KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Problem Solving and Engineering Design part 3

ESAT2A2

Jarle Braeken (r0998124)
Arthur Cukier (r0976603)
Pieter Deferme (r0995734)
Robin Derikx (r0978208)
Willem Hendig (r0995944)
Giel Swenters (r1006315)

Crack the Wi-Fi

PRELIMINARY REPORT

Co-titular
Vincent Rijmen

Coach(es)
John Gaspoz
Dilara Toprakhisar

ACADEMIC YEAR 2024-2025

Declaration of originality

We hereby declare that this submitted draft is entirely our own, subject to feedback and support given us by the didactic team, and subject to lawful cooperation which was agreed with the same didactic team.
Regarding this draft, we also declare that:

1. *Note has been taken of the text on academic integrity (https://eng.kuleuven.be/studeren/masterproef-en-papers/documenten/20161221-academischeintegriteit-okt2016.pdf).*
2. *No plagiarism has been committed as described on https://eng.kuleuven.be/studeren/masterproef-en-papers/plagiaat.*
3. *All experiments, tests, measurements, …, have been performed as described in this draft, and no data or measurement results have been manipulated.*
4. *All sources employed in this draft – including internet sources – have been correctly referenced.*

Contents

List of Figures **II**

List of Tables **II**

0 Introduction **1**

1 Aircrack **2**
1.1 Cracking the WEP access point . . . 2
1.1.1 Structure of a WEP-encrypted packet . . . 2
1.2 Executing the attack . . . 2

2 Reading the network traffic using a sniffer **4**
2.1 WEP protocol . . . 4
2.1.1 RC4 encryption . . . 4
2.1.2 RC4 decryption . . . 6

3 Website **7**
3.1 Website structure . . . 7
3.2 Programming languages . . . 7
3.3 Description of user experience . . . 7
3.4 References . . . 7

4 Man-in-the-Middle **8**
4.1 ARP poisoning . . . 8
4.2 Scapy . . . 8
4.3 Live intervention . . . 9

5 Conclusion and further improvements **10**
5.1 Gantt-chart . . . 10

6 Subject integration **11**

References **12**

Use of Artificial Intelligence (AI) **13**

7 Appendices **15**
7.1 Aircrack commands . . . 15
7.2 Github repository . . . 15
7.3 RC4 encryption pseudocode . . . 16
7.4 Website . . . 18
7.4.1 Main page . . . 18
7.4.2 Login page . . . 19
7.4.3 Getstarted page . . . 20
7.4.4 Keuze page . . . 21
7.4.5 Bank page . . . 22
7.4.6 Fout page . . . 23

List of Figures

1 Network Layout . . . 1
2 The structure of a WEP-encrypted packet . . . 2
3 A WEP key that has been found by Aircrack . . . 3
4 Network layout sniffer . . . 4
5 Network layout MitM . . . 8
6 Gantt chart . . . 10
7 Website main page, part 1 . . . 18
8 Website main page, part 2 . . . 18
9 Website main page, part 3 . . . 19
10 Website login page . . . 19
11 Website getstarted page, part 1 . . . 20
12 Website getstarted page, part 2 . . . 20
13 Website keuze page, part 1 . . . 21
14 Website keuze page, part 2 . . . 21
15 Website bank page, part 1 . . . 22
16 Website bank page, part 2 . . . 22
17 Website fout page . . . 23

List of Tables

1 Example of XOR-encryption using a random keystream . . . 6

0 Introduction

Wi-Fi is an integral part of our lives today. There are billions of people using it every day to do anything from browsing social media to doing business to managing bank accounts. Therefore Wi-Fi security is very important. From the early days there have been attempts to secure Wi-Fi networks. It started with the WEP (Wired Equivalent Privacy) protocol in 1997 followed by WPA (Wi-Fi Protected Access) in 2003, which got then further improved in WPA2 and WPA3 in later years [Firdus et al., 2024]. This project focuses on the 1997 WEP protocol, which has declined in usage because of its vulnerabilities.

This project demonstrates how a single vulnerability in a network can compromise the security of the whole Local Area Network (LAN). The network configuration, illustrated in Figure 1, consists of two interconnected routers: one secured with the weak WEP protocol and another with the stronger WPA2 security. These routers are connected via Ethernet, with a Raspberry Pi 3 hosting a local website on the WPA2-secured network. To demonstrate the vulnerability of the WEP-protected network, the connection between a victim and the first router is exploited to intercept and modify data transmitted to the secure website through Address Resolution Protocol (ARP) manipulation [Chapter 4]. Before the attack, exploiting the LAN is crucial to gather necessary information, like the physical- or MAC address. While network packets are broadcasted through the air and encrypted for security, they can be intercepted without establishing a direct network connection. The encryption key, normally restricted to authorized network devices, can be obtained using the Aircrack-ng suite. This means an attacker can capture, decrypt and see all your activity over the WEP-protected network.

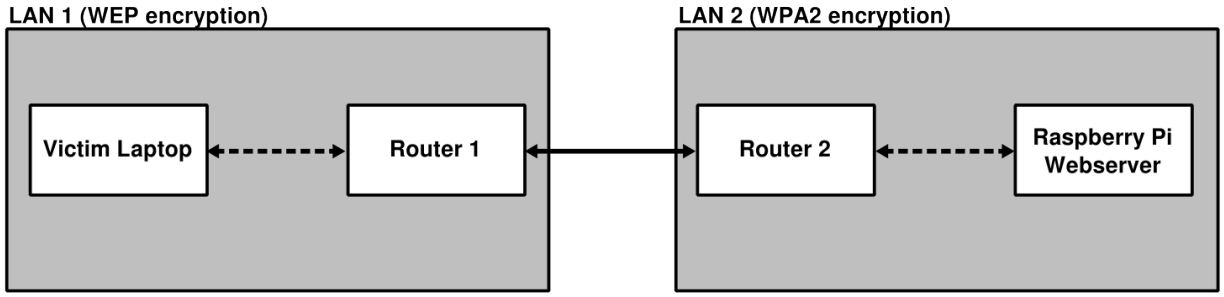

Figure 1: Network Layout

This paper will start in chapter 1 by detailing how the Aircrack-ng network security suite is used to obtain the WEP-encryption key. Chapter 2 describes how packets sent over the WEP network can be intercepted and decrypted, with section 2.1 explaining how the WEP protocol encrypts and decrypts its data using the RC4 algorithm. The implementation of the local website is gone over in chapter 3. In chapter 4 the concept, process and implementation of a man-in-the-middle (MitM) attack gets explained. Chapter 5 contains a brief summary of the project as well as outlining possible further improvements to the project. Details on how knowledge obtained through past courses is integrated in this project can be found in chapter 6.

1 Aircrack

The first step in the project is acquiring the WEP key, as it is a key component for all the following steps. Aircrack-ng is used to find this key. Aircrack-ng, further referred to as Aircrack, is a suite of network security tools used for monitoring, attacking, and cracking WEP and WPA encrypted networks [Devine, 2022]. It is commonly used to assess network security. In this project, Aircrack will aqcuire the WEP key using flaws of the WEP-encryption algorithm.

1.1 Cracking the WEP access point

WEP encryption uses a combination of a shared key and a Initialization Vector (IV) to encrypt network traffic. However, due to the weak IV implementation, it is possible to recover the shared key after capturing enough packets. Aircrack exploits this vulnerability by analyzing captured data and IVs to reconstruct the WEP key.

1.1.1 Structure of a WEP-encrypted packet

The inherent vulnerability of WEP encryption lies in its packet structure. To encrypt a packet, a key is used, which consists of:

- A 24-bit Initialization Vector (IV)
- A WEP key consisting of either 40 bits (for 64-bit encryption) or 104 bits (for 128-bit encryption)

This key is used in an RC4 algorithm to encrypt the packet. However, as the IV is randomly generated for every packet, this causes a problem for the receiver. To enable decryption, the IV is appended to the front of the encrypted packet. This feature is the core weakness of WEP encryption, as it creates two critical vulnerabilities:

1. The IV's limited size (24 bits) leads to the inevitable reuse
2. The transmission of the IV enables attackers to collect and analyze IV patterns

When two packets share the same IV, also known as IV Collision [Barken, 2003], cryptanalysis can be used to determine the static WEP key, allowing for all subsequent packets to be decrypted.

