SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.764

RESEARCH ARTICLE

Secure and transparent network traffic replay, redirect, and relay in a dynamic malware analysis environment

Ying-Dar Lin[1], Tzung-Bi Shih[1], Yu-Sung Wu[1]* and Yuan-Cheng Lai[2]

[1] Department of Computer Science, National Chiao Tung University, Hsin Chu 300, Taiwan
[2] Yuan-Cheng Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan

ABSTRACT

Dynamic analysis is typically performed in a closed network environment to prevent the malware under analysis from attacking machines on the Internet. However, many of today's malwares require Internet connectivity to operate and to be thoroughly analyzed in a closed network environment. We propose a secure and transparent network environment that allows the malware in a dynamic analysis environment to have seemingly unrestricted Internet access in a secure manner. Our environment transparently dispatches malicious network traffic to compatible decoys while allowing harmless control traffic to have Internet access. We use 12 real-world malware samples, which involve Internet connections, to evaluate the effectiveness of the proposed environment. The evaluation shows that the proposed environment can allow malware to exhibit more network activities than a closed network environment and can even outperform the baseline open network environment in some cases. In the meantime, Internet security is maintained by the dispatching of attack and propagation traffic to decoys inside the analysis environment. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS
malware; dynamic analysis; transparent network; botnet

***Correspondence**
Yu-Sung Wu, Department of Computer Science, National Chiao Tung University, Hsin Chu 300, Taiwan.
E-mail: ysw@cs.nctu.edu.tw

1. INTRODUCTION

Malware has been a major threat to computer security for years [2,3]. The defense against malware typically follows a three-step process: analysis of unknown malware, signature development, and deployment of anti-virus scanners for known malware [4]. The analysis of unknown malware is the important first step in the process of malware defense, as the other two steps would not be possible without a proper understanding of the malware first.

Malware analysis comes in two manners: static analysis and dynamic analysis [5]. Static analysis looks for patterns characterizing malicious behaviors in a malware's binary. The main strength of static analysis is that it runs very fast. Efficient algorithms for disassembly and pattern matching are readily available. Static analysis is also very secure because the malware never gets executed during the analysis process. However, the major drawback with static analysis lies in its inability to deal with binary obfuscation [6]. With binary obfuscation, the original malware code may get encrypted, shuffled, or even transformed into opaque instructions to be executed on a virtual machine. As a result, static analysis may have to involve more advanced analysis techniques such as code emulation to deal with obfuscated binary. Even so, the overall result is still far from perfect. This is largely because there are always some new ways to obfuscate a binary, and it would be very difficult to design a static analysis algorithm to encompass all possibilities beforehand.

Complementary to static analysis, dynamic analysis [5,7,8] actually puts a malware into execution in a closed environment and observes the malware's runtime behavior. Binary obfuscation is generally not an issue for dynamic analysis, as the malware should reveal itself automatically during the execution. However, there are still a few other issues with dynamic analysis. First, malware may detect the presence of analysis environment and refrain from showing its true behaviors [9,10]. As an example, malware may check if hardware debug registers have been set, which is a sign of it being debugged (analyzed). Second, malware may be programmed to act only at a specific time or date (i.e., a logic bomb) [4]. Unless the dynamic

analysis is carried out at the right time, it will not be able to capture the full behavior of the malware. Third, malware can be remotely triggered or depend on services on the Internet for its operation (i.e., bots [11,12]). In this case, if the analysis environment does not have network connectivity with the outside world, the dynamic analysis will also fail to capture the full behavior of the malware.

Fortunately, not all the aforementioned issues are difficult to resolve. For instance, for malware that can detect the presence of analysis environment, there is always some way to alter the environment settings to remove the specific signatures the malware is looking for [13]. For logic bombs, one can tweak the machine clock to the dates and times likely to be of interest to the malware [14]. The trickiest issue is with those malware that require Internet connectivity. This kind of malware either depends on network communication with a controller or network services on the Internet to operate or is designed to carry out attacks against targets on the Internet. As a result, if dynamic analysis of the malware is carried out in a closed network environment, it is very likely that most of the malware's behavior will not be observed. On the other hand, if dynamic analysis is carried in an open network environment with unrestricted access to the Internet, then there is the concern about the potential damage the malware can cause to vulnerable machines on the Internet.

To address the dilemma between open and closed network environments for dynamic malware analysis, we propose an alternative environment that can transparently and securely dispatch the network traffic of a malware under analysis. The environment leverages a network intrusion detection system (IDS) to help distinguish attack traffic from benign traffic in addition to the use of port-based attack traffic redirection mechanism. Through a three-phase dispatching process that involves traffic replay, redirect, and relay, the environment allows the malware to communicate freely with its controller or services on the Internet, and at the same time, it can transparently divert attack traffic to decoys within the analysis environment. We manage to achieve a good balance between the open network environment and the closed network environment: The malware will not quit prematurely because of network inaccessibility, and the security of the Internet is ensured. An open-source implementation of the proposed environment is available for download at [1].

2. BACKGROUND

The work is motivated by the abundance of malware that depends on the Internet for operations (e.g., propagation and attack). One notable class of malware that involves heavy Internet usage is bot, a type of malware that takes commands from a controller on the Internet to achieve specific attack goals [11]. In what follows, we shall first give a brief overview of the network activities involved in a bot's operation. We shall also mention how existing dynamic analysis environments handle the network activities of malware and their shortcomings.

2.1. Network traffic of botnet

Bots are a type of malware designed to function in a collective manner as shown in Figure 1. A controller commands a herd of bots that can be used to carry out attacks on target victims. The whole system is often referred to as a "botnet" [11,12], meaning a network of bots. As shown in Figure 1, a botnet involves many network activities. The network activities of a botnet can be roughly put into three categories: propagation, command and control (C&C) communication, and attack [15]. Propagation corresponds to the traffic generated by a bot for infecting other machines to expand the size of a botnet. C&C communication refers to the network traffic between a controller and a bot, which may include commands sent by the controller or sensitive information pilfered from victims (e.g., credit card numbers). Finally, a botnet can be used to launch network attacks such as distributed denial of service or email spamming, which

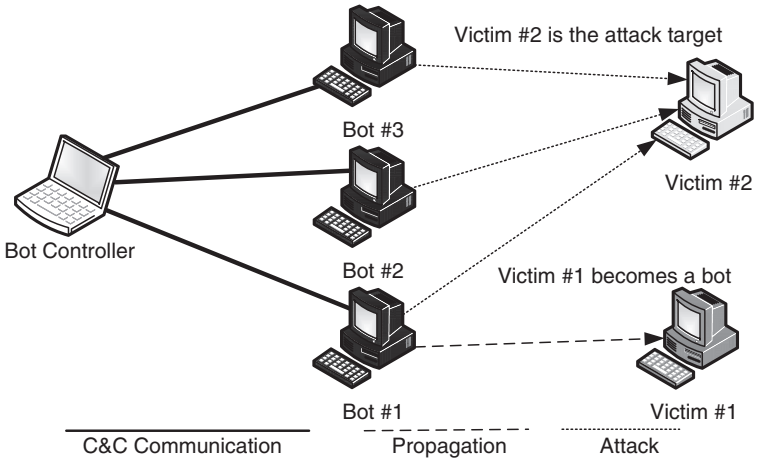

Figure 1. Botnet operations.

correspond to the attack traffic of a botnet. If a bot is put into a closed network environment for dynamic analysis, most of these network activities will not be properly exercised and observed.

