COMP3210 - Promiscous 802.11 packet capture

*

1st Elliot Alexander
Department of Electronics and Computer Science
University of Southampton
E.Alexander@soton.ac.uk

***Abstract*—This document serves as a demonstrator report for the 'Networking Exemplar' Coursework component of COMP3210. The exemplar demonstration involves the use of an Alfa Networks AWUS036ACH, a USB WiFi network adaptor capable of promiscuous packet capture on the 5GHz band. Assuming a known PSK (i.e. A WPA/2 pre-shared key or password), the exemplar makes use of this promiscuous functionality to decrypt packets in-air via the interception of an 802.11i four-way handshake. For the purposes of the demonstration, this handshake is forced via a spoofed deauthentication frame sent to a target client, initializing a reauthentication with the access point. Once captured, this handshake allows the promiscuous capture and decryption of all IP layer traffic between the access point and the client. In the case of the demonstrator, this traffic is focused on DNS queries, allowing a malicious user to promiscuously sniff all DNS queries made across a network without ever connecting to the network.**

I. Introduction

This project was conceived from a combination of known vulnerabilities in WPA2, with the potential to automate and harvest meaningful user data from these vulnerabilities in a manner widely used previously. Particularly, the project was born from an interest in the seemingly ill-exploited promiscuous abilities of some commonly available USB WiFi cards. Promiscuous mode is a driver level feature which allows the interface to return not only packets marked as destined for it's own MAC address, but all packets capture in-air, regardless of the network they exist within, or their target destination. This opens up possibilities for significant userdata capture with very little in the way of discernable evidence left behind - a scary proof of concept. Hence, the initial outlook for this project was as follows - can a system be built which which exploits problems with 802.11i, and allows the promiscuous decryption and capture of all IP layer packets of a certain type, crucially without ever needing to connect to a target network? If a system as such could be implemented, and theoretically placed in a public space, the potential for both building a picture of an individual users activities and wider data-mining on the activities of users is huge. Hence, a secondary objective of the project was to enable a system of sorting, storing or visualizing the captured data.

This report will briefly outline the steps for implementation of the project, as well as highlight the breakdown of work within the team, and how we worked together to solve significant implementation issues within the project.

II. Presumptions

The demonstrator relies on several presumptions, listed below:

- The pre-shared (PSK) of any target network is known - i.e. the access password for the target network is known. The exemplar does not need to connect to the network, only be aware of the networks PSK.
- The target network is operating on WPA2-Personal. WPA2-Enterprise uses uniquely negotiated keys on a per-client basis, and hence is not vulnerable to the same attacks on the construction of a pre-shared key.

It should also be noted that the exemplar requires the interception of a 'Four-way handshake', a validation of pre-shared keys and exchanging of encryption keys between the access point and client prior to the initialization of encrypted communication. This handshake can be forced by deauthenticating a client from the access point, a known vulnerability with multiple well-developed implementations (airmon-ng is used in the demonstrator). Future work for the project might involve including the ability to deauthenticate a target client upon request, potentially automatically deauthenticating all newly observed clients.

A. Focus on DNS

It was decided to focus on DNS packets for the project primarily because DNS is unencrypted at the application layer, unlike TLS for example. Breaking encryption at the next layer is an entire challenge in it's own right, and would be wholly unfeasible. Hence, DNS was chosen as the best option for demonstrating the potential information exposure about a targets internet use, while still remaining a true to life proof of concept.

III. Decrypting 802.11i

IEEE 802.11i was introduced in 2004, and defines WPA2 as a replacement to the temporary solution for the insecurity of WEP introduced with WPA. WPA implements much of the WPA2 standard, and hence is vulnerable to largely the same attacks. However, this project only focuses on WPA2, primarily due to the lack of widespread adoption of WPA2 over WPA.

