University of Bamberg
Professorship for Computer Science

Foundations of Internet Communication KTR-GIK-M Laboratories Summer Term 2020

Assignment 4

NATting, Firewalls, and Virtual Private Networks

Marcel Großmann, Duy Le, Jopaul John, Markus Wolff

Issued:	June 8, 2020
Deadline:	June 21, 23:55, 2020

Prelab

1. Get yourself familiar with the concepts of NAT traversal and read through the VyOS documentation about the descriptions for a source NAT.
2. Read through the documentation of `iptables` and the ability to control packet forwarding [1].
3. Tryout the SoftEther VPN software (https://www.softether.org) and understand the concepts of
   - the `vpnserver`,
   - the `vpnclient` and
   - the `vpncmd` console to configure running `vpnserver` and `vpnclient` instances.
4. Get familiar to use the tool `traceroute`.

1 Network Address Translation (NAT) and Firewalls

Dig out the Kathará laboratory of the previous assignment. Normally, a cloud provider is not routing a web server to the Internet in a way that it can be reached from any host on any port worldwide. We want to change that behaviour in our previous topology. On the one hand, we can enable the load balancer to serve as a single entry point for the web servers, which increases security by restricting access to the machines on the IP level. On the other hand, we can restrict the traffic to the network by setting up a firewall. As Figure 1 shows, we do not need to modify the topology itself, only one image for `r3` needs to be changed (see Table 1). We slightly adjust `r2` and `r3` to restrict the access to our server farms. Both concepts are implemented in the following tasks.

