Hasselt University

Master's Thesis

Wireless Network Privacy

Author:
Pieter Robyns

Promotor:
Prof. dr. Peter Quax

Co-promotor:
Prof. dr. Wim Lamotte

Assessor:
Jo Vermeulen

Mentors:
Ir. Bram Bonné
Arno Barzan

A thesis submitted in fulfillment of the partial requirements for the degree of master in computer science / multimedia

2013 - 2014

Abstract

With the increasing interest in mobile devices such as smartphones and tablets, more sensitive data than ever is being transmitted over wireless links. In this thesis, the impact of this trend on the privacy of wireless network users is investigated. More specifically, existing attacks on wireless networks that are still practically viable today are demonstrated, and new attacks on protocols that are considered secure are explored. These matters are handled respectively in a case study and a feasibility study. For the case study, numerous experiments were conducted to test the practicability of existing attacks, and to estimate the security of different protocols. The feasibility study resulted in a new attack on Apple devices that can be performed in context of WPA-Enterprise authentication. Finally, in both studies, mitigation techniques are discussed that users can implement in order to protect their privacy.

Acknowledgements

First of all, I would like to thank my promotor Peter Quax, my co-promotor Wim Lamotte, Bram Bonné, and Arno Barzan for their invaluable help and feedback on both my thesis and the published WiSec paper. Secondly, I thank my friends Benno Daenen, Laura Coninx, Mathy Vanhoef, and fellow students for their helpful support and suggestions. I would also like to thank my family, who have always been supportive of me. Lastly, special thanks go out to Bart Dierickx, who sparked my interest in computer science from a young age.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the establishment of the first Wi-Fi 802.11 standard by the Institute of Electrical and Electronics Engineers (IEEE) in 1997, the technology has grown to a worldwide scale. According to a recent study by Strategy Analytics, global shipments of Wi-Fi enabled devices grew 19% in 2013 reaching 1.9 billion units [60]. This trend is not surprising, as many consumers and enterprises are attracted to the many advantages of wireless networking. A first advantage is the cost-efficiency of this technology. Enterprises and consumers do not have to install wall jacks and wiring to provide internet access. Instead, one or more wireless Access Points (APs) are often installed that provide ubiquitous access to network resources. Furthermore, Wi-Fi operates in the Industrial, Scientific and Medical (ISM) band, which means no license is required for its usage. A second advantage is mobility: in wireless networks, there is no need to tether a device to a cable in order to access network resources. This is principally useful for handheld devices such as smartphones. As the average smartphone usage grew 50 percent in 2013 according to a study by Cisco [13], this further proves that wireless networking is becoming increasingly popular.

Despite all advantages, wireless networking is a double edged sword. Many of these networks are not properly secured, which may lead to the exposure of privacy sensitive data to cyber criminals. This is especially true for public hotspots, which are often unencrypted open networks. In March 2014, Troels Oerting, head of Europol's cybercrime centre, warned that there is a growing number of attacks being carried out via public Wi-Fi [50]. Even when a proper confidentiality protocol is used, care must be taken that the protocol has no known security vulnerabilities.

In this thesis, we will focus on the privacy aspect of various protocols in context of Wi-Fi networks. The main research question is: "To what degree does wireless networking impact the privacy of its users?". Related questions that we can ask ourselves are: "Are known attacks on older Wi-Fi security protocols still viable today?", and "Are the latest Wi-Fi security protocols strong enough to protect the privacy of the user?". These questions will be answered in a combination of a case study, where we investigate which known Wi-Fi vulnerabilities can be practically exploited by an attacker today, and a feasibility study, where we examine the feasibility of novel attacks on modern Wi-Fi security protocols. Both studies are supplemented with background information that is required to understand the discussed attacks in each chapter.

For our case study, the practicability of attacks was verified by means of experiments, where we performed these attacks on test subjects or devices. Sections detailing such experiments are denoted with the "Experiment: " prefix in the table of contents. However, not all of the existing attacks were verified, as this would not be possible within the given time contraints. Examples of existing attacks that were not verified are attacks on WEP and WPA-Personal. WEP is a very old, deprecated security protocol that already has been extensively researched, whereas WPA-Personal is common and considered secure since existing attacks on WPA-Personal are usually related to brute-force and side-channel attacks. Therefore, the choice was made to focus on open networks (hotspots) and WPA-Enterprise networks, where the existing attacks are more interesting.

The feasibility study focused on finding new ways to attack wireless networks. Given the plethora of authentication methods available in WPA-Enterprise, the choice was made to focus on this technology as this would increase the odds of finding a new vulnerability. A new vulnerability was indeed found in

the WPA-Enterprise implementation of Apple devices, which led to a publication that was submitted and accepted to WiSec 2014.

We will begin this thesis by discussing open network privacy in Chapter 2. This chapter is supplemented briefly with the role of application layer security in open networks. In Chapter 3, we will discuss 802.11i wireless networks and various attacks on its security mechanisms. Elements from this chapter are required for understanding Chapter 4, which details some of the most popular 802.1X authentication methods in the context of wireless network privacy. Chapter 5 will cover a novel WPA-Enterprise implementation vulnerability that we discovered in Apple devices, and finally we will form a conclusion and discuss possible future work in Chapter 6.

Chapter 2

IEEE 802.11 wireless networks

IEEE 802 is a family of standards for Local Area Networks (LANs) and Metropolitan Area Networks (MANs) that deals with the Data Link and Physical layer of the Open Systems Interconnection (OSI) model (see Figure 2.1) [54].

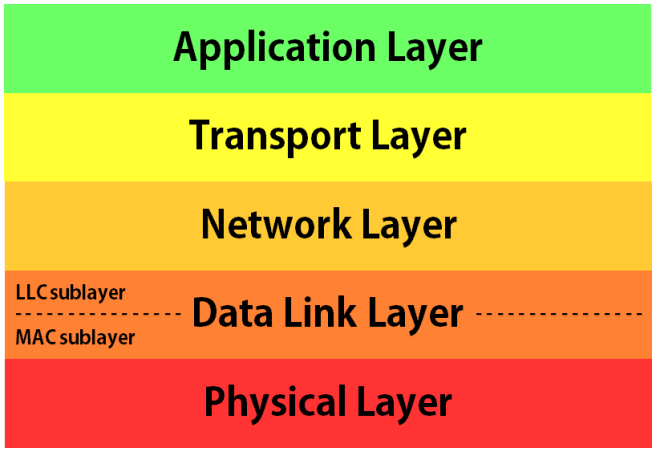

Figure 2.1: OSI model excluding the Session and Presentation layer

The Physical layer is the lowest layer of the OSI model, and encompasses protocols for sending and receiving raw bits between two communicating entities over a physical medium. In essence, these protocols provide the functionality to convert signals such as voltages and radio waves to digital data and vice versa. For example, the 802.11 standard for wireless LAN communication specifies the use of radio waves between 2.412 GHz and 2.484 GHz for a total of 14 channels as shown in Figure 2.2.

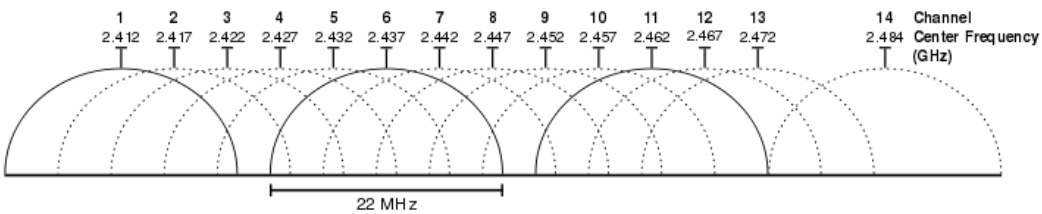

Figure 2.2: 802.11 channels[63]

Directly above the Physical layer is the Data Link layer, which consists of the Medium Access Control (MAC) sublayer and the Logical Link Control (LLC) sublayer. Data Link layer packets transmitted over the network are also called "frames". The MAC sublayer provides host addressing, delimiting of frames,

access control and error protection. The LLC sublayer acts as an interface between the MAC sublayer and the Network layer, and provides flow control functionality [27].

The protocol for the MAC sublayer and Physical layer of wireless LANs is defined in the IEEE 802.11 standard, which is a subset of the IEEE 802 family. In order to understand the concepts described in this thesis, it is essential to have a good understanding of this standard.

2.1 The 802.11 architecture

The 802.11 standard defines a set of components that can be used in a Wireless Local Area Network (WLAN) environment: [27]

- Station (STA): A Station is defined as a single addressable unit within the WLAN network. Any device that is capable of sending and / or receiving a 802.11 message can be considered an STA. Examples are 802.11 capable smart phones, laptops, desktops, or any other device equipped with a wireless Network Interface Controller (NIC).
- Basic Service Set (BSS): The BSS is an abstract definition of a set of multiple STAs. The coverage area within which these STAs can communicate is called the Basic Service Area (BSA).
- Distributed System (DS): The DS is a component that connects multiple BSSs together to form a network with increased coverage. This allows communication from STAs in one BSS to STAs in another BSS.
- AP: An AP is a special case of the STA that allows access to the DS in a wireless manner. There can be only one AP within a BSS. Other STAs must associate to an AP in order to access network resources.
- Extended Service Set (ESS): The ESS is defined as the set of BSSs that make up the same logical network. An ESS is identified by the Service Set Identifier (SSID), which informally is the network name that users see on their device when connecting to an AP.
- Independent Basic Service Set (IBSS): An IBSS is a BSS that forms a self-contained network in which no access to a DS is available.

Figure 2.3 gives an overview of the aforementioned architectural components.

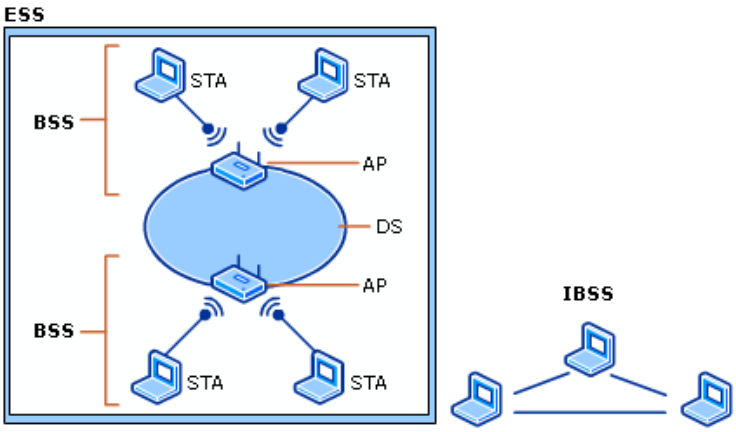

Figure 2.3: IEEE 802.11 architecture overview[36]

2.2 802.11 frames

When considering the data link layer section of 802.11 packets, we refer to 802.11 *frames*. The 802.11 specification defines a specific format for these frames, so that they can support various features. Some of these features are interesting with regard to privacy, and were implemented in a fuzzing tool that was

developed in context of this thesis. This tool will be discussed in Section 2.5.6. Figure 2.4 shows the structure of a generic 802.11 frame [27].

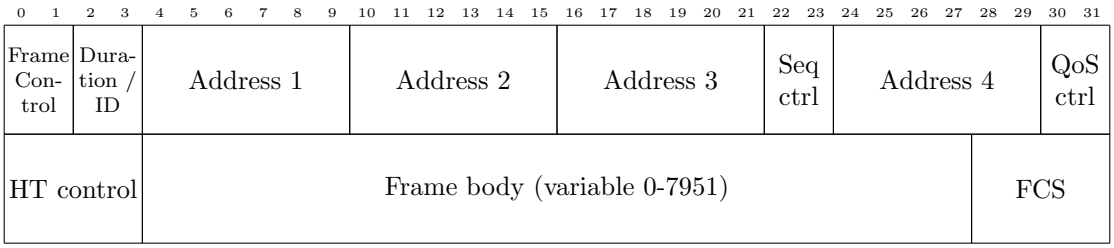

Figure 2.4: 802.11 frame structure and field sizes in bytes

The Frame Control field is special, because it consists of several subfields. These subfields are illustrated in Figure 2.5.

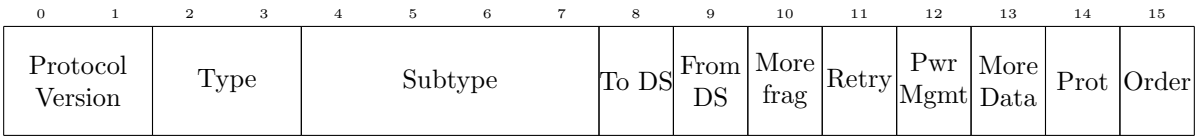

Figure 2.5: Frame Control subfields in bits

The fields in 802.11 frames are transmitted from left to right over the network, and each field has a unique purpose [27, 21]:

- Protocol: Denotes which version of the 802.11 protocol is used. The standard claims that the version number will only increment if a fundamental incompatibility exists between a new revision and the prior edition of the standard. At the time of writing, the latest standard was released in 2012, and no other version than "version 0" exists [27].
- Type: Indicates the frame type of the 802.11 frame. There are three types of frames: *Management* frames (`00`), *Control* frames (`01`), and *Data* frames (`10`). The last possible value (`11`) is reserved, and is therefore not used.
- Subtype: Contains the subtype of the 802.11 frame. Each type has a different set of subtypes, and each subtype serves a different purpose. Table 2.1 gives an overview of the subtypes that are relevant to this thesis. We will discuss the features of these frames in the coming sections.
- To-DS: Bit that indicates whether the frame is destined for the DS.
- From-DS: Bit that indicates whether the frame originated from the DS.
- More Fragments: When a large frame is split over multiple fragments, each nonfinal fragment has this bit set to "1".
- Retry: Indicates whether the frame is a retransmission of a previously sent frame.
- Power Management: If set to "1", this bit indicates that the STA is in power saving mode. A "0" means that the STA is active. An AP cannot enter power saving mode.
- More Data: Indicates that data for an STA that was in power saving mode is buffered at the AP.
- Protected Frame: Specifies whether the content of the frame is encrypted with any data link layer encryption algorithm.
- Order: This bit serves a dual purpose: when Quality of Service (QoS) is not used and this bit is set to "1", frames will be delivered in strictly ordered manner. In the context of QoS frames, the bit indicates whether the frame has a High Throughput (HT) Control field.

- Duration/ID: The Duration/ID field serves different purposes depending on the Type and Subtype fields of the frame. It is sufficient to know that this field is used to reserve the wireless channel for a certain duration to avoid interference of other STAs.
- Address 1: Contains the MAC address of the final recipient or Destination Address (DA) of the frame. In other words it contains the address of the packet's final destination.
- Address 2: Contains the MAC address of the initial sender or Source Address (SA) of the frame.
- Address 3: Contains the MAC address of the intermediary recipient of the frame.
- Sequence Control: The Sequence Control field is split in two parts: the first four bits denote the Fragment Number, and the remaining 12 bits specify the Sequence Number. The former speaks for itself: if the frame is fragmented, this value indicates the fragment number. The latter number is incremented with every transmitted frame.
- Address 4: Contains the MAC address of the intermediary sender of the frame.
- HT Control: The HT Control field was added in the 802.11n standard, and is used in the context of high-throughput communication. The features that this field provides are beyond the scope of this thesis.
- Frame body: The frame body is variable in size and contains the payload from higher-layer protocols.
- FCS: The Frame Check Sequence (FCS) field contains a 32-bit Cyclic Redundancy Check (CRC) over all fields of the MAC header and frame body. This value can be used at the receiving STA to check for errors in the frame.

Now that we have a good understanding of the fields in an 802.11 frame, we will take a look at the different types of frames.

2.2.1 Management frames

When a user wants to connect to a *wired* network, they must first find a wall jack. The wall jack must be enabled, and the user must plug in the network cable before network resources can be accessed. These matters are trivial in a wired network, but in a *wireless* network this is not the case, because there are neither wall jacks nor network cables. The 802.11 standard defines "Management Frames", which were created with the purpose of emulating the physical access procedure for wired networks on a wireless network. We will now briefly discuss all relevant Management Frames, each of which defines and uses a new set of mandatory or fixed length fields, and optional or variable length fields in the frame body of a generic 802.11 frame.

- **Beacon** frames are optionally sent by the AP within the BSA at a certain interval, in order to advertise the presence of an ESS. They are the wireless equivalent to wall jacks in a wired network. Figure 2.6 shows the frame body of a Beacon frame. Here we can see the addition of the following mandatory fields:
  - Timestamp: Contains the number of microseconds that the AP has been active. This value can be used to synchronise STAs within a BSS.
  - Beacon interval: Denotes the number of time units between two subsequent beacon transmissions.
  - Capability information: Gives the capabilities of an STA, including privacy, service set, radio measurement capabilities, and more.
  - SSID: Contains the length of the SSID of the network, followed by the SSID itself.

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	1010	Disassociation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
01	Control	1011	Request to Send (RTS)
01	Control	1100	Clear to Send (CTS)
01	Control	1101	Acknowledgement (ACK)
10	Data	0000	Data

Table 2.1: Table of Type / Subtype field combinations in 802.11 frames

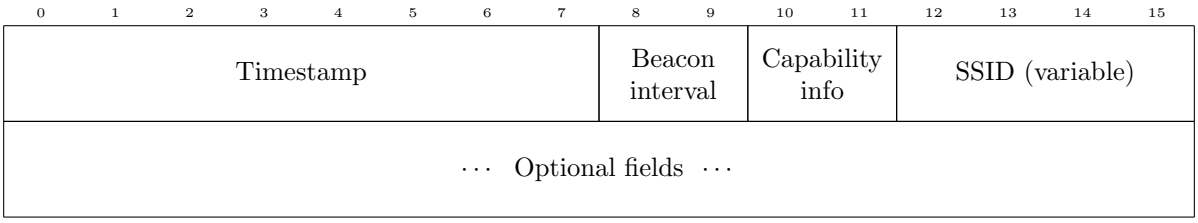

Figure 2.6: Beacon frame body fields in bytes

- **Probe Request and Probe Response** frames: When an STA wants to join an ESS, it will first send a Probe Request frame to share its supported rates (and any optional information), and check whether any AP with the SSID provided in the Probe Request is in range. A Probe Request with a wildcard SSID may also be transmitted, in which case all in-range APs will respond, regardless of the ESS they belong to. Figure 2.7 illustrates the frame body of Probe Requests. The transmission of a Probe Request may happen in three occasions:
  - The user explicitly configured the device to join a (cloaked) network.
  - The Probe Request was sent periodically to check the presence of a network that is already in the Preferred Network List (PNL) of the device.
  - The user requests a list of available in-range APs.

  A Probe Request is always answered with a Probe Response from the AP. The format of a Probe Response is identical to the format of a Beacon frame. However, Probe Responses are unicast to the STA that sent the Probe Request, while Beacons are always broadcast to the entire BSA.

Figure 2.7: Probe Request frame body fields in bytes

- **Authentication and Deauthentication** frames: There are two types of authentication in 802.11:

open system authentication and *shared key* authentication. Authentication frames, which are shown in Figure 2.9, are exchanged to establish the authentication type. Authentication in wireless networks can be compared to enabling a certain wall jack in wired networks.

Open system authentication is defined as a null authentication algorithm, which means any STA requesting authentication will be accepted by the AP. In this case, the Authentication frame exchange is only used to identify the STA.

Shared key authentication is only used when Wired Equivalent Privacy (WEP) was selected as the authentication algorithm. WEP will be discussed in detail in Section 3.1. When the STA requests shared key authentication, the AP will respond with a random challenge text. The STA needs to encrypt this challenge text with a given shared WEP key, and transmit it back to the AP. If the AP can successfully decrypt the challenge response, the authentication is complete, and the AP will send a success message to the STA (see Figure 2.8).

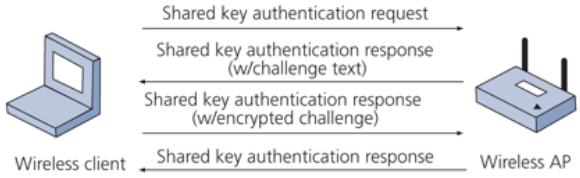

Figure 2.8: 802.11 shared key authentication [28]

Finally, the Deauthentication frame can be transmitted by an STA or AP in order to deauthenticate and disassociate[1] the other party. A deauthentication is always immediately performed upon receipt of a Deauthentication frame, and functions independently of the used authentication method. The format of a Deauthentication frame body is given in Figure 2.10.

0 1	2 3	4 5	6 7 8 9 10 11 12 13 14 15
Authentication alg. number	Authentication trans. seq. no.	Status code	Challenge text (variable)

Figure 2.9: Authentication frame body fields in bytes

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Reason code

Figure 2.10: Deauthentication frame body fields in bits

- **Association Request and Association Response** frames: after Authentication is completed, the final step is association between the STA and the AP. Association is performed to join a wireless network, and can be compared to plugging in the network cable in a wired network. The format of an Association Request is given in Figure 2.11. Here we can see the addition of the Listen Interval field, which indicates how often an STA listens for Beacon frames by the AP.

  Upon receiving the Association Request, the AP will respond with an Association Response (see Figure 2.12). The AP will assign an Association ID (AID) to the STA as part of the association.

[1] Since authentication is a prerequisite for association, a deauthentication will also result in a disassociation.

0 1	2 3	4 5 6 7 8 9	10 11 12 13 14 15
Capability information	Listen interval	SSID (variable)	Supported rates (variable)

Figure 2.11: Association Request frame body fields in bytes

0 1	2 3	4 5	6 7 8 9 10 11 12 13 14 15
Capability information	Status code	AID	Supported rates (variable)

Figure 2.12: Association Response frame body fields in bytes

- **Reassociation Request and Reassociation Response** frames are used in the context of roaming, and are similar to Association Requests and Association Responses. In fact, Association Responses and Reassociation Responses are identical, and the only difference between Association Requests and Reassociation Requests is that the MAC address of the currently associated AP is given in the Reassociation Request. This allows the transfer of association information from the old AP to the new AP. The frame body of a Reassociation Request is illustrated in Figure 2.13.

0 1	2 3	4 5 6 7 8 9	10 11 12	13 14 15
Capability information	Listen interval	Current AP MAC	SSID (variable)	Supported rates (variable)

Figure 2.13: Reassociation Request frame body fields in bytes

- **Disassociation** frames are identical to Deauthentication frames, the difference being that Disassociation frames end the association relationship, whereas Deauthentication frames end both the association and authentication relationship.

2.2.2 Control frames

Control frames offer features regarding the delivery of data frames. Examples of such features are reservation of the wireless medium, reliability, and power management.

- **RTS and CTS** frames: RTS frames are optionally sent by an STA before the transmission of a large unicast data frame in order to reserve the wireless medium for the duration specified in the Duration field. CTS frames are sent as a response to the RTS frame. After receiving the CTS reply, the STA can begin sending the large packet, and other STAs are alerted to hold off for the duration specified in the CTS frame.

  The RTS/CTS mechanism essentially prevents collisions with hidden nodes (see Figure 2.14), and the retransmission of a large data frame as a consequence [66]. The format of these frames is similar to the generic 802.11 frame (see Figure 2.15 and Figure 2.16).

0 1	2 3	4 5 6 7 8 9	10 11 12 13 14 15	16 17 18 19
Frame Control	Duration	Receiver MAC	Transmitter MAC	FCS

Figure 2.15: RTS frame structure and field sizes in bytes

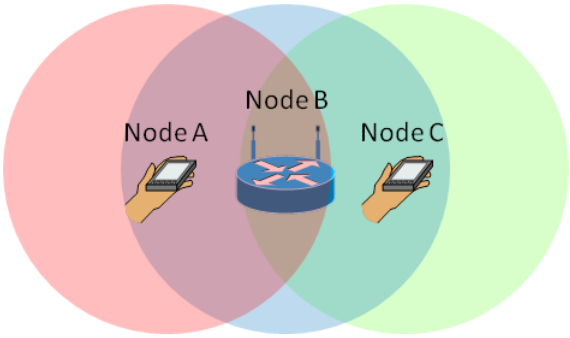

Figure 2.14: The hidden node problem: node A and node C are too far apart to hear each other

0 1	2 3	4 5 6 7 8 9	10 11 12 13
Frame Control	Duration	Receiver MAC	FCS

Figure 2.16: CTS frame structure and field sizes in bytes

- **ACK** frames are used to positively acknowledge the reception of a certain frame to the sender of that frame. The structure of an ACK frame is identical to that of a CTS frame, except the Subtype field is set to "1101".

2.2.3 Data frames

Data frames sent over an 802.11 network carry higher-level protocols in the Frame Body field. The structure of a data frame is identical to the generic frame structure (see Figure 2.4), except the Type field is set to "10", and the Subtype field is set to "0000". If the STA has QoS service support, the Subtype is set to "1000". Details of the 802.11 QoS mechanisms are outside the scope of this thesis.

2.3 802.11 authentication and association

When an STA wants to join a wireless network, a combination of management frames from Section 2.2.1 must first be exchanged with the AP.

As a first step, the AP may advertise itself to all in-range STAs by sending Beacon frames. However, this step is not a requirement, since some APs implement SSID cloaking[2]. In addition to the Beacons sent by the AP, STAs may periodically send broadcast Probe Requests or a list of Probe Requests directed towards one specific SSID in order to discover in-range networks.

