Global Information Assurance Certification Paper

Global Information Assurance Certification Paper

Interception and Automating Blocking of Malicious Traffic Based on NDIS Intermediate Driver

GIAC (GREM) Gold Certification

Author: Lee Ling Chuan, lc.lee@cybersecurity.my
Advisor: Antonios Atlasis

Accepted: 19th October 2010

Abstract

With the evolution of malware technology, modern malware often hide its malicious behavior in various methods. One of the popular manners is to conceal the network communication. This concealment technique poses obstacles to security mechanisms which are used to detect malicious behaviors. In this paper, we give an overview of the automated blocking malicious code technique, a new approach to computer security via malicious software analysis and automatic blocking software. In particular, this technique focuses on building a unified executable program analysis platform and using it to provide novel solutions to a broad spectrum of different security problems. We propose a technique for the Network Driver Interface Specification (NDIS) integrate together with a unified malicious software analysis platform. The NDIS model supports hybrid network transport NDIS drivers, called NDIS intermediate drivers. This driver lies between transport driver and NDIS driver. The advantage of using NDIS intermediate drivers is that it can see the entire network traffic taking place on a system as the drivers lie between protocol drivers and network drivers. By intercepting security-related properties from network traffic directly, this technique enables a principled, root cause based approach to computer security, offering novel and effective solutions.

1. Introduction

Over the past years, the number of malicious programs developed for illegal purpose has grown rapidly. The Monthly Malware Statistics, January 2011 (Zakorzhevsky, 2011) by Kaspersky Lab announced that there are over ten million viruses in circulation, most developed in January 2011. In the past, malicious code has been categorized neatly into different categories such as viruses, loggers or trojan horses based upon functionality and attack vector. Today, the methods used by malware coders to achieve their objectives have substantially evolved. Indeed, criminals are making extensive use of malware to control computers and steal personal, confidential, or otherwise proprietary information for either profit or for fun (Davis, Bodmer & LeMasters, 2009). There are also some malicious codes that conceal the communication pathway and avoid the detection from security protection mechanisms such as firewalls, sniffers, antivirus programs, IDS systems etc (Hoglund & Butler, 2005).

In general, security mechanisms on Windows such as the above rely on the native TCP/IP stack for network traffic related functions. However, Microsoft has imposed restrictions on raw socket (Microsoft MSDN Library, 21 May 2011. TCP/IP Raw Sockets), such as:

1. TCP data cannot be sent over a raw socket.
2. UDP datagram cannot spoof their source address over a raw socket
3. Raw sockets cannot make calls to the bind () function.

A raw socket is a socket that allows direct access to the headers of a network frame. Naturally, the freedom to spoof frame information was abused by malware developers. Hence, these restrictions have been imposed by Microsoft on Windows XP SP2 and later version. The constraints placed on raw sockets are built into tcpip.sys and tcpip6.sys drivers. The only solution that can circumvent the restrictions that places on raw sockets is to roll a dedicated transport layer. This approach gives the authority to control over the created packets. Thus, to see the entire network traffic taking place on a system, rolling Network Driver Interface Specification (NDIS) protocol driver (Dhawan,

1995) is the only solution on Windows, especially in Windows XP SP2, Microsoft Vista and Windows 7. The NDIS is a library of functions that forms the upper sub layer of the OSI data link layer (Stevens, 1994) and acts as an interface between level 3 network protocol drivers and the hardware level MAC drivers (Stevens, 1996).

Table 1 shows the comparison between NDIS interface with Winsock Kernel (WSK) and Winsock Interface (Microsoft MSDN Blog, 1 June 2011. Introduction to Winsock Kernel). Winsock is a technical specification that defines a standard interface between a Windows TCP/IP client application and TCP/IP protocol stack (Wright & Stevens, 1995). The WSK operates in kernel mode and provides Transport Driver Interface (TDI) client developers with a sockets-like programming model similar to those supported in user-mode. The NDIS is a Windows specification as it is a kernel-mode network driver that defines the routines network drivers should implement (Oney, 2003).

Interface	Benefits	Drawbacks
Winsock	Easy to use, well documented	Easier to track down
WSK	Uses the existing TCP/IP stack Not as easy to track down	More demanding and less forgiving than Winsock Must account for protocol-dependent behaviour
NDIS	Offers the most control Can spoof packets Can bypass local firewall	Effort required to implement a new TCP/IP stack Switches may limit one MAC address per port Can be conspicuous in packet capture

Table 1: Comparison with NDIS, WSK and Winsock (Blunden, 2009)

Our approach is to integrate NDIS with a unified malicious software analysis platform. The advantage of our technique is that we can monitor the entire network traffic on the system, including network traffic passing through in kernel level. The work we describe reflects the following contribution:

1. A description of NDIS Intermediate Driver and related programs and functions.
2. Motivated by our description, implementation of an NDIS Intermediate Driver that can be used to supplement malware analysis and detection techniques.
3. Implementation and use of NDIS Intermediate Driver to test our technique against sets of malware binary. Benefits observed by evaluating the results from testing

include the following:

- NDIS has ability to capture the entire network traffic taking place on a system at kernel level.
- NDIS Intermediate Driver can assist malware researcher in malware analysis activity.

In this paper we address the architecture of the system to intercept and automate blocking of malicious network traffic by using NDIS Intermediate driver method (Microsoft MSDN Library, 20 May 2011. NDIS Intermediate Driver). The reason that we choose NDIS intermediate driver is that it can capture all the packets passing through the system, including network packets of user-level, as well as kernel-level which can bypass local firewall. Our approach is to capture the entire network traffic packet and control it in a real time manner. An output of the network traffic log file will be exported into a user readable file where any fields in any headers protocols, can be displayed according to the user's preference.

