5GFIVer: Functional Integrity Verification for 5G Cloud-Native Network Functions

A S M Asadujjaman*, Mohammad Ekramul Kabir*, Hinddeep Purohit*,
Suryadipta Majumdar*, Lingyu Wang*, Yosr Jarraya†, Makan Pourzandi†
*CIISE, Concordia University, Montreal, QC, Canada
Email: {a_asaduj, m_kabi, h_puroh, majumdar, wang}@encs.concordia.ca
†Ericsson Security Research, Ericsson Canada, Montreal, QC, Canada
Email: {yosr.jarraya, makan.pourzandi}@ericsson.com

***Abstract*—5G networks attain a better performance along with a reduction in cost by cloudifying its network functions as Cloud-native Network Functions (CNFs). However, CNF may introduce new security concerns (e.g., data exfiltration and ransomware) due to potential code injection attacks against network functions at runtime. This will potentially result in a breach of functional integrity of these network functions. Towards verifying such functional integrity breaches of CNFs at the 5G-operator-level, existing approaches fell short, as most of them either (i) perform pre-deployment verification (i.e., verifying the CNF image before the deployment) and hence fail to verify integrity breaches occurring after the deployment, or (ii) perform post-deployment verification (i.e., verifying against attack signatures or normal behavior patterns) approaches that require provider-level data (e.g., system calls) which is usually inaccessible to 5G operators. In this paper, we propose 5GFIVer, a new operator-oriented approach for functional integrity verification of CNFs that overcomes the above-mentioned limitations. First, our approach utilizes the side-channel information such as performance metrics (which are already available at the operator level) so that no provider-level data is needed. Second, our approach implements unsupervised machine learning algorithms to detect outliers through time-series analysis of those available performance metrics, and hence no instrumentation for the data collection as well as no training data is required. Third, we leverage the correlation between multiple CNFs to improve the accuracy and minimize false positives (e.g., caused by cloud dynamics). Our experimental results under an open source 5G testbed demonstrate the effectiveness and negligible overhead of our solution.**

I. Introduction

The cloudification of network functions (NFs) in 5G networks is recently empowered by the cloud-native technologies (e.g., containers) to ensure high-speed, cost-efficient, and large-scale connectivity [1]. However, running these software as Cloud-Native Network Functions (CNFs), where the NFs share resources (e.g., operating system kernel) with potentially malicious collocated tenants, increase the risk of many threats including code injection attack that might lead to malicious modifications of the in-memory instructions of the NFs (a.k.a. breach of functional integrity) at runtime [2], [3]. For instance, such vulnerabilities have been reported in open source implementations of 5G core network software (e.g., Open5GS [4]). Additionally, some reused libraries and software components have been shown to be vulnerable to code injection (e.g., Log4j [5] and CVE-2022-28391 [6]). For example, our scanning of the Open5GS [4] project using Trivy [7], an open-source security scanner, reveals numerous bugs (1,309 vulnerabilities, where 41 of those vulnerabilities are critical and 317 vulnerabilities are highly severe). Considering the vital role of 5G in various cyber critical infrastructure and their security, tackling malicious code injection and verifying functional integrity of NFs are very important for the security of 5G networks.

978-1-5386-4633-5/18/$31.00 ©2021 IEEE

However, two major approaches (i.e., pre-deployment and post-deployment) to verifying functional integrity are usually insufficient in the context of CNFs in 5G mainly due to the following reasons: i) *Pre-deployment:* The verification approaches (e.g., [8], [9]) that are conducted before the deployment of the network functions in the cloud cannot detect the integrity breaches that are caused by the malicious code injected at runtime (i.e., after the deployment). (ii) *Post-deployment:* The verification approaches (e.g., [10], [11]) that are conducted after the deployment of the network functions require infrastructure-level data (which is usually inaccessible to 5G operator [12]). On the other hand, relying on an infrastructure-level runtime solution (e.g., using system-call interception) to verify such functional integrity may add significant latency [13], [14] to the performance sensitive of 5G applications. We further illustrate these limitations and our main ideas through a motivating example as follows.

A. Motivating Example

The top of Fig. 1 illustrates a practical scenario of functional integrity breaches in a possible 5G core implementation, and the bottom depicts the limitations of existing verification solutions.

Functional integrity breach. By exploiting the existing vulnerabilities, e.g., Open5GS PFCP [4], an attacker, e.g., collocated tenant ([2], [3]) can inject code into some NFs. Particularly, as shown in the figure, the attacker injects malicious code into the in-memory NF, *Access and Mobility Management Function* (`AMF`), to modify its functionality.

Limitations of existing solutions. The existing verification approaches could be divided into two major categories: 1) the bottom left illustrates the pre-deployment approaches and 2)

