VET5G: A Virtual End-to-End Testbed for 5G Network Security Experimentation

Zhixin Wen
Department of Computer Science
Binghamton University
Binghamton, New York, USA
zwen7@binghamton.edu

Harsh Sanjay Pacherkar
Department of Computer Science
Binghamton University
Binghamton, New York, USA
hpacher1@binghamton.edu

Guanhua Yan
Department of Computer Science
Binghamton University
Binghamton, New York, USA
ghyan@binghamton.edu

ABSTRACT

As 5G networks are gradually rolled out worldwide, it is important to ensure that their network infrastructures are resilient against malicious attacks. This work presents VET5G, a new virtual end-to-end testbed for 5G network security research experiments or training activities such as Capture-The-Flag competitions. The distinguishing features of VET5G include a home-grown 5G core network emulator written in Rust to ensure memory and thread safety, integration of OpenAirInterface's Radio Access Network emulator and the official Android emulator to achieve full end-to-end 5G network emulation, inclusion of a reference P4 software switch to assist with prototyping of defense mechanisms for 5G data planes, implementation of Python APIs for easy 5G network experimentation, and adoption of JupyterHub to support multi-user experimentation. In our experiments we demonstrate how to use VET5G for two attack scenarios in 5G networks as well as its performance when it is used in a 5G hacking project for a Mobile Systems Security course.

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

5G networks, security testbed, programmable switch, Rust

ACM Reference Format:
Zhixin Wen, Harsh Sanjay Pacherkar, and Guanhua Yan. 2022. VET5G: A Virtual End-to-End Testbed for 5G Network Security Experimentation. In *Cyber Security Experimentation and Test Workshop (CSET 2022), August 8, 2022, Virtual, CA, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3546096.3546111

1 INTRODUCTION

The decades-long evolution of mobile communications has led us to today's 5G systems with the promise of transforming every aspect of people's life. Advanced networking technologies such as programmability, virtualization, and edgefication have been widely adopted by 5G networks to enable a variety of application scenarios. The increased complexity of 5G network infrastructures due to these new changes, however, has also drastically enlarged their attack surface. The open nature of 5G networks makes them vulnerable to threats at various places, including mobile devices (e.g., malicious apps), base stations (e.g., rogue small cells), 5G core network functions (e.g., those from untrusted software vendors), and operational environments (e.g., public clouds with ill-intentioned co-tenants). Hence, to secure 5G network operations, new defense techniques are needed that can not only protect them from attacks targeting 5G-specific features such as network slicing and service-based architectures, but also make them resilient to general network-level or software-level attacks that have been plaguing the Internet for decades (e.g., malware and Distributed Denial of Service (DDoS) attacks).

While deployments of 5G networks are progressing rapidly worldwide with global 5G subscriptions expected to reach one billion in 2022 [51], there is a urgent need for testbeds that can assess 5G network security in various attack and/or defense scenarios. In this work we study how to develop a comprehensive 5G network security testbed for dual purposes of research and education. On one hand, the testbed should enable researchers to evaluate the consequences of malicious attacks in a realistic but controlled environment, or assess the effectiveness of proposed defense mechanisms before their deployments in the real world. On the other hand, the testbed can also be used to host Capture-The-Flag (CTF)-like competitions where contestants learn and practice penetration testing techniques against emulated 5G networks without concerns of causing harm to the real ones in operation.

Although there are a plethora of existing open 5G tools, such as OpenAirInterface (OAI) [32] and Open5GS [31], they are immature or inappropriate for such as testbed for the following reasons. *First*, there has been lack of end-to-end 5G network experimentation environments where real-world mobile applications (e.g., Google map and YouTube) are executed to generate large volumes of realistic 5G traffic. High-density high-fidelity 5G traffic generators can be extremely useful to those experiments that explore effective filtering rules to block unwanted packets (e.g., DDoS attack traffic). *Second*, existing open 5G tools mostly focus on implementations of 5G network functionalities instead of their security. However, if the implementations of 5G network functions are too buggy, it would be trivial for an attacker to perform successful low-level software attacks (e.g., software crashes due to memory corruption), making them inappropriate for CTF-like security experiments[1]. *Third*,

CSET 2022, August 8, 2022, Virtual, CA, USA

ACM ISBN 978-1-4503-9684-4/22/08…$15.00
https://doi.org/10.1145/3546096.3546111

[1] In CTF competitions participants' skills are assessed based on how quickly they can successfully exploit the vulnerabilities – which are usually **intentionally** placed – to cause undesirable consequences (e.g., disruption of system operations and leakage of sensitive user data).

programmability, which is a key feature of 5G networks to optimize their operations, has not been fully considered by existing 5G network emulators. For example, 5G User Plane Functions (UPFs) can be implemented on programmable switches to support flexible packet forwarding/routing/inspection rules but code running on such switches usually has to be written in a custom language (e.g., P4 [37]). To the best of our knowledge, none of the existing 5G network emulators have considered using programmable switches in the 5G data planes.

Against this backdrop, this work aims to develop a new testbed called VET5G (a **V**irtual **E**nd-to-End **T**estbed for **5G** Network Security Experimentation) that achieves the following design goals:

- **Full end-to-end 5G network emulation**: To assess resilience of 5G networks against various threat vectors, VET5G is designed to provide a full end-to-end 5G network experimentation testbed with emulated mobile devices, base stations, core network functions, and operational environments mimicking real-world deployments.
- **Secure implementation of 5G core**: As 5G core network functions may be deployed on untrusted public clouds, they create a Pandora's box for new attacks, which were hard to achieve for previous generations of mobile networks whose core network functions were usually implemented by dedicated equipment and operated in closed environments. We plan to use VET5G to hold CTF-like competitions where participants are rated by how quickly they can exploit *intentionally* planted software vulnerabilities in 5G core network functions. To this end 5G core network functions should be implemented securely with few *accidental* exploitable vulnerabilities, while having the flexibility of inserting *intentional* ones for testing contestants' hacking skills.
- **Programmability for prototyping defense mechanisms:** Programmability, which is widely deployed by 5G networks [36], should be reflected in our security-oriented VET5G testbed to allow easy and fast prototyping of defense mechanisms. Ideally, the defense solutions tested by VET5G can be transitioned into real-world deployments with minimal efforts.
- **Strong experiment isolation mechanisms:** Once deployed, the VET5G testbed can be shared by other researchers to study 5G network security as well as students for educational purposes. As multiple users can run experiments simultaneously in the testbed, VET5G should provide strong isolation mechanisms to prevent unauthorized modifications of users' experimentation data.

In a nutshell, we have made the following contributions in the development of VET5G: **(1)** We have implemented essential network functions of a 5G core network based on 3GPP Release 16 specifications in Rust, a secure programming language focused on memory and thread safety to prevent common attacks such as buffer overflow attacks. **(2)** We have integrated Android emulator [1], OAI's nrUE for New Radio-based User Equipment (UE) emulation [32], OAI's gNodeB (gNB) emulator [28], and our own Rust-implemented 5G core network emulator to achieve full end-to-end emulation of a standalone 5G network. **(3)** We have incorporated a reference P4 software switch [2] into VET5G to assist users with developing, testing, and debugging traffic rules for routing/forwarding/inspecting 5G packets. **(4)** We have designed and implemented a set of Python APIs to enable VET5G users to configure, control, and monitor security-related experiments in the testbed. **(5)** We have deployed JupyterHub [24] as a frontend for VET5G users to interact with the testbed while taking advantage of its security mechanisms for multi-user environments. **(6)** Using two example attack scenarios against 5G networks and one course project, we demonstrate the use of the VET5G testbed deployed on a local cluster machine.

