

## Computer Science 530 - Lab Assignment #5 -- Fall 2021

**Due: Friday October 29, 2021, 4:30 p.m.**

### Overview

## Infrastructure for Lab

There are two different Virtual Machines that you will use for this lab, but both are instances of the same basic image. You will use the linux or windows scripts (batch files) to configure the instances of the virtual machines, therefore you should run these VM's within VirtualBox.

### Location of files




The ova file for the one appliance is available in the CSci530 google drive in the folder for Lab 5 [here](#).

**Please note that you may need to login to google drive with your USC account in order to access these files.**

This image is the same as one that you used in previous labs, but I have included them again in the Lab 5 folder for ease of access.

The files in the google drive for lab 5 are:

My Drive > 2021 Lab 5 ▾ 👤

Name	Owner	Last modified
 lab5scripts.windows	me	11:54 F
 lab5scripts.linapp	me	11:54 F
 fedora30-fall20.ova 👤	me	11:53 F

You will note that there is a directory with scripts (or BATCH files) for this lab. There is a directory for windows machines, and another for Linux and apple systems. Download the scripts from the directory that is relevant to your machine. The scripts in this directory are used to clone the virtual machines (populate), start them (poweron), configure the network between them (construct network, this script is only present for the heartbleed part of the lab), power them off, and get rid of them when you are done with the lab.

You will run these scripts at the appropriate time for the experiments below.

### Some notes on this instance of fedora Linux

We have already loaded most of the programs you will need for this lab into the virtual appliance. When you start the virtual machine you will be asked to login. For this lab you will be logging in to the root account or the student account and the Passwords for both accounts is "c\$I@bLinuX". The third character is the letter "I" as in lab.

Unfortunately, in some virtual machines the "student" account was not set up correctly. If you are immediately disconnected after logging in to student, you can log in as root instead. Once you login as root, you may create the directory "/home/student" and then change the owner of the directory to "student". This will allow you to login to that account. I believe this step might not be necessary with this version of the virtual machine.

## CSCI 530 Lab

### Exercising the Wireshark network protocol analyzer ("packet sniffer")

#### Setting up this experiment

Scripts are used. A set of them is provided for each experiment. Please see the "Virtual Machine Usage" section of [this document](#). A summary of the scripts' use and intended execution order follows.

"vmconfigure-populate" is first, creates machines  
 "vmconfigure-construct-network" is next, if it exists  
 "vmconfigure-guestOS-internal-settings" is next, it makes the internal settings after powering the machines on;  
 so if this one exists "vmconfigure-poweron" is not needed  
 "vmconfigure-poweron" is next, if there is no "vmconfigure-guestOS-internal-settings"

Now you can use your VM(s). When you are finished:

```
"vmconfigure-poweroff"  
"vmconfigure-destroy" if you want to delete the machines, but not if you plan to come back to them later  
  
If you do come back to them later,  
  
"vmconfigure-guestOS-internal-settings" should be repeated, if present  
"vmconfigure-poweron" if there is not "vmconfigure-guestOS-internal-settings" present
```

#### Note - on taking screenshots

Below there are several junctures where you are asked to take screenshots (notations appear there in bracketed red italics). To do that, it's recommended to use a screenshot program that you may have on your host machine. Take the screenshot with the VM window open showing the activity of interest.

Alternatively, the guest virtual machine has a screenshot program. You can take a screenshot within the VM by pressing the PrintScreen key. The result is that a file is deposited on the disk. The name of the file will be something like "Screenshot from 2020-09-14 21-29-57.png" and it will be found in the "Pictures" subdirectory of the user's home directory. In the exercise, performed as student, that will be /home/student/Pictures/.

The host option is recommended in order to avoid the need to transfer the file out of the guest VM, since the VMs as provided are not equipped with interfaces that give them internet/external connectivity. It can be done and is documented on our class website but is best avoided.

### 1. Set up

This lab is to be done on two instances of the fedora30 virtual machine. One has hostname CLIENT and the other SERVER. They are networked, with respective IP addresses 192.168.1.2 and 192.168.1.1.

Create the machines and set up the exercise using the scripts provided for that purpose. Run, in order:

```
vmconfigure-populate.bat (or .sh)  
vmconfigure-construct-network.bat  
vmconfigure-guestOS-internal-settings.bat
```

On CLIENT:

Once your machines have booted, log in on CLIENT as student. Start the graphical interface.

