IEICE TRANS. COMMUN., VOL.E90–B, NO.10 OCTOBER 2007

PAPER Special Section on New Challenge for Internet Technology and its Architecture

Detecting and Guarding against Kernel Backdoors through Packet Flow Differentials

Cheolho LEE[†a)], *Nonmember and* **Kiwook SOHN**[†], *Member*

SUMMARY In this paper, we present a novel technique to detect and defeat kernel backdoors which cannot be identified by conventional security solutions. We focus on the fact that since the packet flows of common network applications go up and down through the whole network subsystem but kernel backdoors utilize only the lower layers of the subsystem, we can detect kernel backdoors by employing two host-based monitoring sensors (one at higher layer and the other at lower layer) and by inspecting the packet flow differentials. We also provide strategies to mitigate false positives and negatives and to defeat kernel backdoors. To evaluate the effectiveness of the proposed technique, we implemented a detection system (*KbGuard*) and performed experiments in a simulated environment. The evaluation results indicate that our approach can effectively detect and deactivate kernel backdoors with a high detection rate. We also believe that our research can help prevent stealthy threats of kernel backdoors.

key words: *kernel-mode backdoors, rootkits, backdoors, network monitoring*

1. Introduction

The explosive growth of the Internet helps people easily access remote information by clicking on the mouse. Meanwhile, it also increases the possibility that someone steals others' information. In fact, many attackers have stolen sensitive information using the backdoor which can enable the attackers to access compromised systems without legitimate authentication [1], [2], [21]. Moreover, today's backdoor shows its characteristics for evading Intrusion Detection Systems (IDSs) and hiding not only itself but also its communication channels [12], [13], [24] as shown in cases of Backdoor-ALI [14], SAdoor [18], yyt_hac's ntrootkit [25] and NTrootkit [8], [9].

The backdoor can run in either user-mode or kernel-mode in the view of its running level [17]. A user-mode backdoor can be easily detected by traditional anti-virus solutions or IDSs because it usually opens a specific network port and waits for the access from the attackers (passive open mode). On the other hand, a kernel-mode backdoor (kernel backdoor, also known as a kernel-mode Trojan or a rootkit) cannot be detected by those conventional means because it runs in kernel-mode as a component of network subsystem. In addition, since the kernel backdoor runs in privileged mode, it is so powerful that it can completely control the compromised systems [2], [4]. Many security experts believe that intruders have been using kernel backdoors covertly for years and the lack of them captured in the wild is a reflection of their effectiveness [1], [2], [21]. If they spread all over the Internet by means of self-propagating malicious code such as worms, every aspect of human activities including businesses, economics, and military affairs will be faced with world-wide stealthy threats of kernel backdoors [21].

In this paper, we present a novel technique for detecting and defeating kernel backdoors. The rest of this paper is organized as follows. In the next section, we will mention other researches for detecting user-mode backdoors as well as kernel backdoors. In Sect. 3, we explain the structure of network subsystem and the operation of backdoors. We propose a technique to detect and defeat kernel backdoors and describe its details in Sect. 4. Section 5 then shows experimental results and further considerations. Finally, we summarize our conclusions and future work in Sect. 6.

2. Related Work

Many research efforts [10], [19], [26], [27] including one commercial product [23] have focused on detecting and protecting against kernel backdoors by investigating traffic interactiveness, unintended network connections, changes of system resources, or unauthorized drivers.

Cui et al. proposed a mechanism to detect unknown malwares by identifying user unintended outbound connections [27]. They classified outbound network connections into three classes: user intended, user unintended benign, and user unintended malicious. In their approach, since they assume that user intent is implied by user-driven activities, network connections without user-driven activities can be detected as malicious ones. They used a whitelist to cover system daemons or applications automatically connecting to the remote machines. Their approach is very effective to detect unknown malwares and to detect kernel backdoors as well. However, they considered only outbound connections and focused on the whitelisting mechanism to cover unintended benign connections. On the other hand, our technique considers both inbound and outbound connections and has no whitelisting mechanism. Considering the fact that the whitelist is difficult to manage in various network settings and kernel backdoors may employ both inbound and outbound connections, our technique is more effective than their approach as far as kernel backdoors are concerned.

Zhang and Paxson developed a general algorithm and a set of protocol-specific algorithms to detect interactive traf-

Manuscript received January 29, 2007.
Manuscript revised April 23, 2007.

[†]The authors are with Electronics and Telecommunications Research Institute (ETRI), Daejeon, 305-700 Republic of Korea.
a) E-mail: chlee@etri.re.kr

DOI: 10.1093/ietcom/e90–b.10.2638

fic originated from backdoors through passively monitoring packet size and timing characteristics [26]. They focused on the fact that interactive traffics have smaller packet size and longer idle period than most machine-driven traffics. However, this approach may not be effective because many current backdoors can employ automated machine-driven traffics.

Pennington et al. introduced a mechanism for detecting backdoors by monitoring the changes of storage contents [19]. In their mechanism, if system files are changed or hidden files are found, it is regarded as the existence of malicious behaviors including backdoors, worms, rootkits, and so on. However, this approach may not be useful against kernel backdoors in the sense that it is not easy to detect the files hidden by kernel-level rootkits.

