1 2 9 0

UNIVERSIDADE Ð COIMBRA

Luís Carlos Lopes Loureiro

Security For 5G Network Environments

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Professor Doutor Vasco Pereira and Professor Doutor Tiago Cruz presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2023

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Luís Carlos Lopes Loureiro

Security For 5G Network Environments

Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Prof. Doutor Vasco Pereira and Prof. Doutor Tiago Cruz and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Departamento de Engenharia Informática

Luís Carlos Lopes Loureiro

Segurança Para Ambientes De Redes 5G

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia de Software, orientada pelo Professor Doutor Vasco Pereira e Professor Doutor Tiago Cruz e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro 2023

Acknowledgements

First, I would like to thank my thesis advisors, Professor Vasco Pereira and Professor Tiago Cruz, for integrating me in this project and for all the help, guidance and revisions of this document. I would also like to mention the rest of the team, Diogo and Jorge, for always being there to answer my questions when I started this research work.

I would also like to thank all my friends and colleagues that I met during these years of college, for always being there when I needed you the most. At times when I felt stressed, you were a great source of support, helping me to stay calm and never lose my focus.

Lastly, and most importantly, I would like to express my gratitude to my family for their unconditional support and encouragement in every moment of my academic journey. Thank you for everything, for always being there no matter what, and for making this academic experience possible.

This work is funded by the project POWER (grant number POCI-01-0247-FEDER-070365), co-financed by the European Regional Development Fund (FEDER), through Portugal 2020 (PT2020), and by the Competitiveness and Internationalization Operational Programme (COMPETE 2020). This work was partially supported by the project Smart5Grid (co-funded by FEDER - Competitiveness and Internationalisation Operational Program (COMPETE 2020), Portugal 2020 framework).

Cofinanciado por:

Abstract

The 5G technology brings several improvements to mobile communication systems. The ability to provide much faster connections, low latency and greater scalability enables 5G to be employed in new areas and contexts, where previous generations of network were not feasible enough, such as autonomous driving, remote healthcare, smart manufacturing, and virtual and augmented reality experiences.

To achieve its goals of low latency and high scalability, 5G Networks requires a distributed network architecture with Multi-Access Edge Computing (MEC) at multiple remote points that leads to the introduction of new cloud and virtualization technologies. These access points, combined with the growing number of devices connected to the network, expand the attack surface and create new vulnerabilities, including the heightened risk of Distributed Denial of Service (DDoS) attacks. In addition, the high speed provided by 5G networks enables attackers to generate and distribute massive volumes of malicious traffic more rapidly, further increasing the impact of such attacks on the network.

This research work aims to address critical security aspects and challenges in 5G networks against flooding attacks. It begins with the implementation of a testbed that supports 5G Core Network functionalities empowered by orchestration tools, such as Kubernetes, that will enable a set of security tests. These tests include evaluating the security posture of the testbed through a Penetration Testing process by performing several types of flooding attacks. Based on the findings, a security solution was designed to improve the resilience of the testbed to effectively counter DDoS/flooding attacks, utilizing the capabilities of eBPF XDP.

This internship plan is part of the POWER project, to be developed at the Centre for Informatics and Systems of the University of Coimbra, which it intends to support in a structurally, transversal and complete manner the evolution of the current state of Altice Labs' portfolio of products and services, towards an integrated offer concept, which will enable new approaches to the market, fully aligned with four strong technological transformation vectors: 5G networks, continuum Edge/Cloud computing, AI&data-driven technologies and business models. This internship plan is part of a research effort focused on the development and evolution of products and services in an operator network environment.

Keywords

5G, Security, Testbed, 5G API, Flooding Attacks, DDoS

Resumo

A tecnologia 5G traz imensas melhorias para os sistemas de comunicação móvel. A capacidade de fornecer conexões muito mais rápidas, baixa latência e uma maior escalabilidade, faz com que o 5G impulsione novas áreas e contextos, os quais não eram viáveis para as gerações de rede anteriores, como condução autônoma, saúde remota, manufatura inteligente e experiências de realidade virtual e aumentada.

Para alcançar os seus objetivos de baixa latência e alta escalabilidade, as Redes 5G requerem uma arquitetura de rede distribuída com Computação de Borda de Multi-Acesso (MEC) em vários pontos remotos, o que leva à introdução de novas tecnologias de nuvem e virtualização. Estes pontos de acesso, combinados com o crescente número de dispositivos conectados à rede, ampliam a superfície de ataque e criam novas vulnerabilidades, incluindo o maior risco de ataques de Negação de Serviço Distribuída (DDoS). Além disso, a alta velocidade fornecida pelas redes 5G permite que os atacantes gerem e distribuam grandes volumes de tráfego malicioso de forma mais rápida, aumentando ainda mais o impacto desses ataques na rede.

Este trabalho de investigação visa abordar aspetos críticos de segurança e desafios em redes 5G contra ataques de inundação. Começará pela implementação de um Testbed que suporte funcionalidades de rede Core 5G potencializado por ferramentas de orquestração, como Kubernetes, que possibilitará um conjunto de testes de segurança. Esses testes incluem a avaliação da postura de segurança do Testbed através de um processo de Teste de Penetração, realizando vários tipos de ataques de inundação. Com base nos resultados, foi desenvolvida uma solução de segurança para melhorar a resistência do Testbed, a fim de enfrentar efetivamente ataques DDoS, utilizando as capacidades do eBPF e XDP.

Este plano de estágio está enquadrado no projeto POWER, a ser desenvolvido no Centro de Informática e Sistemas da Universidade de Coimbra, que pretende suportar de forma estrutural, transversal e completa a evolução do estado atual do portfólio de produtos e serviços da Altice Labs, rumo a um conceito de oferta integrada, que possibilitará novas abordagens ao mercado, totalmente alinhado com quatro fortes vetores tecnológicos de transformação: redes 5G, continuum de computação Edge/Cloud, tecnologias e modelos de negócios AI&data-driven. Este plano de estágio enquadra-se no âmbito de um esforço de investigação focado no desenvolvimento e evolução de produtos e serviços num ambiente de redes de operador.

Palavras-Chave

5G, Segurança, Testbed, API 5G, Ataques de Inundação, DDoS

Contents

Glossary

Botnet Network of computers infected by a malicious software developed by an attacker during a distributed cyber-attack.

Ciphertext Text after the encryption process, contained by randomized letters and numbers, which humans cannot understand.

Cloud-native Refers to the concept of building and deploying applications in cloud computing environments. Thus, it also empowers some emerging modern technologies such as Kubernetes and Docker while also giving the developers a comprehensive, standards-based platform for managing cloud native applications such as microservices and serverless functions.

Core network The network's central part, which offers numerous services to the customers who are interconnected by the access network.

IMSI The unique subscriber identifier allocated to the SIM card by a Mobile Network Operator. It's used to confirm a subscriber's identity and monitor their location. The IMSI information should be private.

Load Balancing The process of distributing network traffic across multiple servers. This ensures no single server bears too much demand, improving responsiveness and availability of applications and websites.

Network Function Functional building block within a network infrastructure, which has well-defined external interfaces and a well-defined functional behavior. In 5G, these network functions are virtualized, allowing network operators to manage and expand their network capabilities on using virtual, software based applications where physical boxes once stood in the network architecture.

Network Slicing Method of creating multiple unique logical and virtualized networks over a common multi-domain infrastructure.

NICs hardware components, often in the form of a circuit board or chip, that are integrated into a computer to enable network connectivity.

Plaintext Text before the encryption process with non-randomized characters that can be read by humans.

Radio Access Network It is a key component of any network architecture and is responsible for connecting any device used by an end user to the network.

Acronyms

4G/LTE	4th Generation Long Term Evolution.
AKA	Authentication and Key Agreement.
AMF	Access and Mobility Management Function.
ARP	Address Resolution Protocol.
ARPF	Authentication credential Repository and Processing Function.
AUSF	Authentication Server Function.
BPF	Berkeley Packet Filter.
CA	Certificate Authority.
DDoS	Distributed Denial of Service.
DN	Data Network.
EPC	Evolved Packet Core.
gNB	Next Generation Node B.
GUTI	Global Unique Temporary Identifier.
HPLMN	Home Public Land Mobile Network.
HTTP	HyperText Transfer Protocol.
IoT	Internet of Things.
IPX	Internetwork Packet Exchange.
JWE	JSON Web Encryption.
JWS	JSON Web Signatures.
LLVM	Low Level Virtual Machine.
MCC	Mobile Country Code.
MEC	Multi-Access Edge Computing.
MNC	Mobile Network Code.
MSIN	Mobile Subscriber Identification Number.

NEF Network Exposure Function.

NFV Network Function Virtualization.

NG-RAN New Generation Radio Access Network.

NRF Network Repository Function.

NSA Non-Stand Alone.

NSSF Network Slice Selection Function.

PCF Policy Control Function.

PLMN Public Land Mobile Network.

PTES Penetration Testing Execution Standard.

RAN Radio Access Network.

REST Representational State Transfer.

RKE2 Rancher′s next-generation Kubernetes distribution.

SA Stand Alone.

SAs Security Associations.

SBA Service Based Architecture.

SDN Software Defined Networking.

SEAF SEcurity Anchor Function.

SEPP Security Edge Proxy Protection.

SIDF Subscription Identifier De-concealing Function.

SMF Session Management Function.

SUCI Subscription Concealed Identifier.

SUPI Subscriber Permanent Identifier.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UDM Unified Data Management.

UDR User Data Repository.

UE User Equipment.

UPF User Plane Function.

VNF Virtual Network Function.

VPLMN Visited Public Land Mobile Network.

XDP eXpress Data Path.

List of Figures

List of Tables

Chapter 1

Introduction

This document describes all the work carried out by the student Luís Carlos Lopes Loureiro during his master's internship in informatics engineering.
This chapter aims to introduce the document and contextualize the research work. It will also present the goals of the project and the document structure.

1.1 Context

The 5G technology has arrived with the goals of providing higher throughput, lower latency, higher speed and larger capacity than its predecessor, 4th Generation Long Term Evolution (4G/LTE). Thus, it is expected that 5G will be able to support exciting use cases, including e-health, tactile computing and kinaesthetic communication, holographic interactions and the creation of smart cities, which use Internet of Things (IoT) devices to connect everything from smartphones, homes and cars to traffic systems. In order to achieve these goals, 5G networks need to be flexible, scalable and capable of reacting quickly to the changes in the network.

Software Defined Networking (SDN) and Network Function Virtualization (NFV) are the fundamental technologies that enable 5G to succeed these goals [1] [2]. The main goal behind the introduction of a SDN architecture is to make networks more flexible and agile by separating the Control Plane from the network hardware and enabling external control of data through a logical software entity, the SDN controller, making it easier to manage and introduce new services and changes to the 5G Network [3]. The NFV transforms the way networks are built and services are delivered. Instead of having each Network Function working in isolation in each dedicated hardware, they are now running in Virtual Machines on top of a shared physical infrastructure, and thus enabling Network Slicing [2]. Network Slicing allows the creation of virtualized, isolated, and customized network instances on top of a common physical infrastructure for specific services or use cases.

As 5G networks rely on virtualization and cloud-native technologies, Kubernetes becomes a fundamental tool for achieving the goals of flexibility, scalability, and quick adaptation to changes. The integration of Kubernetes into 5G Networks allows network operators to deploy and manage several Network Functions as containerized applications [4]. These Network Functions can include core network elements, Multi-Access Edge Computing (MEC) nodes, virtualized Radio Access Network (RAN) components, and more. By using Kubernetes, these functions can be dynamically orchestrated and scaled based on demand, ensuring efficient resource utilization and responsiveness to changing network conditions [4].

However, the introduction of these technologies and new specific components to the network architecture comes with new security threats [5]. Furthermore, due to a very high number and variety of devices connected to the network in combination with the new storage and compute capacities, this makes 5G a demanding infrastructure to ensure business continuity and a significant target for malicious attacks to exploit connected devices [5] [6].

1.2 Motivation

With the emergence of 5G Networks, the digital world is ready to experience unprecedented growth in connectivity and services. However, this rapid evolution also brings new and intensified challenges in the domain of cybersecurity. Among these challenges, Distributed Denial of Service (DDoS) attacks come up as a very serious and dangerous threat, capable of causing significant service disruption for subscribers [7]. Historically, such attacks have proven to be highly effective and have caused significant harm to individuals, businesses, and the overall economy. 5G networks are expected to support a vastly larger number of connected devices compared to previous generations of cellular networks, with a significant proportion of these connections involving IoT devices. However, if these types of devices that are vulnerable to security management are infected with malicious code, which triggers large-scale DDoS traffic, they may have a direct impact on the 5G network [7]. Furthermore, in 5G networks, the interoperation between components was unified using web connections based on HTTP, and the openAPI facilitated internal communication service functions and data access for service providers of vertical industry, such as IoT and factory automation [7] [8]. However, by providing API functions to the outside, and since hackers are familiar with the Internet Web technology used in the new 5G architecture and web apps still have security weaknesses, attackers may choose to use it as their preferred method of carrying out the attacks [7]. Therefore, managing vulnerabilities in existing web apps and controlling access to openAPIs will be important. Therefore, given the vulnerability of 5G Networks to DDoS and the significant consequences of such attacks, a robust and proactive security approach becomes highly significant.

1.3 Goals and Contributions

As already mentioned, the main goal of this research work is to address and mitigate critical security challenges posed by DDoS/flooding attacks in 5G Core networks. So, this internship will focuses on two lines of action: validation of the 5G testbed under development from a security point of view and assistance on the implementation of a security mechanism for the detection and prevention of flooding attacks. The validation and refinement processes of the security architecture for 5G environments will cover the following goals:

- Perform a Penetration Testing to discover new types of attack vectors and weaknesses in the 5G version being used in the testbed;
- Evaluate the Resilience of the testbed against DDoS/Flooding Attacks;
- Implementation of mitigation/detection mechanisms for DDoS/flooding Attacks;
- Validation of the effectiveness of the security solution.

Furthermore, this research work offers significant contributions to the field of Network Security. Firstly, it addresses different DDoS attack vectors and their impacts on 5G Network connectivity. Secondly, it focuses on mitigating DDoS/flooding attacks through the development and implementation of detection and prevention mechanisms. Additionally, it serves as a reference to contribute to the secure deployment of a 5G System in Kubernetes clusters. By implementing and validating security solutions, it enhances the resilience and trustworthiness of the 5G Network environment, reducing vulnerabilities and cyber risks. This work is integrated in the project POWER, whose aim is to create an innovative portfolio of products and services, aligned with four strategic technological transformation vectors: 5G networks, Edge/Cloud computing continuum, data-driven technologies and business models, and artificial intelligence.

1.4 Document Structure

This Document is organized as follows:

- **Planning:** Includes the planning for this research work, including the methodology used and the duration of each performed task.
- **Background Knowledge:** Presents some concepts and knowledge that were needed to acquire in order to have a better understanding about the theme of this research work.
- **State of the art:** Describes the current security solutions and mechanisms used for 5G environments, as well as the security problems and threats related to 5G Systems. It also presents research works and strategies used to mitigate and detect DDoS/flooding attacks in the field of cybersecurity.
- **5G Testbed:** Contains the documentation of the 5G Testbed under development and the initial experiments, where we understand how to interact with 5G services and select our target.
- **Penetration Testing:** Here, we conducted a Penetration Testing to find security weaknesses in the testbed.

- **Proposed Solution and Implementation:** Addresses the proposed security solution and how it was implemented.
- **Conclusions and Future Work:** Recap of the Research work and address important aspects to be explored in the Future.

Chapter 2

Methodology and Planning

This chapter will present the planning of the tasks executed during both semesters, the methodology used for the Security Testing and the Management of Risks associated with this research work.

2.1 Methodology

The methodology employed in this research work followed a structured approach to comprehensively assess and enhance the security of the 5G testbed under development. The research started with the implementation of the same testbed and the execution of the Initial Experiments, which played a crucial role in providing a deep understanding of its architecture, attack surface, understanding our target and designing an attack plan.

Subsequently, we conducted a Penetration Testing to try to exploit vulnerabilities in the system by simulating an attacker attempting to compromise the testbed, in order to identify potential weaknesses. For this phase, we are going to use the **Penetration Testing Execution Standard (PTES)** [9] methodology, which works as a pentest guideline to enable testers to perform an effective penetration testing. It consists of seven main phases:

- **Pre-Engagement Interactions:** The scope and goals of the security testing are defined, including the targets that a penetration tester will focus;
- **Intelligence Gathering:** The fist stage of the actual engagement and where data about a target is collected, such as DNS records, IDS/IPS events, possible open Ports, and so on, depending on the type of the target;
- **Threat Modeling:** The penetration tester develops a threat model that outlines the potential attack scenarios;
- **Vulnerability Analysis:** Involves identifying potential vulnerabilities and weaknesses that can be used to gain unauthorized access or compromise the target;
- **Exploitation:** There is an attempt to exploit the identified vulnerabilities;
- **Post-Exploitation:** The activities conducted once a system has been compromised;

- **Reporting:** The last phase and provides a discussion about the main findings and issues to be addressed.

Following the Penetration Testing, the next phase involved the implementation of a security solution designed to mitigate the attacks identified during the Penetration Testing. This implementation primarily focused on addressing DDoS/flooding attacks, which were the primary objective of this research work. Additionally, we validated the effectiveness of this security solution.

2.2 Work Plan

2.2.1 First Semester

The first semester was focused on studying and understanding the theme of this research work and recognize potential security problems in 5G environments. The main tasks performed during the 1st Semester were:

- Obtain background knowledge about 5G and related technologies
- Study State of the Art
- Implementation of the Testbed
- Write the intermediate report

The next image illustrates the work carried out in the first semester:

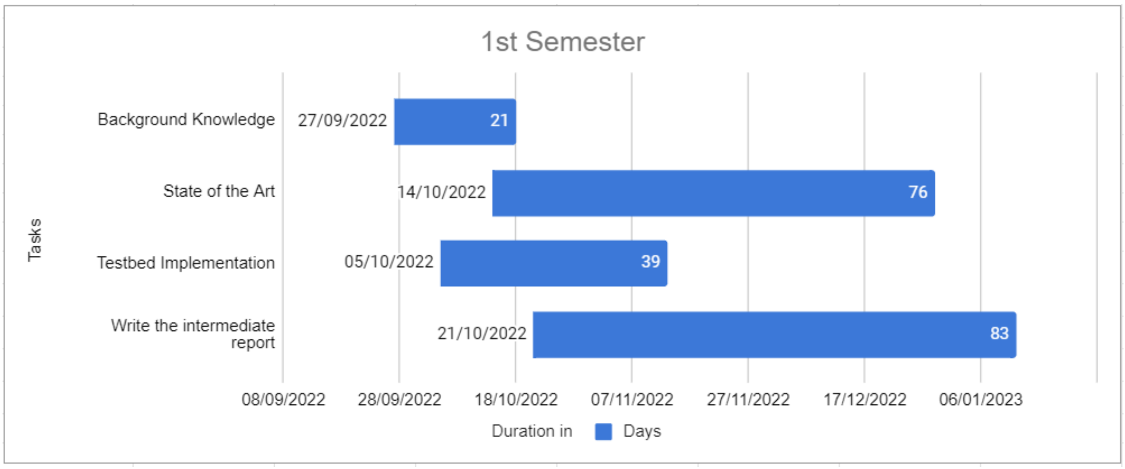


Figure 2.1: Work plan for 1st Semester

It is important to note that the implementation of the testbed was performed based of a research work in the same line of investigation, and for this reason there was no need for such an in-depth investigation of the tools used for the same testbed. The study of the state of the art was focused on understanding 5G Security, potential problems in the same regard, tools that can be used to perform the penetration testing of the platform and the implementation of the attacks, and finally mitigation methods against DDoS/Flooding attacks.

2.2.2 Second Semester

In order to elaborate a better planning of the 2nd Semester, we decided to break down the Main Tasks of the Research Work into sub-tasks. By breaking down complex objectives and goals into smaller, focused sub-tasks, we can allocate resources, time, and efforts more effectively. Each sub-task represents a specific actionable step, making it easier to track progress, identify dependencies, and ensure that the project stays on course.

The Figure 2.2 shows the planning of the sub-tasks to be carried out during the **Document Corrections** phase after the intermediate presentation, taking into account the valuable feedback and suggestions received.

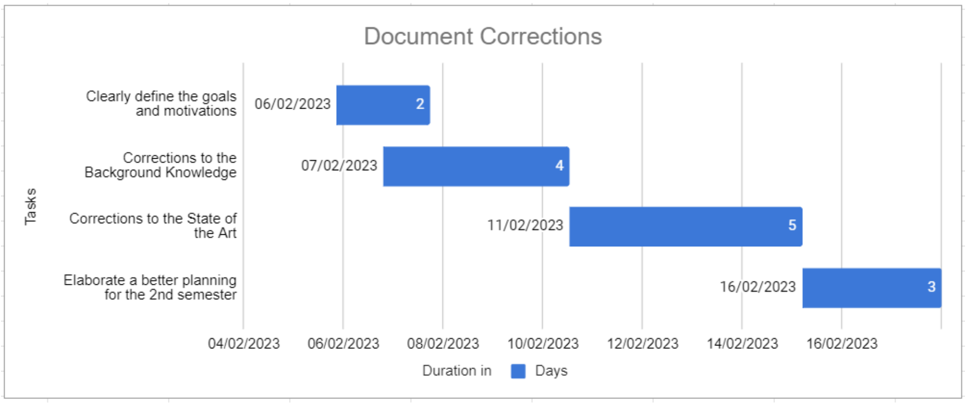


Figure 2.2: Tasks during the Document Corrections

The next Figure 2.3 presents the tasks regarding the **Initial Experiments** of the platform. The goal of this task was to learn more about the 5G version being used in the testbed, learn how to interact with it and develop an attack plan to be executed. With this knowledge, we could start the Penetration Testing.

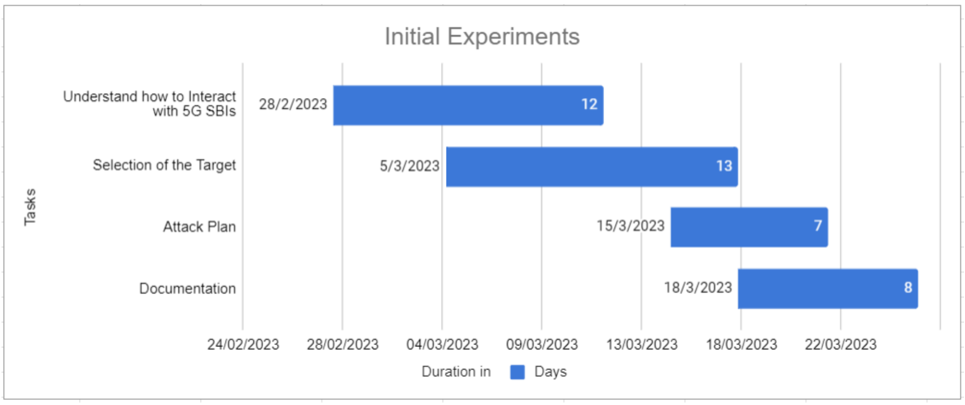


Figure 2.3: Tasks during the Initial Experiments

The Figure 2.4 outlines the tasks involved in the **Penetration Testing** process conducted in our testbed. The purpose of this task was to find vulnerabilities in the security of the testbed, and evaluate the impact of DDoS/flooding attacks.

The Figure 2.5 presents the tasks involved in the **Implementation of the Security Solution** in the Testbed. Building upon the insights gained from the Penetration Testing, these

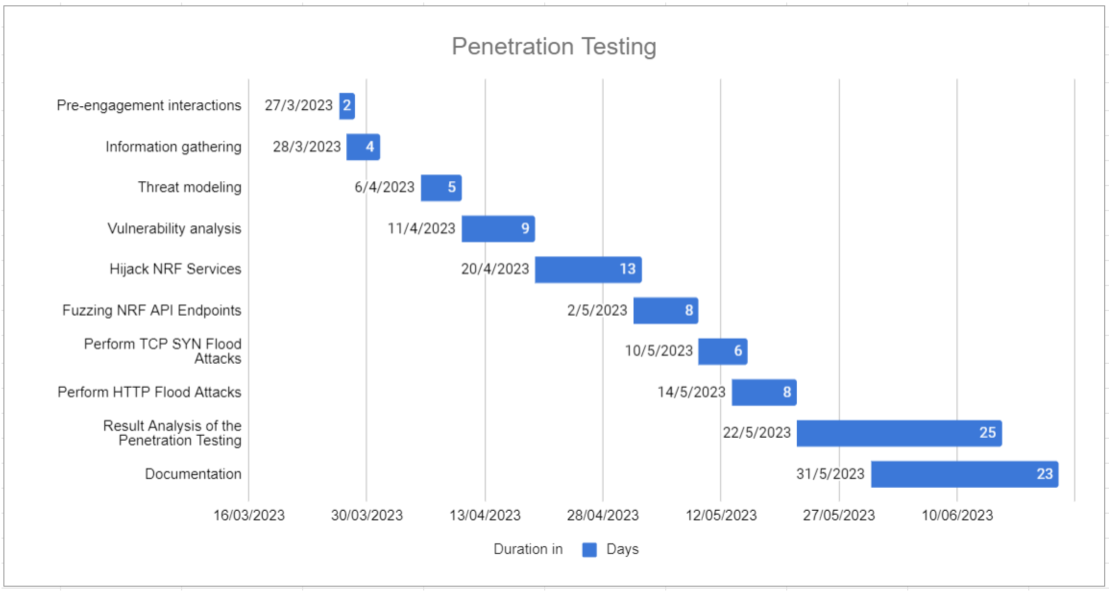


Figure 2.4: Tasks during the Penetration Testing

tasks focus on developing effective mechanisms to mitigate the identified vulnerabilities and enhance the overall security of the testbed. Additionally, we conduct several types of flooding attacks to validate the effectiveness of the implemented security measures.

