Network Security

Assignment 2, Thursday, September 10, 2015, version 1.1

Handing in your answers: Submission via Blackboard (http://blackboard.ru.nl)

Deadline:
Thursday, September 17, 23:59:59 (midnight) (exercises 1–3)
Thursday, September 24, 23:59:59 (midnight) (exercise 4)

In this exercise you will be using the following tools:

- Aircrack-ng suite: http://www.aircrack-ng.org/
- Wireshark: https://www.wireshark.org/
- Arpspoof: http://www.monkey.org/~dugsong/dsniff/
- The Python 3 network sniffer from the previous assignment (a reference implementation will be made available online after the deadline for assignment 1).

For aircrack-ng, wireshark and arpspoof you are allowed to substitute other programs which do the job, as long as you document the steps you took in detail. However, you have to expand the Python 3 network sniffer yourself. Do not compile these programs from source. Rather, use your Linux distribution's package manager to install the packages aircrack-ng, wireshark and dsniff (e.g. Ubuntu / Debian: `apt-get install dsniff`, Arch: `pacman -S dsniff`).

In this exercise, you are going to break into a WEP-"secured" network, map the network and its traffic flows, redirect some of the traffic, and change its contents. Once again, the exercise is long and intended to be challenging to everyone. Feel free to stop after putting in enough time, but include a note saying when and why.

Many commands in this exercise need to be run with root rights. This is denoted by a prefix `#`. When a command should be run without root rights, it will be prefixed with `$`. Do not include the prefix when typing the command.

Now let's test your hardware for compatibility. To successfully complete the exercise your wireless network card must support *monitoring* mode. First, disconnect from any wireless network you are on. Then, temporarily disable any network management software you may have running (e.g. in Ubuntu: `sudo stop NetworkManager`). Next, find out the name of your wireless interface (e.g. `ip l`, `iw dev`). It is likely called something along the lines of `wlan0` or `wlpXsY`.

Then, issue the following commands *as root*, substituting your own interface name for `wlan0`:

```
# airmon-ng stop mon0
# airmon-ng start wlan0
```

The second command should end with a message along the lines of `(monitor mode enabled on mon0)`. If it does not, but instead gives something like `mon0: ERROR while getting interface flags: No such device`, issue the commands:

```
# airmon-ng stop mon0
# airmon-ng stop wlan0
```

If that command complains with the message "You are trying to stop a device that isn't in monitor mode.", that's fine, recent versions of airmon-ng do that. This is just to make sure that the device isn't already accidentally in monitor mode. However, do *not* execute the command it suggests (`iw wlan0 del`).

Continue with:

```
# iw dev wlan0 set monitor none
```

If it complains about `iw` not being found, try

```
# iwconfig wlan0 mode monitor
```

If either command returns an error along the lines of `command failed: operation not supported`, your wireless card does not support monitor mode and you must contact Pol as soon as possible. We will then provide a USB wireless dongle that's guaranteed to work.

Read the entire exercise before starting. If you are stuck on one part, or would like to do some things from home, you can skip parts of exercises and come back to them later (e.g. you can try to edit the sniffer / forwarder without being connected to the network).

1. The WPA2 suite of security protocols is a (currently) secure replacement for WEP and WPA. It can operate in two modes: WPA2-Enterprise, where each client authenticates against a RADIUS master server and then exchanges keys, and WPA2-Personal, or pre-shared key mode, where every client knows the network's master key.

   Do you think a secured network running in WPA2 pre-shared key mode using a strong random passphrase is secure against clients sniffing traffic in the network? If so, why do you think so? If not, explain in general terms the principle behind an attack. Note that you do not need to have a strong understanding of the cryptography used in WPA2; use your intuition. Write your answer to a file called `exercise1`.

2. (a) Your first task is to find and map the network. It will be present at the werkcollege, and at other times it will be reachable from the central common area on the ground floor of the Mercator 1 building. (You may be able to pick up the network from the common area on the first floor, but we do not have any guarantees as to the signal strength and suitability for sniffing there.) Go there. Get comfortable.

   We will use the aircrack-ng suite for this exercise, but feel free to use something else if it has the same functionality.

   Create a folder called `exercise2`. Document all the steps you take in a file in this folder called `exercise2a`.

   Now, as described before, put your wireless card in monitor mode. Don't forget to disconnect from any connected networks and to disable any network management software that might interfere.