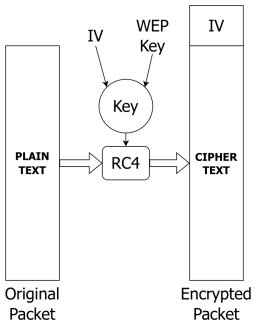

Figure 2: The structure of a WEP-encrypted packet

1.2 Executing the attack

Aircrack consists of several tools, each performing a specific function in WEP password cracking. The following paragraphs aim to provide a high-level overview of the attack and the commands used during it. This attack largely follows the one detailed in [darkAudax, 2010]. The specific commands used are detailed in appendix 7.1.

The main goal of the attack is to capture lots of packets sent on the WEP network in order to find cases of IV Collision. This process consists of a few steps. First, there is the matter of detecting the correct packets. Typically, a Raspberry Pi can only receive packets addressed to itself. This problem necessitates the use of

the *airmon-ng* command. This command enables monitor mode on the Raspberry Pi, allowing it to listen to all packets in the air, thus enabling it to intercept WEP network traffic.

Now that the Raspberry Pi can detect traffic, *airodump-ng* is used to capture the data that the Raspberry Pi detects. By specifying a channel and BSSID, it is configured to exclusively capture data from the WEP-encrypted router. Aircrack will now capture all traffic on the network while keeping track of the IVs and the encrypted data.

Once a sufficient amount of packets have been captured by *airodump-ng*, *aircrack-ng* is employed to crack the WEP key. This command makes use of the weaknesses of WEP encryption detailed in [Section 1.1.1]. Using packets with a shared IV, *aircrack-ng* utilizes cryptanalysis to determine the WEP key. A more detailed explanation of the RC4 algorithm can be found in [Section 2.1].

Figure 3: A WEP key that has been found by Aircrack

2 Reading the network traffic using a sniffer

To capture and read network traffic on the router, a network capture system, known as a sniffer, is implemented. This system is developed using Scapy, which is "a Python program that enables the user to send, sniff, dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks." [Biondi, 2024] In this implementation, the system intercepts and analyzes traffic from the WEP-encrypted network. By combining Scapy with a decryption algorithm, the sniffer captures, decodes and reads network traffic. This decryption algorithm requires the encryption key for the WEP encryption. This key can easily be obtained by the Aircrack attack. Worth noting, the sniffer operates externally to the target network, as illustrated in Figure 4.

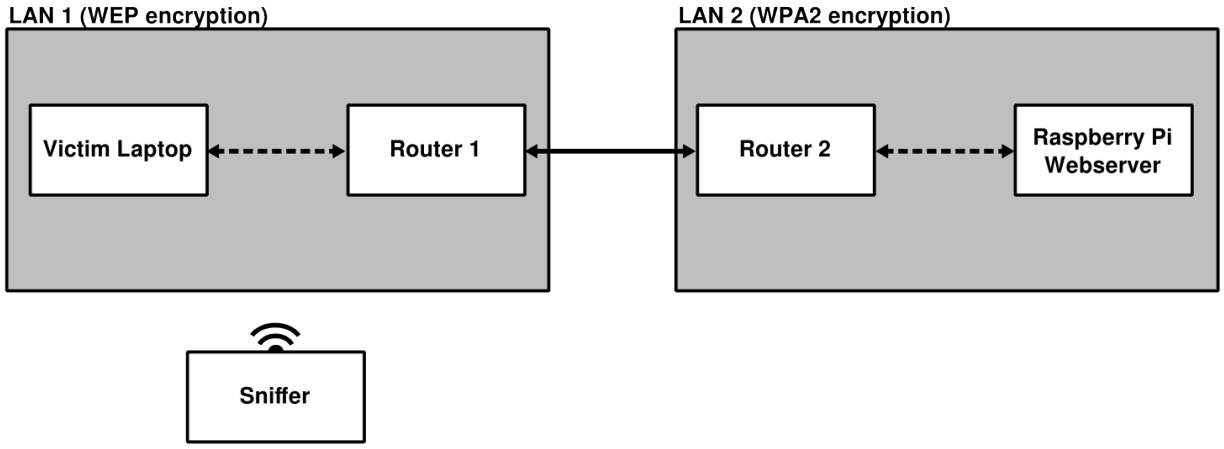

Figure 4: Network layout sniffer

The system utilizes Scapy's "sniff" functionality to intercept network traffic directed at the Raspberry Pi. When using monitor mode, it can detect and sniff all wireless traffic within range. These captured packets can then be filtered based upon the wanted criteria. In this case two filters will be active. First of all, packets gets sent from all different devices. Each device has a unique MAC address, which can be interpreted as a "physical address" for the device. It is important to only filter the packets sent from the device, and thus the MAC address, of the victim. A second filter will be applied to the packets type. For this project, only the WEP-encrypted packets are relevant, since those are the packets that can be decrypted.

2.1 WEP protocol

This paragraph will give a short history of the WEP protocol and explain how it encrypts its data using the RC4 algorithm. This descryption is kept very simple and the algorithm will be simplified, the purpose of this is to gather insight quicker and more easily. To follow along the explanation, a pseudo-code in python is added to appendix 7.3.

WEP stands for Wired Equivalent Privacy, which also describes it's function. This protocol tries to protect your private data in a wireless connection with the same security as a wired connection. In 1999, it was the standard for Wi-Fi security. It makes use of the RC4 encryption algorithm to encrypt the data. On this day, WEP isn't used anymore because flaws were detected and abused. In the configured network used for this report 4, router 1 uses the WEP protocol. This router will be used as a weak point in the network.

The following paragraphs describes how the RC4 algorithm works and how to decrypt the packets using the key gathered by the Aircrack attack [Chapter 1]. It is divided into 3 parts. The first part describes the key-scheduling algorithm, this is an algorithm which uses the key to create a 256-byte array, also called the S-box. The second part goes over the the algorithm which generates the keystream, using the S-box. In the third part the encryption is explained.

2.1.1 RC4 encryption

This explanation on the RC4 encryption is based upon the article written by S Sriadhi, Robbi Rahim and Ansari Saleh Ahmar [S Sriadhi, 2018]. This explanation is a simplified description, but provides enough insight

to understand the use of it in our decryption. Appendix 7.3 gives a pseudocode written based upon the process in this paragraph.

Key-scheduling algorithm

The packet before encryption consists of a plain, readable text. To encrypt this packet, a secret key is necessary. This key could be any string of text with a length between 1 and 256 bytes. In most cases this key is less than 256 bytes long. To obtain a string with the desired length of 256, the shorter key is simply repeated.

This 256 byte long string is now used to generate the S-box. Using the ASCII-table [ASCII-table, 1963], each byte of the extended key is converted to a three digits long decimal code. At the start, the S-box is a simple array with 256 entries. The first entry holds 0, the one after holds 1. Repeating this results in the last entry holding 255. Using a simple repetitive algorithm, this array will be permutated using the extended key. Note the following two variables:

- i holds the amount of iterations the algorithm has been done. It starts at 0 and ends at 255.
- j holds a natural number between 0 and 255 which is calculated by formula 1. Using the modulo operator, j will never exceed these boundaries. For the first iteration, j holds a value 0.

These variables are also used as indices for the key or the S-box. In this case they are noted key[i] or s_box[i].

$$j = (j + \text{s\_box}[j] + \text{key}[i]) \bmod 256 \tag{1}$$

Note that the j on the right side of the equation still holds the value of the previous iteration.

To finish the algorithm, two entries are swapped. These entries are defined by the calculated indices i and j.

$$\text{s\_box}[i], \text{s\_box}[j] = \text{s\_box}[j], \text{s\_box}[i]$$

Note that the swapped values can be any value between 0 and 256 and are only indicated by the indices i and j.

Generating the keystream

Now that the S-box has been created, the keystream can be generated using another algorithm. This algorithm makes use of three variables i, j and k. Note that these variables will be reset at the start of the algorithm. This means that the value of these variables in the key-scheduling algorithm has no impact on the starting values.

The first two variables serve a similar function as in the key scheduling algorithm:

- i holds the amount of iterations completed plus one. This means it starts with a value of 1 and gets the value of the next natural number each iteration. Note that i is using the modulo 256 operator, hence it will never have a value higher than 255.
- j is once again a natural number calculated by formula 2. It makes use of the values stored in the S-box. j has the same upper bound due to the same modulo operator.

$$j = (j + \text{s\_box}[i]) \bmod 256 \tag{2}$$

On the right side of the equation, j holds the value of the previous iteration. This was also the case in the previous algorithm.

With these two variables defined, the next thing to look at is the desired lenght of the keystream. RC4 uses a one-on-one encryption method, which will be discussed in the third part. This means that the desired lenght is the same lenght as the message that has to be encrypted. To ensure this, the algorithm will be executed one time for each character in the message.