2. NDIS PACKET Description

Host based network security software on Windows have made significant advances in both technology and scope of deployment within the past few years. Despite these advances, many challenges remain. One of the biggest challenges is that they are relying on the support of the underlying Operating System for data gathering and monitoring. However, the evolution of malware programs has proved that they are capable to exploit this weakness. The capability to bypass virtually all commodity, host-based firewall and intrusion detection system software in the market today has force security researchers to seek new methods to detect and block the malicious codes; this is the reason why security organizations decided to use NDIS Intermediate Driver as the solution (Sparks, 2009).

Kaspersky Anti-Virus is one of the security products implemented the technology of NDIS Intermediate Driver, although it allows its users to disable the Kaspersky Anti-Virus NDIS Filter functions. However, disabling the NDIS Filtering function will cause

the Anti-Hacker/Firewall function not to perform packet filtering and malicious network traffic will not be intercepted.

According to "Design and implementation of a personal firewall based on NDIS Intermediate Drivers" (Chaokai, 2007), the current firewall technology on defending against external network attacks and threats which deals with the packets under user mode has a lot of limitation. The paper has proposed that the protection can be done better by using NDIS intermediate driver.

The main function of NDIS packets, represented by NDIS_PACKET structure is to ensure all network data can be sent to or from the network in a system. Prior sending data on the network, a protocol driver allocates NDIS packets; filled with data and passed to the next lower NDIS driver. On the contrary, some lowest level NIC drivers will allocate packets to hold received data and pass it to higher layer drivers. There are functions provided by NDIS for allocating and manipulating the substructures that form a packet. A good description of the NDIS_PACKET structure can be found on PCAUSA website (PCAUSA, 2011. NDIS_PACKET Discussion Part 1).

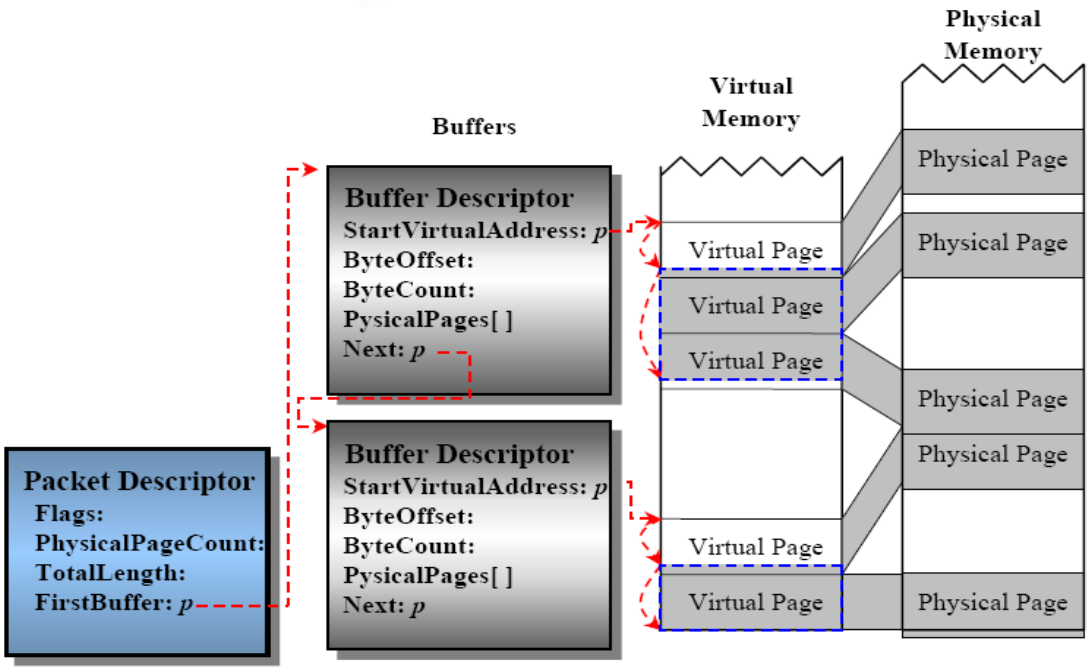

Figure 1: Multi-Buffer NDIS_PACKET Illustration (Sarin, 2008)

In general, each NDIS Packet is a Packet Descriptor, and it has a series of Buffer Descriptors and NDIS_BUFFER (PCAUSA, 2011. NDIS_PACKET Discussion Part2).

The number of chained NDIS_BUFFER is depending on the size of the NDIS packet. If the packet size is small, there will be only one chained NDIS_BUFFER and this buffer is enough to describe the range of virtual memory that contains the complete packet data. Figure 1 shows the NDIS_PACKET, which has two chained NDIS_BUFFER. The first NDIS_BUFFER describes a range of virtual memory that contains the Ethernet header, while the second NDIS_BUFFER describes the range of virtual memory that contains the Ethernet payload. The content of a typical packet descriptor are as the following list:

1. Private areas for the miniport NIC driver and a protocol driver.
2. Flags associated with the packet, defined by a cooperating miniport(s) and protocol driver(s).
3. Number of physical pages that contain the packet.
4. Total length of the packet..
5. Pointer to the first buffer descriptor that maps the first buffer in the packet.

The following list shows the contents of a typical buffer descriptor:

1. Starting virtual address of each buffer.
2. Buffer's byte offset into the page pointed to by the virtual address.
3. Total number of bytes in the buffer.
4. Pointer to the next buffer descriptor, if any.
5. Virtual range, possibly spanning more than one page that makes up the buffer described by the buffer descriptor. These virtual pages map to physical memory.

The virtual range allocates the buffer described by the buffer descriptor. These virtual pages will map to physical memory.

2.1 Principle of Sending Packet Data

The idea of the NDIS Intermediate Driver to implement models of capture and detection of network packet system is based on the ability and capability to intercept the entire network packet as it is the only route for network packet passes through. The workflow of this system consists of the arrival of network packet from Transport Driver

Interface TDI level to NDIS Intermediate Driver through registered MpSend function (Divine, 2003), its initial classification, exportation of unidentified network samples to a robot machine farm, and re-analysis using the results of malicious-code samples.