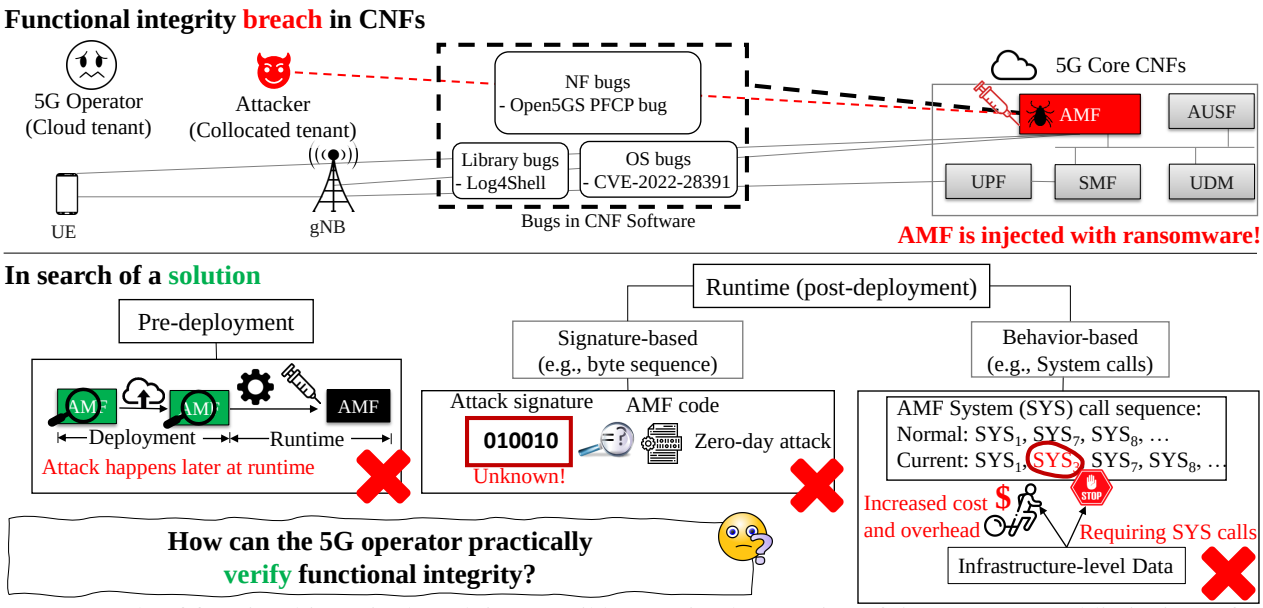


Fig. 1: An example of functional integrity breach in a possible CNF implementation of 5G core (top) and limitations of existing solutions (bottom)

the bottom right depicts the post-deployment approaches. A pre-deployment approach verifies the integrity of AMF before deployment either at the operator level or at the cloud provider level. Therefore, such an approach cannot detect this integrity breach where AMF is modified at the runtime. On the other hand, a post-deployment approach performs either signature-based or behavior-based verification. Signature-based verification (e.g., [15]) relies on the attack signature of the malicious code snippet and checks the binary code of AMF to find a match. On the other hand, behavior-based verification matches the current system call sequence [16] against the normal sequence of system calls to identify that SYS_3 is a mismatch and causing the integrity breach in AMF. However, this verification method requires access to the infrastructure-level data which makes the approach costly (due to the instrumentation) and inefficient (due to the frequent system-level interception). Moreover, the adoption of CNF limits the capacity of 5G operators in accessing provider-level data [12], and hence these post-deployment approaches also become infeasible for the 5G operator.

B. Main Idea

To overcome those limitations in the existing solutions, our main ideas (Fig. 2) are as follows.

Idea 1: Performance metric as a side-channel. As depicted in the middle of Fig. 2, our first idea is inspired by the fact that any change in software functionality (i.e., caused by injected code) may affect the resource (e.g., CPU, memory, power, etc.) consumption [17] in a distinctive way. Particularly, our idea is to perform a time-series analysis on the available performance metrics (e.g., CPU and memory utilization of each container) to identify outliers. The key advantages of this idea are: 1) our approach makes the decision based on the available performance metrics (e.g., CPU and memory utilization of AMF, UDM, AUSF, SMF as shown in Fig. 2), and hence, does not need any access to the infrastructure level data, 2) these performance metrics are made available by public cloud providers (e.g., Amazon AWS, Microsoft Azure, IBM cloud, etc.) for the billing purpose [18], and hence, our proposed approach does not require any instrumentation (i.e., no modification of 5G core NF) to collect data, and 3) the implementation of time-series analysis in detecting outliers provides us the opportunity to detect breaches at runtime. A key challenge here is that, apart from the attacks mentioned earlier, the dynamic behavior of the cloud infrastructure (e.g., workload variation, infrastructural changes, etc.) may also affect the CPU and memory consumption. Therefore, relying only on *Idea 1* would result in false-positive decisions; which motivates us to propose our second idea to address such concerns.

Idea 2: Multi-CNF correlation. Our second idea is to correlate the findings of time-series analysis of the containers whose resource consumptions are correlated with each other (detailed in Section II-B) in order to verify the decisions made by *Idea 1*. The containers of the 5G core NFs are considered correlated with each other when they tend to respond in a similar way to any legitimate changes in their performance metrics (e.g., due to underlying infrastructure change or workload fluctuation). Therefore, the first step of *Idea 2* is to identify the correlated containers and then to implement a time-series analysis for each container to identify respective outliers. If an outlier is identified for a particular container, we correlate this finding with its correlated containers which are expected to show similar outliers, while for the attack, only the infected container may show an outlier. Consequently, *Idea 2* can potentially help to reduce false positives while keeping all advantages of *Idea 1*. We will elaborate on these ideas in Section III.

The main contributions of this work are:

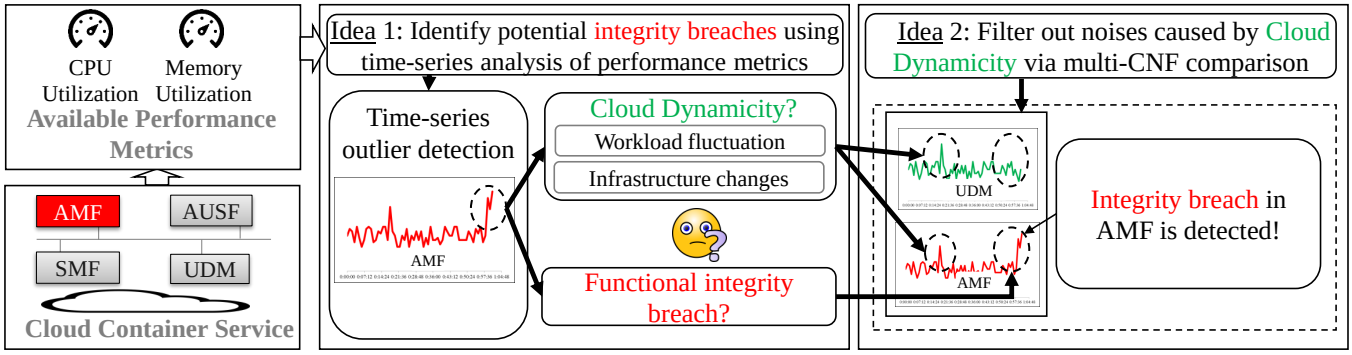


Fig. 2: The main ideas of 5GFIVer

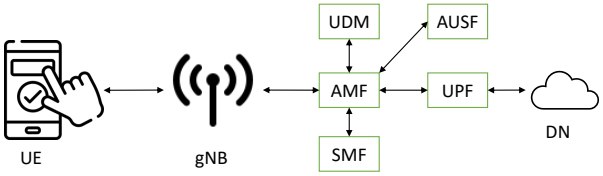


Fig. 3: An excerpt of 5G topology

- We propose an operator-oriented black-box approach to verify functional integrity in 5G core implementation without relying on the provider-level data.
- To improve the accuracy and minimize the false positives, we implement time-series analysis on the available performance metrics (e.g., memory and CPU consumption by each container) to detect outliers and verify whether those are due to attacks or not, through the correlation between containers.
- Our proposed approach does not require any instrumentation to collect required data, i.e., no change is required to the 5G core during the data collection process. This makes our solution more deployable and practical.
- We implement our solution and integrate it with Open5GS [19], a popular open-source 5G core implementation, under our testbed, and our experimental evaluation shows the robustness of our solution.

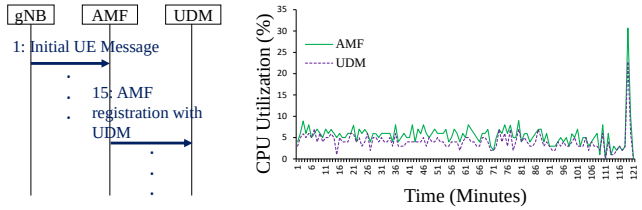
The rest of the paper is organized as follows. Section II provides the background on 5G NFs and defines our threat model. Section III describes the proposed solution. Section IV presents our implementation details along with the experimental results, and Section V describes the literature review. Finally, Section VI concludes the work.

II. Preliminaries

This section discusses 5G core cloudification, describes the correlation among the containers, and then defines our threat model and assumptions.

A. Cloudification of 5G Core

5G core is an essential component of 5G networks that orchestrates various services such as authentication, authorization, session management, routing and switching, data and policy management, and maintaining connectivity with the client. These functionalities are completely virtualized on the cloud deployment and called CNFs. As shown in Fig. 3, the



(a) An example of AMF call flow in 5G [20]

(b) CPU utilization of AMF and UDM

Fig. 4: Example of a metric (CPU utilization), that shows to be highly correlated for AMF and UDM.

5G core is composed of a plethora of CNFs including but not limited to Access and Mobility Management Function (AMF), User Plane Function (UPF), Session Management Function (SMF), and Unified Data Management (UDM). All of these functions serve diverse roles, say, AMF is responsible for letting user equipment (UE), e.g., mobile phones, connect to the core network. 5G core can function only when these CNFs interact. As a result of this interaction, as shown in Fig. 4a, a new task during in one CNF (e.g., processing *Initial UE Message* in AMF) also leads to a new task for another CNF (e.g., *AMF registration* for UDM). Since certain CNFs share the related responsibility, it is likely that their performance metrics may also be correlated (discussed in Section II-B).

B. The Correlation among 5G CNFs

Our experiments indicate that performance metrics (e.g., CPU and memory utilization) of certain CNFs (e.g., AMF) show a strong correlation with other specific CNFs (e.g., UDM). For example, as shown in Fig. 4b, CPU utilization of AMF (shown in solid green) shows a similar pattern as CPU utilization of UDM (shown in dashed purple).

Another way to interpret the above phenomenon is that any "unusual" behavior shown in the performance metrics due to the cloud dynamicity (e.g., change in workload or underlying infrastructure) is likely to affect multiple CNFs instead of only one (i.e., it will affect all the CNFs that are correlated). We can leverage this observation to search for evidence of cloud dynamicity and thus filter them out to avoid false positives. Although the correlation in Fig. 4b is easy to visually identify in this particular case, it may not be the case in the presence of many CNFs and a large volume of data. To this end, we use

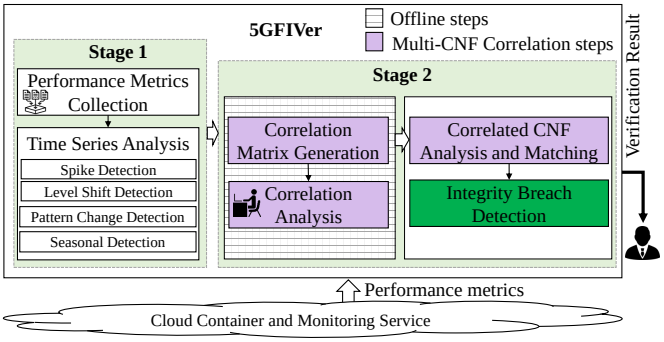


Fig. 5: A high-level overview of 5GFIVer

correlation analysis to identify highly-correlated CNFs, which will be detailed in Section III.

