NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

EVALUATION OF SECURE 802.1X PORT-BASED NETWORK ACCESS AUTHENTICATION OVER 802.11 WIRELESS LOCAL AREA NETWORKS

by

Huseyin Selcuk Ozturk

March 2003

Thesis Advisor: Geoffrey Xie
Second Reader: John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2003	Master's Thesis

4. TITLE AND SUBTITLE Evaluation of Secure 802.1X Port-Based Network Access Authentication over 802.11 wireless Local Area Networks	5. FUNDING NUMBERS
6. AUTHOR (S) Huseyin Selcuk OZTURK	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the U.S. Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited.	A

13. **ABSTRACT** Since wireless technology has been used in Local Area Networks (LAN), our networks are easier to build and are more scalable and mobile than legacy structures. While providing these functionalities, Wireless LAN (WLAN)'s have some security vulnerabilities that should be addressed. Failing to examine the security risks of WLAN technology and take the necessary countermeasures may result in unauthorized entry into the legacy local area networks and other attacks.

A secure connection to an intranet, which holds critical data and applications, must be the utmost consideration in the effort to protect critical resources. This thesis builds an open-source test-bed for evaluating WLAN security protocols. Moreover, it investigates the suitability of the IEEE 802.1X standard to provide the required security framework to WLANs. This research determines that the IEEE 802.1X could enhance the security level in authentication and privacy by the enabling rekeying process, but would not prevent Denial of Service attacks via unauthenticated management frames.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Wireless Local Area Networks, Security, User Authentication Base, Threat Model for Critical Infrastructures, Organizational Security Policy for Wireless LAN'.	191
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EVALUATION OF SECURE 802.1X PORT-BASED NETWORK ACCESS AUTHENTICATION OVER 802.11 WIRELESS LOCAL AREA NETWORKS

Huseyin Selcuk Ozturk
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author: Huseyin Selcuk Ozturk

Approved by: Geoffrey Xie, PhD
Thesis Advisor

John Gibson
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Since wireless technology has been used in Local Area Networks (LAN), our networks are easier to build and are more scalable and mobile than legacy structures. While providing these functionalities, Wireless LAN (WLAN)'s have some security vulnerabilities that should be addressed. Failing to examine the security risks of WLAN technology and to take the necessary countermeasures may result in unauthorized entry into the legacy local area networks and other attacks.

A secure connection to an intranet, which holds critical data and applications, must be the utmost consideration in the effort to protect critical resources. This thesis builds an open-source test-bed for evaluating WLAN security protocols. Moreover, it investigates the suitability of the IEEE 802.1X standard to provide the required security framework to WLANs. This research determines that the IEEE 802.1X could enhance the security level in authentication and privacy by enabling the rekeying process but would not prevent Denial of Service attacks via unauthenticated management frames.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Geoffrey Xie, for his guidance, encouragement, and support throughout my research. I would also like to thank my second reader, John Gibson, for his sound advice and guidance throughout the writing of this thesis. Finally, I would like to thank to my family and great friends for supporting me through my life and giving me encouragement to take on new challenges.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Since the inception of Wireless Local Area Networks (WLAN) in communication domains, they have provided mobility and have improved the ease of network installation in most areas, such as industry, science, education, medicine and the military.

The military has been familiar with wireless communications since the invention of radio. The use of wireless technology in local area networks requires an examination of this new technology in every aspect. Performance, compatibility, and cost are often the first aspects considered, but the security is the most crucial requirement if WLANs are employed in critical infrastructures like military applications.

When IEEE first released the IEEE 802.11 standards in 1997 [3], the implementation of Wired Equivalence Privacy (WEP), which is the security functionality of the standard, was optional for the vendors. As WLANs find more and more use in daily life, new security flaws continue to be discovered. When these flaws are exploited as threats to the WLANs, new security frameworks must be created and the solutions incorporated in the standards.

Wired Equivalence Privacy (WEP) was accepted as a security standard by IEEE because the IEEE 802.11 standard stipulated that security be exportable, reasonably strong, self-synchronizing, and computationally efficient. At the time, WEP was believed to satisfy all these requirements. [5]

It has been demonstrated that WEP is weak and can be broken in a matter of hours [5], [6]. Yet there has been no change to the standard, even though WEP is vulnerable to malicious intent. The IEEE 802.11i workgroup responsible for enhancing the security standards of 802.11 networks is expected to finalize its work by late 2003. The 802.11i security framework is intended to address all the security vulnerabilities inherent in the current standard.

The IEEE 802.1X standard [8], by providing specifications for port-based network access control for IEEE 802 LAN structures, could fill the gap before the IEEE 802.11i standards come to fruition.

However, Arunesh Mishra and William A. Arbaugh, researchers from the University of Maryland, indicated that the IEEE 802.1X standard is also vulnerable to man-in-the-middle and session-hijacking attacks in their technical report, "An Initial Security Analysis of the IEEE 802.1X Standard." [14]

B. THESIS OBJECTIVES

The main objective of this thesis is to build an open-source wireless test-bed for evaluating the IEEE 802.11 family of security protocols. The attacks on 802.1X as suggested in [14], are tested and evaluated with the test-bed to demonstrate its usefulness.

This thesis will try to determine whether the IEEE 802.1X standard is capable of providing secure client authentication to a network that is a part of a critical infrastructure, such as naval headquarters, a supply branch office, or a naval ship.

An open-source test-bed, implementing the IEEE 802.1X standard will be used to demonstrate the facets of the standard. Several client entities will be tested with respect to the threat model and the protocol packet types will be examined while performing secure client authentication and key exchange.

C. THESIS ORGANIZATION

This thesis is organized into six chapters. Chapter II provides an overview of the IEEE 802.11 standard as it applies to the Physical and MAC Layers of LAN and the security mechanisms and vulnerabilities of current IEEE 802.11 WLANs. Chapter III examines, the IEEE 802.1X standard and its security enhancements. Chapter IV explains WLAN entities and the structure of the test-bed. It then presents the experimental results. Chapter V contains a discussion of secure mobile client authentication. Finally, Chapter VI presents conclusions drawn from the research and suggests areas for future work.

II. WIRELESS LOCAL AREA NETWORKS OVERVIEW

A. INTRODUCTION

This chapter gives an overview of the IEEE 802.11 physical and medium access layer specification. In addition to a discussion of the specification, specific security mechanisms and security problems of the IEEE 802.11 WLANs are identified. Finally, a threat model for WLANs is presented at the end of the chapter.

B. HISTORY OF WIRELESS NETWORKS

Network technologies and radio communication were fused for the first time in 1971 at the University of Hawaii, as a research project called ALOHANET. The ALOHANET offered bidirectional communications in a logical star topology, between the central computer and each of the remote stations. The remote stations had to communicate with one another via the centralized computer.

In the 1980's amateur radio hobbyist kept radio networking alive within the United States and Canada by designing and building terminal node controllers [1]. In 1985, the Federal Communication Commission (FCC) made the commercial development of radio-based LAN components possible by authorizing the public use of the Industrial, Scientific, and Medical (ISM) bands between 902 MHz and 5.85 GHz.

In late 1980s, the IEEE 802 Working Group, responsible for developing LAN standards, such as Ethernet and token ring, began developing standards for wireless LANs. This group developed the Wireless LAN Medium Access Control and Physical Layer specification. The IEEE Standards Board approved the standard on June 26, 1997, and the IEEE published the standard on November 18, 1997. The final version of this standard prompted vendors to release 802.11-compliant radio cards and access points in 1998. [1]

C. OVERVIEW OF IEEE 802.11 WIRELESS LAN MEDIUM ACCESS CONTROL (MAC) AND PHYSICAL LAYER (PHY) SPECIFICATION

Figure 1 illustrates the scope of the IEEE 802.11 Standard corresponding to the OSI stack.

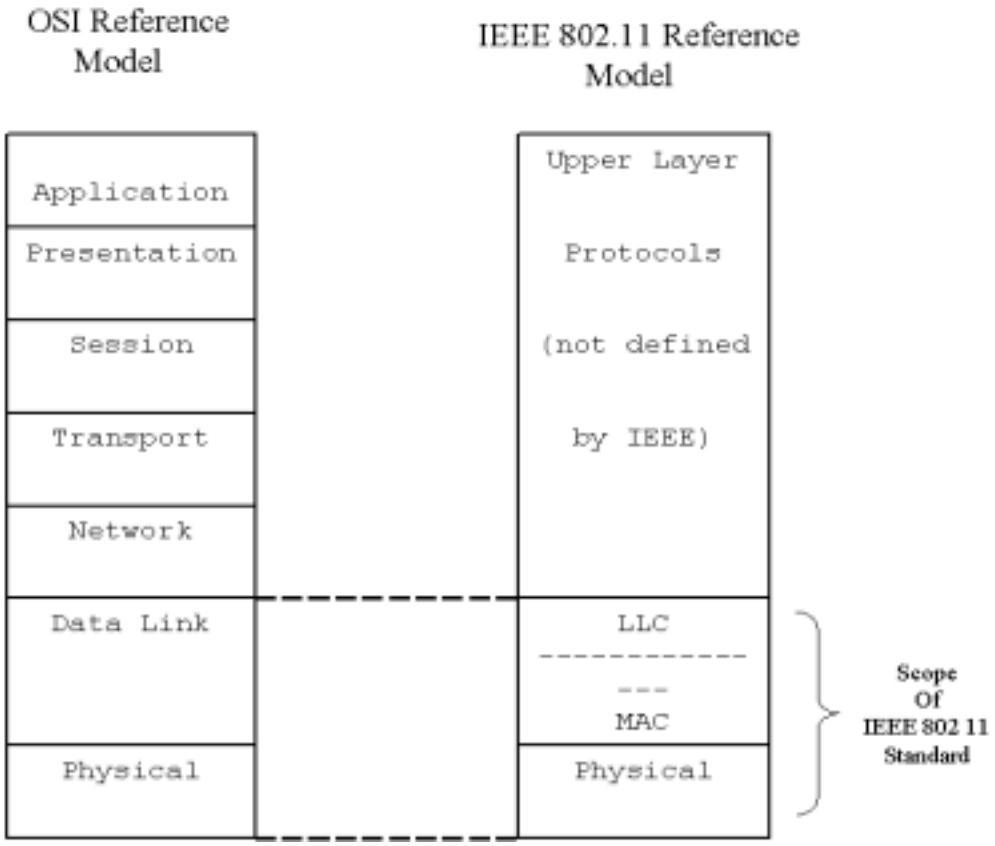


Figure 1. IEEE 802 Reference Model

1. IEEE 802.11 Physical Layer (PHY)

The logical architecture of the 802.11 standard is applied to each station and consists of a single MAC and one of the multiple PHY layers as shown in Figure 2.

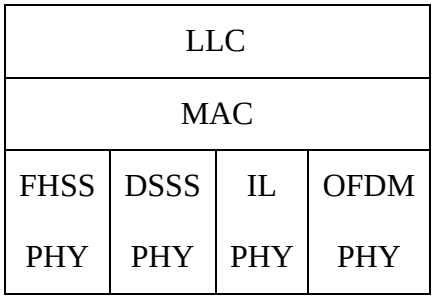


Figure 2. 802.11 Logical Architecture

a. Spread Spectrum Modulation

The IEEE 802.11 Physical Layer uses Radio Frequency (RF) energy to transmit or to receive data through the air. Spread-spectrum modulation is a physical layer function that spreads the digital signal that a network interface card (NIC) transmits. This spreading process makes the data signal much less susceptible to electrical noise. Two methods are used to spread the signal: frequency hopping or direct sequence.

(1) Frequency Hopping Spread Spectrum (FHSS): In a frequency hopping spread spectrum, a network interface card modulates the data signal with a carrier signal and then hops from frequency to frequency as a function of time over a wide frequency band as illustrated in Figure 3.

Both the receiver and the sender should synchronize before transmission according to a hopping code. The hopping code determines the order of frequencies over which the radio transmits. The time spent at a particular frequency during any single hop is called the dwell time. According to the standard, seventy five or more frequency per transmission channels with a maximum dwell time of 400ms should be used to implement FHSS systems.

If interference is encountered during the transmission, the radio will transmit the signal during the subsequent hop. Since faster data rates are susceptible

to an overwhelming number of errors, frequency hopping can achieve a maximum limit of 2 Mbps data rates. [1]

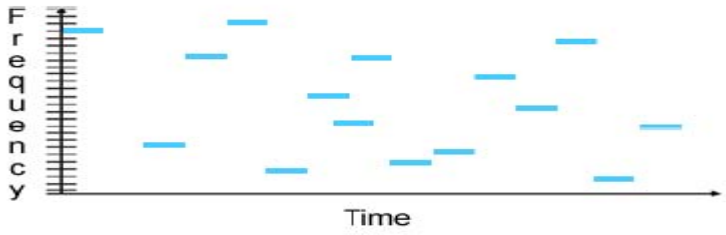


Figure 3. FHSS Example

(2) Direct Sequence Spread Spectrum (DSSS): In direct sequence spread spectrum, a data signal and a higher data bit rate are sequence combined at the sending station. This high bit sequence is referred to as the chipping code. Because the lower bit rate information signal is carried by the higher bit rate composite signal, the number of chips corresponding to each information bit provides redundancy or a processing gain. Figure 4 illustrates the overlapping spread spectrum channels. A high processing gain provides increased resistance to interference. A minimum processing gain of eleven is enforced by the 802.11 workgroup. Direct Sequence Spread Spectrum can provide higher data rates than Frequency Hopping Spread Spectrum.

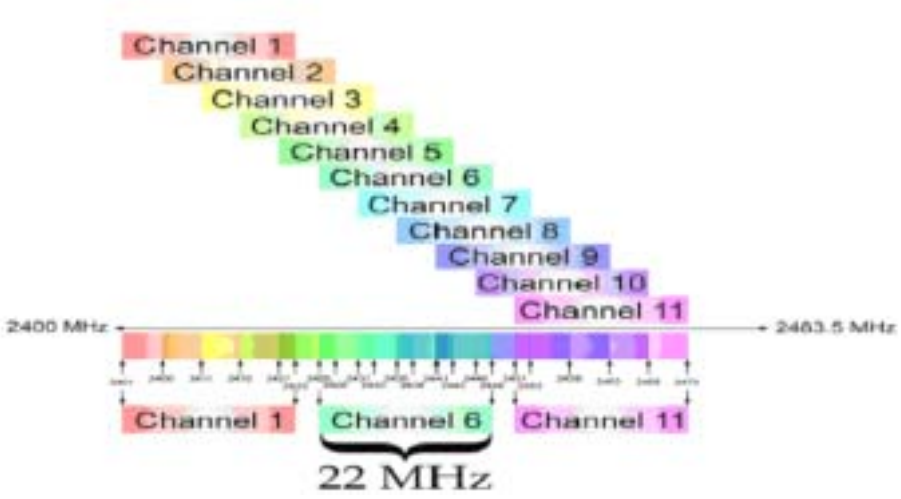


Figure 4. DSSS Example

b. Orthogonal Frequency Division Multiplexing (OFDM)

Orthogonal Frequency Division Multiplexing (OFDM) is a kind of Frequency Division Multiplexing (FDM) that divides bandwidth into sub-carriers. Since FDM requires a guard band between adjacent carriers to prevent interference, OFDM improves on frequency division multiplexing by making the adjacent sub-carriers to each orthogonal. This eliminates the need for protective guard bands between sub-carriers. This results in more efficient frequency bandwidth usage. Quadrature Phase Shift Key (QPSK), 16 Quadrature Amplitude Modulation (16QAM), 64 Quadrature Amplitude Modulation (64QAM), and Binary Phase Shift Keying (BPSK) are modulation schemes used to support the various required data rates.

c. Infrared Light (IL)

The IEEE 802.11 infrared scheme is omnidirectional rather than point-to-point. A maximum range of 20 mile is possible. The modulation scheme for the 1 Mbps data rate is known as 16-PPM (pulse-position modulation). In this scheme, each group of 4 data bits is mapped into one of the 16-PPM symbols. Each symbol is a string of 16 bits consisting of fifteen 0s and one binary 1. The position of the single 1 bit among the fifteen 0 bits defines the pulse position. For the 2-Mbs data rate, each group of 2 data bits is mapped into one of four 4-bits sequences. Each sequence consists of three 0s and one binary 1. The actual transmission uses an intensity modulation scheme in which the presence of a signal corresponds to binary 1 and the absence of a signal corresponds to binary 0 [9].

2. IEEE 802.11 Medium Access Layer (MAC)

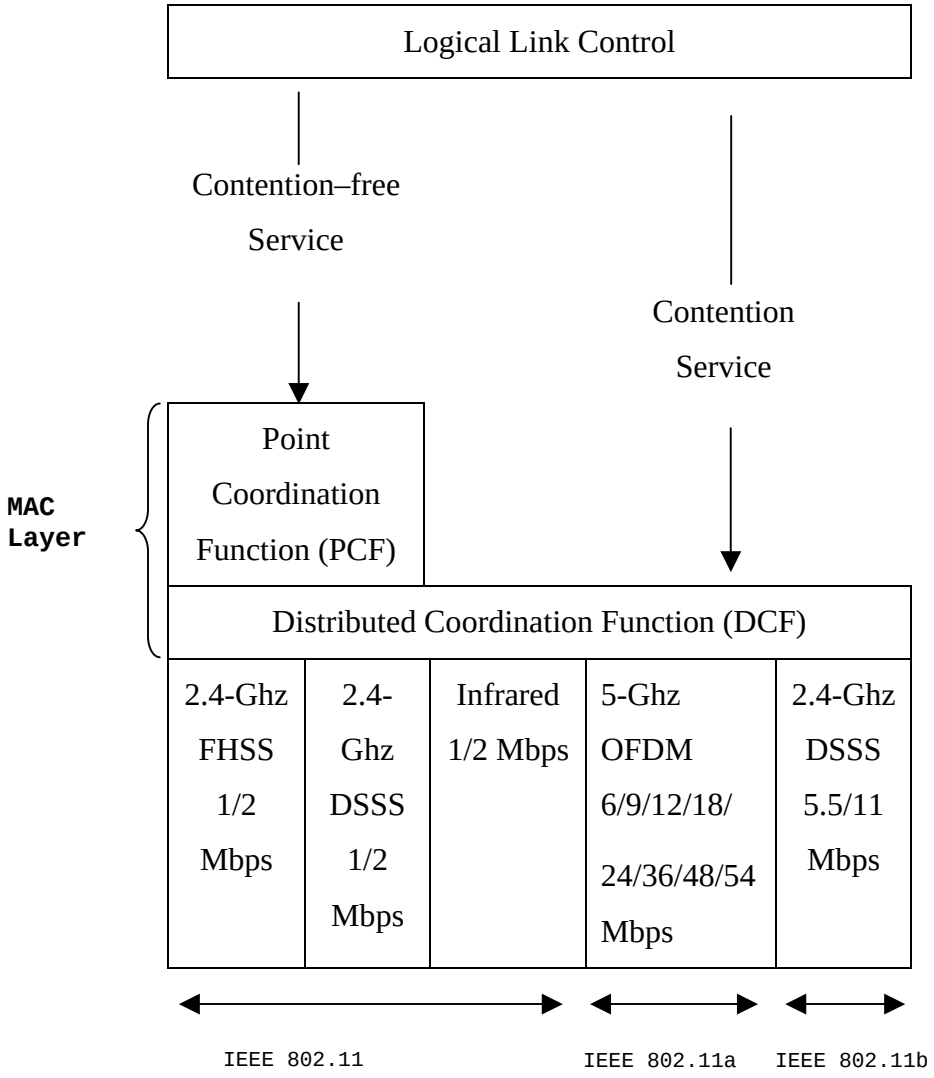


Figure 5. IEEE 802.11 Mac Layer Specification

The IEEE Mac Layer provides three main functions supporting connectivity: data delivery, access control, and security. The role of the MAC layer in the overall 802.11 architecture is illustrated in Figure 5.

a. Reliable Data Delivery

Since an 802.11 WLAN uses air as its transmission medium, noise, interference, and other propagation effects cause the loss of a significant number of

frames. The 802.11 standards have a frame-exchange mechanism to prevent frame loss between stations. When a station receives a data frame from another station, it acknowledges that data frame. This exchange should be an atomic unit that cannot be interrupted by another communication. The receiving station should acknowledge the received frame immediately. If the sender does not receive the acknowledgement (ACK) within the specified time, the sender resends the message. Prior to the exchange of the data and ACK frames, a Request-to-Send (RTS) message, which notifies other stations not to transmit, thus avoiding collisions, is sent to the desired destination. The destination responds with a Clear-to-Send (CTS) message. This notifies other stations beyond the range of the RTS originator that a transmission will occur. The RTS/CTS messages decrease the available network load. This is why the RTS/CTS messages could be disabled to increase the network performance.

b. Access Control

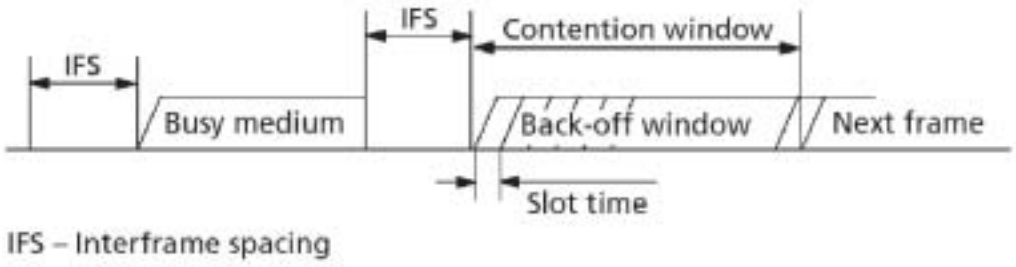
The 802.11 MAC Layer must first gain access to the network by using a distributed coordination function (DCF) or by using the point coordination function (PCF). The general schema is defined in Figure 5.

(1) Distributed Coordination Function (DCF): DCF is the primary protocol that stations and access points must implement in order to share the wireless medium. The DCF sub-layer uses a Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) algorithm. [3]

If a station wants to transmit, first it determines whether the medium is in use. If the medium is idle, then the station starts to transmit; otherwise the station should wait until the current transmission is completed. Since the wireless media is incapable of distinguishing incoming weak signals from noise and the other effects of transmission, the 802.11 standards use delays to avoid collisions. DCF specifies a set of delays known as interframe spaces (IFS):

- **SIFS** (Short IFS): The shortest of the IFS are used for all immediate response actions.

- **PIFS** (Point Coordination Function IFS): The medium-length duration IFS are used by the centralized controller in the PCF scheme when issuing polls.
- **DIFS** (Distributed Coordination Function IFS): The longest IFS is used as a minimum delay for asynchronous frames contending for access.



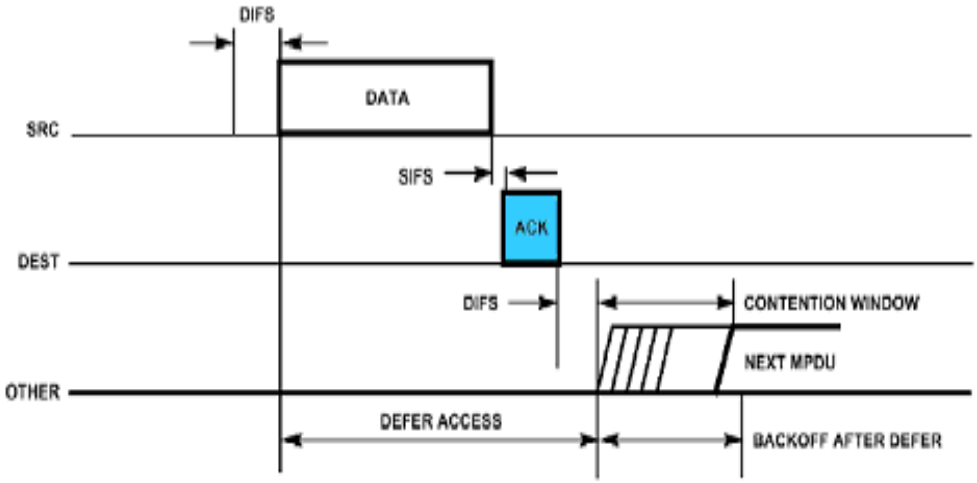


Figure 6. IEEE 802.11 Medium Access Control Logic

As shown in Figure 6, a station first checks the medium to determine whether it is idle or busy before it attempts to transmit a frame. If the medium is idle, it waits to see if the medium remains idle for the IFS time, and if so, the station transmits the frame.

If the medium is busy, the station monitors the medium until the current transmission is over. Once the current transmission is over, the station waits for an additional interframe space. If the medium is idle through the IFS period, then the

station waits for a random period, referred to as an exponential backoff time. If the medium is still idle at the end of the backoff time, then the station transmits the frame.

(2) Point Coordination Function (PCF): The Point Coordination Function is an optional access method that can be implemented on top of the DCF mostly for time sensitive transmissions. PCF uses a centralized, contention-free polling access method. The Point Coordinator, an access point (AP) software functionality, performs polling among attached stations. Current products have not yet implemented PCF.

c. Security

The 802.11 MAC Layer provides authentication and privacy. Since the signals are broadcast through the open air, the standard implementers should provide some level of security to prevent unauthorized users access to the network through which they might forge or steal information. The standard provides Open-system Authentication and Shared-key Authentication methods to enforce authentication and Wired Equivalent Privacy (WEP) to enforce privacy.

(1) Open-System Authentication (OSA): In OSA, both the station and the authenticator agree on exchanging data. The stations must be configured to use the same Service Set Identifier (SSID) in order to authenticate. First, the initiating station sends a MAC authentication frame to the authenticator station. This authentication frame states that it is an open-system authentication type. The authenticator responds to this frame with an authentication frame that indicates whether the authentication frame was accepted or rejected.

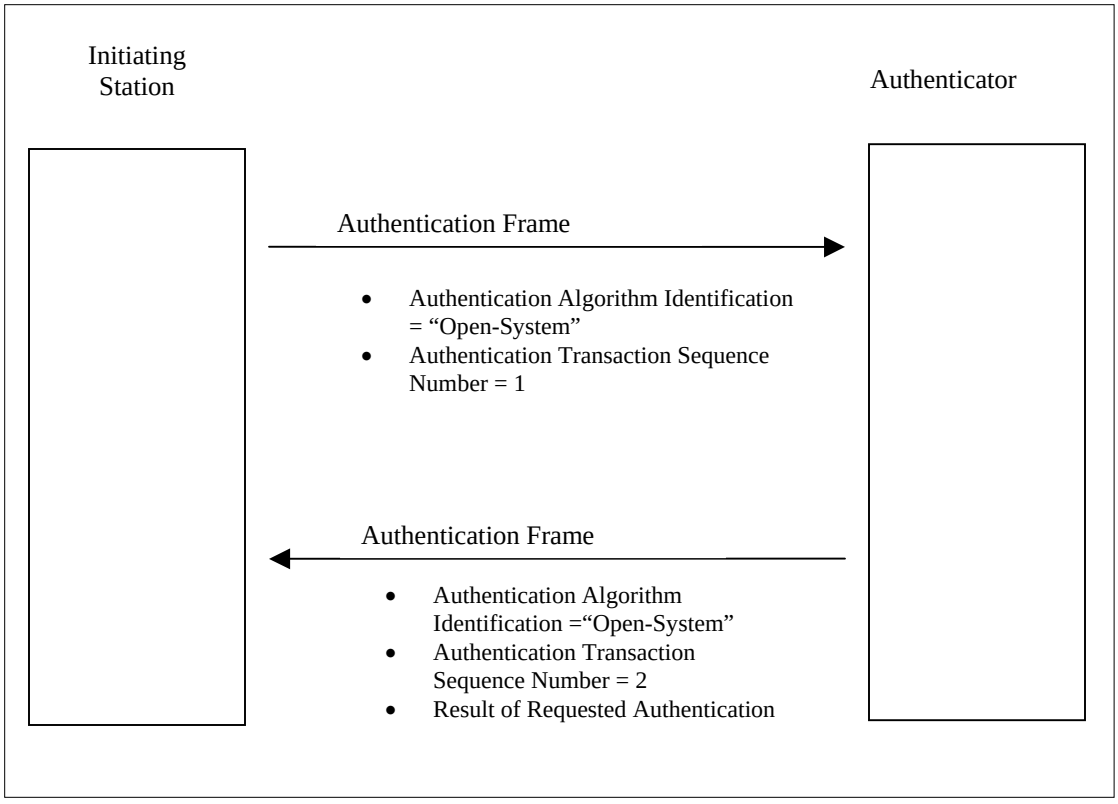


Figure 7. Open-System Authentication

(2) Shared-key Authentication: Shared-Key Authentication requires a shared key, to authenticate the station and authenticator to each other. In order to ensure privacy of the shared key, Wired Equivalent Privacy, as discussed below, should be supported by both parties. The shared-key authentication process is as follows:

- A requesting station sends an authentication frame which defines the authentication algorithm identification as "Shared Key."
- When an authenticator station receives an authentication request frame, it responds with authentication frame that contains 128 octets of challenge text that the WEP service generates.
- When the requesting station gets the authentication challenge, it encrypts the challenge with the shared-key and sends it back to the authenticator station with an authentication frame.

- When the authenticator station receives the authentication message with the encrypted challenge, it decrypts the encrypted challenge with the same shared key. Then decrypted challenge is compared with the challenge text that was sent earlier. If they are the same, the authenticator sends a successful authentication notice; otherwise, a negative authentication frame is sent.

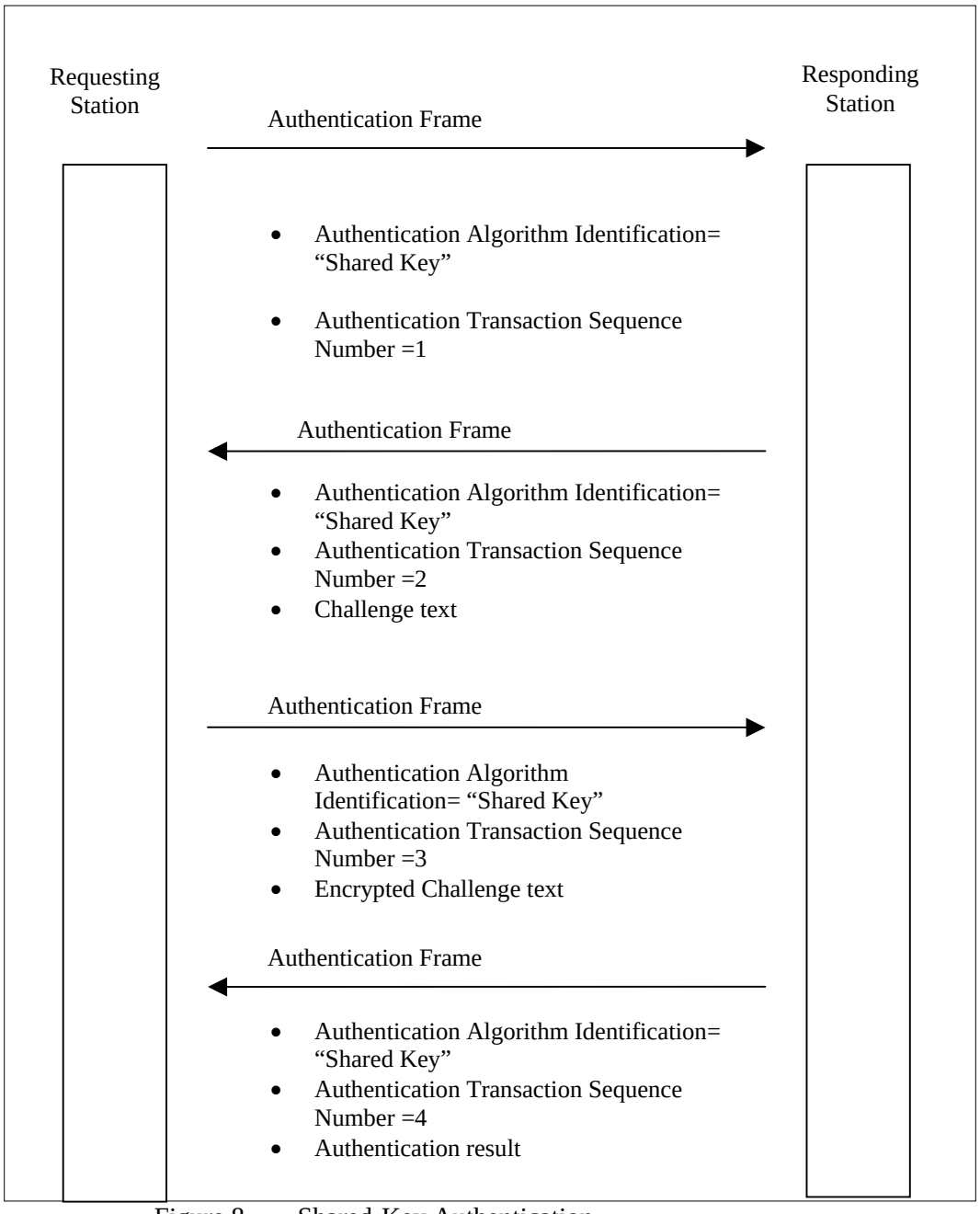


Figure 8. Shared-Key Authentication

(3) Privacy: The IEEE 802.11 standard provides Wired Equivalence Privacy (WEP) to enforce privacy and to provide modest level security and data integrity. The functionality and security vulnerabilities of WEP are discussed later in this chapter.

3. MAC Frame Format

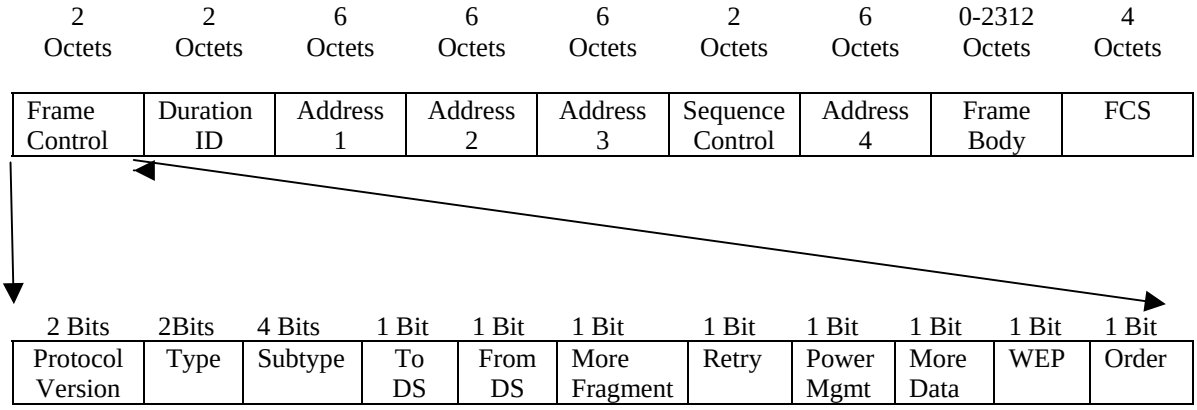


Figure 9. 802.11 MAC Format

- **Frame Control:** This field of the frame contains control information within the frame and determines the type of the frame.

  Protocol Version: Specifies the version of the protocol, which is currently 1.0.

  Type: Identifies whether the frame is a data, control or management frame.

  Subtype: Identifies the function of the frame, which is specified at the type field, as illustrated in at the Figure 9.

  *To Distribution System (To DS):..*Set to "1" if the frame is destined to the distribution system.

From Distribution System (From DS): Set to "1" if the frame is leaving the distribution system.

More Fragment: Set to "1" if there is another fragment of the same LLC data unit (also called an MSDU) following in a subsequent frame.

Retry: Set to "1" if the frame is a retransmission of an earlier frame.

Power Management: Set to "1" if the sending station is in the sleeping mode.

More Data: Set to "1" if the sending station has additional data to send. This data could be sent as one frame or as a group of fragments in multiple frames.

Wired Equivalence Privacy (WEP): Set to '1' if the frame body is processed by the WEP algorithm.

