# 5G Core Implementation Comparison

A School Project for COMP 5900, 5G Networks, Carleton University

Students:
May Wang - 101099445
Parsa Kootzari - 101316375

Course Instructor:
Jun Huang

Teaching Assistant:
Mohammed Abuibaid

April 4, 2024

**Abstract**

This project focuses on the deployment and analysis of 5G networks, utilizing different 5G Core implementations like Open5GS and Free5GC. In this study, the primary aim is to find the most robust, secure, and efficient 5G Core implementation. We will deploy the two most commonly used open-source 5G Core implementations, Free5GC and Open5GS, and conduct comprehensive vulnerability scans using tools like Trivy and Clair with different deployment methodologies. The analysis will also assess each implementation's performance, network latency, response time, and resource consumption. The project's crux lies in identifying weaknesses, the extent of the attack surface, and the types of vulnerabilities.

**Key words and phrases.** 5G, Core, Performance Metric, Security, Vulnerability.

# Contents

# 1 Introduction

The 5G core network is built upon the Evolving Packet Core (EPC) framework and is designed to cater to three main types of uses: Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low-Latency Communication (URLLC), and Massive Machine Type Communications (mMTC). eMBB focuses on providing faster and more efficient mobile internet access, while URLLC ensures highly dependable and low-latency communication critical for applications like autonomous vehicles or remote surgery. mMTC enables the seamless connectivity of a vast number of devices, facilitating the Internet of Things (IoT) and Industrial Internet of Things (IIoT) applications. The 5G system also supports 4C capabilities—Communication, Computing, Control, and Content Delivery—enabling seamless data transfer, computational power at the network edge, efficient resource management, and content delivery. With the introduction of 5G technology, new applications such as real-time online gaming, virtual reality (VR), ultra-high definition (UHD) streaming, autonomous vehicles, tactile Internet with millisecond latency, and IoT and IIoT are emerging, revolutionizing various industries and everyday experiences [5].

In the 5G ecosystem, three key components drive advanced connectivity: the 5G core, Radio Access Network (RAN), and User Equipment (UE). The 5G core functions as the network's central intelligence, managing communication among devices and services through different services. The RAN comprises base stations and antennas, facilitating wireless connections between user devices and the core network using technologies like massive MIMO and beamforming. The UE encompasses user devices like smartphones and IoT gadgets, interacting with the RAN to access network services. Together, these components deliver the high-speed, low-latency connectivity essential for diverse applications, from mobile broadband to IoT and virtual reality experiences [4].

The 5G core is built on a Service-Based Architecture, focusing on delivering services and functions rather than individual network nodes. This approach ensures adaptability and efficiency. Key to this architecture is Control and User Plane Separation (CUPS), which allows for independent scaling of these components. The User Plane Function (UPF) operates at the gateway between the Radio Access Network (RAN) and external networks like

the Internet. It handles tasks such as packet routing, forwarding, quality of service management, and traffic measurement. The UPF also serves as an anchor point for mobility between different radio access technologies (RATs) when needed. The Control Plane Function includes the Session Management Function (SMF), IP address allocation for user equipment (UE), policy enforcement, and session management. Supervising control signalling, user data security, and authentication is the role of the Access and Mobility Management Function (AMF), ensuring seamless communication between different layers of the network [6].

Numerous projects are developing 5G cores based on 3GPP specifications. Some notable open-source implementations include Magma [14], Free5GC [11], Open5GS [13], and SD-Core [17]. In our project, we focus on comparing two well-known implementations: Free5GC and Open5GS. Our analysis covers two main areas. Firstly, we evaluate performance by deploying both in similar environments and measuring metrics like latency, throughput, and resource usage. Secondly, we examine security, conducting thorough vulnerability scans using tools like Trivy [26] and Clair [21] with docker image deployment. Through this analysis, we aim to provide insights into the strengths and weaknesses of these implementations to assist decision-making for future deployments.

# 2 Background

## 2.1 5G Core Implementations

The two 5G core implementations that we are studying in this project are Free5GC and Open5GS. Free5GC, developed by NCTU, is an open-source project written in the Go language, providing a development environment for the 5G core network based on 3GPP Release 15. It comprises three developmental stages for the 5G core network. Initially, Stage 1 focuses on the 5G non-stand-alone (NSA) architecture, defining essential elements like AMF, SMF, and UPF, which function on existing 4G elements. In Stage 2, the 5G stand-alone (SA) architecture is introduced, integrating additional core elements that operate using a service-based interface (SBI). Finally, Stage 3 enhances the architecture by incorporating the uplink classifier (ULCL) and non-3GPP interworking function (N3IWF), essential for supporting non-

3GPP access in Stage 2. Presently, Free5GC has reached Stage 3 of development [7]. Open5GS, on the other hand, is an open-source project for the 5G core, developed in the C language and aligned with the standards of 3GPP Release 17. The commercial licensing for Open5GS is managed by NextEPC. This project has successfully implemented crucial core elements including AMF, SMF, and the user plane function (UPF), all of which are essential components within the 5G core infrastructure [7].

## 2.2 UERANSIM

UERANSIM [19] is an advanced open-source simulator for 5G User Equipment (UE) and Radio Access Network (RAN), including gNodeB, which acts as a 5G mobile phone and a base station, respectively. It's a valuable tool for testing the 5G Core Network and conducting in-depth studies on the 5G System. UERANSIM is the world's first and only open-source implementation for 5G Stand-Alone (SA) UE and gNodeB, setting it apart as a pioneering resource in the field. In this project, we used UERANSIM to simulate the RAN and UE and connect them to the 5G core to test its performance.

## 2.3 Clair

Clair [15] is a vulnerability scanning tool that is open-source and used for evaluating container images. The analysis involves comparing the layers of container images and a database of known vulnerabilities from sources such as Debian, Ubuntu, and Oracle Oval Database [16]. Once the program completes its security scan, it generates detailed reports highlighting any vulnerabilities detected during the process. The provided report identifies vulnerabilities and includes the Common Vulnerability Enumeration (CVE) rating, also known as the Common Vulnerability Scoring System (CVSS) score [2], obtained from the National Vulnerability Database (NVD) [12].

## 2.4 Trivy

Another open-source vulnerability scanner, Trivy [26], is made for thorough security scanning of file systems, git repositories, and container images. Trivy primarily aims to conduct scans on container images to identify any known

vulnerabilities in the installed packages and libraries. The process involves scanning container images by examining the file system and analyzing package manifests to detect vulnerable software components. Trivy utilizes known vulnerability information from sources such as NVD, Red Hat, and Debian [20]. It builds a vulnerability database every six hours and automatically updates it during operation.

# 3 Performance Evaluation

## 3.1 Environment Setup

To assess the performance metrics of various 5G core implementations accurately, they need to be tested in isolated environments with similar specifications and configurations. To establish a fully functional testing environment for evaluating these implementations, we utilized virtual machines (VMs) on a personal laptop to host the different components of the network. The laptop boasts 8GB of RAM and operates on an Intel 11th Gen Core i5-1135G7 processor running at 2.40GHz. Both Free5GC and Open5GS were deployed directly on the VMs, without the use of management or orchestration technologies such as Docker and Kubernetes. This decision was made to ensure that any performance differences between the two 5G core implementations could be accurately measured. By opting for a simpler deployment stack with fewer intermediate technologies, we aimed to achieve more stable and reliable testing conditions. This approach enables us to attribute any variations in the performance of the 5G core networks directly to the implementation itself. Below are the VM configurations that we created to host the components of the 5G core:

- **Free5GC**: deployed on a VM with 2GB of RAM and 2 CPU cores.
- **Open5GS**: deployed on a VM with 2GB of RAM and 2 CPU cores.
- **UERANSIM**: deployed on a VM with 2GB of RAM and 2 CPU cores.

In this setup, the 5G core services are deployed on a single VM and we used the gNB and UE to connect to the Free5GC and Open5GS. Here are

the screenshots that show for both Free5GC and Open5GS, the gNB and UE simulator were successfully connected to the 5G core.