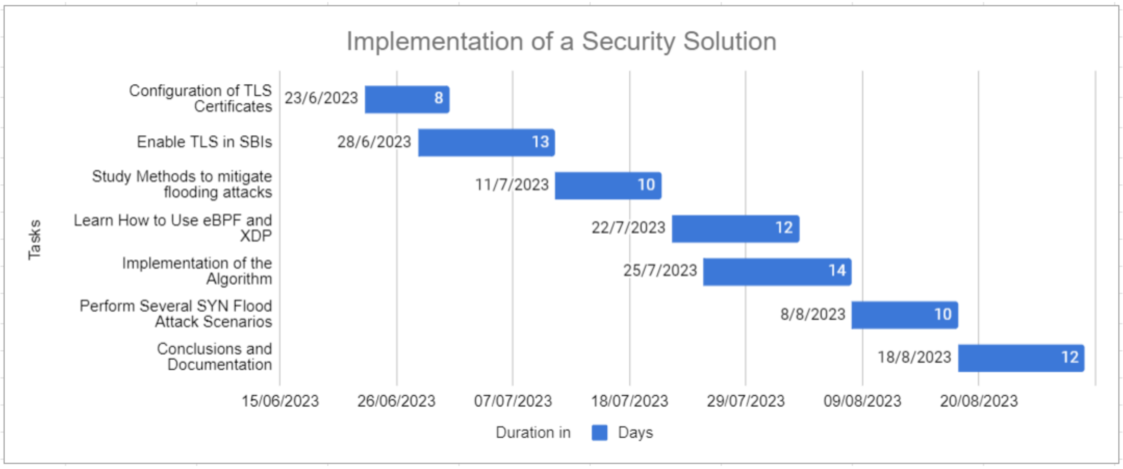


Figure 2.5: Tasks during the Implementation of a Security Solution

2.3 Risk Management

As part of the research work focusing on security testing, it is crucial to conduct a comprehensive risk analysis to identify and address potential risks that may impact the effectiveness and success of the security testing efforts.

2.3.1 Risk Identification

This research work is dedicated to security testing within an established testbed environment. Initially, we had no knowledge about the technologies utilized in developing this testbed or how to operate them. While constructing the entire testbed step by step would grant us basic familiarity with component interactions, potential vulnerabilities, and configurations, it would also introduce added complexity and potentially consume substantial time and effort to learn these technologies. This diversion could shift our attention from the main focus of the research: studying security in 5G Networks. Additionally, given that most of these technologies are relatively new and open-source, they might introduce more additional risks.

The Identified Risks are listed below:

- Difficulties in the testbed implementation;
- Inadequate/Incomplete specifications for the 5GC APIs;
- Lack of comprehensive documentation for several open-source tools;
- Challenges in incorporating new mechanisms into the existing code;
- High learning curve associated with the utilized technologies;
- Limited online resources for fairly recent technologies could make it more challenging to find solutions and information;
- Compatibility issues between open-source tools;
- Lack of consistent maintenance and updates of some open-source projects.

2.3.2 Risk Classification and Evaluation

In order to manage the identified risks, a systematic assessment is conducted to evaluate the impact, probability, consequences, and appropriate mitigation strategies. Risks are classified into two key categories: Impact and Probability.
Impact refers to the extent of the consequences that a risk might have on the project if it were to occur. Impact can be rated as "Low", "Medium", or "High", reflecting the possible extent of damage or disruption in the research work.
Probability assesses the likelihood of a risk actually occurring. Probability is also categorized as "Low", "Medium", or "High", indicating the chances of the risk happening.

The evaluation of the identified risks is represented in the following tables:

Risk ID	1
Description	Difficulties in the testbed implementation
Impact	High
Probability	Low
Consequences	Delayed project
Mitigation Strategy	Ask for guidance and assistance

Table 2.1: Difficulties in the testbed implementation

Risk ID	2
Description	Inadequate/Incomplete specifications for the 5GC APIs
Impact	High
Probability	Low
Consequences	Delayed project, limited testing coverage
Mitigation Strategy	Study the source code of the 5G application and understand how to interact with the APIs, perform security testing techniques to discover endpoints

Table 2.2: Inadequate/Incomplete specifications for the 5GC APIs

Risk ID	3
Description	Lack of comprehensive documentation for several open-source tools
Impact	Medium
Probability	High
Consequences	Increased learning time, potential misconfigurations
Mitigation Strategy	Engage online communities, search tutorials and guides

Table 2.3: Lack of comprehensive documentation for several open-source tools

Risk ID	4
Description	Challenges in incorporating new mechanisms into the existing code
Impact	High
Probability	Medium
Consequences	Performance issues, delayed project, inability to perform some tasks
Mitigation Strategy	Ask for assistance, engage online forums

Table 2.4: Challenges in incorporating new mechanisms into the existing code

Risk ID	5
Description	High learning curve associated with the utilized technologies
Impact	Medium
Probability	Medium
Consequences	Slower development, delayed project
Mitigation Strategy	Dedicate time effort for learning, ask for assistance

Table 2.5: High learning curve associated with the utilized technologies

Risk ID	6
Description	Limited online resources for fairly recent technologies could make it more challenging to find solutions and information
Impact	Low
Probability	High
Consequences	Incomplete understanding, harder to learn how to use these technologies
Mitigation Strategy	Read the documentation, engage forums, ask for assistance

Table 2.6: Limited online resources for fairly recent technologies could make it more challenging to find solutions and information

Risk ID	7
Description	Compatibility issues between open-source tools
Impact	High
Probability	Medium
Consequences	Delayed project
Mitigation Strategy	Investigate other tools, engage online communities

Table 2.7: Compatibility issues between open-source tools

Risk ID	8
Description	Lack of consistent maintenance and updates of some open-source projects
Impact	High
Probability	Low
Consequences	Incompatibility between versions, project delays
Mitigation Strategy	Communication with the owners of the project

Table 2.8: Lack of consistent maintenance and updates of some open-source projects

2.3.3 Risk Prioritization

In order to prioritize risks, we designed a matrix (impact x probability), represented in Table 2.9 to enable us to rank risks according to their potential consequences and the likelihood of occurrence.

		Impact		
		Low	Medium	High
Probability	High	6	3	
	Medium		5	4,7
	Low			1,2,8

Table 2.9: Risk Matrix

After conducting an analysis of the risk matrix, the prioritization of risks becomes evident. It is apparent that risks 3, 4, and 7 hold the highest priority due to their relatively higher combined Impact and Probability scores. Following closely is risk 5, and then risks 1, 2, 6 and 8 are positioned with lower combined scores. This informed prioritization guides the sequential implementation of risk mitigation strategies, ensuring that the most significant risks are addressed first.

Chapter 3

Background Knowledge

This chapter will present the studies that were made in order to understand the theme of this research work. Before we can dive into the security of 5G, we need to understand what is the 5G, how it is structured and how it works. Thus, some important concepts about the 5G architecture will be presented.

3.1 Introduction

A mobile cellular network aims to provide wireless connectivity to devices. These devices, described as User Equipment (UE), correspond to a laptop computer, a smartphone, but increasingly including cars, drones, industrial and agricultural machines, and so on. Any mobile cellular network architecture consists of two main subsystems: The Radio Access Network (RAN) and the Core Network. The RAN corresponds to a distributed set of multiple base stations, also known as Next Generation Node B (gNB) in 5G, and it provides the technology to connect the UE to the Core Network. The Core Network provides several services to the subscribers who are connected by the access network, including authentication, mobility tracking, session management and more. The fourth generation of mobile networks, 4G, had a significant impact in the history of wireless communication, enabling faster data speeds and improved connectivity. However, as the requirements for higher data rates and more scabalable and flexible networks continued to grow, 4G began to face limitations in meeting such demands. This led to the development of 5G. However, since many companies in the telecommunication industry required a fast and less disruptive way to start initial 5G services, instead of relying on a new, less familiar, 5G Core Network architecture, it was developed a solution that would maximize the existing 4G/LTE infrastructure [10]. This solution consists of a Non-Stand Alone (NSA) architecture, where the RAN operates on the existing Evolved Packet Core (EPC), used for 4G/LTE Networks, reinforced with several functionalities to support 5G. Thus, this architecture includes a 4G and a 5G base station, but the 4G base station is more relevant, with 5G NSA still operating with the Core of 4G/LTE Networks. However, this approach brings many disadvantages to the network, namely in terms of latency and speed which are the main requirements for a 5G Network. Differently from the NSA, the Stand Alone (SA) architecture uses the **5G Core**, which is able to **unleash a fully and true 5G network** that performs essential 5G functions.

3.2 5G SA Architecture

3.2.1 5G Architecture Overview

The New Generation Radio Access Network (NG-RAN) is a network infrastructure that consists of a set of multiple gNBs connected to the 5G Core via the NG interface, with the goal of providing several services to the UE. In 5G, 3GPP introduces a split of the 5G base station, the gNB, into gNB-Central Units and gNB-Distributed Units. The gNB-Central Unit controls the operation of multiple gNB-Distributed Units via the F1 interface. This slit architecture can bring several benefits, namely in terms of coordination for performance features, load management, real-time performance optimization, and enables NFV/SDN. Furthermore, the gNB-Central Unit Control Plane part (CU-CP) is connected to the gNB-Central Unit Data Plane part (CU-DP) through the E1 interface. In Figure 3.1, we can see a simplified scheme of the F1 and E1 interfaces between components of the split RAN and the 5G Core.

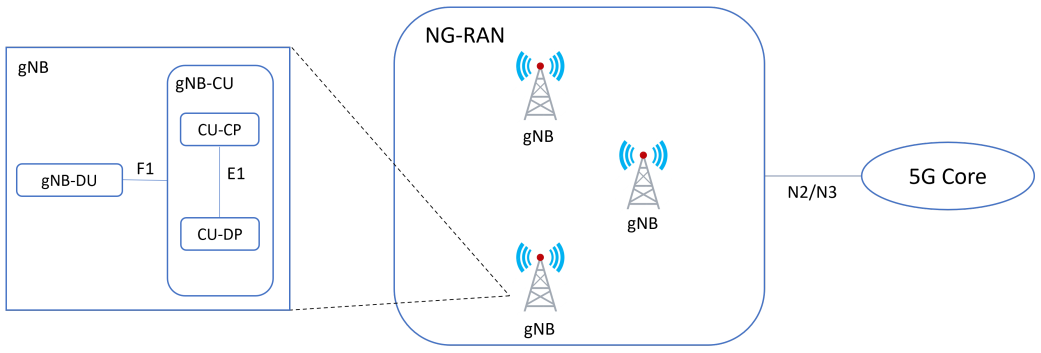


Figure 3.1: NG-RAN Architecture

The 5G Core network, brings evolutionary changes to the Core network with a modular and Cloud-native approach. One key advancement is that it upgrades the traditional monolith architecture of the Core Network of previous generations to a new Service Based Architecture (SBA), enabling more flexible service development. Introduced to improve the modularity of the network system, the SBA lets Network Functions (NFs) in 5G communicate with each other, exposing and making services available to each other. They employ a Representational State Transfer (REST) interface using the HyperText Transfer Protocol (HTTP)/2 protocol.

The schema of the 5G Core architecture represented in the Figure 3.2 can be divided into two simpler concepts: The **User Plane**, also known as the **Data Plane**, which is only composed by the User Plane Function (UPF); The **Control Plane**, constituted by the Network Functions that were evolved from the 4G core network and subdivided into smaller components known as microservices. The processes of forwarding the data are done in the Data Plane, while the Control Plane is the part of the network that controls how the data is forwarded. Because of the separation between the User and Control Plane, this new approach comes with a new set of interfaces defined for the interaction between the Radio Access Network and the core network. These interfaces are referred to as N2 and N3: The N2 interface is the control plane interface between the RAN and the core network. Also, the signaling between the UE and the AMF is transported over the N2 connection for that UE; The N3 interface is the user plane interface between the RAN and the User Plane, being responsible for carrying the traffic from the RAN to the User Plane Function.

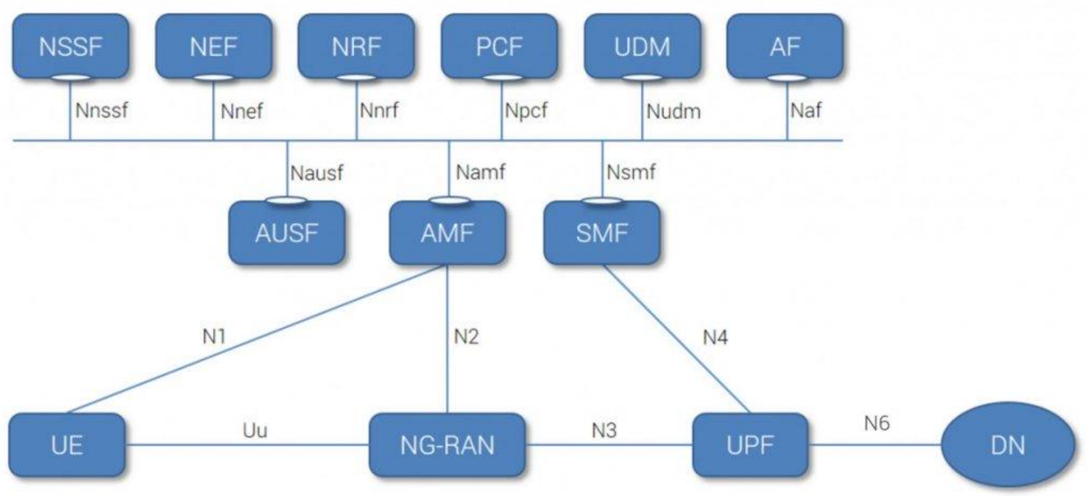


Figure 3.2: 5G Architecture[11]

In the next section, it will be explained more deeply each component inside the Core network, as well as the respective services provided by each NF.

3.2.2 The main 5G Core Functionalities

The core functionality of a network architecture includes ways for the establishment of sessions in a secure way and the forwarding of the user data between mobile devices that provide data connectivity. The core components of any 5G network are the following: Access and Mobility Management Function (AMF), Session Management Function (SMF), UPF, Authentication Server Function (AUSF), Unified Data Management (UDM), User Data Repository (UDR), Policy Control Function (PCF), Network Slice Selection Function (NSSF) and Network Exposure Function (NEF).

The **AMF** interacts with the UE via signaling through the N1 interface and the RAN along the N2 interface. Thus, the AMF aims to perform the following tasks: **Registration Management**, allowing the UE to register and authenticate with the 5G network; **Connection Management**, meaning that it establishes the control plane connections between the UE and the AMF; **Reachability Management**, ensuring that any UE is always reachable and available to be triggered whenever there is a need to establish a mobile-terminated connection; **Mobility Management**, maintaining knowledge of the UE's location within the network. It performs periodic updates that verify that the UE remains on the network (did not move out of range nor stop working).
It is important to note that the AMF does not perform the session management and the device authentication itself. Instead, it sends all the information related to the session management to the SMF and orders the authentication service from the AUSF.

The **SMF** communicates with the AMF, to send and receive all the information related to the session management of the UE to perform the establishment, modification and release of individual sessions and the assignment of IP addresses per session. It also selects and controls the different UPFs in the network across the N4 interface. Lastly, the SMF also interacts with the PCF, which provides policy rules to control the user sessions.

The **UPF** is responsible for the processing and forwarding of user data and is controlled by the SMF. It connects with the RAN and the Data Network (DN) along the N3 and N6 interfaces, respectively, acting as a interconnect point between them. Thus, it provides encapsulation and decapsulation of the data exchanged between the RAN and the DN, can guarantee QoS (Quality of Service), managing packet priority correctly in case of network congestion, and generate traffic usage reports to the SMF.

The **UDM** works with the user subscription data stored in the UDR. It is responsible for generating the authentication data used for authenticating the devices and authorizing access for specific users based on their subscription data.

The **UDR** is the database where multiple types of information are stored, such as the subscription data and the data that defines various types of network or user policies. Its services are used by other network functions, more specifically UDM, PCF and NEF.

The **AUSF** offers the service of authenticating a specific device to the AMF, using the authentication credentials created by the UDM. It also provides services to generate encryption keys that make information updates more secure.

The **PCF** interacts with several network functions, such as the SMF, to perform policy control of user sessions, and the AMF and UDR to allow the service providers to control the way a specific user can access the network, for example restricting the area within which the user can attach or move with retained connectivity.

The **NSSF** can be used by the AMF to assist in the selection of the Network Slice instances that will serve a particular device.

Lastly, the **NEF** provides security services and mechanisms to enable all these Network Functions to expose their functionalities to third-party developers and enterprises in a secure and robust manner. Therefore, it controls every type of data coming from external applications that want to access the internal data of the 5G core.

3.2.3 Service Registration, Discovery and Request

When two of the NFs communicate in the SBA, the one that sends the request has the role of Service Consumer, while the one that provides the service acts as Service Producer. However, this is not enough, since there is a need for a mechanism to identify a Service Producer that is available and can provide the requested services. This mechanism makes use of one of the architecture components represented in figure 3.2, the **Network**

Repository Function (NRF), which supports **Service Discovery** and allows other NFs to register and find each other through an API. Thus, each NF must contain the address to one or more NRFs, but never to any of the other Network Functions configured.

In order to clarify the communication between the NFs, we will take a practical example. Consider the Network Function AMF wants to request a service from the Network Function PCF. This process is divided into three steps: Service Registration, Service Discovery and Service Request.

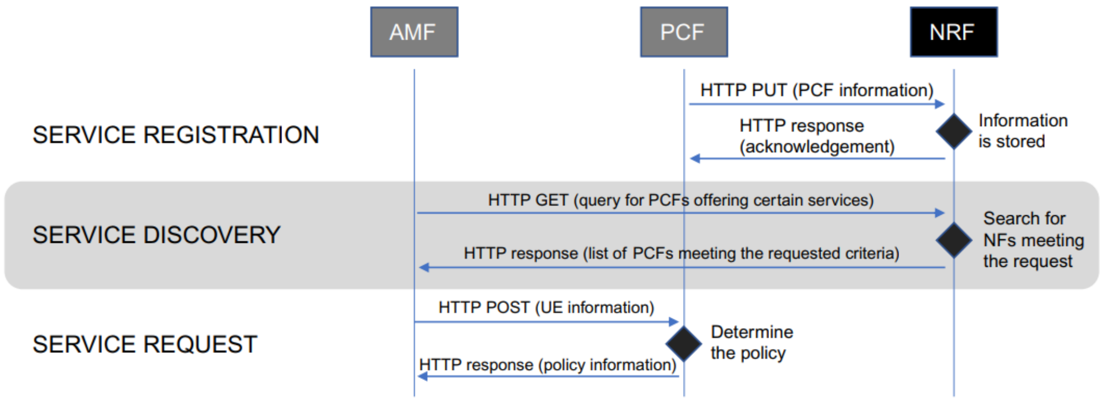

AMF
PCF
NRF
SERVICE REGISTRATION
HTTP PUT (PCF information)
HTTP response (acknowledgement)
Information is stored
SERVICE DISCOVERY
HTTP GET (query for PCFs offering certain services)
HTTP response (list of PCFs meeting the requested criteria)
Search for NFs meeting the request
SERVICE REQUEST
HTTP POST (UE information)
HTTP response (policy information)
Determine the policy

Figure 3.3: Call Flow[10]

In the Service Registration phase, the PCF is considered as a Service Consumer and the NRF as a Service Producer. The PCF sends a HTTP PUT message to the NRF with all the necessary information, such as network address, available services and identity. The NRF verifies if the request is valid, stores the information, and acknowledges the PCF registration with a response back to the PCF.
In the next phase, Service Discovery, it's the AMF that acquires the role of Service Consumer, as it sends a HTTP GET message to the NRF, stating what sort of Network Function it is asking for that provides the services that it needs. Then, the NRF filters out all Network Functions that are registered and are providing the requested services, and then responds back to the AMF.
Finally, it proceeds to the third and last phase, Service Request. The AMF stays as a Service Consumer, while the PCF turns into the Service Producer this time. The Service Consumer sends a HTTP POST message to the Service Producer asking for the service it needs. The PCF determines the applicable policy that is requested by the AMF and responds back with an HTTP response.

3.2.4 Transport Layer Security Overview

To enhance the communication security between 5G Network Function SBIs, including HTTP requests, the utilization of Transport Layer Security (TLS) is mandatory. TLS is a cryptographic protocol designed to provide secure communication over a computer network. It ensures that data exchanged between two systems, such as a client and a server, remains private and did not get tampered during transmission by using encryption and verification methods. Thus, TLS is commonly used to secure sensitive data transmitted over the internet, such as passwords, credit card information, and other personal or confidential data. The initiation of each TLS connection involves a TLS handshake, using asymmetric encryption. Once the handshake is successfully completed, symmetric encryption is applied, ensuring reliable communication by maintaining authenticity and privacy.

Asymmetric encryption, illustrated in Figure 3.4, employs a public key-private key pairing, which means that data encrypted with the private key can only be decrypted with the respective public key, and vice versa. The process of sharing public keys does not have to be secure, but the fact that the public key is mathematically related to the private key means that much larger key sizes are required, so it is unfeasible to derive the private key from the public key. As a result, this makes asymmetric encryption computationally intensive and too slow for many purposes.

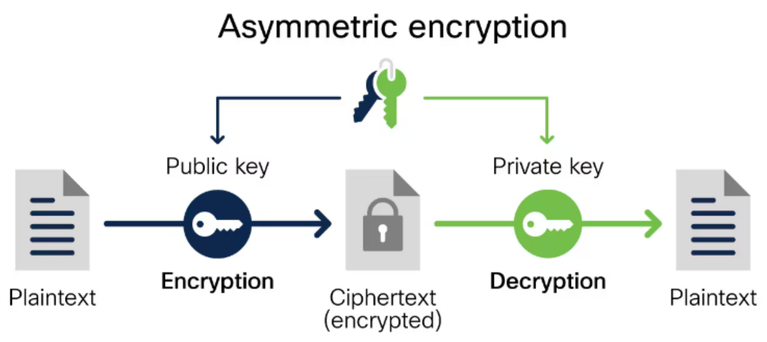


Figure 3.4: Asymmetric Encryption[12]

Symmetric encryption, illustrated in Figure 3.5, involves encrypting and decrypting data using a shared secret key between the client and the server. Because of this, While asymmetric encryption is considered more secure due to the use of two separate keys, symmetric encryption is much faster than asymmetric encryption. With TLS using asymmetric encryption to generate and exchange a session key securely, and symmetric encryption to secure the data transfer, this provides a good compromise between performance and security when transmitting data between two entities over the internet.

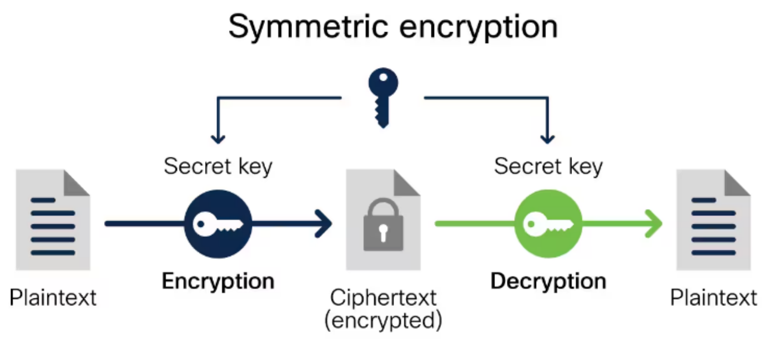


Figure 3.5: Symmetric Encryption[12]

The TLS handshake is crucial for establishing a secure connection as it enables the client and server to agree on encryption algorithms and create session keys for private communication. It is worth noting that the TLS handshake process may vary depending on the

TLS version in use. However, in most cases, the TLS handshake follows the following steps:

- **ClientHello:** The client initiates the handshake by sending a "ClientHello" message, which includes details such as supported cryptographic algorithms and TLS versions.
- **ServerHello:** The server responds with a "ServerHello" message, indicating the chosen encryption algorithm, TLS version and its certificate.
- **Public Key Extraction:** The client verifies the certificate with a Trusted Root Certification Authority to ensure the certificate is legitimate and extracts the public key of the server from the certificate.
- **Asymmetric Encryption:** The client generates a random session key and then uses the public key of the server to encrypt the session key and send it securely to the server.
- **Session Establishment:** The server decrypts the client communication with its private key, and the session is established.
- **Symmetric Encryption:** The session key is now used to encrypt and decrypt data transmitted between the client and server.

After the TLS handshake is completed, both parties are authenticated and the exchanged data will be encrypted and remain confidential. In the 5G Core Network, when Network Functions interact through API calls, the TLS handshake ensures both Network Functions are authenticated and the exchanged information is encrypted and remains confidential.

