POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Development of a framework for Attack/Defense Capture the Flag competition

Supervisor
prof. Antonio Lioy

Candidate
Gianluca Canitano

Academic Year 2021-2022

To my parents,
for their constant presence
and for always pushing me
to do my best,

To my brother and sister,
for their precious advices
and continuous support

Summary

Nowdays cybersecurity is a very important field, thanks to global spread of Internet and smart devices, and demand for cybersecurity expert is continuously growing, so computer security education has become a priority. Together with security courses, security competitions are a very important tool which can be used for educational purposes.

Capture the Flag (CTF) competitions are most popular type of security competitions, because playing them contestants, grouped in teams, can improve their teamwork, and can apply, test and demonstrate practically their skills and theoretical knowledge learned in security courses, by solving challenges and/or exploiting vulnerabilities.

In particular, in Attack/Defense CTF, contestants have to interact and challenge each other, by exploiting services of other teams while they are patching their own services, focusing on both attack and defense and developing offensive and defensive skills. These competitions usually involve a lot of teams, so is not easy to implement an architecture that could be used to host them.

Purpose of this thesis is to describe framework that can be used to host an Attack-/Defense CTF that I have implemented, and some vulnerable services I have developed that could be used for a real Attack/Defense CTF competition.

Acknowledgements

I would like to thank my Fortinet supervisor Axelle Apvrille for her assistance and for proposing me this topic, and Professor Antonio Lioy for his availability and support throughout my thesis work.

Contents

Chapter 1

Introduction

1.1 Computer security and CTF

Nowadays computer, smart devices (televisions, smartphones) and Internet are instruments used all over the world for different reasons (work, communication, free time) and, regardless of the reason, managed documents, software and tool used, protection of networks and computer system has a very important role in preventing the occurrence of events such as theft or disclosure of personal data, damage to software or hardware and consequent interruption of the provision of their services.

For this reasons, computer security education has become a priority, request of security experts is continuously increasing and even basic security skills are really useful in finding a job.

So, together with the rise of courses focused on security and the inclusion of security sections inside other courses, security competitions are organized to allow people interested in this field to test and improve their skills [1] [2] [3].

The most popular security competitions are Capture the Flag competitions, in which participants face some challenges, based on different vulnerabilities which touch many field of computer security (cryptography, binary analysis, reverse engineering, web) and that address different skills, or on focusing simultaneously on attacking other contestants' services trying to exploit their vulnerabilities, while they are defending their own services by patching the same vulnerabilities.

Increasing importance of this type of competition is certified by two factors:

- Increasing number of organized Capture the Flag competitions
- Increasing number of participants of all Capture the Flag competitions

Concerning the increase of number of organized CTF competitions, in the last 10 years this number has increased from about 20 competitions organized in 2011 to about 220 competitions organized in 2021.

The same evolution can be observed analysing number of participants in different editions of some CTF competitions:

If we consider DEF CON, a CTF competition held for the first time in 1996 during the homonymous conference, and which is considered as the final of CTF circuit, we can see that about 1400 teams had registered for 2020 qualification, highlighting a significant growth comparing this number with the 280 teams that had registered for 2011 qualification, and with numbers of previous editions too (in 2003 only 23 teams had registered for qualification).

Also other competitions have been characterized by continuous increasing of number of participants. For example iCTF, a CTF competition organized by UCSB (University of California, Santa Barbara) since 2003, starting from only 7 teams in its first edition, had 123 registered teams in 2013 and almost 400 registered teams in 2019 (from ctftime website [7]).

1.2 CTF overview

CTF (Capture the Flag) are a cybersecurity competitions in which competitors, grouped in teams, have to find some flags (strings usually characterized by a particular format) hidden inside files or services, by exploiting some vulnerabilities and, depending on the type of CTF competition, have also to patch their services' vulnerabilities to not let other teams steal their flags.

CTF competitions are very educational in fact, thanks to the variety and complexity of the challenges, competitors can test their teamwork, because the load and responsibility have to be shared, and each member of a team has to work with other members who have different skills (building a team with person having different skills is very important), trying to take advantage of other teammates' skills to solve challenges and learn some new skills that can be useful in the future.

Contestants can apply all their cybersecurity knowledge, test and improve their skills, not only during the competition itself, but also during training that precedes the competition, in which teams usually spend a lot of time analysing previous CTF competitions and sometimes developing new useful tools that will be used during future competitions.

CTF competitions also stimulate the development of problem-solving skills, because all contestants have to face several challenges and have to propose solutions in order to solve them [8].

There are two main types of CTF: Jeopardy and Attack/Defense.

Jeopardy CTF are characterized by no interaction between teams, there are different categories of challenges, in each category there are some challenges that differ on level of complexity, and teams have to find the single hidden flag of a challenge to obtain points. The amount of points earned when a team submit a flag depends on the complexity of the challenge.

Main challenge categories of Jeopardy CTF are:

- Forensics challenges, which are useful to train investigation skills performing an analysis of the traces of an information system to extract data
- Network challenges, in which participants have to investigate captured traffic, network services and perform packet analysis
- Reverse engineering challenges, which consist in an analysis of given programs to understand how they work and to identify deeper issues.
- Web Challenges, in which participants have to exploit a bug which impacts a website to gain some kind of higher level privilege.

- Cryptography challenges, in which participants have to exploit cryptographic algorithms, usually improperly implemented, or errors in the use of existing libraries
- Binary Exploitation challenges, in which participants usually have to find a vulnerability of a binary or an executable file (usually a Linux ELF file).

At the end of the competition the team which has the highest score is the winner.

1.3 Attack/Defense CTF

In attack/defense CTF all teams have a Virtual Machine preconfigured by competition organizers, all teams' Virtual Machines (VMs) host same services characterized by same vulnerabilities, all teams have to find these services vulnerabilities and then have two tasks:

- exploit vulnerabilities of other teams' services to steal all flags stored inside them.
- patch vulnerabilities of their services, to not let other teams steal their flags.

Thanks to that contestants can learn what means be under attack by other security experts, how to react to these attacks and how to secure vulnerable systems.

We can immediately observe that the main difference between a Jeopardy CTF and an Attack/Defense CTF is that the Attack/Defense CTF is based on interaction between teams, in fact a VM of a team is reachable by all other teams' VMs because all virtual machines are inside the same network, and all teams know IP addresses of all other teams VMs. Competition network is private and teams' VMs are not reachable from outside.

To make interaction between teams easier, teams have usually a dedicated network whose address depends on team ID, such as `10.0.X.0/24` for network of team number X, all teams have a gateway and vulnerable services at which are assigned defined IP addresses, such as `10.0.X.1` for gateway of team x and `10.0.X.3` for all vulnerable services of team X, each of these services is accessible to a different port, and same services of different teams are accessible to the same port.

An Attack/Defense CTF is divided into rounds, duration of the round is decided by organizers and, during each round, one or more new flags are inserted inside all teams vulnerable services (obviously are sent different flags to different teams). All flags have a defined lifetime (usually lifetime is equal to a defined number of rounds decided by organizers), after which they are not valid any more, so teams don't earn point if they submit them.

Insertion of flags is performed by the gameserver, which is the organizers' VM and the core of Attack/Defense CTF, it is in the same network of all teams, and has three main roles:

- As said above, it inserts one or more flags inside all services of all teams at each round.
- It controls if all teams' services are up by sending some requests to each service of each team, and it checks if flags hidden inside vulnerable services can be retrieved or not using some special credentials that teams don't have. If one or more flags of a service can't be retrieved, it means that at least one flag has been deleted or that

the service doesn't work correctly (due to a bad patch attempt or simply because service is down). In both cases, SLA score (see below) related to that team's service has to be decreased.

- It receives all submitted flags, checks their validity, computes the score of all teams and updates the scoreboard.

All packets' source IP addresses that are sent by teams and gameserver to other teams are masqueraded, in fact a team will receive packets all characterized by the same source IP address.

In this way, teams can't distinguish packets sent by gameserver, which wants to check the state of the services, from packets sent by other teams, which contain the exploit written to steal flags, so that they can't build firewall that would block packets coming from their opponents and would let pass packets sent by gameserver.

As Jeopardy CTF, a team needs to have the highest score to win the competition, but the total score of a team is computed in a different way: Scoring criteria used by almost all Attack/Defense CTF are:

- Attack points: these are the points obtained by submitting all valid flags that a team has stolen by exploiting a vulnerability of another team's service.
- SLA (Service Level Agreement) points: points that are assigned to teams that keep their services up, without them all teams would put their services down making them unreachable to not let other teams steal their flags.

Other less used criteria are:

- Defense points: These points are subtracted when a flag is stolen from one of its own vulnerable services, or are added when a team has been able to protect its flag for a certain period of time.
- King of the Hill points: Points that are assigned to teams that provide the best solution to some challenges periodically given by organizers

Usually, during Attack/Defense CTF, teams use ad-hoc tools, in particular are used tools like Destructive Farm [4], that automates the sending of exploits and flag submission (in this way teams don't have to send the exploit and submit flags in each new round manually), and tools built for traffic analysis such as Caronte [5] or Flower [6], which are very useful because using them a team can analyse what has received by other teams, can understand which type of attack has been built and which vulnerability has been exploited, and use these info to attack other teams' services and retrieve some flags.

Attack/Defense CTF are much lower than Jeopardy CTF because, due to their complex architecture which has to include several teams (sometimes more than 100 teams), they are difficult to design, implement and run. Indeed, in 2021, less than 10% of total competitions were Attack/Defense CTF [7].

In some CTF competiions, such as DEF CON, Jeopardy CTF and Attack/Defense CTF are combined because, in order to participate to the final competition, usually structured with Attack/Defense CTF format, teams have to partecipate and pass through a qualification, which is usually structured with Jeopardy CTF format.

1.4 Company and Thesis topic

Main topic of this thesis was to design and implement an architecture for an Attack/Defense CTF competition. This architecture and all vulnerable services have been developed during an internship in Fortinet in Sophia Antipolis, France. Fortinet is a multinational company which is focused on development and sale of cybersecurity products such as antivirus software, firewall and intrusion prevention systems. It is one of the most important company in this field, with more than 500000 customers all over the world.

The reason for which Fortinet has proposed this internship about Attack/Defense CTF competitions is that Fortinet runs NSE Experts Academy, which features a CTF competition that is used as a training for Fortinet employees and as a possibility to show the utility of using Fortinet products for defense.

1.5 State of the art

On internet is not easy to find architectures/frameworks which have been used in the past to host this type of competition. This is probably due to the fact that the number of this type of CTF is not very high, or to the fact that who builds and implements them doesn't want to let them be analysed and that some vulnerabilities are disclosed. Usually, after the end of a competition, organizers publish all vulnerable services details with their code, highlighting which was the vulnerability and a possible solution which describes how they could be exploited and how they could be patched, to let them solved or studied by participants who have not solved them during the competition or by someone else who didn't play the competition. In some other cases, only some parts of the infrastructure are disclosed, such as the scoreboard or, rarely, the competition checksystem. In few other cases is also possible to view details about competition infrastructure (how scoreboard was implemented or how gameserver and services interact among them). An example is the infrastructure used by Shellphish team to host the iCTF competition every year [9]. It has four main components:

- Database, which tracks the state of the game and runs on a Database VM
- Gamebot, which is responsible for advancing the competition. At each round, gamebot decides which action has to be executed by the scriptbot (set flags, retrieve flags or test services)
- Scriptbot, which executes action scheduled by the gamebot.
- Router, which is responsible for routing the traffic between teams in the competition implementing an OpenVPN service. Traffic among teams has to be anonymized to not allow teams distinguish between traffic generated by scriptbot and traffic generated by teams

Together with infrastructure components description are provided some instructions that has to be followed to start an A/D CTF competition.

Another available infrastructure is the one used to run the InCTF Attack/Defense competition organized by Amrita University and Amrita Centre for Cybersecurity Systems and Networks. It is based on the iCTF framework but, unlike the latter, it is an infrastructure that uses Docker containers instead of VMs. To do that, four new components are introduced:

- Container image registry, which provides an easy and secure means to distribute and manage container images
- Service containers host, which is a compute server that run all service containers
- Exploit containers host, which is a compute server that run all exploit containers, useful if organizers decide that teams can't run exploits on their own but that they themselves have to execute the exploits on behalf of the teams (not the best solution because if organizers end up running the exploits, the teams cannot debug exploits if they fail).
- Flag storage volume, which is used to store the database file containing flags to avoid issues related to deletion and recreation of a container which happen when a container has to be updated. In addition, gamebot is modified to synchronize the containers periodically with their corresponding image stored in image registry, and the part concerning VMs creation, which is modified to create containers

The best quality of this architecture is that the use of Docker containers reduces the resource requirements, so it simplifies organizing CTF competitions and scaling them to several teams, while its worst defect is that teams cannot capture exploits from the network and reverse enginner them to identify new vulnerabilities [10] [11]. In this way a fundamental part of Attack/Defense is missing, and this is the reason for which I decided not to use this architecture, together with the other reasons listed below.

There is another example of available A/D CTF infrastructure, and is the "ForcAD" infrastructure [12], developed by RocketClass team, which provides detailed instruction useful about how to use it to run a A/D competition, but that doesn't provide many details about infrastructure description.

After an analysis on these infrastructures, I understood that it was better to use them just to learn what were all different aspects that had to take care of and I decided to create a new infrastructure from scratch, for two different reasons:

- it would have been easier than use an already existent infrastructure, understanding in depth all its characteristics and uses, and then adapting it to my network and my vulnerable services.
- A deeper and more accurate analysis of these infrastructures and their characteristics would have required a huge amount of time, because it implied analysis of hundreds or even thousands of lines of code, usually grouped in specific libraries that teams defined for this project specially.