Every iteration, a new value for the third variable, k, will be calculated as follows:

$$k = (s\_box[i] + s\_box[j]) \bmod 256 \tag{3}$$

Once again the modulo operator has been used as this variable will be used as an index. More specifically, the byte at the entry k in the S-box, or s_box[k], will be added at the end of the keystream. Note that the keystream empty at the beginning. The iteration ends with the same swapping operation that has been discussed earlier.

$$\text{s\_box}[i], \text{s\_box}[j] = \text{s\_box}[j], \text{s\_box}[i]$$

After completing this process, the final keystream is generated. Using this keystream, the original plain text can now be encrypted.

Encryption of the packet

This process starts with converting the ASCII codes to eight digit long binary codes, or bytes. This is done for the keystream and for the plain text. Next, the logical operator XOR is used on both bytes to generate the encrypted output string, which also is one byte.

The visualization in table 1 gives an example of the process described in the paragraph above. The keystream used in this example is completely random.

INPUT			**KEYSTREAM**			**OUTPUT**
index	*byte*	*binary*	*index*	*byte*	*binary*	*binary*
0	P	01010000	0	¥	10100101	11110101
1		10100000	1	D	01000100	11100100
2	e	01100101	2	c	00101110	01001011
3	n	01101110	3	L	01001100	00100010
4		10100000	4	š	10011010	00111010
5	O	01001111	5	ö	11110110	10111001

Table 1: Example of XOR-encryption using a random keystream

One of the benefits of using the XOR operator is how easy it is to reverse. To reverse the encryption, use the XOR operator on the encrypted input and the keystream. This is why XOR is called "a symmetrical operator".

The final step of the encryption is to turn the final binary output back into symbols using the ASCII-table [ASCII-table, 1963].

2.1.2 RC4 decryption

As mentioned in the introduction 2.1, the way the intercepted packets are decrypted will be the exact same way as the victim normally would decrypt them. This is possible because the secret key is known as a result of the successful Aircrack attack 1.

First, the key is used to generate the same keystream as described in the encryption process. Then, the encrypted message and the keystream are both converted to binary. When converted, the XOR operator is used on both binary streams to generate the decrypted binary stream. This illustrates how easily reversible the XOR operator is, as explained above. The last step is to convert the binary stream back to the decrypted message.

3 Website

The website aims to be a clone of Netflix. The idea is to make the victim believe they are consulting the real Netflix website. This would allow the attacker to steal the victim's personal information through the Man In the Middle attack. The following section provides detailed information about the website structure, the programming languages used, the victim's experience, and the references employed.

3.1 Website structure

The website consists of six distinct pages:

- The "index" page, where the user begins their journey. This page of the website is shown in Figure 7, Figure 8 and Figure 9.
- The "login" page, where users can sign in to an existing account. This page of the website is shown in Figure 10.
- The "getstarted" page, allows users to create a new account. This page of the website is shown in Figure 11 and Figure 12.
- The "keuze" page, where users can select from different payment options. This page of the website is shown in Figure 13 and Figure 14.
- The "bank" page, where users finalize the purchase of their chosen Netflix plan. This page of the website is shown in Figure 15 and Figure 16.
- The "fout" page, where the user is directed after entering their information. This page notifies the user of an error and redirects them to the real Netflix website. This way, the user remains unaware that their data has been stolen. This page of the website is shown in Figure 17.

The complete code for these six pages is available in the GitHub repository with additional comments to enhance its clarity. A link of the repository is provided in the appendices section.

3.2 Programming languages

The website uses HTML, CSS, and JavaScript for its code. Each programming language has a clear and distinct purpose. The HTML code adds the structure and the content, the CSS code customizes the style and the JavaScript code changes the language and checks if all fields are correctly filled in.

3.3 Description of user experience

The user first lands on the index page. He can choose to adjust the language, log in to his existing account or create a new one. Sections below highlight the positives of Netflix and a FAQ provides additional information. If the user decides to log in, he will land on the login page where he will need to enter his phone number or email and his password. Once logged in, the user is redirected to an error page with a link to the real Netflix website. This ensures that the user never knows that his information has been stolen.

If the user wishes to register, he will have to provide his full name, email, phone number, password and gender on the getstarted page. The user then chooses one of the three subscriptions offered by Netflix on the keuze page. Once the user has decided, he lands on the bank page where he enters his financial information. After clicking on "Proceed", he arrives once again to an error page with a link to the real Netflix.

3.4 References

In developing the website, inspiration and information were gathered from various sources, including YouTube videos. One video [Coding With Dawid, 2022] provided valuable insights for creating a clone of the Netflix main page. Another video [CodingNepal, 2023] helped replicate the Netflix login page. In order to have a better understanding of registration and validation forms in JavaScript, a third video [JavaScript Academy, 2021] was a useful resource.

4 Man-in-the-Middle

The main target of this project is to intercept Locale Area Network (LAN) traffic packages and alternate them. In this way, a man-in-the-middle (MitM) attack can be performed between the traffic forwarding (forwarding to the Wide Area Network (WAN)) router and the victim's device.

To perform a MitM attack, the victim's device has to think that the attacker's device is the traffic forwarding router and vice versa the router has to think the attacker's device is the victim's device. See figure 5.

This manipulation is done by fooling the Address Resolution Protocol (ARP). "ARP is a commonly used protocol pertaining to computer communications." [Aayush Majumdar, 2021] In a LAN, ARP messages are shared. With these messages, the router in the network knows which IP addresses belong to which computer in the network, through correlating IP addresses and corresponding physical addresses (MAC addresses). Network messages are sent within the LAN, this obligates the attacker to connect to the LAN for carrying out an ARP Poisoning or Spoofing attack.

For this project, the MitM will be demonstrated for two attacks.The first attack is through implementations of redirection links. Redirect links redirect a user that interacts with a webpage to another webpage, which wasn't the intented webpage of the original site. The second attack redirects a subscription payment for renting an account to another account.

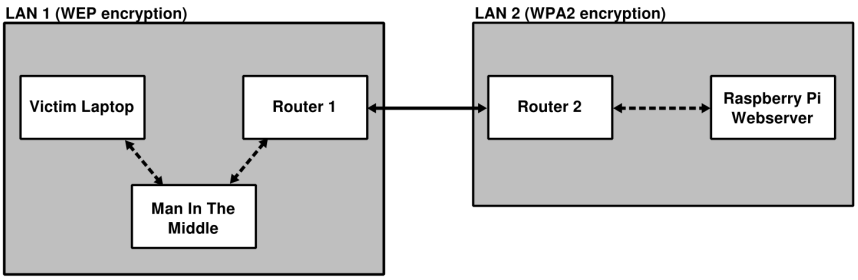

Figure 5: Network layout MitM

4.1 ARP poisoning

An ARP Poisoning attack is an attack in which spoofed ARP messages are sent to the default gateway on the locale network (LAN) with the intent of changing the ARP cache table. [Aayush Majumdar, 2021] The default gateway on a network is a node/gate with the purpose of connecting and so gaining access to another network. In the ARP cache table is information of the relation IP address $\longleftrightarrow$ MAC address stored. [Wikipedia, 2024]

Once the default gateway receives these ARP packets, the changes are broadcast to all devices connected to the network. Broadcasting makes it possible for devices to locate other devices within the network from which it needs to ask something. After swapping the ARP cache table, all devices recognize the attacker's device as the victim's device. Afterwards, all requests are sent to the attacker. Depending on the kind of attack to use, the attacker has to forward the network packets to the victim and in reverse if the packets came from the victim. In this process, the data packages may change. This process of intercepting and alternating the data packets is the man-in-the-middle (MitM) attack.

4.2 Scapy

For ARP-spoofing and the MitM attack, Scapy is used in Python for setting up an ARP poisoning packet, sending it to the gateway and afterward restoring the network by reversing the ARP spoofing. The physical address (MAC address), the gateway and the victim's IP needed for this program are found by Aircrack, chapter 1.

To control if the ARP cache table has changed, the ARP table can be printed using the command:
>>> **ARP -a**

or using the package 'sniff' from Scapy in Python.

4.3 Live intervention

Once the ARP poisoning attack succeeds, the attacker can implement its goal. This project's goal of the MitM attack is first to implement redirect links and second to redirect a subscription payment. Detour links are of interest to the attacker. They redirect the user to other sites with different targets. Targets such as advertising but also malicious websites. The second concept lets the user pay for the attacker by diverting the user's account rent to the attacker's account.

