Exercises for Security of Wireless Networks

edited by
Luka Mališa, Aanjhan Ranganathan, Joel Reardon and Nikos Karapanos

Course responsible:
Prof. Dr. Srđan Čapkun

Assistants on the course:
Luka Mališa
Nikos Karapanos
Hildur Olafsdottir

Email: sown@lists.inf.ethz.ch

Contents

Preface

The booklet is undergoing a major revision this semester as a substantial amount of new content is being introduced. As you, students, are our targeted audience, we kindly ask you to report to us any problems (content- or grammar-wise) you notice with this booklet.

Many people have contributed to this booklet during the years. Parts of this paper were originally written by Mario Čagalj, Jean-Pierre Hubaux and Imad Aad, under the title "Hands-on exercises: IEEE 802.11b standard"[14]. Several sections where later modified by Simon Thoustrup, Sigurd Hilbert Madsen and Frej Laursen Würtz. Later on, parts of these exercises were re-written/edited by Kasper Rasmussen and then edited by Davide Zanetti and Ghassan Karame.

We hope that you will enjoy the exercises as much as we enjoyed creating them!

Luka Mališa
Aanjhan Ranganathan
Joel Reardon

Chapter 1

Preparing Lab Reports

After each set of lab exercises, groups must prepare and submit a lab report. This report introduces the experiments, documents the findings, and provides answers to all the questions raised in the lab manual. The grade depends on the data, results, and analysis provided in the report and how the questions are answered, along with the quality of report's presentation.

1.1 Quality

Writing suitably scientific reports is an important skill, and as such some of the grade will depend on the writing quality. We expect polished, well-formatted, copy-edited, scientific lab reports that clearly introduce the nature of the experiments, the hypotheses, the results, and the analyses. In the next chapter we provide a simplified example report. The end of this chapter includes a LaTeX template report that may be helpful in formatting your reports.

We expect anyone yet unfamiliar with scientific writing to research this independently. Many resources are available online or at libraries to which ETH students have access. The book "The Elements of Style" by Strunk and White is very quick to read and has many useful tips on professional writing.

1.2 Authorship

While the primary author of each lab report may vary throughout the course, we expect every group member to read and understand the contents of every lab report. Before submitting the report, a draft is to be provided to all group members. They will then take the opportunity to edit, correct, and improve the report. The lead author should appear as the first author and should change so that each group member acts as the primary author of a suitable share of the reports.

For *each* lab report, students should **acknowledge** the resources used to answer the questions raised in the lab manual (books, friends, web sites, discussion with other teams, etc.). Although discussion between teams is not prohibited, each solution should reflect your own views on the problem.

1.3 Submitting

Each lab report must be submitted in printed form right at the beginning of the subsequent lab. This allows approximately two weeks to prepare the report. Missed deadlines will receive a grade of zero. In addition to submitting a printed report, you will also send us a pdf version of the document to our e-mail (`sown@lists.inf.ethz.ch`). The digital version of the report can be submitted any time before the beginning of the next lab.

When sending us the e-mail with your report use the following guidelines:

- Subject: Report <group number>_LabX
- Attachment name: `<group number>_LabX.pdf`

If you are group A1 and you are submitting the report for the first lab, then the submission e-mail would have the subject "Report A1_Lab1" while the attached pdf name would be `A1_Lab1.pdf`.

1.4 Template Report

```
\documentclass[12pt,a4paper]{article}

\usepackage{hyperref}

\title{Template for Security-of-Wireless-Networks Reports}
\author{Author Names}
\begin{document}
\maketitle

\begin{abstract}
\end{abstract}

\section{Introduction}

Example use of bibliography\cite{dd_manpage}

\section{Materials and Methods}

\section{Results}

\section{Analysis}

\bibliographystyle{plain}
\bibliography{bibliography}

\end{document}
```

1.5 Template Bibliography

Example contents of the file `bibliography.bib`.

```
@misc{dd_manpage,
      author = "John Doe",
```

```
      title = "dd(1) - Linux man page",
      month = "Septemper",
      year = "2012",
      howpublished = "\url{http://linux.die.net/man/1/dd}"
}
```

Chapter 2

Example Lab Report

2.1 Foreword

This chapter contains a toy example of a lab report for a single small experiment to give an understanding of the reports we expect to receive.

2.2 Introduction

Unix-based operating systems provide virtual character devices to generate data streams. Such streams are useful in a variety of contexts, such as testing throughput for other devices and filling the content of a dummy file. Among these devices are `/dev/zero` to generate a stream of binary zeros, `/dev/random` to generate a stream of hardware (or true) randomness, and `/dev/urandom` to generate a stream of random data without the same constraints of true randomness offered by the `/dev/random` device.

The simplest device, `/dev/zero`, can be implemented by simply clearing the data buffer for all read operations. More complicated are the random devices, which both rely on an entropy pool of random data collected by the measurement of random hardware sources. These sources vary, but a common approach is to sample the duration between hardware interrupts, as interrupts are unpredictable and human driven. The difference between */dev/random* and */dev/urandom* is that the former will always block until it has suitable random data, while the latter will simply reuse stale randomness when necessary instead of blocking, limiting its application to non-cryptographic settings.

In this work, we quantify the rate at which these devices can generate data. Given their mode of operation, we hypothesize that the rapidest will be `/dev/zero`, as no data needs to be read from another source to write the result. The next will be `/dev/urandom`, which needs to read from the entropy pool and possibly perform additional operations on old randomness to generate (cryptographically-unsuitable) pseudorandom data. Finally, `/dev/random` will be vastly slower than both these devices once the entropy pool is exhausted, because it will block until new entropy is added.

source	throughput while idle	throughput while typing
`/dev/zero`	390.1 ± 2.1 MB/s	389.3 ± 5.6 MB/s
`/dev/urandom`	10.0 ± 0.2 MB/s	10.0 ± 0.2 MB/s
`/dev/random`	15.2 ± 8.5 B/s	52.3 ± 18.3 B/s

Table 2.1: Read throughputs for character devices.

2.3 Materials and Methods

We perform our measurements on a commodity Thinkpad T410 laptop computer running Linux version 3.4.0-rc5. The throughput is measured using `dd`[1], reading from the appropriate input device and writing to `/dev/null`; this was done 16 bytes at a time. Each device was read for ten seconds and the average throughput is then reported. We performed the experiment 5 times and report the 95% confidence intervals under the assumption that the resulting throughputs are normally distributed. We execute two variants for each experiment: one where the computer is unused, and another where a human operator types a fixed paragraph at a normal typing speed so as to generate fresh entropy. Between experiments, the computer was used for a minute to generate fresh entropy for the entropy pool.

2.4 Results

The results from our experiment are presented in Table 1. The throughput measure for `/dev/random` suffers from the fact that it is blocking: longer durations of our experiment had no effect once the entropy pool is depleted.

2.5 Conclusion

Our experiment confirmed our hypothesis. The `/dev/zero` had the largest throughput, orders of magnitude larger than `/dev/urandom`. Effectively no throughput is offered by `/dev/random`: once the entropy pool is exhausted, then no data is provided until new entropy is added.

Typing while performing the experiment seems to feed the entropy pool. More data was provided by `/dev/random` when a user typed on the keyboard than when the computer was idle. Keyboard input offered no such benefits for the other, non-blocking devices.

Bibliography

[1] dd manpage
http://linux.die.net/man/1/dd

Chapter 3

Introduction

3.1 IEEE 802.11b standard

The scope of the IEEE 802.11 [5] standard is to provide specifications for wireless connectivity for fixed, portable and moving stations within a local area. It defines over-the-air protocols necessary to support networking in a local area. This standard provides MAC and physical layer functionality. The extension IEEE 802.11b (used in these exercises) gives accommodation of transmission rates of up to 11 Mbps and operates in the 2.4 GHz band. The IEEE 802.11 standard takes into account power management, bandwidth, security and addressing, since these are the significant differences from wireless to wired LANs. The MAC layer specification of the IEEE 802.11 standard provides radio channel access control functions, such as addressing, access coordination etc. The Distributed Coordination Function (DCF) is the primary access protocol for the automatic sharing of the wireless medium between stations and access points. This DCF uses a carrier sense multiple access/collision avoidance (CSMA/CA) protocol for sharing the wireless medium. As shown in Figure 3.1, the DCF delays frame transmissions right after the channel is sensed idle for DIFS (DCF InterFrame Spacing) time. It waits for an additional random time, backoff time, after which the frame is transmitted. The backoff time is bounded by the contention window size CW. This is applied to data frames in the basic scheme, and to RTS frames in the RTS/CTS scheme. The backoff time of each station is decreased as long as the channel is idle. When the channel is busy, backoff time is freezed. When backoff time reaches zero, the station transmits its frame. If the frame collides with another frame (or RTS), the sender times out waiting for the ACK (or the CTS) and computes a new random backoff time with a larger CW to retransmit the frame with lower collision probability. When a frame is successfully transmitted, the CW is reset to CW min. The network allocation vector (NAV) of all other stations is set to the frame duration field value in RTS/CTS and DATA headers.