The series of NDIS send request functions, NdisSend[1], NdisSendPacket[1] and NdisCoSendPacket[1] forward a send request to the underlying driver. Sender using a different function will send the packet contents in different ways, but we can analyse it in the same principle. Thus, the system uses a modified NdisSend function to capture data packets. The NdisSend function prototype is shown below:

```
void NdisSend (
        OUT PNDIS_STATUS  Status,
        IN NDIS_HANDLE        NdisBindingHandle,
        IN PNDIS_PACKET     Packet
);
```

The second NdisSend parameter, NdisBindingHandle[1] specifies the handles which returned by NdisOpenAdapter[1]; these handles will identify the targeted NIC or the virtual adapter of the next lower driver to which the caller is bound. The control code points to an internal structure and the internal structure stored enough information from the upper to lower driver which making the NDIS function to use the correlation function for the next layer drive.

Figure 2 below displays an overview of the entire send request process in a NDIS network driver stack. Starting with the top of the figure, which is the protocol driver, the following are the two intermediate drivers and the bottom is miniport. Before NdisSend is called, a protocol driver can set the flags in the private header which is only reserved for use by the NDIS of the allocated packet descriptor. These flags specify caller-determined information about the requested send operation which is not contained in the packet data. The underlying NIC driver's MiniportSend[2] function is given the send flags as input parameters. When an underlying driver that is serialized has insufficient resources to transmit a valid send packet, it has two alternatives option: First, its MiniportSend

[1] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Library Function Refences.

[2] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Driver Upper-Edge Functions.

function can insert the packet into an internal queue and return NDIS_STATUS_PENDING. The driver holds the packet queued until resources become available and sends the packet when they are ready. Second, MiniportSend function can simply return control with NDIS_STATUS_RESOURCES. The NDIS library holds such a returned packet in an internal queue for resubmission to the serialized miniport driver. The miniport driver can indicate its readiness to accept send packets later by calling NdisMSendComplete[3] or NdisMSendResourcesAvailable[3]. An NDIS Intermediate driver must repackage incoming sends from higher level protocols in fresh packet descriptors before passing such a send request to the underlying miniport driver with NdisSend.

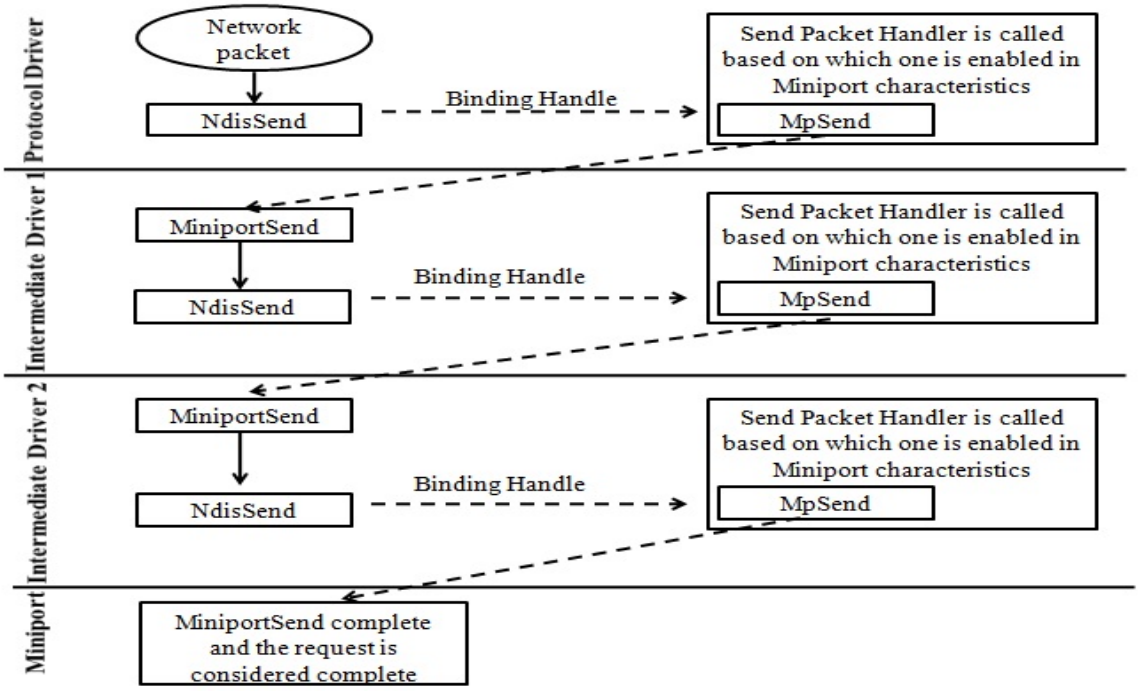

Figure 2: NdisSend forwards a send request to the underlying driver (Microsoft Corporation, 1999. Windows Driver Kit version 7600.16385.1.)

2.2 Principle of Receiving Packet Data

PtReceive and PtReceivePacket (Antognini & Divine, 2003) are the two mutually exclusive functions in receiving packets from NDIS. Although different function used by receiver will get the packet contents in different ways, the overall analysis work can be done in the same principle. Thus, the system uses a modified RtReceive function to

[3] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Library Function Reference.

capture data packets. RtReceive function prototype is shown below:

```
PtReceive (
    IN NDIS_HANDLE          ProtocolBindingContext,
    IN NDIS_HANDLE          MacReceiveContext,
    IN PVOID                HeaderBuffer,
    IN UINT                 HeaderBufferSize,
    IN PVOID                LookAheadBuffer,
    IN UNIT                 LookAheadBufferSize,
    IN UINT                 PacketSize
    );
```

The parameter of LookAheadBufferSize specifies the size in bytes of the LookAheadBuffer. If the size of the network packet is smaller or equal to the parameter, it indicates that the LookAheadBuffer contains all the information of the entire network. The program code below shows the underlying driver that has been programmed to obtain network packet.

```
Packet = NdisGetReceivedPacket[4] (pAdapt->BindingHandle, MacReceiveContext);
```

According to the above code, if the parameter return value is null, it indicates that the bottom level cannot provide a complete network packet. On the contrary, if the return value is not null, it means that bottom level has provided a complete network packet.