Sanctuary Device Control by SecuWave can prevent unauthorized devices from being installed or running on the system [23]. It strongly permits or blocks devices through the device whitelist which ensures that no device, unless authorized, can ever be used. However, since most of benign users rarely know which devices should be authorized or unauthorized, this method is very strong but not applicable to the real field.

Rutkowska proposed a mechanism to detect rootkits by looking for the witnesses of API hooking or changes in kernel structures such as SST (System Service Table), IDT (Interrupt Descriptor Table), etc. [10]. Since rootkits often provide kernel backdoors with the functionalities to hide network connections, processes, files, and drivers, detecting rootkits can be helpful to find out kernel backdoors. Her proposal is very useful to detect rootkits themselves and can be applicable to the real field. Nevertheless, it cannot reveal the evident witnesses of kernel backdoors themselves.

On the other hand, our approach focuses on the kernel backdoor itself, not on its collateral effects as introduced above. By addressing the packet flow differentials between kernel backdoors and the legitimate network applications, our approach can effectively detect and defeat kernel backdoors.

3. Background

In this section, we discover the operational differences between user-mode network applications and kernel backdoors within the network subsystem and find out the clues to differentiate them.

3.1 Network Subsystem Structure

Windows network subsystem structure is described in Fig. 1. The network subsystem has several layered function blocks and each function block is hierarchically related with upper and lower layers — other platforms such as Unix or Linux have similar structures to Windows. Windows has Application Programming Interface (API), Transport Device Interface (TDI), protocols, intermediate drivers, and miniports. Network applications and services provide the users with top layer interfaces for network subsystem, and they are connected to Sockets interface, NetBIOS interface, or others according to their purposes. Then, these network APIs are connected to kernel-mode TDI. Network protocol drivers are the implementation of various protocols such as TCP/IP, IPX/SPX, NetBEUI, and so on. They send and receive network packets to and from Network Interface Cards (NICs) through Network Driver Interface Specification (NDIS). Within NDIS layer, miniports perform NIC-specific operations and intermediate drivers provide standard interface between miniports and protocols [15], [16].

3.2 Backdoors

The operational structures of backdoors are presented in Fig. 2. User-mode backdoors are installed as executable

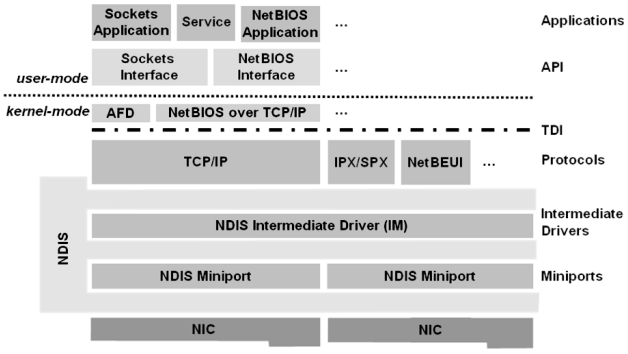
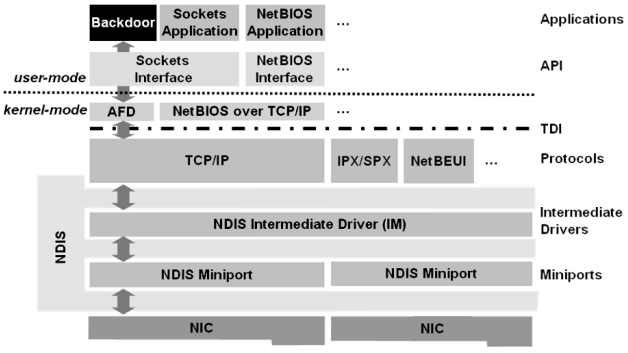
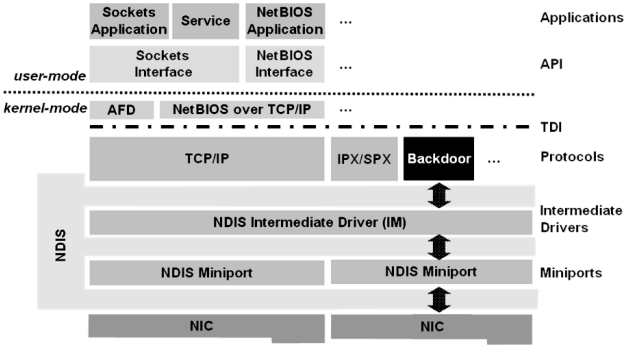


Fig. 1 Windows network subsystem structure.



(a) User-mode



(b) Kernel-mode

Fig. 2 Operational structures of backdoors.

modules and run as processes as legitimate network applications do. Usually, they open specific network ports (e.g. 1243/tcp for SubSeven, 12345/tcp for NetBus, etc.) and wait for the access from the attackers. Moreover, since the network packets from and to user-mode backdoors pass through all layers of a network subsystem, their existence can be exposed by common network monitoring tools or port mappers such as Netstat, FPort [6] and TCPView [22].

