TTM4137 Wireless Security Lab Assignment

NTNU, Department of Telematics, October 23, 2016

Group 8
Md Sakib Nizam Khan
Shiva Prasad Thagadur Prakash
Viswanathan Manihatty Bojan

1 Introduction

Security in wireless network has been evolving continuously. With the emergence of commercial 5G products in the near future, it indeed requires a lot of attention and a good understanding of their security principles. The wireless security lab provides a great platform where the fundamental wireless security concepts can be studied in a practical manner. And in this report, we have presented the knowledge we gathered during the course of the lab experiments by answering the questions and additional notable descriptions, if any. The structure of the report is as follows. Section 2 deals with WEP security, Section 3 deals with WPA security, Section 4 deals with RSN security and finally Section 5 deals with GSM security.

2 WEP Penetration and Cryptanalysis

The **Wired Equivalent Privacy(WEP)** is a security mechanism that is defined and used by 802.11 products. The motive of WEP is to offer security that is equivalent to that of the wired LANs. And it protects the MAC protocol Data Unit (MPDU). However, WEP was detected with multiple problems that compromised its integrity and authentication. This led to the development of WPA/ TKIP. Here, in our experiment, we will be making use of one such weaknesses of WEP to break its security.

WEP has a history of attacks such as Fluhrer, Mantin and Shamir (FMS) attack, Pyshkin, Tews, Weinmann (PTW) attack, etc. Here, we will be making use of the PTW attack in an active mode for our experiment. The active PTW attack involves the procedure of injecting traffic into the network externally in order to collect sufficient samples over which the PTW attack can be performed.

2.1 Result

The experiment was performed as per the lab description and the security key was retrieved. The screen shot of the same can be found in the Appendix 7.1.

2.2 Questions

Q1. Describe the parameters of the AP under analysis, such as the SSID, BSSID, channel number (and optionally the frequency), encryption mechanism, associated clients, the WEP key. How many packets did the PTW attack require? Which web page was the legitimate client browsing?

The target of analysis in our case is a Cisco AP. The various AP parameters are discussed in detail below:

- **SSID**: This corresponds to Service Set Identifier and is the name of the AP. It can be empty if SSID hiding is activated. As a part of this exercise, the SSID of the AP was also kept hidden.

- **BSSID**: This refers to the Basic Service Set Identifier and is the MAC address of the access point. The BSSID of the AP under analysis is: 98:FC:11:B2:D2:67.

- **Channel**: This corresponds to the channel number. All the access points, by default, operate at a specific channel and frequency range. Based on whether the device operates at 2.4GHz band or 5 GHz band, there are 3 and 19 non overlapping channels respectively. The admin takes care of setting the channel to the AP. When using airodump-ng, it needs to be configured to listen to a specific channel so that it can capture the packets of the desired access point. In our case, the AP was operating at channel 6.

- **ENCRYPTION MECHANISM**: This refers to the encryption algorithm that is used in the access point. In our experiment, the encryption algorithm used is WEP.
- **STATION**: This refers to the clients that are connected to the access point. At any time, there can be any number of associated clients from none to many. In the experiment, the desktop machine was the only station that was connected to the access point. It had a MAC address of 50:7A:55:75:4B:2B.
- **BEACONS**: This corresponds to the number of announcement packets sent by the AP. The AP keeps sending these packets at a constant rate so that they can be collected by a station.
- **WEP KEY**: This corresponds to the password value with which the AP can be accessed. After the successful PTW attack, the WEP key was found to be: 46:2E:62:33:21:2B:5F:24:73:72:67:26:52. The ASCII equivalent is: F.b3!+_$srg&R

In order to crack the key, the PTW attack consumed **104947** data packets. **60060** unique IVs were generated during this process.

From the captured Wireshark packets, we understood that the client was sending continuous ping requests to the Google server. Additionally, we were also able to observe that the client was performing an Ubuntu update action.

Q2. Is it possible to run a completely passive PTW attack on 104-bit WEP? Why and under which circumstances?

Yes it is possible to perform a completely passive PTW attack on 104-bit WEP key. Under the passive attack, the attacker needs to listen to the encrypted IP traffic for longer period until the necessary amount of IVs are collected. Active attacks involve using the APR injection where ARP requests are generated more randomly which also helps in finding out the keystream quickly. However, under passive attack, the attacker needs to depend on the surrounding traffic for collecting sufficient samples for the performing the attack. And hence, the probability of a successful passive PTW attack is dependent on the amount of generated traffic [7].

Q3. Why is it possible to send an arbitrary amount of ARP-requests to the AP without knowing the WEP key?

Any node that receives an ARP request will automatically respond with an ARP reply. Here, in the experiment, even though WEP is enabled and the station is unaware of the WEP key, it is still possible to send ARP requests to the AP. This is because, we set the interface of WLAN to monitor mode before sending the ARP requests. Under monitor mode, it is not necessary to know the WEP key to send ARP requests to the AP as it replays the previously captured ARP packets [8].

Q4. What can you do to strengthen your attack when there are some clients, but hardly any traffic at all?

As far as the PTW attack is concerned, the time it takes to break WEP and find the key is proportional to the amount of captured traffic. And this might require the need for several clients in order to generate the traffic. However, at times when there are very few clients connected to the AP, then traffic injection tools can be used to inject the traffic. Following are some of the techniques that can be used when there are only some clients.

- **ARP-Request injection** is one of the methods by which traffic can be injected into the network. It requires the MAC address as one of the parameter. This can be obtained through the Fake authentication method or by impersonating an already connected client.
- **Interactive Packet Replay** is used to inject packets of our own choice. Packet types can be checked in a pre-captured traffic file using Wireshark and similar packet types can be injected.

Q5. How can you obtain packets for injection if no clients are associated with the AP?

Korek chopshop is a traffic injection technique that can be used when there are no connected clients to the AP. This attack helps in decrypting a single packet. With the decrypted packet, we will be able to forge our own ARP requests using packetforge-ng, with which the required traffic for collecting the IVs can be generated.

Q6. Which weaknesses of WEP are we taking advantage of in our attack, and why?

One of the prime weakness of WEP that helped us in our attack is the **weak implementation of RC4 symmetric algorithm**. WEP keys are easily obtained because of its short implementation lengths. Even though WEP allows up to 104-bit keys, they are not sufficient. Moreover, with the limited number of initialization vectors (IV), the key streams also begin to repeat after a certain period of time which is a serious threat against the stream ciphers. With enough collected key streams, the WEP key can be retrieved at a shorter time period.

Q7. Under what circumstances would you consider WEP to be sufficiently secure?

WEP can never be secure as it is very trivial to break. WEP can be considered to be used in a closed network where the SSID is prevented from being transmitted and the MAC addresses are white listed. Such closed networks can be established for domestic purposes, but not for commercial environment. Such closed networks can provide an initial hindrance for an attacker to break into the network, but can never prevent the attack completely.

Q8. Would these attacks be effective against a network deploying WPA with TKIP? If the RC4 cipher used in WEP is considered to be insecure, why is it reused in WPA?

These attacks will not be effective against a system employing WPA/TKIP as it introduces more randomness into the system.

The main purpose of reusing the RC4 cipher in WPA is to make the system backward compatible with the WEP employed systems. Here in the case of WPA/TKIP, we are overcoming the issue over WEP by adding more randomness into the system by introducing 48 bit TKIP sequence numbers and transmit address to the per packet key mixing functionality.

3 Password Dictionary Attack

In this part the objective is to first setup a WPA/WPA2 pre-shared key (PSK) security enabled wireless access point and then perform a password dictionary attack against it. Wi-Fi Protected Access (WPA) is a certification program established by the Wi-Fi Alliance. The Wi-Fi Alliance started this certification program during the preparation of IEEE 802.11i standard. It was introduced due to the findings of several serious weaknesses of Wired Equivalent Privacy (WEP). WPA enabled devices implement a portion of the IEEE 802.11i standard which includes Temporal Key Integrity Protocol (TKIP). Later, when the IEEE 802.11i was standardized then it was termed as WPA2. The key difference between WPA and WPA2 is the encryption algorithm. WPA2 uses Advanced Encryption Standard (AES) based Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP) as the encryption algorithm whereas WPA uses RC4 based TKIP.

