Enhancement of WIRE1x Network Adapter Control Module

Yu-Chen Liu, Yong-Pei Zhang, Shang-Chun Chen, Fu-Wen Chen, Tung-Hsueh Lu, Yi-Ken Ho, and Jyh-Cheng Chen
Wireless Internet Research and Engineering Laboratory
National Tsing Hua University
Hsinchu, Taiwan

December 22, 2008

Abstract

This document presents the design and implementation of WIRE1x for Windows Mobile 6.0. WIRE1x is an open-source implementation of IEEE 802.1x client (supplicant) developed by Wireless Internet Research and Engineering (WIRE) Laboratory, National Tsing Hua University. The former versions only support Windows Vista/XP/2000/98. The most significant difference between the WIRE1x v2.4 and WIRE1x_Mobile is that we use Network Driver Interface Specification (NDIS) to implement the control module of Network Interface Cards (NIC) on PDA. The motivation for developing WIRE1x network adapter control system for WIRE1x is to solve the problem which WIRE1x faced on Windows Mobile 6.0.
.

1. Introduction

Because of the popularity of mobile communication, more and more people want to use their mobile device to connect to Internet by Wireless Local Area Networks (WLAN). The implementation of WIRE1x for Windows Mobile 6.0 is just to satisfy the vast need.

WIRE1x is an open-source implementation of IEEE 802.1x [1] client (supplicant) developed by the Wireless Internet Research & Engineering (WIRE) Laboratory, National Tsing Hua University. [2] IEEE 802.1x is a part of IEEE 802.1 group of network protocol. It provides a port-based mechanism for devices which want to communicate with the authentication server by the Extensible Authentication Protocol (EAP).

The implementation of WIRE1x is based on Open1x which supports Linux only.

Since there are few open-source software provide an usable interface of multi-authenticated methods in Windows environment which is used by most people, the primary purpose of WIRE1x is just to escape from the predicament.

WIRE1x provides several EAP authentication methods, including EAP-MD5 [3], EAP-TLS [4], EAP-TTLS [5], EAP-PEAP [6], EAP-SIM [7], EAP-AKA [8], EAP-FAST [9] and MSCHAPv2 [10]. It also supports WEP, WPA with TKIP, and WPA2 with CCMP in encryption. We have also released versions for XP/2000/98, Vista, and Windows Mobile 6.0, and the customized versions for National Tsing Hua University.

The rest of this document is organized as follows. In Section 2 we introduce to NDIS, WZC service, and how we rewrite the NDIS function to control network adapter system. Finally, Section 3 concludes the contents above.

2. Implementation of NDIS Library

WinPcap[11] offer a library tool function for link-layer network access on Windows environments. But the version of WinPcap for WinCE is not a tested and reliable one. So we have to rewrite the network adapter control module of WIRE1x on Windows Mobile system. WinPcap also use NDIS [12] library, but the difference is that it is pack as a DDL that we can use on any platform. In this project, we use the NDIS library to implement functions for controlling network adapter, sending and receiving frames, connecting to AP and disassociating to AP. The detailed illustrations and these important functions will be explained below. Some of the source code is based on PeekPocket [13], which is a hassle-free Wi-Fi AP scanner for Windows Mobile.

2.1 Introduction to WinPcap and NDIS

The Windows Packet Capture Library (WinPcap) is a set of industry-standard tool using in accessing to Data Link Layer. It allows applications catch and transmit packets without dealing with protocol stack. WinPcap has many other features such as kernel-level packet filtering, a network statistics engine and support for remote packet capture.

WinPcap contains a driver which allows low-level network control provided by operating system, and WinPcap also contains a library which can be used in accessing the low-level network layer. The library above contains the Windows version of libpcap UNIX API.

Because of these features of WinPcap, it is used in many open source and commercial network tools which include protocol analyzers, network monitors, network instruction detection systems, sniffers, traffic generator, and network testers. Some of these well-known tools such as Wireshark, Nmap, Snort, ntop, are known and used throughout the network community.

