Unit 7: Security

1. Firewalls & proxies

Learning objectives

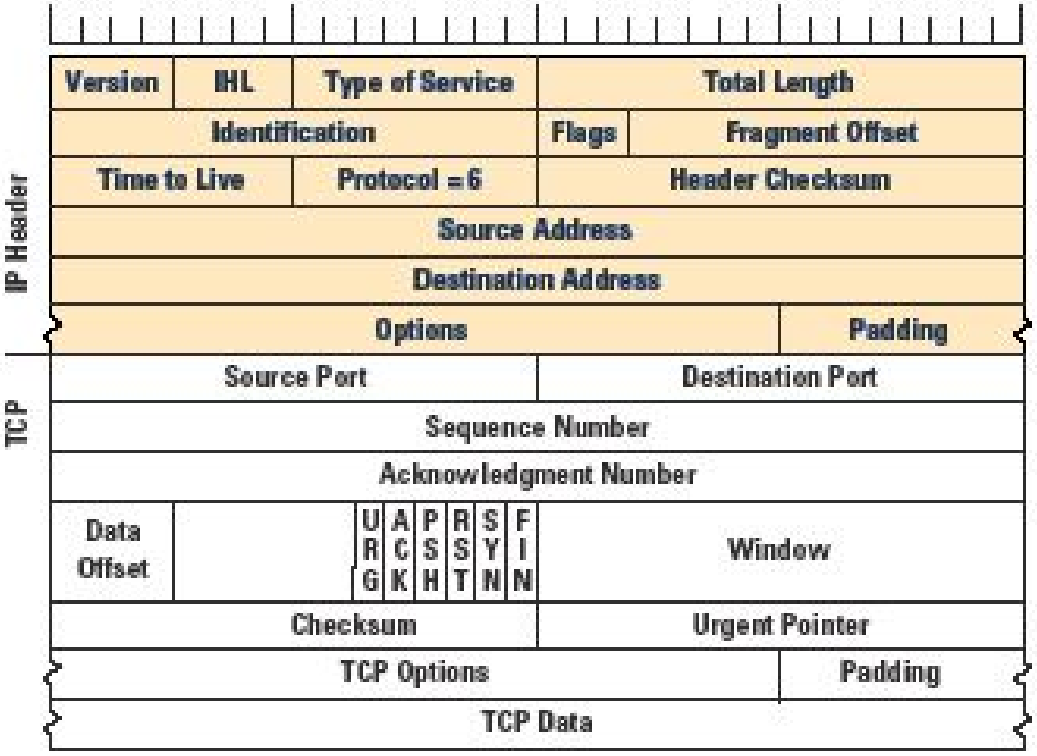
1.4.2 show understanding of how data are kept safe when stored and transmitted, including:

- use of firewalls, both software and hardware, including proxy servers

Firewalls

To understand how a firewall works, you first need to understand how the TCP/IP network protocol works (you all know how that works, remember? We did it in just the previous unit!) Ok, in fairness, here is a little more detail as it is pertinent to firewalls….

This is a diagram showing all the fields & data contained within a TCP/IP packet (one TCP/IP message). The "teeth" across the top represent bytes (so, for instance, the "version" field is 4 bytes in size).



(source)

Firewalls are just software (if you have a "physical" firewall you bought from a store, it is just a simplified computer running dedicated firewall software). Firewall software can be customised by their owners (you) to allow traffic to pass in or out of them based upon "rules" which can be assigned to any combination of these fields.

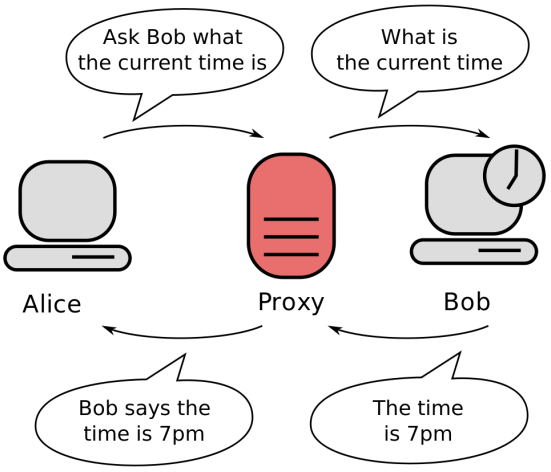
For example, you can create a firewall rule to allow no incoming traffic where the source address is "<insert address of person you don't trust>".