2.2. Related work

There are two mainstream approaches for setting up a network environment for dynamic analysis. The first is to grant the malware full Internet access (i.e., open network environment). The malware should exhibit as much of its behavior in the analysis environment as if it were in the wild. An obvious issue with the open network approach is that the malware can freely attack machines on the Internet and cause damage. In contrast, the more widely embraced approach is to block all Internet access (i.e., a closed network environment) [8,16,17]. This approach is very secure and works well for analyzing malware that does not require network connectivity to the Internet. For malware that depends a lot on Internet connectivity, the closed network environment is very likely to miss most of the malware's runtime behavior.

There have been some works on the redirection of inbound attack traffic to honeypots for attack trace collection [18–22]. Their purpose is fundamentally different from ours, which is instead to provide a transparent and secure network environment to facilitate dynamic malware analysis.

Dynamic malware analysis and honeypot systems have relied on blacklists/whitelists of port numbers [21,23–25] to filter or redirect traffic that is potentially malicious. The GQ honeyfarm [25] employs port-based redirection, and some systems, such as the virtual honeyfarm [26], can also redirect traffic based on flow history and protocol types. However, it is very difficult to foresee the ports or the protocols used by a malware [27,28]. The flow history mechanism employed in [26] only allows outbound traffic related to previous connections (i.e., if a remote host had made a connection to the honeyfarm environment, outbound traffic to the host can be allowed). As a result, if a malware's command and control communication is initiated by the malware (an outbound traffic without any related connection flows), the communication traffic will be filtered, and the malware may not exhibit its full behavior. Instead of using redirection, the work [29] by G. Berger-Sabbatel drops all the packets that match the predefined classes of known attacks. This may disrupt a malware's execution and prevent its complete behavior from being observed during dynamic analysis.

Our environment can be seen as an extension to existing redirection systems in that our redirection mechanism leverages an IDS to automatically determine the portion of a malware's network traffic that should be redirected. This greatly eases the use of traffic redirection in dynamic malware behavior analysis. Our environment also provides the protocol synchronization mechanism to allow the transparent redirection of ongoing network connections. This allows the observation of malware's network behavior in a single round of experiment. In contrast, existing redirection systems require multiple rounds of trials and errors for setting up the redirection rules (e.g., first, redirect all traffic to the decoys regardless of whether the redirection would work, and then, manually allow certain traffic).

Dynamic malware analysis systems such as Berkeley's BitBlaze [30], GFI's Sandbox [31], and iSecLab's Anubis [32] are typically deployed in a closed network environment in favor of absolute network security over observability of malware's network behavior. Our environment is complementary to these systems, as our environment, by itself, does not provide dynamic malware analysis capability, and, on the other hand, our environment can help these systems be more effective on the analysis of malware whose operations require Internet access.

3. NETWORK TRAFFIC REPLAY, REDIRECT, AND RELAY IN DYNAMIC MALWARE ANALYSIS ENVIRONMENT

Many modern malware rely on the Internet to operate. They may use the Internet for propagation. Some of them can deliver attacks over the Internet. Some of them also leverage on the Internet for coordinating attacks (i.e., bots). For the dynamic analysis of modern malware to be effective, it is important for the analysis environment to meet two properties: "network transparency" and "network security." By network transparency, it means that a malware under analysis will have a transparent view of the whole network, notably the Internet. Otherwise, the malware may not exhibit all its behavior such as attack and propagation. By network security, it means that the analysis environment has to ensure that running the malware shall present no threat to the security of the Internet. In the following sections, we shall present an environment that is designed to achieve both network transparency and Internet security for dynamic malware analysis.

3.1. Approach overview

As mentioned earlier, the network traffic of a malware may consist of propagation, C&C communication, and attack traffic. To capture the full behavior of a malware during dynamic analysis, we would like the network traffic to be unobstructed, at least from the malware's perspective. On the other hand, we also want to make sure that the environment is secure so that the malware cannot cause damage to machines on the Internet.

We propose the secure and transparent network environment for dynamic malware analysis as shown in Figure 2. On the left-hand side is the Internet. On the right-hand side is the dynamic malware analysis environment, where the malware is executed and monitored for its runtime behavior such as network connections and system calls. The malware may attack machines (would-be victims)

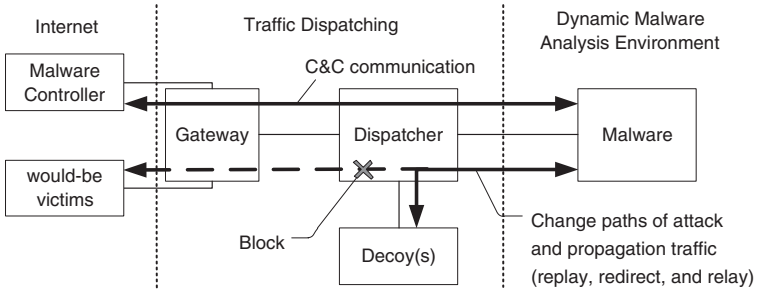

Figure 2. Secure and transparent network environment. C&C, command and control.

on the Internet. The malware may contact with a controller on the Internet (e.g., the C&C server of a botnet). At the core of the proposed environment is the traffic dispatching shown in the middle of Figure 2, which intercepts and reroutes the network traffic between the analysis environment and the Internet.

In the environment, propagation and attack traffic are transparently dispatched to decoys inside our system. Neither of the propagation nor the attack traffic ever reaches the Internet. Traffic dispatching is a three-phase process, which we shall explain in more details in Section 3.2. One important aspect of the traffic dispatching is that it needs be transparent to the malware. The malware should be barely aware of the traffic dispatching for the capture of the malware's full runtime behavior. We noticed that the propagation and attack traffic typically follow well-known protocols. Besides, they often exploit known vulnerabilities in widely deployed network services such as Network Basic Input/Output System (NetBIOS) and Simple Mail Transfer Protocol (SMTP). The strategy maximizes a malware's propagation and attack capability and is a very logical design choice in building malware. As a result, one can reasonably assume that both the propagation and attack traffic can be identified and, with some extra work (Section 3.2), be dispatched to the decoys. The decoys are machines running popular network services with vulnerabilities that will be subject to attacks by the malware during dynamic analysis.

Some malware (notably the bot) require C&C communication with a remote controller on the Internet to operate. As C&C communication protocol can be highly customized [23,33,34], it is generally impractical to assume that one can always find the proper decoy to emulate a controller or assume that C&C communication can be identified in a timely and convenient manner. Because of the aforementioned reasons, we made the design choice not to restrict or dispatch the C&C traffic. Although this might sound risky, our choice is actually based on the fact that C&C communication, by definition, does not carry attack effect in itself. Even if it does, the attack effect is contained within the connection between the malware and the C&C controller. At best, C&C communication may be used for stealing confidential or sensitive information from a victim machine running the malware. However, this is not an issue in our case, because the dynamic analysis environment would not contain any real sensitive information for a malware to steal.

3.2. Design of dispatcher

Figure 3 shows the design of the dispatcher. The dispatcher runs on a machine with three network interface cards

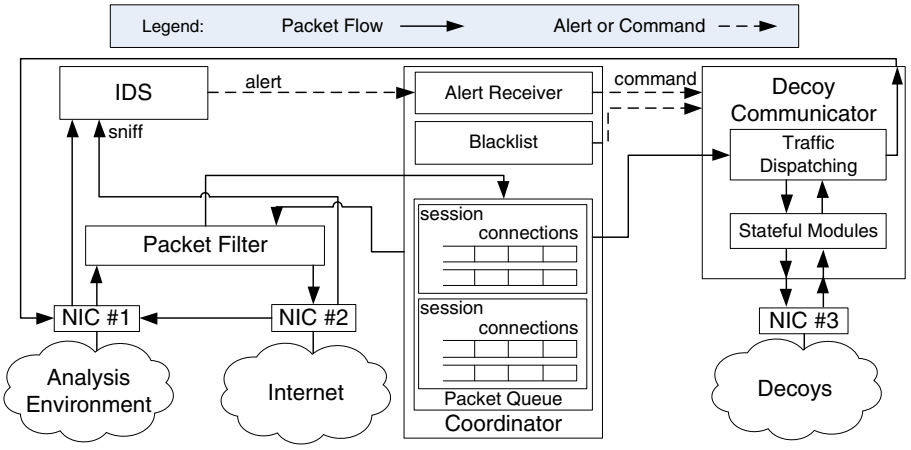

Figure 3. An overview of the dispatcher. IDS, intrusion detection system; NIC, network interface card.