When the STA decides to join a network, the first step is to send an Authentication frame. Here, one can choose between open authentication and shared authentication. Open authentication can be used with any cryptographic protocol. Shared authentication however, must be used in conjunction with WEP.

As a final step, the STA will request to join the network by sending an Association Request frame. The AP will assign an AID to the STA, and reply with an Association Response. If the network does not use an encryption protocol (open network), the STA can immediately begin transmitting data frames (see Figure 2.17). When encryption is enabled, another authentication procedure must be performed before network resources may be accessed.

[2]Cloaked APs do not send Beacons and only reply to Probe Requests containing their SSID in an attempt to hide the SSID from an attacker. However, this security measure can be easily circumvented by sniffing the Probe Request from a legitimate client.

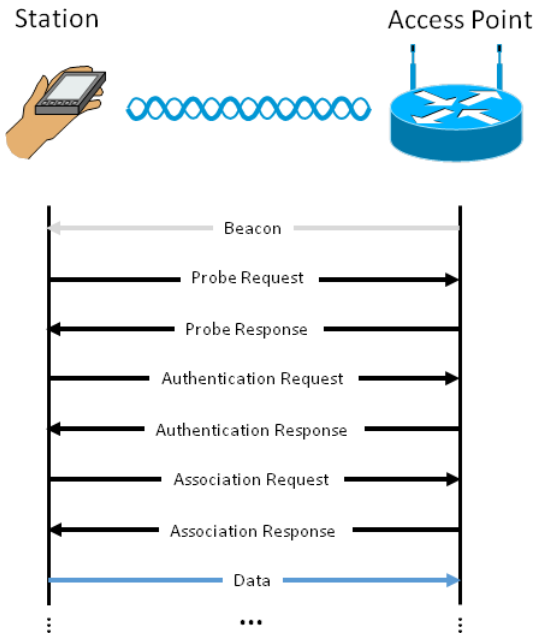

Figure 2.17: 802.11 open network authentication

2.4 NIC modes

A NIC used to connect with 802.11 networks can be configured to operate in several modes. Some of these modes are frequently used in the discussed attacks from this thesis, namely:

- Infrastructure mode: The NIC will act as a client STA connecting to an AP. Only packets destined for the MAC address of this NIC, excluding management and control frames, are passed on to higher layers of the protocol stack.
- Monitor mode: In monitor mode, the NIC will promiscuously capture packets from all in-range 802.11 devices. This includes management and control frames.
- Master mode: The NIC will act in a similar way as infrastructure mode, but as an AP instead of a client STA.

Not all manufacturers support the same set of modes for their NICs. Additionally a NIC might behave slightly differently depending on the chipset and manufacturer.

2.5 Attacks on open 802.11 wireless networks

Given that communication over open 802.11 wireless networks is unencrypted, many attacks can naturally be performed in this context. In this section, these attacks will be detailed, and their applicability today is verified through a number of experiments.

2.5.1 Passive MITM

The passive Man-In-The-Middle (MITM) attack is one of the simplest attacks on 802.11 networks. Here, an attacker configures their NIC to operate in monitor mode. This can be achieved with standard Linux tools such as `iwconfig` and `airmon-ng`.

Because the NIC is in monitor mode, and because open 802.11 networks do not use any form of encryption, it is possible for an attacker to sniff all transmitted packets within range of the NIC on a specific channel. Sniffing on the monitor mode interface can be done with tools like Ettercap and Wireshark.

2.5.2 Experiment: passive MITM with Snoopy

A tool called "SnooPy" was developed to capture data in an automated fashion while performing a passive MITM. The tool provides the following features:

- Interface to MySQL for logging data to disk by using Structured Query Language (SQL) transactions.
- Capture and logging of 802.11 management frames (Associations, Probe Requests).
- MAC address logging of in-range STAs and APs.
- Profiling of the network type of an AP (open network, 802.1X network or 802.11i network) based on the packets sent to and from this AP.
- Capture and logging of 802.1X frames.
- Parsing of Hyper Text Transfer Protocol (HTTP) streams to extract sensitive data. This data can be logged to disk. To protect the privacy of the test subjects, data is obfuscated before it is logged to disk.

Upon execution, SnooPy creates a monitor mode interface for a wireless NIC using `airmon-ng`. Then both the original interface (e.g. `wlan0`) and the monitor mode interface (e.g. `mon0`) are sniffed. The regular interface is used for sniffing HTTP traffic, whereas the monitor mode interface is used for sniffing 802.11 management frames. This has the advantage that, in an active MITM attack (Section 2.5.4), the regular interface will see decrypted packets, but no management frames when spoofing a cryptographically secured network. On the contrary, the monitor mode interface will see management frames, but it will not see decrypted data packets.

2.5.3 Active MITM

When an attacker is able to trick an STA into associating to an AP under their control, the attacker is an active MITM. Contrary to a passive MITM, traffic is destined directly for the attacker's AP. This kind of attack is also known as the Evil Twin attack, Karma attack or Rogue AP attack.

To make matters worse, many devices nowadays connect automatically to an AP that is in the PNL of the user. Furthermore, recall from Section 2.2.1 that Probe Requests are sent at regular intervals. These Probe Requests can be intercepted by an attacker to learn which SSIDs are in the PNL of the user, and a rogue AP can consequently be set up dynamically to perform an active MITM attack. For this reason, users should remove unused open networks from their PNL, so that automatic exploitation of this attack can be prevented.

Despite the fact that many mitigation strategies have been explored by researchers [53, 46, 65], the experiment that we will discuss in Section 2.5.4 shows that the Evil Twin attack is still viable today.

2.5.4 Experiment: active MITM with hostapd_spoof

In a first experiment, we attempted to use a combination of Python and Scapy[3] to send fake Probe Responses to devices. However, we noticed that the time required to send a Probe Response was too large, causing some devices to drop the packet. For this reason, we made modifications to the `hostapd` tool to fit our needs.

`hostapd` is a user space software access point written in C by Jouni Malinen. It uses the `nl80211` driver to create a master mode interface for data traffic and a monitor mode interface for transceiving management frames [9]. This implementation performs much better when compared to the Python + Scapy implementation.

To provide internet connectivity to our software AP, we used `iptables` to route packets from the `hostapd` interface to an internet enabled interface and back. We used `dnsmasq` to set up our own Domain Name

[3]Scapy can be downloaded from `http://www.secdev.org/projects/scapy/`.

Server (DNS) and Dynamic Host Control Protocol (DHCP) server. The commands used to configure this setup are shown in Listing 2.1.

```
# $1: hostapd interface
# $2: internet enabled interface

# Set IP address of hostapd interface
ifconfig $1 up 10.0.0.1 netmask 255.255.255.0

# Disable power management
iwconfig $1 power off

# Enable IPv4 forwarding
sysctl -w net.ipv4.ip_forward=1

# Start dnsmasq of required
if [ -z "$(ps -e | grep dnsmasq)" ]
then
 dnsmasq
fi

# Clear previous iptables configuration
iptables --flush
iptables --table nat --flush
iptables --delete-chain
iptables --table nat --delete-chain

# Route packets to internet enabled interface and back
iptables --table nat --append POSTROUTING --out-interface $2 -j MASQUERADE
iptables --append FORWARD --in-interface $1 -j ACCEPT

# Start hostapd
hostapd <conf_file>
killall dnsmasq
```

Listing 2.1: hostapd_spoof setup script

Our modified `hostapd` has a number of additional options that may be specified in a `hostapd` configuration file:

- `safespoof=[0,1]`: When safespoof is enabled (1), the AP will send a Deauthentication frame after successful association to disconnect the STA[4]. This feature was implemented to prevent a victim from sending application data over our AP, or in other words: to protect the privacy of test subjects.
- `specific=[0,1]`: Specifies whether to answer all Probe Requests (0), or only the Probe Requests containing our SSID (1).
- `answer_broadcast_probereq=[0,1]`: If enabled (1), broadcast or wildcard SSID Probe Requests are answered with Probe Responses containing SSIDs of popular, free hotspots. Examples of such hotspots in Belgium are `FON_BELGACOM`, `TELENETHOTSPOT`, and `TELENETHOMESPOT`.
- `cycle_spoof_ssids=[0,1]`: If enabled (1), the fake AP will store SSIDs from Probe Requests in a circular array. When a broadcast Probe Request with a wildcard SSID is received, one SSID is then chosen from this list for the Probe Response. This feature was implemented because some devices remove an SSID from the list of scanned networks quickly if the AP fails to respond timely

[4]When spoofing an 802.1X secured network, we will deauthenticate the STA after a successful EAP exchange. 802.1X and EAP will be discussed in Section 4

to a broadcast Probe Request. Duplicate entries in the circular array are ignored, so the more popular a network is, the more it will be used in response to broadcast Probe Requests.

- `cycle_probe_response_types=[0,1]`: Specifies whether to cycle between advertising the network as open, Wi-Fi Protected Access (WPA)-Enterprise, or WPA-Personal secured (1). If disabled, only the security protocol configured in the configuration file is advertised.
- `ssid_blacklist=[list]`: Allows the user to specifiy a comma-seperated list of SSIDs that may not be spoofed.

We performed a number of experiments on different locations, where we used the `hostapd_spoof` tool to investigate the current feasibility of an Evil Twin attack. The SQL queries that were used to determine the results are shown in Table 2.2. Here, "Notable subjects" calculates the number of devices that sent a reasonable amount of Probe Requests. We consider these devices in range. The number of "Vulnerable subjects" is calculated by checking which devices associated to an open network (`type = 0`) during the experiment.

Description	SQL Query
Subjects	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests GROUP BY mac) AS c;`
Notable subjects	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests GROUP BY mac HAVING COUNT(*) > 5) AS c;`
Vulnerable subjects	`SELECT COUNT(*) FROM (SELECT DISTINCT mac FROM AssociationRequests WHERE mac IN (SELECT mac FROM ProbeRequests GROUP BY mac HAVING COUNT(*) > 5) AND ssid in (SELECT DISTINCT ssid FROM APs WHERE type = 0)) AS c;`

Table 2.2: SQL queries used to calculate feasibility of the Evil Twin attack

The results of this experiment are shown in Table 2.3. Note that the percentage of vulnerable subjects is highly dependent on when a subject is considered "notable". We can conclude that 212 out of 1317 devices (16%) are vulnerable to the attack.

Location	Duration	Subjects	Notable subjects	Vulnerable subjects	Percentage vulnerable
Test 1: UHasselt	1h26m12s	673	380	58	$\frac{58}{380} = 15\%$
Test 2: EDM Research Center	1h33m05s	61	41	1	$\frac{1}{41} = 2\%$
Test 3: Public transport	1h35m02s	794	314	78	$\frac{78}{314} = 25\%$
Test 4: Public transport	0h44m16s	196	85	11	$\frac{11}{85} = 13\%$
Test 5: UHasselt	1h35m08s	890	497	64	$\frac{64}{497} = 13\%$

Table 2.3: Evil Twin attack test outside lab environment

2.5.5 Experiment: fingerprinting AP authentication modes

Recall from Section 2.2.1 that devices send Probe Requests on a regular basis, and that they connect automatically to known SSIDs. This connection is only initiated when the authentication mode advertised by the AP matches the authentication configuration in the device. We can exploit this behavior to determine the authentication mode of a targeted SSID remotely.

To exploit this knowledge in practice, the hostapd_spoof tool was extended to cycle between an SSID spoofed as open network, WPA-Enterprise network, and WPA-Personal network. SnooPy was extended to sniff the relevant packets that determine whether the STA tried to connect to our rogue AP. If the STA attempted to connect, we know that our currently guessed authentication mode for the SSID was correct. This knowledge is required for the "Vulnerable subjects" SQL query from Table 2.2 (SSID type).

The categorisation of three authentication types (open, enterprise, and personal) was made because we found that devices make no distinction between a WPA-TKIP and a WPA2-CCMP network.

2.5.6 Experiment: 802.11 vulnerability fuzzing with Scapy

When testing a possible 802.11 protocol vulnerability based on an assumption, it might be interesting to interact with a 802.11 standard compliant device to verify whether the vulnerability is indeed present. Similarly it might be worth looking at vendor specific vulnerabilities that result from incompliance to the 802.11 standard.

In both of the above scenarios, there is a need for a tool that allows easy creation and manipulation of 802.11 packets. Altough many tools such as `rfakeap`, `fakeap.pl`, and `airbase-ng` could be used for this purpose, they are designed for specific attack scenarios. Using these tools as a prototype tool is difficult, because packets need to be crafted in a byte-per-byte fashion. Some of these tools are written in C, which is also an inconvenience in terms of lines of code and compilation time.

For these reasons we developed a prototype tool that can emulate a fake AP completely in software and manipulate packets on the fly[5]. This fake AP appears to client devices as if it were legitimate. The tool was written in Python and uses the Scapy library for flexible packet manipulation. Scapy supports a very wide range of network protocols natively, and is therefore ideal for this purpose[6]. On the command line, we can invoke the tool with:

```
prototype.py [-h] [--interface INTERFACE] [--ap AP] [--mac MAC]
             [--mode MODE] [--dns DNS] [--specific SPECIFIC]
```

Arguments that may be provided are:

- `-h`: Display help.
- `--interface <if>`: A monitor mode interface. The default is `mon0`.
- `--ap <ssid>`: SSID of the spoofed AP.
- `--mac <mac>`: MAC address to be used by the AP. By default the tool uses the MAC address of the specified interface.
- `--mode <mode>`: Specifies whether the tool relies on an external program to handle management and control frames (`--mode 0`) or whether this is done entirely with Scapy (`--mode 1`). Mode 1 is slower but allows more control and fuzzing potential.
- `--dns <dns>`: Address of the DNS server that is communicated to STAs upon association.
- `--specific <specific>`: Indicates whether the AP should only reply to Probe Requests directed towards the specified SSID (`--specific 1`) or to all Probe Requests (`--specific 0`).

The tool is capable of associating with multiple STAs. However, routing Application Layer packets using this tool was not implemented, because we found this to be *very* slow. This is due to the fact that the Scapy library dissects[7] all packets in user space. Nevertheless, the tool is useful for vulnerability fuzzing in layers below the Network layer of the OSI model. The vulnerability that we will discuss in Chapter 5 was in fact found using this tool.

2.6 Application layer security

When it comes to privacy, the Application Layer of the OSI model contains a lot of sensitive data. Browsing the web, exchanging Facebook messages, and performing online banking transactions are some examples of activities that are highly privacy-sensitive, and have become part of everyday life.

[5] This tool will become available for download at `http://www.nostack.net/`.

[6] After completion of our prototype tool, a similar tool that uses Python and Scapy was released at GitHub by Dan McInerney on February 8 2014. This tool can be found at `https://github.com/DanMcInerney/fakeAP/blob/master/fakeAP.py`

[7] In Scapy lingo, dissecting a packet means identifying all involved protocols and fields on all layers of the OSI model.

In the event that an attacker successfully launches an active MITM attack on a device, all this sensitive data will be routed to the attacker. However, this does not necessarily mean that a user's privacy has been fully compromised, since the application data may be encrypted with a secure protocol. In practice, this encryption is often performed on the Transport Layer to protect the encapsulated application data, and the most notable protocol for this purpose is Transport Layer Security (TLS). This protocol was previously known as Secure Sockets Layer (SSL).

In this section we will briefly see how TLS works. In addition we will discuss some attacks on TLS.

2.6.1 Transport Layer Security

TLS is a cryptographic protocol that is used to provide privacy and data integrity between two communicating entities [17]. Because it runs on the Transport Layer of the OSI model, the protocol is transparent to applications. A common usage for TLS is the secure transportation of HTTP data.

The first step in establishing a secure channel is to perform the TLS handshake between the two communicating entities. In this process, a secret key is exchanged that will be used to encrypt subsequent communication. The handshake is performed as follows: [17]

1. The client sends a ClientHello message to the server. Hello messages are used to exchange security enhancement capabilities between client and server. For example, the Hello message may indicate which cryptographic algorithms and TLS version will be used. In addition, the ClientHello contains a random nonce, which we will denote as ClientHello.random.
2. The server replies with a ServerHello message, which also contains a random value ServerHello.random. Following the ServerHello message, the server sends a Certificate message which contains the certificate of the server. This certificate will be used by the client to check whether the public key of the server is authentic. Optionally, the server may ask the client for a certificate as well, so that both entities are mutually authenticated. Finally, the server sends a ServerHelloDone message, which concludes the hello-message phase of TLS.
3. The client sends a random 48-byte `pre_master_secret` to the server, encrypted with the server's public key. Both entities then calculate the shared secret key or TLS `master_secret` as follows:

$$\begin{aligned} \text{master\_secret} = \text{PRF}(&\text{pre\_master\_secret}, \text{“master secret”}, \\ &\text{ClientHello.random} + \text{ServerHello.random})[0..47] \end{aligned} \tag{2.1}$$

4. The client and server send each other a Finished message, indicating the completion of the TLS handshake.

2.6.2 Certificate authenticity

To verify whether a public key belongs to a certain name, a certificate that was issued by a trusted entity must be validated by the TLS peer. This trusted entity can be a Certificate Authority (CA). The certificate that this CA presents may itself be signed by another CA higher up in the hierarchy, in which case the CA is an intermediate CA. The user must traverse the hierarchy of CAs until a trusted one is found. This “certification path” is shown in Figure 2.18. [23]

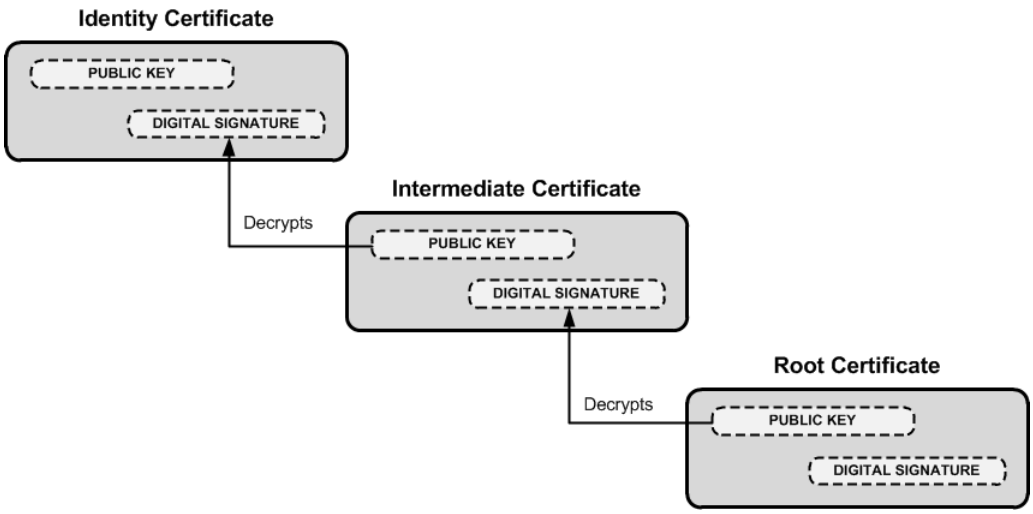

Figure 2.18: The certification path or chain of trust [24]

2.7 Attacks on TLS

Despite the fact that TLS is considered secure when setup correctly, attacks may be performed on the protocol under certain conditions. These conditions include misconfigurations on the client or server, or old versions that are installed on the TLS server. In this section we will discuss some attacks that follow from these mistakes.

2.7.1 Fake certificates

On of the simplest attacks on TLS is to provide a fake certificate once an active MITM attack has been performed. A fake certificate can be generated in the same way as any normal certificate using `openssl`:

```
openssl genrsa -des3 -out fake.key 1024
openssl req -new -key fake.key -out fake.csr
openssl x509 -req -days 365 -in fake.csr -signkey fake.key -out fake.crt
```

Listing 2.2: Commands to generate a fake certificate

As an example, suppose the attacker presents this certificate when a victim user visits `facebook.com` over HTTP Secure (HTTPS). The user's browser will not be able to verify the certificate, resulting in a warning message. However, a recent study by Google indicated that in 70.2% of the cases, the user will simply click through this warning message [5]. Therefore, providing a fake certificate would be a viable attack against an unsuspecting user.

2.7.2 SSLstrip

SSLstrip is a MITM tool for intercepting HTTPS connections, presented by Moxie Marlinspike at Black Hat DC 2009 [41, 33]. Using this tool, an attacker can force a user's User Agent (UA) to access a web page with HTTP instead of HTTPS. The attack is performed in the following way:

1. The attacker becomes an active MITM using any method. For example, the attack described in Section 2.5.3 can be performed.
2. HTTP traffic is intercepted by the SSLstrip tool.

3. All `<a href="https://" >` Uniform Resource Locators (URLs) are rewritten to `<a href="http://" >`. Additionally, a map is kept of which URLs were changed. SSLstrip can also rewrite other fields, such as the Location HTTP header, which is used in HTTP redirects, and the Set-Cookie header, which is used for configuring cookies on the client browser.
4. When the victim device makes a HTTP request for an URL that was mapped, the request is proxied as HTTPS to the server.

With the above exploit, the server does not notice any difference, since it sees the HTTPS requests as expected. The client however, will send data over an unencrypted HTTP channel to the SSLstrip tool, where data can be logged to disk.

SSLstrip can be mitigated by using HTTP Strict Transport Security (HSTS), a mechanism that forces the browser to use HTTPS by means of a preloaded list of websites or a trust-on-first use HSTS header [22, 30].

2.7.3 Experiment: SSLstrip

We conducted an experiment to test the current effectiveness of SSLstrip on high profile websites, in combination with the active MITM attack described in Section 2.5.3. The setup of this experiment is nearly identical to the setup from Section 2.5.4. The only difference is that we run SSLstrip using the commands shown in Listing 2.3. These commands ensure that all HTTP traffic is routed to SSLstrip on port 10 000, so that secure links can be stripped.

```
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000
python2 ./sslstrip/sslstrip.py -l 10000 -k -w /dev/null
```

Listing 2.3: SSLstrip in combination with the active MITM attack

From the results of our experiment we concluded that SSLstrip is still very effective in active MITM attacks today. Table 2.4 shows that many high profile websites are still vulnerable to the SSLstrip attack. The reason why they are vulnerable is because these websites have not been incorporated yet in the preloaded HSTS list of UAs, or because they do not send a HSTS header.

Website	Chrome 35.0.1916.38	Safari 9537.53	Firefox 29.0.1	Internet Explorer 11.0.9600.17107
`google.com`	✓	✓	✓	✓
`accounts.google.com`	✗	✗	✗	✓
`login.live.com`	✓	✓	✓	✓
`facebook.com`	✓	✓	✓	✓
`twitter.com`	✗	✗	✗	✓
`bnpparibasfortis.be`	✓	✓	✓	✓
`kbc.be`	✗*	✗*	✗*	✗*
`belfius.be`	✓	✓	✓	✓

* This website uses Javascript obfuscation to force the browser to a HTTPS link, which is a technique to defeat SSLstrip. The interested reader can read more about this technique at the following blog: `http://securityweekly.com/2012/12/defending-against-ssl-strippin.html`.

Table 2.4: Websites vulnerable to SSLstrip

Note that these results assume that no HSTS header was obtained yet from the target website. In other words, only websites in the HSTS preloaded list are not vulnerable. Internet Explorer performed the worst in this experiment, because this browser does not support HSTS.

2.7.4 BREACH attack

BREACH is a compression side channel attack on SSL that was introduced at Black Hat USA in 2013. The attack can be executed under the following conditions: [42]

- The attacker can inject chosen plaintext in a user's requests.
- The web server uses HTTP compression (gzip / deflate).
- The web server reflects user data on a webpage via query string parameters.
- The web server serves a secret such as a Cross-Site Request Forgery (CSRF) token or session key.

An active MITM can exploit the BREACH attack by repeatedly sending requests to the web server and observing the results that are returned after compression. Because the deflate compression algorithm uses a combination of Huffman encoding and the LZ77 algorithm [16], repeated byte sequences are not repeated in the output. This allows an attacker to guess the secret by observing the length of the output.

As an example, suppose the victim is currently assigned a session key `PHPSESSID=1a79a4d60de6718-e8e5b326e338ae533`. The attacker injects the string `PHPSESSID=a` in the request, and this string is compressed and reflected back in the output. Next, the attacker injects `PHPSESSID=b`, and so on. Because compression is used, we know the shortest output will be the correct guess, e.g. `PHPSESSID=1` will be shorter than `PHPSESSID=a`. Using this method, secret can be guessed byte per byte.

2.7.5 Heartbleed bug

The Heartbleed bug is a recently discovered vulnerability that affected OpenSSL versions 1.0.1 to 1.0.1f. The attack uses crafted Heartbeat Extension packets to trigger a buffer over-read that leaks memory contents of the server. Consequently, an attacker may obtain the server's private key and decrypt any data sent to this server through an active MITM attack. [40]

2.7.6 HSTS bootstrap MITM vulnerability

A HSTS policy can be honored by the UA in two ways. A first possibility is that the policy is incorporated in a preloaded list on the UA. However, this list is very short, and therefore many high profile domain names do not have their policy registered[8] [30].

The second possibility is that the UA receives a `Strict-Transport-Security` HTTP Response header over a valid secure channel before it will honor a HSTS policy. This is a typical example of the "trust on first use" principle. An attacker can intercept the `HTTP` Response header in an active MITM attack, and remove the header to prevent the browser from honoring the HSTS policy [22].