WPA/WPA2 uses a key hierarchy to derive different keys from a single pairwise master key (PMK). In WPA/WPA2 a pseudo random function is used to generate a 512 bits long Pairwise Transient Key (PTK) from the PMK. The function takes PMK, MAC address of AP, MAC address of Station, a Nonce generated by AP (ANonce) and a Nonce generated by Station (SNonce) as input and generates PTK as output. The secureness of WPA/WPA2 depends on the robustness of PMK. There are basically two different flavours of WPA/WPA2 which are: PSK and Enterprise. The Enterprise solution uses one of the available Extensible Authentication Protocol (EAP) for authentication whereas the PSK method uses a single secret key shared among all users for authentication.

In WPA/WPA2-PSK modes of operation usually the pre-shared key in generated from a shared password to make it user friendly. The function that is used to generate PSK is termed as PBKDF2 which takes password and SSID as input. The password can be between 8 to 63 printable ascii characters. In case of pre-shared key mechanism, PSK equals PMK which is known by both AP and station. Next to generate PTK the station and AP use a 4-way handshake mechanism termed as EAPOL 4-way handshake. In the EAPOL 4-way handshake the first 2 messages are exchanged unencrypted between the AP and station because those messages contain the nonce of station and AP which are required to generate the PTK and start encryption. As a result, if the password is not strong then an attacker can eavesdrop the 4-way handshake and perform a dictionary or brute force attack against it. The attacker only needs the first two messages of the 4-way handshake containing ANonce and SNonce to start guessing the password. The goal of this task is to perform a similar dictionary based attack on the WPA/WPA2-PSK mechanism to retrieve the password. The Figure 1 shows the lab setup. The hostapd configuration used to setup WPA2-PSK enabled access point is available in the Appendix 7.2.

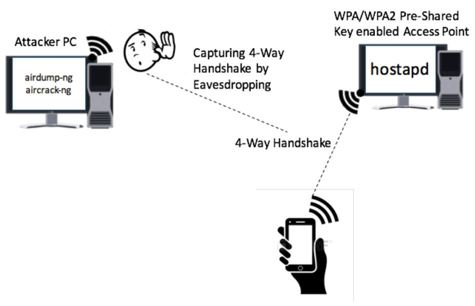

Attacker PC
airdump-ng
aircrack-ng
Capturing 4-Way Handshake by Eavesdropping
WPA/WPA2 Pre-Shared Key enabled Access Point
hostapd
4-Way Handshake

Figure 1: Lab Setup for Password Dictionary Attack

3.1 Result

The screen-shot of successful password recovery is available in the Appendix 7.3

3.2 Questions

Q9. Why is TKIP a more secure encryption mechanism than the one-key WEP alternative?

TKIP is more secure encryption mechanism than one-key WEP because: it uses extended Initialization Vector (IV) (48 bits) than WEP (24 bits) and sends IV as encrypted hash over the network. In addition, TKIP introduces cryptographic message integrity code (MIC) in the MSDU to protect integrity of the message and also per packet key mixing function which generates new encryption key for each packet. Lastly, TKIP also implements sequence counter to protect against replay attacks.

Q10. Which information is used to compute the PMK and the PTK in the password-based PSK scenario? Why is the offline password brute-force attack possible?

In order to calculate the PMK in a password based PSK scenario, the shared password, the SSID of the access point and the length of the SSID is required. However, to calculate PTK the PMK, Mac address of AP, MAC address of Station, nonce of AP and nonce of Station is required. The offline brute force attack is possible because the password is static which in turn makes PMK as well as the derived PTK static for each session. So, if the attacker gets the nonce of AP and Station from the 4-way handshake then he can brute-force the PTK in offline mode until it finds a match.

Q11. Does the compromise of password imply the disclosure of traffic from previous sessions? Justify your answer.

No the compromise of the password does not imply the disclosure of traffic from previous sessions because for each session the station nonce and AP nonce is different and hence the PTK is different. Therefore, if the ANonce and SNonce of the previous are not known then the previous session PTK can not be generated and traffic cannot be decrypted.

Q12. Does the use of AES in WPA2-PSK give an advantage against a password dictionary attack, as compared to RC4 in WPA-PSK?

No, the use of AES in WPA2-PSK does not provide any advantage against password dictionary attacks over RC4 in WPA-PSK. The reason is AES and RC4 are used for encrypting data using the negotiated key during authentication procedure and password dictionary attack is performed against the negotiated key of the authentication procedure. As a result, the attack does not have any relation with encryption algorithm and it remains same for both TKIP and AES.

Q13. What would be advantages and disadvantages of using a precomputed database of PMKs in a PSK password dictionary attack?

The advantage of precomputed database of PMKs is that it will drastically increase the number of passphrase look up per second as the hashes are already computed and hence will reduce the cracking time. However, the disadvantage of precomputed database is that it only contains PMKs calculated using very popular SSIDs because SSID is used as a salt in the PBKDF2 key derivation function. Therefore, if the target SSID is unique then it is not possible to find the key using precomputed database. And then only possibility is to perform brute force or dictionary based attacks by computing fresh key with the SSID of the access point under attack and each possible passphrase.

Q14. Suppose an attacker employs 30 days of processing time on the new supercomputer at NTNU (find out its processing power at [23] and recall the password enumeration speed in the lab). Which minimum requirements would you put on a WPA password to ensure that the probability of successful attack does not exceed 2^{-20}?

The NTNU super computer can perform 275 teraflops per second meaning 275x10^{12} flops per second. According to bitcoinwatch [1] 1 Hash = 12,697 flops. Therefore, 275x10^{12} flops = 275x10^{12} / 12,697 Hash which equals 2.16x10^{10} Hashes/s. That means in 30 days the supercomputer can calculate 2.16x10^{10}x86400x30 equals 5.6x10^{16} Hashes approx. For WPA-PSK to calculate each key it needs to perform 4096 hashes. The supercomputer can calculate 5.6x10^{16}/4096 equals 1.4x10^{13} keys in 30 days, which is close to $2^{43.67}$. Therefore, the entropy of the password should be at least 43.67 bits for 0.5 probability. Hence to achieve such entropy the minimum requirements for the WPA password is that the password must be 11 characters long with at least 1 uppercase 1 lower case and 1 number. If the characters are chosen randomly, uniformly, and independently of each other then such a password will have an entropy of close to 66 bits and the probability of successful attack will not exceed 2^{-20}.

4 Setting up RSN-EAP Wireless Access Point

The goal of this lab experiment was to setup Robust Security Network using Transport Layer Security (TLS) in conjunction with RSN over Extensible Authentication Protocol (EAP). The below diagram depicts the overall setup.

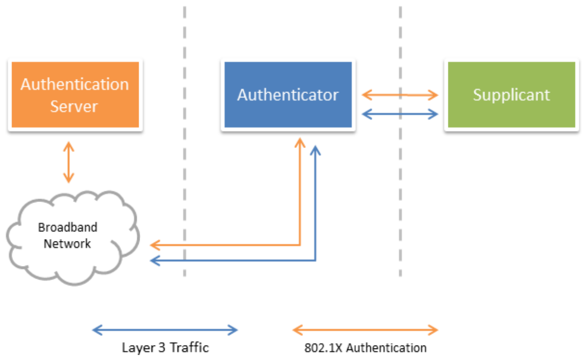


Figure 2: 802.1X (Taken from http://www.qacafe.com)

The communication between the supplicant and the AP is using EAPOL and the communication between the AP and the AS is by EAP over RADIUS(Remote Authentication Dial-In user Service). The setup consists of:

- Supplicant - The device which intends to connect to the internet via Access Point(AP).
- Authenticator - It is usually an AP. It relays the EAP messages between the Supplicant and the authentication server.
- Authentication Server (AS)- The centralized (or logically centralized) machine that performs the authentication of the supplicant. Based on the result of the authentication of a supplicant, appropriate success or failure message is conveyed to the

authenticator. The authenticator in turn grants access to the supplicant based on the success or failure message received from the AS.

In our lab setup, WiFi enabled desktop running Ubuntu 14.04 LTS was configured to act as a supplicant using wpa_supplicant software. The configuration file given in the lab description was used with the appropriate modifications to file paths (certificates path).

A laptop running Ubuntu 16.04 LTS was configured to perform the job of an authenticator or AP using hostapd software. The configuration file fed to hostapd in order to turn the laptop to function as an AP can be found in the Appendix 7.5.

The same laptop was configured to host an authentication server using the FreeRadius software. In reality the AS will be running remotely on a different machine to that of an authenticator and the authenticator is authenticated through a radius key that is shared between the AP and the AS. The files eap.conf and clients.conf were modified appropriately to set-up free radius server and the copy of the same can be found in the Appendix 7.6 and Appendix 7.7 respectively.