(NICs): 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 the coordinator through the packet filter. The coordinator is in charge of the whole traffic dispatching process. It checks each packet against the alert receiver and the blacklist to see if traffic dispatching should be triggered for the corresponding network traffic. The alert receiver receives alerts from an IDS, which is set to inspect on the traffic from the analysis environment. If the IDS signals an alert, we shall treat the traffic as malicious, and the traffic will be dispatched to the decoys. Malicious traffic confirmed by the IDS can never reach the Internet. The blacklist consists of rules for traffic dispatching based on IP addresses and port numbers. For instance, we can blacklist the IP addresses and port numbers of popular SMTP servers (e.g., a.mx.mail.yahoo.com) to prevent the malware from actually sending spam mails through these servers. All the other traffic from the analysis environment will have Internet access via NIC #2.

Inbound traffic from the Internet will be forwarded to the analysis environment (via NIC#1) and also be inspected by the IDS. If malicious traffic is detected by the IDS, traffic dispatching will also be triggered for the corresponding session (details in the succeeding paragraphs). This is especially useful for certain IDS rules that are designed to match malicious patterns in the inbound traffic. For instance, Snort [35] rule 2924 looks in the inbound Server Message Block (SMB [36]) response messages to detect brute-force SMB logon attacks.

A copy of the traffic from the analysis environment is stored in the packet queue. The stored packets will later be used in the traffic dispatching process. Packets in the packet queue are grouped into sessions based on the source and destination IP addresses, so all the packets in a session will have the same source and destination IP addresses. Within each session, the packets are sub-grouped into connections by the source and destination port numbers, so all the packets in a connection will have the same source and destination port numbers plus the same source and destination IP addresses. Packets in each connection are sorted by their arrival times at the packet queue.

3.2.1. Traffic dispatching

During dynamic analysis, a malware may generate three types of traffic: attack, propagation, and miscellaneous such as C&C communication. For security purposes, we cannot allow attack or propagation traffic to reach the Internet. On the other hand, we also need to create an illusion (for the malware) that none of the traffic is blocked or filtered. This is achieved through the traffic dispatching process, which transparently reroutes Internet-bound attack and propagation traffic to the decoys. Traffic dispatching can be initiated in two situations. The first situation is when the IP address or port number of the first packet in a session is matched in the blacklist. All the packets in the session will be relayed to the decoy directly by changing the Media Access Control (MAC) and IP addresses in the packet headers. This is similar to the approach used in [20,21]. The second situation for traffic dispatching is when IDS, with some delay, signals an alert for an ongoing connection. Unlike the first situation, traffic dispatching in this case may occur at any time point in a session. Some connections in the session may have been closed, whereas some connections are still ongoing or about to be established. Packets in these connections cannot be relayed directly, as this would confuse the network protocol stacks. For instance, Transmission Control Protocol (TCP) stack will not accept packets in a partial connection without first seeing the packets of the three-way handshake for establishing that connection.

There is no guarantee that the IDS in the second situation may always detect the attack and propagation traffic by unknown malware. However, as unknown malware is likely to employ well-established exploits, the IDS still offers a layer of additional protection over existing port number-based redirection systems [20,21]. If attack traffic cannot be detected by IDS, manual inspection of the traffic by a human expert will be required, although such manual inspection may not be feasible in practice because of the high amount of unknown malware.

3.2.2. Replay, redirect, and relay

In our system, we have to use a three-phase dispatching process that involves traffic replay, redirect, and relay (Figure 4) to properly handle connections at various stages in a session.

First in the dispatching process is the traffic replay. When the dispatching process begins, some of the malware's network connections may have been closed. Packets in the closed connections may need to be replayed to the decoy, so the decoy can synchronize its states with the running malware. Note that we store all outgoing packets from the analysis environment in the packet queue in Figure 3. When the dispatching process begins, the coordinator will instruct the decoy communicator to replay the packets belonging to the closed connections from the packet queue to the decoy through NIC #3. Corresponding to the example in Figure 4, connections A and B will be replayed. During the replay, the decoy may generate response packets such as TCP ACK. All response packets from the decoy are filtered and not forwarded to the malware, because from the malware's perspective, these connections have been closed.

The second phase is traffic redirect. The phase deals with ongoing connections in a session when the dispatching process begins (e.g., connection C in Figure 4). Packets that have been sent and received (e.g., left portion

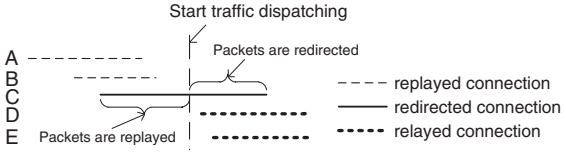

Figure 4. Three-phase traffic dispatching.

of connection C) are processed in the same way as in the replay phase. The decoy communicator takes the packets from the packet queue, adapts the packets for dispatching with the stateful modules (Section 4), and sends the packets to the decoy through NIC #3. During the replay part of traffic redirect, the response packets (e.g., TCP ACK) from the decoy are filtered. For subsequent packet transmission in an ongoing connection (e.g., the right portion of connection C), the decoy communicator will use the stateful modules to adapt the packets (i.e., adjust the sequence numbers) and then redirect the packets to the decoy. The sequence numbers in the response packets from the decoy will also be adjusted by the stateful module. The response packets will then be forwarded to the analysis environment through NIC #1. The adjustment of sequence numbers is necessary because the connection is still ongoing, and we need to ensure that the malware is unaware of the change in the traffic path.

The third phase is traffic relay. The phase changes the traffic paths for future connections in a session (i.e., connections that occur after the start of the traffic dispatching for the session). For instance, connections D and E in Figure 4 will be relayed in the dispatching process. The relay is carried out by modifying the MAC and IP addresses in the packet headers on the fly so that a relayed connection is effectively established between the malware and a decoy. Even though from the perspective of the malware, the connection is still with some victim machine on the Internet.

Through traffic dispatching, our environment transparently replays, redirects, and relays all the connections in a session. From the malware's point of view, the connections are still with some victim machines on the Internet. Although in reality, the underlying traffic has been dispatched to decoys in the secure environment. During the traffic dispatching, there may be states in the upper layer protocols that require additional processing. This is handled by the stateful modules in our system, which will be described in Section 4.

As mentioned earlier, in addition to relying on IDS to flag malicious sessions for traffic dispatching, we also use a blacklist to match known malicious traffic based on port numbers and IP addresses. For instance, an IDS may not have the signature for detecting email spam traffic. To prevent a malware from spamming the Internet, we maintain a blacklist of the IP addresses of well-known SMTP servers for dispatching spam mails from malware to the decoys.

3.3. Maintaining protocol states

Traffic dispatching works extremely well for stateless protocols, where the dispatched traffic will be valid for a decoy as long as the decoy has the corresponding services running on it. For stateful protocols, additional processing is required to ensure that the dispatched traffic is compatible with the protocol states on both ends (the decoy and the malware).