Chapter 2

Architecture

2.1 Network architecture and FortiPoC

In this chapter I describe how I built a network for an Attack/Defense CTF competition.

First step was to decide which tool I had to use to build the network for Attack/Defense CTF competition. I decided to use a tool named FortiPoc to create the architecture which had to host an Attack/Defense CTF, because FortiPoC is an internal Fortinet tool. FortiPoc is used to create simple or complex network topology using the different Fortinet products. A FortiPoC user has to define the required network architecture and the devices that will be connected into this network configuring their network interfaces too. Once the user executes its setup, FortiPoC automatically manages to create all these items. Without FortiPoC the user would have to define and manually create everything in a private or public cloud service. FortiPoC also saves the created environment in a file called "poc" file, which can be reloaded to quickly rebuild a previous environment.

Anyway there were some other interesting tools which I could use to create a network for Attack/Defense CTF competition:

- LogMeIn Hamachi, which is a VPN application developed and released in 2004 by LogMeIn company, which is able to establish direct links between computers that are behind network address translation firewalls without requiring reconfiguration. In this way, computers that are connected over the Internet will interact as if they are connected over a LAN
- NetOverNet, free alternative to Hamachi, with which is possible to create a virtual private network where all IP addresses are universally accessible and permanent
- SoftEther VPN, a free open-source multi-protocol VPN client and VPN server software released in January 2014 by University of Tsukuba's students
- ZeroTier, developed by homonymous company which provides web console for network management and endpoint software for the clients

Once decided to use FortiPoc, as next step, I needed a Virtual Private Server (VPS), which is a form of multi-tenant cloud hosting in which virtualized server resources are made available to an end user over the internet via cloud or hosting provider. So it was necessary to figure out which cloud platform I was going to use. Nowadays most used platforms are:

- Microsoft Azure, a cloud computing service for application management via Microsoft-managed data centers released by Microsoft in February 2010
- Amazon Web Service (AWS), a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs. Launched in 2002, it has the biggest market share for cloud infrastructure in 2022
- Google Cloud Platform, a suite of cloud computing services offered by Google launched on April 2008 which runs on the same infrastructure that Google itself uses internally for its end-user products (such as Google Search, YouTube, Gmail etc).

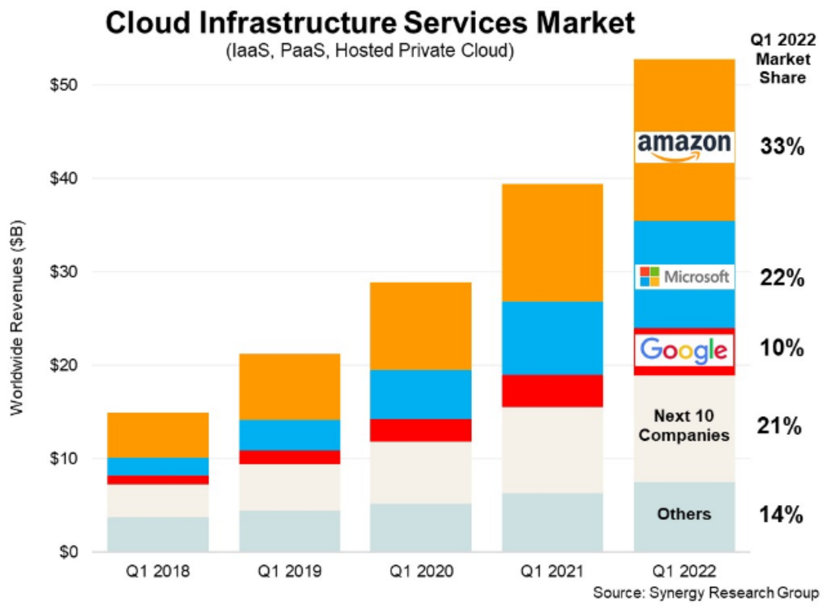


Figure 2.1: Cloud Infrastructure Services Market

Among them I decided to choose Google Cloud Platform, because it is the VPS used at Fortinet when someone wants to build a network using FortiPoc. In this way the network architecture of the game is built entirely in a single virtual machine hosted in the Google Cloud Platform (GCP) and is defined a Google Compute Instance which runs the FortiPoC operating system. Without GCP we would have to define ourself the networks and the VM inside GCP. The compute instance is assigned with a private, non-routable IP address allocated from the Google Private IP address space. In order to access the FortiPoC administrative console, Google automatically assigns a public routed IP address that is mapped to the compute instance's private IP address. Once the public IP address is accessed, Google performs a network address translation (NAT), so it replaces the public IP with the mapped private IP. Because this operation is done on the destination IP address it is called destination network address translation (DNAT).

When we create a new instance, we have to define some important characteristics that will have a great impact on architecture performances, such as number of CPUs and memory. When I defined this instance, I selected a memory of 32 GB with 8 CPUs. I decided not to select too high parameters, because I started implementing my architecture as if there were only few teams' VMs inside it. If this architecture was used for a Attack/Defense demo or for a real Attack/Defense CTF, this values should be significantly increased, because we would need a lot more power to support an architecture with one hundred (or more) teams. Other important features that are defined at instantiation time are networking features, like type of allowed traffic (I chose both http and https traffic), IP addresses and ports that can access this instance and features about boot disk (here I selected FortiPoC image I wanted to use).

2.2 Networks and Devices

After I created the instance, I decided to implement an architecture with only 5 teams virtual machines, then I started defining all networks and devices. First of all I decided the criteria with which to assign IP addresses to all networks, considering that they had to be easily remembered by all contestants, then I implemented following networks:

- One /24 network for each team, with IP address `10.0.teamId.0` (teamId>0)
- One /24 network for gameserver, with IP address `10.0.254.0`
- One /24 "Internet" network, with IP address `10.0.0.0`, to enable Internet connection for all devices

All these networks were connected to a central VyOS router. VyOS is a Linux-based network operating system that provides software-based network routing, firewall and VPN functionality.

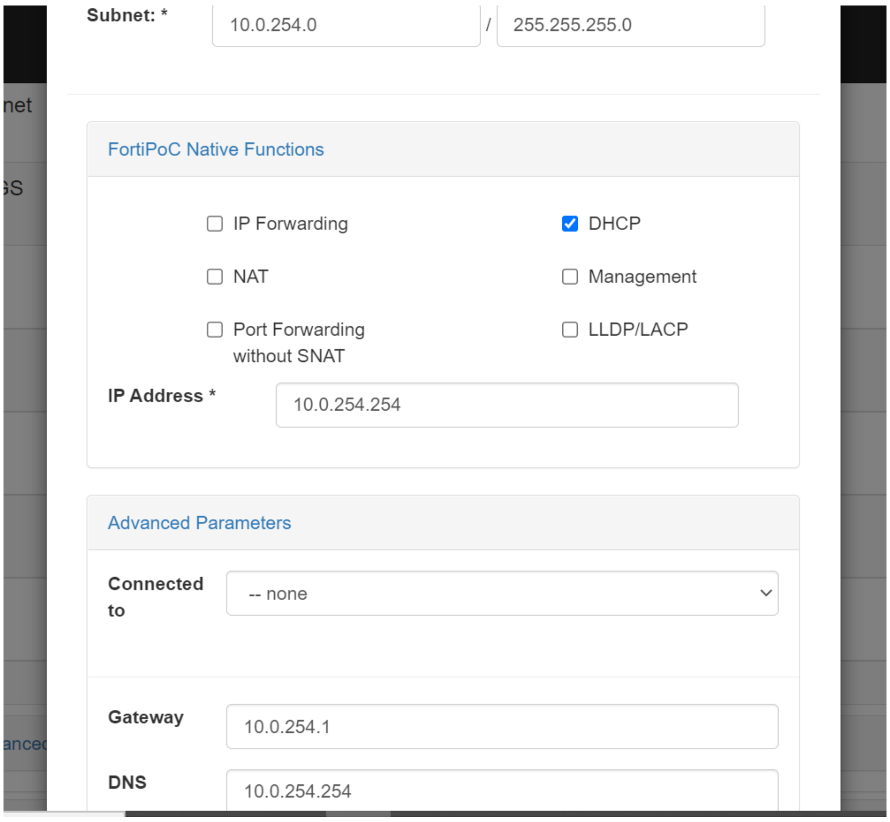
Inside each of these networks, two IP addresses were defined:

- `10.0.teamId.2` (or `10.0.254.2` inside gameserver network) is team (or gameserver) IP address
- `10.0.teamId.1` (or `10.0.254.1` inside gameserver network) is router IP address

Each time I defined a network inside FortiPoC, I had to define DNS, Gateway, and FortiPoC Native Functions:

- For all networks, I selected .254 IP address as DNS.
- For all teams networks and for gameserver networks I selected .1 IP address as gateway (so central router was the gateway for these networks, in this way all internet traffic coming and directed to all other networks would pass through central router and then would reach its destination) while I selected .254 IP address as gateway for "Internet" network.
- For "Internet" network I selected IP forwarding and NAT from FortiPoC native function list, while I selected DHCP for all teams networks and for gameserver network.

I made this choice about DHCP because FortiPoC didn't support yet type of Linux device I instantiated and could not provide their IP address. So I used DHCP protocol to control the IP addresses that had be assigned to the network interfaces of these devices. VyOS router instead is a device type that FortiPoC supports for IP addresses provisioning.