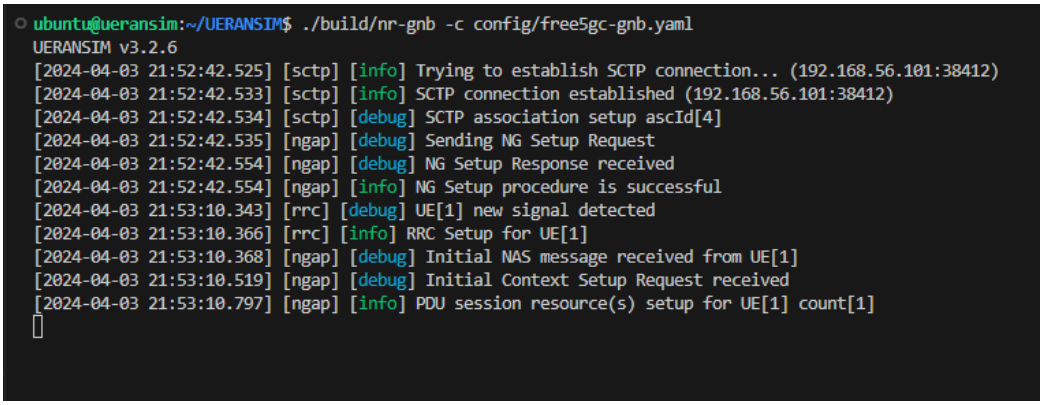


Figure 1: gNB connection to Free5GC core

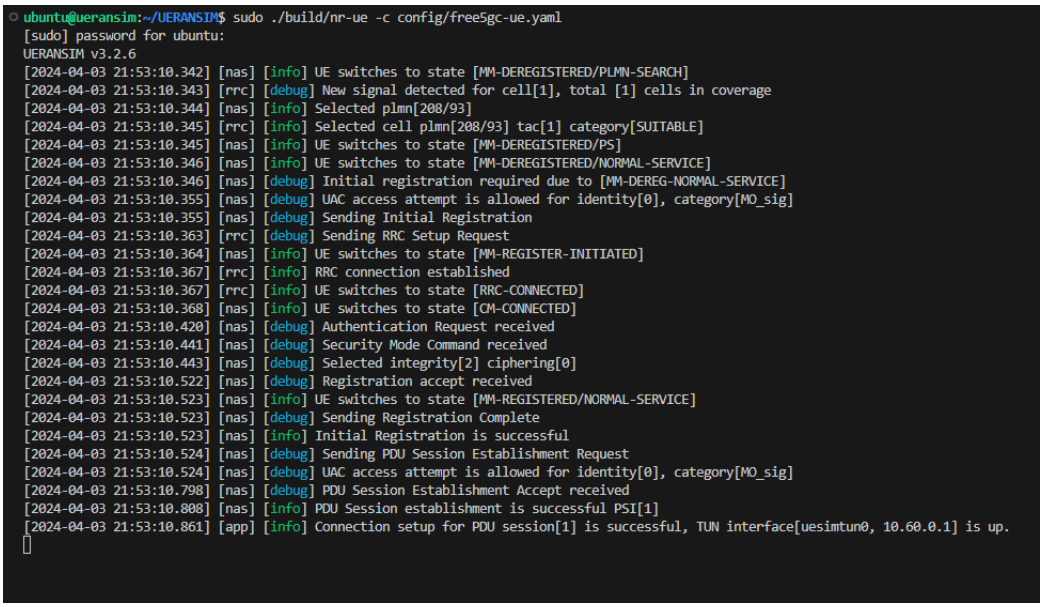


Figure 2: UE connection to Free5GC core

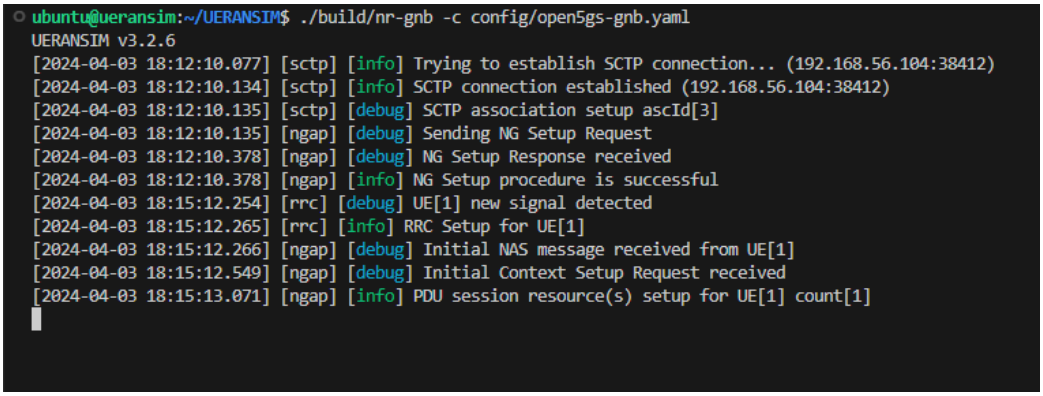


Figure 3: gNB connection to Open5GS core

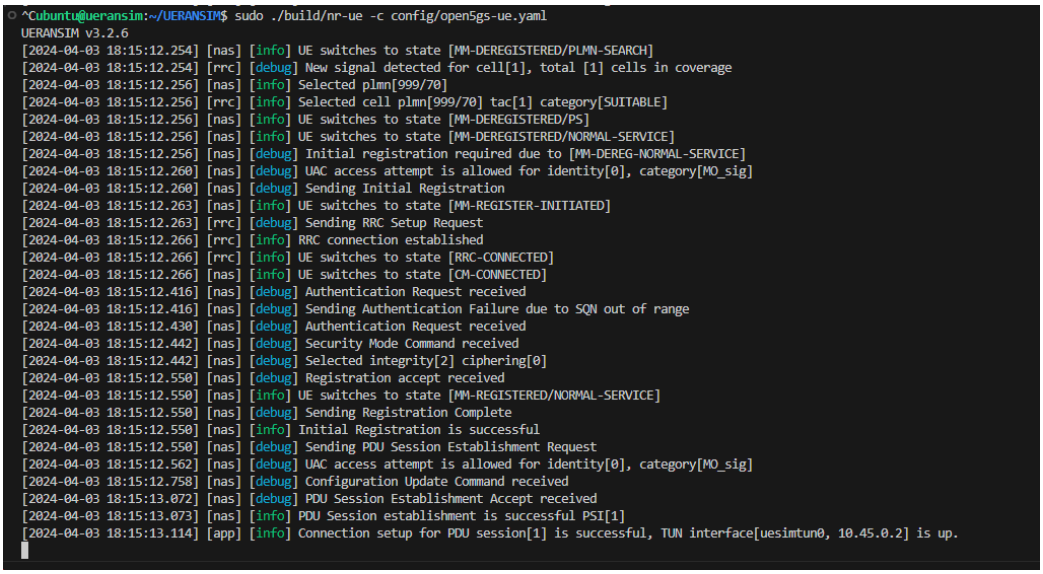


Figure 4: UE connection to Open5GS core

After the connection between UE and 5G core is established, a new interface with name *uesimtun0* will be created and the traffic that is sent to this interface will go through the 5G core network before reaching its destination. Figure 5 shows how we sent ping requests through the 5G core network. The difference between the RTT and latency of requests sent through *uesimtun0* is also visible in this picture.