C. Threat Model and Assumptions

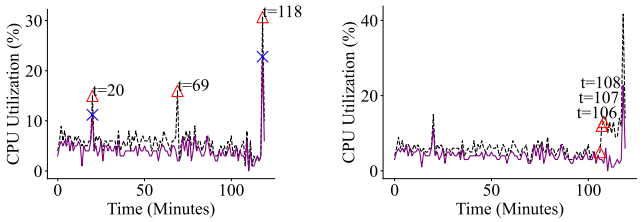
This work considers integrity breaches of containerized network functions (CNFs) caused by a code injected into the CNF by exploiting vulnerabilities [21], [22] in: (i) CNF software (e.g., Open5GS PFCP bug [4]), (ii) libraries used by the CNF (e.g., Log4j [5]), or (iii) underlying host operating systems(e.g., CVE-2022-28391 [6]). This code injection happens at runtime in-memory after the CNF is loaded from its image and does not modify the latter. Once compromised, the CNF cannot be trusted any longer (e.g., for collecting logs via SSH) because it may be under the control of the attacker. In addition, the code injection may be either (i) *transient:* when the code modification is not permanent and only lasts for the duration of the processing of the current packet/request, or (ii) *permanent:* when the code modification is permanent and lasts even after the packet/request is completed/processed. We assume that as the vulnerabilities in each CNF are unique, it will be difficult for the attacker to compromise all the CNFs that are correlated with each other. We do not assume that the attack signature is known (i.e., possibly for a zero-day attack). Also, we do not assume that the tenant has access to the underlying infrastructure to collect information (e.g., logs) apart from what is commonly available to users of cloud container services. On the other hand, the out-of-scope threats for this paper include verifying integrity breaches due to those attacks that: i) compromise all CNFs, ii) similarly affect the correlated CNFs, and iii) do not have much impact on CPU/memory consumption, which will be considered in our future work.

III. Methodology

This section presents the methodology of 5GFIVer in detail.

A. Overview

Fig. 5 shows an overview of 5GFIVer, which contains two stages. First, in *Stage 1*, 5GFIVer performs a time series analysis of the performance metrics (e.g., CPU, memory, etc.) to detect the outliers as an indication of potential attacks. Specifically, in this stage, 5GFIVer first collects and processes the available performance metrics for each container. Then, it deploys an unsupervised learning technique (e.g., level shift detection [23]) to perform a time series analysis of the



(a) Transient code injection (b) Permanent code injection

Fig. 6: 5GFIVer time series analysis; AMF (- - -), UDM (——), analysis of AMF (△), analysis of UDM (×)

collected data to detect outliers for each container (e.g., AMF). In *Stage 2*, 5GFIVer filters out the outliers caused by the cloud dynamicity to identify integrity breaches. Specifically, in this stage, 5GFIVer eliminates false-positive decisions made in *Stage 1* by using the correlated behavior among multiple CNFs (e.g., UDM is correlated with AMF) and then identifies integrity breaches accordingly. We detail each stage as follows.

B. Stage 1: Outliers Detection

This first stage of 5GFIVer performs: (Step 1.1) performance metrics collection, and (Step 1.2) time series analysis.

Step 1.1: Performance metrics collection. This step is to collect available performance metrics for the CNFs to be analyzed for detecting potential integrity breaches. According to our investigation, different performance metrics (e.g., CPU or memory utilization) from different public cloud container services are available to the 5G operator [18]. Our experiments with those available performance metrics show that these metrics are affected by the code injection attack, and hence can be utilized in our solution to detect such breaches. For instance, we collect performance metrics (e.g., CPU utilization) periodically (e.g., at every minute) during the operation of 5GFIVer as shown in Fig. 6. Hence, the outcome of this step can be represented as a stream of timestamp-metric pairs for time t_1 to t_n as follows: $t_1 \rightarrow \mathrm{M}_{t_1}$, $t_2 \rightarrow \mathrm{M}_{t_2}$, $t_3 \rightarrow \mathrm{M}_{t_3}$, ..., $t_n \rightarrow \mathrm{M}_{t_n}$.

Step 1.2: Time series analysis. This step is to identify outliers in the stream of timestamp-metric pairs collected in *Step 1.1*. Fig. 5 lists a number of possible types of time series outlier detection algorithms. In this paper, we demonstrate two of them to detect two types of outliers: 1) detecting spikes caused by the transient code injection and 2) detecting level shift caused by the permanent code injection.

The spike (a.k.a. additive outlier [24]) is a sudden change in the performance metrics for a short span of time. This can be detected by considering three consecutive disjoint sliding windows and tracking the difference in the median value between the short central window and the other two outer windows [24]. For instance, Fig. 6 (a) depicts the spikes as outliers in CPU consumption of AMF at $t = 20$, $t = 69$, and $t = 118$. Hence, the outcome of this spike detector for each CNF is a set of timestamps: $T_S(\text{CNF}) = \{t_{S_1}, t_{S_2}, t_{S_3}, ..., t_{S_n}\}$, where t_{S_n} indicates the time for a spike S_n.

On the other hand, the level shift [23] indicates a shift in the level of the performance metrics that might occur due to a permanent code injection. To determine this level shift, we leverage the LevelShiftAD detector from the anomaly detection toolkit by Arundo [25], which detects a shift in the metrics value level by considering the difference in median value between two adjacent sliding windows. It needs to be mentioned that LevelShiftAD is not spike sensitive [25], and hence any spike generated by transient injection is not detected by this detector. Fig. 6 (b) shows that a level shift occurs at $t = 106$ in the CPU consumption of AMF, while no shift is found for UDM. The outcome of the level shift detector is another set of timestamps: $T_L(\text{CNF}) = \{t_{L_1}, t_{L_2}, t_{L_3}, ..., t_{L_n}\}$, where t_{L_n} indicates the time for a level shift L_n.

However, the key challenge in outlier detection is that these outliers (i.e., spike or level shift) can occur either due to a code injection, or a caused by legitimate changes in the cloud infrastructure. Hence, we verify the detected outliers in *Stage 2* to eliminate false positive decisions.

C. Stage 2: Integrity Breach Detection

The second stage performs: (Step 2.1) multi-CNF correlation, and (Step 2.2) integrity breach detection.