Subnet: * 10.0.254.0 / 255.255.255.0

FortiPoC Native Functions

IP Forwarding
[✓ DHCP]
NAT
Management
Port Forwarding without SNAT
LLDP/LACP

IP Address * 10.0.254.254

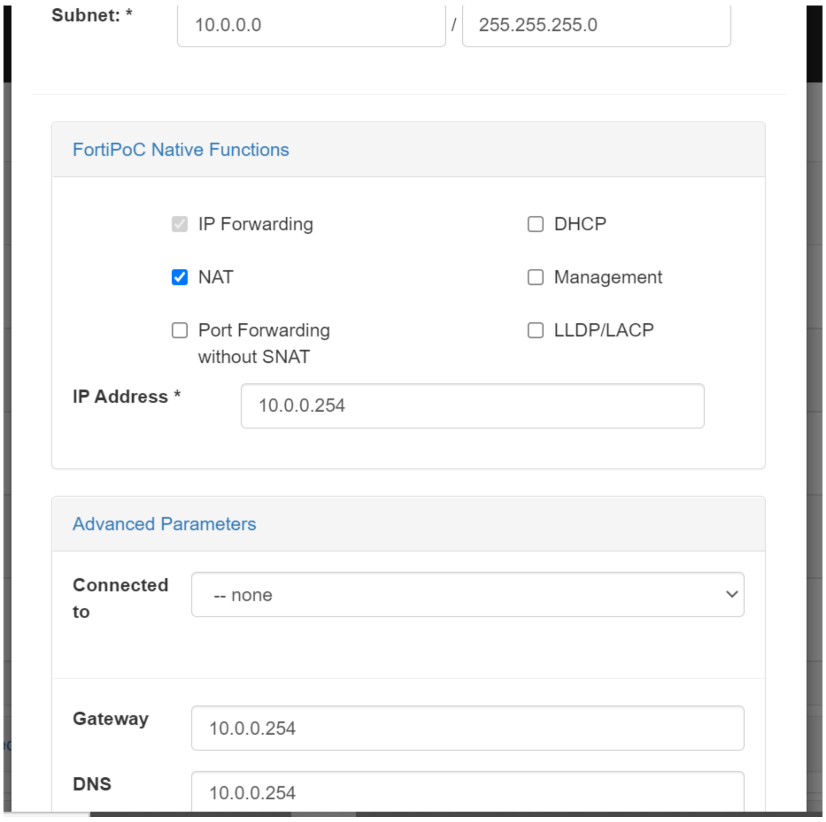
Advanced Parameters

Connected to -- none

Gateway 10.0.254.1

DNS 10.0.254.254

Figure 2.2: Gameserver network configuration



Subnet: * 10.0.0.0 / 255.255.255.0

FortiPoC Native Functions

[✓ IP Forwarding]
DHCP
[✓ NAT]
Management
Port Forwarding without SNAT
LLDP/LACP

IP Address * 10.0.0.254

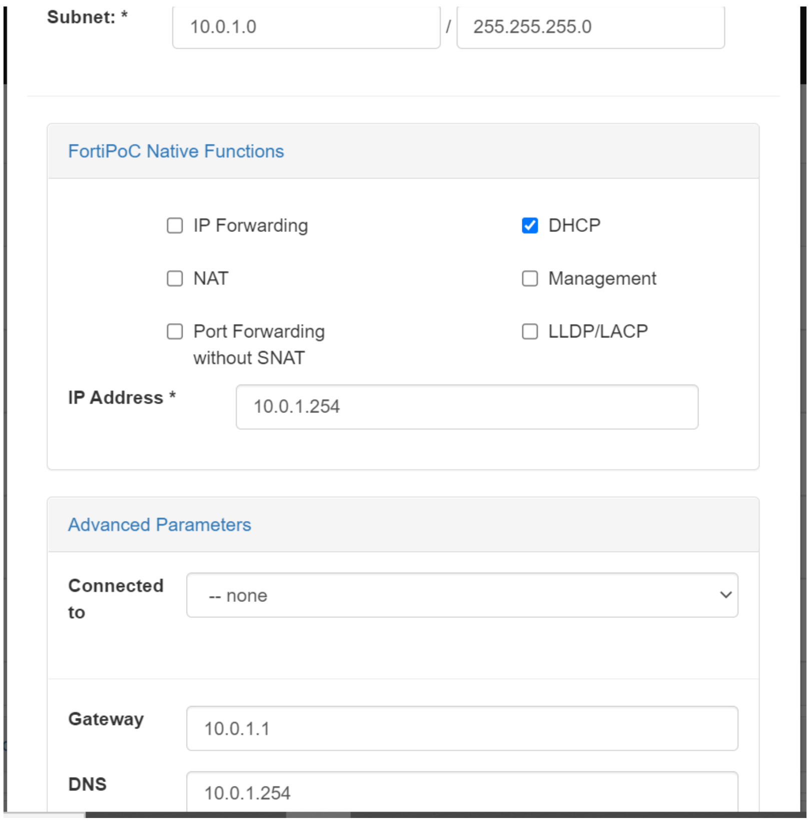
Advanced Parameters

Connected to -- none

Gateway 10.0.0.254

DNS 10.0.0.254

Figure 2.3: “Internet” network configuration



Subnet: *	10.0.1.0 / 255.255.255.0	

FortiPoC Native Functions

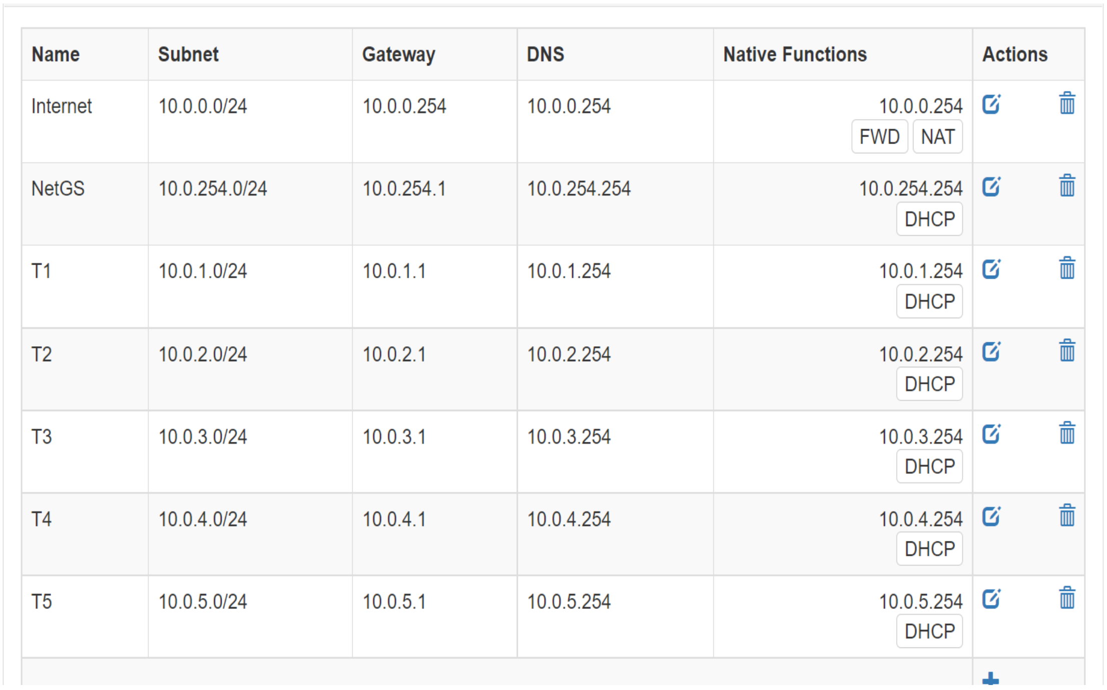
- [ ] IP Forwarding
- [x] DHCP
- [ ] NAT
- [ ] Management
- [ ] Port Forwarding without SNAT
- [ ] LLDP/LACP

IP Address *	10.0.1.254

Advanced Parameters

Connected to	-- none
Gateway	10.0.1.1
DNS	10.0.1.254

Figure 2.4: Team 1 network configuration

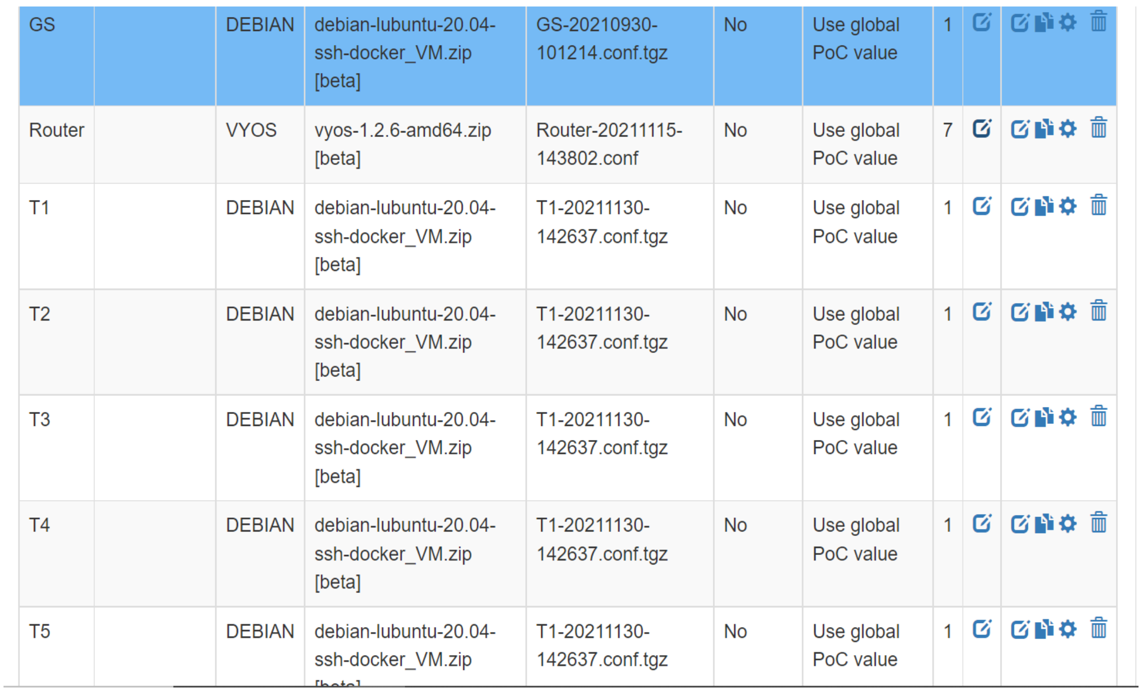


Name	Subnet	Gateway	DNS	Native Functions	Actions
Internet	10.0.0.0/24	10.0.0.254	10.0.0.254	10.0.0.254 FWD NAT	
NetGS	10.0.254.0/24	10.0.254.1	10.0.254.254	10.0.254.254 DHCP	
T1	10.0.1.0/24	10.0.1.1	10.0.1.254	10.0.1.254 DHCP	
T2	10.0.2.0/24	10.0.2.1	10.0.2.254	10.0.2.254 DHCP	
T3	10.0.3.0/24	10.0.3.1	10.0.3.254	10.0.3.254 DHCP	
T4	10.0.4.0/24	10.0.4.1	10.0.4.254	10.0.4.254 DHCP	
T5	10.0.5.0/24	10.0.5.1	10.0.5.254	10.0.5.254 DHCP	

Figure 2.5: All networks configuration

Once implemented all networks, I created all devices: One device for each team, one device for gameserver and another one for central router. For each of them I had to define some parameters:

- Name
- Image (I selected debian-lubuntu with docker for all devices except for central router, for which I selected VyOS image)
- Network parameters, for each device I selected network parameters of their own network (for central router I used "Internet" network parameters)
- Advanced parameters such as memory and CPU cores



GS		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	GS-20210930-101214.conf.tgz	No	Use global PoC value	1		
Router		VYOS	vyos-1.2.6-amd64.zip [beta]	Router-20211115-143802.conf	No	Use global PoC value	7		
T1		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T2		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T3		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T4		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T5		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		

Figure 2.6: All devices configuration

After all these steps, this was the final network

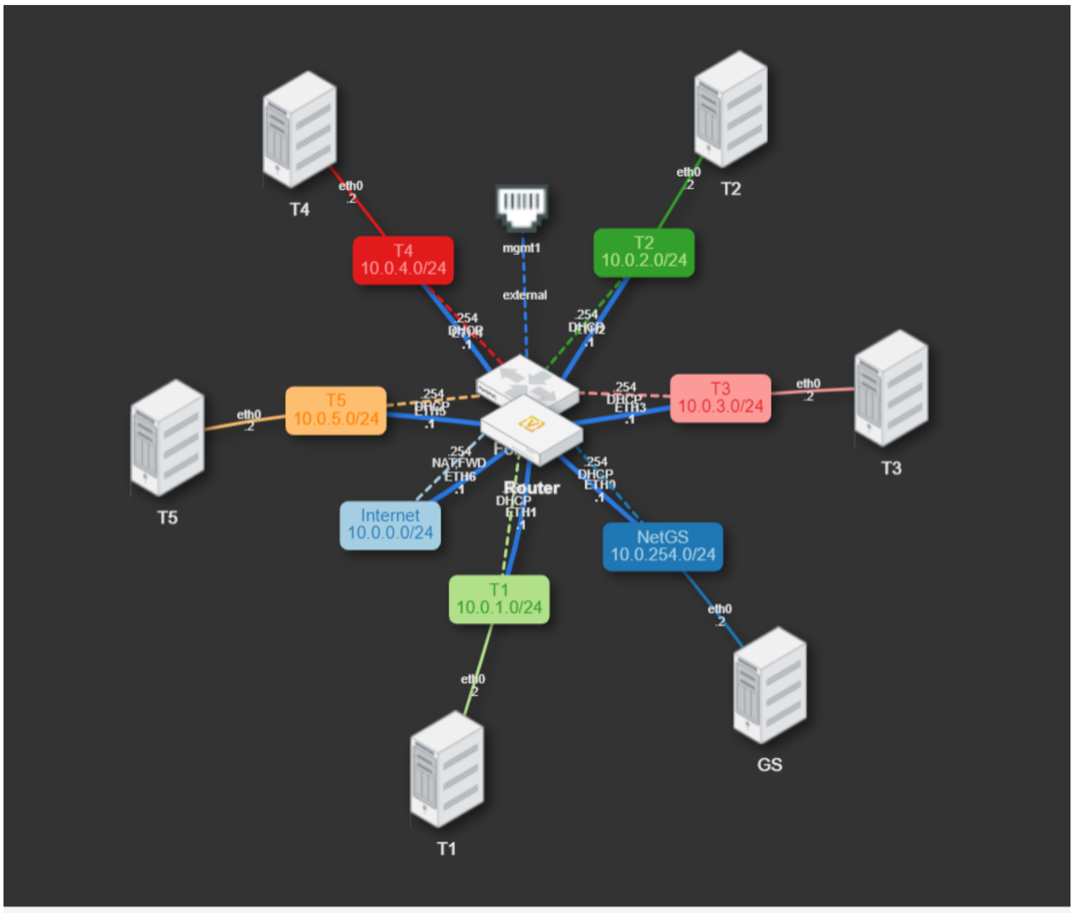
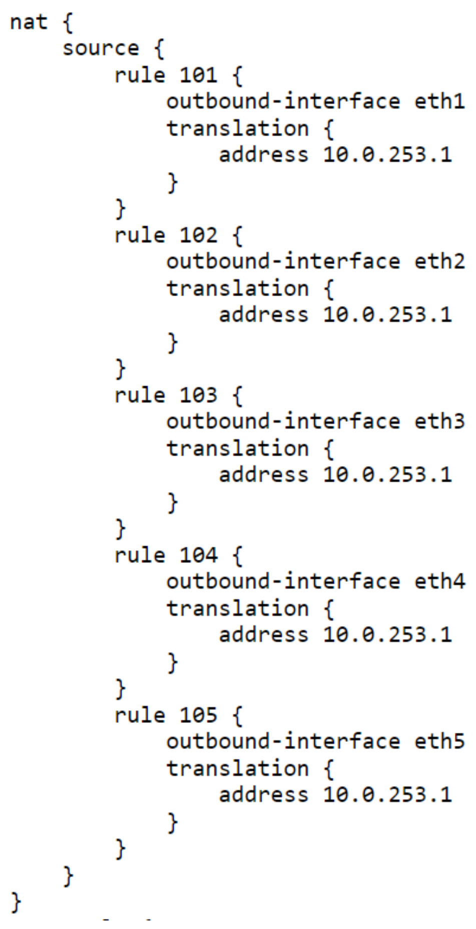


Figure 2.7: Final Network

Last step was implementing IP masquerading inside central VyOS router: what I wanted was that when a team virtual machine received packets from another VM (gameserver or another team VM), it shouldn’t be able to understand who sent them, while gameserver should be able to fogure out which team sent packets when it received them. So, when gameserver or a team sent a packet to another team, source IP address had to be changed by central router to 10.0.253.2.

To do that, I modified router configuration file using following commands:

- `configure` to enter inside configuration mode
- `set nat source rule 101 translation address ’10.0.253.1` and `set nat source rule 101 outbound-interface ’eth1’` taking care of rule number and interface
- `commit` and `save` to save changes inside configuration file



```
nat {
    source {
        rule 101 {
            outbound-interface eth1
            translation {
                address 10.0.253.1
            }
        }
        rule 102 {
            outbound-interface eth2
            translation {
                address 10.0.253.1
            }
        }
        rule 103 {
            outbound-interface eth3
            translation {
                address 10.0.253.1
            }
        }
        rule 104 {
            outbound-interface eth4
            translation {
                address 10.0.253.1
            }
        }
        rule 105 {
            outbound-interface eth5
            translation {
                address 10.0.253.1
            }
        }
    }
}
```

Figure 2.8: IP masquerading rules in router configuration file

2.3 Access to virtual machines

Teams can access their virtual machines terminal through ssh connection at instance IP address with port number equal to 11001+team_id and can even have access VM remote desktop by connecting in display mode using same IP address but port number 15001+team_id (to do it I used VNC viewer). Passwords are random strings that have to be provided to teams before competition starts, and then can be modified by teams itself.