Because of the possibility of partial network connectivity, wireless LAN protocols must take into account the hidden terminal problem (this occurs when a station is able to receive frames from two different stations but these two stations can not hear each other). To solve this, a virtual carrier sense mechanism through the exchange of control frames is used (cf. Figure 2). These are the Request to Send (RTS) and the Clear to Send (CTS) frames. The RTS and CTS frames contain a duration field that defines the period of time that the medium is to be reserved to transmit the actual data frame and the returning ACK frame. All stations within the reception range of either the originating station (which transmits the RTS) or the destination station (which transmits the CTS) shall learn of the medium reservation. Thus a station can be unable to receive from the originating station, yet still know about the impending use of the medium to transmit a data frame. The RTS/CTS control frames should not be used for short data frames, since they would add traffic. According to IEEE 802.11b standard, the use of RTS/CTS mechanism is

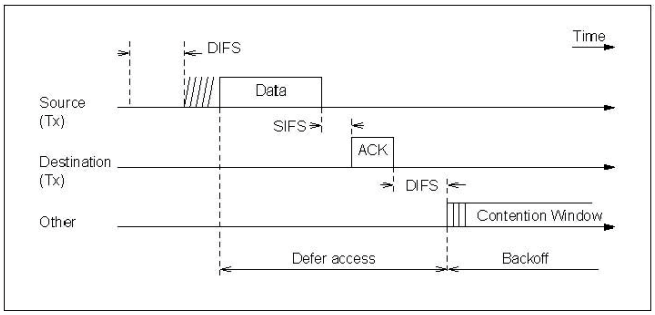


Figure 3.1: The distributed coordination function (DCF) of IEEE 802.11 operating in the basic mode.

optional.

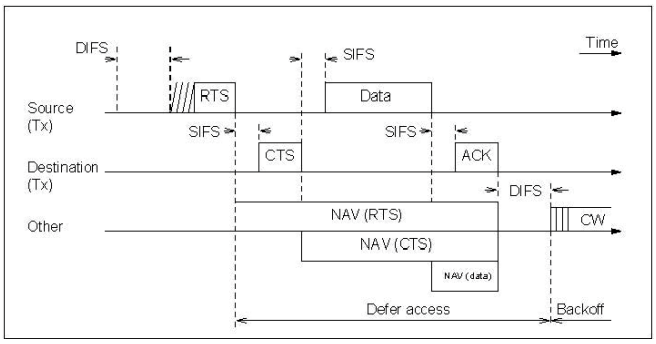


Figure 3.2: The distributed coordination function (DCF) of IEEE 802.11 operating in RTS/CTS mode.

IEEE 802.11 can be used in two different operating modes:

i Infrastructure mode (all communication goes through an access point (AP)).

ii Ad-Hoc mode (no AP is present; all communication is peer-to-peer).

3.2 The Wireless Medium

The wireless medium is a shared medium, where all traffic is available for everyone in the vicinity of the network. It is similar to a wired hub connected network, where every packet in the network is sent to all connected stations. But in a wireless network the user does not even need to be connected in order to eavesdrop on the traffic.

The wireless standard is described in IEEE 802.11[1], but this exercise will focus on the a/b/g subsets.

The 802.11b (11Mb/s) and 802.11g (54Mb/s) standards both operate in the 2.4 GHz band, which may cause intererence with other devices that share the frequency: microwave ovens, cordless telephones, bluetooth devices, etc. The 802.11a (54Mb/s) standard operates in the 5GHz band, thus does not suffer from interference from these products.

The wireless medium is as mentioned a shared medium, and in order to limit the interference from other wireless devices, the standards operates within fourteen different channels. One should

[1]Further information can be found at [4].

attempt to use channels which are currently not in use in the area. Nearby channels do still create interference, but as a rule of thumb, channel 1, 6, 11 should be non overlapping. However, this is not entirely true, and it depends on the transmission power of each station. But for the exercises you should at least attempt, if possible, to select channels which are two apart, i.e., 1, 3, 5, etc.

The shared property of the medium has the further unfortunate property that unauthorized persons in some scenarios can access, eavesdrop or manipulate with the traffic, as will be demonstrated in the following exercises. It is trivial for a spacially-proximate person to become a passive eavesdropper, and active man-in-the-middle-style attacks can be mounted without requiring physical access to network cables.

Chapter 4

Laboratory 1: Selfish Behaviour in Wireless Networks

This chapter will take you through the steps of configuring the wireless adaptors, testing throughput, and describing methods for selfish behaviour. The lab setup is illustrated in Figure 4.1.

Contents

4.1 Configuration of wireless client adaptors

This section makes extensive use of the performance testing tool `iperf`. If you are unfamiliar with it, begin by consulting the manual pages for `iperf`:

```
$ man iperf
```

Your goal in this exercise is to setup an operational **Wireless LAN** (WLAN) and sniff traffic. We use Proxim Orinoco 11a/b/g PCI adaptors as our network card.

TASK 1: **Basic settings**

1. Start a root terminal with password **sown**.

   ```
   $ su
   ```

2. Assign an IP address to the wireless adaptor by typing the following:

   ```
   # ifconfig wlan0 10.0.0.xy netmask 255.255.255.0 up
   ```

   where 10.0.0.xy is the IP address assigned to the machine. Use the following addressing scheme to assign IP addresses to machines. All the machines use the prefix 10.0.0 in their IP address. Then we set X and Y as follows:

   x - to the table number.

   y - to 1 in the case of the Router; to 2 in the case of the Station

   For example, if you sit at table number 3 your router should be 10.0.0.31 and your station 10.0.0.32.

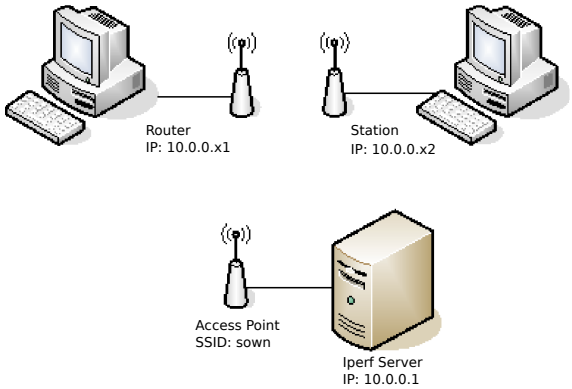


Figure 4.1: Lab overview

3. Verify that the IP was set correctly.

```
# ifconfig wlan0
```

TASK 2: **Infrastructure mode**

This mode is used to have the wireless interface associate with an **Access Point**. As an Access Point is usually connected to a wired network, this provides for access to the LAN. Note that in **Infrastructure Mode**, an adaptor scans frequency channels to find an Access Point; we do not have to set the channel manually.

1. The lab instructor will start an access point and an iperf server.
2. Start a root terminal.
3. Configure the interface with the following properties using the **iwconfig** tool:

   SSID - (*essid*) - Service Set ID (SSID) is a unique identifier that client devices use to associate with either the AP or the other client. This value **MUST** match the SSID of an access point (AP) with which we want to communicate. In our case, the SSID is *sown*. Do not forget that SSIDs are **case sensitive**.

   Network Type - (*mode*) - we want Infrastructure (*managed*).

   802.11-version - (*rate*) - we use the 802.11b (*11M*),

   ```
   # iwconfig wlan0 essid sown mode managed rate 11M
   ```

4. Try to ping other machines in the room to make sure that it worked.
5. There is an iperf server at 10.0.0.1. Try to ping the server.
6. Start the **wireshark** tool to monitor packets exchanged between your station and the server having IP address 10.0.0.1.

   ```
   # wireshark &
   ```

   To start a capture, click *Capture* - *Options* and make sure the *wlan0* interface is selected. Ensure that *Capture packets in promiscuous mode* is *not* enabled. You may want to **update the list of packets in real time** and use **automatic scrolling**.

   To generate traffic that wireshark can capture, ping the server.

7. From the output produced by the wireshark tool, retrieve the MAC address corresponding to the station with IP address 10.0.0.1. Verify that the other communicating party's MAC address is indeed yours (you can find your MAC address with `ifconfig wlan0`). MAC

addresses are used to uniquely identify hardware on the data link layer (in the OSI model) of the network.

8. Wireshark has shown you your messages, however wireless messages can be seen by anyone. To do this, we must put the network card in monitor mode and surveil all the traffic in promiscuous mode. **Restart wireshark completely** and then switch modes:

```
# ifconfig wlan0 down

# iwconfig wlan0 mode monitor channel 3

# ifconfig wlan0 up
```

9. Now run wireshark and start a new packet capture session. This time try with **Capture packets in promiscuous mode** enabled. Observe differences between the promiscuous mode and non-promiscuous mode.

10. **Close wireshark** and return your network card back to managed mode.

As an appendix to your lab report, ensure that the following questions are answered. (Question-answer format is acceptable; it does not need to be organized as a report.) The manual pages for `iwconfig` may be useful: `man iwconfig`. Read through it to be able to answer these questions before proceeding to the next exercise.

- What are all the different modes that iwconfig offers?
- What does each of these modes do?
- Why did we set the channel in monitor mode?
- What is the difference when capturing in promiscuous mode and non-promiscuous mode?