Network Driver Interface Specification is short to NDIS. The primary purpose of NDIS is to define a standard API for "Network Interface Cards" (NIC) to communicate with network systems. For example, Network cards receive packets from Transmission Control Protocol/Internet Protocol (TCP/IP) stack through NDIS and TCP/IP also sends packets to network drivers through NDIS. The details of hardware implementation of NIC is wrapped by a "Media Access Controller" (MAC) device driver so all NICs for the same media (e.g., Ethernet) can be accessed using a common programming interface.

In figure 1, we shows that NDIS also provides a library of functions (sometimes called a "wrapper") that can be used by MAC drivers, miniport drivers, intermediate drivers as well as higher level protocol drivers (such as TCP/IP). The wrapper functions can make development of both MAC and protocol drivers easier as well as hide (some extent) platform dependencies.

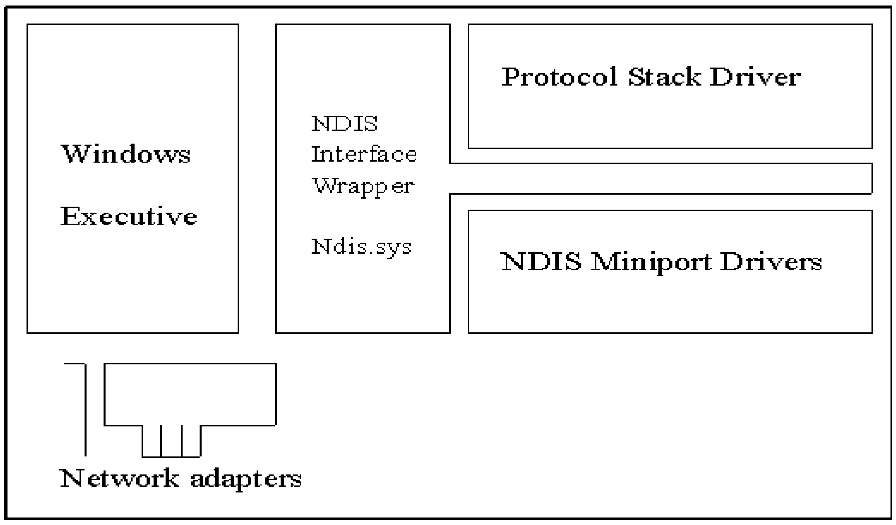

Figure 1 –NDIS architecture [14]

2.2 Introduction to WZC service

Wireless Zero Configuration (WZC)[15] is a wireless connection management utility included with Microsoft Windows XP and later operating systems as a service that dynamically selects a wireless network to connect to based on a user's preferences and various default settings.

The operating system we use is Windows Mobile 6.0. But the Wireless Zero Configuration service on this system attempts to perform an 802.1x shared key authentication if the network adapter has been preconfigured with a WEP shared key.
If the network adapter is not preconfigured with a WEP shared key, the network adapter will revert to open system authentication. Since in our project, we not only use WEP shared key but also WPA and WPA2 thus we have to close WZC. And it won’t break the connection when we use WPA and WPA2 to authenticate. Also it is inconvenient for us to do many settings on WZC if we want to authenticate. So we close WZC to let us to do the settings by ourselves. The additional advantage of setting adapters by ourselves is that we can guarantee the safety of each authentication methods.

We use pcap_SetWzc() to close the WZC service, and open the service after user quit the WIRE1x. Unfortunately, we have to deal with the value of registry for connecting to Internet by using IE. We use pcapHackRegForIE() to solve the question above.

2.3 Function description

We use NDIS to implement network adapter control modules, including initial procedure, scan and get AP information, connect and disconnect to AP, send and receive frame, and other setting function for Windows relation service handler.

We demonstrate the flow and depict the architecture of WIRE1x network adapter control module with figure 2.