Step 2.1: Multi-CNF correlation. This step (marked in light purple background in Fig. 5) is to identify outliers (both spikes and level shifts) caused by cloud dynamics but detected by *Stage 1*. The inputs to this step are a CNF under verification (e.g., CNF1), its correlated CNF (e.g., CNF2), and the sets of timestamps for each of these CNFs from *Stage 1*. This step consists of (i) conducting an offline process to find which CNF metrics are correlated with each other, and (ii) conducting an online process to identify outliers caused by the cloud dynamicity.

The offline process generates a correlation matrix, i.e., the matrix containing the similarity in resource consumption patterns of different containers (e.g., AMF, UDM, AUSF, etc.). For instance, Fig. 6 shows that the CPU consumption, respectively, of AMF and UDM have similarities (evaluated in Fig. 7a) and hence, these two CNFs are correlated with each other. Hence, the outcome of this offline process is a matrix containing the information regarding the correlated CNFs (e.g., AMF is correlated with UDM or SMF is with UPF) which will be available to the online process. This outcome enables the next steps, for any CNF under verification (e.g., *CNF1*), to know its correlated CNF (e.g., *CNF2*).

The online process performs correlation to identify the common ones among the detected outliers (in *Stage 1*) of the correlated CNFs (i.e., *CNF1* and *CNF2*). The outcome of this step is the number of outliers that are in common in *CNF2* and in *CNF1*, for each type. As these CNFs have correlated metrics, the outliers in common can be related to cloud dynamicity issues as discussed in Section II-B. Thus, the number outliers in common of type spike, denoted by N_{C_S}, and the number outliers in common of type level shift, denoted by N_{C_L}, can be computed as follows:

$$N_{C_S} = n(\{T_S(\text{CNF1}) \cap T_S(\text{CNF2})\}) \quad (1)$$

$$N_{C_L} = n(\{T_L(\text{CNF1}) \cap T_L(\text{CNF2})\}) \quad (2)$$

where $n(T)$ is the number of elements in a set T. For instance, in Fig. 6a (it is noteworthy that 6a and 6b are from different experiments and we do not show the corresponding spike detection for 6b due to lack of space), three spikes are detected in CPU consumption of AMF (i.e., *CNF1*) at $t = 20$, $t = 69$, and $t = 118$, while two spikes are detected for UDM (i.e., *CNF2*) at matched timestamps (i.e., at $t = 20$, and $t = 118$). The online process computes that $N_{C_S} = 2$. Similarly, as no level shift is detected in UDM in Fig. 6b, it computes $N_{C_L} = 0$.

Step 2.2: Integrity breach detection. Using the findings from the previous step, this step identifies integrity breaches and classifies them (i.e., transient or permanent).

The input to this step are the same pair of CNFs as in Step 2.1, the number of their respective identified outliers, and the number of outliers common between them. Let N_{CNF1} be the total number of identified outliers (in *Stage 1*) for *CNF1* computed using the following equation,

$$N_{CNF1} = N_{CNF1_S} + N_{CNF1_L} \quad (3)$$

Here, N_{CNF1_S} is the number of outliers detected by the spike detector, and N_{CNF1_L} is the number of outliers detected by the level shift detector in CPU utilization of CNF1 (e.g., AMF). Now a positive value of N_{CNF1} indicates a potential integrity breach, whereas, if N_{CNF1} is zero, it indicates that there are no integrity breaches. If $N_{CNF1} > 0$, we compute the total number of correlated outliers N_C as follows,

$$N_C = N_{C_S} + N_{C_L} \quad (4)$$

Then we define a parameter Δ as follows.

$$\Delta = N_{CNF1} - N_C \quad (5)$$

It should be noted that, according to our assumptions in Section II-C, $N_{CNF1} \geq N_C$ and $N_C \geq 0$. At this point, a positive value of Δ indicates that there is integrity breaches in *CNF1*, and a zero value of Δ indicates that there is no integrity breach.

To further classify the type of code injection (i.e., transient or permanent), the value of spike or level shift can be consulted. Specifically, if $\Delta > 0$, the code is injected in *CNF1*. Now to classify the injected code, N_{CNF1_S} and N_{CNF1_L} can be utilized. Since transient and permanent code injections are exclusive in our threat model, a positive value of N_{CNF1_S} indicates transient code injection. On the other hand, if N_{CNF1_L} is positive, it indicates permanent code injection.

$$\text{Code Injection} = \begin{cases} \text{Transient}, & \text{if } \text{N}_{CNF1_S} > 0 \\ \text{Permanent}, & \text{if } \text{N}_{CNF1_L} > 0 \end{cases}$$

For instance, from Fig. 6b, we can see that the total number (both spikes and level shifts) of outliers in CNF1 (i.e., AMF) is $N_{CNF1} = 3$, while for the correlated CNF2 (i.e., UDM), $N_C = 0$. Hence, we have a value of $\Delta = 3$ and this indicates that there is an integrity breach in AMF. On the other hand, since $N_{CNF1_L} = 3 > 0$, the breach is caused by a permanent injection in AMF.

IV. Implementation and Evaluation

This section presents the implementation details and experimental evaluation of 5GFIVer.

A. Implementation

We implement 5GFIVer as a Linux service using systemd [26] running on a virtual machine (Ubuntu 20.04 server). We use a VM-based deployment as it is easier to port to any server. However, 5GFIVer can also be deployed on any other platform (e.g., baremetal or containers) following a similar architecture as described in this section. 5GFIVer should continuously run where it is deployed (e.g., container, VM, baremetal) to perform its verification. It uses different open-source libraries (e.g., Python library ADTK [27] is used to implement the Level Shift Detector) as well as scripts developed by us in Python. The performance metrics collector invokes a script to collect performance metrics from the underlying cloud service which is specific to the monitoring service (e.g., Amazon CloudWatch, Google Cloud Metrics, Azure Monitor, or Prometheus) and prepares input for the time-series outlier detector in it's required format. Communication between different components and modules is done using a database. We use MongoDB [28], an open-source NoSQL database engine, to implement the database.

