Offensive Security Methods Prof. Dr. Bernhard Jungk
Hochschule Albstadt-Sigmaringen SS 2022

Exercise 3 - Communication Interfaces and Hardware Attacks

Exercise 3.1 - UART Communication Analysis

This excersie is based on the Hack The Box challenge *Debugging Interface.* It is available at `https://app.hackthebox.com/challenges/debugging-interface` and it is necessary to download the hack the box challenge files from the mentioned URL.

General Hint

For the exercise, you have to analyse the recorded UART communication and find the Hack The Box flag using a logic analyzer software.

UART Details

The UART communication works roughly as follows:

1. If the UART communication is inactive, the voltage level is high.
2. The transmission always starts with a start bit (low voltage).
3. Then a number of data bits are transferred (usually 8 bit).
4. The data bits are followed by an optional parity bit.
5. The last bit is the stop bit (voltage level high), signaling the end of the transmission.

A complete UART transmission is depicted in Figure 1.

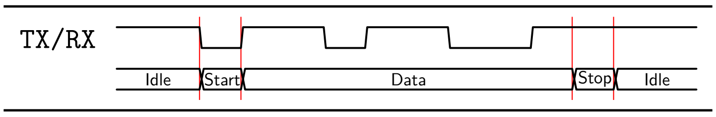

Figure 1: UART timing diagram without parity bits.

Furthermore, the timing of an individual bit (red marks in Figure 1) and the baud rate are related as follows:

$$\text{baudrate} = \frac{1}{t}$$

For example, for $t = 8.68 \mu s$, the baud rate is about 115.200. Note for the exercise, that even though only a few baud rates are typically used, the baud rate can be abitrary, usually only slightly limited by the hardware capabilities.

Tools

The exercise requires that you analyse a UART trace which was captured using a digital logical analyser and to find the HTB flag in the trace. The downloadable UART trace from hack the box can be analysed using the demo version of Saleae Logic 2, which can be downloaded here: `https://www.saleae.com/de/downloads/`.

Furthermore, you need to be able to decode the UART trace. This can be done manually or using a hex editor.

Exercise 3.2 - Binary Data Analysis

This excersie is based on the Hack The Box challenge *The Needle*. It is available at `https://app.hackthebox.com/challenges/the-needle`. It is necessary to download the hack the box challenge files from the URL and to connect to the hack the box instance to find the HTB flag.

General Hint

For this exercise, you have to analyse the binary file, which has to be downloaded from Hack The Box, using tools on Kali Linux such as `file` and `binwalk`. Furthermore, it is necessary to find out which service is running on one of the ports of the target machine to capture the HTB flag.

Exercise 3.3 - Password Cracking - Timing Side Channel Attack

For this exercise, you need to download the Python file `https://elearning.hs-albsig.de/goto.php?target=file_393230_download&client_id=HSALBSIG` from Ilias and develop a timing side channel attack to retrieve at least one randomly generated password of your choice to demonstrate that the feasibility of the attack.

General Hint

Note that you need to acquire timing measurements with the highest resolution possible on your system (see `https://docs.python.org/3/library/time.html`).

Exercise 3.4 - Password Cracking - Simple Power Analysis Attack

For this exercise, you need to download the power traces `https://elearning.hs-albsig.de/goto.php?target=file_393265_download&client_id=HSALBSIG` from Ilias and develop a simple power analysis attack to extract the password from the traces.

General Hint

The power traces contain a reference trace, which can be used to judge, if the guessed character is correct or the guessed character is incorrect. Incorrect power traces are very similar to the reference trace, while the correct power trace is obviously different. This difference can be made visible by plotting several traces for all guesses for a particular length of the password guesses. This feature can also be used to guess the correct length of the password from the given traces.

Format of power traces

The power traces are stored with `numpy.save()` and can be loaded with `numpy.load()`.

For internal reasons of numpy, `numpy.load()` will return a 0-dimensional array. The content of this 0-dimensional array can be accessed with the `item()` member function of the numpy array.

The internal structure of the traces is as follows:

1. The data is structured in several `dict` objects, where the key describes the number of characters in the password guesses. The length of the guess is described by the values of the dict.
2. The value of each entry contains a reference trace, and a further `dict` object, which contains the guessed password (as a string) and the measured trace (as a numpy array).

Exercise 3.5 - AES - Correlation Power Analysis Attack

For this exercise, you need to download the traces `https://elearning.hs-albsig.de/goto.php?target=file_393266_download&client_id=HSALBSIG` from Ilias and develop a correlation power analysis attack to extract the AES-128 key from the traces.