**startx**

Open a terminal window from the menus and become root.

**sudo su -**

Verify CLIENT's IP address:

**ifconfig -a**

On SERVER:

Log in to SERVER as root. Here you do not need the graphical interface. Verify SERVER's IP address

**ifconfig -a**

### 2. Perform a preliminary capture

On CLIENT, run Wireshark from the menus (click "Activities" to show "favorites," among which there's a Wireshark icon).

Go to Capture -> Interfaces.

Use CLIENT's IP Address in a capture filter: "host 192.168.1.2"

Press Enter.

Return to the terminal window (Alt-Tab cycles through running applications).. At the prompt ping the other machine, SERVER:

**ping -c 2 192.168.1.1**

Observe resulting detection activity in Wireshark. Go to Capture -> Stop.

### 3. Get familiar with the Wireshark interface

Observe the three primary, vertically stacked panes:

- packet list pane - one line for every packet you captured (here you click on any packet to select it)
- packet details pane - the decoded breakdown of the selected packet
- packet bytes pane - the actual raw content of the selected packet in hex dump format

Right click on any packet in the packet list pane. From the context menu select "Show Packet in New Window". Maximize the new window. It has only two panes, specifically for the selected packet:

- packet details pane
- packet bytes pane

Select any line item in the details pane and observe the change of highlight in the bytes pane below. Expand the items you see and pick some sub-items. The selected line is the interpretation, or decode, telling the meaning of the highlighted bytes.

Close the single-packet new window you opened.

Wireshark understands how the protocol expresses information. The protocols that Wireshark knows are many:

Go to Analyze -> Enabled Protocols

Examine the long list, then Cancel.

Note colorization.

Go to View -> Colorize Packet List

Uncheck, note visual result, then recheck.

Go to View -> Coloring Rules. Examine, then Cancel.

#### 4. Capture/analyze the operation of a protocol of interest: the echo protocol

The protocol of interest is the echo protocol. It is implemented by an echo server, of which there are two. In order to start the echo servers, edit two files. They are /etc/xinetd.d/echo-dgram and /etc/xinetd.d/echo-stream. In each, change the line that reads "disable = yes" to "disable = no" instead. If you are not comfortable using the vi editor, you can accomplish the same thing programmatically instead by typing carefully, on SERVER:

```
sed -i 's/disable=yes/no/' /etc/xinetd.d/echo-dgram
sed -i 's/disable=yes/no/' /etc/xinetd.d/echo-stream
```

after making the changes, make the echo server run:

```
systemctl restart xinetd
```

One echo server uses UDP, the other TCP. Both use 7 as their port number, either UDP's port 7 or TCP's port 7 (which are separate and independent of one another). As a client, you reach/use the UDP echo server by sending stuff by UDP to port 7, and the TCP echo server by utilizing TCP and port 7. To understand what the echo server does, please read [the RFC document](#) that defines it. Any client that sends something to an echo server is a suitable client for the echo server. A client that prints on screen what is sent back is even better, for a user. A good client is netcat.

Go to Capture -> Start, and start Wireshark capturing again

Get ready to have nc send something to the echo server. On CLIENT:

```
nc -u 192.168.1.1 7
```

At nc's (visually null) prompt type "hi" then enter, followed by "bye" then enter, followed by control-C.

Go to Capture -> Stop, in Wireshark

Go to Statistics -> Flow Graph. Examine then close the window.

In Wireshark's packet list pane, successively highlight each of the 4 packets whose Protocol is given as ECHO. For each, in the packet details and packet bytes panes, note from and to which computers the packet traveled, what transport layer protocol was used (below "Internet Protocol" in the details pane) and with what port numbers, and what the application payload was (final, "Echo" line in the details; expand that and also look at the highlight in the bytes pane). How many bytes of application payload (e.g., the words "hi" and "bye") were carried in each packet? how many total in all 4 packets?

Go to Statistics/Conversation List/UDP [you may be asked to create a screenshot named statistics-udp.png here. Do so only if explicitly asked.]

Of how many bytes did the conversation consist? Is this the same as the total bytes of application payload you saw?

We will now start fresh using TCP instead of UDP to do the same thing.

