Network Security DSV/SU 2021

Laboration 2 (vs 1.2)
2021-02-23

Course Responsible: Yuhong Li
Lab Responsible: Stefan Axelsson

This memo; Stefan Axelsson (with quoted text from Ulf Noring).

Introduction

As we're not able to be in the physical lab this year we'll have to make do with what we have available.

This means that you'll perform the labs on your own computers at home. So you'll need access to a PC running Windows or Linux, or a Mac running MacOS. In addition you need the free tool Wireshark (i.e. just download and install it).

The computer also needs a network connection of some sort that allows you to access the internet. You also need some way of sending and receiving email.

I've tested these labs on a PC running Ubuntu 20.04 and Windows 10 (both with wired Ethernet and WiFi) but they should run on the other systems as well. If they don't then first look at variations in command line parameters between the different systems (i.e. read the documentation). Or you could install Ubuntu, or install/run it as a virtual machine under a free virtualisation environment, i.e. virtualbox.

Note that this lab is based on the memo for the previous version of the lab, i.e. the lab that takes place in our (physical) security lab. I recommend you also read that memo for the self study questions that are not repeated here.

Introduction

This lab focuses mainly on how to exchange messages over the internet securely.

"You are expected to have attended the lectures prior to doing the lab exercises, and done some precursory studying of the topics at hand before attempting to do the lab exercises. For this particular lab exercise, you are urged to read widely beyond the course text, and seek knowledge particularly from Wikipedia, or the Internet at large – an effective and efficient search engine will be your friend during this assignment. The material is readily available out there, in various forms. If you still stumble upon serious problems, after having performed an exhaustive preliminary search for knowledge, you may seek clarification from the course staff. Also, please DO NOT copy-paste commands from this file directly into your terminal window. If you do so, it will cause the issued command to not work properly because of the different symbol formats between Linux and Windows systems."

Handing in

”Please hand your report as a single PDF document. Preferably include the questions as well as the answers in order to speed up marking of the reports. Documents handed in late, in the wrong format, without names, or not using a standardised reference system will not be marked and will require a resubmission. Please note that your hand-ins will be run through a plagiarism checking tool. Make sure that you properly cite everything that is not your own work. Accepted referencing styles are for example IEEE, Oxford, Harvard, and Vancouver.”

”Please do not include any screenshots unless absolutely necessary or asked for in the instructions.”

Traffic Interception in Local Networks

Unfortunately we cannot really perform this part of the lab at home unless we have our own network (or correctly configured simulator – which is difficult to achieve). However, we have already used Wireshark to intercept network communications. When we’re on a switched network, we may not see traffic that is not directly addressed to us. There are ways of working around the limitations of switches however, using e.g. arp cache poisoning and other methods. Please read the explanations in the lab memo from last year.

For this year we’ll have to settle for the local interface capture we have already performed. From that it should be clear that with a combination of layer 2 and layer 3 (routing) attacks we can intercept and modify traffic to our hearts content (given that we have managed to penetrate the network) even if we do not have access to the actual computers on the network.

Question 1

> Check the common protocols that we use on the internet (or parts thereof), i.e. HTTP, HTTPS, ftp, telnet, ssh, pop3, imap, smtp
>
> Which of these are in clear text and which support some form of encryption. If you check *when* these protocols were defined/put into use can you see a trend developing over time with respect to cryptography?

Symmetric Encryption – Block Ciphers

“Based on the previous part of the assignment, it should be obvious that the ability of an attacker to gain access to the traffic of other users necessitates the adoption of appropriate protective measures. Cryptography has been traditionally employed as a method of preventing unauthorized entities from reading the contents of a secret message. Cryptography, besides providing message confidentiality, may also be used for other services such as integrity, authentication, non-repudiation, etc. Symmetric cryptographic algorithms date back to ancient times and are based on a secret key that only the allowed communicating parties are assumed to know.”

The original lab relied on ftp/http in the physical lab, but to make this work in our home setting we will instead focus on email as the only method of communication.