The remainder of the paper is organized as follows. Section 2 summarizes related works. Section 3 introduces the architecture of VET5G. Section 4 presents the implementation details of VET5G. Section 5 shows our experimental results with VET5G. Section 6 discusses the limitations of this work. Section 7 makes concluding remarks and presents our plan for future work.

2 RELATED WORK

Our work has been inspired by previous efforts on building testbeds for cybersecurity research, including, but not limited to, Deter [48], SCADA security testbed [38], PowerCyber [44], and security testbed for Internet-of-Things (IoT) devices [52]. VET5G fills the vacancy of open testbeds for 5G network security experimentation. Moreover, the development of VET5G has also been heavily influenced by existing mature network emulation platforms and tools, including but not limited to, EmuLab [3], Mininet [4], PlanetLab [5], and GENI [6].

There has been a plethora of efforts dedicated to open source software tools for emulating 5G networks, which have been summarized in a recent comprehensive survey [36]. Some of these works [7, 28] focus on the Radio Access Network (RAN) part of 5G networks, while others such as Open5GS [31] and free5GC [22] have implemented the 5G core. Few of existing tools are aimed at providing full end-to-end 5G network emulation capabilities. Open5GCore [30] is a mobile core network testbed platform with rich 5G network features, but based on its project description its focus seems not to be 5G network security experimentation. OAI [32] has recently made public its source code for both RAN and 5G core emulation, but when the development of VET5G started in May 2020, OAI only had a road map for 5G core development. Moreover, in the National Science Foundation (NSF) Workshop on Next-G Security held in October 2020 [8], Prof. Florian Kaltenberger from EUROCOM commented in his talk that security was *not* the primary focus for OAI 5G development by its community. In contrast, VET5G provides various features to facilitate 5G network security experimentation in a controlled environment, including full end-to-end 5G network emulation, secure implementation of 5G core, programmability for prototyping defense mechanisms, and strong experiment isolation mechanisms.

With the emergence of 5G networks, a number of 5G labs and testbeds have been set up by various entities for their specific needs, such as conformance testing, research on wireless technologies, and cybersecurity evaluation [43]. For instance, the NSF Platforms for Advanced Wireless Research (PAWR) project has funded four city-scale advanced wireless testbeds supporting 5G technologies in New York City (COSMOS) [21], Salt Lake City (POWDER) [33], Research Triagle (AERPAW) [18], and Central Iowa (ARA) [20].

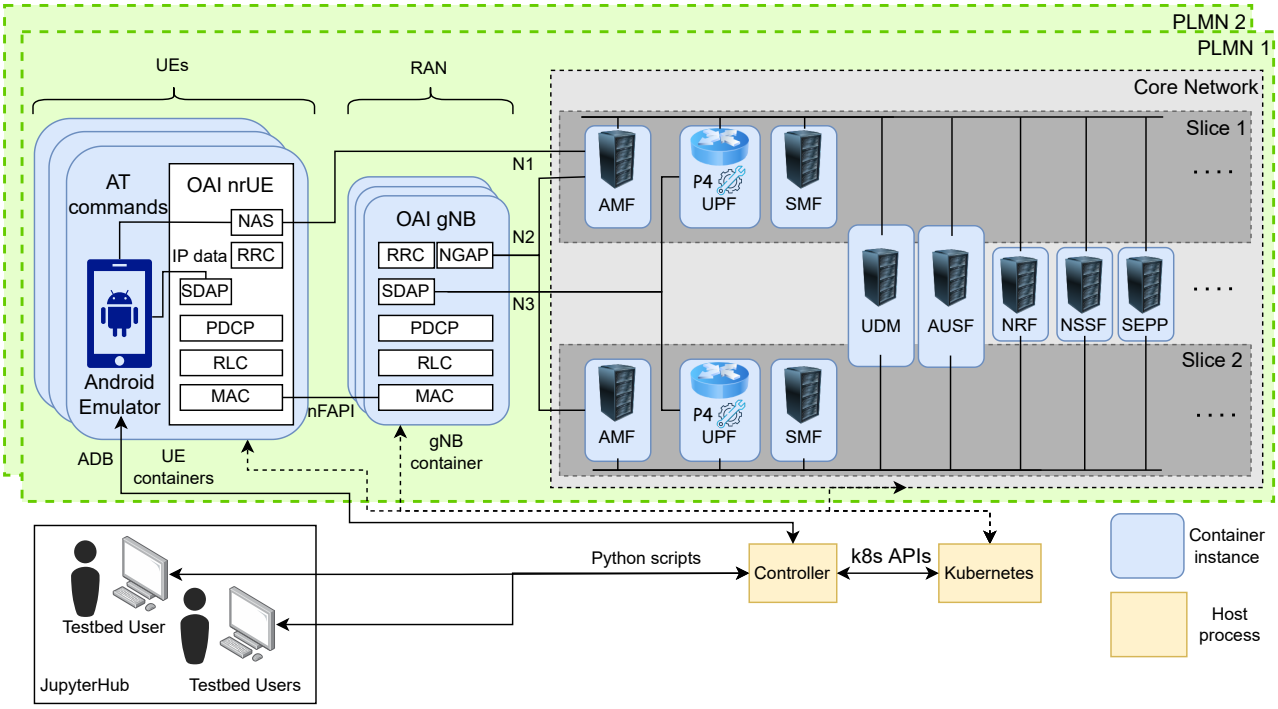

Figure 1: Overall testbed architecture. Rounded boxes marked in blue represents containerized components. The solid arrows mean normal control/data flows whereas the dashed arrows indicate pod management by Kubernetes.

The Colosseum Wireless Network Emulator hosted at Northeastern University [9] provides an open platform for RAN experimentation. The Idaho National Laboratory has set up a 5G wireless testbed with real-world scenarios to test performances of 5G technologies and develop solutions enhancing 5G security [10]. In contrast to these projects primarily focused on wireless communications in 5G RANs, VET5G aims to provide a virtual end-to-end emulation environment for conducting cybersecurity experiments. In our future work we plan to investigate potential integration with these wireless testbeds.

3 TESTBED ARCHITECTURE

This section introduces the architecture of the VET5G testbed, which is illustrated in Figure 1.

Workflow. The workflow of VET5G is shown in the bottom part of the figure. VET5G is operated on a Kubernetes cluster [25]. It uses JupyterHub to provide a multi-user experimentation environment with isolation protection for users' code and data. Inside JupyterHub, each user interacts with the testbed through Python code calling the testbed's client-side APIs. These APIs communicate with the testbed's controller, a Python-written server. By calling Kubernetes (k8s) APIs, the controller commands the underlying Kubernetes cluster to launch experiments and collect logs and/or network traffic files from their executions. The orchestration of the testbed will be explained in detail in Section 4.4 and VET5G's user APIs are defined in Section 4.5.