As some might believe, packets in air over WiFi aren't inherently secure, and are vulnerable to being intercepted by unknown parties. Typically, this is prevented at a driver level, with the majority of consumer WiFi cards excluding the ability to capture packets promiscuously, however this is no defence. Packets in-air are encrypted with CCMP, an implementation of the CCM mode of AES, a widely used and well regarded encryption standard. This standard itself is not known to be vulnerable, however the key exchange is. As a connection is negotiated, four packets, known as 'EAPOL' packets, are transferred between the client and access point. It is presumed that both the client and access point are aware of the PSK, a string constructed from the SSID and Password of the wireless network. The four packets perform the following summarized functions:

1) Message 1 is from the authenticator to the client, establishing the 'ANonce' (Authenticator Nonce). This is randomly generated on the Authenticator.
2) The client then computes the 'SNonce', which is used in conjunction with the 'ANonce' to generate the Pairwise Temporal Key (PMK). The client then transmits the 'SNonce' back to the Authenticator, alongside a Message Integrity Check (MIC).
3) Once received, the Authenticator can also compute the PTK, having generated the ANonce and been informed of the SNonce. The MIC is used to confirm the integrity of the nonce. The Authenticator then returns to the client, instructing the client to install the PTK generated.
4) The client then returns an acknowledgement, and the keys are installed.

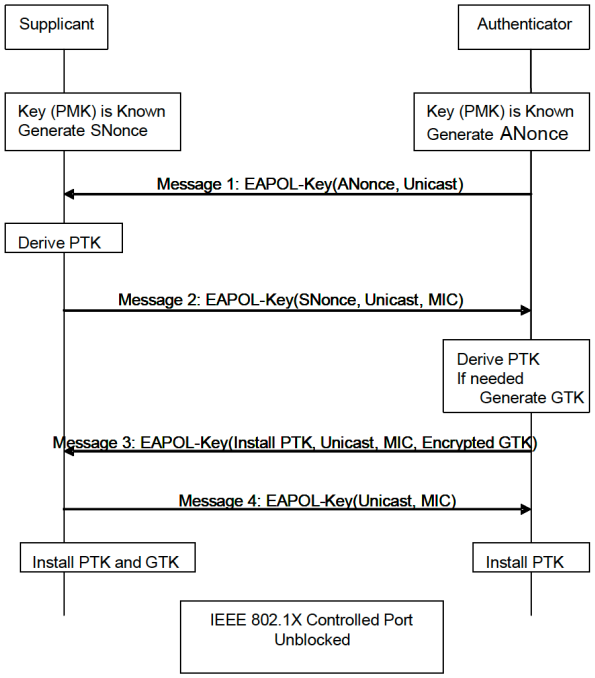

Fig. 1. EAPOL Four way handshake - reproduced from [1]

This process is shown in Figure 2. This handshake establishes the PMK between the client and access point. This PMK is then used to generate the PTK, via function known as the PRF-X, the details of which are omitted here. The PRF-X uses the previously computed nonce values and PMK to generate the final encryption key for the CCMP encryption. Once all four frames of this message have been captured, an external observer can calculate the PTK themselves, allowing all future traffic between the authenticator and client to be decrypted. The PRF-X is constructed from the ANonce, SNonce, the MAC Addresses (STA) of both the Client and Authenticator, as well as a known constant string - "Pairwise key expansion". This is defined in 802.11i [1].

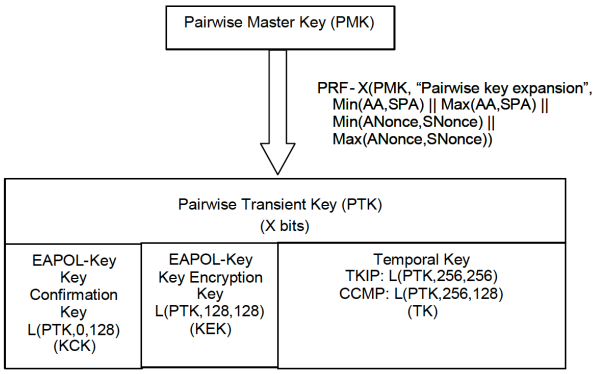

Fig. 2. PTK generation from the PMK, ANonce and SNonce. Reproduced from [1]

A. Decrypted Packets

Once decrypted, the payload of the captured 802.11i packet contains an IP frame. This can be encapsulated, and treated as any normal IP packet.