Go to Capture -> Start, in Wireshark

Give the command

```
nc -t 192.168.1.1 7
```

Type "hi" then enter, followed by "bye" then enter, followed by control-C, in nc.

Go to Capture -> Stop, in Wireshark

Go to Statistics -> Flow Graph, select "TCP flow" and press OK. Examine then close the window. [you may be asked to create a screenshot named statistics-tcp.png here †]

In Wireshark's packet list pane highlight/select packet 1. Successively highlight each packet by repeatedly pressing the down arrow. While doing so, watch the details and bytes panes to locate the application payload ("hi" and "bye"). Those payload tokens are found embedded in four of the packets, but not the others.

Right click in the list on any packet belonging to the conversation, select "Follow" then "TCP Stream" in the context menu. In the new window that opens, client-to-server application data appears red-highlighted while server-to-client data is blue. The size of the "Entire conversation" is given. How many bytes in this conversation? Write it down. Close the Follow TCP Stream window.

Go to Statistics/Conversations/TCP and locate the conversation you had with the server's port 7

Of how many bytes did the conversation consist? Write it down. Is this the same as the total bytes the Follow TCP Stream window gave you?

#### 5. Create and apply a display filter

Wireshark has both "capture" and "display" filters. You already used a capture filter, confining your capture to just packets involving you ("host 11.22.33.44"). The syntax differs, even for the same thing, between the two filter types. Display filters are for after-the-fact viewing, narrowing display to only those captured/stored packets that interest you. As example of syntax differences, these express the same thing:

capture filter: host 11.22.33.44

display filter: ip.addr == 11.22.33.44

The place to enter a capture filter is found in the pre-capture initial screen or, subsequently, the "Capture/Options" dialog box; the place to enter a display filter is in the filter toolbar on the main interface. Wireshark can automatically create display filter expressions modeled on existing packets.

In the packet list pane select one of the packets in your captured conversation that has protocol "ECHO"  
 In the packet details pane right-click the protocol line labeled "Echo," click Apply as Filter, choose Selected  
 Clear the filter by clicking the small X on the filter toolbar to the right of the filter expression

## 6. Auto-generate a rule for your firewall

Suppose you want a firewall to prohibit use of the standard echo protocol. Suppose you use the iptables command in linux to build that firewall. You ask yourself, "What is the iptables command necessary to do the job?"

In the packet list pane select one of the packets in your captured conversation that has protocol "ECHO"  
 Go to Tools -> Firewall ACL Rules  
 In the dialog box open the "Create rules for" drop-down list and select several of the choices  
 Rules are generated characterizing this conversation, in the syntax of the firewall type you chose, which can be for either barring or admitting  
 Set it to "Netfilter (iptables)"

The iptables syntax for what you want appears. Write it down (better, "copy" it and paste it into a file, you'll need it below).

## 7. Capture a cleartext password

Your SERVER virtual machine runs both telnet and ssh servers. On it there's a "cs530" user account with password "Wireshark-CS530". Determine the IP address of one of your fellow students, and log in to his machine; alternatively log in to your own, from your own. Either telnet or ssh will prompt you for a password and grant you a login session. Do both, one after the other, capturing them in Wireshark.

`telnet 192.168.1.1`

and subsequently

`ssh cs530@192.168.1.1`

After you capture the telnet login session, locate the packets that contain the password and piece it together. Since telnet devotes a separate packet to each letter, you'll find successive packets carrying "W", "i", "r", and the other individual letters of the password. But you can find them. Make it easy on yourself, use "Follow TCP Stream".

Capture that screen, showing the in-transit password, into a file named "exposed.jpg", to include in your submittal for this assignment. *[ you are asked to create a screenshot named exposure.png here. Do so. ]*

(Terminate the telnet session on the server with the "exit" command.) After you capture the ssh login session, try to do the same and satisfy yourself that you can't identify the password-bearing packet(s) but even if you could it is illegible. "Follow TCP Stream" makes the encryption completely obvious. (Terminate the ssh session on the server with the "exit" command.)

## 8. Capture an http (browse) session

Here you will connect with an http server. Your instructor may give you a URL for it. Alternatively the virtual machine has one installed, which you or your fellow students could use instead. Create a simple and easily recognizable default web page for it to serve. On SERVER:

`echo "This is a simple web page." > /var/www/html/index.html`

Launch the web server:

`systemctl start httpd`

Return to CLIENT.