Experiments. Within JupyterHub, a user can use VET5G user APIs to launch *experiments*, which resemble *jobs* in supercomputer environments. In each experiment, a user can create one or more Public Land Mobile Network (PLMN) instances. Each PLMN instance includes a 5G core network (which consists of various Network Functions (NFs) summarized in Table 1), a RAN (which consists of one or more gNBs), and one or more UEs. These components are emulated by containerized workloads shown as blue rounded boxes in Figure 1. A 5G core network is emulated by containerized 5G core NFs with home-grown Rust implementations according to 3GPP Release 16 specifications. A gNB container runs OAI's gNB emulator. A UE container integrates a modified Android emulator with OAI's nrUE emulator to support transmissions of AT (ATtention) commands in the control plane and IP data in the data plane. Communications between a UE container and a gNB container are implemented to follow 5G Network Functional Application Platform Interface (nFAPI) standards. In Sections 4.1, 4.2, and 4.3, we shall elaborate on how 5G cores, RANs, and UEs are emulated, respectively.

Compared with a general-purpose 5G network testbed, VET5G offers two important features to support 5G network security research. *First*, VET5G allows a testbed user to control the Android devices through its user APIs, such as uploading new Android Application Packages (APKs) and pinging other machines. We implement this feature by exposing relevant Android Debug Bridge (ADB) commands to the user through the controller, which runs a separate ADB client instance for every Android device emulated in the testbed. *Second*, VET5G supports programmable UPFs which enable users to customize data plane control using P4, a domain-specific language for packet processing [37]. VET5G users can thus

develop, test, and debug defensive packet-forwarding logic (e.g., traffic filtering rules) written in P4 in a controlled environment before deploying them in real 5G networks.

Result collection. VET5G returns two types of experimental results to the user: execution logs and network traffic dumped as pcap files [34]. The latter are transmitted by the controller to the user's workspace in JupyterHub once the experiment is finished. A user can also use ADB commands exposed by the controller to monitor the state of each Android device directly in an experiment.

4 TESTBED IMPLEMENTATION

This section presents the implementation details of VET5G.

4.1 5G Core Emulation

We chose to implement 5G core NFs from scratch using Rust due to the following reasons. (1) When the VET5G project started in May 2020, alternative open source implementations such as open5GS [31], free5GC [22], and OAI [32] were either incomplete or not started yet. (2) As our goal is to achieve full end-to-end emulation of 5G networks, during our own 5G core implementations we ensure that they comply with 5G specifications strictly so as to interact with other 5G network components developed by third parties seamlessly. In contrast, it can be a tedious task to verify whether an existing 5G core emulator has indeed followed 5G protocol standards. (3) We plan to use the VET5G testbed to hold CTF-like competitions where vulnerabilities are *intentionally* added to assess participants' hacking skills. Hence we try to avoid *accidental* vulnerabilities that are exploitable by the contestants. We choose Rust over C/C++ (e.g., used by OAI [32] and Open5GS [31]) or Go (e.g., used by free5GC [22]) because its ownership and type systems can achieve both memory and thread safety efficiently [46]. Without needing expensive runtime garbage collection, Rust's ownership model prevents *accidental* memory vulnerabilities (e.g., use after free, null pointer dereference, use of uninitialized memory, double free, and buffer overflow), which are common to C/C++ programs with manual memory management. Moreover, Rust's use of ownership and type checking ensures that many concurrency errors should be discovered at compile time, effectively avoiding runtime data races; such vulnerabilities are difficult to detect and overcome for multi-threaded computing, which is needed by 5G core NFs to deal with vast parallel sessions. It is noted that, by using Rust's Foreign Function Interface (FFI) feature, *intentional* vulnerabilities can be added to Rust programs by calling unsafe modules implemented in other languages [46].

The 5G core network adopts a Service-Based Architecture (SBA), where NFs communicate with each other as service consumers or providers via HTTP2 RESTful APIs. Using Rust, we have implemented the following essential 5G core NFs: Network Function Repository Function (NRF), Network Slice Selection Function (NSSF), Unified Data Management (UDM), Authentication Server Function (AUSF), Access and Mobility Management Function (AMF), Session Management Function (SMF), UPF, and Security Edge Protection Proxy (SEPP). Their main features are summarized in Table 1.

5G SBA security can be enforced on direct communications among NFs using token-based authentication, or on indirect communications among them using a Service Communication Proxy [35]. Currently VET5G has implemented the former approach where the NRF acts as an authorization server to grant OAuth 2.0 access tokens [29] to consumer NFs for using services from provider ones. We plan to implement the latter in our future work.

Specification compliance. 3GPP has chosen OpenAPI 3.0 to define the interfaces of NFs within the SBA of a 5G core network. The OpenAPI 3.0 specifications of these APIs for Release 16 have been archived here [11] in YAML format. From these YAML files, we use openapi-generator [12] with option *rust-server* to generate both client and server code of each NF in Rust. As the generated code does not support *multipart/related* MIME [47] which is required by AMF and SMF, it is enhanced with our own implementation at places where this MIME is needed.

Abstract Syntax Notation One (ASN.1) is a standard interface description language for defining data structures that can be serialized and deserialized in a vendor-independent way. It is used by the NGAP protocol in AMF for its communications with gNBs (i.e., N2 interface in Figure 1). The NGAP protocol definition provided by 3GPP includes ASN.1 code using the Aligned Packed Encoding Rules (APER) serialization scheme. As we cannot find any existing ASN.1 library to translate it to Rust code directly, we use Rust's FFI feature to incorporate an alternative C library into AMF as follows. We first use the *asn1c* tool [13] to generate C code from the provided NGAP ASN.1 code and then add wrapper code also written in C around generated encoding and decoding procedures for certain NGAP messages (e.g. *InitiatingMessage*). This C code is further compiled into a static library, which is linked by the final AMF and SMF executables.

Asynchronous execution. Key 5G network protocols such as NGAP, NAS, and PFCP need to process large numbers of communication messages from various other entities, making it a challenge to achieve high performance. Our implementation takes advantage of Rust's *async* feature as follows. For each incoming request an asynchronous computation called *future* is created with a unique response identifier, which is stored to later identify the corresponding response. The function making the request is then suspended with its context (e.g., local variables) stored. On the arrival of the response the *waker* handler of the future is used to resume execution of this function. Such an *async* feature not only offers great flexibility to programmers but also increases concurrency to achieve high performance.

P4 switch. 5G UPFs are responsible for interconnections between RANs and Data Networks (DNs), packet routing/forwarding/inspection, QoS management, and usage reporting. VET5G allows testbed users to develop customized packet filtering rules for P4 switches used in UPFs. Our UPF implementation includes three components: a frontend, a backend, and a translation layer in between. The frontend handles communications from SMF and converts its instructions into Intermediate Representations (IRs). These IRs are further forwarded to the translation layer, which translates them into P4 table entries before sending them to the backend. The backend can be any P4 switch implementing P4Runtime [14] APIs. In our implementation we use the P4 BMv2 software switch [15] as the backend.