The workflow of the system consists of the receiver of the network packet, its initial classification, the exportation of unidentified samples to a robot machine farm and the re-analysis if the malicious code pattern exists. This entire process occurs in a pipelined fashion, as shown in Figure 3 below:

[4] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Library Function Reference.

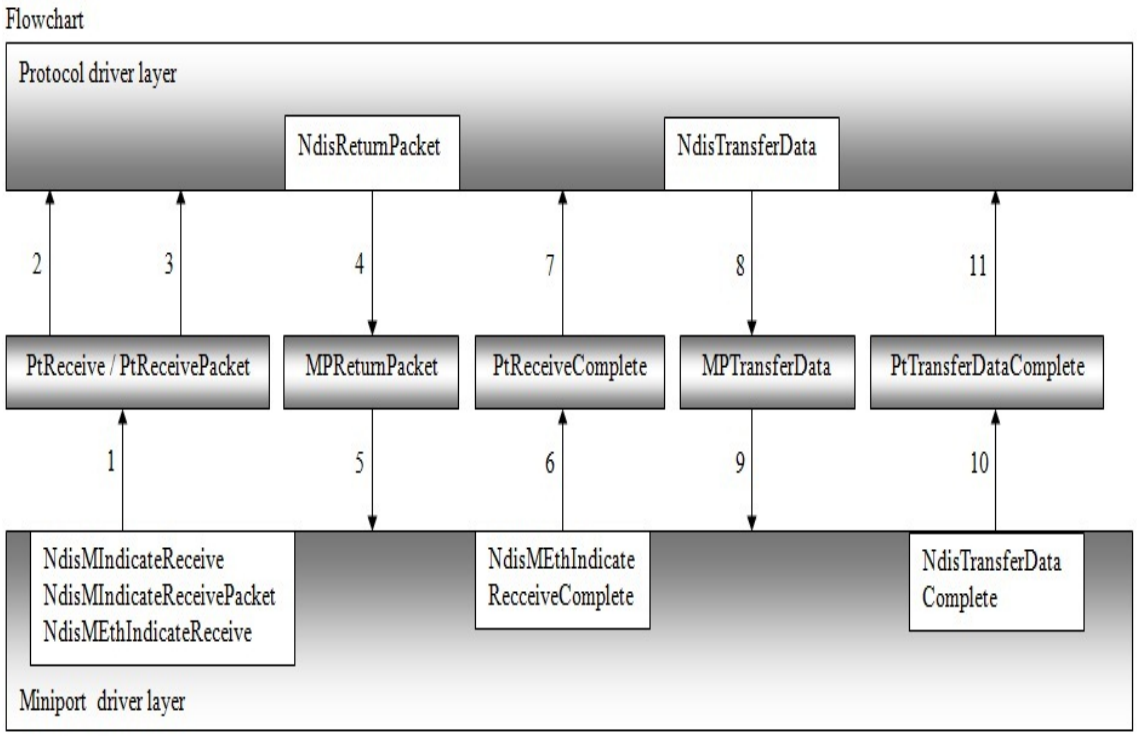

Figure 3: Flowchart of NDIS IM Driver Receive Network Packets (DriveDevelop Forum, 2003. Passthru Send/Receive Operation Flowchart)

Starting with Step 1, miniport driver layer calls NdisMIndicateReceive[5] or NdisMEthIndicateReceive[5] to inform Protocol Driver Layer that Miniport driver layer received packets data. The PtReceive or PtReceivePacket function receives a complete packet through NdisGetReceivedPacket function; in such a case, NdisMIndicateReceivePacket[5] will notify NDIS to call the corresponding upper PtReceive routines. However, if an incomplete packet received by PtReceive or PtReceivePacket function, NdisMEthIndicateReceive[5] will notify NDIS. After the upper protocol driver received a complete packet of data, NdisReturnPacket[5] will be used to notify NDIS. MPReturnPacket (Antognini & Divine, 2003) and NdisReturnPacket[5] functions will be called to release the temporary packet and resource at miniport and protocol layer. In the case of PtReceive/PtReceivePacket receive incomplete packet via NdisGetReceivedPacket, if miniport layer receive the complete packet, miniport layer will use NdisMEthIndicateReceiveComplete[5] to notify NDIS, and PtReceiveComplete (Antognini & Divine, 2003) will be used to inform Protocol layer that complete packet

[5] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Library Function Reference.

have been received. NdisTransferData[6] function will forward a request to copy data received at the miniport driver layer. If NDIS_STATUS_SUCCESS value returns, it indicates that the requested data has been transferred. On the contrary, if NDIS_STATUS_PENDING value returns, the request is being handled asynchronously. After the miniport driver layer receives the complete packet, NdisTransferDataComplete[6] function will be called. The overall process will complete if PtTransferDataComplete (Antognini & Divine, 2003) is send from Miniport Driver Layer to Protocol Driver Layer.

3. Interception and Blocking Architecture

The NDIS model supports hybrid network transport NDIS drivers, called NDIS intermediate driver. The driver lies between transport drivers and NDIS drivers. To an NDIS driver, an NDIS intermediate driver looks like a transport driver; to a transport driver, an NDIS intermediate driver looks like an NDIS driver.