Order: Set to "1" for any data frame sent using the Strictly Order service, which tells the receiving station to process the frames in order. [9]

- **Duration ID:** Each frame contains information that indicates the duration of the next frame transmission (in microseconds). In some control frames, this field contains an association or connection identifier.
- **Address1, 2, 3, 4:** The address spaces contains the basic service set identification (BSSID) source, destination, transmitting station, and receiving station, depending on the type of frame being sent. The addresses are 48 bits in length and can be either individual or group addresses (broadcast or multicast).
- **Sequence Control:** The four leftmost bits contain the Fragment Number Subfield, which is used for fragmentation and reassembly. The next 12 bits of this frame are the sequence number starting form zero and incrementing by one for each frame sent between a given transmitter and receiver.

- **Frame Body:** Contains a variable length field, depending upon the frame type. If the frame is a data frame, this field contains MSDU. If the frame is a control or management frame, this field can contain some specific parameters related to the respective type.
- **Frame Check Sequence (FCS):** The 32-bits frame check sequence is used for error detection and is calculated by using a Cyclic Redundancy Check algorithm at the MAC layer.

4. MAC Frame Types

Type Value	Type Description	Subtype Value	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassociation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	1000	Beacon
00	Management	1001	Announcement TIM
00	Management	1010	Dissociation
00	Management	1011	Authentication
00	Management	1100	De-authentication
01	Control	1010	Power save-poll
01	Control	1011	Request to send
01	Control	1100	Clear to send

01	Control	1101	Acknowledgment
01	Control	1110	Contention-free (CF)-end
01	Control	1111	CF-end + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data+CF-Ack+CF-Poll
10	Data	0100	Null Function (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack+CF-poll (no data)

Table 1. MAC Frame Types

a. Management Frames

2 Octets	2 Octets	6 Octets	6 Octets	6 Octets	2 Octets	0-2312 Octets	4 Octets
Frame Control	Duration ID	Destination Address	Source Address	BSSID	Sequence Control	Frame Body	FCS

Figure 10. Management Frame Structure

	Assoc. Req.	**Assoc. Res.**	**Reass. Req.**	**Reass. Res.**	**Probe Req.**	**Probe Res.**	**Beacon**	**Disass.**	**Authen.**	**De-Authen.**
Authentication Algorithm No.									X	
Authentication Transaction Sequence No.									X	

Beacon Interval					X		X			
Current AP Address			X							
Listen Interval	X		X							
Reason Code								X		X
Association ID		X		X						
Status Code		X		X					X	
Timestamp					X		X			
Service Set ID (SSID)	X		X		X	X	X			
Supported Rates	X	X	X	X	X	X	X			
FH Parameter Set					X		X			
DS Parameter Set					X		X			
CF Parameter Set					X		X			
Capability Information	X	X	X	X	X		X			
Traffic Indication Map (TIM)							X			
IBSS Parameter Set					X		X			
Challenge Text									X	

Table 2. The Frame Body of Management Frame

(Assoc) Association

(Req) Request

(Disass) Disassociation

(Res)	Response
(Reass)	Reassocation
(Authen)	Authentication
(De-Authen)	Deauthentication

Management frames establish the initial communication between stations and their respective access points.

1) *Association Request Frame:* Sent by a station to an Access Point (AP) to request an association with a specified BSSID. The station is associated with the access point after the access point grants permission (authentication).

2) *Association Response Frame:* An AP responses to a station requesting association. This frame indicates whether or not it is accepting the association with the requesting station.

3) *Reassociation Request Frame*: Sent by a station when it moves from one BSS to another and needs to make an association with the new AP. The station uses a Reassociation Request rather than an Association Request so that the new AP knows to negotiate with the old AP for the forwarding of the data frames.

4) *Reassociation Response Frame*: Sent by an AP to notify the station whether or not the reassociation request was accepted.

5) *Probe Request Frame*: Used to determine a specific station or AP.

6) *Probe Response Frame*: When a station or an AP receives a Probe Request frame, it responds with a Probe Response Frame to specify its communication parameters.

7) *Beacon Frame:* Sent by an AP to synchronize stations that are using the same physical layer in an infrastructure mode.

8) *Announcement Traffic Indication Message Frame (ATIM):* Sent by a mobile station to notify other stations, which may have been in low power, mode that this station has frames buffered and waiting to be delivered to the addressed station.

9) *Disassociation Frame:* Sent by either an AP or a participating station to terminate an association.

10) *Authentication Frame:* Used by a station to authenticate itself to an AP or to another station. Authentication consists of several authentication frames according to the type of authentication.

11) *Deauthentication Frame:* Sent by a station to terminate the secure communication.

b. Control Frames

Control frames provide support for the reliable delivery of data frames after establishing an association and completing authentication via management frames. There are six types of control frames:

1) *Request to Send (RTS):* Used when one station alerts another particular station of a desire to transmit data frames. Note that the other station within range of the originator also receives the RTS and is thus aware of the pending busy status of the medium.

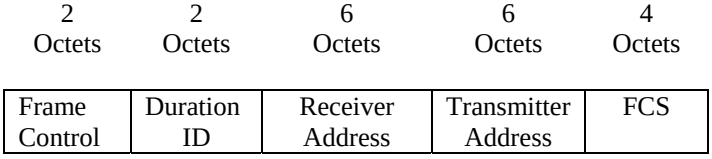


Figure 11. RTS Frame Format

2) *Clear to Send (CTS):* Sent by the destination station to the source of an RTS to grant that station permission to send data frames.

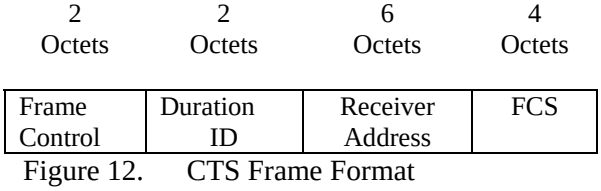


Figure 12. CTS Frame Format

3) *Acknowledgement (ACK):* Sent immediately by a station upon receipt of a data, management, or a PS-Poll frame that was received without detected errors.

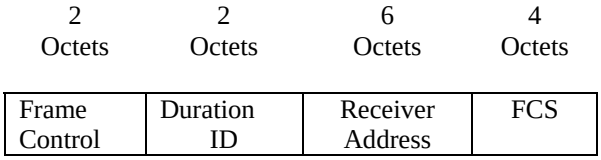


Figure 13. ACK Frame Format

4) *Power-Save Poll (PS-Poll):* Sent by any station, including the AP, to request that the AP transmit a frame that has been buffered for the station while the station was in power-saving mode.

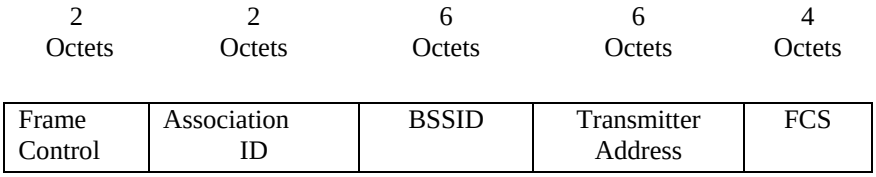


Figure 14. Ps-Poll Frame Format

5) *Contention-Free End (CF End): A*nnounces the end of the contention-free period that is part of the coordination function.

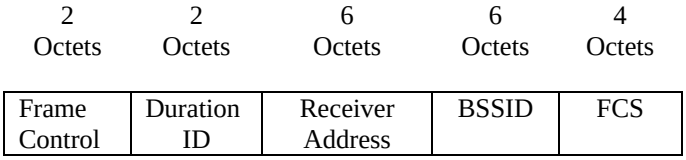


Figure 15. CF End and CF End + CF ACK Frame Formats

6) *CF End + CF-ACK*: Acknowledges the CF-End frame and ends the contention-free period.

c. Data Frames

Data frames can carry data and control information to the destination hosts. Its format is as follows:

1) *Data:* This frame contains only data and can be used during either the contention free or the contention period.

2) *Data + CF-Ack:* This frame contains acknowledgement information about received data and the data being sent. It can be used only at the contention-free period.

3) *Data + CF Poll:* This frame is used by a point coordinator, an AP or a master node to deliver data. It also allows mobile nodes to send buffered data.

4) *Data + CF-Ack + CF-Poll*: This frame combines the functionality of both Data + CF-Ack and Data + CF-Poll frames.

2 Octets	2 Octets	6 Octets	6 Octets	6 Octets	2 Octets	6 Octets	0-2312 Octets	4 Octets
Frame Control	Duration ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	Frame Body	FCS

Figure 16. 802.11 MAC Data Frame Format

To-Ds	**From-Ds**	**Address 1**	**Address 2**	**Address 3**	**Address 4**
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

Figure 17. The To-Ds and From-Ds Fields of Frame Control (Figure 9) Define the Values of Address Fields.

5. Service Sets

A Service Set may be considered to be the basic components of a fully operational wireless LAN. There are three ways to configure a wireless LAN.

a. Independent Basic Service Set (IBSS)

An IBSS is known as an “ad-hoc” network. It has no access point (AP) or any other infrastructure access to a distribution system. A given IBSS covers only one cell. Since there is no AP, clients in the cell send the beacons in a specified order. One of the clients in the IBSS must act as a gateway by hosting special software, which provides communication to hosts not in the IBSS.

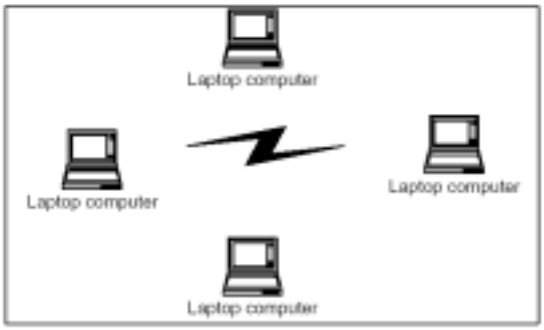


Figure 18. Independent Basic Service Set

b. Basic Service Set (BSS)

A BSS consists of one AP and one or more clients. It uses an infrastructure mode, which includes an AP, to deliver all the traffic between the wireless clients and the clients of the wired distribution system. In the BSS mode, no direct communication is allowed between the clients. The clients must use the AP to communicate with each other or use wired network services.

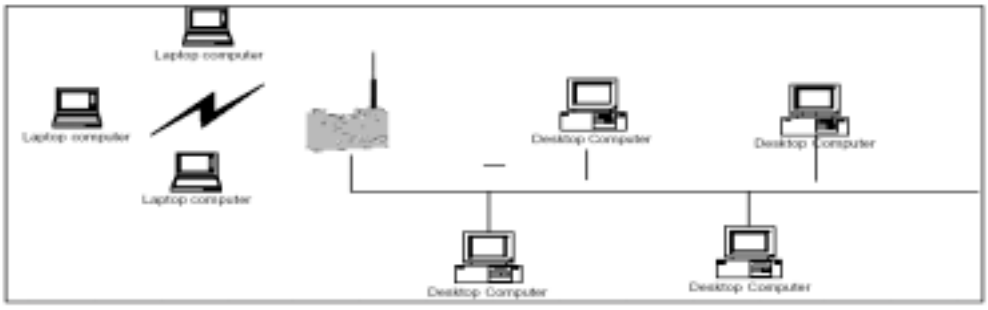


Figure 19. Basic Service Set

c. Extended Service Set (ESS)

AN ESS contains at least two Basic Service Sets connected by a common distribution that which can provide Internet connectivity. There must be at least two APs, one in each BSS, which forward all traffic in their particular ESS.

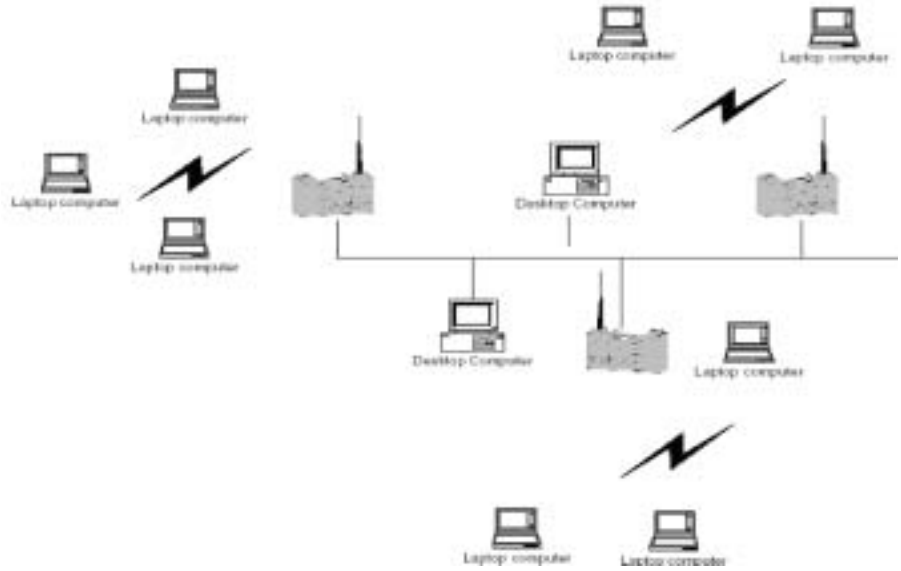


Figure 20. Extended Service Set

Even though an ESS may cover several cells, it does not require any roaming functionality or the assignment of the SSID between cells. Roaming allows wireless clients to move from one cell to another without losing network connectivity. Multiple access points may provide wireless roaming coverage for an entire enterprise or campus.

6. Security Mechanisms of IEEE 802.11 WLANs

a. Wired Equivalent Privacy (WEP)

WEP uses the RC4 stream chipper and a pseudo-random number generator to provide both the authentication and privacy functionalities of a security framework, as described above. The CRC32 integrity check value (ICV) in the WEP function serves only to provide fault detection. Confidentiality, Integrity and Authenticity (CIA), three major requirements for information assurance, are supported by WEP and CRC32. The

primary goal of WEP is to protect the confidentiality of the user data, that is, to protect it from eavesdropping. WEP is an international standard that most vendors integrate into their implementation of the 802.11 standard.

WEP uses four elements to provide data encryption and integrity:

1. A shared-key between all the members.

2. A 24-bit initialization vector (IV), which is changed for each new packet to support a per-packet key change. An IV is appended to the shared-key and used to generate a key stream by pseudo-random number generator (PRNG).

3. A 32-bit Cyclic Redundancy Check (CRC) algorithm to compute an integrity check value (ICV) to provide integrity assurance for the data payload.

4. An RC4 encryption algorithm, a stream chipper that simply XORs the plain text with a key stream that was produced by a pseudo-random number generator, seeded by both the initialization vector (IV) and the shared key. [3]

When WEP is enabled, the sender first computes the ICV value and appends that value to the data payload to support data integrity verification. The sender then picks a new IV value and appends it to the shared key. The RC4 algorithm uses the resulting concatenated bit stream to generate a pseudo-random number (PRN) key stream of the same length as the data to be encrypted. The Initialization Vector is used to support per packet key generation.

The sender simply XORs both of the plaintext data payload and the ICV value with the generated PRN key stream. Then it appends the IV, allowing the receiver to produce the same key stream and decrypt the WEP encrypted frame as illustrated in Figure 21. After this process, the MAC frame header and trailer are appended, in clear text, resulting in an 802.11 frame that is ready to send. It is worth highlighting that the LLC data unit, except for the generating IV, is encrypted while the MAC header and trailer are not.

At the receiver side, the receiver determines whether or not the received frame is WEP encrypted by checking the WEP bit in the frame control part of the frame

header. If the frame is encrypted, then the IV is extracted from the frame in order to produce the same key stream. The data payload is decrypted by XORing the encrypted payload with the key stream. Finally, the receiver verifies the CRC of the decrypted payload data to verify that the frame data was correctly decrypted. [10]

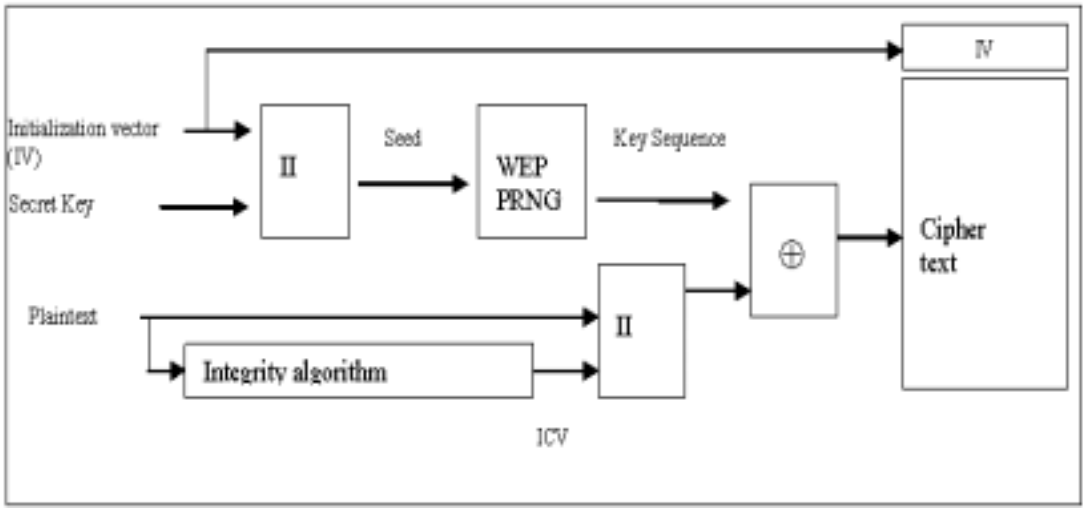


Figure 21. WEP Encryption

(1) Problems with WEP: Since WEP uses an RC4 steam cipher for encryption, which basically is a XOR operation, this operation gives attackers an opportunity to apply statistical attacks. If an attacker obtains two frames that are encrypted by the same key stream, obtaining the XOR of the two plaintexts is possible.

$$c_1(i) = p_1(i) \oplus k(i)$$
$$c_2(i) = p_2(i) \oplus k(i)$$
$$c_1(i) \oplus c_2(i) = p_1(i) \oplus p_2(i)$$

Another vulnerability caused by the RC4 XOR operation is the bit flipping. If an attacker flips a bit in the chipper text, it is simultaneously flipped into the plain text. This means that the chipper text is potentially vulnerable to modification attacks. WEP uses self defense mechanisms, such as the Integrity Check Value (ICV) and

the Initialization Vector (IV), to address these security leaks; however, it has been determined that these are not sufficient mechanisms. [10][11]

ICV is computed by a CRC32 linear checksum algorithm. If attackers have a chance to flip bit n in the message, they could produce the right checksum by also flipping a set of bits in the CRC, as determined by the position of the targeted bit. This allows the attacker to flip arbitrary bits in an encrypted message and to adjust the checksum correctly so that the resulting message appears valid. This demonstrates that the ICV mechanism could be circumvented by experienced attackers.

WEP uses an IV to generate different per-packet key streams. Since an IV is only 24 bits with total value space of 2^24. A busy access point, constantly sending 1500 byte packets at 11Mbps, will exhaust the space of IV's within five hours. As a result, a determined attacker could collect two cipher texts, which are encrypted by the same key within five hours. This might allow attackers to apply statistical attacks to recover a shared key. Some wireless interface cards reset the IV each time they are reinitialized and increase the IV collision probability of capturing a shared-key due to an IV collision. [6]

(2) Attacks on WEP: As noted above, attackers can passively monitor wireless transmissions and can collect traffic. When the attackers gather frames encrypted with the same key, they may be able to apply statistical analysis to recover the shared key. If half of the transmission could be predictable, such as the IP headers, the prediction could increase the attacker's probability of success. An attacker could send some frames with known content from another host on the internet. With these encrypted frames, and knowing most of the plain text, and attacker could produce the key stream.

Since the total IV space is 2^24, after some amount of time the attacker could build an IV–key stream table. This table requires ~15GB storage. After building this table, decryption can be made on the fly by an attacker.

b. Filtering

Filtering is a basic security mechanism used to prevent or to limit unauthorized access. Two types of filtering are mostly applied in WLANs, service set identifier (SSID) and MAC address filtering.

(1) SSID Filtering*:* The SSID is the first security credential that should be known by the authorized user in order to obtain network access. If a station wants to join a service set (BSS, IBSS or ESS), it must enter the appropriate SSID for the wireless interface in order to authenticate with the service set. Since the SSID is broadcast in the beacons and the probe responses, collecting the SSID information from the management frames is quite easy. Therefore it is recommended that SSID information be removed from beacon and probe response messages. Otherwise an attacker could passively obtain the SSID information.

(2) MAC Address Filtering*:* MAC address filtering can be applied at most of the access points. It could be done by building a list of addresses to be excluded or an authorization list specifically authorizing selected addresses. An AP does not associate a client, if the client is not in the authorized list or if the client is on in the excluded list.

Even though the MAC filtering mechanism is considered simple to implement, in a large network it is more appropriate to use a central authentication server like RADIUS to authenticate clients rather than update hundreds of individual AP MAC filter lists. Maintaining individual filter lists, one at each AP, is particularly error prone as a network's size increases.

MAC addresses can easily be captured by an attacker, even if WEP is enabled because the 802.11 MAC frame header is not encrypted by WEP. Since the MAC addresses of the sender and the receiver are in the clear, they might be obtained by a passive sniffer. Some network interface cards allow users to change the respective MAC addresses via the operating system (Linux) or a special software utility. As a result,

an attacker could join a network with what would otherwise be an authorized MAC address.

7. Major Attacks on 802.11 WLANS

a. Jamming Attacks

An attacker could shut down a wireless network by overwhelming the associated RF spectrum of the media. A RF signal generator or a sweep generator can generate high power RF signals, which can make wireless communications impossible. A Denial of Service (Dos) type of attacks can be performed by using jamming.

A RF spectrum analyzer could be used to locate the RF signal generator and thus locate the jamming agent. Currently a handheld device can support RF spectrum functionality with special software and hardware.

Wireless communications may also be intercepted by other devices, without any malicious intent. These unintentional interceptions may be done through baby monitors, microwave ovens, or 2.4 GHZ spectrum wireless telephones.

b. Passive Attacks

If WEP is not enabled during the wireless communication, unencrypted 802.11 sessions are subject to monitoring and hijacking, regardless of how the sessions are authenticated. These types of attacks are known as “parking lot” attacks. A directional antenna, such as a Yogi, can increase the monitoring range from 20 to 30 miles, so an attacker can obtain unencrypted traffic without being physically near the wireless device being monitored. If upper layer encryption is not applied, e-mails, instant messages, and telnet sessions are subject to unauthorized observance.

c. Active Attacks

Nikita Borisov, Ian Goldberg, and David Wagner described advanced techniques on how to compromise WEP in their paper, "Intercepting Mobile Communications: The Insecurity of 802.11."[10]

Since a determined attacker could recover a web key and easily obtain an authorized MAC address—even if MAC filtering is implemented—he could gain a layer II network connection to a private wireless LAN. If there is a DHCP server to distribute the available IP numbers, an attacker could easily map the network or plant Trojan Horses or zombies to invoke later. If the access points provide wired networking connectivity to mobile stations, an attacker not only achieves WLAN connectivity, but also acquires a back door to the entire network.

d. Man-in the-Middle (MiM) Attacks

In order to apply a successful "Man-in-the-Middle Attack" (MiM), an attacker must first obtain the SSID and the WEP key, if it is enabled. Two wireless network interface cards are needed to perform the attack. First the attacker acts like an authorized client and associates with the target network's AP. Second, the attacker acts like one of the networks' AP to send 802.11 data frames to a distribution center. If the attacker is bale to generate more transmission power than the legitimate AP, then the clients try to associate with the attacker's AP rather than the authenticated AP. The authorized network users believe they are interacting with the legitimate AP, while they are, in fact, sending valuable information to the attacker's AP.

e. Denial of Service Attacks by Unauthorized 802.11 Management Messages

The IEEE 802.11 specification does not enforce authentication by the management frames. Beacon, probe request/response, association request/response, reassociation request/response, disassociation, deauthentication frames can be used to generate Denial of Service Attacks.

D. THREAT MODEL

After discussing what the current 802.11 specification provides regarding security mechanisms and their vulnerabilities. The IEEE 802.1X standard addresses some of the vulnerabilities. This thesis is primarily concerns whether the 802.1X is sufficient enough to protect a WLAN when it is implemented as an extension of the legacy network such as an IEEE 802.3 Ethernet LAN in a naval headquarters or on a naval ship.

First, a threat model for that WLAN will be defined, and then the security requirements will be specified appropriate to the defined threat model. These requirements will consider the existing preconditions and the desired post conditions after the security perimeters are applied.

1. Preconditions

- The military organization implemented an Ethernet LAN as its legacy network environment.
- Multilevel information flows exist within the network.
- Application level security mechanisms are applied.
- The intranet is protected via a firewall and an intrusion detection system.
- All network users have sustained a current background check and are authorized, official users.
- There is a strict organizational security policy that controls information flows.
- Unauthorized mobile workstations such as laptops and handhelds devices are prohibited within the intranet without explicit permission.
- There are strict physical security perimeters for unauthorized mobile networking

2. Threats

- An attacker might attempt to eavesdrop 802.11 frames to acquire confidential data and security credentials to use with the authentication mechanism.
- An attacker might try to gain network access by connecting to a legitimate access point.
- An attacker might apply an MiM attack by spoofing a legitimate client and AP point MAC addresses.
- An attacker might generate RF signals to implement a jamming attack.
- An attacker might generate denial of service attacks by using either unauthorized management or control frames, including EAPOL and EAP packets.
- An attacker might attempt a session hijacking attack by gathering session security credentials, such as keys, initialization vectors, random numbers, etc.

3. Security Requirements

- The secure state of the local area network must be protected.
- Secure mobile-client authentication must be enforced by other than open or shared-key authentication mechanisms.
- A session key generation and key management functionality must be enforced on a regular basis to enforce privacy of data flow between mobile clients and network services.
- There must be a mutual authentication mechanism between network entities to address MiM attacks.

E. SUMMARY

Wireless communications basics are covered in this chapter. The IEEE 802.11 specification layers, which are analogous to the OSI physical and medium access control layers, are defined in detail. Without a clear understanding of these layers, it would be meaningless to describe what needs to be secured in order to protect a wireless network.

WEP and filtering are the security mechanisms provided by the current specification. If both mechanisms are applied correctly, they could protect WLANs from unauthorized access. However, when advanced, determined attackers are considered, there are many potential security leaks which may be exploited to access network services, as described in this chapter.

A threat model was built considering current vulnerabilities. A set of security requirements must be fulfilled in order to protect the secure state of the legacy network.

The next chapter will analyze the IEEE 802.1X standard and its entities when it is implemented in an IEEE 802.11 WLAN environment.

THIS PAGE INTENTIONALLY LEFT BLANK

III. IEEE 802.1X PORT BASED NETWORK ACCESS CONTROL

A. INTRODUCTION

As described in the previous chapter, the IEEE 802.11 standard is known to lack any viable security mechanism. IEEE has established another workgroup, IEEE 802.11i, to address all security vulnerabilities of the IEEE 802.11 specification. The IEEE 802.11i workgroup will finalize their specification by the end of 2003.

The IEEE 802.11i workgroup propose to use the IEEE 802.1X Port-Based Network Access Control Standard to provide access control, authentication, and key management for the 802.11 WLANs. After describing weakness of the authentication and privacy mechanisms of the current IEEE 802.11 standard, this chapter discusses if the IEEE 802.1X standard could address these weaknesses.

B. IEEE 802.11 AUTHENTICATION MECHANISM

It is essential to review how the current standard enforces client authentication, and determine whether or not the standard provides such authentication.

There are two types of authentication in the standard:

1. Open-system Authentication

As defined in the Chapter II, Figure 7, open-system authentication welcomes every client who knows the service set identifier (SSID). If WLAN does not enforce WEP, the access points make available network resources to any client able to associate with the access points in WLAN. Since it is easy to obtain SSID information by using free network monitors, such as Ethereal, Air Snort, or Netstumbler, an attacker can capture SSID information from beacons or probe request/responses. Currently, the Windows XP operating system monitors available wireless networks by default. As a result, open-system authentication does not provide any security mechanism other than network availability.

2. Shared-Key Authentication

The previous chapter explained how WEP is weak and can be easily cracked. If we examine Figure 8, which illustrated how shared-key authentication occurs schematically, we see that both a plain and an encrypted challenge are sent within the authentication process. Since the 802.11 specification uses only one key for both authentication and privacy, we could conclude that one key and encryption algorithm (RC4) are used for both the authentication (shared key) and encryption process. If attackers obtain a network key during the authentication process, they also obtain the encryption key. Further, attackers obtain the plain challenge text, *p(i),* and encrypted challenge text, *c(i),* they could easily obtain the key stream.

$$c(i) = p(i) \text{ XOR } k(i)$$

$$p(i) \text{ XOR } c(i) = k(i).$$

Since the RC4 algorithm uses an XOR operation to encrypt messages, it allows an attacker to obtain the key quickly from the cipher text. In this way, although shared-key authentication seems more secure for WLAN services; it serves as a foothold for malicious users to recover the key. That is why open-system authentication is considered to be a more secure option than the shared-key authentication.

C. IEEE 802.1X AUTHENTICATION MECHANISM

The IEEE 802.1X standard was approved by the IEEE in 2001. Its main goal is to provide network access control for the IEEE 802 networks. There are three main entities in the standard.

1). A *Supplicant:* An entity at one end of a point-to-point LAN segment that is being authenticated by an authenticator attached to the other end of that link.[8] A supplicant entity requests network access.

2). An *Authenticator:* An entity at one end of a point-to-point LAN segment that facilitates authentication of the entity attached to the other end of that link.[8] An authenticator entity relays 802.1X frames of a client (supplicant) to an authentication

server and after the authentication server approves the client's authentication, the authenticator provides network connectivity to the supplicant.

3). An *Authentication Server*: An entity that provides authentication service to an authenticator. This service determines from the credentials provided by the supplicant, whether the supplicant is authorized to access the network services provided by the authenticator. An authentication server entity authenticates, authorizes and provides accounting service for clients in the WLAN structure.

A port can be considered as a logical connection between the supplicant and the authenticator. An authenticator supports both the controlled port, which provides network services after the client passes the authentication, and the uncontrolled port, which relays EAP frames between the supplicant and the authentication server. A supplicant first communicates via the uncontrolled port to authenticate. Then, if it is successfully authenticated, the authenticator shifts the supplicant's port from the uncontrolled to controlled state enabling its use of the network services.

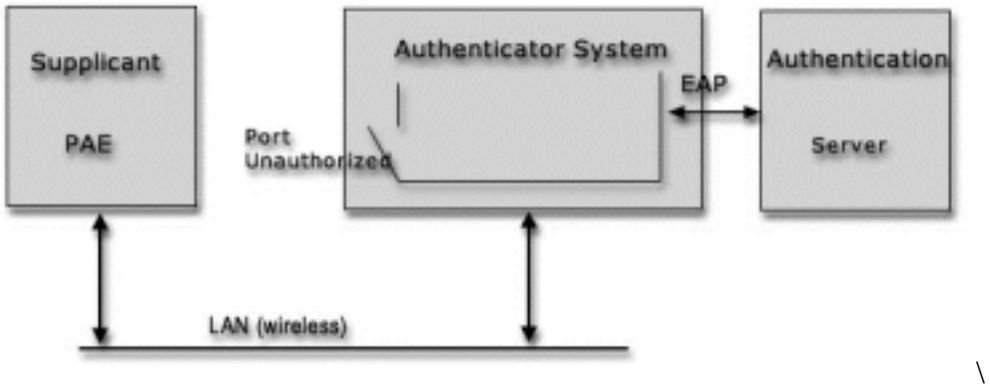


Figure 22. Authenticator, Supplicant, and Authentication Server Roles[8]

When we map these entities to a wireless local area network structure (WLAN), the mobile clients take the supplicant role, and an access point takes the authenticator role, since it provides the network access to the mobile clients. Finally, an authentication

server, such as a Remote Authentication Dial-In User Server (RADIUS), assumes the authentication server role.

The 802.1X standard uses the existing Extensible Authentication Protocol (EAP) and the RADIUS Protocol, which are defined in the IETF RFC protocol stacks and implemented by several standards. The 802.1X standard also defines the new EAP Over LAN (EAPOL) protocol to carry EAP packets in the 802.11 standard. Figure 23 demonstrates the general protocol stack that an 802.11 WLAN uses while applying the 802.1X authentication.

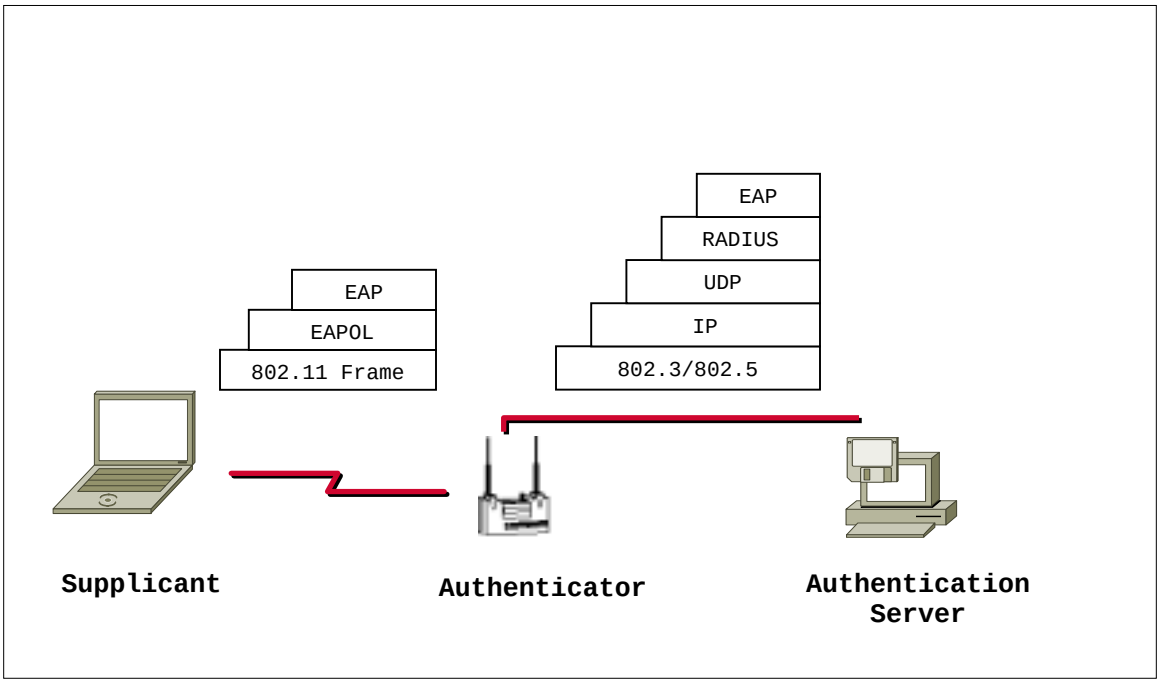


Figure 23. 802.1X Client Authentication Protocol Stack in 802.11 Frame Work

Since this thesis focuses on the 802.1x standard, it is beyond the scope of this research to define 802.3, 802.5, IP and UDP protocols and their specifications. The IEEE 802.1X standard uses these protocols to carry its protocol data units (PDUs).

1. Extensible Authentication Protocol over LAN (EAPOL)

The EAPOL protocol transports EAP packets between the authenticator and the supplicant. By using the EAPOL protocol, both the supplicant and the authenticator can initiate an EAP authentication session. The packet format of the EAPOL is defined in Figure 24.

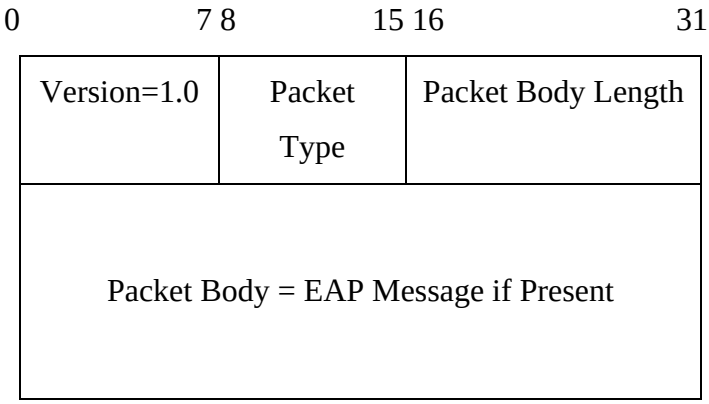


Figure 24. EAPOL Packet Format[8]

- **Version**: 1.0 is the current version.
- **Packet Type:** Five types of EAPOL packets are used currently:

  1). *EAP-Packet:* Indicates that the frame carries an EAP packet (0000 0000).