Chapter 3

Vulnerable services

3.1 General information about services

In this chapter I describe vulnerable services I implemented for Attack/Defense CTF competition. In particular I describe:

- what are services structure and purpose
- how services have been designed
- what are services vulnerabilities
- how services vulnerabilities can be exploited and patched

I will talk about how flags are stored inside them and checked (for SLA points) in next chapter which is focused on gameserver.

The only aspect that all services have in common is that I used Docker to run them all. Docker is an open platform that enables to package and run an application in an isolated environment called container.

To do that, I wrote a DockerFile for each service, which contained instructions about how to build the Docker container image and automated the process of Docker image creation. A Docker image is a read-only template with instructions for creating a Docker container, which is a running instance of a Docker image.

A tool that can be used to speed up service running process is docker-compose. Normally, to run a container, we should use two instructions, one to build Docker image (`docker build`) and one to run the image (`docker run`) but, using a "docker-compose.yml" file, a container (or more than one) can be run with a single command (`docker-compose up`), in background (using "-d" option) and image can be built each time before container starts again (using "–build", very useful if we want to modify our container without put it down).

3.2 First service: inf-pump

Inf-pump is a very basic vulnerable service that I designed in order to let Attack/Defense CTF beginners understand how this type of competition works.

3.2.1 Structure

Inside inf-pump Docker container run two processes: an ssh process, used for communication between gameserver and inf-pump container (with a random ssh password defined inside inf-pump Dockerfile, different from inf-pump passwords of other teams, and known only by team and gameserver), and a nginx process, used to expose html inf-pump page. It is generally recommended to run only one service for each container, but two processes can run inside a single container using a process manager like supervisord. Once I start it inside Dockerfile, it manages all processes.

Main parts of inf-pump service are:

- an `index.html` file which replicates an infusion pump screen
- a `inf-pump.js` javascript file which contains functions used to manage all infusion pump buttons
- a `FileFlag.txt` text file containing all valid flags received from gameserver, it is automatically created when gameserver starts sending flags

Inf-pump html page is visible at teams' IP address on port 3001, and inside it an user can select medicine to be administrated (from default medicines list that includes penicillin, insulin, heparin and paracetamol), administration rate (15, 30, 45 or 60 minutes), and volume to be administered (in ml). When administration starts, our choice is displayed on inf-pump screen with medicine volume already administrated. Page provides new administration availability when previous administration is finished or stopped.

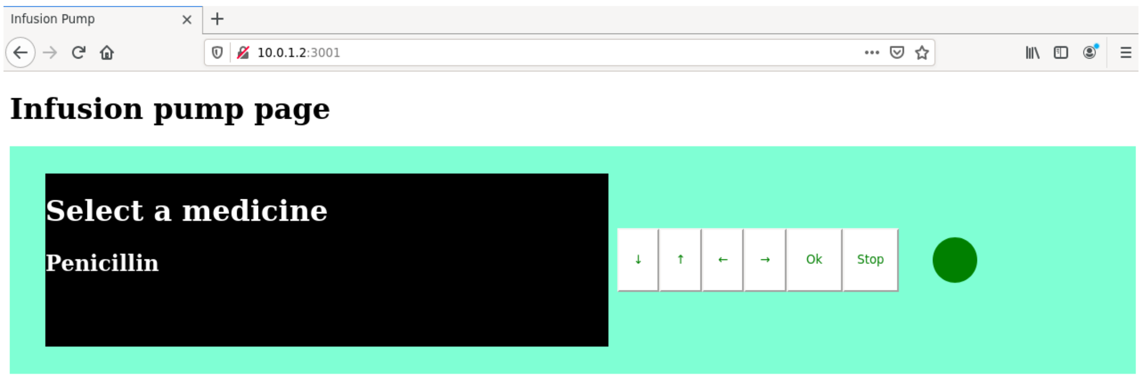
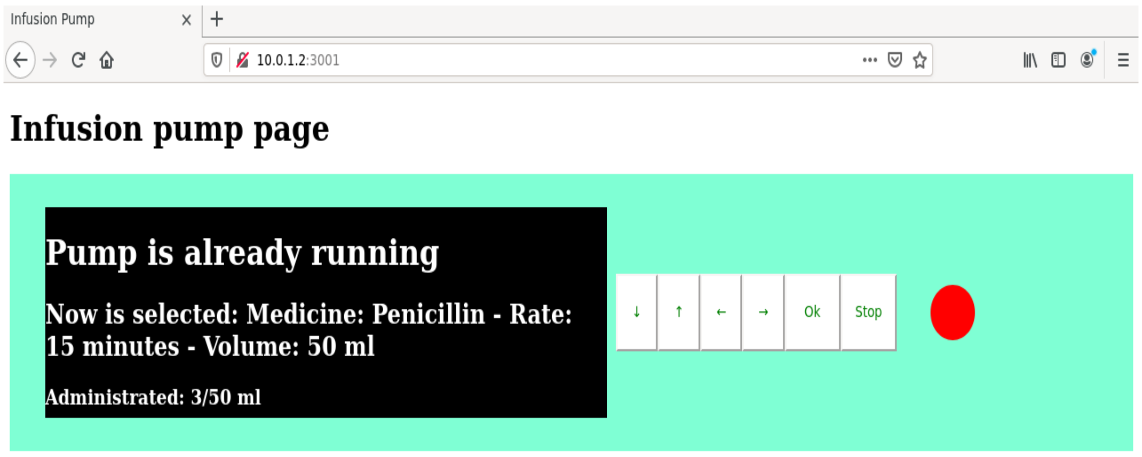


Figure 3.1: Infusion pump ready for selection



Infusion pump page

Pump is already running

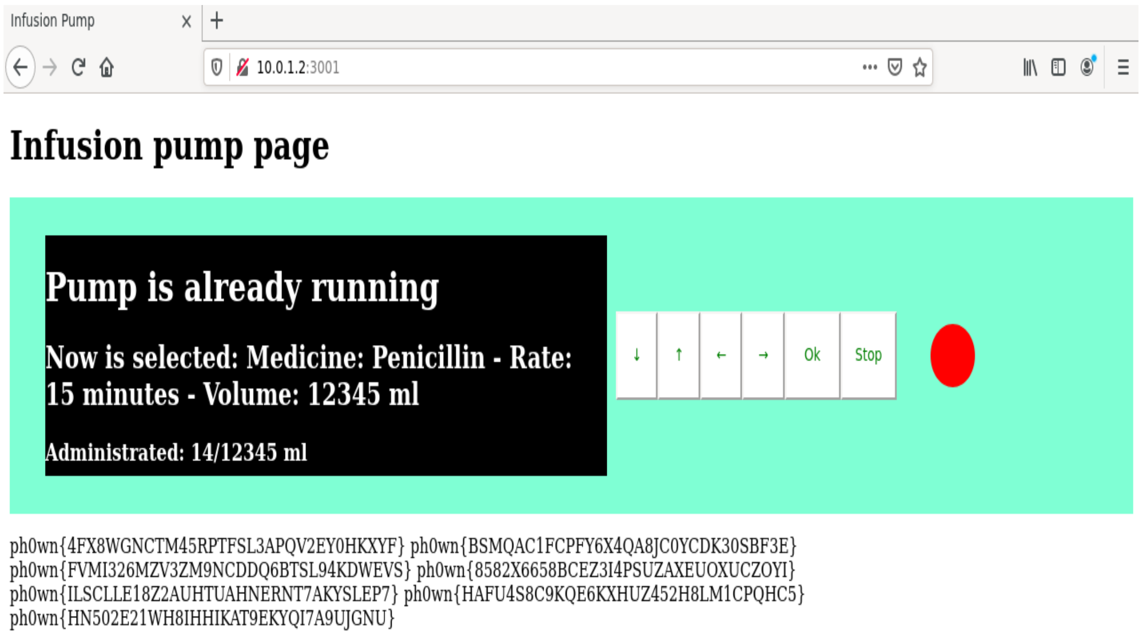
Now is selected: Medicine: Penicillin - Rate: 15 minutes - Volume: 50 ml

Administrated: 3/50 ml

Figure 3.2: Running infusion pump

3.2.2 Vulnerability

The vulnerability that can be exploited to obtain flags stored inside text file is in "volume_selection" function of "inf-pump" javascript file which manages volume selection. Thanks to this vulnerability, if I insert a value bigger than 10000 ml, file containing flags is opened and all flags are read and then shown on html inf-pump page.



Infusion pump page

Pump is already running

Now is selected: Medicine: Penicillin - Rate: 15 minutes - Volume: 12345 ml

Administrated: 14/12345 ml

```
ph0wn{4FX8WGNCTM45RPTFSL3APQV2EY0HKXYF} ph0wn{BSMQAC1FCPFY6X4QA8JC0YCDK30SBF3E}
ph0wn{FVMI326MZV3ZM9NCDDQ6BTSL94KDWEVS} ph0wn{8582X6658BCEZ3I4PSUZAXEUOXUCZOYI}
ph0wn{ILSCLLE18Z2AUHTUAHNERNT7AKYSLEP7} ph0wn{HAFU4S8C9KQE6KXHUZ452H8LM1CPQHC5}
ph0wn{HN502E21WH8IHHIKAT9EKYQI7A9UJGNU}
```

Figure 3.3: Infusion pump service exploited

3.2.3 Patch

Vulnerability can be patched in different ways, for example I can delete javascript code used to read file containing flags when a volume bigger than 10000 ml is inserted, or I can set “maxlength” attribute of volume input field equal to 4, so that values bigger than 9999 can’t be inserted any more.

3.3 Second service: testify

Testify is a service in which users can register and then book appointments with a doctor. I imported this service from Bambi CTF#6 which I played, in order to understand how Attack/Defense CTF vulnerable services are built before start implementing one on my own. I modified couple of things inside this service, while I have modified a lot the code used for service management performed by gameserver, but I will talk about it in gameserver chapter.

3.3.1 Structure

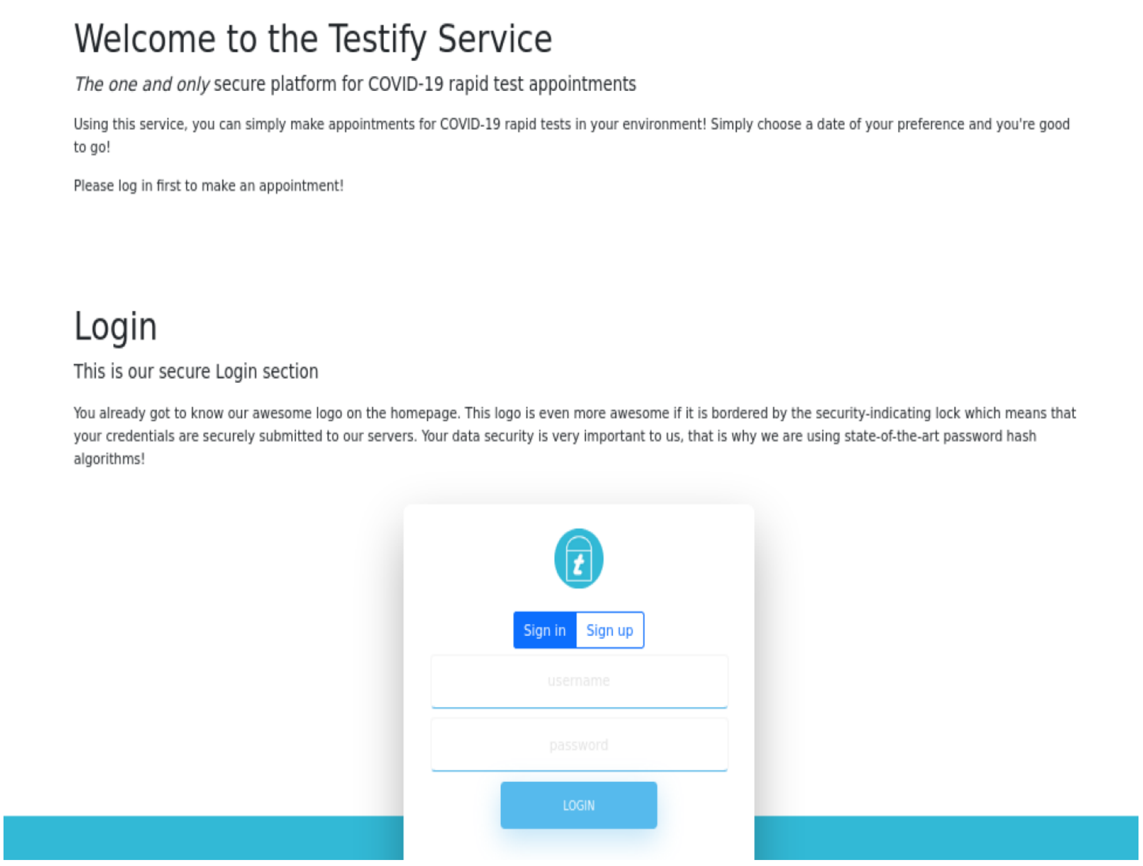
This service consists in two different processes, that run in two different Docker containers.
First one is a python application in which users can register and then login to book an appointment with doctors by inserting their credentials, date and time of the appointment, doctor whom they want to make an appointment with and, optionally, some extra info useful for doctors (and visible by doctors only). App is accessible at teams’ IP address at port 8000.

Second one is a MYSQL database which contains three tables:

- “users” table, in which are stored informations about doctors and users, by default it contains 5 “doctor” users (these are the only users having “is_doctor” field set to 1).
- “appointments” table, in which are stored all info about appointments
- “sessions” table, in which are stored all info about sessions

It is accessible on port 3306, which is the default port for MYSQL protocol.
In this service flags are stored in two different ways:

- inside a `username-info.txt` file which is created automatically when an user makes an appointment, all these files are stored inside an “userdata” folder
- inside extra notes that user provides to doctors, and visible by all users that application recognizes as doctors



Welcome to the Testify Service

The one and only secure platform for COVID-19 rapid test appointments

Using this service, you can simply make appointments for COVID-19 rapid tests in your environment! Simply choose a date of your preference and you're good to go!