On the other hand, kernel backdoors cannot be detected by those conventional means because they run as components of network subsystem such as TCP/IP, NetBEUI, AppleTalk, and so on. They are implemented as kernel-mode drivers which can be inserted into the network subsystem — especially, network protocol drivers [17], [21]. NTrootkit by Greg Houglund [4], [8], [9], SAdoor [18], and yyt_hac's ntrootkit [25] are actually network protocol drivers and they have their own TCP/IP stacks to provide kernel-level communication channels for the attackers without any visible open ports. Moreover, since they work in the privileged mode, they can completely access and control almost all resources of compromised systems such as processes, files, memories, networks, device drivers, and so on. For instance, they can hide specific files, processes, drivers, and network connections from legitimate users by hooking APIs or changing kernel structures [4], [5], [8], [9], [18], [25]. They provide so powerful system controlling ability in order to steal information or prepare next step attacks on the compromised systems without any intervention of conventional security solutions.

4. Packet Flow Differentials

To successfully detect and defeat kernel backdoors which cannot be identified by conventional security solutions, we present a novel technique and develop *KbGuard* to validate the effectiveness of our approach.

4.1 Outline

As described in the previous section, the result of analysis for both network subsystem structure and the operation of kernel backdoors indicates the fact that there are great differences between legitimate network applications and kernel backdoors in terms of their packet flows within the network subsystem. While the packets of common network applications go up and down through the whole network subsystem from API to NDIS and vice versa, kernel backdoors use only NDIS. Accordingly, we can detect kernel backdoors if we have two host-based monitoring sensors — one at higher layer (for TDI) and the other at lower layer (for NDIS). Therefore, we now have a key idea that an IP network connection which does not pass through built-in TCP/IP protocol stack should be regarded as a communication channel of kernel backdoors.

Let us formally define our key idea. To adjust the sensitivity of detection, we have three sensitivity levels; *high*, *medium*, and *low*. Let us denote sensitivity level by s, monitoring time duration with sensitivity level by t_s, inbound direction by *in*, outbound direction by *out*, NDIS layer by N, and TDI layer by T. Then, total number of packets for each network layer during t_s is as follows:

$$\lambda_N(t_s) = N_{in}(t_s) + N_{out}(t_s)$$
$$\lambda_T(t_s) = T_{in}(t_s) + T_{out}(t_s) \quad (1)$$

For a given connection footprint and a given sensitivity level s, we can make a decision on the time of every t_s after the connection is created. We also have a user-defined threshold $\delta_s(> 0)$ of a minimum number of packets ($\lambda_N(t_s)$) in order to say that it is a meaningful connection. The connection footprint should be regarded as a kernel backdoor if $\lambda_N(t_s) \geq \delta_s$ and $\lambda_T(t_s) = 0$, otherwise it must be normal.

4.2 Assumptions

We have following reasonable assumptions to ensure that our key idea will meet the goal to effectively detect and defeat kernel backdoors:

1. **Communication channels of kernel backdoors are bidirectional**: The communication between attackers and kernel backdoors is composed of a bidirectional channel for both commands and their results.
2. **Kernel backdoors employ IP protocol**: If kernel backdoors employ non-IP protocols, their communication will be restricted out of the Internet. Therefore, they employ IP protocol. In fact, all of them employ only IP protocol [9], [14], [18], [25].
3. **Only built-in TCP/IP stack treats legitimate IP packets**: If another protocol stack (not built-in TCP/IP) treats IP packets, we regard it as an illegitimate treatment of IP packets and eventually detect as a footprint of kernel backdoor.
4. **Kernel Backdoors are located at protocol layer**: It is believed that all current kernel backdoors are located at protocol layer [5]. For instance, NTrootkit by Greg Houglund [4], [8], [9], SAdoor [18], and yyt_hac's ntrootkit [25] are network protocol drivers which have their own TCP/IP stacks.
5. **Kernel backdoors employ only non-reflective packets**: If kernel backdoors use reflective packets (e.g. ICMP Echo Requests, TCP/SYNs, etc.), the reflective packets will consequently expose the existence of malicious network behaviors because built-in TCP/IP stack tries to reply for them. Actually, most of kernel backdoors employ only non-reflective packets [18], [25]. We will discuss this issue in the later section.
6. **Kernel backdoors have data payloads in their packets**: When an attacker connects to kernel backdoors, their packets must have data payloads — greater than a specific size — to receive commands and to send their results. Covert channels are beyond the scope of this paper.

4.3 Reflective and Non-reflective

As mentioned in the previous section, it is believed that kernel backdoors are likely to employ non-reflective IP packets. For instance, when ICMP Echo Request packets are employed for the communication between an attacker and the kernel backdoor, for a given packet from the attacker toward the kernel backdoor, not only a responded packet from the kernel backdoor but also additive packets (ICMP Echo Replies) from the built-in TCP/IP stack are sent back to the attacker. Therefore, the attackers have no choice but to employ non-reflective packets to keep secure — from their point of view — communication channels. Table 1 shows an example of non-reflective packets which are employed by yyt_hac's ntrootkit [25] for various communication methods of the kernel backdoor (TCP, UDP, ICMP, and IP user-defined). When an attacker connects to the kernel backdoor through an ICMP channel of yyt_hac's ntrootkit, an attacker sends only ICMP Echo Reply packets and receives only ICMP Destination Unreachable packets, which are not reflective. As for TCP, kernel backdoors set RST flag for TCP channels to avoid packet reflections [18], [25]. UDP and user-defined channels also use non-reflective packets.