3.3 Enabling 5G Core Network Functionality

3.3.1 SDN and NFV

Despite all the advantages that 5G networks bring to mobile communication systems, the 5G network also faces some challenges when used in practice. One of these problems is that the ability to connect a wide variety of devices that use different operating systems and protocols substantially increases the complexity of the structure of a 5G network, making it difficult to manage. And this is where the concept of an SDN comes in. In traditional networks, dedicated hardware devices (routers, switches, base stations, etc.) are used to control and configure network traffic, where the control plane and the data plane are physically present within the same hardware. SDN however, allows to control the network via software, by separating the logical part (control plane) from the hardware. An example of a SDN architecture is represented in Figure 2.4 and it comprises three layers: The Application Layer, which is composed of the applications and services running in the network; The Control Layer (Control plane), constituted by the SDN controllers, which typically run on a server and use protocols to tell switches where to send packets; The Infrastructure Layer (Data plane), constituted by the networking devices, which receive information from the controller about where to move the data. The communication between the SDN controller and data plane is through southbound APIs, and the communication between the SDN Controller and the application layer is through northbound APIs to make the information built from the SDN Controller available for applications.

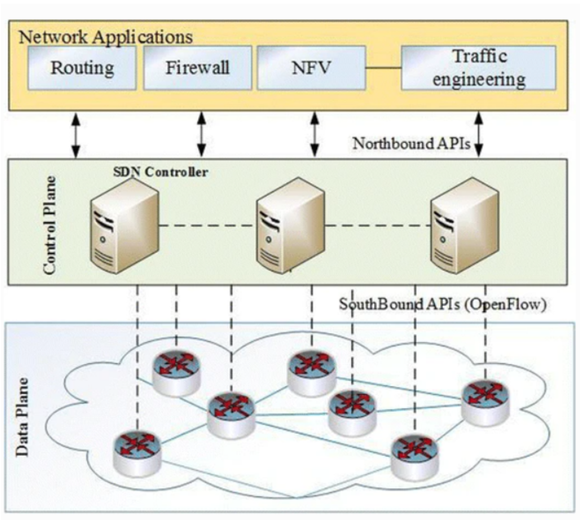


Figure 3.6: Example of an SDN Architecture [13]

One of the key elements that facilitate SDN adoption is the growing use of Network Function Virtualization. SDN and NFV technologies are complementary, with NFV providing many of the actual services managed in a SDN. SDN focuses on the control plane while NFV focuses on optimizing the actual network services managing data flows.

3.3.2 Multi-Access Edge Computing

The increased amount of data generated by connected devices have exceeded the resources of network and infrastructure. Sending all this data to a centralized data center or to the cloud might cause bandwidth and latency issues. The flexibility provided by the split between User and Control Plane allows the User Plane Functions (UPF) to be moved to the edge, which makes Multi-Access Edge Computing (MEC) possible. MEC offers a more efficient alternative, as it brings network capabilities closer to end users, meaning that data is processed and analyzed closer to where it was created, and supports distributed deployment. Because data does not have to travel over a network to be sent to a cloud or data center for processing, latency is significantly reduced. MEC on 5G networks enable faster and more comprehensive data analysis, faster response times and enhanced customer experience. This is relevant for applications that require real-time processing and low latency, such as augmented reality, autonomous vehicles and industrial automation.

3.4 DDoS Attacks

A Distributed Denial of Service (DDoS) attack happens when a large number of compromised machines, including computers and other networked resources such as IoT devices, work together to overwhelm an online service (for example a website) or its surrounding infrastructure, making it really difficult for normal and legitimate users to access and use that service. From a high-level view, a DDoS attack is like a big crowd blocking a doorway, stopping anyone from getting in.

3.4.1 How it Works

In order for the attacker to compromise other computers to get involved in the DDoS attack, the attacker normally uses malicious software. The attacker develops a malware program and distributes it over the Internet and puts it on websites, email attachments and software. So if a vulnerable machine goes to these infected websites or opens these email attachments, the malware will be mistakenly downloaded on the machine, without the user knowing that their machine has been infected. Now, the attacker has the control of an army of infected computers, ready to receive instructions from the attacker. Then, in a specific time, the attacker will send a command to all these computers to start overwhelming the intended server, exhausting its resources and making it unable to operate, causing a denial of service.

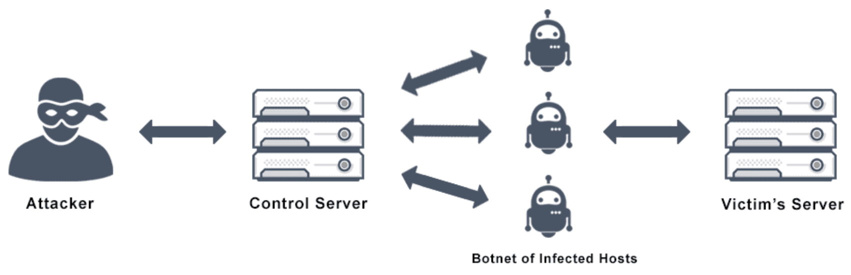


Figure 3.7: Example of a DDoS Attack [14]

3.4.2 Types of DDoS

DDoS attacks happen in diverse forms, each with its own distinct strategy and impact. These attack types capitalize on different vulnerabilities and exploit several aspects of network architecture. Understanding these variations is crucial for developing effective defense strategies against DDoS threats. Typically, they can be classified into one of three different types: Volume Based Attacks, Protocol Attacks, and Application Layer Attacks.

Volume Based Attacks

This category of attacks attempts to create congestion by consuming all available bandwidth between the target and the larger Internet. Large amounts of data are sent to a target by using a form of amplification or another means of creating massive traffic, such as requests from a botnet. An example of this type of DDoS is the **DNS Amplification attack**.

A DNS amplification attack is a type of DDoS attack that takes advantage of the behavior of the Domain Name System (DNS) and the openness of certain DNS servers. In this attack, the attacker sends small requests to publicly accessible DNS servers, using a spoofed IP address that matches the victim address. These servers then respond with much larger responses, which are sent to the victim IP address. Because the responses are much larger than the original requests, the attacker can amplify the volume of traffic sent to the victim, overwhelming their resources and causing service disruptions. This type of attack exploits the asymmetry between the request and response sizes in the DNS protocol.

Protocol Attacks

Protocol attacks, sometimes called state-exhaustion attacks, cause a service disruption by using too much of the server resources or the resources of network equipment like firewalls and load balancers. Protocol attacks utilize weaknesses in layer 3 and layer 4 of the protocol stack to render the target inaccessible. An example of a protocol attack is the **SYN flood attack**.

The SYN Flood attack is a type of cyber attack that targets the Transmission Control Protocol (TCP) handshake process aiming to overwhelm a targeted server or network with a large number of SYN (synchronize) requests. TCP operates at the transport layer of the TCP/IP protocol suite and is responsible for ensuring reliable and ordered delivery of data packets, enabling application programs and computing devices to exchange messages over a network. Thus, before two entities can begin their information sharing within the network, a successful TCP connection must be established. The process for establishing a connection with the TCP protocol is as follows:

- **SYN:** The "SYN" (Synchronize) packet is the initial segment sent by a Client to initiate a TCP connection with a Server. It indicates that the client wants to establish a connection and includes a randomly chosen synchronize sequence number.
- **SYN + ACK:** The "SYN + ACK" (Synchronize + Acknowledge) packet is the response of the Server to the SYN packet, to acknowledge that a Client wants to initialize a connection. This packet also contains its own sequence number.
- **ACK:** The "ACK" (Acknowledge) packet is used to confirm the reception of the "SYN + ACK" packet, establishing a reliable connection with which they will start the actual data transfer.

However, in a SYN flood attack, the client sends overwhelming numbers of SYN requests and intentionally never responds to the server SYN-ACK messages. This leaves the server with open connections awaiting further communication from the client. Each is tracked in the server TCP connection table, eventually filling the table and blocking any more connection attempts from any source.

Application Layer Attacks

Application layer attacks target the top layer of the network protocol stack, focusing on the way applications and services work. These attacks are particularly sneaky because they mimic legitimate user behavior and can be difficult to detect. One example of an application layer attack is the **HTTP Flood attack**, which can include GET requests or POST requests

In a HTTP POST Flood attack, the attacker sends HTTP POST requests to a target server, just like a regular user would when submitting information through a web form. However, these requests are sent very slowly, with each part of the request (headers, data, etc.) trickling in bit by bit, rather than all at once. This is done to hold open as many connections as possible and keep the server waiting for the request to complete. Thus, HTTP flood attacks using POST requests tend to be the most effective from the attacker perspective, since POST requests may include parameters that trigger complex server side processing. On the other hand, HTTP GET-based attacks are simpler to create, and can more effectively scale in a botnet scenario.

The server, while busy waiting, reserves resources for each of these slow connections. Since servers have a limit to how many connections they can handle simultaneously, an attacker can exhaust these resources by maintaining numerous slow connections. Eventually, this overload causes requests of legitimate users to be delayed or even rejected, leading to service disruptions.

3.5 Chapter Conclusions

This chapter helped us understanding how 5G works and how it differs from past network generations. It started by giving an introduction to the entry of 5G in telecom companies, addressing the two possible ways in which 5G can be deployed, NSA and SA. Then, the concept of Split RAN was discussed, which introduced new interfaces, E1 and F1, and the subdivision of the Network Function Mobility Management Entity (MME) used in 4G/LTE networks into smaller Network Functions, which, due to being physically separated from the RAN, introduces new interfaces, N2 and N3. Next, we provided an overview of the Network Functions of 5G Core addressing its respective services. In addition to that, it was shown how the Network Functions manage to find each other in order to provide the respective services and communicate under the SBA and how TLS is used to encrypt the SBI messages between components of the Core Network. Then, we explained how SDN, NFV and MEC technologies can support 5G networks, and lastly we provided an overview of different DDoS Attack vectors.
In the next chapter, we will introduce the security mechanisms specified by 3GPP, address security problems that may exist and mechanisms developed to improve security in 5G Networks.

Chapter 4

State of the Art

This chapter presents the result of the studies of the State of the Art executed during the first stage of this research work. It starts by introducing basic security concepts in mobile networks and address the security architecture of 5G systems defined by 3GPP. Then, it will address some security challenges and opportunities that softwarization and use of new technologies, such as SDN, NFV and MEC brings to 5G. Lastly, it is presented research works that focus in testing the security in 5G environments and developed mechanisms to improve 5G Security against flooding attacks.

4.1 5G Security Architecture

Before the user can access the network, the **authentication** must be performed. During that process, the user proves that he is, in fact, who he claims to be, where each party has access to a secret key or password that only the participating parties know about. In 5G Systems, mutual authentication is required, meaning the network authenticates the user and the user authenticates the network. The network also verifies that the subscriber is **authorized**, meaning the user must have the right privileges to access the requested 5G services using a particular access network [10]. The authentication and authorization processes are usually done at the same time, so the network has the knowledge of who the user is that is requesting the service [10].

Once the user has accessed the network, the signaling traffic and user plane traffic between the UE and the network has to be protected. **Ciphering** and **integrity protection** can be applied to achieve this goal [10]. With ciphering, we can encrypt and decrypt data. A cipher algorithm can convert the original message in Plaintext into Ciphertext using a cryptographic key, making the information unreadable to anyone who manages to intercept it, except for the intended receivers who have the access to the correct keys. With integrity protection, we can detect if the message received was modified by an attacker between the sender and the receiver. If the traffic has been modified, the receiver is able to detect it. In order to perform integrity protection, the parties who are exchanging these messages also need cryptographic keys, but never use the same keys that were already used for different processes. The reason behind this, is because if an attacker manages to recover the ciphering key by breaking, for example, the encryption algorithm, he would also be able to access the other services which use the same key. Also, the key used in one access should not be the same as the key used in another access. If an attacker obtains the key in one access with weak security mechanisms, he would also be able to reuse it to break accesses with stronger security mechanisms. **Key separation** is used to avoid this. This feature, controls the use of keys by separating them into unique types, allowing to use a specific type of key only for its intended purpose [10].

The 3rd Generation Partnership Project, or 3GPP, is the main body developing technical specifications for 5G networks, including security specifications. These specifications bring several security improvements in comparison to previous generations of networks. For an easier understanding of the different security features for 5G Systems, 3GPP divides the security architecture into different domains, since each one has its own set of security threats and solutions:

- **Network Access Security**: Enables a UE to authenticate and access services via the network securely. It also protects the signaling traffic and User Plane traffic in the access. This protection may provide confidentiality and integrity protection of the traffic.
- **User Domain Security**: Secures the access to the mobile equipment, for example, entering a PIN to use the SIM card.
- **Application Domain Security**: Enables applications in the user domain and in the provider domain to securely exchange data.
- **SBA domain security**: Security features regarding the Service Based Architecture, including the network element registration, discovery and authorization security aspects, and also the protection for the service-based interfaces.
- **Visibility and configurability of security**: Enables the user to be informed whether a security feature is in operation, using for example visual indicators or notifications on the UE that may show that the device is connected to a secure network or that data encryption is in use.
- **Network Domain Security**: Allow NFs to securely exchange data and protect against attacks on the network between the NFs, both between NFs within a Public Land Mobile Network (PLMN) and in different PLMNs.

In the next sections we will enter deeply into some of these security domains and understand the 3GPP security specifications for each one, addressing the respective protocols and security mechanisms.

4.2 Network Access Security

The 5G Network Access Security domain specified by 3GPP introduces a group of components contained within the 5G core network functions, more precisely the AMF, UDM and AUSF, described in Chapter 3. These components contain the security mechanisms needed to prevent attackers of accessing the information that traverses via signaling traffic and the user plane traffic, whenever someone is trying to access the network.

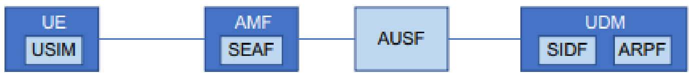


Figure 4.1: Network Access Security Architecture [10]

- **ARPF**: The Authentication credential Repository and Processing Function (ARPF) is located within the UDM and contains the subscriber's credentials and the subscription identifier Subscriber Permanent Identifier (SUPI);

- **SIDF**: The Subscription Identifier De-concealing Function (SIDF) is located within the UDM and is responsible for resolving the SUPI from the Subscription Concealed Identifier (SUCI);

- **SEAF**: The SEcurity Anchor Function (SEAF) is located within the AMF and it handles the authentication in the serving (visited) network, based on information received form the UE and AUSF.

- **AUSF**: The AUSF receives the authentication requests from the AMF and communicates with the UDM to obtain authentication vectors to perform the 5G-Authentication and Key Agreement (AKA) protocol and determinate if the authentication was successful.

4.2.1 Subscriber Privacy

In telecommunication systems, network operators allocate to each SIM card a unique identifier, the IMSI, also known as the SUPI in 5G. In previous generations of network, **the IMSI values were sent in plain text, unprotected**, which would allow an attacker to conduct man-in-the-middle and eavesdropping attacks. This vulnerability is now addressed in 3GPP security specifications for 5G. These include the use of permanent (SUPI), concealed (SUCI) and temporary (Global Unique Temporary Identifier (GUTI)) subscription identifiers, represented in Figure 4.2.

The SUPI is stored in ARPF from UDM and it consists of a 15 decimal digits string where the first 3 digits represent the Mobile Country Code (MCC) and the next 2 or 3 are equivalent to the Mobile Network Code (MNC), which identifies the network operator. The remaining 9 or 10 digits identify the individual user of that particular operator, known as the Mobile Subscriber Identification Number (MSIN). The SUCI is the encrypted version of the SUPI generated by the UE, using a ECIES-based protection scheme with the public key of the Home Network that was securely provisioned to the USIM during the USIM registration. Only the MSIN part of the SUPI is encrypted, since the MCC and MNC need to be sent in clear text to allow the serving PLMN to find the Home Public Land Mobile Network (HPLMN) in roaming processes. The GUTI is a core network temporary identifier allocated by the AMF to the UE and consists of the PLMN and the AMF identifiers to uniquely identify the AMF, and the Temporary Mobile Subscriber Identifier, which is the shorten version of the GUTI.

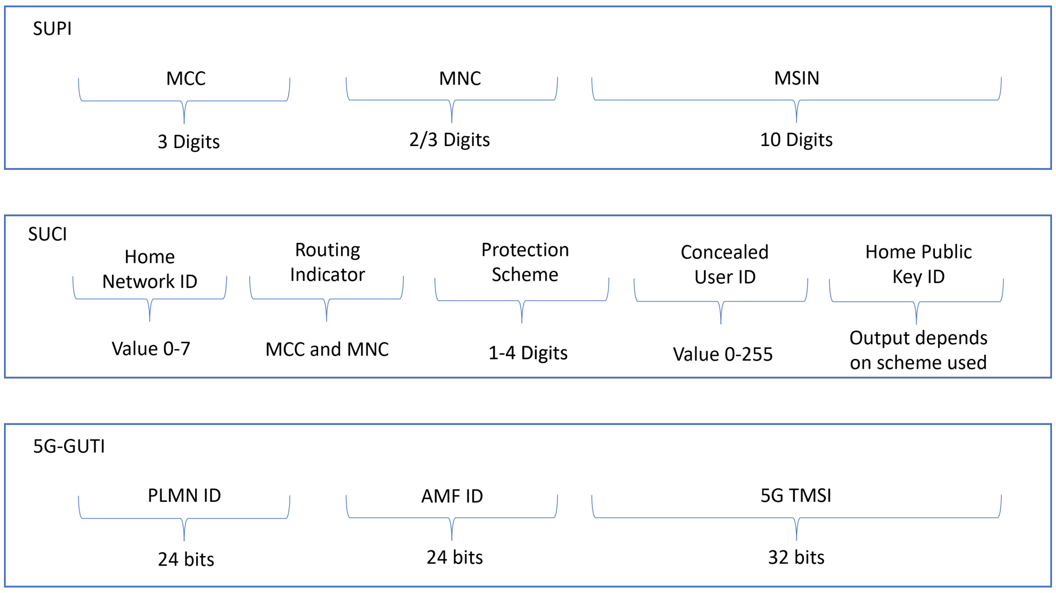


Figure 4.2: SUPI, SUCI and 5G-GUTI

4.2.2 Authentication Framework

The purpose of a primary authentication and key derivation is to enable mutual authentication between the UE and the network and provide keying material (KSeaf) that can be used between the UE and the serving network for subsequent security procedures. For the primary authentication, 3GPP introduces two authentication and key agreement protocol frameworks: EAP-AKA' and 5G-AKA.

One upgrade of the 5G-AKA and EAP-AKA' specified for 5G in comparison with the EPS-AKA and EAP-AKA' used in 4G is that both 5G-AKA and EAP-AKA' can be used for 3GPP access and non-3GPP access, whilst in 4G, the EPS-AKA is used for 3GPP access and the EAP-AKA' is used for non-3GPP access. Another upgrade is that 5GS adds home network control and allows the home network to be in charge of authentication instead of the serving/visiting network. With EPS-AKA, it is the MME that performs the authentication process from the serving network side, and it is the only one that verifies the result. The MME then notifies the HSS from the home network side about the result, without giving the chance for the home network to prove that the authentication was successful. With 5G-AKA however, on the side of the home core network, the Network Function that performs the authentication service is the AUSF. It uses services provided by the UDM and ARPF, which is responsible for selecting authentication methods and computing data and keyring material that AUSF needs. At the same time the SIDF is resolving the SUPI from the SUCI. Inside the serving core network, the key function is the SEAF, which is located within the AMF and works as proxy during the authentication process. The result of the primary authentication and key agreement procedure is an anchor key provided by the AUSF of the home network to the SEAF of the serving network. Keys for more than one security context can then be derived from the KSEAF without the need of a new authentication procedure.

4.2.3 User and signalling data

To ensure confidentiality protection of data exchanged between the UE and the base station, four ciphering algorithms are indicated: NEA0, 128-NEA1, 128-NEA2 and 128-NEA3. The 5G integrity protection algorithms currently supported for RRC, NAS signaling, and User Plane integrity protection are: NIA0, 128-NIA1, 128-NEA2 and 128-NEA3. According to the specification, support for the first three algorithms is mandatory in the UE, gNB, and AMF, while 128-NEA3 and 128-NIA3 are optional.

Name	Algorithm	Comments
NEA0	Does not provide encryption	
128-NEA1	128-bit SNOW 3G based algorithm	Also used in EPS
128-NEA2	128-bit AES based algorithm	Also use in EPS
128-NEA3	128-bit ZUC based algorithm	Also used in EPS

Table 4.1: Ciphering algorithms for NAS, RRC and UP ciphering in 3GPP access [10]

Name	Algorithm	Comments
NIA0	Does not provide encryption	
128-NIA1	128-bit SNOW 3G based algorithm	Also used in EPS
128-NIA2	128-bit AES based algorithm	Also use in EPS
128-NIA3	128-bit ZUC based algorithm	Also used in EPS

Table 4.2: Integrity protection algorithms for NAS, RRC and UP ciphering in 3GPP access [10]

A lack of confidentiality protection of data exchanged between the UE and gNB/AMF makes data vulnerable to interception, giving opportunities for attackers to track user location and conduct other passive or active attacks. As for the integrity protection we ensure that the data remains unchanged.

4.2.4 RAN interfaces

In 5G systems, the RAN architecture is usually divided into two different physical entities: Distributed Units and Central Units. This split introduces a new interface for establishing the communication between each one, the F1 interface. Each Central Unit communicates with each other using the E1 interface. These interfaces may carry important and sensitive types of data, thus 3GPP security specifications demands confidentiality and integrity protection for both E1 and F1 interfaces. With this in mind, IPSec ESP protocol according to IETF RFC 4303 shall be used [10].

Furthermore, in order to communicate using IPSec, the two parties need to establish the required IPsec Security Associations (SAs), which may include attributes such as the cryptographic algorithm and traffic encryption key. Although this can be done manually by configuring both parties with the required parameters, there are several scenarios where a dynamic mechanism for authentication and key generation is needed.

This is where IKEv2 comes into play, and so 3GPP has specified that support for IKEv2 certificate-based authentication shall be implemented.

The N2 and N3 interfaces connect the RAN with the AMF and the UPF. These interfaces also carry sensitive data and must be protected. Hence, 3GPP specifications also mandates for confidentiality and integrity protection using the IPSec ESP and IKEv2 certificate-based authentication for both interfaces.

4.3 Service Based Interfaces Security Domain

The Service Based Interfaces is a new concept in the 5G Network. Thus, 3GPP has defined security mechanisms to ensure the connection between the components of the core network [10]. For example, whenever an NF wants to request the service of another NF, the 5G system ensures that the authentication and authorization processes are performed before that service is considered available.

4.3.1 Mutual authentication and transport security

In order to secure communication between NFs, all the service based interfaces must support TLS 1.2 or 1.3. TLS is a **cryptographic protocol** that provides secure end-to-end communications between networks by encrypting data and authenticating connections that transport data across the internet. As we can see in the protocol stack represented in figure 3.2, it sits between HTTP/2 and TCP.

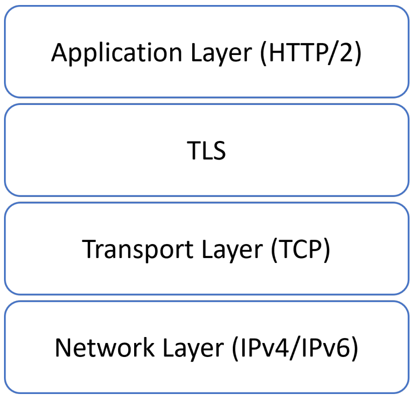


Figure 4.3: SBI Protocol Stack

4.3.2 Authorization of service requests

In addition to the authentication process between NFs, the Server side of a SBI also needs to authorize the client for accessing a service provided by a certain NF. This authorization uses the **OAuth 2.0** framework. This framework is a good way to guarantee authorization in a dynamic and virtualized deployment, as it enables third-party applications to access protected resources on behalf of a resource owner. For example, when a user is trying to access its Spotify account, he may use its Google or Facebook Account for a fast registration, enabling the user to skip the entire registration process and thus improving the customer experience. Furthermore, in the context of 5G SBA, authorization and access control is needed for the communication between NFs. Unlike the previously mentioned method, the OAuth 2.0 authorization process in this context does not involve a consent screen or require direct user interaction. Instead, OAuth 2.0 is based on a central authorization server, corresponding to the NRF, which handles authorization requests from the clients (Service consumers) and provides them access tokens after the authentication has been performed. Tokens are specific to a certain service producer, so when a service consumer tries to access a service, it sends a token with that request. The service producer verifies the token using the NRF key and depending on the result of this verification, the service producer NF may or may not provide the requested service and responds again to the service consumer NF.

The OAuth 2.0 is the standard framework when we consider that a NF wants to request services from another NF. However, since the NRF is the one which provides the OAuth 2.0 and the NF registration services, it plays a special role as a service producer when it interacts with another NF. Apart from the transport security and authentication (based on TLS), the OAuth 2.0 access token is not required, as the NRF authorizes the request based on the profile of the expected service between NFs and the type of the NF service consumer.[10]

4.3.3 Roaming protection

It is also important to talk about **Roaming protection**. Roaming is a process that allows the subscriber to access the network outside the range of its home network, through a visited network. This communication is guided by the signaling between the HPLMN and the Visited Public Land Mobile Network (VPLMN). The 5G Core introduces new mechanisms to improve its security, as well as new protocols for the signaling of the Roaming processes. Security is enabled by each Security Edge Proxy Protection (SEPP) inside both PLMNs. SEPP implements application layer security for all the layer information exchanged between two NFs, ensuring integrity and confidentiality protection. Also includes topology hiding to make sure that the internal network topology is not exposed to external networks [15].