Table 1: 5G core network functions implemented in VET5G

NF	Features implemented	Implementation highlights
NRF	NF registration, discovery, notification; access token	NRF distributes public keys used by OAuth2 during NF registration
NSSF	Slice selection; slice availability update	
SEPP	TLS security between SEPPs; telescopic Fully Qualified Domain Name (FQDN); *3gpp-Sbi-Target-apiRoot* header	Support rate limiting to prevent Denial of Service (DoS) attacks
UDM	Data retrieval; UE authentication; Subscriber Identity Deconcealing Function (SIDF) with Subscription Concealed Identifier (SUCI) protection schema A (based on ed25519)	Directly use MongoDB [27] without Unified Data Repository (UDR)
AMF	UE (de-)registration, mobility, location, service request, and UE context management; N1 signaling security; AES, Snow3G, and ZUC	Asynchronous handling of Non-Access Stratum (NAS) and NG Application Protocol (NGAP) requests; use ASN.1c for NGAP
AUSF	UE authentication	Support 5G Authentication and Key Agreement (5G-AKA)
SMF	Protocol Data Unit (PDU) session creation, update, and removal; PDU session management	Asynchronous handling of Packet Forwarding Control Protocol (PFCP) requests; use ASN.1c for NGAP
UPF	Packet forwarding with support for PDU Session Anchor (PSA), Intermediate UPF (I-UPF), and Uplink Classifier (UL-CL); Quality of Service (QoS) enforcement; packet buffering	Flexible forwarding backend; programmable P4 backend support; optional support for Carrier Grade Network Address Translation (CGNAT)

4.2 RAN Emulation

A 5G RAN consists of one or more gNBs. The current implementation of VET5G uses OAI's gNB emulator due to its full support for 5G-NR protocol stack as well as our familiarity with its code base from our previous projects [40, 41]. OAI's RAN development is however still ongoing [28], suggesting that some features we need for VET5G may not be available or mature. Particularly, its current RAN code cannot emulate multiple UEs connecting to the same gNB, which becomes a hurdle when VET5G is used for simulating large-scale attacks from many mobile devices.

To address this issue, we need to change OAI's implementation of 5G nFAPI. 5G nFAPI enables a gNB Virtual Network Function (VNF) to communicate with multiple Physical Network Functions (PNFs) based on a standardized interface between the MAC layer and the PHY layer. As OAI's gNB emulator supports only one PNF in its nFAPI implementation, we extended it as follows. Assuming that each UE is a PNF and the gNB is a VNF according to 5G nFAPI, communication messages between UEs and the gNB are transmitted directly among their MAC layers without going through the underlying physical layers (see Figure 1). Inside the gNB emulator, we add a global context to store states for PNFs connected to the gNB. Each PNF is uniquely identified by the corresponding UE's 5G New Radio (NR) Cell Radio Network Temporary Identifier (C-RNTI). A message from the VNF (i.e., the gNB) is sent to a PNF (i.e., a UE) if and only if one of the following conditions is satisfied according to the current state: the target of this message is unknown, the identity of this PNF is unknown, or the message's target matches a known PNF's identity. The identity of a PNF is learnt by examining incoming messages from it within a certain period of time (100 5G NR slots or 5 frames in our settings). Compared with a straightforward implementation that broadcasts the VNF's messages to all PNFs, this optimization saves a significant amount of network bandwidth in our testbed. Currently RAN emulation in VET5G does not aim to achieve high-fidelity emulation at the physical layer, so the gain in scalability due to this optimization is favored over the resulting loss in emulation fidelity.

4.3 UE Emulation

UEs in VET5G are emulated by Android Emulator [1] that runs Android 11 (or Android API level 30) with 5G support. In mobile devices, AT commands are commonly used to control their cellular modems such as sending/receiving phone calls and getting device/manufacturer information. In the official Android emulator, AT commands are transmitted between the Android Operating System (OS) and its Global System for Mobile Communications (GSM) Service using a character pipe. In order for the Android OS to communicate with the baseband processor emulated by OAI nrUE, we have modified Android Emulator to support transmission of AT commands between them. On entering the character pipe towards the GSM service, AT commands are intercepted and forwarded to the NAS layer in nrUE's protocol stack using UDP sockets. The responses of the AT commands are sent back from OAI nrUE in a similar way. For data plane connections of Android Emulator, a tunnel interface is created inside the UE container on successful PDU session establishment. This tunnel interface is directly connected to the SDAP layer of OAI nrUE's protocol stack. Thus, all data packets from the Android Emulator are routed via this tunnel interface. As shown in Figure 1, the NAS layer in OAI nrUE is the entry point for control plane packets and its SDAP layer is the entry point for data plane packets.

In VET5G, the native AT Command handler of an Android Virtual Device (AVD) is switched to OAI nrUE's AT command handler implementation. After an AVD successfully registers itself into a 5G network, its screen should show its cellular signal strength and a 5G logo, denoting that its 5G data connection has been set up. However, this cannot be achieved if we forward AT commands from an AVD to OAI nrUE's AT command handler blindly because of the following two reasons. *First*, the network registration procedure of an AVD differs from that of a regular Android device, whose network settings such as the initial Access Point Name (APN) for

connection are obtained from its service provider. By contrast, such settings are predefined on an AVD. *Second*, the AVD sends out AT commands in a specific order to register telephony service for the device. Any changes in the responses to these AT Commands can cause incomplete network registration or no data plane connection. OAI's original AT command handler has not provided full support for all the necessary AT commands sent from the AVD.

To overcome these challenges, we first add a sequence of AT commands and responses between AVD and nrUE to fulfil the following workflow: (1) the AVD is initialized with a predefined APN name and domain; (2) the AVD sends a General Packet Radio Service (GPRS) service request to the nrUE, which forwards it to the core network for establishing a new PDU session for the UE; (3) the AVD obtains the context identifier of the activated APN from the nrUE, and (4) the AVD obtains the latest Packet Data Protocol (PDP) context from the nrUE. We also implement more than 20 missing AT commands in OAI nrUE and change the AT command handler's responses to some existing AT commands in order for them to match the values expected by the AVD. After all these efforts, 5G data connectivity becomes available to the applications running inside the AVD.

4.4 Testbed Orchestration

VET5G uses Kubernetes to orchestrate containerized components, including all 5G core NFs, gNBs, and UEs. To ensure experiment isolation, each PLMN is represented as a unique namespace to identify all the containerized components in it. Namespaces are created by the controller and removed after the experiment is done.

5G Core. Each NF instance in the core network includes three parts in the Kubernetes cluster: a *ConfigMap* storing its configuration, a *StatefulSet* with one replica, and a Service referring to this NF instance. A *StatefulSet* is chosen because a known fixed name pattern is used to name this container instead of a random identifier which is required when constructing a Domain Name System (DNS) name for this NF. The ConfigMap is used to mount custom P4 code into UPF containers.

gNB. For each gNB, a ConfigMap storing its OAI configuration and a StatefulSet is created. As OAI's gNB configuration only works with IP addresses instead of domain names, a gNB instance must be started after the core network so the AMF's IP addresses can be passed to it. If there are multiple AMFs in the core network then a gNB will connect to all of them.