Question 2 (*optional*)

> If you have an email client that connects to a smtp-server directly (i.e. not a web mail service) try and capture the traffic with the smtp-server as you're sending an email (using Wireshark). Were you successful, why/why not?

Designate one of you as "Alice" and another as "Bob". (Two different email accounts of some form suffice.)

Alice decides to send a message (using email) to Bob. However, due to confidentiality concerns Alice decides to encrypt the message . "Through some other communication channel, both Alice and Bob have agreed to use a variation of the Caesar cipher known as ROT13. In order to encrypt and send the message using the ROT13 cipher" If you have a Linux/Unix/Mac OS system available you can issue the command:

```
$ cat message.txt | tr a-mn-zA-MN-Z n-za-mN-ZA-M > message_rot13.txt
```

If you're on Windows, then find a ROT13 command/tool, or do it by hand, it's not too onerous for short messages. (Brevity is after all the soul of wit.)

Question 3

> "What can be considered as the secret key of the ROT13 cipher? Does this cipher use substitution or transposition as encryption technique? Based only on the intercepted known ciphertext, is the attacker capable of retrieving the plaintext and if yes, how?"

"Due to ROT13 being a weak encryption algorithm, Alice and Bob agree on using AES, as a stronger alternative." You're going to use GPG (a free version of Philip Zimmermans PGP – Pretty Good Privacy encryption software).

If you're on Linux the command line client 'gpg' is a good choice. If you're on Windows 'gpg4win' is one good choice, and for MacOS there are several choices, pick one. (Check https://gnupg.org/download/index.html for alternatives)

Read the background on PGP on e.g. Wikipedia (https://en.wikipedia.org/wiki/Pretty_Good_Privacy) and the documentation accompanying your chosen GPG implementation. In the following I won't be giving detailed descriptions on e.g. command line switches or how to run the software, it's up to you to learn how your chosen implementation works. (And note that e.g. gpg4win also install a command line client that you may need to use at times as not all functionality we will use is exposed by the graphical interface)

Question 4

> So use symmetric encryption and encrypt the file containing only the phrase "This is a secret message" (w/o the quote marks) with the crypto AES256 and using "ASCII armour" to make the output printable. Use the key "secret" (again w/o quote marks). If your input file is named secret.txt you should end up with a file secret.txt.asc

The beginning of this file should look something like:

-----BEGIN PGP MESSAGE-----

jA0ECQMCuZNP6HeuwIPH0lEB6V53iuq4Y5wAOJn05geEefUXqrMeudF4OB5R0ZBZ

and then continue with a number of lines.

Include the full encrypted PGP message block in your report.

Now try the same thing one more time exactly like before. Did you get the same result? No? That's because PGP/GPG randomizes the "start" of the encrypted file so that the same message, with the same key encrypted twice, doesn't lead to the same crypto text. (This is to thwart many different cryptographic attacks).

Question 5

With the encryption above, what is the size of the generated key? (Both to the algorithm and the effective key length?) Can you identify any security problems with the above approach?

Message Integrity

"The process of encrypting a message with a cryptographic algorithm, as in the previous part, can surely improve the protection level of the confidentiality of the message against third parties. Depending on qualities such as the strength of the algorithm used, the key length, the encryption mode, the key (and IV) randomness when encrypting multiple messages etc. the communication can achieve a varying level of assurance with respect to the confidentiality of the messages. However, the integrity of the transmitted messages if overlooked may enable attackers to modify messages so as to disrupt the communication or even more importantly modify the contents of the message in an intended way. Hash algorithms due to their property of producing a completely different output upon even slight changes of the input are often employed in message integrity solutions."

"In this section, we will focus on messages whose integrity needs to be ensured without any confidentiality concerns."

