Report

Project: Personal secure wormholes

Subject: Advanced Network Security

Authors:
Daniel Silva 51908
Nuno Marques 65002
Rui Lebre 64090

Index

Introduction

In order to create a personal communication channel over unsecure or, at least, non trustable networks, the design of this project was proposed.
The goal is to build a personal secure wormhole (PSW), a secure cryptographically tunnel that must encapsulate network packets of direct interactions over applications of different clients.
Transparently to common users, the applications from each end-user network should be able to interact each other without reconfiguring their networks.

To accomplish the goal, it was developed a system on Linux operating system, taking profit of technologies like *iptables*. The project was also developed using Python 2.7 (mainly).

Architecture

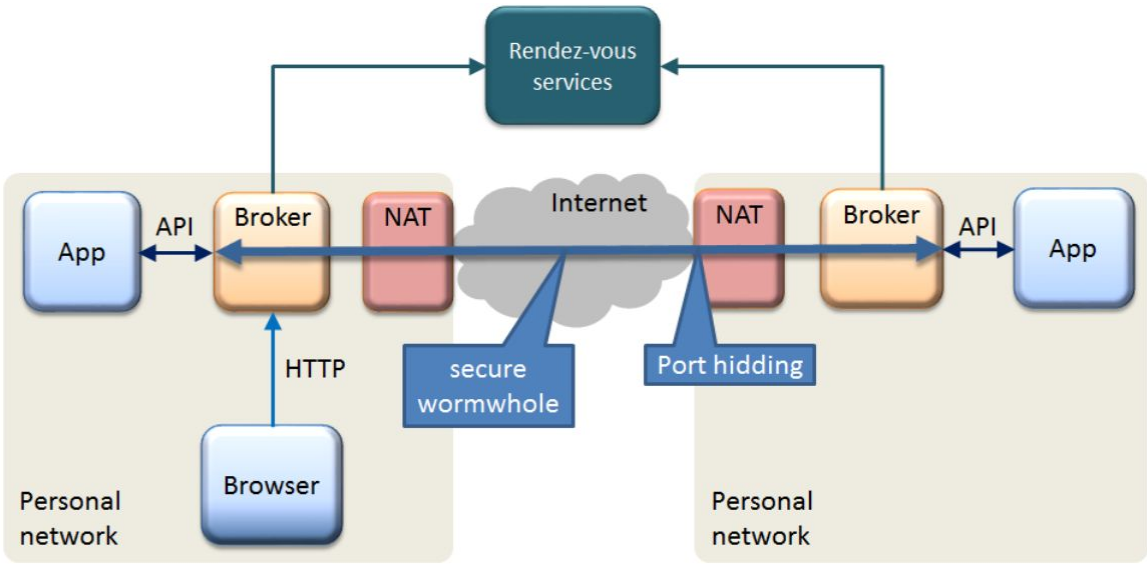
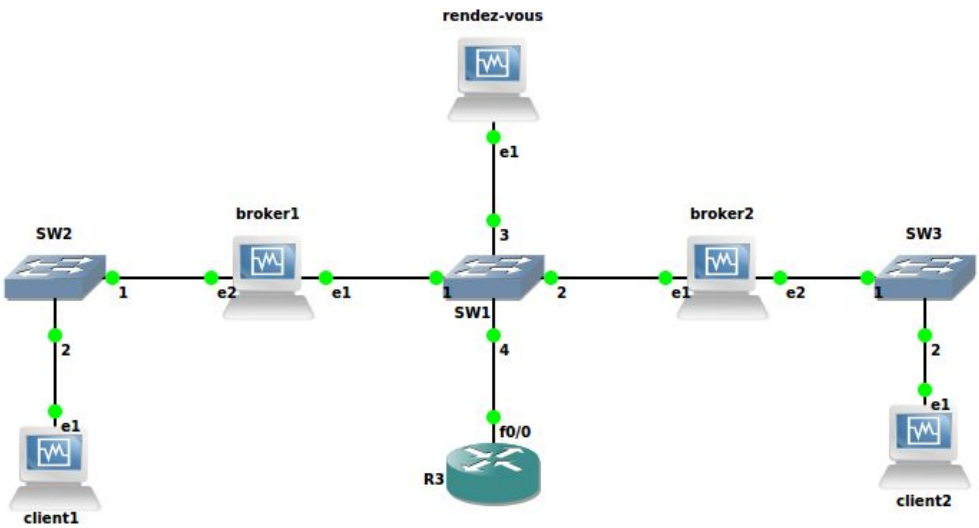


Figure 1 - System wide purposed architecture

It was developed for each part (client) modules and scripts that will configure and allow to run the personal secure wormhole tunnel.
Each one of the applications that want and be able to communicate must be registered at the module called Broker (see image above) so that way, the services available can be transmitted to rendez-vous services and by that way be public to all other clients.