   ```
   # airmon-ng stop wlan0
   # airmon-ng start wlan0
   ```

   A listing of your network interfaces (`ip l`) should now also list `mon0`. If it does not, check whether it says `link/ieee802.11/radiotap` for `wlan0`. If the latter is the case, use `wlan0` instead of `mon0` for the monitor interface.

   Now let's see what's on the network:

   ```
   # airodump-ng mon0
   ```

   This will show you a listing of wireless networks, their security level, the access points' MAC addresses (BSSID), channel they operate on (CH), and some other information. List the networks you see. You do not have to list duplicate network names. Identify your target network's name, the access point's MAC address, and channel. In most of this exercise you will identify the network by the BSSID. If the target network is not obvious at first glance, it is probably down and you should contact Pol as soon as possible.

   This list will also show you wireless network clients ("stations"). Using the information you have on the target network, identify those clients that are connected to the target network. Write down their MAC addresses. When you look at the MAC addresses, does anything strike you as interesting or peculiar? If so, what? Try to explain it.

   If you do see networks and unassociated clients, but do not see any clients connected to the target network, the network is down *or* you've hit a bug we've observed in several machines running Ubuntu. You will not be able to capture enough traffic in a reasonable amount of time to crack the network, so contact Pol as soon as possible.

   (b) Now, let's go ahead and crack this network. First, exit airodump-ng (using ^c).

   Document all the steps you take here in a file called `exercise2b`.

   Cracking WEP is done by capturing enough packets from a network to enable some cryptographic attacks on the algorithms used. Capturing is also done by airodump-ng, with some extra command line switches. See the manual page for airodump-ng (`man airodump-ng`) for a detailed explanation of what each switch does, and substitute the desired values for `<channel>` and `<target BSSID>`:

   ```
   # airodump-ng -c <channel> --bssid <target BSSID> -w outputnetsec mon0
   ```

   Leave this running. Collecting enough packets in the file specified will take a few minutes.

   In a different terminal, open the manual page for aircrack-ng. Read what it does. Identify what option to use to select the target network. This is the only option you really need, but feel free to play around. Just document what you do.

Run aircrack-ng with the option you need, and the file(s) to which airodump-ng is currently writing its output. Once again, document what you use.

If the attack fails at first, just leave it running for a while while airodump-ng captures more packets, and it will retry every 5.000 packets. The attack should succeed in a reasonable amount of time, half an hour at most. I've had success with only 45.000 packets collected, but also had a run where it took nearly 200.000. The network is generating enough traffic to accomodate this.

When the attack finally succeeds, you are provided with the WEP key. Also document this. Leave airodump to capture some more data for good measure, 200.000 frames should be more than enough for the next exercise. Then, exit airodump and put the wireless card back into normal mode (`airmon-ng stop mon0`).

3. This exercise can be done from home after successfully completing exercise 2 and capturing a fair amount of data.

   (a) We now have the WEP key, but we also have a generous chunk of data from the network to work with. Let's work with that and see what we can learn from the network.

   In exercise 2 you had airodump-ng write its output to a few files, all prefixed with "outputnetsec". In the folder where you ran airodump, you should now see at least a file called `outputnetsec-01.cap`.

   Start wireshark. Since we're not going to capture anything, you can run it as a normal user. In wireshark, open the file `outputnetsec-01.cap` (or, if there are multiple `.cap`-files, the file on which you successfully ran aircrack-ng).

   Create a folder called `exercise3`. Describe what you see in wireshark, after opening the capture file, in a file called `exercise3a` in that folder. Try to explain why there is very little useful information in this capture file.

   (b) There is *some* useful information, however, and we are going to try to extract that. Wireshark has several very nice analysis tools built into it. They can be accessed in the `Analyze` and `Statistics` menus. Play around with the tools in the top part of the `Statistics` menu. Document which clients appear to be most active on the network and whom they seem to be communicating with, in a file called `exercise3b`. Also add the output from the `Comments Summary` tool, and try to explain why the `Protocol Hierarchy` looks the way it does.

   If you see more than a few very active clients, limit your description to the clients you identified in the previous exercise as connected to the network.

   If your protocol hierarchy contains more than a few reasonably explainable protocols, there's something wrong with your encrypted capture. Please contact Pol as soon as possible for the encrypted reference file.