1. While user starts to execute WIRE1x_mobile, we use pcap_OpenDriver(), pcap_OpenDevice(), Pcap_GetAdapters(), and Pcap_SetWzc() to set the environment at first.
2. When user triggers the SCAN button to scan AP, pcap_GetApInfo(), pcap_RefreshBSSID(), and pcap_GetBSSID() will search the available AP and then get the information of them to show on the screen for user, such as signal quality, max rate, and security type of selected AP.
3. After user set the ID, password, EAP method, and whether he wants to

verify server’s certificate, he can now trigger ASSOCIATE button to connect to the AP he chose. And then we use pcap_PrepareToConnect() and pcap_ConnectToAPName() to do some setting for connecting to AP. We also rewrite the functions of sending and receiving packets, including send_frame() and get_frame().

4. For the modules we designed, we use pcap_setOid() and pcap_getOid to deal with the NDIS OID. We also use pcapHackRegForIE() to solve the problem of surf the Internet by using Window Internet Explorer.

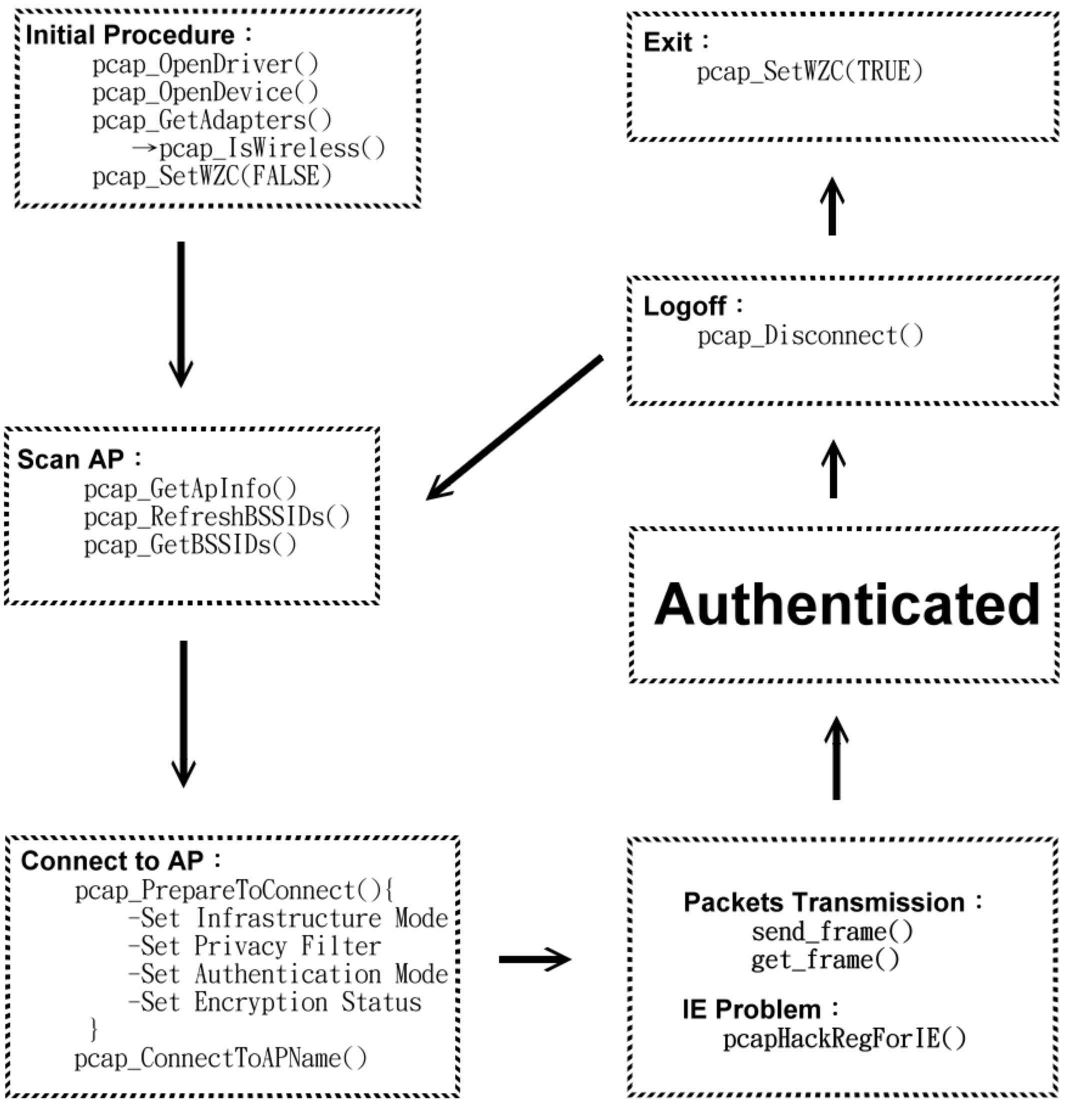

Figure 2 – Main function used in WIRE1x_mobile

2.3.1 Initial procedure while starting:

- **pcap_OpenDriver()**：

In this function, we open a driver of NDIS user mode I/O to set information and inquire information. In this function, we call a function CreateFile(). This function can create or open a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified. We use the returned handle to control the network adapter.

- **pcap_OpenDevice()**：

  In this function we do four things. First, we call CreateFile() to return a new handle and then to let NDIS unbind and then rebind the network adapter to one or all NDIS protocol drivers. Second, we use pcap_OpenDevice() to associates a file handle obtained through a CreateFile() to a network adapter. Third, we use the function pcap_setOid() to set the packet filter for directed packets, which contained a destination address. Last, we set Ethernet Type(0x888e) for eapol packets. By default, the NDISUIO only receive the of the packets with Ethernet Type(0x8001) header.

- **pcap_GetAdapters()**：

  Because there are more adapters at the mobile device, such as adapters for network, infrared, GPRS, or ActiveSync, we use this function to get the network adapter names, and pick the first one to use if there is more than one adapter in this mobile device. However, user can change the network adapter when he clicked the button of “Change Network Interface”, and then we would show the user all available adapter names because we store all names in an array so that he can change the NIC (Network Interface Card) he wants to use.