4.4 Strategies to Mitigate False Positives and Negatives

Since various network applications and services are working on the Internet, our key idea may cause false positives or false negatives. Actually, we found some false positives in following network situations:

1. **DNS Queries and Replies**: When network applications on the local machine try to resolve domain names, built-in TCP/IP stack sends DNS Queries to and receives DNS Replies from a registered DNS server without interactions with TDI layer. The connection for DNS Queries and Replies cause false positives according to our key idea.
2. **ICMP Echo Requests and Replies**: If someone on the remote machine uses Ping utility to check the aliveness of the local machine, built-in TCP/IP stack of the local machine will receive ICMP Echo Requests from and send ICMP Echo Replies to the remote machine regardless of TDI [20], which causes false positives.
3. **ICMP Time Exceeds**: If the gateway processing a packet finds the TTL field is zero or a host reassembling a fragmented packet cannot complete the reassembly within its time limit, they discard the packet and may also send back to the source an ICMP Time Exceed message [20]. Since built-in TCP/IP stack treats the message alone, our key idea may regard it as a kernel backdoor. This situation also occurs with ICMP Destination Unreachables.
4. **Denial-of-Service attacks**: If someone on the remote machine attacks the local system by means of Denial-of-Service (DoS) attacks, the DoS packets cause false positives because built-in TCP/IP stack treats them by itself and our key idea regards this situation as a kernel backdoor. Port scanners such as Nmap [7] may also cause the same situation.

Table 1 Non-reflective packets employed by yyt_hac's ntrootkit [25].

No.	Inbound (toward a backdoor)	Outbound (toward an attacker)	Type
1	ICMP Echo Reply	ICMP Destination Unreachable	ICMP
2	445/tcp (RST set)	445/tcp (RST set)	TCP
3	445/udp	445/udp	UDP
4	Unknown	Unknown	IP—user defined

On the other hand, to connect with the kernel backdoor, if an attacker uses UDP channels toward already opened destination ports (e.g. 137/udp, 138/udp, 445/udp for file sharing services) of the compromised system, the incoming packets toward the opened UDP ports will go up to application level, which in turn causes false negatives because the status of the connection becomes $\lambda_T(t_s) > 0$ even though it is a connection for the kernel backdoor.

To overcome above mentioned false positives and negatives, we adopted the assumptions as described earlier and constructed a decision tree as shown in Fig. 3. Non-bidirectional connections are regarded as normal at a glance according to the first assumption, which is the first step. The second step is our key idea. The third step takes a reflective packet rate (γ) or a non-reflective packet rate ($100 - \gamma$) into consideration according to the fifth assumption. Finally, the sixth assumption makes the fourth step have three classes of decision — *normal*, *suspicious*, and *kernel backdoor* — by considering the average data payload size (denoted by θ) of a given connection footprint. The rate of reflective packets for a given connection is defined as shown in Eq. (2) where

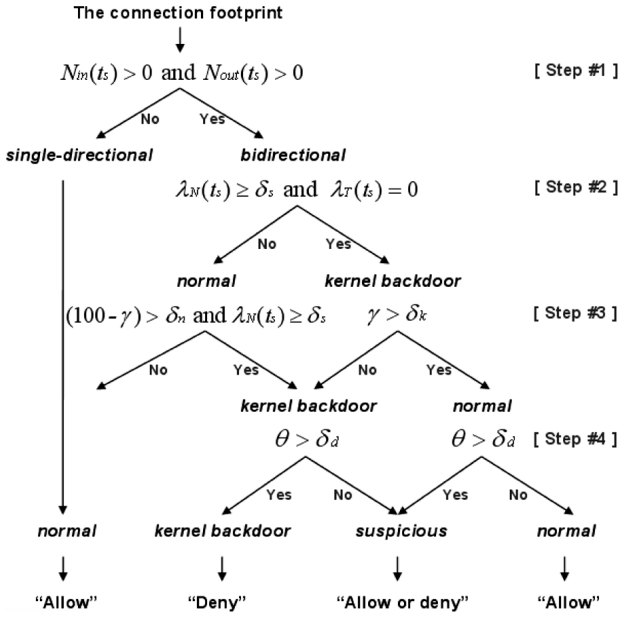


Fig. 3 The decision tree for detecting and defeating kernel backdoors.

λ_R is the total number of reflective packets for a given connection.

$$\gamma = \frac{\lambda_R(t_s)}{\lambda_N(t_s)} \times 100 \quad (2)$$

Let δ_k be maximum γ to say that it is *kernel backdoor* at the third step for the decision of *kernel backdoor* at the second step and δ_n be maximum $(100 - \gamma)$ to say that it is *normal* at the third step for the decision of *normal* at the second step.

An average data size in bytes for a given connection is defined as shown in Eq. (3). We can calculate it by analyzing L4 protocols (e.g. TCP, UDP, ICMP, etc.) for each packet and we also regard option fields in protocol headers as data.

$$\theta = \frac{\sum d(P_i)}{\lambda_N(t_s)} \quad (3)$$

Let P_i be i-th packet within a given connection, $d(P_i)$ be data size of a given packet P_i, and δ_d be maximum θ to say that it is *normal* at the fourth step for the decision of *normal* at the third step or to say that it is *suspicious* at the fourth step for the decision of *kernel backdoor* at the third step.