UE. A UE container includes both OAI's nrUE emulator and an Android Emulator instance. For each UE container a ConfigMap and a StatefulSet is created. Since UE containers require the IP addresses of gNBs, they must be started after gNBs are deployed. Once a UE container is started, it opens port 5555 for connections from ADB clients. This allows testbed users to control the UE through ADB commands, such as installing a new Android APK.

Result collection. We include a statically linked *tcpdump* binary in all 5G core NF containers to collect network traffic. For any NF, an argument can be passed to the container to allow all its packets to be dumped. For a UPF, in addition to the aforementioned feature, it is also possible to dump packets specific to a GPRS Tunnelling Protocol (GTP)-U tunnel or user plane traffic from/to the Internet. All traffic files are copied to the user's local directory using the *kubectl cp* command when the experiment is finished.

Logs can be fetched from pods using Kubernetes' logging APIs whereas Android system logs can be acquired via ADB commands.

Table 2: List of Python APIs supported by VET5G

Python API	Meaning
create_subscribers	Create mobile subscribers within a PLMN
start_core_network	Start a core network instance within a PLMN
start_gnb	Start gNB(s) within a PLMN
start_ue	Start UE(s) within a PLMN
execute_shell_command	Execute a shell command in an Android Emulator
install_apk	Install an Android APK in an Android Emulator and return its package name
start_app	Start an Android APK with a given package name in an Android Emulator
stop_all	Stop an experiment and collect results
get_nf_log	Get logs of an NF instance (it must be called after *stop_all*)

4.5 Testbed User APIs

The Python APIs provided for users to conduct experiments in VET5G are summarized in Table 2. Listing 1 shows how to use these APIs to launch a VET5G experiment on slicing attacks. Here we only explain the skeleton of the Python code while leaving the details of slicing attacks to Section 5.1.

```
1  import asyncio
2  from vet5g.client import VET5GClient
3  from vet5g.models import PlmnId
4  async def main() :
5      plmn = PlmnId('208', '99')
6      vet5g = VET5GClient('user1', [plmn], 'slicing-attack-result/')
7      await vet5g.create_subscribers(plmn,
8            cfg_file = 'scenario1_subscribers.yaml')
9      cn_cfg = await vet5g.start_core_network(plmn,
10           cfg_file = 'scenario1_cn.yaml')
11     gnb_cfg = await vet5g.start_gnb(plmn,
12           cfg_file = 'scenario1_gnb.yaml')
13     ue_cfg = await vet5g.start_ue(plmn,
14           cfg_file = 'scenario1_ue.yaml')
15     await asyncio.sleep(10) # wait for 10 seconds
16     await vet5g.stop_all()
17     attack_logs = await vet5g.get_nf_log(plmn,
18           'slicing-attacker')
19     print(attack_logs)
20 if __name__ == '__main__' :
21     loop = asyncio.get_event_loop()
22     loop.run_until_complete(main())
```

Listing 1: Example Python code for slicing attacks

The code first imports *VET5GClient* from the *vet5g.client* module (Line 2), which serves as the client to interact with the testbed. Next data structure *PlmnId* is imported from the *vet5g.models* module (Line 3) where all 5G data models are defined. The *main* function, which is defined as a Python coroutine, specifies how an experiment should be performed (Lines 4-19); it is invoked within Lines 20-22.

The experiment starts by defining a PLMN, whose mobile country code is 208 and mobile network code is 99 (Line 5). It then creates a VET5G client given username *user1*, the PLMN just defined, and a local workspace directory at *./slicing-attack-result/* (Line 6) where results like pcap dumps are stored. Thereafter instances of subscribers, a core network, a gNB and a UE are created by calling their corresponding APIs along with their respective configuration YAML files (Lines 9-14). By calling *asyncio.sleep*, the main function is suspended to wait for the execution of the experiment for 10 seconds (Line 15). The experiment is terminated after calling *vet5g.stop_all* (Line 16) and the results related to the PLMN are collected from the testbed by calling *vet5g.get_nf_log* (Line 17) where *slicing-attacker* indicates the name of the NF to collect logs from (Line 18).

5 EXPERIMENTAL EVALUATION

This section presents our experimental results from VET5G deployed on a local Kubernetes cluster machine with 36 physical cores (72 logical ones with hyperthreading) and 192GB RAM.

We first show some basic resource usage results when using VET5G to emulate a PLMN with one core network, one gNB, and 15 Android UEs. We run each experiment 10 times and observe its mean resource usage as follows: **(1) Storage:** An UE image, which contains Android Emulator and OAI nrUE, uses about 5GB of disk space. The UPF image takes 160MB storage while all other NF images about 30MB each. **(2) Memory:** The memory footprint of the experiment is about 104.65GB with each UE using around 7GB RAM. **(3) Execution time:** Creation of subscribers (*create_subscribers*) takes 6.34 milliseconds. It takes 41.18 seconds to start the core network (*start_core_network*), 12.07 seconds to start the gNB (*start_gnb*), and 38.93 seconds to start each UE (*start_ue*). Finally it takes 64.22 seconds to stop the experiment and fetch the results.

Next we demonstrate how to use VET5G for two attack scenarios in 5G networks and then present our observations from a course project where students use hacking tools from Kali Linux to attack emulated 5G networks.

5.1 Scenario 1: Slicing Attacks

Network slicing, which is a new concept in 5G networks, partitions the same physical infrastructure to support multiple logical networks called *slices*, each of which is configured for a particular type of service needs. As shown in Figure 1, two slices can run their own NFs (e.g., AMF, UPF, and SMF) while sharing some others (e.g., UDM, AUSF, NRF, NSSF, and SEPP). NFs in different slices can use a shared NRF for service authentication in the 5G SBA. Between two NF instances, which are called *Service Consumer* and *Service Provider* for ease of presentation, NRF acts as an OAuth 2.0 Server to facilitate service authentication. More specifically, the following steps are involved. ① The Service Consumer registers with the NRF. ② The Service Consumer sends an Nnrf_AccessToken_Get request, which includes a slice identifier, to the NRF. ③ The NRF checks whether the slice identifier in the request matches the one stored in the NFProfile of the Service Provider. ④ If the check is successful, an access token is generated by the NRF, which is further sent to the Service Consumer through an Nnrf_AccessToken_Get response message. ⑤ Finally, the Service Consumer sends a Service Request message to the Service Provider, which includes the access token obtained from the NRF.

Attack description. When a Service Provider NF instance is shared by multiple slices, it may contain slice-specific sensitive data. However, the NRF-based service authentication scheme, as described above, does not prevent such data from being leaked to other slices. Based on this observation, Adaptive Mobile has published a few attacks against the service authentication procedures across different 5G network slices [16]. One such attack is illustrated in Figure 2. We assume that a misbehaving NF (attacker) belongs to Slice 2 and has already established a Transport Layer Security (TLS) connection with the NRF. A victim NF, in this case an AMF, is shared by both Slices 1 and 2, but should work on behalf of Slice 1 (or Slice 2) only if the request comes from *the same* slice.