Most firewall rules focus on one or more of the following fields:

- Source address
- Destination address
- Source port
- Destination port

Remember the port number designates the software that will be receiving the packet. So if you are running a webserver, you might choose to block all traffic except that destined for port 80 (http) or 443 (https).

Proxy servers



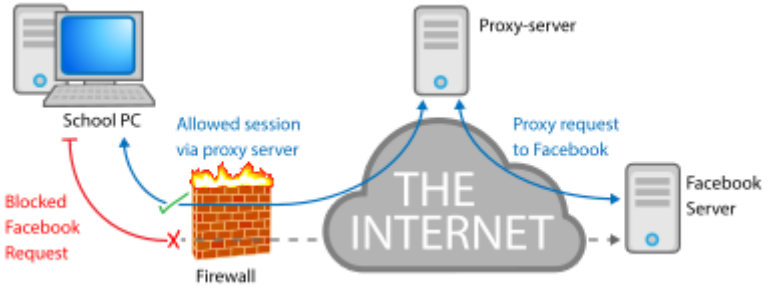
A proxy server is just a server that will receive requests, and forward those on to servers for their reply, before sending the reply to the computer that originally asked it. The wikipedia image opposite gives a good illustration of the general idea.

Proxy servers are commonly used where the person using the proxy server ("Alice") has direct network visibility of the proxy server, but is unable to directly access the desired target ("Bob"). The proxy server, however, can see "Bob" so is able to access it's services on behalf of "Alice".

This commonly occurs when computers are operating on different IP address subnets, such as if Alice is on a private network. This is what happens on home internet connections, or when you access the internet at school. Due to a global shortage of IP addresses, the school would only have 1 external real-world IP address, through which all the staff and students access the internet. Internally, any laptop connected to the school wifi would be issued a private network address (IP addresses starting with 192.168.X.X or 10.X.X.X are designated for private internal network use and are not valid global addresses).

The proxy server has both an internal IP address (192.168.X.X) and a real world IP address (whatever the ISP issues), so it is visible to both internal computers and the external internet, so is able to forward requests to the internet on behalf of internal computers.

As an aside, proxy servers are also commonly used to bypass filters/blocks because your traffic is being directed to the proxy server address instead of the banned address directly.



From a security perspective, be aware, because you are sending your traffic to a proxy, and it is receiving the replies on your behalf, it is able to see the content of the messages being exchanged between you and the ultimate destination. Yet another reason by using encrypted traffic is important.

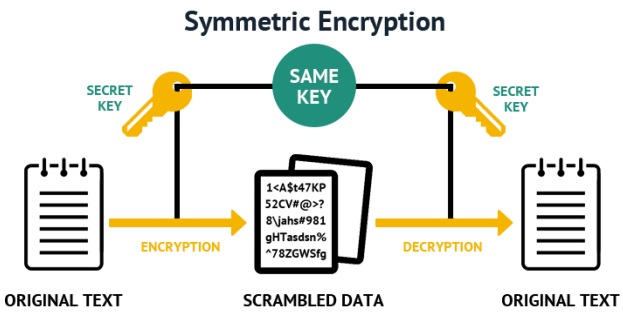
2. Symmetric & asymmetric encryption

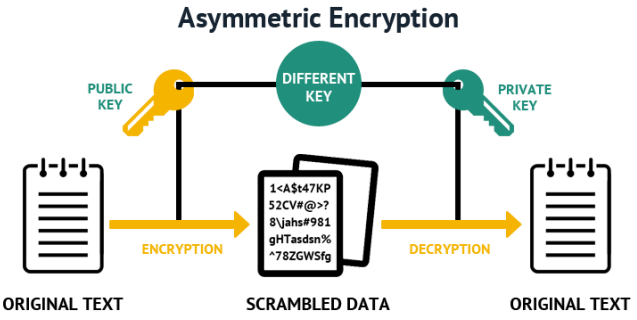
Learning objectives

- use of symmetric encryption (plain text, cypher text and use of a key) showing understanding that increasing the length of a key increases the strength of the encryption

Discussion & learning items

Two types of encryption...