  2). *EAPOL-Start:* Indicates that the frame is an EAPOL-Start packet (0000 0001).

  3). *EAPOL-Logoff:* Indicates that the frame is an explicit EAPOL-Logoff request packet (0000 0010).

  4). *EAPOL-Key:* Indicates that the frame is an EAPOL-Key packet (0000 0011).

  5). *EAPOL-Encapsulated-ASF-Alert:* Indicates that the frame carries an EAPOL-Encapsulated-ASF-Alert (0000 0100).

- **Packet Length:** This field contains the total length of the Packet Body Field in octets.
- **Packet Body:** This field contains information if the packet type is not an EAPOL-Start or Logoff type. If the frame is an EAP packet then the body contains the EAP packet content. If it is an EAPOL-Key packet, it contains a Key Descriptor. If it is an EAPOL-ASF-Alert, it contains the ASF (Alerting Standard Forums) information.

2. Extensible Authentication Protocol (EAP)

The EAP protocol was first defined for the point-to-point protocol (PPP) in Request for Comment 2284 (RFC 2284). It provides negotiation and authentication methods between the end points in the PPP protocol. The IEEE 802.1X standard adopts the EAP protocol to provide an authentication mechanism for the IEEE 802.11 standard. The packet format of the EAP protocol is defined in Figure 25.

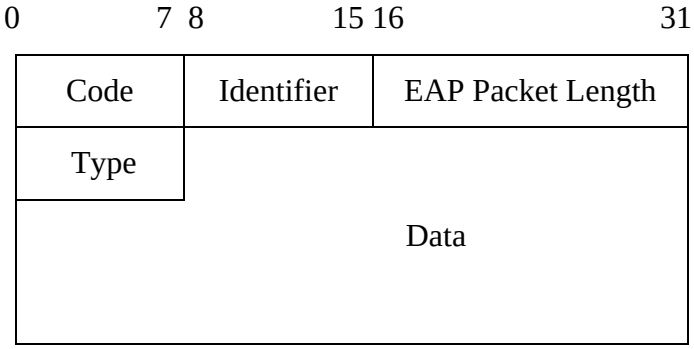


Figure 25. EAP Packet Format[8]

- **Code:** Defines the type of the EAP packet. The type could be request, response, success or failure EAP packet.
- **Identifier:** Contains one octet which allows responses to be matched with requests during the authentication process between the authentication server and the supplicant through the authenticator.

- **Length:** Contains the total length of the packet including code, identifier, length, type and data.
- **Type:** Indicates which authentication method the data payload is intended. The EAP protocol supports several authentication methods: Transport Layer Security (TLS), MD5, One Time Passwords (OTP), and Light-weight EAP (LEAP).
- **Data:** The format of the data field is determined by the Code field. This field could contain zero or more octets.

3. Remote Authentication Dial in User Service (RADIUS)

The authenticator and the authentication server use the RADIUS protocol to carry the EAP packets. The EAP packets are carried in the RADIUS packets as one of the attributes. The RADIUS protocol enforces per-packet authenticity and integrity verification between the authenticator and the authentication server via a shared secret and the HMAC_MD5 secure hash algorithm. The packet format of the RADIUS packet is defined in Figure 26.

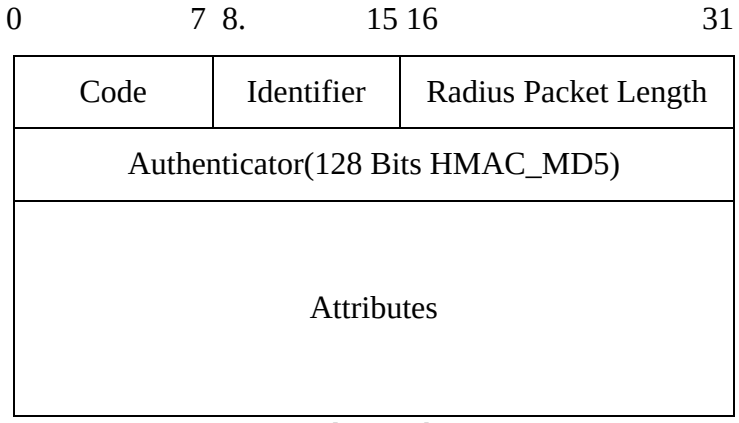


Figure 26. Radius Packet Format

- **Code:** Identifies the type of the RADIUS packet. This could be Access-Request (1), Access-Accept (2), Access-Reject (3), Accounting-Request

(4), Accounting-Response (5), Access-Challenge (11), Status-Server (12) or Status-Client (13).

- **Identifier:** Allows matching the request and replies between the authentication server and the authenticator. The RADIUS server can detect a duplicate request if it has the same set of client source IP address, source UDP port and identifier within a short span of time.[13]
- **Length:** Contains the total length of the RADIUS packet including the code, identifier, length, authenticator, and attributes fields.
- **Authenticator:** Contains a hash value which is computed by the MD5 hash algorithm with a shared secret between the authenticator and the authentication server. This value enforces the per-packet authenticity and integrity verification.
- **Attributes:** The RADIUS protocol supports a variety of authentication mechanism besides EAP. This field contains a variety of attributes. EAP is just one possible attribute in this field. Attributes are stuffed in this field. Attributes could be user-name, user-password, NAS-IP-Address, NAS-Port, Service-Type, and so on. The parameter types could be defined as needed for the authentication method. Each attribute subfield includes an eight-bit type value, which allows 256 different attributes.

4. 802.1X Authentication Message Sequence

All the protocols defined above are used by the IEEE 802.1X standard to provide reliable and secure client authentication for the IEEE 802.11 based WLANs. A complete 802.1X authentication session uses all the protocols as demonstrated in Figure 27.

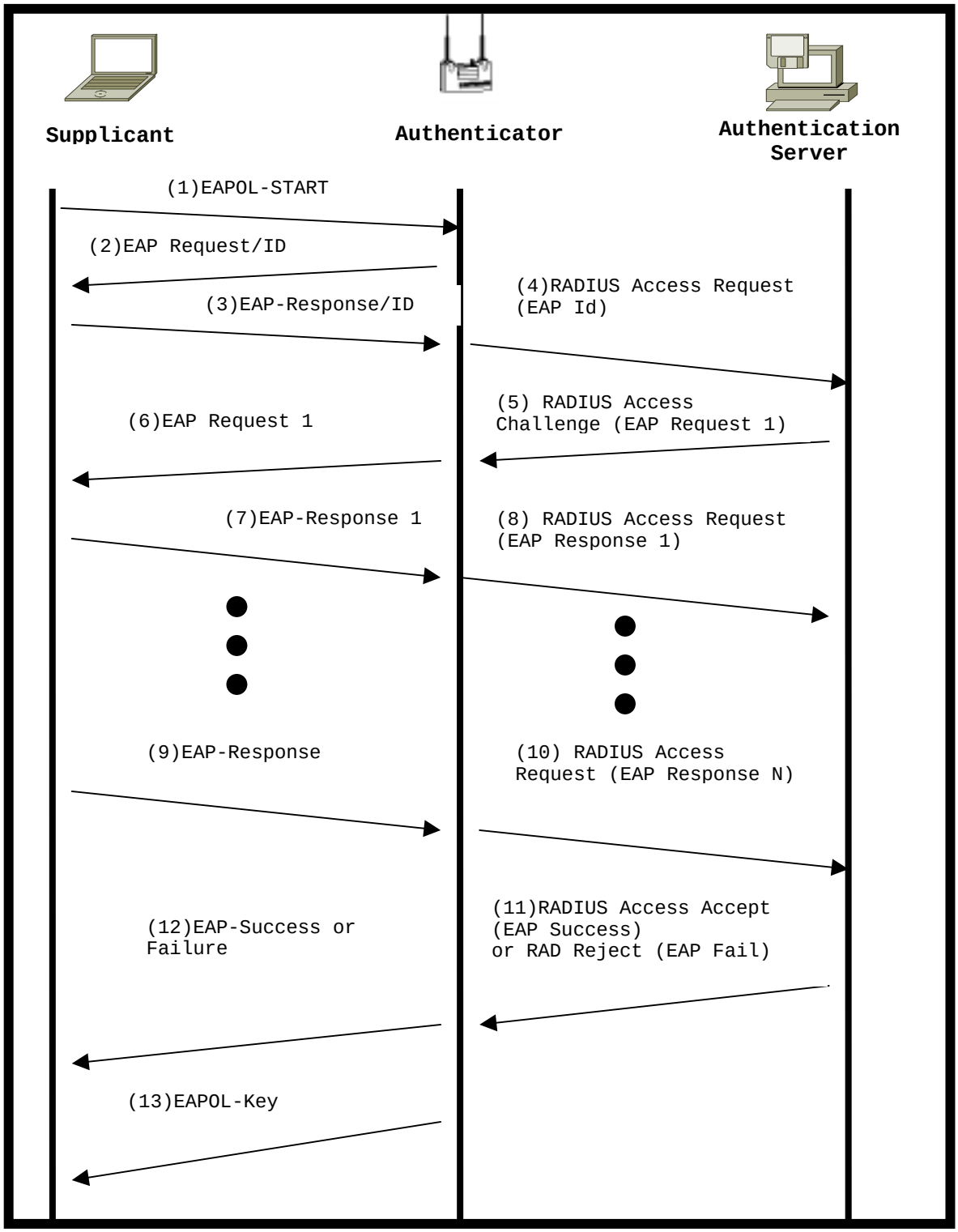


Figure 27. The 802.1X Authentication Session

An 802.1X authentication session follows a handshake sequence as below:

1) A supplicant initiates authentication by sending an EAPOL-Start packet to the authenticator.

2) The authenticator responds with an EAP-Request/Identity packet encapsulated in an EAPOL packet format.

3) The supplicant responds with an EAP-Response/Identity message. The authenticator receives the EAP packet from the EAPOL packet and places it in the RADIUS Access Request-EAP/Identity message. This RADIUS message is authenticated and has an integrity value. The authenticator is responsible for relaying the EAP frames between the supplicant and the authentication server. It repackages the EAP frames from the EAPOL to the RADIUS and vice versa.

4) When the authentication server receives the RADIUS Access-Request message, it verifies that the user ID in the message is matched with its user database.

5) If the authentication server finds an exact match for the user ID, it responds with a RADIUS-Access Challenge message to request security credentials with respect to the requirements of the authentication method. The credential may be a password for a One-Time Password (OTP) protocol, a hash value for an MD5 algorithm or a certificate request for the TLS protocol.

6) The authenticator relays this EAP-Request message in an EAPOL-EAP packet to the supplicant.

7), 8), 9), 10) The EAP-Request/Response messages are sent and received by the supplicant and the authentication server via the authenticator until the authentication is method is completed.

11) The authentication server sends an EAP-Success or Failure message as a result of the authentication method.

12) The authenticator relays this EAP-Success/Failure message to the supplicant. If it is a successful authentication, the authenticator shifts the state of the controlled port to an authenticated state and allows the authenticated supplicant to use the

network resources. Until this time, all the packets have been relayed via an uncontrolled port. In the case of failure, the authenticator disassociates the supplicant from the uncontrolled port.

13) If the supplicant is authenticated to the WLAN infrastructure, a key can be distributed by an EAPOL-Key message to enforce privacy. Key generation should be enforced by the authentication method. However, its implementation is optional.

5. Problems with 802.1X Authentication

The IEEE 802.1X protocol must be accomplished before the 802.11 data frames are transmitted. As defined before, the EAPOL frames are wrapped in the 802.11 frames within the communication path of the supplicant and the authenticator. After the authentication process ends, the client obtains network access and can send data. This authentication mechanism does not enforce authentication for other than layer II (data link layer), which is specified by the IEEE 802.11 standard. Upper layer security mechanisms could be applied to increase the security of the communications. Authentication by the authentication server does not mean that the client is authenticated to a LAN domain or a VPN connection. The 802.1X authentication just enforces layer II network access authentication. Obviously, the 802.1X would increase the composite security level of the WLAN. It definitely provides secure client authentication and optionally assists the key management process.

Since the IEEE 802.1X was not created just for 802.11 networks, the state machines of both standards should be merged correctly, which is the responsibility of the 802.11i workgroup.

The major security leaks of the 802.11 specification were listed in Chapter II. Even though the 802.1X provides security enhancements to the current wireless standard, it opens new holes in the authentication mechanism that the 802.11i workgroup should consider. Since neither EAP nor EAPOL supports authentication and privacy for the protocol handshake message themselves, Denial-of-Service (DoS) attacks could be generated by mimicking EAPOL and EAP frames specified in the 802.1X standard. Since

the main focus of the 802.1X protocol is wired networks, they do not require privacy and authentication at layer II, other than wireless networks. An attacker can spoof an EAPOL-Logoff frame and disconnect a client from the access point or exhaust an access point by flooding it with the EAPOL-Start frames. These unauthenticated EAPOL frames allow DoS attacks.

As defined in the EAP protocol, the EAP frame has an eight-bit identifier field that allows 256 client connections at a time. An attacker could circumvent this space and create a DoS attack by consuming the identifier field. Recall that EAP-failure/success messages are sent by the authenticator to the clients. An attacker could fool a client by sending EAP-failure packets and interrupting the authentication session.

Arunesh Mishra and William Arbaugh released a technical report on 6 February 2002 [13]. In the report, they claimed that the 802.1X standard is subject to man-in the-middle (MiM) attacks because the standard enforces only one-way authentication. If we consider only the IEEE 802.1X specification, we see that the standard does not enforce mutual authentication between the supplicant and the authentication server. The EAP Success message sets the current state of supplicant to authenticated. An attacker could forge this packet on behalf of the authenticator and potentially start a simple MiM attack. The adversary can thus obtain all network traffic from the supplicant [13]. The EAP does not require mutual authentication. MD5 and OTP authentication methods require just client-side authentication so they are vulnerable to MiM attacks as well. But we could choose an authentication mechanism that requires mutual authentication such as Transport Layer Security (TLS) to address this shortfall

Mishra and Arbaugh also reported that an attacker could disassociate a legitimate client from an access point by sending an EAPOL-Logoff frame by spoofing a legitimate AP’s MAC address. Then a malicious client could associate with the access point with the legitimate client’s MAC address at the authenticated state via the controlled port because the AP state information would still be at the authorized state for the legitimate client. This could be done, if the authentication mechanism does not implement the optional distribution of session keys. Without gathering session security credentials, this could be just a DoS attack rather than a Session Hijacking. Mishra and Arbaugh also

ignored the reassociation attempts of the legitimate client after being logged off by the bogus EAPOL-Logoff packet.

Both attack types require excellent knowledge of the IEEE 802.11 and the 802.1X specifications. Attackers might code their special malicious drivers for wireless NIC and access point interfaces. Attackers could create a flexible interface to initialize both interfaces with the required parameters at every state of the specification. However generating these attacks requires a high degree of motivation and excellent low-level coding skills.

D. SUMMARY

The previous chapters explained how the current IEEE 802.11 specification works, its security mechanisms, and security vulnerabilities. As noted, the current IEEE 802.11 specification cannot support secure client authentication to the WLAN and uses a breakable encryption key for privacy.

This chapter introduces how the IEEE 802.1X standard and describes how it can address these vulnerabilities if the right choices of authentication methods, such as EAP-TLS, are made by the system administrators.

Even though the 802.1X standard offers an improvement to the security level provided by the current 802.11 standard, it also increases the possibility of the DoS type attacks by spoofing unauthenticated EAPOL and EAP frames.

Both IEEE 802.11 and 802.1X standards should be revised and merged in order to provide per-packet authentication and integrity, including privacy for the management frames. Bernard Aboba proposed the 802.1X Pre-Authentication [14] to address this vulnerability and the IEEE 802.11i workgroup is revising the standard.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AN OPEN SOURCE WIRELESS PROTOCOL TEST BED

A. INTRODUCTION

The IEEE 802.1X standard addresses some of the IEEE 802.11 security vulnerabilities. This thesis primarily examines whether the IEEE 802.1X framework is sufficient to protect a WLAN when it is implemented as an extension of a legacy network, such as an IEEE 802.3 Ethernet LAN in a naval headquarters or a naval ship.

For verification, an 802.1X test-bed was built on an IEEE 802.11b wireless LAN. This chapter explains how to build and configure the 802.1X entities: the supplicant, the authenticator and the authentication server. For availability and ease of source code verification, open-source software was combined with the Linux Operating System environment.

B. 802.1X AUTHENTICATION TEST BED

This research considers open-source software as a base because it is easier to validate the source code and to manipulate it when required. This section explains how to combine the Linux environment with the required open-source software as illustrated in Figure 28.

After building the test-bed, the EAP-TLS authentication method and the X.509 public key certificate (PKC) are examined. One of the main factors in choosing the right authentication method for the EAP protocol is the ability to provide mutual authentication between the 802.1X entities, namely the supplicant, the authenticator, and the authentication server. Without mutual authentication a WLAN is vulnerable to man-in-the-middle attacks.

First, the available authentication methods were examined. Two types of authentication methods were implemented and supported by the current authentication servers. The first was Cisco's Lightweight Extensible Authentication Protocol (LEAP); the second one was the Transport Layer Security (TLS) [16].

Since LEAP is a proprietary implementation of Cisco, it was not applicable and available for the test experiments. That is why the EAP-TLS authentication method was chosen for mutual authentication. TLS requires certificate based authentication in a public key infrastructure (PKI).

1. Hardware and Software Configuration

There are three roles to fulfill in the 802.1X authentication mechanism—the supplicant (mobile client), the authenticator (access point), and the authentication server.

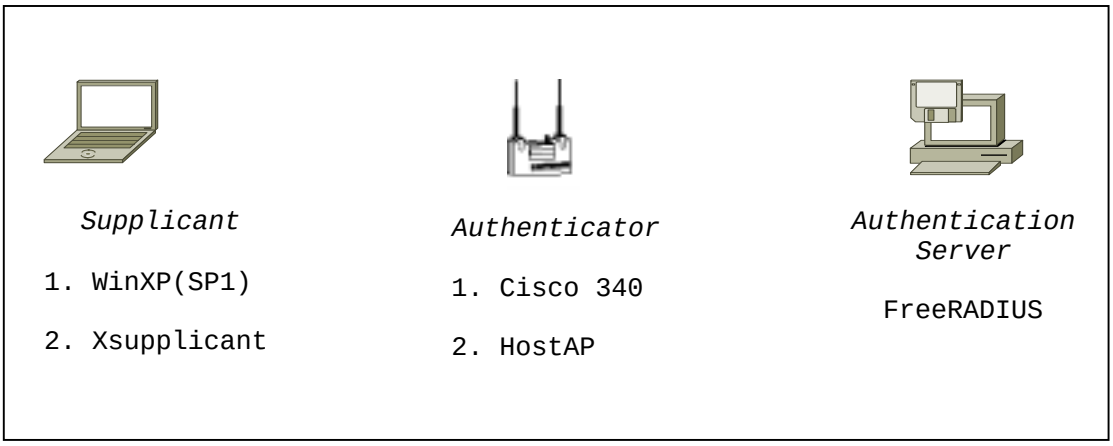


Figure 28. 802.1X Test-bed Schema

a. Supplicant

A Pentium III laptop with PCMCIA support hosts the supplicant. A D-Link DWL-650 IEEE 802.11b compliant wireless network interface card (NIC) was used for wireless communication. Two types of supplicant entities were used for testing purposes.

(1) Windows XP SP1: Microsoft Windows XP Service Pack 1 (SP 1) operating system was used for its embedded IEEE 802.1X supplicant support. Since D-Link officially supports Windows drivers, it was easy to download and to install the WinXP DWL-650 drivers. It is tested with 40-bit and 128-bit WEP encryption. Appendix B covers how to install certificates into the WinXP Supplicant Client for the IEEE 802.1X authentication. WinXP was chosen because of its wide acceptance and tested 802.1X support.

(2) Xsupplicant: Xsupplicant, an open-source supplicant entity, was chosen to fulfill the supplicant role. Xsupplicant is a part of the Open1X project, which is supported by the University of Maryland. Open1X is a complete IEEE 802.1X project, which implements the supplicant and authenticator parts and combines these entities with FreeRADIUS as the authentication server.

Linux Red Hat 8.0 operating system hosts the Xsupplicant on a mobile laptop. The same DWL 650 NIC is used as a wireless interface card. Since D-Link does not officially support Linux drivers, the chip-set of DWL-650 was examined. D-Link DWL-650 NIC uses the Intersil Prism2 chip-set, which is used also by Oronico products. Therefore, Red Hat 8.0 was chosen for its default support of the DWL 650 NIC via the *ornico_cs* driver. The performance of the driver was tested before testing the Xsupplicant support. The wireless NIC could associate with the legacy IEEE 802.11b access points and encrypt packets with the WEP option turned on.

If Red Hat 7.2 is used as an operating system, it does not support DWL 650 NIC by default. The newly released PCMCIA packages must be built to bind with the appropriate drivers for previous versions of the RedHat Linux operating system.

Before installing the Xsupplicant source code, three dependent libraries should be built and installed:

- OpenSSL 0.9.7 (http://www.openssl.org/) or greater
- Libpcap 0.7.1 (http://www.tcpdump.org/) or greater
- Libnet 1.1.0 (http://www.packetfactory.net/libnet/) or greater.

To begin, all the downloaded tarballs should be uncompressed to a desired directory, */usr/src/xsup* suitable for traditional reasons. The “readme” and “install” documents should be reviewed before building libraries. The default configuration methods are enough for building .

cd /usr/src/xsup/directory name
./configure

make

make install

The Above commands are enough for building and installing dependent libraries for the Xsupplicant. The kernel source code tree must be installed while installing Red Hat 8.0. Otherwise, the libraries will not be installed because they need the kernel source tree. The earlier versions of libraries should be uninstalled before installing new ones. It would be a better solution to uncheck these libraries while installing the operating system initially.

Xsupplicant source code can be downloaded from sourceforge.net (http://sourceforge.net/projects/open1x/). After uncompressing the tarball into the */usr/src/xsup* directory, the readme and installation files should be reviewed.

cd /usr/src/xsup/xsupplicant

./configure -–enable-full-debug

make

make install

The "--enable-full-debug" flag allows monitoring compile and run-time errors easily. Compile time monitoring is crucial for determining whether or not the dependent libraries are found by Xsupplicant's configuration script.

Appendix A explains how to create certificates for Xsupplicant. These certificates should be copied into the designated directory structure as defined in the configuration file (*1x.conf*). The Xsupplicant daemon parses this configuration file and loads the certificates into the memory to use through the EAP-TLS handshake, as explained later in this chapter. An example configuration file is provided in the Appendix B. This configuration file must be copied to the */etc/1x/* directory structure because the Xsupplicant daemon requires the *1x.conf* file to be in that directory by default. The Xsupplicant daemon can be invoked after successfully installing of the dependent libraries and Xsupplicant source code, and creating and installing certificates, as defined in the */etc/1x/1x.conf* configuration file.

iwconfig eth0 essid test (network id)

xsupplicant –i eth0 (interface name)

The Xsupplicant daemon is simply invoked by the commands defined above from the command line. Then the wireless NIC is forced to open-system authenticate and associate with the AP (authenticator). Then the Xsupplicant daemon must be invoked by referring to the right interface with which to communicate and enforcing the 802.1X authentication. The Xsupplicant daemon will ask for a secret to decrypt the private key in the certificate. The Certificate creator must distribute this key manually to the legitimate user. The authentication server and the authenticator logs should be examined in order to verify a successful authentication. Xsupplicant does not support WEP rekeying and discard EAPOL-Key messages coming from the authenticator, even in the CVS version of the source code.

b. Authenticator

The Cisco Aironet 340 and the HostAP, an open-source access point project, assume the role of authenticator.

(1) Cisco Aironet 340: Cisco 340 is chosen because it was tested against the EAP-LEAP protocol and works flawlessly with the RADIUS server. Appendix C explains how to configure the access point for the EAP support. Since Cisco supports WEP rekeying with their proprietary RADIUS server, the Cisco 340 could not support the dynamic rekeying process from the authenticator entity.

(2) HostAP: The main purpose of this research is to build an open-source 802.1X test-bed. The first attempt was to use the authenticator part of the Open1X project, but it was just compatible with the supplicant part of the Open1X project and required an access point to upload the authenticator code. Since the research was limited by the hardware, it was not easy to upload the access point firmware into a DWL-650 wireless NIC. The HostAP driver is chosen after unsuccessful attempts were made to upload the access point firmware because the HostAP supports Prism 2 chip set wireless NICs.

The HostAP (http://hostap.epitest.fi/) is a Linux driver for wireless LAN cards based on Intersil's Prism2/2.5 chipset. The driver supports a so-called HostAP mode, which takes care of the IEEE 802.11 management functions in the host computer and acts as an access point. This does not require any special firmware for the wireless LAN card. In addition to this, the HostAP has some support for normal station operations in BSS and possibly also in IBSS. Intersil's station firmware for the Prism2/2.5 chipset supports the HostAP mode in which the firmware takes care of time-critical tasks, like beaconing and frame acknowledging, but leaves other management tasks to the host computer driver. This driver implements the basic functionality needed to initialize and to configure Prism2/2.5-based cards, to send and to receive frames, and to gather statistics. In addition, it includes an implementation of the following IEEE 802.11 functions: authentication (and deauthentication), association (reassociation, and disassociation), data transmission between two wireless stations, power saving (PS) mode signaling and frame buffering for PS stations.

An IBM Think-Pad 600 Pentium II Laptop and a D-Link DWL-650 wireless NIC were the hardware base for the authenticator. First, Linux Red Hat 7.2 with a linux-2.4.7-10 kernel operating system was installed to support the HostAP driver.

The HostAP driver requires a special kernel configuration. Linux-2.4.20 kernel (http://www.us.kernel.org/pub/linux/kernel/v2.4/ or http://www.kernel.org/) is required with special patches. First, the tarball of the new kernel was downloaded and uncompressed into the */usr/src/* directory tree. Then the Wireless Extensions v15 (http://www.hpl.hp.com/personal /Jean_Tourrilhes/Linux/ iw240_we15-6.diff) and v16 (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/iw241_we16-2.diff) was patched to the new kernel source tree via a “patch” command. The diff files were copied into the */usr/src/* directory in order to run the commands below.

patch –p0 <we15.diff

patch –p0 <we16.diff

After the patching process, the new kernel was built by using the commands below.

cd /usr/src/linux-2.4.20

make xconfig

make dep

make bzImage

make modules

make modules_install

There are two configuration menus that must be enabled while configuring a new kernel. After *make xconfig* command, a graphical configuration menu helps to choose the kernel options. The wireless LAN (non-hamradio) option for the HostAP support (Figure 29) and the 802.1d Ethernet Bridging option (Figure 30) for bridging support between wireless and Ethernet must be checked. These options are not checked by default. The authenticator would not work properly without these support options.

Figure 29. Kernel Wireless Extension Support

Figure 30. Kernel Bridging Support

A new kernel image should be created If there was no problem with building new a kernel,

cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.20

cp System.map /boot

To copy a new image and a system map using the commands above is required to boot from the new kernel. LILO, the boot loader for the Linux Operating System, must be modified to recognize the new kernel.

Image = /boot/vmlinuz-2.4.20

Label = "Linux 2.4.20"

Lilo.conf, which is the LILO configuration file, resides at */etc/lilo.conf*. It was modified as described above. If one wants to also run older version kernels, a change might be concatenated to the configuration file.

lilo –v

The lilo –v command reads the changes and modifies the boot sector to allow one to reboot from the new kernel.

The HostAP (http://hostap.epitest.fi/CVS) source code and Wireless Extensions Tools v25 (http://pcmcia-cs.sourceforge.net/ftp/contrib/) must be built on the new kernel.

Wireless Extension Tools v25 were uncompressed into the */usr/src/* directory tree, built, and installed by the command sequence below.

./configure

make

make install

The HostAP source code tarball was uncompressed into the */usr/src/* directory. The HostAP source code must be configured for the special environment first by the command below.

./configure

The configuration file creates a "Make" file for this specific configuration. The KERNEL_PATH value was set to new kernel path in the */usr/src/hostap/Make* file.

KERNEL_PATH=/usr/src/linux-2.4.20

The HostAP source code must be built and installed after the proper configuration and required modification to the *Make* file because the HostAP requires kernel support.

make pccard EXTRA_CFLAGS="-DPRISM_HOSTAPD"

make install_pccard

The "Extra flag" option was required in order to support the 802.1X functionality for hostapd daemon.

Since the new kernel was configured while it was being built, the new kernel supported bridging. If it had not, bridge-utils (http://bridge.sourceforge.net/) could have been downloaded and installed. Bridging is crucial because the wireless side of the network must communicate with the wired side of the network. Otherwise the

HostAP can not relay the EAP packets coming from the supplicant to the authentication server which is on the wired side of the network. There must be two interfaces in the Laptop that hosts the authenticator. One of them is an Ethernet interface connecting the wireless distribution system to the wired side and the other is an 802.11b wireless interface supporting AP functions.

ifconfig wlan0 0.0.0.0.

ifconfig eth0 0.0.0.0

brctl addbr br0

brctl addif br0 eth0

brctl addif br0 wlan0

ifconfig br0 XXX.XXX.XXX.XXX up

Both interfaces' IP addresses must be set to zero and assigned to the bridge interface as defined above. The bridge interface (br0) is a logical interface rather than a physical one like a wireless (wlan0) or Ethernet (eth0) interface. The Bridge interface (br0) relays wired side packets to the wireless side and vice versa.

Once the preceding action were accomplished, all the hardware and software were ready for running *hostapd* daemon to serve as an 802.1X supported access point.

cd /usr/src/hostap/hostapd

./hostapd –d hostapd.conf

As shown above, a configuration file is required in the */usr/src/hostap/hostapd* directory. The configuration file support is added to the project at the CVS version of the project. Previous stable releases must be configured by entering the desired parameters from the command line. That is the reason it is recommended to build and install the HostAP CVS version. Appendix C provides an example configuration file and parameter definitions.

c. Authentication Server

This research uses FreeRADIUS (http://www.freeradius.org/) as the authentication server because it is the only open-source tool available and can run on the Linux environment. FreeRADIUS supports the EAP-TLS authentication method as an embedded module.

The Linux Red HAT 7.2 operating system hosts FreeRADIUS as the authentication server. OpenSSL 0.9.7 (http://www.openssl.org) is required to build FreeRADIUS and the EAP-TLS module in the FreeRADIUS source code. After uncompressing the OpenSSL 0.9.7 tarball into the */usr/src/* directory, the source code was built and installed by using the commands below:

./configure

make

make install

make distclean

./configure –prefix=/usr/local/ shared

make

make install

The same OpenSSL library was installed in two different places: One of them is for FreeRADIUS, and the other is for the EAP-TLS module in the FreeRADIUS.

The FreeRADIUS (FreeRADIUS 0.8.1) modules were uncompressed into the */usr/src/* directory structure. The FreeRADIUS source code was configured to build by using the command below:

cd /usr/src/freeradius0.8.1

./configure –sysconfdir=/etc

The configuration script prepares the Make files to build the source code. The EAP-TLS make file, which was placed in the */usr/src/freeradius0.8.1/src/modules /rlm_eap/types/rlm_eap_tls/* directory, was modified as defined at Appendix D.

make

make install

After building and installing the FreeRADIUS, the user, the client, and the radius configuration files, which were in the */etc/radiusd* directory, they must be modified in order to serve as the 802.1X authentication server. Appendix D covers how to modify the configuration files.

If everything has been installed and configured correctly, FreeRADIUS should be ready to run and to authenticate the legitimate users via the EAP-TLS. A wrapper script is created to run FreeRADIUS with the correct SSL libraries. Appendix D provides a copy of this script.

2. EAP-TLS Authentication Method

The TLS protocol provides mutual authentication, integrity-protected cipher suite negotiation and key exchange between the supplicant and the authentication server.

The TLS was created to provide a security mechanism for applications over the transport layer in the TCP/IP protocol stack. B. Aboba and D. Simon released the PPP EAP-TLS Authentication Protocol RFC 2716 (Experimental), which is the basis for the EAP-TLS authentication for the IEEE 802.1X standard. Even though there are two sub-layers, the TLS Record Protocol and the TLS Handshake Protocol in the TLS protocol [16], the TLS Handshake Sub Layer is enough for EAP-TLS authentication.

The TLS Handshake Protocol provides connection security that has three basic properties:

1). The peer's identity can be authenticated using asymmetric or public key cryptography. This authentication can be made optional but is generally required for at least one of the peers.

2). The negotiation of a shared secret is secure. The negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

3). The negotiation is reliable. No attacker can modify the negotiation communication without being detected by the parties to the communication.

The TLS protocol is invoked at the fifth step of the 802.1X session as described in Figure 27. The protocol frame is defined in Figure 31.

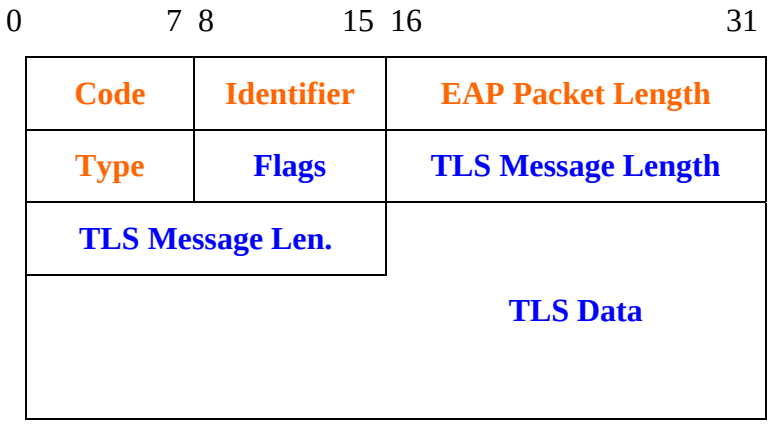


Figure 31. EAP-TLS Packet Format

- **Type**: Type field at the EAP header must be 13 to indicate that it contains a TLS packet.
- **Flags**: (L M S R R R R R)

  L: Length included

  M: More fragments

  S: EAP-TLS Start

  R: Reserved

- **TLS Message Length:** This field is present if the length included bit in the flag field is set to 1. This field provides the total length of the TLS message or set of messages that is being fragmented in octets.

- **TLS Data:** This field consists of the encapsulated TLS packet in TLS record format.

As described before, the authenticator is only responsible for relaying the EAP messages between the supplicant and the authentication server and switching between controlled and the uncontrolled port for the supplicant. Since the TLS protocol data is wrapped in the EAP packet, the authenticator does not know anything about the authentication method. This functionality is extremely powerful as it allows different authentication mechanisms to be used without changing the authenticator firmware or software. The authentication method is carried out between the supplicant and the authentication server. New authentication methods could be added to the supplicant by upgrading the supplicant firmware or software. After the authentication server decides whether the client is authorized or not, it notifies the authenticator (AP) with an Access-Accept or Access-Reject message. The TLS four-way Handshake Protocol is demonstrated in Figure 32 .