OpenSSL tool was used to generate certificates and keys of different participants of the network to establish public key infrastructure (PKI). Participants are the root certification authority (CA), supplicant, authenticator and the AS.

The major advantage of 802.1x is that the user can get seamless internet connection without disclosing his credentials to any other authentication servers than his home AS, provided an agreement exists between the various internet providers with his home service provider (eg Eduroam). In other words, when the supplicant visits other networks and wants to connect to internet, the AS of the visited network functions as a proxy radius server by forwarding the access request to the supplicants home AS (identified through the realm present in the identity response message of the supplicant) provided an agreement exists between them as mentioned earlier. This can be simulated in the lab by configuring a radius proxy server (by appropriately configuring the files proxy.conf and radiusd.conf) which forwards the access request to the appropriate AS and this can be chained i.e the other radius server can in turn be a proxy server.

4.1 Questions

Q15. When would it be appropriate to use WPA2-PSK instead of WPA2-EAP (as a key management scheme)?

Wi-Fi Protected Access-2 Pre-shared Key (WPA2-PSK) also known as WPA2-Personal is intended for small office and home networks as a key management scheme.This is because it is much easier to set-up and maintain WPA2-PSK in comparison to WPA2-EAP which demands more resources. Additionally, there are few situations where WPA-PSK is ideal to use. Firstly, when there are few trusted devices in the network and this could be a small office or home. Secondly, as a means to restrain casual users from accessing Internet in an open network. This may be a guest network or a coffee shop.

Q16. EAP-TLS deploys mutual authentication of two communicating parties. What kind of attack is possible against authentication protocols lacking such authentication?

The protocols lacking mutual authentication of two communicating parties are usually vulnerable to man-in-the-middle (MiM) attacks. Using mutual authentication process we ensure that neither the server nor the supplicant is spoofed or masqueraded.

Q17. Give an overview of the EAP authentication protocols which can be used in WPA2-Enterprise WLANs?

The methods included in WPA2 are: [3]

- EAP-TLS - It is the most secure EAP method and uses Transport Layer Security (TLS) for mutual authentication of supplicant and the authentication server through the exchange of digital certificates.
- EAP-TTLS - EAP Tunneled TLS extends TLS where a secure tunnel is established after the server successfully authenticates to the client by sending its digital certificate signed by a trusted CA (optionally the client may also authenticate by sending certificate to server).This secure channel can then be used to authenticate the client using any existing authentication protocol. Unlike TLS, the client need not have to produce certificate which in turn reduces the complexity at the client.
- EAP-MSCHAPv2- Microsoft's Challenge Handshake Authentication Protocol is the commonly used form of Protected EAP (PEAP). It is the second widely supported EAP method after EAP-TLS. A certificate must be used to authenticate the server to the client before the client submits its credentials for authentication. Otherwise, it would be easy to introduce rouge AP to capture MSCHAP handshake. MSCHAP is used as the inner authentication protocol. In MSCHAPv2, mutual authentication between the peers (server and the supplicant) is achieved via piggybacking of a peer's challenge on the response and the response of an authenticator on the success message. However, many weaknesses of MSCHAPv2 exist which greatly reduce the complexity of brute-force attack [5].
- EAP-GTC- EAP Generic Token Card- Authentication credentials can be exchanged across the network in clear text by employing EAP-GTC method. If the token card password is used properly, this method can alone be used without any tunneling via TTLS as it will not be vulnerable to reply attack. Nevertheless, EAP-GTC should be used inside a tunnel if a re-usable password is being transported over the network to protect it and also for server authentication. [4].
- EAP-SIM (Subscriber Identity Module) - It is used to authenticate mobile stations to the authentication server via EAP-SIM capable APs.
- EAP-FAST- Flexible Authentication via Secure Tunnelling is a protocol proposed as a replacement for LEAP by Cisco Systems. It overcomes the shortcomings of LEAP while keeping the implementation "lightweight". In EAP-FAST server side certificate is optional and it employs Protected Access Credential (PAC) in order to set-up a TLS tunnel inside which the verification of client credentials is done.
- EAP-AKA- Similar to EAP-SIM, EAP- AKS is an EAP method for UMTS (Universal Mobile Telecommunications System) authentication and key agreement(AKA). It is used for the distribution of session keys and authentication using USIM.

Q18. List the security-related protocols used in your RSN-EAP-TLS setup and explain their purpose?

The security-related protocols used in RSN-EAP-TLS are:

- Diffie-Helman key exchange protocol- It is used to negotiate TLS session keys between the client/supplicant and the authentication server. In specific, it is used to derive pre-master key between the client and the server. [2]

- CCMP - Counter Mode Cipher Block Chaining Message Authentication Code Protocol, is the encryption protocol used to encrypt the traffic between the supplicant and the AP (enforcing link confidentiality). [9]
- TLS - Transport Layer Security protocol is used for mutual authentication between the supplicant and the server, integrity-protected ciphersuite negotiation and master key exchange between the supplicant and the server [6]. The mutual authentication is done through the use of digital certificates (X.509 standard) issued by a trusted certificate authority (CA). The client and the server exchange and verify each others certificate.

5 Mobile Network Security

The motive of this experiment is to set up and configure a GSM test network. We made use of a USRP device B200mini which acts as a base station. The functionality of the device is controlled by OpenBTS, an open source software running on a separate computer. Once after establishing the network, we test the network by connecting a Samsung S6 handset to the new network. Later, in the course of the experiment, we made use of another handset to test additional functionality of the network.

The below image shows an overview of the established network.

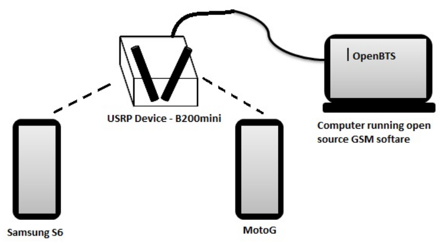


Figure 3: GSM Mobile Network Setup

Apart from the experimental procedure, we also tracked the value of the K_i in the sip_buddies table of the sqlite database and it was found empty. The screen shot of the same can be found in the Appendix 7.4

5.1 Questions

Q20. Make a technical specification of the equipment you used and the mobile network you set up.

The experiment was carried out with the following two equipments:

- **Universal Software Radio Peripheral (USRP)** A B200mini was used as a base station. It is associated with two antennas for receiving and transmitting. The frequency related details are as below:
  - Operating frequency : 1900 MHz with 584 ARFCN
  - Operating Mode : Full duplex

- **Mobile Handset**: Two mobile handsets were used in the experiment: a Samsung Galaxy S6 and a MotoG. Both are smart phones equipped with 2G/3G/LTE features. Before the start of the experiment, the handsets were configured to operate at 2G mode by changing their settings. They are used for connecting to the newly established network. The handsets are able to operate at the frequency 1900 MHz.

Q21. Note similarities and differences between the OpenBTS implementation you experimented with and the ideal textbook/lecture description with respect to the usage of the subscriber's temporary identity (TMSI).

Based on our understanding of the lectures, we were able to note some similarities and differences with respect to the TMSI usage.

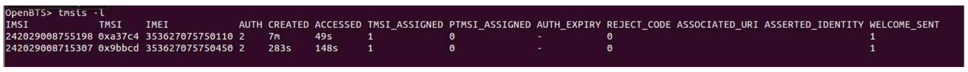
The similarity is that, during the experimentation we were able to observe the TMSIs getting generated for the mobile stations once after their IMSIs were identified by the network. This also proves that the TMSIs are assigned by the authentication servers, which in our case is taken care by OpenBTS.

The difference we observed during the experiment that was contradicting to our lectures was the disclosure of IMSI values even after assigning TMSI. We were able to locate the IMSI values in the captured Wireshark packets even after the TMSI allocation. However, as per the lectures, IMSI is not supposed to be disclosed after the initial authentication.

Q22. Explain the technical differences between 'Open Registration' and 'Cached Authentication' based on the data in your access database (TMSI-table) and the authentication messages that you captured. Exemplify.

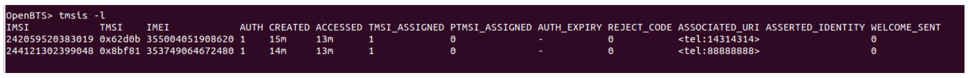
From the access database, we can note that the value within the **AUTH** column differs when the handset was connected to the network under the two different authentication methods.