[8] For example, the Google Chrome HSTS preloaded list can be found at `https://src.chromium.org/viewvc/chrome/trunk/src/net/http/transport_security_state_static.json`. This list does not contain high profile domains such as `facebook.com`, `ing.be`, `bnpparibasfortis.be`, etc.

Chapter 3

Wired Equivalent Privacy and IEEE 802.11i

The MITM attacks described in Section 2.5 can be performed on open networks because of the lack of user privacy. For this reason, the WEP encryption protocol was developed. This protocol was intended to provide privacy equivalent to wired networks. Even though the 802.11 standard specifies that WEP may be used to increase confidentiality of data, it is known to be a very insecure protocol and is currently deprecated for this reason [27].

To address the issues discovered with WEP, the IEEE proposed the 802.11i standard, which describes new security features that may be used by 802.11 devices. In this section, we will discuss the workings of both WEP and the newer algorithms described in the 802.11i standard.

3.1 WEP mechanisms

Standard 64-bit WEP takes a secret key k, composed of a root key k_r of 40 bits and an Initialization Vector (IV) of 24 bits. The root key is typically shared among the users of the network, whereas the IV is randomly generated and different for every packet. The secret key is used as the seed for the Rivest Cipher 4 (RC4) stream cipher. RC4's Pseudo-Random Generation Algorithm (PRGA) generates a *key stream* which is XORed with the concatenation of the plaintext frame body and an Integrity Check Value (ICV). This ICV is designed to protect the frame against tampering, and is calculated as a 4-byte CRC checksum of the frame body.

The result of the WEP operation is called the encrypted data or ciphertext. The IV and KeyID[1] are concatenated with the ciphertext and sent over the network (see Figure 3.1). At the receiving side, this process is reversed to yield the original plain-text data frame.

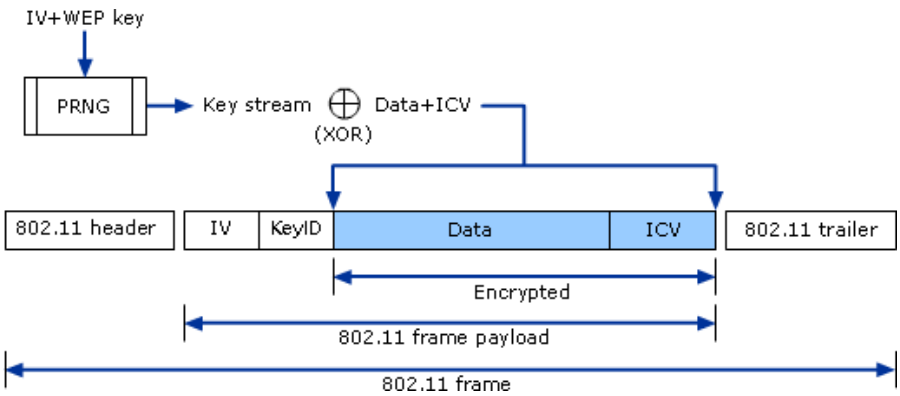

Figure 3.1: WEP encryption algorithm [36]

[1]WEP allows usage of multiple root keys. Which one is used is determined by this KeyID field.

3.2 Attacks on WEP

The mechanism described in Section 3.1 has numerous flaws. In this section we will discuss how these flaws can be exploited in order to compromise the privacy of users.

3.2.1 Replay attack

From Section 3.1 we can observe that WEP does not have the ability to ensure that a message originated from the claimed sender. This ability, known as data authentication, is a crucial security component to prevent packet injection attacks by an attacker [27]. Hence, the protocol is inherently vulnerable to replay attacks.

Since an IV may be reused freely, and since there is no sequence number to check for replayed packets, an attacker can replay frames to generate extra traffic. This is commonly done with Address Resolution Protocol (ARP) Requests, as these packets can be easily distinguished based on their destination MAC[2] or fixed size. Once the ARP Request has been replayed, an ARP Response will be sent in reply to the ARP Request, and hence extra traffic will be generated.

3.2.2 ChopChop attack

The ChopChop attack was introduced by a man with the pseudonym "KoreK" on an internet forum. An attacker can perform the ChopChop attack to decrypt an encrypted WEP packet. This also implies that a keystream can be recovered. However, the root key cannot be recovered using this method.

The attack can be summarized as follows: [57, 18]

1. The attacker authenticates and associates to the network.
2. A packet is sent to the AP by a legitimate client. Before encryption by the RC4 algorithm, an ICV was appended to the plain-text data.
3. The attacker sniffs this packet, and "chops off" the last byte of encrypted data. Denote R as the plain-text value of this truncated byte. The remaining bytes Q will now have an invalid ICV.
4. The attacker guesses R. Since the ICV is calculated with the CRC32 error detection algorithm instead of a secure algorithm, the attacker can modify the ICV to be valid based on the knowledge of R [3].
5. The packet is transmitted to the AP. If the attacker's guess for R was correct, the AP will forward the message to its destination. Otherwise, the packet will be silently discarded. In the latter case, we know the attacker's guess for R was incorrect, and step 4 is repeated with a new value for R.

The above process can be repeated until n bytes of the keystream have been recovered. Note that step 1 is optional in case that a vulnerable AP sends a Disassociation packet if a valid packet is received from an unauthenticated STA. This Disassociation packet can be used to determine whether the guess for R was correct.

3.2.3 Fragmentation attack

The fragmentation attack described by Andrea Bittau can be performed to obtain a keystream from an encrypted WEP packet. In summary, the attacker first guesses part of the keystream, and then reuses this keystream to inject crafted packets of arbitrary length using 802.11 fragmentation. Next, when this packet is forwarded by the AP, it can be sniffed again. A larger keystream can then be recovered, because the attacker knows the plain-text data of their injected packet [10]. Note that contrary to the ChopChop attack (see Section 3.2.2), this attack cannot be used for decrypting any given WEP packet,

[2] Recall from Section 3.1 that only the frame body is encrypted, and that the frame header is visible to anyone.

[3] The reason why this can be done even though the ICV is encrypted, is outside the scope if this thesis. The following website provides an intuitive explanation for the interested reader: `http://www.aircrack-ng.org/doku.php?id=chopchoptheory&DokuWiki=ecfe066435c4bfc667f4417abc7e6079`

because each packet has a different keystream. The fragmentation attack focuses on injecting packets of arbitrary length.

As an example, suppose the attacker sniffs a random packet P. In Chapter 2 we saw that all packets are encapsulated with LLC. The LLC/Subnetwork Access Protocol (SNAP) header is 8 bytes in length and contains almost always constant fields. Therefore, 8 bytes of the keystream can be recovered.

In the next stage, the attacker can inject their own packets using these 8 bytes of recovered keystream. Since the ICV is 4 bytes, and since there may be a maximum of 16 fragments in 802.11, this means $4 \cdot 16 = 64$ bytes of arbitrary data may be sent by the attacker. At this point the attacker could inject a long broadcast frame to recover more bytes of the keystream and repeat the process.

3.2.4 RC4 related attacks

Aside from WEP, the RC4 itself has some vulnerabilities that can be exploited. These vulnerabilities are all of cryptographic nature, and will therefore not be discussed in this thesis.

3.3 WPA and WPA2

In response to the security vulnerabilities that were found in WEP, the IEEE created an amendment to the 802.11 standard named 802.11i. The 802.11i amendment defines new security protocols that use Robust Security Network Associations (RSNAs). An RSNA is an authentication or association between a pair of STAs that includes the 4-Way Handshake, which will be discussed in this section. If the entire network allows only RSNAs, that network is named a Robust Security Network (RSN).

More specifically, the new protocols that were introduced in 802.11i are Temporal Key Integrity Protocol (TKIP) and Counter mode Cipher block chaining Message authentication code Protocol (CCMP). TKIP is a cipher suite which enhances WEP, but also maintains backwards compatibility [25]. When this protocol is used, the network is said to be secured by WPA in consumer terms. CCMP is a more secure protocol that uses Advanced Encryption Standard (AES), and is labeled WPA2 in consumer terms.

3.4 The 4-Way Handshake

Before either TKIP or CCMP may be used, the STAs participating in the communication must perform the 4-Way Handshake. The 802.11i standard states that the 4-Way Handshake: [25]

- Confirms that a common shared secret named the Pairwise Master Key (PMK) exists on both peers.
- Confirms that this PMK is actual.
- Is used to derive Pairwise Transient Keys (PTKs) from the PMK. A PTK is used to encrypt unicast data.
- Installs the Group Transient Key (GTK) in the participating STA and AP. The GTK is used to encrypt broadcast data.
- Confirm the ciphersuite selection.

The 4-Way Handshake is performed by exchanging a series of 802.1X EAPOL-Key frames. This type of frame will be discussed in Section 4.3. Figure 3.2 shows the exchange of EAPOL-Key messages between STA and AP. Here we can observe the following steps: [25]

1. The AP transmits a random value ANonce to the STA.
2. The STA also generates a random value SNonce, and derives the PTK as follows:

$$\begin{aligned} PTK = \text{PRF-X}(PMK, \text{“Pairwise key expansion”}, Min(AA, SPA)||Max(AA, SPA)|| \\ Min(ANonce, SNonce)||Max(ANonce, SNonce)) \end{aligned} \tag{3.1}$$

Where AA is the AP MAC address and SPA is the STA MAC address. The PTK itself consists of three keys: the first 128 bits make up the Key Confirmation Key (KCK), the second 128 bits form the Key Encryption Key (KEK), and the remaining bits are used as the Temporal Key (TK). In summary, these keys are used for the following purposes:

- The KCK is used as a secret key for calculating the Message Integrity Code (MIC), which provides data origin authenticity.
- The KEK is used to encrypt the GTK in the next step.
- The TK is used for the actual encryption of data.

Finally, the STA sends the MIC and SNonce to the AP.

3. Using the received SNonce, the AP performs step 2 to derive the PTK and to acknowledge that there is no MITM by checking the MIC. Then a GTK is generated as a random number, encrypted with the KEK, and sent to the STA.
4. The STA acknowledges reception of the GTK, completing the 4-Way Handshake.

One of the authentication methods that is typically used in home networks is WPA-Pre-Shared Key (PSK), also known as WPA-Personal. Here, the user configures a password that must be entered when access to network resources is desired. From this password, a PSK is derived as: [29, 25]

$$PSK = \text{PBKDF2}(\text{password}, \text{SSID}, \text{ssidlen}, 4096, 256) \tag{3.2}$$

Where 4096 is the number of iterations and 256 is key bit size[4]. The resulting PSK of this algorithm is used as the 256 bit PMK. Note that this PMK is secret, and that it is never transmitted over WPA-Personal networks.

Another method to derive the PMK that is used in enterprise networks is WPA-Enterprise. This method will be discussed in detail in Chapter 4. For now, let us only consider WPA-Personal.

3.5 Temporal Key Integrity Protocol

TKIP is a protocol that was designed to prevent the various security exploits that were possible in WEP (see Section 3.2) while maintaining backwards compatibility with the same hardware that supports WEP. It provides the following improvements to WEP: [25]

- A MIC is added to provide data origin authentication and integrity protection over the DA, SA, Priority field, and plain-text data by using the Michael algorithm. The protection provided by the Michael algorithm can be circumvented as we will see in Section 3.7, but most APs that support TKIP provide countermeasures to prevent exploitation by an attacker.
- TKIP uses a TKIP Sequence Counter (TSC) to prevent replay attacks.
- Instead of one root key, a cryptographic mixing function is introduced that mixes a TK, the Transmitter Address (TA), and TSC. Hence, every packet uses a different key. This key is composed of a 24-bit IV and 104-bit RC4 key.

The TK that is used in the mixing function is derived from a secret Session Key (SK), which can be established on the STA and AP by performing the 4-Way Handshake from Section 3.4.

3.6 CCMP

CCMP is a cipher suite that uses the Counter mode with CBC-MAC (CCM) mode of the AES encryption algorithm. This mode is defined in Request For Comments (RFC) 3610, and as the name implies it uses

[4]The PBKDF2 algorithm performs multiple iterations in order to prevent brute-force attacks against weak passwords, as more iterations increase the time required to calculate a PSK.

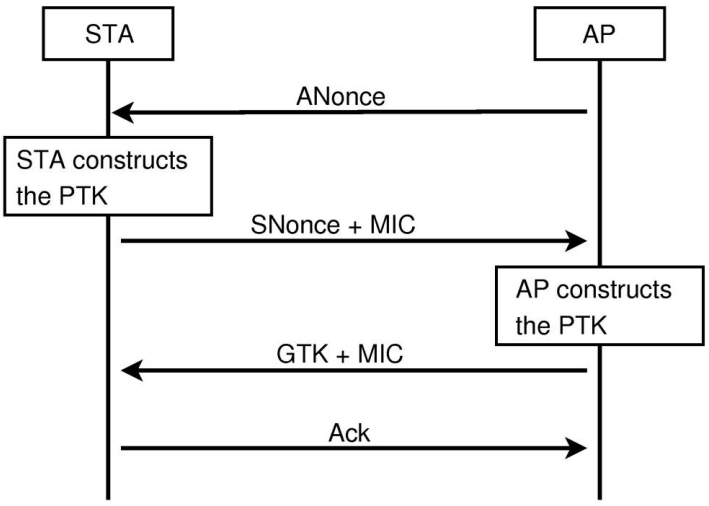

Figure 3.2: The 4-Way Handshake [61]

Counter (CTR) mode for encryption and Cipher Block Chaining Message Authentication Code (CBC-MAC) for authentication and integrity. AES uses a 128-bit key size and a 128-bit block size in context of CCMP. [25]

As with TKIP, CCMP uses a TK which is derived from the 4-Way Handshake. For each packet, a random nonce is derived from a 48-bit Packet Number (PN), the SA and the Priority field. This nonce is used together with the TK as input for the CCM algorithm, which produces the ciphertext and MIC. This algorithm is considered more secure than TKIP, because all RC4-related vulnerabilities are inherently not present in AES, and because the MIC from TKIP is susceptible to attack (see Section 3.7.1).

3.7 Attacks on 802.11i

Though many issues present in the WEP protocol were fixed with the introduction of 802.11i protocols, some minor vulnerabilities still remain. In this section, we show how these vulnerabilities may be exploited by an attacker.

3.7.1 Beck and Tews attack

The Beck and Tews attack is a variation on the ChopChop attack that can be used to send a small number of custom packets on a network that is secured with TKIP. The attack can be exploited under the following conditions: [57]

- The network supports the TKIP protocol.
- Internet Protocol (IP) version 4 is used with an IP range where most addresses are known to the attacker (for example 192.168.0.X).
- A long re-keying interval is used for TKIP, for example 3600 seconds.
- The network supports the IEEE 802.11e QoS features.

Under these conditions, the attack is performed as follows:

1. The attacker sniffs until an encrypted ARP Request or Response is encountered. These packets can be identified by their length, and their contents can be guessed except for the last byte of the IP source, the last byte of the IP destination, the 8-byte MIC and the 4-byte ICV.
2. The attacker launches a modified ChopChop attack on the last byte of the packet on a different QoS channel. Since each QoS channel uses its own TSC, this TSC has a high probability of being lower than the TSC of the replayed packet. Hence, the packet will be accepted.

3. If the attacker's guess for the last byte was incorrect, the packet will be dropped because the ICV is invalid. If it was correct, a MIC failure report will be sent by the AP. To prevent the countermeasures from triggering (see Section 3.5), the attacker needs to wait 60 seconds between each correct guess.
4. The attacker repeats step 3 to decrypt the 12 unknown bytes.
5. After the MIC key and plain-text of the packet are known to the attacker, the Michael algorithm can be reversed to yield the MIC key. Additionally, an AP to STA keystream is known. The attacker is now able to send a custom packet to the STA on every QoS channel that still has a lower TSC value than the captured packet.

In 2013, M. Vanhoef and F. Piessens discovered several new attacks based on the Beck & Tews attack. The authors show how more and bigger packets can be injected, and how any packet sent towards the client can be decrypted [58].

3.7.2 Dictionary attack

From Equation 3.1 we learned that the PTK is derived from the PMK, AA, SPA, ANonce and SNonce. These values are all exchanged in plain-text in the 4-Way Handshake, except for the PMK which is secret. However, in WPA-PSK the PMK is equal to the PSK, and from Equation 3.2 we can see that the only unknown is the password specified by the user. This means that an offline dictionary attack can be performed to obtain the PSK if a 4-Way Handshake is sniffed [31].

3.7.3 Insider attacks

In Section 3.7.2 we discussed that the PTK is derived from the PMK. In WPA-PSK authenticated networks, this PMK is not unique for every user. Hence, if an attacker is in possession of the PMK, they can eavesdrop on any user as long as the 4-Way Handshake between the AP and this user's STA has been captured.

3.7.4 WPS side-channel attack

Wi-Fi Protected Setup (WPS) is a technology that offers a variety of methods to easily configure an AP. One of these methods is the WPS-Personal Identification Number (PIN) method, where the user is required to enter an 8-digit PIN number to gain access to the AP's configuration.

In 2011, Stefan Viehböck discovered two design flaws in WPS-PIN: [59]

- No authentication is required apart from providing the PIN, making this method vulnerable to brute-force attacks.
- If an incorrect PIN is provided by the user, the AP will send an EAP-NAK[5] error message which indicates whether the first or the second half of the PIN was incorrect. This reduces the number of possibilities from $10^8 = 100\,000\,000$ to $10^4 + 10^4 = 20\,000$. Additionally, the 8th digit is always a checksum of the 7 previous digits, further reducing the number of possibilities to $10^4 + 10^3 = 11\,000$.

Given these design flaws, an attacker can brute-force the PIN in about 4 hours using a tool such as `reaver`[6]. Some APs implement a lock-down to prevent this kind of attack, but not all vendors made the lock-down interval long enough to make the attack infeasible.

[5] EAP-NAK messsages will be discussed in Section 4.2.2.
[6] Reaver can be downloaded from `https://code.google.com/p/reaver-wps/`.

Chapter 4

IEEE 802.1X Port Based Authentication

The IEEE 802.1X standard specifies the architecture, functional elements and protocols that support port based authentication of clients in WLANs, LANs or MANs. The goal of this standard is to regulate access to these networks and guard against transmission or reception by unidentified or unauthorized parties [26].

To achieve the above goal, the standard specifies the use of the Extensible Authentication Protocol (EAP). Though EAP can be used with any link-layer protocol, we will focus on the usage of EAP within WLAN environments in this thesis. When a WLAN environment is secured with EAP, it is also known in consumer terms as a WPA-Enterprise or WPA2-Enterprise network.

WPA2-Enterprise networks provide a seperate username and password login for each user through usage of the EAP protocol, instead of a single pre-shared key for the entire network as in WPA2-PSK networks (see Section 3.4). Since different users may require different access rights on the network, access may need to be revoked to former employees, or since the password may unintentionally leak to unauthorized parties when using a pre-shared key, we can naturally see why WPA2-Enterprise is the method of choice for enterprise networks.

In the following sections we will take a look at the mechanisms behind the EAP protocol and compare different EAP methods in terms of security and privacy. Finally, we will discuss and compare some existing attacks on WPA2-Enterprise networks.

4.1 Communicating entities

We can distinguish two types of communicating Port Access Entities (PAEs) in 802.1X: the Supplicant and the Authenticator. A third commonly used entity is the Authentication Server (AS).

- *The Supplicant* or client is the entity that wants to connect to the network, and hence requests authorization. The Supplicant can only send data packets (DHCP, ARP, HTTP, etc.) to the network after one is authenticated to the port. Examples of Supplicants are laptops, smartphones or other WiFi-capable devices.
- *The Authenticator* or Network Access Server (NAS) functions for the larger part as an intermediary between the Supplicant and the AS. It only orchestrates the exchange of authentication data on the link layer between Supplicant and AS, and does not take authentication related decisions. After successful authentication, the Authenticator sets up a secure communication channel with the Supplicant, so network resources can be accessed. This is done through the derivation of WEP, WPA or Wi-Fi Protected Access 2 (WPA2) keys on the Authenticator and Supplicant (see Section 3.4).
- *An Authentication Server* is used to provide the actual authentication service to the Authenticator over higher level protocols, such as Remote Authentication Dial In User Service (RADIUS) and

Diameter. This entity can also be co-located with the Authenticator itself [26].

The ports in port-based authentication can be either in an authorized state or unauthorized state, depending on the credentials provided by the user. Logically, ports are further divided into two entities: the uncontrolled and controlled port. Initially only EAP over LAN (EAPOL) data can be sent over the uncontrolled port, and only after the supplicant successfully authenticates, the controlled port will be opened. Internal network resources will then become available to the user (Figure 4.1). Note that in WLAN networks – because physical ports do not exist – virtual ports are used, but the port authentication principle remains the same.

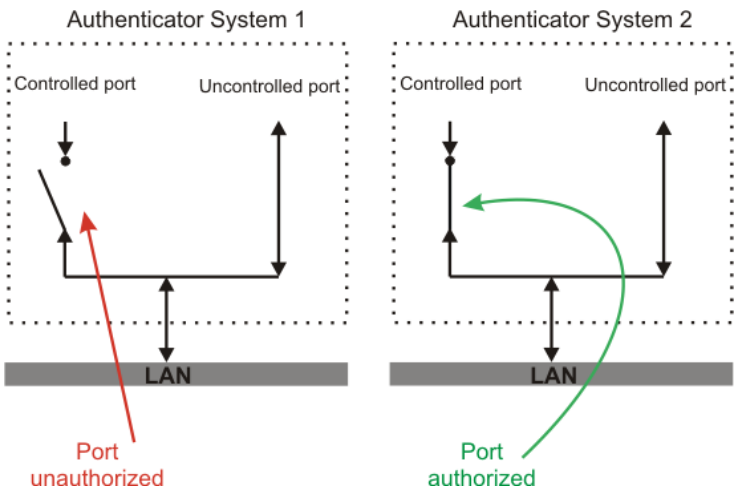

Figure 4.1: Logical entities of a port[56]

Figure 4.2 shows an example setup where the Supplicant gains access to restricted resources after being authenticated. The diagram also shows all protocols involved in 802.1X authentication. These protocols will be detailed further in the coming sections as their understanding is crucial for identifying security weaknesses.

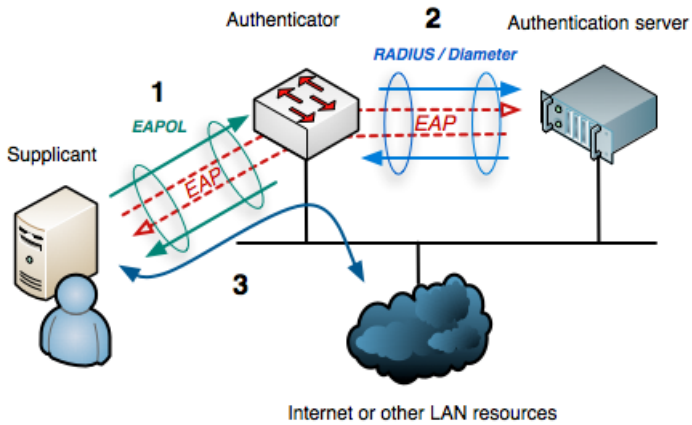

Figure 4.2: Protocols and communicating entities in an 802.1X network[64]

4.2 The Extensible Authentication Protocol

EAP is a framework protocol that supports many different types of port authentication methods, called EAP methods. The protocol is designed to run on top of the data link layer, and hence it does not require IP in order to operate. The protocol stack is shown in Figure 4.3. Note that EAP itself is encapsulated by the EAPOL protocol. We will discuss EAPOL in detail in Section 4.3.

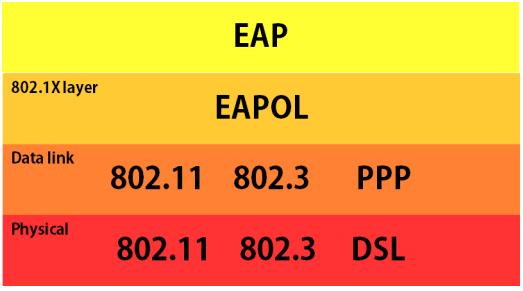

Figure 4.3: EAP protocol stack

Originally the protocol was used exclusively over Point-to-Point Protocol (PPP) or Ethernet wired networks [11], but its usage was later extended to 802.11 wireless networks [3]. The basic architecture of EAP is shown in Figure 4.4.

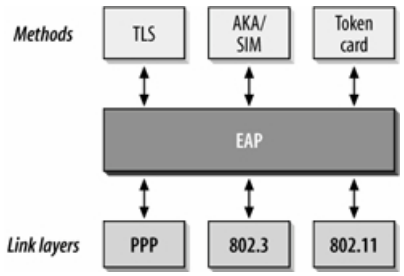

Figure 4.4: Basic EAP architecture[21]

4.2.1 Packet structure

An EAP packet consists out of a *Code*, *Identifier*, *Length* and *Data* field (Figure 4.5). Here, the *Code* field defines whether the EAP packet serves as a Request (1), Response (2), Success (3) or Failure (4) message. The Identifier field can be seen as a sequence number; it indicates retransmissions and matches requests with their reponses. The Length field indicates the size of the packet. Finally, the Data field has a generic purpose depending on the used EAP method (Section 4.5), and contains a variable number of bytes. In EAP Requests and EAP Responses, the first byte of the Data field serves as a *Type* field for identifying the type of EAP Packet. The remainder of the Data field is thereupon denominated *Type-Data*.