In general, when dispatching a connection, the decoy communicator has to ensure that each packet conforms to the protocol states on both ends (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 at each layer of protocols. For instance, we have a MAC stateful module at layer 2 to handle the rewriting of MAC addresses. We also have an IP stateful module at layer 3 for substituting IP addresses and recalculating IP checksums. At layer 4, we have a TCP stateful module to replace TCP sequence numbers, acknowledge numbers, and TCP checksums. For upper-layer protocols, we have implemented 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).

3.4. Example of traffic replay, redirect, and relay

Figure 6 shows an example of the traffic dispatching of a session in action. The session contains three connections A, B, and C. For the precise description of the critical time points in the traffic dispatching process, we further split each connection into sub-connections. For example, connection A is split into A1 representing the start of connection A and A2 representing the end of connection A. Connection B is split into four sub-connections B1, B2, B3, and B4. Here, B2 corresponds to the sending of packet M, and B3 corresponds to the receiving of packet N'. We use the apostrophe mark to signify a dispatched sub-connection. For instance, when connection A1 is replayed by the dispatcher to the decoy, the corresponding replayed connection is marked as A'1. Similarly, we have the redirected connection B'1 that corresponds to the connection B1 and the replayed connection C'1 for the connection C1.

The session begins when the malware makes connection A to the victim 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 connection B (bearing the same source and destination IP addresses as connection A) 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 a packet M containing some malicious payload that triggers an IDS alert. At this time, the dispatcher will flag the session as malicious and begin the traffic dispatching process for the session. Packet M and all subsequent packets bound for the Internet in the session will be dispatched to the decoy.

The first step in the traffic dispatching is to replay packets in connection A, which was finished before the start of the dispatching process. Connection A is replayed to the decoy as connection A'. Subsequently, connection B, which is still ongoing, has to be redirected. For those packets in connection B transmitted before packet M, they are essentially replayed as B'1 to the decoy. During the replay of A' and B'1, the decoy may generate corresponding response packets such as TCP ACKs. We ignore these

response packets from the decoy, because from the malware's point of view, the response packets had been received in connections A and B1. Subsequent transmissions of packets in connection B (i.e., B3 and B4) are essentially relayed between the decoy and the malware. Note that, for this part of connection B, we need to relay the response packet N due to the replayed packet M' back to the malware (i.e., B'3 → B3), or the ongoing connection B can get broken prematurely. Connection C is opened after the dispatching process begins, so it will be simply relayed in each direction by the dispatcher.

3.5. Network security and transparency

The network security guaranteed by the environment is that the outbound network traffic that matches a blacklist rule or triggers an IDS alert (Figure 3) will never reach the Internet. In practice, the blacklist rules may have loopholes and IDS may not detect unknown attack traffic, so the environment does not provide absolute network security. Later in Section 6, we will discuss why the environment is still very useful even with such a weakened network security guarantee.

The environment provides network transparency for a malware under analysis in two aspects. First, the traffic that does not match blacklist rules or trigger IDS alerts is untouched. The traffic can reach the Internet freely, and hence, from the perspective of the malware, the network environment is transparent for the traffic. Second, the traffic that matches a blacklist rule or triggers an IDS alert will be dispatched to the decoys. If the network protocol employed in the dispatched traffic has corresponding stateful modules and decoys, then the malware will not notice the dispatching of the traffic. That is, the malware should continue its follow-up activities as if the traffic had reached its designated targets on the Internet, so the network is transparent to the malware. In practice, one may only have the stateful modules and the decoys for a limited number of network protocols as in the case of the current prototype implementation (Section 4). Traffic without matching stateful modules or decoys can still be dispatched, but the dispatching will disrupt the traffic and interrupt the malware's operation.

4. IMPLEMENTATION

We set up a testbed environment as shown in Figure 2. The malware in the environment has to go through the dispatcher to reach the Internet. Through the dispatcher, the attack and propagation traffic from the malware are redirected to the decoys in the testbed environment. The implementation details of the dispatcher and the decoy are given in the following.

4.1.1. Dispatcher

The open-source implementation [1] of the dispatcher runs on a Linux-based personal computer with three NICs (NICs #1, #2, and #3) as shown in Figure 5. NIC #1 is connected to the analysis environment, and NIC #2 is connected to the Internet. NICs #1 and #2 are bridged together, so traffic from the Internet can reach the malware in the analysis environment directly when allowed by the dispatcher. We use Netfilter [37] to filter the traffic at NIC #1. Non-malicious traffic from the analysis environment is forwarded to NIC#2 through Netfilter. The dispatcher also keeps an eye on the traffic at NIC #1 through the libnetfilter_queue interface. A copy of the traffic at NIC #1 is stored in the packet queue (Figure 3) in case the traffic needs to be replayed. At the same time, the dispatcher relies on a blacklist and the IDS to detect if the traffic is malicious. When malicious traffic is detected at NIC #1, the decoy communicator will initiate the process of traffic dispatching for the corresponding traffic session.

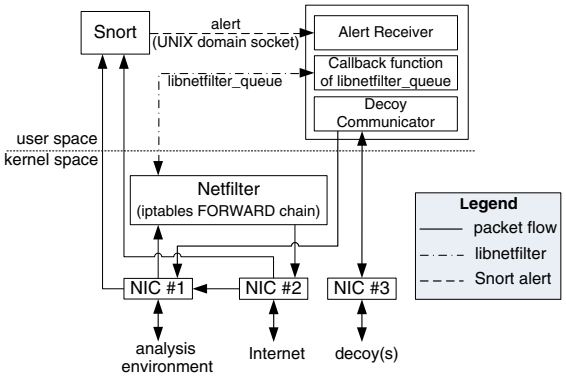

Figure 5. Implementation of dispatcher. NIC, network interface card.

The decoy communicator maintains a pool of threads for traffic dispatching (replay, redirect, and relay; see also Figure 6). For traffic replay, which deals with closed network connections, the decoy communicator just copies the payloads of the saved packets from the packet queue and uses standard socket to regenerate the packets for replay. For traffic redirect and relay, we use raw socket and the stateful modules to create the corresponding packets.

4.1.2. Stateful modules

Stateful modules are used to synchronize protocol states during traffic dispatching. We have implemented the stateful modules for MAC, TCP, SMB, and NT LAN Manager Security Support Provider (NTLMSSP [38]). The SMB stateful module rewrites the tree, process, user, and multiplex id fields in an SMB packet (Figure 7 on the fly during traffic redirect. The NTLMSSP stateful module is used to fix the protocol states during an SMB logon process.