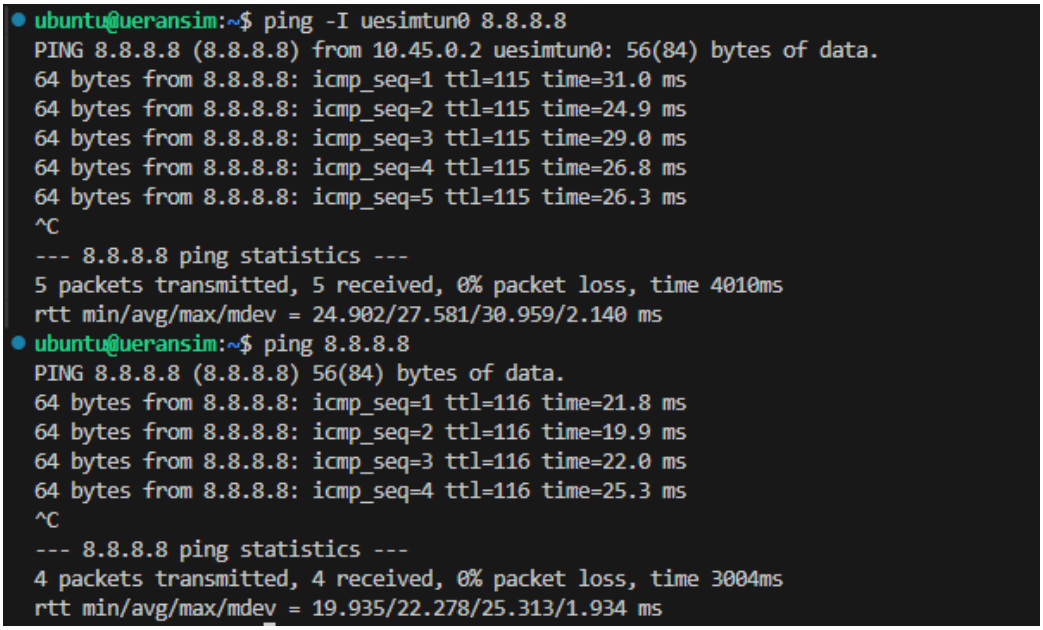


Figure 5: Ping command going through 5G core network using uesimtun0 interface

## 3.2 Findings

To evaluate the performance of the 5G core network, we measured and analyzed four metrics: latency, uplink and downlink throughput, and resource usage. We compared these metrics across various 5G core networks to assess their efficiency. Additionally, we examined the network's performance under two scenarios: first, when routing traffic from User Equipment (UE) through the 5G core network to the internet, and second, when accessing the internet directly from the virtual machine (VM) without deploying the 5G core network. This comparison allowed us to understand the overhead introduced by utilizing the 5G core network. All the metrics that are going to be discussed in the following section were collected by connecting a single UE to each 5G core network.

### 3.2.1 Latency

In Figure 6, the distinction in latency between the virtual machine (VM) and the two 5G core networks is illustrated. To assess latency, we employed the ping command, directing ping requests to the IP address 8.8.8.8. To showcase the latency differences, we utilized the *ping -I uesimtun0* command line argument to designate the interface for data transmission. Latency measurements were conducted 15 times for both scenarios, and the outcomes are visualized in the following figure.

As shown in the illustration, when requests are routed through the 5G core network, there is an additional delay ranging from 2 to 4 milliseconds. However, there's little variance in latency between Free5GC and Open5GS. The primary distinction lies in the First and last requests, where Open5GS exhibits notably higher round-trip times (RTT) compared to Free5GC. This difference can be linked to network statistics at the time of request transmission. Overall, the results from latency measurements reveal no significant distinction in latency between Free5GC and Open5GS.

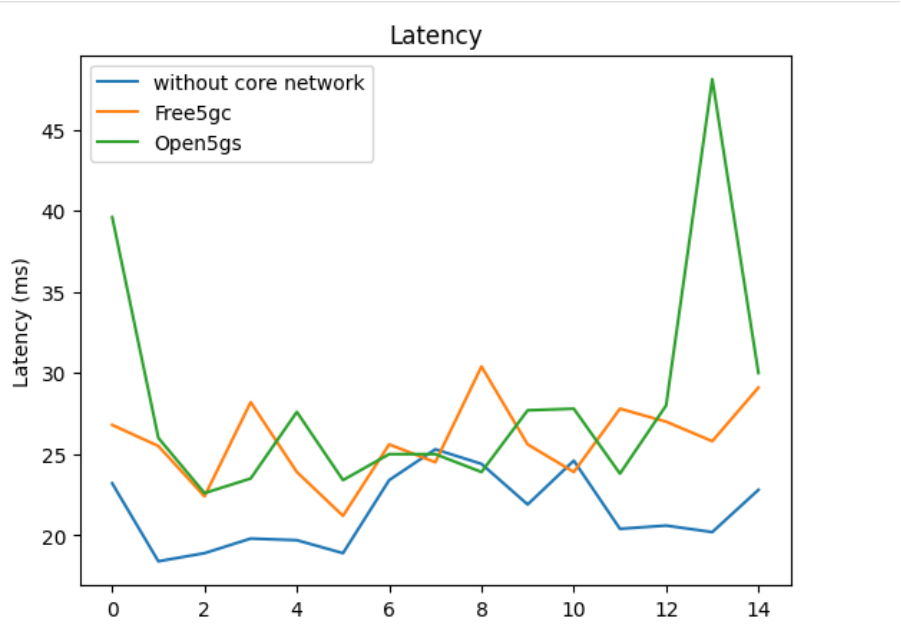


Figure 6: Latency measurements

### 3.2.2 Throughput

Figures 7 and 8 illustrate the downlink and uplink measurements of the network. To gauge throughput, we employed the *speedtest-cli* [18] Python library. In the initial experiment, we utilized the speedtest-cli code without any extra configuration, ensuring that packets were sent and received via the default network interface without traversing the 5G core. Subsequently, in the second scenario, we disabled the default network interface and designated *uesimtun0* as the default. In order to set *usesimtun0* to be the default network interface of the VM first the current default network interface should be disabled using this command *sudo ifconfig enp0s3 down* and then the new default interface should be specified using this command *sudo ip r add default dev uesimtun0*.

After the default interface is changed, we executed the same speed test Python program, enabling the speed test packets to traverse the 5G core, which allowed us to collect the measurements of uplink and downlink speeds under this configuration. This experiment was performed for both Free5GC and Open5GS.

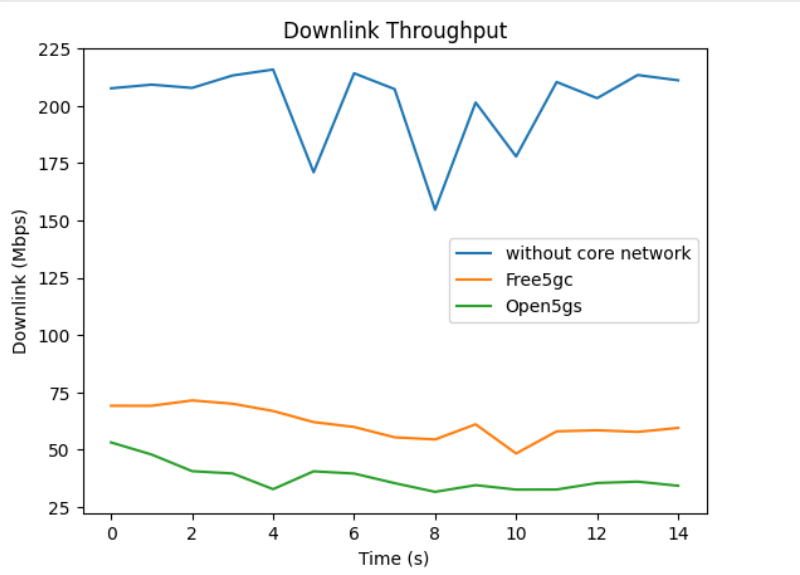


Figure 7: Downlink measurements

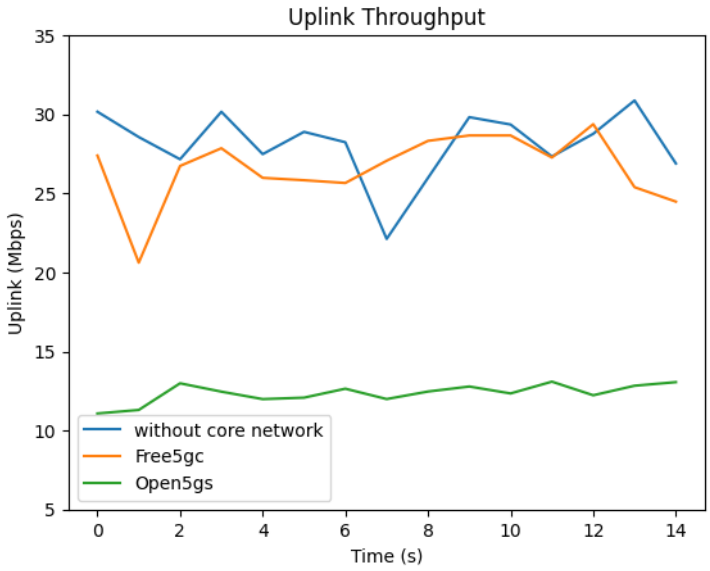


Figure 8: Uplink measurements

The throughput measurements were conducted 15 times for each scenario. Additionally, to validate the reliability of the results, we attempted to measure the downlink speed using *wget —bind-interface=<uesimtun0-IP>*, which operates similarly to ping and directs wget traffic through a designated network interface. We downloaded a 60MB file three times without specifying uesimtun0 as the network interface and three times using uesimtun0. The download speed results for both scenarios remained consistent with the outcomes obtained from the speedtest-cli Python script.