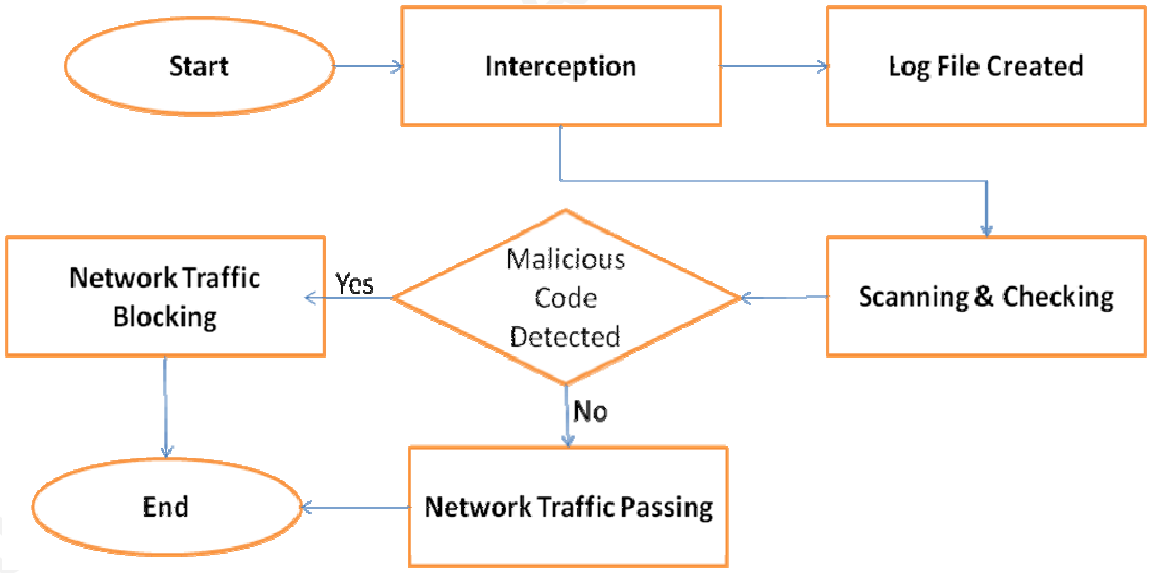

Figure 4: Interception and Blocking Architecture

In this section, we give an overview of the interception and blocking architecture using NDIS intermediate driver. Figure 4 shows the flow chart for the interception and blocking that have been developed. As shown in this figure, all incoming and outgoing

[6] Microsoft MSDN Library (5 October 2010), especially chapter NDIS Library Function Reference.

network traffic of a system will be intercepted by NDIS Intermediate Driver. The intercepted network packet will be passed to scanning and checking module and it will be compared with a set of pre-defined malicious code or malware signature. If any of the network traffic match with a signature, the detected network traffic will be blocked, while the less of the network traffic will continue passing through the system.

As shown in Figure 4, all the network traffic is first intercepted by the interception mechanism. In the process of interception, the entire network packets will dump into a log file in hexadecimal format. The purpose of creating a log file is to see the entire activities that are taking place. The scanning and checking mechanism will execute pattern matching function. This function will compare the intercepted network traffic with pre-defined unique signature strings where each unique string represents malicious portion code signatures.

Knuth-Morris-Pratt (KMP) algorithm was used to observe when a mismatch occurs. The KMP algorithm looks for the malicious code signature pattern in a left-to-right order. It looks like the brute force algorithm but it shifts the pattern more intelligently than the brute force algorithm. Consider an attempt at a left position l, that is when the window is positioned on the text factor y[l ..l+m-1]. Assume that the first mismatch occurs between x[i] and y[i+j] with $0 < i < m$. Then, x[0 .. i-1] = y[l .. i+l-1] =u and $a = x[i] \neq y[i+j]=b$ (Charras & Lecroq, 1997). When shifting, it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of the text. Moreover, if we want to avoid another immediate mismatch, the character following the prefix v in the pattern must be different from a. The longest such prefix v is called the tagged border of u. This introduces the notation: let TT[i] be the length of the longest border of x[0 ..i-1] followed by a character c different from x[i] and -1 if no such tagged border exits, for $0 < i \leq m$. Then, after a shift, the comparisons can resume between character x[TT[i]] and y[i+j] without missing any occurrence of x in y. The value of TT[0] is set to -1.

Appendix A illustrates the fragment code of this function. If the pattern matching occurs, NDIS intermediate driver will immediately drop the interception network traffic. STATUS_DROP function can be used to drop the network packet as shown below:

```
DbgPrint("Signature Match Success");
++j;
if(T[j]=='\0')
{
        DbgPrint("myndis blocking – malicious traffic detected");
        return STATUS_DROP;
```

If none of the pattern is matched, network traffic will allow pass through of the system.

To provide the NDIS Intermediate Driver with logging abilities whose reports will be able to show details such as the source IP, source port, destination IP, destination port, packet payload and the reason for the denial of a request if it exits, log files that records network activities are essential. All the generated reports are in the form of XML files linked to a style sheet that makes them easily viewed. Example of the log file can refer to Figure 5. Our technique is based on method of creating an XML file in the driver layer using ZwCreateFile function (Oney, 2003). However, both ZwReadFile and ZwWriteFile function function (Oney, 2003) can be used to read and write files. Below shows partially source code of log file created function.

```
RtlInitUnicodeString (&usname,L"\\SystemRoot\\System32\\LogFiles\\passthru.log");
InitializeObjectAttributes(&oa, &usname, OBJ_CASE_INSENSITIVE |
                        OBJ_KERNEL_HANDLE, NULL, NULL);
Status = ZwCreateFile(&hfile, GENERIC_WRITE, &oa, &iostatus, NULL,
                        FILE_ATTRIBUTE_NORMAL,
                        FILE_SHARE_READ, FILE_OVERWRITE_IF,
                        FILE_SYNCHRONOUS_IO_NONALERT, NULL, 0);
```

In order to dump all network traffic and write into the log file, a program code as the one shown below can be written:

```
ZwWriteFile(hfile, NULL, NULL, NULL, &iostatus, PrintContent, count, NULL, NULL);
```

Some device drivers and executive components create their own thread dedicated

to process work at passive level; however, most of them use system worker thread instead, which avoid the unnecessary scheduling and memory overhead associated with having additional thread in the system. Our approach calls the executive functions, ExQueueWorkItem function (Oney, 2003) to request a system worker thread's services. This function place a work item on a queue dispatcher object where the threads look for work. Work items include a pointer to a routine and a parameter that the thread passes to the routine when it processes the work item. The routine is implemented by the device driver or executive component that requires passive-level execution.

