14741/18631 HW2 - Write-up

Hao Zhang

TOTAL POINTS

75 / 75

QUESTION 1

1 Q1.1 How to verify public key? 5 / 5

✓ **+ 5 pts** **Metions 'use key ID/fingerprint to verify the integrity of public key'**

+ 0 pts Incorrect answer / no attempt

QUESTION 2

2 Q1.2 CTF flags + key fingerprint 15 / 15

✓ **+ 7 pts** **Correct flag**

✓ **+ 5 pts** **Matching fingerprint**

✓ **+ 3 pts** **Steps taken**

QUESTION 3

3 Q1.3 MIME vs. Inline 5 / 5

✓ **+ 5 pts** **States PGP/MIME is more preferable/desired. Includes supporting justification.**

+ 5 pts States both can be good, depending on the situation. Includes supporting justification.

+ 2.5 pts States inline is more preferable/desired. Includes supporting justification.

+ 0 pts No attempt

QUESTION 4

4 Q2: Hash Extension 30 / 30

✓ **+ 20 pts** **Attack worked / correct flag**

Understanding of state in hashing

✓ **+ 5 pts** **Hash state (h0..h7) itself is preserved in the hash output, so it can be used to continue the hash**

+ 2.5 pts Use of the original hash (h0..h7) but doesn't understand where it comes from

✓ **+ 5 pts** **Explained why include extra padding in the middle of the message since the original hash contained padding so the extension will be double-padded**

- 3 pts Didn't include flag and/or CTF username

- 1 pts Didn't include code in writeup

- 3 pts Didn't match pages

- 15 pts Didn't submit code at all

- 4 pts Hash extension doesn't replace existing data from the chunk, only adding new data

QUESTION 5

5 Q3.A: Explanation of your validation approach 10 / 10

✓ **+ 4 pts** **Correct Trust Store Initialization**

Chain validation

✓ **+ 3 pts** **Correct approach - verify the chain root->intermediate->leaf**

+ 1.5 pts Checking against the root but not checking the entire chain in the root->intermediate->leaf order

✓ **+ 3 pts** **Correct Domain validation**

+ 0 pts No explanation of approach

- 2 pts Didn't match pages

QUESTION 6

6 Q3.B: The flaw and potential solution 10 / 10

✓ **+ 10 pts** **Correctly Identifies CRLs/OCSP (important since CRLs are what are actually used in the real-world for revoking compromised certs)**

+ 6 pts Identifies stolen/compromised cert being abused / needing to be revoked, but not how it is done

+ 0 pts Empty / Incorrect

Question 1

CTF Username: IwishIcansing Flag: 818870150103756680902610936273 28
Fingerprint: 3BE49A0350C5384B7993C19936B601656B23B089

1. Based on this information, how do you verify that the public key you got from the key-server is valid, i.e.,that no one has modified it?

Recall from the class that if Alice sends Bob a plaintext and the hash of the plaintext, Bob will be able to validate the integrity of the message by hashing the plaintext received from Alice and comparing it with the hashed text sent from Alice. Similar approach is shown below. I can get the plaintext (TA's public key) from mailvelope server and its hash (fingerprint) from the shell. I can then hash the plaintext myself and compare it with the received hash from the shell.

1. When I obtained the TA's public key with the "gpg --keyserver" command in the shell, I was also given a key ID. The key ID is a substring of fingerprint, which is the hash of the public key.
2. Go to the mailvelope Key Server , by entering TA's email, I can get TA's public key.
3. Going back to shell, as I import this public key with gpg, it returns a key ID.
4. Verify if this key ID is the same as Key ID in step 1.

2. General steps for completing CTF

   I. Generate a key pair with gpg --gen-key. An verification email with a pgp message was sent to my andrew email. After decrypting it with my private key, I found a link to verify my keys. The public key of my key pair is now pushed to the server, whereas the private key is kept secret by myself.

   II.Enter my email in the CTF Shell prompts. Use my key pair of index 0 as this is the key associated with my andrew ID.

   III. The prompt asked me to prove my identity. Using my private key to sign my email serves this purpose(as presumably no one else in the world can encrypt a message that can be decrypted by my public key). Signing my email with gpg --clearsign my email and pass the base64 value of it into the shell

   IV. Now, I am asked to send TA an encrypted message. I first needs to obtain TA's public key with gpg --keyserver command. Then, I can send encrypted message to TA with gpg -a --sign --encrypt --recipient ta-14741-18631@gmx.com json.txt. This ensures the confidentiality of the message.

   V. Decrypt TA's message with my personal key: gpg -d flag.txt.asc

In summary, I first generated a key pair. Now, both TA and my public key are available on the server. Then, the CTF program authenticates my identification by asking me to sign a message with my private key. After authentication, we start communicating by encrypting our message with the receiver's public key and decrypting the received message with our own private key. I grew an in-depth knowledge of how PKI works in this problem.

