Extensible Authentication Protocol (EAP) and IEEE 802.1x: Tutorial and Empirical Experience

JYH-CHENG CHEN AND YU-PING WANG, NATIONAL TSING HUA UNIVERSITY

Abstract

This article presents the technical details of the Extensible Authentication Protocol (EAP) and IEEE 802.1x by using WIRE1x, an open-source implementation of IEEE 802.1x client (supplicant) and various EAP-based authentication mechanisms. By using a real implementation, 802.1x and EAP should be easily understood.

Introduction

Wireless local area networks (WLANs) have become increasingly more prevalent in recent years. The IEEE 802.11 standard is one of the most widely adopted standards for broadband wireless Internet access. However, security considerations with regard to wireless environments are more complicated than those in wired environments. Due to the wide-open nature of wireless radio, the network is more vulnerable. The original IEEE 802.11 standard [1] has defined the following two basic security mechanisms for securing access to IEEE 802.11 networks:

- Entity authentication, including *open-system* authentication and *shared-key* authentication
- Wired Equivalent Privacy (WEP)

Nevertheless, they have all proven to be vulnerable.

To enhance the security in IEEE 802.11, IEEE 802.11i [2, 3] has been proposed. In addition to introducing protocols for *key management and establishment*, it also defines *encryption* and *authentication* improvements. In order to manage security keys automatically, IEEE 802.11i has defined algorithms and protocols for key management and establishment. As conventional WEP is known to be vulnerable, IEEE 802.11i has specified enhanced encryption algorithms in order to provide stronger privacy. IEEE 802.11i also incorporates IEEE 802.1x [4] as its authentication enhancement. The IEEE 802.1x standard is a port-based network access control used to authenticate and authorize devices interconnected by various IEEE 802 LANs. IEEE 802.11i is expected to play a critical role in improving the overall security of current and future WLANs.

The IEEE 802.1x standard has been well defined. Currently, many manufacturers of 802.11 access point (AP) also support 802.1x. The 802.1x-capable APs have been deployed in many universities, organizations, and companies. To be authenticated using 802.1x, end users also need to be 802.1x capable. Unless 802.1x is embedded in the operating system (OS), users generally will need to install 802.1x client software in order to access to the network. Open1x (http://www.open1x.org/), an open-source implementation of 802.1x, supports Linux. Its earlier versions also supported BSD and Mac OS. Because many users are eager to use free software for the 802.1x client to be able to work with current and earlier versions of MS Windows, we therefore have developed *WIRE1x* to support various versions of MS Windows. As the name suggests, WIRE1x is an open-source implementation of IEEE 802.1x client (supplicant)[1] developed by the Wireless Internet Research & Engineering (WIRE) Laboratory.[2] Both source code and executable code of WIRE1x can be downloaded freely from http://wire.cs.nthu.edu.tw/wire1x/.

Essentially, 802.1x provides a framework for port-based access control. It can work with various authentication mechanisms to authenticate and authorize users. The Extensible Authentication Protocol (EAP, IETF RFC 2284) is a protocol commonly used in 802.1x to authenticate users. Currently, WIRE1x provides various authentication mechanisms, including EAP Message Digest 5 (EAP-MD5, IETF RFC 1321), EAP Transport Layer Security (EAP-TLS, IETF RFC 2716), EAP Tunneled TLS (EAP-TTLS) [5], and Protected Extensible Authentication Protocol (PEAP) [6]. It also supports MSWindows XP, Windows 2000,Windows ME, and Windows 98.

Based on our experience in implementing WIRE1x, this article presents a tutorial of 802.1x and EAP. The article is organized to facilitate the reader's understanding 802.1x and EAP from a real implementation. It is expected that readers will not only understand 802.1x and EAP, but will also be able to examine the source code of WIRE1x.

The rest of the article is organized as follows. In the second section, we introduce WIRE1x and describe the three of its major components. The three components are then presented in detail in the third, fourth, and fifth sections, respectively. The last section concludes the article.

WIRE1x

WIRE1x is an implementation of an IEEE 802.1x client. It is a free as well as open-source software. WIRE1x is licensed under the BSD License and GNU General Public License (GPL). We believe open source is essential for any security-related software because it can be examined as one wishes. As mentioned above, WIRE1x provides various EAP-based authentication mechanisms, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP. It can work with various versions of MS Windows, including Windows XP, 2000, ME, and 98. WIRE1x also works well with various types of WLAN cards. It has been practically used in real-word applications with FreeRADIUS (http://www.freeradius.org/) to secure WLAN environments. The implementation of WIRE1x is based on

[1] *Supplicant is a terminology defined in 802.1x and is described in the third section of this article.*