Therefore, the four-step decision process is performed as follows. For a given connection footprint, if it is not bidirectional, it is finally regarded as *normal* at the first step, otherwise we go on to the next step. At the second step, we calculate $\lambda_N(t_s)$ and $\lambda_T(t_s)$ then initially determine if it is *kernel backdoor* or not. Next, we further decide if it is *kernel backdoor* by analyzing γ or $(100 - \gamma)$ at the third step. Hence, if $\lambda_N(t_s) \geq \delta_s$ and $\lambda_T(t_s) = 0$ at the second step or if $\lambda_T(t_s) > 0$ at the second step and $(100 - \gamma) > \delta_n$ at the third step, we should consider θ at the fourth step. Finally, we allow or deny the connection according to the decision.

4.5 KbGuard

To validate our approach, we implemented *KbGuard* and performed experiments in a simulated environment. Figure 4 shows the overall structure of *KbGuard*. It is made up of three components as follows:

1. ***TDI monitor***: *TDI monitor* implemented as a TDI filer driver captures all IP packets which pass through built-in TCP/IP protocol stack.
2. ***NDIS monitor***: All IP packets which go up and down within NDIS are captured by *NDIS monitor* — implemented as a NDIS intermediate driver. It can selectively filter packets by the command of the *Inspector* to deactivate suspicious connections.
3. ***Inspector***: *Inspector* collects packet information from both *TDI monitor* and *NDIS monitor*. It then analyzes the information to build up the connection footprint table, makes a decision, and allows or denies the connection according to the decision.

TDI monitor and *NDIS monitor* capture only IP packets for both incoming and outgoing traffics according to our second assumption. The connection footprint is consist of several factors as shown in Table 2.

LocalAddr, LocalPort, RemoteAddr, RemotePort, and Protocol are used to uniquely identify each connection. $N_{in}(t_s)$, $N_{out}(t_s)$, $T_{in}(t_s)$, and $T_{out}(t_s)$ represent the numbers of packets of NDIS and TDI for both inbound and outbound directions to analyze packet flow differentials. PacketData is used to calculate a reflective packet rate (γ) and an average data size (θ) for a given connection.

Table 3 shows an example of the connection footprint table when two remote hosts (10.0.0.1 and 10.0.0.2) connect

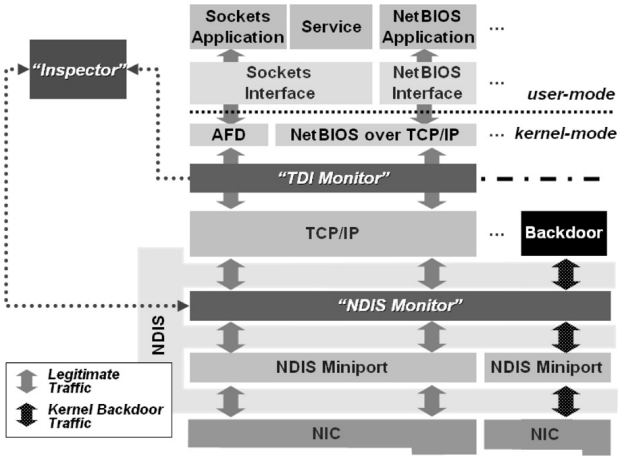


Fig. 4 The overall structure of *KbGuard*.

Table 2 The connection footprint.

Factor	Description
LocalAddr	local IP address
LocalPort	local port (TCP or UDP)
RemoteAddr	remote IP address
RemotePort	remote port (TCP or UDP)
Protocol	protocol (TCP, UDP, or IP)
PacketData	packet data
$N_{in}(t_s)$ and $N_{out}(t_s)$	total number of inbound and outbound packets at NDIS layer
$T_{in}(t_s)$ and $T_{out}(t_s)$	total number of inbound and outbound packets at TDI layer
γ	rate of reflective packets
θ	average data payload size of packets

Table 3 An example of the connection footprint table where $\delta_k = 40$, $\delta_n = 80$, $\delta_s = 5$, $\delta_d = 40$, and $t_s = 3$.

No.	lAddr:lPort-rAddr:rPort (Proto)	$\lambda_N(t_s)$	$\lambda_T(t_s)$	γ	θ	Decision 1	2	3	4
1	10.0.0.166:23-10.0.0.2:1132 (TCP)	10	10	100	2	-	*N*	*N*	-
2	10.0.0.166:80-10.0.0.2:2195 (TCP)	31	31	94	329	-	*N*	*N*	-
3	10.0.0.166-10.0.0.2 (IP)	20	0	50	16	-	*K*	*N*	*N*
4	10.0.0.166:445-10.0.0.1:445 (UDP)	45	15	0	51	-	*N*	*K*	*K*

to the local host 10.0.0.166. Let us denote *normal* by *N*, *suspicious* by *S*, and *kernel backdoor* by *K*. The first connection is for Telnet and the second one is for HTTP. They are to be decided as *normal* because they are bidirectional at the first step of the decision process, $\lambda_T(t_s) > 0$ at the second step, and $(100 - \gamma) < \delta_n$ at the third step. The third connection is a set of legitimate ICMP Echo Requests and Replies. It is initially determined as *kernel backdoor* through the first and the second steps because it is bidirectional and $\lambda_N(t_s) \geq \delta_s$ and $\lambda_T(t_s) = 0$. However, it is finally decided as *normal* because $\gamma > \delta_k$ and $\theta < \delta_d$ at the third and the fourth step respectively. On the other hand, the fourth connection is for *kernel backdoor*. It is initially *normal* through the first and the second steps because it is bidirectional and $\lambda_T(t_s) > 0$, but it is finally decided as *kernel backdoor* because $(100-\gamma) > \delta_n$ at the third step and $\theta > \delta_d$ at the fourth step.