Open Firefox  
 Type in the target server IP 192.168.1.1 but do not press enter yet.  
 Go to Capture -> Start in Wireshark  
 In the browser, press enter to launch the http exchange  
 Go to Capture -> Stop in Wireshark  
 On any packet in the exchange (choose an early one) right-click and select Follow TCP Stream  
 Look for appearance in the capture of the text "This is a simple web page."

If you do not see it, you can use Ctrl-F5 in Firefox to have it refresh the page, overriding its cache (in case this is the second time or more you are visiting that page)

Observe the HTML code responsible for the page that appeared in the browser.

## 9. Observe protocols embodied in stored sessions (capture files)

In Wireshark, view how some other protocols operate. We'll examine **dhcp**, **kerberos**, and **ftp**.

Browse to <http://wiki.wireshark.org/SampleCaptures>  
 Search and find files "dhcp.pcap" and "krb-816.zip"  
 Download them (unzip the zipped one)  
 Go to File -> Open in Wireshark, open and study the two sessions that were captured.

To make better sense of a capture it's useful to know the rules of the protocol you captured. For dhcp you could read [the dhcp RFC document](#) (when you have time). Who is R. Droms? For kerberos, [the kerberos RFC document](#) (when you have *more* time). Who is C. Neuman?

As a 3rd example, observe how ftp operates. I downloaded a file. While doing so, I captured the datastream using Wireshark then saved it.

Download, unzip, and open [the capture file](#)  
 Add a column showing packets' sizes; Go to Edit -> Preferences/User Interface/Columns/New  
 Set Title to the word "Size" and, from the dropdown list, set Format to "Packet length (bytes)"

Select the new column within the column list, and move it up so it's the second column  
 exit Wireshark, restart it, and reopen the file  
 sort the packets by size in the packet list pane by clicking on any the "Size" column title

Several of the questions in the assignment refer to the resulting screen.

# 10. Exercise Wireshark's SSL/TLS decrypt capability

As user root on CLIENT create a directory:

```
cd
mkdir snakeoil
cd snakeoil
```

Obtain the capture of an SSL/TLS exchange together with the key that was used during that exchange, in a sample file. It may be on your local machine, as /home/server/snakeoil2\_070531.tgz. Otherwise get it from <https://wiki.wireshark.org/SampleCaptures> (search to heading "SSL with decryption keys"). [ depending on your browser, try right clicking this link and something like "save link" in the ensuing context menu; normal left clicking may not get it due to file type, it isn't an html file ]. Put it in /root/snakeoil (the browser may have deposited it in /home/student/Downloads or /root/Downloads).

Extract:

```
tar -xvzf snakeoil2_070531.tgz
```

You get:

```
[root@CentOSvm snakeoil]# ll
total 60
-rwx----- 1 1010 513 25057 Jun 27 2006 rsasnaeoil2.cap
-rwx----- 1 1010 513 910 Jun 27 2006 rsasnaeoil2.key
-rwx----- 1 1010 513 230 May 31 2007 rsasnaeoil2.README
-rw-r--r-- 1 root root 23205 Sep 30 15:00 snakeoil2_070531.tgz
[root@CentOSvm snakeoil]#
```

Before we invoke Wireshark let's take a look at the key we're given:

```
cat rsasnaeoil2.key
```