The attack is carried out in the following steps. ① The misbehaving NF from Slice 2 sends an Nnrf_AccessToken_Get request to the NRF to access Slice 1 on the victim NF. ② As the victim NF is shared by both Slices 1 and 2, the NRF approves the request and generates a valid token for Slice 1 and the victim NF. ③ The NRF sends an Nnrf_AccessToken_Get response to the misbehaving NF, including the access token just generated. ④ The misbehaving NF sends a request for service in Slice 1 with the access token to the victim NF. In the example shown in Figure 2, the attacker requests the location information of UE with IMSI 2089900007487. ⑤ The victim NF approves the request because the access token is valid. ⑥ The victim NF responds to the service request from the misbehaving NF, which may contain sensitive UE data.

Attack simulation. To simulate the slicing attack, the example Python code shown in Listing 1 is used. Its core network configuration file *scenario1_cn.yaml* (Line 10) includes the following section describing a custom NF simulating the slicing attacker:

```
--- !custom NF section
custom_nfs:
  - name: slicing-attacker
    image_name: 'slicing-attacker-nf.tar'
    image_args: []
    slices: # Slices this NF belongs to
      - name: 'slice-2'
        sst: 1 # Slice/Service Type
        sd: '000002' # Slice Differentiator
    allowed_slices: null # what slices are allowed
    # to use NF's resources, use `null` to be the
    # same as `slices`
    dump_pcap: true # To dump pcap file or not
    dump_logs: true # To dump NF log or not
```

In the example, the *slicing-attacker* NF is instantiated from a container image named *slicing-attacker-nf.tar*. To facilitate custom NF development, VET5G provides a set of scaffolding code written in Rust as well as container building scripts. The testbed user can use these resources to implement the aforementioned steps involved in a slicing attack and build the corresponding container image.

Attack result. Figure 2 presents the Wireshark traffic analysis in a slicing attack. The two boxes following ① and ③ show that the attacker successfully obtains an access token from the NRF for the victim NF shared by Slices 1 and 2. Next the two boxes following ④ and ⑥ demonstrate that this access token is used to obtain the location of UE with IMSI 2089900007487 which belongs to Slice 1.

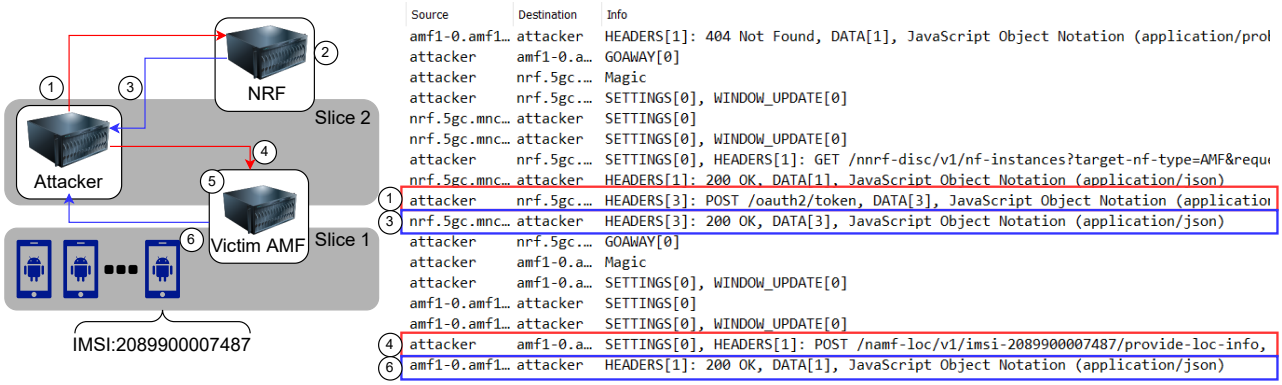

Figure 2: Wireshark analysis of slicing attack traffic. The red boxes highlight the attacker's requests and and blue ones the responses received. One of the phones in Slice 1 has IMSI 2089900007487 whose location is requested in Step ④.

Table 3: Attack topics in the 5G hacking course project, where 40 participating students are divided into 15 groups.

Attack Type	Description	Number of groups
Fuzzing	Use tools such as wfuzz and sfuzz to fuzz test 5G core NFs	14
Reconnaissance	Use tools such as nmap to find attack targets in 5G networks	9
GTP attack	Guess TEIDs (Tunnel Endpoint Identifiers) used by legitimate GTP sessions	3
DDoS attack	Use tools such as slowloris to perform DDoS attacks	3
Password attack	Use tools such as Hydra to guess passwords	3
MITM attack	Use tools such as Bettercap to perform MITM (Man-In-The-Middle) attacks	2
SSH attack	Use tools such as metasploit to attack vulnerable SSH services	1

5.2 Scenario 2: Cellular Botnets

A cellular botnet is a collection of cellular devices (e.g., smart phones and IoT devices) that have been infected by bot malware and can be controlled by a botmaster remotely through its Command and Control (C&C) channel. Previous research has shown that cellular botnets have the potential of impairing operations of cellular network cores [53], delaying setups of voice or video calls through paging storm attacks [41], and causing disruption or degradation of Internet services (e.g., web services) [42]. As 5G networks are *not* immune to the attacks from these cellular botnets, it has become an important research topic to investigate 5G-oriented solutions to detect and mitigate these threats [49].

Attack description. To demonstrate the use of VET5G for research on cellular botnets, we consider emulation of the *Matryosh* Android botnet, which was discovered spreading via open ADB port 5555 in early 2021 [17]. Our experiment models both the propagation and attack phases of Matryosh-like cellular botnets within a 5G network, as illustrated in Figure 3: ① The attacker scans devices with open port 5555. ② If the 5G network deploys CGNAT, scanning packets from the external Internet cannot reach vulnerable end devices assigned with non-routable private IP addresses. ③ Otherwise, ADB connections can be established with vulnerable devices so the attacker can execute shell commands to install bot malware on them. ④ The bot malware launches DDoS attacks against a victim server.

Attack simulation. We develop two Android apps for the attack, one for the botmaster and the other for individual bots. The botmaster scans for vulnerable devices with TCP port 5555 open,

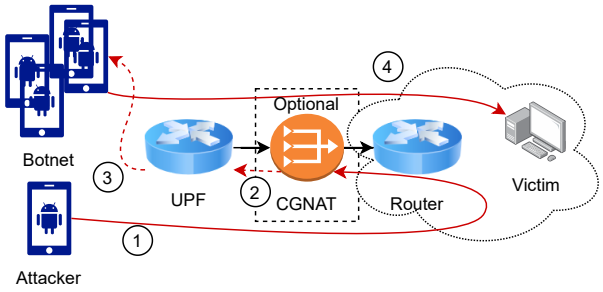

Figure 3: Illustration of a botnet attack via exposed ADB ports

uploads the bot malware onto any vulnerable device found, and commands new bot-infected devices to launch an HTTP pipelining DDoS attack against a server within the same mobile network. Vulnerable devices are emulated by rooted Android images in UE containers.

In our experiment, the attack involves six vulnerable UEs. The botmaster commands the bots to start their DDoS attacks sequentially, separated by the same time intervals of two seconds. The result of this attack is shown as the red dotted line in Figure 4.

