A Chipset Level Network Backdoor: Bypassing Host-Based Firewall & IDS

Sherri Sparks, Shawn Embleton, Cliff C. Zou
School of Electrical Engineering and Computer Science
University of Central Florida
4000 Central Florida Blvd., Orlando, FL USA 32816-2362
+1-407-823-5015
{sparks, embleton}@clearhatconsulting.com, czou@eecs.ucf.edu

ABSTRACT
Chipsets refer to a set of specialized chips on a computer's motherboard or an expansion card [12]. In this paper we present a proof of concept chipset level rootkit/network backdoor. It interacts directly with network interface card hardware based on a widely deployed Intel chipset 8255x, and we tested it successfully on two different Ethernet cards with this chipset. The network backdoor has the ability to both covertly send out packets and receive packets, without the need to disable security software installed in the compromised host in order to hide its presence. Because of its low-level position in a computer system, the backdoor is capable of bypassing virtually all commodity firewall and host-based intrusion detection software, including popular, widely deployed applications like Snort and Zone Alarm Security Suite. Such network backdoors, while complicated and hardware specific, are likely to become serious threats in high profile attacks like corporate espionage or cyber terrorist attacks.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – *invasive software, security kernels*

General Terms
Security

Keywords
Rootkit, network backdoor, hardware security

1. INTRODUCTION
Host-based firewalls and intrusion detection systems have made significant advances in both technology and scope of deployment within the past few years. Despite these advances, two challenges remain: they focus mostly on defending against outside attacks instead of inside information exfiltration, and they are mostly relying on the underlying Operating System's support for data gathering and monitoring. In this paper, we present a network rootkit / backdoor that exploits these two problems. This network backdoor is capable of bypassing virtually all commodity, host-based firewall and intrusion detetection software on the market today, including popular, widely deployed products like Snort and Zone alarm.

Traditionally, firewalls, network based intrusion detection systems (IDS) and intrusion prevention systems (IPS have been focused on outsider threats. These types of systems monitor incoming network traffic or system behavior for malicious code or attacks. When an attack is detected, the system reacts in real-time to block or prevent it (e.g. by dropping the malicious packets while allowing other network traffic to pass). Unfortunately, many of these systems only filter inbound traffic, still leaving the protected machine vulnerable to a large class of insider threats resulting from the free flow of unauthorized, outbound traffic. The firewall provided with the Windows XP operating system is one such example [11]. The implications include leaving the machine vulnerable to the exfiltration of sensitive information as well as delaying detection of malware threats resulting from unmonitored outgoing traffic. Extrusion detection is to deal with this security issue, which focuses "primarily on the analysis of system activity and outbound traffic in order to detect malicious users, malware or network traffic that may pose a threat to the security of neighboring systems [27]."

The potential for sensitive data exfiltration is perhaps the most significant threat arising from unrestricted outbound traffic flow. The exfiltration of sensitive information can occur either inadvertently or deliberately and affects both corporate organizations and individuals. For example, spyware and adware infestations are extremely prevalent on home PC's with the AOL/NCSA study showing that 80% of home computers are infected and that the average infected user has 93 spyware or adware components on their computer [10]. Additional threats that remain inadequately addressed by existing IDS, IPS, and firewall technology's failure to filter outbound traffic include delayed detection of DDOS attacks, Botnets, and Internet Worms.

The second problem concerns the reliance of host-based firewall and intrusion detection tools on the trustworthiness of the underlying Operating System. Unfortunately, malware authors have developed an arsenal of techniques to exploit this reliance and cheat the data returned to applications and drivers that rely on the OS API. These techniques range from preventing security software from loading to complex hooks in Operating System network stack [15]. This problem should not be understated. To illustrate this potential threat, we present a network backdoor in this paper that operates at the physical network card interface and successfully

ASIACCS'09, March 10-12, 2009, Sydney, NSW, Australia.

bypasses virtually all commodity host-based firewall and IDS / IPS software on the market today.

The proposed network backdoor is essentially a rootkit (a malicious program that tries to hide its existence on an infected computer), thus it relies on vulnerability exploits, such as through worms or email viruses, or other attack mechanisms to install it on a computer. How to compromise a remote computer is not the focus of the proposed network backdoor.

Many people may think that attackers can simply deactivate any defense systems running on a computer once the computer is compromised, and hence, it is not necessary for attackers to deploy any advanced hiding techniques. This is true for computers where their users or security managers are careless. For other computers, however, deactivating security defense systems can be easily noticed by security-minded users via some simple system checks. Hiding malicious codes and their activities on an infected computer with as small as possible system change is still essential to serious attackers, especially in botnet attacks or long-term business espionage.

The contributions of this paper are as follows: First, we provide a design and implementation of a network rootkit / backdoor that is capable of bypassing virtually all currently available commodity, host-based firewalls and intrusion detection systems. Our backdoor possesses the ability to both covertly send and receive network packets over a compromised host's network interface. Secondarily, we test our attack against several well-known firewalls and intrusion detection systems. Finally, we seek to raise awareness of the data exfiltration problem as it relates to both data loss prevention and malware propogation and consider potential defenses against such attacks.

This paper is organized as follows. In section 2, we give an overview of related work in the area of extrusion detection and prevention as well as discuss previous methods used by malware authors to bypass IDS and IPS systems. Section 3 describes our implementation of a network backdoor capable of bypassing a large class of firewalls and intrusion detection tools. Results from the testing of our implementation against several well known systems are provided in section 4. Section 5 discusses possible defensive measures. We conclude in section 6.

2. BACKGROUND & RELATED WORK

Several extrusion detection tools have been developed and discussed in the literature.

Cui et al. described an extrusion-based break in detector for personal computers called BINDER [16]. They note that many malware applications send malicious outgoing network traffic from compromised computers and make the observation that most legitimate network activities are directly or indirectly triggered by user input. BINDER detects compromises by correlating network activity with user input on the premise that malicious code typically runs in the background and generates connections without user input.