3. Imagine you were interacting with us using email and had to send us an attachment, would you use PGP/MIME or PGP/inline, if the email client(s) you and we use support both methods? Justify your answer.

   I would use PGP/MIME. The crucial difference between these two mechanisms is that PGP/MIME encrypts message, metadata, and attachments, as a whole, whereas inline does it separately. Therefore, if an interceptor sees the PGP/inline message, he/she would easily identify the sections of the message, attachments, and the metadata. When we are doing security, it is never a good thing to leak any information to the evil people. Therefore, we would use MIME[1].

[1] https://protonmail.com/support/knowledge-base/pgp-mime-pgp-inline/

1 Q1.1 How to verify public key? **5 / 5**

✓ **+ 5 pts** **Metions 'use key ID/fingerprint to verify the integrity of public key'**

+ 0 pts Incorrect answer / no attempt

Question 1

CTF Username: IwishIcansing Flag: 818870150103756680902610936273 28
Fingerprint: 3BE49A0350C5384B7993C19936B601656B23B089

1. Based on this information, how do you verify that the public key you got from the key-server is valid, i.e.,that no one has modified it?

Recall from the class that if Alice sends Bob a plaintext and the hash of the plaintext, Bob will be able to validate the integrity of the message by hashing the plaintext received from Alice and comparing it with the hashed text sent from Alice. Similar approach is shown below. I can get the plaintext (TA's public key) from mailvelope server and its hash (fingerprint) from the shell. I can then hash the plaintext myself and compare it with the received hash from the shell.

1. When I obtained the TA's public key with the "gpg --keyserver" command in the shell, I was also given a key ID. The key ID is a substring of fingerprint, which is the hash of the public key.
2. Go to the mailvelope Key Server , by entering TA's email, I can get TA's public key.
3. Going back to shell, as I import this public key with gpg, it returns a key ID.
4. Verify if this key ID is the same as Key ID in step 1.

2. General steps for completing CTF

   I. Generate a key pair with gpg --gen-key. An verification email with a pgp message was sent to my andrew email. After decrypting it with my private key, I found a link to verify my keys. The public key of my key pair is now pushed to the server, whereas the private key is kept secret by myself.

   II.Enter my email in the CTF Shell prompts. Use my key pair of index 0 as this is the key associated with my andrew ID.

   III. The prompt asked me to prove my identity. Using my private key to sign my email serves this purpose(as presumably no one else in the world can encrypt a message that can be decrypted by my public key). Signing my email with gpg --clearsign my email and pass the base64 value of it into the shell

   IV. Now, I am asked to send TA an encrypted message. I first needs to obtain TA's public key with gpg --keyserver command. Then, I can send encrypted message to TA with gpg -a --sign --encrypt --recipient ta-14741-18631@gmx.com json.txt. This ensures the confidentiality of the message.

   V. Decrypt TA's message with my personal key: gpg -d flag.txt.asc

In summary, I first generated a key pair. Now, both TA and my public key are available on the server. Then, the CTF program authenticates my identification by asking me to sign a message with my private key. After authentication, we start communicating by encrypting our message with the receiver's public key and decrypting the received message with our own private key. I grew an in-depth knowledge of how PKI works in this problem.

3. Imagine you were interacting with us using email and had to send us an attachment, would you use PGP/MIME or PGP/inline, if the email client(s) you and we use support both methods? Justify your answer.

   I would use PGP/MIME. The crucial difference between these two mechanisms is that PGP/MIME encrypts message, metadata, and attachments, as a whole, whereas inline does it separately. Therefore, if an interceptor sees the PGP/inline message, he/she would easily identify the sections of the message, attachments, and the metadata. When we are doing security, it is never a good thing to leak any information to the evil people. Therefore, we would use MIME[1].

[1] https://protonmail.com/support/knowledge-base/pgp-mime-pgp-inline/

2 Q1.2 CTF flags + key fingerprint 15 / 15

- ✓ **+ 7 pts** Correct flag
- ✓ **+ 5 pts** Matching fingerprint
- ✓ **+ 3 pts** Steps taken

Question 1

CTF Username: IwishIcansing Flag: 818870150103756680902610936273 28
Fingerprint: 3BE49A0350C5384B7993C19936B601656B23B089

1. Based on this information, how do you verify that the public key you got from the key-server is valid, i.e.,that no one has modified it?

Recall from the class that if Alice sends Bob a plaintext and the hash of the plaintext, Bob will be able to validate the integrity of the message by hashing the plaintext received from Alice and comparing it with the hashed text sent from Alice. Similar approach is shown below. I can get the plaintext (TA's public key) from mailvelope server and its hash (fingerprint) from the shell. I can then hash the plaintext myself and compare it with the received hash from the shell.