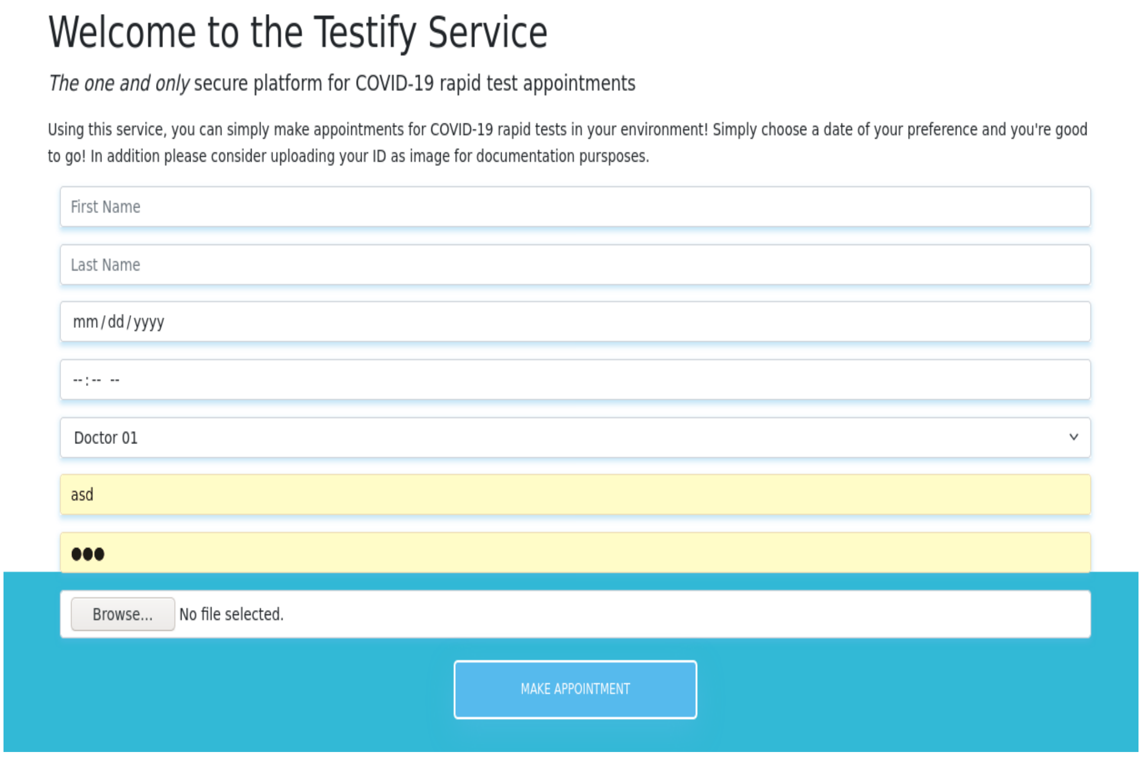
Please log in first to make an appointment!

Login

This is our secure Login section

You already got to know our awesome logo on the homepage. This logo is even more awesome if it is bordered by the security-indicating lock which means that your credentials are securely submitted to our servers. Your data security is very important to us, that is why we are using state-of-the-art password hash algorithms!

Figure 3.4: Testify Register/Login page



Welcome to the Testify Service

The one and only secure platform for COVID-19 rapid test appointments

Using this service, you can simply make appointments for COVID-19 rapid tests in your environment! Simply choose a date of your preference and you're good to go! In addition please consider uploading your ID as image for documentation pursposes.

First Name
Last Name
mm/dd/yyyy
--:-- --
Doctor 01
asd
●●●
Browse... No file selected.

Figure 3.5: Make Appointments page

3.3.2 Vulnerabilities

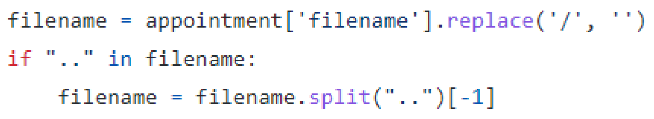
This service has two vulnerabilities, and each one can be exploited to retrieve flags stored inside one of positions listed above.

First vulnerability consists in understanding that all files belonging to users created by gameserver (and that are files contain flags) are named as `username-info.txt` (with username automatically appended every time) and that gameserver uses same usernames for all teams services (usernames can be retrieved from MYSQL database or from files stored inside “userdata” folder). Then attacker has to make appointments creating files named as users files generated by gameserver containing flags. This is possible thanks to a “split” and a “replace” method that allow to replicate files names (see image below) by providing filenames built as `..victim-username-info.txt`. In this way attacker’s username will be appended but then removed, and last part containing only victim’s username will be used as name of new file every time. At this point service returns appointment id linked with file, and the attacker can retrieve file containing flags using them.

Second vulnerability consists in understanding how application recognizes if an user is a doctor or not. To do that, application checks:

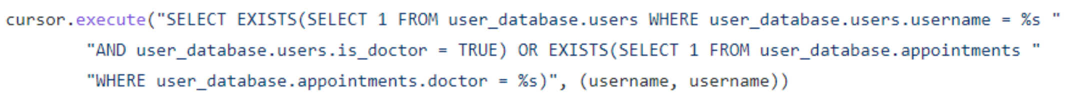
- if inside “appointments” table there is at least one appointments in which the user is indicated as doctor
- if user has “is_doctor” field = equal to 1

So an attacker first has to make an appointment indicating itself as doctor, and then can retrieve all flags stored inside users extra notes.



```
filename = appointment['filename'].replace('/', '')
if ".." in filename:
    filename = filename.split("..")[-1]
```

Figure 3.6: First vulnerability



```
cursor.execute("SELECT EXISTS(SELECT 1 FROM user_database.users WHERE user_database.users.username = %s "
        "AND user_database.users.is_doctor = TRUE) OR EXISTS(SELECT 1 FROM user_database.appointments "
        "WHERE user_database.appointments.doctor = %s)", (username, username))
```

Figure 3.7: Second vulnerability

3.3.3 Patches

To patch first vulnerability, code which allows file name manipulation has to be removed while, to patch second vulnerability, only default “doctor” users (only users having “is_doctor” field set to 1) have to be recognized as doctors (second part of MYSQL query shown above has to be removed)

3.4 Third service: postit

Postit is a service in which registered users can login and post some notes. As for testify service I imported this service from Bambi CTF#6.

3.4.1 Structure

This service consists in a single process, which runs in a Docker container reachable at team IP address on port 9337. Once connected, different actions can be performed:

- registration of a new user by providing an username and RSA public keys (exponent and modulus)
- login, by signing a random challenge given by service using user private key
- creation of a new post with a message inside it, or listing all user posts. User has to be logged to do it
- see list of all registered users, or see public keys of a specific user by providing its username. Be logged is not necessary

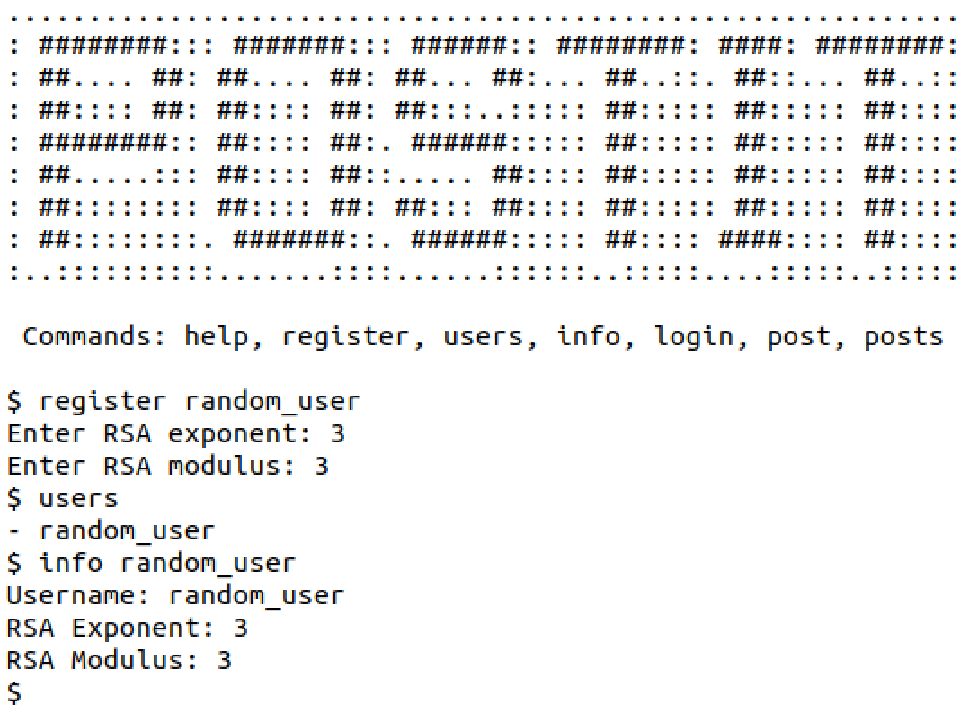
All informations are stored inside a sqlite3 database.

3.4.2 Vulnerability

Vulnerability consists in possibility of forging signature for short challenges (default challenge size is 16 bytes), due to the fact that gameserver uses value 3 for all users it generates.

3.4.3 Patch

To avoid fake signs, size of random challenge, given to user by service, has to be increased



```
...........................................................
: ########::: #######::: ######:: ########: ####: ########:
: ##.... ##: ##.... ##: ##... ##:... ##..::. ##::... ##..::
: ##:::: ##: ##:::: ##: ##:::..::::: ##::::: ##::::: ##::::
: ########:: ##:::: ##:. ######::::: ##::::: ##::::: ##::::
: ##.....::: ##:::: ##::..... ##:::: ##::::: ##::::: ##::::
: ##:::::::: ##:::: ##: ##::: ##:::: ##::::: ##::::: ##::::
: ##::::::::. #######::. ######::::: ##:::: ####:::: ##::::
:..:::::::::.......::::......::::::..::::::....::::::..:::::

 Commands: help, register, users, info, login, post, posts

$ register random_user
Enter RSA exponent: 3
Enter RSA modulus: 3
$ users
- random_user
$ info random_user
Username: random_user
RSA Exponent: 3
RSA Modulus: 3
$
```

Figure 3.8: Postit interface

3.5 Fourth service: spring

This service contains a Spring Boot application. Spring Framework is a Java platform that provides infrastructure support for developing Java applications, while Spring boot is an open-source framework which provides Java developers a platform to get started with an auto configurable Spring application. With it, developers can simply run their Spring based application without losing time on preparing and configuring it.

Inside Spring Docker container run two processes: an ssh process, used for communication between gameserver and spring container like inf-pump service, and a Spring Boot java application. As for inf-pump service, I used a supervisord to run both processes inside same Docker container, and flags sent by gameserver are stored inside a text file. Spring Boot application can be accessed at teams' IP address on port 8080.

3.5.1 Vulnerability

This Spring Boot web application has a Log4Shell vulnerability (CVE-2021-44228). Log4Shell was a vulnerability of Log4j Apache Java framework, used for logging messages in application.

Log4shell was publicly disclosed on 9 December 2021, and Apache gave it a CVSS (Common Vulnerability Scoring System, standard for security vulnerabilities evaluation) severity rating of 10, which is the maximum possible score. It has been defined as the largest vulnerability ever by many experts, due to the fact that exploit is simple to execute and is estimated to affect hundreds of millions of devices.

In particular, the problem was the package "JNDILookup plugin". JNDI stands for Java Naming and Directory Interface, and it provides an API to interact with directory services, such as LDAP. Using it a Java application can find and invoke a Business Object from an LDAP server running on the same machine or on a remote machine. This means that if an attacker can control the LDAP url, he can cause a Java program to instantiate a class from a LDAP server under their control, and he can cause the application to load and execute arbitrary Java code.

To exploit this vulnerable service, it should be forced to connect to an LDAP server controlled by the attacker itself, and malicious code has to be injected. For example, an attacker can use netcat in listener mode waiting for a connection on a specific port, and then can inject code to push file containing flags from the vulnerable service to itself.

3.5.2 Patch

To patch this vulnerability can be added a constraint about log4j version library, which has to force the use of a log4j version in which this vulnerability has been patched.

Chapter 4

Gameserver

4.1 Gameserver tasks

In this chapter I describe which are the tasks of gameserver (GS), that is CTF organizers VM.

Gameserver has to take care of all tasks required for CTF running and management:

- Provide a scoreboard showing all teams overall scores, number of flags stolen by a team from other teams' services and all services SLA score
- Generate a new flag for each service at each round and store it both on gameserver side and on teams' services side
- Check if all teams' vulnerable services are up and if all valid flags are available inside them
- Compute SLA points coherently with flags availability
- Manage flags submission

In following paragraphs I describe how all this tasks are performed.