	Symmetric Encryption	Asymmetric Encryption
The Basics	Symmetric encryption makes use of a single key for both encryption and decryption.	Asymmetric encryption uses different keys for encryption and decryption. It applies a public key for encryption, while a private key is used for decryption.
Algorithms	Symmetric encryption using algorithms like 3DES, DES, RC4, AES and QUAD.	Whereas, Asymmetric encryption uses RSA and ECC algorithms to create the public and private keys.
Performance	While Symmetric encryption is fast in its execution.	Asymmetric encryption tends to be slower in execution as a result of more complex algorithms which come with a high computation burden.
Purpose	Symmetric encryption is utilized for bulk data transmission.	Asymmetric encryption is mostly used for securely exchanging secret keys.

(source)

Symmetric encryption might seem logical enough, some kind of algorithm that encrypts/decrypts when you both know the secret key. We've done similar (at an extremely simple level) with Caesar cipher etc. But what about asymmetric encryption? How can it possibly work when only one person knows the "key"?

The basis of asymmetric encryption is calculations that are "easy" in one direction but very difficult in reverse. For example compare:

- What is 7919 multiplied by 6323?
- What two prime numbers are the factors of 50 071 837?

One of the above you could easily do, the other you'd probably throw your hands up in despair!

Modern cryptography is currently using at least 512 bit numbers. That's numbers in the range of 0 to 1.34 x 10^{154}. Try brute forcing the prime factors of a number at that scale!

This is the basis of the idea of public and private key cryptography. One number is public, one number is kept private. You need both numbers to decode.

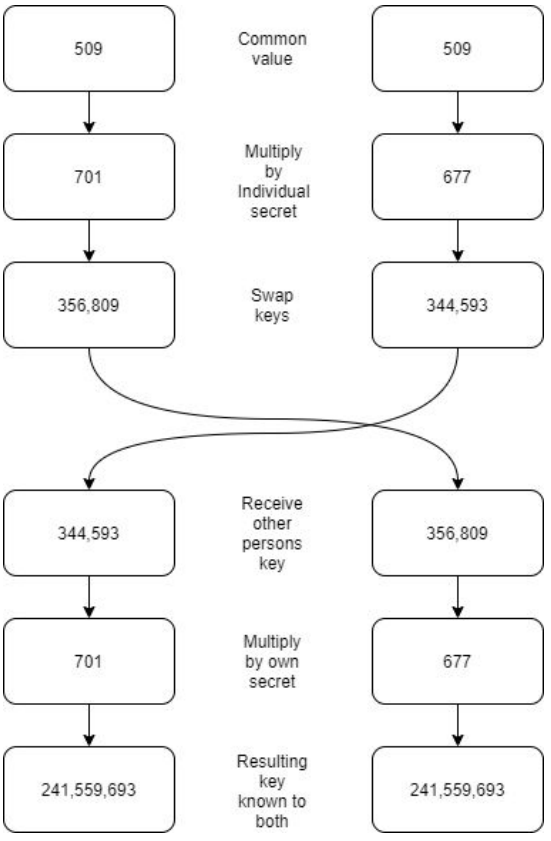
Diffie Hellman key exchange

https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

Diffie-Hellman key exchange is a method by which two parties can maintain their own private key and yet communicate together with a common secret key.

It is used by a number of asymmetric encryption systems.

The wikipedia article provides a good illustration using colours. To turn this into something more resembling crypto, I'll recreate it using large prime numbers…



Further practice & resources

This is an example of using symmetric encryption with the pycryptodome library.

- To install on your local machine, pip install `pycryptodemo`
- To install on repl.it, install via the packages icon ---------->

```
# Based on examples provided here
# https://pycryptodome.readthedocs.io/en/latest/src/examples.html
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def generate_key():
    # Your key must be 16, 24 or 32 bytes long
    key = get_random_bytes(32)
    return key

def encrypt( message, key ):
    byte_array = message.encode("utf-8")
    # If we were given a string, convert it to a bytes array
    if type(key) == str:
        key = key.encode("utf-8")
    if len(key) not in [16,24,32]:
        print("Key must be 16, 24 or 32 bytes long")
        exit()
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(byte_array)
    return ciphertext, tag, cipher.nonce

def decrypt( encrypted, key, tag, nonce):
    key = key.encode("utf-8")
    cipher = AES.new(key, AES.MODE_EAX, nonce)
    byte_array = cipher.decrypt_and_verify(encrypted, tag)
    return byte_array.decode('utf-8')


def main():
    while True:
        task = input("(e)ncrypt, (d)ecrypt or (q)uit?")
        if task == "e": # Encrypt
            message = input("Your secret message: ")
            key = input("Your encryption key: ")
            ciphertext, tag, nonce = encrypt(message, key)
            print("You can safely send these...")
            print("Cipher text: "+ciphertext.hex())
            print("Tag:         "+tag.hex())
            print("Nonce:       "+nonce.hex())
        elif task == "d": # Decrypt
            ciphertext = input("Encrypted text (hex): ")
            tag = input("Your ciphertext tag (hex): ")
```

```
            nonce = input("Your ciphertext nonce (hex): ")
            key = input("Your encryption key: ")
            # Convert from hex strings to byte array
            ciphertext = bytearray.fromhex(ciphertext)
            tag = bytearray.fromhex(tag)
            nonce = bytearray.fromhex(nonce)
            message = decrypt(ciphertext, key, tag, nonce)
            print(message)
        elif task == "q": # Quit
            return()

main()
```