Defense deployed by P4 switch. To defend against this attack we consider the method proposed in [39] where a modified count-min sketch data structure is used to detect if the number of UEs connecting to the same destination exceeds a certain threshold. If this condition is satisfied then a *TrTCM meter* [45] is used to limit the maximum bit rate going to that destination. Such a rate limiting

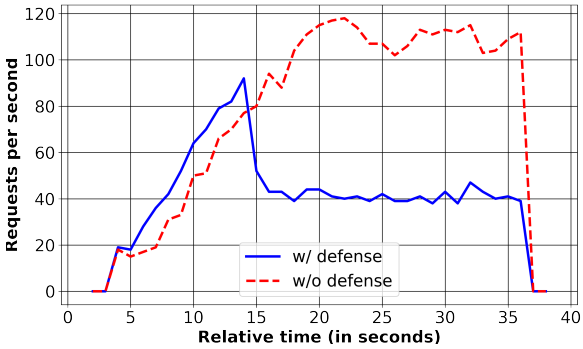

Figure 4: Rate of HTTP requests received by the victim server in a botnet attack

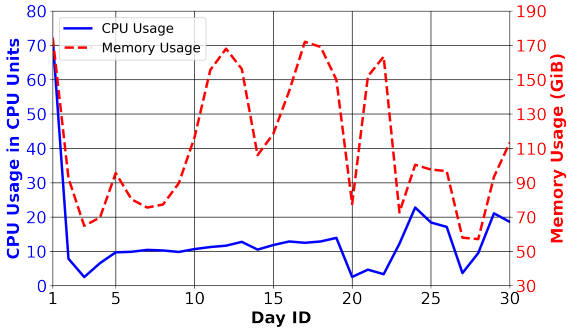

Figure 5: CPU and memory usage of the testbed during the course project

scheme can be implemented with ~100 lines of additional P4 code and ~40 lines of additional UPF code written in Rust. The defense effect is shown as the blue solid line in Figure 4. Clearly, after the defense mechanism is deployed, the traffic rate received by the victim server is limited to around 40 requests per second.

5.3 5G Hacking Course Project

The VET5G testbed has been used for a 5G hacking project by 40 students who are taking a Mobile Systems Security course in our university. To foster teamwork, the 40 students form 15 groups, each of which needs to deliver a final project report summarizing its findings. In this project, each student can create a separate PLMN with one 5G core, one gNB, and one UE. In addition to a PLMN, a student also has access to a container running tools from Kali Linux to perform offensive attacks against any components inside the PLMN.

Table 3 summarizes the types of attacks attempted by each team. Almost all the teams (14/15) have used fuzzing tools to test the 5G NFs listed in Table 1. Nine teams have used reconnaissance tools (e.g., nmap and traceroute) to infer the details of 5G networks. Some teams have also performed specific attacks such as GTP attacks[2], DDoS attacks, password guessing attacks, MITM attacks, and SSH attacks.

Over the one-month period of the course project (04/14/2022 - 05/13/2022), students have conducted 298 experiments on VET5G. The duration of these experiments has a mean of 860.24 minutes and a standard deviation of 1558.61 minutes. Clearly there is large variation among the lengths of these experiments performed by the students. Using the measurements from Kubernetes' Metrics Server, we derive that the mean and standard deviation of the CPU usage consumed by each experiment is 0.080 and 0.152 CPU units, respectively; the mean and standard deviation of the memory usage by each experiment is 1.390 and 1.589 GiB (Gibibytes), respectively.

Figure 5 presents the CPU and memory usage of the cluster machine deploying VET5G over the one-month period of the course project. Recall the cluster machine has 72 logical cores and 192GB RAM. It is noted that at the beginning of the course project, both the CPU and memory usages approached the physical limits of the cluster machine. This is probably because the students were busy with learning with how to use VET5G at that time. After the first few days, CPU usage became low (usually less than 20 CPU units), but there were a few spikes on memory usage. After examining the logs, we find that these spikes were caused by the fact that some students forgot to call *stop_all* to terminate their experiments explicitly from their interactive sessions in JupyterHub. Hence, these inactive experiments did not consume much CPU usage, but their memory was never released back to the system.

[2]GTP vulnerabilities are well-known issues for mobile core networks [26, 50]. 5G networks use GTP on their N2 and N3 interfaces (see Figure 1) and thus inherit the same security risks. To create fake GTP packets an attacker needs to know TEIDs of legitimate sessions, which sometimes can be found through brute force attacks.

6 LIMITATIONS

The current implementation of VET5G has the following limitations.

First, its RAN emulator, which is based on OAI's gNB emulator, ignores the wireless communications at the physical layer, making it inappropriate to emulate any realistic attack scenarios through the 5G air interface. Due to the same reason, VET5G does not support wireless communications from real 5G phones, suggesting that it cannot be used to investigate security issues of these devices.

Second, an attacker may compromise the underlying computing infrastructure of 5G networks to disrupt their operations. Within the VET5G testbed a user can attack an emulated 5G network by exploiting security vulnerabilities of Kubernetes or Docker containers, but these activities may affect other users because VET5G has only deployed mechanisms to isolate users' experiments at the network instance level, but not at the infrastructure level.

Third, currently VET5G uses the BMv2 P4 software switch [15] to emulate programmable switches for UPFs. As BMv2 is *not* a production-level software switch, it can only be used for developing, testing, or debugging the P4 code written for UPFs. Any throughput or latency measurements obtained from the BMv2 software switch in the testbed do *not* correctly capture the UPF's performances in an operational 5G network where a real P4 switch is used.

Last, although we strive to provide a secure and comprehensive implementation of 5G core networks according to 3GPP specifications, due to their complexity as well as our limited staff, the current implementation of VET5G only includes some basic functions of the essential 5G core NFs listed in Table 1. Other features that we have not implemented may impact the fidelity of emulation results observed from VET5G.

7 CONCLUSIONS AND FUTURE WORK

In this paper we present the design and implementation details of VET5G, a new virtual end-to-end testbed dedicated to 5G network security experimentation. We also demonstrate the use of VET5G for two attack scenarios and a Mobile Systems Security course project.

The VET5G testbed is still under active development. In the future we plan to enrich its features and improve its usability. For example, we plan to integrate Software Defined Radio (SDR)-based RAN emulators (e.g., OAI gNB [32], srsRAN [7], and ARAMI Callbox [19]) to enable security experiments conducted through the 5G air interface. We are working towards a UPF implementation based on a real Tofino-based P4 switch [23]. Once it is ready we plan to integrate it into VET5G so some cybersecurity experiments (e.g., DDoS attacks in the data plane) can produce more accurate measurements of attack effects from the testbed. We will also explore how to insert intentional vulnerabilities into emulated 5G networks to support CTF-like competitions. We will investigate effective yet efficient techniques to automatically validate whether our Rust-implemented 5G core network is immune to existing or new software exploitation attacks, or whether it contains any deviations from 3GPP specifications that pose potential security risks. We also plan to integrate various end devices, such as IoT, VR (Virtual Reality), and AR (Augmented Reality) devices into VET5G to study their security issues. Particularly, the inclusion of XR (eXtended Reality) technologies will enable us to understand limitations of 5G networks and explore new solutions for 6G and beyond.

VET5G runs on a local cluster machine inside Binghamton University's campus network. To support access from external users, we plan to deploy VET5G on a server in the DMZ (Demilitarized Zone) of the university's network. More details can be found at the project's website: http://cybersec.cs.binghamton.edu/vet5g.