[2] *http://wire.cs.nthu.edu.tw/.*

0163-6804/05/$20.00 © 2005 IEEE

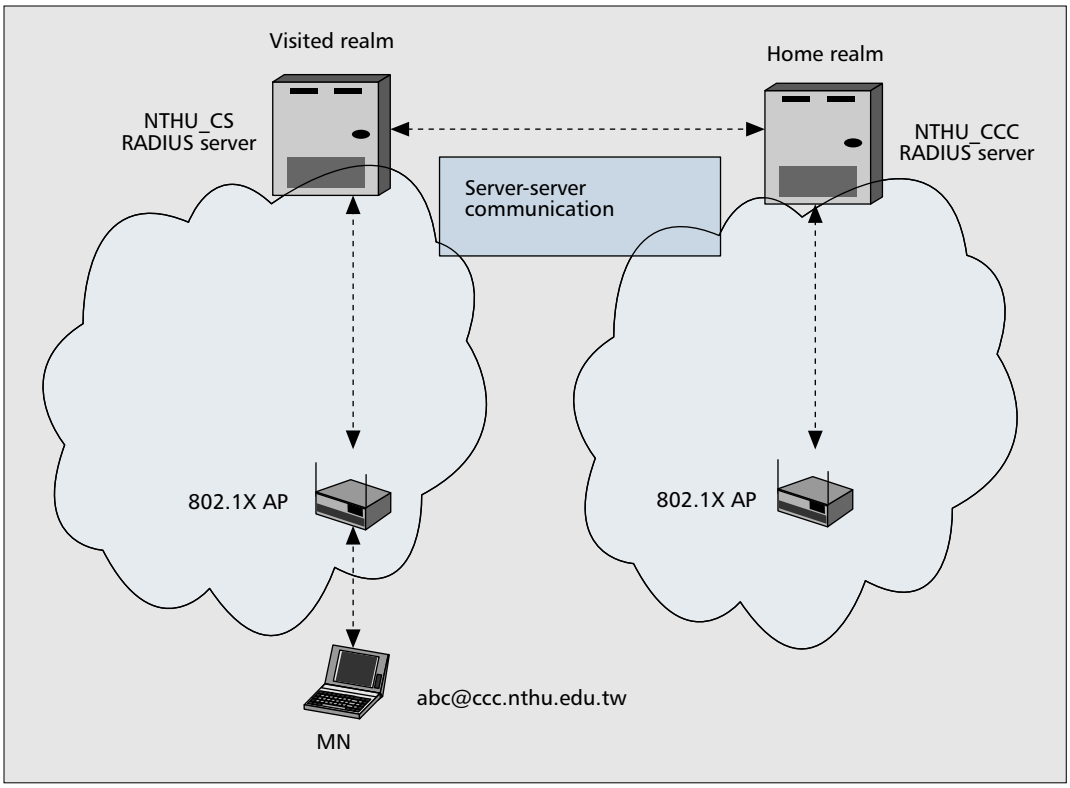

FIGURE 1. *Authentication on a visited network.*

Open1x and developed by using MS Visual C++. It also utilizes open-source libraries of WinPcap (http://winpcap.polito.it/), Libnet (http://libnet.sourceforge.net/), and OpenSSL (http://www.openssl.org/). *WinPcap* and *Libnet* are responsible for capturing and writing packets to and from the data link layer. Additionally, *OpenSSL* is used only for TLS-based authentication mechanisms.

WIRE1x has been used practically at the National Tsing Hua University (NTHU). At NTHU, each department/institute is responsible for the deployment of networking facilities inside its own building(s). The university has no authority over the areas owned by the department/institute. The Computer & Communication Center (CCC) operated by the university is responsible for the networking facilities in public areas on campus. Thus, following standards is essential for roaming and integration of WLAN environments, even inside the same university.

Both CCC and the Computer Science (CS) department at NTHU have deployed WLANs by using 802.1x and RADIUS (IETF RFC 2865) to authenticate users. To roam between different administrative domains, both of their RADIUS servers can be connected together, as shown in Fig. 1. We assume a user *abc* who has an account *abc@ccc.nthu.edu.tw* and belongs to the CCC. Once the user roams into the WLANs covered by the CS department, the CS RADIUS server can authenticate the user by relaying the authentication messages back to the CCC RADIUS. The CS RADIUS server acts as a proxy client to the CCC RADIUS server. With only one account at CCC, the 802.1x client can still roam into other WLANs.

The software architecture of WIRE1x can roughly be divided into three components, as illustrated in Fig. 2:

- Supplicant Port Access Entity (PAE) state machine
- EAP and authentication mechanisms
- WinPcap, Libnet, and OpenSSL

The *supplicant PAE state machine* follows the specifications defined in IEEE 802.1x. The third section describes 802.1x and the supplicant PAE state machine. As the name suggests, the *EAP and authentication mechanisms* support EAP and various authentication mechanisms. The forth section presents and compares the EAP-based authentication mechanisms. The fifth section describes the open-source libraries, including WinPcap, Libnet, and OpenSSL, used in WIRE1x.