4.2 How much do we get out of 11 Mbps?

IEEE 802.11b adaptors operate at the maximum data rate of 11 Mbps. Supported Data Rates for the Proxim adaptor including the following: **1Mbps, 2 Mbps, 5.5 Mbps, and 11 Mbps**. The data rate of 11 Mbps sounds fairly good (∼1.3 MBps). Note, however, that this is the 802.11b raw data rate in perfect conditions at the physical level of the network. (Recall the OSI network stack: physical/link/network/transport/application. Each of these adds its own overhead and traffic.) In this exercise, you will perform an experiment to measure the observed data rate at the network layer. You will connect to the iperf server in the lab and observe the throughput. (The tool `iperf` is used to check network performance; if you are unfamiliar with it or its parameters, start by reading the corresponding man pages.) Measurements will be performed in a real-life scenario (i.e., several stations competing for the available bandwidth).

In the following tasks you will use the program **iptraf**, which monitors IP network statistics (e.g., data rates). To start this tool, open a new terminal and type **iptraf** at the command line. Then select **"Detailed interface statistics"** and choose the wlan0 interface.

TASK 1: **Wireless iperf server**

In this exercise, an iperf server is running on the AP machine (10.0.0.1). Do the following on *both* machines of your table:

1. Configure the machine to work in infrastructure mode. Using **iwconfig** and **ifconfig**, associate with the SSID **sown**. (Make sure you are in managed mode.)

2. Set up an iperf session between your machines and the server (10.0.0.1), by typing:

```
# iperf -c 10.0.0.1 -i 2 -t 1000 -u -b 100000000
```

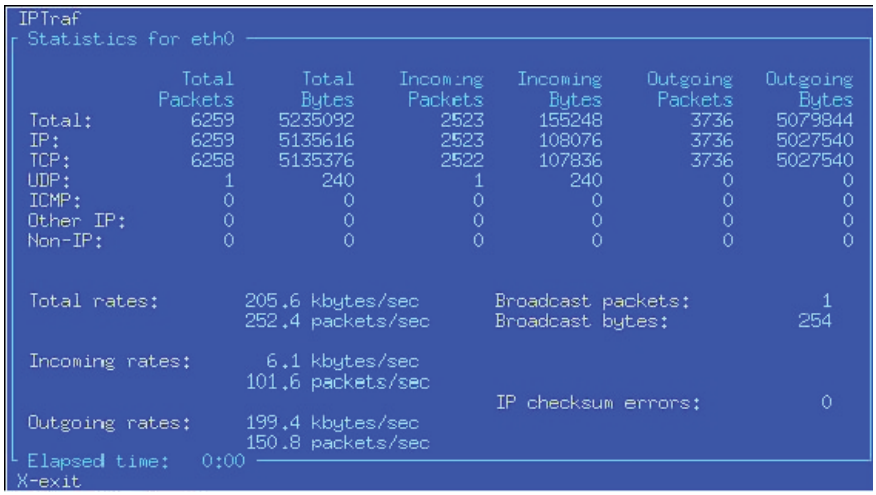


Figure 4.2: Iptraf screen

3. Observe the data rate with **iptraf**. Collect results on the observed throughput.
4. Wait until the measurements stabilize. Observe the distribution of bandwidth over the machines.

TASK 2: **Different Channels** In the previous exercise, all machines competed for bandwidth on a single channel. Now, we divide the class into 3 groups of 4 machines (i.e., two teams pair together). The teams will work together to complete the exercise. Each group will use a unique channel for communication in this exercise. One member of each team (e.g., the station of the lower group number) acts as the wireless base station, which we call the **AP machine**. The others will be clients in managed mode. We use the tool `hostapd` to act as a wireless access point. There is already a configuration file for this exercise called `hostapd.basic.conf`, which can be found in the `lab_material` subdirectory in the user's home directory: e.g., `/home/router7/lab_material/hostapd.basic.conf` – if there were a machine called **router7**. Lab material will always be found in this directory. It should provide the correct communication channel for this exercise, but confirm it by reading the contents of the file first.

1. In this exercise, only three computers will compete for bandwidth. The remaining machines in the lab are separated into different channels. We will measure the resulting throughput. **Formulate a hypothesis** on the resulting throughput as compared to the previous exercise.
2. The computer that acts as the host should first stop all programs using the network (iptraf, wireshark, etc.) and start the access point:

   ```
   # hostapd -d hostapd.basic.conf
   ```

3. Other computers should then select the channel and essid provided by the hostapd configuration on the machine running the access point:

   ```
   # iwconfig wlan0 essid <essid> channel <channel>
   ```

4. All computers should then set their IP address to communicate:

   ```
   # ifconfig wlan0 10.0.0.XY netmask 255.255.255.0 up
   ```

5. The AP machine should run an iperf server

   ```
   # iperf -s -u
   ```

6. The other machines now connect to the running iperf server using the *group's* AP machine's IP address:

   ```
   # iperf -c <AP's IP> -i 2 -t 1000 -u -b 10000000 -d
   ```

   The extra command `-d` tests both upload and download throughputs. Be sure to use it for all the experiments in your group.

7. Wait until the throughput stabilizes and observe the result. Compare it with your hypothesis. Observe how it compares with other clients on your channel. Compare bandwidth with other groups. Are the data transfer rates similar?

8. Try moving around the wireless antennas closer and further from the group's AP and observe the results. Observe the fairness properties of the medium.

9. Stop and start the iperf server.

The next two tasks will look at how to interfere with the throughput of neighbouring machines.

TASK 3: **Rate Changing** In this task, some machines will change their data transfer rate and observe the effect this has on all the machines in their group. While the transfers from the previous task continue, one member of each group will change their rate from 11M to 1M: decreasing their data transfer rate to about 10% of its previous value. As always as a group, *formulate a hypothesis* before beginning on the result that decreasing the data rate will have on the throughput both the lowered-rate machine and the non-lowered-rate machines.

1. Ensure that your group has an iperf server and clients running with a stable, shared bandwidth.

2. Select one non-AP member of your group to decrease their data rate. They should then run the following command:

   ```
   # iwconfig wlan0 rate 1M
   ```

3. Wait until the throughput stabilizes. Observe the result and contrast it to your hypothesis.

4. Now, on the remaining clients, decrease the data rate to 1M. Wait until the throughput stabilizes and observe the results.

5. Return the rate-reduced machines to 11M before continuing with the next exercise.

   ```
   # iwconfig wlan0 rate 11M
   ```

6. Stop and start the iperf server.

TASK 4: **Selfish Backoff** The wireless medium, being shared and relatively uncoordinated, has techniques to detect and avoid network congestion, thus improving bandwidth utilization and fairness for all participants. By simply not adhering to the techniques, one can unilaterally act against the best social good for private gain.

In this task, we will experiment with using different backoff strategy; backoff refers to the time one waits without sending data if one observes congestion on the wireless medium. We've prepared a modified wireless driver that sets the backoff timer to zero, effectively performing no backoff.

As always, read through all the steps of the exercise and *as a group formulate a hypothesis* on the results before beginning. We will test the results for one machine with the modified driver and all machines with the modified driver. **Write down your hypotheses** on the resulting throughput for all machines for all experiments. We assume that the setup continues from the previous task: one AP machine running both hostapd and an iperf server along with 3 clients associated with the AP running iperf clients.

1. Stop all iperf clients and restart the iperf server running on the AP.
2. Select one non-AP member of your group to change their wireless driver. In addition to stopping the iperf client, they should stop any other usage of the wireless card, e.g., iptraf or wireshark) and take down their wireless network:

   ```
   # ifconfig wlan0 down
   ```

3. To swap drivers, go to the `lab_material/modified_driver` subdirectory of the user's home directory and run the following script:

   ```
   # sh driver-modified.sh
   ```

4. If swapping the driver produced error-like output, inform a lab assistant!
5. Swapping drivers makes resetting the parameters for both `iwconfig` and `ifconfig` necessary. Do this as before to connect to the group's AP, set an IP, and bring back up the wireless card. (The non-modified machines do not need to do this.)
6. Connect to the group's iperf server.
7. Wait until the throughput stabilizes. Observe the result for the modified machine and the other machines. Contrast it to your hypothesis. Check if it is still fair.
8. Move the antennas closer and further from your group's AP. Observe if this has any effect on the modified AP or the non-modified AP.
9. Now, its not just one person who can cheat. Make the other clients use the modified driver in the same way as above. Make sure that the AP machine always uses the normal driver.
10. Observe the results when all modified machines are using the modified driver.
11. Move the various cheating clients closer and further from the AP. Observe the results. Compare with moving the antennas when the wireless cards were not cheating.
12. If you want to return to the original driver to compare moving around the antenna, you can run the `driver-vanilla.sh` script.

In your lab report, ensure that the following questions are answered, appropriately divided into introduction-like, theory-like, experiment-like, and analysis-like sections.

- What were your hypotheses for each task?
- What was the reasoning behind these hypotheses?
- How does the data rate compare to the actual throughput?
- How is the available bandwidth shared when all the computers connected to the same AP?
- How does the throughput change when computers are spread to different channels?
- What happens to the throughputs when one person reduces their data rate?
- What is backoff and what does it do?
- What happens when one machine ignores backoff?
- What happens when all machines ignore backoff?
- Why does backoff effect throughput as it did?
- How do the results compare to your hypothesis?
- If the results differed from expectation, explain and analyze why this was the case. Otherwise, the explanation of the reasoning is sufficient and claim instead that the experiment gave evidence towards your hypothesis.

Chapter 5

Laboratory 2: Unauthorized Access in Wireless Networks

This chapter explores the weaknesses in various wireless security mechanisms.