5. Evaluation

A screenshot of *KbGuard* is shown in Fig. 5. Three connections of kernel backdoors are successfully detected and immediately deactivated. With *KbGuard*, we performed experiments in our laboratory environment as shown in Fig. 6 for the evaluation of our approach. The victim with *KbGuard* installed is compromised by kernel backdoors — yyt_hac's ntrootkit 1.22 [25] and NTrootkit 0.40 [9]. All systems used for experiments are constructed on Windows 2000 professional and the network bandwidth is 100 Mbps. When the client connects to the victim with normal network services such as HTTP, E-mail, etc and the attacker connects to the victim through kernel backdoors, *KbGuard* constructs the connection footprint table, makes a decision for each connection, and finally filters the packets of *kernel backdoor* connections. In the experiments, almost all kernel backdoors were successfully detected and deactivated by *KbGuard*. Moreover, false positives and negatives were effectively mitigated.

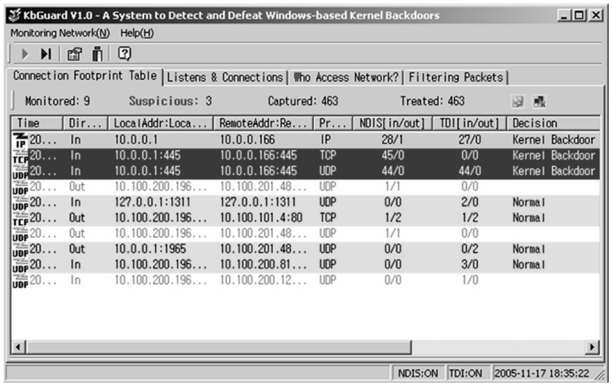


Fig. 5 The screenshot of *KbGuard*.

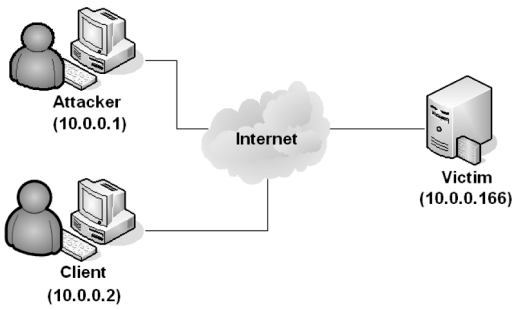


Fig. 6 Experimental setting of *KbGuard*.

Table 4 Experimental results in various network situations where $\delta_k = 40$, $\delta_n = 80$, $\delta_s = 5$, $\delta_d = 40$, and $t_s = 3$.

No.	Class	Case	$\frac{\lambda_T(t_s)}{\lambda_N(t_s)}$	γ	θ	Decision 1	2	3	4
1		HTTP	1.00	99	266	-	*N*	*N*	-
2		HTTPS	1.00	99	281	-	*N*	*N*	-
3		SMTP	1.00	100	96	-	*N*	*N*	-
4		FTP	1.00	100	762	-	*N*	*N*	-
5		NetBIOS	0.85	89	32	-	*N*	*N*	-
6		Microsoft-ds	0.94	94	67	-	*N*	*N*	-
7	*N*	MSN	1.00	100	18	-	*N*	*N*	-
8		Telnet	1.00	100	2	-	*N*	*N*	-
9		NNTP	1.00	99	276	-	*N*	*N*	-
10		ICMP Echo Request/Reply	0.00	50	16	-	*K*	*N*	*N*
11		ICMP Time Exceed	0.00	0	20	*N*	-	-	-
12		ICMP Destination Unreachable	0.00	0	20	*N*	-	-	-
13		DNS	0.00	100	32	-	*K*	*N*	*N*
14		send ONLY	0.00	-	-	*N*	-	-	-
15	*S*,*K*	send and receive	0.00	$> \delta_k$	$> \delta_d$	-	*K*	*N*	*S*
					$< \delta_d$	-	*K*	*N*	*N*
				$< \delta_k$	$> \delta_d$	-	*K*	*K*	*K*
					$< \delta_d$	-	*K*	*K*	*S*
16		ntrootkit (open UDP port)	0.79	0	48	-	*N*	*K*	*K*
17		ntrootkit (open TCP port)	0.00	0	48	-	*K*	*K*	-
18		ntrootkit (closed TCP/UDP port)	0.00	0	48	-	*K*	*K*	-
19	*K*	ntrootkit (ICMP)	0.00	0	48	-	*K*	*K*	-
20		ntrootkit (IP-user defined)	0.00	0	62	-	*K*	*K*	-
21		NTrootkit (open TCP port)	0.00	100	54	-	*K*	*N*	*S*
22		NTrootkit (closed TCP port)	0.00	100	54	-	*K*	*N*	*S*