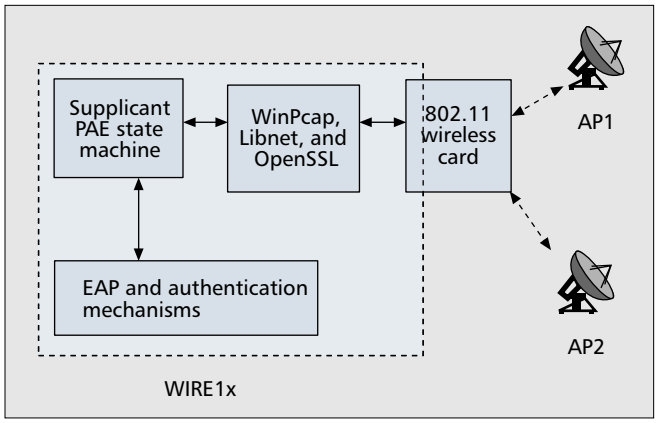

FIGURE 2. *Software architecture of WIRE1x.*

802.1x

IEEE 802.1x defines a mechanism for port-based network access control. It is based upon EAP to provide compatible authentication and authorization mechanisms for devices interconnected by IEEE 802 LANs. As depicted in Fig. 3, there are three main components in the IEEE 802.1x authentication system: supplicant, authenticator, and authentication server. In a WLAN, the *supplicant* is usually a mobile node (MN). The AP usually represents an *authenticator*. An authentication, authorization, and accounting (AAA) server such as

