Zero Trust Networking

1 - Introduction

In this challenge, you will see how to implement a Zero Trust Networking solution using OpenZiti.

2 - OpenZiti

OpenZiti is a free and open source project created by the Netfoundry company, focused on bringing zero trust networking principles directly into any application. The project provides all the pieces required to implement a zero trust overlay network and provides all the tools necessary to integrate zero trust into existing solutions (interesting for brown field applications).

1.1 - Network Infrastructure chosen

There is multiple Network Infrastructure to choose in OpenZiti before implement it:

- Local - No Docker
- Local - With Docker

- Local - Docker Compose
- Host OpenZiti Anywhere

We are going to use the 4th infrastructure “Host OpenZiti Anywhere” because we want to implement it over multiple physical/virtual devices.

1.2 - Components

OpenZiti Controller:

The Controller is the masterpiece of the network. It authenticates and authorizes communication between services, store policies, provide configuration plane…

OpenZiti Router:

The OpenZiti Router in combination with the Ziti Controller is responsible for authenticating and authorizing OpenZiti Edge Clients.
At least one Router is mandatory in a network infrastructure. Routers can mesh themself automatically in order to route packets more efficiently.

OpenZiti Edge Clients:

There are two types of OpenZiti Edge Clients.

- Tunnelers : are built-in software designed that we place in front of an already existing application to make it Zero Trust, to be compliant with the OpenZiti Network.
- SDK : is a Software Development Kit that provides you to create a fully Zero Trust application from scratch if the application is not already developed.

1.3 - Functioning

OpenZiti works with identities, services and policies.

1.3.1 - Identities

An identity is a way to identify a device on the OpenZiti overlay network and give it the permission it should have. By default, a non-configured identity can just do nothing on the overlay (this is the least privilege).
Roles, also called attributes, can be added to an entity in order to manage it and then distribute services access linked with these attributes.

1.3.2 - Services & Policies

A service encapsulates the definition of any resource that could be accessed by a client on a traditional network.

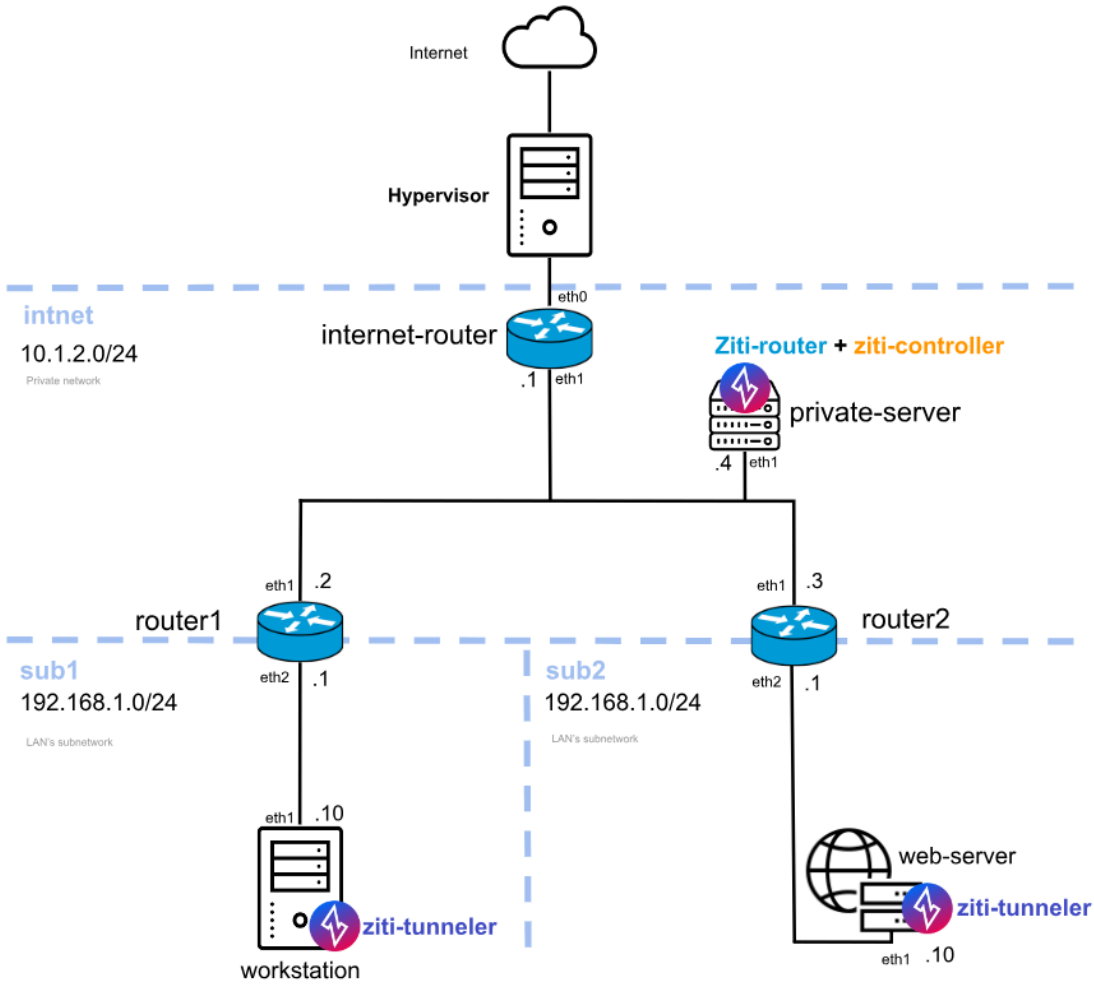
A service is defined by the following components:

