2006

Microsoft .NET and Security Provided by High - Level Internet Protocols

Tatiana Melnik
University of North Florida

Follow this and additional works at: http://digitalcommons.unf.edu/ojii_volumes

Part of the Physical Sciences and Mathematics Commons

Microsoft .NET and Security Provided by High -Level Internet Protocols

Tatiana Melnik

Faculty Sponsor: Dr. Zornitza Genova Prodanoff,
Assistant Professor of Computer Science

Abstract

This paper describes a class of insecure .NET client applications, which avoid higher layer protocol protection through using a "raw" send and receive API. The .NET Framework rests on many other Microsoft components, including the Windows Driver Model (WDM). This model supports four driver types, two of which were considered in this paper: protocol and miniport drivers. By compiling and executing client applications using the "raw" sockets interface, we demonstrate that insecure clients can be written with minimal programming effort (lines of code).

Introduction

The Microsoft .NET (pronounced dot NET) provides an integrated development environment for construction of Web services using eXtensible Markup Language (XML) [16],[6] and Simple Object Access Protocol (SOAP) [17]. The goal of the .NET platform, and Microsoft's larger vision, is to allow better integration of information technology. In the .NET world, people can access their home appliances or office computers from their cellular telephones or personal digital assistants (PDA's).

.NET Framework

Figure 1 shows the two basic .NET Framework components: the Common Language Runtime (CLR) and the Framework Class Library (FCL) [24]. The CLR environment allows the execution of code written in any of several languages. The CLR also provides numerous services to applications, such as elimination of "DLL Hell," and prevention of malicious code execution.
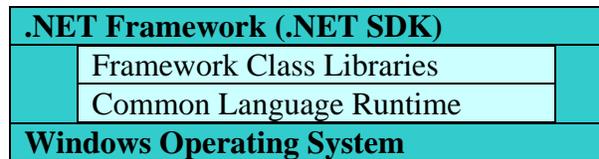

Figure 1. Basic Components of .NET Framework.

The FCL is a collection of classes, interfaces, and value types and is included in the Microsoft .NET Framework SDK [13]. Through the FCL programmers can easily reuse common functions as well as gain access to other .NET high-level services.

Client-Server vs. Distributive Architecture

Using distributive computing, Microsoft wants to create a distributed operating system [25], where all applications are located on the network level and users request an application as a Web service. Currently in distributive computing, the workload is spread over multiple computers where each computer performs a smaller piece of the larger task. The .NET platform, however, has programs located on the network level. Thus "a single application may comprise services accessed from different

computers at various locations around the town" [25].

Presently, much of Internet communication follows the client-server model, which is based on communication between two computers: a client and a server. The client establishes a connection and requests the information or service, and the server provides the requested information or service. Once the server transmits the information or service requested, the communication ends. The .NET platform allows for similar communication. Programmers can write programs that provide services and transfer them to the platform. Users (i.e., other clients) can request the services from the platform.

The .NET approach differs from the traditional client-sever model in that programmers can use modules written by other programmers in their own code. More specifically, when a programmer writes code and sends it to the platform, other programmers can use that code, or that service, without physically seeing the code. They would simply "plug-in" the code from the platform into their own work. Therefore, the communication between the client and server does not end immediately. Rather, the communication is semi-continuous. Since both of the programs will be located on the network level, they will be linked. That is, the application relying on the foreign code will not function properly without access to the foreign code.

As demonstrated in Figure 2, in order to write and run the code, both computers must install .NET Framework, including the Common Language Runtime (CLR). This is necessary because the code is compiled into the Microsoft Intermediate Language (MSIL), which allows for machine specific execution. One of the goals of .NET is to allow