Another outbound intrusion, or extrusion, detection tool called FROID was developed and presented by Salvador Mandujano [17]. FROID attempts to protect a set of nodes in a network by having each member monitor its own outbound traffic for evidence of compromise. It was built using the JADE agent framework and takes an ontology-based approach to the detection of malicious code [18]. The prototype features a misuse based detection based on signatures derived from network traffic and process execution.

Web Tap is an anomaly based intrusion detection tool specifically focused on detecting malicious, covert outbound HTTP traffic, like spyware, in an otherwise firewalled network [19]. By analyzing outbound HTTP traffic, the authors developed filters capable of detecting several covert web tunneling programs, a backdoor, and several spyware / adware applications.

Zhang and Paxson tackle the problem of generically identifying backdoors, specifically those that provide interactive access on non standard ports [20]. They note that interactive traffic has different traits than application generated traffic. In order to search for traffic containing these traits, they successfully propose and test a passive network monitoring algorithm based upon keystroke characteristics including directionality of the connection, packet sizes, and packet interarrival times.

Although these tools seek to address the outbound malicious activities, the implementations described in these papers remain vulnerable to the second problem. This is, they all rely on host-based network monitoring for the correct operation of their tool. In order for these tools to monitor network traffic, they must rely upon the network API support provided by the Operating system to intercept that network traffic. This is a common weakness. A variety of malware techniques exist to subvert this reliance. These techniques exploit the fact that modern Operating Systems like Linux and Windows, are built upon a layered architecture. In general, by inserting themselves lower in the architecture a malware application gains more stealth and power. For example, a malicious kernel driver is more powerful and capable of evading detection than a malicious usermode application. A stealthier malware application does not rely upon the OS at all, but instead interacts directly with the hardware.

Clearly, the ability to evade a host-based firewall or IDS is a valuable asset for malware like worms or botnets who would like to delay detection for as long as possible. A number of methods have been previously proposed and /or implemented.

Perhaps the simplest approach for a malicious kernel module is to register a driver load notification callback. When a new driver is loaded, the OS calls the malware defined callback function giving it a chance to scan it for signatures corresponding to known firewall drivers. If a firewall is detected, the malware simply prevents it from successfully loading. The drawback to this method is clearly the fact that the malware must be resident and active in memory prior to the firewall.

More advanced attacks attempt to hook into the OS network subsystem in order to make the OS return false information to the IDS or firewall. NT Rootkit by Greg Hoglund is an example of this type of rootkit [23]. For example, the two primary components of the network subsystem of interest to malware authors on Windows Operating Systems are TDI (Transport Driver Interface) and NDIS (Network Driver Interface Specification) [21]. Figure 1 illustrates the relationship between these components. Both of these components are also used by security software to implement firewalls and IDS.

TDI defines an upper level kernel network interface. Under Windows 2000/XP/2003 based systems, tcpip.sys is the primary

driver that exposes the Transport Data Interface. It creates 4 devices including TCP, UDP, Raw IP, and ICMP. A firewall may intercept the TDI interface to control network access at a per process granularity and to simplify detection and prevention of attacks at the application layer. For example, TDI may be used to decide if a given process is allowed to open a TCP / UDP port or send and receive network data. The interception is usually performed with a special driver, called a filter driver. The filter driver attaches itself above tcpip.sys in the Windows network stack. From this position, it is able to transparently and invisibly snoop communications to and from tcpip.sys devices. Unfortunately, such a filter is limited by the fact that it sits at top of the kernel network subsystem. As a result, it is only able to control the network communications for drivers that exist above it. In practice, this limits the usefulness of TDI interception to malware using the kernel mode sockets interface.

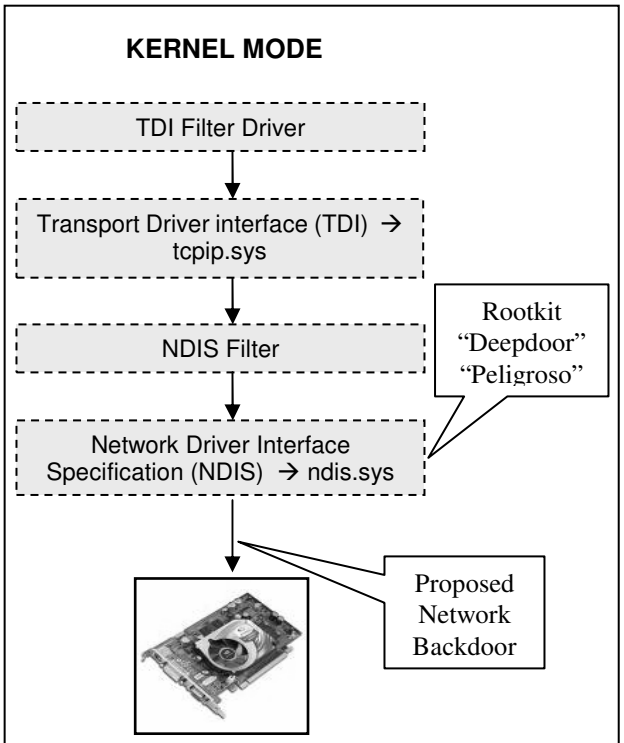

Figure 1: Windows Network Architecture

NDIS operates below TDI. Its primary purpose is to abstract the physical network hardware from network drivers. It is possible for a firewall to intercept NDIS functions to filter traffic at a lower level than TDI based driver. Windows provides several methods of hooking NDIS functions including development of an NDIS Intermediate Driver, developing a Filter Hook Driver (Windows firewall hook driver), and registering a new protocol to NDIS to force the NDIS protocol characteristics table to hook the TCPIP protocol NDIS functions. Although it is more powerful, NDIS based firewall solutions are more complex and will have difficulty associating opened ports with application layer processes. Malware may also intercept NDIS functions in an effort to bypass host-based firewalls or intrusion detection systems. One approach is to simply replace the firewall hooks with their own malicious hooks. This technique was demonstrated in the "DeepDoor" rootkit by Joanna Rutkowska [5] and the "Peligroso" rootkit by Greg Hoglund [6]. Such changes, however, may be detected by more advanced firewalls which validate the presence of their hooks and the integrity of their handlers.