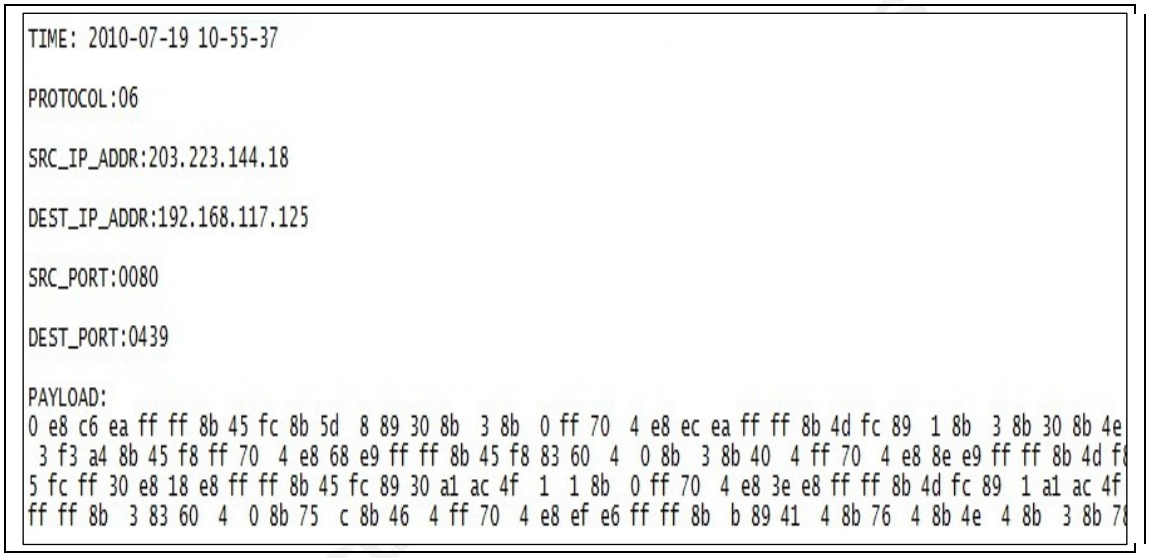

Figure 5: Hexadecimal Log File

4. Case Study

We implemented a system for malicious network traffic interception and analysis with the above component and technique. The experimental environment was the following: two testing machines were selected, including a Web Server installed with Apache and PHP and a machine used as a client with installed NDIS Intermediate Driver (Microsoft MSDN Library, 27 May 2011. Passthru Ndis Intermediate Sample Driver).

In this case study, an Adobe Flash Player exploit (Down, 2008) was used to exploit the testing machine. All incoming and outgoing network traffic was intercepted by the NDIS intermediate driver. The signatures of the Adobe Flash Player exploit was collected and put under the list of signature matching. Figure 6 shows the unique string

for Adobe Flash Player exploit payload that has been selected. The selected signature is "$1/1@1K1Q1t1z1".

As discussed in the previous section, all intercepted network traffic was dumped into a hexadecimal log file. Figure 5 shows the example of hexadecimal log file that match with the malicious payload signature.

