On the Insecurity of Personal Firewall

Raymond Chiong and Sandeep Dhakal
School of Computing & Design
Swinburne University of Technology (Sarawak Campus)
State Complex, 93576 Kuching,
Sarawak, Malaysia
rchiong@swinburne.edu.my, sandeep.dhakal@gmail.com

Abstract

Personal Firewall is a popular and common installation choice on personal computers nowadays. It is expected to provide higher Internet security by employing Internet access controls. In this paper, we discuss the insecurity of most personal firewalls on Microsoft Windows platform and the methodologies to bypass them. We show that personal firewalls can be bypassed using various methods at different layers in Windows Network Architecture. We reckon that in order to better secure personal computers, more effective methods have to be implemented to enforce the security strength of personal firewalls. In addition, personal firewalls should be used in conjunction with good antivirus programs to provide in-depth security defenses.

1. Introduction

The Internet provides great connections around the globe. However, the open environment of the Internet causes many potential security problems at the same time. With the wide availability of the globally connected networks, there are always some parties trying to get into others' computers seeking sensitive data, spoiling the services, writing and releasing viruses and worms, and so on. A firewall is therefore becoming very important in this situation.

Nowadays, most of the companies and organizations install firewalls between their own networks and the public network, enforcing access control policies which define what traffic is allowed and what is not. This ensures that their networks are well-protected from any kind of hacking activities. Besides companies and organizations, there are also more and more personal computers connected to the Internet. Following this trend, much attention from the attackers has been redirected to the personal users. As a result, more and more personal users suffer from Trojans, spyware as well as other Internet hacking activities [1]. There is thus an increasing public awareness for Internet security, that the single point of access should not be the only place to take security measurement, but every vulnerable computer connecting to the Internet. Inspired by traditional firewall, the concept of personal firewall has attracted the security professionals, companies and customers [2]. A personal firewall is installed on personal user's computer and enforces access control between the Internet and the personal computer.

Although personal firewalls are expected to guard the computers from various hacking attempts, this is an illusion built on tests that did not take into account the real world, but were tested straight out of the box on a clean personal computer [3]. There are many underlying issues that are not identified, such as the false security where the computer has to learn from the users who are exhausted from answering questions they do not really understand, and most of the time the users simply turn off or ignore the security alerts, rendering the theoretical protection useless. Besides that, firewall can also be disabled by certain viruses or worms - a security loophole that cannot be closed.

In this paper, we carry out a study on the personal firewall on Microsoft Windows platform to understand its working principles and also ways to bypass it. We show that the personal firewall is far from a perfect solution and users will never be 100% secure from someone who really wants to penetrate their computers.

2. Personal Firewall

A firewall is a system or group of systems that enforces an access control policy between two networks [4]. Ideally, a decent personal firewall should include the following essential functions:

- *Packet Interception*: for a personal firewall, it is the basic requirement that it should intercept all traffic to and from the computer.
- *Packet filtering*: personal firewall should decide whether to allow the traffic based on preset rules. Rules should be flexible and powerful enough to fulfill different requirements.
- *Event Logging and Alert*: personal firewall should record all significant or suspicious events in the log file for future consultation. It should also give real time alert to notify the user of the serious activities.
- *Stealth*: make the computer invisible to others while it is online, reducing the chance of being attacked.

Additional features may include cookies control, content filtering, intrusion detection, etc. Among these, packet interception is the most basic function of a personal firewall. This essentially determines how powerful a personal firewall can be. To understand how a personal firewall intercepts packets, we must first look at the Microsoft Windows Network Architecture.

2.1. Windows Network Architecture

Basically, the Windows Network Architecture consists of four layers: Network API Layer (Winsock API and others), Transport Data Interface (TDI) Layer (afd.sys and others), Network Protocol Layer (TCP/IP stack and others), and Network Driver Interface Specification (NDIS) Layer. Figure 1 shows these basic layers.

Layer	Example	
Applications	Internet Explorer	
Network API	Winsock	*User mode*
TDI	(interface)	*Kernel mode*
Network Protocol Layer	TCPIP.SYS	
NDIS	(interface)	

Fig. 1. Layers of Windows Network Architecture [5].

Winsock is a protocol-independent API that allows Windows based applications to access the transport protocols. It can be extended with other protocol/socket types. It is worth mentioning that Microsoft has controversially added raw IP capability to the Windows Sockets in Windows 2000/XP, which means that applications can easily craft (malicious) TCP/IP packets and inject them into the network.