2.3.2 Scan AP Informations:

1. **pcap_GetApInfo()**：

   In this function, we get the information of the scanned AP, such as name, intensity of the signal, MAC address, transfer rate, and security support way of AP, etc. And then we use arrays to store the information. Also, between calling pcap_RefreshBSSIDs() and pcap_GetBSSIDs(), we wait for 2 seconds to let the device have enough time to scan the available AP.

2. **pcap_RefreshBSSIDs()**：

This function will be called in the function pcap_GetApInfo(). When calling this function, the OID_802_11_BSSID_LIST_SCAN OID requests that the miniport driver perform a network scan of BSSIDs and SSIDs. The driver will then be asked for the scan results and the device will use active scanning methods, passive scanning methods or a combination of both methods to perform a scan.

3. **pcap_GetBSSIDs()**：

This function will be called in pcap_GetApInfo(), and it uses pcap_getOid() get all APs' attributes after that pcap_GetApInfo() called pcap_RefreshBSSIDs() and wait for 2 seconds, including length, MAC address, Service set identifier(SSID), Received Signal Strength Indication(RSSI), authentication type, Infrastructure Mode, frequency, max rate, authentication information element(IEs, IELength).

At first, we store those attributes of different AP, and then we set authentication type as privacy, and then use the information return by pcap_GetAuthModeFromIEs() to decided if we should change the authentication type. If WPA or WPA2 is enabled, there must be a corresponding IE data, so we change the attribute of the AP's Authentication into Ndis802_11AuthModeWPA_XP or Ndis802_11AuthModeWPA2_XP. After that, we use pcap_driver_ndis_get_ies() to get IE data from AP. Then we call pcap_wpa_parse_wpa_ie_wpa() to parse WPA IE data and call pcap_wpa_parse_wpa_ie_rsn() to parse WPA2 IE data and use pcap_wpa_supplicant_set_suites() to save the parsing IE result back into scan result structure

2.3.3 Setting before connecting to AP

We use pcap_PrepareToConnect() to do some setting before connecting to the AP. There are four steps. At first, we reset the device by use OPEN_NONE type to run the four steps. After that, we set the real type that user selected.

After calling pcap_PrepareToConnect(), we then use pcap_ConnectToAPName() to set OID for connecting to selected AP.