It was also inserted a module to prevent unwanted or non authorized communications named port hiding.

All traffic transmitted after the previous configurations will be cryptographically secured using ssl tunnels.



Implementation

Firewall

It was developed a firewall so that communications from outside cannot communicate with the machine if not allowed or aware of ongoing protocols. To do it so, a set of Linux *iptable* rules were created:

1. ICMP packets (*ping*) are accepted and reply is provided
2. Communications on loopback interface are allowed
3. TCP connections to port 8000 are fully allowed (required to access Broker interface)
4. A pre-defined set of ports (usually 3) are open but no answer is provided when interactions occurs (see below)

To start connections (like a SSH session, on port 22), it is required that a set of interactions is made. In this case, it must be performed a set of TCP connections on defined ports (e.g. 1111 -> 2222 -> 3333) and no one of them will take any response.

After the sequence is achieved, for 30 seconds (configurable value), it is allowed the establishment of a connection on port 22 (SSH), which remains open as much as is required (once connection is established, and only for that connection).

See Figure 2 below for further graphical details.

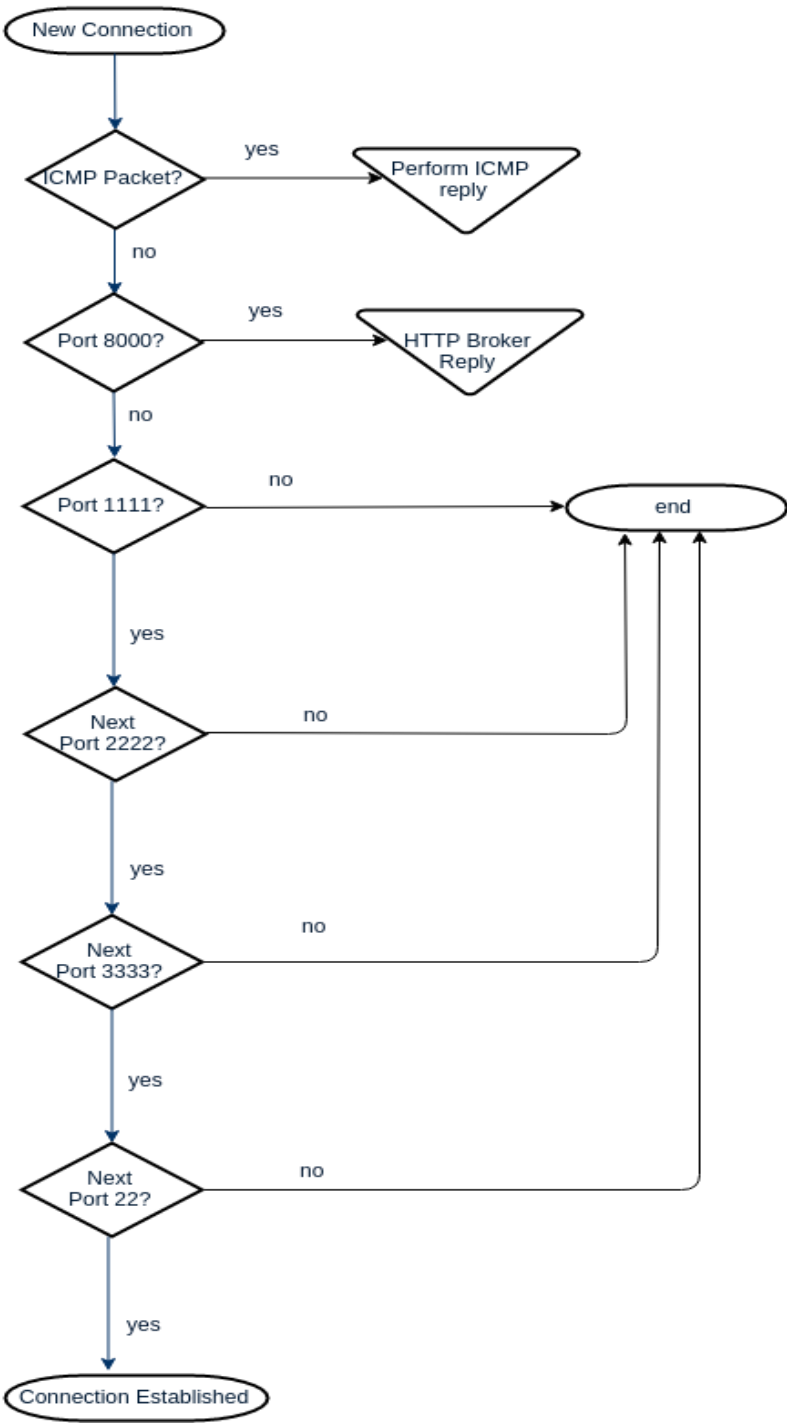


Figure 2 - Flowchart of communication establishment

Port-knocking

After previous configuration, it is needed to perform the actions so port 22 can be accessed. To do that, it was developed a script, written in Python, that allows the user to transparently to do the *port knocking* automatically.
So that can be executed in Linux operating systems, it is requested to enable port forwarding, so a bash script was written, and it execution must be with root privileges (e.g. $sudo activate_ip_forward.sh)