5 Conclusion and further improvements

To conclude this preliminary report, this paragraph will go over the parts of the project which have already been successfully executed, and that will be made in the near future. The Aircrack attack can gather the WEP key, as wanted. The use of this key to decrypt the RC4 encryption is also understood and complete. The current version of the website is valid for testing, though adjustments will probably be made in the future.

While the key aspects of the project, such as retrieving the WEP key and decrypting the RC4 encryption, have been successfully completed, the implementation of the sniffer remains a challenge. The main obstacle is getting the sniffer to work without being connected to the network, which has not yet been fully achieved. The second part of the project which will take some time in the future will be the modification of the data. The Man in the Middle attack has to be studied and understood in order to modify the victim's data, without the victim having this knowledge.

To improve the project, more focus should be placed on resolving the issues surrounding the sniffer, particularly its ability to operate without a network connection. Further research into the implementation of the Man-in-the-Middle attack is also needed.

5.1 Gantt-chart

Included below is a Gantt chart detailing the distribution of subtasks in the project.

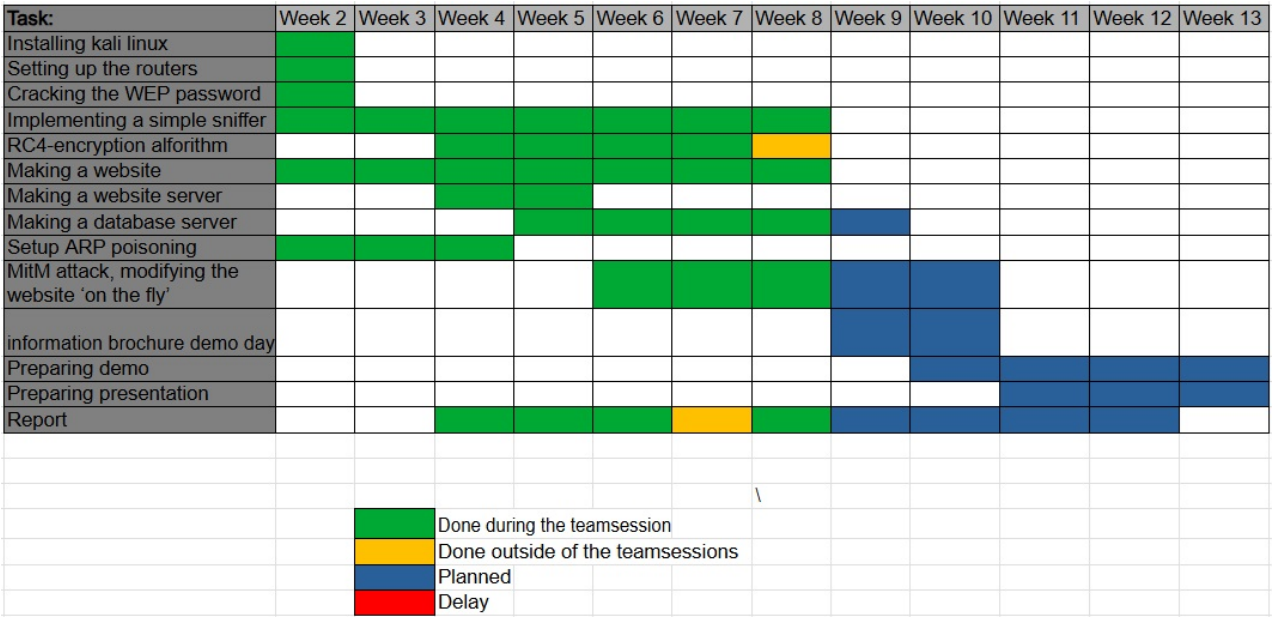

Figure 6: Gantt chart

6 Subject integration

The project is divided in multiple different interconnected domains (Software setup, Aircrack, sniffer, man-in-the-middle attack (MitM) using ARP-poisoning, website hosting). These domains use a combination of certain subjects seen in the first and second year of the Bachelor of Engineering.

The integrated subjects:

- Methodiek van de Informatica
- Toegepaste algebra
- Informatieoverdracht en -verwerking (IOV)
- Kansrekenen en statistiek

Aircrack For this domain *Kansrekenen en statistiek* are used. The more data gathered by the sniffer, the greater the chances of cracking the WEP key.

Sniffer The sniffer integrates knowledge from multiple subjects, including *Methodiek van de informatica*, *Toegepaste algebra*, and *IOV*. The sniffer is programmed in Python, applying techniques from *Methodiek van de informatica*. Capturing data packets from the network involves working with transmission protocols, which is part of the *IOV* subject area. Additionally, the decryption of captured, encrypted data relies on the RC4 algorithm, which uses algebraic techniques covered in *Toegepaste algebra*.

MitM De MitM is programmed in Python and uses techniques from the subject *Methodiek van de informatica*.

Web Hosting The website is programmed with HTML, CSS and JavaScript.

References

[Aayush Majumdar, 2021] Aayush Majumdar, Shruti Raj, T. S. (2021). Arp poisoning detection and prevention using scapy. *Journal of Physics*.

[ASCII-table, 1963] ASCII-table (1963). https://www.ascii-code.com/.

[Barken, 2003] Barken, L. (December 23, 2003). Wep vulnerabilities—wired equivalent privacy? https://www.informit.com/articles/article.aspx?p=102230&seqNum=6. [Online; accessed 14-October-2024].

[Biondi, 2024] Biondi, P. (2024). Introduction — scapy 2.6.0 documentation. https://scapy.readthedocs.io/en/latest/introduction.html.

[Coding With Dawid, 2022] Coding With Dawid (2022). Netflix clone with html and css (tutorial for beginners). Accessed: 2024-10-01.

[CodingNepal, 2023] CodingNepal (2023). Create a netflix login page in html and css — netflix login page clone in html and css. Accessed: 2024-10-01.

[darkAudax, 2010] darkAudax (January 11, 2010). Tutorial: Simple wep crack. https://www.aircrack-ng.org/doku.php?id=simple_wep_crack.

[Devine, 2022] Devine, C. (May 10, 2022). Aircrank-ng. https://www.aircrack-ng.org/.

[Firdus et al., 2024] Firdus, E., Aghababayev, R., Aliyev, V., Mustafayeva, G., Mayilov, R., Sardarova, I., and Bakhshaliyeva, S. (2024). Wifi from past to today, consequences that can cause and measures of prevention from them, wifi security protocols. https://doi.org/10.1051/e3sconf/202447402004.

[Ibrahim, 2021] Ibrahim, A. (March 16, 2021). Wired equivalent privacy (wep) explained. https://www.youtube.com/watch?v=6cKgoA3Qj60. [Online; accessed 14-October-2024].

[JavaScript Academy, 2021] JavaScript Academy (2021). Form validation using javascript on the client side for beginners. Accessed: 2024-10-01.

[S Sriadhi, 2018] S Sriadhi, Robbi Rahim, A. S. A. (2018). Conf. ser. 1028 012057. *Journal of Physics*.

[Wikipedia, 2024] Wikipedia (2024). Default gateway — Wikipedia, the free encyclopedia. [Online; accessed 14-October-2024].

Use of Artificial Intelligence (AI)

Gebruik van een AI-schrijfassistent bij het schrijven van dit P&O3-verslag

Teamnaam: ESAT2A2

Teamleden: Jarle Braeken, Arthur Cukier, Pieter Deferme, Robin Derikx, Willem Hendig & Giel Swenters

Duid aan met "X":

O **Dit verslag maakte geen gebruik** van een AI-schrijfassistent.

X **Dit verslag maakte wel gebruik** van een AI-schrijfassistent (vb.: ChatGPT, Microsoft Copilot, Google Gemini, Grammarly, …), nl.:
ChatGPT

Duid aan met "X" (meerdere opties mogelijk) op welke manier dit gebruikt werd.

X ~~**Uitsluitend**~~ **als hulp bij de taal**

- *Gedragscode*: Dit is gelijkwaardig met het gebruik van een spellingscontrole

X **Als zoekmachine om te leren over een bepaald onderwerp**

- *Gedragscode*: Dit gebruik is vergelijkbaar met bijvoorbeeld een Google-zoekopdracht of het raadplegen van Wikipedia. Wees je ervan bewust dat de output van de AI-schrijfassistent evolueert en in de loop van de tijd kan veranderen.

O **Voor een literatuurstudie**

- *Gedragscode*: Dit gebruik is vergelijkbaar met bijvoorbeeld een Google Scholar-zoekopdracht. Houd er echter rekening mee dat sommige AI-schrijfassistenten geen of onjuiste verwijzingen kunnen geven. Als student ben je verantwoordelijk voor het verder controleren en verifiëren van de afwezigheid of juistheid van verwijzingen.