- **pcap_PrepareToConnect()**：

Step 1. Set Infrastructure Mode：

When the OID_802_11_INFRASTRUCTURE_MODE OID set this mode by using pcap_setOid(), the function requests that the miniport driver set its network mode to a specific mode such as ad-hoc mode, extended service set (ESS) mode, or switch between ad-hoc mode and ESS mode.

Step 2. Set Privacy Filter：

When the OID_802_11_PRIVACY_FILTER OID set this mode by using pcap_setOid(), the function requests that the miniport driver set its IEEE 802.1X privacy filter to a specific mode such as Accept-all mode to receive any non-encrypted packets or packets which encrypted successfully, and 802.1xWEP mode to receive IEEE 802.1X packets.

Step 3. Set Authentication Mode：

When the OID_802_11_AUTHENTICATION_MODE OID set this mode by using pcap_setOid(), the function requests that the miniport driver set its IEEE 802.11 authentication mode to the specific mode as shown in Table 2.

Step 4. Set Encryption Status：

When the OID_802_11_ENCRYPTION_STATUS OID set this mode by using pcap_setOid(), the function requests that the miniport driver change its encryption mode. Three encryption modes show as in the Table 1.

Encryption1	WEP encryption is supported.
Encryption2	WEP and TKIP encryption is supported.
Encryption3	WEP, TKIP, and AES encryption is supported.

Table 1- Encryption Status

The table below describes the required parameters of Authentication Mode, Privacy Filter, and Encryption Status corresponding to each type user selected.

Parameter / Type	**Authentication Mode Privacy Filter Encryption Status**
OPEN_NONE	**Ndis802_11AuthModeOpen Ndis802_11PrivFilterAcceptAll Ndis802_11EncryptionDisabled**
OPEN_WEP	**Ndis802_11AuthModeOpen Ndis802_11PrivFilterAcceptAll**

	Ndis802_11Encryption1Enabled
SHARED_NONE	**Ndis802_11AuthModeShared** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11EncryptionDisabled**
SHARED_WEP	**Ndis802_11AuthModeShared** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11Encryption1Enabled**
WPA	**Ndis802_11AuthModeWPA** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11Encryption2Enabled**
WPA_PSK	**Ndis802_11AuthModeWPAPSK** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11Encryption2Enabled**
WPA2	**Ndis802_11AuthModeWPA2** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11Encryption3Enabled**
WPA2_PSK	**Ndis802_11AuthModeWPA2PSK** **Ndis802_11PrivFilter8021xWEP** **Ndis802_11Encryption3Enabled**

Table 2- Parameters of each type

- **pcap_ConnectToAPName()**：

  In this function, at first we convert the name of AP from Wide-character format to ANSI format. And then we set the OID_802_11_SSID OID by using pcap_setOid(), the miniport driver would set the name of AP of the basic service set(BSS) with which this mobile device can connect.

2.3.4 Send packets and receive packets

The original function of transmit packets can not be used on mobile device, so we rewrite the following functions.

- **Send packets**：

  In send_frame(), we use WriteFile() to send out packets to AP.

- **Receive packets**：

  Unlike sending packets, the action of receiving packets is always in motion. First, we create a thread for pcap_ReadFile() to receive packets continuously, and store packets in a circular queue. After that, when calling get_frame(), the packets in queue will be taken out.

The use of queue for storing packets is that to avoid missing packets. When receiving packets, we call pcap_ReadFile() to receive packet from AP continuously, and then call get_frame() to get the packets. If the interval of coming packets is too small, we will get the wrong packets. To solve the problem, we store the packets received in a queue at pcap_ReadFile() and let get_frame() to get the packets from queue.

Because the queue of storing temporary packets is circular, the old packets will be replaced by the new ones.

2.3.5 Other functions

- **pcap_setOid() and pcap_getOid()**：

  OID represents a cryptographic object identifier and the NDIS miniport driver provides control of NIC for sending and receiving data. pcap_setOid() allows an application to set an OID of a miniport driver. We set some OID values of the miniport such as Infrastruture_Mode, Authentication_Mode, Privacy_Filter. pcap_getOid() allows an application to query an OID of a miniport driver. We query some data from the miniport. For example, we use it to check whether the interface card is wireless and also to query information of available AP list.