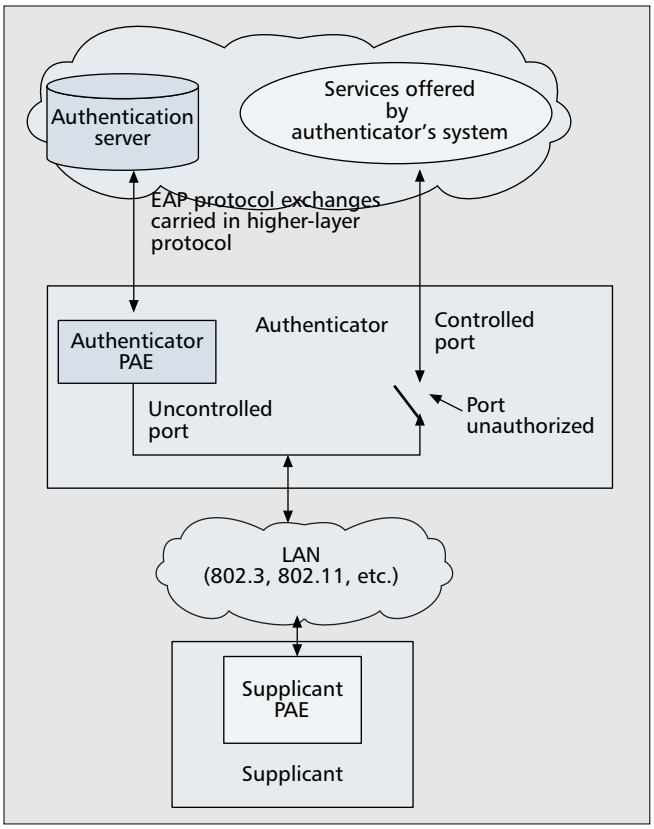

FIGURE 3. *Supplicant, authenticator, and authentication server.*

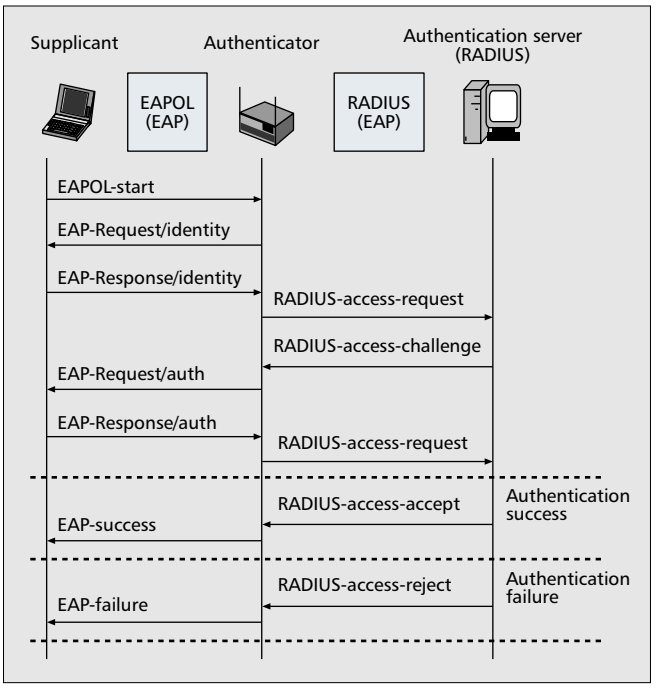

FIGURE 4. *A typical 802.1x message flow.*

the RADIUS server is the *authentication server*. The *port* in 802.1x represents the association between the supplicant and the authenticator. Both supplicant and authenticator have a PAE that operates the algorithms and protocols associated with the authentication mechanisms. In Fig. 3, the authenticator's *controlled port* is in unauthorized state, that is, the port is open. Messages will be directed only to the *authenticator PAE*, which will further direct 802.1x messages to the authentication server. The authenticator PAE will close the controlled port after the supplicant is authenticated successfully. Thus, the supplicant is able to access to other services through the controlled port.

Based upon EAP, the IEEE 802.1x standard can use a number of authentication mechanisms. The authentication mechanisms are outside the scope of the IEEE 802.1x standard. Many authentication mechanisms such as MD5, TLS, TTLS, and PEAP can be used. The IEEE 802.1x also defines EAP over LANs (EAPOL) in order to encapsulate EAP messages between the supplicant and the authenticator. The authenticator PAE relays all EAP messages between the supplicant and the authentication server. The 802.1x is utilized to enforce the use of specific authentication mechanism and to route authentication messages properly, while the authentication mechanisms define the actual authentication exchanges that take place. Fig. 4 shows a typical 802.1x message exchange. In Fig. 4, RADIUS serves as the authentication server. This does not limit the use of other AAA servers such as Diameter (IETF RFC 3588) as the authentication server. A detailed discussion of the PAE state diagrams and state transitions in supplicant and authenticator can be found in [3].

The supplicant PAE state machine is the major component of any implementation of the 802.1x supplicant. It specifies the behavior of the supplicant and interacts with the authenticator. In WIRE1x, roughly speaking, the supplicant PAE state machine is implemented in four files: dot1x_globals.cpp, eap.cpp, eapol.cpp, and os_generic.cpp. All variables of the state machine are defined in dot1x_globals.h. Additionally, the EAP *code field* and *type field* specified in IETF RFC 2284 are defined in eap.h. The eap.cpp is responsible for building the response frames and decoding the EAP packets. Moreover, the EAPOL header and the Ethernet header are defined in eapol.h. The eapol.cpp is responsible for starting the EAPOL process, performing necessary PAE state actions, transiting to proper states, decoding the EAPOL packets, and transmitting EAPOL frames. Furthermore, os_frame_funcs.h comprises get_frame(), send_frame(), more_frames(), and so on. In os_generic.cpp, get_frame() employs pcap_dispatch() to capture EAP frames. The send_frame() employs libnet_write_link() to send EAP frames.

EAP and Authentication Mechanisms

This section introduces EAP. In addition, we also use the message exchanges depicted in Fig. 4 to demonstrate a typical authentication procedure of WIRE1x. Additionally, we provide a description and comparison of several authentication mechanisms implemented in WIRE1x.

Overview of EAP

EAP was originally proposed for the Point-to-Point Protocol (PPP, IETF RFC 1661) for an optional authentication phase after the PPP link has been established. It is also a general-purpose authentication protocol. EAP supports multiple authentication methods, such as token card, Kerberos (IETF RFC 1510), one-time password, certificate, public key authentication, and smart card. Figure 5 shows that there can be many different authentication mechanisms in the *Authentication Layer*. The authentication mechanisms are based upon EAP. Any new authentication mechanisms can be added easily. WIRE1x is expected to be versatile in authentication mechanisms. It has been implemented to support the most common authentication methods, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP.

When using EAP, it is not necessary to prenegotiate a particular authentication mechanism at the *Link Control Phase*. Instead, the authenticator usually sends an initial *Identity Request* followed by one or more Requests to authenticate the

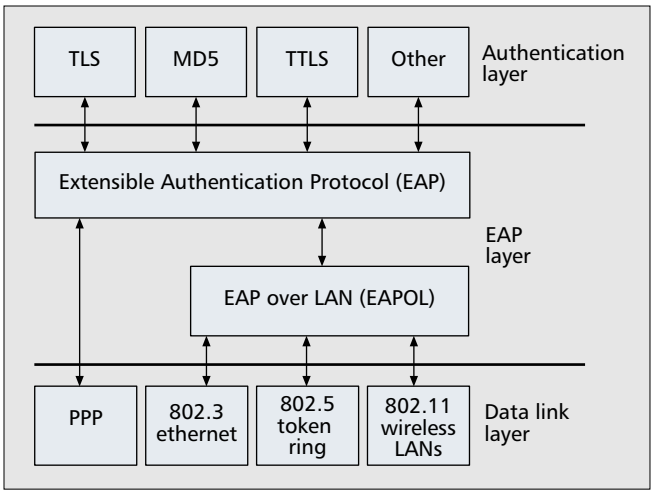

FIGURE 5. *EAP and associated layers.*

supplicant. A Request contains a *type field* to indicate what information is being requested. The *MD5-challenge* is one example of the type field. The supplicant then replies a *Response* for each Request. The Response also contains a type field according to the type field in the Request. Based on the specific authentication mechanism, a series of Requests and Responses will be exchanged. The authenticator then either sends an authentication Success or Failure to the supplicant.