O **Voor ondersteuning bij korte stukken tekst (bv. parafraseren, samenvatten, herschrijven, …)**

- *Gedragscode*: Dit gebruik is vergelijkbaar met bijvoorbeeld Google Docs, dat wordt aangedreven door generatieve taalmodellen.

X Voor het genereren van programmeercode

- *Gedragscode*: Vermeld correct het gebruik van een AI-schrijfassistent en citeer het. Je kunt ook vragen aan het AI-schrijfassistent hoe je het moet citeren.

O **Voor het genereren van nieuwe onderzoeksideeën**

- *Gedragscode*: Controleer in dit geval verder of het idee nieuw is of niet. Het is waarschijnlijk dat het gerelateerd is aan bestaand werk, waar dan naar moet worden verwezen.

O **Voor het genereren van (grote) tekstblokken**

- *Gedragscode*: Het invoegen van tekstblokken zonder aanhalingstekens van een AI-schrijfassistent in je verslag is niet toegestaan. Volgens Artikel 84 van het onderwijs- en examenreglement moet bij het evalueren van je werk correct worden beoordeeld op je eigen kennis. Indien het echt nodig is om een tekstblok van een AI-schrijfassistent in te voegen, maak hier dan een citaat van door gebruik te maken van aanhalingstekens. Dit moet hoe dan ook tot een absoluut minimum beperkt blijven.

O **Ander gebruik**

- *Gedragscode*: Bespreek het gebruik van een AI-schrijfassistent in dit geval met je begeleiding. Motiveer hoe je voldoet aan Artikel 84 van het onderwijs- en examenreglement. Leg het gebruik en de toegevoegde waarde van een AI-schrijfassistent in deze situatie uit:

.....

.....

.....

.....

7 Appendices

7.1 Aircrack commands

These are the Aircrack commands used for cracking the WEP key.

```
airmon-ng check kill
airmon-ng start wlan0
airodump-ng wlan0mon
```

Replace CHANNEL en BSSID with the right channel and bssid.

```
airodump-ng -c CHANNEL --bssid BSSID -w dump wlan0mon
```

After enough packets are recieved the encryption key can be cracked using:

```
aircrack-ng -b BSSID dump-01.cap
```

After cracking the password the wlan0mon interface can be stopped using:

```
airmon-ng stop wlan0mon
```

And the regular interdace can be restarted using:

```
systemctl start NetworkManager.service
```

7.2 Github repository

All the code for the website, ARP spoofing, Aircrack, and MitM can be found in a GitHub repository, accessible via the following link: `https://github.com/Jarle-student/ESAT2A2`.

7.3 RC4 encryption pseudocode

```
import pyperclip

def extend_key(key):
    key_lenght = len(key)
    extended_key = []
    for i in range(256):
        extended_key.append(key[i % key_lenght])
    return extended_key

def convert_to_ASCII(list):
    for i in range(len(list)):
        list[i] = ord(list[i])
    return list

def generate_sbox(key):
    keystream = [i for i in range(256)]
    j = 0
    for i in range(256):
        j = (j + keystream[j] + key[i]) % 256
        keystream[i], keystream[j] = keystream[j], keystream[i]
    return keystream

def generate_keystream(extended_key, packet):
    S_box = generate_sbox(extended_key)
    keystream = []
    i = j = 0
    for l in range(len(packet)):
        i = (i + 1) % 256
        j = (j + S_box[i]) % 256
        S_box[j], S_box[i] = S_box[i], S_box[j]
        k = (S_box[i] + S_box[j]) % 256
        keystream.append(S_box[k])
    return keystream

def convert_to_list(text):
    output = []
    for i in range(len(text)):
        output.append(text[i])
    return output

def convert_to_binary(list):
    for i in range(len(list)):
        list[i] = format(list[i],'08b')
    return list

def XOR(bin1, bin2):
    output_bin = ""
    for i in range(len(bin1)):
        if bin1[i] == bin2[i]:
            output_bin = output_bin + "0"
        else:
            output_bin = output_bin + "1"
    return output_bin

def crypt_bin(keystream, text):
    encrypted_bin_list = []
    for i in range(len(text)):
        encrypted_bin_list.append(XOR(keystream[i], text[i]))
    return encrypted_bin_list

```

```
def convert_to_standard(list):
    encrypted_message = ""
    for i in range(len(list)):
        char = chr(int(list[i], 2))
        encrypted_message = encrypted_message + char
    return encrypted_message

def encrypt():
    plain_text = input("Insert message here: ")
    text_list = convert_to_list(plain_text)
    binary_text = convert_to_binary(convert_to_ASCII(text_list))
    key = input("Insert secret key here: ")
    extended_key = extend_key(key)
    ASCII_key = convert_to_ASCII(extended_key)
    keystream = generate_keystream(ASCII_key, text_list)
    bin_keystream = convert_to_binary(keystream)
    encrypted_bin = crypt_bin(bin_keystream, binary_text)
    encrypted_message = convert_to_standard(encrypted_bin)
    pyperclip.copy(encrypted_message)
    return encrypted_message, key

def decrypt(encr_message, key):
    messge_lst = convert_to_list(encr_message)
    extended_key = extend_key(key)
    ascii_key = convert_to_ASCII(extended_key)
    keystream = generate_keystream(ascii_key, messge_lst)
    bin_keystream = convert_to_binary(keystream)
    decrypted_bin = crypt_bin(bin_keystream, convert_to_binary(
        convert_to_ASCII(messge_lst)))
    decrypted_message = convert_to_standard(decrypted_bin)
    return decrypted_message


def main():
    encrypted_message, key = encrypt()
    print("This is the encrypted message: " + encrypted_message)
    decrypted_message = decrypt(encrypted_message, key)
    print("This is the decrypted message: " + decrypted_message)
    return

main()
```