As indicated in the figures, there's a noticeable drop in downlink speed when traffic is routed through the 5G core. This decline is particularly significant with Open5GS since it is experiencing a more pronounced decrease in downlink throughput. In order to be sure that these changes in the throughput are not the result of maximum throughput configuration in the time of UE registration, we checked the parameters that the UEs were registered with, in both Free5GC and Open5GS. In the Free5GC network, the UE's maximum downlink throughput was configured at 200 Mbps, and in the case of Open5gc, the UE was registered with a downlink capacity of 2 Gbps. Therefore, this reduction cannot be attributed to the maximum downlink

and uplink capacity of the User Equipment (UE).

In terms of uplink throughput, Free5GC doesn't show a significant drop in supported uplink speed for the UE. However, Open5GS does demonstrate a significant decline in uplink throughput. Overall, based on the collected metrics from both Free5GC and Open5GS, it's clear that Free5GC provides better service and throughput for connected UEs.

### 3.2.3 Resource Consumption

This section will concentrate on evaluating the resource utilization of 5G networks. Regarding the performance analysis, we focus on two metrics: CPU consumption and memory consumption. In our experiment, the initial state is idle, meaning that the User Equipment (UE) is neither transmitting nor receiving any traffic through the 5G core network. However, after a few seconds, we initiated the download of a large file from the UE using this command *wget –bind-interface=<uesimtun0-IP> <requested file>*. This action compels the 5G network to deliver the highest achievable throughput to the UE for a specific duration. Throughout the whole experiment, we monitored the CPU and memory consumption of the network using a Python script that is running on the VM that is hosting the 5G core. The experiment shows the extent of resource utilization when servicing the UEs compared to when it is idle.

Figure 9 shows the CPU consumption and as it is shown the CPU consumption will significantly increase when the 5G network is sending data to the UE. In the case of Free5GC, as soon as the download of the file starts from the UE end the CPU consumption will increase and it fluctuates between 15 to 40 percent. The Mean CPU consumption of Free5GC VM is 27.08 percent. However, in the case of Open5GS, serving the UE requires much more processing resources and the CPU consumption range is between 50 to 80 percent. The Mean CPU consumption of Open5GS VM is 62.13 percent. The mean CPU consumption for both scenarios was measured after the file download had started.

Figure 10 shows the memory consumption of the 5G core network and as it is shown the memory consumption doesn't change whether the 5G core is sending data to the UE or not. The displayed memory utilization reflects the total memory employed by the 5G core Virtual Machine (VM).

Since both 5G core implementations operate on the same OS (Ubuntu 20.04 server) and have only essential software dependencies and services installed, any variation in memory usage can be directly attributed to the 5G core network services. Based on the results of resource consumption measurements, Open5gc demands more memory compared to Free5GC, with Free5GC consuming approximately 530 MB while Open5GS utilizes around 860 MB of memory. Overall, it's clear that Free5GC is more efficient in utilizing host resources, needing less processing power and memory to serve connected User Equipments (UEs).

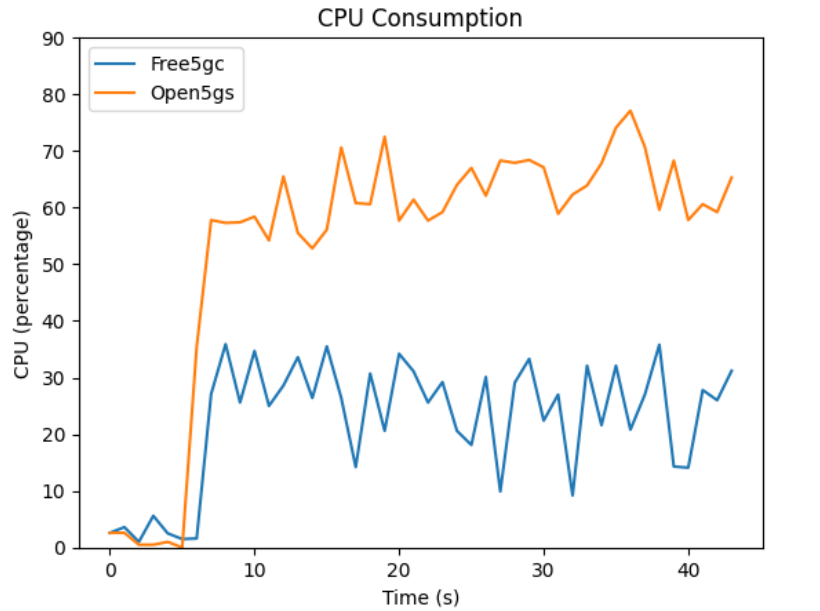


Figure 9: Free5GC CPU consumption measurements

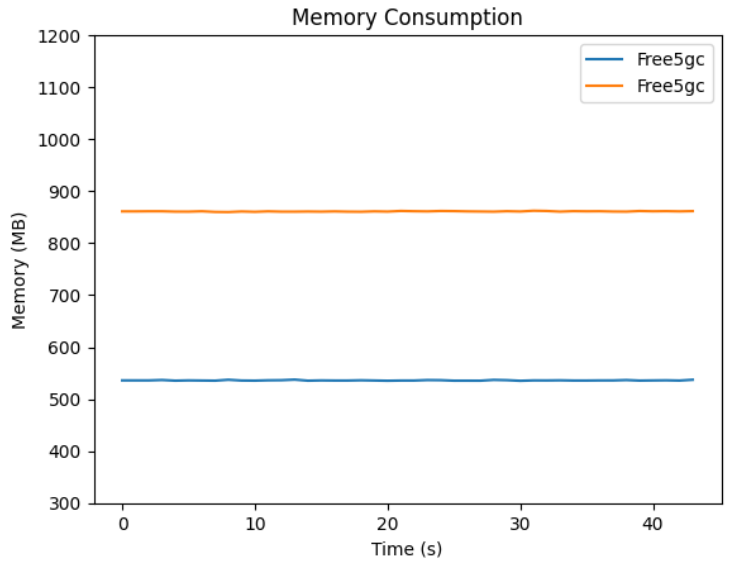


Figure 10: Free5GC memory consumption measurements

# 4 Security Evaluation

## 4.1 Methodology

### 4.1.1 Docker Deployment

Docker was employed for the deployment of Free5GC [11] and Open5GS [13], utilizing their official docker-compose files that are compatible with their most recent versions. This approach was chosen because these files are readily available and updated, ensuring compatibility and ease of deployment. For Open5GS, version 2.6.6 [25] was selected, while Free5GC was deployed using version 3.3.0 [24].