[1] https://en.wikibooks.org/wiki/Communication_Networks/IP_Tables

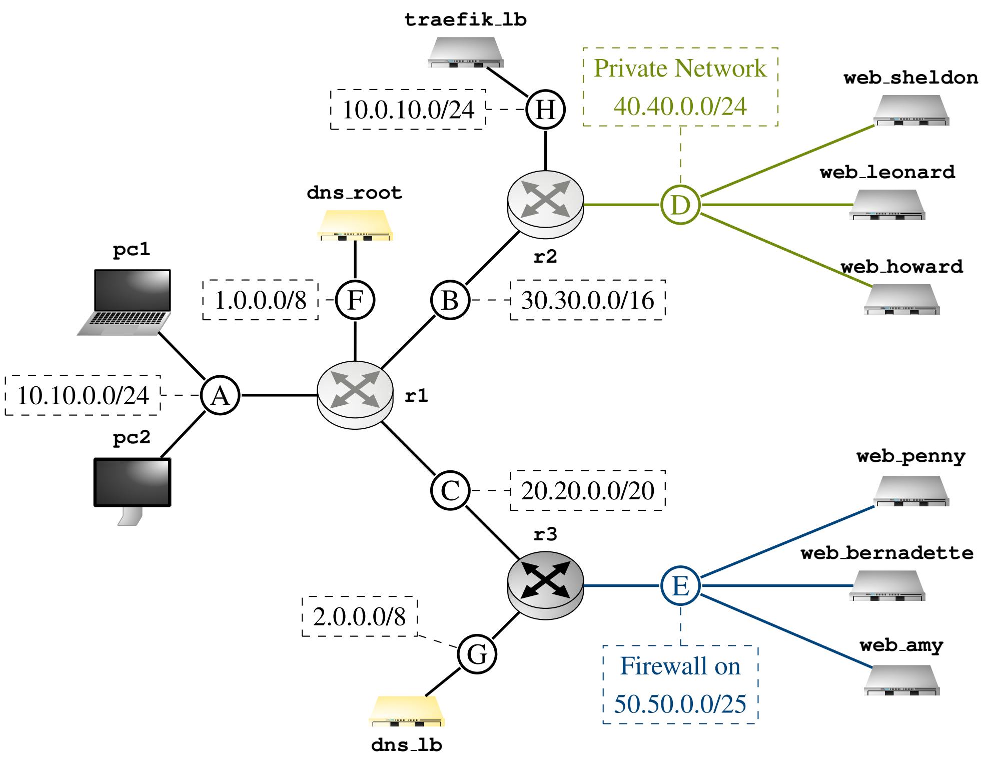

Figure 1: NAT & firewall experiment configuration

Name	Docker Image	IP Address	Interface	CD
pc1	unibaktr/alpine	10.10.0.10/24	eth0	A
pc2	unibaktr/alpine	10.10.0.11/24	eth0	A
r1	unibaktr/vyos	10.10.0.1/24	eth0	A
r1	unibaktr/vyos	30.30.0.1/16	eth1	B
r1	unibaktr/vyos	20.20.0.1/20	eth2	C
r1	unibaktr/vyos	1.0.0.1/8	eth3	F
r2	unibaktr/vyos	30.30.0.2/16	eth0	B
r2	unibaktr/vyos	40.40.0.2/24	eth1	D
r2	unibaktr/vyos	10.0.10.2/24	eth2	H
r3	unibaktr/alpine	20.20.0.3/20	eth0	C
r3	unibaktr/alpine	50.50.0.3/25	eth1	E
r3	unibaktr/alpine	2.0.0.3/8	eth2	G
dns_root	unibaktr/alpine:coredns	1.1.1.1/8	eth0	F
dns_lb	unibaktr/alpine:coredns	2.2.2.2/8	eth0	G

Name	Docker Image	IP Address	Interface	CD
traefik_lb	unibaktr/alpine:traefik	10.0.10.1/24	eth0	H
web_sheldon	unibaktr/alpine:whoami	40.40.0.100/24	eth0	D
web_leonard	unibaktr/alpine:whoami	40.40.0.101/24	eth0	D
web_howard	unibaktr/alpine:whoami	40.40.0.102/24	eth0	D
web_penny	unibaktr/alpine:whoami	50.50.0.100/25	eth0	E
web_bernadette	unibaktr/alpine:whoami	50.50.0.101/25	eth0	E
web_amy	unibaktr/alpine:whoami	50.50.0.102/25	eth0	E

Table 1: Experiment configuration for a NAT and a firewall

1.1 Network Address Translation (NAT)

We consider our network `D` to be a private network and thus achieve that only the load balancer serves as an entrypoint to `gik.de`. Therefore, we change the configuration in the following way:

1. For the first server farm, we configure on `r2`:
   - a source NAT with the gateway `40.40.0.2/24` in network `D`,
   - and masquerade domain `D` to reach domains `B`, `A` and `F`.
2. On `r1` and `r3` **remove** the route to reach collision domain (CD) `D`.
3. From `pc2` ensure that, e.g., `web_sheldon` is not pingable, but `curl gik.de` still delivers a result.
4. On CD `B` and CD `D` capture the call `curl gik.de` with wireshark and explain the behaviour.

1.2 Firewall

On network `E`, we want to restrict the access to our servers and drop all packets except the ones, which establish `tcp` connections on port `80`.

1. Therefore, we need to enable a firewall with `iptables` on `r3`.
   (a) Create a default filter policy to **drop** all packets,
   (b) than **allow** unlimited traffic on the loopback interface `lo` again.
   (c) **Forward** packets of the DNS network `2.0.0.0/8` without any restrictions.
   (d) **Allow** all web servers in CD `E` to accept connections on **tcp port 80**.
2. On `pc2` ensure that `ping gik.org` fails, but `curl gik.org` displays the website content.

2 Virtual Private Network (VPN)

Unfortunately, our administrator, who currently works at home, needs unrestricted remote access to the servers. Since we do not want to get rid of our security improvements, we setup a VPN to achieve that goal. Therefore, we use a software solution called Softether (https://www.softether.org).

