TCP TTL:128 TOS:0x0 ID:84 IpLen:20 DgmLen:1450 DF
***AP*** Seq: 0xAC7D3BBB  Ack: 0xF845FBE6  Win: 0xFC99  TcpLen: 20
```

Figure 14: Snort issues an alert for e-mail content

```
From: austria@msdirectservices.com
To: user@domain.com
Subject: Mail Delivery (failure user@domain.com)
Date: Thu, 21 Apr 2011 22:40:07 -0700
MIME-Version: 1.0
Content-Type: multipart/related;
.type="multipart/alternative";
.boundary="----=_NextPart_000_001B_01C0CA80.6B015D10"
X-Priority: 3
X-MSMail-Priority: Normal

This is a multi-part message in MIME format.

------=_NextPart_000_001B_01C0CA80.6B015D10
```

Figure 15: Partial content of the spam e-mail

5.5 Case Study: An unexpected case

Before this case study, we have confirmed that our implementation of relaying connections with blacklist works. Suppose in the worst case, our system may fail to function in replay and redirect phases, it should work in the relay phase. Interestingly, we still fail in the relay phase in the following case.

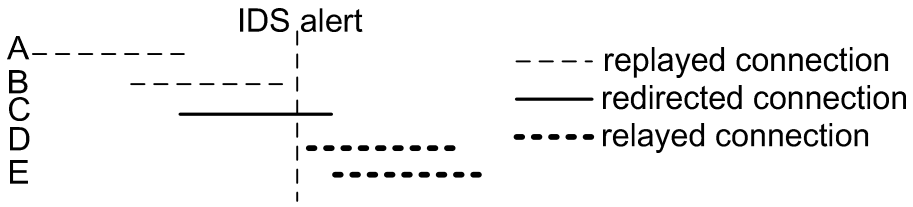


Figure 16: An unexpected case

Figure 16 shows a case where all phases of traffic retargeting fail. Every segment in Figure 16 represents a SMB logon connection, and x-axis represents time. When the number of failed SMB logons (A and B in Figure 16) reaches a threshold value, Snort will generate alert “NETBIOS SMB-DS repeated logon failure”. Then, our system will engage traffic retargeting (i.e. replay, redirect, and relay). Connection A and B are designed to replay, connection C is designed to redirect, and connection D and E are designed to relay.

Each connection in the case is independent. That is, effectiveness of replaying connection A and B in the case may be nullified, because no states need to reestablish. Ideally, we can redirect connection C to the decoy. But we find that we always fail in this case. This is because the connection C almost finishes, from the malware’s perspective of view, the connection C is no longer useful. For instance, packets of connection C are like packets shown in Figure 8. When the IDS sees the packet (6) in Figure 8, it triggers an alert. Even though we try to redirect the connection C, from the malware’s point of view, the connection C already logon failure, and should be closed. As a result, we replace fields by stateful modules and the logon to the decoy successfully, the malware still send TCP RESET or TCP FIN.

Finally, we consider that the logon should success in the relay phase. But it fails again. The most possible reason is that the decoy adopts an empty password and the empty password has been tried in the early stage of the password guessing. Once the malware find a password cannot logon success, it will not try it again. Connection A

and B in Figure 16 are mostly about an empty password. Consequently, it makes logon still fail in connection E and F.

In our original design, we only apply layer 2 and 3 stateful modules for the relay phase. In this case, we try to additionally apply a layer 7 stateful module for the relay phase. The module replaces the challenge-response fields to help the malware to logon to the decoy. By doing so, we can see a successful logon, malware binary transmission (packet #1 in Figure 17), and scheduler registration (packet #146 in Figure 18).