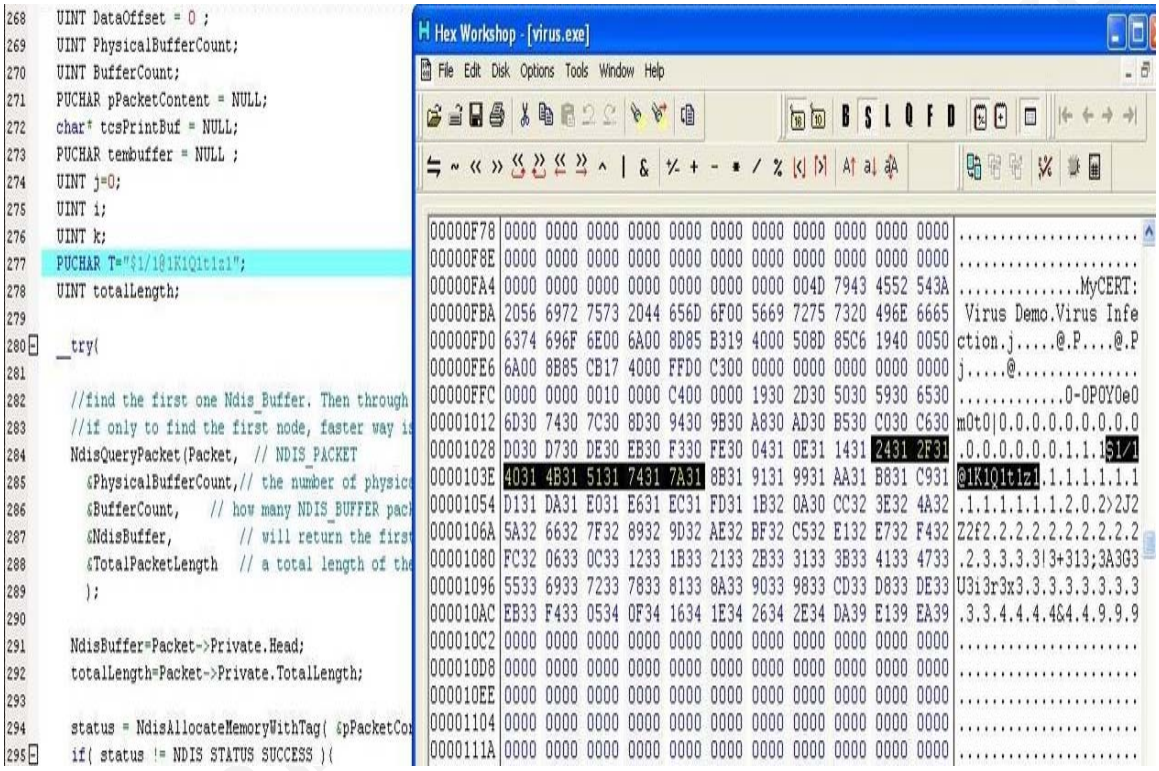

Figure 6: Unique String of Malicious Payload

After executing the NDIS Intermediate driver, all network packets passing through the testing machine was intercepted. Pattern matching function will executed to scan the intercepted network traffic if the Adobe Flash Player exploit payload matches. The incoming network traffic created by the Adobe Flash Player exploit was detected and the NDIS intermediate driver dropped the malicious network packet. Figure 7 shows the dropping malicious network packet when the signature was found, indicated by the "Signature Match Success" message.

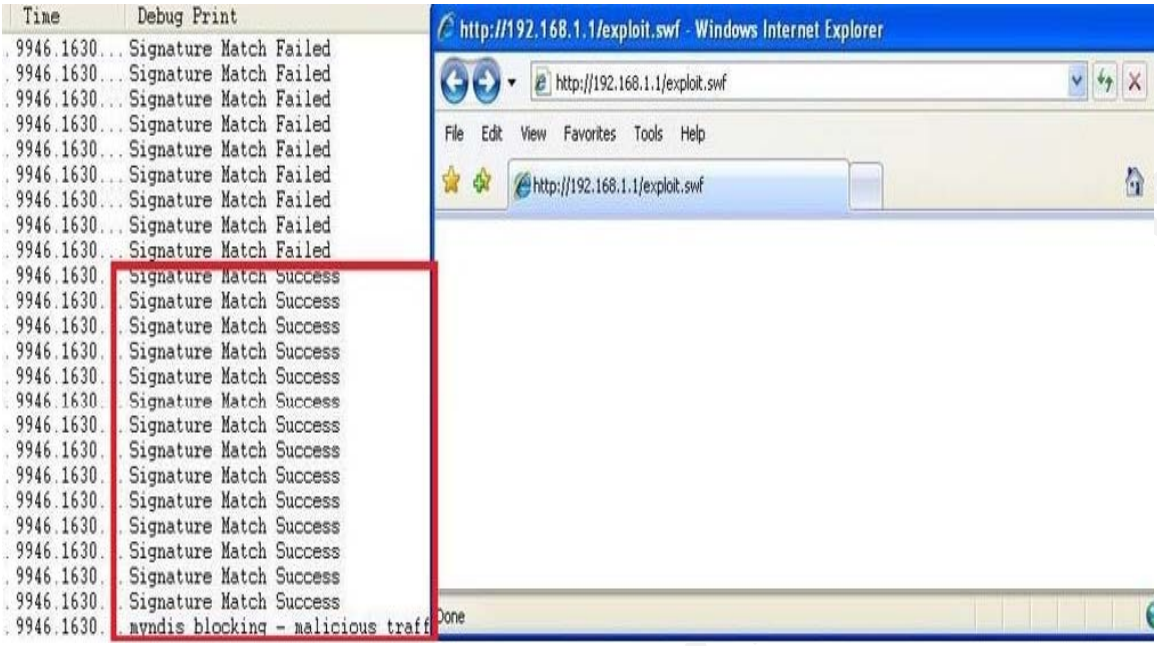

Figure 7: Blocking Malicious Network Traffic Packets

5. Conclusions

Malware scanning engine is an essential malware protection tool for today's computer system. Many users use it as a first line of defense against various kinds of network attacks and system threats. With the evolution of malware technology, attacking victim's computers is evolved into kernel mode. However, most personal scanning engine only deals with the network packet under user mode, and cause a lot of limitation. In order to provide better protection for the user, security prevention mechanisms need to be done in kernel mode. This paper takes an in-depth look at the pervasive dynamic security with interaction of IDS and Firewall concept using the NDIS Intermediate Driver to achieve malicious network traffic capture and filtering. The method presented in this paper is for the handling and analyzing the entire network traffic taking place on a system. Our future work will focus on the study of architecture for string matching which aims to optimize for speed and scalability.

Author Name, email@address

6. References

Antognini, J. , & Divine, T. F. (2003). Extending The Microsoft PassThru NDIS Intermediate Driver Part2: Two IP Address Blocking NDIS IM Drivers. Retrieved 24 May 2011, from wd-3 Web site: http://www.wd-3.com/archive/ExtendingPassthru2.htm

Blunden, P. B. (2009). The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System, Wordware Publishing Inc.

Chaokai, H. (2007). Design and Implementation of a Personal Firewall Based on NDIS Intermediate Drivers. Eighth ACIS International Conference.

Charras, C., & Lecroq, T. (1997). Handbook of Exact String Matching Algorithms. Computer Science Department and LITIS EA 4108, Faculty of Science University of Rouen.

Davis, M. A., Bodmer, S., & LeMasters, A. (2009). Hacking Exposed Malware & Rootkits: Malware & Rootkits Security Secrets & Solutions. McGraw Hill Professional.

Dhawan, S. (1995). Networking Device Drivers. John Wiley & Sons Inc.

Divine, T. F. (2003). Extending The Microsoft PassThru NDIS Intermediate Driver Part1: Adding a DeviceIoControl Interface. Retrieved 24 May 2011, from wd-3 Web site: http://www.wd-3.com/archive/ExtendingPassthru.htm

Dowd, M. (2008). Application-Specific Attacks: Leveraging the ActionScript Virtual Machine. IBM Internet Security Systems.