7.4 Website

7.4.1 Main page

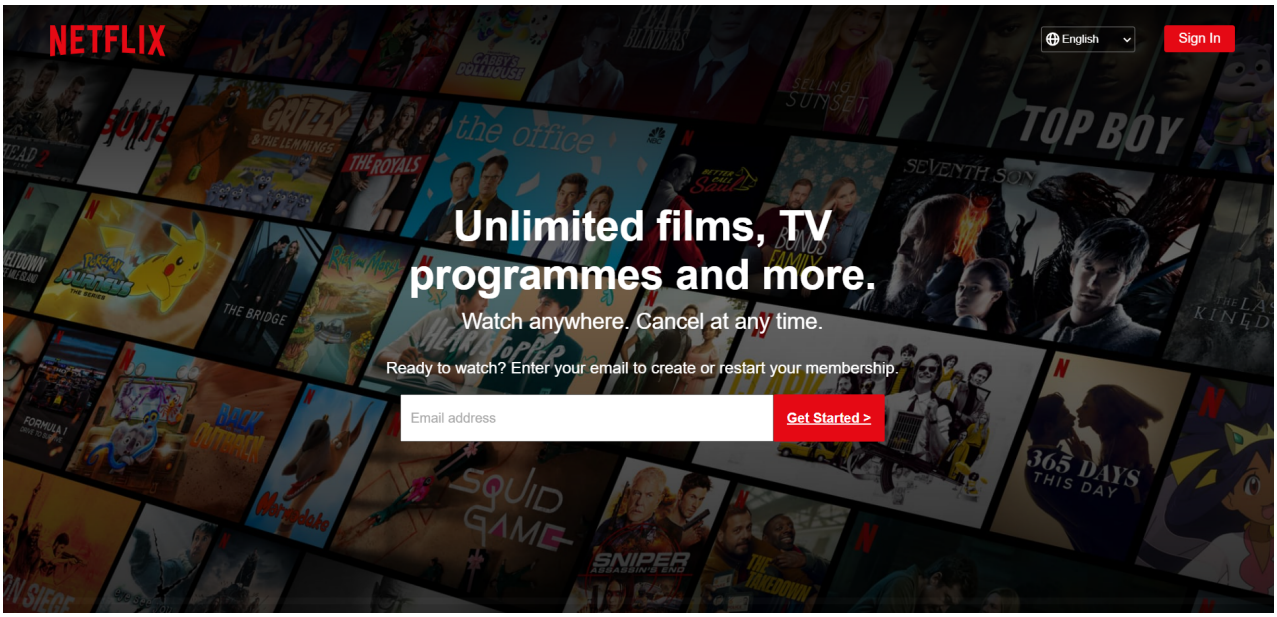

Figure 7: Website main page, part 1

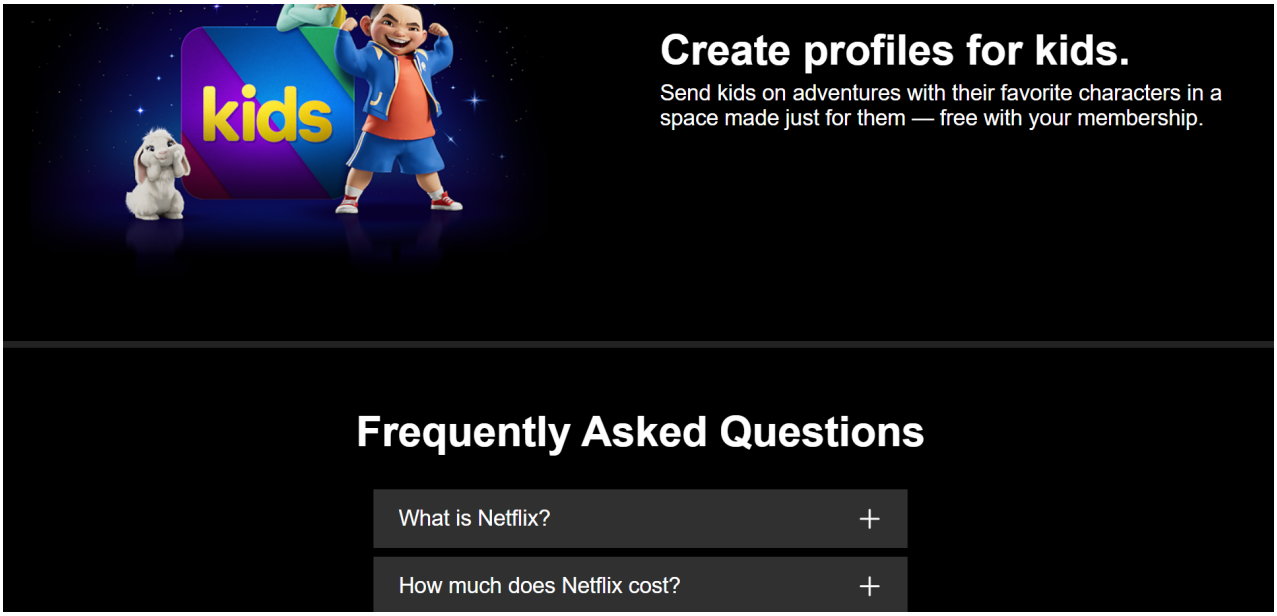

Figure 8: Website main page, part 2

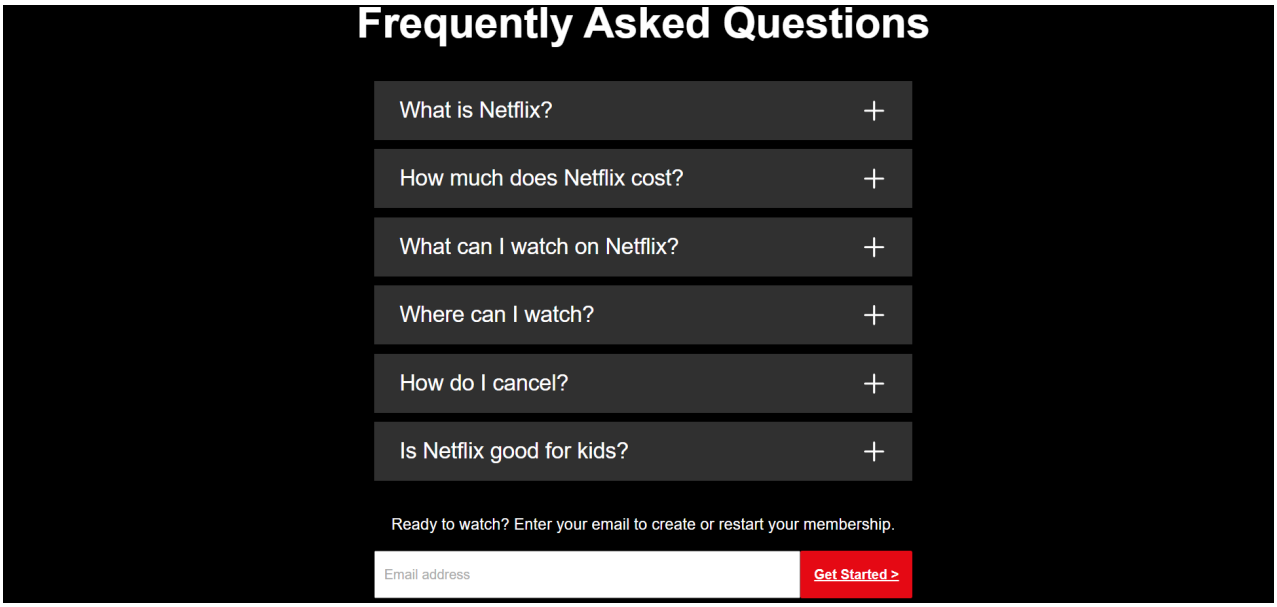

Figure 9: Website main page, part 3

7.4.2 Login page

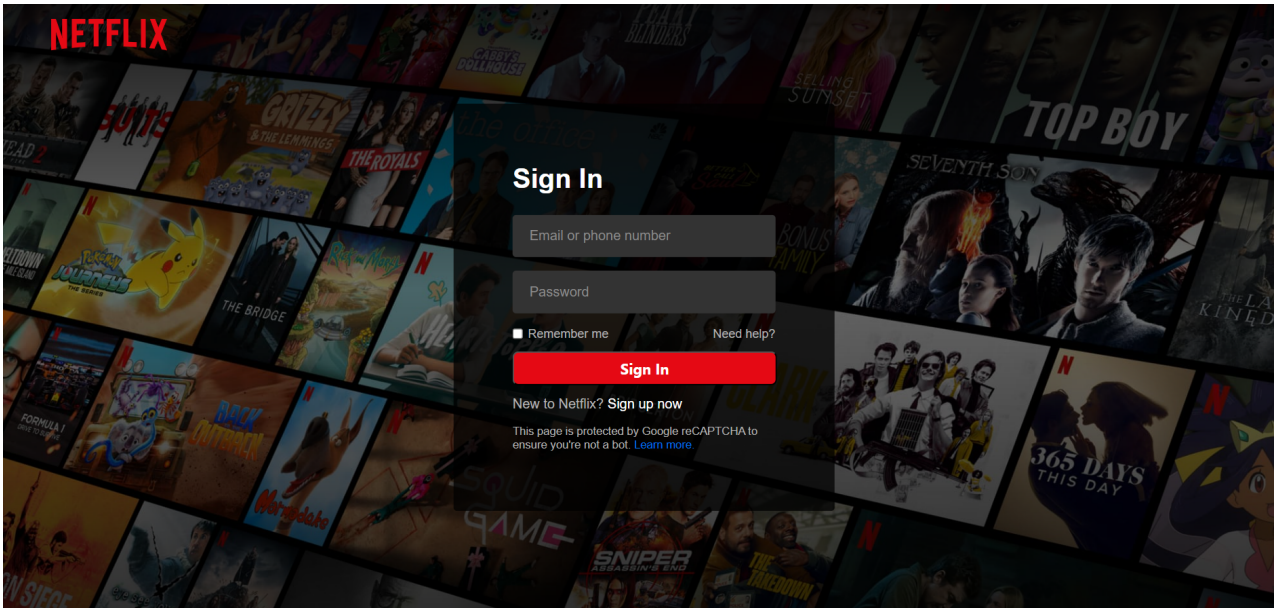