As mentioned, TDI and NDIS hooking techniques are used both by firewall & IDS developers as well as malware authors in an elaborate game of cat and mouse. The general trend that can be observed, however, is that the lower (e.g. closer to the hardware) one goes, the greater their power and stealth. Both TDI and NDIS hooking as used by malware authors may be viewed as a form of "man in the middle" attack. Extending this form of attack to its logical conclusion would be the development of a stealthy network exfiltration backdoor that exists as low as is physically possible (directly above the physical hardware). While relatively rare due to their complexity, there has been some prior research into hardware level rootkits. John Heasman discussed the development of proof of concept BIOS and PCI rootkits on Windows NT systems [24].

The remainder of this paper discusses the feasibility and development of a proof of concept chipset level network backdoor. This backdoor successfully evades all firewall and intrusion detection tools that we tested it against. We provide the details of our implementation and experimentation on 2 popular network cards in the following sections and provide recommendations for mitigating this form of malware attack.

3. DESIGN & IMPLEMENTATION

In this section, we discuss our development of a chipset level network backdoor capable of both receiving remote commands and exfiltrating information across the boundary of most host-based commercial firewalls and intrusion detection systems. Our backdoor resides below both the NDIS and TDI Operating System interfaces at the physical hardware layer of the network card. Thus, it is capable of bypassing any malicious code detection / prevention software running at an abstracted level above the hardware. We chose to develop and install our proof of concept code as a Windows kernel driver to simplify testing and debugging. Unfortunately, we must sacrifice hardware dependence for OS independence and increased stealth. As a result, our current implementation is limited to cards compatible with the Intel 8255x chipset. It is neither necessary nor desirable for us to extend our implementation to support a larger number of chipsets. Our intention is to provide a proof of concept that addresses a critical hole in existing security technology, not provide the blueprints for malware authors to develop a fully featured attack tool. Nevertheless, it remains that the Intel 8255x chipset is compatible with many existing Intel ethernet cards.

We break the details of our implementation down into 2 subtasks: data exfiltration and data infiltration.

Offset	Upper Word	Lower Word
0x00	SCB Command Word	SCB Status Word
0x04	SCB General Pointer	

Figure 2: 8255x System Control Block (SCB)

3.1. Data Exfiltration

Data exfiltration refers to the process involved in sending data out from the compromised host. We send data out by interacting directly with the LAN controller hardware over the PCI bus. The LAN controller acts as both a master and a slave on the PCI bus. In the role of master, it interacts with system memory to access transmit and receive data buffers. As a slave, the host processor accesses the LAN controller's internal structures to read and write information to its on-chip registers. These registers may be either I/O mapped or memory mapped. The method to use is determined by system software. First, we give a brief overview of the Intel 8255x frame transmission and reception architecture.

The Intel 8255x chipset consists of 2 primary components: the Command Unit (CU) and the Receive Unit (RU). Software issues commands to control these components through a memory mapped data structure referred to as the System Control Block (SCB). The layout for this structure is shown in Figure 2. The System Command Block consists of a command word, a status word, and a general pointer. Because the 8255x can interrupt the CPU for multiple events, the status word is checked to determine the cause of interrupts. The command word is used to mask device interrupts and send commands to the device while the value of the general pointer varies depending on the command being sent to the device. Various commands cause the device to activate, suspend, resume, or idle. The CU is primarily involved with frame transmission while the RU is primarily involved with receiving frames.

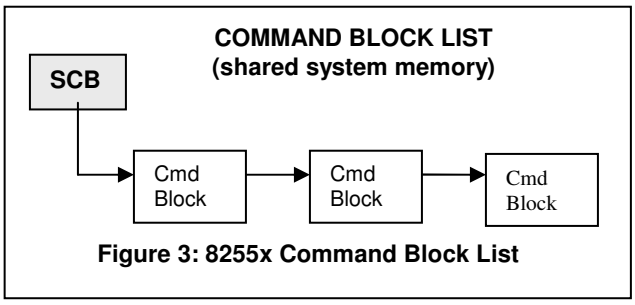

Figure 3: 8255x Command Block List

The CU's frame transmission function operates upon another data structure called the Command Block List (CBL). The CBL is a linked list data structure in shared system memory consisting of Command Blocks containing command parameters and status information. These blocks include diagnostic and configuration commands in addition to the transmit command. Figure 3 shows the layout of the command block list.

Transmitting a packet is, in fact, a fairly straightforward process. We must first construct 2 essential data structures: the data packet and the Transmit Command Block (TCB), a special type of Comand Block for the transmit command. The steps are outlined below and illustrated in Figure 4.

1. First, we (on an infected computer, "we" refers to the rootkit program) construct the data packet. Because we don't have access to the upper level NDIS or TDI drivers, this process must be performed manually. For simplicity, we chose to use the UDP protocol in our proof of concept implementation. Thus, the basic packet structure consists of an Ethernet header followed by an IP header, followed by a UDP header followed by the payload.
2. Second, we build a Transmit Command Block. The exact format of this data structure is contained in the Intel 82558 chipset documentation. Typically, the Transmit Command Block is followed in memory by the transmit data buffer.
3. After the data packet and Transmit Command Block are defined, we check the LAN controller to ensure that it is in an idle state and load its System Control Block's General Pointer field with the physical address of the Transmit Command Block.
4. Finally we initiate execution of the LAN controller by sending it a CU Start command. This causes it to begin executing the Transmit Command Block that will send the data packet out over the network.

Data exfiltration is highly stealthy because it does not require any long term detectable changes to any of the host Operating System networking components or data structures. Furthermore, there is no easy way to monitor the LAN controller on the x86 architecture because the 8255x data structures are addressed in physical memory. The x86 is capable of monitoring virtual memory accesses, but not physical memory accesses.