4.2 Scoreboard

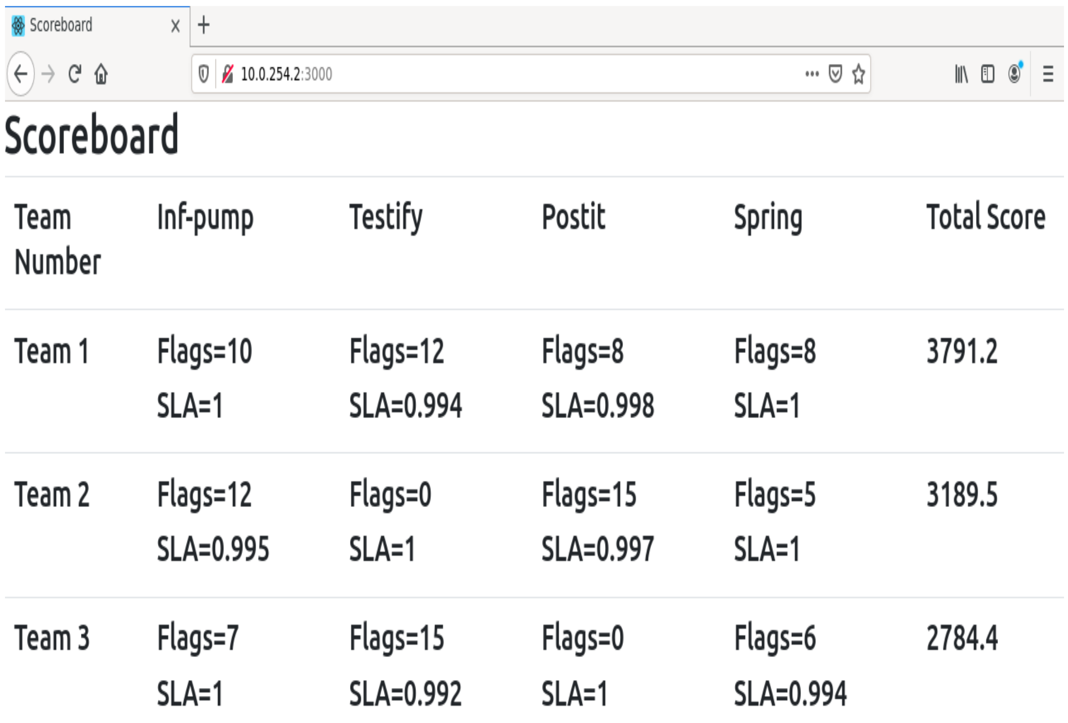
To set up a scoreboard I decided to implement a node js web server, which is divided into server part and client part. In client part I defined scoreboard html page layout and I implemented requests that client sends to gameserver through APIs each time it wants to see all teams scores on scoreboard page. In server part I implemented clients requests management and I saved database `teams.db` in which are stored all information about teams scores.

Scoreboard page is available at gameserver IP address `10.0.254.2` on port 3000, and it shows a table which has a row for each team, a column for teams' names, a column for each service implemented inside CTF, and a column for teams overall scores. For each service, teams can see number of flags stolen from that service of other teams and SLA scores of their services.

To compute overall scores, first is computed partial score of each service, using SLA points of that service and the number of flags stolen from that service of other teams, accordingly to the formula

$$Score = nflags * 100 * SLA$$

. Then all services partial scores are sum to obtain team overall score.



Scoreboard

10.0.254.2:3000

Scoreboard

Team Number	Inf-pump	Testify	Postit	Spring	Total Score
Team 1	Flags=10 SLA=1	Flags=12 SLA=0.994	Flags=8 SLA=0.998	Flags=8 SLA=1	3791.2
Team 2	Flags=12 SLA=0.995	Flags=0 SLA=1	Flags=15 SLA=0.997	Flags=5 SLA=1	3189.5
Team 3	Flags=7 SLA=1	Flags=15 SLA=0.992	Flags=0 SLA=1	Flags=6 SLA=0.994	2784.4

Figure 4.1: Scoreboard page

4.3 Generation and storage of new flags

To have a simple structure for management, I decided to create a "service name" folder for each vulnerable service and I used same technique for all vulnerable services to generate flags and save them on gameserver side. Significant aspect that differentiates services is how I decided to save flags on vulnerable services side, because it depends on how services work.

Inside each of "service name" folders I wrote a python program, named `servicename_generate_and_test_flags.py`, which, at each round (round duration can be defined inside program, but it has to be the same for all services), starts a new thread that generates a new flag and saves it both on gameserver side and on services side. I defined flag as a "ph0wn" string concatenated to a 32 uppercase random characters string enclosed in curly brackets, to make it easily recognizable.

On gameserver side, python programs create, inside their "service name" folders, one text file `TeamX_flag_and_info.txt` for each team, containing all valid flags that should be stored and available inside that team's vulnerable services. I named these text files in this way because, for testify and postit services, I saved also some information about users that are useful for flags checking.

Flags are also saved inside other files, named `FlagsValidX.txt` but, inside these files, are saved flags generated for other teams and that are valid in case of submission (e.g. inside `FlagsValid1.txt` are inserted all flags generated for team2, team3, ... services, inside `FlagsValid2.txt` are inserted all flags generated for team1, team3, ... services etc). These programs also keep only last X flags inside all text files, where X is flags lifetime expressed in number of rounds.

On services side, I used different techniques to save flags inside vulnerable services docker containers:

- For inf-pump and spring services, I set up a SSH connection with inf-pump and spring docker containers and then I append new flag to `FileFlag.txt` file containing all flags. All passwords for SSH connection are random and saved in a dedicated `SSHpasswords` file, and ssh connection is managed using paramiko python library.
- For testify service, I save each flag twice, once for each vulnerability. For both vulnerabilities I first create a new user and then I create an appointment for both of them but, while for first vulnerability flag is saved inside user filename, for second vulnerability flag is saved inside user notes. In same round, two different users are created for the two vulnerabilities, but same flag is stored for both of them.
- For postit service, I create a new user using a random username and a RSA public key, always equal to 3, then I login by signing challenge sent by postit service using user's private key, and finally I add flag to user notes

In these last two services, I modified code taken from original version, because code which implemented services management performed by gameserver relied on a on-purpose library which consisted in an high volume of code lines, and it was not easily understandable. So I decided to modify code structure deleting this library and using well known libraries, such as "request" python library, useful for making HTTP requests to a specified URL, in testify service, and pwn python library, for remote connections in postit service.

4.4 Check flags availability

To check valid flags availability inside all vulnerable services, I used two different strategies:

- For inf-pump and spring services, I open a SSH connection (always using paramiko python library) with inf-pump and spring docker containers, I get `TeamX_flag_and_info.txt` file containing list of valid flags, and then I compare it with text file containing valid flags saved inside gameserver VM.
- For testify and postit services, I check all flags contained inside `TeamX_flag_and_info.txt` file by logging using same users' credentials used when gameserver registered users and inserted flags on vulnerable services side (I save this credentials together with flags inside text file) and retrieving flags one by one

If at the end of this process not all valid flags are available, which means that at least one of them has been modified or deleted, or that container is down or that it is not correctly working, I overwrite a dedicated file, one for each team and for each service, named `SLAX.txt`, by inserting a “0”, if instead all valid flags are available, I overwrite file by inserting a “1”. Note that if a flag is sent by gameserver to a team’s service while its container is down, this flag will not be saved and team will lose SLA points for the whole time this flag will be valid.

4.5 Computation of SLA points

SLA points computation is performed by scoreboard web server. At the beginning all services start with a SLA score equal to 1 and, at each round, value contained inside `SLAX.txt` file is checked. If a “1” is stored, SLA score of that team is not modified but, if a “0” is stored, SLA points will be decreased, which means that SLA value of that team for that service inside `teams.db` file will be reduced. In this way teams will not put all their containers down to not let flags be stolen from their services, because they would lose a considerable amount of points.

During my simulation I decided to reduce SLA points by 0.01 each time a value “0” was found inside `SLAX.txt` but, in a real Attack/Defense CTF, SLA points should be decreased accordingly to the duration of the competition (and related total number of rounds), because it should cover percentage of rounds in which all valid flags were correctly available.

4.6 Management of flags submission

For flag submission management, I decided to implement a python TCP server, because for me it was the simplest and fastest way in which teams could send flags.

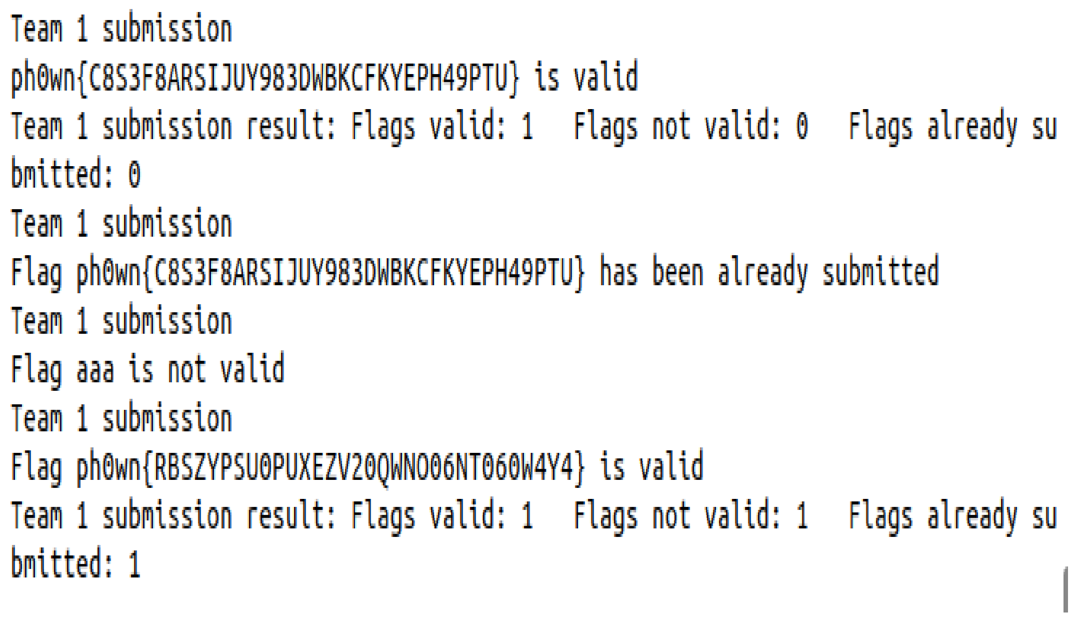
Scoreboard page is available at gameserver IP address `10.0.254.2` on port 5000 and, each time a team connects to it (can be done simply using netcat command), server understands which team has opened connection by looking at its IP address, and then is available to receive flags. For a correct flags submission, teams have to send flags (at least one) split by a blank space with the name of exploited service at the end. If this format is not respected, a “No valid service” message is displayed.

When flags and service name are received correctly, server checks if received flags are stored inside `FlagsValidX.txt` of team that has sent flags, stored inside “service name” folder. Two different aspects have to be checked:

- If flag is valid or not valid. If a flag is valid it has to be stored so that it is not counted if it is submitted again
- If flag has been already submitted, by checking all flags previously submitted.

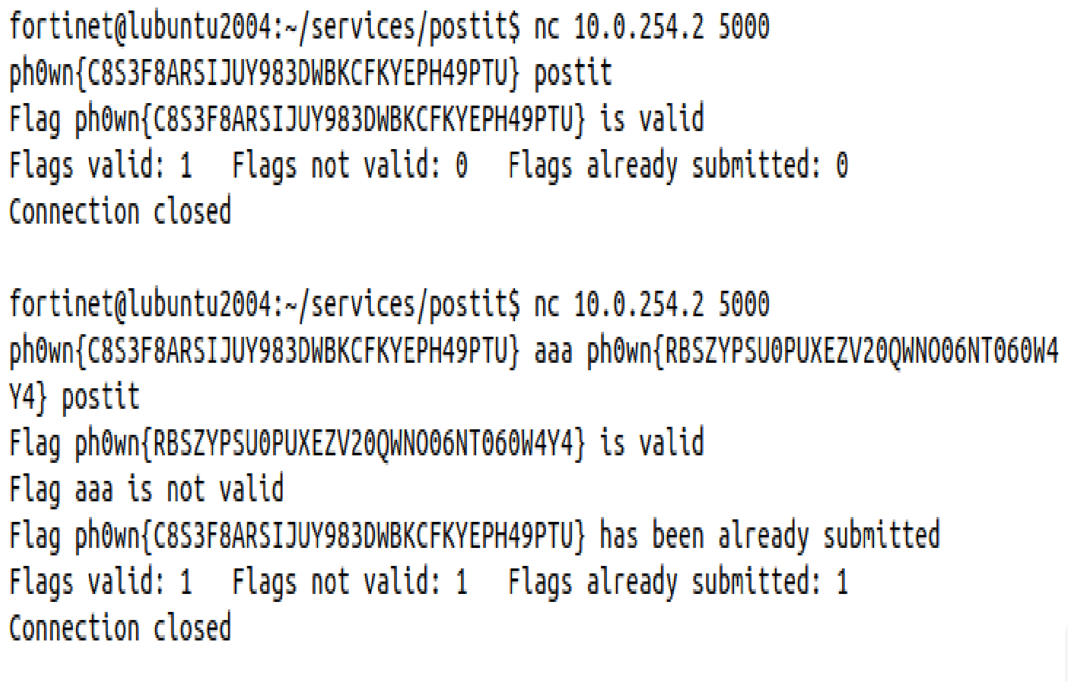
Once this check has been performed on all submitted flags, server has to modify number of valid flags submitted for that service and overall team score inside `teams.db`, so that new values could be seen on scoreboard web page.