1. When I obtained the TA's public key with the "gpg --keyserver" command in the shell, I was also given a key ID. The key ID is a substring of fingerprint, which is the hash of the public key.
2. Go to the mailvelope Key Server , by entering TA's email, I can get TA's public key.
3. Going back to shell, as I import this public key with gpg, it returns a key ID.
4. Verify if this key ID is the same as Key ID in step 1.

2. General steps for completing CTF

   I. Generate a key pair with gpg --gen-key. An verification email with a pgp message was sent to my andrew email. After decrypting it with my private key, I found a link to verify my keys. The public key of my key pair is now pushed to the server, whereas the private key is kept secret by myself.

   II.Enter my email in the CTF Shell prompts. Use my key pair of index 0 as this is the key associated with my andrew ID.

   III. The prompt asked me to prove my identity. Using my private key to sign my email serves this purpose(as presumably no one else in the world can encrypt a message that can be decrypted by my public key). Signing my email with gpg --clearsign my email and pass the base64 value of it into the shell

   IV. Now, I am asked to send TA an encrypted message. I first needs to obtain TA's public key with gpg --keyserver command. Then, I can send encrypted message to TA with gpg -a --sign --encrypt --recipient ta-14741-18631@gmx.com json.txt. This ensures the confidentiality of the message.

   V. Decrypt TA's message with my personal key: gpg -d flag.txt.asc

In summary, I first generated a key pair. Now, both TA and my public key are available on the server. Then, the CTF program authenticates my identification by asking me to sign a message with my private key. After authentication, we start communicating by encrypting our message with the receiver's public key and decrypting the received message with our own private key. I grew an in-depth knowledge of how PKI works in this problem.

3. Imagine you were interacting with us using email and had to send us an attachment, would you use PGP/MIME or PGP/inline, if the email client(s) you and we use support both methods? Justify your answer.

   I would use PGP/MIME. The crucial difference between these two mechanisms is that PGP/MIME encrypts message, metadata, and attachments, as a whole, whereas inline does it separately. Therefore, if an interceptor sees the PGP/inline message, he/she would easily identify the sections of the message, attachments, and the metadata. When we are doing security, it is never a good thing to leak any information to the evil people. Therefore, we would use MIME[1].

[1] https://protonmail.com/support/knowledge-base/pgp-mime-pgp-inline/

3 Q1.3 MIME vs. Inline 5 / 5

✓ **+ 5 pts** **States PGP/MIME is more preferable/desired. Includes supporting justification.**

+ 5 pts States both can be good, depending on the situation. Includes supporting justification.

+ 2.5 pts States inline is more preferable/desired. Includes supporting justification.

+ 0 pts No attempt

Question 2

CTF Username: IwishIcansing
Flag: 6929e20ffd868b4f16f5ffade6b19531
See my teammate's submission:
His name: Zhichao Chen
His andrew ID: zhichaoc

4 Q2: Hash Extension 30 / 30

✓ **+ 20 pts** **Attack worked / correct flag**

Understanding of state in hashing

✓ **+ 5 pts** **Hash state (h0..h7) itself is preserved in the hash output, so it can be used to continue the hash**

+ 2.5 pts Use of the original hash (h0..h7) but doesn't understand where it comes from

✓ **+ 5 pts** **Explained why include extra padding in the middle of the message since the original hash contained padding so the extension will be double-padded**

- 3 pts Didn't include flag and/or CTF username

- 1 pts Didn't include code in writeup

- 3 pts Didn't match pages

- 15 pts Didn't submit code at all

- 4 pts Hash extension doesn't replace existing data from the chunk, only adding new data

Question 3

Part A

Below are the steps taken to complete the program.

1. **Retrieve certificate Chain**: Starter codes already provided the certificate chain of a given domain name by sending the TLS request and parse the reply. Pop the first certificate as it is the leaf certificate. The certificates left in the chains will either be root CA or intermediate CAs.
2. **Create an X509 Store object** for our trusted certificate authority.
3. **Load CA root Certificates**: Import certifi library, the root CA certificates is stored at certifi.where. Therefore, I read the content of all root CA certificates into a very long string cacert with open file function. By using regular expression, I am able to parse each certificate and load the certificates to the X509 Store. Note: I should not load all ca root certificates as one file, because it might confuse the X509 store considering it as only one certificate. You can verify it by calling root_cert.get_subject(), and you will get only one root ca certificate's subject name compared to a list of subject names.
4. **Verify root and intermediate certificates**: Write a for loop under a try exception clause. In this for loop, loop through all root and intermediate certificates in the chain with a reverse orders (starting from the root CA). Setup the context with store_ctx = OpenSSL.crypto.X509StoreContext(store,inter_med)
   and verify each certificate in the chain by store_ctx.verify_certificate(). This verify function will validate each certificates' validity including expiration dates and if it or its parent certificate is in the store. If it is a valid intermediate certificate, the program will add it into the X509 store.
5. **Verify leaf certificate**: Again, using store_ctx.verify_certificate() for validation. However, we do need to worry about edge cases. The leaf certificate can be hello.pandu.xyz.com, but it should not be considered as a part of *.xyz.com. A typical case can be for example, my website is hello.pandu.xyz.com. I don't have a valid certificates so I download a certificate for *.xyz.com. If the certificate chain verification software is not robust, it might consider me as valid since my website domain name ends with .xyz.com. Taking considerations of this edge case, here are what I did:
    a. **Parse all DNS names from the leaf certificate**: Using a for loop to loop through all leaf certificate extensions and concatenate them into a string. Then, only extract the DNS part with regular expression.
    b. **Validate if there is an extra dot**: strip off the domain name with the wildcard from target domain. If there is no extra dot, then it is a valid domain.