programmers to write code in any language, thus making it platform independent [25]. To achieve such independence, there must be an intermediate step to allow integration into .NET. MSIL is this intermediate step. Once the code is compiled, it can be run on any OS containing the .NET Framework. That is, the application is platform independent. When run, the application employs the CLR.
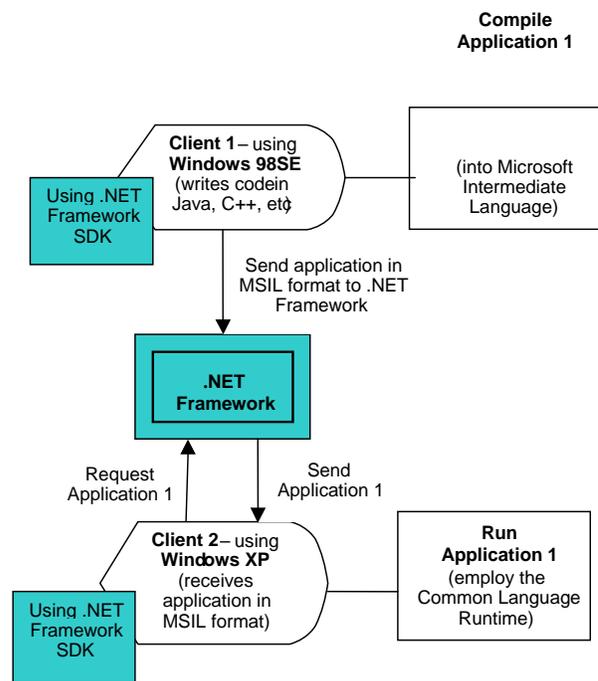

Figure 2. .NET Framework Platform.

The Scope of this Paper

Microsoft is making a great effort to provide security for the users of their products. Although some publications addressed security issues with the .NET Framework [8] [10], few have addressed security issues regarding any specific implementation of communication protocol stacks.

Through writing client applications using "raw" send and "raw" receive packet handling, users avoid the protection provided by these protocols and can gain access to server resources. With this paper we demonstrate that the Microsoft .NET suite allows insecure client applications to surpass the protocol logic of higher layer protocols and use the device driver protocol directly to communicate with the machine hosting the server.

Web Services Security and Communication Protocols

The Internet is a network of networks [11]. That is, wires, routers, and numerous other different devices connect clients and servers, through multiple networks, to each other. When a client attempts to communicate with a server, or network, via the Internet, the two systems, along with the connecting devices, must follow the same protocols. Web service protocols such as Simple Object Access Protocol (SOAP) run on top of Transmission Control Protocol/Internet Protocol (TCP/IP) [21] and Hyper Text Transfer Protocol (HTTP) [1], [7].

The TCP/IP protocol suite is modular. That is, the protocol uses a layered-design approach, where each layer is encapsulated and independent of the layer above and the layer below. The protocol is divided into five layers: application, transport, internet, network and physical medium. As Figure 3 demonstrates, each layer provides services to the layer above, and requests services from the layer below. The transport layer, for example, provides a service to the application layer, and requests a service from the internet layer. When the sender sends data the data traverses through each layer from the top to the bottom.

Sometimes, the data is very large. In order to adequately transfer the data to the receiver, it must be broken down into smaller pieces, or packets. As packets traverse through the layers, each layer adds headers (i.e., overhead) to the original data. The overhead is necessary because it provides essential information regarding the formation and reassembly of packets. When the receiver accepts the packet, the packet traverses from the bottom layer to the top layer, where each layer reads and removes the added overhead.
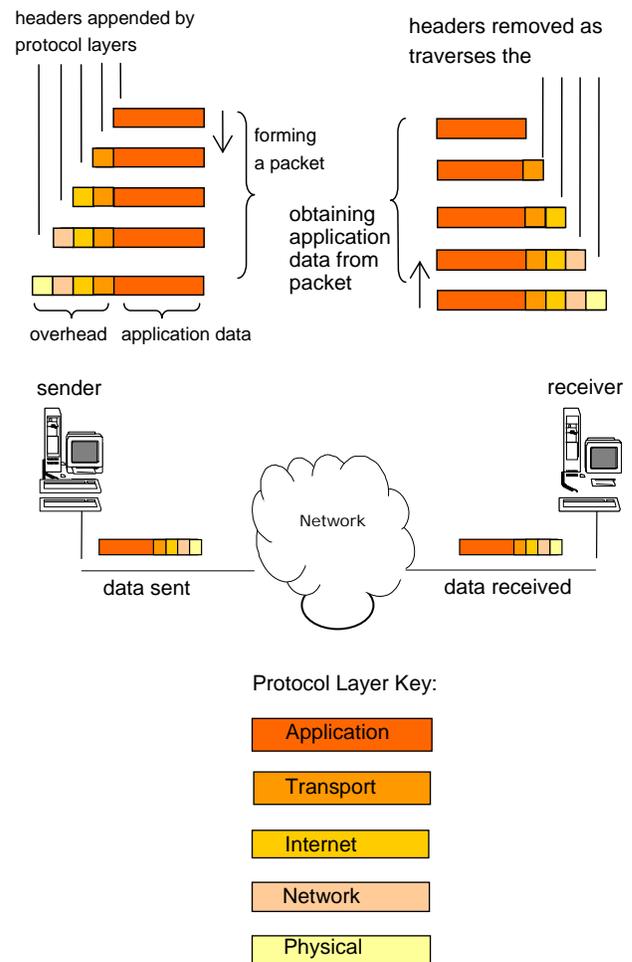