Winsock does not access the transport protocols directly, because the transport protocols do not have a sockets-style interface which applications can talk to directly. Instead, the transport protocols implement a much more general interface called the TDI. The generality of this API brings the advantage that the Windows does not have to stick to particular network programming interfaces. A socket's emulation is provided by the Winsock kernel mode driver afd.sys. This driver is responsible for the connection and buffer management, providing a sockets-style interface to an application. To access the transport protocols, Winsock talks to afd.sys, which in turn talks to the transport protocol drivers through TDI.

The Network Protocol Layer provides data transmission services for TDI clients, by implementing the set of functions and call mechanisms described by TDI. Protocols are standard for packet format and transmission that makes it possible to share information through network. Many protocols are supported in Windows platform, including TCP/IP, ATM, NetBEUI, AppleTalk, etc. It is also called transport protocols layer or protocol driver layer.

The lowest layer is the NDIS layer. It is a specification that allows network protocols to communicate with the underlying physical network adapter. All network adapter drivers must conform to the NDIS specification, which provides a general interface to the network protocols. As long as a network protocol conforms to NDIS, it can communicate with all network adapters that Windows supports.

2.2. Packet Interception Techniques

By examining the Windows Network Architecture from top down, it is perceptible that there is more than one place where packet interception/filtering functions can be achieved. This section explores the available packet interception techniques.

2.2.1. Layered Service Provider. A Layered Service Provider (LSP) is part of the Microsoft Windows Winsock 2.0 interface. It is often used by spyware and adware vendors [6]. For example, an optional LSP layer can be inserted between the Winsock 2.0 DLL and the underlying protocol stack. This allows the attacker to forward all of the user's traffic to an unauthorized external site, where it is data-mined to find the user's special interests to bombard him/her with targeted advertisements, as well as spam e-mail. Additional services like authentication, encryption and packet interception can be provided in this layer. However, it is still possible for an application to call the kernel mode TCP/IP protocol driver directly via the TDI interface, thus bypassing the packet interception service in the LSP layer.

2.2.2. Winsock DLL Replacement. Before the introduction of Winsock LSP facility, the only way to extend the functionality of Winsock was to replace certain Microsoft-provided Winsock DLLs with replacement DLLs [6, 7], either statically or dynamically. It is difficult to develop a robust Winsock replacement DLL mainly because Microsoft Winsock DLL includes private internal support functions that are not documented. The replacement DLL must deal with at least some of these undocumented functions. This mechanism has the same problem as the LSP mechanism that it intercepts the packets at a too high layer and applications can easily bypass it.

2.2.3. Kernel-mode Sockets Filter. Another way to intercept all sockets traffic is to intercept all calls from the msafd.dll, which is the lowest level of user mode Winsock DLLs, to the afd.dll, which is a kernel mode DLL for sockets emulation of the underlying protocol drivers [8]. However, though this approach is at a lower layer than the LSP layer, it is no better than LSP since it is still above the TDI interface. Applications can still bypass it by accessing TDI directly.

2.2.4. Packet Filtering with Iphlpapi.dll. Windows 2000 and later offer a finer degree of programmatic control over TCP/IP, including the ability to perform packet filtering. The iphlpapi.dll provides API which can be used by user-mode applications to install a set of 'filter descriptors' [9]. The TCP/IP protocol driver will use it for packet filtering. Unfortunately, this new API is not documented very well, and rules for filtering are rather limited. This serious limitation makes it practically useless for a personal firewall.

2.2.5. Global Hook. Windows hook mechanism is a powerful tool for many purposes as Windows relies heavily on message-exchanging for intercommunication between procedures and modules to control their behaviour. For the purpose of packet interception, global hook can be installed for all network-related functions, so that any network-related activity can be monitored [7]. Although theoretically feasible, this approach can be rather difficult. A global hook that has a bug may result in an unexpected impact on the system stability, therefore likely to have a significant negative impact on the system performance.

2.2.6. TDI Filter Driver. TDI clients talk to the protocol drivers which are bound to them through the TDI filter driver interface [8]. If all calls through TDI are intercepted, then all packets passing through TDI between the TDI clients and protocol drivers can be intercepted.

Another approach is to hook the TDI device objects by using IoAttachDevice() [7]. It attaches the caller's device object to a target device object, so those I/O requests to the target device are routed first to the attached device. By attaching a filter object on the top of every transport driver device object, a filter layer is virtually inserted between the TDI transport drivers and clients, which intercepts all calls to the transport drivers [10].