   (c) Now, to solve the problem identified in exercise 3a, aircrack-ng has the tool airdecap-ng. Open its manual page (`man airdecap-ng`), identify what options you need to pass, then run it with these options on `outputnetsec-01.cap`. This will create a file `outputnetsec-01-dec.cap`.

   Close the file you currently have open in wireshark, then open this new file.

   IMPORTANT NOTE: Last year, airdecap-ng on Ubuntu 14.04-LTS had a bug which mangles WEP-encrypted traffic on decryption. We do not know whether this bug has been fixed by now, but if you encounter it, this is evident when viewing the decrypted capture in wireshark: missing UDP traffic, all sorts of weird protocols in the protocol hierarchy, and more. (TCP retransmissions, however, are just normal behaviour on a wireless network). If you are on Ubuntu you will likely hit the bug. If you are on another Linux distribution, but see stuff like IPv6 and all kinds of weird protocols in the protocol hierarchy, UDP packets going to the Bank of Scotland, no UDP conversations in the "Conversations" view at all, and other weird stuff, you have also probably hit this bug. In that case, contact Pol and we will work around this problem.

   Once again, describe what you see in wireshark, now in `exercise3c`. Explain why this is different from what you saw in exercise3a.

(d) Similar to exercise 3b, using the statistics tools, document which clients are communicating with whom, in a file called `exercise3d`. Also include their IP addresses. You should be able to go into much more detail this time. Therefore, this time, also zoom in on the conversations themselves (if you right-click on a conversation in the output from the `Conversations` tool, you can apply this conversation as a filter so that you only see its packets). There will likely be more than just a few conversations. Pick a few larger ones, or ones that look interesting. Briefly describe what each conversation is, whether it looks interesting, and why.

(e) There should be at least one connection which is consistently being rejected. See if you can find it. Document how you did this in `exercise3e`. Note you can filter out conversations which you are not interested in by simply right-clicking on that conversation in the `Conversations` tool, and preparing a filter. Filter out multiple conversations by using the `... and not...` entry on each. Then, apply the filter.

Most of the time, you'll simply want to build filters using the graphical interface, but for more documentation on their syntax, see
https://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html.

(f) Finally, there should be several conversations which, when you look at the packets' contents, are obviously interesting for you. Find them. Document the process in `exercise3f`, and also include why you think they are interesting.

4. This exercise requires you to be nice to fellow students. The attack you will perform requires you to use ARP spoofing to redirect traffic to your machine. Obviously, only one machine can do this at a time to a single conversation, so if there are more people working on the exercise at the same time, communicate with the others about which conversation you will use.

   We're going to attack one of the conversations you identified in exercise 3f. Recall from the lecture that WEP stands for Wired Equivalent Privacy. A wireless network presents itself, at a logical level, just like a wired ethernet. What this means is that e.g. it's not a given that you can just capture other clients' traffic if your drivers and hardware do not support this, even though it's all just radio waves. With specialized hardware, much more is possible.

   However, it also means that attacks which work on wired networks will often also work on wireless networks, once you're a "legitimate" client on the network. WEP was designed to provide only Wired *Equivalent* Privacy, i.e. it's just as bad as a wired network. Even worse, in fact, when you consider that the crypto is broken, but even if it were not, the attack we are about to carry out works regardless of whether you broke in to the network or whether you're a legitimate user.

   Start by actually connecting to the wireless network, using the WEP key you recovered in exercise 2. If you are using the virtual machine we provided, you will need to edit the file `/etc/network/interfaces` as root. The comments in it should make clear how it needs to look for WEP.

   You may notice that this network does not have a DHCP service running. Select a free static IP address in the network range, based on the clients you identified earlier. The network mask for this network is /24, or 255.255.255.0. It does not have an Internet gateway. If you run network management software that forces you to enter a gateway anyway, use any address that is not actually in use, e.g. the address ending in .2.

   If you are using the virtual machine, after editing the file, use `# ip l set dev wlan0 up` to connect, and `# ip l set dev wlan0 down` to disconnect.

   Create a folder called `exercise4`.

   (a) Run wireshark with root rights, and let it sniff your wireless interface. It should *not* use either monitor mode or promiscuous mode. Ping one of the other clients you identified in exercise 3, and see whether these pings show up in wireshark. Note that you may have to specify which interface to ping on, use the manual page (`man ping`) to figure out how.

   If this works, enable IP forwarding:

   ```
   # echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

   If using sudo, you will need a slightly different command:

```
$ sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

This is due to nuances in when shell redirection happens.