Next, we present a typical EAP authentication procedure of WIRE1x in a WLAN environment according to Fig. 4.

1. User opens and selects the device to be authenticated by pcap_findalldevs(). Supplicant starts to associate with authenticator. Both supplicant and authenticator will then transition to the CONNECTING state.
2. Supplicant sends an EAPOL-start frame by libnet_write_link() to the authenticator in order to initialize the authentication process.
3. When the authenticator receives EAPOL-Start, it will reply EAP-Request/Identity to obtain the supplicant's identity. When the supplicant captures the EAP frame by pcap_dispatch(), the EAP frame is parsed by eap_decode_packet() and eapol_decode_packet() located in eap.cpp and eapol.cpp, respectively. Moreover, according to the result determined by eap_decode_packet() and eapol_decode_packet(), the supplicant PAE state machine transits to ACQUIRED state if the request is received successfully.
4. The supplicant sends back EAP-Response/Identity, containing supplicant's identity, to the authenticator. Subsequently, the authenticator and authentication server will perform necessary message exchanges according to the authentication mechanism. Note that the initial identity exchange is transmitted in cleartext. Therefore, the identity exchange is optional in some EAP types. In PEAP, for instance, the supplicant may put a *routing realm* instead of its real identity in the EAP-Response/Identity. The routing realm will route the EAP messages to the authentication server. The real identity of the supplicant can be established at a later phase [6].

5 Let us take MD5 as an example. When the supplicant receives EAP-Request/Auth, which contains RADIUS-Access-Challenge, the supplicant PAE state machine transits to the AUTHENTICATING state. The supplicant replies an EAP-Response/Auth to the authenticator in which the RADIUS-Access-Request is encapsulated.

6 Based on the result of the authentication method, the RADIUS server decides whether or not to authorize the user. If the user is authorized, the supplicant captures the EAP-Success and the supplicant PAE state machine transits to the AUTHENTICATED state. Otherwise, the supplicant captures the EAP-Failure and transits to the HELD state.

EAP-MD5

MD5 is primarily based on a *one-way hash function*. Essentially, a hash function is a cryptographic checksum. A one-way hash function takes an arbitrarily long input message and produces a fixed-length, pseudorandom output called a hash. With a hash, it is computationally difficult to find the message that produced that hash. In addition, it is almost impossible and difficult to find different messages that will generate the same hash.

MD5 takes an input message of arbitrary length and produces an output of a 128-bit *fingerprint* or *message digest*. When using MD5, an authentication server can authenticate a user without storing the user's password in cleartext. When an account is created and a user types in his/her password, the authentication server stores the hash generated by a one-way hash function which has the password as the input message. When the user wants to login to the system later, the supplicant computes the hash with the password the user enters now as the input of the same one-way hash function. The hash is transmitted over the network. As mentioned above, even if one knows the hash, it is computationally difficult to derive the original password that produced the hash. If the hash received is same as the one stored in the authentication server, the user is authenticated. Because the password is not stored in cleartext, the user password will not be disclosed, even when the password file is revealed.

The EAP-MD5 is one of the most popular EAP types because it is easy to use. A user simply types in username. A *Challenge/Response* is then followed to authenticate the user. The authentication server asks for the password by sending *RADIUS-Access-Challenge*, as shown in Fig. 4. The password hash is then sent using *EAP-Response/Auth*, which is further encapsulated by *RADIUS-Access-Request*. This is a simple and reasonable choice for wired LANs in which there is low risk for attackers to intercept the transmission. In wireless LANs, however, attackers can easily sniff a station's identity and password hash. Therefore, MD5 is more vulnerable than other authentication methods. One such attack is *replay attack*. By using replay attack, an attacker can pretend to be an authorized user in order to access a network even when the password is encrypted. For example, an attacker could simply intercept and replay (i.e., resend) a station's identity and password hash to be authenticated.

In WIRE1x, the MD5 algorithm described in IETF RFC 1321 is implemented in md5.h and md5.cpp. We ported the MD5 algorithm from Open1x and then implemented a graphical user interface (GUI) for MSWindows. A user only needs to type in username and password, and then select a proper network interface to be authenticated.