The main advantage of this TDI filter driver technique is that it provides stateful TCP connections inspection based on real operating system TCP/IP stack. The disadvantage is that the personal firewall that operates at this level cannot filter packets before TCP/IP stack, which means at least ICMP and other network layer packets can get around the firewall.

2.2.7. IP Filtering Driver in RRAS. Starting from Windows NT 4.0, there is an integrated service providing both remote access and multi-protocol routing called the Routing and Remote Access Service (RRAS). This service includes an IP Filtering Driver which provides the Windows routers the capability to specifically allow or disallow the flow of a particular type of IP traffic [11]. Theoretically, all IP packets have to pass through it. However, this API is not well documented by Microsoft. Its filtering function is also very limited. It can only filter IP, TCP, UDP and ICMP packets based on some but not all fields of these protocol headers.

The functionality of the IP Filtering Driver can be extended by a kernel mode driver, the filter-hook driver. However, this is only available on Windows 2000/XP [12].

2.2.8. NDIS Filter Intermediate Driver. NDIS Intermediate Driver (NDIS IM Driver) is an optional layer which interfaces between the protocol drivers and NDIS miniport drivers. One of the design purposes of NDIS IM Driver is exactly, packets filtering (see [13]). This filter driver can do everything to the packets before sending them to or after receiving them from the underlying NDIS miniports. This even includes encryption and compression.

The NDIS IM Driver is a powerful technique for personal firewall applications. However, it is not widely used currently due to the implementation difficulties and installation inconvenience. In Windows 2000/XP, the IM driver has to be digitally signed at Microsoft, otherwise the use of it will get a warning message claiming that the driver may have unpredictable impact on the system stability. In the future, it is foreseeable that this powerful filter driver may be more widely employed given the better support in the later Windows versions such as Vista.

2.2.9. NDIS Hooking Filter Driver. The NDIS is essentially a library that exports a full set of functions, for communication between miniports and transport protocol drivers, and for the miniports to control the physical Network Interface Controller (NIC). The miniports and transport protocol drivers in turn have to expose a set of entry points (functions) which NDIS calls for its own purposes or on behalf of transport protocol drivers or miniports [13]. This actually makes the NDIS a layer lying between the miniports and protocol drivers. Thus, if it is possible to install a hook for the NDIS, then the hook should be able to effectively intercept all communication between the miniports and protocol drivers.

In Windows 2000/XP, NDIS is implemented in ndis.sys. There are two approaches to implement NDIS hook in Windows 2000/XP, namely the Export Table Modification and the Dummy Protocol Registration.

For Export Table Modification, all sys files are in Portable Executable (PE) format in Windows 2000/XP. The export table is part of the PE, which contains the addresses of the exported functions. If the address of a particular function in the export table is modified to the address of the hook function, whenever this function is called, the call is actually directed to the hook function first. To install an NDIS hook driver for a personal firewall, only several functions which are exported by ndis.sys are necessary to hook. The hook driver should be loaded at a start stage of the operating system and cannot be unloaded from memory later. The Windows 2000/XP also protects a kernel image in the memory from possible modification, so the hook installation also needs to modify the registry to disable this protection.

For Dummy Protocol Registration, when a protocol driver registers itself with the NDIS library in Windows 2000/XP, a NDIS_PROTOCOL_BLOCK structure is returned. By registering a dummy protocol, all the structures are retrieved and then necessary modification can be applied to install the hook driver. This approach can be done in any stage of the system, and the hook driver can be unloaded.

Using NDIS Hooking Filter Driver has many advantages. It is flexible and concise, and provides high performance. It is also powerful and hard to bypass because it works in possibly the lowest level in Windows Network Architecture. The hook driver can do everything to the packets and is relatively easy to install. All these advantages make it a widely used technique for personal firewalls. However, some structures used in this technique are not well documented. The implementation also differs on different version of Windows.

3. Practical Implementations

Although there are many packet interception techniques available, not all of them are suitable for a personal firewall, especially for a commercial product. For a commercial personal firewall, it should be capable of intercepting everything from and to the Internet. This section examines what practical functions are available in today's personal firewalls, and how these functions are achieved. Three different personal firewalls will be discussed, namely Microsoft Internet Connection Firewall (ICF), ZoneAlarm Pro[1] and Outpost Firewall Pro[2]. These products representatively include all functions worthy of discussion.