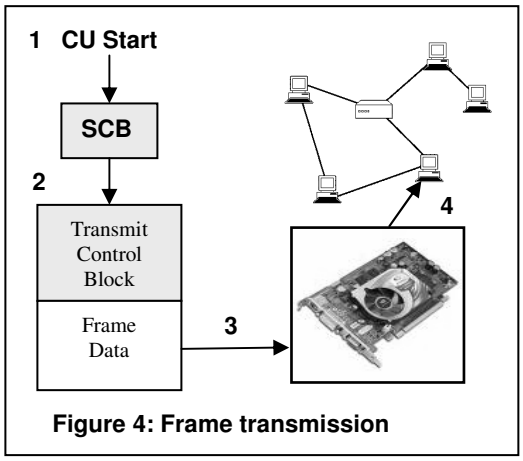

Figure 4: Frame transmission

3.2. Data Infiltration

In contrast to exfiltration, data infiltration refers to the process of receiving incoming data from an external source.

Packet reception on the 8255x is based upon the concept of a Receive Frame Area (RFA). The layout of RFA is shown in Figure 5. The RFA is a region of physical memory that is shared between the NIC and the CPU. It is subdivided into blocks called Receive Frame Descriptors (RFDs). The Receive Frame Descriptor is a data structure consisting of two parts: a header followed by a data buffer capable of holding the maximum Ethernet packet size. Every frame received by the NIC controller is described by one RFD. The RFD layout is shown in Figure 6. The NIC's RFA can be located by reading the "general pointer" field from the NIC's Status Control Block. The last RFD in the list is indicated by setting the EL bit.

Frame reception occurs when the device detects a frame on the link with an address that matches either the individual address, a multicast address, or broadcast address. It transfers the frame to the receive FIFO which in turn causes the NIC's receive DMA unit to transfer the frame to main memory on the host machine. Successful frame reception, in turn, causes the NIC to raise a Frame Receive (FR) interrupt on the host machine. The FR interrupt handler is responsible for extracting the RFD data, setting the appropriate

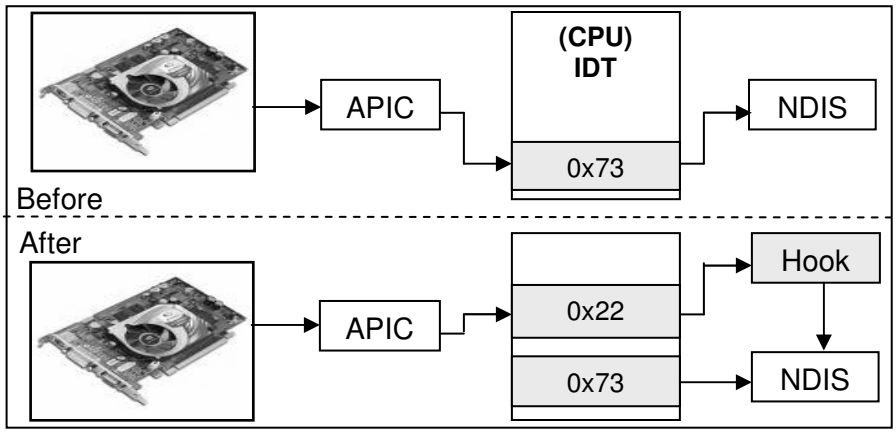

Figure 7: Interception of Packet Receive using IOAPIC Interrupt Redirection

status bits in the RFD header, and ensuring that it is passed to kernel and user components higher in the networking stack.

On Windows, during normal operating, the RFA is cooperatively managed between the Windows NDIS driver and the Intel Bus Driver (e100b325.sys). A malicious driver can circumvent the normal operation of packet arrival by inserting itself between the physical hardware interface and the Operating System. This is in contrast to previous stealthy network backdoors like Joanna's DeepDoor rootkit [5] which inserted themselves in NDIS, deep in the OS networking stack, yet still above the physical hardware interface. Our backdoor operates one level lower. By intercepting the NIC's FR interrupt that indicates packet arrival, we can inspect arriving frames prior to the OS or any firewall software running on the host machine.

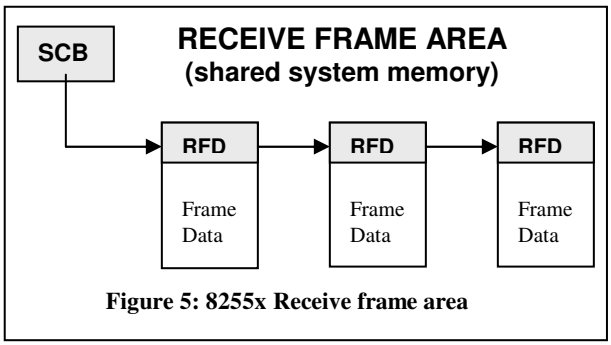

Figure 5: 8255x Receive frame area

Offset	Command Word						Status Word			
0x00	EL	S	00000000	H	SF	000	C	0	OK	Status
0x04	Link Address									
0x08	Reserved									
0x0C	0		0		Size		EOF	F	Count	

Figure 6: 8255x Receive Frame Descriptor

The Intel Advanced Programmable Interrupt Controller (APIC) is used to manage communication between the CPU, chipset, and external peripheral devices. When it receives interrupts, the APIC dispatches them to the processor, one at a time, based upon their priorities. The processor looks up the handler for the interrupt in the Interrupt Descriptor Table (IDT) [2]. Each interrupt is assigned a unique identifier, called a vector. The processor uses this value as an index into the IDT. The Interrupt Descriptor Table is a processor specific data structure containing one entry for each of 255 defined vectors. Kernel rootkits often use IDT hooking to intercept processor interrupts and exceptions [15]. This involves replacing the Operating System handler contained in the IDT with a pointer to a malicious hook routine.