EAP-TLS

The EAP-TLS (IETF RFC 2716) is based on TLS (IETF RFC 2246) to provide protected cipher-suite negotiation, mutual authentication, and key management. After the EAP-TLS negotiation is completed, the two end-points can securely communicate within the encrypted TLS tunnel. Therefore, user's identity and password will not be revealed. Because TLS provides a way to use certificates for both user and server to authenticate each other, a user, in addition to being authenticated, can also authenticate the network. Therefore, forged APs could be detected. Both supplicant and authentication server need to have valid certificates when using EAP-TLS.

Figure 6 illustrates the authentication process and message exchanges of EAP-TLS in a WLAN [7]. After the authenticator receives the supplicant's identity in EAP-Response/Identi-

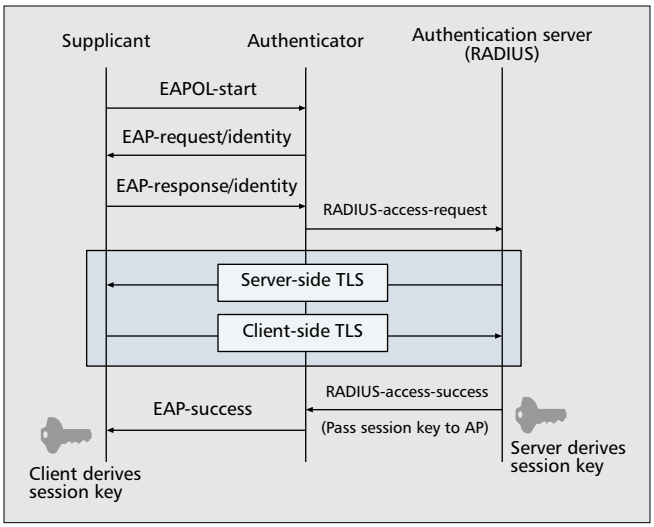

FIGURE 6. *Message flow of EAP-TLS.*

ty (flow 3), it initiates a RADIUS-Access-Request, which also carries the supplicant's identity, to the authentication server. The authentication server then provides its certificate to the supplicant and requests the supplicant's certificate. The supplicant validates the server's certificate and responds an with EAP-Response which contains the supplicant's certificate. The supplicant also initiates the negotiation for cryptographic material. After the supplicant's certificate is validated, the server responds the cryptographic material for the session. The session keys derived at both ends can be used for data encryption. Because both the supplicant and authentication servers need to have valid certificates when using EAP-TLS, to some extent EAP-TLS is difficult to manage.

In WIRE1x, EAP-TLS is implemented in tls_funcs.h and tls_funcs.cpp. WIRE1x supports the server certificate in privacy-enhanced electronic mail (PEM) [8] format. The client certificate can be in either distinguished encoding rules (DER) [9] format or basic encoding rules (BER) [9] format. The GUI of EAP-TLS includes a *select* button which is responsible for choosing the certificate in the supplicant. We use several functions from Windows' software development toolkit (SDK) in mfcDlg.cpp, as described below:

1. CertOpenSystemStore() for opening a system certificate store
2. CryptUIDlgSelectCertificateFromStore() for selecting a new certificate when using the GUI
3. CertGetNameString() for finding and printing the name of the subject of the retrieved certificate
4. CertOpenStore() for opening the certificate store to be searched
5. CertCloseStore() for closing the system certificate store

To use these functions, the system header files of wincrypt.h and cryptuiapi.h must be included. Additionally, the libraries of crypt32.lib, advapi32.lib, and cryptui.lib must be linked. While we use these functions to obtain the certificate structure, we must replace SSL_CTX_use_certificate_file() by SSL_CTX_use_certificate_ASN1() in eapcrypt.cpp in order to receive the structure provided by Windows SDK functions.

EAP-TTLS

The EAP-TTLS extends EAP-TLS to exchange additional information between client and server by using the secure tunnel established by TLS negotiation. A EAP-TTLS negotiation comprises two phases: the *TLS handshake phase* and the *TLS tunnel phase*. During phase one, TLS is used for the client to authenticate the server. Optionally, the server can also authenticate the client. Similarly as in EAP-TLS, the authentication is done by using certificates. A secure TLS tunnel is also established after the phase-one handshake. In phase two, the secure TLS tunnel can be used for other information exchanges, such as additional user authentication key, communication of accounting information, and so forth.

In a WLAN environment, the EAP-TTLS usually is used as follows. In phase one, TLS is used as a supplicant to authenticate the authentication server by using a certificate. Once the authentication server is authenticated, the authentication server authenticates the supplicant by using the supplicant's username and password in phase two. The username and password are carried in the attribute-value pairs (AVPs) defined by the AAA server, which usually is a RADIUS server or Diameter server. The message exchanges are protected by the TLS tunnel established in phase one. The authentication of supplicant in phase two can use any non-EAP protocols such as PPP Authentication Protocols (PAP, IETF RFC 1334), PPP Challenge Handshake Authentication Protocol (CHAP, IETF RFC 1994), Microsoft PPP CHAP Extensions (MS-CHAP, IETF RFC 2433), or Microsoft PPP CHAP Extensions, Version 2 (MS-CHAP-V2, IETF RFC 2759). Because only the authentication server needs to have a valid certificate, EAP-TTLS is more manageable than EAP-TLS.