As introduced earlier, there are many places in the Windows Network Architecture where packet interception can be achieved. Generally, the lower level a personal firewall operates at, the more packets it can intercept, and the harder it is for other applications to bypass it. The actual level a personal firewall works at may be inferred from its behaviour.

Microsoft ICF is a 'starter' firewall with basic functionalities. It generally operates at IP layer and blocks all communication initiated by the outside network by default. Nevertheless, the setting can be changed and new rules can be set up to allow some particular services to pass the ICF.

The basic functionalities of ICF are acceptable for a 'starter' personal firewall, but not sufficient for a more serious personal firewall as it does not get packets other than TCP/IP and is not too hard to bypass. It is observed that ZoneAlarm and Outpost Firewall work at a much lower level below the system TCP/IP stack, and also they are working at multiple levels to provide better security. They employ TDI level packet

[1] http://www.zonelabs.com

[2] http://www.agnitum.com

interception to provide application control, which is now recognized as a very important function of a personal firewall. This is because TDI is considerably the lowest level where process information is still available.

It is further observed that the emphasis of ZoneAlarm and Outpost personal firewalls are different. Generally speaking, ZoneAlarm is more application-concerned, which means it looks after every application started. On the other hand, Outpost is more packet-based.

For the three personal firewalls, they all share one capability, which is to be set for each single network connection. This feature is quite useful in practice, although there are some slight differences between the implementation of these different firewalls.

4. The Bypassing Techniques

There are many techniques to bypass the personal firewalls. This is mainly because personal firewalls are just software, in which they have to allow some traffic or applications, and their architectures to date are not perfect. In this section, we discuss some of the bypassing techniques in theory.

4.1. NIC Adapter Driver

A straightforward idea to bypass any personal firewall will be to make the program work at a lower level than the firewall. Since commercial personal firewalls run at NDIS level, between NIC driver and protocol driver; if the program works at NIC driver level, it should be able to bypass the personal firewalls [14]. Although theoretically feasible, the Trojan written for a particular NIC will not work for others, so in practice this is not a good idea.

4.2. Prevent Loading

Sometimes personal firewalls store information in the registry. By modifying the registry, it may be possible to prevent some firewalls from loading after reboot [15]. However, as this requires rebooting the computer, the users are likely to notice the absence of their firewalls.

4.3. Uninstall

Since personal firewalls are software only, it is possible to ‘bypass’ by uninstalling them. There are already some hacking programs in the market that are specifically designed to uninstall some well-known firewalls (see [16]). Certainly, these hacking programs need to know the personal firewalls in advance in order to uninstall them. As a counter-measure, some firewalls explicitly ask for confirmation when they are being uninstalled or their services are stopped.

4.4. Protocol Tunneling

Protocol tunneling is a technique to wrap a protocol into the payload of packets of another protocol [17]. This, however, requires support from both sides of communication, so that they can understand the payload of packets correctly. It is well known as a technology for Virtual Private Network (VPN), but can also be used by Trojans to bypass firewalls.

4.5. ACK Tunneling

ACK Tunneling is actually quite different from protocol tunneling even though the name is similar. Normally, a TCP connection is set up by a three-way handshake. Relying on this fact, some firewalls only apply rules on packets with SYN flag set. A Trojan can exploit this by communicating only using ACK packets, thus the name ACK Tunneling. What makes it better for the Trojan is that there is a high possibility that these packets are not logged, because the firewalls may only log the starting SYN packets [18].

4.6. Application Masquerade

The firewall does not block all the traffic. There are always some applications which are allowed or even trusted for Internet access. So by masquerading to be permitted applications, a malicious program can get around the personal firewall and communicate with outsider [19]. This used to be quite an effective technique as simple as rename, but today’s commercial personal firewalls identify programs not only by names but also the encrypted signature, thus making masquerading hard to work.

4.7. Application Control

Sometimes it is possible for a program to launch a firewall-permitted application and use its function to send and receive message, provided that it is able to control the behaviour of the application. For example, a program can simply launch a hidden Internet Explorer with a parameter URL which contains messages to send, and gets reply message from the Internet Explorer window title (see [20]). Simple as it is, this method is very effective in practice since most users will set their firewalls to allow Internet Explorer to access Internet.