Figure 10: Website login page

7.4.3 Getstarted page

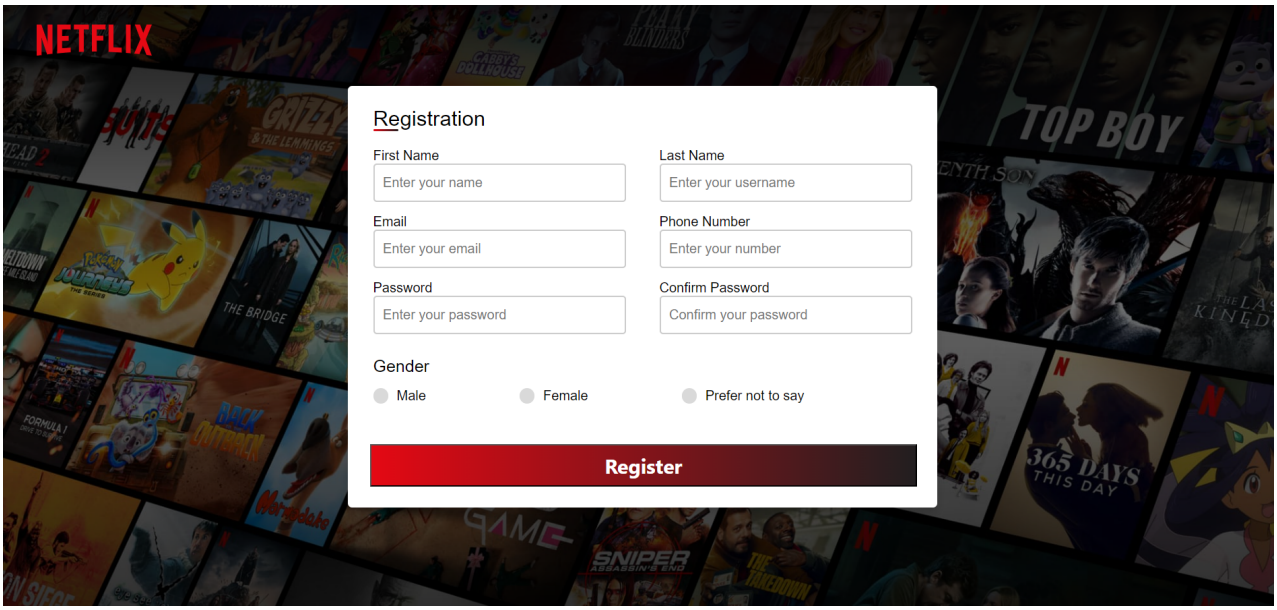

Figure 11: Website getstarted page, part 1

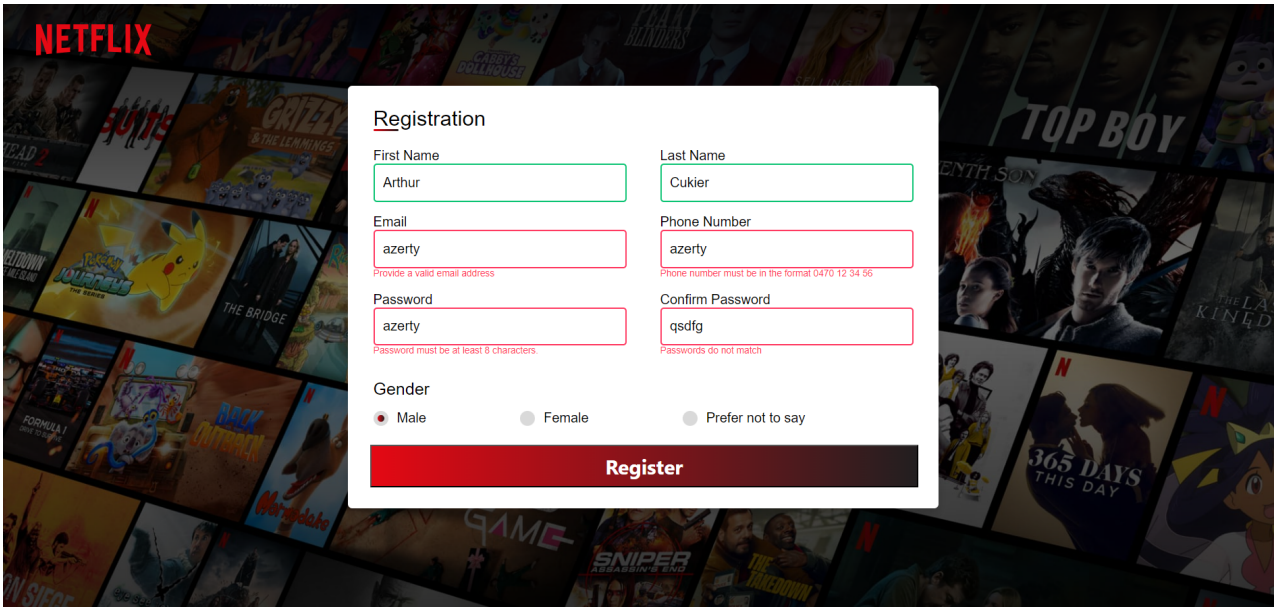

Figure 12: Website getstarted page, part 2

7.4.4 Keuze page

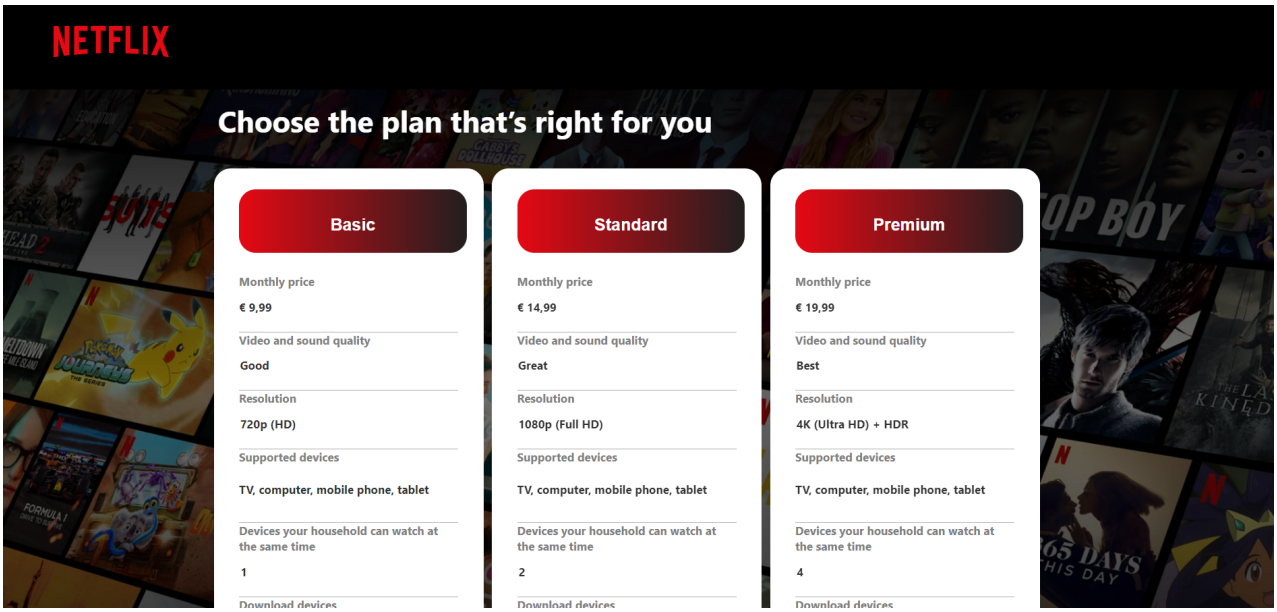

Figure 13: Website keuze page, part 1