5.1 Performance

Table 4 shows the experimental results for various network applications and situations. We explain the experiments in more detail. To test normal cases, when a host (10.0.0.166) connects to the Internet for various network services, all network packets are monitored by *KbGuard* on the host. To test the cases of kernel backdoors, an attacker (10.0.0.1) tries to connect to the kernel backdoor of a victim (10.0.0.166) when *KbGuard* installed on the victim performs the detection processes. As a special case, when WinPcap — a packet capturing and generating utility — on the victim generates IP packets, they are also monitored by *KbGuard* on the host. We considered various network situations to evaluate the detection rate of *KbGuard*. To test true negatives and false positives for normal cases, we used popular Internet services and some specific network situations. Meanwhile, we used WinPcap [11], yyt_hac's ntrootkit 1.22 [25], and NTrootkit 0.40 [9] to test true positives and false negatives.

$\frac{\lambda_T(t_s)}{\lambda_N(t_s)}$ of normal network applications (from the first to the ninth cases) are almost 1.00, they are bidirectional, and their γ are nearly 100, which in turn makes them regarded as *normal* at both the second and the third steps. ICMP Time Exceed and Destination Unreachable are finally decided as *normal* at the first step because they are single-directional. Meanwhile, since ICMP Echo Request/Reply and DNS are bidirectional and can be seen by only *NDIS monitor*, they are initially determined as *kernel backdoor* through the first and the second steps, but the third and the fourth steps finally regard them as *normal* in the sense that $\gamma > \delta_k$ and $\theta < \delta_d$. Therefore, *KbGuard* accurately makes decisions as *normal* for various normal network situations.

If WinPcap generates IP packets, our decision varies according to the types of generated packets and the data payload size (θ). We will consider this issue in the next section. With yyt_hac's ntrootkit [25], its connection footprints are successfully detected without regard to its communication channels such as TCP, UDP, ICMP, and user-defined IP. However, the second step regards it as *normal* in cases of open UDP ports because of the connectionlessness of UDP. On the other hand, NTrootkit [9] is initially detected as *kernel backdoor* at the second step, but it is regarded as *normal* at the third step and finally *suspicious* at the fourth step. It actually employs 100% reflective packets with its own TCP/IP stack and we can even connect to it with legitimate Telnet clients. However, it cannot get along with the built-in TCP/IP stack in order not to cause packet reflections by built-in TCP/IP stack or to treat all IP packets alone [8], [9], which in turn makes it inapplicable to the real field. It is just a proof of concept for kernel backdoors as the author says [8].

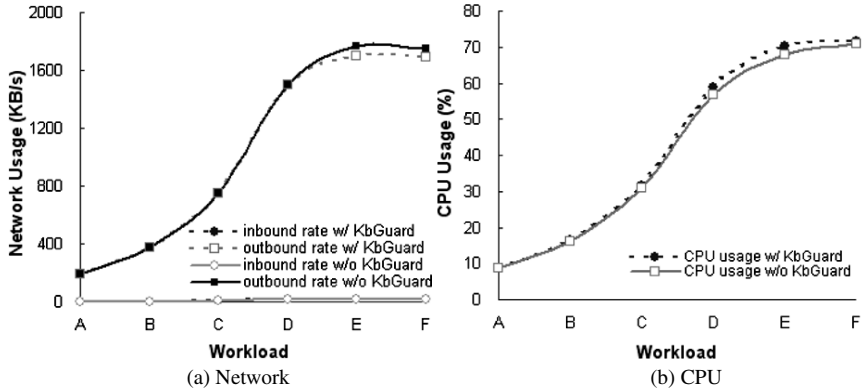
Considering overall evaluation results as mentioned above, our approach effectively detects and defeats kernel backdoors.

5.2 Considerations

If WinPcap sends and receives IP packets, its connections may be detected as *suspicious* or *kernel backdoor* in some special cases as shown in Table 4. However, they are not false positives because generating artificial packets is suspicious enough in itself. In fact, some types of kernel backdoors such as yyt_hac's ntrootkit [25] and SAdoor [18] employ WinPcap to communicate with the attackers. It seems that they make bad use of WinPcap to avoid detection assuming that most of security solutions regard it as one of known legitimate network tools. Therefore, the activities of sending and receiving packets by WinPcap must not be regarded as *normal*. *KbGuard* successfully detects the connections related with WinPcap as *suspicious* or *kernel backdoor*.

5.3 Overheads

To evaluate network and CPU overheads of *KbGuard*, we examined the changes of network inbound/outbound rates and CPU usages by Microsoft Web Application Stress Tool (MWAST) and Performance Counter (Windows diagnostic mechanism) respectively. While the MWAST on the client concurrently connected to the web server on the victim with or without *KbGuard*, we changed network workloads by



(a) Network (b) CPU

Fig. 7 Overheads.

adjusting concurrent connections of the MWAST — 1(A), 2(B), 4(C), 8(D), 16(E), and 32(F). The web server also had three types of mixed documents in their size (10 K, 20 K, and 30 K) to vary traffic patterns. As shown in Fig. 7, we have found maximum degradation effects with workload E — network(inbound) 3.73%, network(outbound) 3.84%, and CPU usage 2.30%, which would be acceptable to the real field.