IV. Demonstrator Paramaters

A. Forcing EAPOL Frames

Forcing EAPOL frames is relatively easy, with existing tools such as airmon-ng being capable of capturing a beacon frame from the target network, and then spoofing a deauthentication frame from the client to the access point. These tools work with an adaptor in promiscous mode, and do not require connection to the network to deauthenticate. Once deauthenticated, the majority of clients will immediately attempt to reconnect (having not sent the deauthentication frames themselves), initiating a new EAPOL connection with minimal service disruption to the client. The airmon-ng deauthenication can be initiated with the following command:

```
aireplay-ng --deauth 0 -c <client STA> -a
<authenticator STA> <interface>
```

V. System Setup

The demonstrator setup has been tested on a 2018 Macbook Pro, running Kali rolling release 2019.1 inside VMware Fusion, with the AWUS036ACH passed through as a USB device from the host machine. The following series of commands are used to place the network interface into monitor mode before setting the channel (A fixed channel 2 was used for all testing). The network interface for the wireless adaptor is 'wlan0'.

```
airmon-ng start wlan0
sudo iwconfig wlano channel <channel>
```

A. Wireless Network

The wireless network used to test the demonstartor was running on a GL-iNet GL-AR150-POE on WPA2-PSM (CCMP). It should be noted that it is important that the USB WiFi adaptor and wireless network are on the same channel, and that the channel of the wireless network is fixed.

B. Wireless adaptor

The chosen wireless adaptor is the Alfa Networks AWUS036ACH, one of a few 5GHz wireless adaptors with promiscuous mode available in the United Kingdom. This adaptor was chosen based on reviews praising it's relative stability and driver support on Linux hosts. This said, during testing issues were discoverd where the adaptor needed to be restarted after the drivers crashed, especially when the adaptor was running in promiscous mode. Adaptors operating on the 2.4GHz spectrum are more widely available, however a 5GHz adaptor was chosen with an outlook to keep the finished proof of concept as close to a real world scenario as possible.

VI. Implementation

A. Wireless interface

The need to pull packets directly from the wireless network interface is generally provided through the use of some skew of pcap-lib, which offers low level access to network interfaces on a host system. This library is implemented in C, and is used in low-level code of many popular programs, including Wireshark. However, with ambitions to make the data captured by the client accessible over a network, Java was chosen with Pcap-4J implementing the majority of required features. Pcap-4j is a Java implementation of Pcap-lib, with support for encapsulating packets in an object oriented structure and directly accessing a network interface. Pcap-4j also has the useful functionality of dumping a packet objects directly to a .pcap file. This made debugging significantly easier, allowing us to inspect packet captures inside existing tools such as Wireshark [3].

The choice of Java offered some other benefits, such as far greater integration with message queues and other networked clients, however led to performance issues which required a heavy use of threading in order to ensure that all packets were processed in real time. It was essential to ensure that no significant operations were performed on the thread responsible for handling packet ingestion and initial processing, such as not to block this thread. With the adaptor in promiscuous mode, it is possible that the adaptor may be ingesting a large volume of packets per second, particuarly in a busy area with multiple networks active. Consider - a busy location with ten different active WiFi networks in the adaptors area of effect, each with 10 clients. the number of potential packets in-air at any one moment is huge, so these need to be processed quickly.

B. Backend Client

As previously mentioned, the backend client (i.e. the promiscuous adaptor running on a host machine) was specified in Java. In order to ensure that incoming packets were processed, broken down, open sessions identified and decrypted in real time, a heavy use of threading was required. The following briefly summarizes each of the thread types, as well as their potential number of instances.

- Main Thread - The main thread handles opening interfaces and message queue connections, before starting the packet listener thread. Once packet listener thread is started, Main is responsible for monitoring the synchronous Connection mapping objects to prune old connections on a timeout.
- Packet Listener Thread - This thread handles all interfacing with incoming packets. The PLT has two sub-roles:
  1) Identify new EAPOL packets and build connection objects for them. Once all four EAPOL packets are seen, start the generation of the PTK inside Decoder Thread.
  2) Identify incoming packets where an open connection exists. If all four EAPOL frames have been seen and the PTK calculated, start a new Decryption thread for the incoming packet.
- Decoder Thread - This thread is responsible for calculating the PTK once all four EAPOL frames have been seen.
- Decryption Thread - A new instance of this thread is started for the decryption of each packet identified to have an existing open connection. This thread is responsible for decrypting the packet, and then passing the packet to the Message Queue. There may be many hundreds of instances of Decryption thread active at any one time, however it's lifespan is relatively short lived.