As an example for showing the role of stateful modules in traffic dispatching, let us consider that a malware is attempting an SMB logon with a target victim machine through brute-force password guessing as shown in Figure 8. Assume that before the malware can successfully guess the victim's password, the IDS generates an alert (due to the excessive number of failed SMB logon attempts)

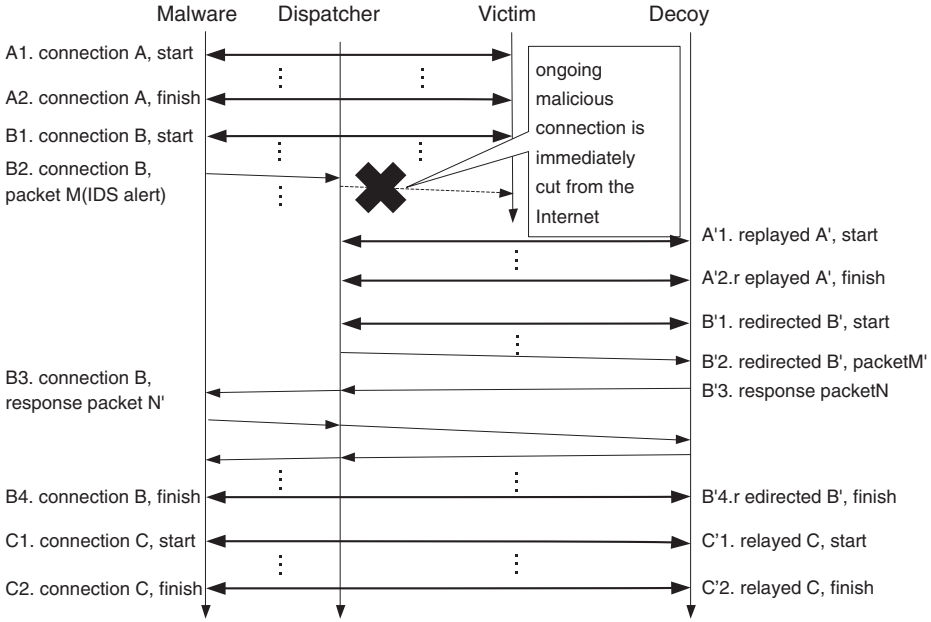

Figure 6. An example of traffic replay, redirect, and relay of connections in a malicious session.

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. Server Message Block packet format.

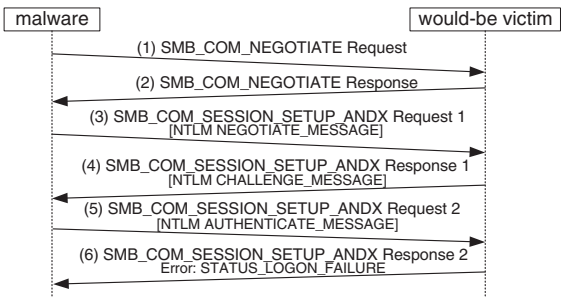

Figure 8. A Server Message Block (SMB) logon failure process.

and causes the dispatcher to initiate traffic dispatching for the connections in the corresponding session. Through the traffic dispatching, the ongoing SMB logon connection will be redirected to the decoy. Assume that the connection has three transmitted packets (1), (3), and (5), which are kept in the packet queue. Now, the dispatcher will first replay packet (1) SMB_COM_NEGOTIATE Request to the decoy. Then, the dispatcher will replay packet (3) SMB_COM_SESSION_SETUP_ANDX Request, for which the decoy will reply with an NTLM_CHALLENGE_MESSAGE (similar to packet (4) but with a different nonce value). If the dispatcher continues to replay packet (5), the logon process will fail, as the response in packet (5) is only good for the original challenge nonce in packet (4). In this case, the NTLMSSP stateful module is used to recalculate the corresponding response value for the new challenge nonce, so the redirected logon process may proceed correctly.

4.1.3. Decoy

The implementation of the decoy is a (virtual) machine loaded with operating system and network services treated as the targets for the dispatched attack and propagation traffic. In our implementation, we have a decoy acting as an SMTP server and decoys running vulnerable SMB and NetBIOS services. Services on the decoy may require extra settings to ensure the malware cannot cause damage to the Internet via them. For instance, the SMTP decoy is set to silently skip the delivery of outgoing emails to the Internet.

5. EXPERIMENT STUDIES

For evaluation, we compared the proposed secure and transparent network environment with both the closed network environment and the open network environment (Figure 9). The comparisons are based on dynamic analysis results from running 12 real-world malware samples in each of the network environments. The selection criteria for malware samples and the setup of the experiment environment are presented in Section 5.1. The comparison

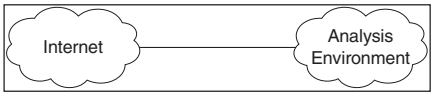

Figure 9. Open network.

on network transparency with respect to each of the three environments is given in Section 5.2. The comparison on network security is given in Section 5.3. Two case studies are provided in Section 5.4 and Section 5.5 to give more details about the operation of the secure and transparent network environment.

5.1. Sample selection and experiment environment

We collected more than 2000 suspicious malware samples from different sources including peer-to-peer file sharing, email attachments, phishing websites, and Nepenthes honeypots [39]. As the focus of this work is about achieving network security and network transparency in dynamic malware analysis environment, we concentrated only on those samples that do exhibit malicious network behavior. Therefore, we scanned the samples with four different anti-virus scanners and kept only those flagged by all the scanners. This resulted in a total of 124 samples. Next, we executed each of the remaining 124 malware samples and waited for 2 min to see if they exhibit any network activities. We removed those samples that exhibit no network activity at all. We also removed those samples that could neither establish a successful TCP connection nor receive any incoming packet transmission from a remote server. In the end, we had a selection of 12 malware samples as shown in Table I. The 12 malwares were separated into two groups: malware without C&C and malware with C&C. The discovery time is based on [3].

The first group, malware without C&C, consists of three worms and one email spammer. The worms (m10.exe, m11.exe, and m12.exe) propagate by brute-force attack on weak SMB logon password. After a successful logon, the worm binaries are copied to the target machine and get executed. The email spammer (m7.exe) propagates by sending spam emails with copies of its binary "Worm/NetSky.P" in the attachment.

The second group corresponds to malware with C&C communication. In this group, we have two spammers (m8.exe and m9.exe) that communicate with C&C servers for retrieving updated versions of spam mail contents and recipient lists. We also have botnet agents (m1.exe, m2.exe, m3.exe, m4.exe, m5.exe, and m6.exe), which connect to C&C servers awaiting commands for follow-up attack or propagation actions. We observed that upon receiving commands from the C&C server, these malwares would start scan machines randomly on the network (both LAN and Wide Area Network addresses) and attack machines running vulnerable NetBIOS services.

The samples in Table I also confirmed our assumption that the attack and propagation traffic employed by new malware are not necessarily as new. For instance, the malware m1.exe and m4.exe that were discovered in 2010 still relied on the same SMB password guessing attack as had been employed by m2.exe in 2004. Although it is indeed impossible to guarantee that an IDS can detect new types of attack traffic, we believe that in practice, it still offers a stronger protection over purely port number-based redirection systems, which will not be able to distinguish attack traffic that slips through a non-blacklisted port.

We set up an experiment environment according to the architecture in Figure 3. The high-level network topology of the secure and transparent network environment is shown in Figure 10. We also set up a closed network environment (Figure 11) and an open network environment (Figure 9) for comparison. The decoys are machines loaded with vulnerable network services.

Table I. Selected samples.

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

C&C, command and control; SMB, Server Message Block.

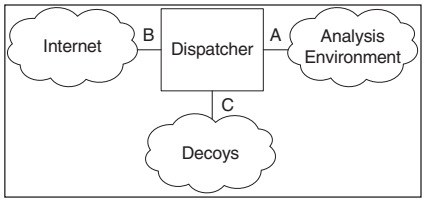

Figure 10. Secure and transparent network (our environment).

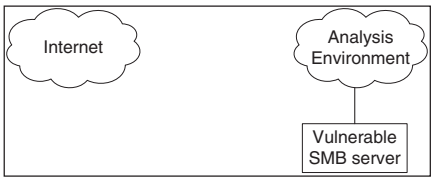

Figure 11. Closed network. SMB, Server Message Block.

5.2. Effectiveness of transparent network environment

In this part of the experiment, we evaluate the network transparency by studying the network traffic collected from running malware in each of the three environments. Each malware was executed in an analysis environment for 10 min. We use TCPDUMP [40] to record the network traffic.