5.4 Limitations and Weaknesses

Although our approach successfully detects and defeats kernel backdoors as shown earlier, it also has some limitations and weaknesses. First, we have two monitoring sensors according to our fourth assumption. However, our approach will not work properly if future kernel backdoors are located at NDIS intermediate drivers or even miniports. To avoid this situation, the lower layer one (*NDIS monitor*) should go down or cover whole NDIS layers to sandwich the future kernel backdoors. Second, we need to consider self-protection mechanisms to prevent some types of attacks toward *KbGuard*.

6. Conclusions

In this paper, we have proposed a novel technique to effectively detect and defeat kernel backdoors, and implemented *KbGuard* to evaluate the performance of our technique. It successfully detected the network connections between kernel backdoors and attackers by analyzing packet flow differentials for the connections within the network subsystem according to our four-step decision process. Furthermore, it could immediately deactivate kernel backdoors by filtering the suspicious connections. We believe that our technique is one of the most effective countermeasures against stealthy threats of kernel backdoors. We will extend our technique to scaled-up network environments, apply to another platforms (e.g. Unix, Linux, etc.), and try to overcome its limitations and weaknesses. We will also elaborate the proposed four-step decision process.

References

[1] Admin, "Backdoors," Unix Security, Internet Software Marketing Ltd., 2002.
[2] B. Bobkiewicz, "Hidden backdoors, trojan horses and rootkit tools in a Windows environment," Windows OS Security, Internet Software Marketing Ltd., 2004.
[3] J. Braun, "What port numbers do well-known Trojan horses use?," SANS Intrusion Detection FAQ, 2001.
[4] F.J. Cibelli, "Beware of geeks bearing gifts: A Windows NT rootkit explored," Incident Handling and Hacker Exploit Practical, 2001.
[5] firew0rker, "Kernel-mode backdoors for Windows NT," Phrack Magazine, vol.62-6, 2004.
[6] Foundstone, "FPort v2.0 — TCP/IP process to port mapper," 2000.
[7] Fyodor, "Nmap — The art of port scanning," 1997.
[8] G. Houglund, "A *REAL* NT rootkit, patching the NT kernel," Phrack Magazine, vol.55-5, 1999.
[9] G. Houglund, "NTrootkit v0.40," 2001.
[10] R. Joanna, "Rootkits detection on Windows systems," Proc. ITUnderground Conference, pp.40–44, Warsaw, 2004.
[11] Network Research Group, "Libpcap," Lawrence Berkeley National Labs., 2004.
[12] Y. Liu, "W32.HLLW. Doomjuice," Technical Description, Symantec Security Response, 2004.
[13] Marcus, "The art of rootkits (2nd ed)," The Infosec Writers Text Library, 2004.
[14] McAfee, "BackDoor-ALI," McAfee, Networks Associates Technology, 2001.
[15] Microsoft, NetBIOS over TCP/IP, MSDN Library, Microsoft Corporation, 2004.
[16] Microsoft, Network Devices and Protocols, MSDN Library, Microsoft Corporation, 2004.
[17] NISCC, "Trojan horse programs and rootkits," NISCC Technical Note, 2003.
[18] C.M. Nyberg, "SAdoor winserver version 1.1," 2003.
[19] A.G. Pennington, J.D. Strunk, J.L. Griffin, C.A. Soules, G.R. Goodson, and G.R. Ganger, "Storage-based intrusion detection: Watching storage activity for suspicious behavior," Proc. 12th USENIX Security Symposium, pp.137–152, Washington, DC, Aug. 2003.
[20] J. Postel, "Internet control message protocol," Network Working Group, RFC 792, Sept. 1981.
[21] K. Poulsen, "Windows rootkits a stealthy threat," SecurityFocus HOME News, 2003.
[22] M. Russinovich, "TCPView for NT/2000/XP/9x," Sysinternals, 2002.
[23] SecureWave, "Sanctuary device control," 2004.
[24] A. Vidstrom, "ACK tunneling Trojans," NTSecurity.nu, 2001.
[25] yyt_hac, "yyt_hac's ntrootkit 1.22," 2004.
[26] Y. Zhang and V. Paxson, "Detecting backdoors," Proc. 9th USENIX Security Symposium, pp.157–170, 2003.
[27] W. Cui, R.H. Katz, and W. Tan, "Design and implementation of an extrusion-based break-in detector for personal computers," Proc. 21st ACSAC, pp.361–370, 2005.

Cheolho Lee received the B.S. degree in Information and Computer Engineering from Ajou University in 2002 and the M.S. degree in Information and Communications Engineering from Ajou University in 2004. Since 2004, he has been with Electronics and Telecommunications Research Institute (ETRI), where he is a member of research staff. His research interests include security monitoring, behavioral analysis, network security, and malicious code detection.

Kiwook Sohn received the B.S. and M.S. degrees in Information Engineering from Sungkyunkwan University in 1990 and 1992, respectively. He received the Ph.D. degree in Electricity, Electronics, and Computer Engineering from Sungkyunkwan University in 2002. Since 1992, he has been with Electronics and Telecommunications Research Institute (ETRI), where he is a principal member of research staff. His research interests include cryptographic protocol, network security, and malicious code detection.