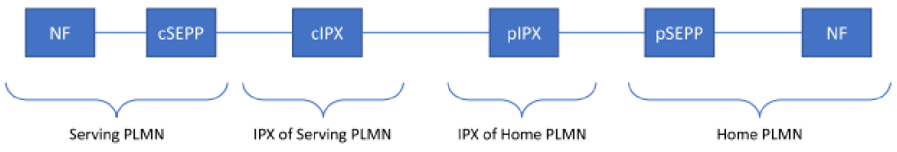


Figure 4.4: Overview of security between PLMNs (N32)[10]

Between the PLMNs, it is likely that there is also an Internetwork Packet Exchange (IPX). The IPX works as an intermediate network that provides the mediation services between PLMNs. The communication between NFs from different PLMNs occurs along the N32 interface that is represented in the Figure 4.4. Since the operators pIPX and cIPX may want to modify the messages exchanged between PLMNs to provide the mediation services, TLS cannot be used since it does not allow intermediaries to see or modify a mes-

sage. Because of this, application layer security is needed for protection between the SEPPS [10]. Thus, the JSON Web Encryption (JWE) encryption is used to protect messages. Furthermore, IPXs can perform modifications according to a policy specified by the networks, which must be recorded and digitally signed by the JSON Web Signatures (JWS).
If there are no IPX entities between the SEPPs, the TLS protocol is used to protect the signaling messages, between each NF [10].

4.4 Security Challenges in 5G Networks

The introduction of many technologies like Software Defined Networking (SDN), Network Function Virtualization (NFV), Network Slicing and MEC allowed 5G Networks to achieve high performance goals, namely in terms of scalability, flexibility, speed, latency large capacity. However, all these technologies also give rise to additional security risks.

4.4.1 Orchestration Tools

The complexity of 5G networks has empowered the development of artificial intelligence and machine learning algorithms in the orchestration layer. Within a SDN/NFV environment, an orchestrator is able to manage VNFs based on the network conditions [5]. If the network is under an attack, the orchestrator will communicate with the SDN controller that controls the firewalls and routers to mitigate the attack [5]. At the same time, the orchestrator will also instantiate additional VNFs if the traffic load increases. This flexibility introduces new security threats, as it allows an attacker who has access to the orchestrator to modify its configuration in order to take control of a Virtual Network Function (VNF) [16] [5].

4.4.2 SDN Controller

Software Defined Networking has emerged as a new network architecture with the aim of reducing hardware limitations. The main idea of introducing SDN is to separate the control plane out of the switches and to allow for external data control via its logical software component, the SDN Controller. Hence, the role of the SDN Controller is to manage flow control to the switches/routers through its southbound APIs and the applications and business logic through northbound APIs [17]. Since SDN uses a centralized controller with software applications, one of the biggest benefits of implementing SDN is its flexibility. Because the SDN controller assumes most complex functions, the network devices just need to accept orders from the SDN controller. As a result, this gives network administrators great flexibility to configure, manage, secure and optimize network devices [18]. However, an SDN could be very vulnerable to attacks because of the separation of control and data planes. A miscommunication between both planes can result in a major flaw that attackers can capitalize on, by sending specific threat vectors to the SDN Controllers [18], [5].

4.4.3 Network Slicing

Although network slicing allows resources to be shared more efficiently in the network and facilitate the allocation of resources to support different types of applications, this also brings security problems. [5] states that it is mandatory to ensure a proper isolation of slices and a trusted virtualization infrastructure, by addressing potential security challenges and the respective potential mitigation techniques. For example, the capability of allocating appropriate amount of network resources to a specific slice based on service, brings a new security challenge where an attacker may cause a Denial of service, by exhausting resources common to multiple slices. A mitigation technique for this problem would be to implement limitation of resources for individual slices.

4.4.4 Mobile Edge Computing

Mobile Edge Computing is a key enabler for 5G to achieve low latency and high bandwidth requirements, by bringing the computing, storage and networking resources to the base station. This combined with the increased attack surface in 5G environments due to the higher number of different devices connected to the network, makes the edge more susceptible to cyber-attacks. [5] addresses the importance of configuring a proper separation of third-party applications and the network functions in robustly segregated Virtual Machines, since badly designed applications might introduce risks and allow attackers to infiltrate the platform. Furthermore, the higher speed in 5G dramatically increases the impact of DDoS attacks over the network. This effect is also amplified by the wide diversity of IoT devices, which is expected in 5G, as IoT devices tend to be more vulnerable to cyber-attacks.

4.4.5 More Devices, More Chances Of DDoS Attacks

DDoS attacks top the list of primary security concerns for mobile operators now that 5G is advancing as the number of connected devices grows. A DDoS is a type of cyber attack on a specific server or network with the purpose of disrupting normal operation by flooding the target with a constant flood of traffic. By using malicious software, an attacker can develop a malware program and distribute it over the network. So, if a more vulnerable device opens one of these infected programs, the malicious software will be installed in their device without the owner even noticing it. The attacker then utilizes all these infected devices (also known as a Botnet) to perform a DDoS attack. The most threatening DDoS attack vectors to 5G network are addressed in [19]: SYN Flood; UDP Flood; UDP Fragmentation and DNS Flood.

SYN Floods exploit the vulnerabilities in TCP handshake process. In a normal TCP connection, there is a message request from the client to synchronize with the server, which then responds back with a SYN-ACK message aknowledging that the client wants to connect. The client then sends back an ACK to perform the connection. In this type of attack, the client never sends the final ACK message and therefore the connection is never completed. The temporary connection will eventually close, but in a Distributed attack where multiple requests are sent in the same time, the server will be overwhelmed with incomplete connections. The Figure 4.5 illustrates this process.

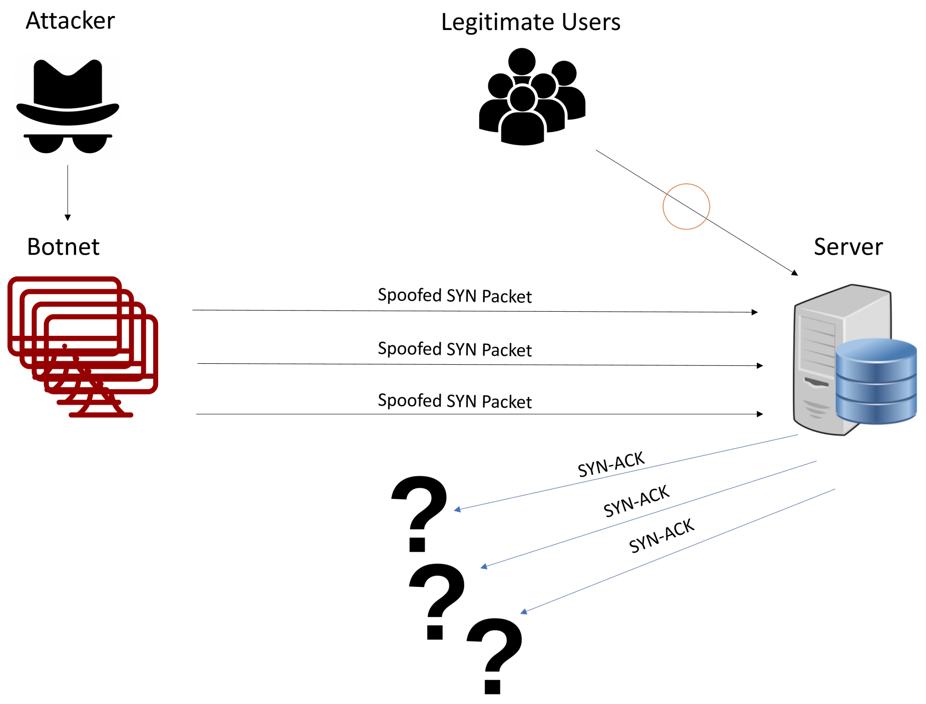


Figure 4.5: SYN Flood Attack

In a UDP Flood, attackers send small UDP packets to random ports on the system of the victim using a large range of source IPs. This consumes essential network element resources on the victim's network which are overwhelmed by the large number of incoming UDP packets.

UDP Fragmentation Attacks consist of sending large UDP packets which consume more network bandwidth. Since the fragmented packets usually cannot be reassembled, they consume significant resources on stateful devices such as firewalls along the traffic path.

DNS flood attacks constitute a relatively new type of DNS-based attack that has escalated with the rise of high bandwidth Internet of Things (IoT) botnets. DNS flood attacks use the connections of IoT devices to directly overwhelm the DNS servers of major providers. The volume of requests from IoT devices overwhelms the services of the DNS provider and prevents legitimate users from accessing the DNS servers.

4.5 Enhancing Security In 5G Networks

Several mitigation strategies have been proposed to address security challenges in 5G networks. These strategies aim to protect networks and services against DDoS attacks by using emerging technologies and effective approaches. In this Section we will explore some of these strategies, aiming to enhance prevention and detection measures against this type of attacks.

4.5.1 Rate Limiting and Packet Filters

Rate limiting involves controlling the rate of incoming packets from specific sources, preventing any given traffic source from sending too many requests. There are several types of rate limiting algorithms, with the most popular ones being:

- **Token Bucket Algorithm:** In this algorithm, a token bucket is used to hold tokens, and each incoming request or packet requires a certain number of tokens to be processed. The bucket is filled at a fixed rate, and if there are not enough tokens for an incoming request, it is either delayed or dropped [20];
- **Leaky Bucket Algorithm:** Similar to the token bucket, the Leaky Bucket algorithm also involves a bucket that holds requests. However, instead of tokens, the bucket "leaks" requests at a steady rate. If the bucket is full, incoming requests are dropped [20];
- **Fixed-window Algorithm:** Requests are tracked within fixed time windows, for example 1 sec. If the number of requests in a window exceeds a predetermined threshold, subsequent requests in that window can be delayed or rejected [20].

Packet filters are complementary strategies that examine the header of each IP packet based on a set of rules and decides to prevent it from passing (drop) or allow it to pass (accept). For instance, if we consider a scenario involving a SYN Flood attack, packet filters play a critical role by inspecting the header of the IP packet to see whether it corresponds to a SYN Packet or not. This inspection allows a rate limiter to effectively track the number of SYN Packets, enabling it to apply controlled limitations on incoming traffic and mitigate the impact of potential SYN Flood attacks.

However, DDoS attacks have unique challenges because they distribute requests among many different sources, sometimes millions of IP addresses allowing each source to avoid exceeding the rate limit. Therefore, the security solution should identify the requests from different locations as part of a single attack, treating them as a single source. Furthermore, rate limiting does not differentiate between harmless and harmful sources of traffic, making it unable to tell if a connection is safe or not. Additionally, it treats all sources of traffic the same way, which means both good and bad sources get penalized equally. This leads to a lot of false alarms where innocent sources are treated as malicious [21].

4.5.2 The Power of eBPF for Enhanced Linux Kernel Functionality

While exploring approaches to implement mechanisms to mitigate DDoS attacks, several tools came into perspective. The first one was **Iptables**, which allows us to define rules for filtering and manipulating network packets. Iptables is a user-space tool that provides a firewall framework within the Linux kernel. It offers a set of predefined tables and chains that can be customized to filter, modify, or redirect packets based on specified criteria such as source and destination IP addresses, port numbers, and protocols. With iptables, we can establish rules that determine whether packets should be accepted, rejected, or forwarded to another destination. While iptables has been a widely adopted solution for network security, its effectiveness can be limited in the face of complex and high-speed DDoS attacks. As packets pass through iptables user-space processing, the overhead of context switches and user-to-kernel transitions can hinder its ability to efficiently handle large volumes of incoming traffic. This can lead to performance bottlenecks and delays in responding to rapidly evolving attacks. Furthermore, iptables rule-based approach might not offer the level of precision required to effectively mitigate certain types of DDoS attacks. The predefined rules may not always capture the specific characteristics of emerging attack patterns, leaving the network vulnerable to sophisticated attacks that bypass the existing rules. This led us to investigate the capabilities of **eBPF**.

eBPF, an extended version of the **Berkeley Packet Filter (BPF)**, is a technology with origins in the Linux kernel, used to safely and efficiently extend the capabilities of the kernel without requiring to change kernel source code or load kernel modules. Historically, the operating system has always been an ideal place to implement observability, security, and networking functionality due to the kernel privileged ability to oversee and control the entire system. At the same time, an operating system kernel is hard to evolve due to its central role and high requirement towards stability and security. The rate of innovation at the operating system level has thus traditionally been lower compared to functionality implemented outside of the operating system. eBPF changes this formula fundamentally. By enabling the execution of sandboxed programs within the operating system, eBPF opens doors for developers to seamlessly incorporate additional capabilities to the system in real-time. This approach not only grants developers the power to craft low-level monitoring, tracing, or networking applications within Linux, but also does so while ensuring optimal performance standards.

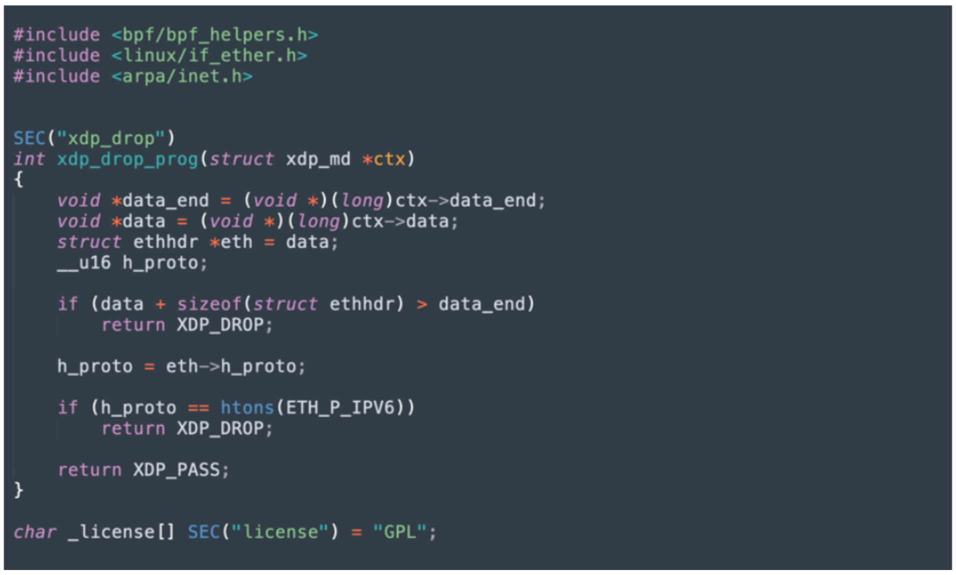
4.5.3 The Role of XDP in eBPF

eXpress Data Path (XDP) is a technology that allows developers to attach eBPF programs to low-level hooks, implemented by network device drivers in the Linux kernel, as well as generic hooks that run after the device driver. XDP can be used to achieve high-performance packet processing in an eBPF architecture, primarily using kernel bypass. This greatly reduces the overhead needed for the kernel, because it does not need to process context switches, network layer processing, interrupts, and so on. Nonetheless, the kernel bypass method does bear its limitations:

- eBPF programs have to write their own drivers. This creates extra work for developers.
- XDP programs run before packets are parsed. This means that eBPF programs must directly implement functionality they need to do their job, without relying on the kernel.

These limitations created the need for XDP. XDP makes it easier to implement high-performance networking in eBPF, by allowing eBPF programs to directly read and write network packet data, and determine how to process the packets, before reaching the kernel level.

XDP programs can be attached directly to a network interface, and they are triggered whenever a new packet is received on that interface. This immediate callback mechanism allows XDP programs to process the packet swiftly and make decisions based on its contents, making it an efficient way to handle network traffic at an early stage within the Linux kernel. One of the basic functions of XDP in eBPF is to use `XDP_DROP`, which tells the driver to drop packets at an early stage. This lets a developer apply a variety of efficient network strategies, while keeping the cost of each packet very low. This is great for situations where it is necessary to deal with any type of DDoS attack, but more generally, using XDP, eBPF can implement any type of firewall policy with very little overhead. An example of an XDP program that analyzes traffic and drops only IPv6 packets is shown in Figure 4.6.



```
#include <bpf/bpf_helpers.h>
#include <linux/if_ether.h>
#include <arpa/inet.h>


SEC("xdp_drop")
int xdp_drop_prog(struct xdp_md *ctx)
{
    void *data_end = (void *)(long)ctx->data_end;
    void *data = (void *)(long)ctx->data;
    struct ethhdr *eth = data;
    __u16 h_proto;

    if (data + sizeof(struct ethhdr) > data_end)
        return XDP_DROP;

    h_proto = eth->h_proto;

    if (h_proto == htons(ETH_P_IPV6))
        return XDP_DROP;

    return XDP_PASS;
}

char _license[] SEC("license") = "GPL";
```

Figure 4.6: Drop Only IPv6 Packets [22]

In the study conducted in [23], researchers aimed to address the challenge of mitigating DDoS attacks effectively. To achieve this, they focused on enhancing the capabilities of network servers to handle and mitigate such attacks. The initial approach involved using Iptables in combination with Berkeley Packet Filter (BPF) through the xt bpf module. This methodology allowed them to express complex filtering rules, where basic packet patterns were handled by Iptables, while more intricate matching logic was accomplished using BPF. However, as their network infrastructure faced larger DDoS attacks, they encountered performance limitations with the Iptables-based approach. This situation led to issues where servers were overwhelmed by the sheer volume of network packets, causing application processing to suffer. To enhance their mitigation strategy further, they planned to transition from their existing approach (classical BPF and Iptables/userspace offload) to leverage eBPF and XDP. This transition aimed to streamline their mitigation process by consolidating filtering rules into a singular XDP program. The anticipated benefits of this migration included simplified rule expression and improved overall performance.

Furthermore, within the domain of 5G Networks, particularly the UPF of the 5G SA Architecture, there has been a growing exploration of eBPF/XDP applicability. A research work in this context is presented in [24], which explored the potential benefits of integrating eBPF XDP to enhance UPF performance and efficiency. While the Data Plane Development Kit (DPDK) has gained recognition for its exceptional throughput and low latency, certain limitations arise in typical MEC deployments. These drawbacks include inadequate support by Network Interface Controllers (NICs), resource wastage through the continuous occupancy of the CPU via DPDK Poll Mode Drivers even during periods of low traffic, and the complexity of integrating non-DPDK applications with DPDK counterparts. To address these challenges, the authors proposed the integration of eBPF/XDP as a means to achieve faster packet processing, lower latency, and optimized resource utilization within the UPF. Through a comprehensive evaluation, the results showcased the ability of the solution to achieve remarkable scalability and performance. Specifically, the proposed UPF solution achieved a throughput of 10 million packets per second while utilizing only 85% of the CPU with 6 cores. This performance enhancement was attributed to the integration of eBPF/XDP, which demonstrated the potential to outperform DPDK in terms of resource utilization and efficient packet processing.

4.5.4 Observability in the Context of 5G Networks

Observability is essential for security because they provide insights into what is happening within a system or network. It provides visibility into the internal state of the system, including its performance, behaviour, and interactions with other systems. This information is crucial for identifying problems, optimizing performance, and ensuring the system meets its service level objectives. It also offers several ways to enhance security. This involves utilizing observability data for real time detection and response to security incidents. For instance, through analysis of logs, network activity, and other relevant data, security teams can identify indications of a data breach or security incident and immediately respond to mitigate potential damage [25] [26].

As 5G SA networks are beginning to roll out, there is a continuous discussion about how to effectively provide observability in these platforms that heavily use containers. Traditional tools are no longer viable, and this is widely accepted within the industry. The previous way of using certain tools for observing is outdated, and even relying on virtual tools for "cloud resources" is not as useful anymore. We are now in the "cloud-native" era, which means a big change in how we use cloud technology. Cloud-native focuses on making the most of virtual resources and distributed computing, moving from one type of technology to another. The challenge now is determining how to best introspect these containerized environments. There are two main options that have emerged for providing observability of 5G SA environments, namely eBPF and sidecar containers.

Many observability providers have turned to using sidecar containers as a solution. This approach involves linking into the application layer of the container traffic. The idea is that a sidecar container is added to each pod, sitting alongside the main application/container (such as a 5G Network Function). However, this method comes with drawbacks. The primary container and the sidecar container share resources like volume and network, and so adding a sidecar increases the resources needed for the main workload (the NF). This issue has sparked a debate between network function vendors and observability vendors. Some vendors do not want sidecar containers near their network functions, as they affect resources and market share. Moreover, the observability vendor providing the sidecar solution may need cooperation from NF vendors to work effectively. This involves potential code changes by NF vendors or discussions about integrating at the application level and sharing pod resources. Using sidecar observability solutions raises concerns for NF vendors. Not only does it create resource-sharing issues at the application level, but it also introduces unwanted network latency, especially troubling for 5G mobile environments. Sidecars act as proxies, causing pod-to-pod communications to pass through them, resulting in double latency for every communication to/from a network function. This is significant in a 5G architecture, which promises faster speeds and lower latency. The added latency due to sidecar containers contradicts the goals of a 5G SA architecture.

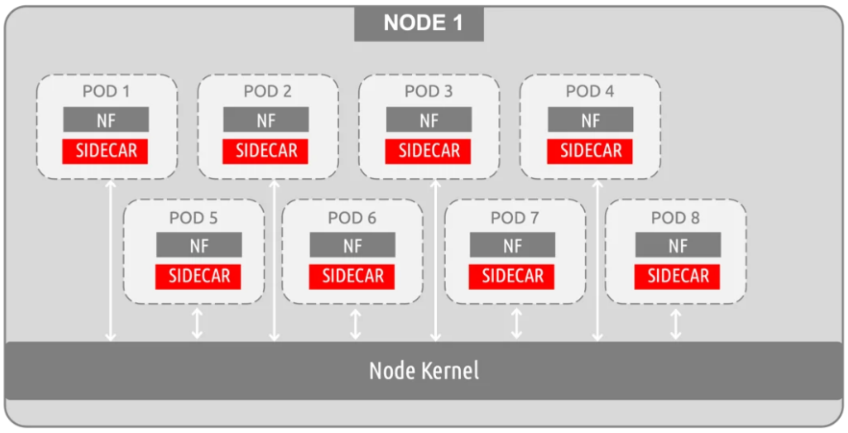


Figure 4.7: Sidecar deployed in each NF pod [27]

With eBPF, things are different. This technology operates by connecting to the core elements of the operating system through deployment at the kernel level, setting it apart from the application/network level approach used by sidecar containers. Illustrated in the Figure 4.8 is a basic eBPF program employed for monitoring 5G resources. Unlike kernel modules, eBPF solutions function as standalone programs within the kernel space, eliminating the need for kernel modifications. In this scenario, the eBPF program, serving as an event handler within the kernel space, can assess the entire node environment (including pod-to-pod communications, processes, and interfaces) when deployed as a single pod/container.

This deployment model using eBPF agents contrasts with the approach involving sidecar containers. The advantages of eBPF monitoring span several areas, with its exceptional performance being a standout feature. Unlike sidecar containers that act as intermediaries for all pod-to-pod communications (requiring resource-intensive sifting through these communications), eBPF programs function as event handlers at the kernel level. These programs define specific guidelines for targeted events (such as 5G-specific SBI messaging) and initiate monitoring activities upon event occurrence. This approach offers an efficient foundation that can perform various functions and capabilities. In many cases, this incurs a minimal resource cost of less than 1% of a virtual CPU, dependent on the specific use case [27].

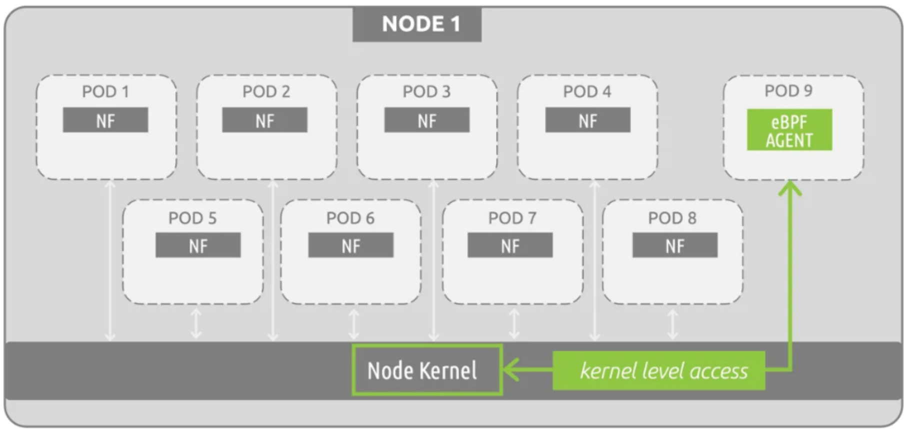


Figure 4.8: eBPF Program for 5G Resource Monitoring [27]

Therefore, eBPF technology becomes essential in cloud-native setups due to its extensibility in kernel subsystems [26]. With impressive programmability, eBPF greatly enhances safety, visibility, and policy enforcement, making it suitable for networking, observability, and cloud security with low overhead. This technology empowers deep observability across clusters in cloud orchestrators such as kubernetes, offering a comprehensive view of applications and cloud-native functions [26] [28].

4.6 Chapter Conclusions

This chapter presented several topics regarding security in 5G. It started by addressing the 3GPP security specifications in its latest release TS 33.501, where it introduced a set of security features and improvements, such as the encryption of the SUPI to resolve the SUCI, so the subscriber identifier is not sent in clear text over the access network and the introduction of a new authentication framework, 5G-AKA, which adds home network control and allows the home network to be in charge of authentication instead of the serving/visiting network. It also presented the requirements for securing the connections between SBA Network Functions, namely mutual authentication (for example, TLS 1.2/1.3 with x.509 certificates) and authorization (for example, OAuth2.0). Then we presented security challenges and opportunities associated with 5G, addressing problems that can take place when novel technologies are used in order to support 5G networks. Lastly, we also explored strategies to enhance security in 5G environments, including fast packet processing to mitigate DDoS attacks and methods to enhance observability of 5G components within a Kubernetes cluster.