Here is a message for you to decrypt. It has a 16 byte key that should be relatively easy for you to guess.

Cipher text	`4f7658764d3a538d3a8d8051187199345d9aa38bc6755d40861dc1`
	`3d096859e1415ebbb69216d7e9608374fd936b2d06785242413e04`
	`B4e1d50bdb32adec8230d6ea889707ab0ba19b20515336bc44c0f0`
	`7ed54126c7bb473670b4c0e64eefce56dbaa5ad4c8f07aecc4b6fa`

Tag	`70927dcb435058b6a94a280cfbff4fff`
Nonce	`16ea29729cda6a86c986ec3dad0b147a`

(note to self: since it took me an hour to remember it last time, the key is in my stc keep notes!)

3. Hashing algorithms

Learning objectives

- Hashing algorithms are a critical tool for ensuring the integrity of data transmitted

What is a hash algorithm?

Hashing Algorithms and Security - Computerphile (Tom Scott)
https://www.youtube.com/watch?v=b4b8ktEV4Bg (8:11)

The Python library containing the hash functionality is `hashlib`.

Commonly known hashing algorithms include:

- md5 - broken
- sha1 - broken
- sha2 - partially broken
- sha3 - not broken

To see just how broken md5 and sha1 are, try the following website:

- https://project-rainbowcrack.com/table.htm
  (yes, you can even download full reverse lookups for md5 and sha1 if you fancy wasting the bandwidth and disk space)

```
import hashlib

original = input("Message: ")

# Convert string to array of bytes
byte_array = original.encode('utf-8')

md5 = hashlib.md5(byte_array).digest()
sha1 = hashlib.sha1(byte_array).digest()
sha3_256 = hashlib.sha256(byte_array).digest()

print("md5 = "+md5.hex())
print("sha1 = "+sha1.hex())
print("sha3_256 = "+sha3_256.hex())
```

The field of password hash algorithms is constantly changing. If you find yourself implementing a system that will store password hashes, you should check the current advice at the time of programming. Here is a good summary of the situation as of 2019,
https://medium.com/analytics-vidhya/password-hashing-pbkdf2-scrypt-bcrypt-and-argon2-e25aaf41598e

4. Security protocols

Learning objectives

- use of security protocols such as Secure Socket Layer (SSL) and Transport Layer Security (TLS)

Discussion & learning items

What is SSL and TLS?

This article provides a good overview
https://www.csoonline.com/article/3246212/what-is-ssl-tls-and-how-this-encryption-protocol-works.html

Post article questions:

1. How are SSL and TLS related?
2. What is the role of an SSL certificate?
3. Why must the issuer of SSL certificates be trusted by both parties?
4. What version of TLS should you be using now?
5. How can you determine what protocol you are using on a given website (not in the article, google for instructions on how to check this with Chrome or your browser of choice)

SSL/TLS in action

Download Wireshark (I would put money on this not working from the school network, download at home. If at school, I'll have a copy on a USB)