When the LAN controller receives an interrupt, the APIC dispatches it to the CPU where it is looked up in the Interrupt Descriptor Table. Normally, the interrupt handler for the network card is managed by the Windows NDIS driver. We can intercept it by replacing the pointer with our own. Thus, when a packet arrives, we will receive the first notification and will be able to inspect the receive buffer prior to any Operating System software. Figure 6 illustrates this process.

This technique, however, can be detected by checking if the NIC interrupt in the IDT points to the OS where it should. To improve the stealthiness of our network backdoor, we can redirect the NIC's interrupt to another interrupt that is not being currently used by the OS. As mentioned previously, the IOAPIC's primary function is to receive and route peripheral hardware interrupts to the Local APIC for delivery to the CPU. For this purpose, the IOAPIC architecture defines a Redirection Table. The Redirection table contains a dedicated entry for each interrupt pin. It is used to translate the physical, hardware signal into an APIC message on the bus. This table can be used to specify the destination of the interrupt, the vector, and the delivery mode. We can therefore, change interrupt vector for the NIC and redirect it to a different, unused entry in the IDT. From this handler, after we inspect the incoming frame we can pass it on to the OS handler. Figure 7 illustrates this redirection technique.

The implementation of the process for monitoring incoming traffic can be described as follows:

1. Identify the interrupt for 8255x compatible network card.

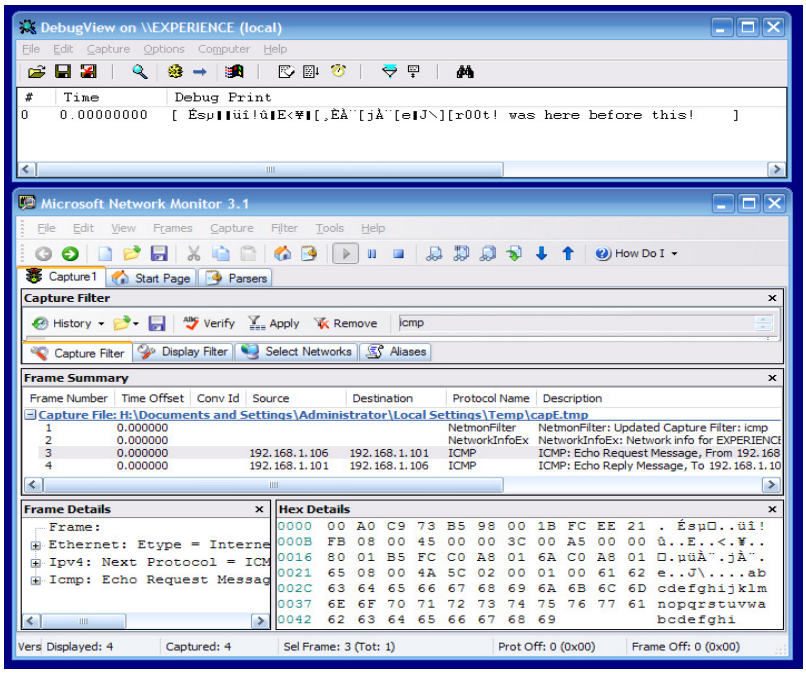

Figure 8: Packet Spoofing — Data Infiltration

2. Look up the OS handler for that interrupt in the CPU interrupt descriptor table (IDT) and save the pointer.
3. Locate an unused interrupt in the IDT and *hook* it by replacing the handler address with the address of our backdoor's handler.
4. Redirect the NIC interrupt to our new, hooked IDT vector by modifying the chipset's APIC Redirection Table.

When an interrupt from the NIC is received, the following steps occur:

1. Determine if the interrupt is due to a frame arrival (check bit in Status Control Block). If it's for some other reason, call the OS handler.
2. If the interrupt is due to frame arrival, locate the start of the Receive Frame Area (RFA) from the SCB General Pointer Field.
3. Scan the data portion of the Receive Frame Descriptors in the RFA. This scan is used to identify a "special" ICMP packet.
4. If a "special" packet belonging to the backdoor is identified, then erase it.
5. Else, pass control to the OS handler and let it process the packet normally.

4. TESTING

We tested our implementation against several popular firewalls and intrusion detection systems for both data infiltration and data exfiltration. These detection systems include the Windows XP Firewall, Zone Alarm Security Suite and the Snort Intrusion Detection System [7][8][9]. We tested on 2 different network cards, Intel Pro 100B and Intel Pro 100S, which are compatible with the Intel 8255X chipset. Both cards were installed in Intel Core 2 Duo systems running Windows XP SP2. The implementation should also work on other cards using the Intel 8255X chipset.

For all of the experiments, we used specially crafted ICMP packets containing the data payload "r00t was here before this!". This string serves as a form of "signature" in the packet receive interrupt handler to indicate that the packet is destined for the backdoor rather than the Operating System. In order to test data infiltration, we used a secondary laptop running the Network Packet Generator (NPG) program to craft these special packets. NPG is a free GNU GPL Windows Packet injector. It uses WinPcap to send packet out the network interface. The packets are defined in a packet file and it is possible to craft any kind of packet, regardless of headers or payload.

We conducted two sets of experiments, one on data infiltration and the other on data exfiltration. Our data infiltration experiments test the ability to intercept network packets before they reach any software firewall or Intrusion Detection System (IDS). Once intercepted, we must also prevent the OS or other security software from raising an alert. We tested 2 different approaches to this problem: packet spoofing and packet erasing. In the packet spoofing approach, we modify the malicious ICMP packet payload ("r00t was here!") in the receive interrupt handler so that it appears as the default Windows ICMP packet payload. Then we forward it to the OS normally. The destination address and MAC header information can also be spoofed before forwarding the packet. Although the OS receives the packet, it just looks like a normal ping. Because ping traffic is so common, it is unlikely for a rogue ping to raise any red flags. If ICMP traffic is blocked by an upstream router or firewall, another kind of packet could be used to relay information to the backdoor (For example, HTTP traffic is usually allowed).