Chapter 5

5G Testbed

This chapter contains the analysis of the Testbed that is being developed. It will start by presenting the architecture, and address the respective components and technologies used. Finally, it will present the initial experiments.

5.1 Testbed Overview

The Testbed implementation was performed from another research work in the same line of investigation and it uses several different technologies, mainly the VMware ESXi, which is the **Hypervisor** that runs and creates Virtual Machines, and the Rancher's next-generation Kubernetes distribution (RKE2), an **Orchestration tool** used for managing applications and services inside clusters.

The RKE2 cluster consists of three RKE2 Server Nodes in order to achieve High Availability, meaning that they work as a single system and ensure continous operation and uptime. These Nodes have the role of controlling the cluster and make it function. Each one consists of multiple components: The **API-Server**, which the user and the other components communicate with; The **Scheduler**, that assigns the worker node to each deployable component of the application; The **Controller-Manager**, which performs cluster-level functions, such as replicating components, keeping track of worker nodes and handling node failures; Lastly, the **etcd**, a reliable distributed data store that persistently stores the cluster configuration [29]. Thus, the components inside each Server Node hold and manage the Cluster, but they do not run the application, as this is done by the Worker Nodes. The Worker Nodes are responsible for running, monitoring and provide services to the application [29].

Another important component to talk about is the **Kube-VIP**. Kube-VIP is a service that provides Kubernetes clusters with a virtual IP and Load Balancing for the control plane and Kubernetes Services with the load balancer type, without relying on any external hardware or software [30]. The cluster deployed in the testbed runs in the Address Resolution Protocol (ARP) mode. In this mode, all kube-VIP pods will elect a Leader from the Server Nodes, which will be responsible for exposing all services addresses through ARP. This mode also enables new worker nodes to be installed within the kubernetes cluster without disrupting the functionality of the system, including the pods that run the applications.

The Rancher Server adds a complete Graphical User Interface and workload management layer, that simplifies adoption and integrates CI/CD.

The deployment of the applications running in the kubernetes cluster was done by using **Helm**. Helm is a kubernetes tool that helps managing applications, by making it easier to install, uninstall and update packages for kubernetes applications.

The next figure represents the RKE2 cluster architecture:

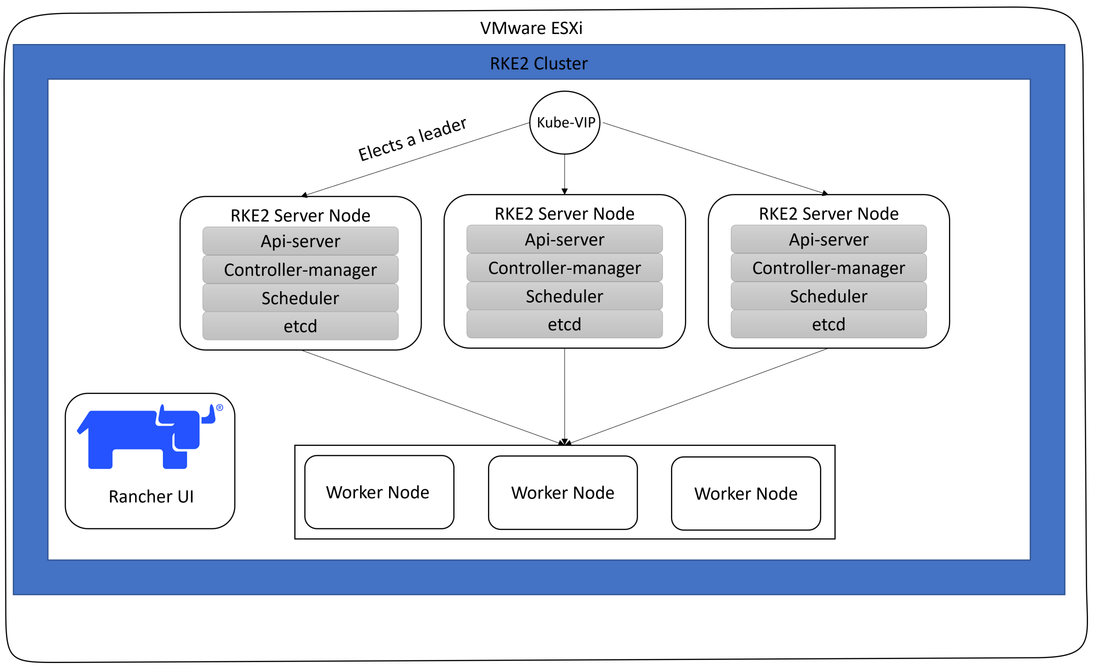


Figure 5.1: RKE2 Cluster Architecture

Next, we will take a look into the current version of the 5G Core simulator that is going to be used in the testbed, presenting the Open5GS and the UERANSIM.

5.2 Open5GS and UERANSIM

Open5GS is an open source implementation that provides 5G Core Network functionalities, for building a private 5G network. Its main goal is to implement the 5G Core network which was defined by 3GPP (currently in Release 17) and although it supports the two different architecture segments, 5G NSA and 5G SA, our focus will be solely on the 5G SA Core network functions.

The network functions running in the current version of Open5GS 5G SA Core are the following: NRF, SCP, AMF, SMF, UPF, AUSF, UDM, UDR, PCF, NSSF and BSF. The role of each NF was already explained in Section 3.2.2. It uses MongoDB as its database to store the information needed for NRF,PCF and UDR. **OpenEBS** was used to provide a fast local storage for MongoDB.

Open5GS also provides a WebUI, represented in the Figure 5.2, allowing users to interactively add and edit subscriber data. While not mandatory, the WebUI simplifies the process of adding UE subscribers to the AMF in the 5G Core by enabling users to perform this task with a simple click of a button.

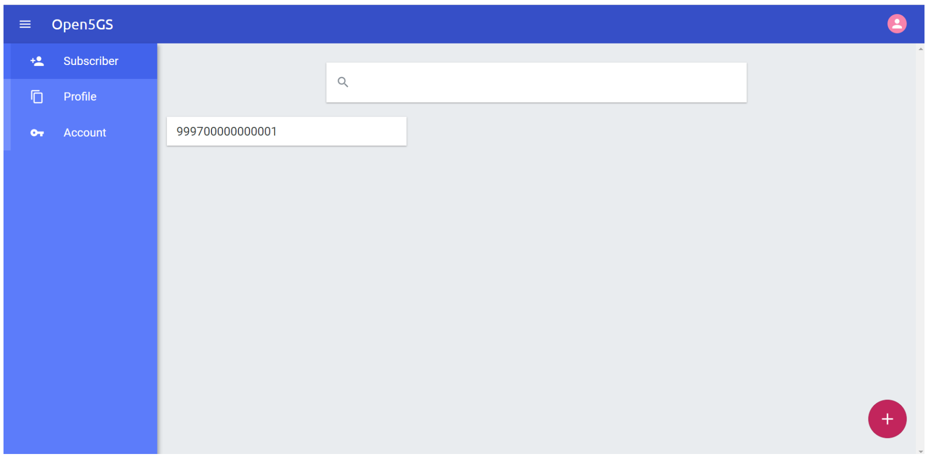


Figure 5.2: Open5GS Web Interface

To ensure that the 5G testbed is fully up and running, and is able to connect to the NG-RAN and the UE, we need to simulate the devices connected to the network and the RAN functionality. Thus, **UERANSIM**[31] was deployed within the cluster, which is the open source state of-the-art 5G UE and RAN (gNodeB) simulator, and can be used to test the 5G Core Network and study the 5G System. After deploying ueransim-gnb, it established a SCTP connection with the AMF, and then the NGAP procedure was successfully performed, which sets up the N2 interface between the gNB RAN and the AMF. Finally, we installed ueransim-ue, in order to add a subscriber to the network. Once ueransim-ue starts running, a PDU session and a TUN interface (uesimtun0) are created. This interface is associated with an IP address, and works as a virtual network interface that allows the UE to send and receive IP packets to and from the 5G Core over the established PDU session, enabling the simulated device to connect to the internet via the core network and furthermore providing several services such as data transfer, voice over IP and other network applications. For example, after entering the UE terminal, by using a simple curl or a ping command such as *ping www.google.com -I uesimtun0*, we could successfully ensure that the traffic was being sent from the UE to the internet via the 5G Core.

Both Open5GS and UERANSIM were deployed using Helm Charts Collection of Openverso Project [32]. This repository maintains helm charts generated by Gradiant in OpenVerso, with the goal of facilitating the deployment and management of 5G network components, evolvable to 6G technologies.

The scheme in Figure 5.3 illustrates the final deployment architecture of Open5GS and UERANSIM in the Kubernetes cluster, showcasing how each Network Function is deployed as a separate component. These components represented as pods are assigned with individual ClusterIP addresses, allowing pods to communicate with other pods on the same node or other nodes. These IP addresses are private and only accessible from within the Kubernetes cluster. Within each Node, there are also three primary components: The **kubelet**, which talks to the API server and manages containers on its node; The **kube-proxy**, to load-balance network traffic between application components; And the **Container-runtime**, such as Docker or containerd, responsible for running containers in the pods. These components enable the execution of Network Functions within the pods, ensuring their availability, proper resource allocation (such as CPU and memory), and communication within the cluster.

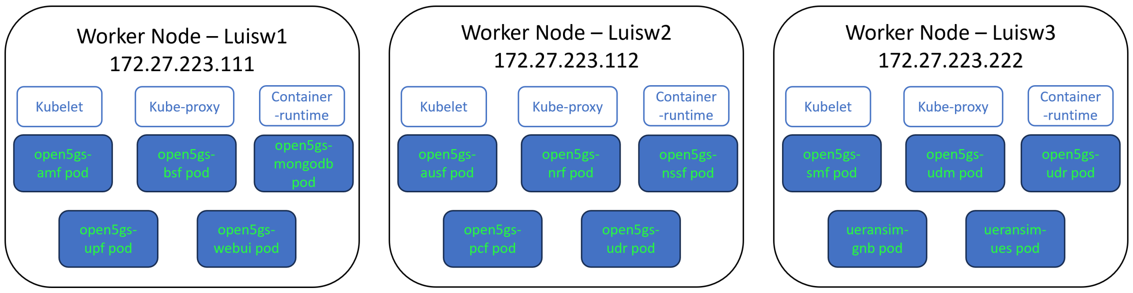


Figure 5.3: Open5GS and UERANSIM deployment

This testbed has been developed to enhance and investigate the security measures against DDoS/flooding attacks targeting 5G Networks. Having now acquired a comprehensive understanding of our 5G Testbed architecture and the underlying technologies that power it, our focus shifts to the upcoming section, dedicated to our Initial Experiments.

5.3 Initial Experiments

In this Section, we present the preliminary tests conducted within the 5G Testbed, providing insights into the earliest findings, showcasing our initial ideas and how we formulated the attack plan.

5.3.1 Understanding 5G SBIs and Selecting Our Target

An API usually provides the same services that a web application of the same provider does, just without the use of a graphical interface. Thus, they are built as an interface to answer automated requests, normally provided by processes instead of people. Due to this interface, there is a specific ruleset to enable a correct communication with an API, which includes specific endpoints, request methods, input parameters, data format and so on. All the information about these rules that apply to the 5G Core APIs, also known as SBIs, is documented in OpenAPI 3.0 format in [33], which has been created by 3GPP, as part of the official 3GPP Technical Specifications. Therefore, this repository can be very helpful in order to understand how to interact with the SBIs and furthermore we can find potential exploits and evaluate the types of information that could be explored by attackers who successfully penetrated a 5G SA Network. In the beginning, we considered using the tool **Postman** as our client to craft and send our requests and retrieve responses from the SBIs. However, since these interfaces communicate with each other using the HTTP/2 protocol, which is not supported by Postman, we had to find an alternative solution. We decided to use **libcurl** in C and had to include the flag **–http2-prior-knowledge** to ensure that our requests would work with the HTTP/2 protocol, since it was the only way to interact with the SBIs of Open5GS. This flag tells libcurl to use HTTP/2 from the start of the connection, rather than negotiating the protocol version with the server.

Each Network Function holds a unique and crucial role within the 5G Core Network contributing to its consistent and efficient operation, explained in Section 3.2.2, and so identifying a specific target was not a straightforward task. However, aligning with the research goals of examining the 5GC APIs, focusing on malicious requests for data manipulation and information gathering, and evaluating the impact of subjecting a Network Function to a denial of service situation, we posed three significant considerations:

- Which Network Function maintains the highest degree of interaction with other Network Functions;
- Which Network Function holds critical and sensitive data, including Network configurations, components and Subscriber information;
- Which Network Function plays an indispensable role in ensuring uninterrupted connectivity, preventing disruptions for UE access to the Internet.

After exploring the repository containing the 5GC APIs documentation and considering the types of information that could be gathered or manipulated through the API endpoints, three main components drew my attention: the UDM, SMF, and NRF. The NRF Network Function serves as a central repository for network function discovery and management, already explained in Section 3.2.3, and could potentially be an attractive target for attackers in order to learn information about other Network Function instances, or even add, delete and modify them without having the permission to do so. The SMF Network Function plays a critical role in managing sessions between the UE and the network, and its disruption can result in loss of connectivity or potential data breaches. The UDM component holds subscriber-related information, such as authentication and authorization keys and credentials, and if an attacker gains access to it, he might be able to impersonate legitimate subscribers and gain unauthorized access to network services. However, the testbed we are utilizing employs the Open5GS project, and just like many open-source projects, its implementation is centered more on functionality. Consequently, several endpoints specified in the 3GPP Specifications **are yet to be implemented**.

Regarding the UDM component, we tried to send many requests to its SBI endpoints. However, after an inspection of the source code files [34] and [35], it became evident that only the PUT and PATCH methods of the **UE Context Management Service** service, more precisely the /registrations/amf-3gpp-access API endpoint, have been implemented, enabling the registration and update of an AMF Instance for 3GPP access. Furthermore, we performed requests to the **UE Authentication Service**, and followed the same approach. By looking into the source code files [34] and [35] we discovered that there was only the implementation of two endpoints, namely /security-information/generate-auth-data to generate authentication data for the UE and /auth-events to delete the authentication result in the UDM. Lastly, after performing requests to the **Subscriber Data Management Service**, we could only retrieve information specifying the maximum uplink and downlink rates on the /am-data endpoint, and the default network slice chosen for a subscribed UE on the /smf-select-data endpoint.

While investigating the implemented endpoints on the NRF SBI in [34], we discovered that Open5GS has successfully implemented all the endpoints specified in the 3GPP specifications for the NRF. Considering this, **we decided to concentrate our efforts on the NRF as our primary target**. The NRF is expected to be an interesting choice due to its crucial role, and if the NRF is susceptible to flooding attacks in its SBI and becomes a victim of a denial-of-service situation, its capability to interact with other Network Functions might be compromised, which can cause disruptions that can affect the entire 5G Core Network. This is because all Network Functions rely on interactions with the NRF to discover other Network Functions and request their services.

5.3.2 Attack Plan

Having selected our target and understanding the critical role of the NRF within the 5G Core Network, it is crucial to develop a plan for the attacks that will be executed against it. Firstly, we established the following goals:

- Gather information about other Network Function instances;
- Remove registered instances of other Network Functions;
- Alter information about registered Network Functions;
- Fuzzing by inputting massive amounts of random data to the NRF API in an attempt to make it crash;
- Cause a Denial of Service in NRF SBI to block communication with other Network Functions, potentially disrupting the connectivity of UE subscribers to the Internet.

To carry out the planned attacks, we have an attacking Virtual Machine serving as our penetration testing platform and it is positioned outside the Kubernetes cluster, symbolizing an external entity attempting to access the Network Functions. This Virtual Machine is equipped with all the necessary tools to perform the intended attacks. As already mention, one of the tools we are going to use is **libcurl**, which allows us to craft and send requests to the NRF SBI. This tool provides us with the flexibility to create custom requests, manipulate parameters, and interact with the NRF endpoints. Additionally, we will use the **Hping3**[36] and **Scapy**[37]. Hping3 is a versatile packet crafting tool that enables us to generate and send network packets with specific characteristics. It plays a crucial role in executing flood-based attacks, such as SYN floods, that can overwhelm the NRF SBI. Scapy is a Python-based tool that offers packet manipulation capabilities. While Hping3 is specialized for certain types of network-based attacks, Scapy provides us with a more dynamic approach, allowing us to design and execute custom packet-based attacks tailored to the specific requirements of our testing scenarios. This multi-tool approach ensures that we can tackle a range of attack vectors effectively, from crafting precise requests using libcurl to simulating high-volume network traffic using Hping3 and Scapy.

As already demonstrated in Figure 5.3, the 5G Services, including the NRF component, are running inside a Kubernetes Cluster and assigned with private ClusterIP addresses. However, in a real world environment, these services might be exposed for specific use cases [38]. When we aim to make services accessible in Kubernetes, there are typically two well-known methods: using a **Load Balancer** or a **NodePort**. In our context, we opted to use NodePort for our testing purposes. This decision reflects the need for a fast and straightforward approach suitable for exposing our NRF, making it accessible through any worker node that is listening on the assigned NodePort for incoming requests for the service. Although Load Balancers provide advanced features for production environments, NodePort aligns better with our testing goals, balancing simplicity and efficiency.

5.4 Chapter Conclusions

In this chapter, we presented the analysis of the 5G testbed that was implemented, focusing on its architecture, components, and technologies used. Furthermore, we carried out a preliminary set of experiments, simply to familiarize ourselves with the Open5GS application. Now that we have a solid knowledge of the testbed functionalities, in the next Chapter we will focus on the Penetration Testing process, aiming to achieve the goals outlined in our attack plan.

Chapter 6

Penetration Testing

In this chapter, we present the next phase of this research work, where we dive into the practical evaluation of the security of the overall testbed, following the **Penetration Testing Executive Standard** methodology. Despite our comprehensive knowledge of the implemented testbed, we will approach this testing from an attacker's perspective. Our goal is to simulate an environment where the attacker possesses minimal prior knowledge, aiming to make the penetration testing as realistic as possible.

6.1 Pre-Engagement Interactions

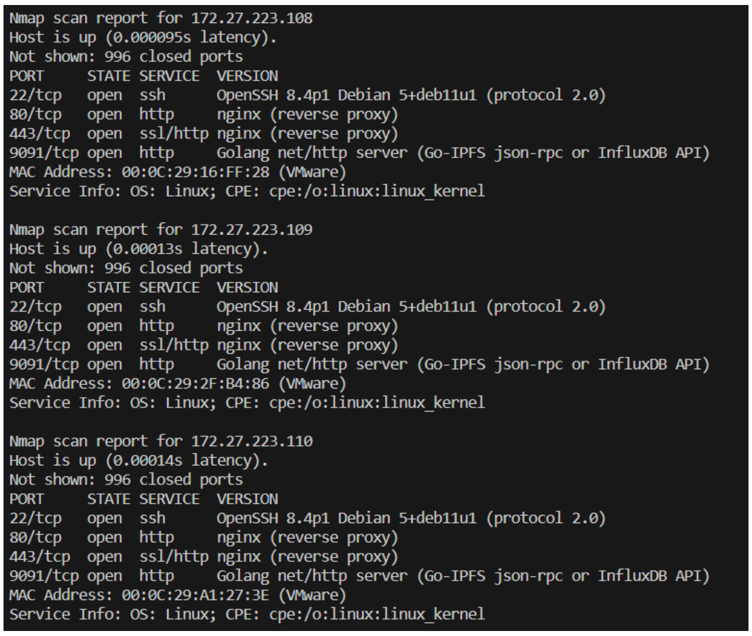
Before initiating any penetration testing activity, a crucial step is the Pre-Engagement Interactions. This phase is designed to establish effective communication between the testing team and the client, ensure a clear understanding of the project scope and objectives, and align expectations.

The goal of this Penetration Test is to identify vulnerabilities within the infrastructure supporting the 5G application, with the aim of gaining unauthorized access to the Kubernetes cluster. Subsequently, we will evaluate the security of the 5G Core APIs in the testbed, with a particular focus on the NRF component. This evaluation will involve executing malformed requests and conducting Flooding attacks, following the designed attack plan from Section 5.3.2.

6.2 Information Gathering

The information gathering phase aims to gather essential details about the target environment, facilitating a comprehensive understanding of its structure, services and the attack surface. We decided to use **NMAP**[39], a free and open source utility for network discovery that works as a fingerprinting tool, which is the first step of hacking where we can find information about the target, for example an IP address or a website. NMAP uses IP packets to give detailed, real-time information on the network, as well as the devices (servers, routers, switches, mobile devices) connected to them, and it can be used to scan active IP addresses and the network, including a list of live hosts and open ports, and the operative system of every connected device. We started Nmap using the command: "nmap -sV 172.27.223.1/24". This command checks which services are running on the devices within a specific IP range (172.27.223.1 to 172.27.223.254) and tries to determine the respective version of the services.

After reviewing the scanning results, with a portion of it shown in Figure 6.1, we discovered numerous connected hosts within the VMware ESXi hypervisor. Since the implementation of the testbed was made by us, we hold prior knowledge about which hosts correspond to the Master and Worker nodes within the Kubernetes cluster. However, adopting an attacker's perspective sheds light on the complexity of identifying specific nodes within the Kubernetes cluster, given only scan results. From an attacker's point of view, we attempted to gain further insights using tools such as **whois**[40] in an attempt to acquire additional knowledge about these hosts and potentially uncover more information. However, the results of the whois command did not provide meaningful insights directly pertinent to comprehending or targeting the Kubernetes cluster.



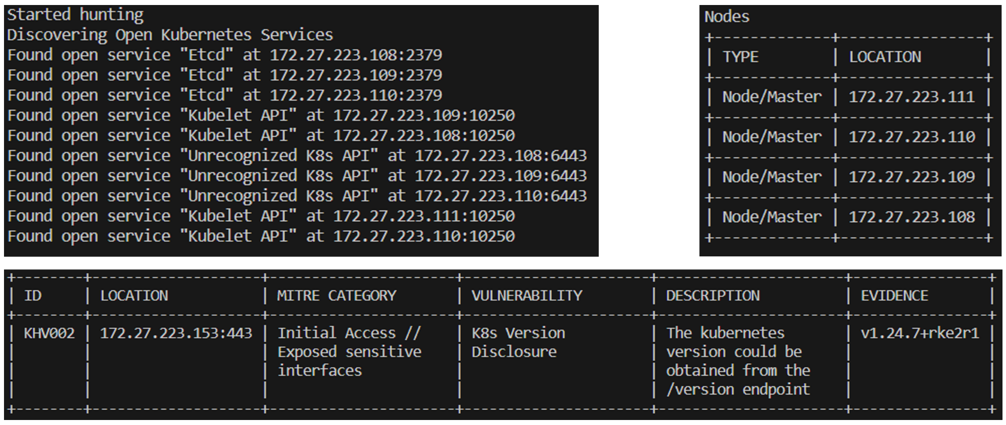
```
Nmap scan report for 172.27.223.108
Host is up (0.000095s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
80/tcp   open  http     nginx (reverse proxy)
443/tcp  open  ssl/http nginx (reverse proxy)
9091/tcp open  http     Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
MAC Address: 00:0C:29:16:FF:28 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap scan report for 172.27.223.109
Host is up (0.00013s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
80/tcp   open  http     nginx (reverse proxy)
443/tcp  open  ssl/http nginx (reverse proxy)
9091/tcp open  http     Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
MAC Address: 00:0C:29:2F:B4:86 (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap scan report for 172.27.223.110
Host is up (0.00014s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
80/tcp   open  http     nginx (reverse proxy)
443/tcp  open  ssl/http nginx (reverse proxy)
9091/tcp open  http     Golang net/http server (Go-IPFS json-rpc or InfluxDB API)
MAC Address: 00:0C:29:A1:27:3E (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 6.1: Nmap Scanning Results of the Master Nodes

So, we decided to further investigate other tools commonly used for assessing the security of kubernetes clusters, and we discovered a very interesting one, which is **kube-hunter**[41]. Kube-hunter is an open-source tool specially designed to hunt for security weaknesses in Kubernetes clusters. While its primary focus aligns with the Vulnerability Analysis phase, it offers broader capabilities. In addition to identifying vulnerabilities, kube-hunter also has the capability to map and discover nodes within a cluster, as well as open services running on these nodes, offering a comprehensive view of potential attack surfaces. We made this possible by issuing the "kube-hunter –mapping" command, aligned with a specific range of IP addresses. The Figure 6.2 displays the outcome of our kube-hunter scan. The scan effectively identified several nodes within the Kubernetes cluster and open services, including 'Etcd' and 'Kubelet API' on several IP addresses within the range. Moreover, we successfully retrieved the version of the Kubernetes distribution being used in the testbed, as depicted in Figure 6.2, revealing the utilization of v1.24.7+rke2r1.



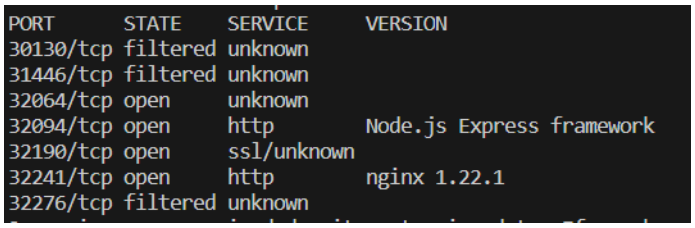
```
Started hunting
Discovering Open Kubernetes Services
Found open service "Etcd" at 172.27.223.108:2379
Found open service "Etcd" at 172.27.223.109:2379
Found open service "Etcd" at 172.27.223.110:2379
Found open service "Kubelet API" at 172.27.223.109:10250
Found open service "Kubelet API" at 172.27.223.108:10250
Found open service "Unrecognized K8s API" at 172.27.223.108:6443
Found open service "Unrecognized K8s API" at 172.27.223.109:6443
Found open service "Unrecognized K8s API" at 172.27.223.110:6443
Found open service "Kubelet API" at 172.27.223.111:10250
Found open service "Kubelet API" at 172.27.223.110:10250
```

```
Nodes
+-------------+----------------+
| TYPE        | LOCATION       |
+-------------+----------------+
| Node/Master | 172.27.223.111 |
+-------------+----------------+
| Node/Master | 172.27.223.110 |
+-------------+----------------+
| Node/Master | 172.27.223.109 |
+-------------+----------------+
| Node/Master | 172.27.223.108 |
+-------------+----------------+
```

```
+--------+--------------------+---------------------+-------------------+--------------------+----------------+
| ID     | LOCATION           | MITRE CATEGORY      | VULNERABILITY     | DESCRIPTION        | EVIDENCE       |
+--------+--------------------+---------------------+-------------------+--------------------+----------------+
| KHV002 | 172.27.223.153:443 | Initial Access //   | K8s Version       | The kubernetes     | v1.24.7+rke2r1 |
|        |                    | Exposed sensitive   | Disclosure        | version could be   |                |
|        |                    | interfaces          |                   | obtained from the  |                |
|        |                    |                     |                   | /version endpoint  |                |
+--------+--------------------+---------------------+-------------------+--------------------+----------------+
```

Figure 6.2: Kube-Hunter mapping results

Moreover, an attacker could potentially target services exposed through the NodePort mechanism in an attempt to discover additional exposed services within the Kubernetes Cluster. The NodePort mechanism allocates services to random ports within the range of 30,000 to 32,767. To investigate further, we conducted a scan using the command "nmap -sV -p 30000-32767 172.27.223.108", and found several open ports depicted in Figure 6.3. Based on the open port 32094, and the service and version parameters, it is likely that this port is hosting a web application built using the Node.js Express framework. To determine the nature of the exposed web service, an attacker could use the **whatweb**[42] tool. Utilizing this tool revealed that the web service is related to the Open5GS application.