As feedback, both on server and on client side are printed number of valid flags, number of not valid flags, and number of already submitted flags, together with all flags and their evaluation. This could be useful on team side to understand how all flags have been considered, but also on gameserver side to understand if a team is not trying to exploit vulnerable services, but it is trying to randomly send some possible valid flags. If number of not valid flags is too high, or a team sends flags that have never been generated, organizers can give a penalty and reduce team overall score.



```
Team 1 submission
ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} is valid
Team 1 submission result: Flags valid: 1   Flags not valid: 0   Flags already su
bmitted: 0
Team 1 submission
Flag ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} has been already submitted
Team 1 submission
Flag aaa is not valid
Team 1 submission
Flag ph0wn{RBSZYPSU0PUXEZV20QWNO06NT060W4Y4} is valid
Team 1 submission result: Flags valid: 1   Flags not valid: 1   Flags already su
bmitted: 1
```

Figure 4.2: Server side submission



```
fortinet@lubuntu2004:~/services/postit$ nc 10.0.254.2 5000
ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} postit
Flag ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} is valid
Flags valid: 1   Flags not valid: 0   Flags already submitted: 0
Connection closed

fortinet@lubuntu2004:~/services/postit$ nc 10.0.254.2 5000
ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} aaa ph0wn{RBSZYPSU0PUXEZV20QWNO06NT060W4
Y4} postit
Flag ph0wn{RBSZYPSU0PUXEZV20QWNO06NT060W4Y4} is valid
Flag aaa is not valid
Flag ph0wn{C8S3F8ARSIJUY983DWBKCFKYEPH49PTU} has been already submitted
Flags valid: 1   Flags not valid: 1   Flags already submitted: 1
Connection closed
```

Figure 4.3: Team side submission

Chapter 5

Results

This chapter presents main results of my thesis.

I tested my architecture and services by organizing couple of Attack/Defense CTF demos in which I played as competition organizer (managing gameserver virtual machine) and some Fortinet colleagues (with little experience in Attack/Defense CTF) played contestants roles, by managing each one a team VM.

After fixing some problems, my architecture has been able to guarantee the smooth progress of the demos, gameserver performed its task correctly by managing flag submission, detecting changes of services containers state and flags tampering.

During these demos only inf-pump service has been exploited, confirming its utility in let beginner contestants understand how this type of competition works, while other services were too difficult, so I had to provide them exploits in order to test them.

A positive aspect of this architecture was that actions like flags generation or checking if flags were available in all teams' vulnerable services were all correctly performed in each round, not only in some of them, and that in each round was not performed only one of them, like some architecture available on the Internet, but all these actions were performed simultaneously.

However, before using this architecture (and these services) to host a real Attack/Defense CTF, it should be tested by people with more experience in this type of competition, because they could discover some unwanted vulnerabilities that can prevent smooth progress of competition, and with a bigger amount of user, to check architecture behavior in conditions as close as possible to those of a real CTF competition (scalability is the main problem in implementation of architecture for this type of competition). Main aspects related to scalability that have to be checked in this case are:

- power that has to be provided to gameserver, which will have to handle a way higher number of teams
- router behavior, because it will have a much greater load, and for its maximum interfaces interfaces supported. While there shouldn't be problems due to the amount of load (according to a Fortinet network expert who has a huge experience in networks implementation using FortiPoC), in case of an high number of teams, we should implement a kind of router tree structure by inserting two (or more) routers and splitting teams among them.

Chapter 6

Conclusions

Development of a framework for an Attack/Defense CTF and development of some Attack/Defense CTF challenges are very difficult tasks, which require a lot of software, hardware and human resources. A good level of networking and programming skills, experience in the field of CTF and a significant amount of time are necessary to support the creation of this architecture and to monitor and test its behavior and all challenges running inside it, but all of this is rewarded by a significant improvement of all these abilities that are put to the test while these tasks are performed.

In fact, as in the case of playing CTF competitions, the development of this framework and of these challenges, performed working in a team, are very useful because each member of development team can learn new skills from more experienced and more skilled teammates. A very significant difference between playing Attack/Defense CTF and developing an architecture and some challenges for an Attack/Defense CTF is that, in this second case, together with teamwork skills, are developed networking skills, useful to build network which has to host gameserver and all teams that have to play in CTF competition, security skills and developing skills because for each CTF competition some challenges have to be build, and the best method to build interesting challenges consists in analysing challenges of other competitions. Testing skills are developed too, because the architecture and all challenges have to be tested to check that don't exists unwanted vulnerabilities or points of failure.

So, despite all the resources and skills required, and all the critical issues that may emerge during its implementation, the development of an architecture for an Attack-/Defense CTF is a valid tool in the field of computer security as well as playing CTF competition itself.

Appendix A

Developer Manual

A.1 Developer Manual

This section contains more detailed information about the proposed architecture, in order to make easier its replication or improvement. Main used language are Python, Javascript and MySQL.

A.1.1 How to set up FortiPoC instance

To create network topology which should host my competition I use FortiPoC, an internal Fortinet tool used to create simple or complex network topology using the different Fortinet products. I define the network architecture I need, the devices that will run into this network, then I run the whole stuff and FortiPoC automatically manage to create all these item for me. FortiPoC also saves all this information in a poc file that can be replayed on any other FortiPoC instance.

Then I chose Google Cloud Platform as Virtual Private Server (VPS), because it is the one normally used at Fortinet when someone wants to build a network using FortiPoC. In this way network architecture of the game is built entirely in a single virtual machine hosted in the Google Cloud Platform (GCP) and is defined a Google Compute Instance which runs the FortiPoC operating system. The compute instance is assigned with a private, non-routable IP address allocated from the Google Private IP address space. In order to access the FortiPoC administrative console, Google automatically assigns a public routed IP address that is mapped to the compute instance's private IP address. Once the public IP address is accessed, Google performs a network address translation (NAT), so it replaces the public IP with the mapped private IP. Because this operation is done on the destination IP address it is called destination network address translation (DNAT).

When I create a new instance, I define some characteristics such as labels, region in which instance will be hosted, type and number of CPUs needed and memory needed. I chose 32 GB of memory with 8 CPUs, because I thought they would be enough for an infrastructure with a limited number of teams VMs. From networking point of view, I can define which type of traffic is allowed from internet (I chose both http and https traffic) and which IP addresses and ports can access this instance. Then I can define all disk info (size, type, deletion rule) and all boot disk info (FortiPoC image and size).

Now I can start working on my instance. First of all I have to enter inside the new instance by inserting IP address provided by GCP and then resolve "Your connection is not private" error by typing "thisisunsafe". Then I have to resolve "not registered" FortiPoC error by login (using default "admin" username and no password) inside system → registration using Fortinet personal credentials. After that I can change the password and refresh all repositories (repositories → sources). To use the latest FortiPoc version I can open a ssh session on instance IP using admin account, and then run "execute upgrade".

At this point I can create my PoC inside PoC → Definitions, where I define name of this PoC, password of all devices inside it (except for ones that have an image with a default password, such as VyOS routers) and start mode (parallel).

A.1.2 How to build competition infrastructure

I built an infrastructure which could host an Attack/Defense CTF with 5 teams. It could be easily modified to support an higher number of teams, but then I have to worry about increasing memory and CPU parameters previously described.

First I had to define all networks inside "networks" subsection: for each device (gameserver VM and all teams VMs) I created a /24 network. Choice of team and gameserver IP addresses is particularly important, because IP addresses have to be deductible and easy to remember by all people involved in the competition.

To achieve that, I decided to implement my architecture following these rules:

- Build one /24 network for each team, with IP address `10.0.teamId.0` (teamId>0)
- Build one /24 network for gameserver, with IP address `10.0.254.0`
- Build one /24 "Internet" network, with IP address `10.0.0.0`, used to provide Internet connection to the whole architecture

Inside each of these networks, two IP addresses are defined:

- `10.0.teamId.2` (or `10.0.254.2` inside gameserver network) is team (or gameserver) IP address
- `10.0.teamId.1` (or `10.0.254.1` inside gameserver network) is central VyOS router IP address

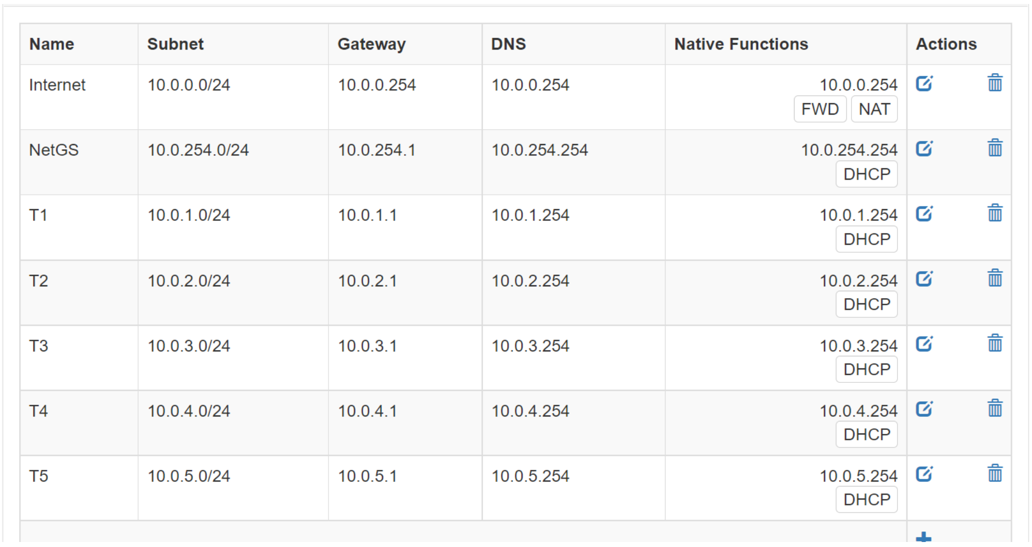
When I create a network, FortiPoC asks me some advanced parameters and about Native Functions. Advanced parameters are:

- Network gateway IP address, and here I chose .1 IP address for gameserver and for all teams networks, because, in this way, all traffic generated by all networks will have to pass through central router, whose access is reserved only to organizers. For "Internet" network I selected .254 IP address as gateway.
- Network DNS IP address, and for all networks I chose .254 IP address as DNS

Instead Native Function are:

- IP forwarding and NAT, that I selected only on "Internet" network, because it has to be connected to Internet and has to provide Internet connection to the rest of the architecture (all Internet traffic coming from gameserver and teams networks will pass first through central router and then through this "Internet" network)

- DHCP (Dynamic Host Configuration Control), that I selected only on gameserver and teams networks because FortiPoC doesn't support yet type of Linux device I instantiated (see below) and their IP address can't be provisioned. So I used DHCP protocol to control the IP addresses that will be assigned to the network interfaces of these devices. I didn't select it for "Internet" network, because VyOS router instead is a device type that FortiPoC support for IP addresses provisioning.



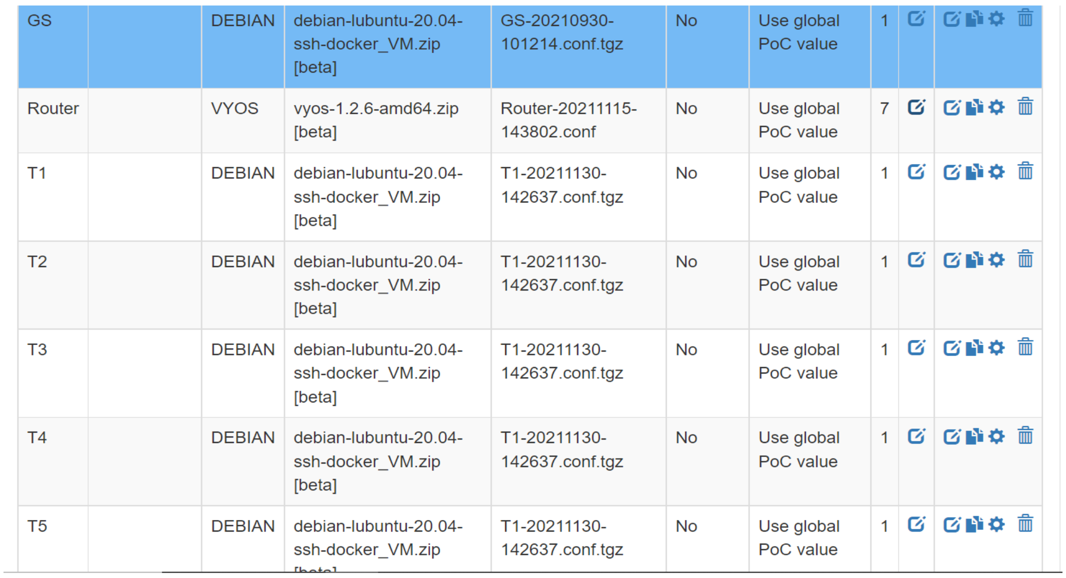
Name	Subnet	Gateway	DNS	Native Functions	Actions
Internet	10.0.0.0/24	10.0.0.254	10.0.0.254	10.0.0.254 FWD NAT	
NetGS	10.0.254.0/24	10.0.254.1	10.0.254.254	10.0.254.254 DHCP	
T1	10.0.1.0/24	10.0.1.1	10.0.1.254	10.0.1.254 DHCP	
T2	10.0.2.0/24	10.0.2.1	10.0.2.254	10.0.2.254 DHCP	
T3	10.0.3.0/24	10.0.3.1	10.0.3.254	10.0.3.254 DHCP	
T4	10.0.4.0/24	10.0.4.1	10.0.4.254	10.0.4.254 DHCP	
T5	10.0.5.0/24	10.0.5.1	10.0.5.254	10.0.5.254 DHCP	

Figure A.1: All networks configuration

Now I can define all devices, inside "Devices" subsection. For each of them I have to decide some parameters:

- Name, I chose "T1, T2 ..." for teams devices and "Router" for central router.
- Image, I chose debian-lubuntu image with Docker for gameserver and teams devices, because Docker is already installed, while I chose VyOS image for central router
- Network from which it has to take all parameters, for "Router" I selected "Internet" network, while for all other devices I selected their own network
- Advanced parameters such as memory and CPU cores, I selected 4 Gb and two cores for each device.