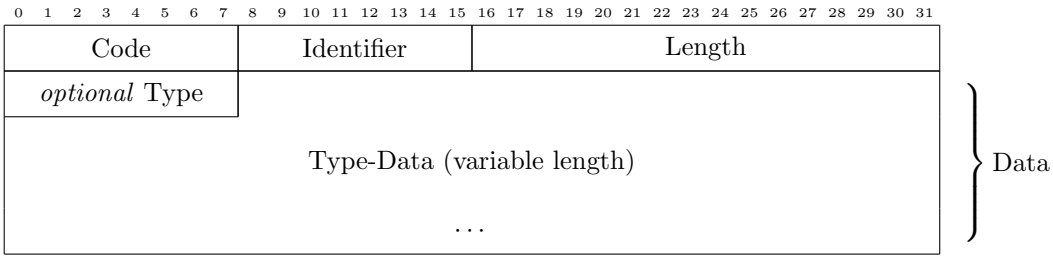

Figure 4.5: EAP packet structure and field sizes in bits

When experimenting with the EAP protocol by sending custom packets, it is important that all of the above fields are present. Otherwise, a Supplicant that respects the 802.1X Standard will drop the packet. It is also required that the EAP packet be encapsulated properly depending on the context in which it is used. For example, between Supplicant and Authenticator on a LAN, EAP packets must be encapsulated within an EAPOL frame whereas between Authenticator and a RADIUS AS, EAP packets must be encapsulated in RADIUS packets. Both EAPOL and RADIUS will be explained more in detail in Sections 4.3 and 4.4.

4.2.2 Authentication procedure

Most EAP methods use the same principal steps in order to authenticate a Supplicant to the AS. As the first step, the Supplicant listens for an Identity Request (Type = 1) from the Authenticator. Alternatively, the Supplicant can transmit an EAPOL Start packet to trigger the Identity Request transmission by itself. Upon receiving an Identity Request, the Supplicant should reply with an Identity Response containing the Network Access Identifier (NAI) of the user. This NAI has the format "`username@realm.com`", and will be used by the AS to look up the corresponding RADIUS realm and user credentials.

In the next step of the authentication procedure, the AS will send a method Request message containing the desired authentication method, depending on the configuration of the RADIUS server. The Supplicant can agree to authenticate using this method or send an EAP-Negative-Acknowledgment (NAK) Response (Type = 3). In the latter case, the Supplicant will suggest to use a different EAP method by providing this method's Type value in the Data field. When both parties agree on the used method, the authentication procedure continues as defined by the method specification. In Section 4.5, some interesting methods will be detailed further.

After the port authentication is acknowledged by the AS, the Supplicant and AS derive the Master Session Key (MSK) through a procedure defined in the used EAP method. The AS then derives the PMK and sends it to the Authenticator over a secure channel. On the Supplicant, the PMK is derived from the MSK as well. Finally, the Authenticator will send an EAP-Success packet (Code = 3) to the Supplicant, and a 4-Way Handshake (see Section 3.4) is performed between Supplicant and Authenticator in order to derive the PTKs. Note that the PMK is never sent over the air between these entities. The PMK can be cached for fast reauthentications.

Should the authentication method fail for any reason, the port remains unauthorized and an EAP-Failure packet (Code = 4) should be sent instead. When the Supplicant wants to log off, an EAPOL-Logoff message can be sent to unauthorize the port. The full authentication procedure is summarized in Figure 4.6. Table 4.1 gives an overview of some relevant EAP Code and EAP Type combinations.

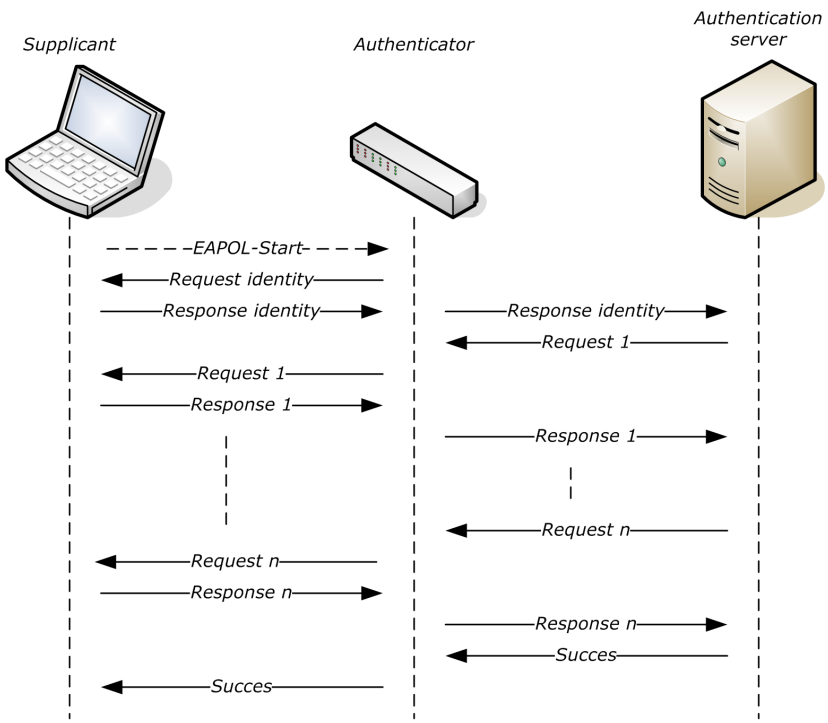

Figure 4.6: Simplified EAP authentication procedure[62]

EAP Code	EAP Type	Description
1	1	Request Identity
2	1	Response Identity
2	3	Response NAK
1	4	Request EAP-MD5
2	4	Response EAP-MD5
1	6	Request EAP-GTC
1	17	Request EAP-LEAP
1	21	Request EAP-TTLS
1	25	Request EAP-PEAP
1	29	Request EAP-MSCHAPv2
3	N/A	EAP-Success
4	N/A	EAP-Failure

Table 4.1: Table of EAP Code and EAP Type combinations

4.3 EAP over LAN

As explained in Section 4.2.1, EAP packets sent over the LAN must be encapsulated in EAPOL frames. The structure of these frames is illustrated in Figure 4.7.

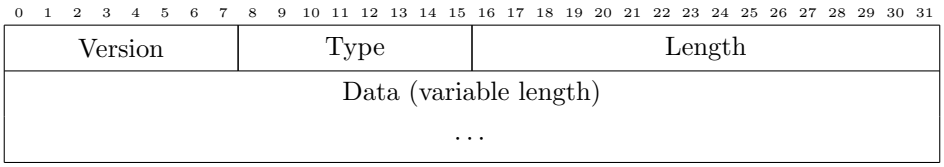

Figure 4.7: EAPOL packet structure and field sizes

Note that the Type field in EAPOL packets is different from the Type field in EAP packets. The following EAPOL types are relevant to our research:

- EAPOL-EAP-Packet (0): the Data field of the EAPOL frame contains an encapsulated EAP packet.
- EAPOL-Start (1): frame sent by the Supplicant to discover available Authenticators and to trigger the start of port-based authentication.
- EAPOL-Logoff (2): frame sent by the Supplicant to the Authenticator to reset the controlled port to unauthorized state and log off the network.
- EAPOL-Key (3): contains cryptographic key information which is exchanged between entities after successful port authentication (see Section 3.4).

4.4 Remote Authentication Dial In User Service

Though it is possible to perform Authentication, Authorization and Accounting (AAA) completely on the Authenticator in 802.1X networks, in some situations it is desirable to have a dedicated AS for this task. Most enterprise networks have a dedicated AS that either implements the RADIUS or the newer Diameter protocol.

From a functional point of view, these protocols are largely the same. RADIUS is the older protocol, dating from RFC 2058, which was written in January 1997 [44]. The protocol runs on the application layer and uses User Datagram Protocol (UDP)/IP for communication with the NAS (see Figure 4.8). Diameter is a more recent protocol which supports more advanced features in comparison to RADIUS, including native transmission-level security, reliable transport by using Transport Control Protocol (TCP), server-initiated messages, capability negotiation, and more [19].

Despite Diameter's apparent advantages towards RADIUS, we will only consider RADIUS, because this protocol is widely implemented[1] and because both protocols are similar nevertheless. Furthermore, in context of 802.1X, the main objective is encapsulation of EAP packets, which happens in a similar fashion in both protocols.

RADIUS was initially designed to provide AAA management on dispersed serial lines and modem pools for users that wish to use a certain network service such as PPP, rlogin or telnet [45]. Nowadays EAP support has been added, so that RADIUS could be used in the context of 802.1X.

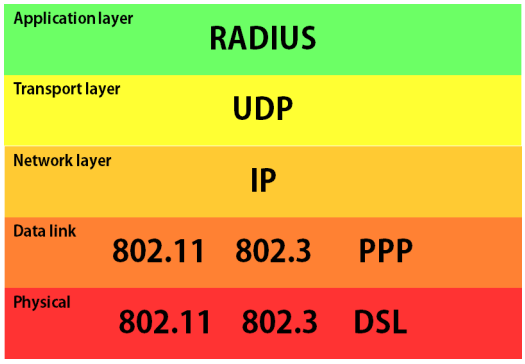

Figure 4.8: RADIUS protocol stack

AAA management is provided through the exchange of RADIUS attributes, called Attribute Value Pairs (AVPs), with the NAS. Later on, RFCs 3579 and 3580 added new guidelines and AVPs to support EAP encapsulation [4, 14]. The packet structure of an AVP is given in Figure 4.9.

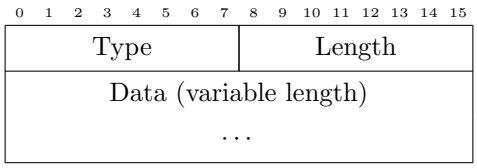

Figure 4.9: AVP structure and field sizes

Two AVPs that were added to support EAP over RADIUS are the EAP-Message and Message-Authenticator AVP. The EAP-Message AVP is used to encapsulate EAP messages, and the Message-Authenticator AVP is used to protect against spoofing and tampering of RADIUS messages [43]. In 802.1X networks, we can distinguish between four message types that can carry these AVPs from Authenticator to AS and vice versa: Access-Request, Access-Challenge, Access-Accept and Access-Reject messages:

- Access-Request (1): Access-Request messages are sent from the Authenticator to the AS. When the Authenticator decides that a certain EAP Response message needs to be handled by the AS, the EAP Response will be encapsulated within the EAP-Message AVP of the Access-Request RADIUS message. Additional AVPs, such as `Calling-Station-Id`, `User-Name`, `NAS-IP-Address`, etc. may also be included. Because Access-Request messages may query the AS for sensitive information such as user passwords, a 128-bit pseudo-random nonce called the Request Authenticator (RA) is

[1] Some of our experiments were performed on the campus 802.1X network "eduroam". Eduroam is an 802.1X network which is implemented in universities all over the world, and uses RADIUS for its AS hierarchy [8].

sent with each Access-Request message. The nonce will be used in the Access-Accept message to encrypt this sensitive information.

- Access-Challenge (11): Access-Challenge messages are structurally very similar to Access-Request messages. They carry EAP Requests encapsulated in EAP-Message AVPs from the AS to the Authenticator, and also contain a 128-bit pseudo-random RA field to prevent spoofing and tampering.
- Access-Accept (2): When EAP authentication is successful, the AS will send an Access-Accept message to the Authenticator, which contains the encrypted PMK[2] in the `MS-MPPE-Recv-Key` AVP (Section 3.4) and an encapsulated EAP Success message. The PMK will be used to encrypt traffic between the Supplicant and the Authenticator / AP after authentication. Since the derivation of this PMK requires keying material based on user credentials, and since the Authenticator has no access to this material, the RADIUS server has to send the PMK to the Authenticator. It is naturally essential that the PMK may not be eavesdropped during this process, since all communication between Supplicant and Authenticator can then be intercepted and decrypted by a third party. Therefore, a shared RADIUS secret S is used in combination with the RA from the Access-Request message to encrypt the PMK before it is transmitted to the Authenticator:

$$B = MD5(S \mid\mid RA) \tag{4.1}$$
$$C = B \oplus PMK \tag{4.2}$$

  Now C contains the encrypted PMK. This value is sent to the Authenticator, where the process is reversed to yield the plain-text PMK.
- Access-Reject (3): In the case of unsuccessful authentication, the AS will send an EAP Failure message encapsulated within an Access-Reject message. The Authenticator will forward the EAP Failure to the Supplicant.

4.5 EAP Authentication Methods

As mentioned earlier, EAP allows a variety of authentication methods called EAP methods. The Type field of EAP Requests and EAP Responses indicates which method should be used for authentication. This decision is negotiated between Supplicant and AS after the EAP Identity exchange.

We will now consider EAP methods that are popular and relevant for the attacks on EAP described in this thesis. These methods will be evaluated in terms of functionality and security, and in Section 4.6.3, we will take a look at which EAP Methods are still accepted by modern client devices. Most of the EAP methods discussed here have significant security vulnerabilities, yet they can be useful when encapsulated by a secure Transport Layer protocol such as TLS.

4.5.1 EAP-MD5

EAP-MD5 authentication is an EAP method based on Challenge Handshake Authentication Protocol (CHAP) [3] and is specified in paragraph 5.4 of RFC 3748. It has an EAP Type of 4. The authentication procedure for this EAP method is straightforward: [52]

1. The AS sends a challenge message to the Supplicant.
2. The Supplicant responds with a 16 byte Message Digest 5 (MD5)-hash over the Identifier field, shared secret and the challenge value.
3. The AS calculates the same MD5 value analogous to the previous step, and compares the result. If the values match, the authentication is acknowledged.
4. At random intervals, a new challenge is sent and the process is repeated.

The EAP-MD5 RFC states that all EAP implementations *must* support EAP-MD5. However, the Supplicant can freely decide to NAK this method, meaning a different method has to be chosen by

[2] In some RFC's, the PMK is also referred to as the Master Key (MK), which may be confusing.

the Authenticator. In Section 4.6.4, we compare the behaviour of different devices when an EAP-MD5 challenge is transmitted to them.

Currently, the EAP-MD5 algorithm is rarely used because of significant security vulnerabilities [2]. An attacker could capture the MD5 challenge and challenge response in order to launch a dictionary attack on the secret. Another contributing factor is that dynamic key derivation is not supported with EAP-MD5. Finally, EAP-MD5 does not provide mutual authentication, meaning an attacker could set up a rogue AS and trick a victim user into providing their credentials.

4.5.2 EAP-GTC

One of the simplest EAP methods is EAP-Generic Token Card (GTC), which is also defined in RFC 3748 as EAP Type 6. The Authenticator begins by sending an EAP Request message where the Type-Data can contain a displayable message. The EAP Response coming from the Supplicant contains plain-text data read from a token card, hence the name GTC. As with EAP-MD5, the Supplicant can also reply with an EAP-NAK instead of an EAP Response message [3].

Because all data is sent (optionally hashed) in plain-text, EAP-GTC should always be used with the protection of a TLS tunnel by using for example PEAP (see Section 4.5.6). If not, an attacker can eavesdrop on the user's credentials.

4.5.3 MSCHAPv1

MSCHAPv1 is a Microsoft proprietary CHAP dialect defined in RFC 2433. The protocol was designed for use over PPP, and cannot be implemented as standalone method for EAP, because there is no EAP method Type reserved for MSCHAPv1. In general, MSCHAP is similar to EAP-MD5, with the exception that MSCHAP: [70]

- Provides additional mechanisms for password changing and authentication retries.
- Defines a set of reason-for-failure error codes.

The authentication algorithm used can be summarized as follows:

1. Receive a random Challenge C from the other peer.
2. Calculate the `NtPasswordHash`. This is identical to an Message Digest 4 (MD4) hash calculation over the unicode or `wchar` password of the user.
3. Use the `NtPasswordHash` to generate three Data Encryption Standard (DES) keys k_0, k_1 and k_2 where:

$$\begin{aligned} k_0 &= \texttt{NtPasswordHash}_{[0:7]} \\ k_1 &= \texttt{NtPasswordHash}_{[7:14]} \\ k_2 &= \texttt{NtPasswordHash}_{[14:16]} \mid\mid \texttt{00:00:00:00:00} \end{aligned} \tag{4.3}$$

4. Generate the Challenge Response R using k_0, k_1, k_2 from the previous step and the Challenge C from step one:

$$R = \mathrm{DES}_{k_0}(C) \mid\mid \mathrm{DES}_{k_1}(C) \mid\mid \mathrm{DES}_{k_2}(C) \tag{4.4}$$

   Where "||" stands for the concatenation operator. The result of this operation is 16-byte Challenge Response value.

5. Send the Challenge Response to the other peer. If this peer calculates the exact same Challenge Response, the authentication is acknowledged.
6. After the authentication is complete, an EAP peer can use keying material from the method in order to dynamically generate Microsoft Point-to-Point Encryption (MPPE) RC4 keys for subsequent communication. However, since MSCHAPv1 is never used in wireless networks, this key derivation process ([67]) will not be detailed further in this thesis.

Though more secure than EAP-MD5, the above algorithm still has several security issues. A first issue is that since MSCHAPv1 exchanges are not encrypted, an attacker could eavesdrop on the challenge and challenge response. Using a dictionary or brute-force attack, the password of a victim user can then be derived (see Sections 4.6.7 and 4.6.8). An example of a tool that can be used in practice for executing a dictionary attack is the `asleap` tool, which was developed by Joshua Wright [1].

A second security issue is that MSCHAPv1 on itself does not provide mutual authentication between peers. This means that an attacker could impersonate the peer that sends the MSCHAPv1 request message in order to steal a user's credentials and become an active MITM (see Section 2.5.3).

4.5.4 MSCHAPv2

In response to the security vulnerabilities noted in Section 4.5.3 about MSCHAPv1, Microsoft created a second version of the MSCHAP protocol. This new version is more complex and not backwards compatible with MSCHAPv1 [69], but it benefits from added security measures such as default mutual authentication.

MSCHAPv2 can be used in the context of EAP, in which case it has Method Type 29[47]. EAP-MSCHAPv2 can be used as a standalone method, but it is generally used in conjunction with an outer authentication method such as PEAP. We will discuss PEAP in Section 4.5.6. The MSCHAPv2 authentication algorithm is performed as follows:

- The AS starts by generating a 16-byte random server challenge $C_s = Random16(seed)$ and sends it to the Supplicant.
- The Supplicant also generates a random 16-byte peer challenge C_p. Then the challenge response is calculated as $R_p = ChallengeResponse(Challenge(C_s), H)$, where the $ChallengeResponse$ function is identical to steps 3 and 4 from MSCHAPv1 (Section 4.5.3), $Challenge(C_s) = SHA1(C_p||C_s||U)[0:8]$, U is the username of the user, $H = MD4(Unicode(PW))$, PW is the password of the user and $[0:8]$ means the first eight bytes of the data. This challenge response is transmitted back to the AS, along with C_p and U.
- The AS calculates R_{check} analogous to R_p in step 4.5.4. R_{check} and R_p must match, or the authentication will fail.
- The AS calculates a peer challenge response

$$R_s = PeerResponse(MD4(Unicode(H)), M_1, R_p, Challenge(C_p), M_2) \tag{4.5}$$

  where M_1 is the literal ASCII string "Magic server to client signing constant" and M_2 is the literal ASCII string "Pad to make it do more than one iteration". This result is SHA1-hashed and sent to the Supplicant.
- The Supplicant authenticates the server, completing the MSCHAPv2 authentication. Since both Supplicant and Authenticator have to know the NtPasswordHash in order to mutually authenticate, it should be impossible for an attacker to set up a rogue Authenticator.

After successful authentication, a MK is derived as follows: [68]

$$MK = SHA1(MD4(Unicode(H))||R_p||\text{"This is the MPPE Master Key"}) \tag{4.6}$$

From the MK, the `MPPE-Send-Key` and `MPPE-Recv-Key` are derived as shown in Table 4.2.

	MPPE-Send-Key	**MPPE-Recv-Key**
Server	$SHA1(MK\|\|SHSpad_1\|\|$ "On the client side, this is the receive key; on the server side, it is the send key." $\|\|SHSpad_2)$	$SHA1(MK\|\|SHSpad_1\|\|$ "On the client side, this is the send key; on the server side, it is the receive key." $\|\|SHSpad_2)$
Client	$SHA1(MK\|\|SHSpad_1\|\|$ "On the client side, this is the send key; on the server side, it is the receive key." $\|\|SHSpad_2)$	$SHA1(MK\|\|SHSpad_1\|\|$ "On the client side, this is the receive key; on the server side, it is the send key." $\|\|SHSpad_2)$

Table 4.2: MSCHAPv2 MPPE key derivation

Here, $SHSpad_1$ is 40 times the octet `0x00` and $SHSpad_2$ is 40 times the octet `0xF2`. The resulting MPPE keys are either used directly for derivation of WEP / WPA2 keys, or as input for the Compound Session Key (CSK) as explained later in Section 4.5.6.

Note that in the previous algorithm, it is still possible for an attacker to capture the MSCHAPv2 credentials by setting up a rogue AP, because the user is authenticated before the AS itself. However, since after that the AS is required to authenticate itself to the Supplicant, it is no longer possible to become an active MITM as was the case with MSCHAPv1. To see why, observe that a rogue AS does not know $MD4(Unicode(H))$, and that therefore the peer challenge response cannot be calculated. Hence, the authentication cannot complete.

4.5.5 LEAP

Lightweight Extensible Authentication Protocol (LEAP) is a Cisco proprietary protocol for EAP authentication. It is also known as "Cisco EAP Wireless" and has EAP Type 17. LEAP supports dynamic generation of WEP keys and key rotation for enhanced security. Even though the protocol is proprietary, the mechanisms behind it were deduced by analysis of packets passed between an Aironet and Cisco ACS [32].

The authentication procedure of LEAP is shown in Figure 4.10, and is performed as follows:

1. The AS performs the MSCHAPv1 algorithm to authenticate the Supplicant. Denote the challenge sent by the AS as C_s and the response sent by the Supplicant as R_p.
2. In case of success, an EAP-Success message is sent from Authenticator to the Supplicant. Then, AS and Supplicant switch roles and repeat step 1. This time we denote the challenge sent by the Supplicant as C_p, and the response by the AS as R_s.
3. The AS derives the SK as

$$SK = MD5(MD4(Unicode(H))||C_s||R_p||C_p||R_s) \tag{4.7}$$

   where "||" is the concatenation operator, $H = MD4(Unicode(PW))$ and PW is the password of the user. The AS encrypts this value with the RADIUS secret (see Section 4.4). Then the encrypted message is sent to the Authenticator. The Supplicant also derives the SK, so this key can be used for WEP encrypted unicast communication. Finally, a random broadcast key is generated by the Authenticator and sent encrypted with the unicast key to the Supplicant.

Despite the extra security measures, LEAP still has a number of shortcomings. Because credentials are sent in the clear, they can easily be sniffed and captured by an attacker. Furthermore, since the calculation of R_p is relatively cheap, an offline dictionary attack can be performed on the MSCHAPv1 credentials in order to obtain a victim's `NtPasswordHash` [12]. If the password is weak, it can be retrieved in plain-text by looking up the hash in a rainbow table.

4.5.6 PEAP

Protected EAP (PEAP) is an EAP method designed for mutual authentication and session key generation in a roaming environment, and is specified in an Internet-Draft by H. Andersson et al. [38]. In essence, PEAP performs an second or "inner" EAP (for example EAP-MSCHAPv2 or EAP-GTC) authentication after establishing a TLS tunnel between Supplicant and AS. The authentication procedure of the latest version of PEAP, PEAPv2, is as follows:[38]

1. In Phase 1, the Supplicant and AS set up a TLS tunnel similar to the procedure described in RFC 5246 [17]. From the TLS master secret T_m, an MSK is derived via a one-way function:

$$K = \text{TLS-PRF-128}(T_m, \text{"client EAP encryption"}, N_c||N_s) \tag{4.8}$$

$$MSK = K[0:64] \tag{4.9}$$

   where N_c is the nonce generated by the TLS client and N_s is the nonce generated by the TLS server (see Section 2.6.1). The one-way `TLS-PRF-128` function prevents reversing of the MSK to retrieve the TLS master secret. Taking the first 64 bytes of the `TLS-PRF-128` output results in

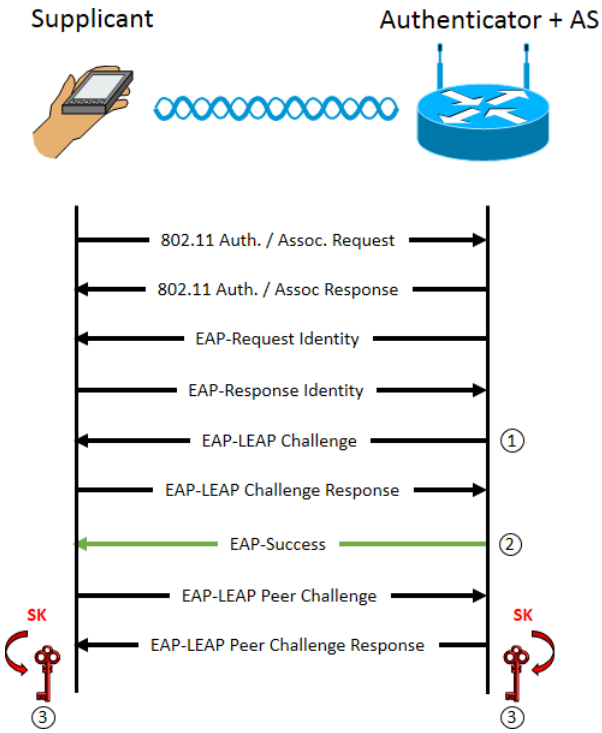

Figure 4.10: LEAP method mechanism

the MSK, which serves a comparable purpose to the SK from LEAP. When using cryptographic binding however, the MSK derived here is instead named the Tunnel Key (TK), and only 40 bytes are used instead of 64 [38].