```
PORT      STATE    SERVICE     VERSION
30130/tcp filtered unknown
31446/tcp filtered unknown
32064/tcp open     unknown
32094/tcp open     http        Node.js Express framework
32190/tcp open     ssl/unknown
32241/tcp open     http        nginx 1.22.1
32276/tcp filtered unknown
```

Figure 6.3: NodePort scanning results

In summary, after the information gathering phase, we acquired knowledge about the target environment's active hosts, open ports, and service versions. While certain nodes within the Kubernetes cluster could be identified due to our familiarity, we decided to take an additional step in attempting to distinguish hosts linked to the designated Kubernetes cluster. Our aim extended to determining the version of the employed Kubernetes distribution and uncovering services linked to these identified nodes.

6.3 Threat Modeling

Threat modeling is a proactive cybersecurity approach that aims to anticipate and evaluate potential vulnerabilities and threats within a system or application. By analyzing the system's architecture, components, and data flows, we can identify weak points and possible attack vectors. From the Information Gathering phase, we've gathered the following insights about the Kubernetes Cluster:

- The Kubernetes cluster consists of at least three Master Nodes, distinguished by IP addresses 172.27.223.108, 172.27.223.109, and 172.27.223.110;
- Each master node hosts atleast two open Kubernetes services: 'etcd' on port 2379 and 'kubelet API' on port 10250;
- The distribution version in use is v1.24.7+rke2r1;
- Multiple application services are deployed within the cluster and exposed through NodePort.

Each master node hosts at least two open Kubernetes services: 'etcd' on port 2379 and 'kubelet API' on port 10250. These services provide critical functionalities for managing the cluster. However, exposing them to untrusted networks without proper authentication controls poses a significant security risk [43]. This could lead to a range of malicious actions, such as unauthorized modification of cluster configurations, unauthorized data access, and potentially even complete cluster compromise, allowing them to create or delete pods, scale deployments, update configurations, and even delete resources. To mitigate this risk, security controls such as API Server Authentication, API Server Authorization, and Kubelet Authentication should be implemented. Additionally, for services that do not support authentication, such as cAdvisor or the read-only Kubelet, access should be restricted to a whitelist of trusted source addresses [43] [44].

The knowledge of the Kubernetes distribution version, which is v1.24.7+rke2r1 in this case, can potentially expose the system to security risks. Attackers might exploit known vulnerabilities associated with specific distribution versions. However, during our research we discovered a single vulnerability referenced in [45]. Notably, the information also indicated that the current version being utilized has already addressed this issue. Hence, to the best of our knowledge, no vulnerabilities have been identified within this version.

When considering services exposed via NodePort within the Kubernetes cluster, it is essential to highlight potential security risks. If proper authentication and authorization mechanisms are not implemented in these exposed services, they become vulnerable points of entry for attackers. Furthermore, these applications could potentially serve as a pathway for an attempt to gain control over the entire kubernetes cluster. Another security risk specifically associated with exposing services through NodePort is that it can lead to an imbalance in the workload among the Nodes in the cluster. When a service is accessed via NodePort, the specific Node it connects to may become overwhelmed, while other Nodes remain idle. This situation has the potential to cause a DoS on the Node in question. To address this issue, using a load balancer to distribute the workload evenly across all Nodes in the cluster can mitigate this, depending on the intensity and type of DoS attack. However, it is crucial to emphasize that even with load balancing, authentication and authorization mechanisms remain essential to safeguard these exposed services.

6.4 Vulnerability Analysis

Vulnerability analysis is a crucial phase in the cybersecurity assessment process that dives into discovering and evaluating potential security weaknesses within a system or application. This phase involves a systematic exploration of the target's architecture, configurations, components, and exposed services to identify vulnerabilities that could be exploited by malicious actors.

Firstly, we utilized Kube-hunter to identify potential vulnerabilities and weaknesses across the cluster, offering insights into several attack vectors that adversaries might exploit. Apart from the mapping and discovery capabilities, the kube-hunter scan performs several tests:

- **Kubelet Readonly Ports Hunter:** Hunts specific endpoints on open ports in the readonly Kubelet server;
- **Kubelet Secure Ports Hunter:** Hunts specific endpoints on an open secured Kubelet;
- **API Server Hunter:** Checks if API server is accessible;
- **Pod Capabilities Hunter:** Checks for default enabled capabilities in a pod;
- **Certificate Email Hunting:** Checks for email addresses in kubernetes ssl certificates;
- **Kubectl CVE Hunter:** Checks if the kubectl client is vulnerable to specific important CVEs;
- **Dashboard Hunting:** Hunts open Dashboards, gets the type of nodes in the cluster;
- **Etcd Remote Access:** Checks for remote availability of etcd, its version, and read access to the DataBase;
- **Access Secrets:** Accessing the secrets accessible to the pod;

After conducting kube-hunter scans across all nodes in the kubernetes cluster and performing its range of tests, the tool reported an affirmative outcome: **"No vulnerabilities were found"**. This result indicates that the evaluated aspects of the cluster have demonstrated a reliable security posture, effectively countering potential vulnerabilities and threats. For further investigation we decided to explore the use of **kubeletctl**[46]. This tool operates as a command-line utility, implementing the kubelet's API as a client to facilitate API requests directed at the kubelet itself. With this tool, we attempted to execute unauthorized actions within the cluster. In line with the findings from kube-hunter, our subsequent execution of multiple API calls towards the etcd and kubelet services led to specific response messages, notably **"The response failed with status: 401"** and **"Message: Unauthorized"**. These outcomes confirm the effective presence of authentication and authorization mechanisms.

Turning our attention to the final phase of our vulnerability analysis, we dived into the examination of services exposed through NodePort within the Kubernetes cluster. Naturally, following the Initial Experiments section 5.3, we had already explored how to communicate with the Open5GS APIs. To the best of our knowledge, there are only two tools availabe for requesting Open5GS services: **curl** and **nghttp2**[47]. From the perspective of a potential attacker seeking to exploit vulnerabilities in NRF of the 5G architecture, they would have to start with a passive information gathering phase, where they gather information from documents and online resources, similar to what we did in the Section 5.3.1.

We attempted to start sending requests to the NRF service, and it became evident that these services do not have authentication support. This is crucial because now the attacker gets unrestricted access to manipulate and interfere with the NRF component from the 5G Core Network. This could lead to several security risks, such as disruptions, data breaches, or unauthorized system access. Moreover, the lack of authentication opens up the possibility of Man-in-the-Middle (MitM) attacks. In MitM attacks, an adversary can intercept and manipulate communications between two parties without their knowledge. In the context of our testbed, this could lead to severe security vulnerabilities. An attacker could intercept and modify communication between 5G services, potentially compromising the integrity and confidentiality of data.

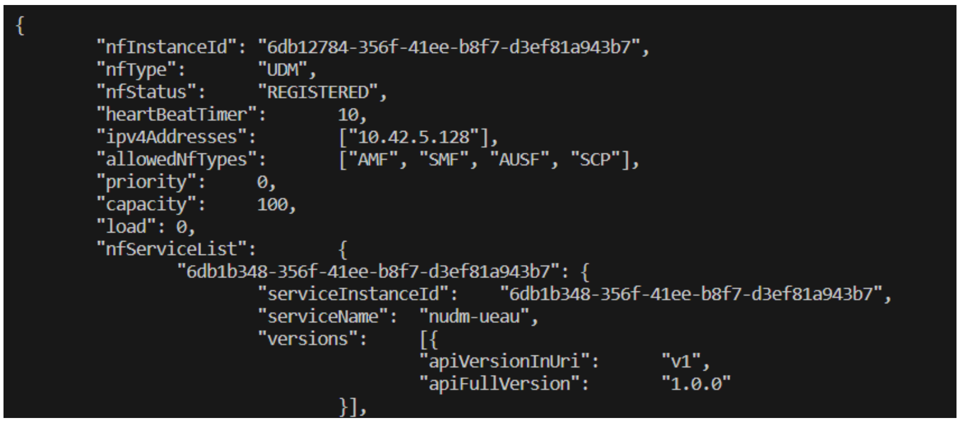
6.5 Exploitation

The exploitation phase focuses on actively leveraging the identified vulnerabilities to assess their potential impact and gain insights into potential security weaknesses.

6.5.1 Hijacking NRF Services

Information Gathering about other Network Functions

When looking into the documentation of 5G Core Network APIs in [33], we can see that NRF has two services: **NF Management** and **NF Discovery**. We started by focusing on the **NF Management Service**, and issued a GET request to the '/nf-instances' endpoint, which returned a payload containing the instance ID of each registered NF. We could go even further by adding a simple query to the endpoint, such as '/nf-instances?nf-type=UDM', to instead of retrieving all the NF instances, only retrieve the instance ID of the UDM. Consequently, by adding the instance ID of the UDM to the endpoint, '/nf-instance/<UDMInstanceID>, we could obtain a JSON payload containing all the information about this NF, including its IP address, status, allowed NF types and registered service instances associated with the UDM, their versions, name, scheme, status, IP endpoints and more. The **NF Discovery Service** is the one used by all other Network Functions to find and locate Services that they need in order to operate. For example, when the AUSF needs to discover a UDM to utilize the authentication credentials generated by it in order to provide the service of authenticating a specific device, it issues a request such as '/nf-instances?target-nf-type=UDM&requester-nf-type=AUSF". Upon sending this request, we received a JSON payload that follows the same format observed in our previous interaction. A fraction of this Json payload is depicted in Figure 6.4.



```
{
        "nfInstanceId": "6db12784-356f-41ee-b8f7-d3ef81a943b7",
        "nfType":       "UDM",
        "nfStatus":     "REGISTERED",
        "heartBeatTimer":       10,
        "ipv4Addresses":        ["10.42.5.128"],
        "allowedNfTypes":       ["AMF", "SMF", "AUSF", "SCP"],
        "priority":     0,
        "capacity":     100,
        "load": 0,
        "nfServiceList":        {
                "6db1b348-356f-41ee-b8f7-d3ef81a943b7": {
                        "serviceInstanceId":    "6db1b348-356f-41ee-b8f7-d3ef81a943b7",
                        "serviceName":  "nudm-ueau",
                        "versions":     [{
                                        "apiVersionInUri":      "v1",
                                        "apiFullVersion":       "1.0.0"
                                }],
```

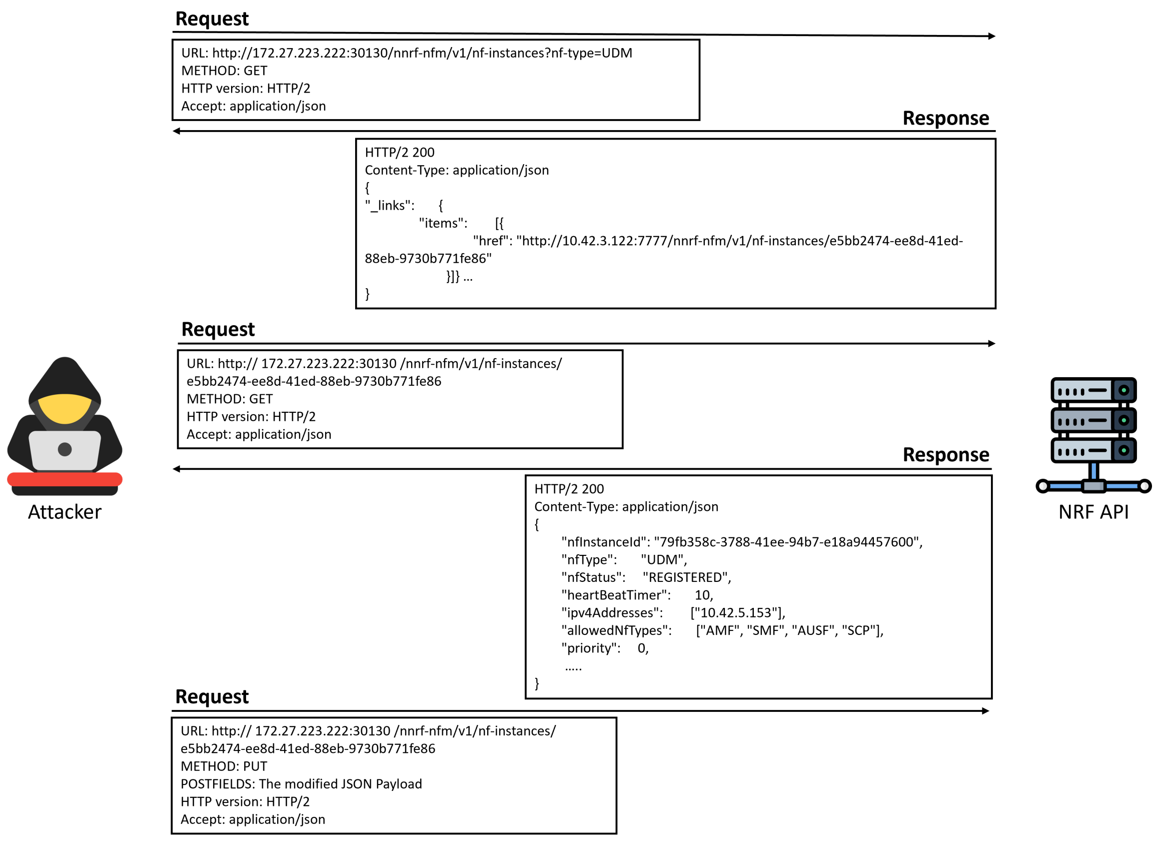
Figure 6.4: JSON payload with UDM Information

The act of gathering information can be very dangerous, as it exposes attackers to critical details about other Network Functions. This could potentially lead to malicious activities that compromise the integrity of the entire network ecosystem. It underscores the importance of safeguarding such sensitive information and implementing authentication measures to prevent unauthorized access and potential exploits.

Modifying and Deleting NF instances

Following the information gathering phase, aimed at acquiring information regarding other Network Functions, the attacker gains the ability to execute potentially disruptive actions. Upon closer examination of the extensive UDM information, a specific parameter stood out – the 'allowedNfTypes'. This parameter holds a list of authorized NFs that can request services from the UDM, including AMF, SMF, AUSF, and SCP. This discovery prompted a question: What if I altered the UDM data by removing AUSF from this list? Since AUSF depends on UDM communication for authentication-related services, my thought process led me to wonder whether removing AUSF from the 'allowedNfTypes' could potentially disrupt the registration process for UE subscribers requiring authentication services within the network. Following this, we provide a comprehensive breakdown of all the steps undertaken to execute this attack, illustrated in Figure 6.5.

Firstly, we extracted the JSON payload containing all the UDM information and removed the 'AUSF' from the 'allowedNfTypes' array. Then, we initiated a PUT request containing the modified payload, resulting in the successful update of the UDM information. Finally , we conducted the redeployment of UE subscribers using UERANSIM to restart the registration process, and the observed outcomes aligned with the anticipated expectations. After analyzing the logs of the ueransim-ue pod, we noticed that the TUN interface (uesimtun0) connecting the UE to the Internet could not be established. Further investigations into the behaviour of AUSF revealed that attempts were made to discover the services of the UDM, including the UE Authentication Service [nudm-ueau]. However, due to the impact of the executed attack, the AUSF encountered challenges in accessing these services and processing the corresponding SBI messages. As a result, this disruption led to the impossibility of completing the UE registration process, thereby demonstrating the potential implications of the attack.



```
URL: http://172.27.223.222:30130/nnrf-nfm/v1/nf-instances?nf-type=UDM
METHOD: GET
HTTP version: HTTP/2
Accept: application/json
```

```
HTTP/2 200
Content-Type: application/json
{
"_links":    {
             "items":     [{
                          "href": "http://10.42.3.122:7777/nnrf-nfm/v1/nf-instances/e5bb2474-ee8d-41ed-
88eb-9730b771fe86"
                       }]} ...
}
```

```
URL: http:// 172.27.223.222:30130 /nnrf-nfm/v1/nf-instances/
e5bb2474-ee8d-41ed-88eb-9730b771fe86
METHOD: GET
HTTP version: HTTP/2
Accept: application/json
```

```
HTTP/2 200
Content-Type: application/json
{
      "nfInstanceId": "79fb358c-3788-41ee-94b7-e18a94457600",
      "nfType":      "UDM",
      "nfStatus":    "REGISTERED",
      "heartBeatTimer":     10,
      "ipv4Addresses":      ["10.42.5.153"],
      "allowedNfTypes":     ["AMF", "SMF", "AUSF", "SCP"],
      "priority":    0,
       .....
}
```

```
URL: http:// 172.27.223.222:30130 /nnrf-nfm/v1/nf-instances/
e5bb2474-ee8d-41ed-88eb-9730b771fe86
METHOD: PUT
POSTFIELDS: The modified JSON Payload
HTTP version: HTTP/2
Accept: application/json
```

Figure 6.5: Attack Flow

Similar to this type of attack, other researchers have demonstrated the process of eliminating Network Function instances [48]. Taking inspiration from their approach, we attempted to replicate a similar attack within our testbed. We executed a DELETE request directed towards the endpoint /nf-instances/<instanceID> to remove a specific Network Function, such as the UDM. Our analysis confirmed that the UDM successfully de-registered from the NRF following the attack. However, it was observed that the UDM quickly re-registered with the NRF almost immediately afterward. This behavior can be attributed to automated processes within the UDM, which continually attempts to register itself with the NRF, in cases it has not already. Therefore, the conducted attack had no significant impact on the operation of the 5G Core Network.

6.5.2 Fuzzing NRF API endpoints

Fuzzing or Fuzz Testing is an automated software testing method that injects invalid, malformed, or unexpected inputs into a system to reveal software defects and vulnerabilities. A fuzzing tool injects these inputs into the system and then monitors for exceptions such as crashes or information leakage [49]. In this Section, we will explore this testing method on the NRF API.

There are many types of fuzzing, but usually when it comes to web applications, we can take one of two approaches. The first one involves utilizing a list of common words, such as directory names, to uncover unknown API endpoints. The second one is using random or generated data that could be for example very long or very short, to try and elicit some odd behavior from within the application. Thus, the list that we choose for each one of the approaches is very important, since an inappropriate selection can result in oversights. To generate a diverse range of payloads, we utilized the SecLists Repository [50]. This repository provides specialized wordlists for security testing purposes. Since we want to perform Fuzzing of the Json Payloads, we selected the wordlist file JSON.Fuzzing.txt, which consists of many several distinct JSON payloads, mainly containing special characters and exceptionally large input strings.

To perform this testing, we developed a custom C program using the libcurl library. The program reads different JSON payloads from this wordlist file, with each line representing a distinct payload. Then we sent multiple requests to the target endpoint **NF Instances** and **NF Instance ID** of the NF Management Service and **NF Instances** of the NF Discovery Service and tracked the server responses. During intervals between executions of our custom C program, we also attempted manual Fuzzing. This involved trying unexpected HTTP methods and submitting requests without payloads for actual HTTP methods that typically require payloads, such as POST and PUT.

The results of the Fuzz Testing are represented in Table 6.1. The first column indicates the name of the endpoint used, followed by the method utilized for each request in the second column, in the third column we mention whether the wordlist of Json Fuzzing is applied and the fourth column displays the response received from the server.

Endpoint	**Method**	**JSON.Fuzzing.txt**	**Response**
NF Instances	GET	No	Status 200
NF Instances	GET	Yes	Status 200
NF Instances	OPTIONS	No	Server Crash
NF Instances	Any Unexpected Method	No	Server Crash
NF Instance ID	DELETE	Yes	Status 400
NF Instance ID	PUT	Yes	Status 400
NF Instance ID	PATCH	Yes	Status 400
NF Instance ID	GET	Yes	Status 400
NF Instance ID	Any Unexpected Method	Yes	Status 403

Table 6.1: Fuzzing Tests Results

Throughout the fuzzing tests, we can conclude that in cases where JSON Fuzzing was applied, the server consistently responded with a status code of 400, indicating that the requests were not accepted due to malformed or invalid JSON payloads. This behavior underscores the NRF API robustness in handling incorrect Json Payloads, as it properly detected and rejected malformed requests. However, a few scenarios demonstrated more critical outcomes. For instance, sending an OPTIONS request to the 'NF Instances' endpoint resulted in a server crash, which can be critical since it is a method that is supposed to handle [33]. Similarly, sending unexpected HTTP methods, like those the server was not expected to receive, led to crashes as well. We can conclude that the endpoint '/nf-instances' (NF Instances) lacks verification or validation of the HTTP Method. Additionally, the tests on the '/nf-instances/nf-instanceID' (NF Instance ID) endpoint revealed that this endpoint already has validation of the HTTP method used. Consequently, whenever unexpected HTTP methods were sent, the server responded with a 403 status code, indicating that the method is not supported.

6.5.3 Flooding Attacks

In this subsection, we will focus on assessing the vulnerability of the NRF within the 5G Core Network to flooding attacks. By subjecting the NRF to controlled attacks, we aim to evaluate its resilience and the potential impact on the connectivity of the registered UE devices, and the communication with other NFs via the SBI. DoS attacks are intentional attempts to disrupt or disable a network service, rendering it inaccessible to legitimate users. In the context of the NRF, such attacks can lead to service degradation, loss of connectivity, and negative user experiences.

TCP SYN Flood

We started by evaluating the SBI's resilience against a high volume of incoming SYN packets and determine its ability to handle such attacks effectively. We initiated the TCP SYN flood attacks using the following command:

```
hping3 -d 1000 -S -p 30130 --rand-source --flood 172.27.223.222
```

For a brief explanation of the selected parameters:

- **-d 1000**: Sets the data size of the TCP packet to 1000 bytes;
- **-S**: Specifies that the packets should be sent with the SYN flag set;
- **-p 30130**: Designates the destination port of the NRF Service;
- **–rand-source**: Generates packets with randomized source IP addresses for added anonymity and adds an additional layer of complexity for defensive mechanisms;
- **–flood**: Triggers the flood mode, sending packets as rapidly as possible.
- **172.27.223.222**: Refers to the target IP address of one Worker Node in the Cluster.

After executing this command, we obtained interesting results. First, attempting to send requests to the NRF interface via the address 172.27.223.222:30130 was not possible. This outcome was a direct consequence of the attack targeting this very address, since the NRF Services were still accessible through other Nodes in the Cluster using the same Node-Port, meaning that the impact of the denial-of-service was more on the Worker Node than the NRF component itself. However, the situation escalated further as the NRF crashed due to a Segmentation Fault error and entered in a "CrashLoopBackOff" state. This term refers to a condition in which a Kubernetes pod repeatedly crashes immediately after starting, triggering the system to restart the pod in an ongoing loop. This mode of operation serves as a safety mechanism to prevent the pod from overloading the resources in its attempt to run [51]. During this state, the NRF Services were completely unavailable from any Node in the Cluster or any other component of the Core Network. Through a straightforward and simple process, we utilized libcurl in C to perform a few rounds of HTTP requests for interaction with the NRF, which alongside insights from the NRF logs, we determined that the average restart time was approximately 76 seconds. Adding to this complexity, the fact that this Worker Node experienced the most impact of the DoS attack, caused all the Network Functions provisioned by this node to stop working. This cascade effect rendered the entire 5G core network unable to operate correctly. The TUN interface that the UE relied upon to establish an Internet connection ceased to exist, and its PDU session also disappeared. This highlights how flooding attacks can impact both a 5G Core Network and a Kubernetes cluster.

In adittion, we also developed a straightforward Scapy program to simulate this attack, to see if we could have different results. The Algorithm 1 shows how this attack was implemented. This python script generates 500 random source IP addresses, simulating different computers. For each of these IP addresses, it sends 15,000 SYN packets to a specific target IP address and port. To optimize the process and send these packets concurrently, the script uses threads. Each thread represents one of the random IP addresses, and they all work together to launch the attack simultaneously. This approach mimics a coordinated attack from multiple sources, increasing its effectiveness. Nevertheless, our experiments did not reveal a significant difference in outcomes between using this custom script and the hping3 tool.

HTTP Flood

We developed a custom C program to conduct an HTTP flood attack on the NRF SBI. This program utilizes the libcurl library, which provides functions for making HTTP requests. It also creates multiple threads, each responsible for sending HTTP requests to a target URL. The targetted endpoint was the '/nf-instances'. Within each thread, an HTTP GET request is sent. The flood attack is initiated by creating a large number of threads (10.000 in this case) and starting them concurrently. Each thread repeatedly sends HTTP/2 requests to the target URL, overwhelming the server's resources and simulating a flood of requests. The completion of the flood attack is indicated when all threads have finished executing.

During our tests, we also explored the use of **h2load**, which is a benchmarking and testing tool for HTTP/2 and HTTP/1.1 protocols. While h2load was not extensively utilized, it provided an alternative approach to validate and compare the results obtained from our custom C program. We started the HTTTP Flood attack using this tool with the following command:

```
h2load http://<serviceIP>/nnrf-nfm/v1/nf-istances\
  -H 'content-type:␣application/json' \
  -t 100 -c 1000 -n 20000
```

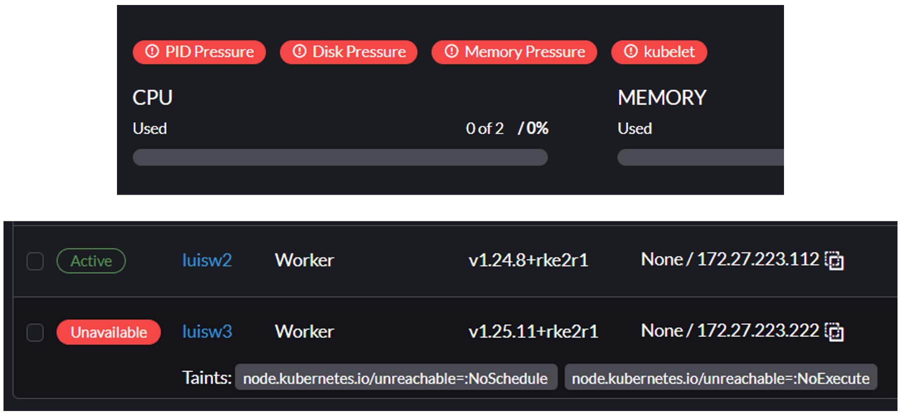
While the results of this test were not as impactful as those observed in the TCP SYN flood attack, they still provided valuable insights into the service behavior under increased load. As the h2load test progressed, we noticed a point at which the tool began displaying the message 'client could not connect to host', indicating that the tool was

```
Algorithm 1: Scapy Program
target_ip ← "172.27.223.222";
target_port ← 30130;
num_rounds ← 15;
num_threads ← 500;
packets_sent ← 0;
send_syn_packets();
create_threads();
Function send_syn_packets
    while true do
        // Generate a random source IP
        src_ip ← ".".join(str(randrange(1, 256)) for _ in range(4));
        for _ in range(num_rounds) do
            // Create IP, TCP, and Raw packets
            ip ← IP(src=src_ip, dst=target_ip);
            tcp ← TCP(sport=RandShort(), dport=target_port, flags="S");
            raw ← Raw(b"X" * 1024);
            packet ← ip / tcp / raw;
            send(packet * 1000, verbose=0);
            packets_sent ← packets_sent + 1000;
        end
    end
Function create_threads
    threads ← [];
    for _ in range(num_threads) do
        thread ← threading.Thread(target=send_syn_packets);
        threads.append(thread);
        thread.start();
    end
Function wait_for_threads
    foreach thread in threads do
        thread.join();
    end
```

unable to establish further connections to the server. This scenario suggested that the server had reached a potential limit in handling incoming connections, which may have led to reduced availability. Interestingly, despite this apparent limitation in connections, we observed that it was still possible to send requests to the server using different IP addresses. Comparing the results of the HTTP flood attack with those of our custom C program, we noticed that the h2load tool was able to exert a more considerable impact on the NRF Service's availability. This contrast underscores the importance of using realistic and specialized tools designed for conducting specific types of attacks, as they can provide more accurate insights into a service behavior under real-world conditions. In conclusion, while the HTTP flood attack did not disrupt the overall functionality of the Core Network or significantly impact the NRF service availability, it highlights the importance of evaluating an application resilience against different attack vectors.

Continuing our research, we examined all the Nodes of the cluster through the Rancher

UI. Notably, the Worker Node which was targeted by the attack, was marked with the Status of "Unavailable". Further investigation revealed that the Master Node was experiencing issues related to Disk and Memory pressure, as shown in Figure 6.6. Additionally, when we attempted to directly access the same VM in the VMware ESXi environment, it was unresponsive and unable to execute any actions.



Active	luisw2	Worker	v1.24.8+rke2r1	None / 172.27.223.112
Unavailable	luisw3	Worker	v1.25.11+rke2r1	None / 172.27.223.222

Taints: node.kubernetes.io/unreachable=:NoSchedule node.kubernetes.io/unreachable=:NoExecute

Figure 6.6: Worker Node Status in Rancher UI

To summarize, both the HTTP and SYN flood attacks overloaded the resources of the Worker Node, causing the NRF Service to be unavailable of receiving requests through the IP address associated with that Node. Although requests could still be routed through other Cluster Nodes using the same NodePort, the prospect of a widespread attack across all Nodes implies that the NRF Service could become unavailable through any Node in the Cluster. Both flooding attacks highlight the potential risks associated with leaving 5G APIs exposed without any form of packet rate control, which can significantly threaten its regular operation.

6.6 Results and Conclusions

In this chapter, we performed a comprehensive cybersecurity assessment following the PTES methodology, starting with the Information Gathering Phase. Through an extensive network scan, we looked to identify hosts and open ports across the network.

Subsequently, we entered the Threat Modeling and Vulnerability Analysis phases, where we proactively assessed potential security risks within the system. Building on insights from our information gathering, we dived into the Kubernetes Cluster, revealing several findings.

- We identified the cluster's Master Nodes and associated services, including 'etcd' and 'kubelet API';
- Notably, the Kubernetes distribution version, v1.24.7+rke2r1, revealed no known vulnerabilities;
- Emphasizing the importance of robust authentication and authorization mechanisms when exposing services outside of the kubernetes cluster;

- Recognizing potential workload imbalances and denial-of-service risks, particularly when accessing services via NodePort.

Utilizing tools like kube-hunter and kubeletctl, we aimed to uncover vulnerabilities within the cluster. Kube-hunter returned a reassuring outcome, reporting no vulnerabilities. Subsequent tests with kubeletctl reinforced the presence of vital authentication and authorization mechanisms of the kubernetes etcd and kube API services.

Moving into the Exploitation Phase within the testbed revealed substantial findings and insights regarding the security weaknesses and vulnerabilities of the NRF component. The Table 6.2 summarizes the results of the tests we conducted for the Kubernetes cluster and the NRF component.

Tests	**Result**
Kubelet Readonly Ports	Pass
Kubelet Secure Ports	Pass
API Server has proper authentication and authorization	Pass
kubectl client is not vulnerable to specific important CVEs	Pass
etcd has proper authentication and authorization	Pass
Presence of authentication and authorization in SBIs	Fail
The NRF service demonstrates resilience against a DoS SYN Flood	Fail
The NRF service demonstrates resilience against a DDoS SYN Flood	Fail

Table 6.2: Tests made in the Penetration Testing

Based on these findings, the next Chapter will focus on the development and implementation of security mechanisms, such as authentication and strategies to mitigate flooding attacks, to address the identified vulnerabilities.

Chapter 7

Proposed Solution and Implementation

This Chapter presents the implementation of the security measures to mitigate the vulnerabilities identified during the Penetration Testing of the 5G testbed. It starts by addressing the proposed security architecture, followed by the development and implementation of the security solution.

7.1 Security Architecture

The security architecture of the 5G testbed was designed to ensure encryption and integrity of the 5G APIs against potential threats and vulnerabilities. In this Section, we will discuss the key elements of the proposed security architecture and how they contribute to a secure environment.

A fundamental aspect of the security architecture regarding the services provided by the Network Functions is the implementation of Transport Layer Security (TLS), in accordance with the 3GPP security specifications [10]. TLS provides encryption, authentication and data integrity, ensuring that the communication between the Network Functions within the testbed remains encrypted and confidential. Within the TLS protocol, mutual authentication is employed to verify the identities of both parties before estalishing a connection, providing an enhanced level of security and ensuring that both the client and the server authenticate each other's identities. This two-way authentication mechanism prevents unauthorized entities from accessing and interacting with the testbed, offering a more robust and reliable means of authentication, reducing the risk of unauthorized access. A practical example showing the communication between NRF and AMF is illustrated in Figure 7.1.

Moreover, the challenge of mitigating Flooding Attacks, such as SYN Flood, targeting the TCP protocol at the transport layer presents a critical security concern that cannot be solely addressed by TLS encryption. SYN Flood attacks exploit the inherent vulnerability in the TCP handshake process, overwhelming the network resources by sending a high volume of incomplete SYN requests, causing legitimate connection requests to be ig-

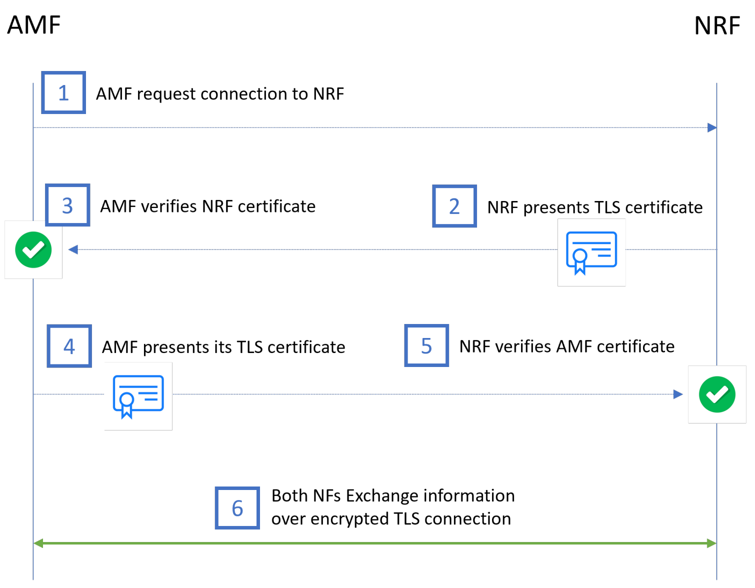

AMF
NRF
1 AMF request connection to NRF
3 AMF verifies NRF certificate
2 NRF presents TLS certificate
4 AMF presents its TLS certificate
5 NRF verifies AMF certificate
6 Both NFs Exchange information over encrypted TLS connection

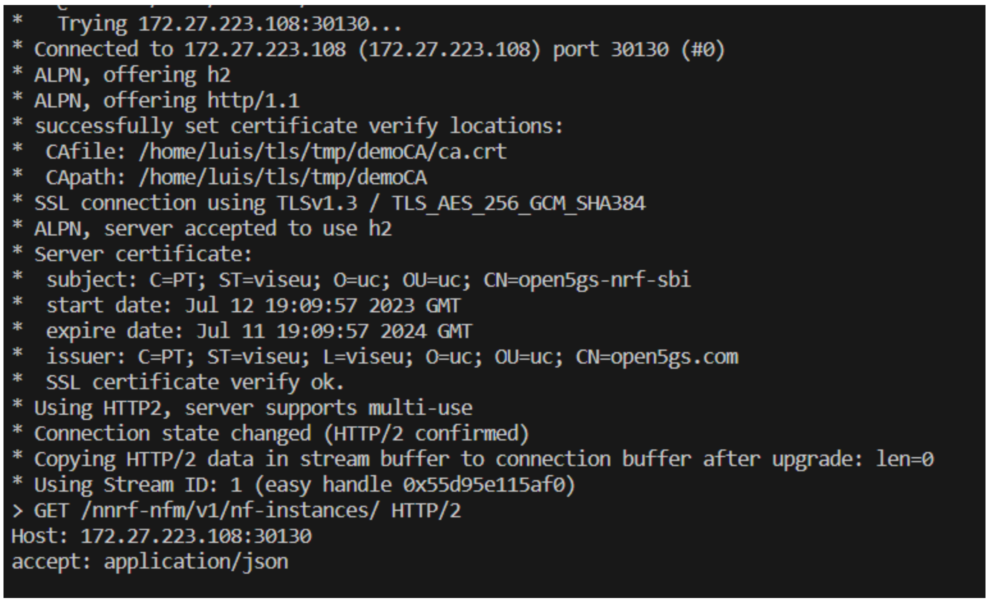
Figure 7.1: AMF and NRF Mutual Authentication Procedure

nored. These attacks always occur before the TLS handshake even takes place, rendering the TLS encryption ineffective against these types of attacks. To effectively counter these attacks at an earlier stage, a multi-layered defense strategy is essential. Implementing eBPF XDP agents on each Worker Node within the Kubernetes cluster provides a comprehensive approach to SYN Flood detection and prevention. These agents are linked to the public network interface through which external applications can access 5G NF Services, and operate at the network level, enabling real-time monitoring of incoming traffic patterns. By leveraging their capabilities, the eBPF XDP agents can quickly identify and filter out malicious SYN Flood traffic, mitigating the impact of the attack before it reaches the Network Function APIs.

7.2 Configuration of Mutual Authentication

In the latest release of Open5GS v2.6.1, significant advancements have been made to enhance the security of the Core Network, including the implementation of TLS support [52]. However, during the deployment of Open5GS within the Kubernetes Cluster, we encountered a challenge with the Openverso Project repository. The repository contains helm charts for each component of the 5G Core Network of Open5GS, but at the time of our research work, it did not provide native support for configuring cacert, key, and cert values in the helm chart [53]. To address this limitation, we took an active role in customizing the deployment configuration, carefully adjusting the configmap files of each 5G component, incorporating the necessary paths and values for the key and certificates. In doing so, we made a deliberate decision not to use the Network Functions' default TLS private keys and certificates provided within the Open5GS source code. The reason behind this is because, although these privates keys and certificates are available, they are also stored in a public repository, making them accessible to anyone and thus posing a potential security risk, since a malicious user can use these default private keys to decrypt the communications between the Network Functions and gain access to sensitive information. To mitigate this risk, we created our own self-signed Certificate Authority (CA) and generated the certificates for each Network Function using the CA private key. These certificates contain the NF public key and a digital signature from the CA. The signed certificates are then distributed to the respective components of the 5G Core, along with their private keys. The generation of certificates and keys was performed using OpenSSL 1.1.1, a widely-used open-source toolkit for SSL/TLS protocols.

In Figure 7.2, we can see the details of the communication scenario where a client initiates a GET request to the NRF API, exposed through NodePort 30130, trying to establish a secure connection:

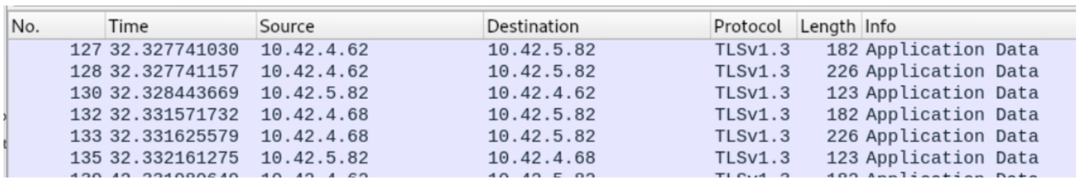


```
*   Trying 172.27.223.108:30130...
* Connected to 172.27.223.108 (172.27.223.108) port 30130 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*  CAfile: /home/luis/tls/tmp/demoCA/ca.crt
*  CApath: /home/luis/tls/tmp/demoCA
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=PT; ST=viseu; O=uc; OU=uc; CN=open5gs-nrf-sbi
*  start date: Jul 12 19:09:57 2023 GMT
*  expire date: Jul 11 19:09:57 2024 GMT
*  issuer: C=PT; ST=viseu; L=viseu; O=uc; OU=uc; CN=open5gs.com
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x55d95e115af0)
> GET /nnrf-nfm/v1/nf-instances/ HTTP/2
Host: 172.27.223.108:30130
accept: application/json
```

Figure 7.2: Client and NRF Communication Details using HTTPS

During the TLS handshake process, the client verified the authenticity of the server's certificate against the CA certificate. The TLS version used for the connection was TLSv1.3, which is the latest version of the TLS protocol providing enhanced security features, along with the `TLS_AES_256_GCM_SHA384` cipher suite. After the TLS handshake, the client successfully upgraded the connection to HTTP/2, a more efficient protocol for data transfer. The client then sent a GET request to the server's API endpoint, representing the application data being exchanged between the client and NRF API. This request and subsequent responses are encrypted and protected from unauthorized access due to the established secure TLS connection. Furthermore, the analysis conducted using Wireshark illustrated in Figure 7.3, with filters applied specifically to the ClusterIP address of the NRF component and the TLS protocol, provides conclusive evidence that the application data transmitted between the components of the 5G Core Network inside the Cluster is encrypted using TLSv1.3.

While the TLS handshake ensures the secure exchange of information and establishes a trusted connection, the choice of TLS version plays a significant role in determining the strength of encryption and the supported cipher suites. To ensure the security of the communications of the SBIs, we used the command line `s_client` from OpenSSL. We confirmed that the server supports TLSv1.3 and TLSv1.2, both of which are secure



```
No.      Time          Source         Destination    Protocol  Length Info
    127 32.327741030  10.42.4.62     10.42.5.82     TLSv1.3    182 Application Data
    128 32.327741157  10.42.4.62     10.42.5.82     TLSv1.3    226 Application Data
    130 32.328443669  10.42.5.82     10.42.4.62     TLSv1.3    123 Application Data
    132 32.331571732  10.42.4.68     10.42.5.82     TLSv1.3    182 Application Data
    133 32.331625579  10.42.4.68     10.42.5.82     TLSv1.3    226 Application Data
    135 32.332161275  10.42.5.82     10.42.4.68     TLSv1.3    123 Application Data
```

Figure 7.3: Wireshark capture of Open5GS traffic

versions offering strong encryption and integrity protection. Additionally, the server refused to support TLSv1.1, which is an outdated version with known vulnerabilities and has been deprecated in Mar 2021 with IETF RFC 8996 [54]. Therefore, it aligns with the desired security standards of the 3GPP Security Specifications for the SBI Domain, addressed in Section 4.3.1.

7.3 Prevention and Detection of SYN Flooding Attacks

In addressing the challenge of mitigating Flooding attacks, we aimed at designing a solution utilizing the capabilities of eBPF and XDP. This solution serves as a countermeasure against SYN Flood vectors by harnessing the potential of in-kernel packet processing. The implemented framework encompasses two essential components: an eBPF program executed within the kernel state and a user-space program used to facilitate several interactions with the eBPF program. Currently, BPF programs running in the kernel state can only be developed in C, or more precisely in **restricted C syntax**, and the only one that can perfectly compile C source code into BPF target files is the **clang compiler** (clang is a compiler front-end for C, C++, Objective-C and other programming languages, using Low Level Virtual Machine (LLVM) as the back-end). The user state programs used to load and unload BPF programs can be developed in multiple languages, either C, Python, Go or many others. We decided to write it in C which works with the **libbpf** and **libxdp** libraries, both providing an abstraction layer that allows the developer to interact with eBPF programs and stored maps in a controlled manner.

7.3.1 eBPF Program

The core of our solution resides within the eBPF program, designed to perform early-stage packet filtering and processing at the network layer. This program employs a modified version of the **Fixed Window Algorithm** to implement rate limiting, a technique for mitigating SYN Flood attacks. The Fixed Window algorithm tracks the frequency of specific events—in this case, SYN packets—and allows only a predefined number of these events within a specific time window. This approach is designed to mitigate SYN Flood attacks, by limiting the rate at which SYN packets are processed, thereby preventing the network from being overwhelmed.

Although rate limiting algorithms can be effective at mitigating DoS attacks originating from a single source, they fall short when confronted with Distributed attacks. As mentioned in Section 4.5.1, these algorithms lack the capability to distinguish between legitimate and compromised users. Consequently, standard rate limiting mechanisms affect all forms of traffic equally. Furthermore, when dealing with a flood of SYN packets originating from millions of different sources, the attempt to counteract this by imposing a pre-defined limit of SYN packets per IP address becomes irrelevant.

The code starts by defining two types of eBPF maps, both serving as key-value data structure that enables programs to retrieve and store information and acts as a bridge for passing information across the user space and the kernel space programs:

- **tcp_count:** In this hash map, each key corresponds to an IP Address that has attempted a TCP connection. For every key, there is a corresponding data structure named **tcp_info_packet**, holding essential information including the number of SYN packets originating from the specific IP address, the time values that facilitate the calculation of time windows, and a flag. This flag works as a dynamic indicator within the algorithm, toggling between states to track if an IP address is considered infected.
- **zero_flag_counter:** This map only has one entry, which points to a data structure called **ddos_stats** containing information regarding a Distributed SYN Flood attack, including the time frame it was detected, a variable which defines how many IP addresses have sent too many SYN packets, and the number of current open tcp connections.

The Algorithm 2 shows a simplified pseudo-code representation of the logic implemented to develop this solution. The **xdp_drop_tcp()** function is executed at the XDP hook and serves as the heart of the packet processing logic. It begins by extracting relevant header information from the packet to determine if it is a TCP SYN packet. If so, the algorithm proceeds to track and enforce rate limiting. The algorithm fetches the previously stored information about the source IP address from the **tcp_count map** using the **bpf_map_lookup_ elem()** function. It then calculates the time elapsed since the last recorded packet and assesses whether the incoming packet complies with the allowed rate. If the rate limit is exceeded, the packet is dropped. If the rate is within acceptable limits, the program updates the stored information in the map with the current time and an incremented count of packets. However, if the SYN packets originate from legitimate users, a subsequent ACK packet is usually present. The code utilizes this contextual information to monitor potentially suspicious connections, leveraging a flag that transitions to '1' upon receiving a SYN packet and resets to '0' after receiving an ACK packet. As a result, the rate limiting measures are selectively applied to those transmitting consecutive SYN packets—specifically, the compromised IP addresses launching a SYN Flood attack. Meanwhile, legitimate connections can freely submit requests to the service without facing rate limitations.

This approach would be feasible enough if we wanted to prevent flooding attacks from a single source. However, in order to mitigate distributed attacks, we needed to introduce more complexity. Therefore, when we identify a significant number of infected IP addresses, we take an additional step: we prevent any new IP addresses from establishing TCP connections for a predefined duration. It is important to note that already established connections are not affected by this DROP (unless they exhibit malicious behavior). Moreover, attackers might use a more subtle strategy, for example sending just

```
Algorithm 2: Handling SYN Flood Attacks using eBPF and XDP
Initialize BPF maps and configuration parameters;
Function xdp_drop_tcp(event);
Retrieve and process incoming packet Packet ← GetNextPacket()
if IsNotDestinationPort(Packet, TargetPort) then
    PassPacket(Packet)
end
if IsSYNPacket(Packet) then
    if IsNotACKPacket(Packet) then
        info ← Pointer to TCP Info Packet;
        stats ← Pointer to DDoS Stats;
        if stats is NULL then
            Create a new DDoS Stats entry if it does not exist
        else
            if info is NULL then
                if stats.open_tcp_conn $\geq$ 40 seconds then
                    DropPacket(Packet)
                end
                if stats.amount_of_infected_IP_Addresses $\geq$ 5 then
                    DropPacket(Packet)
                end
                Store New IP Address with its new info
            else
                if info->count_syn $\geq$ MAX_TCP_SYN_PACKETS_PER_SEC then
                    if now_usec1 - stats->start_time < 60 seconds then
                        DropPacket(Packet)
                    end
                    if info->flag then
                        It is an infected IP Address -> Gets blocked for 20 seconds
                         UpdateConnectionStats(info, stats)
                    end
                    if elapsed_time_usec $\geq$ 20 seconds then
                        20 seconds have passed -> Open connections
                         UpdateConnectionStats(info, stats) PassPacket(Packet)
                    end
                    DropPacket(Packet)
                else
                    PassPacket(Packet)
                end
            end
        end
    end
end
else if IsACKPacket(Packet) then
    UpdateConnectionStats(info, stats)
end
```

a few malicious packets per IP address to avoid triggering the rate limit while still overwhelming the server. To counter this, we also keep track of the number of open TCP connections. If this count surpasses a certain threshold, we proactively DROP new connection attempts regardless of the source IP.

In essence, this eBPF program combines packet inspection, rate limiting, and map-based state tracking to effectively mitigate and detect SYN Flood attacks originating from several IP addresses. This approach mitigates a range of scenarios involving SYN flooding attacks, improving the protection against different attack vectors.

7.3.2 User-space Program

The integration of the eBPF XDP program into our architecture was facilitated through the user-space program developed as part of our solution. Given that we chose to expose our 5G Core Services through the NodePort mechanism, external requests naturally flow through the public interface, **ens192**. By attaching the XDP program to this interface, a layer of packet analysis is imposed upon every packet destined for any of our 5G Network Functions. This integration fortifies the robustness of our entire 5G Core Network by enhancing its resilience against potential malicious activities, notably SYN Flood attacks.

7.4 Validation of the eBPF program

To effectively validate the performance of this mechanism, we decided to evaluate it under several types of SYN Flooding Attacks against the NRF component exposed through NodePort, which was the target of our Penetration Testing. We thought about three possible scenarios:

- **1. Single Source SYN Flood:** This involves sending a substantial volume of SYN packets from one single source, similar to a DoS attack;
- **2. Multi-Source SYN Flood:** This scenario comprises sending a high number of SYN packets from various different sources;
- **3. Rate-Limited SYN Flood:** Here, a minimal number of SYN packets are sent from several sources, with the aim of trying to not activate the rate limiting.

To simulate these attack scenarios, we employed the same tools used in the Penetration Testing, namely **Scapy** and **Hping3**. For the first attack scenario, we employed Hping3 and executed a straightforward command, resembling the one utilized in the Section Flooding Attacks 6.5.3, with the exception of excluding the "–rand-source" flag to mimic a DoS attack:

```
hping3 -d 1000 -S -p 30130 --flood 172.27.223.222
```

After conducting the SYN Flood attack for approximately 80 seconds, during which a total of 15.622.227 packets were sent, the NRF service demonstrated a satisfactory level of resilience. It neither crashed nor experienced any resource-related issues, such as excessive pressure on CPU utilization. Throughout the attack, the NRF service also remained available to receive legitimate external or internal requests from trusted sources that possessed the necessary SSL/TLS certificates, effectively blocking only the attacking IP address. These findings were further validated by the results obtained from **Wireshark**, as depicted in Figure 7.4, with filters applied to only capture SYN Packets in the port of the NRF Service. Consider the attacking Virtual Machine IP address as 172.27.223.227, the IP address of the legitimate user as 172.27.223.108, and the pre-defined rate limiting value as 3 SYN Packets per second.

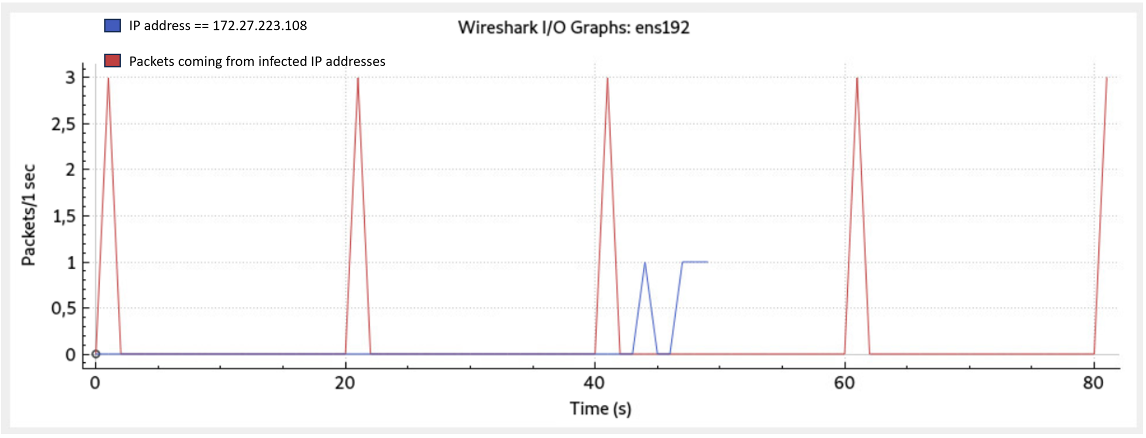
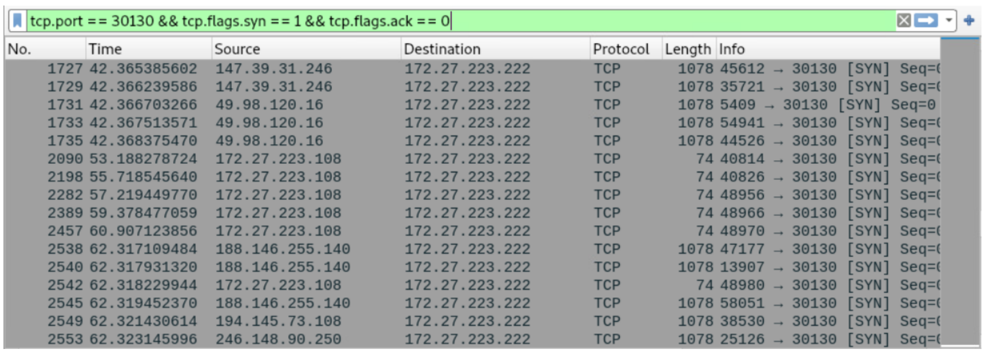


Figure 7.4: Wireshark IO graph during the DoS attack

By analyzing the IO graph, it becomes evident that the source IP address responsible for initiating TCP connections exhibited malicious behavior, allowing for a maximum of three concurrent SYN packets. Once this threshold was reached, the eBPF agent immediately recognized this IP as malicious and initiated a preemptive measure where the subsequent TCP SYN packets originating from this malicious IP address were effectively dropped for the next 20 seconds. Moreover, during this 20-second period, other legitimate users could access and submit requests without interruption. We selected this time window duration because it aligns best with our testing purposes; it is a fairly low duration enabling us to speed up the testing process, but can easily be adjusted for real-world scenarios.

In the second attack scenario, we decided to use the same scapy program as we used in the Penetration Testing, in Section 6.5.3. After launching the attack, the eBPF program effectively countered the DDoS attack initiated by the Scapy program. It instantly detected the potential DDoS threat and took action by blocking any new connections made via SYN requests for a pre defined duration of 40 seconds. This proactive defense not only strengthened the NRF service against subsequent SYN Flood attacks but also ensured uninterrupted service for legitimate users who were already engaged with the NRF service. Results from **wireshark** depicted in Figure 7.5 also confirmed the attack mitigation, as it displayed the infected IP addresses trying to get into the NRF service, like a group effort. But there was one address, 172.27.223.108, that belonged to a real user. Even with all the other traffic, this user could still send as many requests as he wanted. In summary, we could tell who was causing problems and who was not.



```
tcp.port == 30130 && tcp.flags.syn == 1 && tcp.flags.ack == 0

No.    Time          Source           Destination      Protocol Length Info
  1727 42.365385602  147.39.31.246    172.27.223.222   TCP      1078 45612 → 30130 [SYN] Seq=
  1729 42.366239586  147.39.31.246    172.27.223.222   TCP      1078 35721 → 30130 [SYN] Seq=
  1731 42.366703266  49.98.120.16     172.27.223.222   TCP      1078 5409 → 30130 [SYN] Seq=0
  1733 42.367513571  49.98.120.16     172.27.223.222   TCP      1078 54941 → 30130 [SYN] Seq=
  1735 42.368375470  49.98.120.16     172.27.223.222   TCP      1078 44526 → 30130 [SYN] Seq=
  2090 53.188278724  172.27.223.108   172.27.223.222   TCP        74 40814 → 30130 [SYN] Seq=
  2198 55.718545640  172.27.223.108   172.27.223.222   TCP        74 40826 → 30130 [SYN] Seq=
  2282 57.219449770  172.27.223.108   172.27.223.222   TCP        74 48956 → 30130 [SYN] Seq=
  2389 59.378477059  172.27.223.108   172.27.223.222   TCP        74 48966 → 30130 [SYN] Seq=
  2457 60.907123856  172.27.223.108   172.27.223.222   TCP        74 48970 → 30130 [SYN] Seq=
  2538 62.317109484  188.146.255.140  172.27.223.222   TCP      1078 47177 → 30130 [SYN] Seq=
  2540 62.317931320  188.146.255.140  172.27.223.222   TCP      1078 13907 → 30130 [SYN] Seq=
  2542 62.318229944  172.27.223.108   172.27.223.222   TCP        74 48980 → 30130 [SYN] Seq=
  2545 62.319452370  188.146.255.140  172.27.223.222   TCP      1078 58051 → 30130 [SYN] Seq=
  2549 62.321430614  194.145.73.108   172.27.223.222   TCP      1078 38530 → 30130 [SYN] Seq=
  2553 62.323145996  246.148.90.250   172.27.223.222   TCP      1078 25126 → 30130 [SYN] Seq=
```

Figure 7.5: Wireshark capture during the DDoS attack

The IO graph illustrated in Figure 7.6 further confirms that the eBPF program quickly identifies a potential DDoS attack when confronted with a continuous inundation of SYN Packets in rapid succession. Consequently, it responds by dropping any new TCP connections for the next 40 seconds, as it was pre-defined in the Algorithm 2. In this graphical representation, the blue line distinctly illustrates the uninterrupted establishment of complete TCP connections by legitimate users with the NRF service.

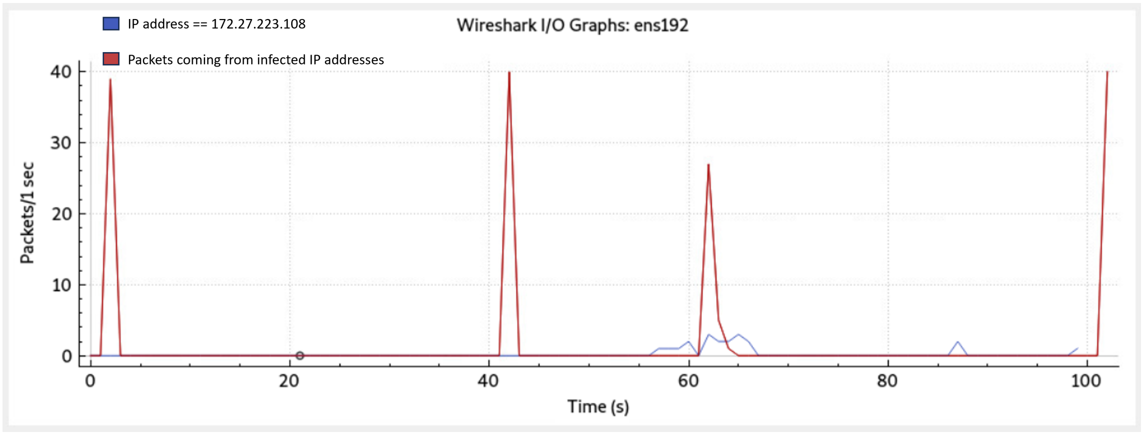


Figure 7.6: Wireshark IO graph during the DDoS attack

In the third attack scenario, we used the command "hping3 -d 1000 -S -p 30130 –rand-source –flood 172.27.223.222", during which a total of 27.748.287 packets were sent. In this case, the attacker sent packets as quickly as possible, each from a random IP address. However, most of these packets were limited to just one per IP address, so they rarely reached the predefined rate limit. Yet, even in this situation, the eBPF program proved effective. It monitors the number of open TCP connections and, once it reaches our predefined limit of 40 connections, it starts rejecting new requests, as shown in the IO graph represented in Figure 7.7. The overall results were similar to those of the second attack scenario, but instead the time window dropping new TCP connection attempts coming from different sources, was 60 seconds this time. This is because, in the Algorithm 2, we can see that there is a condition that checks whether the DDoS attack detection was triggered by the fact that it received 40 concurrent SYN packets, dropping new TCP connections for 60 seconds, or by the fact that 5 different IP addresses sent 3 SYN packets concurrently, dropping new TCP connections for 40 seconds.

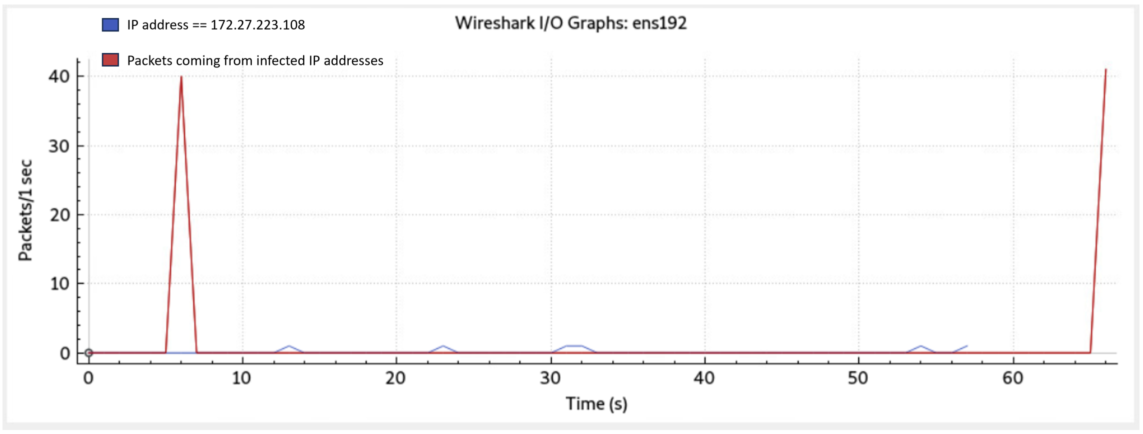


Figure 7.7: Wireshark IO graph during the DDoS attack

7.5 Chapter Conclusions

This Chapter presented the steps taken to enhance the 5G testbed against the identified vulnerabilities in the Penetration Testing. The chapter initiates with the conceptualization and execution of a security architecture designed to safeguard the 5G APIs through the incorporation of TLS and mutual authentication to ensure encryption and data integrity. Furthermore, Flooding attacks were addressed through the implementation of an eBPF XDP agent to help mitigating multiple types of SYN Flood attacks. We evaluated the effectiveness of this solution through simulations involving various scenarios. The results indicated that the algorithm is sufficiently capable of preventing both the NRF component and the targeted Node of the Kubernetes Cluster from becoming overwhelmed. It is important to note that all the results presented here were obtained using pre-defined values, chosen for the purpose of testing the eBPF XDP program's effectiveness. In real-world situations, these parameters would certainly need to be adjusted depending on specific requirements.

Chapter 8

Conclusions and Future Work

In this research work, we focused on evaluating and enhancing the security of 5G Networks. In order to achieve this goal, we started by studying 5G network security. This exploration involved a deep dive into 3GPP technical specifications to gain a comprehensive understanding of the specific security requirements associated with 5G Core Networks. Then, we extended our efforts to investigate potential security challenges and attacks that might be present, study security mechanisms to improve security against DDoS/flooding attacks, and ways to provide visibility and observability of SBI messages in the 5G Core. Then, we proceeded to the implementation of a pre-defined testbed aiming to address its security and conducted the Initial Experiments. Next, we conducted a Penetration Testing following the PTES methodology to address the security of the kubernetes infrastructure and Open5GS. In this Section we found interesting types of attacks that might be possible through 5G SBIs, some vulnerabilities in several API endpoints that are specifically to the current 5G version being used in the testbed (open5GS) and evaluated the potential impacts of SYN flooding attacks against the 5G Core, more precisely the NRF component. Building upon the findings from our Penetration Testing, we initiated the implementation of security measures. This step included the generation of SSL/TLS certificates and the adjustment of deployment configurations to enable mutual authentication between Network Functions SBIs. Furthermore, we also implemented a mechanism to mitigate and detect SYN Flooding attacks using eBPF XDP and evaluated its performance against different types of SYN Floods to see if they would be feasible enough to mitigate the attacks from the Penetration Testing.

During this research work we faced several challenges. Initially, it was very difficult to have a starting point, on what would be the real focus of the security testing efforts. Secondly, sending and retrieving requests from open5GS components was not a straightforward task. Since open5GS uses nghttp2 as the implementation of the http/2 protocol, the only way it was possible to invoke services was using curl or the nghttp/2 client with the –http2-prior-knowledge. Thus, we could not use tools such as Postman, Burp Suite and some automated tools to build requests and make API calls to 5G services. Another major challenge was the implementation of the eBPF XDP program. The documentation on these tools was not detailed and there were almost no tutorial on the setup of these tools or the implementation of mechanisms using eBPF.

Something that could have been very interesting, though unfortunately, we could not successfully put it into action, was enhancing observability and visibility within the realm of 5G Core Networks, as discussed in the State of the Art Chapter, using eBPF. This would have been our next step if we had more time to execute it. The reason this is so appealing is because having observability in 5G Core networks enables us to detect and establish guidelines for specific events, especially when it comes to 5G-specific SBI messaging. This would be incredibly valuable for understanding and managing the entire network's behavior.

References

[1] F. Z. Yousaf, M. Bredel, S. Schaller, and F. Schneider, "Nfv and sdn—key technology enablers for 5g networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2468–2478, 2017.

[2] C. Bouras, A. Kollia, and A. Papazois, "Sdn nfv in 5g: Advancements and challenges," in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 107–111, 2017.

[3] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and communication networks*, vol. 9, no. 18, pp. 5803–5833, 2016.

[4] B. Chun, J. Ha, S. Oh, H. Cho, and M. Jeong, "Kubernetes enhancement for 5g nfv infrastructure," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1327–1329, 2019.

[5] A. Dutta and E. Hammad, "5g security challenges and opportunities: A system approach," in *2020 IEEE 3rd 5G World Forum (5GWF)*, pp. 109–114, 2020.

[6] I. Ahmad, S. Shahabuddin, T. Kumar, J. Okwuibe, A. Gurtov, and M. Ylianttila, "Security for 5g and beyond," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3682–3722, 2019.

[7] H. Kim, "5g core network security issues and attack classification from network protocol perspective.," *J. Internet Serv. Inf. Secur.*, vol. 10, no. 2, pp. 1–15, 2020.

[8] Q. Tang, O. Ermis, C. D. Nguyen, A. D. Oliveira, and A. Hirtzig, "A systematic analysis of 5g networks with a focus on 5g core security," *IEEE Access*, vol. 10, pp. 18298–18319, 2022.

[9] "The penetration testing execution standard (ptes)." (accessed: 07/05/2023, from: `https://github.com/penetration-testing-execution-standard`).

[10] S. Rommer, P. Hedman, M. Olsson, L. Frid, S. Sultana, and C. Mulligan, *5G Core Networks: Powering Digitalization*. Academic Press, 2019.

[11] 3GPP, "System architecture for the 5g system," *3GPP TS 23.501 V15. 3.0*, 2018.

[12] "What is encryption?." (accessed: 07/05/2023, from: `https://www.cisco.com/c/en/us/products/security/encryption-explained.html/`).

[13] S. K. Tayyaba and M. A. Shah, "Resource allocation in sdn based 5g cellular networks," *Peer-to-Peer Networking and Applications*, vol. 12, no. 2, pp. 514–538, 2019.

[14] "What is a ddos attack?." (accessed: 07/05/2023, from: `https://avinetworks.com/glossary/ddos-attack/`).

[15] A. Tacca, E. C. V. Boas, and G. P. Aquino, "Aspectos de segurança cibernética em redes móveis 5g,"

[16] S. Lal, T. Taleb, and A. Dutta, "Nfv: Security threats and best practices," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 211–217, 2017.

[17] M. Zolanvari, "Sdn for 5g," 2015. (accessed: 12/12/2022, from: `https://www.cse.wustl.edu/~jain/cse570-15/ftp/sdnfor5g.pdf`).

[18] M. Dargin, "Secure your sdn controller," 2018. (accessed: 12/12/2022, from: `https://www.networkworld.com/article/3245173/secure-your-sdn-controller.html`).

[19] D. Bloom, R. Filkovsky, and B. Lifshitz, "The top 4 ddos attack vectors threatening 5g networks," 2022. (accessed: 12:17, 14/01/2023, from: `https://www.allot.com/blog/top-ddos-attack-vectors-threatening-5g/`).

[20] "Understanding rate limiting algorithms." (accessed: 07/05/2023, from: `https://www.quinbay.com/blog/understanding-rate-limiting-algorithms`).

[21] A. El Kamel, H. Eltaief, and H. Youssef, "On-the-fly (d) dos attack mitigation in sdn using deep neural network-based rate limiting," *Computer Communications*, vol. 182, pp. 153–169, 2022.

[22] "ebpf xdp: The basics and a quick tutorial." (accessed: 07/05/2023, from: `https://www.tigera.io/learn/guides/ebpf/ebpf-xdp/`).

[23] G. Bertin, "Xdp in practice: integrating xdp into our ddos mitigation pipeline," in *Technical Conference on Linux Networking, Netdev*, vol. 2, pp. 1–5, The NetDev Society, 2017.

[24] T. A. N. do Amaral, R. V. Rosa, D. F. C. Moura, and C. E. Rothenberg, "An in-kernel solution based on xdp for 5g upf: Design, prototype and performance evaluation," in *2021 17th International Conference on Network and Service Management (CNSM)*, pp. 146–152, 2021.

[25] "The importance of a unified observability solution in security implementation." (accessed: 07/05/2023, from: `https://www.elastic.co/pt/blog/unified-observability-solution-security-implementation`).

[26] D. Soldani, P. Nahi, H. Bour, S. Jafarizadeh, M. F. Soliman, L. Di Giovanna, F. Monaco, G. Ognibene, and F. Risso, "ebpf: A new approach to cloud-native observability, networking and security for current (5g) and future mobile networks (6g and beyond)," *IEEE Access*, vol. 11, pp. 57174–57202, 2023.

[27] "ebpf vs. sidecar containers for 5g visibilityl." (accessed: 07/05/2023, from: `https://www.mantisnet.com/blog/ebpf-v-sidecar-containers-5g-observability`).

[28] F. Parola, F. Risso, and S. Miano, "Providing telco-oriented network services with ebpf: the case for a 5g mobile gateway," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, pp. 221–225, 2021.

[29] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[30] T. L. F. Kube-VIP, "Kube-vip," 2022. (accessed: 21/12/2022, from: https://kube-vip.io/.

[31] aligungr, "Ueransim," 2022. (accessed: 23/02/2023, from: https://github.com/aligungr/UERANSIM/).

[32] "Helm charts collection of openverso project." (accessed: 09/03/2023, from: https://github.com/Gradiant/openverso-charts).

[33] jdegre, "Openapi descriptions of 3gpp 5g apis (release 18)." (accessed: 10/04/2023, from: https://github.com/jdegre/5GC_APIs).

[34] S. Lee, "Implemented endpoints open5gs." (accessed: 20/05/2023, from: https://github.com/open5gs/open5gs/blob/873cf398c733cf5e780619eab1117604de30931f/lib/sbi/message.h#L83).

[35] S. Lee, "Ue-sm.c file open5gs." (accessed: 20/05/2023, from: https://github.com/open5gs/open5gs/blob/main/src/udm/ue-sm.c).

[36] Antirez, "Hping3 network tool." (accessed: 12:17, 20/03/2023, from: https://github.com/antirez/hping/).

[37] "Scapy." (accessed: 07/05/2023, from: https://github.com/secdev/scapy/).

[38] "Network exposure." (accessed: 07/05/2023, from: https://www.ericsson.com/en/core-network/network-exposure).

[39] nmap, "Nmap." (accessed: 22/03/2023, from: https://github.com/nmap/nmap).

[40] "whois." (accessed: 05/07/2023, from: https://github.com/rfc1036/whois).

[41] "kube-hunter." (accessed: 05/07/2023, from: https://github.com/aquasecurity/kube-hunter).

[42] "Whatweb." (accessed: 05/07/2023, from: https:https://github.com/urbanadventurer/WhatWeb).

[43] "K8s threat model." (accessed: 05/07/2023, from: https:https://cloudsecdocs.com/container_security/theory/threats/k8s_threat_model/).

[44] M. Lancini, "The current state of kubernetes threat modelling," 2020. (accessed: 12:17, 25/08/2023, from: https://blog.marcolancini.it/2020/blog-kubernetes-threat-modelling/).

[45] macedogm, "[cve-2022-3172]," 2022. (accessed: 15:03, 25/08/2023, from: https://github.com/rancher/rancher/issues/38994).

[46] cyberark, "kubeletctl," 2020. (accessed: 17:03, 25/08/2023, from: https://github.com/cyberark/kubeletctl).

[47] T. Tsujikawa, "nghttp2," 2023. (accessed: 20:09, 26/08/2023, from: https://github.com/nghttp2/nghttp2).

[48] S. Dudek, "Intruding 5g sa core networks from outside and inside." (accessed: 14/04/2023, from: `https://penthertz.com/blog/Intruding-5G-corenetworks-from-outside-and_inside.html`).

[49] "What is fuzz testing and how does it work?." (accessed: 07/05/2023, from: `https://www.synopsys.com/glossary/what-is-fuzz-testing.html/`).

[50] danielmiessler, "Seclists." (accessed: 25/04/2023, from: `https://github.com/danielmiessler/SecLists`).

[51] "Network exposure." (accessed: 07/05/2023, from: `https://komodor.com/learn/how-to-fix-crashloopbackoff-kubernetes-error/`).

[52] S. Lee, "v2.6.1 - upgrade to release-17." (accessed: 03/07/2023, from: `https://open5gs.org/open5gs/release/2023/03/08/release-v2.6.1.html`).

[53] "Tls on sbi issue 131." (accessed: 03/07/2023, from: `https://github.com/Gradiant/openverso-charts/issues/131`).

[54] "Rfc 8996 - deprecating tls 1.0 and tls 1.1." (accessed: 05/07/2023, from: `https://datatracker.ietf.org/doc/html/rfc8996`).