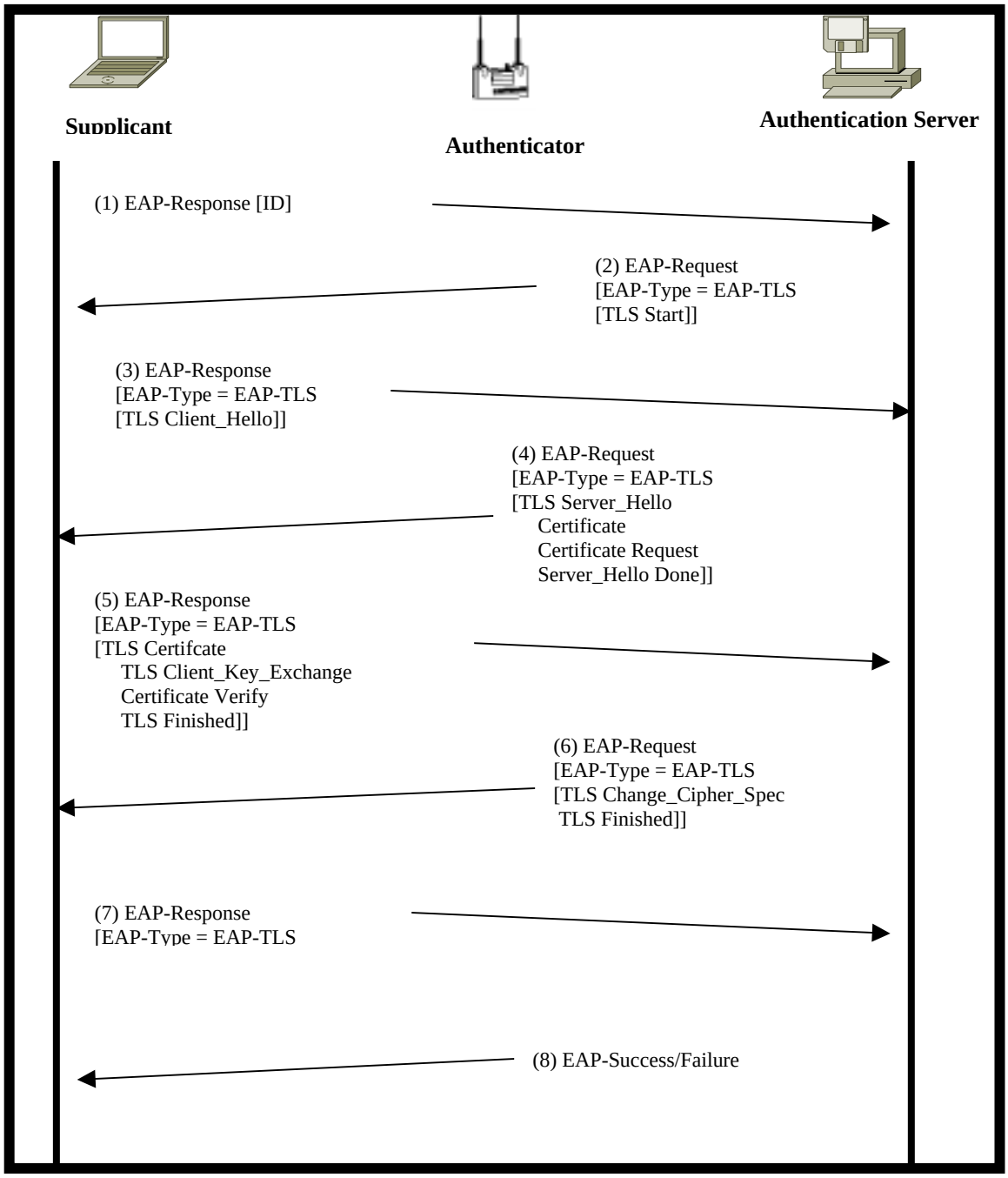


Figure 32. TLS Handshake Protocol Message Flow

Since the authenticator traverses the EAP packets between the supplicant and the authentication server, the EAPOL and RADIUS message flow is omitted in Figure 32.

1) The supplicant sends its identification after it obtains the EAP-Identity request from the authenticator. The authenticator then relays the ID information to the backend authentication server.

2) The authentication server checks its user's database. If it can find the same ID information, it determines whether the authentication method corresponds with the ID information. In this specific example, it sends an EAP-Request packet with an EAP-Type= EAP-TLS and sets the S flag to "1" which indicates that it is a TLS-Start message.

3) When the supplicant receives the TLS-Start message, it interprets that its ID matched with EAP-TLS method at the authentication server. Then it sends a TLS-Client_Hello message containing the protocol version, the session-ID, the chipper suites supported by the client, and a random value (Client_Hello.Random). This random value provides randomness and protects the key for replay attacks and is used as input entropy to generate a master secret.

4) When the authentication server receives the Client_Hello message. It responds with four messages.

The Server_Hello message contains the server-version, the session-ID that matches the Client.sessionID, a random value (ServerHello.random), and a chipper suite which is selected by the server from the list in the Client_Hello.cipher_suites.

The Server_Certificate message generally contains an X.509v3 certificate. This certificate should contain a key type correspondent with the negotiated chipper type in the hello messages. This message can contain a chain of certificates starting with the sender's certificate except root for the certificate authority (CA) because the rootCA public key must be distributed independently in order to verify the certificate chain.

The Certificate_Request message contains information about the certificate types and the certificate authorities that the server accepts.

The Server_Hello.done message notifies the client that the Server_Hello message sequence is ended and the server waits for the client response.

5) When the client receives the Server_Hello.done message, the client sends a sequence of information to the server.

The Client_Certificate message is sent if it is requested by the server on the list of Server_Certificate_Request list. This is also generally an X509v3 certificate.

Client_Key_Exchange message contains a random premaster-secret which is an input for computing a 48 byte master secret at both ends. A key block is generated from this master secret to generate encryption keys. The lengths of the premaster secret and key block depend on the key exchange method. The premaster value is encrypted with the server's public key and sent along with the server certificate.

master_secret = PRF(premaster-secret,

"master secret",

Client_Hello.random +

Server_Hello.random)

key_block = PRF (master_secret,

"key expansion",

Server_Hello.random +

Client_Hello.random.)

The pseudo-random function (PRF) needs entropy parameters. PRF function is computed as follows.

A(0) = seed

A(i) = HMAC_hash (secret, A (i-1))

$$P\_hash\ (secret, seed) = HMAC\_hash\ (secret, A(1) + seed) +$$
$$HMAC\_hash\ (secret, A(2) + seed) +$$
$$HMAC\_hash\ (secret, A(3) + seed) + \ldots\ldots$$

$$PRF\ (secret, label, seed) = P\_MD5\ (S1, label + seed)\ XOR$$
$$P\_SHA\ (S2, label + seed)$$

The Certificate_Verify message provides explicit verification for client authentication. The client first generates a hash value from the previous handshake messages including hello messages by using MD5 and SHA algorithms. This hash value is encrypted with the client's private key and sent with this message.

TLS_Finished message contains a verification value for the recipient.

$$Verify\_data = PRF(master\_secret,$$
$$\text{“client finished”},$$
$$MD5(handshake\_messages)+$$
$$SHA(handshake\_messages)\ )$$

6) When the server receives the client certificate, it first validates if the client is one of the legitimate clients who has a certificate signed by a trusted certificate authority. Then it decrypts the premaster secret and generates the master secret. The same key_block is generated from the same master-secret and random values which both the client and the authentication server share with the hello messages. The server could verify keys by applying the same algorithms with the same keys. By comparing the results with the received Certificate_Verify and Tls_Finished messages, the authentication server concludes the session keys whether correctly distributed or not. After verification, the

server sends a Server_Finished message containing the verification data for the client to validate the session-keys distribution.

Verify_data = PRF(master_secret,

“server finished”,

MD5(handshake_messages)+

SHA(handshake_messages))

7) When the client receives the Server_Finished message, it concludes that session keys exchanged correctly. Then the client sends an EAP-Type = EAP-TLS message.

8) If the authentication server accepts all the security credentials that the client presents, the client is authorized to use the network services. The authentication server sends an EAP success message via the authenticator (AP) to notify the client to use the network services and the authenticator to authorize the controlled port for the authenticated client. The authentication server sends the session keys which are derived from the key block in the RADIUS access accept message. This message also carries the EAP success message for the client. These keys are carried as a RADIUS attribute and encrypted with the shared secret between the authentication server and the authenticator. If dynamic WEP keying is supported by both the authenticator and the supplicant, the AP generates the WEP key and sends this key via an EAPOL-Key message to enforce privacy.

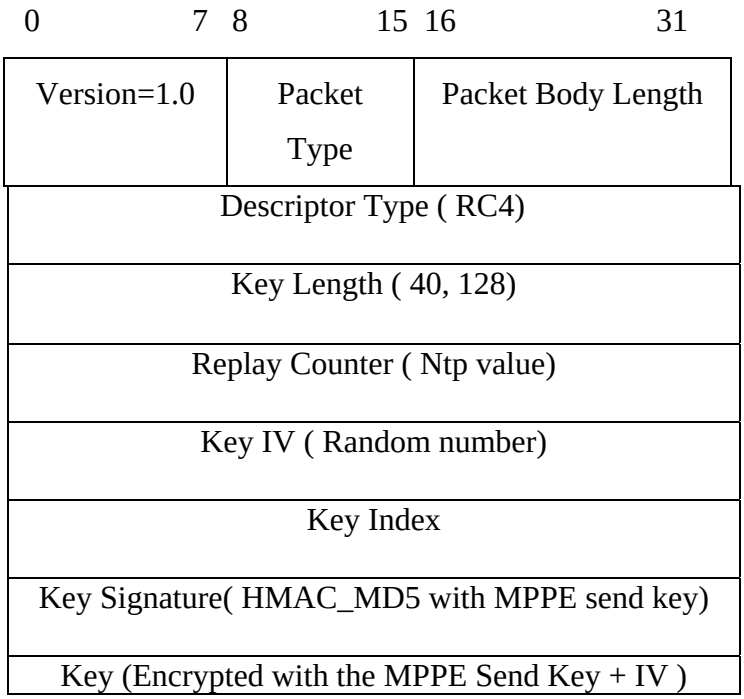


Figure 33. EAPOL Key Message

3. X.509v3 Certificates

The EAP-TLS authentication is a certificate-based authentication method. X.509v3 certificates are created for root certificate authority (rootCA), authentication server, and supplicant.

A certificate refers to a Public Key Certificate containing a public key of an end entity and some other information. It is digitally signed by the private key of the CA that issued it. A Certificate Authority (CA) is trusted by one or more users to create and to assign public key certificates. The CA could create the user's keys.

The OpenSSL security tool kit was used for creating certificates. Both the authentication server and the supplicant require OpenSSL libraries to manipulate certificate data.

The certificates could be created independently and installed into both the authentication server and the supplicant. A trusted, self-signed rootCA should be created first. Then supplicant and authentication server certificates are created and signed by this

trusted rootCA. This mechanism provides authentication verifying another party with regards to the trust relationship with the trusted rootCA.

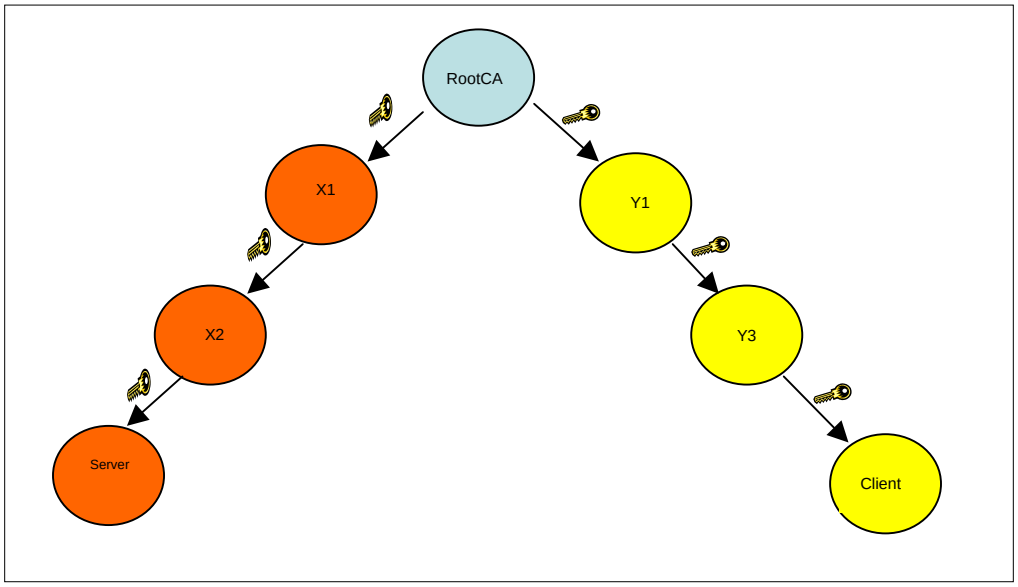


Figure 34. Certificate Trust Model

As shown in Figure 34, there could be intermediate certificates with signing capability. Both the authentication server and the supplicant client must have a trusted rootCA certificate to authorize the other party's identity. The RootCA must be installed before the authentication and then the certificate chain could be send via TLS handshake messages.

The authentication server can be used as a certificate generation entity and hosts the certificate authority. Three scripts were created to generate rootCA, the supplicant and the authentication server certificates. Two versions of OpenSSL were used for creating certificates because WinXP requires a special version of the OpenSSL release. Appendix A covers how to build OpenSSL libraries and explains certificate generation scripts.

D. SUMMARY

This chapter covers the main focus of this thesis: an open-source implementation of the 802.1X standard with a specific authentication method, transport layer security (TLS) over extensible authentication protocol (EAP).

TLS provides mutual secure authentication via certificates between the supplicant and the authentication server and creates session-based keys at both the supplicant and the authentication server side.

A detailed installation and configuration manual for building an 802.1X test-bed is provided. The supplicant, the authenticator, and the server roles are fulfilled with the required hardware and software options.

V. DISCUSSION

A. INTRODUCTION

This thesis has examined the security vulnerabilities of the IEEE 802.11 standard, a proposed solution, embodied in the IEEE 802.1X standard, to address these vulnerabilities, and finally I address how to apply these security perimeters in an open-source environment. This chapter discusses the initial utility of the test-bed implementation. It further examines whether a possible 802.1X solution could fulfill the security requirements defined in Chapter II.

Even though the 802.1X standard specifies the EAP protocol to enforce authentication, this thesis specially considered the EAP-TLS protocol. Since the EAP-TLS method could support mutual authentication and session key generation, the EAP-TLS method is used in the 802.1X evaluation process.

B. UTILITY OF TEST BED

As noted earlier, A. Mishra and W. Arbaugh claim that the IEEE 802.1X is vulnerable to session hijacking and man-in-the-middle attacks in their paper, *"An Initial Security Analysis of the IEEE 802.1X Standard"* [13]. They proposed that after a legitimate client authenticates itself to a WLAN and obtains network connectivity from the authenticator, and after the appropriate message (12) exchange, which is depicted in Figure 27, an adversary can send an 802.11 MAC disassociate management frame using the AP's MAC address. This causes the supplicant to be disassociated from the AP, but keeps the authenticator state machine at the authenticated state. Then the adversary gains network access using the MAC address of the affected authenticated supplicant. [6]

A. Mishra and W. Arbaugh also noted that an EAP-Success message sets the *eapSuccess* flag, which makes a direct transition to the a*uthenticated* state irrespective of the current state. Typically this would cause the interface to come up and provide network connectivity. Thus, an attacker could forge this packet on behalf of the

authenticator and potentially start a simple Man-in-the-Middle (MiM) attack. The adversary can thus obtain all network traffic from the supplicant to pass through it. [6]

Even though the session hijacking and MiM attacks are applicable, they need an open-source platform. The aforementioned attacks require spoofing MAC addresses, generating management and EAP-Success messages, and relaying data frames. These types of attacks cannot be evaluated without open-source test-bed support. The 802.1X test-bed explained in Chapter IV could be used to generate individual management, EAP, or EAPOL messages and to spoof the MAC addresses of the 802.1X entities.

1. Demonstration of a Denial of Service Attack on a 802.11B WLAN

A DoS attack was applied to demonstrate the utility of the test-bed. The WinXP supplicant served in the role of the legitimate client, the Cisco Aironet 340 access point performed in the role of the legitimate authenticator, the FreeRADIUS functioned in the role of the authentication server, and the hostap took the role of the attacker.

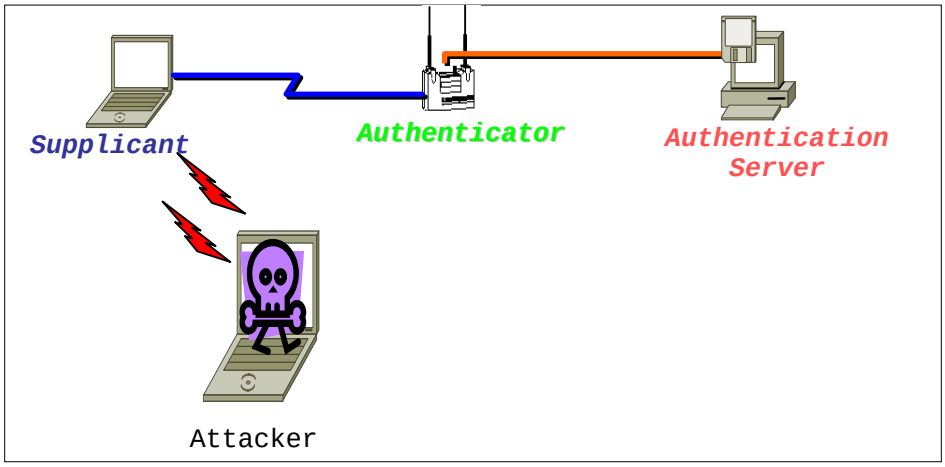


Figure 35. DoS Attack Entities

HostAP was used as an attacker, because the Linux operating system allows changing the MAC address of the wireless NIC. It is crucial to change the MAC address to impersonate the legitimate authenticator.

Once the successful authentication and key distribution was finished at the legitimate connection, the attacker monitored the WLAN communication. Netstumbler, a freeware network monitor, was used to determine the MAC address and the SSID information of the legitimate access point. These are enough information for the attacker to generate a DoS attack. The legitimate AP's MAC address and SSID were spoofed by the attacker AP with the following shell commands.

ifconfig wlan0 hw ether XX:XX:XX:XX:XX:XX

iwconfig wlan0 essid "?????"

./hostapd –d hostap.conf

These commands were enough to invoke the malicious access point with the spoofed MAC address and the SSID of the legitimate AP. The hostap driver generated the deauthentication frames by default at the startup [Appendix E].

When the legitimate client received these unauthenticated deauthentication messages, it tried to reauthenticate with the AP. Since the attacker generated a stronger RF signal than the legitimate AP, the client tried to associate with the attacker's AP.

Even though the client believed that it tried to authenticate with the legitimate AP, the malicious AP, impersonating the legitimate AP, directed the client's RF signal to associate with itself. The client tried to complete the 802.11 open-system authentication and association procedure with the malicious AP. However, the client could not complete the 802.1X authentication because there was no authentication server. The client could not use the network services as it did with the legitimate AP, thus suffering from a denial of service as desired by the attacker.

Figure 36 illustrates the DoS attack in detail.

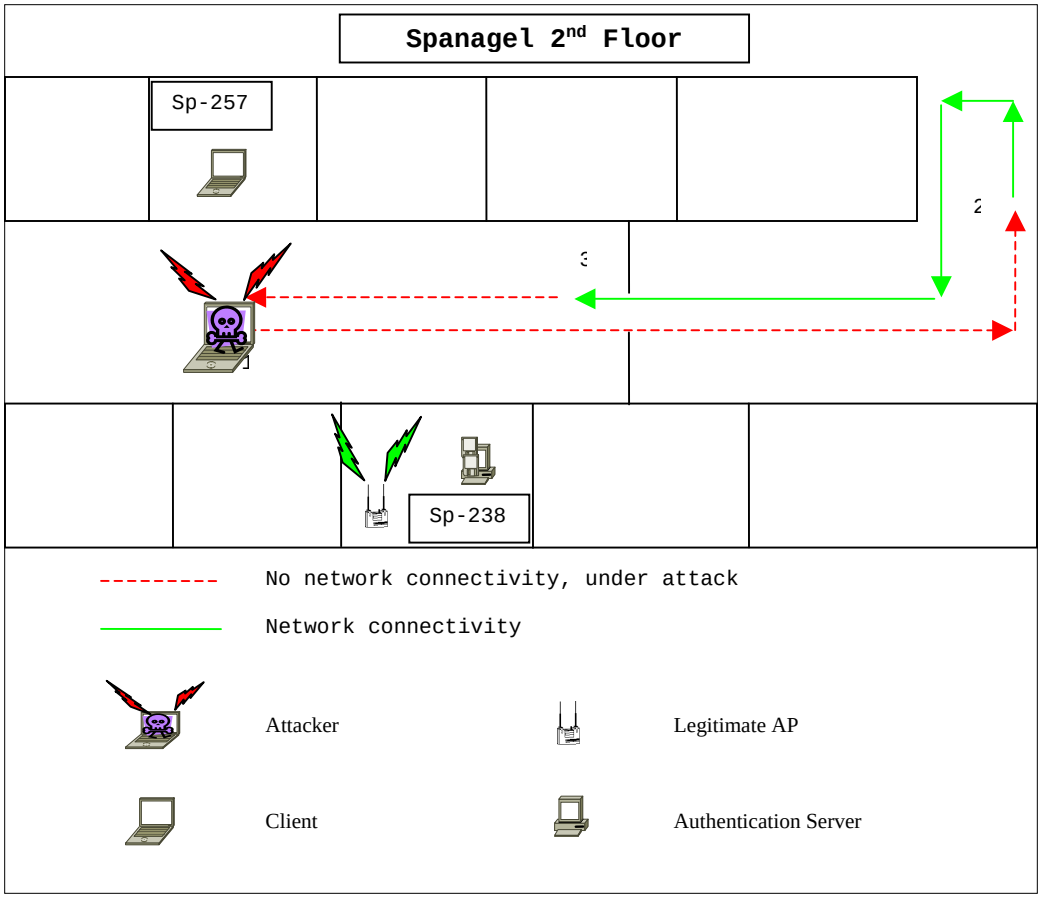


Figure 36. DoS Attack Placement

1). Client first completes an open-system authentication and association with the legitimate AP in room SP-238. The 802.1X authentication and key distribution is finished after the 802.11 open-system authentication and association is completed. The attacker then starts to run a malicious access point. That access point generates the deauthentication messages by impersonating the legitimate AP MAC address and SSID. Since the attacker's RF signal strength is more powerful than the legitimate AP, the client is forced to reauthenticate, but with the attacker's AP. At this moment, the client cannot use the network services, which the legitimate AP provides. The DoS attack is successfully applied.

2). The effective attack range is examined by moving the malicious AP. It is transported through a 200 foot hallway. The dotted lines indicate the attack period. The "Ping" command is used to check the connectivity with the authentication server because

only the legitimate AP has a wired connection with the authentication server. The client finally connects with the legitimate AP after the malicious AP's RF signal fades.

3). When the malicious AP's RF signal again overwhelms the legitimate AP, the client comes under attack again at point "3." The timers at the state machines most probably cause the connectivity difference in the "interruption range."

Since the 802.11 management frames and the 802.1X EAP/EAPOL frames are unauthenticated, these frames can be generated by changing the HostAP source code. Being able to generate these individual frames gives attackers the ability to apply DoS attacks.

C. EVALUATION OF 802.1X

The security policies defined in Chapter II could address possible threats and must be fulfilled in order to implement a WLAN in a military infrastructure. These include:

1. The secure state of the LAN must be protected.
2. Secure mobile client authentication mechanisms, other than open or shared-key authentication, must be enforced.
3. A session-key generation and key-management functionality must be enforced on a regular basis to enforce the privacy of data flow between the mobile clients and the network services.
4. There must be a mutual authentication mechanism between the supplicant-authenticator, the supplicant-authentication server, and the authenticator-authentication server in order to address man-in-the-middle attacks

1. Proposed Solutions

The scope of this section assumes that the WLAN will be implemented as an extension of an Ethernet LAN. The legacy network has a strong physical security, and it has been well protected from the Internet. The WLAN will use only the BSS topology,

rather than ESS, to decrease the network load and to ease manageability. The proposed solutions will be explained in light of the security policies.

a. The Secure State of the Legacy Network Must Be Protected.

The IEEE 802.11 specification resides in physical (I) and datalink (II) layers of the TCP/IP stack. Chapter II explained the functions of both the Physical Layer (PHY) and Medium Access Control (MAC) sub-layers.

As the media of a WLAN is publicly insecure, the IEEE 802.11 WLANs are vulnerable to data forging. Regardless of the security solutions of the IEEE 802.11, Virtual Private Networking (VPN) technology must be used to protect the upper layer data in the TCP/IP protocol stack. This could be done by using L2TPv3 or IPSEC technology. These technologies can protect upper layer protocol data between the VPN client and the server. VPN client software must be built in the supplicant client. The VPN server part could be implemented either in the authenticator or in a special VPN server used in the wireless broadcast domain.

The WLAN must be built as a separate subnet because of the broadcast behavior of the wireless communication. A firewall between the wireless subnet and the protected wired side of the network must be established. A VPN gateway with the firewall support would be a suitable solution for a small to medium WLAN substructure.

The SSID information must be removed from the beacon and probe request frames to hide the network information from the curious attackers. MAC filtering is recommended at the RADIUS server for central control.

WEP must be enabled whether it is dynamically or manually changed. Even though the WEP is considered weak, changing keys before the average initialization vector (IV) collision time makes the WEP key much less vulnerable.

Jamming and DoS attacks are ignored in this model because they are related to availability rather than confidentiality or authentication. Since wireless communication uses air as a media and the frequency band is in the public ISM and UNI

bands, it is legal for another electronic device to use the same frequency band. An attacker could generate a high power RF signal to jam the network communication. Since it is so expensive to capture the entire ISM or UNI band, generating that much power is unlikely. However, an RF survey is an important prerequisite before implementing a WLAN to prevent jamming from cordless phones, microwave ovens, etc.

b. Secure Client Authentication

Both the open-system and the shared secret authentication in the IEEE 802.11 specification are insecure. Open-system authentication welcomes everyone. Shared-secret authentication allows the WEP key to be obtained easily. EAP-TLS certificate based client authentication provides strong client authentication by using a central RADIUS authentication server.

The weakest point of the Public Key Infrastructure (PKI) is the blind trust to the root certificate authority. When PKI is implemented in a military infrastructure, this trust is established explicitly between the client and the system administrator. Since rootCA certificates are installed by system administrators or with their permission, it is quite hard for an attacker to obtain this trusted rootCA certificate unless there is an insider, a situation which must be carefully guarded against and continuously monitored. Military users have security clearances with regards to their level. It is the client's responsibility to protect the hardware and the organizational security policy must enforce this rule.

The trust relation starts with the physical authentication of the client by the system administrator. The system administrator must install or distribute the rootCA PKC to the legitimate users. Users are responsible for guarding the confidentiality, integrity, and availability of the rootCA PKC in the mobile supplicant device. The server is always under the control of the administrator. Without having rootCA PKC, it is impossible to validate certificates that are digitally signed by rootCA is impossible. The other security perimeter is the private-key decryption secret that is generated while creating certificates. Without this secret, the private key of the client certificate cannot be activated and the TLS four-way handshake never ends and the client cannot authenticate itself to the

server. This secret key must be distributed after physically authenticating the user. XSupplicant requires entering the shared secret manually. The WindowsXP client gives an option to choose if one wants to enter a password with each connection or not. The WindowsXP supplicant must be configured to require the password for every connection.

An attacker must have three security credentials in order to connect to a WLAN; the rootCA PKC, a legitimate client PKC, and the private key with the client PKC, and the secret to decrypt the private key to enable the user to use the system. There should be a certificate policy aligned with the security policy.

c. Session Key Generation

The methods to break the WEP key were explained briefly in Chapter II. [6], [7] and [11] are approved technical papers that explain how to crack the WEP key by using the weakness of the RC4 algorithm, IV space collision, and various attack methods.

There are two ways to prevent WEP key breaking. The first is to upgrade the encryption algorithm, which is the IEEE 802.11i workgroup's responsibility. Advanced Encryption System (AES) is expected to be the next encryption algorithm. Since the algorithm will not change until unless the 802.11i workgroup finishes its task and the IEEE approves it, in the interim the WEP key must be changed before IV collisions occur.

The second method is changing the encryption keys and allowing dynamic WEP keying, since the IEEE 802.1X allows changing the WEP key by sending EAPOL-Key messages from authenticator.

Static WEP keying can be used with the EAP-TLS authentication as in the Cisco 340 configuration. Since the EAPOL-Key message is optional to implement in the IEEE 802.1X standard, Cisco 340 AP does not support a dynamic WEP keying process.

The HostAP supports dynamic WEP keying with broadcast and unicast WEP keys. The broadcast WEP key is used for broadcasting messages within the broadcast domain. Unicast Keys are unique between the supplicant and the authenticator.

The system administrator can configure the WEP rekey period in response to the network load.

The supplicant and the authentication server create a key block at the end of the TLS handshake. MS-MPPE-Send and MS-MPPE-Receive keys are generated from this key block. The authentication server sends these keys to the authenticator with the RADIUS-Access-Accept message for the legitimate supplicant. These keys are sent to the authenticator as an attribute in this RADIUS packet and encrypted with the shared secret between the authentication server and the authenticator (AP). Their privacy is provided by encrypting them with a shared secret and their integrity is assured by using an HMAC_MD5 value between the server and the AP. The authenticator generates broadcast and Unicast keys and encrypts them with the MS-MPPE-Send Key, which is sent by the authentication server. The authentication server then assumes a sleep mode until receipt of a new authentication request.

The authenticator is responsible for generating new keys and distributing these keys to the legitimate and authenticated clients within the rekeying period as configured. MS-MPPE Send and Receive Keys are unreachable to the attackers because they are generated at both the supplicant and the authentication server, and the authentication server sends these keys to the authenticator (AP) encrypted, integrity protected and authenticated at the wired side.

These secret session keys also address session hijacking attacks. A. Mishra and W. Arbaugh claim that the IEEE 802.1X is vulnerable to session hijacking attacks [14]. They propose that after a legitimate client authenticates itself to a WLAN and obtains network connectivity from the authenticator after the message (12) in Figure 27, an adversary can send a 802.11 MAC disassociate management frame using APs MAC address. This causes the supplicant to be disassociated from the AP, but keeps the authenticator state machine at the authenticated state. Then the adversary gains network access using the MAC address of the authenticated supplicant. [6]

This is doable at the EAP message session, as in Figure 27. If the EAP-TLS authentication method is used, the session keys are generated at the end of the TLS

handshake process, both at the server and the supplicant. The adversary cannot send or receive any data frame without having session keys because it cannot decrypt EAPOL-Key messages that the legitimate AP sends. Further, the adversary must boot its card from the authenticated state of its supplicant state machine; otherwise, AP enforces the adversary to reauthenticate. Mishra and Arbaugh also ignore the legitimate supplicant's authentication attempts with the same MAC address at the same broadcast domain. This attack only causes a DoS rather than a session hijacking.

d. Mutual Authentication

In order to prevent man-in-the-middle attacks, there must be mutual authentication:

- Between the supplicant and the authenticator
- Between the authenticator and the authentication server
- Between the supplicant and the authentication server pairs.

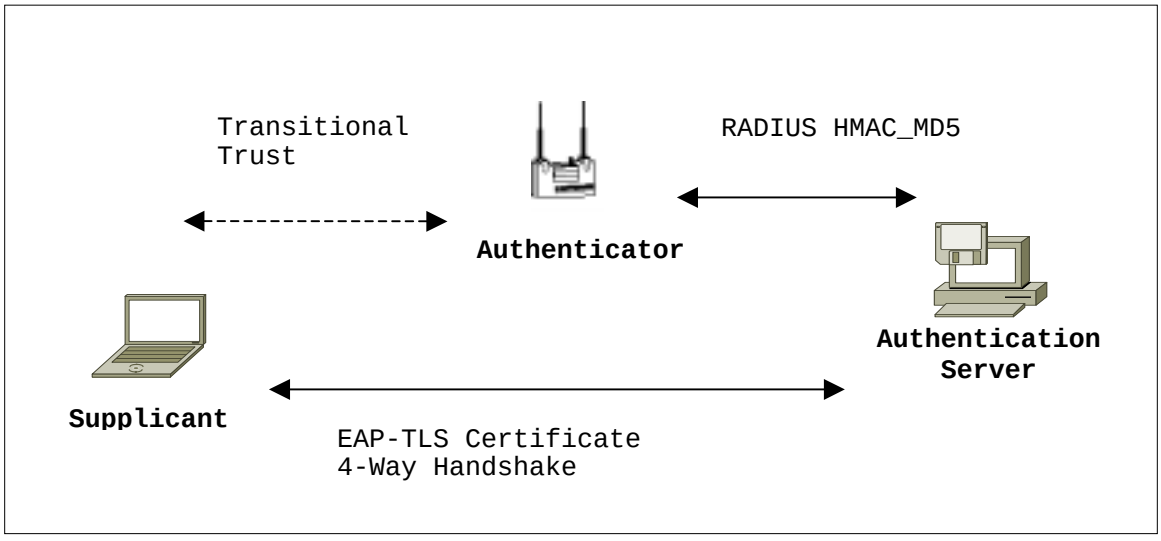


Figure 37. Entity Trust Model

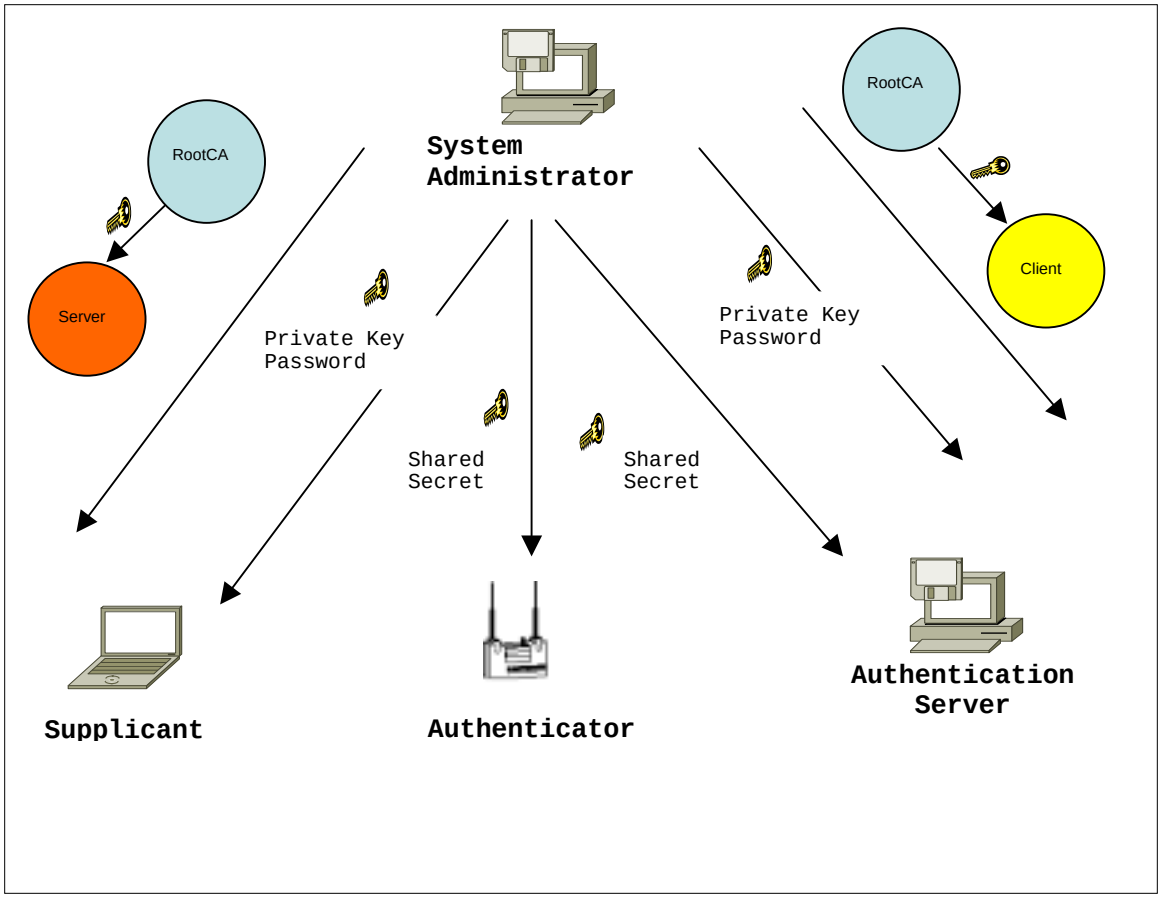


Figure 38. Certificate and Secret Distribution Scheme

The authentication server and authenticator are under the control of the system administrator on the wired side. The shared secret is set manually. This secret provides mutual authentication between the authentication server and the authenticator by generating HMAC_MD5 value for per RADIUS packet.

The system administrator also generates and distributes the rootCA PKC, the client PKC and its private key, and the server PKC and its private key for legitimate users. Since both the supplicant and the authentication server have the same rootCA PKC to verify each other's PKC, which is signed by the rootCA private key, both ends could authenticate each other during the TLS handshake. The system administrator also distributes the passwords to activate private keys at both ends.

As there is a mutual trust between the supplicant and the authentication server; as well as the authentication server and the authenticator, there must be a mutual trust between the supplicant and the authenticator because of the transitional property of trust. The session keys generated after the TLS handshake also support mutual trust between entities.