Upon successful deployment, Free5GC will contain more images compared to Open5GS. This is because Free5GC will have separate images for each individual component, whereas Open5GS only has a single primary image during runtime.

| | | | | |
|---|---|---|---|---|
| free5gc/chf | <none> | 4555a3c2427a | 5 months ago | 46.1MB |
| free5gc/smf | v3.3.0 | f03dc13fa8aa | 9 months ago | 24.6MB |
| free5gc/udm | v3.3.0 | 9b6215bf0a15 | 9 months ago | 22.6MB |
| free5gc/nrf | v3.3.0 | b4b1ecf1ac94 | 9 months ago | 28MB |
| free5gc/ausf | v3.3.0 | 7c1952789afd | 9 months ago | 23.9MB |
| free5gc/nssf | v3.3.0 | efaae85d547d | 9 months ago | 21.8MB |
| free5gc/webui | v3.3.0 | 724f77370f79 | 9 months ago | 115MB |
| free5gc/amf | v3.3.0 | 527021e3cbea | 9 months ago | 27.7MB |
| free5gc/upf | v3.3.0 | 6abd9349af50 | 9 months ago | 123MB |
| free5gc/pcf | v3.3.0 | a84fca3ed266 | 9 months ago | 28.4MB |
| free5gc/udr | v3.3.0 | 854121fab907 | 9 months ago | 28.3MB |
| free5gc/n3iwf | v3.3.0 | 4f964e0a9aff | 9 months ago | 27MB |

Figure 11: Free5GC Docker Images

| | | | | |
|---|---|---|---|---|
| /ubuntu-latest-open5gs-build | latest | c2f307dc1784 | 8 hours ago | 1.04GB |
| ueransim | latest | 01c91702cec2 | 44 hours ago | 155MB |
| n3iwue | latest | 43a6b6d036a1 | 44 hours ago | 170MB |
| /ubuntu-latest-open5gs-base | latest | 2bd04d195d01 | 4 days ago | 573MB |

Figure 12: Open5GS Docker Images

### 4.1.2 Container Image scanning with Trivy

We conducted static security analysis using Trivy and Clair on both implementations of 5GC. To conduct a scan with Trivy, first need to install the software, which is available for various operating systems. Once installed, the scanning process can be initiated by executing the command:

```
$ trivy image [OPTIONS] TARGET_IMAGE
```

Where `TARGET_IMAGE` specifies the name of the container image to be scanned, Trivy then accesses a comprehensive vulnerability database and analyzes the specified image for known vulnerabilities, including those in OS packages (like Debian, Ubuntu, and Alpine) and application dependencies (such as Ruby Gems and Python pip). The output of this scan provides a comprehensive list of detected vulnerabilities, including their levels of severity. This allows developers to effectively identify and resolve potential security concerns in their container images.

### 4.1.3 Container Image scanning with Clair

Using Clair, an open-source vulnerability scanning tool, can greatly enhance the security of container images. The initial step involves setting up a Clair database, which can be obtained from the GitHub repository [21]; this is accomplished by executing the command:

```
$ docker run -p 5432:5432 -d --name db arminc/clair-db
   :latest
```

Which deploys the latest version of the Clair database in a Docker container. Subsequently, the Clair application itself is started with the command:

```
$ docker run -p 6060:6060 --link db:postgres -d --name
    clair arminc/clair-local-scan:latest
```

Linking it to the previously started database container. This setup ensures that Clair is fully operational and ready to scan container images for vulnerabilities.

For the actual scanning process of a container image, the `clair-scanner` utility is employed. The scanning command is

```
$ sudo ./clair-scanner --ip YOUR_LOCAL_IP IMAGE_NAME
```

where `YOUR_LOCAL_IP` should be replaced with the local IP address of the machine running the scan, and `IMAGE_NAME` with the name of the container image to be scanned. This process identifies known vulnerabilities in the container image, leveraging Clair's comprehensive vulnerability database, and outputs a detailed report.

## 4.2 Results

The complete set of results can be accessed through our GitHub link: `https://github.com/may010/5G-Core-Implementation-Comparison`.

Below are a few snippets of output generated by Clair and Trivy:

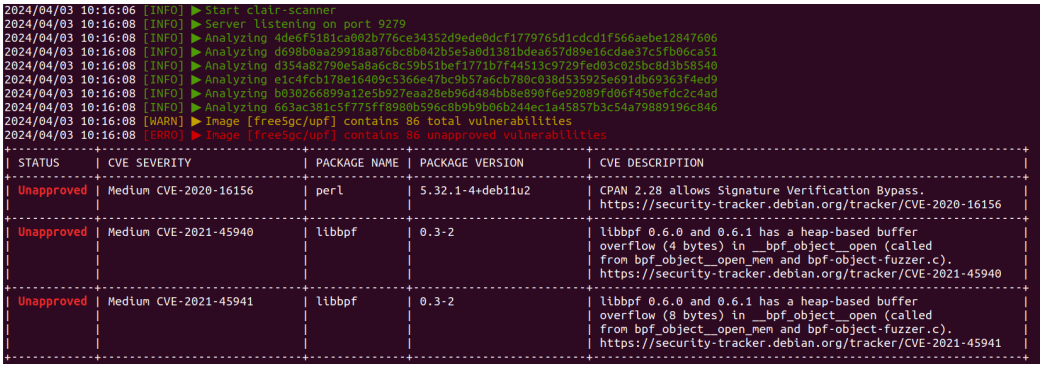


Figure 13: Partial Output Sample from Clair Security Scanner Scanning free5gc/upf

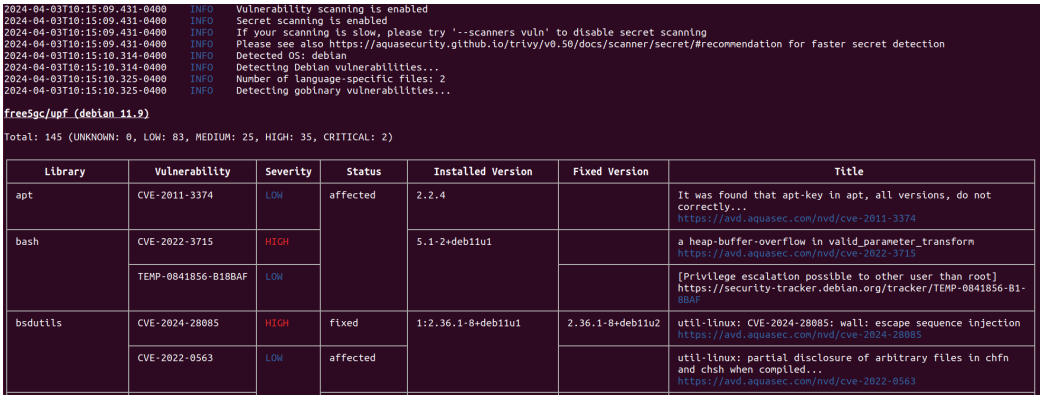


Figure 14: Partial Sample Output Generated by Trivy Security Scanner Scanning free5gc/upf

### 4.2.1 Vulnerabilities Detected in Each Docker Image

**Higher Detection by Trivy**: Based on Table 1, Trivy shows higher detection capabilities by identifying a greater number of vulnerabilities in comparison to Clair. This suggests that Trivy may have a more extensive ability to identify vulnerabilities or perhaps access alternative databases or sources of vulnerability information.

**High Number of Shared Vulnerabilities**: Both programs share a significant number of vulnerabilities, which amounts to 2,382 in total, indicating a significant agreement on the identified flaws. The presence of this overlap

suggests that these vulnerabilities are widely recognized and included in the databases of both programs.