Contents

5.1 Hacking MAC filtering

MAC is an abbreviation for Media Access Control[7] and the MAC address is a unique identifier for the network card. The MAC address identifies the network card at the link layer and is used to map an upper layer address to a specific interface. Each network interface is provided with a unique MAC address from the manufacturer. The MAC address is a series of twelve hexadecimals, e.g. `00:08:74:4C:7F:1D`. The address `FF:FF:FF:FF:FF:FF` is the broadcast address. A multicast address is identified by a `01` in the first octet.

MAC filtering is the security process of granting or denying service (i.e., wireless access) by discrimination on the client's reported MAC address. MAC filtering is done by creating a list of authorized MAC addresses (a white-list) on the AP. If the client's MAC address is not listed, then the traffic corresponding with that MAC address is simply dropped.

The problem with this kind of protection, however, is that the client is trusted to accurately report their MAC address. By inaccurately reporting this value (spoofing), the client can instead report a MAC address that is authorized for service (or alternatively, one that is not unauthorized for service). The client need only to tell the operating system to use a different MAC address.

The exercise requires three computers: An access point AP, a client C and an attacker A. Coordinate your activities with the other group at your table.

Start out by setting up an access point on AP. We use the program `hostapd` to turn computer hosts into APs. This tool has a wealth of configuration options and we will continue to use it throughout this section. You may want to consult the man pages for hostapd to familiarize yourself with its usage. We only need `hostapd` on the machine that is to run as an access point. We have already written configuration files for each table and experiment that we will perform, but we expect you to read the configuration file before starting `hostapd`.

TASK 1: **Configuration of the Access Point**

1. Bring down the interface.

   ```
   # ifconfig wlan0 down
   ```

2. Configure the AP. Read the configuration file `hostapd.basic.conf` which is located in your user's `lab_material` directory. The file should specify which channel to use for communication along with a distinct essid.

3. Start `hostapd`

   ```
   # hostapd -d hostapd.basic.conf
   ```

4. In another root terminal, set the IP address for the network interface.

   ```
   # ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up
   ```

   where **xy** is chosen as in chapter 4.

TASK 2: **Configuration of the Client**

1. Bring down the interface.

   ```
   # ifconfig wlan0 down
   ```

2. Connect to AP in Managed mode.

   ```
   # iwconfig wlan0 mode managed essid <essid> rate 11M channel <channel>
   ```

   where <essid> and <channel> are specified in `hostapd.basic.conf`.

3. Configure the network address of the interface.

   ```
   # ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up
   ```

   where **xy** is chosen as in chapter 4.

TASK 3: **Adding MAC filtering**

Now we will add MAC address filtering to the running features of our AP. There are two kinds of filters we can use: white lists and black lists. A white list is a list of MAC addresses that have been specifically granted access. A black list is a list of MAC address that have been specifically revoked access. In our lab, we will use a white list of approved MAC addresses.

1. Obtain the MAC address of C (e.g., by running the command on C).

   ```
   # ifconfig wlan0
   ```

2. Add the MAC address to list of approved address. The location of the white list file can be found in `hostapd.mac.conf`.

3. Stop the previous hostapd service on AP and start it again using the configuration file that includes MAC address filtering.

   ```
   # hostapd -d hostapd.mac.conf
   ```

4. Reconnect C to AP.

   ```
   # ifconfig wlan0 down

   # iwconfig wlan0 mode managed essid <essid> rate 11M

   # ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up
   ```

5. Check that you are still able to connect and ping the AP.

TASK 4: **Hacking the filter**

The white list ensures that only devices with MAC addresses on the list will be granted access by the server. However, the expectation of honest self-reporting is not very reasonable when dealing with unscrupulous adversaries.

1. Try to connect A to AP.

   ```
   # ifconfig wlan0 down

   # iwconfig wlan0 mode managed essid <essid> rate 11M

   # ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up
   ```

   where **xy** is chosen as in chapter 4.

2. Put the card in **monitor mode** and start **wireshark** on A, and start monitoring the traffic.

   ```
   # ifconfig wlan0 down

   # iwconfig wlan0 mode monitor channel <channel>

   # ifconfig wlan0 up
   ```

   For this exercise, set <channel> to have the same value as the channel of the AP you created. In case you do not know the channel or the MAC of the AP, you can obtain it (when the device is in **managed mode** by typing *iwlist wlan0 scanning* and looking for the resulting information for the given AP SSID.

3. Start wireshark and try to find the MAC address of some client (i.e., C) that has successfully authenticated with the AP. (Yes, this is going to be the same MAC address as the white list, but don't ask your partner what it is—be the attacker.) A filter on the AP MAC address (*wlan.da == <AP mac>*) may help. If you cannot find the MAC address of C, try to ping AP from C. When you have successfully observed the MAC address of C, write it down for later use. Now, put A's interface back in managed mode and disconnect C.

   The reason for specifying the channel for the monitor mode machine is that, when in monitor mode, the interface does not associate with any AP. In order to capture the traffic, you need to have the interface tuned to the right frequency (channel).

4. Let's now test the security of MAC filtering by changing (spoofing) the MAC address of A, thus shamelessly lying to the harmless router about our MAC address. We will make use of the software tool *macchanger* to change A's MAC address. Macchanger is an open source Linux utility for viewing/manipulating the MAC addresses of network interfaces.

   ```
   # ifconfig wlan0 down

   # macchanger wlan0 -m <mac address of C>
   ```

5. Return the card back into managed mode with the following command

   ```
   # iwconfig wlan0 mode managed
   ```

   Re-associate A with AP.

   ```
   # iwconfig wlan0 essid <essid> mode managed rate 11M

   # ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up
   ```

   Check that you are able to connect to AP.

In your lab report, be sure to answer the following questions, divided appropriately into introduction, results, and analysis section.

- How does MAC filtering work?
- Which computers, and during which situations, are able to connect and not connect to the AP?

- Is MAC filtering a good way of securing networks against unauthorized access?
- Why is MAC filtering still widely used, despite its obvious vulnerabilities? (E.g., the ETH uses MAC filtering for wired connections.)
- Can you think of situations where black listing or white listing would provide reasonable security?

Stop MAC filtering on *AP* by stopping `hostapd`. The dirty machine who lied shamelessly about its MAC address should be rebooted as punishment.

5.2 Cracking the WEP encryption

WEP is an abbreviation for Wired Equivalent Privacy and it was the first attempt to provide the wireless medium the same confidentiality as a traditional wired network. Unfortunately, it was pessimally designed, and has many different vulnerabilities that has rendered it completely insecure.

It works by encrypting every package between the AP and its clients. All the packets for the same network are encrypted with the same shared key; consequently it provides no protection to individual users against other authorized users in same the network.

WEP uses a user-chosen secret of various sizes (40-bits is typical) and it concatenates a 24-bit initialization vector (IV) to each packet. The result of a 40-bit secret is then a 64-bit encryption key that is then used to seed the RC4 stream cipher. Of course, 24 bits of the key corresponding to the initialization vector are not secret. The IV is changed with each transmitted packet, either by incrementing a counter or randomly selecting a new IV. The reason to include an IV is that, given the same key, stream ciphers will always produce the same sequence of random bits to XOR with the message. Reuse of the same key for different plain-text messages leads to the two-time pad problem, the use of random IVs as part of the key avoids this problem.

WEP is flawed in many ways, one of which is that a 24-bit IV is too short to prevent eventual repetition. The 24-bit allows roughly 17 million possibilities; according to the birthday paradox[1] it is likely that there will be IV reuse in the order of thousands of packets[1]. Whenever an IV is retransmitted, it is possible to retrieve the plain text through statistical analysis, and when the plain text is retrieved, it is trivial to retrieve the key, thus gaining access to the network [9].

This part of the exercise will take a look at the security in a WEP-protected network. For this exercise three machines are required: Two peers *P1* and *P2* and an attacker *A*.

We make use of the following two tools from the suite of tools called **Aircrack-ng**:

Airodump-ng for capturing network traffic.

Aircrack-ng for cracking the encryption through the captured traffic.

Let's proceed with the evaluation of WEP as a security mechanism for wireless networks.

TASK 1: **Setting up WEP encryption**

1. Put both *P1* and *P2* into ad-hoc mode and check they can communicate.

```
# ifconfig wlan0 down

# iwconfig wlan0 mode ad-hoc

# ifconfig wlan0 inet 10.0.0.xy netmask 255.255.255.0 up

# iwconfig wlan0 essid <essid> channel <channel>
```

[1] See [9] for details.

2. Add WEP encryption to *P1*.

```
# ifconfig wlan0 down
```

```
# iwconfig wlan0 key <key>
```

where <key> is the encryption key.

```
# ifconfig wlan0 up
```

The <key> parameter should be a 40-bit value expressed in hexidecimal (without the starting 0x). As an example, faced12345.

3. Add WEP encryption to *P2* in the same way, except use a different key. Try to communicate with the peer. Observe the results.

4. Now, performing the same proceedure, use the same key for *P2* as *P1*. Try to communicate between the peers. Observe the results.

TASK 2: **Collection of IVs**

1. Try to connect *A* to the ad-hoc network with the peers (but without setting any key). Observe the results.

2. Since we, as the attacker, do not know the key, we will try to find it. Put the attacker in monitor mode (in the correct channel) and start **wireshark** to capture traffic from the peers. (You may need to generate some traffic, e.g., ping requests.) You might want to add some filters like

```
wlan.sa == <source mac>
```

or

```
wlan.da == <destination mac>
```

Observe the appearance of the traffic.