The EAP-TTLS in WIRE1x is implemented in tls_funcs.h, tls_funcs.cpp, ttlsphase2.h, and ttlsphase2.cpp. The GUI consists of five major input objects. The username and password are used for phase-two authentication. The user also needs to choose one of the authentication protocols which can be PAP, CHAP, MS-CHAP, or MS-CHAP-V2 for phase-two authentication.

PEAP

PEAP provides an encrypted and authenticated tunnel based on TLS. Therefore, the EAP messages encapsulated inside the TLS tunnel are protected against various attacks. Similar to EAP-TTLS, PEAP also comprises two phases. In the first phase, a TLS session is negotiated and established. The client also authenticates the server by using a certificate. Optionally, the server can also authenticate the client. In the second phase, EAP messages are encrypted by using the key negotiated in phase one. The basic idea of PEAP and EAP-TTLS are identical. However, PEAP can only use EAP protocols (for example, EAP-MS-CHAP-V2[3] [10]) in the second phase, while EAP-TTLS can use EAP or non-EAP protocols (for example, PAP, CHAP, MS-CHAP, and MS-CHAP-V2).

When using PEAP in WLANs, typically, an authentication server is authenticated by a supplicant based on the server certificate. A secure TLS tunnel is also created. A supplicant is then authenticated using username and password, which are protected by the TLS tunnel.

The PEAP in WIRE1x is implemented in tls_funcs.h, tls_funcs.cpp, eapmschapv2.h, eapmschapv2.cpp, peapphase2.h, and peapphase2.cpp. The GUI of PEAP consists of three major input objects. A user must type in username and password for phase-two authentication. Currently, only EAP-MS-CHAP-V2 is supported. A user also needs to select a proper network interface to be authenticated.

To conclude this section, a comparison of the authentication mechanisms discussed in this section is provided in Table 1.

Open-Source Libraries

This section describes the open-source libraries used in WIRE1x. WinPcap and Libnet are used to capture/write packets from/to data link layer. OpenSSL is used for TLS-based authentication methods.

[3] *EAP-MS-CHAP-V2 encapsulates the MS-CHAP-V2 within EAP.*

	EAP-MD5 (RFC 1321)	EAP-TLS (RFC 2716)	EAP-TTLS (Internet draft)	PEAP (Internet draft)
Server authentication	No	Public key (certificate)	Public key (certificate)	Public key (certificate)
Supplicant authentication	Password hash	Public key (certificate or smart card)	Certificate, EAP, or non-EAP protocols	Certificate or EAP protocols
Mutual authentication	No	Yes	Yes	Yes
Dynamic key delivery	No	Yes	Yes	Yes
Basic protocol architecture	Challenge/response	Establish TLS session and validate certificates for both client and server	1. Establish TLS between client and TTLS server 2. Exchange attribute-value pairs between client and server	1. Establish TLS between client and PEAP server 2. Run EAP exchanges over TLS tunnel
Server certificate	No	Required	Required	Required
Client certificate	No	Required	Optional	Optional
Protection of user identity	No	No	Yes, protected by TLS	Yes, protected by TLS

TABLE 1. *Comparison of authentication mechanisms.*

WinPcap

WinPcap is used for packet capturing and network analysis in the Win32 platform. It includes a kernel-level packet filter, a low-level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll, which is based on libpcap version 0.6.2). It is in charge of the following tasks:

1. The pcap_findalldevs() in wire1xDlg.cpp prints the list of network interfaces. Therefore, a user can select a proper network interface to be authenticated.
2. The pcap_dispatch() in os_generic.cpp captures packets from the AP.
3. The setup pcap() in os_generic.cpp can be used to adjust parameters in filter, which can make the supplicant receive EAP frames only.
4. The pcap_close() in os_generic.cpp shuts down WinPcap.
5. The pcap_open_live() in os_generic.cpp selects promiscuous mode or nonpromiscuous mode for the network interface.

Libnet

Libnet is a generic networking API that provides access to several protocols. In WIRE1x, it is used only for libnet_write_link() in os_generic.cpp to write packets to AP.