Under Open Registration, the **AUTH** column had a value of **2** whereas under Cached Authentication, the **AUTH** column had a value of **1**. Additionally, in the TMSI table, we can observe the MSIN values under the **ASSOCIATED_URI** column for cached authentication. However, the Open registration method has no entry of the MSISDN. The same can be seen in the screen shot below.



```
OpenBTS> tmsis -l
IMSI            TMSI     IMEI            AUTH CREATED ACCESSED TMSI_ASSIGNED PTMSI_ASSIGNED AUTH_EXPIRY REJECT_CODE ASSOCIATED_URI ASSERTED_IDENTITY WELCOME_SENT
242029008755198 0xa37c4  353627075750110 2    7m      49s      1             0              -           0                                             1
242029008715307 0x9bbcd  353627075750450 2    283s    148s     1             0              -           0                                             1
```

Figure 4: TMSI table for Open Registration



```
OpenBTS> tmsis -l
IMSI            TMSI     IMEI            AUTH CREATED ACCESSED TMSI_ASSIGNED PTMSI_ASSIGNED AUTH_EXPIRY REJECT_CODE ASSOCIATED_URI ASSERTED_IDENTITY WELCOME_SENT
242059520383019 0x62d0b  355004051908620 1    15m     13m      1             0              -           0           <tel:14314314>                    0
244121302399048 0x8bf81  353749064672480 1    14m     13m      1             0              -           0           <tel:88888888>                    0
```

Figure 5: TMSI table for Cached Authentication

On probing the captured packets of both Open Registration and Cached Authentication, we were able to identify that the Open Registration did not involve in the exchange of authentication requests and response. However, the Cached Authentication was involved in the handshake procedure, where the authentication messages were exchanged between the mobile station and the authentication server. Apart from that, both the methods involve in the exchange of packets related to location registration, identity registration, TMSI allocation, etc.

Q23. Compare the OpenBTS encryption behaviour to the GSM encryption standard. Describe the technical differences that you can find when OpenBTS encryption is disabled and when it is enabled, based on data in your 'TMSI-table', the key Ki, and the protocol messages that you captured. Exemplify.

The key is only stored in the SIM and in the authentication server. And in this case, since that we do not own the SIM, we are unaware of the key value. There is no difference between the encryption and non-encryption methods as we do not possess the key. As far as the TMSI-table is concerned, there are no differences in the entries. Even after enabling the encryption, it was found that the captured traffic was unencrypted. Also, the OpenBTS manual confirms that the encryption is not supported in OpenBTS.

Q24. Which security modes of operation are signalled to the user by the phone's display? In particular, consider various types of registration, authentication, and encryption modes.

Once after the base station network is established through the B200mini, the following modes of operation can be seen in the by the user in the handset:

- The user will be able to see the newly established network in the list of available networks.
- Once the user tries to connect to the new network, the users IMSI is registered by the authentication server.
- Once after successful authentication, the user is sent with a registration message with the details of the IMSI. At this point, the user will also be able to view the new network name if the users device is configured to show it.
- No security related information was shared by the network operator to the handset, even though it is a mandatory concern.

Q25. Compare OpenBTS 'Cached Authentication' to the textbook/lecture/standard description of the GSM authentication. Why cannot we do the complete authentication protocol in this lab? What are required to perform the complete GSM authentication protocol, and how do you suggest we should do this in the next year's assignment?

Both OpenBTS authentication and GSM authentication follow a similar procedure. The only difference is that the OpenBTS replaces the BSS and the subscriber registry replaces the HLR/AuC. We are not able to perform the complete authentication here as we neither own the key nor we have the triplet (K_c, XRES,RAND) from the HLR/AuC that is required for completing the authentication. The key is only stored in the SIM and in the home authentication server. And since we do not own the SIM, we dont have the master key. As a result we are unable to perform the complete authentication of the user. In order to perform the complete GSM authentication we either need the master key or the triplet from the HLR. We suggest to fabricate our own dummy SIM cards with our master key embedded in the SIM, provided it is legal to do so.

6 Discussion & Concluding Remarks

The primary goal of the lab is to have a detailed idea of the wireless security concepts in both WLAN and mobile networks. All the milestones were met successfully and the results as well as the procedures were analysed. The experiments helped us to understand how tedious it is to design a fully secured wireless network heeding to all the necessary requirements.

The experiments helped us to gather sufficient knowledge about the practical implementation of the wireless security concepts and the lab manual was very descriptive providing a step by step guide for approaching the experiments. However, we found few of the questions to be ambiguous that required us to do extensive research in order to understand the question requirements and come up with the correct answer which was difficult within the given time frame. Apart from the improvement suggested in Q25, we can have inter group activities where each group can give a challenge to the opposite group to be completed. Based on the content requirements, it would have been better if the page count is a bit relaxed.

References

[1] Bit coin. `http://bitcoinwatch.com/`, note = Accessed: 2016-10-23.

[2] Eap-tls deployment guide for wireless lan networks [wireless, lan (wlan)] - cisco systems. `http://www.cisco.com/en/US/tech/tk722/tk809/technologies_white_paper09186a008009256b.shtml`. (Accessed on 10/22/2016).

[3] Wi-fi certified expanded to support eap-aka and eap-fast authentication mechanisms — wi-fi alliance. `http://www.wi-fi.org/news-events/newsroom/wi-fi-certified-expanded-to-support-eap-aka-and-eap-fast-authentication`. (Accessed on 10/22/2016).

[4] Bernard Aboba, L Blunk, J Vollbrecht, James Carlson, and Henrik Levkowetz. Rfc 3748-extensible authentication protocol (eap). *Network Working Group*, 2004.

[5] Jochen Eisinger. Exploiting known security holes in microsofts pptp authentication extensions (ms-chapv2). *University of Freiburg,[cit. 2008-27-05] Dostupné z:¡ http://penguinbreeder. org/pptp/download/pptp_mschapv2. pdf*, 2001.

[6] Dan Simon, Bernard Aboba, and Ryan Hurst. The eap-tls authentication protocol. Technical report, 2008.

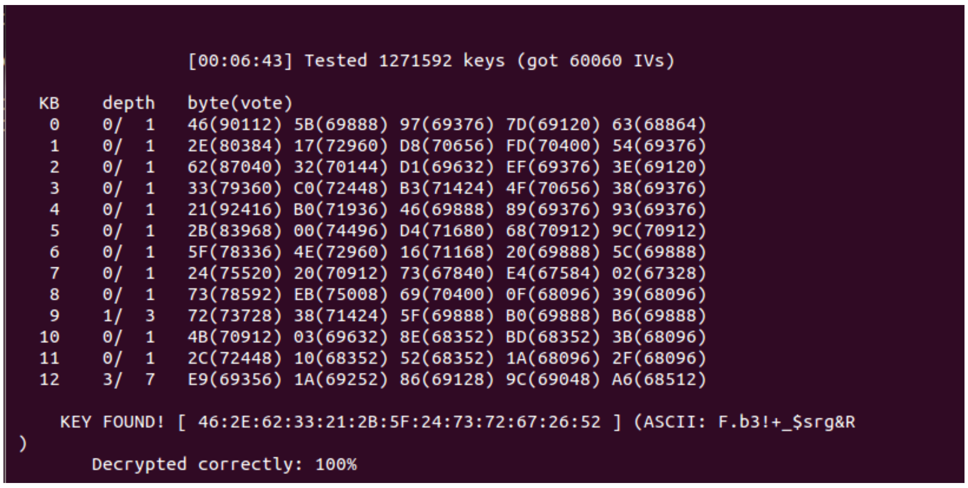
[7] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. In *International Workshop on Information Security Applications*, pages 188–202. Springer, 2007.

[8] Alfonso Valdes and Diego Zamboni. *Recent Advances in Intrusion Detection: 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, 2005, Revised Papers*, volume 3858. Springer, 2006.

[9] Doug Whiting, Niels Ferguson, and Russell Housley. Counter with cbc-mac (ccm). 2003.