DriveDevelop Forum (2003). Passthru Send/Receive Operation Flowchart. Retrieved from DriverDevelop Forum Web site: http://bbs.driverdevelop.com/read.php?tid-40727.html

Hoglund, G., & Butler, J. (2005). Rootkits: Subverting the Windows Kernel. Addison-Wesley Professional.

Microsoft Corporation, 1999. Windows Driver Kit version 7600.16385.1. PASSTHRU.SYS – Sample NDIS Intermediate Driver.

Microsoft MSDN Blog. Introduction to Winsock Kernel (WSK). Retrieved 1 June 2011, from Microsoft MSDN Blog Web site:

http://blogs.msdn.com/b/wndp/archive/2006/02/24/538746.aspx

Microsoft MSDN Library. NDIS Intermediate Driver. Retrieved 20 May 2011, from Microsoft MSDN Library Web site: http://msdn.microsoft.com/en-us/library/ff557012(v=vs.85).aspx

Microsoft MSDN Library. Passthru Ndis Intermediate Sample Driver. Retrieve 27 May 2011 from Microsoft MSDN Library Web site: http://msdn.microsoft.com/en-us/library/ff569982(v=vs.85).aspx

Microsoft MSDN Library. TCP/IP Raw Sockets. Retrieved 21 May 2011, from Microsoft MSDN Library Web site: http://msdn.microsoft.com/en-us/library/ms740548(v=vs.85).aspx

Oney, W. (2003). Programming the Microsoft Windows Driver Model. Microsoft Press.

PCAUSA (2011). NDIS_PACKET Discussion Part 1-NDIS Packet Data. Retrieved 24 May 2011, from PCAUSA Web site: http://www.ndis.com/ndis-ndis5/ndispacket/ndispacket1.htm.

PCAUSA (2011). NDIS_PACKET Discussion Part 2-NDIS_PACKET Reserved Areas. Retrieved 24 May 2011, from PCAUSA Web site: http://www.ndis.com/ndis-ndis5/ndispacket/ndispacket2.htm

Sarin, A. (2008). NDIS – Part 1. Retrieved 27 May 2011, from Microsoft MSDN Web site: http://blogs.msdn.com/b/ntdebugging/archive/2008/09/19/ndis-part-1.aspx

Sparks, S., Embleton, S., & Zhou, C. C. (2009). A chipset Level Network Backdoor: Bypassing Host-Based Firewall&IDS. School of Electrical Engineering and Computer Science University of Central Florida USA.

Stevens, W. R. (1994). TCP/IP Illustrated, Vol.1: The Protocol. Addison-Wesley Professional Computing Series.

Stevens, W. R. (1996). TCP/IP Illustrated, Vol. 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols. Addison-Wesley Professional.

Wright, G. R., & Stevens, W. R. (1995). TCP/IP Illustrated, Vol.2: The Implementation. Addison-Wesley Professional.

Zakorzhevsky, V. (2011). Monthly Malware Statistic January 2011. Retrieved 21 May 2011, from securelist Web site: http://www.securelist.com/en/analysis/204792159/Monthly_Malware_Statistics_J

Author Name, email@address

anuary_2011?print_mode=1

Author Name, email@address

7. Appendix A: Fragment Code Applied Knuth-Morris-Pratt Algorithm

```
TT[0]=-1;
TT[1]=0;
p=2;
k=0;
while(T[p]!='\0'){
        if(T[p-1]==T[k]){
                TT[p]=k+1;
                k++;
                p++;
        }
        else if (k>0){
                k=TT[k];
        }
        else {
                TT[p]=0;
                p++;
                k=0;
        }
}
if(pPacketContent[34]==0&&pPacketContent[35]==80){
  DbgPrint("Received Traffic Http Port 80");
        for(i=54;i<=DataOffset;i++){
          while(pPacketContent[i+j]!='\0'&&T[j]!='\0'){
            if(pPacketContent[i+j]==T[j]){
```

```
            DbgPrint("Signature Match Success");
            ++j;
            if(T[j]=='\0'){
              DbgPrint("myndis blocking – malicious traffic detected");
              return STATUS_DROP;
               }
              }
            else{
                  i+=j-TT[j];
                  if (j>0)
                  j+=TT[j];
              }
          }
      }
}
```

Author Name, email@address

Last Updated: April 8th, 2014

Upcoming Training

SANS Security West 2014	**San Diego, CA**	**May 08, 2014 - May 17, 2014**	**Live Event**
Mentor Session - FOR 610	**Columbia, MD**	**May 21, 2014 - Jul 23, 2014**	**Mentor**
Digital Forensics & Incident Response Summit	**Austin, TX**	**Jun 03, 2014 - Jun 10, 2014**	**Live Event**
Community SANS Ottawa	**Ottawa, ON**	**Jun 16, 2014 - Jun 21, 2014**	**Community SANS**
SANSFIRE 2014	**Baltimore, MD**	**Jun 21, 2014 - Jun 30, 2014**	**Live Event**
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	**FOR610 - 201407,**	**Jul 14, 2014 - Aug 20, 2014**	**vLive**
SANS Virginia Beach 2014	**Virginia Beach, VA**	**Aug 18, 2014 - Aug 29, 2014**	**Live Event**
SANS Baltimore 2014	**Baltimore, MD**	**Sep 22, 2014 - Sep 27, 2014**	**Live Event**
SANS DFIR Prague 2014	**Prague, Czech Republic**	**Sep 29, 2014 - Oct 11, 2014**	**Live Event**
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	**FOR610 - 201410,**	**Oct 13, 2014 - Nov 19, 2014**	**vLive**
Community SANS Paris @ HSC - FOR610 (in French)	**Paris, France**	**Nov 24, 2014 - Nov 28, 2014**	**Community SANS**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**