**Comparing Free5GC Components to Base Image and WebUI image**: Upon observation, it was found that the Free5GC individual components present barely any vulnerabilities. However, the base image and webui image are very susceptible to flaws. Several potential explanations were: 1) The individual components of Free5GC may have been designed, developed, and tested with security in mind. These components may be focused on specific functionalities such as authentication, session management, or data routing. Due to this focused approach, the developers may have been more diligent in implementing security measures correctly. 2) The base image and webui images likely depend on external libraries, frameworks, or tools. These dependencies may introduce vulnerabilities that are beyond the control of the Free5GC developers. Vulnerabilities in dependencies can stem from outdated versions, lack of security patches, or inherent flaws in the external software. 3) The base image and webui images may have a larger attack surface compared to individual components. For example, webui images expose functionality to external users, increasing the risk of attack vectors such as cross-site scripting (XSS) or SQL injection. Similarly, the base image may include various services and configurations, increasing the potential for vulnerabilities. 4) Individual components of Free5GC may receive more regular updates and patches to address security vulnerabilities. However, the base image and webui images may receive a different level of attention or may not be updated as frequently, leading to a higher likelihood of vulnerabilities persisting over time.

| Image Name | Clair CVE Count | Trivy CVE count | Shared Vulnerabilities Number |
|---|---|---|---|
| free5gc_amf | 1 | 0 | 0 |
| free5gc_ausf | 1 | 0 | 0 |
| free5gc_base | 744 | 1174 | 731 |
| free5gc_chf | 1 | 2 | 0 |
| free5gc_n3iwf | 1 | 0 | 0 |
| free5gc_n3iwue | 375 | 482 | 374 |
| free5gc_nrf | 1 | 0 | 0 |
| free5gc_nssf | 1 | 0 | 0 |
| free5gc_pcf | 1 | 0 | 0 |
| free5gc_smf | 1 | 0 | 0 |
| free5gc_udm | 1 | 0 | 0 |
| free5gc_udr | 1 | 0 | 0 |
| free5gc_ueransim | 366 | 462 | 365 |
| free5gc_upf | 87 | 145 | 78 |
| free5gc_webui | 77 | 128 | 68 |
| open5gs_base | 0 | 244 | 0 |
| open5gs_build | 0 | 244 | 0 |
| open5gs_dev | 0 | 257 | 0 |
| open5gs_webui | 768 | 1942 | 766 |

Table 1: Docker Image Vulnerabilities Detected by Clair and Trivy

#### 4.2.2 Vulnerability in Each Prototype Implementation

Based on Figure 15, Free5GC has a higher total number of detected CVEs compared to Open5GS. The number of CVEs detected for both prototypes varies significantly between the tools used. For instance, Trivy detects more CVEs than Clair in both cases, suggesting it might be more thorough or has a broader database of CVEs it checks against. The discrepancy in detection between Clair and Trivy might indicate differences in the scope of their scans, their CVE databases, or their detection capabilities.

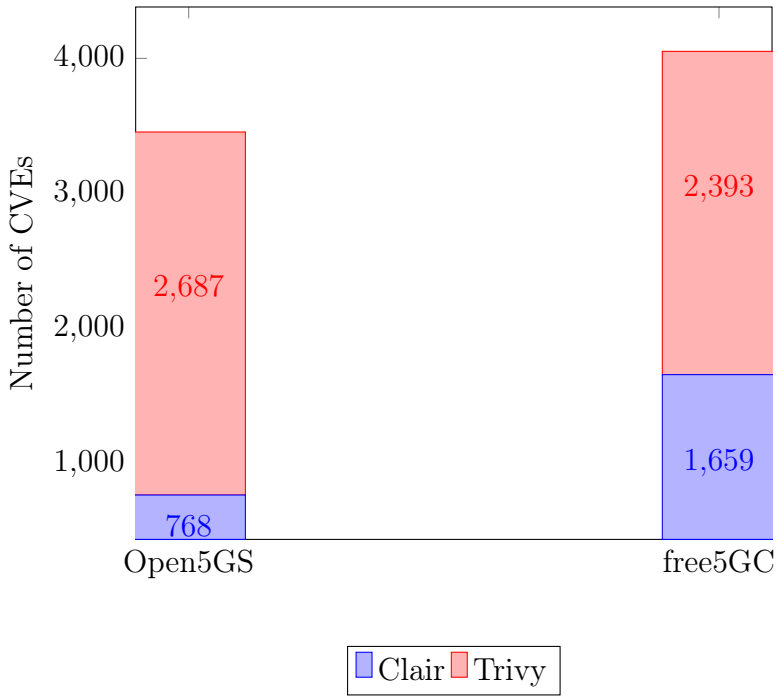


Figure 15: Comparison of CVE counts between Open5GS and free5GC using Clair and Trivy scanners

An attack vector is a path or means by which a hacker (or threat actor) can gain access to a computer or network server in order to deliver a payload or malicious outcome. According to Figure 17, the LOCAL vector has the highest count across both prototypes and both tools, suggesting a significant number of vulnerabilities require local access to exploit. The NETWORK vector follows, indicating that some vulnerabilities could potentially be exploited remotely [23].

| | PHYSICAL | LOCAL | ADJACENT NETWORK | NETWORK |
|---|---|---|---|---|
| free5GC in Clair | 37 | 796 | 68 | 394 |
| free5GC in Trivy | 37 | 1084 | 76 | 658 |
| Open5GS in Clair | 11 | 402 | 24 | 211 |
| Open5GS in Trivy | 38 | 1371 | 50 | 856 |

Table 2: Attack Vector Distribution Across Implementations

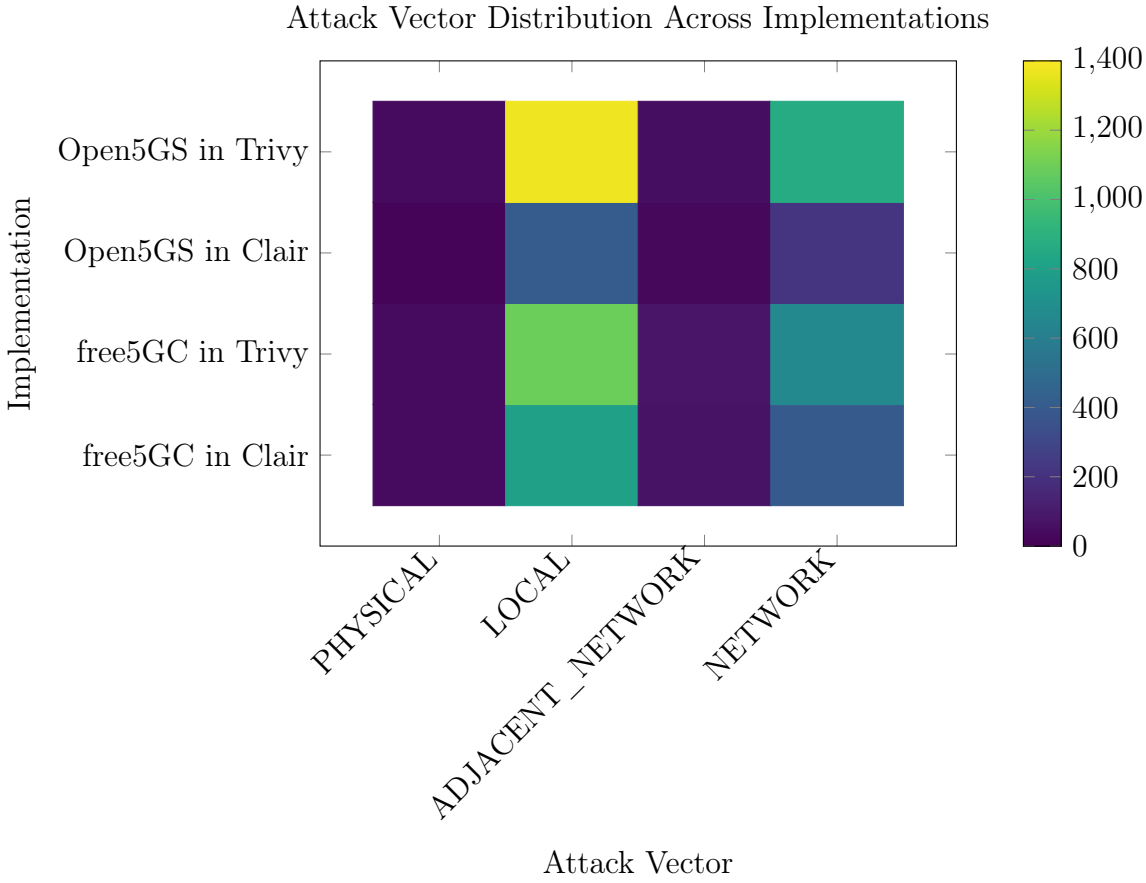


Figure 16: Attack Vector Distribution Across Implementations

Some vulnerabilities detected through these scanning tools have not yet been assigned a CVE identifier or are being rejected [22]. Across both prototypes, the severity levels tend to follow a similar pattern, with the majority of vulnerabilities classified as Medium, followed by High, and then Low and Critical. This pattern suggests that while significant vulnerabilities could impact the system to a moderate or high degree, the proportion of vulnerabilities posing a critical threat, though present, is smaller.