- **pcapHackRegForIE()**：

  We use this function to set the value of registry for IE(Windows Internet Explorer).When we start to execute WIRE1x_mobile, WZC(Wireless Zero Configuration) would be closed by WIRE1x_mobile because it would interfere with the authentication process. But close WZC, we will not surf the Internet by using IE, so we have to set the value of registry as WZC does. First, we find which values should be set by WZC originally and query some values of the mobile device. Then create a new key to set the values, including Adapter, AlwaysOn (data connection type), ConnectionGUID, DestId, Secure, SecureLevel, and Default(SSIDName).

- **pcap_SetWzc()**：

  Deactivate WZC service when starting WIRE1x_mobile and activate it when closing WIRE1x_mobile. When we want to activate WZC service, we just use the function ActivateDevice(WZC_DRIVER,0). On the other hand, if we want to deactivate WZC service, we use the function DeactivateDevice((HANDLE) hnd). Before using the deactivate function, we should find a handle for WZC driver at first. To get this handle, we must get the first handle of key, which is obtained from the mobile device. Then we enumerate first handle of key to get the second handle. Finally, we get the handle from second handle for WZC driver.

- **pcap_IsWireless()**：

  This function is called in pcap_GetAdapters(), and we use it to check whether the network adapter we get is wireless or not. We set the OID_GEN_PHYSICAL_MEDIUM OID by using pcap_getOid(), and then the miniport driver would return a value that which type of physical medium that NIC supports. If the returned value is equal to NdisPhysicalMediumWirelessLan, the network adapter is wireless.

3. Conclusion

Due to the popularity of mobile communication, the need of WLAN linking by mobile will be urgency. After a year hard working, we finally implement the WIRE1x for Windows Mobile 6.0, and this version has been tested and released on WIRE1x website.

This document describes the control module of how we port WIRE1x to Windows Mobile 6.0. At first, we introduce WinPcap and NDIS before explaining the functions we rewrite. Then we particularly describe each function rewrite from WinPcap, and explain the reason why we have to close the WZC service. We additionally show how we close WZC service.

It is admitted that the implementation of NDIS on Windows Mobile 6.0 will be extensively used by whom are interesting in develop application on mobile device, and it is a pleasure to improve WLAN research. After our implementation, we have a convenient way to connect to Internet via mobile device. We are sure that WIRE1x will be a more usable, more functional, and more portable software via WIRE1x Mobile.

4. Reference

[1] IEEE Standard 802.1x-2004, “IEEE standard for local and metropolitan area

networks, port-based network access control," December 2004

[2] "WIRE1x" http://wire.cs.nthu.edu.tw/wire1x

[3] "EAP-MD5" RFC3748: Extensible Authentication Protocol (EAP) (June 2004)

[4] "EAP-TLS" RFC2716: PPP EAP TLS Authentication Protocol (October 1999)

[5] "EAP-TTLS" RFC5281: Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0) (August 2008)

[6] "EAP-PEAP" RFC2284: PPP Extensible Authentication Protocol (EAP) (March 1998)

[7] "EAP-SIM" RFC4186: Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) (January 2006)

[8] "EAP-AKA" RFC4187: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA) (January 2006)

[9] "EAP-FAST" RFC4851: Flexible Authentication via Secure Tunneling (May 2007)

[10] "MS-CHAPv2" RFC2759: Microsoft PPP CHAP Extensions, Version 2 (January 2000)

[11] "WinPcap" http://www.winpcap.org

[12] "NDIS" http://www.ndis.com/

[13] "PeekPocket" http://dzolee.blogspot.com/

[14] "NDIS Architecture" http://www.microsoft.com/technet/prodtechnol/windows2000serv/reskit/cnet/cnad_arc_vepi.mspx?mfr=true

[15] "Windows CE WZC" http://msdn.microsoft.com/en-us/library/ms885812.aspx