ACKNOWLEDGMENTS

We thank our paper shepherd David Balenson and the anonymous reviewers for their valuable feedback on this paper. This work is partially supported by US National Science Foundation under Grant CNS-1943079.

REFERENCES

[1] [n. d.]. https://developer.android.com/studio/run/emulator.
[2] [n. d.]. https://github.com/p4lang/behavioral-model.
[3] [n. d.]. https://www.emulab.net/.
[4] [n. d.]. http://mininet.org/.
[5] [n. d.]. https://planetlab.cs.princeton.edu/.
[6] [n. d.]. https://www.geni.net/.
[7] [n. d.]. https://www.srsran.com/.
[8] [n. d.]. https://nsf-nextg-security.cs.ucsb.edu/.
[9] [n. d.]. https://www.northeastern.edu/colosseum/.
[10] [n. d.]. https://inl.gov/trending-topic/5g-wireless-technology/.
[11] [n. d.]. https://www.3gpp.org/FTP/Specs/archive/OpenAPI/Rel-16.
[12] [n. d.]. https://github.com/OpenAPITools/openapi-generator.
[13] [n. d.]. https://github.com/vlm/asn1c.
[14] [n. d.]. https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html.
[15] [n. d.]. https://github.com/p4lang/behavioral-model.
[16] [n. d.]. https://info.adaptivemobile.com/5g-network-slicing-security.
[17] [n. d.]. https://blog.netlab.360.com/matryosh-botnet-is-spreading-en/.
[18] [n. d.]. AERPAW: Aerial Experimentation and Research Platform for Advanced Wireless. https://aerpaw.org/.
[19] [n. d.]. AMARI Callbox Series. https://www.amarisoft.com/products/test-measurements/amari-lte-callbox/.
[20] [n. d.]. ARA Wireless Living Lab for Smart and Connected Rural Communities. https://arawireless.org/.
[21] [n. d.]. COSMOS: Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment. https://cosmos-lab.org/.
[22] [n. d.]. free5GC. https://www.free5gc.org/.
[23] [n. d.]. Intel Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.
[24] [n. d.]. JupyterHub: A multi-user version of the notebook designed for companies, classrooms and research labs. https://jupyter.org/hub.
[25] [n. d.]. Kubernetes: Production-Grade Container Orchestration. https://kubernetes.io/.
[26] [n. d.]. The Mobile Core under Attack. https://www.vsec.infinigate.co.uk/hubfs/A10%20Networks/Resources/A10-WP-21154-EN-White-Paper-the-Mobile-Core-Under-Attack.pdf.
[27] [n. d.]. MongoDB. https://www.mongodb.com/.
[28] [n. d.]. OAI 5G RAN Project Group. https://openairinterface.org/oai-5g-ran-project/.
[29] [n. d.]. OAuth 2.0. https://oauth.net/2/.
[30] [n. d.]. open5Gcore. https://www.open5gcore.org/.
[31] [n. d.]. open5GS. https://www.open5gs.org/.
[32] [n. d.]. OpenAirInterface. https://www.openairinterface.org/.
[33] [n. d.]. Powder: the Platform for Open Wireless Data-driven Experimental Research. https://powderwireless.net/5g.
[34] [n. d.]. Tcpdump and libpcap. https://www.tcpdump.org/.
[35] 3GPP. 2021. *Security architecture and procedures for 5G system.* Technical Specification (TS). 3rd Generation Partnership Project (3GPP). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169 Version 16.4.0.
[36] Leonardo Bonati, Michele Polese, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2020. Open, programmable, and virtualized 5G networks: State-of-the-art and the road ahead. *Computer Networks* 182 (2020), 107516.
[37] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.
[38] C. M. Davis, J. E. Tate, H. Okhravi, C. Grier, T. J. Overbye, and D. Nicol. 2006. SCADA cyber security testbed development. In *Proceedings of the 38th North American Power Symposium*. IEEE, 483–488.
[39] Damu Ding, Marco Savi, Federico Pederzolli, Mauro Campanella, and Domenico Siracusa. 2021. In-network volumetric DDoS victim identification using programmable commodity switches. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1191–1202.
[40] Kaiming Fang and Guanhua Yan. 2018. Emulation-instrumented fuzz testing of 4G/LTE Android mobile devices guided by reinforcement learning. In *European Symposium on Research in Computer Security*. Springer, 20–40.
[41] Kaiming Fang and Guanhua Yan. 2020. Paging storm attacks against 4G/LTE networks from regional Android botnets: rationale, practicality, and implications. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 295–305.
[42] Paolo Farina, Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. 2016. Are mobile botnets a possible threat? The case of SlowBot Net. *Computers & Security* 58 (2016), 268–283.
[43] Federal Mobility Group. 2020. 5G Framework to Conduct 5G Testing. https://www.cio.gov/assets/files/Framework-to-Conduct-5G-Testing-508.pdf.
[44] Adam Hahn, Aditya Ashok, Siddharth Sridhar, and Manimaran Govindarasu. 2013. Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid. *IEEE Transactions on Smart Grid* 4, 2 (2013), 847–855.
[45] J. Heinanen and R. Guerin. 1999. *A Two Rate Three Color Marker.* RFC 2698. RFC Editor.
[46] Steve Klabnik and Carol Nichols. 2019. *The Rust Programming Language (Covers Rust 2018).* No Starch Press.
[47] E. Levinson. 1998. *The MIME Multipart/Related Content-type.* RFC 2387. RFC Editor.
[48] Jelena Mirkovic, Terry V Benzel, Ted Faber, Robert Braden, John T Wroclawski, and Stephen Schwab. 2010. The DETER project: Advancing the science of cyber security experimentation and test. In *Proceedings of the IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 1–7.
[49] Manuel Gil Pérez, Alberto Huertas Celdrán, Fabrizio Ippoliti, Pietro G Giardina, Giacomo Bernini, Ricardo Marco Alaez, Enrique Chirivella-Perez, Félix J García Clemente, Gregorio Martínez Pérez, Elian Kraja, Gino Carrozzo, Jose M. Alcaraz Calero, and Qi Wang. 2017. Dynamic reconfiguration in 5G mobile networks to proactively detect and mitigate botnets. *IEEE Internet Computing* 21, 5 (2017), 28–36.
[50] Positive Technologies. 2020. Threat Vector: GTP Vulnerabilities in LTE and 5G networks 2020. https://www.politico.eu/wp-content/uploads/2020/06/POLITICO-Positive-Technologies-report-Threat-vector-GTP-June-2020.pdf.
[51] Felix Richter. 2022. Global 5G Adoption to Hit One Billion in 2022 . https://www.statista.com/chart/9604/5g-subscription-forecast/.

[52] Shachar Siboni, Vinay Sachidananda, Yair Meidan, Michael Bohadana, Yael Mathov, Suhas Bhairav, Asaf Shabtai, and Yuval Elovici. 2019. Security testbed for Internet-of-Things devices. *IEEE transactions on reliability* 68, 1 (2019), 23–44.

[53] Patrick Traynor, Michael Lin, Machigar Ongtang, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. 2009. On cellular botnets: measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 16th ACM conference on Computer and communications security*. 223–234.