Trivy tends to identify more vulnerabilities across all severity levels than Clair, which is consistent with the earlier observation from the attack vector analysis. This could indicate a more comprehensive scan or a broader definition of what constitutes a vulnerability by Trivy. Open5GS has a higher

count of High and Critical severity vulnerabilities compared to free5GC when analyzed with Trivy, highlighting a potentially greater risk or a need for more stringent security measures and prioritization in remediation efforts.

| | LOW | MEDIUM | HIGH | CRITICAL |
|---|---|---|---|---|
| free5GC in Clair | 71 | 753 | 443 | 28 |
| free5GC in Trivy | 111 | 1105 | 597 | 42 |
| Open5GS in Clair | 25 | 355 | 252 | 16 |
| Open5GS in Trivy | 104 | 1383 | 755 | 72 |

Table 3: Severity Distribution Across Implementations

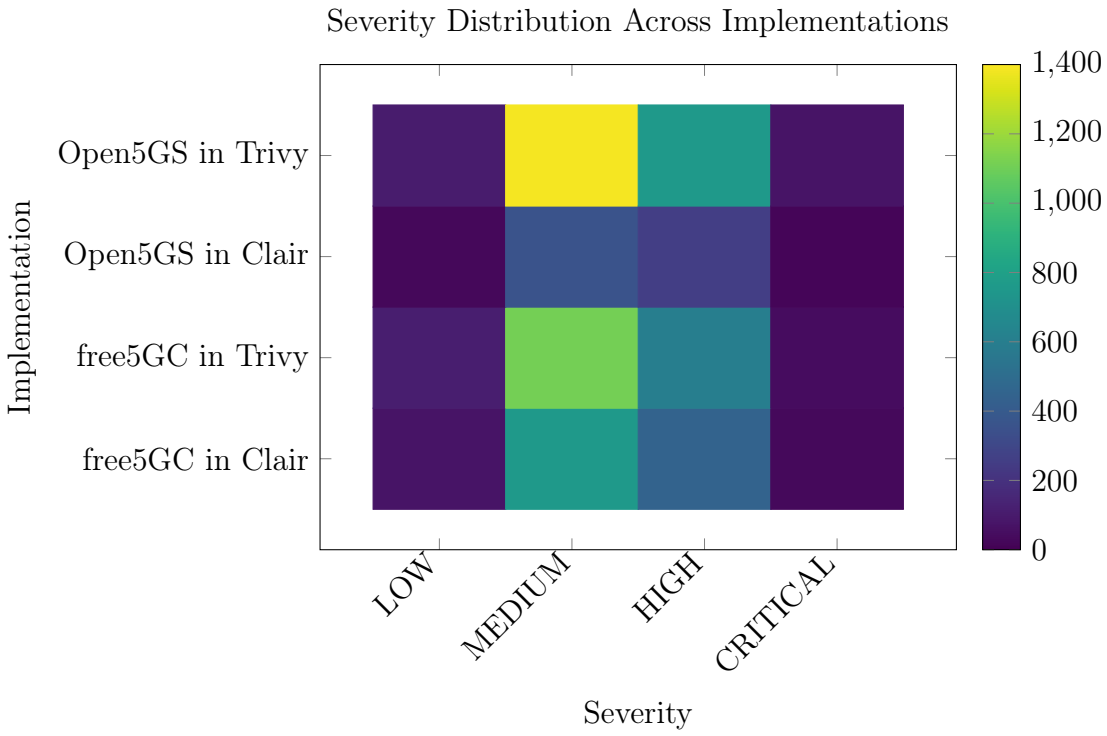


Figure 17: Severity Distribution Across Implementations

# 5 Challenges & Limitations

## 5.1 Performance Aspect

Deploying the 5G core networks and setting up the environment for conducting performance measurements posed several challenges.

## 5.2 Software Dependency Versions

We faced challenges deploying 5G core networks because both core implementations required MongoDB as a database for storing UE subscriber information. The problem arose because MongoDB 5.0 and newer need a CPU with AVX support. If the environment lacks AVX support, MongoDB service crashes without clear error messages or warnings. To solve this, we downgraded to MongoDB version 4.4, compatible only with Ubuntu 20.04, not the latest Ubuntu 22.04. Thus, we had to stick with Ubuntu 20.04. Additionally, Free5GC needed a kernel module named gtp5g, compatible only with the 5.0.0-23-generic or 5.4.x Linux kernel versions. Hence, we had to find a Linux distribution compiled with these specific kernel versions.

## 5.3 Communication Between Core and UERANSIM VMs

In the early stages of our project, we obtained virtual machines (VMs) from the University's infrastructure servers to set up our 5G networks. We planned to deploy the 5G core network and UERANSIM on separate VMs. For communication between these VMs, we used OpenStack, the infrastructure management software. However, we encountered a challenge: OpenStack's default security groups imposed strict limitations on ports and communication protocols. This made it difficult for our VMs to exchange traffic effectively. Enabling the Stream Control Transmission Protocol (SCTP) was particularly important for the 5G core and UERANSIM to communicate. Since the provided OpenStack VMs couldn't meet our needs, we decided to deploy the 5G core networks on VMs hosted on our personal laptops. This gave us the freedom to adjust host and VM configurations as required.

## 5.4 Security Aspect

### 5.4.1 Absence of Recent Vulnerabilities Recorded in NVD

Our first challenge regarding security was the need for up-to-date vulnerability information and CVSS scores, despite the excellent community support of Clair, which provides regular updates on new vulnerability content [1]. For instance, the most recent version of Open5GS is v2.7.0 [13]. However, the most recently identified vulnerability is present in version 2.6.6. Our vulnerability scanning program identifies patterns of vulnerability based on known vulnerabilities. However, in some cases, the tool may recognize a vulnerability that is not published in the database and does not have an associated CVE name or CVSS score. As a result, this vulnerability might not be detected in the scanning report. According to the research conducted by Khanmohammadi and Khoury [9], the average time for the CVSS score to update is 11.62 days. A zero-day vulnerability is a type of security weakness that has not yet been publicly published. The average duration for a zero-day vulnerability to be detected and disclosed in IT systems is roughly 5.3 years [3]. In this scenario, if we solely depend on the public vulnerability scanning tool, which utilizes known vulnerabilities and static analysis, it may not detect the most recent vulnerabilities. Hence, numerous studies in the 5G core (5GC) domain have focused on dynamic scanning techniques, such as fuzzy testing [8], to identify vulnerabilities that are directly associated with the 5GC exclusively.

### 5.4.2 Discrepancies Among Varied Scanning Tools

Based on the section 4.2.1, it can be inferred that even if the scanning tool scans the exact same image, it will produce different reports on vulnerabilities, possibly with a few overlaps.
One possible explanation is that these two vulnerability scanning tools are derived from various vulnerability databases. Due to differences in the extent of coverage and frequency of updates among these databases, there can be discrepancies in the reported vulnerabilities. Clair primarily utilizes the National Vulnerability Database (NVD) to identify known vulnerabilities. However, in the report log of Trivy, we have observed numerous vulnerability IDs, such as "TEMP-0841856-B18BAF," which originates from the

Debian database and has not been assigned a Common Vulnerabilities and Exposures (CVE) name [10].
The other explanation involves to the distinct detection methodologies employed by each scanning tool. Section 2.3 and 2.4 discuss how Clair analyzes the layers of container images and their packages to find vulnerabilities. On the other hand, Trivy examines the filesystem and analyzes package manifests to identify vulnerable software components in container images. In this scenario, Clair can more easily identify vulnerabilities associated with particular package versions. In contrast, Trivy excels at detecting vulnerabilities in individual files or those related to misconfigurations, exposed sensitive information, or insecure file permissions. These types of vulnerabilities may not be detected by Clair's package-focused methodology.