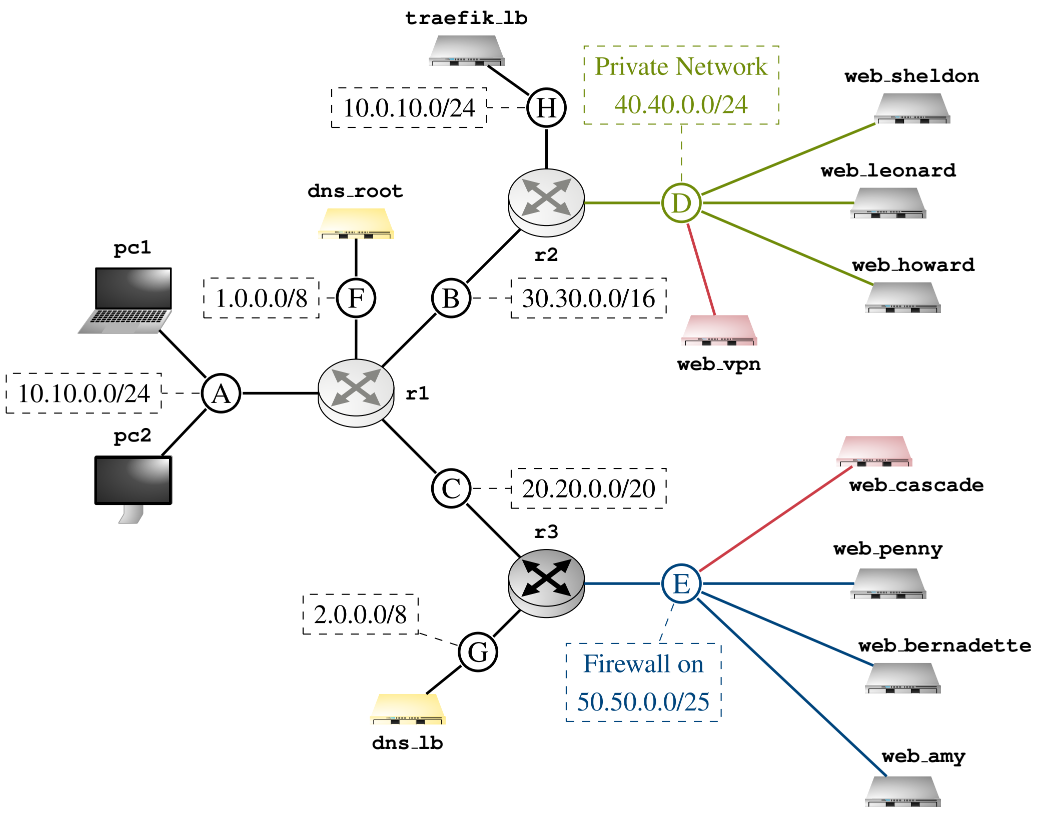

Figure 2: VPN configuration

Name	Docker Image	IP Address	Interface	CD
pc1	unibaktr/alpine:softether	10.10.0.10/24	eth0	A
web_vpn	unibaktr/alpine:softether	40.40.0.99/24	eth0	D
web_cascade	unibaktr/alpine:softether	50.50.0.99/25	eth0	E

Table 2: Modifications of Table 1 to enable VPN

User	Password
ktr	tcpip-admin
s2s	cascade-admin

Table 3: VPN users and passwords

2.1 Softether Server & Client

First, we want to create a remote access VPN to give our administrator, who uses `pc1`, unrestricted access to the servers in our private network `D`. If she is connected to the VPN server, her network topology looks like the one of Figure 3 and her machine seems to be in the same local area network (LAN).

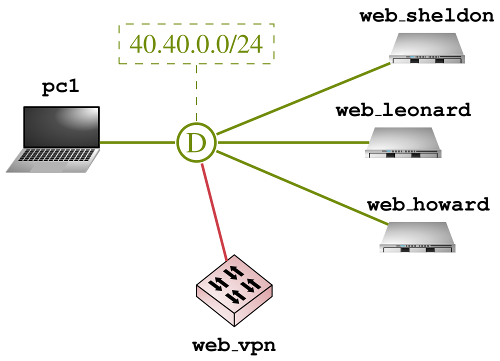

Figure 3: Topology appearance after a connection to the VPN is established.

- To achieve the described behaviour, we create a Softether server, called `web_vpn` in the topology of Figure 2.
  1. Our VPN server is started with the command `vpnserver start` and can be configured by `vpncmd`.
     (a) First we create a hub and append a **tap** device called **intern** to it.
     (b) On the hub, we add the users of Table 3.
  2. A new network interface card (NIC) `tap_intern` was created, which needs to be connected to `eth0` by a bridge.
- Unfortunately, `web_vpn` is inside a NAT and not reachable for our administrator on `pc1`. To facilitate this, we need to open **tcp port 443** on `r2` to forward traffic from it's `eth0` interface to `web_vpn`.
- Now let us try, if our administrator can connect from `pc1` by

1. starting a VPN client with `vpnclient start` on it.
   (a) We add a virtual interface called **intern** and
   (b) create an account for the user **ktr**, which connects to `web_vpn` via `r2`.
2. If the connection succeeds, we can add a private IP address to the newly created NIC `vpn_intern`, which connects to the network as depicted in Figure 3.