Figure 3. TCP/IP Layers and Data Transfer over Network.

In order to send or request information from a server (or network), a client must have a physical method of communication with the physical medium (e.g., coaxial cable, twisted-pair cable, fiber optic cable, or air - radio frequency). The network interface card (NIC), such as an Ethernet (IEEE 802.3) adapter, facilitates the communication between a client and the rest of the network. That is, the NIC allows a client to connect to the network.

The Hyper-Text Transfer Protocol (HTTP) is the protocol of the Web [1], [7]. HTTP defines the way Web clients (i.e., software allowing access to the Internet) and servers communicate with each other via the Internet [11]. Users must implement HTTP on both the client side and the server side for communication to occur. Users employ the HTTP protocol to define how each side (client and server) structure a request for the other side (client or server) to understand. In addition, users use HTTP to define the format of a client or server reply to a request.

To communicate with the .NET Framework, programmers use more than just the standard protocols. Microsoft employs the Simple Object Access Protocol (SOAP) to allow .NET implementations to communicate with non-.NET platforms [13]. SOAP is a simple eXtensible Markup Language (XML)-based protocol allowing applications to exchange information over the Internet using HTTP [2]. XML is a standard, is based on the Standard Generalized Markup Language (SGML) and is a format for structured documents and data on the Internet [3]. According to the World Wide Web Consortium (W3C) "XML has been designed for ease of implementation and for interoperability with both SGML and HTML" [3].

Similar to XML, SOAP is also a standard defined by the W3C. In addition, similar to the TCP/IP protocol, the SOAP protocol is also modular. As defined by the W3C, SOAP consists of three parts:

- The SOAP envelope construct defines an overall framework for expressing **what** is in a message; **who** should deal with it, and **whether** it is optional or mandatory.
- The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
- The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

The SOAP protocol is operating system and programming language independent and can work in conjunction with other protocols, such as HTTP [22]. Similar to protocols previously discussed, for a message to be understood, both the sender and receiver must implement SOAP.

Web Services

The basic goal of the .NET Framework is to provide users an easy way to build and disseminate Web services. A Web service is XML-based. It is transmitted over the Internet and allows applications to communicate with each other without consideration of the operating system or programming language [9]. Programmers use SOAP in the implementation of a Web service.

Communication between Web services is based on the client-server communication model, where a client is a machine housing the client application requesting the service and the server is a machine housing the server application responding to the request. Using the

SOAP protocol, the client makes a request and the server responds (see Figure 4).

The creator of a Web service must define the functionality of the Web service as well as the interface. Programmers use the Web Services Description Language (WSDL) for such definition [5]. The WSDL provides the user with information regarding the functionality and function for a Web Service [5]. The WSDL is basically an XML file describing the message encapsulated within a SOAP packet [4].
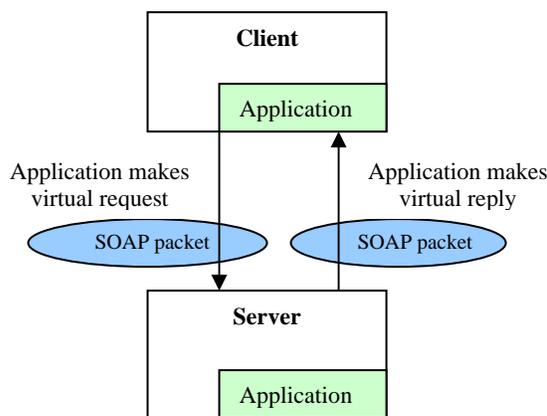

Figure 4. Web Service Communication.