Figure 8 illustrates the packet spoofing approach. The machine installed with network backdoor has the IP address of 192.168.1.101 and the name of "EXPERIENCE". The secondary laptop, with IP address of 192.168.1.106 and the name of "AMD32-LAPTOP", sends the specially crafted ICMP packet. We

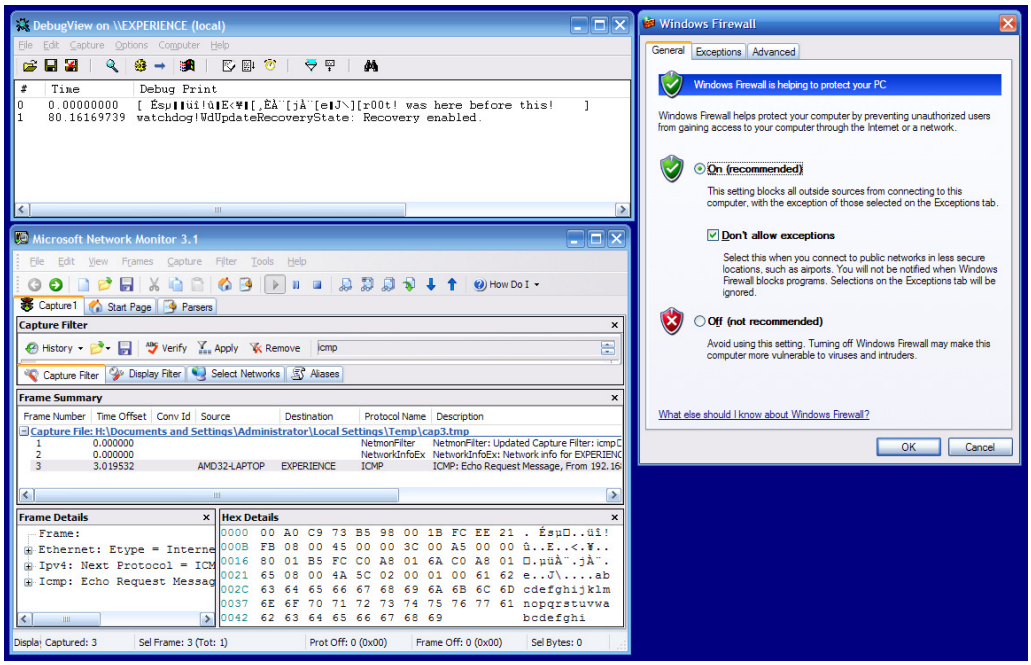

Figure 9: Windows firewall – Data Infiltration

use Network Packet Generator to send the malicious ICMP packets and we view them using Microsoft Network Monitor [22][16] on the laptop (the bottom window shown in this figure).

Note that the network monitor is not running on the machine with backdoor---it cannot capture malicious packets on a backdoor installed machine since the backdoor is designed to avoid detection.

The backdoor prints the packet headers and payload to Windows debug output for comparison (the top window shown on Figure 8 is the debug output). From the output, it is clear that the backdoor is installed and receiving the malicious packets, however, Microsoft Network Monitor reports it as a default ping.

The other approach we tested was packet erasing. In this method, we zero out the data portion of the receive frame descriptor including the MAC, IP, TCP, and ICMP headers. In this case, the OS drops the packet without sending it up the network stack. When we compared the backdoor's Debug output with Microsoft Network Monitor, we saw that Microsoft Network Monitor failed to report any kind of network activity, ICMP or otherwise. Because of this, packet erasing may be the stealthiest approach.

4.1. Data Exfiltration Test

In the data exfiltration test, the installed network backdoor program on the first machine sends out specially crafted ICMP packets to our secondary laptop. The laptop successfully received all these packets no matter which of those three tested detection systems was used. In addition, none of these detection systems provided any alert for such exfiltration traffic.

Windows XP Firewall does not monitor or block outbound traffic. Thus it cannot detect any data exfiltration attack. Zone Alarm and Snort have the capability to monitor and log outgoing traffic. However, because both of them rely on the network APIs provided by the Operation System for traffic monitoring, they cannot detect any exfiltration packets generated by our network backdoor, which is in the lower level than Operating System's network APIs as illustrated in Fig. 1.

4.2. Data Infiltration Test

4.2.1. Testing against the Windows XP firewall

We first looked at the Windows XP Firewall. We wanted to test our backdoor on the strictest settings. Therefore, we set the Windows firewall to block all outside sources from connecting to the computer and disallow any exceptions to that rull. We then used NPG to send a malicious ICMP packet to the backdoor and monitored the output of Microsoft Network Monitor and the backdoor's dump of the packet over the Debug output interface. The network backdoor used the packet erasing approach to conceal the packet from the OS.

Figure 9 shows the results of this experiment. The right window shows that the Windows Firewall is turned on to block all outside connections. The network monitor window on this figure shows that the secondary laptop has sent out an ICMP packet to the backdoor machine (the monitor runs on the laptop, not the backdoor machine). The packet is clearly successfully received by the backdoor despite the firewall's blocking-all policy (as shown in the "DebugView" window on this figure). This is evidenced by its output of the header and payload data. It also remains invisible to Microsoft Network monitor on the backdoor machine.

4.2.2. Testing against Zone Alarm

Zone Alarm is a software firewall and intrusion detection system. In addition to providing inbound intrusion detection, Zone alarm also monitors and controls the ability for programs to create outbound connections. It also contains a LOCK feature which allows the user to lock his / her computer so that applications can neither send nor receive data from the Internet or local network.

Like the Windows Firewall, we chose to test the backdoor under the strictest conditions, with the internet lock enabled. We also used the packet erasing approach. Figure 10 shows that the malicious ICMP packet was received by the backdoor, but that Zone alarm installed on the backdoor machine did not detect any access attempt.