The aim with the heavy amount of threading is to avoid time consuming or potentially blocking operations being executed on Packet Listener Thread. This will allow PLT to process all incoming packets in real time, without missing potentially vital packets (if a single EAPOL frame when opening a connection was missed while a data structure was being accessed, for example, the connection could never be decrypted.). All signficiant arithmetic and network operations are kept outside of the Packet Listener Thread, ensuring that potentially blocking operations are not preventing packets from being captured.

Included in Figure 3 is a sequence diagram for the threading structure of the program.

C. Frontend Client

The frontend client is a combination of a secondary reciprocatory Java application, responsible for handling communication between the database layer and the backend, and an InfluxDB / Grafana combination configured within the Dockerfile.

1) Docker: The entire frontend client runs in a series of Docker containers, with one container being responsible for building the frontend Java project (responsible for passing packets from the message queue to the frontend database) and another containing the database and Granada installation.

2) MQTT / Influx DB: The issue of removing data from the backend and passing it into a accessible form is handled in part by MQTT, a lightweight connectivity protocol designed for extremely lightweight publish / subscribe messaging [2]. MQTT connects directly to the backend, where 'Decryption Thread' pushes packets decrypted from 802.11 frames to one of two topics - 'root/packets' or 'root/packets/dns', depending on whether the packet type is a DNS packet or not. This allows the easier accessing of DNS queries specifically as part of a proof of concept for data visualization and acquisition.

Once a packet is added to the MQTT queue, it is detected by a separate listener thread in the frontend client, where the packet is processed into a query and added to InfluxDB, the backend database. From there, it is accessible from Grafana.

3) Grafana: The true frontend of the project is handled by Grafana, which connects to InfluxDB and offers a range of querying, visualization, analytics and analysis on incoming data. Grafana runs alongside Influx DB in docker, allowing the entire frontend to run in a single container (once built).

VII. Distribution of work

This project was completed in collaboration between the author (Elliot Alexander) and Dan Harris. Work breakdown was a combination of collaborative working, almost to an extent of pair programming through particularly difficult sections of the backend, to working entirely independently on separate areas of the project. My responsibilities in the project were entirely focused on the backend, involving handling the interfacing, threading, decryption and processing of packets from the network interface, in addition to the configuration of the interface and deauthentication of clients. Some ares of the backend, namely the understanding of the breakdown of EAPOL frames, were completed together, mainly due to the sheer complexity of the implementation of the decryption element of the project.

VIII. Results

Ultimately, the project was able to intercept a target users' DNS request, as well as their other IPv4 traffic in real time, without ever needing to connect to the network. As a proof of concept, the project has been extremely successful. The time delay between a target user making a DNS request on their WiFi connected device and the address the user is querying appearing in Grafana's visualization tools is a matter of seconds. The project has been successful in in demonstrating the potential for the collection of userdata on a wider scale from a target network, focusing on analysis and processing of a large pool of data over targeting a specific client.

When used in combination with a deauthentication tool (such as airmon-ng), the targeted attack of a particular client is relatively easy. The target client disconnects and reauthenticates with the network in a matter of seconds, leaving it unlikely that a user will observe the deauthentication, and the sniffing client itself leaves no trace on the targets network. Once EAPOL frames are captured, all the targets traffic is visible.

IX. Glossary

DNS - Domain Name System - an IP layer service for resolving domain names to 'records', containing the IP addresses of a service.

PSK - Pre Shared Key, the primary method of securing WPA and WPA2-Personal networks.

PMK - Pairwise Master Key - the negotiated key generated between the Authenticator and Client during an EAPOL transfer.

PTK - Pairwise Transient Key - generated from the

ANonce - Authenticator Nonce - a random nonce generated by the authenticator.

SNonce - Supplicant Nonce - a random nonce generated by the supplicant (client).

Supplicant - The authenticating client during an EAPOL exchange.

STA - An 802.11 supported station address (MAC address).

GTK - Group temporal key - The encryption key used for unicast traffic over the network. This is outside the scope of this proof of concept.

References

[1] 802.11i-2004 - IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements," 2004

[2] "MQTT," MQTT RSS. [Online]. Available: https://mqtt.org/.