https://www.wireshark.org/download.html

On Windows 10 machines you will also need to install Win10pcap

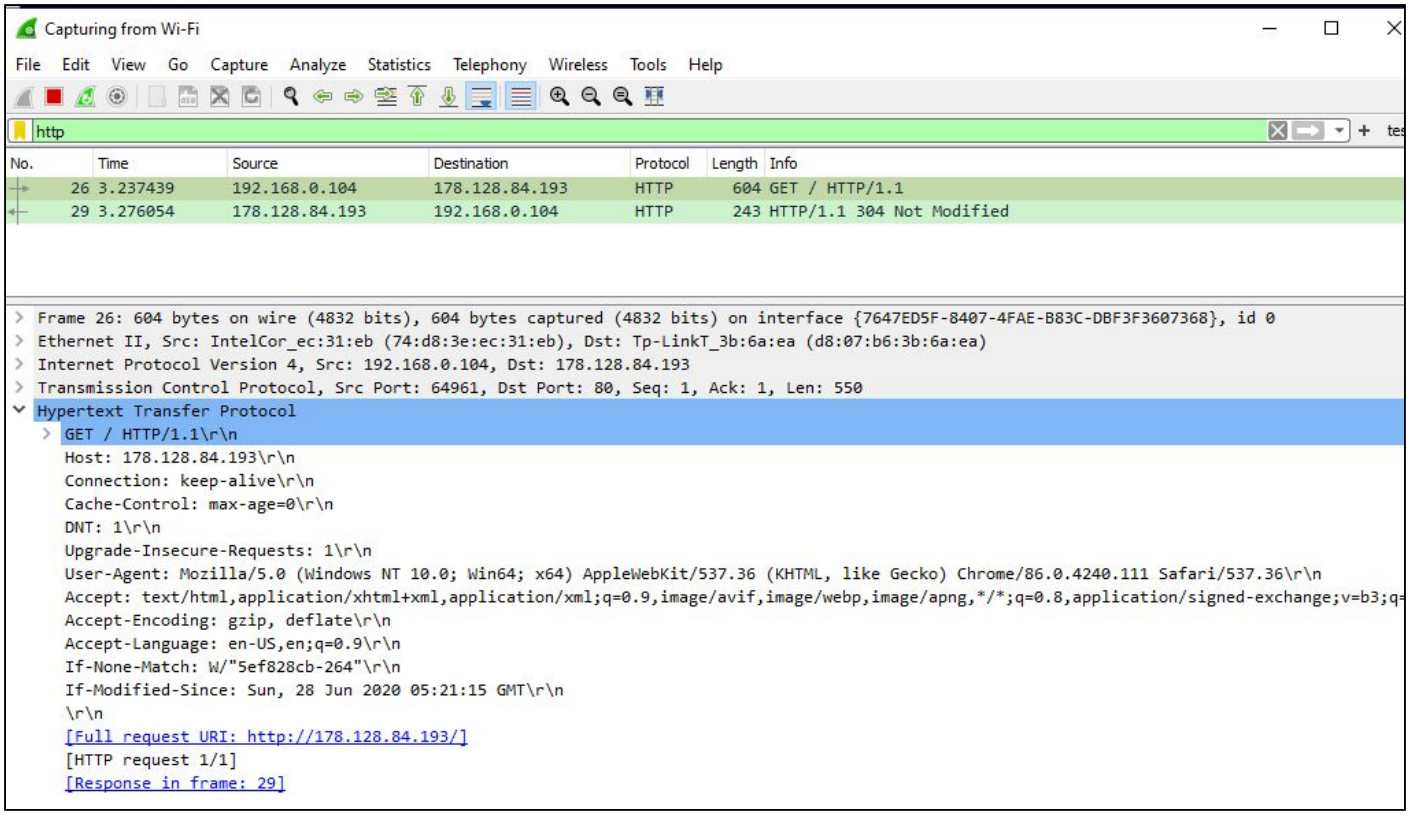
http://www.win10pcap.org/download/

Make sure you start Wireshark with Administrator privileges (right click on the start menu icon, "Run as Administrator")