4.2.3. Testing against Snort

Snort is an open source firewall and intrusion detection/ prevention system. It uses a rule based language. This gives it the flexibility to incorporate signature, protocol, and anomaly based detections. It is also the most widely deployed intrusion detection and prevention systems. We crafted a special rule file for Snort that logs all TCP packets and stores the logged information in a file called alert.ids. We verified its operation by first sending normal ping traffic using the Windows 'ping' command. After verifying that this traffic is correctly logged, we sent (via the secondary laptop) the malicious ICMP packets over TCP to the backdoor installed machine. The network backdoor used the packet erasing approach as well to conceal the malicious ICMP packets from the OS.

Like the previous two experiments, Snort failed to log this attack activity while these malicious ICMP packets are intercepted successfully by the backdoor. Due to the similarity, we do not use another figure to show this experiment result again.

Table 1 summarizes the testing results for data infiltration and data exfiltration by the proposed network backdoor.

Table 1: Testing results summary

Product	Monitors Incoming Traffic	Monitors Outgoing Traffic	Detects Incoming Backdoor Packets	Detects Outgoing Backdoor Packets
Win XP Firewall	YES	NO	NO	N/A
Zone Alarm	YES	YES	NO	NO
Snort	YES	YES	NO	NO

4.3. Network Backdoor Performance

It is difficult to estimate the performance of the network backdoor on the overall system. In order to estimate performance, it is necessary to obtain a comparison with the performance of the Operating System's network subsystem, which is a difficult task due to the architecture. It is also difficult to measure the execution time for the Operating System's network handling code because only a small portion of the code runs in the interrupt handler and the remaining majority is executed as deferred procedure calls which are scheduled to run asynchronously when CPU resources are not needed for critical tasks. Because the network processing code does not run continuously and linearly from start to finish, estimating actual execution time is problematic.

We can however, make the observation that the backdoor will add a relatively constant overhead to network processing. To measure that overhead, we calculated the number of clock cycles it takes to execute our backdoor's network interrupt handler using the CPU timestamp counter. On our test system, this averaged out to an overhead of approximately 4000 additional clock cycles per network packet received. Subjectively, however, this overhead did not produce any human detectable lag in performance, even while the network was subjected to heavy loads (such as downloading a large file).

5. DEFENSE

We have shown that it is relatively easy for an attacker to develop a network backdoor capable of evading a large number of popular, widely deployed firewalls and intrusion detection systems. The first problem lies in the fact that most of these systems fail to monitor outbound traffic. This deficiency may result in the leakage of potentially secure data and the delayed detection of malware threats like worms, and botnets. Support for extrusion detection would be a valuable addition to many commodity firewalls and intrusion detection / prevention systems.

However, the second issue concerns the reliable implementation of such support. Systems which rely upon the trustworthiness of the Operating System for monitoring network data may be easily spoofed using a variety of existing rootkit techniques (e.g. TDI / NDIS hooking). In general, malware becomes more stealthy and difficult to detect as it insinuates itself deeper in the OS and closer to the physical hardware. We take this paradigm to its logical conclusion by developing a network backdoor that operates at the network card chipset interface. Detecting such malware becomes a difficult problem for several reasons.

First, it is difficult for security vendors to operate at this level. Second, there is no network protocol stack support from the OS at this level. Finally, the hardware specific nature of the code becomes an obstacle to producing a generic, robust product. From the software side, we can break defense into two related challenges: detecting outbound traffic, and detecting inbound traffic. Between these two challenges, detecting inbound traffic is easier. This is due to the fact that in order to intercept incoming traffic, the malware must be able to intercept the card's frame arrival interrupt. If it hooks the OS interrupt handler directly, it will be detectable by the changes it makes to the Interrupt Descriptor Table (i.e. the pointer no longer points within the OS handler). Our method of redirecting the interrupt at the IOAPIC redirection table increases its stealth because we are not directly hooking the OS interrupt handler for the network. Instead, we take an unused interrupt and reprogram the chipset to interrupt on the new vector. In addition to scanning the IDT for changes, security software should also check chipset level data structures, like the APIC redirection table, for suspicious modifications.

Detecting outbound traffic is more difficult. This is due to the fact that the malicious code does not need to make any permanent changes to the OS or architectural data structures (e.g. the IDT) to send data frames out over the network. It merely needs to know the location of the card's shared memory space and write to a few registers on the card. If one were able to detect and validate reads and writes on the shared memory region of the card, it might be possible to monitor outgoing traffic. Unfortunately, the card addresses memory physically rather than virtually and the x86 does not support monitoring physical memory accesses.

In order to detect packet erasing approach used by the proposed backdoor, a host must cooperate with a network firewall/gateway that take charge of this host's incoming traffic. The network firewall could provide the exact number of packets incoming

targeting the host. By comparing the host's monitored number of incoming packets, the host could possibly detect if there are some packets being erased or not. This approach does not place much burden to network firewalls and should be able to be implemented without much difficulty.

The best software option may, in fact, be moving the firewall into a virtual machine monitor (VMM) with support for I/O virtualization. The new Intel and AMD CPU's have the hardware support for this [25]. This would allow the virtual machine monitor to receive notification on hardware accesses and validate them accordingly. In addition, VMsafe from VMware [29] and the XenAccess [30] provide software based virtual machine monitor platform. Alternatively, the Operating System could provide a trusted virtual machine monitor that abstracts critical components like the networking hardware and provides an interface to kernel drivers.

Another hardware supported defense is to use virtualization for directed I/O. For example, Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts [26], thus only memory blessed by the OS can be accessed by devices for DMA.

The best defense, however, is likely to be a hardware firewall capable of inspecting and blocking outgoing traffic. A hardware firewall will be immune to the attacks discussed in this paper; however, detecting malicious outbound traffic is still likely to pose challenges if it is encrypted or obsfucated using steganographic techniques.

Finally, we can rely on network-based intrusion detection systems (NIDS), to detect the backdoor, or any other rootkit secret traffic since they do not rely on host's integrity for malicious traffic detection. The drawback is that a network-based detection system only has packet-level monitoring capability without any knowledge of host-level information, which makes it difficult to detect advanced malicious activities that hide with either encryption or embedded within normal traffic.

