14-741/18-631: Homework 6

All sections Due: Thursday November 18, 2021 by 10:00am EST

Name:

Andrew ID:

Total (100 pts max):

Guidelines (Please read before starting!)

- Be neat and concise in your explanations.
- You must use at most one page for your explanation for each Problem (code you wrote may be on additional pages). Start each problem on a new page. You will need to map the sections of your PDF to problems in Gradescope.
- Some questions are marked as **Team work**. For those, you will be allowed to collaborate within your team.
- Please check your English. You won't be penalized for using incorrect grammar, but you will get penalized if we can't understand what you are writing.
- Proofs (including mathematical proofs) get full credit. Statements without proof or argumentation get no credit.
- There is an old saying from one of my math teachers in college: "In math, anything partially right is totally wrong." While we are not as loathe to give partial credit, please check your derivations.
- Write a report using your favorite editor. Note that **only PDF submissions will be graded.**
- Submit to Gradescope a PDF file containing your explanations and your code files before 1:29pm Eastern Standard Time on the due date. You can find the timing for EST here: `https://time.is/EST`. Late submissions incur penalties as described on the syllabus (first you use up grace credits, then you lose points).
- If you choose to use a late day, you do not have to inform the instructors. We will calculate the number of late days used at the end of the semester based on the time of submission on Gradescope.
- Post any clarifications or questions regarding this homework to Piazza.
- **Team work** As a team, beyond designated team questions, you are encouraged to shared resources (e.g., TA's help, online resources you found helpful); you are encourages to set up virtual study sessions with your teammate(s) to check each other's progress and discuss homework assignment solutions.
- **This is not a group assignment. Beyond your teammate, feel free to discuss the assignment in general terms with other people, but the answers must be your own.** Our academic integrity policy strictly follows the current INI Student Handbook `http://www.ini.cmu.edu/current_students/handbook/`, section IV-C.

1 Teamwork: Running Kerberos and Wireshark (33 pts)

Kerberos is slightly more complicated than what we discussed in the lecture. While the steps were overall correct, we only presented a simplified version of the protocol. In Kerberos version 4, a client C, an authentication server AS (trusted), a ticket granting server TGS (which is often the same machine as AS but doesn't have to be), and a server V are involved. C wants to access a service on V. The following exchange takes place:

1. $C \to AS$: ID_C, ID_{TGS}, TS_1
2. $AS \to C$: $\{K_{C,TGS}, ID_{TGS}, TS_2, L_2, Ticket_{TGS}\}_{K_C}$
3. $C \to TGS$: $ID_V, Ticket_{TGS}, Authenticator_C$
4. $TGS \to C$: $\{K_{C,V}, ID_V, TS_4, Ticket_V\}_{K_{C,TGS}}$
5. $C \to V$: $Ticket_V, Authenticator'_C$
6. (optional) $V \to C$: $\{TS_5 + 1\}_{K_{C,V}}$

where

- $Ticket_{TGS} = \{K_{C,TGS}, ID_C, AD_C, ID_{TGS}, TS_2, L_2\}_{K_{TGS}}$
- $Ticket_V = \{K_{C,V}, ID_C, AD_C, ID_V, TS_4, L_4\}_{K_V}$
- $Authenticator_C = \{ID_C, AD_C, TS_3\}_{K_{C,TGS}}$
- $Authenticator'_C = \{ID_C, AD_C, TS_5\}_{K_{C,V}}$

and ID_x is the identity of x, AD_y is the authentication domain of y. TS and L denote timestamps and lifetimes, respectively.

The purpose of this exercise is to observe in practice the establishment of a Kerberos connection. You can use the Ubuntu virtual machine we provide by downloading it from Canvas. *Please note that the package is about 1.5 GB, so make sure to download it outside of peak hours. Use a wired connection as much as possible (it is faster anyway).* Once downloaded, the VM is in `tar.bz2` format. This can readily be unarchived on MacOS and most UNIX variants (FreeBSD, Linux). For Windows, you likely will need 7zip (`http://www.7-zip.org/`) to unarchive it.