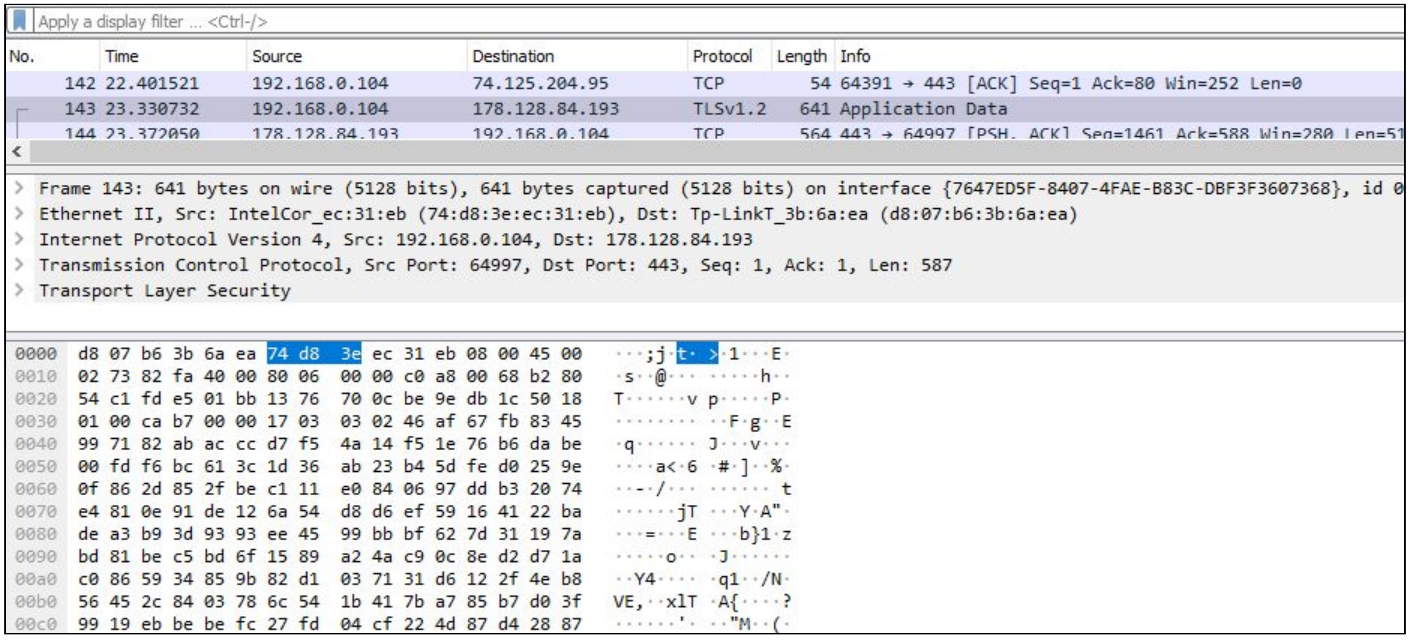
Have a go at connecting to various websites in HTTP and HTTPS and compare how much you can see.

If you wish to use your Raspberry Pi, install and research into using the tool, `tcpdump`.

Browsing using HTTP without the “S”



Browsing using HTTPS



5. Understanding security threats

Learning objectives

- 1.4.3 - show understanding of the need to keep online systems safe from attacks including denial of service attacks, phishing, pharming

Discussion & learning items

Terminology:

- Malware
    - Viruses
    - Spyware
- Social engineering
    - Phishing
    - Pharming
- Hacking
- Cracking

Term	Definition	Best practices to protect yourself as a user	Best practices to protect your product as a developer
Malware			
Virus			
Spyware			
Social engineering			
Phishing			
Pharming			
Hacking			
Cracking			

6. Threat mitigation

Learning objectives

- 1.4.1 - show understanding of the need to keep data safe from accidental damage, including corruption and human errors
- 1.4.1 - show understanding of the need to keep data safe from malicious actions, including unauthorised viewing, deleting, copying and corruption
- 1.4.4 - describe how the knowledge from 1.4.1, 1.4.2 and 1.4.3 can be applied to real-life scenarios including, for example, online banking, shopping

Discussion & learning items

Validate your inputs!

Heart bleed is the name given to a vulnerability found in the OpenSSL library in 2014. OpenSSL forms a key part of the cryptographic library used by TLS.

Randall Munroe, the comic artist for XKCD did an excellent illustration of how the bug works here:

- https://www.explainxkcd.com/wiki/index.php/1354:_Heartbleed_Explanation

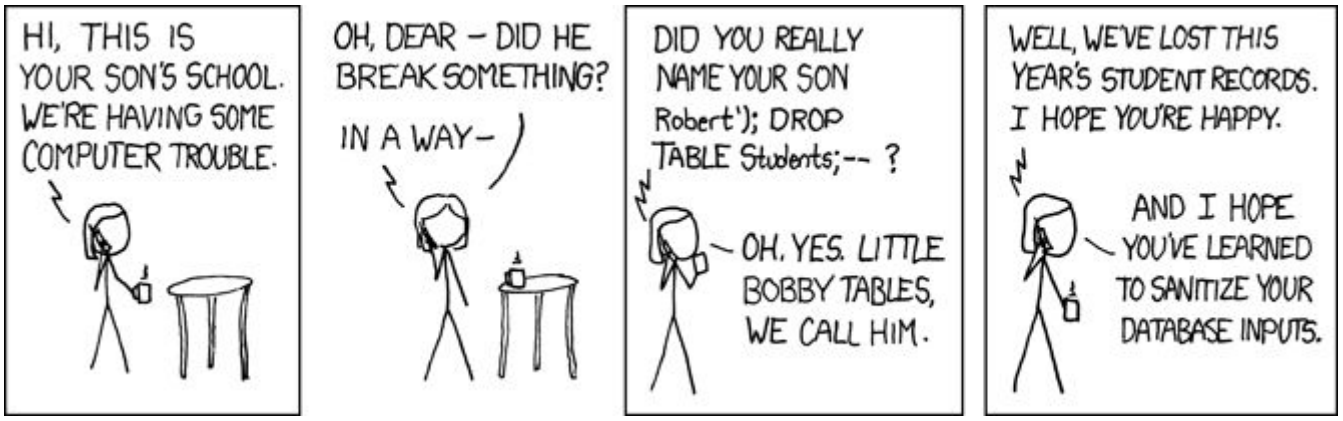
For details on the different real-world ways the vulnerability could expose information, read here:

- http://www.pabr.org/heartbleedtax/heartbleedtax.en.html

The bug is a great illustration of why input validation is such a critical part of security programming!

Validate your inputs!, part 2

This is a famous computing cartoon about the importance of validating your inputs, https://xkcd.com/327/



Remind me to revisit this when we look at databases. Here is a brief overview of the scenario though.

Databases are commonly used for storing app information. They are generally manipulated through a programming language known as SQL. Typically your main program (written in Python or whatever) would send SQL commands to the database to update/retrieve data as desired. The complication comes when you are using information provided from the user within those SQL commands. In the example of the comic above, it might be asking the database to "create a new student called Bobby".

In a badly designed program, one way of doing this might look like…

```
sql = "INSERT INTO Students VALUES ('" + first_name + "', '" + family_name + "')";
```

In the above scenario, the following was entered as Bobby's first name.

```
Robert'); DROP TABLE Students; --
```

If you substitute what the user supplied for "first_name" without validating it, this could end up as…

```
sql = "INSERT INTO Students VALUES ('Robert'); DROP TABLE Students; --', 'Doe')"
```

SQL allows multiple commands in one string when separated by semicolons, so this actually turns into three commands!

1. `INSERT INTO Students VALUES ('Robert');`
2. `DROP TABLE Students;`
3. `--', 'Doe')` note the -- is a comment operator so this line will be ignored

The key being instruction #2, DROP TABLE which is the command to delete the entire Students table.

Never trust the user! Always validate your inputs!

Malware protection tools

Antivirus software scans a file, program, or an application and compares a specific set of code with information stored in its database. If it finds code that is identical or similar to a piece of known malware in the database, that code is considered malware and is quarantined or removed. (quoted)

Don't use USB devices you don't trust

https://foundersec.substack.com/p/usb-keys-will-end-you

Backups

Follow the 3-2-1 rule of backups, so if the worst happens, you can get back to where you were.

With automated synchronising tools such as Google Drive, Microsoft OneDrive, DropBox etc there is no excuse to not have current, regular backups of your files. But the cloud copy by itself is not enough!

7. Best practices for authenticating users

When it comes to identifying and authenticating users on secure systems, the methods are typically broken into these three categories:

- Something you know - eg: a password, mothers maiden name, favourite movie
- Something you have - eg: something physical such as your phone (Authy, SMS, Email)
- Something you are - biometrics eg: fingerprint, retina scan, face recognition

It is commonly accepted that to properly authenticate a person, two of the three categories should be used.

Passwords

To be honest, passwords are no longer considered good practice. There are just too many ways for things to go wrong. Read the Microsoft Whitepaper on going passwordless.

- https://www.microsoft.com/en-us/security/business/identity/passwordless

If you do need passwords, follow the best practice guidelines on what passwords to accept

- https://www.troyhunt.com/passwords-evolved-authentication-guidance-for-the-modern-era/

Watch: How not to store passwords https://www.youtube.com/watch?v=8ZtInClXe1Q

- Don't use clear text
- Don't use the same encryption key for all passwords
- Don't hash just the raw password
- Don't use the same salt for all passwords
- Don't use an insecure hash algorithm!