6. CONCLUSIONS & FUTURE WORK

The greatest limitation of our implementation is the fact that its hardware and chipset specific. Our implementation is limited to cards using the Intel 8255x chipset. While this may appear to limit the usefulness of such an attack, the Intel 8255x is a widely deployed chipset that is compatible with a large number of network cards. The complexity and level of effort required to implement chipset specific malicious code places this type of attack out of the reach of most casual hackers and malware developers. It is more suited to advanced, targeted attacks where the attackers are capable of investing considerable resources in terms of time and money. Such attacks are likely to be profit driven goal oriented, and target specific as in the case of economic espionage or cyber terrorism attacks.

The range of the threat would be increased if one were able to target a wider subset of commodity networking hardware. For example, our network backdoor might be extended to support the Intel Centrino wireless network card specification. Since Intel Centrino mobile platform defines a built-in wireless capability, it has created a homogenous networking environment for Intel laptops. Were adversaries capable of creating a similar backdoor to the one proposed in this paper that would work on the Intel Centrino chipset, it would greatly reduce the hardware specific limitation and greatly increase the threat. Intel has not published the specifications for the Intel Centrino wireless chipset, but experienced adversaries could reverse engineer the Intel drivers to figure out how it works. This is one potential area of future work we'd like to persue. We intend to study this wireless chipset and figure out whether it is possible and whether it is easy for adversaries to produce such a backdoor.

Another limitation of our current design is that it is non persistent. Non persistent malware is incapable of persisting across reboots. To persist across reboots, malware must usually have some method of gaining control of execution during the boot sequence so that it can install itself. It must also have some means of storing itself on non-volatile media (e.g. a hard disk) so that it can be loaded into volatile RAM. Adding the persistent capability to a malicious application decreases its stealth and becomes another vector for detection because many anti virus and security applications inspect the boot process and scan non persistent media like hard disks. In practice, non persistence is not a big limitation. Many servers remain active for weeks or months at a time between reboots. This is almost certain to give an attacker adequate time to inspect and exfiltrate sensitive information from the target computer.

In conclusion, our design and implementation serves to highlight two important weaknesses in commodity host-based firewall and intrusion detection technology: the lack of support for outbound traffic monitoring and a continuing reliance on the trustworthiness of a potentially compromised OS.

7. ACKNOWLEDGMENTS

This work was supported by NSF Grant CNS-0627318 and Intel Research Fund.

8. REFERENCES

[1] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2. May 2007.

[2] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1. May 2007.

[3] Intel Corporation. Intel 8255x 10/100 Mbps Ethernet Controller Family: Open Source Software Developer Manual, January 2006.

[4] R. Bejtlich. Extrusion Detection: Security Monitoring for Internal Intrusions. AddisonWesley, first edition, 2006.

[5] Joanna Rutkowska. "Rootkits vs. Stealth by Design Malware", Presented at Black Hat, Europe 2006.

[6] Alexander Tereshkin. "Rootkits: Attacking Personal Firewalls", Presented at Black Hat USA, 2006.

[7] Windows XP Firewall. http://www.microsoft.com/windowsxp/using/networking/security/winfirewall.mspx

[8] Zone Alarm. http://www.zonealarm.com/store/content/home.jsp

[9] Snort. http://www.snort.org/

[10] AOL/NCSA Online Safety Study. Conducted by America Online and the National Cyber Security Alliance. Dec. 2005.

[11] Microsoft Corporation. Windows XP Firewall.

[12] Chipset. http://en.wikipedia.org/wiki/Chipset

[13] Gramm-Leach Bliley Act. http://www.ftc.gov/privacy/privacyinitiatives/glbact.html

[14] Payment Card Industry Data Security Standard. https://www.pcisecuritystandards.org/
[15] J. Bulter and G. Hoglund. “Rootkits: Subverting the Windows Kernel.” Addison Wesley. 2005.
[16] W. Cui, R.H. Katz, and W. Tan. BINDER: An Extrusion-based Break-In Detector for Personal Computers. In 2005 USENIX Annual Technical Conference. 2005.
[17] Salvador Mandujano. “Identifying Attack Code through an Ontology-Based Multiagent Tool: FROID.” In Proceedings of the World Academy of Science, Engineering, and Technology, June 2005.
[18] F. Bellifemine, A. Poggi, and G. Rimassa. “JADE – A FIPA-compliant agent framework.” In Proceedings of Practical Applications of Intelligent Agents, 1999.
[19] K. Borders and A. Prakash. “Web Tap: Detecting Covert Web Traffic”. In ACM Conference on Commputer and Communications Security. 2004.
[20] Y. Zhang and V. Paxson. “Detecting Backdoors”. In Proceedings of the 9th USENIX Security Symposium. August, 2000.
[21] NDIS. http://en.wikipedia.org/wiki/Network_ Driver_Interface_Specification
[22] Network Packet Generator. http://www.wikistc.org/wiki/Network_packet_generator
[23] Greg Hoglund. “A *REAL* NT Rootkit, patching the NT Kernel”. In Phrack Vol. 9, Issue 55. 1999.
[24] J. Heasman. Implementing and Detecting an ACPI BIOS Rootkit. Presented at Black Hat Federal, 2006.
[25] x86 virtualization. http://en.wikipedia.org/wiki/X86_virtualization
[26] Intel® Virtualization Technology for Directed I/O. http://www.intel.com/technology/itj/2006/v10i3/2-io/7-conclusion.htm
[27] Extrusion detection. http://en.wikipedia.org/wiki/Extrusion_detection
[28] D. Whyte, P. Oorschot, E. Kranakis. Exposure Maps: Removing Reliance on Attribution during Scanning Detection. USENIX HotSec 2006.
[29] VMware VMsafe Security Technology. http://www.vmware.com/technology/security/vmsafe.html
[30] XenAccess Library. http://code.google.com/p/xenaccess/