Part B

The current code only has the capacity of adding certificates to the store. However, in the case if any company gets compromised, the original code we have will still validate it. Therefore, we need a function to periodically check the validity of all certificates in the store. If we find out any company (e.g. google.com) is compromised. We will remove the certificate and put the certificate in the Certificate Revocation List. Personally, I will be more conservative by revocating the entire certificate chain. Because since the root CA cannot ensure the security of its intermediate certificate, then the entire branch might not be that trustworthy.

5 Q3.A: Explanation of your validation approach 10 / 10

✓ **+ 4 pts** **Correct Trust Store Initialization**

Chain validation

✓ **+ 3 pts** **Correct approach - verify the chain root->intermediate->leaf**

+ 1.5 pts Checking against the root but not checking the entire chain in the root->intermediate->leaf order

✓ **+ 3 pts** **Correct Domain validation**

+ 0 pts No explanation of approach

- 2 pts Didn't match pages

Question 3

Part A

Below are the steps taken to complete the program.

1. **Retrieve certificate Chain**: Starter codes already provided the certificate chain of a given domain name by sending the TLS request and parse the reply. Pop the first certificate as it is the leaf certificate. The certificates left in the chains will either be root CA or intermediate CAs.
2. **Create an X509 Store object** for our trusted certificate authority.
3. **Load CA root Certificates**: Import certifi library, the root CA certificates is stored at certifi.where. Therefore, I read the content of all root CA certificates into a very long string cacert with open file function. By using regular expression, I am able to parse each certificate and load the certificates to the X509 Store. Note: I should not load all ca root certificates as one file, because it might confuse the X509 store considering it as only one certificate. You can verify it by calling root_cert.get_subject(), and you will get only one root ca certificate's subject name compared to a list of subject names.
4. **Verify root and intermediate certificates**: Write a for loop under a try exception clause. In this for loop, loop through all root and intermediate certificates in the chain with a reverse orders (starting from the root CA). Setup the context with store_ctx = OpenSSL.crypto.X509StoreContext(store,inter_med)
   and verify each certificate in the chain by store_ctx.verify_certificate(). This verify function will validate each certificates' validity including expiration dates and if it or its parent certificate is in the store. If it is a valid intermediate certificate, the program will add it into the X509 store.
5. **Verify leaf certificate**: Again, using store_ctx.verify_certificate() for validation. However, we do need to worry about edge cases. The leaf certificate can be hello.pandu.xyz.com, but it should not be considered as a part of *.xyz.com. A typical case can be for example, my website is hello.pandu.xyz.com. I don't have a valid certificates so I download a certificate for *.xyz.com. If the certificate chain verification software is not robust, it might consider me as valid since my website domain name ends with .xyz.com. Taking considerations of this edge case, here are what I did:
    a. **Parse all DNS names from the leaf certificate**: Using a for loop to loop through all leaf certificate extensions and concatenate them into a string. Then, only extract the DNS part with regular expression.
    b. **Validate if there is an extra dot**: strip off the domain name with the wildcard from target domain. If there is no extra dot, then it is a valid domain.

Part B

The current code only has the capacity of adding certificates to the store. However, in the case if any company gets compromised, the original code we have will still validate it. Therefore, we need a function to periodically check the validity of all certificates in the store. If we find out any company (e.g. google.com) is compromised. We will remove the certificate and put the certificate in the Certificate Revocation List. Personally, I will be more conservative by revocating the entire certificate chain. Because since the root CA cannot ensure the security of its intermediate certificate, then the entire branch might not be that trustworthy.

6 Q3.B: The flaw and potential solution 10 / 10

✓ **+ 10 pts Correctly Identifies CRLs/OCSP (important since CRLs are what are actually used in the real-world for revoking compromised certs)**

+ 6 pts Identifies stolen/compromised cert being abused / needing to be revoked, but not how it is done

+ 0 pts Empty / Incorrect