2. Phase 2 is performed inside the TLS tunnel and implies the use of an inner EAP authentication method encapsulated in EAP-Payload Type-Length-Value (TLV) messages. MSCHAPv2 is often used and relatively secure for this purpose.

3. An EAP-Result-TLV exchange is performed between AS and Supplicant to indicate the result of the PEAP authentication. If cryptograhic binding is enabled, a Cryptobinding TLV is exchanged between AS and Supplicant as well. The `Compound_MAC` field of the TLV contains a value that proves both Phase 1 and Phase 2 of the PEAP authentication terminated at the same two EAP peers. This value is calculated as follows:

   - The TK is retrieved from step 1.
   - An Inner Session Key (ISK) is calculated where

$$\text{Peer } ISK = InnerMPPESendKey||InnerMPPERecvKey \tag{4.10}$$
$$\text{Server } ISK = InnerMPPERecvKey||InnerMPPESendKey \tag{4.11}$$

     The InnerMPPE keys are basically the MKs derived by the used inner EAP method. Though MPPE is in the field name, the keys are not encrypted with the RADIUS secret.

   - The Intermediate PEAP MAC Key (IPMK) Seed is calculated as

$$IPMKSeed = \text{“}InnerMethodsCompoundKeys\text{”}||ISK \tag{4.12}$$

     This seed will be used later in the Pseudo-Random Function+ (PRF+) function.

   - Now the IPMK itself is calculated by combining the TK and IPMK Seed:

$$Result = \text{PRF+}(TK[0:40], \text{IPMK Seed}) \tag{4.13}$$
$$IPMK = Result[0:40] \tag{4.14}$$
$$CMK = Result[40:60] \tag{4.15}$$

The PRF+ function performs a number of `HMAC-SHA1` operations on the key using the IPMK Seed. The Compound MAC Key (CMK) is used as the key for another `HMAC-SHA1` operation, which will result in our final value, the `Compound_MAC`.

Then the CSK is calculated as such:

$$CSK = \text{PRF+}(IPMK, \text{“}SessionKeyGeneratingFunction\text{”}, 128) \tag{4.16}$$

Finally, the CSK is split into parts of 32 bytes to serve as the final MSK. Table 4.3 shows which keys correspond to the MPPE send and receive keys. If cryptographic binding is not used, the MSK from step 1 is split analogously, instead of the CSK. The resulting MPPE keys are sent encrypted to the Authenticator.

It should be mentioned that an Extended Master Session Key (EMSK) is also derived from the last 64 bytes of the CSK, but there is no use yet for this key in PEAP [38]. The EMSK is only derived because this is a requirement for EAP methods according to RFC 3748 [3].

4. Finally, an EAP-Success message is sent in plain-text to indicate the result of the authentication. The Authenticator decrypts the `MS-MPPE-Recv-Key` value and uses it as the PMK. Secure transmission of data can begin when the 802.11i 4-Way Handshake (see Section 3.4) between Authenticator and Supplicant is completed.

The above algorithm is summarized in Figure 4.11.

4.5.7 EAP-TTLS

The EAP-TTLS method was designed by Paul Funk and Simon Blake-Wilson, and is described in RFC 5281. This method was created to allow usage of legacy authentication protocols (PAP, MSCHAP, CHAP, etc.) within a TLS tunnel. Hence, the method is similar to PEAP, with the exception that:

- EAP-TTLSv0 uses the TLS record layer after the TLS handshake to exchange AVP messages containing *any legacy authentication method*, whereas PEAP performs a complete second *EAP method* authentication after the TLS handshake.
- Cryptographic binding is not supported in EAP-TTLSv0. The newer version, EAP-TTLSv1, has expired as a draft despite being more secure by implementing cryptographic binding, and is no longer supported since GnuTLS 3.0.0. The changelog reads:

  > “*libgnutls-extra: Inner application extension was removed. It was never standardized nor published as an RFC.*” [35]

The authentication procedure of EAP-TTLSv0 is as follows: [20]

- Identical to Phase 1 in PEAP, the Supplicant and AS set up a TLS tunnel for secure communication. The client must compare its configured certificate with the certificate that the server provided in order to authenticate the AS. Analogous authentication from client to server through certificates is optional.
- In Phase 2, the AS and Supplicant exchange AVP messages (see Section 4.4) over the TLS record layer to perform one-way or mutual authentication. Any authentication method (`PAP`, `MD5-Challenge`, `MSCHAP`, etc.) may be used here.
- When the previous steps successfully concluded, the method will derive the keying material, MSK, and EMSK as follows:

$$K = \text{TLS-PRF-128}(T_m, \text{“ttls keying material”}, N_c||N_s) \tag{4.17}$$

$$MSK = K[0:64] \tag{4.18}$$

$$EMSK = K[64:128] \tag{4.19}$$

Note that this is identical to PEAP, except for the literal string “ttls keying material” that is used as input for the `TLS-PRF-128` function. The `MPPE-Recv-Key` and `MPPE-Send-Key` can finally be transported to the Authenticator / AP as respectively the first 32 bytes and second 32 bytes of the MSK.

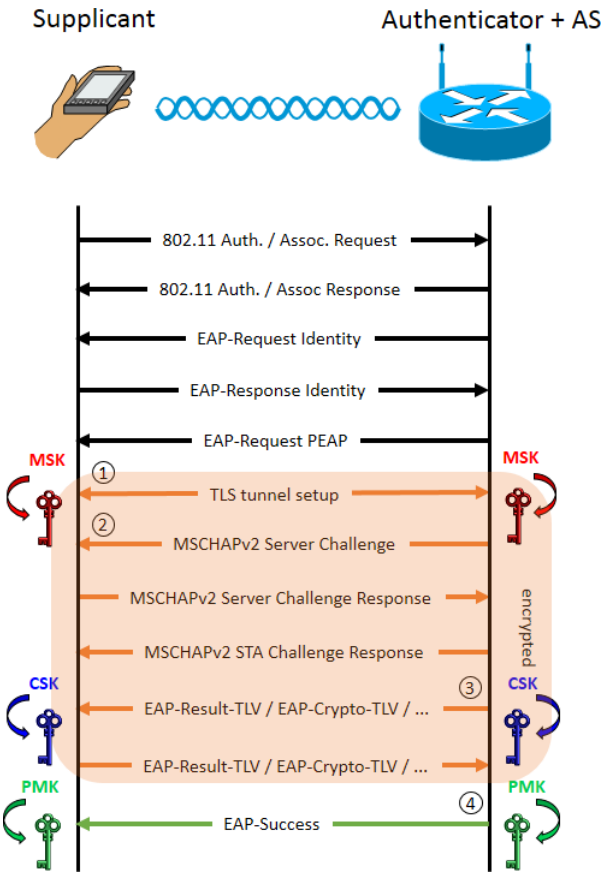

Figure 4.11: PEAP method mechanism

	$CSK[0:32]$	$CSK[32:64]$
EAP peer	MS-MPPE-Send-Key	MS-MPPE-Recv-Key
EAP server	MS-MPPE-Recv-Key	MS-MPPE-Send-Key

Table 4.3: MS-MPPE keys in PEAP

4.5.8 EAP-TLS

Another method that utilizes TLS is EAP-TLS, which was designed by D. Simon et al. Contrary to EAP-TTLS and PEAP however, this method does not tunnel an inner EAP method. Instead, the authentication is completely performed using TLS, and keys are derived from the TLS master secret. Hence, the authentication procedure is performed as follows: [51]

1. The Supplicant and AS set up a TLS tunnel (see Section 2.6.1). In EAP-TLS, *both* parties are required to present a valid certificate to each other.
2. After successful authentication, the MSK and EMSK are derived from the TLS master secret similarly to PEAP without cryptobinding:

$$K = \text{TLS-PRF-128}(T_m, \text{“client EAP encryption”}, N_c||N_s) \tag{4.20}$$
$$MSK = K[0:64] \tag{4.21}$$
$$EMSK = K[64:128] \tag{4.22}$$
$$\tag{4.23}$$

4.5.9 Security claim comparison

RFC 3748 states that each EAP method must include a section on the claimed security properties of that method. The set of terms used to distinguish these security properties are as follows: [3]

- Ciphersuite negotiation: The ability of the EAP method to negotiate the ciphersuite that protects the EAP exchange itself.
- Mutual authentication: The ability of the EAP method to authenticate the client to the server and vice versa.
- Integrity protection: The EAP method can protect against spoofing and tampering of the messages.
- Replay protection: The method protects against replaying packets.
- Confidentiality: EAP messages exchanged by the method are encrypted.
- Key derivation: The EAP method derives exportable keying material.
- Key strength: If the key strength is n bits, methods to brute-force the key should take 2^{n-1} operations of a typical block cipher.
- Dictionary attack protection: The method should protect against offline dictionary attacks.
- Fast reconnect: The ability of the EAP method to resume a previously established session in a faster way than full reauthentication.
- Cryptographic binding: The EAP method must assure that the same client and server participated in the EAP authentication during the complete exchange.
- Session independence: The method must assure that no passive or active attack may compromise the MSK or EMSK of prior or subsequent sessions.
- Fragmentation: The ability of the EAP method to fragment and reassemble messages that exceed the minimum Maximum Transmission Unit (MTU) of 1020 octets.
- Channel binding: The EAP method may support a mechanism that protects the exchange of channel properties. Examples of channel properties are endpoint identifiers such as `NAS-Identifier`, `Called-Station-Id`, `Calling-Station-Id`, etc.

The following table summarizes all the the security claims made by each discussed EAP method in their respective RFCs. Claims that were not mentioned are indicated with a "**?**".

	Cip. Neg.	Mut. auth.	Int. Prot.	Rep. Prot.	Conf.	Key der.	Key str.	Dict. prot.	Fast rec.	Crypt. binding	Sess. Indep.	Frag.	Ch. Bind.
EAP-MD5	✗	✗	✗	✗	✗	✗	✗	✗	✗	N/A	✗	✗	✗
EAP-GTC	✗	✗	✗	✗	✗	✗	✗	✗	✗	N/A	✗	✗	✗
EAP-MSCHAPv2	✗	✓	✓	✓	✗	✓	✓[3]	✗	✗	N/A	✓[4]	✗	✗
EAP-TTLSv0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✗
EAP-TLS	✓	✓	✓	✓	✓	✓	✓	✓	✓	N/A	✓	✓	✗
PEAPv2	✓	✓[5]	✓	✓	✓	✓	Variable	✓	✓	✓	✓	✓	**?**

Table 4.4: Comparison of security claims made by EAP methods

The MSCHAPv1 algorithm is not discussed here because there is no EAP-MSCHAPv1 RFC, nor is there an EAP method Type value reserved for this algorithm. LEAP is also not mentioned because this protocol is proprietary. However, we will determine whether these methods meet the security requirements discussed in Section 4.7.1 based on our experiments.

[3] Depends on passsword policy
[4] Depends on password policy
[5] Depending on the inner EAP method used and TLS certificate configuration.

4.6 Attacks on EAP

When examining the security claim comparison from Section 4.5.9, it is obvious that certain EAP methods are flawed by design in the sense that they were never meant to be used as secure wireless EAP methods. A prime example is EAP-MD5, which is viable when used in a low security, wired EAP environment. If used in a wireless environment however, usage of this method would be a major security risk.

Aside from the EAP methods that were designed to lack certain security features, there also exist EAP methods that became widely used prior to the discovery security flaws. An example is LEAP, which was found to be vulnerable to dictionary attacks after it became widely implemented by vendors and network administrators. These kind of vulnerabilities are naturally more attractive to an attacker.

In the next sections we will discuss some attacks against EAP methods that exploit unintended vulnerabilities. Because not all vendors implement the same set of EAP methods, we tested devices that use Linux, Windows 8, Mac OS X, iOS or Android operating systems. Finally, we will grade the EAP methods based on their accordance to a set of requirements, and compare them in a single overview.

4.6.1 Supplicant identity snooping

In section 4.2.2 we saw that the first step in the EAP authentication procedure is the exchange of a user's NAI between AS and Supplicant. The AS requires this identity in order to match the corresponding credentials of the user, and to decide whether this user is allowed access to internal network resources. Per protocol specification this exchange is not encrypted, which means it is possible for third parties to eavesdrop on the identity credentials.

A trivial attack to exploit this fact would therefore be to set a NIC in monitor mode and passively capture all EAP Identity Responses that pass by in a WPA2-Enterprise network. In addition to a user identity, EAP Identity Response messages also contain the sender MAC address in the 802.11 frame. Hence, as EAP Identities often contain a username or (part of) the real name of a user, an attacker can essentially perform a MAC address to user name mapping. A typical setup of this attack is shown in Figure 4.12.

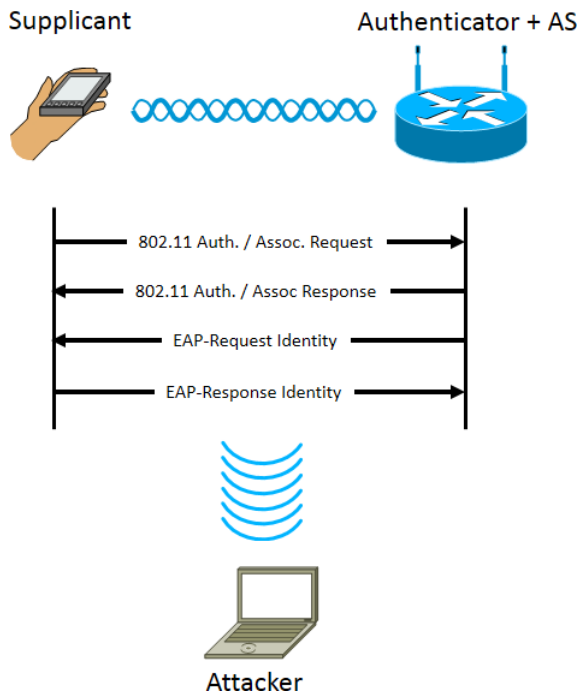

Figure 4.12: Passive identity snooping

Aside from passively sniffing Identity Responses, an attacker could also set up a rogue AP in order to actively send Identity Request messages to victim users. Most WiFi-capable devices send Probe Requests

for known APs (see Section 2.2.1), so identifying which SSID to spoof is a trivial task. Once the attacker starts sending Beacon frames and Probe Responses, most devices will automatically associate to it (see Section 2.5.4). Since no authentication happens prior to the NAI exchange, the victim will not be able to distinguish between a legitimate and a rogue AP. Iterating through all network SSIDs in the PNL of the victim's device in this way may yield more than one user identity, so this method might leak more sensitive information than the passive listening method. Furthermore, the attacker does not need to be in range of the legitimate network to perform this attack. Figure 4.13 illustrates this setup.

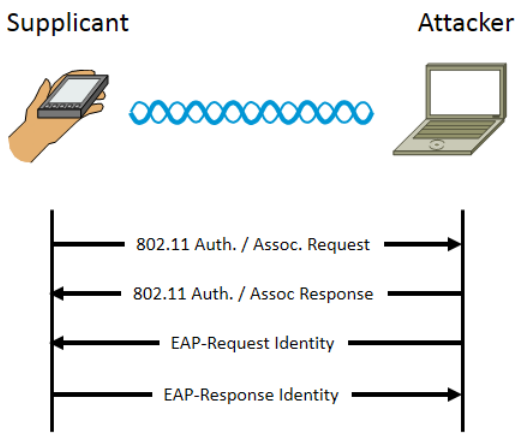

Figure 4.13: Active identity snooping

Because all EAP methods are required to perform the NAI exchange, they are inherently vulnerable to this attack. However, some EAP methods support *anonymous identities* or *outer identities*. When outer identities are used, the Supplicant will first exchange a known EAP Identity, e.g. "anonymous@example.com", with the AS. The real or inner identity of the user is only exposed after a TLS tunnel has been established. This approach effectively mitigates passive eavesdropping, since it is not possible for an attacker to monitor communication that happens inside the TLS tunnel.

4.6.2 Experiment: Supplicant identity snooping

In this experiment, the Python prototype tool from Section 2.5.6 was used to sniff Identity Requests in both an active and passive manner. This can simply be achieved with Scapy by sniffing EAP Response packets with Type "1".

We found that the most effective way to gather NAIs was to actively spoof the SSID of an existing network while being in range of this network. This caused closer devices to reassociate to our fake AP because of stronger signal strength. Note that most devices[6] automatically trust our fake AP, since no authentication of the Authenticator happens before the Identity exchange, and since the SSID is already known to the device. Passively sniffing Identity Responses was less effective, because we had to wait for a coincidential Identity exchange to take place between a legitimate AP and a device.

Further improvements to the above method could be made by actively transmitting spoofed EAPOL-Logoff packets or 802.11 Deauthentication frames to devices that are connected to the legitimate network. In the case of EAPOL-Logoff packets, the victim device will attempt to reestablish the 802.1X session with the NAS, and the Identity exchange may be passively sniffed. If we send a 802.11 Deauthentication frame instead, the victim device will scan for other APs immediately, which allows us to become an active MITM using a fake AP with better signal strength than the legitimate AP [48].

4.6.3 EAP method iteration

When a user connects to an EAP enabled network, their device stores the given credentials and EAP method in a configuration file. The next time the user is in range of the network, this *configuration*

[6] Based on our experiments from Section 2.5.3, we concluded that most devices automatically associate to a known SSID.

profile can be retrieved in order to automatically connect. However, some devices allow usage of EAP methods that were not advertised by the AS. We shall henceforth name these methods *alternative EAP methods*.

By using the same principle from the rogue AP Identity Request attack described in Section 4.6.1, we can check which alternative EAP methods are allowed by a certain device, when the device already possesses a configuration profile for a certain network. This is valuable information to an attacker, because many (older) EAP methods exist with security vulnerabilities that can be exploited if they are supported by the Supplicant device. For example, allowed EAP methods that send credentials in clear text or EAP methods that do not support mutual authentication could be interesting targets to an attacker when performing an active MITM.

To check whether an EAP method is an allowed alternative, we can spoof the SSID (see Section 2.5.3) and send an EAP Request message with the Type field set to the desired EAP method. This scenario is illustrated in Figure 4.14. If the Supplicant sends an EAP-NAK message in the response, we know that the method is not allowed as an alternative. In the case that the Supplicant sends an EAP Response of the correct type, we know we can use this method to authenticate the Supplicant.

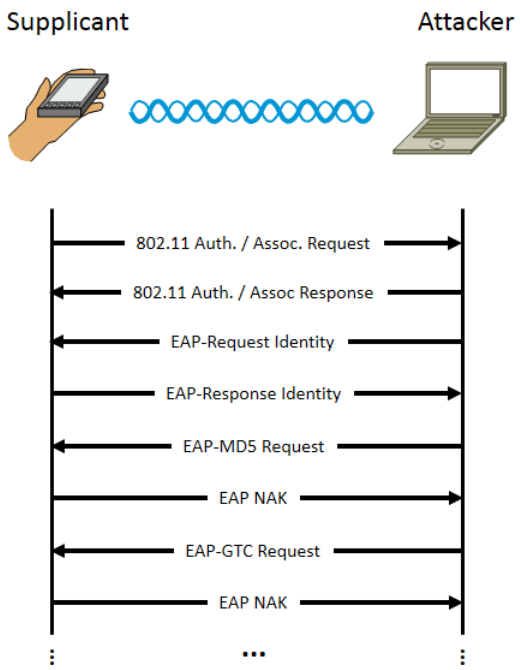

Figure 4.14: EAP method iteration

4.6.4 Experiment: EAP method iteration

With the above method, we can iterate over all EAP method types and derive a list of alternative EAP methods that are allowed *by default* for an existing PEAP network configuration profile. Here, "by default" means that the user only enters an SSID, username, and password in the configuration profile. We tested the allowed default alternative methods for some of the most popular Operating Systems (OSs). The results are shown in Table 4.5.

OS / EAP Type	EAP-MD5	EAP-GTC (inner)	MSCHAPv2 (inner)	LEAP	PEAP	EAP-TTLS	EAP-FAST
Linux 3.14.3-1-ARCH	✗⁕	✗⁕	✓⁕	✗⁕	✓⁕	✗⁕	✗⁕
Android 4.4.2	✗	✓	✓	✗	✓	✗	✗
Mac OS X 10.8.2	✗	✓⁕	✓	✓	✓	✓	✓
iOS 6.1.6 and iOS 7.1	✗	✓⁕	✓	✓	✓	✓	✓
Windows 8.1	✗	✗	✓	✗	✓	✗	✗
Windows Phone 8	✗	✗	✓	✗	✓	✗	✗

⁕ Depends on the tool used to connect to the network. The values shown are for the default NetworkManager (wrapper for `wpa_supplicant`) configuration. If the `wpa_supplicant` tool would be used without the `eap=` option, every EAP method would be accepted. Such configurations are highly insecure.

⁕ In a real life attack scenario, the default certificate pinning mechanism in these OSs will prevent changing the inner EAP method in PEAP.

Table 4.5: Alternatives for PEAP per OS, as derived from our experiment

Note that insecure methods such as EAP-GTC and LEAP are allowed by default on some devices. Fortunately, a user can configure their device to disallow alternative methods:

- Linux: Use a network client that disallows alternative methods by default. An example is NetworkManager.
- Android: Set the "Phase 2 authentication method" field when connecting to a network.
- Mac OS X: Install a "configuration profile". These will be discussed in Section 4.6.6.
- iOS: Identical to Mac OS X.
- Windows: Windows devices are fixed to PEAP and MSCHAPv2 *by default* for password based authentication. On Windows 8.1, a server certificate is required by default as well. Hence, using alternative methods is not possible.

4.6.5 EAP dumb-down attack

From Section 4.6.3 we were able to deduce which OSs support which EAP methods. We can exploit this knowledge by acting as a rogue AP in order to force the client to use the weakest of all supported EAP methods. This attack is known as the EAP dumb-down attack, and was introduced at RootedCON by Raul Siles in 2013 [49]. The attack may allow an attacker to steal plain-text credentials, encrypted credentials, or even become an active MITM.

The initial steps to perform the attack are similar to the EAP method iteration attack: we set up a rogue AP that spoofs the target network. When answering the client's Probe Request with a Probe Response, the client will associate automatically[7] to our AP, because at this stage the authenticity of the AP cannot be verified yet.

After association, the attacker first chooses one of the supported EAP methods, and sends the EAP Request message for this method to the Supplicant. More secure EAP methods will establish a TLS tunnel between the two parties before continuing the authentication. Then, the attacker can choose from the available inner EAP methods. A popular choice here from the attacker's point of view is to choose EAP-GTC, because if the Supplicant device accepts this EAP method, the user's credentials will be sent in plain text to the attacker. Note that this also allows an attacker to become an active MITM, because the secret password is then known. A less optimal scenario for the attacker is that only MSCHAPv2 is accepted, in which case only the challenge C_p and challenge response R_p are exposed. Nevertheless, the attacker can still retrieve the plain text password from these credentials, as we will see in Section 4.6.7. Figure 4.15 gives a graphical representation of the attack.

[7] This behaviour can be disabled by the user to prevent the attack from happening in an automated fashion.

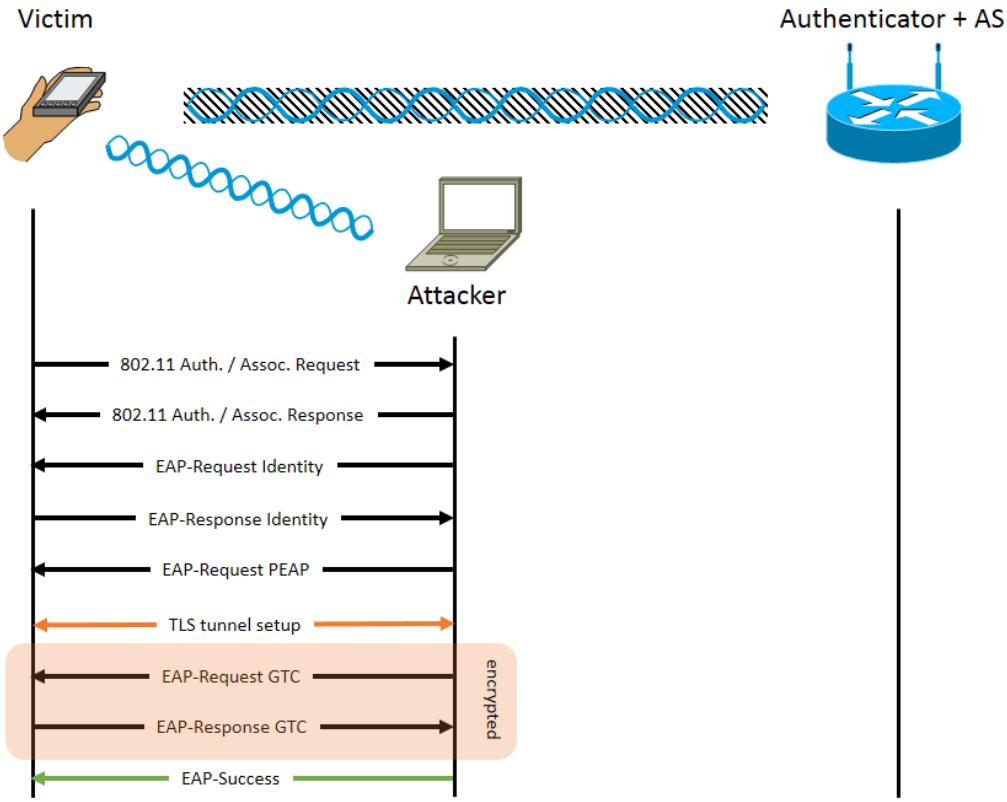

Figure 4.15: EAP dumb-down attack

4.6.6 Experiment: EAP dumb-down attack

To perform the dumb-down attack, we modified the source code of an open-source implementation of a RADIUS server[8] to force usage of a specific EAP method. In order to get clients to connect to us automatically, we also used the modified hostapd implementation that was discussed earlier in Section 2.5.3. We attempted to force the Supplicant to use PEAP with EAP-GTC in order get the plain-text credentials of the user. If the Supplicant would send a NAK message, we attempted to switch to MSCHAPv2 as a fallback to capture the MSCHAPv2 credentials instead of the plain-text credentials.