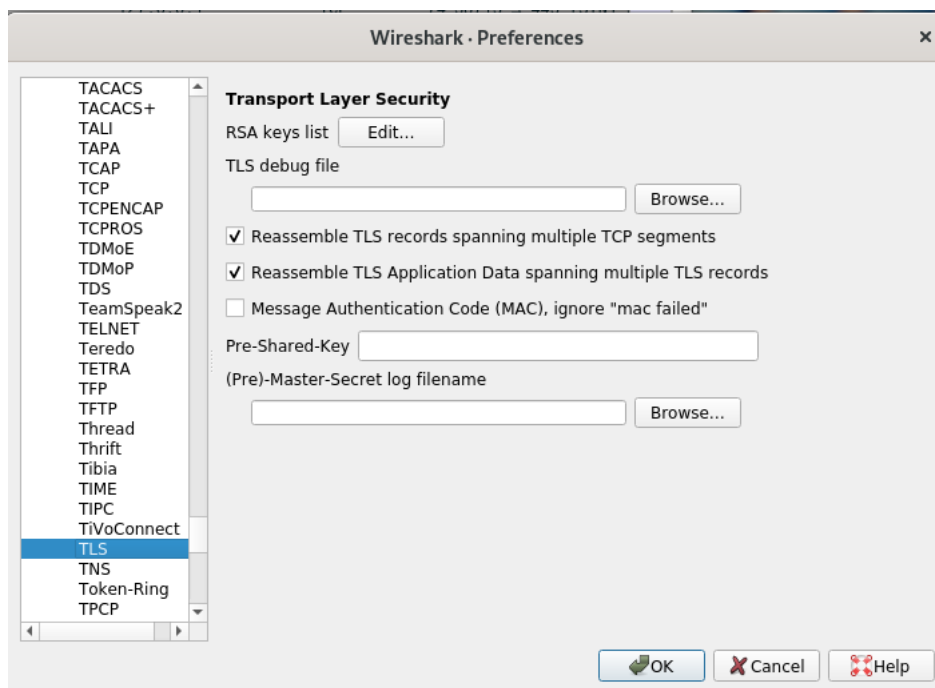
"RSA PRIVATE KEY," it says. We will give it to Wireshark soon.

Now in Wireshark open the resultant capture file rsasnaeoil2.cap. It holds 58 frames. The first 3 are a standard tcp handshake establishing a connection. What follows is whatever went back-and-forth across that connection. This is normal and routine. What went across this particular connection was an ssl/tls negotiation and data exchange. tcp carries tls, which in turn carries the application data. Unless the application happens to engage in some encryption of its own, what tls carries will be clear while, because tls will (almost certainly) encrypt it, what tcp carries will be obscure. The application in this case was apparently a browser, because (once decrypted) you'll see it to have "emitted" http. Regular cleartext http. There's no application layer encryption here. Make a mental snapshot of the set of frames in the packet list pane, so you can loosely compare it with what you'll see later.

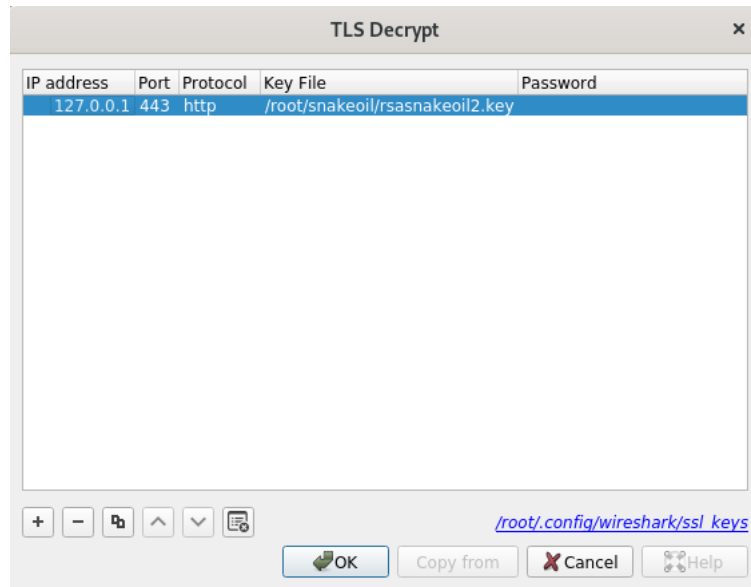
Select the 4th, "Client Hello" frame. It's the start of the tls exchange. Right click on it. The context menu offers both "Follow TCP Stream" and "Follow TLS Stream." You can look at either. You expect the tcp stream to be obscure, as noted, but the tls stream to be clear. First try tcp, "Follow TCP Stream.". Red stuff was client-to-server, blue server-to-client. Legible or illegible? Close that window. Now follow the tls stream. This one doesn't show encrypted stuff, however it's altogether blank. That's because Wireshark lacks the all-important key with which to decrypt the tls stream. The same one that the 2 tls's on either side of the connection had used for that. Wireshark shows you nothing, because it can't. But we are supplied with tls's key in a form Wireshark can assimilate, file rsasnaeoil2.key you viewed earlier. Very generous. Granted that same key tls used for decrypting, Wireshark will now be able to do so too.