4.8. DLL Injection

DLL Injection is a technique used to run code within the address space of another process, by forcing it to load a dynamic-link library. It relies on the powerful feature of Windows hook that, if one places an intercept function inside a DLL, that DLL then gets loaded up with any subsequent application and is treated as being in the same process space as that application [21]. This means it is possible for a Trojan to masquerade as a component of the permitted application and thus get permission to access Internet. DLL injection allows the Trojan to use not only web browser, but also any other application which access Internet, such as Outlook Express. The hook function can do virtually anything and can be loaded up by applications already accessing Internet, which makes it even more difficult to notice.

4.9. Parallel Network Stack

It is noticed that network-monitoring tools [22, 23] are generally ignored by personal firewalls. A program that works similar to network monitoring tools can therefore bypass personal firewalls in two phases. First, it comes with its own network stack that is parallel to system network stack. The parallel network stack function can be as simple as raw packet generation and received packet forwarding. Second, even if the direct communication with the NIC driver is inspected by the NDIS level filter of personal firewall, it can still get around it with the help of protocol tunneling.

5. Experiments

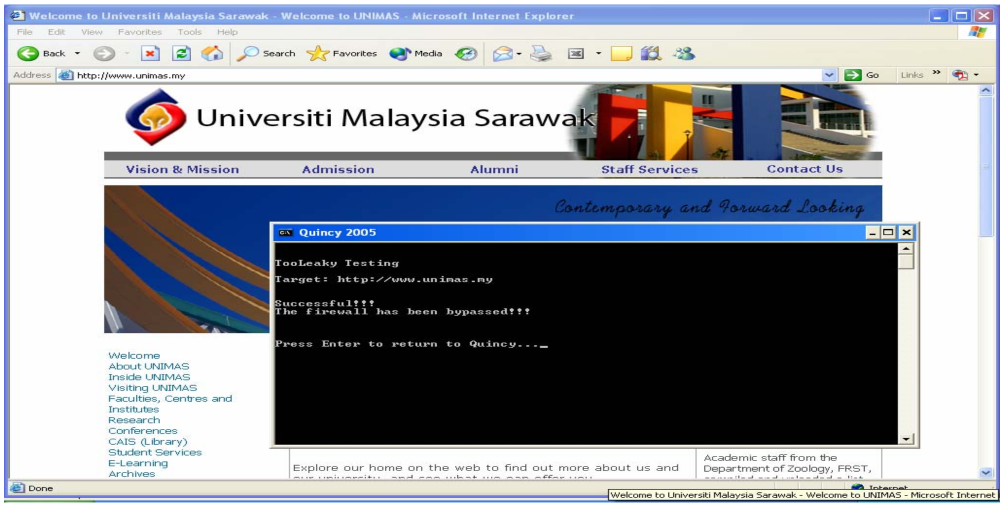
While many firewall-bypassing techniques have been discussed in the previous section, not all of them are of practical value. For experimentation purposes of this paper, we are more interested in bypassing the firewall by exploiting the weaknesses of its working principles. For this reason, application related bypassing and parallel network stack are selected for our implementation testing.

5.1. TooLeaky.NET

It was mentioned earlier that we can bypass personal firewalls by simply launching a web browser and controlling its behaviour. To our best knowledge, no personal firewall can defend effectively against this attack. In our first experiment, we wrote a simple program based on the well-known "TooLeaky" [20] using C. Figure 2 below shows the pseudocode of our program, and Figure 3 is the result.

```
locate Internet Explorer executable;
create an instance of ProcessStartInfo for IE executable;
compile ProcessStartInfo.Arguments from the user input;
set ProcessStartInfo.WindowStyle to Normal;
start a new process using the instance of ProcessStartInfo;
loop to check the MainWindowTitle of the new process until it
changes or timeout;
show the new MainWindowTitle;
```

Fig. 2. TooLeaky.NET pseudocode.


Welcome to Universiti Malaysia Sarawak - Welcome to UNIMAS - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.unimas.my Go Links
Universiti Malaysia Sarawak
Vision & Mission
Admission
Alumni
Staff Services
Contact Us
Contemporary and Forward Looking
Quincy 2005
Welcome
About UNIMAS
Inside UNIMAS
Visiting UNIMAS
Faculties, Centres and Institutes
Research
Conferences
CAIS (Library)
Student Services
E-Learning
Archives
Explore our home on the web to find out more about us and
Academic staff from the Department of Zoology, FRST,
Done
Welcome to Universiti Malaysia Sarawak - Welcome to UNIMAS - Microsoft Internet

```
TooLeaky Testing
Target: http://www.unimas.my

Successful!!!
The firewall has been bypassed!!!

Press Enter to return to Quincy..._
```

Fig. 3. TooLeaky.NET screenshot.