- **Name** - the name of the service.
- **Termination** - Ziti only provides access to a network service, it does not provide the service itself. The service must be able to get network traffic to whatever application or application cluster is actually providing the service, whether that provider has Ziti embedded or has no knowledge of Ziti.
- **Configuration** - Ziti allows application specific configuration to be stored for services. See Configuration Store.
- **Authorization** - Policies are applied to services to manage attributes permissions of an identity.

3 - Lab

This guided lab will show you how to implement the simplest OpenZiti architecture in a private network.

3.1 - Network Architecture



3.2 - Controller and Router installation

The Edge Controller and Edge Router will be installed on the private-server which is accessible from all endpoints on the private network.
Open a shell on the private-server.

3.2.1 - Firewall

Before starting to install OpenZiti components, we have to open 3 ports for the edge controller and edge router.

Ports explanation:

- **8440/tcp**: edge controller providing router control plane
- **8441/tcp**: edge controller providing client sessions
- **8442/tcp**: edge router providing client connections

These ports are arbitrary, we are going to define them further in the installation.

Log as root to do all steps of the installation.
We going to use iptables to open necessary ports on the private-server:

```
iptables -A INPUT -p tcp --dport 8440 -m conntrack --ctstate NEW -m
comment --comment "Edge Controller providing router control plane" -j
ACCEPT
iptables -A INPUT -p tcp --dport 8441 -m conntrack --ctstate NEW -m
comment --comment "Edge Controller providing client sessions" -j ACCEPT
iptables -A INPUT -p tcp --dport 8442 -m conntrack --ctstate NEW -m
comment --comment "Edge Controller providing client connections" -j
ACCEPT
```

We can also define default policy rules to DROP on the INPUT and FORWARD chain of the firewall to drop all traffic except the three we just opened above.
Before setting DROP default policy, make sure the SSH rule on INPUT is active, or you will be disconnected!

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
```

3.2.2 - Express Installation

Prerequisites:

```
apt update && apt install tar hostname jq curl screen wget lsof -y
```

We can now install the controller and the router using a one-liner command provided by the documentation of OpenZiti: Express Install.
First setup some environment variables to specify multiple parameters like the Controller's and Router's IP and ports.

```
export ZITI_BIN_DIR="/root/ziti-bin"
mkdir $ZITI_BIN_DIR
export EXTERNAL_IP="10.1.2.4"
export EXTERNAL_DNS="10.1.2.4"
export ZITI_CTRL_EDGE_ADVERTISED_ADDRESS="${EXTERNAL_IP}"
export ZITI_CTRL_ADVERTISED_ADDRESS="${EXTERNAL_IP}"
export ZITI_CTRL_EDGE_IP_OVERRIDE="${EXTERNAL_IP}"
export ZITI_ROUTER_IP_OVERRIDE="${EXTERNAL_IP}"
export ZITI_CTRL_ADVERTISED_PORT=8440
export ZITI_CTRL_EDGE_ADVERTISED_PORT=8441
export ZITI_ROUTER_PORT=8442
export ZITI_CONFIG_FILE="ziti-controller-conf.yaml"
export ZITI_DIR="/root/.ziti/quickstart/$(hostname)"
```

Once variables of the root user environment are created, we can start to download the ziti binary that we are going to use to create configuration files, identities, policies, …

```
wget
'https://github.com/openziti/ziti/releases/download/v1.1.4/ziti-linux-am
d64-1.1.4.tar.gz' --no-check-certificate
tar -xf 'ziti-linux-amd64-1.1.4.tar.gz'
mv ziti $ZITI_BIN_DIR
rm 'ziti-linux-amd64-1.1.4.tar.gz'
ln -s $ZITI_BIN_DIR/ziti /usr/bin
```

We can now use “ziti” as a command.
Next we are going to download and execute a bash script to do the expressInstall. Specify the password you want to use for the default admin account.
Do not accept to update ziti binary

```
source /dev/stdin <<< "$(wget -O- --no-check-certificate
https://get.openziti.io/ziti-cli-functions.sh)";expressInstall
```

Configuration files are stored at $ZITI_DIR.
We can now start the controller using the ziti binary with the following command:

```
ziti controller run $ZITI_DIR/$(hostname).yaml
```

And then start the edge-router.