We must supply that key to Wireshark, then try to again to follow tls with the key's benefit. Use keystroke ctrl-shift-P to open the Preferences dialog. There, expand the "Protocols" menu option and on the resulting submenu navigate down to "TLS".

You obtain:



At "RSA keys list:" press the "Edit" button. The "TLS Decrypt" dialog appears. There, press the plus sign to add a new key. Supply the values shown here for IP address, Port, Protocol, and Key File.



Again open /root/snakeoil/snakeoil2.cap (try File/Open Recent among the menus). Look at the set of frames in the packet list (top) pane. Is it different than before? Again right-click on frame 4 and follow tcp. Still obscure. But then follow tls. Hot dog! you got something this time. Even better, you can clearly read it. It's an unmistakable back-and-forth between a web client and server, http requests and replies in cleartext. This is what tls saw and handed to the browser when the .cap file was created, and is now what Wireshark sees and hands to you.

[ you are asked to create a screenshot named snakeoil-decrypt.png or .jpg here. Do so. ]

This is artificial. Outsiders, attackers, are not supposed to be privy to the tls key! We only have the key because the Wireshark people put it on their website for us. How did they get it? Where could *we* get it if we run *our own* tls conversations? Some browsers (Firefox is one) are able to obtain the key (or, with similar outcome, pre-master secret) and log it into a file during a tls conversation. You can then pick it up and hand it to Wireshark. Our focus here is on the Wireshark feature, not the tls protocol to which it relates. However if you have further interest, in all these aspects, here's a [thorough tutorial](#).

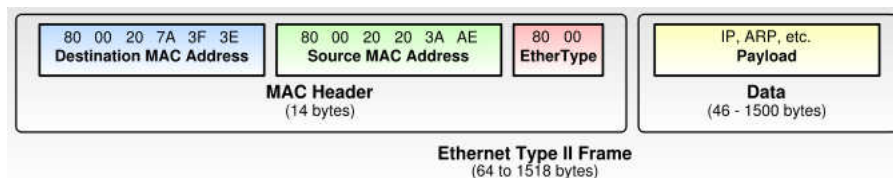
Don't be disturbed at the ease of "breaking" tls here. I argue that you don't break it if you have the key. I don't "break" the lock on my front door, I "unlock" it because I have the key. Attackers "break," I "unlock." Wireshark only has the key if given it. And these investigations (and Firefox's key logging) are done, and doable, *only* on the tls conversation endpoint machine where the key is *supposed* to be. So it's OK. A Wireshark user on some intermediate router would never have the key so would not be able to get the cleartext datastream. The Wireshark feature is not absolute or *carte blanche*. No blank check here. It only works in conjunction with gaining the key, and it normally can only gain it legitimately. If illegitimately, the illegitimacy is upstream of the Wireshark tool itself in however the key got obtained or purloined.

### The assignment:

Prepare your answers in a Microsoft Word doc file named **snifflab.doc**. In it, give numbered answers to these questions:

1. The number of bytes in the echo protocol exchange in section 4 above, according to the "Follow TCP Stream" window, is \_\_\_\_\_
2. The number of bytes in the echo protocol exchange in section 4 above, according to the Statistics/Conversation List/TCP window, is \_\_\_\_\_
3. The iptables command syntax to create a firewall rule prohibiting use of the standard echo protocol (Section 6 above) is:  
\_\_\_\_\_
4. OMITTED - Do not answer this question.
5. The number of frames in section 9's datastream was \_\_\_\_\_
6. The average length/size (in bytes) of the frames in section 9's datastream was \_\_\_\_\_
7. The most common frame size among the frames in section 9's datastream was \_\_\_\_\_
8. The maximum frame size among the frames in section 9's datastream was \_\_\_\_\_
9. For any of those max-sized frames, the size of its ethernet payload portion was \_\_\_\_\_
10. For that frame, the size of the remainder of the packet (ie, its header) was \_\_\_\_\_
11. For that frame (and all the others like it) Wireshark names its highest-level payload (see the packet details pane). It's \_\_\_\_\_
12. The observed value of the maximum frame size is interesting. It could not be any larger because (consider the reference graphic that follows):  
\_\_\_\_\_

For reference, question 12:



13. At the bottom of your file, insert two screen captures you made:
- the one you generated in section 7 above (telnet login showing cleartext password)
  - the one you generated in section 10 ("snakeoil" tls decrypt)