5.2. DLL Injection

TooLeaky can only initiate an outbound connection, thus bypasses only outbound detection. This means the target URL has to be hard-coded in the program and can be traced. On the contrary, the DLL injection technique introduced in [21] can wait for the inbound connection and perform masquerading with little or no notice from the firewall or user.

DLL injection relies on the powerful feature of a hook. The hook procedure DLL is normally injected into the process space of the hooked application. There is no difference between the legal components and the hook procedure from the system's point of view. Although the hook is intended for message processing, the hook procedure for bypassing personal firewalls can just ignore the message handling and do its own business. Obviously, anything can be done in the hook procedure, including both initiating outbound connection and listening for inbound connection.

As the sample executable provided in [21] only tries to initiate outbound connection, we wrote a simple program using C++ to listen for incoming connection. Figure 4 below shows the pseudocode for our DLL Injection program, while Figure 5 is the result.

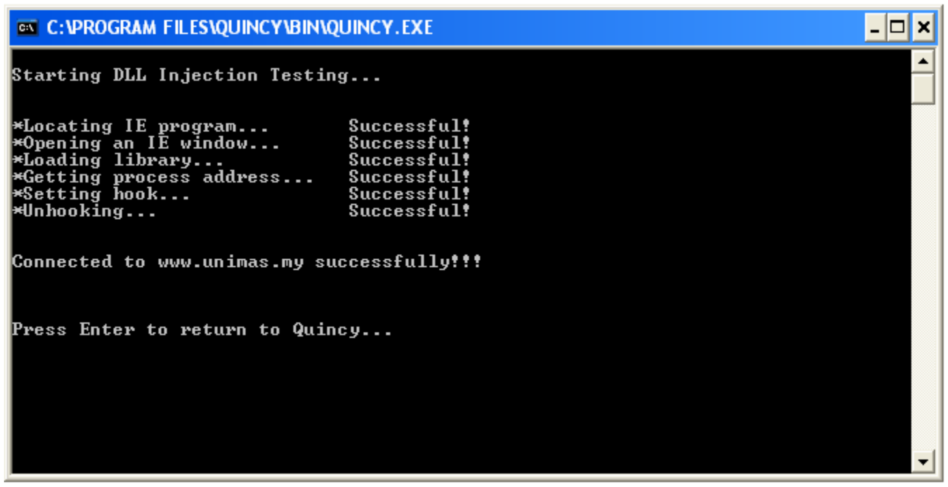
```
Main:
{
  open and query a registry key;

  if(key value is 0)
  {
    locate IE executable;
    compile STARTUPINFO;
    start a new process;
    load DLL module;
    get hook procedure address;
    set hook for the new process;
    wait for a second and unhook;
  }
  else
    prompt user of the existing hook;
}
```

```
Hook Procedure:
query the registry key;

if key value is 0
{
    WSAStartup;
    set key value to 1;
    create a socket;
    connect to a remote host;
    indicate connection result;
    close socket;
    create a new socket;
    bind and listen;
    accept connection;
    indicate result;
    close socket;
    WSACleanup;
    set key value to 0;
    end process;
}
```

Fig. 4. DLL Injection pseudocode.



```
Starting DLL Injection Testing...

*Locating IE program...       Successful!
*Opening an IE window...      Successful!
*Loading library...           Successful!
*Getting process address...   Successful!
*Setting hook...              Successful!
*Unhooking...                 Successful!

Connected to www.unimas.my successfully!!!


Press Enter to return to Quincy...
```

Fig. 5. DLL Injection screenshot.

5.3. Parallel Network Stack

As discussed earlier, an application with a parallel network stack which talks to the NIC driver directly should be able to bypass personal firewalls in two phases. There is actually an implemented packet capture driver called WinPcap [24], published under the GNU Lesser General Public License as free software. WinPcap works parallel to the existing system TCP/IP network stack. It includes a kernel mode Packet Capture Driver, a user mode low level dynamic library, and a user mode high level static library. The architecture of WinPcap is shown in Figure 6.

WinPcap is a free software library and widely used to create network monitoring tools. As it has the ability to receive and send packets, it is also a perfect tool for bypassing personal firewall using parallel network stack. In our final experiment, we wrote another program using C++ that can send packets using WinPcap. Figure 7 shows part of the pseudocode of this program, and Figure 8 is the result.