Using the secret.txt file from above, calculate the SHA256 hash sum of the file. (On windows there is actually a built in tool since Win 7 that can do this: certutil, see below. On linux the sha256sum command works.)

```
C: certutil -hashfile secret.txt SHA256
```

Question 6

What are the hash values of the file? What are the hash algorithms' output sizes? Briefly explain why a larger output size of a hash algorithm is preferable." Note that most/all editors/operating systems add line-feed characters to a file. And these are different for different operating systems The file I encrypted contained a single carriage return character (ASCII CR) at the end.

Now let's communicate in a way that allows us to check that the message has not been tampered with. Construct a short text file. Calculate the SHA256 checksum of the file. Attach the file to an email message, and write the SHA256 checksum in the email as well. Send it. Have the receiver verify that the SHA256 signature matches.

Now assume the attacker manages to change the message. Make a small change (one character) in the message and check the new checksum against the old.

Question 7

> Do the checksums match after the change? How different are the checksums? Also, is the protocol you're using (i.e. sending the checksum in clear text with the message) secure? Why/why not?

Extra/optional reading:

> "As an alternative, Message Authentication Code (MAC) algorithms combine the input with a key in order to produce a tag that is used to ensure the data integrity of the message. The involvement of a key protects also the authenticity of the message assuming that only the necessary communicating parties have knowledge of it. MAC algorithms are implemented commonly using either block ciphers (e.g. CBC-MAC) or cryptographic hash algorithms (e.g. HMAC).
>
> In the case of cryptographic hash algorithms though, caution is needed to avoid custom schemes of how the key and the algorithm are employed. An insecure but common approach is to concatenate the key along with the message and calculate the hash of this result (H(k||m)). Flickr has been a notorious example of using this erroneous approach in enabling its API service consumers to provide authenticity in their API requests (e.g a third party page uses the Flickr Authentication API to authenticate users)"
>
> "The problem with this approach is that hash algorithms such as MD0-MD5 and SHA0-SHA2 are known to be vulnerable to length extension attacks due to being based on the Merkle-Damgård construction model. As an example, the MD5 algorithm splits the given input into 512bit blocks (padding the last one if needed) and using a 128-bit stream as a chaining value (the 128-bit result of the first block is mixed with the second input block etc.) as shown in Figure I. The last chaining value comprises the actual hash algorithm result. However, the last value can be used as a chaining value for a new input block or multiple blocks (padded properly if needed) to produce a valid new hash value thus effectively calculating the valid hash value of H(key||original_data + padding + added_message
> +additional_padding) without knowing the key."

Public Key Cryptography

"In contrast to symmetric cryptography where a single secret key is shared between the communicating parties, public key cryptography uses a pair of keys consisting of a public

and a private key providing a much more efficient solution to key management issues. The two keys are tied with mathematical relations such that if the one key is used for encryption then the other can only be used for decryption. The private key is supposed to be kept confidential by the owner while the public key is assumed to be freely distributed to the public (e.g. posted on a website). The mathematical relation of the two keys must not allow other parties to infer the private key of an entity based on knowledge of the public key."

Use GPG to generate a public/private keypair with one of your identities. I.e one for Alice, Bob and Charlie. You can generate "proper" keypairs with your real identities, and then substitute your names for Alice and Bob in the following, but you don't have to. In the case you make "fake" keys, just make sure you don't publish them to a key-server.

Question 8

> What key-size and key-type did you chose? I.e. what type of public key cryptography algorithms. Why?

Question 9

> Are there any security issues with the secret key file you generated? Why/why not?

Make a short text file (Say it's from Alice in the text file) and encrypt it with your (Bob's) public key. Send it (via email, or copy on the same computer, or copy via network share to another computer) to Bob. Have Bob decrypt it. It is a good idea to ASCII-armour the message.

Question 10

> How did you make sure that Alice really used Bob's public key? If you cannot check Bob's public key, how could someone that has complete control of the communication channel between Alice and Bob perform a man-in-the-middle attack?

Now try the same thing again, but first have Bob confirm their key by telling Alice their key fingerprint (on a channel that cannot be controlled by a man-in-the-middle attacker).