In a first test, the attack was tested in a lab environment on an Apple iPhone 4, a Samsung GT-S5570, a Google Nexus 7 2013, a Windows Phone 8, and a HP EliteBook 8530p. We wanted to see whether the attack still works, despite being reported by Raul Siles to all vendors in 2013 [49]. Before performing the attack, we let the device connect to a legitimate PEAP network using the default configuration to simulate a real-life situation. We then proceeded by spoofing this network and performing the dumb-down attack. Table 4.6 shows on which OSs the attack was successfully executed in a way that does not require user intervention.

OS	Vulnerable
iOS 6.1.6 (iPod Touch) and iOS 7.1 (iPhone 4)	✗
Android 2.3.6 (Samsung GT-S5570)	✓
Android 4.4.2 (Google Nexus 7 2013)	✓
Windows Phone 8	✱
Linux (HP EliteBook 8530p)	✱

Table 4.6: EAP dumb-down attack applicability test

[8]This open-source implementation is named FreeRADIUS and can be found at `http://freeradius.org`.

From this test we concluded that Android devices seem to be vulnerable to the attack, because alternative *inner EAP methods* are allowed by default on this operating system. Even EAP-GTC could be forced, which is the worst case scenario as the victim's credentials can be sniffed in plain-text in this case. Apple devices are not vulnerable, because certificate pinning is employed. This ensures that the device will not automatically connect unless the user explicitly accepts the fake certificate provided by the attacker's AS. Other devices (✱) are vulnerable to a lesser degree, in the sense that they connected to our fake AP, but did not dumb-down the EAP method. This means an attacker could still capture the MSCHAPv2 credentials and brute-force them in a later stage.

Fortunately all device vendors have features available that can mitigate the attack. However, some of these features are not enforced or enabled by default, so security is essentially put in the hands of the end user. The mitigation strategies are also different depending on the vendor:

- Apple: Apple devices offer several ways to mitigate the attack. A first is that the devices automatically use certificate pinning by default, which means that each time a new X.509 certificate is presented, the user has to manually accept this certificate. This prevents the attacker from executing the dumb-down attack automatically, since the user has to confirm the attacker's fake certificate before authentication proceeds. A second method is offered through configuration profiles (see Figure 4.16). Though this method is cumbersome[9], it allows a user to disable specific EAP methods and manually provide an X.509 ceriticate.
- Android: Android devices offer the functionality to manually set a certificate and inner EAP method[10]. This way a user can force their device to only connect to an AP when a valid certificate is presented, and fix the inner EAP method to a safe variant. However, configuring the inner EAP method and the certificate is not required by default.
- Windows: Windows devices only allow the PEAP method in combination with MSCHAPv2 (in case of a password based login) or EAP-TLS (in case of a smart card based login) [37]. Furthermore, a certficate can be set manually. On Windows Phone 8, setting this certificate is not a requirement by default, and hence a user's MSCHAPv2 credentials may still be exposed to an attacker when the certificate is not set. Microsoft is aware of this problem and recommends configuring a certificate [39].

A second test was performed in a crowded environment to see how many people are practically vulnerable to the attack outside a lab environment. Our implementation was run from a Raspberry Pi that was concealed in a backpack. We tested how many people would automatically connect to the "eduroam" network, and how many would provide their plain-text or MSCHAPv2 credentials.

We captured and stored all relevant EAP packets together with all Probe Requests in a MySQL database. Then, by analyzing the number of Probe Requests, we derived how many subjects were exposed to the attack. Table 4.7 shows how which SQL queries were used to calculate the exposure.

The "Subjects" column indicates how many unique MAC addresses sent a Probe Request. Similarly, "Notable subjects" gives the number of unique MAC addresses that sent a Probe Request, but with the constraint that the device should have sent at least 5 of them. This ensures the device was in range long enough to perform a full PEAP authentication. The "Subjects with eduroam" columns are identical, except we only consider the devices that connected to "eduroam" in the past.

[9] Creating a configuration profile requires the user to download a special tool, such as the iPhone Configuration Utility (`http://support.apple.com/kb/dl1465`). The user cannot make the same changes from within the iOS Graphical User Interface (GUI).

[10] Sometimes, the inner EAP method is also called the "Phase 2 method".

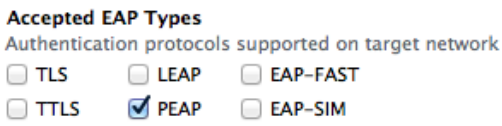

Figure 4.16: iDevice configuration profile example

Description	SQL Query
Subjects	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests GROUP BY mac) AS c;`
Notable subjects	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests GROUP BY mac HAVING COUNT(*) > 5) AS c;`
Subjects with eduroam	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests WHERE SSID = 'eduroam' GROUP BY mac) AS c;`
Notable subjects with eduroam	`SELECT COUNT(*) FROM (SELECT mac, COUNT(*) AS count FROM ProbeRequests WHERE SSID = 'eduroam' GROUP BY mac HAVING COUNT(*) > 5) AS c;`

Table 4.7: SQL queries used to calculate exposure to the dumb-down attack

The results from this test are shown in Table 4.8. The percentage of vulnerable devices is calculated by taking the sum of the MSCHAPv2 and EAP-GTC vulnerable devices, divided by the number of notable subjects that connected to "eduroam" at some point in time before. We can conclude that in total, 28 out of 99 devices (28%) are vulnerable to the EAP dumb-down attack if the target network is present in the PNL.

Location	Duration	Subjects	Notable subjects	Subjects w. eduroam	Not. subjects w. eduroam	MSCHAPv2 credentials	Plain-text credentials	Percentage vulnerable
UHasselt	2h36m7s	625	347	78	32	6	4	$\frac{10}{32} = 31\%$
UHasselt	1h56m0s	684	383	68	30	3	2	$\frac{5}{30} = 16\%$
EDM Research Center	3h59m17s	66	47	6	4	1	2	$\frac{3}{4} = 75\%$
Town café	1h56m23s	75	47	2	2	0	0	$\frac{0}{2} = 0\%$
Public transport	0h36m38s	249	86	10	4	2	1	$\frac{3}{4} = 75\%$
UHasselt + Bus	4h6m42s	961	466	56	27	3	4	$\frac{7}{27} = 26\%$

Table 4.8: EAP dumb-down attack test outside lab environment

4.6.7 MSCHAPv2 dictionary and brute-force attacks

We already saw in Section 4.5.4 that MSCHAPv2 is vulnerable to a dictionary attack. To see why, consider a scenario where an attacker managed to sniff an MSCHAPv2 exchange of messages. This may be accomplished by performing a passive MITM attack on the victim. The credentials in possession of the attacker at this point are C_s, C_p, R_p, and U. Recall that the challenge response R_p is calculated from the challenge as follows:

$$C = SHA1(C_p||C_s||U)[0:8] \tag{4.24}$$

$$k_0 = \texttt{NtPasswordHash}_{[0:7]} \tag{4.25}$$

$$k_1 = \texttt{NtPasswordHash}_{[7:14]} \tag{4.26}$$

$$k_2 = \texttt{NtPasswordHash}_{[14:16]} \;||\; \texttt{00:00:00:00:00} \tag{4.27}$$

$$R_p = \mathrm{DES}_{k_0}(C) \;||\; \mathrm{DES}_{k_1}(C) \;||\; \mathrm{DES}_{k_2}(C) \tag{4.28}$$

Notice that `NtPasswordHash` is the only unknown in the above equations. In a naïve approach, the attacker can iterate over a list of passwords and calculate a challenge response R using the `NtPassword-`

`Hash` derived from this guessed password. If the calculated value matches with R_p, the password guess was correct.

There are several methods to speed up this dictionary attack. A first performance increase can be achieved by precalculating the `NtPasswordHash` for every password in the dictionary. We could do this ourselves or use an existing `NtPasswordHash` rainbow table from the internet, since the `NtPasswordHash` is unsalted.

A second way to increase performance is to first brute-force k_2. Note that this key is only 2 bytes in size, so we can simply perform all 65536 combinations of $\text{DES}_{k_2}(C)$ and compare the result to the last 8 bytes of R. When k_2 is recovered, the two last bytes of the `NtPasswordHash` are revealed. With this knowledge, we only need to calculate R in its entirety when the last two bytes of our *guessed* `NtPasswordHash` match the first two bytes of the now known k_2. The "asleap" tool by Joshua Wright [1] is capable of performing this dictionary attack with both optimisations.

Aside from a dictionary attack, another method to recover the `NtPasswordHash` from MSCHAPv2 credentials was devised and commercialised by Moxie Marlinspike and Marsh Ray in 2012 [34]. This method, dubbed CloudCracker, uses a cloud-based array of Field-Programmable Gate Array (FPGA) devices to attack the weak 56-bit key strength of the DES keys. The duo proves how the strength of the challenge encryption can be reduced to a single DES operation. The FPGA array performs the algorithm presented in Listing 4.1 to quickly brute-force all possible k_0 and k_1 DES key combinations in under a day.

```
keyOne = NULL;
keyTwo = NULL;

for(int i = 0; i < 2^56; i++) {
        result = DESkey[i](plaintext);

        if(result == ciphertext1)
                keyOne = result;
        else if(result == ciphertext2)
                keyTwo = result;
}
```

Listing 4.1: CloudCracker cracking algorithm [34]

Though knowledge of the `NtPasswordHash` is sufficient to decrypt traffic[11], gain authorization to the network, or become an active MITM, note that one still has to perform a rainbow table lookup or brute-force attack on the hash in order to obtain the plain-text password of the user.

4.6.8 Experiment: MSCHAPv2 brute-force attack

One disadvantage of the asleap tool discussed in Section 4.6.7 is that this tool always requires a dictionary, since pure brute-force on the user's password is considered infeasible. We decided to challenge this claim by extending the tool with a threaded brute-force functionality, and perform an attack without usage of a dictionary. In this experiment we attempt to crack an 8-character random alphanumeric (`[a-zA-Z0-9]`) password to verify the feasibility of a brute-force attack on medium-length plain-text passwords. This makes the total amount of possible combinations:

$$62^8 = 218\ 340\ 105\ 584\ 896 \tag{4.29}$$

Consider now that we take one random combination x from this set. If our guessed x is random, the `NtPasswordHash` of x, which we will define as X from now on, will be random as well. Also, the last two bytes of the hash to guess, which we denoted earlier as H, are already known: these are equal to the

[11] Naturally, this assumes the MSCHAPv2 exchange itself was not encrypted by any means. For example in the case of PEAP, the attacker should be in possession of the TLS private key in order to capture the MSCHAPv2 exchange.

first two bytes of k_2. In case that $H[14] \neq X[14]$ or $H[15] \neq X[15]$, we know that X will never match H, and our partial calculation of `NtPasswordHash` was sufficient to check this. Hence, we only need to perform the entire calculation of R when $H[14] == X[14]$ and $H[15] == X[15]$. The chance that this event occurs is $\frac{1}{256} \cdot \frac{1}{256} = \frac{1}{65536}$ or roughly 0.0015 percent. Therefore, 99.9985 percent of the time a simple `NtPasswordHash` operation is sufficient to check whether our guess X is correct.

Our asleap extension divides all possible permutations of the 8-character password over a number of worker threads. Each thread performs an `NtPasswordHash` operation on its current permutation, and checks the result against the known k_2 bytes. Table 4.9 shows the performance of this algorithm when executed on different machines.

	Checks / s	Worst-case time
Intel(R) Core(TM) 2 Duo CPU P8600 @ 2.40GHz	7 047 175	358 days
Intel(R) Core(TM) i7-4770K CPU @ 3.50 GHz	35 235 878	72 days

Table 4.9: Brute-force attack on an 8-character password

From these results we concluded that with today's modern Central Processing Units (CPUs), it would be feasible for a determined attacker to brute-force the plain-text password on a single machine. The process can be further accelerated through usage of the Graphics Processing Unit (GPU), though we did not implement this technique. To give an idea of the potential speedup with GPU acceleration: a single AMD HD6990 GPU can check 20 853 000 000 MD4 hashes per second using the OCLhashcat tool [7]. This translates to finding an 8-character alphanumeric password worst-case in three hours, or a 9-character alphanumeric password worst-case in eight days.

If GPU acceleration is not an option, one could consider distributed computing to speed up the cracking process. Cracking time will then decrease in a roughly linear fashion with the number of machines used.

4.7 Privacy of EAP

Now that we have a good understanding of the operations and vulnerabilities of different EAP methods, we can assess the impact on user privacy by evaluating these aspects. The severity of an EAP method vulnerability is determined by the impact on the security requirements and security claims (see Section 4.5.9) made by the method. At the end of this section we will grade all EAP method based on this information.

4.7.1 Requirements

Before we can quantify an EAP deployment's security, we have to define requirements that should be met in a secure deployment. Fortunately, these requirements are already defined in RFC 4017 [55], and further detailed in a paper by R. Dantu et al. [15]. These papers distinguish between 3 types of requirements: *mandatory* requirements, *recommended* requirements and *optional* requirements.

- **Mandatory requirements**
  1. Generation of symmetric keying material: the EAP method must export a MSK of at least 64 octets, and an EMSK of at least 64 octets. This corresponds to the *key derivation* and *key strength* security claims.
  2. Mutual authentication
  3. Dictionary attack protection
  4. MITM attack resistance: the EAP method must resist MITM attacks by implementing *cryptographic binding*, *integrity protection*, *replay protection*, and *session independence*.
  5. Protected ciphersuite negotiation

- **Recommended requirements**
  1. Fragmentation support
  2. Identity hiding: the EAP method should have support for anonymous EAP identities.
- **Optional requirements**
  1. Channel binding
  2. Fast reconnect

A requirement is met when all of the matching security claims from Section 4.5.9 are present.

4.7.2 Evaluation

Table 4.10 shows the evaluation of all discussed EAP methods based on the requirements and discovered vulnerabilities. EAP methods are penalized based on these vulnerabilities. Penalty points for an EAP method are calculated as: *severity* $*$ *importance* for each column. Here, the severity can be none (0), low (1), medium (2) or high (3). The importance can be optional (1), recommended (2), or mandatory (3). In the table, these values are represented as colors, where a darker shade indicates a higher value.

	Sym. Key. Mat. (3)	Mut. auth. (3)	Dict. Prot. (3)	MITM Resist. (3)	Prot. Ciph. Neg. (3)	Fragmentation support (2)	Identity hiding (2)	Channel binding (1)	Fast reconnect (1)
EAP-MD5	(3)	(3)	(2)	(3)	(3)	(3)	(3)	(3)	(3)
EAP-GTC	(3)	(3)	(3)	(3)	(3)	(3)	(3)	(3)	(3)
EAP-MSCHAPv2			(1)	(1)	(3)	(3)	(3)	(3)	(3)
PEAPv2		(1)	(1)	(2)			(1)	(3)	
EAP-TTLSv0		(1)	(1)	(2)			(1)	(3)	
EAP-TLS							(1)	(3)	

Table 4.10: Security of EAP methods based on vulnerabilities

We will now briefly motivate our choices of penalty points for each EAP method.

- **EAP-GTC** *63 points*: EAP-GTC does not meet any security requirement.
- **EAP-MD5** *60 points*: Dictionary attacks are slightly more difficult to perform on EAP-MD5, because the shared secret is MD5-hashed. However, this hash is not salted, which means it can easily be cracked using rainbow tables in case the attacker desires a plain-text value.
- **EAP-MSCHAPv2** *33 points*: MSCHAPv2 offers generation of symmetric keying material and mutual authentication. However, dictionary and MITM attacks can still be performed.
- **PEAP** *14 points*: PEAP adds protected ciphersuite negotiation, fragmentation support, and fast reconnect to meet extra requirements. However, mutual authentication via TLS or the inner EAP method is optional, which means execution of a MITM attack, dumb-down attack or relay attack is simplified. Similarly, identity snooping attacks may be performed as identity hiding is not a requirement.
- **EAP-TTLS** *14 points*: Analogous to PEAP, though it should be noted that the lack of crypto-binding may have security implications in context of relay attacks.
- **EAP-TLS** *5 points*: EAP-TLS completely relies on certificates for authentication, which eliminates the possibility of brute-force attacks. Furthermore, mutual authentication between Supplicant and AS is *required* in EAP-TLS, which prevents MITM attacks and relay attacks. However, identity hiding is still optional.

From this evaluation we can conclude that EAP-TLS appears to be the most secure EAP method, because of the mutual authentication requirement and certificate-based authentication. Despite its advantages, a major drawback is that the implementation of EAP-TLS requires a lot of maintenance, as valid certificates need to be installed on the AS and on all Supplicants.

Chapter 5

Experiment: LEAP relay attack

In January 2014, we discovered a novel vulnerability in Apple devices that allows an attacker to gain unauthorized access to PEAP WPA2-Enterprise networks by impersonating a victim user. The vulnerability is caused by the sharing of credentials over EAP methods on Apple devices, and by a flaw in the MSCHAPv2 protocol. By performing a similar attack to the MITM attack proposed by N. Asokan et al. [6], these vulnerabilities can be exploited. A more detailed explanation will be provided in Section 5.2. Aside from the vulnerability itself, we will also discuss the disclosure procedure and proof-of-concept implementation in this chapter.

5.1 Disclosure procedure

After discovery of the vulnerability, we chose to responsibly disclose it in order to minimize the impact on enterprise network security. The initial report was made to Computer Emergency Response Team (CERT) Belgium. They contacted Apple on our behalf so we could remain anonymous.

In the meantime, we submitted a paper that details the discovered vulnerability to WiSec 2014. We also contacted Ars Technica in an attempt to warn the public about which devices would be vulnerable under which conditions, without giving away practical details required to exploit the vulnerability. Unfortunately, we never received a response to this report.

Some time later we contacted CERT US, because we still had not received news about a fix date from Apple. This fix date is currently still unknown.

The full vulnerability disclosure timeline can be found below:

- January 2014: Discovery of the vulnerability and development of proof-of-concept implementation.
- January 31, 2014: Reported existence of vulnerability to CERT.be
- February 5, 2014: Fully detailed report sent to CERT.be. This report was forwarded to Apple.
- February 20, 2014: Vulnerability acknowledged as distinct issue by Apple. Fix date unknown.
- March 18, 2014: Paper detailing the vulnerability submitted to WiSec 2014.
- March 24, 2014: Contacted Ars Technica about the vulnerability in an attempt to warn users, without giving too much information about the exploit. No response received.
- April 17, 2014: Existence of vulnerability reported to CERT.org. Immediate reply requesting more details.
- April 18, 2014: Some details provided to CERT.org.
- May 9, 2014: Paper conditionally accepted.
- May 28, 2014: Acknowledgement of the vulnerability by CERT.org. Fix date unknown.
- May 28, 2014: Paper revision submitted.

- May 30, 2014: Paper accepted.

5.2 Paper submission

The discovered vulnerability was submitted and accepted as short paper to WiSec 2014. The original submission explains all details of the vulnerability, and can be found attached below.

Exploiting WPA2-Enterprise Vendor Implementation Weaknesses through Challenge Response Oracles

ABSTRACT

Many of today's enterprise-scale wireless networks are protected by the WPA2-Enterprise Protected Extensible Authentication Protocol (PEAP). In this paper it is demonstrated how an attacker can steal a user's credentials and gain unauthorized access to such networks, by utilizing a class of vulnerable devices as MSCHAPv2 challenge response oracles. More specifically this paper explains how on these devices, Lightweight EAP (LEAP) MSCHAPv1 credentials can be captured and converted to PEAP MSCHAPv2 credentials by using a rogue Access Point. This man-in-the-middle vulnerability was found to be present in all current versions of Apple's iOS and OS X operating systems, and may impact other devices as well. A proof-of-concept implementation is available that shows how Authentication Server certificate validation and certificate pinning mechanisms may be bypassed. Mitigation strategies for the attack and protective actions which can be undertaken by end-users are also described in this paper.

Keywords

Network security, WPA2-Enterprise, PEAP, LEAP

Categories and Subject Descriptors

C.2.0 [**Computer-communication networks**]: Security and protection.

General Terms

Experimentation, Security

1. INTRODUCTION

Since its inception, wireless networking has become increasingly popular. More and more users desire access to network resources or the internet without having to struggle with network cables. As anyone with a wireless network card can eavesdrop on data sent wirelessly, it is self-evident that data security and user privacy are crucial aspects. This is especially true for enterprises, where confidential company data may be transmitted over the air. Fortunately, this data can be encrypted using a secure communication protocol.

For the average home user, the procotol that is considered most secure for wireless communication is WPA2-PSK. Here, the user configures a single password that is used for authentication. This password is shared with all users that require access to the network. For enterprises, this approach is infeasible: different users may require different access rights on the network, access may need to be revoked to former employees or the password may unintentionally leak to unauthorized parties. Therefore, the most popular choice for enterprises is WPA2-Enterprise. When this protocol is used, each user has their own login and password.

Though WPA2-Enterprise is considered secure in general, many attacks exist that are based on the man-in-the-middle principle. Here, a victim user is tricked into connecting to a malicious Access Point (AP) that has the same SSID as the enterprise network. To add to the problem, many devices on the market automatically join a wireless network in their Preferred Network List (PNL) by default. This is convenient for the user, as it reduces the amount of actions that have to be taken in order to access internet or network resources. Conversely, the user has no control over which network from the list is joined, meaning they could inadvertedly join a network under control of an attacker [10].

To solve these man-in-the-middle issues, the authenticity of the APs themselves can be verified by the device. This verification happens in the background, so the user fully relies on the used network protocol for its security. In context of WPA2-Enterprise networks, the IEEE 802.1X Standard specifies that the EAP protocol should be used for this purpose. EAP is an extensible authentication protocol that implements a wide variety of authentication procedures, called EAP methods.

Though EAP methods are well-defined and thoroughly examined for flaws by security experts, a correct protocol implementation is the responsibility of the device vendor. Unsurprisingly, there are subtle differences between various vendor implementations. Some of these may contribute to significant security vulnerabilities, such as those described in this paper.

For our research we focused mainly on the PEAP method, because it is popular, widely supported and considered

Submitted for review to ACM WiSec 2014.

secure. We tested the PEAP implementation of some of the most popular operating systems used today: Windows, MAC OS X, Android and iOS [6]. A practical attack for which Apple devices are particularly vulnerable resulted from our findings. The vulnerability has been reported to Apple prior to the release of this paper, on February 5, 2014.

2. ATTACK DESCRIPTION

Our attack exploits a combination of two vulnerabilities. The first vulnerability is the fact that some devices accept the older LEAP method for authentication. This EAP method is Cisco proprietary and uses the MSCHAPv1 algorithm to authenticate users. Past research has already proven that both MSCHAPv1 and MSCHAPv2 are insecure for various reasons when used without the protection of a TLS tunnel [15]. Since the LEAP method does not establish a TLS tunnel from client (or "Supplicant") to Authentication Server (AS) prior to exchanging credentials, it is vulnerable to a rogue AS man-in-the-middle attack [3].

The second vulnerability is that when the user configures or joins a PEAP network, some devices reuse the supplied credentials for all supported EAP methods. Hence, LEAP credentials do not have to be entered explicitly by the user. Existing man-in-the-middle attacks try to capture these LEAP credentials using a rogue AS, and then crack them with dictionary attack tools like `asleap`[1]. In our attack, we will use the credentials for a different purpose.