We extended Anomaly Detection Toolkit (ADTK) [27] to develop our spike detector, called SpikeAD, which detects spikes by tracking the difference between median values at the central and the other two outer windows of three sliding time windows next to each other. Thus, SpikeAD can detect spikes while ignoring level shifts.

B. Experimental Settings

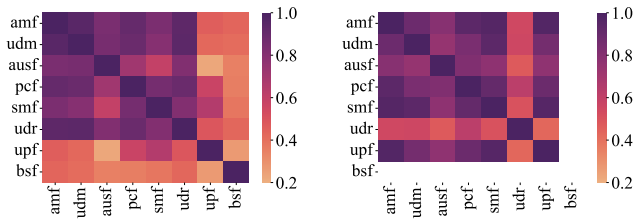
We adopted Open5GS-2.4.8 [19] to create images of the CNFs for the 5G core. We used UERANSIM [29] to emulate the Radio Access Network (RAN) and user equipment (UE). We emulated up to 10000 UEs in our experiments. We limited the allowed CPU cores (between 1 and 8) and memory (512 MB) for each CNF.

To investigate the behavior of CNFs in terms of consuming the resources from the cloud, we deployed containers on Amazon Elastic Container Services (ECS). We then collected performance metrics for our deployed containers from Amazon CloudWatch metric monitoring service.

C. Experimental Results

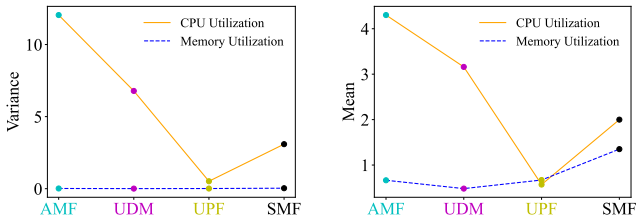
This section presents the experimental results to evaluate the effectiveness of 5GFIVer.

Metrics selection and Multi-CNF correlation. These sets of experiments identify the correlated behavior among different CNFs and the effectiveness of performance metrics in using to detect code injection attacks. The heatmaps in Fig. 7 demonstrate the correlation among multiple CNFs in terms of consuming CPU and memory. Fig. 7a) depicts that CPU utilization of some CNFs is highly correlated (the darker the shade, the higher the co-relation) with that of some other CNFs



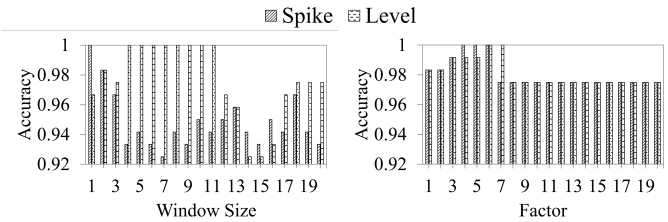
(a) Correlation (CPU utilizations) (b) Correlaton (Memory utilizations)

Fig. 7: Heatmaps showing correlation in CPU and memory utilizations of different CNFs (blank cells mean no correlation)



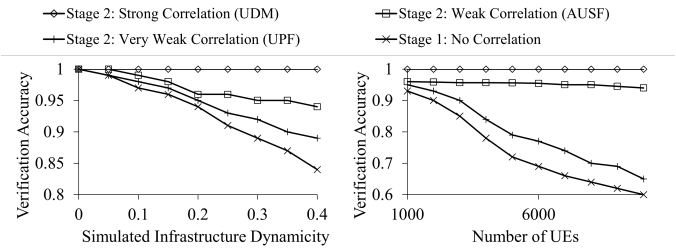
(a) Variance (b) Mean

Fig. 8: The variance and mean of CPU and memory utilizations in different CNFs



(a) Impact of window sizes on accuracy (b) Impact of factors on accuracy

Fig. 9: Accuracy of time series algorithms for different hyper-parameter value settings



(a) Imapct of infrastructure dynamicity (b) Impact of workload variation

Fig. 10: Impact of cloud dynamicity on the accuracy of verification on AMF for different correlated CNFs; Stage 1 (without multi-CNF correlaton), Stage 2 (with multi-CNF correlation)

(e.g., AMF and UDM are correlated with each other), while, Fig. 7b) shows the correlation for Memory utilization. Hence, to verify the outliers (i.e., due to attack or cloud dynamicity) found in AMF, we can utilize its correlation with UDM.

On the other hand, Fig. 8 shows the variance and mean of CPU and Memory utilization for different CNFs. Although the variance (Fig. 8a) of the CPU utilization is quite high for the different CNFs, the variance of memory utilization is very low (indicating a lack of useful information [30]). Hence, to attain a higher entropy, we select CPU utilization throughout

TABLE I: Performance profile of 5GFIVer on a lightweight Amazon EC2 virtual machine (VM) of type t2.medium (i.e., two vCPUs and 4 GB memory)

Performance measure	Average	Min	Max
Execution time for each iteration	1.2s	0.7s	2s
CPU utilization during each iteration	31%	11.0%	52%
Memory utilization during each iteration	1.1%	0.7%	4.0%

the rest of our experiments. However, memory utilization of other performance metrics can also be used for this purpose.

Accuracy of detectors. The detection accuracy is highly dependent on the adopted detection algorithms and their parameter selection. Fig. 9 shows the effectiveness of detecting outliers using time series analysis algorithms (as mentioned in Section III-B) for different hyper-parameters (i.e., *window size* and *factor* value) [25]. As shown in Fig. 9a, the level shift detector (i.e., LevelShiftAD) is most effective when *window size* is between 4 and 11, as indicated by accuracy of 1.0. Similarly, Fig. 9a, and Fig. 9b show effectiveness of detection when the hyper-parameters of the detection algorithms are configured correctly. Hence, by tuning these parameters, we can achieve higher accuracy in outliers detection.