[3] "PCAP-4J - A Java library for capturing, crafting, and sending packets.," Pcap4J. [Online]. Available: https://www.pcap4j.org/. [Accessed: 21-May-2019].

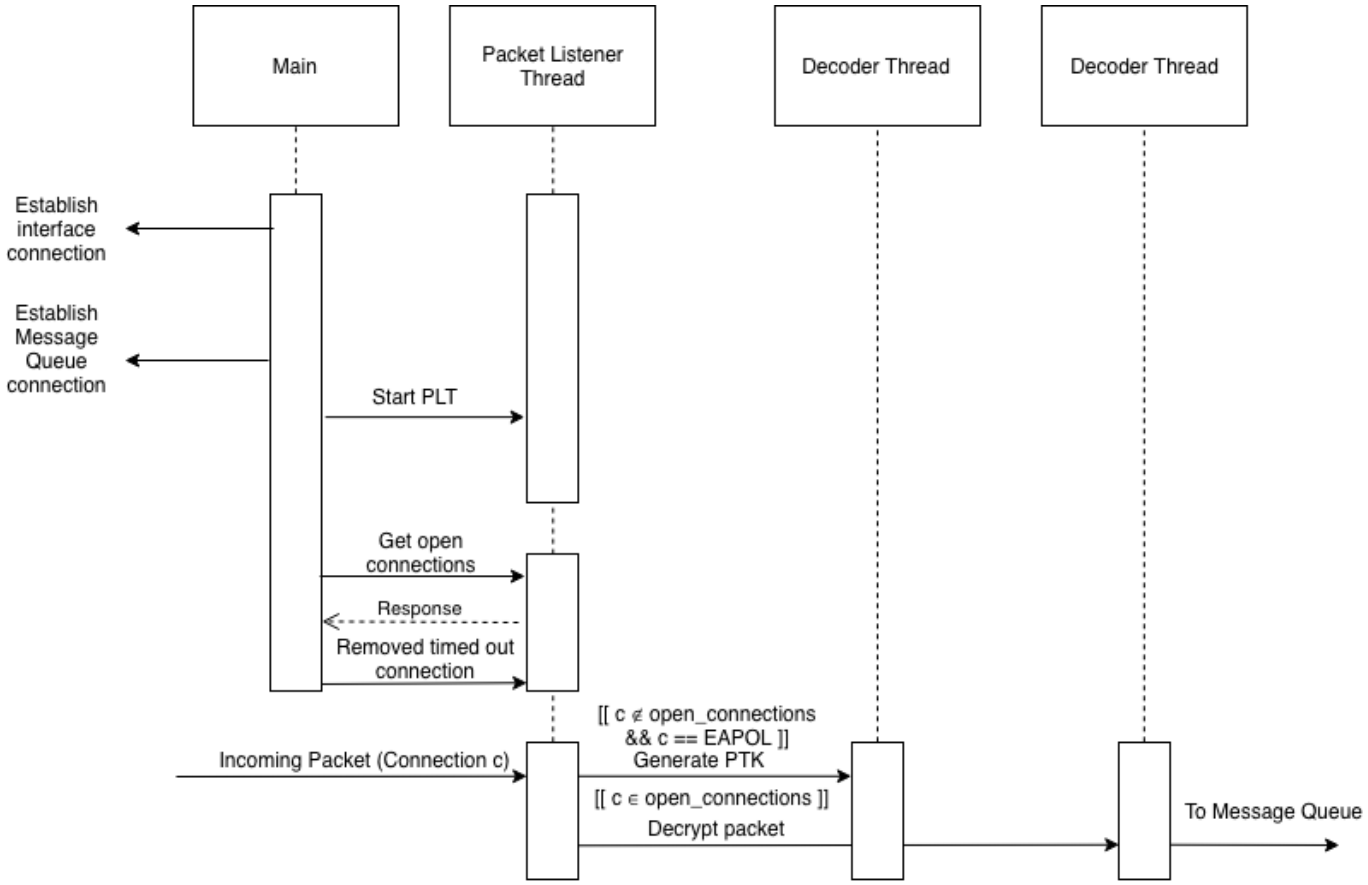

Fig. 3. Threading sequence diagram for the Backend Client