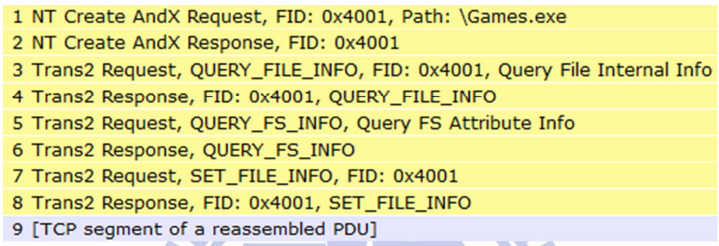


Figure 17: Transfer the malware binary via SMB

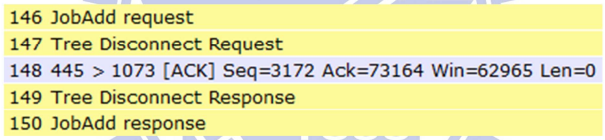


Figure 18: Using ‘at’ scheduler

Chapter 6 Conclusions and Future Works

Dynamic malware analysis traditionally runs in a closed network environment without Internet connection. This prevents the malware from causing damages to the outside world. However, for malware that involves significant amount of network activities, a closed network environment defeats the purpose of dynamic analysis, as much of the malware's network behavior will not be exhibited and captured.

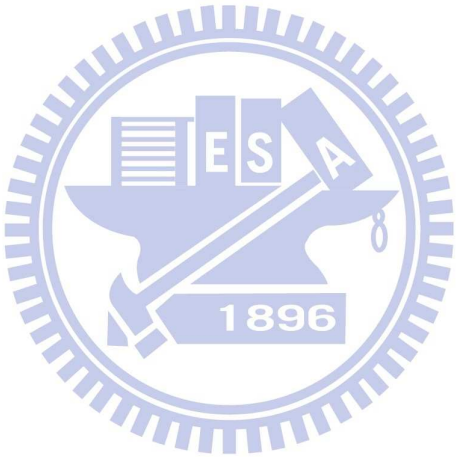
We propose a system to allow malware exhibiting network behavior in a dynamic malware analysis environment while also ensuring that the malware can do no harm beyond the boundary of the analysis environment. Our system transparently retargets propagation and attack traffic, instead of blocking them, to decoys inside the analysis environment. At the same time, we allow the malware's control traffic, which is deemed to be harmless, to cross the boundary of the analysis environment.

The evaluation result shows that our system significantly increases the amount of observed network activities during dynamic malware analysis when compared with a traditional closed network environment. The overall effect is having a dynamic analysis environment, which is useful for those malware with lots of network activities.

The use of traffic retargeting and decoys in our system can improve the effectiveness of dynamic analysis beyond what an open network environment (with unrestricted Internet access) can offer. This happens when a malware requires accessing machines on the Internet, which for some reason are not accessible during the time of analysis. An example is a spam-ware sending spam e-mails through a hard-coded SMTP server that was known to accept public relays. If the hard-coded SMTP is no longer functioning, a dynamic analysis of the malware will fail to reveal the full picture of the malware's behavior. In our experiments, we were able use our

system to retarget the SMTP traffic of such a spam-ware and extract both the recipient list and the mail content (including a backdoor program in the attachment part) from the spam-ware.

From the second case study, we can see some cases of malware may be unexpected in our design. In the future work, we will attempt to execute more malware samples and apply different stateful modules for each protocol in different case. Besides, in our experiment, we use a simple dynamic malware analysis. It may be observed more meaningful activities by using a sophisticated dynamic malware analysis environment.



References

[1] "Symantec Malware Threat Explorer," [online], available from World Wide Web; http://www.symantec.com/business/security_response/threatexplorer/index.jsp.

[2] "Kaspersky Monthly Malware Statistics," [online], available from World Wide Web; http://usa.kaspersky.com/resources/knowledge-center/statistics.

[3] "Avira Virus Lab," [online], available from World Wide Web; http://www.avira.com/en/support-virus-lab.

[4] P. Szor, "The art of computer virus research and defense," *Addison-Wesley Professional*, 2005.

[5] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," *Department of Computer Science, The University of Auckland, New Zealand*, 1997.

[6] C. Greamo and A. Ghosh, "Sandboxing and Virtualization: Modern Tools for Combating Malware," *Security Privacy, IEEE*, vol. 9, pp. 79 -82, 2011.

[7] X. Chen, J. Andersen, Z. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pp. 177-186, 2008.

[8] P. Ferrie, "Attacks on more virtual machine emulators," *Symantec Technology Exchange*, 2007.

[9] M. Carpenter, T. Liston, and others, "Hiding virtualization from attackers and malware," *IEEE Security and Privacy, Published by the IEEE Computer Society*, pp. 62-65, 2007.