General Hint

For this attack, you have to perform the following steps:

1. Develop code to compute the intermediate AES-128 state before the last round, based on the ciphertext and a hypothetical last round key. This is equivalent to the computation of the first round of the AES-128 decryption.
2. Compute intermediate AES-128 values for all possible key hypotheses and ciphertexts and compute the Hamming weight of each intermediate values to simulate hypothetical power consumptions. This should be performed on individual bytes of the key and the corresponding ciphertext byte.
3. Compute Pearson's correlation coefficient between your hypothetical power consumption values and the power traces.
4. Identify the most likely hypothesis from the computed correlation.
5. Repeat your attack for the other AES sub round keys.
6. If you retrieved the complete last round key, compute the inverse key schedule for AES to retrieve the original AES key.

Basic correlation power analysis strategy

According to the lecture slides, the basic attack strategy works as follows:

1. Choose an approriate intermediate result of the algorithm to attack.
2. Choose an approriate model to compute hypothetical power consumption values.
3. Measure n traces with n different inputs.
4. Compute nk hypothetical intermediate values for k (key) hypotheses.
5. Compute Pearson's correlation coefficient between the traces and the hypotheses.

For the attack to be developed, the following guidelines are given:

- For the attack, the first round of AES's decryption mode can be used. The best intermediate value for attacking the last round of AES is the intermediate state after the first inverse SubBytes transformation.
- The appropriate model for this attack is to compute the Hamming weight of individual bytes of the intermediate state after the inverse SubBytes transformation.

Pearson's Correlation Coefficient

Pearson's correlation coefficient is defined as follows:

$$r(X,Y) = \frac{C(X,Y)}{\sqrt{\mathbb{V}(X)\mathbb{V}(Y)}}$$

With $C(X,Y)$ begin the co-variance of X and Y and $\mathbb{V}(X)$ ($\mathbb{V}(Y)$) the variance of X (Y).

You can use numpy's or scipy's implementation of Pearson's correlation coefficient or implement your own variant using numpy, which computes all correlation values in parallel without involving any loops in the Python code.

Identifying The Most Likely Key Hypothesis

You are given the ciphertext output of many encryptions of random plaintexts. To attack a single key byte, you have to compute 256 different hypothetical values for a particular byte of the ciphertext. This computation is repeated for n ciphertexts, resulting in a matrix of $256 \times n$ hypothetical intermediate values.

In addition to the n ciphertexts, n power traces are contained in the trace file. Each trace was acquired during the AES-128 computation to produce the ciphertext (in ECB mode). A trace consists of m sampling points over the time of the encryption. Hence, the power traces form a $n \times m$ array.

Since the same amount of samples mast be used in the computation of Pearson's correlation coefficient for both random variables, the logical conclusion is, that for each of the m sample points and for each of the 256 key hypotheses, you will get one correlation value. This will give you a $256 \times m$ matrix of correlation values, which allows you to find the most likely key hypothesis, by identifying the maximum absolute correlation (i.e., the sign of the correlation does not matter).

Computing The Inverse Key Schedule Of AES

Based on the basic description of the AES key schedule, it is obvious, that it can be inversed, if one of the round keys is extracted.

A C implementation of the inverse key schedule can be found at `https://github.com/kavka1983/key/blob/master/atmega8/aes/aes-min/aes-otfks-decrypt.c`.

Exercise 3.6 - ARP spoofing on the Metasploitable 2 VM

The goal of this exercise is to use an ARP spoofing attack to recover the login data to the DVWA installed on the Metasploitable 2 VM.

ARP Demo Setup

The setup consists of:

1. The Metasploitable 2 VM as target system.
2. Your local computer or any other Windows or Linux VM.

Metasploitable 2 VM

The VM for Metasploitable 2 can be downloaded at `https://sourceforge.net/projects/metasploitable/`.

Required Tools

Since the VMware Player network behavior with several hosts on the same computer in non-bridged network mode is actually a hub-like behavior, the Kali linux VM would see the complete traffic of the victim host without ARP spoofing.

Therefore, you have several options:

1. Run the VMs on different hosts connected to a bridged network, which uses a switch.
2. Run the VMs on the same host, connected to a bridged network, which uses a switch.
3. Run the VMs with VirtualBox instead of VMware Player.

For the ARP poisoning and sniffing attacks, there are several tools available, such as `ettercap`, `arpspoof` and `wireshark`.