Table II summarizes the observed network activities from running malware without C&C in the closed network environment and in our secure and transparent network environment. In the closed network environment, the majority of the packets are TCP SYNs, as connections to the outside world are blocked. On the other hand, we saw many more network activities in our environment. For instance, we observed that m7.exe attempted to initiate SMTP connections for sending spam emails, and malware m10.exe generated many more traffic on ports 139 and 445 (369199 packets) compared with the result from the closed network environment (707 packets). We were also able to observe the hypertext transfer protocol (HTTP) connections made by m10.exe in our environment. In fact, the network traffic from m10.exe triggered the Snort alert "NetBIOS SMB-DS repeated logon failure," and it was the subsequent traffic dispatching that allowed us to have a more thorough view of the network behavior by m10.exe. The situations with m11.exe and m12.exe are similar to m10.exe. The spam mail content from m7.exe can be completely captured with our environment if we provide a proper SMTP decoy (details given in Sections 5.3 and 5.4).

Whereas none of the malware in Table II involved C&C communication, we found that they still require Internet connections. For instance, m7.exe requires Internet connection for sending spam mails, and m10.exe, m11.exe, and m12.exe all made HTTP connections to advertising sites on the Internet. If we execute these malwares in the closed network environment, none of these behaviors can be observed. This shows the importance of network transparency in dynamic malware analysis.

The results for malware with C&C are shown in Table III. Most of the network traffic in the closed network environment corresponds to unsuccessful connection attempts (i.e., TCP SYNs). On the other hand, many more network activities were observed in the secure and transparent network environment. We successfully observed that m4.exe connected to an Internet Relay Chat server on port 47221 and downloaded the malicious payload "TR/Kazy.15451.21" via a separate HTTP connection. Soon after that, m4.exe began to scan for vulnerable machines (on both LAN and WAN) at port 445 for propagation. Malwares m8.exe and m9.exe are two spammers that also involved C&C communication. They connected to C&C servers on port 80 to download the latest spam content and the recipient list. In the secure and transparent network environment, we could observe the complete process of C&C communication and capture the spam emails sent by these two malwares.

5.2.1. Open network is not always more transparent

We noticed that in some situations, the secure and transparent network environment could actually show more behaviors about a malware than the open network environment. This happened when a malware depended on some network service on the Internet that was no longer functioning but could be simulated by a decoy. The secure and transparent network environment could help dispatch

Table II. Network activities by malware without C&C.

Malware	Closed network	Our environment
m7.exe	No response for DNS MX record.	Nine spam email attempts.
m10.exe	707 packets for TCP ports 139 and 445 (707 incomplete connections).	369 199 packets for TCP ports 139 and 445 (102 connections). HTTP traffic to advertising websites.
m11.exe	795 packets for TCP ports 139 and 445 (792 incomplete connections).	23 161 packets for TCP ports 139 and 445 (100 connections). HTTP traffic to advertising websites.
m12.exe	Probe machines by ICMP echo request.	Probe machines by ICMP echo request. 60 285 packets for TCP ports 139 and 445 (24 connections). HTTP traffic to advertising websites.

C&C, command and control; TCP, Transmission Control Protocol; HTTP, hypertext transfer protocol; ICMP, Internet Control Message Protocol.

Table III. Network activities by malware with C&C.

Malware	Closed network	Our environment
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	No response for DNS A query.	TCP C&C connection (74.117.174.122:16667)
m5.exe		TCP SYN at port 445 after receiving ".advscan asn445 100 5 0 -b -r -s" command
m6.exe		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 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 at port 139 34 spam emails
m9.exe	No response for DNS MX query.	TCP C&C connection (208.77.45.146:80) 179 spam emails

C&C, command and control; TCP, Transmission Control Protocol; HTTP, hypertext transfer protocol.

traffic to the decoy and allowed the malware to continue its execution. For example, we observed that the SMTP servers used by m7.exe for sending email spams no longer accepted SMTP connections from the malware anymore. With the secure and transparent network environment, we were able to set up relay rules so that SMTP connections from the analysis environment were transparently dispatched to an SMTP decoy. By doing so, we captured 14 spam emails sent by m7.exe during the experiment. In contrast, zero spam emails were captured for m7.exe in the open network environment as shown in Table IV.

Table IV. Number of spam emails sent.

Malware	Our environment	Open network	Improvement rate (%)
m7.exe	14	0	N/A
m8.exe	117	68	72.06
m9.exe	118	70	68.57

N/A, not applicable.

The other two spammer malwares m8.exe and m9.exe both targeted Yahoo email accounts. Some of their SMTP connections were blocked by the anti-spam mechanism of Yahoo [41]. If we use the secure and transparent network environment together with the SMTP decoy, all the spam emails could be successfully delivered and captured.

5.3. Effectiveness of secure network environment

We evaluate the security of our environment by scrutinizing the Internet-bound traffic (traffic at point B in Figure 10) in the experiments of Section 5.2. If the environment is secure, the Internet-bound traffic should contain no attack or propagation traffic that may damage machines on the Internet.

As shown in Table V, we observed that the only attack or propagation traffic leaked to the Internet was part of the SMB password guessing attack traffic (i.e., m10.exe, m11.exe, and etc.) and email spamming traffic (i.e., m8.exe and m9.exe). The reason why the SMB password guessing traffic could reach the Internet is because the IDS rule (Snort

Table V. Leaked attack or propagation traffic from the secure and transparent environment.

Malware	Leaked attack and propagation traffic	Dispatched attack and propagation traffic
m7.exe	None. (SMTP server is no longer working.)	Spam emails with malware in the attachment
m1.exe, m4.exe, m5.exe,	SMB password guessing	SMB password guessing
m6.exe, m11.exe, m12.exe		Transfer of malware binary via SMB
m2.exe, m3.exe	SMB password guessing	SMB password guessing Shellcode injection
m10.exe,	SMB password guessing	SMB password guessing Transfer of malware binary via SMB Shellcode injection
m8.exe, m9.exe	Several spam emails are delivered to Yahoo mail server.	Spam emails

SMTP, Simple Mail Transfer Protocol; SMB, Server Message Block.

rule 2924) for detecting the attack has a threshold value. The rule is triggered only when there are more than 10 failed SMB logon attempts in 60 s. For the email spamming traffic, it was because Snort did not have a corresponding rule for detecting it. In fact, the spam mails sent by m8.exe and m9.exe are just plain text mails, which carry no attachments. However, it is possible that the body text of the mail could carry messages that may be used for social engineering or phishing attacks. In the future, we can integrate a spam mail filter into the IDS infrastructure to further enhance the security of our environment for this particular type of traffic.

From Table V, we can see that the secure and transparent network environment was able to dispatch the SMB password guessing traffic early on (after about 10 failed logon attempts) and prevented the propagation of malware binary to the Internet. However, if a victim machine had used a weak SMB password, hypothetically the malware might have been able to break into the victim machine before the traffic dispatching kicked in. On the other hand, for m7.exe, if a working SMTP server had been present, hypothetically the malware could have also succeeded in its propagation beyond the border of our environment.

To show that our environment is secure even in these hypothetical situations, we extended the experiment testbed with an SMB server that takes empty logon password and an SMTP server that unconditionally accepts and relays emails from m7.exe as shown in Figure 12. We redid the experiment and found that the SMB password guessing attack could succeed within three attempts, and Snort rule 2924 was never triggered. However, when malware started transmitting its executable binary through the established SMB connection, Snort immediately raised alert "NetBIOS SMB-DS ADMIN$ Unicode share access" (Snort rule 2473), and the connection was dispatched right away. For m7.exe, the malware could succeed in establishing connection with the SMTP server. However, once the malware started sending its malicious payload as the email attachment over the SMTP connection (details about the payload is described in Section 5.4), Snort immediately raised alert "SHELLCODE x86 inc ecx NOOP" (Snort rule 1394), and the SMTP connection was dispatched before the SMTP server could finish receiving the email with the payload.