3. Now, since we would like to gain access to the network, we can exploit the weakness in WEP by comparing IV reoccurrences. Let's start **airodump** to collect IVs.

```
# airodump-ng wlan0 -w <dump file> -c <channel>
```

where <dump file> is the name of the file to store the recorded traffic and <channel> is the channel of the ad-hoc network. The number at `# Data` corresponds to the number of IVs sent. If it is not generating data very quickly, try generating some traffic between the peers, e.g. by issuing a ping flood

```
# ping -f <ip>
```

While collecting data, write down the MAC address of the station. Leave the collection running, we can always use some more IVs. You may use this time to read the paper entitled *Weaknesses in the Key Scheduling Algorithm of RC4*[2]. When you have done that, you will know exactly why all this actually works.

TASK 4: **Cracking the encryption**

While still collecting IVs, you can test to see if you have enough by trying to find the encryption key on the current set. Use the following command to try to break WEP

```
# aircrack-ng -n 64 -z -b <ap mac> <dump file>.cap
```

where -n specifies the encryption size and -z to invokes the PTW WEP[13] cracking method.

[2]http://www.drizzle.com/∼aboba/IEEE/rc4_ksaproc.pdf

After you have retrieved the encryption key, use it as the key parameter in `iwconfig` on the attacker's machine and try to join the ad-hoc network. Observe the results.

When the exercise is successfully completed, turn of the encryption.

```
# iwconfig wlan0 key off
```

In your lab report, be sure to answer the following questions, divided appropriately into introduction, results, and analysis section.

- Was the attacker originally able to connect?
- How did the sniffed traffic look?
- Was the attacker able to join after discovering the key?
- How many packets did it take until the key was recovered?
- How long might it take to generate enough traffic to recover an IV for a home network? For an office network?

5.3 Breaking WPA2-Personal Passwords

WPA is somewhat more secure then WEP, and was developed in order to deal with the severe weaknesses in WEP. It used the same cryptographic scheme as WEP so as to be compatible with existing hardware. Later, WPA2 was developed without the goal of backwards compatibility. WPA2 is actually an umbrella term that covers a whole class of systems and protocols. What we refer to as WPA2 in this exercise is in fact WPA2-PSK (pre-shared key mode, also known as WPA2-Personal), which uses the standardized AES cryptosystem. WPA2-Personal is widely-deployed in personal networks and generally considered secure.

However, WPA2's authentication protocol is vulnerable to a dictionary attack: a class of attacks that permit an adversary to check the correctness of potential passwords in a very fast and offline (i.e., discreet) manner. (In fact, WPA suffers the exact same vulnerability.) During an on-line attack, the access point may throttle the rate of authentication attempts or disable access to the user after observing repeated failed password attempts. However, an offline dictionary attack can be done without attempt limits and is trivially distributed among any available computational resources (e.g., a botnet). Consequently, the security of WPA relies on the ratio between the adversary's computational power and the security (or unguessability) of the group passphrase.

For this part of the exercise we will take a look at the security in a WPA2 protected network. For this exercise three machines are required: An access point (AP), a client (C) and an attacker (A). In this section, we will use a network in infrastructure mode with an access point running `hostapd`.

TASK 1: **Setting up WPA2 encryption on AP**

1. Start by choosing a passphrase. It should not be something fancy, but rather an english word or frequently-used password. (Remember, this is a dictionary attack, so if it is not in the dictionary the attack won't work.)
2. We use the `hostapd` configuration file `hostapd.wpa2.conf`. Read the contents of the file and edit it to reflect your desire pre-shared key (PSK).
3. Start Hostapd on AP

   ```
   # hostapd -d hostapd.wpa2.conf
   ```

TASK 2: **Setting up WPA2 encryption on C**

1. In order to connect to a WPA2 encrypted AP, a program called **wpa_supplicant** is needed. Edit the config file (`wpa_supplicant.conf`, in your computer's *user*'s home directory) to contain your SSID and passphrase information.
2. Put C in Managed mode as described in 5.1.
3. Start wpa_supplicant on C.

   ```
   # wpa_supplicant -i wlan0 -c wpa_supplicant.conf -D nl80211
   ```

4. Bring up the client.

   ```
   # ifconfig wlan0 10.0.0.XY netmask 255.255.255.0 up
   ```

5. Try pinging AP to ensure that it works.

TASK 3: **Collecting encrypted data (on *A*)**

1. Try connecting A to AP without applying encryption. Observe the results. Start wireshark and capture the traffic from C (remember to actually generate some traffic on C and to put A in monitor mode). The data is encrypted and secured by AES, so its unlikely that much sense will be made from it. (If you are able to decrypt the data in your head, please inform a laboratory assistant for an extra credit assignment.)

   We perform the attack by determining the group password. To do this, we only need a tiny bit of information: a brief successful authentication handshake that initiates any legitimate connection. This alone is sufficient to begin our offline dictionary attack. As the attacker, start airodump to collect packets. (Be sure that you collect the entire packet).

   ```
   # airodump-ng wlan0 -w <dump file> -c <channel>
   ```

2. Now, in the wild, we would need to wait until a user authenticates to get the data that we need. The impatient active attacker may reduce this delay by performing a de-authentication attack, resulting in the client promptly re-authenticating and thus producing the required packets. To perform a de-authentication attack, use the following command on the attacking node

   ```
   # aireplay-ng -0 1 --ignore-negative-one -a <ap mac> wlan0
   ```

   Remember that `wpa_supplicant` must stay running.

TASK 4: **Cracking the encryption**

That's it. We've collected what we need in order to perform a dictionary attack. Stop the collection of packets and use **aircrack** with the following parameters.

```
# aircrack-ng -a 2 -b <ap mac> -w <word file> <dump file>
```

where <word file> is the name of the file containing a list of words. The file

```
/usr/share/dict/words
```

contains the list of words used for spell checking. The file

```
/usr/share/john/password.lst
```

contains a list of common passwords. If your password is in neither of these files (grep for it if you want to know), then you can try adding it to make the attack succeed. Of course, in practice, password lists can be very large indeed and generated algorithmically.

The tool `john` is designed for system administrators who wish to determine if their users are using weak passwords. (Hence its list of common passwords.) It also generates passwords algorithmically based on password lists and configurable rules (i.e., capitalizing, punctuation appending, etc.). Whenever you are generating a password

in the future, remember this attack and these kinds of tools. The time required for an adversary to find your password is determined exclusively by the rate at which it can check passwords and how far down the guessing list your password sits.

In your lab report, be sure to answer the following questions, divided appropriately into introduction, results, and analysis section.

- Was the attacker able to connect without applying encryption?
- How did the observed traffic look?
- Why is a handshake alone sufficient to break the password?
- How would you relate the security (or entropy, or unguessability) of a password with regards to a list of passwords?
- How fast was it to recover the password?
- How many passwords a minute could be checked with this computer?
- Were your password a number, how big would it need to be to prevent a day's worth of consecutive number guessing on the computer?

Chapter 6

Laboratory 3: Global Positioning System and DSSS

Contents

6.1 Global Positioning System

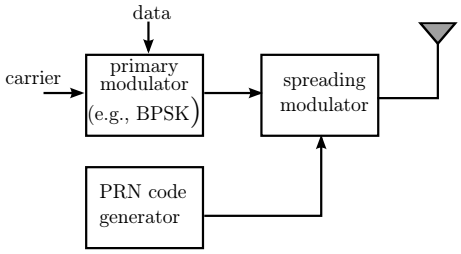
First part of the lab will be a demo of GPS spoofing on a commercial satellite simulator. The goal is to show you how realistic (and cheap) such attacks are.

6.2 Spread Spectrum Techniques

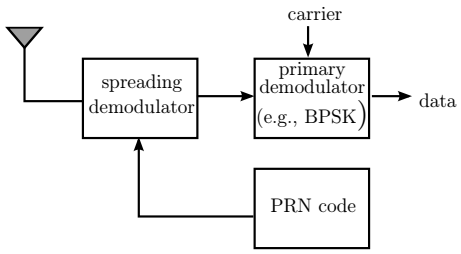
Most communication systems utilize a limited range of frequencies to transmit and receive information. For example, the AM and FM radio stations occupy less than few hundred kilo hertz of bandwidth and are broadly classified under "narrowband systems". Narrow band systems are easy to detect, eavesdrop and also jam, making it trivial to execute a denial of service attack. In contrast signals that utilize more bandwidth ("broadband systems") are less susceptible to external interference (unintentional and intentional). Spreading the signal energy over a large bandwidth reduces the effective signal strength, making it harder for an unauthorized receiver to detect and decode the signals. Additionally, these signals are more robust against channel multipath and fading effects.

There are three main ways in which the communication signal is spread over a large bandwidth.

1. Change frequency of transmission periodically (**F**requency **H**op **S**pread **S**pectrum) in a known (based on a keycode?) pattern.
2. Increase or decrease frequency continuously over time (chirping). For example, an increasing frequency signal can represent a '1' bit and '0' bit represented by a decreasing frequency signal.



(a) DSSS transmitter blocks.



(b) DSSS receiver blocks.