E. SUMMARY

The IEEE 802.1X security framework could provide mutual authentication between standard entities by using the EAP-TLS authentication method. The authentication method is crucial to enforce mutual authentication and to generate session keys.

This thesis showed that if EAP-TLS is used as an authentication method, it could increase the security level of the WLAN and could be implemented even in military infrastructures. It could address MiM and session-hijacking attacks.

The system or the security administrator is the key player because he must generate certificates and shared secrets, distribute them to legitimate users and entities and configure the devices.

Appendix E presents a full log of the authentication server and the authenticator during the EAP-TLS handshake and session key generation. The IEEE 802.1X message traffic, as shown in Figure 27, could be analyzed using the Log files.

VI. CONCLUSION AND FUTURE WORK

A. CONCLUSION

This thesis examined one of the major LAN technologies, the IEEE 802.11 compatible WLANs. The functions of the physical and medium access control layers are explained briefly. The security vulnerabilities of the current standard are surveyed and possible solutions recommended.

This thesis focused on whether the IEEE 802.1X standard is capable of addressing security vulnerabilities of the current IEEE 802.11 standard. The IEEE 802.11i workgroup consider the IEEE 802.1X standard a key part of its solution to the security vulnerabilities of the IEEE 802.11 standard. They have proposed to use the 802.1X standard to enforce authentication and assist key management. The 802.11i workgroup will finalize their work by the end of 2003.

An open-source test-bed implementing the 802.1X standard was built to evaluate the 802.1X security framework. A denial of service attack was demonstrated to show the utility of the test-bed.

In the test-bed, the EAP-TLS method was chosen as the authentication method between the supplicant and authentication server. This is because EAP-TLS can provide mutual authentication and session key generation. The EAP-TLS method could enhance the security of a WLAN in a military domain but could not address all the vulnerabilities that the IEEE 802.11 specification has, such as DoS attacks via unauthenticated management frames. The EAP-TLS method uses Public Key Certificates (PKC) for authentication. Therefore, a Public Key Infrastructure (PKI) is created for the test-bed. A PKI can be securely implemented in a military domain because the military domain can create its special root certificate authority and distribute the public certificates of various entities securely and trustfully.

The IEEE 802.11i sponsored security framework is progressing and will be released within months. The IEEE 802.11 standard is too weak to provide security and should be integrated with this new security framework when it is released. Before the

integration, a 802.11 WLAN should not be deployed where critical or real time information is processed. It should not be used without upper layer security protection, such as, virtual private networking, firewalls, secure socket layer or virtual local area networking.

B. FUTURE WORK

An open-source test-bed is built to evaluate the 802.1X security standard. The entities: the supplicant, the authenticator and the authentication server form an ideal platform for implementing and evaluating new authentication methods and attack scenarios

New key generation and distribution protocols could be integrated into the entities because the source code is open to manipulate. Management packets, such as deauthentication, disassociation, EAP-Success, EAPOL-Start, or EAPOL-Logoff could be generated to evaluate more types of DoS attacks.

Tools for generating arbitrary management packets should be created by revising the source code of the HostAP. The IEEE 802.11 and 802.1X standard header and source files are well documented. Experience with C or C++ programming and Linux environment is a prerequisite to this work.

APPENDIX A

A. CERTIFICATE GENERATOR CONFIGURATION

1. OpenSSL Configuration File

Certificates generation must be done prior to the 802.1X authentication. WinXP and XSupplicant require different types of OpenSSL versions to generate certificates. WinXP requires OpenSSL 0.9.7beta3, and XSupplicant requires OpenSSL 0.9.8dev. The certificate generator has to install two versions. After two versions of OpenSSL are uncompressed into the */usr/src/* directory, both of them must be installed in different directory trees.

OpenSSL 0.9.8 dev:

./config --prefix=/usr/local/openssl-certgen shared

make

make install

OpenSSL 0.9.7 beta3:

./config --prefix=/usr/local/openssl-Xsupcertgen shared

make

make install

These libraries will be referred in the certificate generation scripts. The configuration file in the */usr/local/opensssl-certgen or openssl-Xsupcertgen /ssl/openssl.conf*, should be modified as desired to save time while generating certificates.

```
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#
# This definition stops the following lines choking if HOME isn't #defined.
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd
# Extra OBJECT IDENTIFIER info:
#oid_file               = $ENV::HOME/.oid
oid_section             = new_oids
# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions            =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)

[ new_oids ]

# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6

####################################################################
[ ca ]
default_ca      = CA_default            # The default ca section

####################################################################
[ CA_default ]

dir             = ./demoCA              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
new_certs_dir = $dir/newcerts                   # default place for new certs.
certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE    = $dir/private/.rand    # private random number file
x509_extensions     = usr_cert              # The extensions to add to the cert
# Comment out the following two lines for the "traditional"
# (and highly broken) format.
```

```
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default            # Certificate field options
# Extension copying option: use with caution.
# copy_extensions = copy
# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crl_extensions        = crl_ext
default_days    = 365                   # how long to certify for
default_crl_days= 30                    # how long before next CRL
default_md      = md5                   # which md to use.
preserve        = no                    # keep passed DN ordering
# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that
policy          = policy_match
# For the CA policy

[ policy_match ]

countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName      = optional
commonName              = supplied
emailAddress            = optional
# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.

[ policy_anything ]

countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName      = optional
commonName              = supplied
emailAddress            = optional

####################################################################
[ req ]
default_bits            = 1024
default_keyfile         = privkey.pem
distinguished_name= req_distinguished_name
attributes              = req_attributes
```

```
x509_extensions        = v3_ca         # The extensions to add to the self signed cert
# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret
# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix   : PrintableString, BMPString.
# utf8only: only UTF8Strings.
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: current versions of Netscape crash on BMPStrings or UTF8Strings
# so use this option with caution!
string_mask = nombstr
# req_extensions = v3_req # The extensions to add to a certificate request

# !!!!!!!!!!Modified by Selcuk OZTURK 02.01.2003!!!!!!!!!!
# The default values should be modified as desired. Common must change # different
certificates. Default values are important values while
# verifying certificates, because they are checked against policy type.

[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = US
countryName_min                         = 2
countryName_max                         = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default             = California

localityName                    = Locality Name (eg, city)
localityName_default            = Monterey

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = NPGS

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default=  World Wide Web Pty Ltd

organizationalUnitName                  = Organizational Unit Name (eg, section)
organizationalUnitName_default          = SAAM

# Common name must be correspondent to the names as defined at the
# RADIUS user database.
```

```
commonName              = Common Name (eg, YOUR name)
commonName_max                  = 64
commonName_default              = WirelessSAAM

emailAddress            = Email Address
emailAddress_max        = 64
emailAddress_default    = hozturk@nps.navy.mil

#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# SET-ex3                       = SET extension number 3

[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min           = 4
challengePassword_max           = 20

unstructuredName        = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.
# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.
basicConstraints=CA:FALSE
# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.
# This is OK for an SSL server.
# nsCertType                    = server
# For an object signing certificate this would be used.
# nsCertType = objsign
# For normal client use this is typical
# nsCertType = client, email
# and for everything including object signing:
# nsCertType = client, email, objsign
# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment
# This will be displayed in Netscape's comment listbox.

nsComment                       = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
# This stuff is for subjectAltName and issuerAltname.
```

```
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move
# Copy subject details
# issuerAltName=issuer:copy
#nsCaRevocationUrl              = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]

# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer:always

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA
```

```
# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crl_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.
# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
```

B. CERTIFICATE GENERATION SCRIPTS

1. Root Certificate Authority Generation Script

```
# CA.root

#!/bin/sh
# SSL system variable must be map to right  OpenSSL version
# To generate certificates for XSupplicant
# SSL=/usr/local/openssl-Xsupcertgen
#
 SSL=/usr/local/openssl-certgen
 export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
 export LD_LIBRARY_PATH=${SSL}/lib
# needed if you need to start from scratch otherwise the CA.pl -newca
# command doesn't copy the new
# private key into the CA directories
 rm -rf demoCA
 echo
"**********************************************************************"
 echo "Creating self-signed private key and certificate"
 echo "When prompted override the default value for the Common Name field"
 echo
"**********************************************************************"
 echo
# Generate a new self-signed certificate.
# After invocation, newreq.pem will contain a private key and
# certificate
# newreq.pem will be used in the next step
```

```
 openssl req -new -x509 -keyout newreq.pem -out newreq.pem -passin pass:whatever -
passout pass:whatever
 echo
"**********************************************************************"
 echo "Creating a new CA hierarchy (used later by the "ca" command) with the
certificate"
 echo "and private key created in the last step"
 echo
"**********************************************************************"
 echo
 echo "newreq.pem" | CA.pl -newca >/dev/null
 echo
"**********************************************************************"
 echo "Creating ROOT CA"
 echo
"**********************************************************************"
 echo
#Create a PKCS#12 file, using the previously created CA certificate/key
# The certificate in demoCA/cacert.pem is the same as in newreq.pem. #Instead of
# using "-in demoCA/cacert.pem" we could have used "-in newreq.pem" and #then
omitted
# the "-inkey newreq.pem" because newreq.pem contains both the private #key and
certificate

 openssl pkcs12 -export -in demoCA/cacert.pem -inkey newreq.pem -out root.p12 -
cacerts -passin pass:whatever -passout pass:whatever

# parse the PKCS#12 file just created and produce a PEM format #certificate and key in
root.pem

 openssl pkcs12 -in root.p12 -out root.pem -passin pass:whatever -passout pass:whatever

# Convert root certificate from PEM format to DER format

openssl x509 -inform PEM -outform DER -in root.pem -out root.der

#Clean Up
 rm -rf newreq.pem
```

2. Server Certificate Generation Script

```
# CA.srv
#!/bin/sh
# SSL system variable must be map to right  OpenSSL version
# To generate certificates for XSupplicant
# SSL=/usr/local/openssl-Xsupcertgen
#
 SSL=/usr/local/openssl-certgen
 export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
 export LD_LIBRARY_PATH=${SSL}/lib
 echo
"**********************************************************************"
 echo "Creating server private key and certificate"
 echo "When prompted enter the server name in the Common Name field."
 echo
"**********************************************************************"
 echo
# Request a new PKCS#10 certificate.
#First, newreq.pem will be overwritten with the new certificate #request

 openssl req -new -keyout newreq.pem -out newreq.pem -passin pass:whatever -passout
pass:whatever

# Sign the certificate request. The policy is defined in the
# openssl.cnf file.
# The request generated in the previous step is specified with the –
# infiles option and
# the output is in newcert.pem
# The -extensions option is necessary to add the OID for the extended
# key for server authentication

# “-extensions xpserver_ext -extfile xpextensions” arguments is
# required for XP supplicant certificates
# xpextension files must be at the same directory as this script is
# run.

 openssl ca -policy policy_anything -out newcert.pem -passin pass:whatever -key
whatever -extensions xpserver_ext -extfile xpextensions -infiles newreq.pem

# Create a PKCS#12 file from the new certificate and its private key
# found in newreq.pem
# and place in file specified on the command line

 openssl pkcs12 -export -in newcert.pem -inkey newreq.pem -out $1.p12 -clcerts -passin
pass:whatever -passout pass:whatever
```

```
# parse the PKCS#12 file just created and produce a PEM format
# certificate and key in certsrv.pem

 openssl pkcs12 -in $1.p12 -out $1.pem -passin pass:whatever -passout pass:whatever

# Convert certificate from PEM format to DER format
 openssl x509 -inform PEM -outform DER -in $1.pem -out $1.der

# Clean Up
 rm -rf newert.pem newreq.pem
```

3. Supplicant Certificate Generation Script

```
#CA.clt
#!/bin/sh
 SSL=/usr/local/openssl-certgen
 export PATH=${SSL}/bin/:${SSL}/ssl/misc:${PATH}
 export LD_LIBRARY_PATH=${SSL}/lib
 echo
"**********************************************************************"
 echo "Creating client private key and certificate"
 echo "When prompted enter the client name in the Common Name field.
This is the same"
 echo " used as the Username in FreeRADIUS"
 echo
"**********************************************************************"
 echo

# Request a new PKCS#10 certificate.
# First, newreq.pem will be overwritten with the new certificate
# request

openssl req -new -keyout newreq.pem -out newreq.pem -passin pass:whatever -passout
pass:whatever

# Sign the certificate request. The policy is defined in the
# openssl.cnf file.
# The request generated in the previous step is specified with the –
# infiles option and
# the output is in newcert.pem

# The -extensions option is necessary to add the OID for the extended
# key for client authentication. “-extensions xpserver_ext -extfile
# xpextensions” arguments is  required for XP supplicant certificates
# xpextension files must be at the same directory as this script is
```

```
# run. This argument must be deleted while generating XSupplicant
# certificates.

 openssl ca -policy policy_anything -out newcert.pem -passin pass:whatever -key
whatever -extensions xpclient_ext -extfile xpextensions -infiles newreq.pem

# Create a PKCS#12 file from the new certificate and its private key
# found in newreq.pem
# and place in file specified on the command line

 openssl pkcs12 -export -in newcert.pem -inkey newreq.pem -out $1.p12 -clcerts -passin
pass:whatever -passout pass:whatever

# parse the PKCS#12 file just created and produce a PEM format
# certificate and key in certclt.pem

 openssl pkcs12 -in $1.p12 -out $1.pem -passin pass:whatever -passout pass:whatever

# Convert certificate from PEM format to DER format

openssl x509 -inform PEM -outform DER -in $1.pem -out $1.der

# clean up
rm -rf newcert newreq.pem
```

4. XP Specific Extension Files

```
[ xpclient_ext]
 extendedKeyUsage = 1.3.6.1.5.5.7.3.2

[ xpserver_ext ]
 extendedKeyUsage = 1.3.6.1.5.5.7.3.1
```

5. Generating Certificates

All certificates: root, server, client could be created after running the root, server and client scripts in the order below.

./CA.root

./CA.srv servername

./CA.clt clientname

The ".p12, .der, .pem" formatted certificates must be seen in the same directory as the scripts are run. *Servername*.pem and root.pem are required for the FreeRADIUS. *Clientname*.p12; root.der are required for WinXP supplicant client and clientname.pem , clientname.der and root.der are required for the XSupplicant client(http:// www.impossiblereflex.com/8021x/eap-tls-HOWTO.htm, http:// www.missl.cs.umd.edu/ wireless/eaptls/)

APPENDIX B

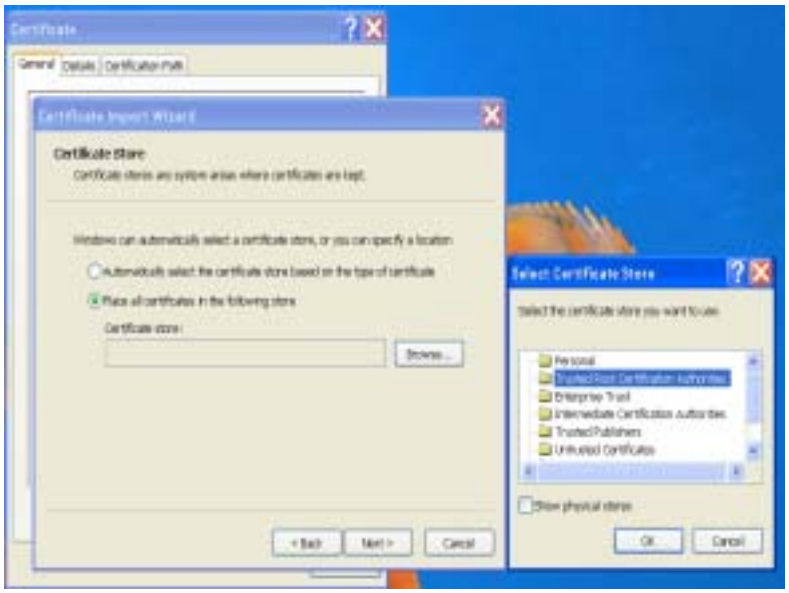
A. WINDOWS XP CERTIFICATE INSTALLATION FOR SUPPLICANT SUPPORT

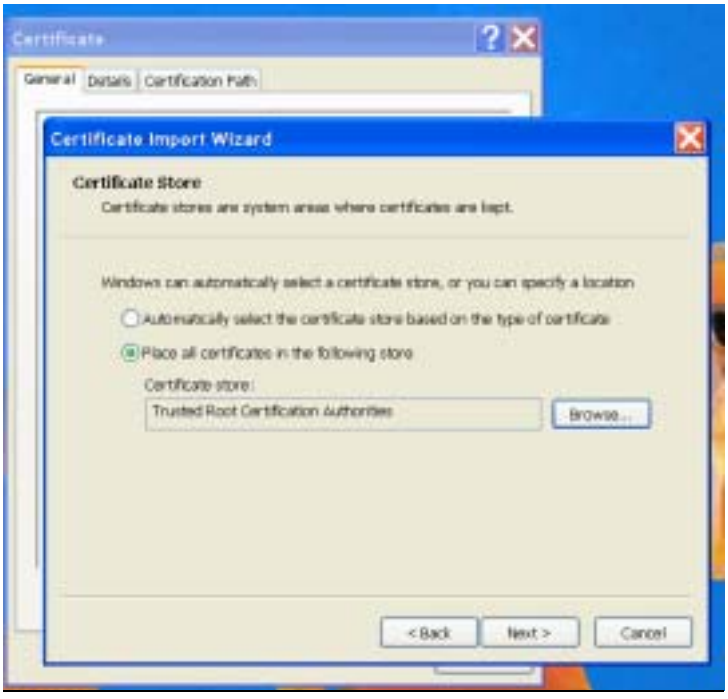
Appendix A explained how to create X.509v3 certificates and private keys which correspond to these public key certificates. Windows XP-Sp1's embedded supplicant requires a client public key certificate, a private key corresponding to this public key and a rootCA public key certificate to verify the server's certificate authentication.

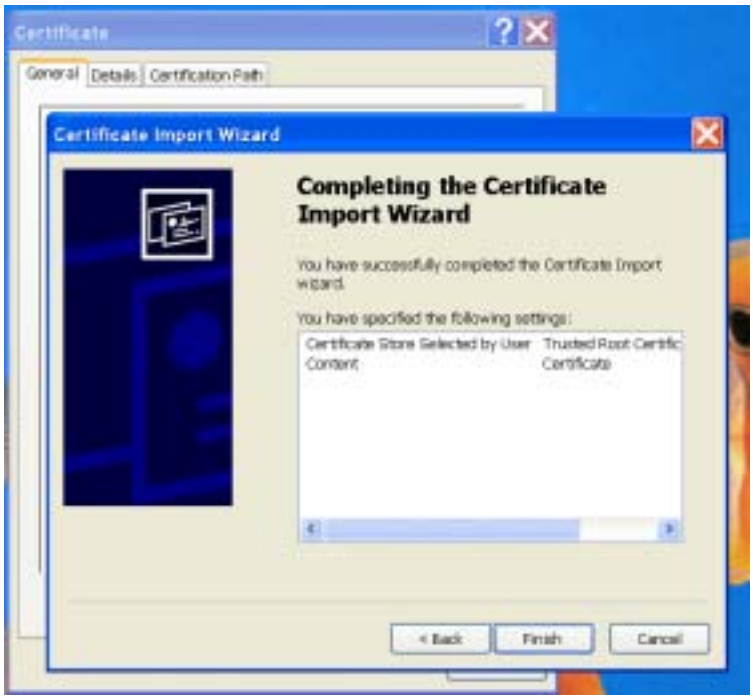
The "hozturk.p12" file containing the public key certificate and private key of the client, and the root.der containing the public key certificate (PKC) of the root certificate authority are required. It has been assumed that both of these files are generated in a trusted environment and there is a strong trust relationship between the client and the root certificate authority. The rootCA' PKC must be installed manually because the trust model requires this.

The screen shot demonstration below will explain how to install these certificates:

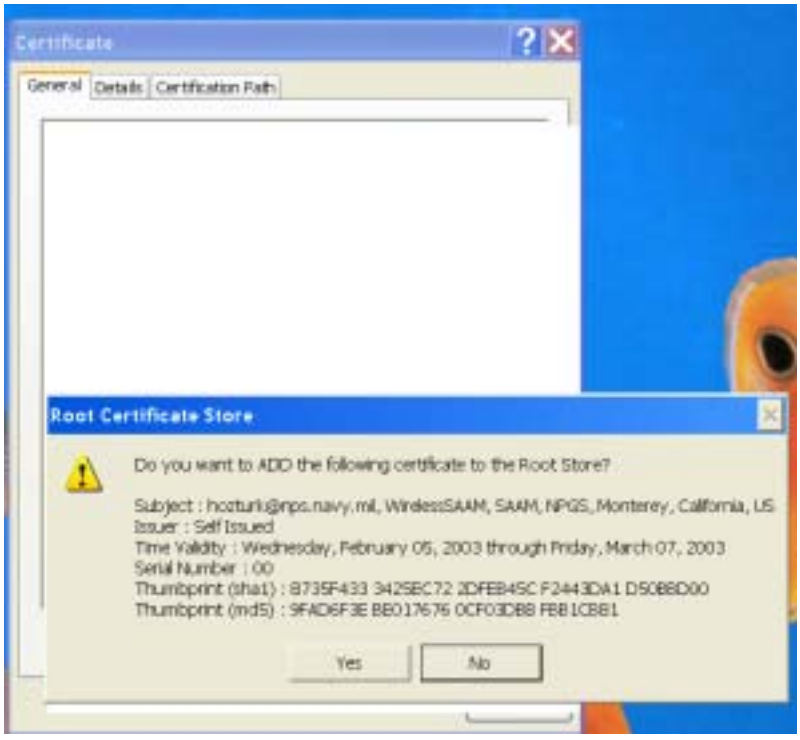
B.A1. First the trusted rootCA must be installed. When the root.der file is double-clicked, the certificate interface comes up. Then the "Install Certificate" button must be clicked.

	B.A2. The “Place all certificate in the following store” radio button must be checked and the “Trusted Root Certification Authorities” must be highlighted. Then click OK.

	B.A3. Click Next to continue

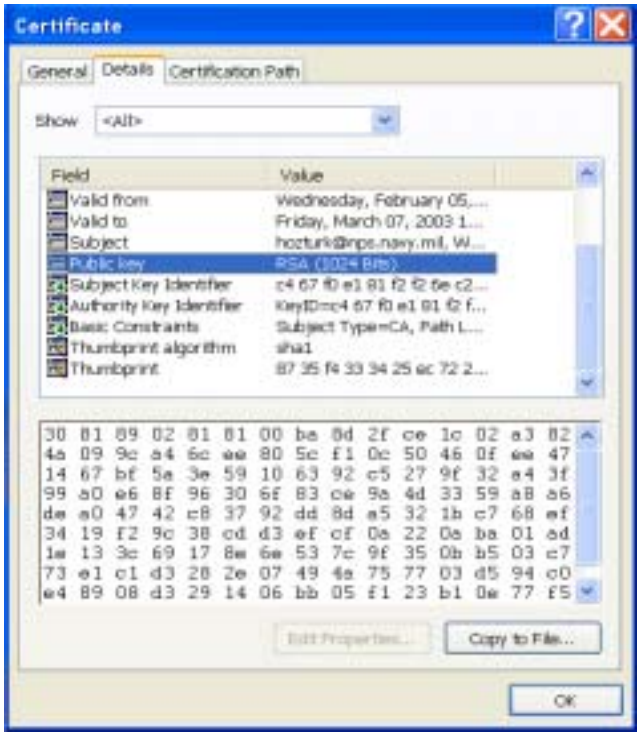


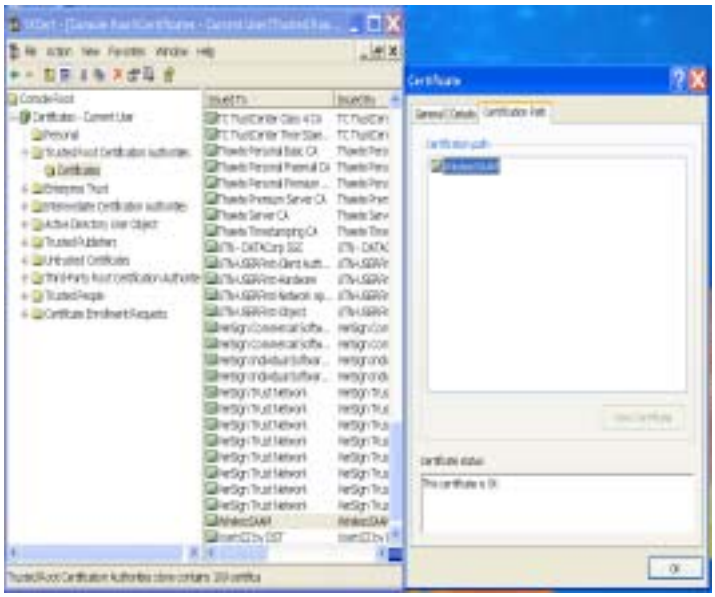
B.A4. Click Finish at the " Completing the Certificate Import " window

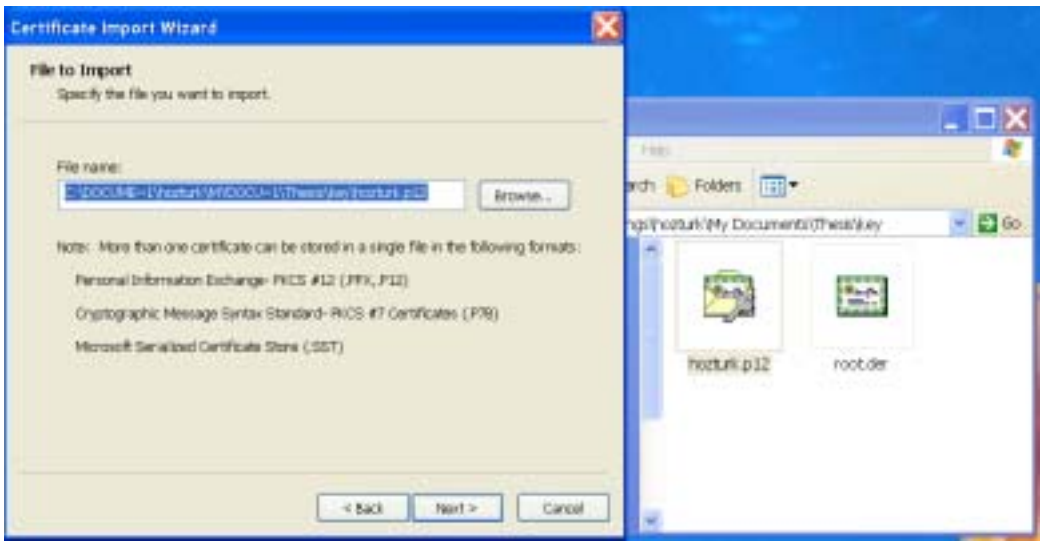


B.A5. Click yes to confirm installation of the self-signed and issued rootCA.

	B.A6. If everything has been right. The system responds with the "successful import window."

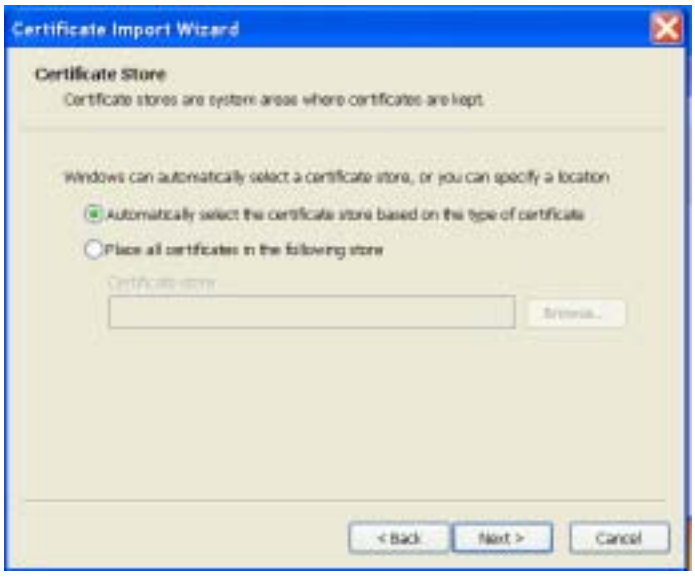
	B.A7. 1024 bits RSA public can be monitored through the "details" tab.

	B.A8. The rootCA PKC can be verified via the MMC console whether it is under Trusted Certificate Authorities or not.

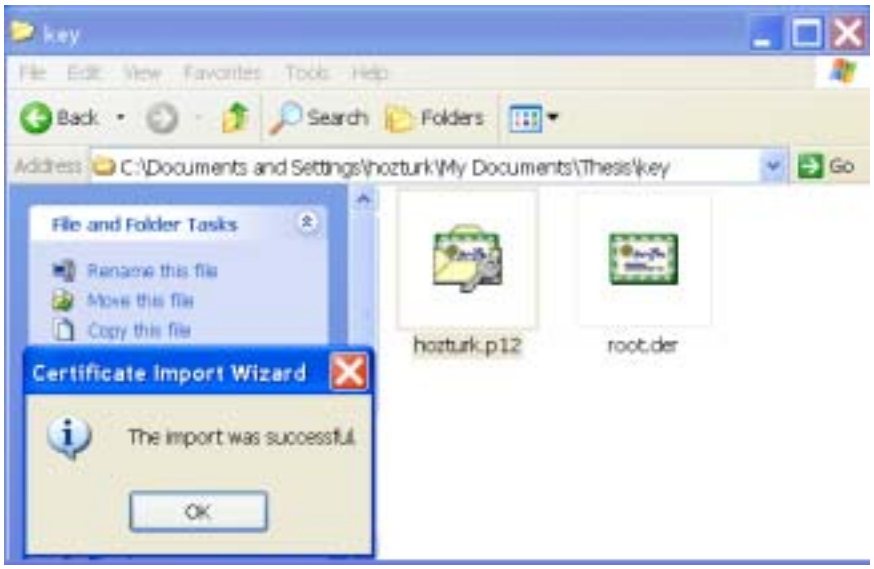


B.A9. When the client.p12 file is double-clicked, the certificate installation wizard appears on the screen. Click next to continue.

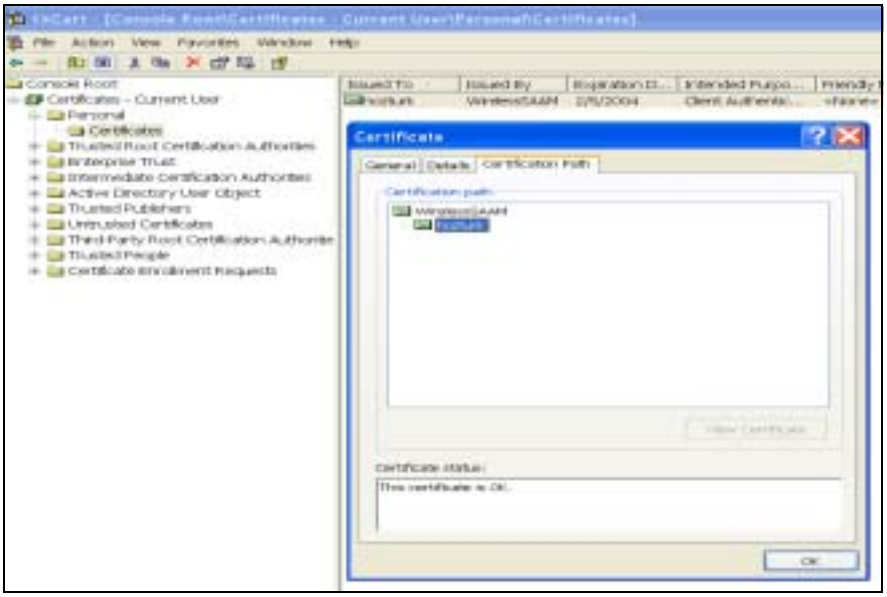
B.A10. The password to decrypt the private key for the client must be entered correctly. The password can be obtained manually from the certificate manager. All the passwords set to "whatever" for test purposes. Then click next to continue.



B.A11 Leave the default values and click next.

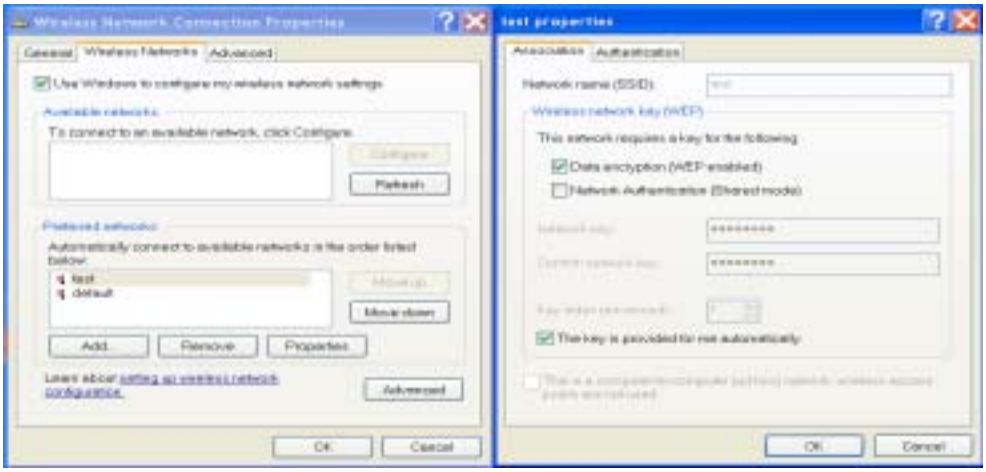


B.A12. If everything has been correctly done, the system will notify the user about the successful import.

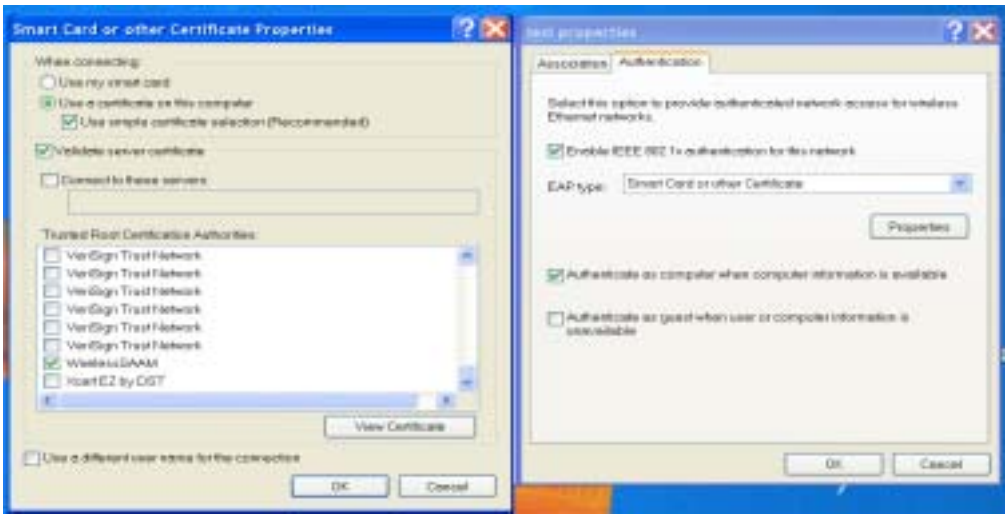


B.A13. The certification path should be verified in the MMC window.

B. WINDOWS XP WIRELESS CLIENT 802.1X CONFIGURATION



B.B1. Thw WEP and dynamic key options must be checked in order to support the dynamic key generation from the authenticator



B.B2 The “Enable IEEE 802.1X authentication for this network” radio button must be checked with the “Smart Card or other Certificates” option. When the “Properties” button is clicked, the “Use a certificate on this computer” radio button must be checked. The “Validate server certificate” radio button must be checked; otherwise, only the client certificate is validated at the server. The server certificate must also be validated to enforce mutual authentication.