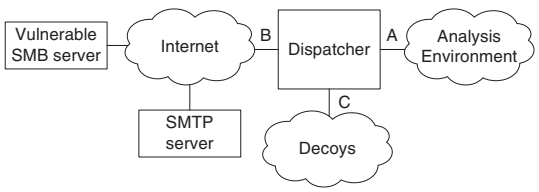

Figure 12. Test environment with vulnerable Server Message Block (SMB) server and working Simple Mail Transfer Protocol (SMTP) server for public relay.

5.4. Case study: running m7.exe

In Section 5.3, m7.exe queried Domain Name System Mail Exchanger (DNS MX) records to look for SMTP servers (Figure 13) for sending spam emails. We spoofed the DNS records to trick m7.exe into using the SMTP server that we set up. Traffic dispatching occurred during the sending of spam email by m7.exe when Snort rule 1394 was triggered (Figures 14 and 15). This caused the dispatcher to initiate the three-stage traffic dispatching process (replay, redirect, and relay) on the SMTP session. The spam email never reached the Internet. On the other hand, the transparent traffic dispatching allowed the decoy to capture the entire content of the spam email. We were able to determine from the captured content that the email comes with a copy of the malware itself Worm/NetSky.P in the attachment.

5.5. Case study: stateful module and traffic dispatching

The use of stateful modules to rewrite MAC address (layer 2), IP address (layer 3), and TCP ACK number (layer 4) is essential for transparent traffic dispatching. However, the use of stateful modules for upper-layer protocols may not be as obvious. In this case study, we shall give an example showing the need for layer 7 stateful module in traffic dispatching.

One example where layer 7 stateful module was needed is the SMB protocol (a layer 7 protocol) used by some of the malware in our experiments (Table I). If there was no layer 7 stateful module, generally the traffic dispatching

```
 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 14. Simple Mail Transfer Protocol (SMTP) session by m7.exe during sending spam email.

```
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 13. DNS queries by m7.exe.

```
[**] [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 15. Snort alert for m7.exe.

would not succeed: The malware would not be able to propagate to or cause damage to the decoy. Pertaining to our environment, the reason is twofold: On one hand, such a malware tends to start SMB logon attempts with weak passwords such as the blank password first. Fortunately, these weak passwords did not work for the victim in our experiments and would result in a series of failed logon attempts. This eventually triggered the Snort alert "NetBIOS SMB-DS repeated logon failure" followed by the traffic dispatching. From the malware's perspective, none of the logon attempts thus far succeeded. On the other hand, when the SMB connections were replayed and redirected to the decoy, the SMB logon request that used the blank password (exactly what we set for the decoy) would succeed in the logon process with the decoy. This resulted in an inconsistency in the SMB protocol states between the malware and the decoy.

The situation did not get any better even in the relay phase. In the relay phase, the malware would continue with its SMB logon attempts with the decoy (via the relayed connections). However, the malware was smart enough not to use the same password more than once, so it could never succeed in the logon process as the decoy was expecting the blank password, which the malware had used once long ago (Figure 16).

In the end, we needed to rely on a layer 7 module to resolve the inconsistency in the SMB protocol states. In the current implementation, the layer 7 stateful module acted as a proxy for the relayed SMB connections between the malware and the decoy. The module replaced the challenge-response fields in the SMB packets so that the malware could logon to a decoy with arbitrary password through a relayed SMB connection. With the help of the layer 7 stateful module, we were able to capture the propagation of malware m10.exe (being injected as Games.exe onto a victim machine as shown in Figure 17) and the scheduler registration activity on the victim machine that was used to invoke the injected malware binary as shown in Figure 18.

6. LIMITATION AND DISCUSSION

The use of IDS for distinguishing attack traffic from benign traffic may be regarded as insecure because no existing IDS can guarantee the detection of unknown

```
1 NT Create AndX Request, FID: 0x4001, Path: \Games.exe
2 NT Create AndX Response, FID: 0x4001
3 Trans2 Request, QUERY_FILE_INFO, FID: 0x4001, Query File Internal Info
4 Trans2 Response, FID: 0x4001, QUERY_FILE_INFO
5 Trans2 Request, QUERY_FS_INFO, Query FS Attribute Info
6 Trans2 Response, QUERY_FS_INFO
7 Trans2 Request, SET_FILE_INFO, FID: 0x4001
8 Trans2 Response, FID: 0x4001, SET_FILE_INFO
9 [TCP segment of a reassembled PDU]
```

Figure 17. Transfer the malware binary via Server Message Block.

```
146 JobAdd request
147 Tree Disconnect Request
148 445 > 1073 [ACK] Seq=3172 Ack=73164 Win=62965 Len=0
149 Tree Disconnect Response
150 JobAdd response
```

Figure 18. Using "at" scheduler.

attacks. Unknown malware running in the proposed environment can possibly evade traffic redirection and attack machines on the Internet. However, we believe the risk is well worth for the timely dynamic analysis of malware binaries. On one hand, if an unknown malware collected from the Internet can evade IDS detection, there is really no good reason why the proposed environment would make the Internet less secure, as the unknown malware most likely has been rampaging through the Internet. On the other hand, we believe that unknown malwares are likely to employ a known attack because, as a matter of fact, the volume of new malware is obviously growing at a much higher rate than the volume of new zero-day exploits. An IDS can be handy in dealing with these malware.

Anomaly-based IDS can also be applied to the environment to help detect attack and propagation traffic by unknown malware. However, anomaly-based IDS tend to have a high false-positive detection rate. False positives will cause unnecessary traffic dispatching and will increase the chance of traffic without corresponding stateful modules or decoys to be dispatched.

The evaluation only employs a limited number of malware samples, as the focus of the evaluation is to demonstrate the feasibility of transparent traffic dispatching and the use of IDS in the dispatching process. For a large-scale evaluation against a sizeable volume of malware samples, we need to implement the stateful modules for all well-known protocols (current prototype only implements MAC, TCP, SMB, and NTLMSSP) and the corresponding decoys. However, the dispatching process will just be the same, and the samples used in the evaluation suffice to demonstrate the validity of the proposed environment.