3. Combine data signal with a high bit rate chip or code signal (**D**irect **S**equence **S**pread **S**pectrum).

In this exercise, we will try to understand the working of direct sequence spread spectrum through `Matlab` simulations. You will need to download the `Matlab` code files from the course website. The code simulates (almost!) a typical DSSS transmitter and receiver system.

6.2.1 Direct Sequence Spread Spectrum (DSSS)

Due to its resilience against intended (adversarial jamming, eavesdropping etc.) and unintended (channel noise) interference, use of DSSS is common in both military and civilian communication systems.

DSSS Transmitter

Figure 6.1a shows a block diagram of a generic DSSS transmitter. There are two main signal components in a DSSS system: (i) The message signal $m(t)$ and (ii) Pseudo random code sequence ($code(t)$). The pseudo random code signal has a higher bit rate than that of the message signal. At the transmitter, in order to spread the message signal, it is multiplied with the code sequence. Due to the pseudo random nature of the coding signal, the frequency spectrum of the resulting output signal is spread evenly over a wider range of frequencies.

DSSS Receiver

At the receiver, there is a spreading demodulator which uses the same pseudo random code as that of the transmitter to recover the message signal. A generic block diagram of a DSSS receiver is shown in Figure 6.1b.

Running Matlab

We will use Matlab to simulate and understand the working of DSSS technique. To run `Matlab` use the command

```
$ matlab
```

After Matlab loads, you should be seeing a screen like the one shown in Figure 6.2.

Use the left pane to navigate to the folder where the `dsss.m` script is saved. Double click on the filename to open it in an editor (Figure 6.3). From the editor panel it is possible to run your simulations using the `Run` icon in the toolbar.

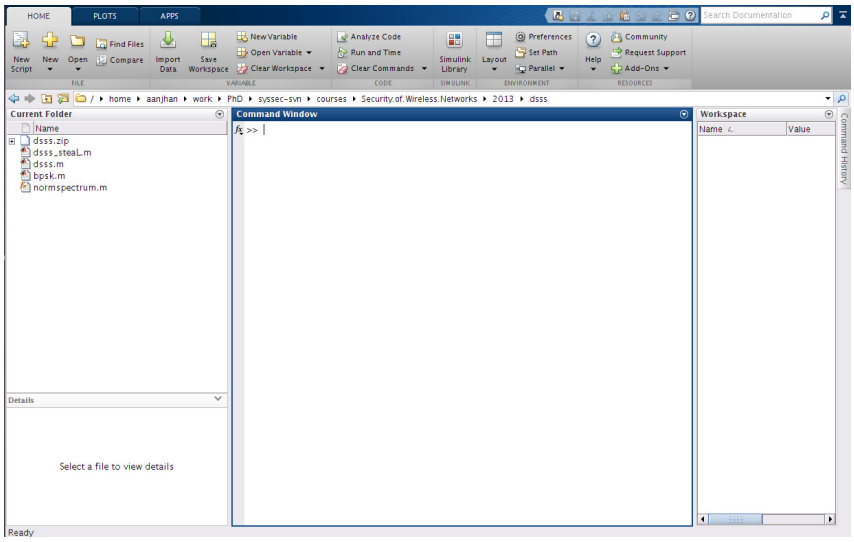


Figure 6.2: `Matlab` start screen

You can either run selectively portions of a `Matlab` code or execute the entire script. In order to selectively execute a portion of code, simply select the code segment you wish to simulate and press `F9`. For complete execution, from the editor window press `F5`. It is strongly recommended to go through the `Matlab` code sectionwise and complete the following tasks.

Tasks

- **Task 1:** Create your own data signal by assigning values to the `data` array as instructed in the code.
- **Task 2:** Run the code and compare the recovered data signal with the transmitted signal. Are they the same or is it just random noise? Correct the receiver code to recover the data signal.
- **Task 3:** Try to analyze the power spectral density for various lengths of the pseudorandom code and data signal.

Questions

- Briefly explain the DSSS system configuration you used to simulate in Matlab. For example, what is the length of the message signal, pseudo random code sequence? How many bits of the pseudo random code sequence are used to modulate one bit of the information signal? What is the processing gain?
- Why are broadband signaling schemes (e.g., DSSS) more difficult to jam? Would DSSS be effective against wideband jammers?
- What does it mean by DSSS signal is "below the noise"? What are the inherent security properties provided by a DSSS system? Can you think of any mechanism to detect the presence of a DSSS signal?
- Plot and comment on the power spectrum of the message, code and the spread DSSS signal. Briefly comment on why and how the narrow band message signal gets spread over a wider bandwidth when multiplied with the code signal.

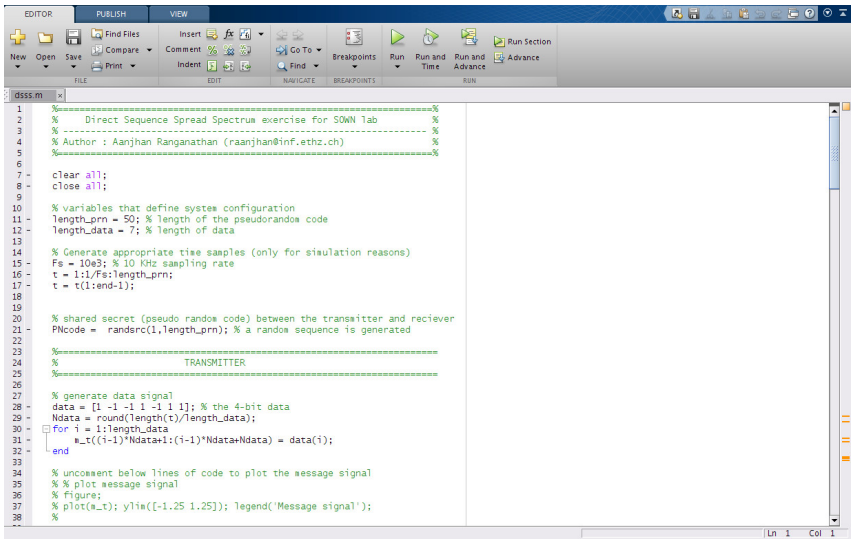


Figure 6.3: `Matlab` editor

- What is the effect of increasing the length of the pseudo random code sequence? *(Hint: Processing gain)*
- In addition to the security properties DSSS systems provide, can you think of other use cases and advantages (non-security) of using DSSS (or in general any code based spreading technique)? *Hint: CDMA, GPS*

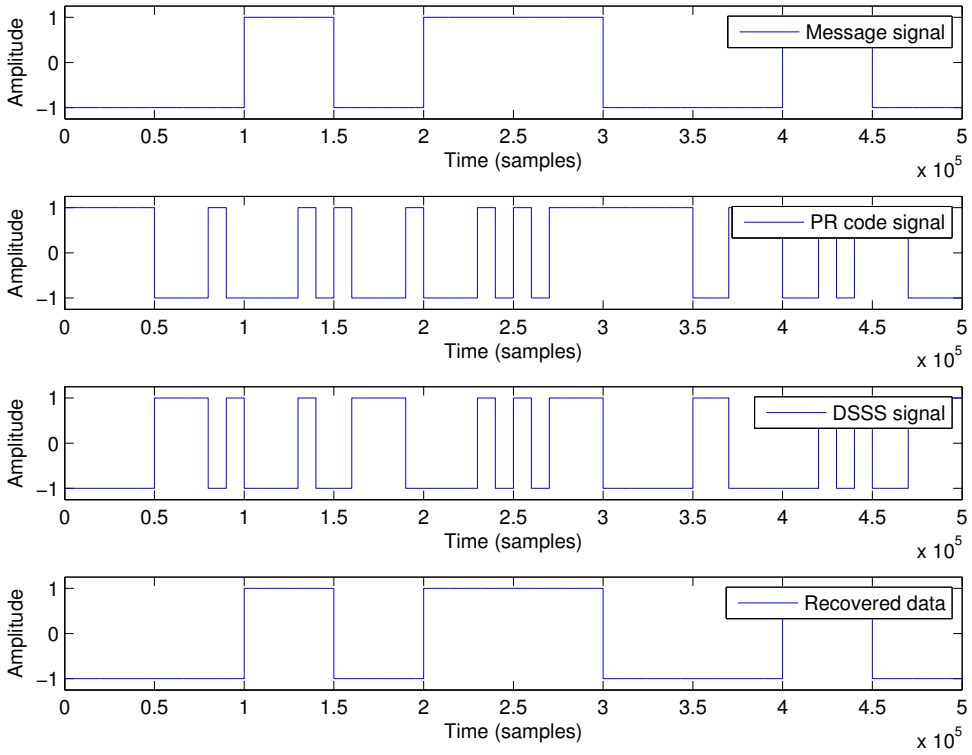


Figure 6.4: Plot showing the various signals in time domain.