Alternatively, you can install the necessary packages on your own machine. You will need Wireshark, a Kerberos client that can be accessed from the command line, and support for the OpenAFS filesystem (`http://www.openafs.org/`). On the VM we provide, the login is `user`, password `user`, and that user is provided `root`-equivalent privileges via `sudo`.

Teamwork is allowed for this problem, but each member of the team is required to submit individual answers due to the nature of some operations required.

1. Why is the last step (Step 6) optional?
2. From the command line, run `kdestroy` to remove any stale tickets. Then, run Wireshark, and start a packet capture. While the capture is running, contact the Carnegie Mellon authentication server by executing the following command line: `kinit` [your andrew id]`@ANDREW.CMU.EDU` (Note that Kerberos is case-sensitive.)

In Wireshark, filtering out any non-Kerberos packets from the capture. In this trace, identify the packet corresponding to the request to the server, and the response from the server. Answer the following questions, justifying your answers **with screen captures** from Wireshark (highlight the important part of the screen capture):

(a) Which version of Kerberos are you running?
(b) What is the server name? (AS)
(c) What is the client name? (C)
(d) What encryption method is being used?
(e) What is the encrypted value of the ticket $Ticket_{TGS}$ in the lecture notes (use only the first 8 hexadecimal digits)?

3. For the `klist`:

   (a) Provide the output of `klist` **(screenshot required)**.
   (b) How many tickets are currently valid?
   (c) For how long are they valid?
   (d) Who are the principals involved?

4. Destroy all your tickets (`kdestroy`) and make sure that AFS is not running:
   (`sudo /etc/init.d/openafs-client force-stop`). Now, while running a Wireshark capture, start AFS and complete an AFS operation. This can be done with the following commands:

   - `sudo /etc/init.d/openafs-client force-start`
   - `kinit` [your andrew id]`@ANDREW.CMU.EDU`
   - `touch /afs/andrew.cmu.edu/usr/`[your andrew id]`/14741-test`
   - `rm /afs/andrew.cmu.edu/usr/`[your andrew id]`/14741-test`
   - `sudo /etc/init.d/openafs-client force-stop`

   Answer the following questions:

   (a) Show the four Kerberos messages that preside over the establishment of the AFS connection. **(screenshot required)**
   (b) What is the name of the AFS server? (V)
   (c) Show that the ticket ($Ticket_{TGS}$) the authentication server gave you is sent to the ticket granting server. **(screenshot required)**
   (d) Identify the ticket that the TGS returned to you ($Ticket_V$), and show that it is sent to the AFS server when you are trying to create the file `14741-test`. Show only the first eight hexadecimal digits of the ticket. Hint: the AFS decoder for Wireshark is not the greatest thing on Earth, and your Kerberos ticket may be buried a bit. Look carefully! **(screenshot required)**
   (e) Is mutual authentication used?

5. For this `klist`

   (a) Once again, provide the output of `klist` **(screenshot required)**.
   (b) How many tickets are currently valid?
   (c) For how long are they valid?
   (d) Who are the principals involved?

2 Decrypting SSL traffic (17 pts)

The file picoctf_ssl_flag1.pcapng contains a capture of encrypted traffic between a client machine and picoctf.com, a webserver running HTTPS.

1. Using wireshark, decrypt the traffic and tell us what the 14741 flag is. The file premaster.txt will help.
2. What environment variable was set to cause Chrome to generate premaster.txt?
3. If you can decrypt a file containing this HTTPS traffic, why is web browsing with HTTPS still secure? Hint: Google ”Wireshark decrypt ssl premaster.txt”

3 A Simple Authentication Protocol (20 pts)

Alice designs a simple authentication protocol between a garage remote control and the door. When a user presses a button on the remote control (e.g., door open button), the door and the remote will first run this protocol so the door knows that the correct remote is present and the remote knows that the door is present. We assume that the remote (R) and the door (D) share a secure, symmetric key K. Below are the protocol steps.

1. R $\rightarrow$ D: open, N_R
2. D $\rightarrow$ R: $\{N_D, N_R\}_K$
3. R $\rightarrow$ D: N_D

First, the remote generates a nonce and sent it to the door. Second, the door generates a new nonce and sends the encrypted nonces using key K to the remote. Next, The remote decrypts the message, check the received N_R is the same the one it generated, and returns N_D to the door. Finally, the door checks received N_D is the same as the one it generated and complete the authentication. If the checks in the late two steps fail, the protocol aborts.