C. XSUPPLICANT CONFIGURATION FILE

XSupplicant uses a configuration file to load the certificates automatically. All the required certificates must be copied into the same directory structure.

```
########################################
#/etc/1x/1x.conf
#<networked>:<tag>=<value>
########################################
test:id=hozturk
test:cert=/etc/1x/hozturk.der
test:key=/etc/1x/hozturk.pem
test:root=/etc/1x/root.pem
test:auth=EAP
test:type=wireless
########################################
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

A. CISCO 340 EAP CONFIGURATION SCHEMA

C.A1 The Cisco Aironet 340 has a WEB based configuration tool. Ethernet connection is recommended to configure. The Setup tab and security option must be clicked in order to interact with this window. The Authentication Server and Radio Data Encryption (WEP) functionality must be configured to support the EAP authentication.

C.A2. The protocol version must be set to “Draft10” from the drop down menu. The Authentication server’s IP address and the shared secret, which the authentication server (FreeRADIUS) and the authenticator (Cisco 340) should have, must be set in this menu. Alpha-numeric and more than 12 character secrets are recommended for preventing dictionary attacks. The EAP authentication radio button must be checked for the authentication server services.

C.A3. Since there is no dynamic WEP key generation functionality for Cisco 340 AP, full encryption must be enabled. The WEP privacy is supported with the EAP authentication for this configuration.

B. HOSTAP CONFIGURATION FILE

```
##############################################################################
# it's under /usr/src/hostap/hostapd directory for my setup.
# this configuration file must be modified to enable EAP-TLS
# authentication and dynamic WEP keying support
##############################################################################
# Empty lines and lines starting with # are ignored
# AP net device name (without 'ap' prefix, i.e., wlan0 uses wlan0ap for
# management frames)
interface=wlan0

# Debugging: 0 = no, 1 = minimal, 2 = verbose, 3 = msg dumps
debug=3

# Dump file for state information (on SIGUSR1)
dump_file=/tmp/hostapd.dump
```

```
# Daemonize hostapd process (i.e., fork to background)
daemonize=1


##### IEEE 802.11 related configuration
#######################################

# SSID to be used in IEEE 802.11 management frames
ssid=test

# Station MAC address -based authentication
# 0 = accept unless in deny list
# 1 = deny unless in accept list
# 2 = use external RADIUS server (accept/deny lists are searched first)
macaddr_acl=0

# Accept/deny lists are read from separate files (containing list of
# MAC addresses, one per line). Use absolute path name to make sure
#that the  files can be read on SIGHUP configuration reloads.
#accept_mac_file=/etc/hostapd.accept
#deny_mac_file=/etc/hostapd.deny

# Associate as a station to another AP while still acting as an AP on the same
# channel.
#assoc_ap_addr=00:12:34:56:78:9a


##### IEEE 802.1X (and IEEE 802.1aa/D4) related configuration ##################

# Require IEEE 802.1X authorization
ieee8021x=1

# Use internal minimal EAP Authentication Server for testing IEEE
# 802.1X.This should only be used for testing since it authorizes all
# users that support IEEE 802.1X without any keys or certificates.
minimal_eap=0

# Optional displayable message sent with EAP Request-Identity
eap_message=hello

# WEP rekeying (disabled if key lengths are not set or are set to 0)
# Key lengths for default/broadcast and individual/unicast keys:
# 5 = 40-bit WEP (also known as 64-bit WEP with 40 secret bits)
# 13 = 104-bit WEP (also known as 128-bit WEP with 104 secret bits)
```

```
wep_key_len_broadcast=5
wep_key_len_unicast=5
#Rekeying period in seconds. 0 = do not rekey (i.e., set keys only once)
wep_rekey_period=300

# EAPOL-Key index workaround (set bit7) for WinXP Supplicant (needed
# only if only broadcast keys are used)
eapol_key_index_workaround=1


##### IEEE 802.11f - Inter-Access Point Protocol (IAPP) ######################

# Interface to be used for IAPP broadcast packets
#iapp_interface=eth0


##### RADIUS configuration
#########################################################
# for IEEE 802.1X with external Authentication Server, IEEE 802.11
# authentication with external ACL for MAC addresses, and accounting

# The own IP address of the access point (used as NAS-IP-Address)
own_ip_addr=131.120.8.136

# RADIUS authentication server
auth_server_addr=131.120.8.155
auth_server_port=1812
# This secret must be the same as defined at Radiusd client.conf file
auth_server_shared_secret=whatever

# RADIUS accounting server
#acct_server_addr=127.0.0.1
#acct_server_port=1813
#acct_server_shared_secret=secret
#########################################################################
```

APPENDIX D

A. FREERADIUS EAP-TLS MODULE MAKE FILE

```
# Generated automatically from Makefile.in by configure.
TARGET      = rlm_eap_tls
SRCS        = rlm_eap_tls.c eap_tls.c cb.c tls.c mppe_keys.c

# OpenSSL 0.9.7 or higer is required
RLM_CFLAGS  = $(INCLTDL) -I../.. -I/usr/local/openssl/include
HEADERS     = eap_tls.h
RLM_INSTALL =
RLM_LDFLAGS += -L/usr/local/openssl/lib
RLM_LIBS    +=  -lcrypto -lssl

$(STATIC_OBJS): $(HEADERS)

$(DYNAMIC_OBJS): $(HEADERS)

RLM_DIR=../../
include ${RLM_DIR}../rules.mak
```

B. RADIUSD CONFIGURATION FILE

etc/raddb/radiusd.conf

```
######################################################################
## radiusd.conf          -- FreeRADIUS server configuration file.
##
##      http://www.freeradius.org/
##      $Id: radiusd.conf.in,v 1.121 2002/10/27 15:41:49 aland Exp $
##
#       The location of other config files and
#       logfiles are declared in this file
#       Also general configuration for modules can be done
#       in this file, it is exported through the API to
#       modules that ask for it.
#       The configuration variables defined here are of the form ${foo}
#       They are local to this file, and do not change from request to
#       request.
#       The per-request variables are of the form %{Attribute-Name}, and
#       are taken from the values of the attribute in the incoming
#       request.  See 'doc/variables.txt' for more information.
```

```
prefix = /usr/local
exec_prefix = ${prefix}
sysconfdir = /etc
localstatedir = ${prefix}/var
sbindir = ${exec_prefix}/sbin
logdir = ${localstatedir}/log/radius
raddbdir = ${sysconfdir}/raddb
radacctdir = ${logdir}/radacct

#  Location of config and logfiles.
confdir = ${raddbdir}
run_dir = ${localstatedir}/run/radiusd
#
#  The logging messages for the server are appended to the
#  tail of this file.
#
log_file = ${logdir}/radius.log
#
# libdir: Where to find the rlm_* modules.
#
#   This should be automatically set at configuration time.
#
#   If the server builds and installs, but fails at execution time
#   with an 'undefined symbol' error, then you can use the libdir
#   directive to work around the problem.
#
#   The cause is usually that a library has been installed on your
#   system in a place where the dynamic linker CANNOT find it.  When
#   executing as root (or another user), your personal environment MAY
#   be set up to allow the dynamic linker to find the library.  When
#   executing as a daemon, FreeRADIUS MAY NOT have the same
#   personalized configuration.
#
#   To work around the problem, find out which library contains that
# symbol,
#   and add the directory containing that library to the end of
# 'libdir',
#   with a colon separating the directory names.  NO spaces are
# allowed.
#
#   e.g. libdir = /usr/local/lib:/opt/package/lib
#
#   You can also try setting the LD_LIBRARY_PATH environment variable
#   in a script which starts the server.
```

```
#
#   If that does not work, then you can re-configure and re-build the
#   server to NOT use shared libraries, via:
#
#         ./configure --disable-shared
#         make
#         make install
#
libdir = ${exec_prefix}/lib

#  pidfile: Where to place the PID of the RADIUS server.
#
#  The server may be signalled while it's running by using this
#  file.
#
#  This file is written when ONLY running in daemon mode.
#
#  e.g.:  kill -HUP `cat /var/run/radiusd/radiusd.pid`
#
pidfile = ${run_dir}/radiusd.pid


# user/group: The name (or #number) of the user/group to run radiusd
# as.
#   If these are commented out, the server will run as the user/group
#   that started it.  In order to change to a different user/group, you
#   WILL need to be root ( or have root privleges ) to start the
#   server.
#
#   We STRONGLY recommend that you run the server with as few
#   permissions as possible.  That is, if you're not using shadow
#   passwords, the user and group items below should be set to
#   'nobody'.
#
#    On SCO (ODT 3) use "user = nouser" and "group = nogroup".
#
#  NOTE that some kernels refuse to setgid(group) when the value of
#  (unsigned)group is above 60000; don't use group nobody on these
#  systems!
#
#  On systems with shadow passwords, you might have to set 'group =
#  shadow'for the server to be able to read the shadow password file.
#  If you canauthenticate users while in debug mode, but not in daemon #  mode, it may
bethat the debugging mode server is running as a user
#  that can read thehadow info, and the user listed below can not.
```

```
#
#user = nobody
#group = nobody

#  max_request_time: The maximum time (in seconds) to handle a request.
#
#  Requests which take more time than this to process may be killed,
#  and a REJECT message is returned.
#
#  WARNING: If you notice that requests take a long time to be handled,
#  then this MAY INDICATE a bug in the server, in one of the modules
#  used to handle a request, OR in your local configuration.
#
#  This problem is most often seen when using an SQL database.  If it
#  takesmore than a second or two to receive an answer from the SQL
#  database,then it probably means that you haven't indexed the
#  database.  See your SQL server documentation for more information.

#  Useful range of values: 5 to 120
#
max_request_time = 30

#  delete_blocked_requests: If the request takes MORE THAN
#  'max_request_time'to be handled, then maybe the server should delete
#  it.
#
#  If you're running in threaded, or thread pool mode, this setting
#  should probably be 'no'.  Setting it to 'yes' when using a threaded
#  server MAY cause the server to crash!
#
delete_blocked_requests = no

#  cleanup_delay: The time to wait (in seconds) before cleaning up
#  a reply which was sent to the NAS.
#
#  The RADIUS request is normally cached internally for a short period
#  of time, after the reply is sent to the NAS.  The reply packet may
#  be lost in the network, and the NAS will not see it.  The NAS will
#  then re-send the request, and the server will respond quickly with
#  the cached reply.
#
#  If this value is set too low, then duplicate requests from the NAS
#  MAY NOT be detected, and will instead be handled as seperate
#  requests.
#
```

```
#  If this value is set too high, then the server will cache too many
#  requests, and some new requests may get blocked.  (See
#  'max_requests'.)
#
#  Useful range of values: 2 to 10
#
cleanup_delay = 5

#  max_requests: The maximum number of requests which the server keeps
#  track of.  This should be 256 multiplied by the number of clients.
#  e.g. With 4 clients, this number should be 1024.
#
#  If this number is too low, then when the server becomes busy,
#  it will not respond to any new requests, until the 'cleanup_delay'
#  time has passed, and it has removed the old requests.
#
#  If this number is set too high, then the server will use a bit more
#  memory for no real benefit.
#
#  If you aren't sure what it should be set to, it's better to set it
#  too high than too low.  Setting it to 1000 per client is probably
#  the highest it should be.
#
#  Useful range of values: 256 to infinity
#
max_requests = 1024

#  bind_address:  Make the server listen on a particular IP address,
#   and send replies out from that address.  This directive is most useful for machines with
multiple IP addresses on one interface.
#
#  It can either contain "*", or an IP address, or a fully qualified
#  Internet domain name.  The default is "*"
#
bind_address = *

#  port: Allows you to bind FreeRADIUS to a specific port.
#
#  The default port that most NAS boxes use is 1645, which is
#  historical. RFC 2138 defines 1812 to be the new port.  Many new
# servers and NAS boxes use 1812, which can create interoperability
# problems.
#
#  The port is defined here to be 0 so that the server will pick up
#  the machine's local configuration for the radius port, as defined
```

```
#  in /etc/services.
#
#  If you want to use the default RADIUS port as defined on your
#  server, (usually through 'grep radius /etc/services') set this to 0
#  (zero).
#
#  A port given on the command-line via '-p' over-rides this one.
#
port = 0

#  hostname_lookups: Log the names of clients or just their IP
#  addresses e.g., www.freeradius.org (on) or 206.47.27.232 (off).
#  The default is 'off' because it'd be overall better for the net if
#  people had to knowingly turn this feature on, since enabling it
#  means that each client request will result in AT LEAST one lookup
#  request to the nameserver.
#
#  Turning hostname lookups off also means that the server won't block
#  for 30 seconds, if it sees an IP address which has no name
#  associated with it.
#
#  allowed values: {no, yes}
#
hostname_lookups = no

#  Core dumps are a bad thing.  This should only be set to 'yes'
#  if you're debugging a problem with the server.
#
#  allowed values: {no, yes}
#
allow_core_dumps = no

#  Regular expressions
#
#  These items are set at configure time.  If they're set to "yes",
#  then setting them to "no" turns off regular expression support.
#
#  If they're set to "no" at configure time, then setting them to "yes"
#  WILL NOT WORK.  It will give you an error.
#
regular_expressions    = yes
extended_expressions = yes

#  Log the full User-Name attribute, as it was found in the request.
#
```

```
# allowed values: {no, yes}
#
log_stripped_names = no

#  Log authentication requests to the log file.
#
#  allowed values: {no, yes}
#
log_auth = no

#  Log passwords with the authentication requests.
#  log_auth_badpass  - logs password if it's rejected
#  log_auth_goodpass - logs password if it's correct
#
#  allowed values: {no, yes}
#
log_auth_badpass = no
log_auth_goodpass = no

# usercollide:  Turn "username collision" code on and off.  See the
# "doc/duplicate-users" file
#
usercollide = no

# lower_user / lower_pass:
# Lowercase the username/password "before" or "after"
# attempting to authenticate.
#
# If "before", the server will first modify the request
# and then try to auth the user.  If "after", the server
# will first auth using the values provided by the
# user.  If that fails it will reprocess the request
# after modifying it as you specify below.
#
# This is as close as we can get to case insensitivity.  It is
# the admin's job to ensure that the username on the auth
# db side is *also* lowercase to make this work
#
# Default is 'no' (don't lowercase values)
# Valid values = "before" / "after" / "no"
#
lower_user = no
lower_pass = no

# nospace_user / nospace_pass:
```

```
# Some users like to enter spaces in their username or
# password incorrectly.  To save yourself the tech support
# call, you can eliminate those spaces here:
#
# Default is 'no' (don't remove spaces)
# Valid values = "before" / "after" / "no" (explanation above)
#
nospace_user = no
nospace_pass = no

#  Which program to execute check doing concurrency checks.
checkrad = ${sbindir}/checkrad

# SECURITY CONFIGURATION
#
#  There may be multiple methods of attacking on the server.  This
#  section holds the configuration items which minimize the impact
#  of those attacks
#
security {
        #
        #  max_attributes: The maximum number of attributes
        #  permitted in a RADIUS packet.  Packets which have MORE
        #  than this number of attributes in them will be dropped.
        #
        #  If this number is set too low, then no RADIUS packets
        #  will be accepted.
        #
        #  If this number is set too high, then an attacker may be
        #  able to send a small number of packets which will cause
        #  the server to use all available memory on the machine.
        #
        #  Setting this number to 0 means "allow any number of attributes"
        max_attributes = 200

        #
        #  delayed_reject: When sending an Access-Reject, it can be
        #  delayed for a few seconds.  This may help slow down a DoS
        #  attack.  It also helps to slow down people trying to brute-force
        #  crack a users password.
        #
        #  Setting this number to 0 means "send rejects immediately"
        #
        #  If this number is set higher than 'cleanup_delay', then the
        #  rejects will be sent at 'cleanup_delay' time, when the request
```

```
        #  is deleted from the internal cache of requests.
        #
        #  Useful ranges: 1 to 5
        reject_delay = 1
}

# PROXY CONFIGURATION
#
#  proxy_requests: Turns proxying of RADIUS requests on or off.
#
#  The server has proxying turned on by default.  If your system is NOT
#  set up to proxy requests to another server, then you can turn
#  proxying off here.  This will save a small amount of resources on
#  the server.
#
#  If you have proxying turned off, and your configuration files say
#  to proxy a request, then an error message will be logged.
#
#  To disable proxying, change the "yes" to "no", and comment the
#  $INCLUDE line.
#
#  allowed values: {no, yes}
#
proxy_requests  = yes
$INCLUDE  ${confdir}/proxy.conf



# CLIENTS CONFIGURATION
#
#  Client configuration is defined in "clients.conf".  If you don't
#  use the "clients.conf", you can comment the following.  The use of
#  "clients.conf" is recommended over the old "clients", though both
#  are supported.
#
$INCLUDE  ${confdir}/clients.conf


# SNMP CONFIGURATION
#
#  Snmp configuration is only valid if you enabled SNMP support when
#  you compiled radiusd.
#
$INCLUDE  ${confdir}/snmp.conf
```

```
# THREAD POOL CONFIGURATION
#
#  The thread pool is a long-lived group of threads which
#  take turns (round-robin) handling any incoming requests.
#
#
#  You probably want to have a few spare threads around,
#  so that high-load situations can be handled immediately.  If you
#  don't have any spare threads, then the request handling will
#  be delayed while a new thread is created, and added to the pool.
#
#  You probably don't want too many spare threads around,
#  otherwise they'll be sitting there taking up resources, and
#  not doing anything productive.
#
#  The numbers given below should be adequate for most situations.
#
thread pool {
        #  Number of servers to start initially --- should be a
        #reasonable
        #  ballpark figure.
        start_servers = 5

        #  Limit on the total number of servers running.
        #
        #  If this limit is ever reached, clients will be LOCKED OUT, so
        #it
        #  should NOT BE SET TOO LOW.  It is intended mainly as a brake
        #to
        #  keep a runaway server from taking the system with it as it
        #spirals
        #  down...
        #
        #  You may find that the server is regularly reaching the
        #  'max_servers' number of threads, and that increasing
        #  'max_servers' doesn't seem to make much difference.
        #   If this is the case, then the problem is MOST LIKELY that
        #   your back-end databases are taking too long to respond, and
        #   are preventing the server from responding in a timely manner.
        #   For more information, see 'max_request_time', above.
        #
        max_servers = 32

        #  Server-pool size regulation.  Rather than making you guess
        #  how many servers you need, FreeRADIUS dynamically adapts to
```

```
        #  the load it sees, that is, it tries to maintain enough
        #  servers to handle the current load, plus a few spare
        #  servers to handle transient load spikes.
        #
        #  It does this by periodically checking how many servers are
        #  waiting for a request.  If there are fewer than
        #  min_spare_servers, it creates a new spare.  If there are
        #  more than max_spare_servers, some of the spares die off.
        #  The default values are probably OK for most sites.
        #
        min_spare_servers = 3
        max_spare_servers = 10

        #  There may be memory leaks or resource allocation problems with
        #  the server.  If so, set this value to 300 or so, so that the
        #  resources will be cleaned up periodically.
        #
        #  This should only be necessary if there are serious bugs in the
        #  server which have not yet been fixed.
        #
        #  '0' is a special value meaning 'infinity', or 'the servers
        #never
        #  exit'
        max_requests_per_server = 0
}

# MODULE CONFIGURATION
#
#  The names and configuration of each module is located in this section.
#
#  After the modules are defined here, they may be referred to by name,
#  in other sections of this configuration file.
#
modules {

        # PAP module to authenticate users based on their stored password
        #
        #  Supports multiple encryption schemes
        #  clear: Clear text
        #  crypt: Unix crypt
        #    md5: MD5 ecnryption
        #   sha1: SHA1 encryption.
        #  DEFAULT: crypt
        pap {
                encryption_scheme = crypt
```

```
}

# CHAP module
#
#  To authenticate requests containing a CHAP-Password
#
chap {
        authtype = CHAP
}

# Pluggable Authentication Modules
#
#  For Linux, see:
#       http://www.kernel.org/pub/linux/libs/pam/index.html
#
pam {
        #
        #  The name to use for PAM authentication.
        #  PAM looks in /etc/pam.d/${pam_auth_name}
        #  for it's configuration.  See 'redhat/radiusd-pam'
        #  for a sample PAM configuration file.
        #
        #  Note that any Pam-Auth attribute set in the 'users'
        #  file over-rides this one.
        #
        pam_auth = radiusd
}

# Unix /etc/passwd style authentication
#
#
unix {
        #
        #  Cache /etc/passwd, /etc/shadow, and /etc/group
        #
        #  The default is to cache them.
        #
        #  For FreeBSD, you do NOT want to enable the cache,
        #  as it's password lookups are done via a database, so
        #  set this value to 'no'.
        #
        #  Some systems (e.g. RedHat Linux with pam_pwbd) can
        #  take *seconds* to check a password, if the password
        #  file contains 1000's of entries.  For those systems,
        #  you should set the cache value to 'yes', and set
```

```
        #  the locations of the 'passwd', 'shadow', and 'group'
        #  files, below.
        #
        # allowed values: {no, yes}
        cache = yes

        # Reload the cache every 600 seconds (10mins). 0 to
        #disable.
        cache_reload = 600

        #
        #  Define the locations of the normal passwd, shadow, and
        #  group files.
        #
        #  'shadow' is commented out by default, because not all
        #  systems have shadow passwords.
        #
        #  To force the module to use the system password
        #functions,
        #  instead of reading the files, comment out the 'passwd'
        #  and 'shadow' configuration entries.  This is required
        #  for some systems, like FreeBSD, and Mac OSX.
        #
                passwd = /etc/passwd
                shadow = /etc/shadow
                group = /etc/group


        #
        #  Where the 'wtmp' file is located.
        #  This will be moved to it's own module soon..
        #
        radwtmp = ${logdir}/radwtmp
}
# Extensible Authentication Protocol
#
#  For all EAP related authentications
eap {
        # Invoke the default supported EAP type when
        # EAP-Identity response is received
        default_eap_type = tls

        # Default expiry time to clean the EAP list,
        # It is maintained to co-relate the
        # EAP-response for each EAP-request sent.
```

```
timer_expire     = 60

# Supported EAP-types
#md5 {
#}

## FIXME: EAP-TLS is highly experimental EAP-Type at the
#moment.
#       Please give feedback.
tls {
        # the private key to decrypt privete key of server
        private_key_password = whatever

        # server private key , two directories are
        # created to contain certificates, the /etc/XSUP/
        # directory is used instead of XP1X in test
        # configuration
        private_key_file = /etc/XP1x/radius.pem

#       If Private key & Certificate are located in the
#       same file, then private_key_file & certificate_file
#       must contain the same file name.
        certificate_file = /etc/XP1x/radius.pem


#       Trusted Root CA list, root CA PKC
        CA_file = /etc/XP1x/root.pem

        # These random files must be created to support
        # random values for DH Public Key support
        dh_file = /etc/XP1x/dh
        random_file = /etc/XP1x/random
#
#       This can never exceed MAX_RADIUS_LEN (4096)
#       preferably half the MAX_RADIUS_LEN, to
#       accomodate other attributes in RADIUS packet.
#       On most APs the MAX packet length is configured
#       between 1500 - 1600. In these cases, fragment
#       size should be <= 1024.
        fragment_size = 1024
#
#       include_length is a flag which is by default set to
#       yes If set to yes, Total Length of the message is
#       included in EVERY packet we send.
#       If set to no, Total Length of the message is included
```

```
        #       ONLY in the First packet of a fragment series.
                include_length = yes
        }
}

# Microsoft CHAP authentication
#
#  This module supports SAMBA passwd file authorization
#  and MS-CHAP, MS-CHAPv2 authentication.  However, we recommend
#  using the 'passwd' module, below, as it's more general.
#
#mschap {
        # if given, passwd shows location of
        # SAMBA passwd file
        #       passwd = /etc/smbpasswd
        # please note that smbpasswd authorization in
        # mschap is for compatibility only. It works
        # slow and shouldn't be used.
        # use rlm_passwd module instead in authorize section
        # you can find configuration example for
        # passwd etc_smbpasswd
        # below

        # authtype value, if present, will be used
        # to overwrite (or add) Auth-Type during
        # authorization. Normally should be MS-CHAP
        authtype = MS-CHAP

        # if ignore_password set to yes mschap will
        # ignore password set by any other module during
        # authorization and will always use password file
        #       ignore_password = yes

        # if use_mppe is not set to no mschap will
        # add MS-CHAP-MPPE-Keys for MS-CHAPv1 and
        # MS-MPPE-Recv-Key/MS-MPPE-Send-Key for MS-CHAPv2
        #       use_mppe = no

        # if mppe is enabled require_encryption makes
        # encryption moderate
        #       require_encryption = yes

        # require_strong always requires 128 bit key
        # encryption
        #       require_strong = yes
```

```
        #}

        # Lightweight Directory Access Protocol (LDAP)
        #
        #  This module definition allows you to use LDAP for
        #  authorization and authentication (Auth-Type := LDAP)
        #
        #  See doc/rlm_ldap for description of configuration options
        #  and sample authorize{} and authenticate{} blocks
        #ldap {
                server = "ldap.your.domain"
                # identity = "cn=admin,o=My Org,c=UA"
                # password = mypass
                basedn = "o=My Org,c=UA"
                filter = "(uid=%{Stripped-User-Name:-%{User-Name}})"

                # set this to 'yes' to use TLS encrypted connections
                # to the LDAP database by using the StartTLS extended
                # operation.
                start_tls = no
                # set this to 'yes' to use TLS encrypted connections to the
                # LDAP database by passing the LDAP_OPT_X_TLS_TRY option to
                # the ldap library.
                tls_mode = no
                # default_profile = "cn=radprofile,ou=dialup,o=My Org,c=UA"
                # profile_attribute = "radiusProfileDn"
                access_attr = "dialupAccess"

                # Mapping of RADIUS dictionary attributes to LDAP
                # directory attributes.
                dictionary_mapping = ${raddbdir}/ldap.attrmap

                # ldap_cache_timeout = 120
                # ldap_cache_size = 0
                ldap_connections_number = 5
                # password_header = "{clear}"
                # password_attribute = userPassword
                # groupname_attribute = cn
                # groupmembership_filter =
"(|(&(objectClass=GroupOfNames)(member=%{Ldap-
UserDn}))(&(objectClass=GroupOfUniqueNames)(uniquemember=%{Ldap-
UserDn})))"
                # groupmembership_attribute = radiusGroupName
                timeout = 4
                timelimit = 3
```

```
        net_timeout = 1
        # compare_check_items = yes
        # access_attr_used_for_allow = yes
#}

# passwd module allows to do authorization via any passwd-like
# file and to extract any attributes from these modules
#
# parameters are:
#   filename - path to filename
#   format - format for filename record. This parameters
#          correlates record in the passwd file and RADIUS
#          attributes.
#
#       Field marked as '*' is key field. That is, the parameter
#       with this name from the request is used to search for
#          the record from passwd file
#
#       Field marked as ',' may contain a comma separated list
#          of attributes.
#   authtype - if record found this Auth-Type is used to
#   authenticate
#          user
#   hashsize - hashtable size. If 0 or not specified records are
#not
#          stored in memory and file is red on every request.
#   allowmultiplekeys - if few records for every key are allowed
#   ignorenislike - ignore NIS-related records
#   delimiter - symbol to use as a field separator in passwd
#file,
#          for format ':' symbol is always used. '\0', '\n' are
#     not allowed
#
#passwd etc_smbpasswd {
#        filename = /etc/smbpasswd
#        format = "*User-Name::LM-Password:NT-Password:SMB-Account-
#        CTRL-TEXT::"
#        authtype = MS-CHAP
#        hashsize = 100
#        ignorenislike = no
#        allowmultiplekeys = no
#}

#  Similar configuration, for the /etc/group file. Adds a Group-
#Name
```

```
#  attribute for every group that the user is member of.
#
#passwd etc_group {
#       filename = /etc/group
#       format = "Group-Name:::*,User-Name"
#       hashsize = 50
#       ignorenislike = yes
#       allowmultiplekeys = yes
#       delimiter = ":"
#}

# Realm module, for proxying.
#
#  You can have multiple instances of the realm module to
#  support multiple realm syntaxs at the same time.  The
#  search order is defined the order in the authorize and
#  preacct blocks after the module config block.
#
#  Two config options:
#       format     -  must be 'prefix' or 'suffix'
#       delimiter  -  must be a single character

#  'username@realm'
#
realm suffix {
        format = suffix
        delimiter = "@"
}

#  'realm/username'
#
#  Using this entry, IPASS users have their realm set to "IPASS".
realm realmslash {
        format = prefix
        delimiter = "/"
}

#  'username%realm'
#
realm realmpercent {
        format = suffix
        delimiter = "%"
}

#  rewrite arbitrary packets.  Useful in accounting and
```

```
#authorization.
## FIXME:  This is highly experimental at the moment.  Please
#give  feedback.
#
# The module can also use the Rewrite-Rule attribute. If it is
#set and matches the name of the module instance, then that
# module instance will be the only one which runs.
#
# Also if new_attribute is set to yes then a new attribute will
#be created containing the value replacewith and it will be added
#to searchin (packet, reply or config).
# searchfor,ignore_case and max_matches will be ignored in that
#case.
#
#attr_rewrite sanecallerid {
#         attribute = Called-Station-Id
          # may be "packet", "reply", or "config"
#         searchin = packet
#         searchfor = "[+ ]"
#         replacewith = ""
#         ignore_case = no
#         new_attribute = no
#         max_matches = 10
#         ## If set to yes then the replace string will be appended
#to the original string
#         append = no
#}

# Preprocess the incoming RADIUS request, before handing it off
# to other modules.
#
#  This module processes the 'huntgroups' and 'hints' files.
#  In addition, it re-writes some weird attributes created
#  by some NASes, and converts the attributes into a form which
#  is a little more standard.
#
preprocess {
          huntgroups = ${confdir}/huntgroups
          hints = ${confdir}/hints

          # This hack changes Ascend's wierd port numberings
          # to standard 0-??? port numbers so that the "+" works
          # for IP address assignments.
          with_ascend_hack = no
          ascend_channels_per_line = 23
```

```
        # Windows NT machines often authenticate themselves as
        # NT_DOMAIN\username
        #
        # If this is set to 'yes', then the NT_DOMAIN portion
        # of the user-name is silently discarded.
        with_ntdomain_hack = no

        # Specialix Jetstream 8500 24 port access server.
        #
        # If the user name is 10 characters or longer, a "/"
        # and the excess characters after the 10th are
        # appended to the user name.
        #
        # If you're not running that NAS, you don't need
        # this hack.
        with_specialix_jetstream_hack = no

        # Cisco sends it's VSA attributes with the attribute
        # name *again* in the string, like:
        #
        #   H323-Attribute = "h323-attribute=value".
        #
        # If this configuration item is set to 'yes', then
        # the redundant data in the the attribute text is stripped
        # out.  The result is:
        #
        #  H323-Attribute = "value"
        #
        # If you're not running a Cisco NAS, you don't need
        # this hack.
        with_cisco_vsa_hack = no
}

# Livingston-style 'users' file
#
files {
        usersfile = ${confdir}/users
        acctusersfile = ${confdir}/acct_users

        #  If you want to use the old Cistron 'users' file
        #  with FreeRADIUS, you should change the next line
        #  to 'compat = cistron'.  You can the copy your 'users'
        #  file from Cistron.
        compat = no
```

```
        }

        # See doc/rlm_fastusers before using this
        # module or changing these values.
        #
        fastusers {
                usersfile = ${confdir}/users_fast
                hashsize = 1000
                compat = no
                # Reload the hash every 600 seconds (10mins)
                hash_reload = 600
        }

        # Write a detailed log of all accounting records received.
        #
        detail {
                #  Note that we do NOT use NAS-IP-Address here, as that
                #  attribute MAY BE from the originating NAS, and NOT
                #  from the proxy which actually sent us the request.
                #  The Client-IP-Address attribute is ALWAYS the address
                #  of the client which sent us the request.
                #
                #  The following line creates a new detail file for every
                #  radius client (by IP address or hostname).  In addition,
                #  a new detail file is created every day, so that the
                #detail
                #  file doesn't have to go through a 'log rotation'
                #
                #  If your detail files are large, you may also want to
                #  add a ':%H' (see doc/variables.txt) to the end of it:
                #
                #   ..../detail-%Y%m%d:%H
                #
                #  This will create a new detail file for every hour.
                #
                detailfile = ${radacctdir}/%{Client-IP-Address}/detail-
%Y%m%d
                detailperm = 0600
        }

        # Create a unique accounting session Id, as many NASes re-use
        # or repeat values for Acct-Session-Id, causing no end of
        #confusion.
        #
        #  This module will add a (probably) unique session id
```

```
        #  to an accounting packet based on the attributes listed
        #  below found in the packet.  see doc/rlm_acct_unique
        #
        acct_unique {
                key = "User-Name, Acct-Session-Id, NAS-IP-Address, Client-IP-Address,
NAS-Port-Id"
        }


        # Include another file that has SQL-related stuff in it.
        # This is another file solely because it tends to be big.
        #
        #  The following configuration file is for use with MySQL.
        #
        # For Postgresql, use ${confdir}/postgresql.conf
        # For MS-SQL, use ${confdir}/mssql.conf
        #
        $INCLUDE  ${confdir}/sql.conf

        # Write a 'utmp' style log file, of which users are currently
        # logged in, and where they've logged in from.
        #
        radutmp {
                filename = ${logdir}/radutmp

                # Set the file permissions, as the contents of this file
                # are usually private.
                perm = 0600
                callerid = "yes"
        }

        # "Safe" radutmp - does not contain caller ID, so it can be
        # world-readable, and radwho can work for normal users, without
        # exposing any information that isn't already exposed by who(1).
        #
        # This is another instance of the radutmp module, but it is given
        # then name "sradutmp" to identify it later in the "accounting"
        # section.
        radutmp sradutmp {
                filename = ${logdir}/sradutmp
                perm = 0644
                callerid = "no"
        }

        # attr_filter - filters the attributes received in replies from
```

```
# proxied servers, to make sure we send back to our RADIUS client
# only allowed attributes.
attr_filter {
        attrsfile = ${confdir}/attrs
}

#  This module takes an attribute (count-attribute).
#  It also takes a key, and creates a counter for each unique
#  key.  The count is incremented when accounting packets are
#  received by the server.  The value of the increment depends
#  on the attribute type.
#  If the attribute is Acct-Session-Time or an integer we add the
#  value of the attribute. If it is anything else we increase the
#  counter by one.
#
#The 'reset' parameter defines when the counters are all reset to
#  zero.  It can be hourly, daily, weekly, monthly or never.
#  It can also be user defined. It should be of the form:
#  num[hdwm] where:
#  h: hours, d: days, w: weeks, m: months
#  If the letter is ommited days will be assumed. In example:
#  reset = 10h (reset every 10 hours)
#  reset = 12  (reset every 12 days)
#
#
#  The check-name attribute defines an attribute which will be
#registered
#  by the counter module and can be used to set the maximum
#allowed value
#  for the counter after which the user is rejected.
#  Something like:
#
#  DEFAULT Max-Daily-Session := 36000
#         Fall-Through = 1
#
#  You should add the counter module in the instantiate section
#so that it
#  registers check-name before the files module reads the users
#file.
#
#  If check-name is set and the user is to be rejected then we
#send back a
#  Reply-Message and we log a Failure-Message in the radius.log
#
#  The counter-name can also be used like below:
```

```
#
#  DEFAULT  Daily-Session-Time > 3600, Auth-Type = Reject
#      Reply-Message = "You've used up more than one hour today"
#
# The allowed-servicetype attribute can be used to only take
# into account specific sessions. For example if a user first
# logs in through a login menu and then selects ppp there will
# be two sessions. One for Login-User and one for Framed-User
# service type. We only need to take into account the second one.
#
# The module should be added in the instantiate,authorize and
#accounting sections.
# Make sure that in the authorize section it comes after any
#module which
# sets check-name
#
counter {
        filename = ${raddbdir}/db.counter
        key = User-Name
        count-attribute = Acct-Session-Time
        reset = daily
        counter-name = Daily-Session-Time
        check-name = Max-Daily-Session
        allowed-servicetype = Framed-User
        cache-size = 5000
}

# The "always" module is here for debugging purposes. Each
#instance
# simply returns the same result, always, without doing anything.
always fail {
        rcode = fail
}
always reject {
        rcode = reject
}
always ok {
        rcode = ok
        simulcount = 0
        mpp = no
}