Before we discuss a practical implementation of our attack, let us first examine how credentials are exchanged in LEAP. The three entities participating in the authentication are the Supplicant, the Authenticator, and the AS. For simplicity, assume that Authenticator and AS reside on the same machine. The LEAP authentication procedure is performed as follows [3]:

1. The Supplicant associates with the AP and exchanges its identity with the AS. This step is identical for all EAP methods.
2. The AS sends an 8-byte challenge C_s, where $C_s = Random8(seed)$, to the Supplicant.
3. The Supplicant generates a 24-byte challenge response R_p, where $R_p = ChallengeResponse(C_s, H)$, $H = MD4(Unicode(PW))$ and PW is the password of the user. R_p is then sent to the AS.
4. The AS calculates $R_{check} = ChallengeResponse(C_s, H)$. The exchange is successful if R_p and R_{check} match.
5. In case of success, an EAP-Success message is sent from Authenticator to the Supplicant. Then, AS and Supplicant switch roles and repeat steps 2 to 4. This time we denote the challenge sent by the Supplicant as C_p, and the response by the AS as R_s.
6. The AS derives the Session Key as

$$SK = MD5(MD4(Unicode(H)) \quad ||C_s||R_p||C_p||R_s) \quad (1)$$

[1]This tool can be downloaded from the following URL: `http://www.willhackforsushi.com/?page_id=41`

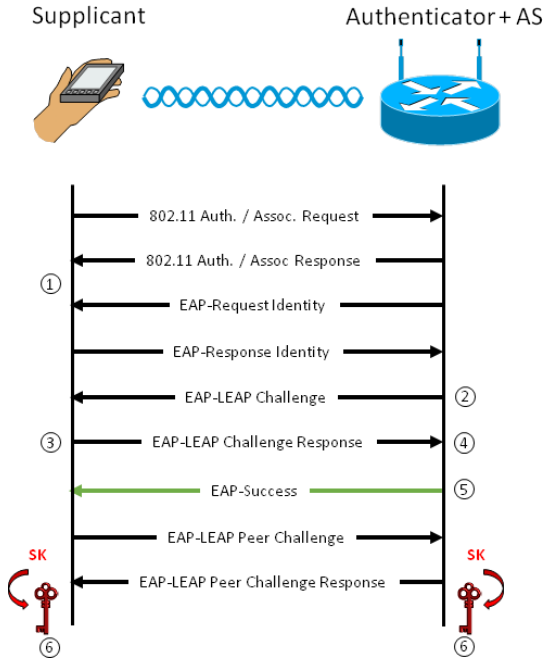

Figure 1: General LEAP authentication sequence

where "||" is the concatenation operator. The AS encrypts this value with the RADIUS secret and sends it to the Authenticator. The Supplicant also derives the SK, so this key can be used for WEP encrypted unicast communication. Finally, a random broadcast key is generated by the Authenticator and sent encrypted with the unicast key to the Supplicant.

Figure 1 shows a diagram of the previous algorithm. Note that a LEAP exchange is practically identical to performing two MSCHAPv1 authentications (steps 2 to 4): one from AS to Supplicant ($C_s \rightarrow R_p$) and one from Supplicant to AS ($C_p \rightarrow R_s$). [20].

Next, let us examine PEAP in Figure 2. This authentication method is significantly more complex, and among other features supports MSCHAPv2 mutual authentication to protect against man-in-the-middle attacks [13, 14]. Assuming cryptographic binding is not used (see Section 6.3), PEAP authentication is performed as follows:

1. The Supplicant associates with the AP and exchanges its identity with the AS.
2. In Phase 1, the Supplicant and AS set up a TLS tunnel similar to the procedure described in RFC 5246 [4]. From the TLS master secret, a Master Session Key (MSK) is derived via a one-way function. This key serves a comparable purpose to the Session Key from LEAP.
3. Phase 2 is performed inside the TLS tunnel and implies usage of an EAP inner authentication method encapsulated in EAP-Payload Type-Length-Value (TLV) messages. MSCHAPv2 is often used and relatively

secure for this purpose. Assuming MSCHAPv2 is used, the AS starts by generating a 16-byte random server challenge $C_s = Random16(seed)$ and sends it to the Supplicant.

4. The Supplicant also generates a random 16-byte peer challenge C_p. Then the challenge response is calculated as $R_p = ChallengeResponse(Challenge(C_s), H)$, where $Challenge(C_s) = SHA1(C_p||C_s||U)[0:8]$, U is the username of the user, $H = MD4(Unicode(PW))$, PW is the password of the user and $[0:8]$ means the first eight bytes of the data. This challenge response is transmitted back to the AS, along with C_p and U.

5. The AS calculates R_{check} analogous to R_p in step 4. R_{check} and R_p must match, or the authentication will fail.

6. The AS calculates a peer challenge response

$$R_s = PeerResponse(MD4(Unicode(H)), M_1, R_p, Challenge(C_p), M_2) \quad (2)$$

where M_1 is the "Magic server to client signing constant" and M_2 is the "Pad to make it do more than one iteration" constant. This result is SHA1-hashed and sent to the Supplicant.

7. The Supplicant authenticates the server, completing the MSCHAPv2 inner authentication.

8. An EAP-Result-TLV exchange is performed between AS and Supplicant to indicate the result of the PEAP authentication. Then an EAP-Success message is sent.

9. The MSK is used to derive the WPA2 Pairwise Master Key (PMK) and subsequent keys. Secure transmission of data can begin when the 802.11i four-way handshake [8] is completed.

When comparing the core differences between MSCHAPv1 and MSCHAPv2 credentials from RFCs 2433 and 2759, we can see that they are in fact very minor. Table 1 shows a comparison between the two methods [19, 20].

Though RFC 2759 states that MSCHAPv2 is incompatible with MSCHAPv1 [19], the insignificance of the aforementioned differences led us to the conclusion that some MSCHAPv1 messages can be converted to MSCHAPv2 messages and vice versa.

We will now show that C_s from MSCHAPv1 is identical to $Challenge(C_s)$ from the MSCHAPv2 AS and that R_p from the MSCHAPv1 peer is identical to R_{check} at the MSCHAPv2 server. This way we can be sure that all messages converted from MSCHAPv1 to MSCHAPv2 or vice versa will be accepted by the destination host. For the challenges we derive:

$$\begin{aligned} Challenge(C_s) &= SHA1(C_p||C_s||U)[0:8] && (3)\\ &= SHA1(x)[0:8] \ (C_p \text{ and } C_s \text{ are random}) && (4)\\ &= Random8(seed), \quad \text{if x = random} && (5)\\ &= C_s && (6) \end{aligned}$$

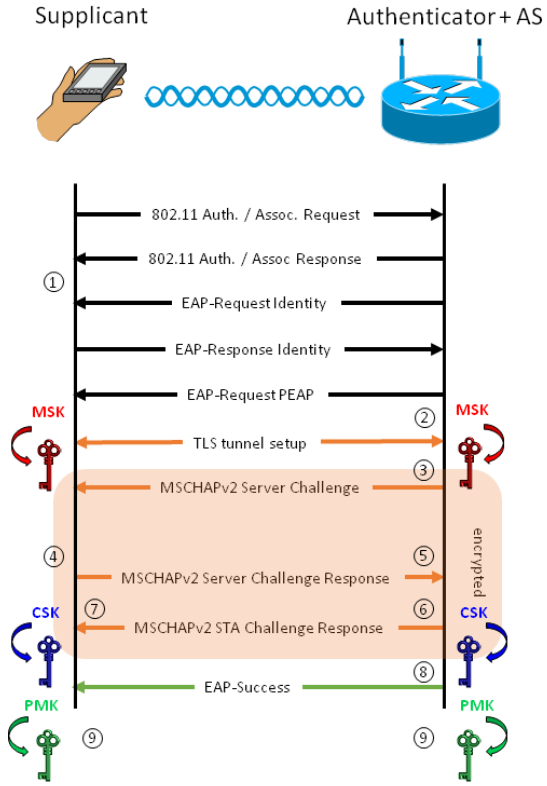

Figure 2: General PEAP authentication sequence

Given that the $ChallengeResponse$ function is the same in MSCHAPv1 and MSCHAPv2, we derive for the challenge responses:

$$\begin{aligned} R_s &= ChallengeResponse(C_s, H) && (7)\\ &= ChallengeResponse(Challenge(C_s), H) \quad \text{(Eq. 6)} && (8)\\ &= R_{check} && (9) \end{aligned}$$

With the knowledge that the challenge we get from the PEAP MSCHAPv2 AS can be converted to an MSCHAPv1 challenge (Equation 6), and that the challenge response we get from our LEAP MSCHAPv1 victim can be converted to an MSCHAPv2 challenge response that matches R_{check} on the AS (Equation 9), we devised a relay attack that uses a vulnerable device as an MSCHAPv2 challenge response oracle in order to gain unauthorized access to PEAP networks. Figure 3 shows a schematic representation of our attack.

3. PRACTICAL LEAP RELAY ATTACK

In this section we will show how the MSCHAPv1 to MSCHAPv2 conversion can be exploited in practice. First we will discuss the preconditions for the attack. Then, a practical implementation for attacking Apple devices will be demonstrated.

3.1 Preconditions

A device connecting to a PEAP network is considered vulnerable to our attack when all of the following preconditions are met:

	MSCHAPv1	**MSCHAPv2**
C_s	$C_s = Random8(seed)$	$C_s = Random16(seed)$
R_s	N/A	$R_s = PeerResponse(MD4(Unicode(H)), M_1, R_p, Challenge(C_p), M_2)$
C_p	N/A	$C_p = Random16(seed)$
R_p	$R_p = ChallengeResponse(C_s, H)$	$R_p = ChallengeResponse(Challenge(C_s), H)$

Table 1: Differences between MSCHAPv1 and MSCHAPv2 exchanges

- The device supports the LEAP method.
- The device connects automatically to the PEAP network. This is the default behavior.
- The Authenticator does not require and validate client certificates. Server certificate validation and certificate pinning may be enabled on the client.
- The MSCHAPv2 or MSCHAPv1 inner authentication EAP method is supported and allowed on the AS.

Note that most of the preconditions listed here are commonly fulfilled by default in enterprise network setups.

3.2 Case study: Apple devices

We will now demonstrate how the exploit can be practically applied to Apple devices (see Figure 3). Our proof-of-concept implementation uses a simple state machine to perform the attack (Figure 4). After successful execution, an attacker gains unauthorized access to the target network by impersonating a legitimate user.

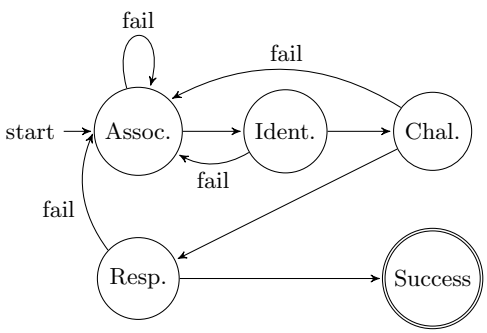

Figure 4: State machine of our attack

3.2.1 State 1: Association

Before wireless clients can begin the exchange of EAP packets to secured networks, they require association with a wireless AP. Clients will transmit Probe Requests every few seconds to search for in-range APs. Then, when the AP responds with a Probe Response, the client will automatically associate to it. Note that this only happens automatically if the client already successfully connected to this network at some point in time. We exploit this default auto-join behavior to have clients associate to an AP under our control. In order to accomplish this, we set up a fake wireless AP with the same SSID as the target network. This fake AP broadcasts beacon packets and replies to Probe Requests from clients.

The client will associate or reassociate to our fake network AP when it is closer to the target network AP, because better signal strength is preferred [5]. Since we do not want to receive requests from devices which are not vulnerable, our implementation uses the MAC Organizationally Unique Identifier (OUI) to identify the device vendor. We can filter out all non-Apple devices this way.

3.2.2 State 2: Identification

The first step after association in WPA2-Enterprise networks is identification. The AS has to know which user wants to authenticate in order to match corresponding credentials. We can learn the identity of the vulnerable device by sending an EAP Identity Request. The device will then reply with an EAP Identity Response which contains the username of our victim.

At this point, data sent over the air is still not encrypted. Hence, some PEAP implementations use anonymous identities. In this case the real username is only disclosed when a TLS tunnel has been established between the Supplicant and the AS. Nonetheless, we can still get the real username in a later phase of our attack.

Our next goal is to get the challenge value from the target AP. We created a modified version of the `wpa_supplicant`[2] tool for this purpose. At the end of this state, the binary executable of this modified version is called from our implementation.

[2]The `wpa_supplicant` tool is the de facto standard 802.1X Supplicant implementation in Linux-based operating systems. It was created by Jouni Malinen and is open source. We modified the source code to fit our needs instead of starting from scratch.

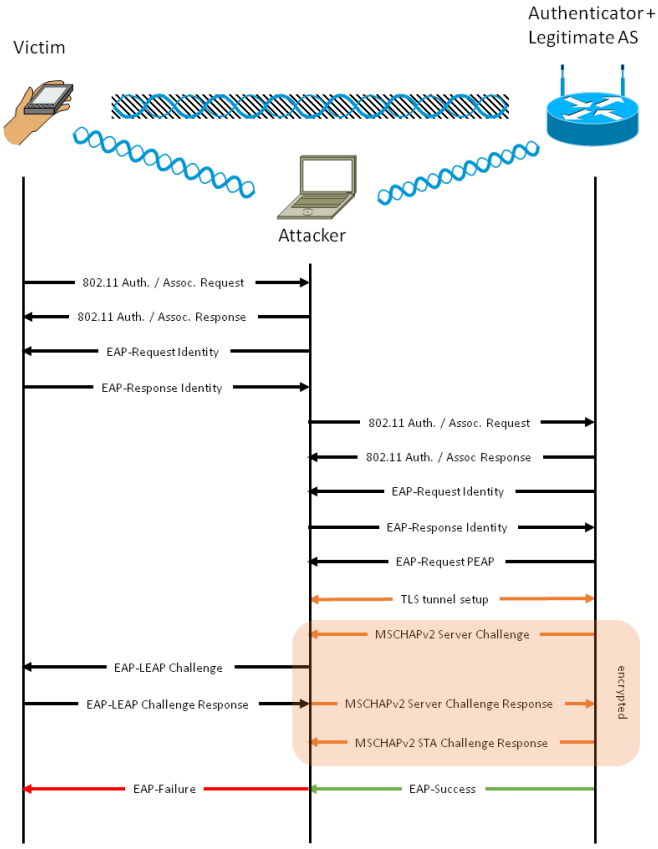

Figure 3: Schematic representation of the Apple LEAP attack

3.2.3 State 3: Challenge

In State 3, we wait for the `wpa_supplicant` tool to establish a TLS tunnel with the target AS and extract an MSCHAPv2 challenge from the inner authentication. We can now see why usage of client certificates would mitigate the attack, as the client certificate validation would not be successful in this case.

When the MSCHAPv2 challenge is retrieved, we pass it on to our tool. Upon receipt, the tool will periodically send LEAP Request messages (containing the extracted challenge) to the Apple device in order to keep the session alive.

3.2.4 State 4: Response

After receiving the LEAP Request, our victim will reply with a LEAP Response which contains an MSCHAPv1 challenge response to our MSCHAPv2 challenge. Should the target PEAP network enforce anonymous identities, the real or inner identity of the victim will also be revealed to the attacker through this LEAP Response. Next, our implementation will forward the received MSCHAPv1 challenge response as an MSCHAPv2 challenge response to the modified `wpa_supplicant` tool, which will in turn forward the challenge response to the legitimate PEAP network AS.

3.2.5 State 5: Success

When the AS receives our modified challenge response, authentication proceeds as usual, which means the AS has to authenticate to our Supplicant. However, since we are not in posession of the NT password secret, we cannot derive H. Hence, when receiving the peer challenge response from the AS, we are forced to accept any sent value.

After this, the MSCHAPv2 inner authentication will complete successfully and the port will be authenticated. The AS and our Supplicant will derive the MSK, and from this we can derive the PMK. We now have all components required to access resources on the internal network.

4. VARIANTS

In this section we will discuss some of the possible variations on the attack described above.

4.1 Remote challenge response collecting

Since devices periodically send Probe Requests for networks in their PNL, an attacker may sniff these Probe Requests and respond to each of them by posing as a WPA2-Enterprise network. When the attacker finds an SSID that truly is a WPA2-Enterprise network, the device will attempt to automatically connect. Next, a LEAP challenge can be sent in order to retrieve the LEAP challenge response. We can then perform a dictionary attack on these credentials by using existing tools, such as `asleap`.

4.2 Elevation of privileges

A side effect of our attack is that the user of the targeted vulnerable device is impersonated. Therefore, it is possible for the attacker to gain access to a user's VLAN when the attack is executed. For example, suppose there are two users currently connected to the network with a vulnerable device: Jane and Bob. Bob is an accountant and only has access to VLAN 0. Jane is a sales manager and only has access to VLAN 1. Because the VLAN is assigned by the AS based on username, an attacker can perform our attack on either Jane's or Bob's device in order to get access to their corresponding VLANs.

5. TEST RESULTS

We tested our attack on devices from multiple vendors. Table 2 shows the devices we tested and whether they were vulnerable to the LEAP relay attack.

Assuming that the same network protocol stack is used on all Apple operating systems, we concluded from these results that all Apple devices are vulnerable. The vulnerability was executed on multiple different APs using different AS implementations. These included:

- A TP-Link WN422G using `hostapd` and the latest `freeradius` implementation on the same machine.
- A Linksys WRT54G AP using the latest `freeradius` implementation on a dedicated machine.
- A Ubiquity UniFi AC 3.x AP using Windows RADIUS server on a dedicated machine.

Device	Vulnerable
iPod Touch (iOS 6.1.6)	Yes
iPhone 4 (iOS 7.1)	Yes
iPhone 4S (iOS 7.1)	Yes
Mac OS X 10.8.2 (Mountain Lion)	Yes
Samsung GT-S5570 (Android 2.3.4)	No
Google Nexus 7 (Android 4.4.2)	No
Samsung GT-I8750 (Windows Phone 8.0)	No
Windows 7 Desktop	No

Table 2: Devices vulnerable to the LEAP relay attack

6. MITIGATION

The attack we described in this paper can be mitigated in various ways. We will discuss four methods in this section.

6.1 Client certificates

In State 3 of our attack, a TLS tunnel has to be established between the attacker and the target network AS. When using client certificates, each client's certificate must be provided in the "Client Hello" phase of the TLS tunnel setup. When this verification fails, the TLS setup will be aborted and hence, our attack will fail because the MSK cannot be derived from the TLS master secret.

This countermeasure is very effective and by far the most secure. However, it would require a lot of administration effort for enterprises. Especially in enterprises with a Bring Your Own Device (BYOD) policy, because a signed certificate for every device allowed on the network must be installed on the AS.

6.2 iPhone Configuration Utility

A network administrator can distribute and install network configuration profiles on Apple devices. Alternatively, a user can create their own configuration profile using tools like the iPhone Configuration Utility[3].

These configuration profiles allow the network administrator to choose which EAP methods clients should use (Figure 5). They are the only way in which LEAP can be disabled on Apple devices. If this method is chosen to mitigate the attack, care must be taken in BYOD environments: if one user does not install the network profile, the attack can nevertheless be executed. Furthermore, network profiles can be accidentally removed by the user. For these reasons, security is put in the hands of the end user and therefore this method is not as secure as using client certificates.

[3]The iPhone Configuration Utility can be downloaded from the official Apple website at: http://support.apple.com/kb/dl1466

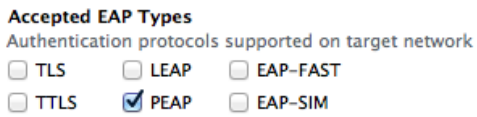

Figure 5: Disabling LEAP on Apple devices through profiles

6.3 Cryptobinding

An optional feature described in the PEAP version 2 internet draft is cryptographic binding [14]. This feature introduces the use of a new Type-Length-Value (TLV), the CryptoBinding TLV, to address man-in-the-middle attacks. A two-way handshake containing a Compound MAC Key (CMK) proves that the two authentications terminate at the same PEAP peer and PEAP server [12].

To calculate the CMK, the Supplicant is required to use keying material from both Phase 1 and Phase 2 of the PEAP exchange. In pratical terms this involves the calculation of the Tunnel Key (TK) and the Inner Session Key (ISK). These keys are combined in the cryptobinding algorithm to form the CMK (see Figure 2).

The TK is calculated similarly to the MSK from the TLS master secret, and would be available to an attacker. The ISK however, is calculated at the Supplicant as $ISK = InnerMPPESendKey||InnerMPPERecvKey$. The $InnerMPPESendKey$ and $InnerMPPERecvKey$ are both derived from the inner MSCHAPv2 Master Key (MK), which is derived as

$$MK = GetMasterKey(MD4(Unicode(H))[0:16], R_s) \quad (10)$$

Since H is unknown to the attacker, the ISK cannot be derived and authentication will fail.

If all consumer devices would support cryptobinding, this method would probably be the best way to mitigate our attack. However, from our experiments we concluded that Apple devices do not support cryptographic binding at this time. When support for cryptographic binding is implemented, we recommend to enable this feature on the authentication server.

6.4 Intrusion detection

Though it does not prevent an attack, real time intrusion detection may be helpful for enterprise system administrators to detect an attack. This way, appropriate measures may be taken by the administrator. For wireless networks, several Wireless Intrusion Detection Systems (WIDS) exist; both commercial (AirDefense, AirMagnet Distributed) and open source (Kismet, NetStumbler) [9].

A signature based WIDS might be able to detect our attack by passively scanning for LEAP requests. Since these packets will never be sent by a legitimate AP, IDS

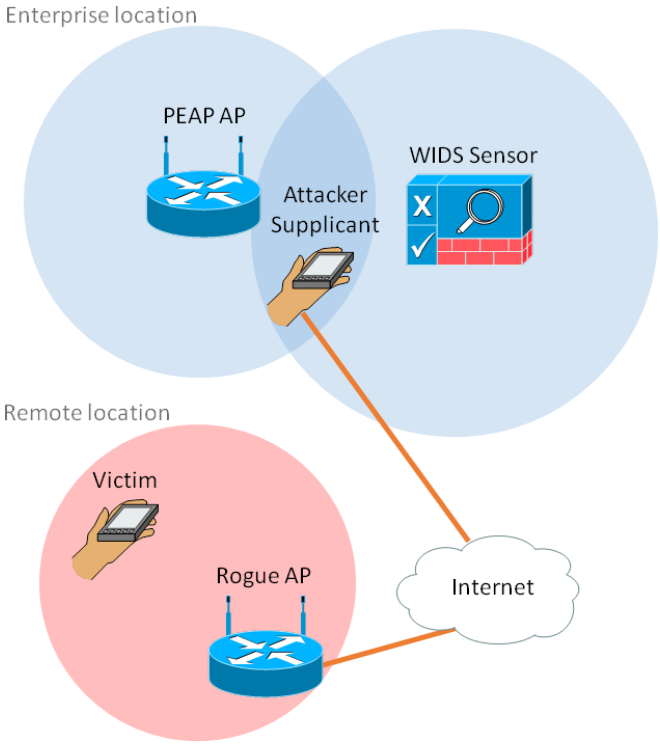

Figure 6: Remote LEAP relay attack

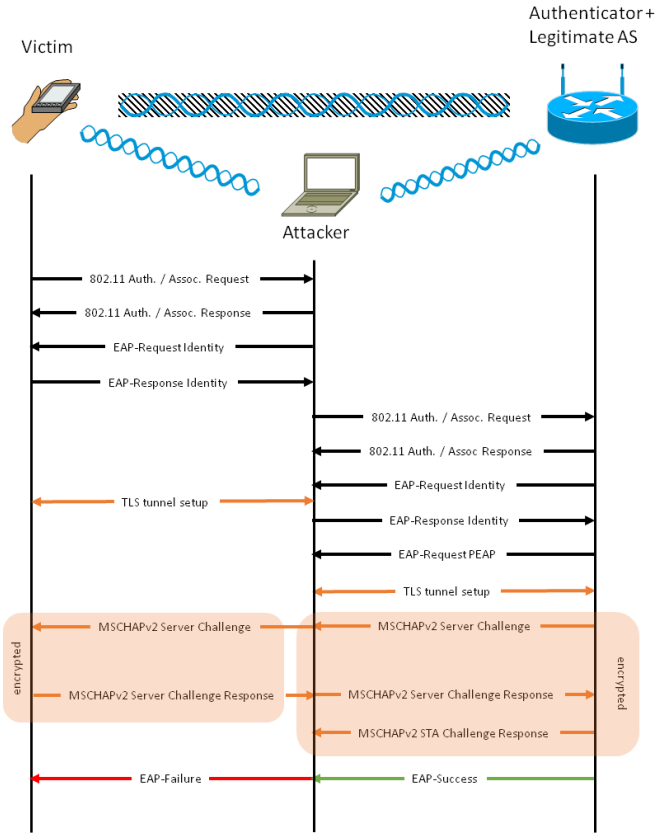

Figure 7: Generic PEAP relay attack

sensor nodes have a clear indication that the network is under attack. Analytic approaches to detect our attack may include station counts, association counts, OS fingerprinting and RSSI value analysis, though these methods often lead to false positives [1].

As a final note, we would like to indicate that care must be taken when relying on a WIDS for detection of an attack, as we believe that in many cases the IDS may be bypassed. For example, a victim may be in range of the rogue AP, while the latter is out of range from a WIDS sensor. The only entity that is required to be in range of the enterprise network is the attacker's client device. In Figure 6, an example scenario is shown where the relay attack is executed over the internet.

7. FUTURE WORK

Future work could be done by using the same attack principles described in this paper. From our experiments we determined that other devices, for example Android devices, do not employ certificate pinning by default. If the victim user did not configure a server certificate, we believe a more generic man-in-the-middle attack may be executed as shown in Figure 7. This attack is similar to the attack described in RFC 7029 [7]. Note that in this case, the preconditions are stricter: it is required that server certificates are not used by the Android device, which was not the case for Apple devices.