1. (10 pts) Can N_D be implemented as a simple counter? Explain why or why not.
2. (10 pts) Can N_R be implemented as a simple counter? Explain why or why not.

Note the following

1. you can safely assume that the counter space is sufficiently large. No need to worry about the counter overflowing or wrapping around.
2. Assume attackers are capable of observing and injecting messages, but not intercepting any messages (i.e., cannot prevent D or R from seeing messages meant for them).
3. Assume that the remote control R is in a safe location. R will not initiate the first message without the user pressing the button. R will not respond to D in the 3rd step if R did not initiate the first step.

4 Alice and Bob (30 pts)

Bob wants to work for Alice's security consulting company. Alice insists that Bob proves his worth by designing a simple secure communication protocol before she agrees to hire him. The protocol has to 1) be based on public key cryptography, and 2) be able to transmit and acknowledge receipt of any message m_X securely between two parties A and B.

Bob comes up with the following protocol for a message m_X:

1. A sends $\{pk_A, m_X\}_{pk_B}$ to B, where pk_A and pk_B are A and B's public key respectively.
2. B in turn acknowledges receipt by sending $\{B, m_X\}_{pk_A}$ to A.

Alice is not impressed. She says the protocol is unnecessarily complicated, and that the following protocol would produce the *exact same level of security* as Bob's protocol while being simpler:

1. A sends pk_A,$\{m_X\}_{pk_B}$ to B, where pk_A and pk_B are A and B's public key respectively.
2. B in turn acknowledges receipt by sending $\{m_X\}_{pk_A}$ to A.

Bob disagrees and asks you to be the judge.
Questions:

1. Which one(s) of the following security properties does Bob's protocol enforce? (1) The secrecy of m_X. (2) Upon decrypting the message from Bob and checking the message is the same as m_X, Alice knows that Bob has received m_X. (3) Upon decrypting the message, Bob knows that Alice is the one who has sent m_X. For each of the above property that holds, explain why or why not. To explain why not, use a concrete attack scenario.

   For extra credit, Analyze the above properties in ProVerif (See Problem 5).

2. Show that Alice is wrong – in that one of the security properties Bob's protocol enforces is not maintained anymore. Explain how the property is violated in your writeup.

3. Bob's protocol unfortunately has a major problem: It is vulnerable to a replay attack in case where A sends to B m_X multiple times. An attacker Mallory can impersonate B and acknowledge the receipt of A's message. Enhance the protocol proposed by Bob to prevent this attack. Explain why your enhancement(s) work.

5 Extra credits (20 pts)

Before you proceed with the extra credit, please review the following:

- **The goal of this extra credit is to give an opportunity for students who have finished this problem set early to explore further in protocol security.**
- **We strongly discourage students who are already short on time to finish regular course load from trying to complete this problem. Nonetheless, everyone can try it out and provide feedback to us to gain partial credit.**
- **We expect students to explore this problem largely on their own.**
- **Please direct all questions on this problem to Zichao Zhang's TA hours.**

For the above problem, luckily, there is a software called ProVerif[1] that can help you analyze their protocols. Please refer to the document in your homework handout on how to install and get started with ProVerif. **Note:** Please include all the properties you are checking in one ProVerif file. That is, for questions 3-5 you will include a single screenshot of ProVerif's output but you will have to interpret/explain the results for each question in your writeup. You will upload your ProVerif file on gradescope for question 6.

1. (3pt) How long did you spend on this problem and how far have you gotten (e.g., installed the tool, read the manual, finished coding Alice and Bob, entered properties, finished verification).
2. (2pt) Suggestions for making this problem more accessible for you.
3. (4 pts) Proverif results for analyzing property (1) in Q4.1.
4. (3 pts) Proverif results for analyzing property (2) in Q4.1.
5. (3 pts) Proverif results for analyzing property (3) in Q4.1.
6. (5 pts) Proverif code (upload separately on gradescope)

[1] `https://bblanche.gitlabpages.inria.fr/proverif/`