Accuracy of verification. We investigate the impact of cloud dynamicity (i.e., changes in the infrastructure (Fig. 10a) or workload variation (Fig. 10b) in our verification accuracy and highlight the importance of the *Stage 2* verification. To that end, we simulate an increasing level of cloud dynamicity by incrementing the number (by 0% to 40%) of occurrences of fluctuations in the performance metrics of the CNFs caused by cloud dynamicity. On the other hand, we vary the workload dynamicity by increasing the number of UEs (from 1k to 10k). Finally, we intend to detect the breaches in AMF for four different scenarios: 1) Implement *Stage 1* only (i.e., no verification from *Stage 2*), 2) Consider UDM (i.e., strongly correlated with AMF) as the correlated CNF in *Stage 2*, 3) Consider AUSF (i.e., weakly correlated with AMF) as the correlated CNF in *Stage 2*, and 4) Consider UPF (i.e., very weakly correlated with AMF) as the correlated CNF in *Stage 2*.

Fig. 10a illustrates that the accuracy of *Stage 1* (i.e., without verifying by *Stage 2*) decreases with an increased value of dynamicity, while this decreasing trend is almost linear with the increased workload. Then to improve the performance (i.e., eliminate false positive decisions due to cloud dynamicity/-workload), we implement *Stage 2* for three correlated CNFs and find that the accuracy can be regained high when the considered CNF (i.e., UDM) has a strong correlation with the AMF. On the other hand, considering UPF as the correlated CNF cannot make any significant improvement due to its very weak correlation with AMF.

Overhead evaluation. We examine the added overhead by 5GFIVer to evaluate its effectiveness. We listed the average value along with the observed maximum and minimum values of required time, CPU, and memory consumption in Table I, which shows that 5GFIVer is very fast to detect the breaches by adding a negligible amount of CPU and memory overhead.

V. Related Work

In search of a solution to detect functional integrity breaches in virtualized 5G core, we investigated existing works close to our problem area. To that end, we looked at the area of 5G security, malware detection, network virtualization, and performance metrics correlation.

Functional integrity breach in 5G. Existing works in 5G security can be categorized by the related key technologies as follows: 1) Software Defined Networking (SDN), 2) virtualized Network Functions (NFs), 3) Network Slicing and 4) Mobile Edge Computing (MEC). Among these, works that discuss security issues related to virtualized NFs are the closest to our research. Security issues in 5G related to virtualized NFs. Among the works in this area, authors report the possibility of 1) code injection attack [2], 2) migration of NFs to a less secure host to compromise it subsequently [3], 3) propagation of malicious code to a co-hosted NF [31], however, they do not provide any mechanism to perform integrity verification against these attacks.

Malware detection. In the literature, extensive work has been done on detecting malware in applications [16]. However, existing malware detection works cannot be applied in our case for the following reasons: (i) we classify works that statically extract features from binaries for detection [8], [9] as pre-deployment. However, these solutions might not be applicable against our threat model, especially while an attack will happen after the binaries (i.e., CNF images) are deployed and run, (ii) on the other hand, a few works that perform during runtime by using a signature-based approach cannot detect zero-day attacks [32], and (iii) works [10], [11] that attempt to detect zero-day attacks but use system features (e.g., system calls) usually cannot be applied due to challenges in collecting those features (e.g., lack of access to the underlying host operating system by a tenant) and even when those features can be collected, collecting them would require significant cost in terms of overhead (considering the high throughput requirement of 5G) due to interception and need for instrumentation of each CNF [33].

Network virtualization auditing. Most existing works in the area of network virtualization mainly focus on verifying the integrity of virtualized networks at a higher level of abstraction (e.g., deployment according to given specification [34], [35], correct packet forwarding [36]–[38]). The closest work we found in terms of objective is [39]. However, [39] is very specific to security function outsourcing and cannot be extended to our scenario of 5G core NFs.

Performance metrics correlation. In [40], the correlation between different metrics has been studied, while [41] studied the correlation between performance metrics from different cells. But, none of the above works studied the correlation among metrics of different network functions (multi-network function metric correlation) and as a ramification, cannot be adapted to serve our purpose.

VI. Conclusion

This paper proposed an operator-oriented, lightweight side-channel-based black-box approach, namely, 5GFIVer, to verify the functional integrity of CNFs without relying on any underlying cloud infrastructure data. To that end, 5GFIVer first analyzed performance metrics available to the 5G operators to detect outliers that may contain many false positives due to the dynamic behavior of clouds or noises. To filter out the false positives, 5GFIVer then verified service chain integrity by correlating the outliers with the outliers found in other correlated CNFs. We integrated 5GFIVer with a popular open-source 5G core implementation (Open5GS [19]), under our testbed and the experimental results showed that our approach can effectively verify functional integrity by adding a negligible overhead. In future, we plan to extend our approach for performing verification when the integrity of multiple CNFs can be breached, ensemble the findings from multiple detection algorithms to attain more accuracy and further optimize other parameters of 5GFIVer. Furthermore, we plan to perform extensive security analysis and more experimental evaluations of 5GFIVer to show its effectiveness in other 5G core implementations in the future.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair in SDN/NFV Security and the Canada Foundation for Innovation under JELF Project 38599.

References

[1] *Cloud native is transforming the telecom industry*. [Online]. Available: https://www.ericsson.com/en/cloud-native

[2] A. Aljuhani and T. Alharbi, "Virtualized network functions security attacks and vulnerabilities," in *IEEE CCWC*, 2017, pp. 1–4.