```
4 Negotiate Protocol Request
5 Negotiate Protocol Response
6 Session Setup AndX Request, NTLMSSP_NEGOTIATE
7 Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED
8 Session Setup AndX Request, NTLMSSP_AUTH, User: BB-UGOIOR2EBSR3\Administrator
9 Session Setup AndX Response, Error: STATUS_LOGON_TYPE_NOT_GRANTED
```

Figure 16. Server Message Block (SMB) logon packets (no layer 7 stateful module).

7. CONCLUSION AND FUTURE WORK

Dynamic malware analysis is conventionally performed in a closed network environment without Internet connection. This effectively prevents the malware under analysis from attacking machines on the Internet. However, for malware that depends on Internet connections to operate, a closed network environment would defeat the purpose of dynamic analysis, as much of the malware's network behavior will not be manifested and get analyzed.

We propose the secure and transparent network environment that allows a malware under dynamic analysis to manifest most of its network behavior. Non-malicious control traffic is allowed access to the Internet, whereas malicious attack and propagation traffic are dispatched to decoys within the analysis environment.

To ensure that the traffic dispatching is transparent to the malware, network connections at different stages have to be handled in different ways. We devise a three-stage process that involves traffic replay for closed connections, traffic redirect for ongoing connections, and traffic relay for subsequent new connections.

The evaluation result shows that our system significantly increases the amount of observed network activities during dynamic malware analysis compared with the closed network environment. In some situations, the use of traffic dispatching and decoys in our system can even improve the effectiveness of dynamic analysis beyond what an open network environment can offer. Except for a few spam mails with advertising contents, our system successfully dispatches all outbound malicious traffic to decoys throughout the evaluation. The leakage of spam mails can be addressed by setting up dispatching rules based on protocol types (i.e., dispatch all SMTP traffic by default) or employing an anti-spam filter as part of the IDS infrastructure. Both are items we plan to pursue as part of the future works.

The widespread use of mobile devices including smartphones and tablets has created a new venue for malware [42]. Some of these malware also involve Internet connections in their attack vectors [42]. Our environment can be used to redirect the attack and propagation traffic sent from the Wi-Fi interface of a mobile device provided that corresponding decoys and stateful modules are in place. However, mobile devices can also send traffic through the telephony network (e.g., the Long-Term Evolution network), which is currently not supported by the environment. As a future work, we will investigate on possibilities for extending the environment to support the telephony network interface of mobile devices.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Council (NSC) and the National Center of High-Performance Computing (NCHC) in Taiwan.

REFERENCES

1. Network Benchmarking Lab. BotMasquerader, 2012/12/19. Available: http://sourceforge.net/projects/botmasquerader/files/
2. Symantec. Threat Explorer—Spyware and Adware, Dialers, Hack tools, Hoaxes and other risks, 2011/9/16. Available: http://www.symantec.com/business/security_response/threatexplorer/
3. Kaspersky. Kaspersky Securelist, 2011. Available: http://www.securelist.com/en/find
4. Szor P. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional: Boston, USA, 2005.
5. Bayer U, Moser A, Kruegel C, Kirda E. Dynamic analysis of malicious code. *Journal in Computer Virology* 2006; **2**:67–77.
6. Collberg C, Thomborson C, Low D. *A Taxonomy of Obfuscating Transformations*. Department of Computer Science, The University of Auckland: New Zealand, 1997; 1173–3500.
7. Willems C, Holz T, Freiling F. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE* 2007; **5**:32–39.
8. Bayer U, Kruegel C, Kirda E. TTAnalyze: a tool for analyzing malware. In *15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, 2006.
9. Greamo C, Ghosh A. Sandboxing and virtualization: modern tools for combating malware. *Security & Privacy, IEEE* 2011; **9**:79–82.
10. Chen X, Andersen J, Mao ZM, Bailey M, Nazario J. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Dependable Systems and Networks*. IEEE/IFIP: Anchorage, Alaska, USA, 2008; 177–186.
11. Puri R. Bots & botnet: an overview. In *SANS Institute*. SANS Institute: Bethesda, Maryland, 2003.
12. Barford P, Yegneswaran V. An inside look at botnets. In *Malware Detection*. Springer: Berlin, Germany, 2007; 171–191.
13. Carpenter M, Liston T, Skoudis E. Hiding virtualization from attackers and malware. *Security & Privacy, IEEE* 2007; **5**:62–65.
14. Crandall JR, Wassermann G, de Oliveira DAS, Su Z, Wu SF, Chong FT. Temporal search: detecting hidden malware timebombs with virtual machines. *ACM SIGPLAN Notices* 2006; **41**:25–36.
15. Dagon D, Gu G, Lee C, Lee W. A Taxonomy of Botnet Structures. *IEEE Annual Computer Security Applications Conference*. Miami Beach, Florida 2007.
16. Norman. Norman Sandbox, 2011. Available: http://www.norman.com/security_center/security_tools/
17. Stewart J. The Reusable Unknown Malware Analysis Net, 2011. Available: http://www.secureworks.com/cyber-threat-intelligence/tools/truman/

18. Kim M, Mun Y. Design and implementation of the honeypot system with focusing on the session redirection. In *Computational Science and Its Applications–ICCSA 2004*. Springer: Berlin, 2004; 262–269.
19. Kim I, Kim M. The DecoyPort: redirecting hackers to honeypots. *Network-Based Information Systems*. IEEE: Regensburg, Germany, 2007:59–68.
20. Alberdi I, Alata E, Nicomette V, Owezarski P, Kaâniche M. Shark: Spy honeypot with advanced redirection kit. In *IEEE Workshop on Monitoring, Attack Detection and Mitigation (MonAM'07)*, 2007; 47–52.
21. Alata É, Alberdi I, Nicomette V, Owezarski P, Kaâniche M. Internet attacks monitoring with dynamic connection redirection mechanisms. *Journal in Computer Virology* 2008; **4**:127–136.
22. John JP, Moshchuk A, Gribble SD, Krishnamurthy A. Studying spamming botnets using Botlab. In *Symposium on Networked System Design and Implementation*, 2009.
23. Grizzard JB, Sharma V, Nunnery C, Kang BBH, Dagon D. Peer-to-peer botnets: overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007; 1–1.
24. Ion A. *Malicious Traffic Observation Using a Framework to Parallelize and Compose Midpoint Inspection Devices*. Universite de Toulouse, 2010.
25. Kreibich C, Weaver N, Kanich C, Cui W, Paxson V. GQ: practical containment for measuring modern malware systems. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, 2011; 397–412.
26. Vrable M, Ma J, Chen J, *et al*. *ACM SIGOPS Operating Systems Review*. Scalability, fidelity, and containment in the potemkin virtual honeyfarm ACM: Brighton, UK, 2005; 148–162.
27. Comparetti PM, Wondracek G, Kruegel C, Kirda E. Prospex: protocol specification extraction. In *Symposium on Security and Privacy*, 2009; 110–125.
28. Cho CY, Shin ECR, Song D. Inference and analysis of formal models of botnet command and control protocols. In *Computer and Communications Security*. ACM: Chicago, IL, USA, 2010; 426–439.
29. Berger-Sabbatel G, Duda A. Analysis of Malware Network Activity. In *Multimedia Communications, Services and Security*. Springer: Berlin, Germany, 2011; 207–215.
30. Song D, Brumley D, Yin H, *et al*. BitBlaze: a new approach to computer security via binary analysis. In *Information systems security*. Springer: Berlin, Germany, 2008; 1–25.
31. GFI.com. Malware analysis with GFI Sandbox, 2013/1/14. Available: http://www.gfi.com/malware-analysis-tool
32. International Secure Systems Lab. Anubis: analyzing unknown binaries, 2013/1/14. Available: http://anubis.iseclab.org/
33. Starnberger G, Kruegel C, Kirda E. Overbot: a botnet protocol based on Kademlia. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, 2008; 13.
34. Chiang K, Lloyd L. A case study of the rustock rootkit and spam bot. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007; 10–10.
35. Snort. Snort, 2011. Available: http://www.snort.org/
36. SMB Protocol, 2011. Available: http://www.protocols.com/pbook/ibm.htm
37. Netfilter. Netfilter, 2011. Available: http://www.netfilter.org/
38. Microsoft. NT LAN Manager (NTLM) Authentication Protocol Specification, 2011. Available: http://msdn.microsoft.com/en-us/library/cc236621.aspx
39. Baecher P, Koetter M, Holz T, Dornseif M, Freiling F. The nepenthes platform: an efficient approach to collect malware. In *Recent Advances in Intrusion Detection*. Springer: Hamburg, Germany, 2006; 165–184.
40. TCPDUMP, 2011. Available: http://www.tcpdump.org/
41. Yahoo. Yahoo 421 SMTP Error Code, 2011. Available: http://help.yahoo.com/kb/index?page=content&y=PROD_MAIL_ML&locale=en_US&id= SLN3434
42. Felt AP, Finifter M, Chin E, Hanna S, Wagner D. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011; 3–14.