7 Appendix

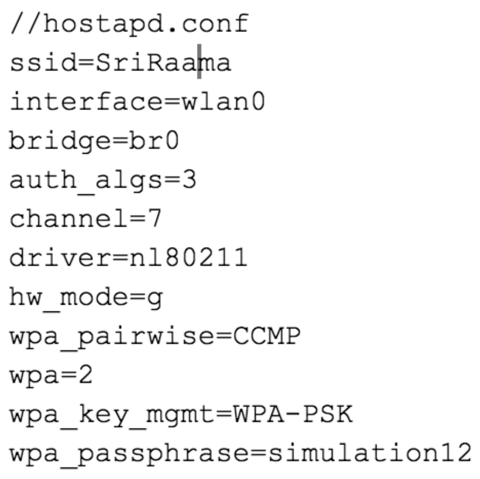
7.1 PTW Attack



```
                 [00:06:43] Tested 1271592 keys (got 60060 IVs)

   KB    depth   byte(vote)
    0    0/  1   46(90112) 5B(69888) 97(69376) 7D(69120) 63(68864)
    1    0/  1   2E(80384) 17(72960) D8(70656) FD(70400) 54(69376)
    2    0/  1   62(87040) 32(70144) D1(69632) EF(69376) 3E(69120)
    3    0/  1   33(79360) C0(72448) B3(71424) 4F(70656) 38(69376)
    4    0/  1   21(92416) B0(71936) 46(69888) 89(69376) 93(69376)
    5    0/  1   2B(83968) 00(74496) D4(71680) 68(70912) 9C(70912)
    6    0/  1   5F(78336) 4E(72960) 16(71168) 20(69888) 5C(69888)
    7    0/  1   24(75520) 20(70912) 73(67840) E4(67584) 02(67328)
    8    0/  1   73(78592) EB(75008) 69(70400) 0F(68096) 39(68096)
    9    1/  3   72(73728) 38(71424) 5F(69888) B0(69888) B6(69888)
   10    0/  1   4B(70912) 03(69632) 8E(68352) BD(68352) 3B(68096)
   11    0/  1   2C(72448) 10(68352) 52(68352) 1A(68096) 2F(68096)
   12    3/  7   E9(69356) 1A(69252) 86(69128) 9C(69048) A6(68512)

     KEY FOUND! [ 46:2E:62:33:21:2B:5F:24:73:72:67:26:52 ] (ASCII: F.b3!+_$srg&R
)
        Decrypted correctly: 100%
```

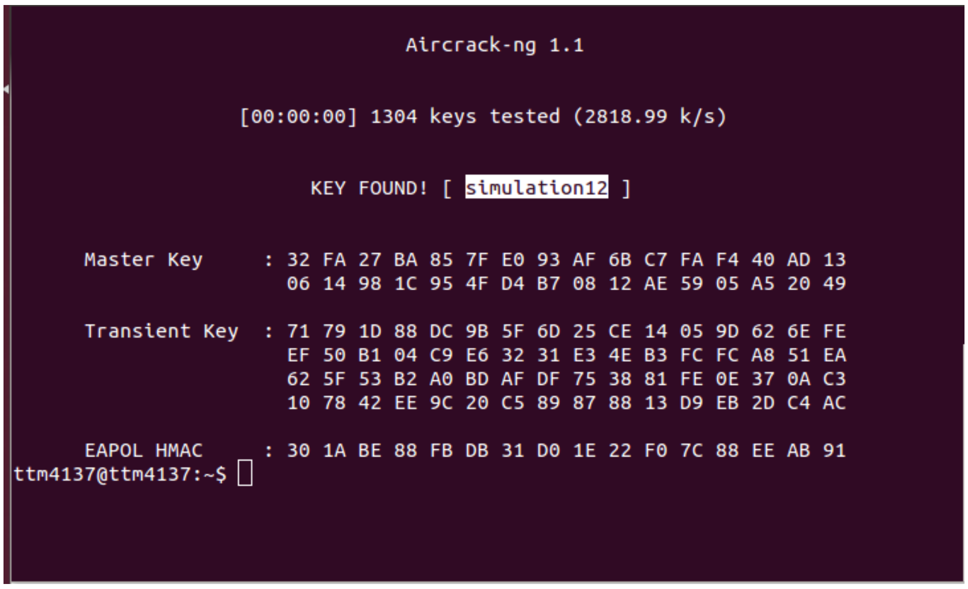
Figure 6: PTW attack using aircrack-ng

7.2 Hostapd configuration for WPA-PSK



```
//hostapd.conf
ssid=SriRaama
interface=wlan0
bridge=br0
auth_algs=3
channel=7
driver=nl80211
hw_mode=g
wpa_pairwise=CCMP
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_passphrase=simulation12
```

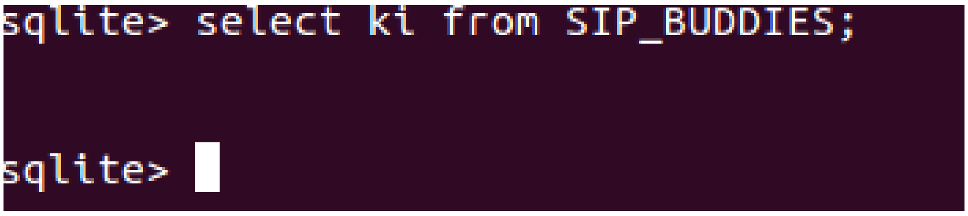
7.3 WPA/WPA2 Pre-shared Key Recovery



```
                              Aircrack-ng 1.1


                   [00:00:00] 1304 keys tested (2818.99 k/s)


                          KEY FOUND! [ simulation12 ]


      Master Key     : 32 FA 27 BA 85 7F E0 93 AF 6B C7 FA F4 40 AD 13
                       06 14 98 1C 95 4F D4 B7 08 12 AE 59 05 A5 20 49

      Transient Key  : 71 79 1D 88 DC 9B 5F 6D 25 CE 14 05 9D 62 6E FE
                       EF 50 B1 04 C9 E6 32 31 E3 4E B3 FC FC A8 51 EA
                       62 5F 53 B2 A0 BD AF DF 75 38 81 FE 0E 37 0A C3
                       10 78 42 EE 9C 20 C5 89 87 88 13 D9 EB 2D C4 AC

      EAPOL HMAC     : 30 1A BE 88 FB DB 31 D0 1E 22 F0 7C 88 EE AB 91
ttm4137@ttm4137:~$ 
```

Figure 7: Successful WPA/WPA2 Pre-shared Key Recovery

7.4 Value of K_i in sip_buddies



```
sqlite> select ki from SIP_BUDDIES;


sqlite> 
```

Figure 8: K_i in sip_buddies

7.5 HostAppd Configuration File

```
# -*- text -*-
##
##  hostapd.conf -- Configuration for AP supporting EAP
##
##

#######################################################################
ssid=SriRaama
interface=wlo1
bridge=br0
auth_algs=3
channel=7
driver=nl80211
hw_mode=g
max_num_sta=5
rsn_pairwise=CCMP
wpa=2
wpa_key_mgmt=WPA-EAP
ieee8021x=1
eapol_version=2
own_ip_addr=127.0.0.1

auth_server_addr=127.0.0.1
auth_server_port=1812
auth_server_shared_secret=radius123

ca_cert=/home/shiva/CA/cacert.pem
server_cert=/home/shiva/CA/newcerts/ascert.pem
private_key=/home/shiva/CA/private/asreq.pem
private_key_passwd=1234
```