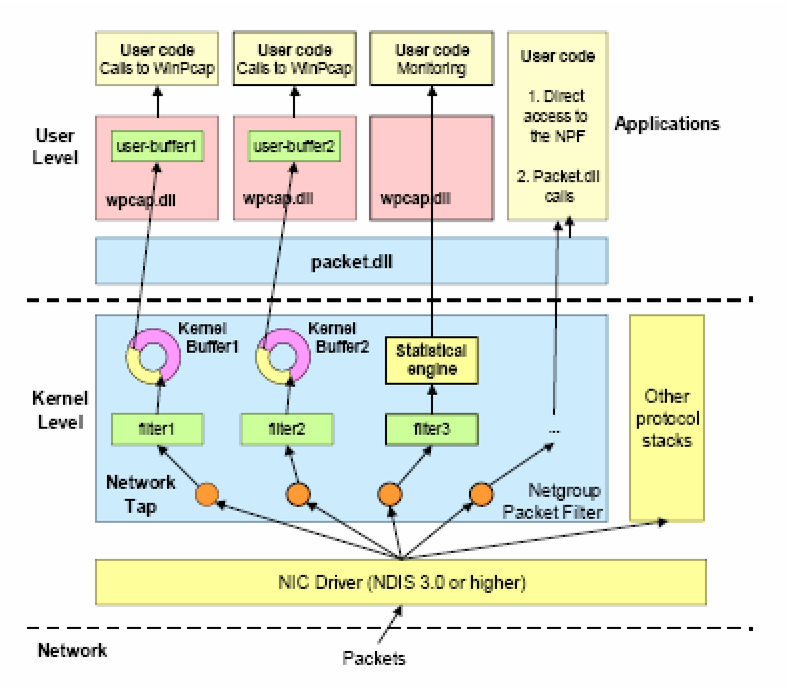
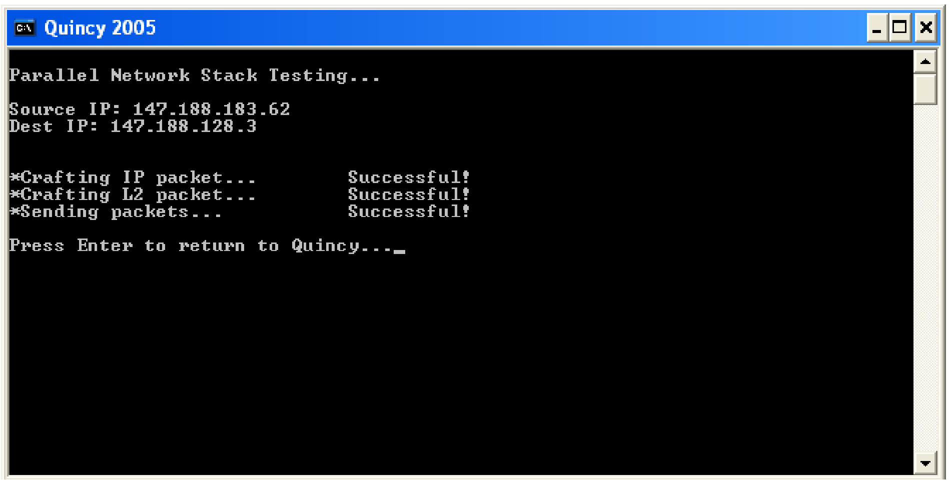


Fig. 6. The WinPcap packet capture architecture [25].

```
Main:
{
  protocol = packet.dll
  craftIPPacket(protocol, SrcIP, DestIP);
  craftL2Packet(IPBuffer);
  sendPacket(L2Buffer);
}

sendPacket(buffer)
{
  get adapter list;
  open adapter;
  allocate packet;
  initiate packet;
  set number of packets to send;
  send packets;
  free packet;
  close adapter;
}
```

Fig. 7. Part of the pseudocode for WinPcap.



```
Parallel Network Stack Testing...

Source IP: 147.188.183.62
Dest IP: 147.188.128.3

*Crafting IP packet...          Successful!
*Crafting L2 packet...          Successful!
*Sending packets...             Successful!

Press Enter to return to Quincy..._
```

Fig. 8. Parallel network stack testing screenshot.

From the architecture above we know that WinPcap kernel mode Packet Capture Driver works partly below NDIS level packet interception. WinPcap architecture has the capability of sending packets regardless of the existence of personal firewall. It also has the capability of receiving packets without the notice from the firewall. This makes it a perfect basis for developing applications that can bypass personal firewalls. It is just perfect for developing a remote network sniffer Trojan which remotely captures information on a local network and bypasses the personal firewall to send information back to the attacker.