Another option for future work would be to complete the LEAP exchange from Figure 3, instead of the MSCHAPv2 exchange with the PEAP AS. This way, an attacker could become an active man-in-the-middle, so data traffic sent by the victim can be captured. However, the attacker would need to derive SK, which requires knowledge of the password hash H (see Equation 1).

8. RELATED WORK

A similar attack on PEAP vendor implementations is the EAP dumb-down attack introduced by Raul Siles in 2013. This attack exploits the default lack of certificate validation in mobile devices. When these devices trust the AS, any inner EAP method may be chosen, and credentials may be stolen. However, for Apple devices, the dumb-down attack requires user intervention whereas our attack is automatic [16]. Furthermore, a correct configuration of authentication server certificates does not mitigate our attack for Apple devices.

Other related attacks were proposed at numerous security conferences. In 2008, Joshua Wright and Brad Antoniewicz demonstrated how EAP credentials such as MSCHAPv2 exchanges can be collected using `freeradius-wpe`, a rogue AS implementation [17]. By using the `asleap` tool, these credentials can then be cracked with a dictionary attack [2]. More recently, in 2012, Moxie Marlinspike showed how MSCHAPv2 credentials can be cracked in less than 24 hours using cloud-based FPGA nodes [11]. Finally, Josh Yavor indicated the dangers of BYOD and default certificate validation behavior of mobile devices in 2013 [18].

9. CONCLUSIONS

We demonstrated how MSCHAPv1 challenges and challenge responses can be converted to MSCHAPv2 challenges and challenge responses. Then, we indicated how this can be exploited in practice when a Supplicant supports the insecure LEAP method and when credentials are reused between EAP methods.

Many devices, in particular all Apple devices, are currently vulnerable to our attack. Mitigation is possible in various ways. However, we noted why some mitigation strategies might not be feasible for enterprises. Therefore, users and network managers should take care when their devices satisfy all mentioned vulnerability preconditions.

10. REFERENCES

[1] M. Ciampa. *CWNA Guide to Wireless LANs*. Cengage Learning, 2012.

[2] Cisco. Dictionary Attack on Cisco LEAP Vulnerability, 2003. `http://www.cisco.com/en/US/tech/tk722/tk809/technologies_security_notice09186a00801aa80f.html`.

[3] A. DeKok and A. Sulmicki. Cisco LEAP protocol description, 2001. `http://freeradius.org/rfc/leap.txt`.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol. RFC 5246, IETF, August 2008.

[5] M. S. Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition*. O'Reilly, 2005.

[6] A. Gupta, R. Cozza, and C. Lu. Market Share analysis: Mobile phones, worldwide, 4Q13 and 2013. *Gartner*, 2014.

[7] S. Hartman and M. Wasserman. Extensible Authentication Protocol (EAP) Mutual Cryptographic Binding. RFC 7029, IETF, October 2013.

[8] C. He and J. C. Mitchell. Analysis of the 802.11i 4-Way Handshake. October 2004.

[9] K. Hutchison. Wireless Intrusion Detection Systems. *SANS Institute InfoSec Reading Room*, October 2004.

[10] Ludwig Nussel. The Evil Twin problem with WPA2-Enterprise. *SUSE Linux Products GmbH*, 2010.

[11] M. Marlinspike. Divide and Conquer: Cracking MS-CHAPv2, 2012. `https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/`.

[12] Microsoft. Cryptobinding, 2014 (accessed). `http://msdn.microsoft.com/en-us/library/cc238384.aspx`.

[13] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson. Protected EAP Protocol (PEAP). Work in Progress 6, IETF, March 2003.

[14] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson. Protected EAP Protocol (PEAP) Version 2. Work in Progress 10, IETF, October 2004.

[15] B. Schneier, Mudge, and D. Wagner. Cryptanalysis of Microsoft's PPTP Authentication Extensions. *CQRE '99*, October 1999.

[16] R. Siles. EAP dumb-down attack. In *RootedCON 2013*, pages 27–28. DinoSec, 2013.

[17] J. Wright. FreeRADIUS-WPE, 2008. `http://www.willhackforsushi.com/?page_id=37`.

[18] J. Yavor. The BYOD PEAP Show. In *DefCon 21*. iSEC Partners, 2013.

[19] G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759, IETF, January 2000.

[20] G. Zorn and S. Cobb. Microsoft PPP CHAP Extensions. RFC 2443, IETF, October 1998.

5.3 PEAPwn tool

PEAPwn is a proof-of-concept implementation of the Apple LEAP relay attack described in this chapter. The proof-of-concept consists of two components:

- The PEAPwn Python script, which performs the actual exploit.
- A modified version of the `wpa_supplicant` tool, which performs the TLS tunnel setup, the credential relay to PEAPwn via Unix domain sockets, and the 802.11i 4-Way Handshake with the target AP (see Section 3.4).

The paper from Section 5.2 explained how the tool works. In this section we will discuss which options are available, and how the user can use the tool to gain unauthorized access to a PEAP WPA2-Enterprise network. This proof-of-concept implementation is currently only supported on Linux, and can be invoked on the command line as shown in Listing 5.1.

```
peapwn.py [-h] [--debug] [--nooui] infra_if mon_if essid bssid
```

Listing 5.1: PEAPwn options

Arguments that may be provided are:

- `infra_if`: The interface that should be used as the "client" interface to connect with the target SSID.
- `mon_if`: The interface that should be used to spoof the target SSID and attack the victim's device.
- `essid`: The SSID of the network to attack.
- `bssid`: The MAC address of the AP to spoof. Make sure that you are closer to the victim's device than the real AP with this MAC address.
- `-h`: Display help.
- `-d`: Print verbose wpa_supplicant output.
- `-n`: Disables usage of an Organizationally Unique Identifier (OUI) list to identify vulnerable devices.

Chapter 6

Conclusion and future work

Despite the fact that many protocols are available to secure wireless networks, a lot of networks still use open authentication to provide access to the network resources. Notable examples of such networks are public hotspots, over which sensitive user data is transmitted every day. Our experiments have shown that the presence of only one open network in the device's PNL is enough to compromise the privacy of a user. We concluded that on average, 16% of the mobile device users are directly vulnerable to an active MITM attack.

Fortunately, many protocols exist to secure wireless networks. We explained the mechanisms behind the most popular wireless security protocols in order to gain an insight in their known vulnerabilities. Some of the vulnerabilities were tested to verify their applicability today. Notable results from these tests come from the dumb-down attack, which despite being reported in 2013 is still a risk to roughly 28% of mobile device users.

Concluding this thesis, we have demonstrated how a vendor implementation vulnerability in Apple devices allows an attacker to gain unauthorized access to any PEAP network. This discovery resulted in a paper that was submitted and accepted to WiSec 2014. At the time of writing, the vulnerability has not yet been fixed. This proves again that vulnerabilities may remain feasible for extended periods of time, and that users and administrators should always be on their guard for both existing and novel attacks on wireless networks in order to protect their privacy.

Future work

A first possibility for future work was discovered in experiments with the Google Nexus 7 (2013). When connecting to a new 802.1X secured wireless network on this device, the user is required to specify the configuration. Among the available options are mandatory fields such as the username and password field, and optional fields such as the TLS certificate and the PEAP inner authentication method.

While experimenting, we noticed that the PEAP inner authentication setting can be overridden by an attacker if the victim is quickly disconnected from one network, and then quickly rejoins another that shares the same BSS MAC address. For example, suppose a user is connected to a network "gtcnet", and that this user has configured their Android device to use PEAP with GTC for this network. If an attacker quickly deauthenticates the user and spoofs a different SSID "testnet" (also using the same MAC address), the attacker may force the client to use PEAP with EAP-GTC, even if the client configured "testnet" to use PEAP with MSCHAPv2.

The exact conditions under which this event occurs are not clear. It is also unknown why not all Android devices behave this way. Because of time contraints, we could not investigate this further, and hence work regarding this issue could be done in the future.

A second possibility for future work is to implement the LEAP relay attack from Chapter 5 for EAP-TTLS. The LEAP relay attack was only tested on PEAP networks. However, since PEAP and EAP-TTLS are very similar protocols, we believe the attack may also work on EAP-TTLS secured enterprise networks.

As a third option, future work could be done by investigating new attacks on WPA-Personal. In this thesis, we focused on WPA-Enterprise 802.1X for our novel attacks feasibility study, but new vulnerabilities may be found in WPA-Personal as well.

Lastly, recall that we extended the `asleap` tool with a threaded brute-force functionality for MSCHAPv2 credentials. This cracking process could be improved in the future by making use of the GPU for calculating MD4 hashes.

Acronyms

AAA Authentication, Authorization and Accounting. 30, 31, 63

ACK Acknowledgement. 7, 10, 63

AES Advanced Encryption Standard. 22–24, 63

AID Association ID. 8–10, 63

AP Access Point. 1, 4–15, 21–23, 25, 32, 35, 37, 40–43, 45, 60, 63

ARP Address Resolution Protocol. 21, 24, 63

AS Authentication Server. 26, 28–33, 35–38, 40–42, 45, 49, 63

AVP Attribute Value Pair. 31, 32, 37, 63

BSA Basic Service Area. 4, 6, 7, 63

BSS Basic Service Set. 4, 6, 61, 63

CA Certificate Authority. 16, 63

CBC-MAC Cipher Block Chaining Message Authentication Code. 24, 63

CCM Counter mode with CBC-MAC. 23, 24, 63

CCMP Counter mode Cipher block chaining Message authentication code Protocol. 22–24, 63

CERT Computer Emergency Response Team. 50, 63

CHAP Challenge Handshake Authentication Protocol. 32, 33, 63

CMK Compound MAC Key. 37, 63

CPU Central Processing Unit. 48, 63

CRC Cyclic Redundancy Check. 6, 20, 21, 63

CSK Compound Session Key. 35, 37, 63

CSRF Cross-Site Request Forgery. 19, 63

CTR Counter. 24, 63

CTS Clear to Send. 7, 9, 10, 63

DA Destination Address. 6, 23, 63

DES Data Encryption Standard. 33, 47, 63

DHCP Dynamic Host Control Protocol. 13, 63

DNS Domain Name Server. 12, 15, 63

DS Distributed System. 4, 5, 63

EAP Extensible Authentication Protocol. 26–29, 33, 36–39, 42, 43, 45, 46, 49, 50, 63

EAPOL EAP over LAN. 27–30, 63

EMSK Extended Master Session Key. 37–39, 48, 63

ESS Extended Service Set. 4, 6, 7, 63

FCS Frame Check Sequence. 6, 9, 10, 63

FPGA Field-Programmable Gate Array. 47, 63

GPU Graphics Processing Unit. 48, 62, 63

GTC Generic Token Card. 33, 43, 61, 63

GTK Group Transient Key. 22, 23, 63

GUI Graphical User Interface. 45, 63

HSTS HTTP Strict Transport Security. 18, 19, 63

HT High Throughput. 5, 6, 63

HTTP Hyper Text Transfer Protocol. 12, 16–19, 63

HTTPS HTTP Secure. 17, 18, 63

IBSS Independent Basic Service Set. 4, 63

ICV Integrity Check Value. 20–22, 24, 25, 63

IEEE Institute of Electrical and Electronics Engineers. 1, 3, 4, 20, 22, 26, 63

IP Internet Protocol. 24, 27, 31, 63

IPMK Intermediate PEAP MAC Key. 36, 37, 63

ISK Inner Session Key. 36, 63

ISM Industrial, Scientific and Medical. 1, 63

IV Initialization Vector. 20, 21, 23, 63

KCK Key Confirmation Key. 23, 63

KEK Key Encryption Key. 23, 63

LAN Local Area Network. 3, 4, 26, 28, 30, 63

LEAP Lightweight Extensible Authentication Protocol. 35, 39, 43, 60, 61, 63

LLC Logical Link Control. 3, 4, 22, 63

MAC Medium Access Control. 3, 4, 6, 9–12, 15, 21, 23, 40, 45, 60, 61, 63

MAN Metropolitan Area Network. 3, 26, 63

MD4 Message Digest 4. 33, 48, 62, 63

MD5 Message Digest 5. 32, 33, 49, 63

MIC Message Integrity Code. 23–25, 63

MITM Man-In-The-Middle. 11, 12, 16–20, 23, 34, 35, 41–43, 46–50, 61, 63

MK Master Key. 32, 34, 36, 63

MPPE Microsoft Point-to-Point Encryption. 33, 35–37, 63

MSK Master Session Key. 29, 35–39, 48, 63

MTU Maximum Transmission Unit. 39, 63

NAI Network Access Identifier. 29, 40, 41, 63

NAK Negative-Acknowledgment. 29, 32, 33, 42, 44, 63

NAS Network Access Server. 26, 31, 41, 63

NIC Network Interface Controller. 4, 11, 12, 40, 63

OS Operating System. 42–44, 63

OSI Open Systems Interconnection. 3, 15, 16, 63

OUI Organizationally Unique Identifier. 60, 63

PAE Port Access Entity. 26, 63

PEAP Protected EAP. 35, 37, 38, 42, 43, 45, 50, 60, 61, 63

PIN Personal Identification Number. 25, 63

PMK Pairwise Master Key. 22, 23, 25, 29, 32, 37, 63

PN Packet Number. 24, 63

PNL Preferred Network List. 7, 12, 41, 46, 61, 63

PPP Point-to-Point Protocol. 28, 31, 33, 63

PRF+ Pseudo-Random Function+. 36, 37, 63

PRGA Pseudo-Random Generation Algorithm. 20, 63

PSK Pre-Shared Key. 23, 25, 63

PTK Pairwise Transient Key. 22, 23, 25, 29, 63

QoS Quality of Service. 5, 10, 24, 25, 63

RA Request Authenticator. 31, 32, 63

RADIUS Remote Authentication Dial In User Service. 26, 28, 29, 31, 32, 36, 63

RC4 Rivest Cipher 4. 20–24, 33, 63

RFC Request For Comments. 23, 31–33, 35, 37, 39, 63

RSN Robust Security Network. 22, 63

RSNA Robust Security Network Association. 22, 63

RTS Request to Send. 7, 9, 63

SA Source Address. 6, 23, 24, 63

SD Secure Digital. 63

SK Session Key. 23, 35, 36, 63

SNAP Subnetwork Access Protocol. 22, 63

SQL Structured Query Language. 12, 14, 45, 46, 63

SSID Service Set Identifier. 4, 6, 7, 10, 12–15, 23, 41, 42, 60, 61, 63

SSL Secure Sockets Layer. 16, 19, 63

STA Station. 4–15, 21–23, 25, 63

TA Transmitter Address. 23, 63

TCP Transport Control Protocol. 31, 63

TK Tunnel Key. 36, 63

TK Temporal Key. 23, 24, 63

TKIP Temporal Key Integrity Protocol. 22–24, 63

TLS Transport Layer Security. 16, 17, 32, 35–39, 41, 47, 49, 60, 61, 63

TLV Type-Length-Value. 36, 63

TSC TKIP Sequence Counter. 23–25, 63

TTLS Tunneled TLS. 63

UA User Agent. 17–19, 63

UDP User Datagram Protocol. 31, 63

URL Uniform Resource Locator. 18, 63

WEP Wired Equivalent Privacy. 8, 10, 20–24, 26, 35, 63

WLAN Wireless Local Area Network. 4, 26, 27, 63

WPA Wi-Fi Protected Access. 14, 22, 23, 25, 26, 50, 63

WPA2 Wi-Fi Protected Access 2. 26, 35, 63

WPS Wi-Fi Protected Setup. 25, 63

Bibliography

[1] ASLEAP – Exploiting Cisco LEAP, 2008. http://www.willhackforsushi.com/Asleap.html.

[2] EAP-MD5-Challenge Authentication Protocol, 2013. http://www.juniper.net/techpubs/software/aaa_802/sbrc/sbrc70/sw-sbrc-admin/html/EAP-029.html.

[3] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz. Extensible Authentication Protocol. RFC 3748, IETF, June 2004.

[4] B. Aboba and P. Calhoun. RADIUS support for EAP. RFC 3579, IETF, September 2003.

[5] D. Akhawe and A. P. Felt. A Large-Scale Field Study of Browser Security Warning Effectiveness. 2013. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/41323.pdf Accessed: 2013-11-18.

[6] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *Security Protocols*, pages 28–41. Springer, 2005.

[7] Atom. oclhashcat advanced password recovery, January 2014. http://hashcat.net/oclhashcat/.

[8] Bauer. RADIUS Server Configuration for joining eduroam-US, 2009. https://www.eduroam.us/radius_configuration.

[9] J. Berg. hostapd Linux documentation page, 2013. http://wireless.kernel.org/en/users/Documentation/hostapd.

[10] A. Bittau. The fragmentation attack in practice. In *IEEE Symposium on Security and Privacy, IEEE Computer Society*, 2005.

[11] L. Blunk and J. Vollbrecht. PPP Extensible Authentication Protocol (EAP). RFC 2284, IETF, March 1998.

[12] Cisco. Dictionary Attack on Cisco LEAP Vulnerability, 2003. http://www.cisco.com/en/US/tech/tk722/tk809/technologies_security_notice09186a00801aa80f.html.

[13] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20132018, February 2014. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[14] P. Congdon, B. Aboba, A. Smith, G. Zorn, and J. Roese. IEEE 802.1X RADIUS Usage Guidelines. RFC 3580, IETF, September 2003.

[15] R. Dantu, G. Clothier, and A. Atri. EAP methods for wireless networks. *Computer Standards and Interfaces 29*, pages 289 – 301, 2006. http://nsl.cse.unt.edu/~dantu/cae/Dantu/EAP_Methods_for_Wireless_Networks.pdf Accessed: 2014-01-06.

[16] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, IETF, May 1996.

[17] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol. RFC 5246, IETF, August 2008.

[18] Erik Tews. Attacks on the WEP protocol. *Diploma thesis Fachgebiet Theoretische Informatik*, 2007.

[19] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn. Diameter Base Protocol. RFC 6733, IETF, October 2012.

[20] P. Funk and S. Blake-Wilson. EAP Tunneled TLS Authentication Protocol Version 0 (EAP-TTLSv0). RFC 5281, IETF, August 2008.

[21] M. S. Gast. *802.11 Wireless Networks: The Definitive Guide, Second Edition.* O'Reilly, 2005.

[22] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security. RFC 6797, IETF, November 2012.

[23] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, IETF, April 2002.

[24] http://www.fir3net.com. Image of the chain of trust, 2014 (accessed). `http://www.fir3net.com/images/PKI_ChainofTrust-Chain.png`.

[25] IEEE Computer Society. Amendment 6: Medium Access Control (MAC) Security Enhancements. IEEE Standard, IEEE, July 2004.

[26] IEEE Computer Society. IEEE Standard for local and metropolitan area networks. *IEEE Std 802.1X–2010*, 2010.

[27] IEEE Computer Society. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard, IEEE, March 2012.

[28] International Society of Automation. Image of the 802.11 shared key authentication, 2014 (accessed). `http://www.dks.co.id/knowledge/images/20070454-1.gif`.

[29] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898, IETF, September 2000.

[30] M. Kato. HTTP Strict Transport Security, 2013. `https://developer.mozilla.org/en-US/docs/Security/HTTP_Strict_Transport_Security`.

[31] V. Kumkar, A. Tiwari, P. Tiwari, A. Gupta, and S. Shrawne. Vulnerabilities of wireless security protocols (wep and wpa2). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1(2):34–38, 2012.

[32] C. Macnally. Cisco LEAP protocol description. Protocol description, IETF, September 2001.

[33] M. Marlinspike. New Tricks For Defeating SSL In Practice. In *Black Hat DC 2009*, pages 1–99. Black Hat, 2009.

[34] M. Marlinspike and M. Ray. Divide and Conquer: Cracking MS-CHAPv2 with a 100% success rate, 2012. `https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/`.

[35] N. Mavrogiannopoulos and S. Josefsson. GnuTLS 3.0.0 Release Notes, July 2011. `https://lists.gnu.org/archive/html/info-gnu/2011-07/msg00011.html`.

[36] Microsoft. How 802.11 Wireless Works, March 2003. `http://technet.microsoft.com/en-us/library/cc757419%28v=ws.10%29.aspx`.

[37] Microsoft. Supported EAP methods in Windows, 2014 (accessed). `http://technet.microsoft.com/en-us/library/hh945104.aspx`.

[38] Microsoft Corporation. Protected Extensible Authentication Protocol (PEAP). Microsoft Open Specifications Documentation, Microsoft, February 2014.

[39] Microsoft Security Advisory 2876146. Wireless PEAP-MS-CHAPv2 Authentication Could Allow Information Disclosure, August 2013. `https://technet.microsoft.com/library/security/2876146`.

[40] MITRE. Heartbleed bug: CVE-2014-0160, 2014 (accessed). `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`.

[41] Moxie Marlinspike. SSLstrip, 2009. `http://www.thoughtcrime.org/software/sslstrip/`.

[42] A. Prado, N. Harris, and Y. Gluck. BREACH attack, 2014 (accessed). `http://breachattack.com/`.

[43] C. Rigney, W. Willats, and P. Calhoun. RADIUS Extensions. RFC 2869, IETF, June 2000.

[44] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2058, IETF, January 1997.

[45] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS). RFC 2865, IETF, June 2000.

[46] V. Roth, W. Polak, E. Rieffel, and T. Turner. Simple and effective defense against evil twin access points. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 220–235, New York, NY, USA, 2008. ACM.

[47] Ryan Hurst and Ashwin Palekar. Microsoft EAP CHAP Extensions. Internet Draft, IETF, June 2007.

[48] T. Schmoyer, Y.-X. Lim, and H. L. Owen. Wireless intrusion detection and response: a classic study using main-in-the-middle attack. In *Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE*, volume 2, pages 883–888 Vol.2, March 2004.

[49] R. Siles. Wi-Fi: Why iOS (Android and others) Fail inexplicably? In *RootedCON 2013*, pages 27–28. DinoSec, 2013.

[50] D. Simmons. Free wi-fi hotspots pose data risk, Europol warns, March 2014. `http://www.bbc.com/news/technology-26469598`.

[51] D. Simon, B. Aboba, and R. Hurst. The EAP-TLS Authentication Protocol. RFC 5216, IETF, March 2008.

[52] W. Simpson. PPP Challenge Handshake Authentication Protocol (CHAP). RFC 1994, IETF, August 1996.

[53] K. Bauer, H. Gonzales, and D. McCoy. Mitigating evil twin attacks in 802.11. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 513–516, Dec 2008.

[54] I. C. Society. IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture. IEEE Standard, IEEE, February 2002.

[55] D. Stanley, J. Walker, and B. Aboba. Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs. RFC 4017, IETF, March 2005.

[56] L. Strand. Logical port entities diagram, 2004. `http://www.tldp.org/HOWTO/8021X-HOWTO/intro.html`.

[57] E. Tews and M. Beck. Practical attacks against wep and wpa. In *Proceedings of the Second ACM Conference on Wireless Network Security*, WiSec '09, pages 79–86, New York, NY, USA, 2009. ACM.

[58] M. Vanhoef and F. Piessens. Practical verification of wpa-tkip vulnerabilities. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (AsiaCCS 2013)*, pages 427–436, 2013.

[59] S. Viehböck. Brute forcing Wi-Fi Protected Setup, 2014 (accessed). `http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf`.

[60] D. Watkins. Embedded WLAN (Wi-Fi) CE Devices: Global Market Forecast. *Strategy Analytics*, February 2014.

[61] Wikimedia. Image of the 802.11i 4-Way Handshake, 2005. `http://upload.wikimedia.org/wikipedia/commons/6/60/4-way-handshake_WPA2.png`.

[62] Wikimedia. EAP message flow diagram, 2006. `http://upload.wikimedia.org/wikipedia/commons/8/8a/EAP_message_flow.png`.

[63] Wikimedia. Image of the 802.11 channels, 2009. `http://en.wikipedia.org/wiki/File:2.4_GHz_Wi-Fi_channels_%28802.11b,g_WLAN%29.svg`.

[64] Wikimedia. 802.1X involved protocols diagram, 2010. `http://upload.wikimedia.org/wikipedia/commons/1/1f/802.1X_wired_protocols.png`.

[65] Z. Yang, A. C. Champion, B. Gu, X. Bai, and D. Xuan. Link-layer protection in 802.11i wlans with dummy authentication. In *Proceedings of the Second ACM Conference on Wireless Network Security*, WiSec '09, pages 131–138, New York, NY, USA, 2009. ACM.

[66] F. Ye, S.-T. Sheu, T. Chen, and J. Chen. The Impact of RTS Threshold on IEEE 802.11 MAC Protocol . *Tamkang Journal of Science and Engineering*, 6:57–63, 2003.

[67] G. Zorn. Deriving MPPE Keys From MS-CHAP V1 Credentials. Internet Draft 0, IETF, September 1998.

[68] G. Zorn. Deriving MPPE Keys From MS-CHAP V2 Credentials. Internet Draft 2, IETF, November 1998.

[69] G. Zorn. Microsoft PPP CHAP Extensions, Version 2. RFC 2759, IETF, January 2000.

[70] G. Zorn and S. Cobb. Microsoft PPP CHAP Extensions. RFC 2433, IETF, October 1998.