Device Drivers

A device driver is software that controls the functionality of hardware or a peripheral device. Traditionally, programmers wrote drivers to interface with the CPU and to link with the kernel. The driver is part of the lowest level of the Operating System (OS). Through the kernel, drivers have access to other OS areas such as passwords as well as other important user information and functions. OS access to such sensitive information caused Windows users, as well as Microsoft, severe problems. Loopholes within the driver itself leave the OS open to spoofing and other hacker attacks. In

addition, if the driver crashes, the entire OS is susceptible to failure and corruption.

Currently, Microsoft uses the generic Windows Driver Model (WDM) based on the International Standards Organization's (ISO) Open Systems Interconnection (OSI) model. The OSI model emphasizes a layered-design approach where the application layer is the highest layer, and the physical layer is the lowest layer [12]. Similar to the TCP/IP model, each layer in the OSI model has a specific function and provides services to higher and lower layers.

When a client attempts to communicate with the server, most of the time the packet bits are communicated by going through a modem or a network interface card (NIC). NIC logic together with the specific OS compatible device driver would ensure that the packet bits are transmitted to the server application. The Network Driver Interface Specifications (NDIS) defines a standard Application Program Interface (API) for network driver development [19]. Developers use the NDIS to develop higher-layer (e.g., protocol driver) and lower-layer (e.g., miniport driver) drivers. The Media Access Controller (MAC; see Figure 5) hides the implementation details of the NIC so that it provides an interface to higher and lower layers. Therefore, "all NIC's for the same media (e.g., Ethernet) can be accessed using a common programming interface"[19]. Almost every version of Microsoft Windows has different NDIS's. Having different specifications is problematic for developers because they must constantly update their product drivers, which becomes very costly in terms of both capital and labor. In addition, new specifications are problematic for users because often times, drivers do not function properly with the OS or hardware

or users cannot locate new drivers. This paper will focus on network device driver design for Microsoft Windows XP because Microsoft recommends this OS for use of the .NET Framework.

Device drivers within Microsoft's Windows XP OS use the bottom four layers of the OSI model or the TCP/IP model (see Figure 5). In terms of the OSI model, the bottom four layers are the physical, data link, network and session. In terms of the TCP/IP model, the bottom four layers are the physical, network access, internet and transport. The network interface card (NIC) enables the physical layer and part of the network access layer (i.e., the Media Access Control (MAC) portion). Protocol drivers carry out the transport layer, internet layer, and a portion of the network access layer (i.e., the Logical Link Control (LLC) portion). As visible in Figure 5, the two models do not correspond exactly; however, both models are presented for descriptive purposes. Although the WDM is based on the ISO model, Microsoft (as most everyone else) actually implements the TCP/IP model.

Microsoft's Windows Driver Model (WDM) supports many hardware and peripheral devices, however, some devices require specific drivers. Due to poor security as well as numerous complaints from programmers, Microsoft is moving to the Windows Driver Foundation (WDF). The implementation of the model is similar to the WDM, however the WDF provides three main distinctions from the WDM: it has a kernel-mode driver framework, a user-mode driver framework, and driver verification tools [12].

Layer: OSI	Layer: TCP/IP	Implementation
Session	Transport (TCP)	- implemented by protocol drivers
Transport		
Network	Internet (IP)	- implemented by protocol drivers
Data Link: LLC	Network Access	- implemented by protocol drivers
Data Link: MAC		- implemented by NIC (miniport drivers)
Physical	Physical	- implemented by NIC, transceiver, and medium to which NIC is attached

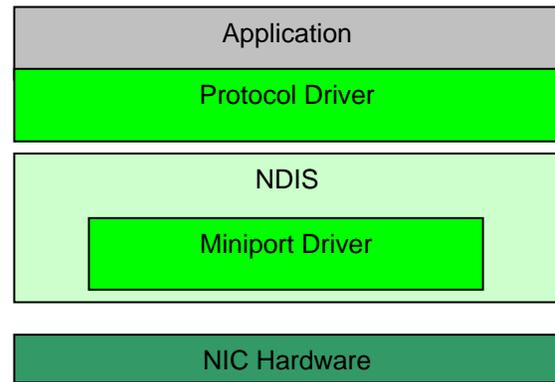**Figure 5. Implementation of the Four Layers used in Windows XP [18].**

Figure 6. Simple Driver Structure.