6. Conclusions

In this paper, we discussed the insecurity of the personal firewall on Microsoft Windows platform. Personal firewalls are intended to protect personal computers by enforcing access control between the Internet and the computers. We demonstrated that a personal firewall has to reside at the lowest level possible in the Windows Network Architecture to intercept packets. The tight integration with system also makes it possible to provide application control. A personal firewall inspects network activities at multi-levels, blocking unwanted traffic to stop communication of Trojans or spyware, restrict access to system vulnerability, and provide online stealth. Personal firewall can also provide additional features such as cookie control, content filtering, or even intrusion detection. It shows to be a good add-on for network security of personal computers.

However, personal firewall alone is not the cure for network security. As examined in both theory and experimental implementations, personal firewalls are vulnerable to many attacks. It seems to be always true that defense is always after attacks in the network security field. It is still in question as to how a personal firewall can defend against these attacks.

For better protection, we reckon that a personal firewall should be used in conjunction with good antivirus software. Meanwhile, future research is needed to provide personal firewall with more solid functions that are harder to bypass from hardware point of view.

7. References

[1] http://www.net-spec.com/whitepapers/MigrationtoDPI.pdf

[2] S. Hendison, *Do you need a Firewall?,* Special Report, Scotts Computer Help Articles, 2005; http://www.pdxtc.com/200110-firewall.htm

[3] http://download.smoothwall.net/pdf/white.personal-firewalls.pdf

[4] E.D. Zwicky, S. Cooper and D.B. Chapman, *Building Internet Firewalls*, 2nd Edition, O'Reilly, London, 2000.

[5] C. Ries, *Defeating Windows Personal Firewalls: Filtering Methodologies, Attacks, and Defenses,* White Paper, VigilantMinds Inc., 2005; http://www.vigilantminds.com/files/defeating_windows_personal_firewalls.pdf

[6] E. Chien, *Techniques of Adware and Spyware,* White Paper, Symantec Corp., Santa Monica, CA, 2005; www.symantec.com/avcenter/reference/techniques.of.adware.and.spyware.pdf

[7] http://www.ndis.com/papers/winpktfilter.htm

[8] W. Buchanan and D. Llamas, "Covert Channel Analysis and Detection using Reverse Proxy Servers," *Proc. of the 3rd International Conference on Electronic Warfare and Security* (ECIW 2004), London, 2004.

[9] T. Plooy, "Packet Filtering with iphlpapi.dll," *Windows Developer's Journal*, vol. 11, no. 10, 2000.

[10] S. S. M. Chow, L. C. K. Hui, S. M. Yiu, K. P. Chow and R. W. C. Lui, "A generic anti-spyware solution by access control list at kernel level," *Journal of Systems and Software*, vol. 75, no. 2, 2004, pp. 227-234.

[11] https://ccv4.cryptocard.com/site/CryptoCard_2.0_41/pdf/MAS_Microsoft_RRAS_Implementation_Guide.pdf

[12] http://www.microsoft.com/windows2000/techinfo/reskit/en/Intwork/internetworking.htm

[13] http://msdn.microsoft.com/library/default.asp

[14] G.A. Stewart III, R. Dodge and D. Ragsdale, "Embedded Firewall Defense," *United States Military Academy*, 2004.

[15] http://www.securiteam.com/securitynews/5HP0C0K35S.html

[16] http://www.megasecurity.org/Firewalls/Firecracker.html

[17] C. Maple, H. Jacobs and M. Reeve, "Choosing the Right Wireless LAN Security Protocol for the Home and Business User," *Proc. of the 1st International Conference on Availability, Reliability and Security* (ARES 2006), Vienna, Austria, 2006.

[18] http://www.megasecurity.org/Trojaninfo/ACK_Tunneling_Trojans.htm

[19] W.R. Cheswick and S.M. Bellovin, *Firewalls and Internet Security*, Addison-Wesley, 1994.

[20] http://tooleaky.zensoft.com

[21] http://keir.net/firehole.html

[22] http://network-spy.com/netspy.php

[23] http://www.ethereal.com

[24] http://netgroup-serv.polito.it/WinPcap

[25] F. Risso and L. Degioanni, "An Architecture for High Performance Network Analysis," *Proc. of the 6th IEEE Symposium on Computers and Communications* (ISCC 2001), Hammamet, Tunisia, 2001, pp. 686-693.