#
#  The 'expression' module current has no configuration.
#
```

```
        expr {
        }

        # ANSI X9.9 token support.  Not included by default.
        # $INCLUDE  ${confdir}/x99.conf

}

# Instantiation
#
# This section orders the loading of the modules.  Modules
# listed here will get loaded BEFORE the later sections like
# authorize, authenticate, etc. get examined.
#
# This section is not strictly needed.  When a section like
# authorize refers to a module, it's automatically loaded and
# initialized.  However, some modules may not be listed in any
# of the following sections, so they can be listed here.
#
# Also, listing modules here ensures that you have control over
# the order in which they are initalized.  If one module needs
# something defined by another module, you can list them in order
# here, and ensure that the configuration will be OK.
#
instantiate {
        #
        #  The expression module doesn't do authorization,
        #  authentication, or accounting.  It only does dynamic
        #  translation, of the form:
        #
        #       Session-Timeout = `%{expr:2 + 3}`
        #
        #  So the module needs to be instantiated, but CANNOT be
        #  listed in any other section.
        #
        expr
}

# Authorization. First preprocess (hints and huntgroups files),
# then realms, and finally look in the "users" file.
# The order of the realm modules will determine the order that
# we try to find a matching realm.
# Make *sure* that 'preprocess' comes before any realm if you
# need to setup hints for the remote radius server
authorize {
```

```
        #
        # The preprocess module takes care of sanitizing some bizarre
        # attributes in the request, and turning them into attributes
        # which are more standard.
        #
        # It takes care of processing the 'raddb/hints' and the
        # 'raddb/huntgroups' files.
        #
        # It also adds a Client-IP-Address attribute to the request.
        #
        preprocess

        #
        # The chap module will set 'Auth-Type := CHAP' if we are
        # handling a CHAP request and Auth-Type has not already been set
        #
        #chap

#       counter
#       attr_filter
        eap
        suffix
        files
#       etc_smbpasswd

        #
        #  Uncomment 'mschap' if the users are logging in with an
        #  MS-CHAP-Challenge attribute for authentication.  The mschap
        #  module will find the MS-CHAP-Challenge attribute, and add
        #  'Auth-Type := MS-CHAP' to the request, which makes it use
        #  the mschap module for authentication.
        #
#       mschap

# The ldap module will set Auth-Type to LDAP if it has not already been set
#       ldap
}


# Authentication.
#
# This section lists which modules are available for authentication.
# Note that it does NOT mean 'try each module in order'.  It means
# that you have to have a module from the 'authorize' section add
# a configuration attribute 'Auth-Type := FOO'.  That authentication
```

```
# type is then used to pick the apropriate module from the list below.
#
# The default Auth-Type is Local.  That is, whatever is not included
# inside an authtype section will be called only if Auth-Type is set to
# Local.
#
# So you should do the following:
# - Set Auth-Type to an appropriate value in the authorize modules
# above.
#   For example, the chap module will set Auth-Type to CHAP, ldap to
#   LDAP, etc.
# - After that create corresponding authtype sections in the
#   authenticate section below and call the appropriate modules.
authenticate {
#       pam
        unix

        # Uncomment it if you want to use ldap for authentication
#       authtype LDAP {
#               ldap
#       }
#       mschap
        eap

        # Most people want CHAP authentication
        # A back-end database listed in the 'authorize' section
        # MUST supply a CLEAR TEXT password.  Encrypted passwords
        # won't work.
        #authtype CHAP {
        #       chap
        #}

        # PAP authentication, when a back-end database listed
        # in the 'authorize' section supplies a password.  The
        # password can be clear-text, or encrypted.
        #authtype PAP {
        #       pap
        #}
}


# Pre-accounting. Look for proxy realm in order of realms, then
# acct_users file, then preprocess (hints file).
preacct {
        preprocess
```

```
        suffix
        files
}


# Accounting. Log to detail file, and to the radwtmp file, and maintain
# radutmp.
accounting {
#       acct_unique
        detail
#       counter
        unix
        radutmp
#       sradutmp
}


# Session database, used for checking Simultaneous-Use. Either the radutmp
# or rlm_sql module can handle this.
# The rlm_sql module is *much* faster
session {
        radutmp
#       sql
}


# Post-Auth. Run the ippool module.
post-auth {
#       main_pool
}
#############################################################################
```

C. CLIENTS CONFIGURATION FILE

etc/raddb/clients.conf

```
#######################################################################
# clients.conf - client configuration directives
#
# This file is included by default.  To disable it, you will need
# to modify the CLIENTS CONFIGURATION section of "radiusd.conf".
#
#######################################################################

#######################################################################
#
#  Definition of a RADIUS client (usually a NAS).
#
#  The information given here over rides anything given in the
#  'clients' file, or in the 'naslist' file.  The configuration here
#  contains all of the information from those two files, and also
#  allows for more configuration items.
#
#  The "shortname" can be used for logging, and the "nastype",
#  "login" and "password" fields are mainly used for checkrad and are
#  optional.
#

#
#  Defines a RADIUS client.  The format is 'client [hostname|ip-
#  address]'
#
#  '127.0.0.1' is another name for 'localhost'.  It is enabled by
#  default, to allow testing of the server after an initial
#  installation.  If you are not going to be permitting RADIUS queries
#  from localhost, we suggest that you delete, or comment out, this
#  entry.
#
client 127.0.0.1 {
        #
        #  The shared secret use to "encrypt" and "sign" packets between
        #  the NAS and FreeRADIUS.  You MUST change this secret from the
        #  default, otherwise it's not a secret any more!
        #
        #  The secret can be any string, up to 32 characters in length.
        #
        secret          = test
```

```
        #
        #  The short name is used as an alias for the fully qualified
        #  domain name, or the IP address.
        #
        shortname       = localhost

        #
        # the following three fields are optional, but may be used by
        # checkrad.pl for simultaneous use checks
        #

        #
        # The nastype tells 'checkrad.pl' which NAS-specific method to
        #  use to query the NAS for simultaneous use.
        #
        #  Permitted NAS types are:
        #
        #       cisco
        #       computone
        #       livingston
        #       max40xx
        #       multitech
        #       netserver
        #       pathras
        #       patton
        #       portslave
        #       tc
        #       usrhiper
        #       other           # for all other types

        #
        nastype     = other     # localhost isn't usually a NAS...

        #
        #  The following two configurations are for future use.
        #  The 'naspasswd' file is currently used to store the NAS
        #  login name and password, which is used by checkrad.pl
        #  when querying the NAS for simultaneous use.
        #
#       login       = !root
#       password    = someadminpas
}

#..This is the IP number of the authenticator for test purposes
#  Shared secret must be same as defined in the authenticator
```

```
#  configuration
client 131.120.8.136 {

        secret   = whatever
        shortname       = test
}

#
#  You can now specify one secret for a network of clients.
#  When a client request comes in, the BEST match is chosen.
#  i.e. The entry from the smallest possible network.
#
#client 192.168.0.0/24 {
#       secret          = testing123-1
#       shortname       = private-network-1
#}
#
#client 192.168.0.0/16 {
#       secret          = testing123-2
#       shortname       = private-network-2
#}


#client 10.10.10.10 {
#       # secret and password are mapped through the "secrets" file.
#       secret      = testing123
#       shortname   = liv1
#      # the following three fields are optional, but may be used by
#      # checkrad.pl for simultaneous usage checks
#       nastype     = livingston
#       login       = !root
#       password    = someadminpas
#}
#############################################################################
```

D. USERS CONFIGURATION FILE

etc/raddb/users.conf

```
#########################################################################
#       Please read the documentation file ../doc/processing_users_file,
#       or 'man 5 users' (after installing the server) for more
#       information.
#
#       This file contains authentication security and configuration
#       information for each user.  Accounting requests are NOT processed
#       through this file.  Instead, see 'acct_users', in this directory.
#
#       The first field is the user's name and can be up to
#       253 characters in length.  This is followed (on the same line)
#       with the list of authentication requirements for that user.  This
#       can include password, comm server name, comm server port number,
#       protocol type (perhaps set by the "hints" file), and huntgroup
#       name (set by the "huntgroups" file).
#
#       If you are not sure why a particular reply is being sent by the
#       server, then run the server in debugging mode (radiusd -X), and
#       you will see which entries in this file are matched.
#
#       When an authentication request is received from the comm server,
#       these values are tested. Only the first match is used unless the
#       "Fall-Through" variable is set to "Yes".
#
#       A special user named "DEFAULT" matches on all usernames.
#       You can have several DEFAULT entries. All entries are processed
#       in the order they appear in this file. The first entry that
#       matches the login-request will stop processing unless you use
#       the Fall-Through variable.
#
#       If you use the database support to turn this file into a .db or
#       .dbm file, the DEFAULT entries _have_ to be at the end of this
#       file and you can't have multiple entries for one username.
#
#       You don't need to specify a password if you set Auth-Type +=
#       System on the list of authentication requirements. The RADIUS
#       server will then check the system password file.
#
#       Indented (with the tab character) lines following the first
#       line indicate the configuration values to be passed back to
#       the comm server to allow the initiation of a user session.
#       This can include things like the PPP configuration values
```

```
#       or the host to log the user onto.
#
#       You can include another `users' file with `$INCLUDE users.other'
#


#
#       For a list of RADIUS attributes, and links to their definitions,
#       see:
#
#       http://www.freeradius.org/rfc/attributes.html
#


#
# Deny access for a specific user.  Note that this entry MUST
# be before any other 'Auth-Type' attribute which results in the user
# being authenticated.
#
# Note that there is NO 'Fall-Through' attribute, so the user will not
# be given any additional resources.
#
#lameuser       Auth-Type := Reject
#               Reply-Message = "Your account has been disabled."

#
# Deny access for a group of users.
#
# Note that there is NO 'Fall-Through' attribute, so the user will not
# be given any additional resources.
#
#DEFAULT    Group == "disabled", Auth-Type := Reject
#               Reply-Message = "Your account has been disabled."
#


#
# This is a complete entry for "steve". Note that there is no Fall-
# Through entry so that no DEFAULT entry will be used, and the user
# will NOT get any attributes in addition to the ones listed here.
#
#steve  Auth-Type := Local, User-Password == "testing"
#       Service-Type = Framed-User,
#       Framed-Protocol = PPP,
#       Framed-IP-Address = 172.16.3.33,
#       Framed-IP-Netmask = 255.255.255.0,
#       Framed-Routing = Broadcast-Listen,
#       Framed-Filter-Id = "std.ppp",
```

```
#       Framed-MTU = 1500,
#       Framed-Compression = Van-Jacobsen-TCP-IP

#
# This is an entry for a user with a space in their name.
# Note the double quotes surrounding the name.
#
#"John Doe"   Auth-Type := Local, User-Password == "hello"
#               Reply-Message = "Hello, %u"

# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# The legitimate user names are defined in this section. Names are must
# be the same as client certificate common names as entered while
# creating client certificates.

"adam-ctl"      Auth-Type := EAP

"hozturk"       Auth-Type := EAP

#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

"test"          Auth-Type :=Local,User-Password=="test"

#
# Dial user back and telnet to the default host for that port
#
#Deg   Auth-Type := Local, User-Password == "ge55ged"
#       Service-Type = Callback-Login-User,
#       Login-IP-Host = 0.0.0.0,
#       Callback-Number = "9,5551212",
#       Login-Service = Telnet,
#       Login-TCP-Port = Telnet

#
# Another complete entry. After the user "dialbk" has logged in, the
# connection will be broken and the user will be dialed back after
# which he will get a connection to the host "timeshare1".
#
#dialbkAuth-Type := Local, User-Password == "callme"
#       Service-Type = Callback-Login-User,
#       Login-IP-Host = timeshare1,
#       Login-Service = PortMaster,
#       Callback-Number = "9,1-800-555-1212"

#
```

```
# user "swilson" will only get a static IP number if he logs in with
# a framed protocol on a terminal server in Alphen (see the huntgroups
# file).
#
# Note that by setting "Fall-Through", other attributes will be added
# from the following DEFAULT entries
#
#swilson        Service-Type == Framed-User, Huntgroup-Name == "alphen"
#               Framed-IP-Address = 192.168.1.65,
#               Fall-Through = Yes

#
# If the user logs in as 'username.shell', then authenticate them
# against the system database, give them shell access, and stop processing
# the rest of the file.
#
#DEFAULT    Suffix == ".shell", Auth-Type := System
#               Service-Type = Login-User,
#               Login-Service = Telnet,
#               Login-IP-Host = your.shell.machine


#
# The rest of this file contains the several DEFAULT entries.
# DEFAULT entries match with all login names.
# Note that DEFAULT entries can also Fall-Through (see first entry).
# A name-value pair from a DEFAULT entry will _NEVER_ override
# an already existing name-value pair.
#

#
# First setup all accounts to be checked against the UNIX /etc/passwd.
# (Unless a password was already given earlier in this file).
#
DEFAULT     Auth-Type := System
        Fall-Through = 1

#
# Set up different IP address pools for the terminal servers.
# Note that the "+" behind the IP address means that this is the "base"
# IP address. The Port-Id (S0, S1 etc) will be added to it.
#
#DEFAULT    Service-Type == Framed-User, Huntgroup-Name == "alphen"
#               Framed-IP-Address = 192.168.1.32+,
#               Fall-Through = Yes
```

```
#DEFAULT    Service-Type == Framed-User, Huntgroup-Name == "delft"
#                   Framed-IP-Address = 192.168.2.32+,
#                   Fall-Through = Yes

#
# Defaults for all framed connections.
#
DEFAULT     Service-Type == Framed-User
        Framed-IP-Address = 255.255.255.254,
        Framed-MTU = 576,
        Service-Type = Framed-User,
        Fall-Through = Yes

#
# Default for PPP: dynamic IP address, PPP mode, VJ-compression.
# NOTE: we do not use Hint = "PPP", since PPP might also be auto-
# detected
#       by the terminal server in which case there may not be a "P"
#       suffix.
#       The terminal server sends "Framed-Protocol = PPP" for auto PPP.
#
DEFAULT     Framed-Protocol == PPP
        Framed-Protocol = PPP,
        Framed-Compression = Van-Jacobson-TCP-IP

#
# Default for CSLIP: dynamic IP address, SLIP mode, VJ-compression.
#
DEFAULT     Hint == "CSLIP"
        Framed-Protocol = SLIP,
        Framed-Compression = Van-Jacobson-TCP-IP

#
# Default for SLIP: dynamic IP address, SLIP mode.
#
DEFAULT     Hint == "SLIP"
        Framed-Protocol = SLIP

#
# Last default: rlogin to our main server.
#
#DEFAULT
#       Service-Type = Login-User,
#       Login-Service = Rlogin,
```

```
#       Login-IP-Host = shellbox.ispdomain.com

# #
# # Last default: shell on the local terminal server.
# #
# DEFAULT
#       Service-Type = Shell-User

# On no match, the user is denied access.
#########################################################################
```

E. RADIUSD RUNNING SCRIPT

FreeRADIUS requires OpennSSL 0.9.7 or greater to be installed. This script enforces FreeRADIUS to load right version of the OpenSSL before running.

/usr/local/sbin/radiusd_run

```
################################

#!/bin/sh -x

LD_LIBRARY_PATH=/usr/local/openssl/lib

LD_PRELOAD=/usr/local/openssl/lib/libcrypto.so

export LD_LIBRARY_PATH LD_PRELOAD

/usr/local/sbin/radiusd -X -A $@

########################################
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E

A. AUTHENTICATION SERVER SUCCESSFUL AUTHENTICATION LOGS

```
############################################################

[root@buffaloberry root]# ./run-radiusd
bash: ./run-radiusd: No such file or directory
[root@buffaloberry root]# cd  /usr/local/sbin
[root@buffaloberry sbin]# ./run-radiusd
+ LD_LIBRARY_PATH=/usr/local/openssl/lib
+ LD_PRELOAD=/usr/local/openssl/lib/libcrypto.so
+ export LD_LIBRARY_PATH LD_PRELOAD
+ /usr/local/sbin/radiusd -X -A
Starting - reading configuration files ...
reread_config:  reading radiusd.conf
Config:   including file: /etc/raddb/proxy.conf
Config:   including file: /etc/raddb/clients.conf
Config:   including file: /etc/raddb/snmp.conf
Config:   including file: /etc/raddb/sql.conf
 main: prefix = "/usr/local"
 main: localstatedir = "/usr/local/var"
 main: logdir = "/usr/local/var/log/radius"
 main: libdir = "/usr/local/lib"
 main: radacctdir = "/usr/local/var/log/radius/radacct"
 main: hostname_lookups = no
 main: max_request_time = 30
 main: cleanup_delay = 5
 main: max_requests = 1024
 main: delete_blocked_requests = 0
 main: port = 0
 main: allow_core_dumps = no
 main: log_stripped_names = no
 main: log_file = "/usr/local/var/log/radius/radius.log"
 main: log_auth = no
 main: log_auth_badpass = no
 main: log_auth_goodpass = no
 main: pidfile = "/usr/local/var/run/radiusd/radiusd.pid"
 main: user = "(null)"
 main: group = "(null)"
 main: usercollide = no
 main: lower_user = "no"
 main: lower_pass = "no"
 main: nospace_user = "no"
```

main: nospace_pass = "no"
main: checkrad = "/usr/local/sbin/checkrad"
main: proxy_requests = yes
proxy: retry_delay = 5
proxy: retry_count = 3
proxy: synchronous = no
proxy: default_fallback = yes
proxy: dead_time = 120
proxy: servers_per_realm = 15
security: max_attributes = 200
security: reject_delay = 1
security: status_server = no
main: debug_level = 0
read_config_files: reading dictionary
read_config_files: reading naslist
read_config_files: reading clients
read_config_files: reading realms
radiusd: entering modules setup
Module: Library search path is /usr/local/lib
Module: Loaded expr
Module: Instantiated expr (expr)
Module: Loaded System
unix: cache = yes
unix: passwd = "/etc/passwd"
unix: shadow = "/etc/shadow"
unix: group = "/etc/group"
unix: radwtmp = "/usr/local/var/log/radius/radwtmp"
unix: usegroup = no
unix: cache_reload = 600
HASH: Reinitializing hash structures and lists for caching...
HASH: user root found in hashtable bucket 11726
HASH: user bin found in hashtable bucket 86651
HASH: user daemon found in hashtable bucket 11668
HASH: user adm found in hashtable bucket 26466
HASH: user lp found in hashtable bucket 54068
HASH: user sync found in hashtable bucket 42895
HASH: user shutdown found in hashtable bucket 71746
HASH: user halt found in hashtable bucket 7481
HASH: user mail found in hashtable bucket 79471
HASH: user news found in hashtable bucket 5375
HASH: user uucp found in hashtable bucket 38541
HASH: user operator found in hashtable bucket 21748
HASH: user games found in hashtable bucket 47657
HASH: user gopher found in hashtable bucket 47357
HASH: user ftp found in hashtable bucket 56226

```
  HASH:  user nobody found in hashtable bucket 99723
  HASH:  user mailnull found in hashtable bucket 78086
  HASH:  user rpm found in hashtable bucket 72383
  HASH:  user xfs found in hashtable bucket 17213
  HASH:  user ntp found in hashtable bucket 21418
  HASH:  user rpc found in hashtable bucket 72373
  HASH:  user rpcuser found in hashtable bucket 552
  HASH:  user nfsnobody found in hashtable bucket 51830
  HASH:  user nscd found in hashtable bucket 36306
  HASH:  user ident found in hashtable bucket 40304
  HASH:  user radvd found in hashtable bucket 66743
  HASH:  user pcap found in hashtable bucket 55326
  HASH:  user hozturk found in hashtable bucket 95961
 HASH:  Stored 28 entries from /etc/passwd
 HASH:  Stored 38 entries from /etc/group
 Module: Instantiated unix (unix)
 Module: Loaded eap
  eap: default_eap_type = "tls"
  eap: timer_expire = 60
  tls: rsa_key_exchange = no
  tls: dh_key_exchange = yes
  tls: rsa_key_length = 512
  tls: dh_key_length = 512
  tls: verify_depth = 0
  tls: CA_path = "(null)"
  tls: pem_file_type = yes
  tls: private_key_file = "/etc/XP1x/radius.pem"
  tls: certificate_file = "/etc/XP1x/radius.pem"
  tls: CA_file = "/etc/XP1x/root.pem"
  tls: private_key_password = "whatever"
  tls: dh_file = "/etc/XP1x/dh"
  tls: random_file = "/etc/XP1x/random"
  tls: fragment_size = 1024
  tls: include_length = yes
 rlm_eap_tls: conf N ctx stored
 rlm_eap: Loaded and initialized the type tls
 Module: Instantiated eap (eap)
 Module: Loaded preprocess
  preprocess: huntgroups = "/etc/raddb/huntgroups"
  preprocess: hints = "/etc/raddb/hints"
  preprocess: with_ascend_hack = no
  preprocess: ascend_channels_per_line = 23
  preprocess: with_ntdomain_hack = no
  preprocess: with_specialix_jetstream_hack = no
  preprocess: with_cisco_vsa_hack = no
```

```
Module: Instantiated preprocess (preprocess)
Module: Loaded realm
 realm: format = "suffix"
 realm: delimiter = "@"
Module: Instantiated realm (suffix)
Module: Loaded files
 files: usersfile = "/etc/raddb/users"
 files: acctusersfile = "/etc/raddb/acct_users"
 files: preproxy_usersfile = "/etc/raddb/preproxy_users"
 files: compat = "no"
Module: Instantiated files (files)
Module: Loaded detail
 detail: detailfile = "/usr/local/var/log/radius/radacct/%{Client-IP-Address}/de
tail-%Y%m%d"
 detail: detailperm = 384
 detail: dirperm = 493
 detail: locking = no
Module: Instantiated detail (detail)
Module: Loaded radutmp
 radutmp: filename = "/usr/local/var/log/radius/radutmp"
 radutmp: username = "%{User-Name}"
 radutmp: perm = 384
 radutmp: callerid = yes
Module: Instantiated radutmp (radutmp)
Listening on IP address *, ports 1812/udp and 1813/udp, with proxy on 1814/udp.
Ready to process requests.
rad_recv: Access-Request packet from host 131.120.8.136:32805, id=4, length=190
        User-Name = "hozturk"
        NAS-IP-Address = 131.120.8.136
        NAS-Port = 1
        Called-Station-Id = "00-05-5D-D9-55-A5:test"
        Calling-Station-Id = "00-05-5D-D9-57-59"
        Framed-MTU = 2304
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 11Mbps 802.11b"
        EAP-Message = "\002\007\000\014\001hozturk"
        State =
0x1ca134a889c3f86d7b9edef57f878e292e334c3e78e44625063ca5945ebf26b691e5d689
        Message-Authenticator = 0x0bf02b6e40f53e591ecf5e5660f2a160
modcall: entering group authorize
  modcall[authorize]: module "preprocess" returns ok
  modcall[authorize]: module "eap" returns updated
    rlm_realm: No '@' in User-Name = "hozturk", looking up realm NULL
    rlm_realm: No such realm NULL
  modcall[authorize]: module "suffix" returns noop
```

```
    users: Matched hozturk at 99
  modcall[authorize]: module "files" returns ok
modcall: group authorize returns updated
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate
rlm_eap: processing type tls
  modcall[authenticate]: module "eap" returns ok
modcall: group authenticate returns ok
Sending Access-Challenge of id 4 to 131.120.8.136:32805
        EAP-Message = "\001\010\000\006\r "
        Message-Authenticator = 0x00000000000000000000000000000000
        State =
0xcdc3039be24294b182ae6bb900b0253033334c3e80bcf0c2cacda2c80599e1f622f519a6
Finished request 4
Going to the next request
rad_recv: Access-Request packet from host 131.120.8.136:32805, id=5, length=258
        User-Name = "hozturk"
        NAS-IP-Address = 131.120.8.136
        NAS-Port = 1
        Called-Station-Id = "00-05-5D-D9-55-A5:test"
        Calling-Station-Id = "00-05-5D-D9-57-59"
        Framed-MTU = 2304
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 11Mbps 802.11b"
        EAP-Message =
"\002\010\000P\r\200\000\000\000F\026\003\001\000A\001\000\000=\003\001>L2\211\
350\267;\003\311\223\250Cg\262\314N\004\254\3335(\302\322\264\274\331\346\223C\
204.\276\000\000\026\000\004\000\005\000\n\000\t\000d\000b\000\003\000\006\000\02
3\000\022\000c\001"
        State =
0xcdc3039be24294b182ae6bb900b0253033334c3e80bcf0c2cacda2c80599e1f622f519a6
        Message-Authenticator = 0xaa457cf62bdcb022049c092659620abf
modcall: entering group authorize
  modcall[authorize]: module "preprocess" returns ok
  modcall[authorize]: module "eap" returns updated
    rlm_realm: No '@' in User-Name = "hozturk", looking up realm NULL
    rlm_realm: No such realm NULL
  modcall[authorize]: module "suffix" returns noop
    users: Matched hozturk at 99
  modcall[authorize]: module "files" returns ok
modcall: group authorize returns updated
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate
```

rlm_eap: Request found, released from the list
rlm_eap: EAP_TYPE - tls
rlm_eap: processing type tls
rlm_eap_tls: Length Included
undefined: before/accept initialization
TLS_accept: before/accept initialization
<<< TLS 1.0 Handshake [length 0041], ClientHello

TLS_accept: SSLv3 read client hello A
>>> TLS 1.0 Handshake [length 004a], ServerHello

TLS_accept: SSLv3 write server hello A
>>> TLS 1.0 Handshake [length 0648], Certificate

TLS_accept: SSLv3 write certificate A
>>> TLS 1.0 Handshake [length 00a1], CertificateRequest

TLS_accept: SSLv3 write certificate request A
TLS_accept: SSLv3 flush data
TLS_accept:error in SSLv3 read client certificate A
rlm_eap_tls: SSL_read Error
 Error code is 2
 SSL Error 2
  modcall[authenticate]: module "eap" returns ok
modcall: group authenticate returns ok
Sending Access-Challenge of id 5 to 131.120.8.136:32805
	EAP-Message =
"\001\t\004\n\r\300\000\000\007B\026\003\001\000J\002\000\000F\003\001>L33\215+\3
41*?\334\211\223\374\212\364s\371u\323\224\225\200vsC=\356\317v\235\312P
\327?}S\211\337\267Nu\311\3713\304n\302\007\326\234)\014(\307\004<\315\032\262\3
24\200\310\027\037\000\004\000\026\003\001\006H\013\000\006D\000\006A\000\002\2
610\202\002\2550\202\002\026\240\003\002\001\002\002\001\0010\r\006\t*\206H\206\3
67\r\001\001\004\005\0000\201\2171\0130\t\006\003U\004\006\023\002US1\0230\021\0
06\003U\004\010\023\nCalifornia1\0210"
	EAP-Message =
"H\206\367\r\001\t\001\026\024hozturk@nps.navy.mil0\036\027\r030206061717Z\027\r
040206061717Z0\201\2171\0130\t\006\003U\004\006\023\002US1\0230\021\006\003U\
004\010\023\nCalifornia1\0210\017\006\003U\004\007\023\010Monterey1\r0\013\006\0
03U\004\n\023\004NPGS1\r0\013\006\003U\004\013\023\004SAAM1\0250\023\006\00
3U\004\003\023\014buffaloberry1#0!\006\t*\206H\206\367\r\001\t\001\026\024hozturk
@nps.navy.mil0\201\2370\r\006\t*\206H\206\367\r\001\001\001\005\000\003\201\215\0
000\201\211\002\201\201\000\266LS\365"
	EAP-Message = "\313\333R\330DU\n\027\261t\345\351q\262\007.
"\241H\030\217k\332\331+C&\360~\336\267|o\314\326\277\307s\325\243\004D\023\23
3\314p\002-

N5l\252\303Q\\y0\347\213yl\337\370s\373\002;&\032@\267\362\224\333\307\027\0141
\027>\r"\233\001\346\242\n\204\336'\337
\321\037\376\3731V\314\024\316\004\022\274\014\240\335L5\275s\002\003\001\000\0
01\243\0270\0250\023\006\003U\035%\004\0140\n\006\010+\006\001\005\005\007\003\
0010\r\006\t*\206H\206\367\r\001\001\004\005\000\003\201\201\000%F\225'\351\025.!r
m\242@\326s\021[\237"
EAP-Message = "}\326\212q'\021\301@\374.\003.#i\301\017\224\005\360:\377-
\344\007b\3617\023<\340\235W\244\333\360g\244\000\003\2120\202\003\2060\202\00
2\357\240\003\002\001\002\002\001\0000\r\006\t*\206H\206\367\r\001\001\004\005\00
00\201\2171\0130\t\006\003U\004\006\023\002US1\0230\021\006\003U\004\010\023\n
California1\0210\017\006\003U\004\007\023\010Monterey1\r0\013\006\003U\004\n\023
\004NPGS1\r0\013\006\003U\004\013\023\004SAAM1\0250\023\006\003U\004\003\02
3\014WirelessSAAM1#0!\006\t*\206H\206\367\r\001\t\001\026\024"
EAP-Message =
"1\0130\t\006\003U\004\006\023\002US1\0230\021\006\003U\004\010\023\nCa"
Message-Authenticator = 0x00000000000000000000000000000000
State =
0x48213789b1d169d3b91e433eefe2c74833334c3ed6b7186ae78d3ab5b8dc7dfbae0b9bee
Finished request 5
Going to the next request
Cleaning up request 3 ID 3 with timestamp 3e4c332e
Waking up in 1 seconds...
rad_recv: Access-Request packet from host 131.120.8.136:32805, id=6, length=184
User-Name = "hozturk"
NAS-IP-Address = 131.120.8.136
NAS-Port = 1
Called-Station-Id = "00-05-5D-D9-55-A5:test"
Calling-Station-Id = "00-05-5D-D9-57-59"
Framed-MTU = 2304
NAS-Port-Type = Wireless-802.11
Connect-Info = "CONNECT 11Mbps 802.11b"
EAP-Message = "\002\t\000\006\r"
State =
0x48213789b1d169d3b91e433eefe2c74833334c3ed6b7186ae78d3ab5b8dc7dfbae0b9bee
Message-Authenticator = 0x157f0728ccd7e66786c92e11b94d8f2e
modcall: entering group authorize
modcall[authorize]: module "preprocess" returns ok
modcall[authorize]: module "eap" returns updated
rlm_realm: No '@' in User-Name = "hozturk", looking up realm NULL
rlm_realm: No such realm NULL
modcall[authorize]: module "suffix" returns noop
users: Matched hozturk at 99
modcall[authorize]: module "files" returns ok
modcall: group authorize returns updated
rad_check_password: Found Auth-Type EAP

auth: type "EAP"
modcall: entering group authenticate
rlm_eap: Request found, released from the list
rlm_eap: EAP_TYPE - tls
rlm_eap: processing type tls
rlm_eap_tls: Received EAP-TLS ACK message
  modcall[authenticate]: module "eap" returns ok
modcall: group authenticate returns ok
Sending Access-Challenge of id 6 to 131.120.8.136:32805
	EAP-Message =
"\001\n\003L\r\200\000\000\007Blifornia1\0210\017\006\003U\004\007\023\010Monter
ey1\r0\013\006\003U\004\n\023\004NPGS1\r0\013\006\003U\004\013\023\004SAAM1\
0250\023\006\003U\004\003\023\014WirelessSAAM1#0!\006\t*\206H\206\367\r\001\t\0
01\026\024hozturk@nps.navy.mil0\201\2370\r\006\t*\206H\206\367\r\001\001\001\005\
000\003\201\215\0000\201\211\002\201\201\000\272\215/\316\034\002\243\202J\t\234\
244l\356\200\\\361\014PF\017\356G\024g\277Z>Y\020c\222\305'\2372\244?\231\240\3
46\217\2260o\203\316\232M3Y\250\246"
	EAP-Message =
"\003\307s\341\301\323(.\007IJuw\003\325\224\300\344\211\010\323)\024\006\273\005\
361#\261\016w\365\002\003\001\000\001\243\201\3570\201\3540\035\006\003U\035\01
6\004\026\004\024\304g\360\341\201\362\362n\302\325(B\367\230\320\010\025\217\35
5(0\201\274\006\003U\035#\004\201\2640\201\261\200\024\304g\360\341\201\362\362
n\302\325(B\367\230\320\010\025\217\355(\241\201\225\244\201\2220\201\2171\0130\t
\006\003U\004\006\023\002US1\0230\021\006\003U\004\010\023\nCalifornia1\0210\01
7\006\003U\004\007\023\010Monte"
	EAP-Message =
"s.navy.mil\202\001\0000\014\006\003U\035\023\004\0050\003\001\001\3770\r\006\t*\2
06H\206\367\r\001\001\004\005\000\003\201\201\0003\261\207\255^\261\247~\217<\30
2\t\276\024\265\004\375h\2646\350>\355\347%J\213bb\262\212\264\030\227-
\2411\032\323n\315\240;e1\344\024r\r+\237\203\216\337\001\0250\377\377\372\374
\235\311
Mj\033\345\356\304^\\D\234\260\2248v\262\010E\007\327wV|\366~M\273d\313\355\0
31\025\244|\014\314\205@#C\3520TAC\037\302\254\270\263\3060l\030\204\2603+\24
5\201\367\361\273\026\003\001\000\241"
	EAP-Message =
"U\004\n\023\004NPGS1\r0\013\006\003U\004\013\023\004SAAM1\0250\023\006\003
U\004\003\023\014WirelessSAAM1#0!\006\t*\206H\206\367\r\001\t\001\026\024hoztur
k@nps.navy.mil\016\000\000"
	Message-Authenticator = 0x00000000000000000000000000000000
	State =
0xea982fac5f81ef74120d4798073aab8f33334c3e4bf29b2bc8ea21a34f3e0ca5bae8405e
Finished request 6
Going to the next request
Waking up in 1 seconds...
rad_recv: Access-Request packet from host 131.120.8.136:32805, id=7, length=1206