The WDF allows for a separation of the driver and the OS. First, unlike the WDM, the WDF is not OS specific. The OS version can be changed without having to modify the driver. Second, drivers do not have unrestricted access to the OS. The kernel-mode framework provides access into the kernel, however, not direct access. The framework is shielded from the rest of the OS, thus if the driver crashes, it will not cause system corruption. In addition, many of the drivers today do not need to run in kernel mode, thus they can use the user-mode driver framework. In this framework, drivers "do not have access to system structures or to the system virtual space" [12]. Therefore, if there is a problem with the driver, the system is not compromised. Lastly, programmers can employ the

driver verification tools to test their drivers to ensure Microsoft compliance.

Microsoft Windows XP supports four different drivers that run in the kernel-model: protocol drivers, miniport drivers, intermediate drivers and filter-hook drivers [14]. This paper will *not* focus on intermediate drivers and filter-hook drivers because they are not relevant in this discussion. Protocol drivers implement the transport layer; network layer and the logical link control of the data link (LLC) of the OSI model (see Figure 5). Figure 6 demonstrates that at its upper boundary, the protocol driver is a software interface between these layers and the higher-layers (i.e., application layer when using the TCP/IP model). At its lower boundary, the protocol driver interfaces with the NDIS. At its lower boundary, the NDIS interfaces with the media access control (MAC) of the data link layer. In Windows XP, the MAC is implemented by the miniport driver, which manages the NIC. At its lower boundary, the miniport driver employs the NDIS to communicate with the NIC hardware. At its upper boundary the miniport driver presents "an interface to allow protocol drivers to configure the adapter, as well as to send and receive packets over the network" [15].

When a client communicates with a server, the client-server connection occurs at the transport layer (based on the TCP/IP model). The client initiates the communication with the server by first sending a connection request. The server then responds stating that it is available and is ready to receive requests. As the data traverses through the TCP/IP layers on the client side, it is broken into packets and each layer adds a header. The transport layer also adds a sequence number to each packet to ensure reliable data transfer. That is, by providing a sequence number at the sender, the protocol ensures that all packets are received and reassembled in sequence by the receiver. The packets are sent through the NIC and onto the physical medium.

Programmers can also use "raw" sends and "raw" receives, which packets are sent using only the physical layer protocol. Packets would not need to traverse the upper-layer protocols before they are transferred because the data is not coming from the application or transport layers. Thus, programmers could write code to avoid higher-layer protocols. In avoiding the higher-layer protocols, the programmer is responsible for writing code that performs all the functions of the higher-layer protocols. That is, the programmer is responsible for the integrity and security of the data as well as making sure the code is not harmful.

Security

When considering web services, security is also a concern. Many Web services require users to divulge private information such as a credit card number. Internet users are increasingly concerned about identity theft and credit card fraud. Often, thieves obtain private information through hacking into corporate servers. The Microsoft Corporation, as well as companies and PC users, are well known for being victims of hacker attacks due to security issues within Microsoft products. Hackers can attack the server or the client.

One possible server attack is a denial of service, which hackers can accomplish through continuously sending requests to a server [23]. These requests force the server to do 'busy work' leading to the consumption of server resources. The server is unable to handle legitimate requests and eventually crashes or stalls. Another possible server attack is an

authorization attack [23]. Here the hacker attempts to access the server as a legitimate user through misrepresentation to the server. Such attacks are often successful due to poorly written code where fields are improperly validated or input is automatically assumed to be valid [10]. These attacks are devastating to corporations, where hackers break in and often steal credit card numbers or sensitive company information.