7.6 EAP configuration File

```
# -*- text -*-
##
##  eap.conf -- Configuration for EAP types (PEAP, TTLS, etc.)
##
##      $Id$

#######################################################################
#
#  Whatever you do, do NOT set 'Auth-Type := EAP'.  The server
#  is smart enough to figure this out on its own.  The most
#  common side effect of setting 'Auth-Type := EAP' is that the
#  users then cannot use ANY other authentication method.
#
#  EAP types NOT listed here may be supported via the "eap2" module.
#  See experimental.conf for documentation.
#
        eap {
                #  Invoke the default supported EAP type when
                #  EAP-Identity response is received.
                #
                #  The incoming EAP messages DO NOT specify which EAP
                #  type they will be using, so it MUST be set here.
                #
                #  For now, only one default EAP type may be used at a time.
                #
                #  If the EAP-Type attribute is set by another module,
                #  then that EAP type takes precedence over the
                #  default type configured here.
                #
                default_eap_type = tls

                #  A list is maintained to correlate EAP-Response
                #  packets with EAP-Request packets.  After a
                #  configurable length of time, entries in the list
                #  expire, and are deleted.
                #
                timer_expire     = 60

                #  There are many EAP types, but the server has support
                #  for only a limited subset.  If the server receives
                #  a request for an EAP type it does not support, then
                #  it normally rejects the request.  By setting this
                #  configuration to "yes", you can tell the server to
                #  instead keep processing the request.  Another module
                #  MUST then be configured to proxy the request to
                #  another RADIUS server which supports that EAP type.
                #
                #  If another module is NOT configured to handle the
                #  request, then the request will still end up being
                #  rejected.
                ignore_unknown_eap_types = no

                # Cisco AP1230B firmware 12.2(13)JA1 has a bug.  When given
                # a User-Name attribute in an Access-Accept, it copies one
                # more byte than it should.
                #
                # We can work around it by configurably adding an extra
                # zero byte.
                cisco_accounting_username_bug = no

                #
                #  Help prevent DoS attacks by limiting the number of
                #  sessions that the server is tracking.  For simplicity,
                #  this is taken from the "max_requests" directive in
                #  radiusd.conf.
                max_sessions = ${max_requests}

                # Supported EAP-types

                #
                #  We do NOT recommend using EAP-MD5 authentication
                #  for wireless connections.  It is insecure, and does
                #  not provide for dynamic WEP keys.
                #
                md5 {
                }

                # Cisco LEAP
                #
                #  We do not recommend using LEAP in new deployments.  See:
                #  http://www.securiteam.com/tools/5TP012ACKE.html
                #
                #  Cisco LEAP uses the MS-CHAP algorithm (but not
                #  the MS-CHAP attributes) to perform it's authentication.
                #
                #  As a result, LEAP *requires* access to the plain-text
                #  User-Password, or the NT-Password attributes.
                #  'System' authentication is impossible with LEAP.
                #
                leap {
                }

                #  Generic Token Card.
                #
                #  Currently, this is only permitted inside of EAP-TTLS,
                #  or EAP-PEAP.  The module "challenges" the user with
                #  text, and the response from the user is taken to be
                #  the User-Password.
                #
                #  Proxying the tunneled EAP-GTC session is a bad idea,
                #  the users password will go over the wire in plain-text,
                #  for anyone to see.
                #
                gtc {
                        #  The default challenge, which many clients
                        #  ignore..
                        #challenge = "Password: "

                        #  The plain-text response which comes back
                        #  is put into a User-Password attribute,
                        #  and passed to another module for
                        #  authentication.  This allows the EAP-GTC
                        #  response to be checked against plain-text,
                        #  or crypt'd passwords.
                        #
                        #  If you say "Local" instead of "PAP", then
                        #  the module will look for a User-Password
                        #  configured for the request, and do the
                        #  authentication itself.
                        #
                        auth_type = PAP
                }

                ## EAP-TLS
                #
                #  See raddb/certs/README for additional comments
                #  on certificates.
                #
```

linktodoc=true

```
#  If OpenSSL was not found at the time the server was
#  built, the "tls", "ttls", and "peap" sections will
#  be ignored.
#
#  Otherwise, when the server first starts in debugging
#  mode, test certificates will be created.  See the
#  "make_cert_command" below for details, and the README
#  file in raddb/certs
#
#  These test certificates SHOULD NOT be used in a normal
#  deployment.  They are created only to make it easier
#  to install the server, and to perform some simple
#  tests with EAP-TLS, TTLS, or PEAP.
#
#  See also:
#
#  http://www.dslreports.com/forum/remark,9286052~mode=flat
#
#  Note that you should NOT use a globally known CA here!
#  e.g. using a Verisign cert as a "known CA" means that
#  ANYONE who has a certificate signed by them can
#  authenticate via EAP-TLS!  This is likely not what you want.
tls {
        #
        #  These is used to simplify later configurations.
        #
        certdir = /home/shiva/CA
        cadir = /home/shiva/CA

        private_key_password = 1234
        private_key_file = ${certdir}/private/asreq.pem

        #  If Private key & Certificate are located in
        #  the same file, then private_key_file &
        #  certificate_file must contain the same file
        #  name.
        #
        #  If CA_file (below) is not used, then the
        #  certificate_file below MUST include not
        #  only the server certificate, but ALSO all
        #  of the CA certificates used to sign the
        #  server certificate.
        certificate_file = ${certdir}/newcerts/ascert.pem

        #  Trusted Root CA list
        #
        #  ALL of the CA's in this list will be trusted
        #  to issue client certificates for authentication.
        #
        #  In general, you should use self-signed
        #  certificates for 802.1x (EAP) authentication.
        #  In that case, this CA file should contain
        #  *one* CA certificate.
        #
        #  This parameter is used only for EAP-TLS,
        #  when you issue client certificates.  If you do
        #  not use client certificates, and you do not want
        #  to permit EAP-TLS authentication, then delete
        #  this configuration item.
        CA_file = ${cadir}/cacert.pem

        #
        #  For DH cipher suites to work, you have to
        #  run OpenSSL to create the DH file first:
```

```
                        #
                        #       openssl dhparam -out certs/dh 1024
                        #
                        dh_file = ${certdir}/dh
                        random_file = ${certdir}/random


                        #
                        #  This can never exceed the size of a RADIUS
                        #  packet (4096 bytes), and is preferably half
                        #  that, to accomodate other attributes in
                        #  RADIUS packet.  On most APs the MAX packet
                        #  length is configured between 1500 - 1600
                        #  In these cases, fragment size should be
                        #  1024 or less.
                        #
                #       fragment_size = 1024

                        #  include_length is a flag which is
                        #  by default set to yes If set to
                        #  yes, Total Length of the message is
                        #  included in EVERY packet we send.
                        #  If set to no, Total Length of the
                        #  message is included ONLY in the
                        #  First packet of a fragment series.
                        #
                #       include_length = yes

                        #  Check the Certificate Revocation List
                        #
                        #  1) Copy CA certificates and CRLs to same directory.
                        #  2) Execute 'c_rehash <CA certs&CRLs Directory>'.
                        #    'c_rehash' is OpenSSL's command.
                        #  3) uncomment the lines below.
                        #  5) Restart radiusd
                #       check_crl = yes

                        # Check if intermediate CAs have been revoked.
                #       check_all_crl = yes

                        CA_path = ${cadir}

                       #
                       #  If check_cert_issuer is set, the value will
                       #  be checked against the DN of the issuer in
                       #  the client certificate.  If the values do not
                       #  match, the cerficate verification will fail,
                       #  rejecting the user.
                       #
                       #  In 2.1.10 and later, this check can be done
                       #  more generally by checking the value of the
                       #  TLS-Client-Cert-Issuer attribute.  This check
                       #  can be done via any mechanism you choose.
                       #
                #       check_cert_issuer = "/C=GB/ST=Berkshire/L=Newbury/O=My
Company Ltd"

                       #
                       #  If check_cert_cn is set, the value will
                       #  be xlat'ed and checked against the CN
                       #  in the client certificate.  If the values
                       #  do not match, the certificate verification
                       #  will fail rejecting the user.
                       #
```

```
			#  This check is done only if the previous
			#  "check_cert_issuer" is not set, or if
			#  the check succeeds.
			#
			#  In 2.1.10 and later, this check can be done
			#  more generally by checking the value of the
			#  TLS-Client-Cert-CN attribute.  This check
			#  can be done via any mechanism you choose.
			#
	#		check_cert_cn = %{User-Name}
	#
			# Set this option to specify the allowed
			# TLS cipher suites.  The format is listed
			# in "man 1 ciphers".
			cipher_list = "DEFAULT"

			#
			# As part of checking a client certificate, the EAP-TLS
			# sets some attributes such as TLS-Client-Cert-CN. This
			# virtual server has access to these attributes, and can
			# be used to accept or reject the request.
			#
	#		virtual_server = check-eap-tls

			# This command creates the initial "snake oil"
			# certificates when the server is run as root,
			# and via "radiusd -X".
			#
			# As of 2.1.11, it *also* checks the server
			# certificate for validity, including expiration.
			# This means that radiusd will refuse to start
			# when the certificate has expired.  The alternative
			# is to have the 802.1X clients refuse to connect
			# when they discover the certificate has expired.
			#
			# Debugging client issues is hard, so it's better
			# for the server to print out an error message,
			# and refuse to start.
			#
			make_cert_command = "${certdir}/bootstrap"

			#
			#  Elliptical cryptography configuration
			#
			#  Only for OpenSSL >= 0.9.8.f
			#
			ecdh_curve = "prime256v1"

			#
			#  Session resumption / fast reauthentication
			#  cache.
			#
			#  The cache contains the following information:
			#
			#  session Id - unique identifier, managed by SSL
			#  User-Name  - from the Access-Accept
			#  Stripped-User-Name - from the Access-Request
			#  Cached-Session-Policy - from the Access-Accept
			#
			#  The "Cached-Session-Policy" is the name of a
			#  policy which should be applied to the cached
			#  session.  This policy can be used to assign
			#  VLANs, IP addresses, etc.  It serves as a useful
			#  way to re-apply the policy from the original
```

```
#  Access-Accept to the subsequent Access-Accept
#  for the cached session.
#
#  On session resumption, these attributes are
#  copied from the cache, and placed into the
#  reply list.
#
#  You probably also want "use_tunneled_reply = yes"
#  when using fast session resumption.
#
cache {
      #
      #  Enable it.  The default is "no".
      #  Deleting the entire "cache" subsection
      #  Also disables caching.
      #
      #  You can disallow resumption for a
      #  particular user by adding the following
      #  attribute to the control item list:
      #
      #         Allow-Session-Resumption = No
      #
      #  If "enable = no" below, you CANNOT
      #  enable resumption for just one user
      #  by setting the above attribute to "yes".
      #
      enable = no

      #
      #  Lifetime of the cached entries, in hours.
      #  The sessions will be deleted after this
      #  time.
      #
      lifetime = 24 # hours

      #
      #  The maximum number of entries in the
      #  cache.  Set to "0" for "infinite".
      #
      #  This could be set to the number of users
      #  who are logged in... which can be a LOT.
      #
      max_entries = 255
}

#
#  As of version 2.1.10, client certificates can be
#  validated via an external command.  This allows
#  dynamic CRLs or OCSP to be used.
#
#  This configuration is commented out in the
#  default configuration.  Uncomment it, and configure
#  the correct paths below to enable it.
#
verify {
        #  A temporary directory where the client
        #  certificates are stored.  This directory
        #  MUST be owned by the UID of the server,
        #  and MUST not be accessible by any other
        #  users.  When the server starts, it will do
        #  "chmod go-rwx" on the directory, for
        #  security reasons.  The directory MUST
        #  exist when the server starts.
        #
```