Now you can use arpspoof to trick the sending endpoint in the conversation to send its data to you instead of to the receiving endpoint, and your machine will forward the data. Use the manual page of arpspoof (`man arpspoof`) for details on how to use it. Note that you need to keep arpspoof running as long as you want traffic to be redirected, and you should close it down as soon as you're done. Note that you will need to tell arpspoof what interface to use, since that determines how and where it spoofs the targets.

Verify that the traffic of that conversation is flowing through your machine. You should see, among other things, two identical sets of IPv4 packets, interleaved. One set should have as a source address the MAC address of the endpoint you just started ARP spoofing to, and as destination address your own MAC address. The other set should have as source address your own MAC address and as destination address the MAC address of the original receiving endpoint of the conversation. If you do not see the second set, IP fowarding is not working. If you do not see the first set, spoofing is not working.

If this works, turn off arpspoof. Save the wireshark capture to a file called `exercise4a.cap` in the folder `exercise4`. Document the commands you used in a file called `exercise4a`, and explain why you're seeing these two sets of packets in wireshark.

Turn off IP forwarding:

```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

or

```
$ sudo sh -c "echo 0 > /proc/sys/net/ipv4/ip_forward"
```

(b) Start with your implementation of the sniffer from exercise 4 of last week, or use the implementation provided. Rename it to `mitm.py`, and place it in the folder `exercise4`. Change it so that it rewrites the packets according to the instructions contained inside the packet, then sends them on. For this, you will need to make several changes:

- You will need to bind the socket to the wireless interface.
- You will need to select the right packets (based on e.g. the combination of ethernet source, destination, IP destination, and port).
- You will need to find and replace the destination MAC address in the ethernet frame.
- You will need to find the correct part of the packet payload and rewrite it.
- You will need to find and zero out the packet checksum in the packet header in each frame. For this protocol, the checksum is optional, and a zeroed checksum indicates it's not being used. Not as sophisticated as recaculating the checksum, but just as effective. If you do not change the checksum, the destination will discard the packet before it reaches the application.
- You will need to use the `socket.send()` function to send the frame on. The documentation for the `socket` library does not state this clearly, but then again, we're not exactly doing legitimate stuff here.

If you don't feel confident about the above, another option is that you retrieve the payload from each frame, create a different UDP socket, and send the payload on through that socket.

Add some `print()` output to show you, when it runs, that forwarding actually works.

Once you've made these adaptations, run the forwarder, and then run wireshark and arpspoof the same way you did in exercise 4a. If everything goes correctly, the following will happen:

- Your forwarder will rewrite and forward the packets.
- Wireshark will show you a similar output, i.e. two sets of packets, this time there will be changes in payloads as well as in addresses.
- The final endpoint will register the modified packets including the changes you've made.
- The final endpoint will acknowledge receiving a valid modified packet by broadcasting the student numbers in a special packet. This way you have some feedback on whether or not you succeeded, so look out for those with e.g. wireshark.

Let this run for a few packets, then stop arpspoof and the forwarder. Save the wireshark capture file as `exercise4b.cap` in the folder `exercise4`.

5. As a completely optional extra, if you feel this was too easy, and have finished the assignment, take a look at http://www.aircrack-ng.org/doku.php?id=simple_wep_crack.

   If a network is not generating enough traffic for an attack by itself, there are more sophisticated attacks using e.g. ARP replay and disassociation to force traffic to be generated. Feel free to play around with this, but note that if your network driver does not support packet injection (the linked tutorial contains instructions on how to test that) these attacks will likely not work without patching and recompiling your driver first. Only do this if you have time left over, and *DO NOT USE A DISASSOCIATION ATTACK*. I will be very unhappy if you do because it messes with the exercise setup. Document what you do in a file called `exerciseBONUS`.

6. Place the files and directories `exercise1`, `exercise2`, `exercise3`, `exercise4`, and `exerciseBONUS` (optional) and all their contents in a folder called `netsec-assignment2-STUDENTNUMBER1-STUDENTNUMBER2`. Also include the airodump capture file (`outputnetsec-xx.cap`) you used to crack the network, and the corresponding airdecap file (`outputnetsec-xx-dec.cap`) analyzed in wireshark. Replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers, and accomodate for extra / fewer student numbers. Make a `tar.gz` archive of the whole `netsec-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Blackboard.