User-Name = "hozturk"
NAS-IP-Address = 131.120.8.136
NAS-Port = 1
Called-Station-Id = "00-05-5D-D9-55-A5:test"
Calling-Station-Id = "00-05-5D-D9-57-59"
Framed-MTU = 2304
NAS-Port-Type = Wireless-802.11
Connect-Info = "CONNECT 11Mbps 802.11b"
EAP-Message =
"\002\n\003\374\r\200\000\000\003\362\026\003\001\003\302\013\000\002\262\000\002\
257\000\002\2540\202\002\2500\202\002\021\240\003\002\001\002\002\001\0020\r\006\
t*\206H\206\367\r\001\001\004\005\0000\201\2171\0130\t\006\003U\004\006\023\002U
S1\0230\021\006\003U\004\010\023\nCalifornia1\0210\017\006\003U\004\007\023\010
Monterey1\r0\013\006\003U\004\n\023\004NPGS1\r0\013\006\003U\004\013\023\004S
AAM1\0250\023\006\003U\004\003\023\014WirelessSAAM1#0!\006\t*\206H\206\367\r
\001\t\001\026\024hozturk@nps.navy.m"
EAP-Message =
"\021\006\003U\004\010\023\nCalifornia1\0210\017\006\003U\004\007\023\010Montere
y1\r0\013\006\003U\004\n\023\004NPGS1\r0\013\006\003U\004\013\023\004SAAM1\0
200\016\006\003U\004\003\023\007hozturk1#0!\006\t*\206H\206\367\r\001\t\001\026\0
24hozturk@nps.navy.mil0\201\2370\r\006\t*\206H\206\367\r\001\001\001\005\000\003\
201\215\0000\201\211\002\201\201\000\256\021\223\262\231\335\330\315>\331M\216
P\032\356\232\2352\347@60\320P\224h\004y\020\330\210\004\320\227(\341\206\337\3
60iP\355\333\260C<]KL\251\336\251\030"
EAP-Message =
"\270\021s\270\344"\332\345j\0270\206U\200\3373\034[\314(uHA\017\314\247\002\00
3\001\000\001\243\0270\0250\023\006\003U\035%\004\0140\n\006\010+\006\001\005\0
05\007\003\0020\r\006\t*\206H\206\367\r\001\001\004\005\000\003\201\201\000\266l\3
56\326\271}\2278\026\365\2231\335\341\026\246\r\311r\261P\021ON\215~$\301zVG\0
37~\203a]\246\037\202\350Zz\307m\222\240d\372M\327p\243\200w\336\034\204PL\34
0\377"\016\247\240*v\371Y\034\351:n\357\001\335\375\266\201\366~\373\267\203Q\32
6\362\207\254.\331\200\341c\026\273"
EAP-Message =
"M\370\364\323\233\257\332\273\034\311\017\360b\325\267\324\203\324\311\024\306\
235_0\307N\377a"\303_`&/\232\263\2172\331X\330X\017,\217\307*\344\235m\306\37
7(J\010~\2174\004\266\252\020\255cK\353\270\005t\246\216\237\301\256eRN\237\026
dk\2068C\017\000\000\202\000\200b\273\327\327\204>W\214\013L\230+,,v-
\215\251\020\320FJ\022\354$\225\036\346\021\206\244\036\r?\253U\306g\207X\016c\2
17M\022oML\252W\337\242\036\352\203I\21276\342=\032\222#\376`\225]\3354\3670\
335F<8@5\203q\227{O\333\\B}\363\322~\307\232\234\247"
EAP-Message = "f\241\324\271\0202\350!"
State =
0xea982fac5f81ef74120d4798073aab8f33334c3e4bf29b2bc8ea21a34f3e0ca5bae8405e
Message-Authenticator = 0x73f2c05a0982580584dbc2107c49943c
modcall: entering group authorize

```
  modcall[authorize]: module "preprocess" returns ok
  modcall[authorize]: module "eap" returns updated
    rlm_realm: No '@' in User-Name = "hozturk", looking up realm NULL
    rlm_realm: No such realm NULL
  modcall[authorize]: module "suffix" returns noop
    users: Matched hozturk at 99
  modcall[authorize]: module "files" returns ok
modcall: group authorize returns updated
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate
rlm_eap: Multiple EAP_Message attributes found
rlm_eap: Request found, released from the list
rlm_eap: EAP_TYPE - tls
rlm_eap: processing type tls
rlm_eap_tls:  Length Included
<<< TLS 1.0 Handshake [length 02b6], Certificate

chain-depth=1,
error=0
--> User-Name = hozturk
--> BUF-Name = WirelessSAAM
--> subject =
/C=US/ST=California/L=Monterey/O=NPGS/OU=SAAM/CN=WirelessSAAM/emailAd
dress=hozturk@nps.navy.mil
--> issuer  =
/C=US/ST=California/L=Monterey/O=NPGS/OU=SAAM/CN=WirelessSAAM/emailAd
dress=hozturk@nps.navy.mil
--> verify return:1
chain-depth=0,
error=0
--> User-Name = hozturk
--> BUF-Name = hozturk
--> subject =
/C=US/ST=California/L=Monterey/O=NPGS/OU=SAAM/CN=hozturk/emailAddress=h
ozturk@nps.navy.mil
--> issuer  =
/C=US/ST=California/L=Monterey/O=NPGS/OU=SAAM/CN=WirelessSAAM/emailAd
dress=hozturk@nps.navy.mil
--> verify return:1
TLS_accept: SSLv3 read client certificate A
<<< TLS 1.0 Handshake [length 0086], ClientKeyExchange

TLS_accept: SSLv3 read client key exchange A
<<< TLS 1.0 Handshake [length 0086], CertificateVerify
```

```
TLS_accept: SSLv3 read certificate verify A
<<< TLS 1.0 ChangeCipherSpec [length 0001]

<<< TLS 1.0 Handshake [length 0010], Finished

TLS_accept: SSLv3 read finished A
>>> TLS 1.0 ChangeCipherSpec [length 0001]

TLS_accept: SSLv3 write change cipher spec A
>>> TLS 1.0 Handshake [length 0010], Finished

TLS_accept: SSLv3 write finished A
TLS_accept: SSLv3 flush data
undefined: SSL negotiation finished successfully
rlm_eap_tls: SSL_read Error
 Error code is ..... 2
 SSL Error ..... 2
  modcall[authenticate]: module "eap" returns ok
modcall: group authenticate returns ok
Sending Access-Challenge of id 7 to 131.120.8.136:32805
        EAP-Message =
"\001\013\0005\r\200\000\000\000+\024\003\001\000\001\001\026\003\001\000
Hs\303\235\370\260\254O/\340t\223y\264.\376\305\276\250rP\2631\223\024D\005\357\
033\002\344\361"
        Message-Authenticator = 0x00000000000000000000000000000000
        State =
0x224c775ef0ac027b62140d14570e145533334c3eb4a3a23d64cfd04ed71f250dff870a62
Finished request 7
Going to the next request
Waking up in 1 seconds...
rad_recv: Access-Request packet from host 131.120.8.136:32805, id=8, length=184
        User-Name = "hozturk"
        NAS-IP-Address = 131.120.8.136
        NAS-Port = 1
        Called-Station-Id = "00-05-5D-D9-55-A5:test"
        Calling-Station-Id = "00-05-5D-D9-57-59"
        Framed-MTU = 2304
        NAS-Port-Type = Wireless-802.11
        Connect-Info = "CONNECT 11Mbps 802.11b"
        EAP-Message = "\002\013\000\006\r"
        State =
0x224c775ef0ac027b62140d14570e145533334c3eb4a3a23d64cfd04ed71f250dff870a62
        Message-Authenticator = 0xe77ab297814f807cf03d2218d5b63c54
modcall: entering group authorize
```

```
  modcall[authorize]: module "preprocess" returns ok
  modcall[authorize]: module "eap" returns updated
    rlm_realm: No '@' in User-Name = "hozturk", looking up realm NULL
    rlm_realm: No such realm NULL
  modcall[authorize]: module "suffix" returns noop
    users: Matched hozturk at 99
  modcall[authorize]: module "files" returns ok
modcall: group authorize returns updated
  rad_check_password:  Found Auth-Type EAP
auth: type "EAP"
modcall: entering group authenticate
rlm_eap: Request found, released from the list
rlm_eap: EAP_TYPE - tls
rlm_eap: processing type tls
rlm_eap_tls: Received EAP-TLS ACK message
  modcall[authenticate]: module "eap" returns ok
modcall: group authenticate returns ok
Sending Access-Accept of id 8 to 131.120.8.136:32805
        MS-MPPE-Recv-Key =
0x962f275093783a304806f7c1b0dea448ad9a43198639e108564a685bb17aff9ae84316d1
ddbcfc650934a84f69be9fa6da87
        MS-MPPE-Send-Key =
0x962cc718ebd9115f8fc15b026d064230c5c1ee09af77c8d31e776405dc643ac14724fbd3e
985f0e0bd2c919915aae3dc9945
        EAP-Message = "\003\013\000\004"
        Message-Authenticator = 0x00000000000000000000000000000000
Finished request 8
Going to the next request
Waking up in 1 seconds...
--- Walking the entire request list ---
Waking up in 5 seconds...
--- Walking the entire request list ---
Cleaning up request 4 ID 4 with timestamp 3e4c3333
Cleaning up request 5 ID 5 with timestamp 3e4c3333
Cleaning up request 6 ID 6 with timestamp 3e4c3333
Cleaning up request 7 ID 7 with timestamp 3e4c3333
Cleaning up request 8 ID 8 with timestamp 3e4c3333
Nothing to do.  Sleeping until we see a request.
##############################################################################
```

B. AUTHENTICATOR SUCCESSFUL SUPPLICANT AUTHENTICATION LOG

```
############################################################

[root@selcuk root]# cd /usr/src/hostap/hostapd/
[root@selcuk hostapd]# ./hostapd -d hostapd.conf
Opening raw packet socket for ifindex 4
Using interface wlan0ap with hwaddr 00:05:5d:d9:55:a5 and ssid 'test'
Default WEP key - hexdump(len=5): 61 38 16 16 07
Flushing old station entries
Deauthenticate all stations
Received 30 bytes management frame
  dump: b0 00 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 60 05 00 00
01 00 00 00
MGMT
mgmt::auth
authentication: STA=00:05:5d:d9:57:59 auth_alg=0 auth_transaction=1 status_code=0
Station 00:05:5d:d9:57:59 authentication OK (open-system)
Received 30 bytes management frame
  dump: b2 00 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 50 3d 00 00
02 00 00 00
MGMT (TX callback) ACK
mgmt::auth cb
Station 00:05:5d:d9:57:59 authenticated
Received 40 bytes management frame
  dump: 00 00 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 70 05 11 00
01 00 00 04 74 65 73 74 01 04 82 84 0b 16
MGMT
mgmt::assoc_req
association request: STA=00:05:5d:d9:57:59 capab_info=0x11 listen_interval=1
  old AID 1
Station 00:05:5d:d9:57:59 association OK (aid 1)
Received 36 bytes management frame
  dump: 12 00 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 60 3d 01 00
00 00 01 c0 01 04 82 84 0b 16
MGMT (TX callback) ACK
mgmt::assoc_resp cb
Station 00:05:5d:d9:57:59 associated (aid 1)
Received 37 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 80 05 aa aa
03 00 00 00 88 8e 01 01 00 00 00
DATA
IEEE 802.1X: 5 bytes from 00:05:5d:d9:57:59
   IEEE 802.1X: version=1 type=1 length=0
   ignoring 1 extra octets after IEEE 802.1X packet
```

```
  EAPOL-Start
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state CONNECTING
IEEE 802.1X: Sending EAP Request-Identity to 00:05:5d:d9:57:59 (identifier 7)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 46 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 80 3d aa aa
03 00 00 00 88 8e 01 00 00 0a 01 07 00 0a 01 68 65 6c 6c 6f
DATA (TX callback) ACK
Received 48 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 90 05 aa aa
03 00 00 00 88 8e 01 00 00 0c 02 07 00 0c 01 68 6f 7a 74 75 72 6b
DATA
IEEE 802.1X: 16 bytes from 00:05:5d:d9:57:59
  IEEE 802.1X: version=1 type=0 length=12
  EAP: code=2 identifier=7 length=12 (response)
  EAP Response-Identity
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state AUTHENTICATING
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE

Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=4 length=190
  Attribute 1 (User-Name) length=9
    Value: 'hozturk'
  Attribute 4 (NAS-IP-Address) length=6
    Value: 131.120.8.136
  Attribute 5 (NAS-Port) length=6
    Value: 1
  Attribute 30 (Called-Station-Id) length=24
    Value: '00-05-5D-D9-55-A5:test'
  Attribute 31 (Calling-Station-Id) length=19
    Value: '00-05-5D-D9-57-59'
  Attribute 12 (Framed-MTU) length=6
    Value: 2304
  Attribute 61 (NAS-Port-Type) length=6
    Value: 19
  Attribute 77 (Connect-Info) length=24
    Value: 'CONNECT 11Mbps 802.11b'
  Attribute 79 (EAP-Message) length=14
  Attribute 24 (State) length=38
  Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 84 bytes from authentication server
```

Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=4 length=84
  Attribute 79 (EAP-Message) length=8
  Attribute 80 (Message-Authenticator) length=18
  Attribute 24 (State) length=38
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 8)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 42 bytes management frame
 dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 20 3e aa aa
03 00 00 00 88 8e 01 00 00 06 01 08 00 06 0d 20
DATA (TX callback) ACK
Received 116 bytes management frame
 dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 a0 05 aa aa
03 00 00 00 88 8e 01 00 00 50 02 08 00 50 0d 80 00 00 00 46 16 03 01 00 41 01 00 00 3d
03 01 3e 4c 32 89 e8 b7 3b 03 c9 93 a8 43 67 b2 cc 4e 04 ac db 35 28 c2 d2 b4 bc d9 e6
93 43 84 2e be 00 00 16 00 04 00 05 00 0a 00 09 00 64 00 62 00 03 00 06 00 13 00 12 00
63 01 00
DATA
IEEE 802.1X: 84 bytes from 00:05:5d:d9:57:59
  IEEE 802.1X: version=1 type=0 length=80
  EAP: code=2 identifier=8 length=80 (response)
  EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=5 length=258
  Attribute 1 (User-Name) length=9
    Value: 'hozturk'
  Attribute 4 (NAS-IP-Address) length=6
    Value: 131.120.8.136
  Attribute 5 (NAS-Port) length=6
    Value: 1
  Attribute 30 (Called-Station-Id) length=24
    Value: '00-05-5D-D9-55-A5:test'
  Attribute 31 (Calling-Station-Id) length=19
    Value: '00-05-5D-D9-57-59'
  Attribute 12 (Framed-MTU) length=6
    Value: 2304
  Attribute 61 (NAS-Port-Type) length=6
    Value: 19
  Attribute 77 (Connect-Info) length=24
    Value: 'CONNECT 11Mbps 802.11b'

Attribute 79 (EAP-Message) length=82
Attribute 24 (State) length=38
Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 1120 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=5 length=1120
Attribute 79 (EAP-Message) length=254
Attribute 79 (EAP-Message) length=254
Attribute 79 (EAP-Message) length=254
Attribute 79 (EAP-Message) length=254
Attribute 79 (EAP-Message) length=28
Attribute 80 (Message-Authenticator) length=18
Attribute 24 (State) length=38
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 9)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 1070 bytes management frame
dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 50 3e aa aa
03 00 00 00 88 8e 01 00 04 0a 01 09 04 0a 0d c0 00 00 07 42 16 03 01 00 4a 02 00 00 46
03 01 3e 4c 33 33 8d 2b e1 2a 3f dc 89 93 fc 8a f4 73 f9 75 d3 94 95 80 76 73 43 3d ee cf
76 9d ca 50 20 d7 3f 7d 53 89 df b7 4e 75 c9 f9 33 c4 6e c2 07 d6 9c 29 0c 28 c7 04 3c
cd 1a b2 d4 80 c8 17 1f 00 04 00 16 03 01 06 48 0b 00 06 44 00 06 41 00 02 b1 30 82 02
ad 30 82 02 16 a0 03 02 01 02 02 01 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 30
81 8f 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c
69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30
0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 15
30 13 06 03 55 04 03 13 0c 57 69 72 65 6c 65 73 73 53 41 41 4d 31 23 30 21 06 09 2a 86
48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c
30 1e 17 0d 30 33 30 32 30 36 30 36 31 37 31 37 5a 17 0d 30 34 30 32 30 36 30 36 31 37
31 37 5a 30 81 8f 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13
0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65
79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41
41 4d 31 15 30 13 06 03 55 04 03 13 0c 62 75 66 66 61 6c 6f 62 65 72 72 79 31 23 30 21
06 09 2a 86 48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61 76 79
2e 6d 69 6c 30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00 30 81 89
02 81 81 00 b6 4c 53 f5 2d 7d 49 9e d3 1a 39 6c 6f 34 77 b1 cb db 52 d8 44 55 0a 17 b1
74 e5 e9 71 b2 07 2e 20 22 a1 48 18 8f 6b da d9 2b 43 26 f0 7e de b7 7c 6f cc d6 bf c7 73
d5 a3 04 44 13 9b cc 70 02 2d 4e 35 6c aa c3 51 5c 79 30 e7 8b 79 6c df f8 73 fb 02 3b
26 1a 40 b7 f2 94 db c7 17 0c 31 17 3e 0d 22 9b 01 e6 a2 0a 84 de 27 df 20 d1 1f fe fb 31
56 cc 14 ce 04 12 bc 0c a0 dd 4c 35 bd 73 02 03 01 00 01 a3 17 30 15 30 13 06 03 55 1d
25 04 0c 30 0a 06 08 2b 06 01 05 05 07 03 01 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05

00 03 81 81 00 25 46 95 27 e9 15 2e 21 72 6d a2 40 d6 73 11 5b 9f 59 9a 10 4c af 94 11
87 f9 69 f2 04 2e 6f 9d c8 9e f6 dd b9 43 22 16 65 e9 24 c8 26 28 1e e1 96 d9 fa ae 33 00
90 35 94 ea c7 ad 2a fb d7 5f 30 66 93 8a 89 17 01 cb 3d 85 6d f3 2e d2 4a c0 73 9a 4b
41 6a 0f 5c f7 09 b3 60 7d d6 8a 71 27 11 c1 40 fc 2e 03 2e 23 69 c1 0f 94 05 f0 3a ff 2d
e4 07 62 f1 37 13 3c e0 9d 57 a4 db f0 67 a4 00 03 8a 30 82 03 86 30 82 02 ef a0 03 02
01 02 02 01 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00 30 81 8f 31 0b 30 09 06 03
55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31
11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04
4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 15 30 13 06 03 55 04 03 13
0c 57 69 72 65 6c 65 73 73 53 41 41 4d 31 23 30 21 06 09 2a 86 48 86 f7 0d 01 09 01 16
14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c 30 1e 17 0d 30 33 30 32
30 36 30 36 31 35 31 39 5a 17 0d 30 33 30 33 30 38 30 36 31 35 31 39 5a 30 81 8f 31 0b
30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61
DATA (TX callback) ACK
Received 42 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 b0 05 aa aa
03 00 00 00 88 8e 01 00 00 06 02 09 00 06 0d 00
DATA
IEEE 802.1X: 10 bytes from 00:05:5d:d9:57:59
  IEEE 802.1X: version=1 type=0 length=6
  EAP: code=2 identifier=9 length=6 (response)
  EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=6 length=184
  Attribute 1 (User-Name) length=9
    Value: 'hozturk'
  Attribute 4 (NAS-IP-Address) length=6
    Value: 131.120.8.136
  Attribute 5 (NAS-Port) length=6
    Value: 1
  Attribute 30 (Called-Station-Id) length=24
    Value: '00-05-5D-D9-55-A5:test'
  Attribute 31 (Calling-Station-Id) length=19
    Value: '00-05-5D-D9-57-59'
  Attribute 12 (Framed-MTU) length=6
    Value: 2304
  Attribute 61 (NAS-Port-Type) length=6
    Value: 19
  Attribute 77 (Connect-Info) length=24
    Value: 'CONNECT 11Mbps 802.11b'
  Attribute 79 (EAP-Message) length=8
  Attribute 24 (State) length=38
  Attribute 80 (Message-Authenticator) length=18

IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 928 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=6 length=928
  Attribute 79 (EAP-Message) length=254
  Attribute 79 (EAP-Message) length=254
  Attribute 79 (EAP-Message) length=254
  Attribute 79 (EAP-Message) length=90
  Attribute 80 (Message-Authenticator) length=18
  Attribute 24 (State) length=38
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 10)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 880 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 60 3e aa aa
03 00 00 00 88 8e 01 00 03 4c 01 0a 03 4c 0d 80 00 00 07 42 6c 69 66 6f 72 6e 69 61 31
11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04
4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 15 30 13 06 03 55 04 03 13
0c 57 69 72 65 6c 65 73 73 53 41 41 4d 31 23 30 21 06 09 2a 86 48 86 f7 0d 01 09 01 16
14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c 30 81 9f 30 0d 06 09 2a
86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00 30 81 89 02 81 81 00 ba 8d 2f ce 1c 02 a3 82
4a 09 9c a4 6c ee 80 5c f1 0c 50 46 0f ee 47 14 67 bf 5a 3e 59 10 63 92 c5 27 9f 32 a4 3f
99 a0 e6 8f 96 30 6f 83 ce 9a 4d 33 59 a8 a6 de a0 47 42 c8 37 92 dd 8d a5 32 1b c7 68
ef 34 19 f2 9c 38 cd d3 ef cf 0a 22 0a ba 01 ad 1e 13 3c 69 17 8e 6e 53 7c 9f 35 0b b5 03
c7 73 e1 c1 d3 28 2e 07 49 4a 75 77 03 d5 94 c0 e4 89 08 d3 29 14 06 bb 05 f1 23 b1 0e
77 f5 02 03 01 00 01 a3 81 ef 30 81 ec 30 1d 06 03 55 1d 0e 04 16 04 14 c4 67 f0 e1 81
f2 f2 6e c2 d5 28 42 f7 98 d0 08 15 8f ed 28 30 81 bc 06 03 55 1d 23 04 81 b4 30 81 b1
80 14 c4 67 f0 e1 81 f2 f2 6e c2 d5 28 42 f7 98 d0 08 15 8f ed 28 a1 81 95 a4 81 92 30
81 8f 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04 08 13 0a 43 61 6c
69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65 72 65 79 31 0d 30
0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04 53 41 41 4d 31 15
30 13 06 03 55 04 03 13 0c 57 69 72 65 6c 65 73 73 53 41 41 4d 31 23 30 21 06 09 2a 86
48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61 76 79 2e 6d 69 6c
82 01 00 30 0c 06 03 55 1d 13 04 05 30 03 01 01 ff 30 0d 06 09 2a 86 48 86 f7 0d 01 01
04 05 00 03 81 81 00 33 b1 87 ad 5e b1 a7 7e 8f 3c c2 09 be 14 b5 04 fd 68 b4 36 e8 3e
ed e7 25 4a 8b 62 62 b2 8a b4 18 97 2d a1 31 1a d3 6e cd a0 3b 65 31 e4 14 72 0d 2b 9f
83 8e df 01 15 30 ff ff fa fc 20 9d c9 20 4d 6a 1b e5 ee c4 5e 7f 5c 44 9c b0 94 38 76 b2
08 45 07 d7 77 56 7c f6 7e 4d bb 64 cb ed 19 15 a4 7c 0c cc 85 40 23 43 ea 30 54 41 43
1f c2 ac b8 b3 c6 30 6c 18 84 b0 33 2b a5 81 f7 f1 bb 16 03 01 00 a1 0d 00 00 99 02 01
02 00 94 00 92 30 81 8f 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06 03 55 04
08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f 6e 74 65
72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04 0b 13 04

53 41 41 4d 31 15 30 13 06 03 55 04 03 13 0c 57 69 72 65 6c 65 73 73 53 41 41 4d 31 23
30 21 06 09 2a 86 48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e 6e 61
76 79 2e 6d 69 6c 0e 00 00 00
DATA (TX callback) ACK
Received 1056 bytes management frame
  dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 c0 05 aa aa
03 00 00 00 88 8e 01 00 03 fc 02 0a 03 fc 0d 80 00 00 03 f2 16 03 01 03 c2 0b 00 02 b2
00 02 af 00 02 ac 30 82 02 a8 30 82 02 11 a0 03 02 01 02 02 01 02 30 0d 06 09 2a 86 48
86 f7 0d 01 01 04 05 00 30 81 8f 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 13 30 11 06
03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07 13 08 4d 6f
6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b 06 03 55 04
0b 13 04 53 41 41 4d 31 15 30 13 06 03 55 04 03 13 0c 57 69 72 65 6c 65 73 73 53 41 41
4d 31 23 30 21 06 09 2a 86 48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73
2e 6e 61 76 79 2e 6d 69 6c 30 1e 17 0d 30 33 30 32 30 36 30 36 31 38 31 32 5a 17 0d 30
34 30 32 30 36 30 36 31 38 31 32 5a 30 81 8a 31 0b 30 09 06 03 55 04 06 13 02 55 53 31
13 30 11 06 03 55 04 08 13 0a 43 61 6c 69 66 6f 72 6e 69 61 31 11 30 0f 06 03 55 04 07
13 08 4d 6f 6e 74 65 72 65 79 31 0d 30 0b 06 03 55 04 0a 13 04 4e 50 47 53 31 0d 30 0b
06 03 55 04 0b 13 04 53 41 41 4d 31 10 30 0e 06 03 55 04 03 13 07 68 6f 7a 74 75 72 6b
31 23 30 21 06 09 2a 86 48 86 f7 0d 01 09 01 16 14 68 6f 7a 74 75 72 6b 40 6e 70 73 2e
6e 61 76 79 2e 6d 69 6c 30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d
00 30 81 89 02 81 81 00 ae 11 93 b2 99 dd d8 cd 3e d9 4d 8e 50 1a ee 9a 9d 32 e7 40 36
30 d0 50 94 68 04 79 10 d8 88 04 d0 97 28 e1 86 df f0 69 50 ed db b0 43 3c 5d 4b 4c a9
de a9 18 f6 4c d2 31 67 5a da 41 5a d0 a8 ad c4 38 f0 1b b3 f3 6e c6 34 49 a6 43 b9 a0
d6 58 e3 90 41 b2 77 8f 49 9c 4b fb 0f 23 2c 74 01 70 97 ed 2b e6 38 b8 11 73 b8 e4 22
da e5 6a 17 30 86 55 80 df 33 1c 5b cc 28 75 48 41 0f cc a7 02 03 01 00 01 a3 17 30 15
30 13 06 03 55 1d 25 04 0c 30 0a 06 08 2b 06 01 05 05 07 03 02 30 0d 06 09 2a 86 48 86
f7 0d 01 01 04 05 00 03 81 81 00 b6 6c ee d6 b9 7d 97 38 16 f5 93 31 dd e1 16 a6 0d c9
72 b1 50 11 4f 4e 8d 7e 24 c1 7a 56 47 1f 7e 83 61 5d a6 1f 82 e8 5a 7a 7f c7 6d 92 a0 64
fa 4d d7 70 a3 80 77 de 1c 84 50 4c e0 ff 27 0e a7 a0 2a 76 f9 59 1c e9 3a 6e ef 01 dd fd
b6 81 f6 7e fb b7 83 51 d6 f2 87 ac 2e d9 80 e1 63 16 bb 8f f7 73 c1 d9 cd 31 fa cc 4a 3d
02 ac 38 b3 14 39 04 ba b2 ed 5a 2a 4a b3 0e 9d 4b 49 cd 78 10 00 00 82 00 80 00 62 31
8f 08 c3 ea c8 44 62 12 7b d6 3e a7 c5 79 9a f5 e5 90 15 7c 67 c2 f6 e1 c5 a1 b9 02 5a a6
1b c4 c1 be 95 a8 cf af da ef 66 4d f8 f4 d3 9b af da bb 1c c9 0f f0 62 d5 b7 d4 83 d4 c9
14 c6 9d 5f 30 c7 4e ff 61 22 c3 5f 60 26 2f 9a b3 8f 32 d9 58 d8 58 0f 2c 8f c7 2a e4 9d
6d c6 ff 28 4a 08 7e 8f 34 04 b6 aa 10 ad 63 4b eb b8 05 74 a6 8e 9f c1 ae 65 52 4e 9f 16
64 6b 86 38 43 0f 00 00 82 00 80 62 bb d7 d7 84 3e 57 8c 0b 4c 98 2b 2c 2c 76 2d 8d a9
10 d0 46 4a 12 ec 24 95 1e e6 11 86 a4 1e 0d 3f ab 55 c6 67 87 58 0e 63 8f 4d 12 6f 4d
4c aa 57 df a2 1e ea 83 49 8a 37 36 e2 3d 1a 92 23 fe 60 95 5d dd 34 f7 30 dd 46 3c 38
40 35 83 71 97 7b 4f db 5c 42 7d f3 d2 7e c7 9a 9c a7 aa 22 2a 76 c6 91 5f 82 a8 75 6e
95 33 97 a9 4a a7 93 c3 d0 47 aa 89 4c 57 d7 b6 c0 7e 6a f9 8e bb cc 14 03 01 00 01 01
16 03 01 00 20 71 b0 85 18 2d 47 9a 35 ee 09 60 b6 ac 4d 67 a6 cc ac 36 cc 1d 4f 9c dc
66 a1 d4 b9 10 32 e8 21
DATA
IEEE 802.1X: 1024 bytes from 00:05:5d:d9:57:59
  IEEE 802.1X: version=1 type=0 length=1020

```
  EAP: code=2 identifier=10 length=1020 (response)
  EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=7 length=1206
  Attribute 1 (User-Name) length=9
    Value: 'hozturk'
  Attribute 4 (NAS-IP-Address) length=6
    Value: 131.120.8.136
  Attribute 5 (NAS-Port) length=6
    Value: 1
  Attribute 30 (Called-Station-Id) length=24
    Value: '00-05-5D-D9-55-A5:test'
  Attribute 31 (Calling-Station-Id) length=19
    Value: '00-05-5D-D9-57-59'
  Attribute 12 (Framed-MTU) length=6
    Value: 2304
  Attribute 61 (NAS-Port-Type) length=6
    Value: 19
  Attribute 77 (Connect-Info) length=24
    Value: 'CONNECT 11Mbps 802.11b'
  Attribute 79 (EAP-Message) length=255
  Attribute 79 (EAP-Message) length=255
  Attribute 79 (EAP-Message) length=255
  Attribute 79 (EAP-Message) length=255
  Attribute 79 (EAP-Message) length=10
  Attribute 24 (State) length=38
  Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 Port Timers TICK (timers: 29 0 3599 29)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 131 bytes from authentication server
Received RADIUS message
RADIUS message: code=11 (Access-Challenge) identifier=7 length=131
  Attribute 79 (EAP-Message) length=55
  Attribute 80 (Message-Authenticator) length=18
  Attribute 24 (State) length=38
RADIUS packet matching with station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state REQUEST
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 11)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 89 bytes management frame
```

dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 c0 3e aa aa
03 00 00 00 88 8e 01 00 00 35 01 0b 00 35 0d 80 00 00 00 2b 14 03 01 00 01 01 16 03 01
00 20 48 73 c3 9d f8 b0 ac 4f 2f e0 74 93 79 b4 2e fe c5 be a8 72 50 b3 31 93 14 44 05 ef
1b 02 e4 f1
DATA (TX callback) ACK
Received 42 bytes management frame
dump: 08 01 02 01 00 05 5d d9 55 a5 00 05 5d d9 57 59 00 05 5d d9 55 a5 d0 05 aa aa
03 00 00 00 88 8e 01 00 00 06 02 0b 00 06 0d 00
DATA
IEEE 802.1X: 10 bytes from 00:05:5d:d9:57:59
IEEE 802.1X: version=1 type=0 length=6

EAP: code=2 identifier=11 length=6 (response)
EAP Response-TLS
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state RESPONSE
Encapsulating EAP message into a RADIUS packet
Sending RADIUS message to authentication server
RADIUS message: code=1 (Access-Request) identifier=8 length=184
Attribute 1 (User-Name) length=9
Value: 'hozturk'
Attribute 4 (NAS-IP-Address) length=6
Value: 131.120.8.136
Attribute 5 (NAS-Port) length=6
Value: 1
Attribute 30 (Called-Station-Id) length=24
Value: '00-05-5D-D9-55-A5:test'
Attribute 31 (Calling-Station-Id) length=19
Value: '00-05-5D-D9-57-59'
Attribute 12 (Framed-MTU) length=6
Value: 2304
Attribute 61 (NAS-Port-Type) length=6
Value: 19
Attribute 77 (Connect-Info) length=24
Value: 'CONNECT 11Mbps 802.11b'
Attribute 79 (EAP-Message) length=8
Attribute 24 (State) length=38
Attribute 80 (Message-Authenticator) length=18
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
Received 160 bytes from authentication server
Received RADIUS message
RADIUS message: code=2 (Access-Accept) identifier=8 length=160
Attribute 26 (Vendor-Specific) length=58
Attribute 26 (Vendor-Specific) length=58
Attribute 79 (EAP-Message) length=6

Attribute 80 (Message-Authenticator) length=18
RADIUS packet matching with station 00:05:5d:d9:57:59
MS-MPPE-Send-Key (len=32): ef 82 03 c1 ec ea 8e c6 a1 94 85 3c 0a e0 f8 7b ad a1 a5 3b 7f 6e 2e 89 93 11 e4 1c be 6f 9f 6c
MS-MPPE-Recv-Key (len=32): fd c0 5a 5d de 88 65 78 79 07 49 db da 5a 80 23 eb ac 59 97 6b 29 89 34 92 26 82 4f 4f 75 d0 b8
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state SUCCESS
IEEE 802.1X: Sending EAP Packet to 00:05:5d:d9:57:59 (identifier 11)
IEEE 802.1X: 00:05:5d:d9:57:59 REAUTH_TIMER entering state INITIALIZE
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_KEY_TX entering state KEY_TRANSMIT
IEEE 802.1X: Sending EAPOL-Key(s) to 00:05:5d:d9:57:59 (identifier 11)
IEEE 802.1X: Sending EAPOL-Key to 00:05:5d:d9:57:59 (broadcast index=1)
Individual WEP key - hexdump(len=5): c7 ad 84 75 6a
IEEE 802.1X: Sending EAPOL-Key to 00:05:5d:d9:57:59 (unicast index=0)
IEEE 802.1X: 00:05:5d:d9:57:59 AUTH_PAE entering state AUTHENTICATED
IEEE 802.1X: Authorizing station 00:05:5d:d9:57:59
IEEE 802.1X: 00:05:5d:d9:57:59 BE_AUTH entering state IDLE
Received 40 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 00 3f aa aa 03 00 00 00 88 8e 01 00 00 04 03 0b 00 04
DATA (TX callback) ACK
Received 85 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 10 3f aa aa 03 00 00 00 88 8e 01 03 00 31 01 00 05 c1 f6 af a5 88 a9 ee 49 d7 41 01 17 36 47 dc 0a 2a a6 8d 99 5a 91 67 67 81 87 e0 9f 39 ea 79 fd 17 21 b3 f4 85 3d e1 92 7a d8 6e 26 99 d9
DATA (TX callback) ACK
Received 85 bytes management frame
  dump: 0a 02 3a 01 00 05 5d d9 57 59 00 05 5d d9 55 a5 00 05 5d d9 55 a5 40 3f aa aa 03 00 00 00 88 8e 01 03 00 31 01 00 05 c1 f6 af a5 8c 97 d7 be bc f3 27 ff 84 7e 4c 5f 8d e8 30 c4 e5 36 e8 c8 80 3c 4e 80 3e f0 10 2d 9e bc b2 fa 3d 59 fb d4 8a 21 fa 51 ca 7e
DATA (TX callback) ACK
IEEE 802.1X: 00:05:5d:d9:57:59 Port Timers TICK (timers: 29 0 3599 28)

##

LIST OF REFERENCES

1. Geier Jim, *Wireless LANs Implementing Interoperable Networks*, Macmillan Network Architecture & Development Series, 1999.

2. Behrouz A. Forozuan, *Local Area Networks*, McGraw Hill, 2003.

3. Institute of Electrical and Electronics Engineers, "*ANSI/IEEE Std 802.11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications"*, 20 August 1999.

4. Institute of Electrical and Electronics Engineers, "*ANSI/IEEE Std 802.11b/D8.0, DRAFT Supplement to STANDART for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHZ Band"*, September 2001.

5. *Certified Wireless Network Administrator Official Study Guide*, Planet3 2000.

6. Scott Fluhrer, Itsik Mantin, and Adi Shamir, "*Weakness in the Key Scheduling Algotrithm of RC4"*, SAC 2001.

7. Adam Stubblefield, John Ioannidis, and Aviel D. Rubin, "*Using Fluhrer, Mantin, and Shamir Attack to Break WEP"*, AT&T Labs Technical Report 2001.

8. Institute of Electrical and Electronics Engineers, *IEEE Standard for Local and Metropolitan Area Networks Port-Based Network Access Control*, IEEE Std 802.1X-2001.

9. W. Stallings, *Wireless Communications and Networks*, Prentice Hall 2002.

10. Jesse R. Walker, "*Unsafe at Any Key Size: An analysis of the WEP Encapsulation"*, doc: IEEE 802.11-00/362, October 2000.

11. Nikita Borisov, Ian Goldberg, and David Wagner, "*Intercepting Mobile Communications: The Insecurity of 802.11"*, ACM SIGMOBILE 7/01 Rome, Italy, 2001 ACM ISBN 1-58113-422-3/01/07.

12. B.Aboba and D.Simon, "*PPP EAP Authentication Protocol"*, RFC 2716 Experimental, October 1999

13. C. Rigney, S. Willens, A. Rubens, W. Simpson, "*Remote Authentication Dial In User Service"*, RFC2138 June 2000.

14. A. Mishra, W. Arbaugh, *"An Initial Security Analysis of the IEEE 802.1X Standard"*, CS-TR-4328 UMIACS-TR-2002-10 Technical Report 6 February 2002.

15. Bernard Aboba, *"IEEE 802.1X Pre-Authentication"*, doc.: IEEE 802.11-02/389r0 June 2002.

16. T.Dierks, C. Allen *"The TLS Protocol Version 1.0"*, RFC-2246 January 1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia 22060

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California 93943

3. Professor Geoffrey Xie
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943

4. Mr. John Gibson
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943

5. Deniz Kuvvetleri Komutanligi
   Kutuphane
   Bakanliklar, Ankara, TURKEY

6. Deniz Harp Okulu Komutanligi
   Kutuphane
   Tuzla, Istanbul, TURKEY

7. Bilkent Universitesi Kutuphanesi
   Bilkent, Ankara, TURKEY

8. Orta Dogu Teknik Universitesi Kutuphanesi
   Balgat, Ankara, TURKEY

9. Bogazici Universitesi Kutuphanesi
   Bebek, Istanbul, TURKEY

10. Dz. Utgm. H. Selcuk Ozturk
    Deniz Kuvvetleri Komutanligi
    Bakanliklar, Ankara, TURKEY