linktodoc=true

```
                                #  You should also delete all of the files
                                #  in the directory when the server starts.
                #               tmpdir = /tmp/radiusd

                                #  The command used to verify the client cert.
                                #  We recommend using the OpenSSL command-line
                                #  tool.
                                #
                                #  The ${..CA_path} text is a reference to
                                #  the CA_path variable defined above.
                                #
                                #  The %{TLS-Client-Cert-Filename} is the name
                                #  of the temporary file containing the cert
                                #  in PEM format.  This file is automatically
                                #  deleted by the server when the command
                                #  returns.
                #               client = "/path/to/openssl verify -CApath
${..CA_path} %{TLS-Client-Cert-Filename}"
                        }

                        #
                        #  OCSP Configuration
                        #  Certificates can be verified against an OCSP
                        #  Responder. This makes it possible to immediately
                        #  revoke certificates without the distribution of
                        #  new Certificate Revokation Lists (CRLs).
                        #
                        ocsp {
                              #
                              #  Enable it.  The default is "no".
                              #  Deleting the entire "ocsp" subsection
                              #  Also disables ocsp checking
                              #
                              enable = no

                              #
                              #  The OCSP Responder URL can be automatically
                              #  extracted from the certificate in question.
                              #  To override the OCSP Responder URL set
                              #  "override_cert_url = yes".
                              #
                              override_cert_url = yes

                              #
                              #  If the OCSP Responder address is not
                              #  extracted from the certificate, the
                              #  URL can be defined here.

                              #
                              #  Limitation: Currently the HTTP
                              #  Request is not sending the "Host: "
                              #  information to the web-server.  This
                              #  can be a problem if the OCSP
                              #  Responder is running as a vhost.
                              #
                              url = "http://127.0.0.1/ocsp/"

                              #
                              # If the OCSP Responder can not cope with nonce
                              # in the request, then it can be disabled here.
                              #
                              # For security reasons, disabling this option
                              # is not recommended as nonce protects against
                              # replay attacks.
```

linktodoc=true

```
                        #
                        # Note that Microsoft AD Certificate Services OCSP
                        # Responder does not enable nonce by default. It is
                        # more secure to enable nonce on the responder than
                        # to disable it in the query here.
                        # See http://technet.microsoft.com/en-us/library/
cc770413%28WS.10%29.aspx
                        #
                        # use_nonce = yes

                        #
                        # Number of seconds before giving up waiting
                        # for OCSP response. 0 uses system default.
                        #
                        # timeout = 0

                        #
                        # Normally an error in querying the OCSP
                        # responder (no response from server, server did
                        # not understand the request, etc) will result in
                        # a validation failure.
                        #
                        # To treat these errors as 'soft' failures and
                        # still accept the certificate, enable this
                        # option.
                        #
                        # Warning: this may enable clients with revoked
                        # certificates to connect if the OCSP responder
                        # is not available. Use with caution.
                        #
                        # softfail = no
                }
        }

        #  The TTLS module implements the EAP-TTLS protocol,
        #  which can be described as EAP inside of Diameter,
        #  inside of TLS, inside of EAP, inside of RADIUS...
        #
        #  Surprisingly, it works quite well.
        #
        #  The TTLS module needs the TLS module to be installed
        #  and configured, in order to use the TLS tunnel
        #  inside of the EAP packet.  You will still need to
        #  configure the TLS module, even if you do not want
        #  to deploy EAP-TLS in your network.  Users will not
        #  be able to request EAP-TLS, as it requires them to
        #  have a client certificate.  EAP-TTLS does not
        #  require a client certificate.
        #
        #  You can make TTLS require a client cert by setting
        #
        #       EAP-TLS-Require-Client-Cert = Yes
        #
        #  in the control items for a request.
        #
        ttls {
                #  The tunneled EAP session needs a default
                #  EAP type which is separate from the one for
                #  the non-tunneled EAP module.  Inside of the
                #  TTLS tunnel, we recommend using EAP-MD5.
                #  If the request does not contain an EAP
                #  conversation, then this configuration entry
                #  is ignored.
                default_eap_type = md5
                #  The tunneled authentication request does
                #  not usually contain useful attributes
                #  like 'Calling-Station-Id', etc.  These
                #  attributes are outside of the tunnel,
                #  and normally unavailable to the tunneled
                #  authentication request.
                #
                #  By setting this configuration entry to
                #  'yes', any attribute which NOT in the
                #  tunneled authentication request, but
                #  which IS available outside of the tunnel,
                #  is copied to the tunneled request.
                #
                # allowed values: {no, yes}
                copy_request_to_tunnel = no

                #  The reply attributes sent to the NAS are
                #  usually based on the name of the user
                #  'outside' of the tunnel (usually
                #  'anonymous').  If you want to send the
                #  reply attributes based on the user name
                #  inside of the tunnel, then set this
                #  configuration entry to 'yes', and the reply
                #  to the NAS will be taken from the reply to
                #  the tunneled request.
                #
                # allowed values: {no, yes}
                use_tunneled_reply = no

                #
                #  The inner tunneled request can be sent
                #  through a virtual server constructed
                #  specifically for this purpose.
                #
                #  If this entry is commented out, the inner
                #  tunneled request will be sent through
                #  the virtual server that processed the
                #  outer requests.
                #
                virtual_server = "inner-tunnel"

                #  This has the same meaning as the
                #  same field in the "tls" module, above.
                #  The default value here is "yes".
        #       include_length = yes
        }

        ##################################################
        #
        #  !!!!! WARNINGS for Windows compatibility  !!!!!
        #
        ##################################################
        #
        #  If you see the server send an Access-Challenge,
        #  and the client never sends another Access-Request,
        #  then
        #
        #               STOP!
        #
        #  The server certificate has to have special OID's
        #  in it, or else the Microsoft clients will silently
        #  fail.  See the "scripts/xpextensions" file for
        #  details, and the following page:
```