Best practice to store passwords

- Do use different, randomised salt per user!
- Do use a modern, secure hash algorithm
- Do let others handle the difficult task for you if you can (eg: Sign in with Google).

As a user, some password advice:

1. Do not use the same password for more than one service.
2. Do use two-factor authentication if available (such as Authy)
3. Do use a password manager
4. Passwords are actually quite a weak form of security and are being constantly compromised. Check to see if any of yours have been compromised without being aware of it. Enter your email addresses into the https://haveibeenpwned.com/ website (safe to use as all you enter is an email address not any actual passwords - that would not be safe!)

One time codes

If you don't use the Authy app, I highly recommend you set this up for any service that offers it.

As a programmer, you should also add OTC to any service you create. It is trivially easy to add (as you'll see in the exercise) and it will give your users a much better security experience. If you only had one authentication tool, use this instead of a password.

How do these work? The app author creates an encryption key. This is hashed and shared with the user via the QR code you scan to load the app into your Authy.

When you want to login using a one time code, both the server and the Authy app look up the current time and add it to the encryption key to generate a hash. If both arrive at the same hash, then it means the client with the Authy app has the correct encryption key and is therefore authorised to proceed.

Biometrics

I couldn't find an easy to use Python library that incorporates the fingerprint reader found in Windows laptops. This library was the best I could find. I'm sure better solutions will appear soon.

https://github.com/luspock/FingerPrint

Programming exercise

Here is a proof of concept for getting the one-time-passcodes to work in Python.

Library	https://github.com/pyauth/pyotp
To install	`pip install pyotp qrcode`

```
import pyotp
import qrcode

def register_user(key):
    auth = pyotp.TOTP(key)
    # Generate QR code data
    qr_data = auth.provisioning_uri(issuer_name='Mr B secure app')
    print("QR Data: ",qr_data)
    # Render a QR code image
    img = qrcode.make(qr_data)
    img.save("qrcode.png")      # Save to PNG image file
    img.show()                  # Display image in a preview window
    return qr_data

def verify(key, code):
    auth = pyotp.TOTP(key)
    return auth.verify(code)

# Cheat mode: Get the current valid code
def cheat(key):
    auth = pyotp.TOTP(key)
    print( "The phone code would be ", auth.now() )

#### MAIN
# Warning - Anyone who knows the key can generate codes for your "secure app"
# Typically this key should be customised per-user
encryption_key = 'mysupersecretpassword'
while True:
    choice = input("(r)egister, (v)alidate, or (q)uit? ")
    # Register a new user
    if choice == "r":
        data = register_user(encryption_key)
        print(data)
    # Verify an existing user
    elif choice == "v":
        print("Open Authy on your phone and get the validation code...")
        confirm = input("Code shown on app: ")
        if verify(encryption_key, confirm):
            print("Your code is correct")
        else:
            print("Access denied!!!")
    # Quit
    elif choice == "q":
        break
```

8. Ethics revisited

Learning objectives

1.5 Ethics
Candidates should be able to:

- show understanding of computer ethics, including copyright issues and plagiarism
- distinguish between free software, freeware and shareware
- show understanding of the ethical issues raised by the spread of electronic communication and computer systems, including hacking, cracking and production of malware

Introduction

Most tools tend to be neutral. They can be used for good and malicious purposes. This includes Wireshark, encryption, hashing algorithms and so forth.

It may seem obvious to say it, but not create or distribute malware (viruses, spyware); or seek to crack into others systems!

Privacy issues

As a user:

- Awareness of how data is handled at the program level should make you a more cautious user of technology.

As a developer

- You have an ethical obligation to treat your user's data with respect. Secure it as you would want your own data secured. Eg:
    - Encrypt the database
    - Firewall settings to be strict,
    - Limit which members of the developer team have access to the customer data,
    - Log all access to customer data
- How much data do you legitimately need to be collecting?
- Be aware of any data protection laws, privacy laws that you may have to comply with.

Anonymity issues

- People behave differently online when they think they are anonymous. Eg: Trolling.
- If you are going to develop software that allows for anonymous interaction between users, you need to consider what safeguards to put in place.

Software licensing

Be aware of copyright laws and intellectual property regulations in your jurisdiction!

What is the difference between these software licensing models?

- Commercial
- Freeware
- Shareware
- Opensource

(this is a very common question in the exams)