- On `pc1` we start a `traceroute` to any web server of CD `D` and confirm that our target is only one hop away.
- From `pc1` we capture a `curl` to the IP address of one web server of CD `D` with wireshark on the domains `B` and `D`. Analyze and explain the captured VPN packets.

2.2 Softether Cascade Connections

Finally, we connect both server farms over a site-to-site (s2s) VPN with a cascade connection. Our administrator can connect to both sites over a single VPN connection and reach all nodes (`web_`$\star$) without any restrictions. Her topology would look like the one depicted in Figure 4. A further advantage of the cascade is that we can enable every web server to communicate with each other. So if we would replace our simple web servers with enhanced applications, which may consist of several microservices, they could securely communicate and use the ressources of both sites.

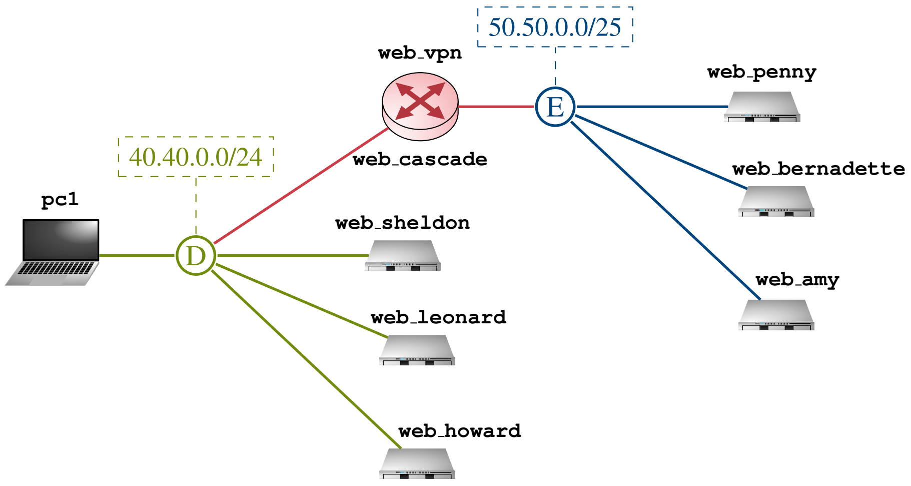

Figure 4: Topology appearance after a connection to the site-to-site VPN is established.

- We create a Softether server, called `web_cascade`
  1. and start it with the command `vpnserver start`.
     (a) We create a hub and a add **tap** device called **intern** to it.
     (b) On the created hub we add a cascade, which connects to the remote hub on `web_vpn` with the credentials of **s2s**.
  2. After the configuration, a NIC `tap_intern` was created, which needs to be connected to `eth0` by a bridge and
  3. additonally, we can add a route to `40.40.0.0/24`.
- On `r3` we need to modify the firewall to allow unrestricted access to `web_cascade`.
- If the cascade connection is established, we can add a route on `web_vpn` to CD `E`.
- Within the networks being routable over the gateways `web_vpn` and `web_cascade`, we can setup all web servers to reach each other:
  1. On `web_sheldon`, `web_leonard`, and `web_howard` we add a route to CD `E` over `web_vpn`.
  2. On `web_penny`, `web_bernadette`, and `web_amy` we add a route to CD `D` over `web_cascade`.
- Finally, for the administrator on `pc1`, we add a route to `50.50.0.0/25` over `web_vpn`, which acts as a gateway for both restricted networks.
- For the evaluation, start a `traceroute` to any web server of CD `E` from `pc1` and confirm that your target is only two hops away and gets routed over `web_vpn`.
- From `pc1` capture a `curl` to the IP address of one web server of CD `E` with wireshark on the domains `B` and `D`. Analyze and explain the captured VPN packets.

Lab Report: Write a short summary of your observations in the experiments and the encountered pitfalls in the exercise, provide screenshots whenever possible to justify your statements. Include the commands and write a short explanation with screenshots. **Upload** your Kathará files to gitlab and try to automate the setup by using configuration files.