Question 11

> You've only done encryption so far. Alice can be certain that only Bob can read the message (if precautions against man-in-the-middle attacks have been taken), but can Bob be certain that the message comes from Alice? Why/why not?

Public Key Cryptography – Digital Signatures

“As in the symmetric cryptography environment, any message transmitted over an insecure channel (either in plain or encrypted form depending on the case) is susceptible to modification attempts by intermediate parties. Similarly to the symmetric case, hash algorithms can be employed as part of solutions that ensure the message integrity due to their collision resistance and irreversibility features. However, sending the message digest in clear text does not provide for message authenticity since an intercepting entity can change both the message and the message digest so as to match. In the RSA digital signature, the message digest of a file is encrypted with the sender’s private key and verified by the recipient using the sender’s public key.”

So now take the message you made before and *clear sign* it with Alice’s key, i.e. not using any encryption. Send it to Bob and have him verify that the signature is correct.

You have the same problem as last time, i.e. how does Bob know that he really has Alice’s key to make the verification. Read up on GPG/PGP’s “Web of trust”. Note that when you’re using normal TLS/SSL i.e. surfing the web using https there is a tree of “trust”, called a PKI (Public Key Infrastructure) where your web browser is delivered with a number of root certificates, that have been used to sign other certificates that sign other certificates that finally sign the name of the web site you’re visiting.

There have been many problems with this trust hierarchy; one is that if one root is compromised, it can sign for any site, anywhere. Even if the owners of the site have had no dealings with the operator of that signing agency/company and have in fact contracted with another signer. This has led large operators like Google, who develop their own web browser (Chrome) to implement certificate pinning. That means that Chrome ignores the PKI for Google’s own domains, instead relying on a hard coded certificate built in to Chrome “from the factory” to validate Google’s own domains. So subverting a higher level certificate authority would not work in that case. Google know who they trust to sign for their domains.

However, PGP predates the introduction of functioning PKIs so it is based on a “web of trust” model, where users can sign for other users keys. And PGP/GPG users can assign different levels of trust for keys *signing other keys*. That is to say; you can as a user decide to trust (say) Bob’s key. But not put too much trust in Bob when he signs other keys. (Trust is not transitive. While Bob may be trustworthy, he may also be naive. Just because he trusts someone, that doesn’t necessarily mean that you trust them).

So here you need three entities: Alice, Bob, and Charlie. (Make a Charlie if you don’t have one already).

Now clear all trusted keys from both Alice and Bob. I.e. they should not trust any other key. Have Charlie sign both Alice and Bob’s keys. Then introduce Charlie’s key to both Bob and Alice.

Now sign and encrypt a message from Alice to Bob. Get the respective keys and check Charlie’s signature, i.e. trust Charlie to sign for other’s with a high degree of certainty.

<u>Question 12</u>

> Do Charlie have to be present when Alice and Bob communicate with each other? Do they have to communicate with Charlie beyond having trusted his key to sign for other keys? Why/why not? Were there any problems to your understanding what was going on, and how the web of trust works? What/why? Do you have any suggestions for improvements in that regard?

You should now have gotten a feel for the intricacies of public key signature hierarchies and how one can use them to make (more) certain that one is communicating with the right person/entity, and that only the intended recipient can read the message. This is used not only on when surfing the web, but also in banking applications, computer hardware that only runs signed operating systems (the company's public key is stored in ROM/HW in such a way that it cannot be modified, that is used to check the signature of the firmware/OS before it is run. As only the public key is exposed, an attacker cannot derive the secret key used for signing).

Also note that while PGP/GPG was the only game in town for many years, now usage has fallen by the wayside. Secret/signed communication is today done mainly using apps such as Telegram/Signal etc. or services like Hushmail. PGP/GPG turned out to be too difficult to use for ordinary users (See e.g. "Why Johnny can't encrypt" and "Why Johnny still can't encrypt") and integration in popular email/messaging software was always lacking. Today, while still useful to the initiated, it's mostly for historical reference, and useful for illustrating public key signing/encryption and trust relationships.

That concludes lab 2