OpenSSL

OpenSSL is an open-source toolkit which implements the Secure Sockets Layer (SSL) v2/v3 and TLS v1 protocols. It also includes a full-strength general-purpose cryptography library. The TLS-based authentication methods in WIRE1x use OpenSSL library in eapcrypt.cpp. There are many OpenSSL functions in eapcrypt.cpp. Here, we only itemize some of them:

1. The SSL_CTX_load_verify_locations() loads the server certificate in PEM format. As discussed above, all TLS-based authentication methods need a server certificate in order to authenticate the server.
2. The SSL_CTX_use_certificate_file() loads the client certificate in DER format. Alternatively, the SSL CTX use certificate ASN1() loads the client certificate in BER format.
3. The SSL_CTX_use_PrivateKey_file() loads the client private key in PEM format. This function is used by EAP-TLS only.

Summary

Supplicant software is indispensable to the IEEE 802.1x standard. We observe that the success of 802.1x greatly depends on end users. We believe that a free 802.1x supplicant software, which works with various versions of MS Windows and supports most of authentication mechanisms in EAP, will boost the deployment of 802.1x, and thus 802.11i. Therefore, we have developed WIRE1x and hope that most users will access WLANs in a more secure manner.

We believe that WIRE1x is a good choice for people eager for 802.1x client software. It has been downloaded worldwide. Since WIRE1x was released on June 18, 2003, the WIRE1x website has been visited more than 24,000 times. There have been more than 3600 downloads of source code and 6800 downloads of executable code up until August 2005.

This article has presented all the components of WIRE1x exhaustively. In addition to providing a tutorial of 802.1x and EAP, the objective of this article is to share our experience in implementing the 802.1x supplicant. Currently, WIRE1x supports several wireless cards and provides various authentication mechanisms, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP. It is versatile, as compared to many other implementations of the 802.1x supplicant. We believe open source is essential for any security-related software because it can be examined as one wishes. Based on this article, readers should be able to comprehend the source code of WIRE1x easily.

Acknowledgments

We thank other members of the Wireless Internet Research & Engineering (WIRE) Lab, especially Yi-Wen Liu, Chin-Hsing Lin, Wen-Ting Wu, and Jui-Hung Yeh, for their help in the development of WIRE1x. We also thank the Computer and Communication Center, National Tsing Hua University, and the Computer Center, Department of Computer Science, National Tsing Hua University, for their support. This work was sponsored in part by National Science Council (NSC) under grant nos. 94-2752- E-007-003-PAE, 94-2213-E-007-073, 93-2219-E-007-002, and 93-2213-E-007-004, and Industrial Technology Research Institute (ITRI) under contract no. T1-94081-13.

References

[1] ANSI/IEEE Std 802.11, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications," 1999.
[2] IEEE Std 802.11i, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements," July 2004.
[3] J.-C. Chen, M.-C. Jiang, and Y.-W. Liu, "Wireless LAN Security and IEEE 802.11i," *IEEE Wireless Commun.*, vol. 12, 2005, pp. 27–36.
[4] IEEE Std 802.1X-2001, "IEEE Standard for Local and Metropolitan Area Networks, Port-based Network Access Control," Oct. 2001.
[5] P. Funk and S. Blake-Wilson, "EAP Tunneled TLS Authentication Protocol Version 1 (EAP-TTLSv1)," IETF Internet Draft (work in progress), Feb. 2005, available at draft-funk-eap-ttls-v1-00.txt
[6] A. Palekar *et al.*, "Protected EAP Protocol (PEAP), Version 2," IETF Internet Draft (work in progress), Oct. 2004, available at draft-josefsson-pppext-eap-tls-eap-10.txt
[7] Cisco Systems, "White paper: EAP-TLS Deployment Guide for Wireless LAN Networks," 2004, available at http://www.cisco.com/
[8] B. Kaliski, "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services," IETF RFC 1424, Feb. 1993.
[9] ITU-T Rec. X.690, "ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)," July 2002.
[10] V. Kamath and A. Palekar, "Microsoft EAP CHAP Extensions," IETF Internet Draft (work in progress), Apr. 2004, available at draft-kamath-pppext-eap-mschapv2-01.txt

Biographies

JYH-CHENG CHEN [SM] (jcchen@cs.nthu.edu.tw) is an associate professor in the Department of Computer Science and the Institute of Communications Engineering, National Tsing Hua University, Hsinchu, Taiwan. Prior to joining National Tsing Hua University as an assistant professor, he was a research scientist at Bellcore/Telcordia Technologies, Morristown, New Jersey, from August 1998 to August 2001. He received his Ph.D. degree from the State University of New York at Buffalo in 1998. He is a coauthor of the book *IP-Based Next-Generation Wireless Networks* (Wiley, 2004).

YU-PING WANG (ichiro@wire.cs.nthu.edu.tw) received his B.B.A. degree in information management from Shih Hsin University, Taipei, Taiwan, in 2002, and his M.S. degree in communications engineering from National Tsing Hua University in 2004. He is now with CyberTAN Technology, Hsinchu, Taiwan.