### 5.4.3 Limitations of Static Analyzers

This project focuses on comparing two distinct 5G core prototype implementations. Our primary objective is to identify each implementation's vulnerabilities and comprehensively compare them. The primary purpose of a static analyzer is to identify vulnerabilities that can lead to potential security breaches. These vulnerabilities may include problems related to code quality, security vulnerabilities, or compliance with coding guidelines. These scanners perform their analysis by looking for patterns or signatures in the code that match known vulnerabilities, coding errors, or bad practices. However, the dynamic nature of the 5G network means that many potential threats only become apparent during runtime when interacting with the network, such as with a DoS attack. A static analyzer can identify buffer overflow vulnerabilities, but dynamic analysis is essential to simulate and detect Denial-of-Service (DoS) attacks. This involves creating a controlled environment that mimics real-world network conditions and systematically introducing different attack vectors to observe how each prototype responds to stress.

Upon analyzing our generated CVE reports from Clair and Trivy, we have observed that most of the vulnerability attack vectors are classified as "Local" based on figure 16. This attacker vector indicates that the attacker would typically require physical access to the device or authenticated access, such as through a login session [23]. Open5GS and free5GC have open-sourced specific Common Vulnerabilities and Exposures (CVE). These vulnerabili-

ties are associated with the "Network" attack vector and can be exploited remotely [23]. This attack vector means that the attacker does not need any access to the target system in order to exploit the vulnerability.

We are considering the implementation of a dynamic scanner or analyzer to gain further insight into how a vulnerability could be exploited in real-world scenarios. This tool would simulate attacks and help identify potential paths for exploitation within the system that may not be immediately apparent from static analysis or CVE reports.

# 6 Conclusion and Future Work

## 6.1 Future Work

In this project, we assessed the performance of Free5GC and Open5GS by employing a single User Equipment (UE) to gather metrics from each network. Future studies could expand on this by conducting similar experiments with a larger number of UEs connected to each 5G core network. This would enable researchers to explore whether the results vary and to ascertain which implementation is better suited for delivering service on a larger scale. Moreover, the same experiments can be conducted to compare the performance of other 5G core networks such as SD-Core and Magma.

In terms of security, future analysis and remedial procedures may be developed. One possible way is to create an attack graph, which depicts the potential attack paths that a system may face. Security professionals can acquire valuable insights into potential threats and weaknesses by thoroughly examining the interconnectedness of various system components and detecting potential vulnerabilities. Second, scan with a dynamic scanner or perform dynamic analysis to capture and stress test the 5G network. Furthermore, using machine learning algorithms and artificial intelligence may improve the efficiency and accuracy of vulnerability analysis.

## 6.2 Conclusion

In conclusion, this study evaluated the deployment and analysis of 5G networks, focusing on Open5GS and Free5GC implementations. Our aim was to determine the most robust and efficient 5G Core implementation. Through

comprehensive vulnerability scans and performance assessments, we identified strengths, weaknesses, and vulnerability types in each implementation.

Based on the collected metrics from both Free5GC and Open5GS core networks, it is evident that Free5GC provides better service and higher throughput to connected User Equipments (UEs). Additionally, Free5GC requires less memory and processing resources compared to Open5GS. This difference in performance and resource consumption may be attributed to Free5GC's development in Go language. Go is specifically designed to support concurrent applications, offering built-in features for efficient management of concurrency. Its concurrency primitives and efficient runtime make it well-suited for building microservices architectures. Furthermore, Go's small binary size and low memory footprint contribute to efficient microservices deployment. This could explain why Free5GC demonstrates superior performance compared to Open5GS, which is developed using the C language.

The examination of security reports from two distinct prototype implementations, which were scanned by various scanning technologies, offers an extensive evaluation of the security status of the prototypes, taking into account potential attack vectors and the severity levels of vulnerabilities. It emphasizes the significance of employing multiple vulnerability scanning techniques to detect a diverse array of potential threats. It highlights the importance of addressing detected vulnerabilities in a prioritized manner, taking into account their severity and exploitability.

These findings offer valuable insights for future 5G network deployments and security strategies.

# References


[1] Kyriakos Kritikos et al. "A survey on vulnerability assessment tools and databases for cloud-based web applications". In: *Array* 3 (2019), p. 100011.

[2] FIRST. *Common Vulnerability Scoring System Version 3.1: Specification Document*. `https://www.first.org/cvss/v3.1/specification-document`. 2020.

[3] Richard J Thomas et al. "Catch me if you can: An in-depth study of cve discovery time and inconsistencies for managing risks in critical infrastructures". In: *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*. 2020, pp. 49–60.

[4] Francisco Joaquim De Souza Neto et al. "Analysis for comparison of framework for 5G core implementation". In: *2021 International Conference on Information Science and Communications Technologies (ICISCT)*. IEEE. 2021, pp. 1–5.

[5] William Stallings. *5G Wireless: A Comprehensive Introduction*. 2021.

[6] Saurabh Joshi et al. "5G Deployment and Simulation Using Firecracker for Medical Application". In: *International Conference on Computer Engineering and Networks*. Springer. 2022, pp. 1250–1259.

[7] Taeyun Kim et al. "An implementation study of network data analytic function in 5g". In: *2022 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2022, pp. 1–3.

[8] Hongxin Wang et al. "An automated vulnerability detection method for the 5g rrc protocol based on fuzzing". In: *2022 4th International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*. IEEE. 2022, pp. 1–7.

[9] Kobra Khanmohammadi and Raphael Khoury. "Half-Day Vulnerabilities: A study of the First Days of CVE Entries". In: *arXiv preprint arXiv:2303.07990* (2023).

[10] *Debian Security Tracker*. `https://security-tracker.debian.org/tracker/data/fake-names`. Accessed: April 1, 2024.

[11] *Free5GC*. URL: `https://github.com/free5gc/free5gc`.

[12] National Institute of Standards and Technology (NIST). *National Vulnerability Database*. https://nvd.nist.gov/. Accessed: [insert date here].

[13] *Open5GS*. URL: https://github.com/open5gs.

[14] Linux Foundation Project. *Magma*. URL: https://magmacore.org/.

[15] Quay. *Clair*. https://github.com/quay/clair. [Online; accessed 29-March-2024].

[16] Red Hat. *Red Hat Quay Documentation*. https://access.redhat.com/documentation/en-us/red_hat_quay/3.6/html/manage_red_hat_quay/clair-intro2. Accessed: Insert Date Here.

[17] *SD-Core*. URL: https://opennetworking.org/sd-core/.

[18] *speedtest-cli*. URL: https://github.com/sivel/speedtest-cli.

[19] *UERANSIM*. URL: https://github.com/aligungr/UERANSIM.

[20] Aqua Security. *Trivy Vulnerability Database*. https://github.com/aquasecurity/trivy-db/pkgs/container/trivy-db. Accessed: <insert access date here>.

[21] Armin Coralic. *Clair Scanner*. https://github.com/arminc/clair-scanner. Accessed: Date the website was last accessed. Year the website was last accessed.

[22] CVE. *CVE - Common Vulnerabilities and Exposures FAQs*. [Accessed: April 4, 2024]. CVE - Common Vulnerabilities and Exposures. 2024. URL: https://www.cve.org/ResourcesSupport/FAQs.

[23] FIRST. *Common Vulnerability Scoring System v3.0 Specification Document*. [Accessed: April 4, 2024]. FIRST - Forum of Incident Response and Security Teams. 2015. URL: https://www.first.org/cvss/v3.0/specification-document.

[24] Free5GC Contributors. *Free5GC-Compose: A Docker Compose setup for Free5GC*. https://github.com/free5gc/free5gc-compose. Accessed: <insert access date here>.

[25] Open5GS Contributors. *Open5GS Dockerfiles*. `https://github.com/open5gs/open5gs/tree/main/docker`. Accessed: <insert access date here>.

[26] Aqua Security. *Trivy: A Simple and Comprehensive Vulnerability Scanner for Containers*. `https://github.com/aquasecurity/trivy`. Accessed: <insert date here>.