Hackers also attack clients. Most clients are individuals using their personal computer (PC) at home. The news often reports stories of hackers finding 'loop-holes' in the Microsoft Operating System's and devising viruses or directly accessing their system. One way to access the users (PC) is through holes in device drivers. Because device drivers are currently systems dependent, programmers must rewrite and continuously update their drivers. Unfortunately, sometimes the code is sloppy, or not well tested. Hackers therefore, can gain access into the PC and to the users' information. In addition, the hacker can than use the PC to launch attacks on a server.

Evaluation

This evaluation will demonstrate how a class of client applications can be developed to compromise the security of a .NET server. Through using "raw" packet sends and receives the application surpasses all protocols running higher than the Network Access layer protocol. "raw" sends and receives allow the application developer to append potentially compromised protocol headers for all layers higher than the physical layer. In this way the application can misrepresent any number of header fields (including client MAC and IP addresses) to the server.

We built a simple client application using the winpcap set of library calls [20] and compiled it with the Microsoft Visual C++ 2002, Standard. Our testbed consisted of a 3.06GHz Dell Optiplex Pentium 4 PC with a WindowsXP Pro operating system. The applications used in this evaluation were accessing a server machine through an IEEE 802.3 Ethernet compliant network interface card.

Since the .NET client is an HTTP aware application that can communicate with the server, it can run at the Microsoft protocol driver layer. Other layers below the protocol driver layer, e.g. miniport and intermediate drivers, cannot communicate with user-mode applications (see [21] for elaboration). Other possible implementations can be realized at the NDIS miniport driver level and below. Such applications are not implemented as part of this evaluation, because the class of applications demonstrated in this section have sufficient functionality to interact with the server in "raw" mode, that is, surpassing high layer protocols.

Figure 7 presents the algorithm for sending a "raw" packet. Lines 1 to 3 show the use of the pcap_findalldevs() WINPCAP function. Returned is a list of pcap_if structures with an internal name and human readable description of the device. pcap_open_live(), shown in lines 7 to 9, is used to obtain a packet capture descriptor to look at packets on the network. An argument is passed to pcap_open_live() to put the adapter in a promiscuous mode and allow for all traffic to be seen. In normal situations, an adapter only extracts the network traffic destined to it; the packets exchanged by other hosts are therefore ignored. On shared media (like non-switched Ethernet) WinPcap will be able to capture the packets of its own host as well as other hosts. Promiscuous

mode is the default for most capture applications.

```
    ...
 1.  obtain adapter list //
 pcap_findalldevs()
 2.  if no device available
 3.  exit
 4.  select adapter name
 5.  if no valid name selected
 6.  exit
 7.  open capture device //
 pcap_open_live()
 8.  if no adapter opened
 9.  exit
10.  send a "raw" packet //
 pcap_sendpacket()
11.  if no  packet
12.  return error
    ...
```

Figure 7. "Raw" Packet Send Algorithm.

To send a "raw" packet pcap_sendpacket() is called. It takes as an argument a buffer containing the data to send, its length and the adapter that will send it. The buffer is sent to the net as is, without any manipulation. Thus, the application has to create the correct protocol headers in order to send meaningful data. A similar set of calls is used to receive and parse a "raw" packet.

Summary and Future Work

This paper identifies a class of insecure .NET client applications, which avoid higher layer protocols protection through using the "raw" sockets API. The .NET Framework rests on many other Microsoft components, including the Windows Driver Model. This model supports four driver types, two of which were considered in this paper: protocol and miniport drivers. By compiling and executing client applications using the "raw" send and receive interface, we demonstrated that insecure clients can be written with minimal programming effort (lines of code) by using library calls, e.g. WINPCAP calls. Microsoft is currently moving towards a new driver model, Windows Driver Foundation, which is not yet available. Future work will include testing using this new driver model.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.1", IETF RFC 1945, May 1996; http://www.rfc-editor.org/rfc/rfc1945.txt.

[2] D. Box, et al., "Simple Object Access Protocol (SOAP) 1.1", World Wide Web Consortium (W3C) note, May 2000; http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[3] T. Bray, et al., "Extensible Markup Language (XML) 1.1", World Wide Web Consortium (W3C) recommendation, April 2004; http://www.w3.org/TR/2004/REC-xml11-20040204/.