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

[4] "Open5GS PFCP bug." [Online]. Available: https://research.nccgroup.com/2021/10/06/technical-advisory-open5gs-stack-buffer-overflow-during-pfcp-session-establishment-on-upf-cve-2021-41794/

[5] *Exploiting, Mitigating, and Detecting CVE-2021-44228: Log4j Remote Code Execution (RCE)*. [Online]. Available: https://sysdig.com/blog/exploit-detect-mitigate-log4j-cve/

[6] *CVE-2022-28391 Detail*. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2022-28391

[7] *Trivy*. [Online]. Available: https://aquasecurity.github.io/trivy/dev/

[8] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *IEEE S&P*, 2000.

[9] K. Hahn and I. Register, "Robust static analysis of portable executable malware," *HTWK Leipzig*, vol. 134, 2014.

[10] G. Wagener, A. Dulaunoy *et al.*, "Malware behaviour analysis," *Journal in computer virology*, vol. 4, no. 4, pp. 279–287, 2008.

[11] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE TIFS*, vol. 11, no. 2, pp. 289–302, 2015.

[12] A. Asadujjaman, M. Oqaily, Y. Jarraya, S. Majumdar, M. Pourzandi, L. Wang, and M. Debbabi, "Artificial packet-pair dispersion (APPD): A blackbox approach to verifying the integrity of NFV service chains," in *2021 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2021, pp. 245–253.

[13] M. Gebai and M. R. Dagenais, "Survey and analysis of kernel and userspace tracers on linux: Design, implementation, and overhead," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–33, 2018.

[14] A. Darki, A. Duff, Z. Qian, G. Naik, S. Mancoridis, and M. Faloutsos, "Don't trust your router: Detecting compromised router," in *IEEE CoNEXT*, vol. 16, 2016.

[15] E. S. Parildi, D. Hatzinakos, and Y. Lawryshyn, "Deep learning-aided runtime opcode-based windows malware detection," *Neural Computing and Applications*, vol. 33, no. 18, pp. 11 963–11 983, 2021.

[16] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[17] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *IEEE INFOCOM*, 2017, pp. 1–9.

[18] *Amazon CloudWatch Pricing*. [Online]. Available: https://aws.amazon.com/cloudwatch/pricing/

[19] *Open5GS*. [Online]. Available: https://open5gs.org/open5gs/

[20] *ETSI TS 123 502: Procedures for the 5G System*. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.02.00_60/ts_123502v150200p.pdf

[21] *Code Injection*. [Online]. Available: https://owasp.org/www-community/attacks/Code_Injection

[22] *A Complete Guide to Cloud-Native Application Security*. [Online]. Available: https://www.trendmicro.com/en_no/devops/21/k/a-complete-guide-to-cloud-native-application-security.html

[23] H. Dehling, R. Fried, and M. Wendler, "A robust method for shift detection in time series," *Biometrika*, vol. 107, no. 3, pp. 647–660, 2020.

[24] *Outliers Detection and Intervention Analysis*. [Online]. Available: https://datascienceplus.com/outliers-detection-and-intervention-analysis/

[25] *LevelShiftAD*. [Online]. Available: https://arundo-adtk.readthedocs-hosted.com/en/stable/notebooks/demo.html#LevelShiftAD

[26] *Systemd*. [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/

[27] *Anomaly Detection Toolkit*. [Online]. Available: https://adtk.readthedocs.io/en/stable/

[28] *MongoDB*. [Online]. Available: https://www.mongodb.com/

[29] *UERANSIM on GitHub*. [Online]. Available: https://github.com/aligungr/UERANSIM

[30] *Feature Selection Using Variance Threshold in sklearn*. [Online]. Available: https://lifewithdata.com/2022/03/13/feature-selection-using-variance-threshold-in-sklearn/

[31] A. H. Anwar, G. Atia, and M. Guirguis, "It's time to migrate! a game-theoretic framework for protecting a multi-tenant cloud against collocation attacks," in *IEEE CLOUD*, 2018, pp. 725–731.

[32] R. Tahir, "A study on malware and malware detection techniques," *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018.

[33] Y. Ji, Q. Li, Y. He, and D. Guo, "Overhead analysis and evaluation of approaches to host-based bot detection," *International Journal of Distributed Sensor Networks*, vol. 11, no. 5, p. 524627, 2015.

[34] Y. Yue and B. Cheng, "EasyOrchestrator: A NFV-based network service creation platform for end-users," in *IEEE IPCCC*, 2018.

[35] M. Bonfim, F. Freitas, and S. Fernandes, "A semantic-based policy analysis solution for the deployment of NFV services," *TNSM*, vol. 16, no. 3, pp. 1005–1018, 2019.

[36] X. Zhang, Q. Li, J. Wu, and J. Yang, "vSFC: Generic and agile verification of service function chains in the cloud," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2020.

[37] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Flowcloak: Defeating middlebox-bypass attacks in software-defined networking," in *IEEE INFOCOM*, 2018.

[38] ——, "Securing middlebox policy enforcement in SDN," *Computer Networks*, vol. 193, 2021.

[39] P. Zhang, "Towards rule enforcement verification for software defined networks," in *IEEE INFOCOM*, 2017.

[40] A. Zafeiropoulos, E. Fotopoulou, M. Peuster, S. Schneider, P. Gouvas, D. Behnke, M. Müller, P.-B. Bök, P. Trakadas, P. Karkazis *et al.*, "Benchmarking and profiling 5g verticals' applications: An industrial iot use case," in *IEEE NetSoft*, 2020, pp. 310–318.

[41] P. Munoz, I. De La Bandera, E. J. Khatib, A. Gómez-Andrades, I. Serrano, and R. Barco, "Root cause analysis based on temporal analysis of metrics toward self-organizing 5g networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 3, pp. 2811–2824, 2016.