```
ziti router run $ZITI_DIR/$(hostname)-edge-router.yaml
```

I recommend you to use the "screen" binary to create virtual terminals that can be detached from your actual terminal. See https://linuxize.com/post/how-to-use-linux-screen/ for more details.
For example: Starting openziti controller via ssh without screen, when the ssh session ends the controller ends too. With screen, you can detach the process and keep running it even if you disconnect from your ssh session.
With screen:

```
screen -S controller -d -m ziti controller run
$ZITI_DIR/$(hostname).yaml
screen -S router -d -m ziti router run
$ZITI_DIR/$(hostname)-edge-router.yaml
```

Show running screen:

```
screen -ls
```

Access to a screen:

```
screen -R [screen name]
```

Detach from a screen: Ctrl + A, then D.

3.3 - Tunnelers installation

The controller and router are now installed and running. We can start to download and enroll the OpenZiti Tunellers on the workstation and the web-server.

Open a shell on workstation and web-server as ROOT. And follow the guide

Commands are the same for both machines.
Install the ziti binary:

```
export ZITI_BIN_DIR=~/ziti-bin
mkdir $ZITI_BIN_DIR
wget
https://github.com/openziti/ziti/releases/download/v1.1.4/ziti-linux-amd
64-1.1.4.tar.gz --no-check-certificate
tar -xf ziti-linux-amd64-1.1.4.tar.gz
mv ziti $ZITI_BIN_DIR
rm ziti-linux-amd64-1.1.4.tar.gz
ln -s $ZITI_BIN_DIR/ziti /usr/bin
```

Install the ziti-edge-tunnel binary with the one-line installer:

```
(set -euo pipefail
 cd $(mktemp -d)
 wget -q
"https://github.com/openziti/ziti-tunnel-sdk-c/releases/latest/download/
ziti-edge-tunnel-Linux_$(uname -m).zip"
 unzip ./ziti-edge-tunnel-Linux_$(uname -m).zip
 sudo install -o root -g root ./ziti-edge-tunnel /usr/local/bin/
 grep -q '^ziti:' /etc/group || sudo groupadd --system ziti
 sudo mkdir -pv /opt/openziti/etc/identities
 ziti-edge-tunnel version
 )
```

3.3.1 - Identity creation & enrollment

Once ziti-edge-tunnel and ziti binaries are installed on both workstation and web-server, we can continue by creating an identity using ziti (which use the API on the Controller)
First you have to login and specify the Controller IP + Controller Port that is in charge to manage client sessions. In our case this is port 8441.

```
ziti edge login
```

It was ask you to enter the controller host[:port], you will put **10.1.2.4:8441**
It will ask you if you want to accept the certificate provided by private-server, enter “Y”
And finally, log as “admin” with the password you defined on the private-server during the express installation of the controller/edge-router.

You are now successfully authenticated to the Controller and you can use the API via the ziti command line to create identities, manage policies, and so on!

Let's create an identity !

```
ziti edge create identity $(hostname) -o ~/$(hostname)_token.jwt
```

With this command, an identity is created via API and stored into the Controller. It gives us an JWT file that lets us enroll our identity to an OpenZiti's component.
In our case we want to enroll our ziti-edge-tunnel we just installed before.

You can do this with this command:

```
ziti-edge-tunnel enroll -j ~/$(hostname)_token.jwt -i
~/$(hostname)-tunneler-identity.json
```

This command takes in input (-j) the JWT file and outputs (-i) the identity of the enrolled identity.
Do this process for workstation and web-server.

Great! To summarize at this point, we have now:

- A Controller and an Edge-Router on the private-server.
- An enrolled tunneler named "workstation" on workstation.
- An enrolled tunneler named "web-server" on web-server.

3.4 - Service Configuration

In the context of this lab, we want to access the webserver via HTTP from the workstation.
A service takes in input two configuration file, so first we have to create two configurations:

- Intercept.v1: This config type configures an intercepting Ziti tunneler as a proxy for a particular Ziti service (workstation side).
- Host.v1: This config type configures a hosting Ziti tunneler to forward connections to the destination server for a particular Ziti service (web-server side).

3.4.1 - Intercept.v1

A configuration file is represented by a JSON string and can take multiple keys:values.
We need at least three parameters:

- Address: the address used to communicate with the endpoint. Can be whatever address
- Port: the port used to communicate with the endpoint. Can be a range.
- Protocol: the protocol used to communicate with the endpoint. Can be tcp or udp.

First on our workstation, we need an address that can be handled by the tunneler, a specific and arbitrary address that we will define.
The Top-Level Domain (TLD), for example in this domain name: google**.com**, the TLD is **.com**.
To avoid address conflict we are going to use the **.ziti** TLD which is not used on the Internet.
You can check the full list of TLDs managed by the ICANN and used on the Internet here: https://data.iana.org/TLD/tlds-alpha-by-domain.txt or https://iana.org/domains/root/db

So you can choose another TLD that is not in these lists.
Well, let's define this address to **website.ziti**. This address should trigger the tunneler which will forward this traffic to the OpenZiti's Network Overlay by an Edge Router.

The port will be 80, because we want to use HTTP.

And the protocol will be TCP, because HTTP is encapsulated in the TCP protocol.

In JSON, it looks like:

```
{
    "protocols":["tcp"],
    "addresses":["website.ziti"],
    "portRanges":[{"low":80, "high":80}]
}
```

We can now, on any machine where you are logged with the "ziti edge login" command, create via API an intercept.v1 configuration.

```
ziti edge create config website.intercept.v1 intercept.v1
'{"protocols":["tcp"],"addresses":["website.ziti"],
"portRanges":[{"low":80, "high":80}]}'
```

3.4.2 - Host.v1

This configuration concerns how the tunneler on the web-server will receive the traffic from the OpenZiti Network Overlay and where it has to forward it.
Same as Intercept.v1 config type, this configuration requires 3 essentials parameters, but not with the same definition:

- Address: the destination address where the traffic has to be forwarded (endpoint address), it can be 127.0.0.1 on the tunneler's host or any other IPs.
- Port: the destination port of the endpoint, same principle of the destination address. Can be a range if the intercept.v1 is configured for.
- Protocol: the protocol used to communicate with the endpoint. Can be tcp or udp.

Our address will be 127.0.0.1 because the tunneler is on the web-server, and we want to access to the web-server.
The port will be 80 and the protocol TCP.

Here is the JSON representation of this configuration:

```
{
    "protocol":"tcp",
    "address":"127.0.0.1",
    "port":80
}
```

We can now, on any machine where you are logged with the “ziti edge login” command, create via API an host.v1 configuration.

```
ziti edge create config website.host.v1 host.v1 '{"protocol":"tcp",
"address":"127.0.0.1", "port":80}'
```

3.4.3 - Service

Once we have our two configuration files stored into the controller database, we can start to create the service.
The command takes the two configurations in input:

```
ziti edge create service website.svc --configs
website.intercept.v1,website.host.v1
```

3.5 - Policies Configuration

The configurations and the service are now created but we can’t use them because no policies are applied to them. By default, services as well as identities have no rights (least privilege).
So we have to apply two policies on the service:

- Dial: this policy allows an identity (loaded on a tunneler for example) with a specific role/attribute to Dial a service. In other words, it allows an identity to initiate a connection to an endpoint.
- Bind: this policy allows an identity with a specific role/attribute to Bind a service. In other words, it allows an identity to listen/wait for a connection.

We have to define arbitraries identity’s roles for both client and server.
Let’s say the client (workstation) will have the “webclient” role and the server (web-server) will have the “webserver” role.

Create the first policy for letting workstation to Dial the service:

```
ziti edge create service-policy website.policy.dial Dial --service-roles
"@website.svc" --identity-roles '#webclient'
```

Create the second policy for letting web-server to Bind the service:

```
ziti edge create service-policy website.policy.bind Bind --service-roles
'@website.svc' --identity-roles "#webserver"
```

We are almost done! All we have left to do is to update the two identities “workstation” and “web-server” by adding their proper roles:

```
ziti edge update identity "workstation" -a "webclient"
```

And for the "web-server":

```
ziti edge update identity "web-server" -a "webserver"
```

You are done! If you have both tunnelers running, they will automatically receive an access to the service regarding the policies set up.
If the tunnelers are not started, start them with this command:

```
ziti-edge-tunnel run -i ~/$(hostname)-tunneler-identity.json
```

You can now from the workstation access to the web page with curl:

```
curl website.ziti
```

You should see the website code returned by the web-server:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
[...]
```