linktodoc=true

```
	#
	#	http://support.microsoft.com/kb/814394/en-us
	#
	#  For additional Windows XP SP2 issues, see:
	#
	#	http://support.microsoft.com/kb/885453/en-us
	#
	#
	#  If is still doesn't work, and you're using Samba,
	#  you may be encountering a Samba bug.  See:
	#
	#	https://bugzilla.samba.org/show_bug.cgi?id=6563
	#
	#  Note that we do not necessarily agree with their
	#  explanation... but the fix does appear to work.
	#
	##################################################

	#
	#  The tunneled EAP session needs a default EAP type
	#  which is separate from the one for the non-tunneled
	#  EAP module.  Inside of the TLS/PEAP tunnel, we
	#  recommend using EAP-MS-CHAPv2.
	#
	#  The PEAP module needs the TLS module to be installed
	#  and configured, in order to use the TLS tunnel
	#  inside of the EAP packet.  You will still need to
	#  configure the TLS module, even if you do not want
	#  to deploy EAP-TLS in your network.  Users will not
	#  be able to request EAP-TLS, as it requires them to
	#  have a client certificate.  EAP-PEAP does not
	#  require a client certificate.
	#
	#
	#  You can make PEAP require a client cert by setting
	#
	#	EAP-TLS-Require-Client-Cert = Yes
	#
	#  in the control items for a request.
	#
	peap {
		#  The tunneled EAP session needs a default
		#  EAP type which is separate from the one for
		#  the non-tunneled EAP module.  Inside of the
		#  PEAP tunnel, we recommend using MS-CHAPv2,
		#  as that is the default type supported by
		#  Windows clients.
		default_eap_type = mschapv2

		#  the PEAP module also has these configuration
		#  items, which are the same as for TTLS.
		copy_request_to_tunnel = no
		use_tunneled_reply = no

		#  When the tunneled session is proxied, the
		#  home server may not understand EAP-MSCHAP-V2.
		#  Set this entry to "no" to proxy the tunneled
		#  EAP-MSCHAP-V2 as normal MSCHAPv2.
	#	proxy_tunneled_request_as_eap = yes

		#
		#  The inner tunneled request can be sent
		#  through a virtual server constructed
		#  specifically for this purpose.
```

linktodoc=true

```
			#
			#  If this entry is commented out, the inner
			#  tunneled request will be sent through
			#  the virtual server that processed the
			#  outer requests.
			#
			virtual_server = "inner-tunnel"

			# This option enables support for MS-SoH
			# see doc/SoH.txt for more info.
			# It is disabled by default.
			#
#			soh = yes

			#
			# The SoH reply will be turned into a request which
			# can be sent to a specific virtual server:
			#
#			soh_virtual_server = "soh-server"
		}

		#
		#  This takes no configuration.
		#
		#  Note that it is the EAP MS-CHAPv2 sub-module, not
		#  the main 'mschap' module.
		#
		#  Note also that in order for this sub-module to work,
		#  the main 'mschap' module MUST ALSO be configured.
		#
		#  This module is the *Microsoft* implementation of MS-CHAPv2
		#  in EAP.  There is another (incompatible) implementation
		#  of MS-CHAPv2 in EAP by Cisco, which FreeRADIUS does not
		#  currently support.
		#
		mschapv2 {
			#  Prior to version 2.1.11, the module never
			#  sent the MS-CHAP-Error message to the
			#  client.  This worked, but it had issues
			#  when the cached password was wrong.  The
			#  server *should* send "E=691 R=0" to the
			#  client, which tells it to prompt the user
			#  for a new password.
			#
			#  The default is to behave as in 2.1.10 and
			#  earlier, which is known to work.  If you
			#  set "send_error = yes", then the error
			#  message will be sent back to the client.
			#  This *may* help some clients work better,
			#  but *may* also cause other clients to stop
			#  working.
			#
#			send_error = no
		}
	}
```

7.7 Clients Configuration

```
# -*- text -*-
##
## clients.conf -- client configuration directives
##
##      $Id$

#######################################################################
#
#  Define RADIUS clients (usually a NAS, Access Point, etc.).

#
#  Defines a RADIUS client.
#
#  '127.0.0.1' is another name for 'localhost'.  It is enabled by default,
#  to allow testing of the server after an initial installation.  If you
#  are not going to be permitting RADIUS queries from localhost, we suggest
#  that you delete, or comment out, this entry.
#
#

#
#  Each client has a "short name" that is used to distinguish it from
#  other clients.
#
#  In version 1.x, the string after the word "client" was the IP
#  address of the client.  In 2.0, the IP address is configured via
#  the "ipaddr" or "ipv6addr" fields.  For compatibility, the 1.x
#  format is still accepted.
#
client localhost {
        #  Allowed values are:
        #       dotted quad (1.2.3.4)
        #       hostname    (radius.example.com)
        ipaddr = 127.0.0.1

        #  OR, you can use an IPv6 address, but not both
        #  at the same time.
#       ipv6addr = ::   # any.  ::1 == localhost

        #
        #  A note on DNS:  We STRONGLY recommend using IP addresses
        #  rather than host names.  Using host names means that the
        #  server will do DNS lookups when it starts, making it
        #  dependent on DNS.  i.e. If anything goes wrong with DNS,
        #  the server won't start!
        #
        #  The server also looks up the IP address from DNS once, and
        #  only once, when it starts.  If the DNS record is later
        #  updated, the server WILL NOT see that update.
        #

        #  One client definition can be applied to an entire network.
        #  e.g. 127/8 should be defined with "ipaddr = 127.0.0.0" and
        #  "netmask = 8"
        #
        #  If not specified, the default netmask is 32 (i.e. /32)
        #
        #  We do NOT recommend using anything other than 32.  There
        #  are usually other, better ways to achieve the same goal.
        #  Using netmasks of other than 32 can cause security issues.
        #
        #  You can specify overlapping networks (127/8 and 127.0/16)
        #  In that case, the smallest possible network will be used
        #  as the "best match" for the client.
```

linktodoc=true

```
        #
        #  Clients can also be defined dynamically at run time, based
        #  on any criteria.  e.g. SQL lookups, keying off of NAS-Identifier,
        #  etc.
        #  See raddb/sites-available/dynamic-clients for details.
        #

#       netmask = 32

        #
        #  The shared secret use to "encrypt" and "sign" packets between
        #  the NAS and FreeRADIUS.  You MUST change this secret from the
        #  default, otherwise it's not a secret any more!
        #
        #  The secret can be any string, up to 8k characters in length.
        #
        #  Control codes can be entered vi octal encoding,
        #       e.g. "\101\102" == "AB"
        #  Quotation marks can be entered by escaping them,
        #       e.g. "foo\"bar"
        #
        #  A note on security:  The security of the RADIUS protocol
        #  depends COMPLETELY on this secret!  We recommend using a
        #  shared secret that is composed of:
        #
        #       upper case letters
        #       lower case letters
        #       numbers
        #
        #  And is at LEAST 8 characters long, preferably 16 characters in
        #  length.  The secret MUST be random, and should not be words,
        #  phrase, or anything else that is recognizable.
        #
        #  The default secret below is only for testing, and should
        #  not be used in any real environment.
        #
        secret          = radius123

        #
        #  Old-style clients do not send a Message-Authenticator
        #  in an Access-Request.  RFC 5080 suggests that all clients
        #  SHOULD include it in an Access-Request.  The configuration
        #  item below allows the server to require it.  If a client
        #  is required to include a Message-Authenticator and it does
        #  not, then the packet will be silently discarded.
        #
        #  allowed values: yes, no
        require_message_authenticator = no

        #
        #  The short name is used as an alias for the fully qualified
        #  domain name, or the IP address.
        #
        #  It is accepted for compatibility with 1.x, but it is no
        #  longer necessary in 2.0
        #
#       shortname       = localhost

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

        #
```

linktodoc=true

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

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

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

        #
        #  As of 2.0, clients can also be tied to a virtual server.
        #  This is done by setting the "virtual_server" configuration
        #  item, as in the example below.
        #
#       virtual_server = home1

        #
        #  A pointer to the "home_server_pool" OR a "home_server"
        #  section that contains the CoA configuration for this
        #  client.  For an example of a coa home server or pool,
        #  see raddb/sites-available/originate-coa
#       coa_server = coa
}

# IPv6 Client
#client ::1 {
#       secret          = testing123
#       shortname       = localhost
#}
#
# All IPv6 Site-local clients
#client fe80::/16 {
#       secret          = testing123
#       shortname       = localhost
#}

#client some.host.org {
#       secret          = testing123
#       shortname       = localhost
#}

#
```

linktodoc=true

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


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

#######################################################################
#
#  Per-socket client lists.  The configuration entries are exactly
#  the same as above, but they are nested inside of a section.
#
#  You can have as many per-socket client lists as you have "listen"
#  sections, or you can re-use a list among multiple "listen" sections.
#
#  Un-comment this section, and edit a "listen" section to add:
#  "clients = per_socket_clients".  That IP address/port combination
#  will then accept ONLY the clients listed in this section.
#
#clients per_socket_clients {
#       client 192.168.3.4 {
#               secret = testing123
#        }
#}
```