Network Security

Assignment 2, Wednesday, 22 April, 2020, version 1.1

Handing in your answers: Submission via Brightspace (https://brightspace.ru.nl)

Deadline: Wednesday, 6 May, 23:59:59 (midnight)

Goals: In this exercise, you are going to break the encryption of a WEP-"secured" wireless network, map the network and its traffic flows. Then on a *wired* network, you will use ARP spoofing to do a man-in-the-middle attack and change the contents of some network traffic. 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.

Tools: 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 network sniffer from Assignment 1 yourself. All the tools you need should already be installed on the Kali VM.

Some 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.

1. Do you think a secured network running in WPA2 pre-shared key mode using a strong random passphrase is secure against *connected* 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. In other years, we would have a wireless network set up at the university for you to play with. Of course that is not possible this year. Unfortunately we will have to make due with a network capture that we saved from last year. Download this capture from https://cs.ru.nl/~dsprenkels/netsec2020/outputnetsec-01.cap.

   The file you just downloaded contains the unaltered traffic from an WEP-protected network. Open the capture in Wireshark.

   For this exercise, write your answers to a file called `exercise2`.

   (a) What kind of traffic do you see? (Hint: Use the `Statistics > Protocol Hierarchy` tool.) Why do you not see any IP/TCP or other conventional data in the packet capture?

   (b) Even though the data may seem opaque at first, we can still extract *some* information from it. Look at the `Statistics > Endpoints`, `Statistics > Conversations` and `Wireless > WLAN Traffic` tools. Document which clients appear to be most active on the network. What is the network name, and which of the nodes is the wireless access point?

   (c) What is remarkable about the MAC addresses? What can you probably conclude from this? (Hint: How are MAC addresses allocated?)

(d) *(Optional bonus question.)* What kind of devices do you think these MAC adresses belong to (phones, laptops, etc.)?

(e) Read the manual page of `aircrack-ng`. Use `aircrack-ng` to get the WEP key that the network is protected with[1].

(f) Use the `airdecap-ng` tool to strip the WEP protection from the packet capture. This will create a new file called `outputnetsec-01-dec.cap`.

Close the file you currently have open in wireshark, then open the new `outputnetsec-01-dec.cap` file.

(g) Just like in exercise 2a, describe what you see in Wireshark[2]. Explain why this is different from what you saw in 2a.

(h) Similar to exercise 2b, using the statistics tools, document which clients are communicating with whom. Also include their IP addresses. Something should immediately strike you as peculiar when doing this. What do you think is going on here?

(i) 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.

(j) There should be at least one connection which is consistently being rejected. See if you can find it. 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.

(k) 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.

3. In this exercise, we will use a new set of VMs that will be on the same network the `netsec-kali` VM that you have been using already[3]. Download the new "Assignment2" VMs from https://cs.ru.nl/~dsprenkels/netsec2020 and import them into VirtualBox.

   [Update 21/04/2020] Make sure to **not** randomize the MAC addresses when importing the VMs.[4]

   Select all three new VMs (`netsec-gateway`, `netsec-peer1`, `netsec-peer2`), and press `Start`. This will start the VMs. After some time a login prompt should show on each of them. You don't have to interact with them. To do a graceful shutdown of the VMs, right-click, then press `Close > ACPI shutdown`.

   The virtualized network in the next exercises is different from the one you analyzed in 2, though I have tried to make it look similar. However, because we cannot simulate a wireless network using VMs, you will be dealing with a *wired* network in this exercise. The gateway on this network behaves like *switched* ethernet, not *hubbed* like a WEP-encrypted wireless network would.

   The `netsec-kali` machine is connected to the network through it's `eth1` interface. It is configured with the IP address 172.21.152.101/23. The gateway is at 172.21.152.1.

   Create a file called `exercise3`. Put your answers for this exercise in that file.

[1] There currently seems to be a bug with `aircrack-ng`, where the `-e` argument is broken. I recommend you to just omit the `-e` argument. `aircrack-ng` will know what to do.

[2] *Note:* In previous years, certain versions of airdecap-ng had a bug which mangles WEP-encrypted traffic on decryption. 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. If you see stuff like IPv6, obscure protocols you have never heard of in the protocol hierarchy, UDP packets going to the Bank of Scotland, no UDP conversations in the "Conversations" view at all, no packets being decrypted at all, and other weird stuff, contact us and we will work around the problem.

[3] If you didn't, follow the guide in Assignment 1 (appendix) first; then continue here.

[4] We have already noticed that this causes problems for some people!

(a) In other to get some knowledge about the nodes on the network, we run this command:

```
sudo arpspoof -i eth1 172.21.152.1
```

Fire up wireshark with root rights and start listening on the `eth1` interface. It should *not* use either monitor mode or promiscuous mode; make sure that that checkmark is disabled.

What are the nodes that you see on this network? (Hint: Use what you learned in exercise 2.)

(b) Read the `arpspoof` man page. Explain broadly what the command from 3a does.

(c) The conversation from question 2j also happens periodically on this network. Using the `arpspoof` man page (`man arpspoof`) and your answer to 3b, explain why we only see *one* of the sides of the conversation.

(d) Knowing that `172.21.152.1` is the gateway, and given your answer in 3b, explain what kind of traffic you are currently able to see, in terms of who communicates and where the traffic goes. Can you think of a kind of traffic that you are *not* yet able to see?

(e) What method for host discovery do you know that is more reliable than the one in exercise 3a?

4. We are now going to attack the conversations you identified in exercise 2k. In our virtual network, this data is sent from `172.21.153.20` to `172.21.153.10`.

Before continuing, make sure to terminate the `arpspoof` command from exercise 3a.

Create a directory called `exercise4`.

(a) Run wireshark with root rights, and let it sniff the `eth1` interface. Again, it should *not* use either monitor mode or promiscuous mode.

Ping one of these clients, 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:

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

Now you can use arpspoof to trick the endpoints in the conversation to send their data to you instead of to the other, and your machine will forward the data. Use the manual page of 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[5].

If this works, turn off arpspoof. Save the wireshark capture to a file called `exercise4/exercise4a.cap`. 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"
```

[5]One of the first things you can check if whether your OS has set up a restrictive firewall. If you run `# iptables --table filter --list FORWARD` you should see that the `FORWARD` chain is empty and its policy is `ACCEPT`. If this is not the case, find a way to disable your firewall.

(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 `eth1` 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 source and destination MAC addresses 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 `exercise4/exercise4b.cap`.

5. Place the files and directories `exercise1`, `exercise2`, `exercise3`, `exercise4` and all their contents in a folder called `netsec-assignment2-STUDENTNUMBER1-STUDENTNUMBER2`. Replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers, and accommodate for extra / fewer student numbers.

   Include capture files if they are relevant to your answers. However, please don't include files that are larger than 10 MB. Instead, use the Wireshark tool `File > Export Specified Packets` to trim them down.

   Make a `tar.gz` archive of the whole `netsec-assignment2-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Brightspace.