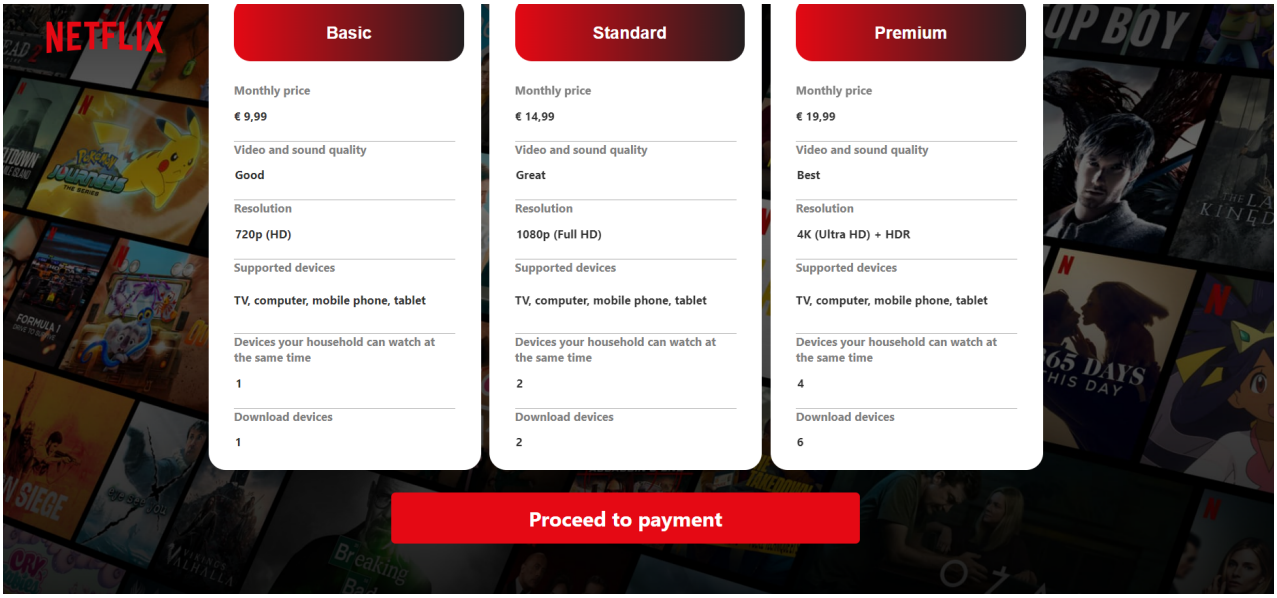

Figure 14: Website keuze page, part 2

7.4.5 Bank page

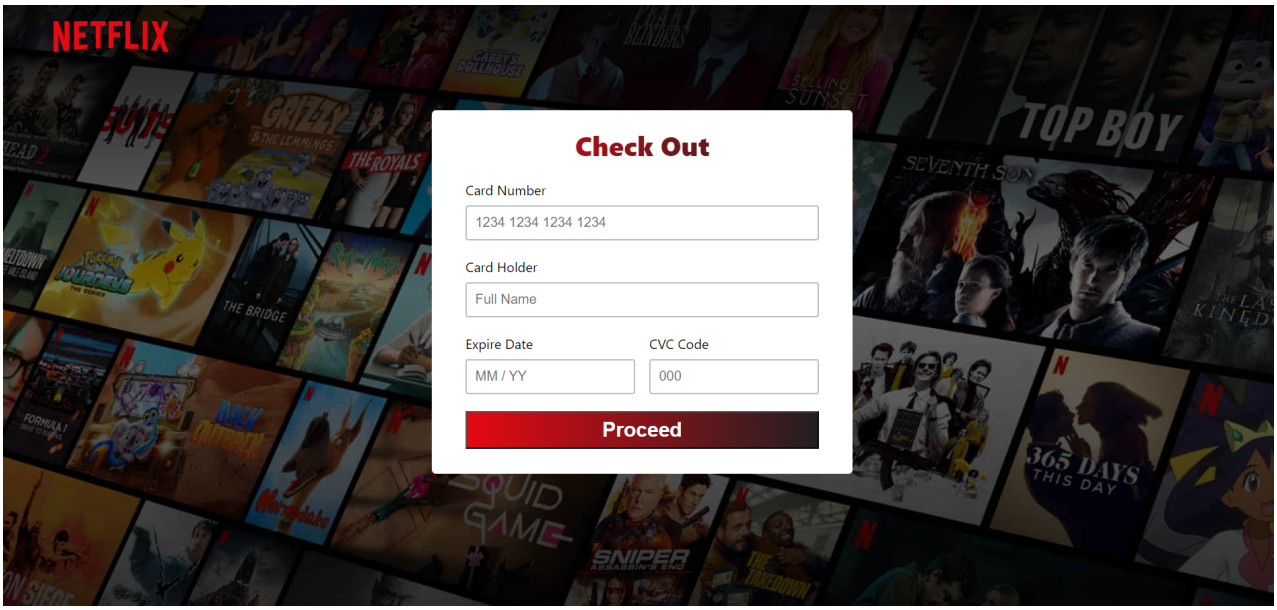

Figure 15: Website bank page, part 1

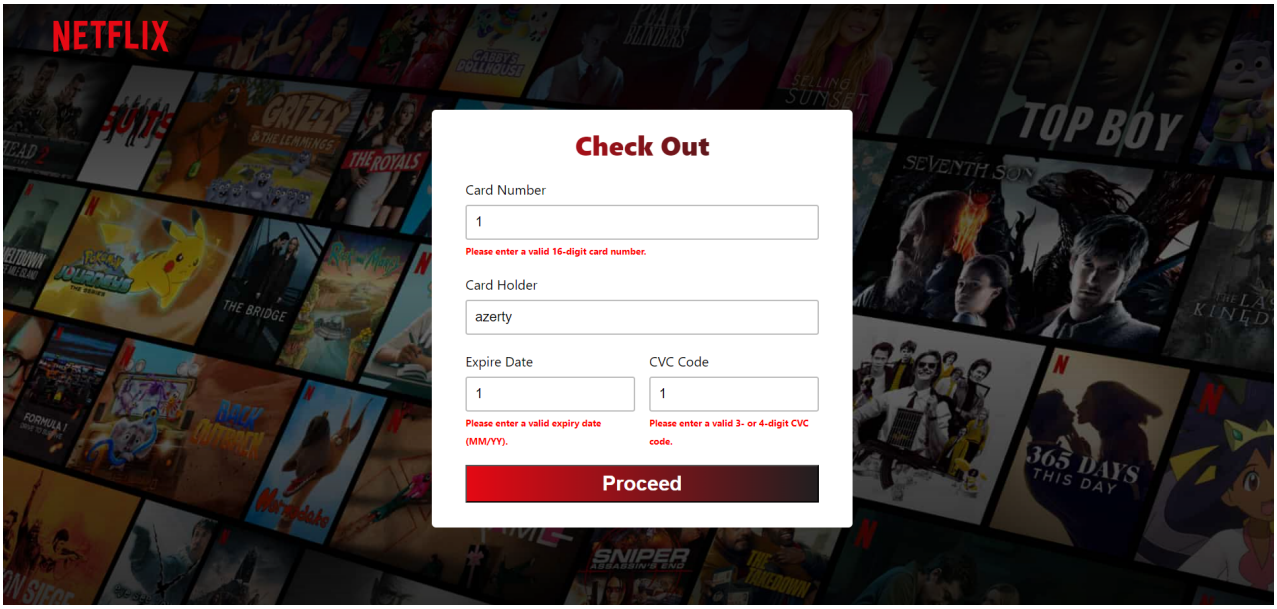

Figure 16: Website bank page, part 2

7.4.6 Fout page

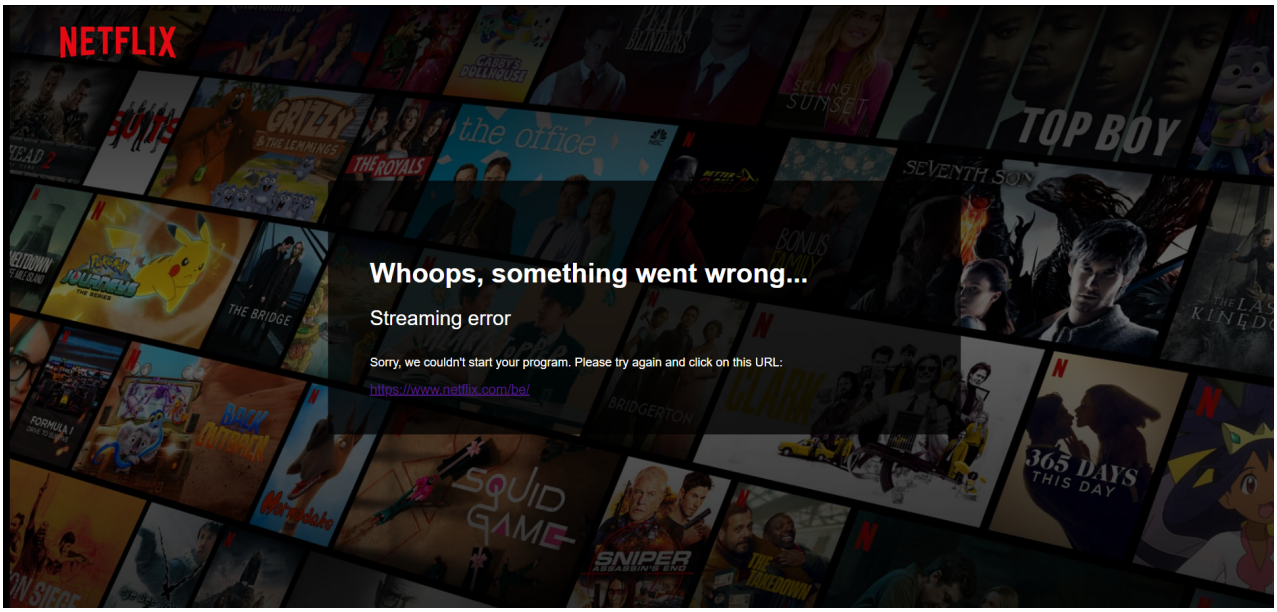

Figure 17: Website fout page