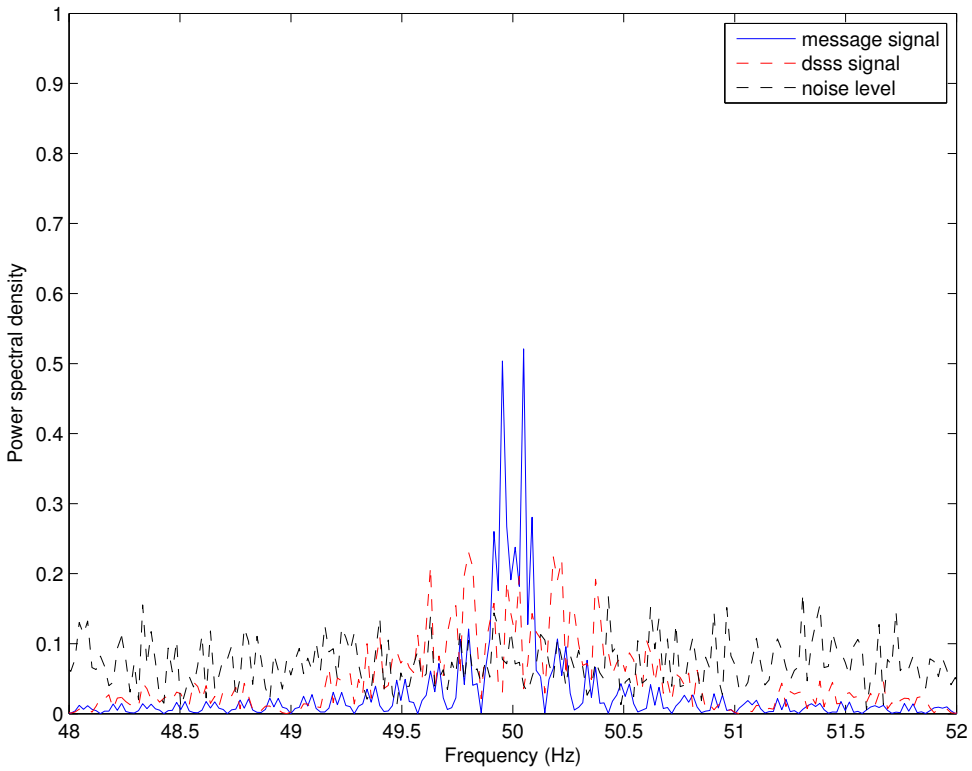


Figure 6.5: Plot showing the power spectrum of the original message signal, the spread DSSS signal and the noise level.

Chapter 7

Laboratory 4: Jamming Resistant Communication

Contents

7.1 Jamming in Wireless Networks

A *software-defined radio*[10] (SDR) is a hardware device capable of synthesizing arbitrary radio signals. The user provides a series of samples (amplitude values) which the SDR then uses to produce the corresponding signal waveform. In this exercise we will use USRP2 [12] (Figure 7.1) and USRP B210 [11] (Figure 7.2) software-defined radios to perform a jamming[8] attack on 802.11. The main difference between USRP2 and USRP B210 is that the former is accessible over the local LAN while the latter provides a USB interface. From now on we will refer to those USRPs as *USRP-LAN* and *USRP-USB* respectively. The following splitting applies:

- Teams *1, 2, 3 and 4* will be using USRP-LAN devices (one USRP per team).
- Teams *5 and 6* will be using USRP-USB devices (one USRP per team).

Assistants will connect the USRP-LAN devices to the local LAN (each one will be assigned a unique IP addresses and will be accessible directly over the local network) and the USRP-USB devices to the PCs of teams 5 and 6.

Teams will be divided in 3 groups. Each group will be operating on a different 802.11 channel and will create a separate access point:

- Teams 1 and 2: *Channel 1*, *SSID: router1*.
- Teams 3 and 4: *Channel 6*, *SSID: router6*.
- Teams 5 and 6: *Channel 11*, *SSID: router11*.

In each each group, one of the teams will perform jamming using their USRP, while the second team will monitor the effects of jamming using their USRP. Once the exercise is completed, we will reverse the team roles and jamming will now be performed by the other team in each group.

Figure 7.1: USRP software-defined radio with *ethernet* interface

Figure 7.2: USRP software-defined radio with *USB* interface

1. Each group will choose one machine from one team which will serve as an access point and one machine from the other team that will serve as a client connecting to access point.
2. Refer to previous labs for exact commands on how to start an access point using `hostapd` as well as how to associate a client. Each group should configure their access point to use a different channel and SSID as described above.
3. Using `hostapd` on the access point machine, start an access point on the specified channel. Assign the following IP address to the access point:

   ```
   # ifconfig wlan0 10.0.0.1 up
   ```

4. Connect the client machine to associate with the access point and assign an IP:

   ```
   # ifconfig wlan0 10.0.0.2 up
   ```

5. At this point, the client should be able to talk to its corresponding access point. Verify connectivity for the client by pinging the access point:

   ```
   # ping -i 0.001 10.0.0.1
   ```

   The issued command will both rapidly and constantly ping the access point. Leave the ping command running.

We will now proceed to configure the USRP software-defined radios.

USRP-LAN instructions. In order to be able to talk to the USRP-LAN, we need to setup our local LAN interface by assigning the correct IP address.

The **attacking** team will do the following steps:

1. Setup the local interface.

   ```
   # ifconfig eth0:1 11.0.0.21X up
   ```

   where X is the team number.

2. Verify that the USRP-LAN device is reachable.

```
# ping 11.0.0.20X
```

where X is the team number.

The **monitoring** team will do the following steps:

1. Setup the local interface:

```
# ifconfig eth0:1 11.0.0.21X up
```

where X is the team number.

2. Verify that the USRP-LAN device is reachable.

```
# ping 11.0.0.20X
```

where X is the team number.

Finally, **both teams** should execute the following two lines as root on the two machines which will control the USRP-LAN devices.

```
# sysctl -w net.core.wmem_max=1048576

# sysctl -w net.core.rmem_max=50000000
```

USRP-USB instructions. For the USPR-USB devices, teams should execute the following two lines as root on the machines to the USRP-USB devices are connected.

```
# sysctl -w net.core.wmem_max=1048576

# sysctl -w net.core.rmem_max=50000000
```

Starting GNU Radio. Now we should be able to issue commands to the USRPs. We will use `gnuradio-companion` from the *GNU Radio*[3] project to control our USRP2 devices. Run `gnuradio-companion` with the following command as root:

```
# gnuradio-companion
```

You should see a screen much like the one in Figure 7.3. First thing you need to do is double click the *Options* box on the upper left part of the screen and change the value of *generate options* to *QT GUI*.

We will now proceed to build the receiving (monitoring) and transmitting (jamming) chains in `gnuradio-companion`. Blocks are created by means of drag-and-drop. You can find various blocks on the right side of the screen. Each block has ports which serve as means to interconnect blocks. In order to connect two blocks, click the port of one block, followed by the click on the port of another block. There should now be a solid line between the two blocks, indicating that a successful connection was made. Assistants will help you through the process of creating the gnuradio blocks. We now know the basics and we are ready to construct blocks.

7.1.1 Attacking Team

The attacking team will setup a USRP2 to transmit a jamming signal.

1. Set the `samp_rate` variable to 1MHz (or 1e6 in scientific notation).
2. Create a random noise source (*Sources* $\rightarrow$ *Noise Source*) from the available selection in the right portion of the screen. Verify that the amplitude of the noise source is set to 1.

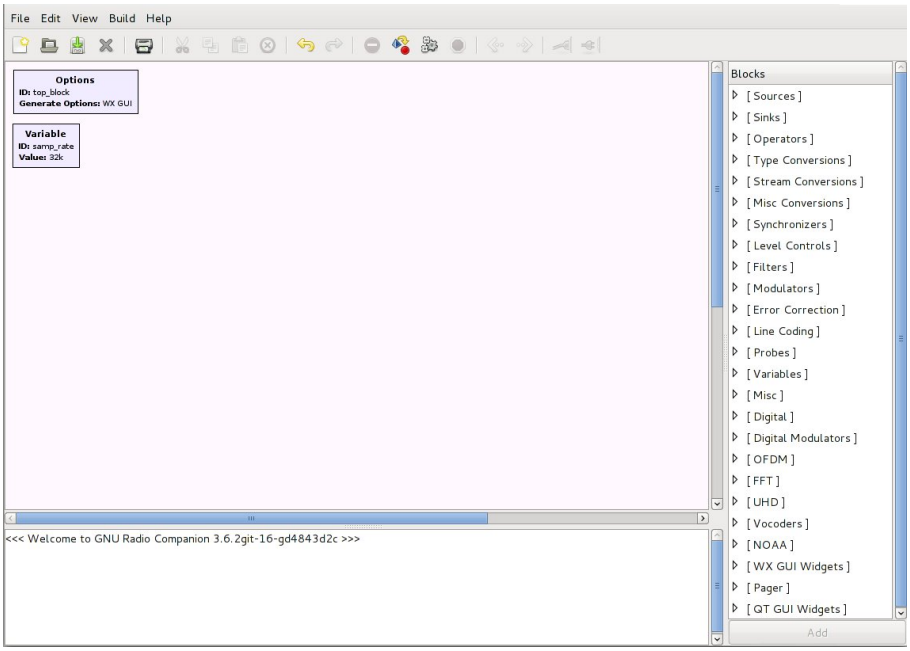


Figure 7.3: `gnuradio-companion` default screen

3. Next, create a low-pass filter[6] (*Filters* → *Low Pass Filter*). Set the *cutoff frequency* to 1KHz (1e3) and the *transition width* to 1KHz (1e3). Leave all other settings as they are.
4. Create a UHD USRP sink (*UHD* → *UHD : USRP Sink*). Set the *device address* field to value:
   - `addr=11.0.0.20X` where X is your team number, if you are using a USRP-LAN device.
   - Leave the field empty, if you are using a USRP-USB device.