[10] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology, Springer*, vol. 2, pp. 67-77, 2006.

[11] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy, IEEE Computer Society*, pp. 32-39, 2007.

[12] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A tool for analyzing malware," *15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, 2006.

[13] R. Puri, "Bots & botnet: An overview," *SANS Institute 2003*, .

[14] P. Barford and V. Yegneswaran, "An inside look at botnets," *Malware Detection, Springer*, pp. 171-191, 2007.

[15] K. Yoshioka, Y. Hosobuchi, T. Orii, and T. Matsumoto, "Vulnerability in Public Malware Sandbox Analysis Systems," *2010 10th Annual International*

Symposium on Applications and the Internet, pp. 265-268, 2010.

[16] J. Crandall, G. Wassermann, D. de Oliveira, Z. Su, S. Wu, and F. Chong, "Temporal search: Detecting hidden malware timebombs with virtual machines," *ACM SIGARCH Computer Architecture News, ACM*, vol. 34, pp. 25-36, 2006.

[17] D. Dagon, G. Gu, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton, "A taxonomy of botnets," *Unpublished paper, c, Citeseer*, 2005.

[18] "Norman Sandbox," [online], available from World Wide Web; http://www.norman.com/security_center/security_tools/.

[19] "The Reusable Unknown Malware Analysis Net," [online], available from World Wide Web; http://www.secureworks.com/research/tools/truman/.

[20] M. Kim, M. Kim, and Y. Mun, "Design and Implementation of the HoneyPot System with Focusing on the Session Redirection," *Computational Science and Its Applications--ICCSA 2004, Springer*, pp. 262-269, 2004.

[21] I. Kim and M. Kim, "The DecoyPort: redirecting hackers to honeypots," *Network-Based Information Systems, Springer*, pp. 59-68, 2007.

[22] L. Spitzner, "The honeynet project: Trapping the hackers," *IEEE Security and Privacy, Published by the IEEE Computer Society*, pp. 15-23, 2003.

[23] I. Alberdi, E. Alata, V. Nicomette, P. Owezarski, and M. Kaâniche, "Shark: Spy Honeypot with Advanced Redirection Kit," *IEEE Workshop on Monitoring, Attack Detection and Mitigation (MonAM'07)*, pp. 47-52, 2007.

[24] E. Alata, I. Alberdi, V. Nicomette, P. Owezarski, and M. Kaâniche, "Internet attacks monitoring with dynamic connection redirection mechanisms," *Journal in Computer Virology, Springer*, vol. 4, pp. 127-136, 2008.

[25] J. Grizzard, V. Sharma, C. Nunnery, B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pp. 1-1, 2007.

[26] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot: a botnet protocol based on Kademlia," *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pp. 1-9, 2008.

[27] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," *The First Workshop in Understanding Botnets*, 2007.

[28] G. Berger-Sabbatel and A. Duda, "Analysis of Malware Network Activity," *Multimedia Communications, Services and Security, Springer*, pp. 207-215, 2011.

[29] "Netfiler," [online], available from World Wide Web; http://www.netfilter.org/.

[30] "Snort," [online], available from World Wide Web; http://www.snort.org/.

[31] "SMB Packet Header," [online], available from World Wide Web; http://www.protocols.com/pbook/ibm.htm.

[32] "MS-NLMP - NT LAN MANAGER (NTLM) Authentication Protocol

Specification," [online], available from World Wide Web; http://msdn2.microsoft.com/en-us/library/cc207842.aspx.

[33] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The nepenthes platform: An efficient approach to collect malware," *Recent Advances in Intrusion Detection*, pp. 165-184, 2006.

[34] "Kaspersky Securelist," [online], available from World Wide Web; http://www.securelist.com/en/find.

[35] "TCPDUMP," [online], available from World Wide Web; http://www.tcpdump.org/.

[36] "421 4.16.55 [TS01] Messages from x.x.x.x temporarily deferred due to excessive user complaints," [online], available from World Wide Web; http://help.yahoo.com/l/us/yahoo/mail/postmaster/errors/421-ts01.html.