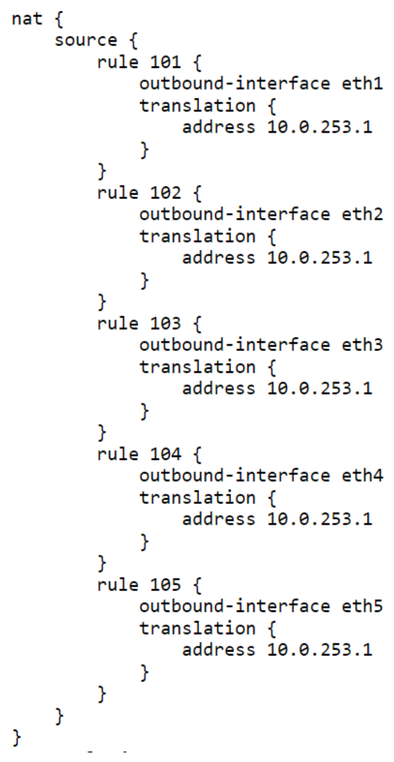
I have also to define ports for each device, selecting "edit ports" option, choosing DHCP as Address Configuration Mode and the network from which I want to get an IP.



GS		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	GS-20210930-101214.conf.tgz	No	Use global PoC value	1		
Router		VYOS	vyos-1.2.6-amd64.zip [beta]	Router-20211115-143802.conf	No	Use global PoC value	7		
T1		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T2		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T3		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T4		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip [beta]	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		
T5		DEBIAN	debian-lubuntu-20.04-ssh-docker_VM.zip	T1-20211130-142637.conf.tgz	No	Use global PoC value	1		

Figure A.2: All devices configuration

Last step is implementation of IP masquerading inside VyOS router. I do this because when a team receives a packet, it shouldn't be able to understand if it has been sent by gameserver or by another team. So I decided that, when a packet is sent by gameserver or a team to another team, source IP address has be changed by router to `10.0.253.2`. To do that, I have to access to VyOS router using default VyOS credentials (username: vyos, password: vyos), enter in configuration mode using `configure` command, use commands `set nat source rule 10X outbound-interface ethX` and `set nat source rule 10X translation address 10.0.253.1`, where X is team_id, and save new configuration using `commit` and `save` commands.



```
nat {
    source {
        rule 101 {
            outbound-interface eth1
            translation {
                address 10.0.253.1
            }
        }
        rule 102 {
            outbound-interface eth2
            translation {
                address 10.0.253.1
            }
        }
        rule 103 {
            outbound-interface eth3
            translation {
                address 10.0.253.1
            }
        }
        rule 104 {
            outbound-interface eth4
            translation {
                address 10.0.253.1
            }
        }
        rule 105 {
            outbound-interface eth5
            translation {
                address 10.0.253.1
            }
        }
    }
}
```

Figure A.3: IP masquerading rules in router configuration file

This is the final network

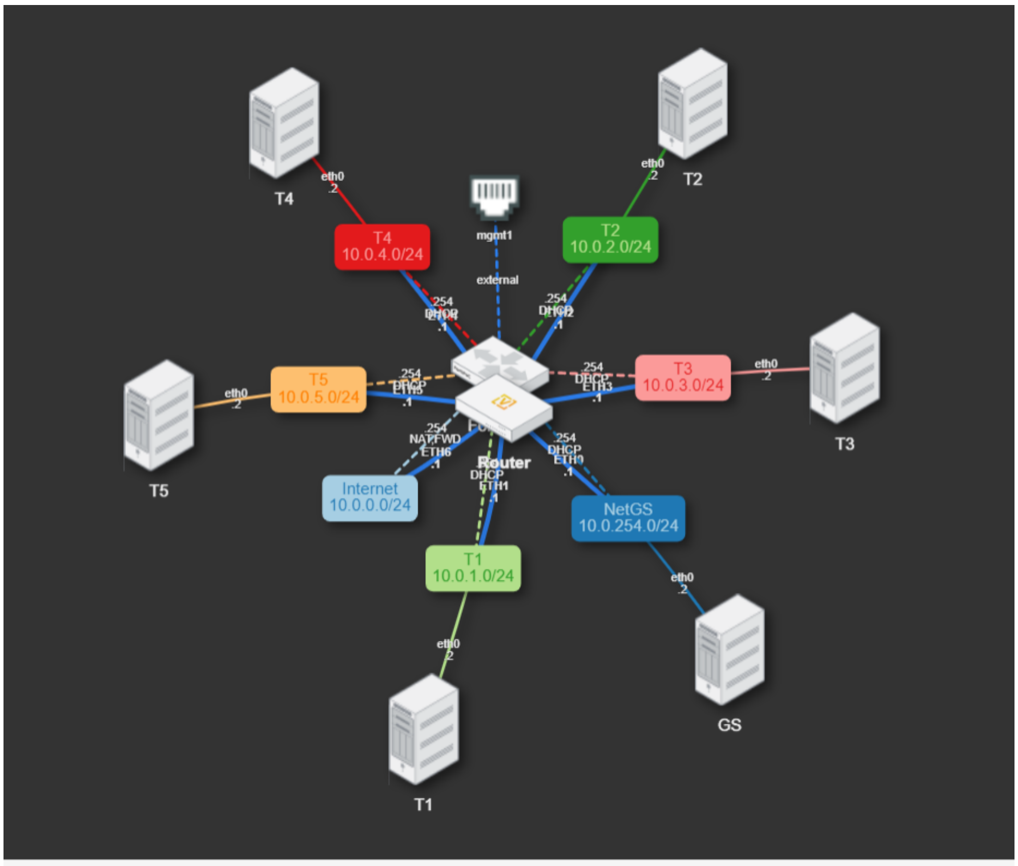


Figure A.4: Final Network

A.1.3 How to start a competition

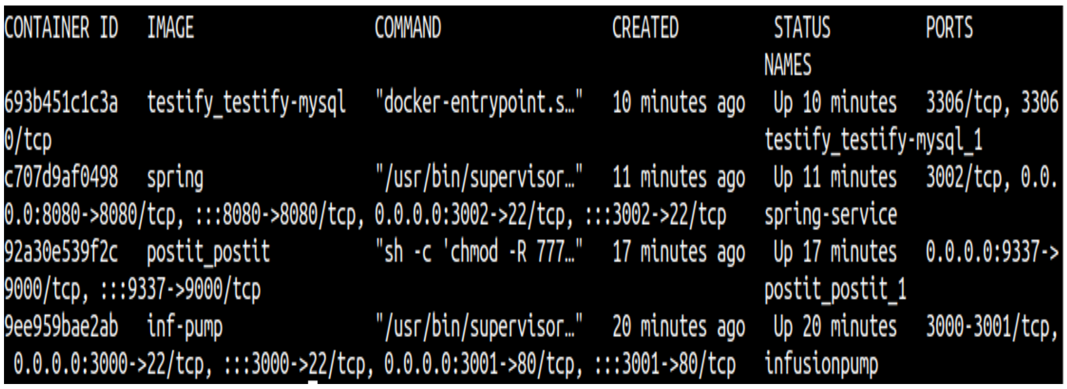
To start a competition, I have to run scoreboard and submission flag server on gameserver VM, I have to start all vulnerable services Docker containers on all teams VMs, and then I have to start flags generation. To run scoreboard I have to follow two steps:

- inside scoreboard/client/src folder, I have to run scoreboard client side using command `npm init` and then `npm start`
- inside scoreboard/server folder, I have to run scoreboard server side using command `nodemon team_server.js path_to_scoreboard_server_folder`

To run submission flag server I have to enter inside scoreboard/server folder and use command `python3 submission_flag_server`

Then I have to start docker container of vulnerable services of all teams, because they have to be up to receive flags from gameserver and because when competition starts I have to provide a VM with all services up and properly functioning to all teams.

To do it, inside all teams VMs, I have to enter inside each vulnerable service folder (all folders are inside "services" folder) and run command `docker-compose up -d -build`, which will start service Docker container named as the folder in which I have launched the command. Once done that, I can check that all containers are up using command `docker ps`, which will list all containers currently running.



```
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS
                                                                              NAMES
693b451c1c3a   testify_testify-mysql "docker-entrypoint.s…"   10 minutes ago   Up 10 minutes   3306/tcp, 3306
0/tcp                                                                         testify_testify-mysql_1
c707d9af0498   spring                "/usr/bin/supervisor…"   11 minutes ago   Up 11 minutes   3002/tcp, 0.0.
0.0:8080->8080/tcp, :::8080->8080/tcp, 0.0.0.0:3002->22/tcp, :::3002->22/tcp   spring-service
92a30e539f2c   postit_postit         "sh -c 'chmod -R 777…"   17 minutes ago   Up 17 minutes   0.0.0.0:9337->
9000/tcp, :::9337->9000/tcp                                                   postit_postit_1
9ee959bae2ab   inf-pump              "/usr/bin/supervisor…"   20 minutes ago   Up 20 minutes   3000-3001/tcp,
 0.0.0.0:3000->22/tcp, :::3000->22/tcp, 0.0.0.0:3001->80/tcp, :::3001->80/tcp   infusionpump
```

Figure A.5: List of all docker containers correctly running

Then, in gameserver VM, I can start flags generation for all services. To do that, I have to enter inside gameserver folder where, with "scoreboard" folder, are stored all services folders. Inside each of them I have to run command saved inside "runPython", which will run"generate_and_test_flags.py", taking care to change parameters such as number of teams (inside python files) or folder paths (passed as arguments), if number of teams joining the competition is different or if path to "scoreboard" folder is different.

As last step, passwords of all teams VMs, of gameserver VM and of VyOS router have to be randomized and credentials (IP address, username, team_id and password of each team) have to be given to all teams joining the competition.

Appendix B

User Manual

B.1 User Manual

B.1.1 How to access VMs

When competition starts, each team has its own VM. Once credentials (IP address, username, team_id and password) are received, each team can connect to its VM at port number 11000+team_id through SSH using command `ssh username@IP_address -p port_number` and inserting password when requested. Teams can also use a tool like PuTTY, a free and open-source terminal and serial console which supports several network protocols, including SSH. Once opened PuTTY, teams have only to configure IP address and port number field and then insert username and password when requested.

B.1.2 How to play A/D CTF

Once a team is inside its own VM, it can immediately see which are vulnerable services of this Attack/Defense competition, using command `docker ps`, which will show what are Docker container of services which are currently running, and can start analysing them. Each service has its own folder, named as service itself.

Competition network is structured as follows:

- Gameserver has IP address 10.0.254.2
- Team x has IP address 10.0.x.2

Goal of this competition is to find vulnerabilities inside vulnerable services, exploit them to steal flags from service of other teams and patch its own services to not let other team steal flags from them.

Flags are strings with "ph0wn" string concatenated to a 32 uppercase random characters string enclosed in curly brackets. Competition is divided into rounds, and at each rounds new flags are generated for each service. Flags have a lifetime, which is usually a multiple of duration of a round, so that if a team submits too old flags, it obtains no points.

When some flags are stolen, they have to be submitted to the gameserver (VM of competition organizer), to obtain points. To do that, team has to connect to flag submission server running on gameserver VM, using command `nc 10.0.254.2 5000`, and then has to submit list of stolen flags, all separated by a blank space and with name of exploited service at the end. Once list of flags is submitted correctly, submission summary will be displayed, in which will be shown amount of points earned and how each flag has been considered. If submission has not been done correctly, an error message “No valid service” will appear.

If a team finds a vulnerability, it has also to patch it on its service. To redeploy service once code has been modified, team should use command `docker-compose build`, because it rebuilds service’s image using the existing code. Team has also to worry about if service is correctly running after image is rebuilt and to do that, it can use again command `docker ps` or check SLA score of that service. SLA (Service Level Agreement) is second parameter used to compute teams score, and is computed considering amount of time in which services work correctly and flags were available inside them.

In fact number of flags submitted is not the only parameter used for score computation. Gameserver checks at each round if all docker containers, in which vulnerable services run, are up and generated flags are still available (so they can be stolen by other teams), and if this is not true SLA points are reduced. In this way a team, not to let other team steal flags from their services, cannot simply put their services down or delete/modify flags.

Scores of all teams can be visualized at gameserver IP address `10.0.254.2` on port 3000.. For each team are visualized number of flags stolen and SLA score for each service, and team overall score.

Bibliography

[1] Erik Trickel, Francesco Disperati, Eric Gustafson, Faezeh Kalantari, Mike Mabey, Naveen Tiwari, Yeganeh Safaei, Adam Doupe and Giovanni Vigna, “Shell We Play A Game? CTF-as-a-service for Security Education” 2017 USENIX Workshop on Advances in Security Education, Vancouver (Canada), August 15, 2017, pp. 1-3

[2] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupe, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat and Yan Shoshitaishvili, “Ten Years of iCTF: The Good, The Bad, and The Ugly”, 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education, 3GSE, San Diego (CA, USA) August 18, 2014, pp. 1-3

[3] Andy Davis, Tim Leek, Michael Zhivich, Kyle Gwinnup, and William Leonard, “The Fun and Future of CTF”, 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education, 3GSE, San Diego (CA, USA) August 18, 2014, pp. 2

[4] Destructive Farm `https://github.com/DestructiveVoice/DestructiveFarm`

[5] Caronte `https://github.com/eciavatta/caronte`

[6] Flower `https://github.com/secgroup/flower`

[7] CTF Time, https://ctftime.org, 2014.

[8] Alexander Mansurov,“A CTF-Based Approach in Information Security Education: An Extracurricular Activity in Teaching Students at Altai State University, Russia”, Online published, August 17, 2016, pp. 159-162, DOI 10.5539/mas.v10n11p159.

[9] Shellphish iCTF platform https://github.com/shellphish/ictf-framework

[10] Arvind S Raj, Bithin Alangot, Seshagiri Prabhu, and Krishnashree Achuthan,“Scalable and lightweight CTF infrastructures using application containers”, 2016 USENIX Workshop on Advances in Security Education (ASE 16), Austin (TX, USA) August 9, 2016, pp. 2-8

[11] InCTF platform https://github.com/inctf/inctf-framework

[12] ForcAD platform https://github.com/pomo-mondreganto/ForcAD