   Also, set the *gain* field to 20 and the *center frequency* to:
   - 2.412GHz (2.412e9), if your access point operates on channel 1 (ssid router1).
   - 2.437GHz (2.437e9), if your access point operates on channel 6 (ssid router6).
   - 2.462GHz (2.462e9), if your access point operates on channel 11 (ssid router11).
5. Create a FFT scope (*INSTRUMENTATION* → *QT* → *QT GUI Sink*).
6. Connect the random source to the low-pass filter. Connect the low-pass filter to the USRP sink as well as to the FFT scope.
7. Confirm that your setup looks as the one shown in Figure 7.4.

We are now ready to start jamming. Coordinate with the other team of your group and confirm they are ready to measure the effects of jamming. Furthermore, make sure that the client machine is still successfully pinging the access point.

1. Before starting the jammer, make a hypothesis as to what effect the jammer will have on your established 802.11 network.
2. Start the jammer by navigating the `gnuradio-companion` menus (*Run* → *Execute*).
3. Inspect the output of `gnuradio-companion` (lower window) for any error messages.
4. You should see an FFT[2] plot displaying the transmitted signal.

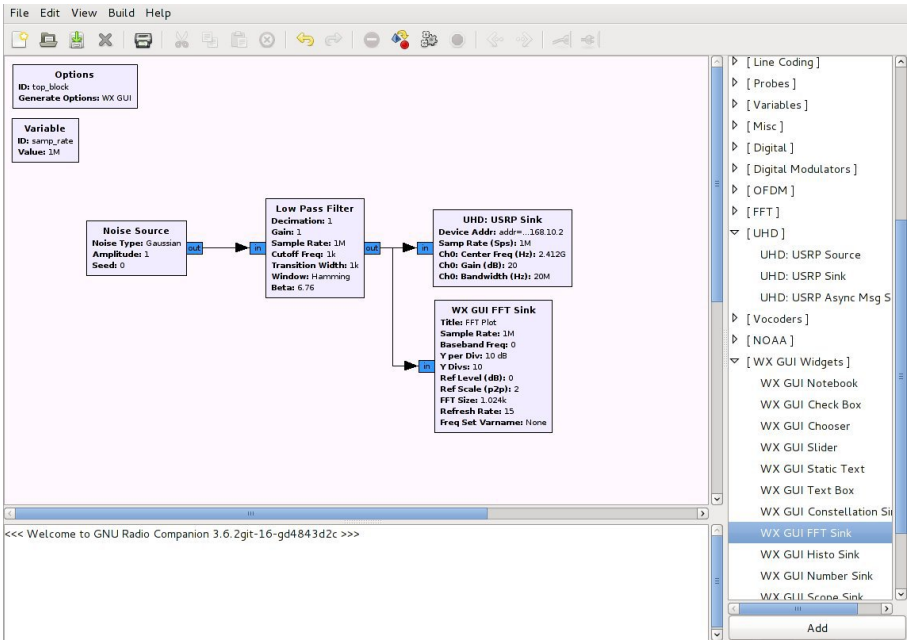


Figure 7.4: `gnuradio-companion` attacker screen

5. Observe the bandwidth of the jamming signal.
6. Observe the effect of jamming on the pinging client.
7. Once observations are made, make sure that the jammer is shutdown ($Run \rightarrow Kill$).

- What is the center frequency of channels 1 and 6 and 11 of 802.11?
- Why did we specifically choose to create networks on channels 1, and 11?
- What is the bandwidth of a 802.11 channel?
- What was the jamming signal bandwidth? (You can observe this from the FFT plot.)
- What was the low-pass filter used for?
- What would happen to the transmitted signal energy if no filter was used?
- What happened to your established 802.11 network during jamming? Does your initial hypothesis match observed results? If not, why?

You will now increase the bandwidth of the jamming signal to 100KHz and observe the effects on your established wireless network.

1. Make sure that the jammer is not running.
2. Change the *cutoff frequency* of the low-pass filter to 100e3.
3. Coordinate with the other group to check if they are ready to observe the effects of the jamming.
4. Start the jammer ($Run \rightarrow Execute$).
5. Observe the effects on the pinging client.

- Is the jamming effect different on the pinging client? If yes, why?

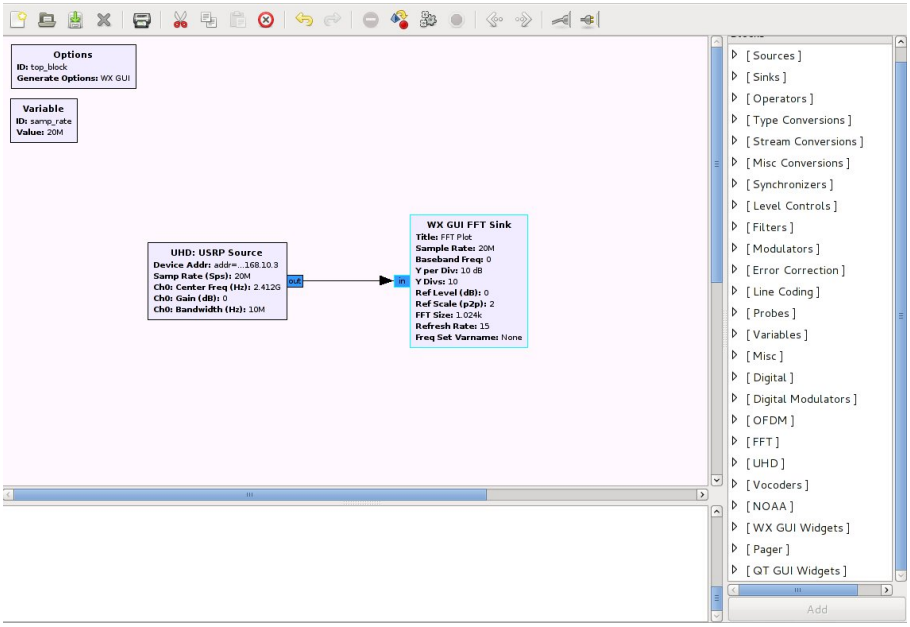


Figure 7.5: `gnuradio-companion` measurement screen

7.1.2 Monitoring Team

The monitoring team will setup their USRP device to constantly monitor the wireless channel.

1. Set the `samp_rate` variable to 10MHz (10e6).
2. Create a UHD USRP source ($UHD \rightarrow UHD: USRP\ Source$). Set the *device address* field to value:
   - `addr=11.0.0.20X` where X is your team number, if you are using a USRP-LAN device.
   - Leave the field empty, if you are using a USRP-USB device.

   Also, set the *gain* field to 20, the *bandwidth* to 1MHz (1e6) and the *center frequency* to:
   - 2.412GHz (2.412e9), if your access point operates on channel 1 (ssid router1).
   - 2.437GHz (2.437e9), if your access point operates on channel 6 (ssid router6).
   - 2.462GHz (2.462e9), if your access point operates on channel 11 (ssid router11).
3. Create a FFT scope ($INSTRUMENTATION \rightarrow QT \rightarrow QT\ GUI\ Sink$).
4. Connect the USRP sink to the FFT scope.
5. Confirm that your setup matches the one shown in Figure 7.5.
6. We are now ready to monitor the effect of the jamming. Start monitoring the channel by running the created project ($Run \rightarrow Execute$).
7. Observe the FFT plot. At this point, no jamming should be running and the only traffic should be the one performed by the pinging client.

- What could you observe from the FFT plot while the jammer was not running?
- Can you see the effects of the pinging client?

Coordinate with the other team of your group and wait until they start jamming the medium.

- What effects did the jammer have on the wireless medium?

Wait until the other team increases the bandwidth of the jamming signal.

- Can you see any difference in the observed FFT plot?

Bibliography

[1] Birthday problem. http://en.wikipedia.org/wiki/Birthday_problem.

[2] Fast fourier transforms. http://hyperphysics.phy-astr.gsu.edu/hbase/math/fft.html.

[3] Gnu radio. http://gnuradio.org/redmine/projects/gnuradio/wiki.

[4] Ieee 802.11. http://en.wikipedia.org/wiki/802.11.

[5] Lan/man standards committee. ansi/ieee std 802.11: Wireless lan medium access control (mac) and physical layer (phy) specifications. ieee computer society, 1999. http://standards.ieee.org/getieee802/802.11.html.

[6] Low-pass filter. http://en.wikipedia.org/wiki/Low-pass_filter.

[7] Mac address. http://en.wikipedia.org/wiki/MAC_address.

[8] Radio jamming. http://en.wikipedia.org/wiki/Radio_jamming.

[9] Related-key attack. http://en.wikipedia.org/wiki/Related_key_attack.

[10] Software-defined radio. http://en.wikipedia.org/wiki/Software-defined_radio.

[11] Usrp b210: Software-defined radio with usb interface. ttps://www.ettus.com/product/details/UB210-KIT.

[12] Usrp2: Software-defined radio. http://www.ettus.com/product/details/UN200-KIT.

[13] Martin Beck. Practical attacks against wep and wpa. http://dl.aircrack-ng.org/breakingwepandwpa.pdf.

[14] Jean-Pierre Hubaux Mario Čagalj and Imad Aad. Hands-on exercises: Ieee 802.11b standard. In *Laboratory for computer Communications and Applications (EPFL-I&C-LCA)*, November 9, 2004.