[4] K.F. Chavda, "Anatomy of a Web Service", *Journal of Computing Sciences in Colleges*, The Consortium for Computing in Small Colleges, Little Rock, AR, Jan 2004, pp.124 – 134.

[5] E. Christensen, et al., "Web Services Description Language (WSDL) 1.1", World Wide Web Consortium (W3C) note, March 2001; http://www.w3.org/TR/wsdl.

[6] D. III. Eastlake, J. Reagle, and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", IETF and W3C RFC 3275, March 2002; http://www.rfc-editor.org/rfc/rfc3275.txt.

[7] R. Fielding, et al., "Hypertext Transfer Protocol - HTTP/1.1", IETF RFC 2616, June 1999; http://www.rfc-editor.org/rfc/rfc2616.txt.

[8] Foundstone, Inc., and CORE Security Technologies, "Security in the Microsoft .NET Framework", Foundstone Strategic Security White Paper, 2003; http://www.foundstone.com/resources/whitepapers/dotnet-security-framework.pdf.

[9] H. Haas, "Web Services Activity Statement", World Wide Web Consortium (W3C) activity statement, April 2004; http://www.w3.org/2002/ws/Activity.

[10] A. Klein, "Secure Coding Practices for Microsoft .NET Applications", Sanctum, Inc., White Paper, 2003.

[11] J.F. Kurose, and K.W. Ross, *Computer networking: A Top-Down Approach Featuring the Internet,* Pearson Education, New York, NY, 2003.

[12] Microsoft Corporation, "Introduction to the Windows Driver Foundation", June 2004; http://msdn.microsoft.com/library/default.asp?url=/library/enus/dndevice/html/WDF_intro.asp.

[13] Microsoft Corporation, ".NET Framework Developer's Guide: Overview of the .NET Framework"; http://msdn.microsoft.com/library/default.asp?url=/library/enus/cpguide/html/cpovrintroductiontonetframeworksdk.asp.

[14] Microsoft Corporation, "Network Devices and Protocols: Windows DDK: NDIS Protocol Drivers", June 2004; http://msdn.microsoft.com/library/default.asp?url=/library/

enus/network/hh/network/302pro_a648250f-4e5c-4140-bd7ca1b8f3b7a154.xml.asp.

[15] Microsoft Corporation, "Network Devices and Protocols: Windows DDK: Obtaining and Setting Miniport Driver Information and NDIS Support for WMI", June 2004; http://msdn.microsoft.com/library/default.asp?url=/library/enus/network/hh/network/205mpinfo_ab586ccb-6399-420f-89d9-6a32b20b9e6f.xml.asp.

[16] M. Murata, S. St.Laurent, and D. Kohn, "XML Media Types", IETF RFC 3023, January 2001; http://www.rfc-editor.org/rfc/rfc3023.txt.

[17] E. O'Tuathail, and M. Rose, "Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP)", IETF RFC 3288, June 2002; http://www.rfc-editor.org/rfc/rfc3288.txt.

[18] Open Systems Resources, Inc. "Windows 2000 and Later Network Architecture and the OSI Model", April 2003; http://www.osr.com/ddk/network/102gen_07vr.htm.

[19] PCAUSA, "What is NDIS?", June 2004; http://www.ndis.com/.

[20] "Packet Capture Architecture for Windows, Version 3.0 (WinPcap)", URL: http://winpcap.polito.it/.

[21] M. Rose and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", IETF RFC 1155, May 1990; http://www.rfc-editor.org/rfc/rfc1155.txt.

[22] A. Skonnard, "Understanding SOAP", Microsoft Corporation, March 2003; http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnsoap/html/understandsoap.asp.

[23] W. Stallings, *High-Speed Networks and Internets: Performance and Quality of Service, 2nd,* Pearson Education, New York, NY, 2001.

[24] P. Tapadiya, *.NET programming: A Practical Guide Using C#,* Prentice Hall, Upper Saddle River, NJ, 2002.

[25] A. Weiss, "Microsoft's .NET: Platform in the Clouds", *netWorker*, vol. 5, no. 4, Dec 2001, pp. 26 – 31.