Broker

Broker is a service that runs on port 8000 of each client and it goal is the registry of all locally available services.
Alongside graphical interface (Web) available, it is also provided a REST API to, programmatically, so it be possible the automatic registry of available apps.
Here is stored the basic information of each example such as Name, IP, Port and Description.

Available REST API

HTTP Method	**URL**	**Parameter type**	**Parameter format**	**Return**
GET	/api/services	-	-	- 200 OK - 400 BAD REQUEST - 404 NOT FOUND

POST	/api/services	Json Body	{
			"name": "Chat em pc",
			"description": "Chat point to point",
			"ip": "10.0.0.10",
			"port": "12345"
			}
				- 200 OK.
				- 400 BAD REQUEST

DELETE	/api/services/del/{pk}	Path Param	{pk} = ID of registered service	- 200 OK. - 404 NOT FOUND - 400 BAD REQUEST
DELETE	/api/services/del/{ip}/{port}	Path Param	{ip} = IP where service is running {port} = Port where service is running	- 200 OK. - 404 NOT FOUND - 400 BAD REQUEST

Available interface

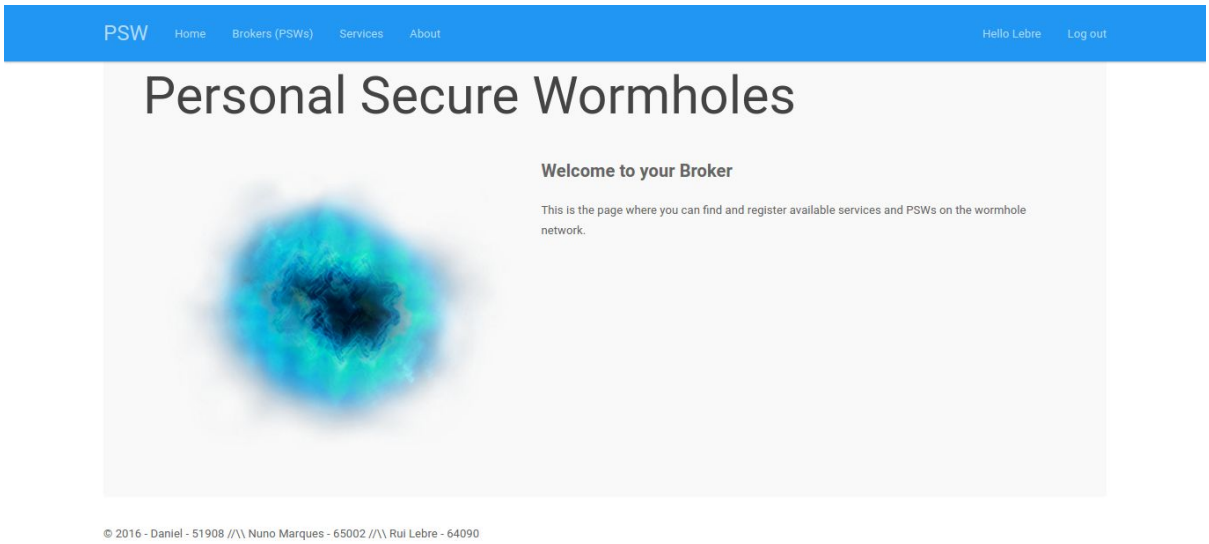


Figure 3 - Broker's home page

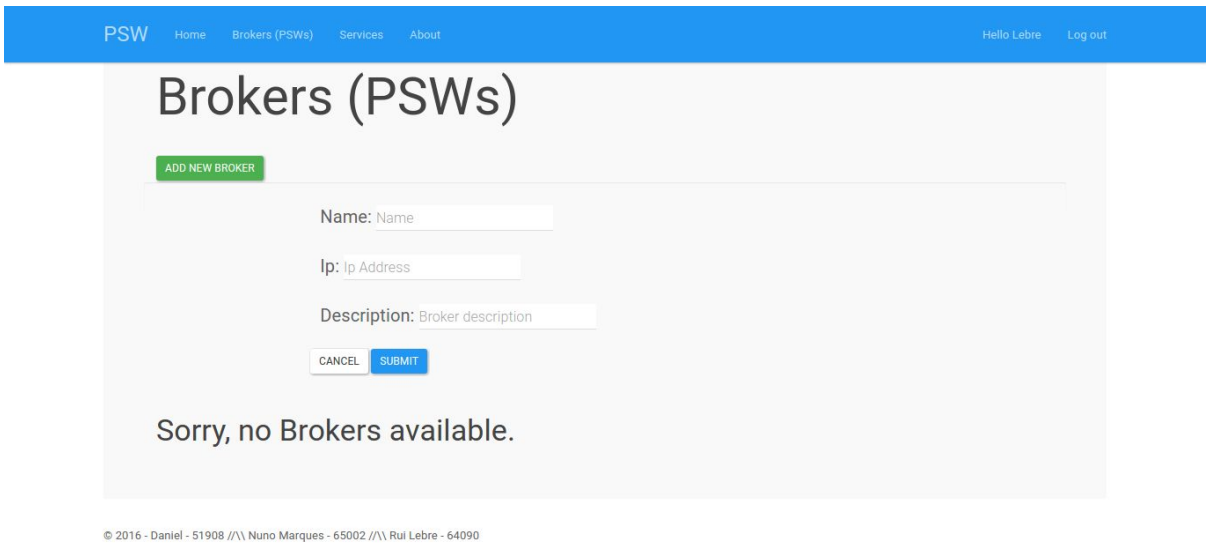


Figure 4 - On this web page form, it can be added new brokers so that services from that broker can be listed

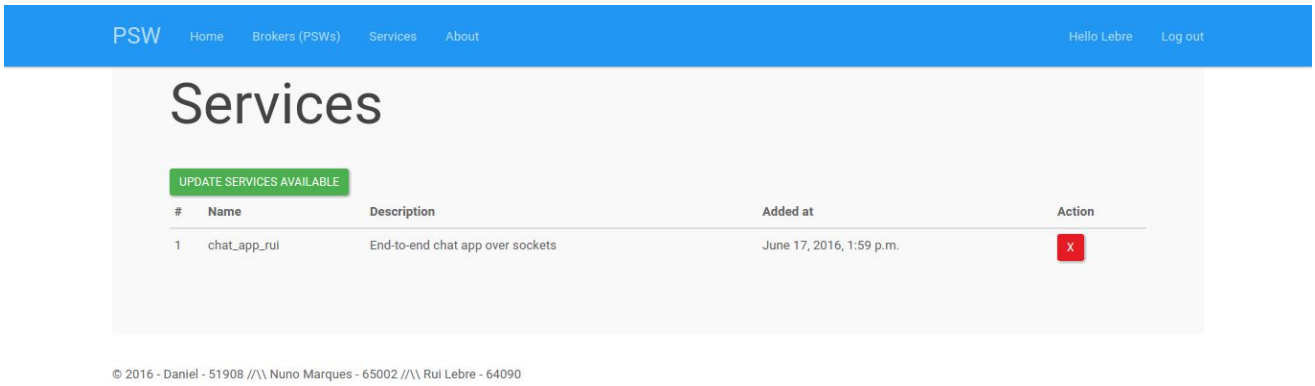
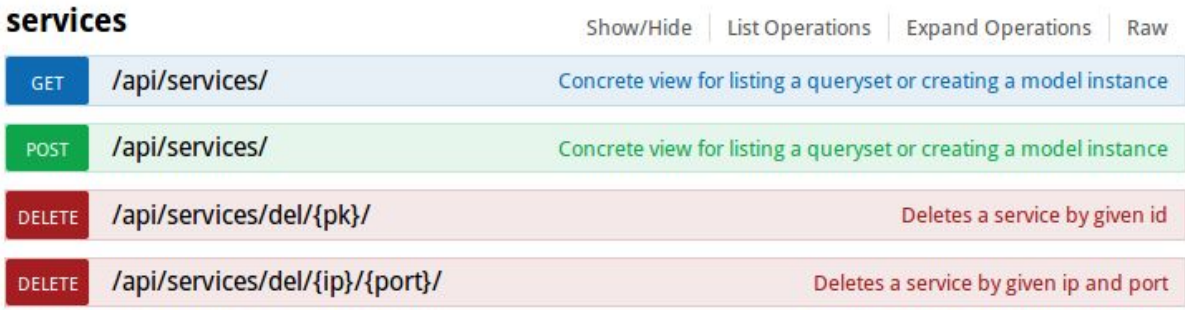


Figure 4 - On this web page form, it can be added new broker services so that those services can be available to communicate

Rendez-vous

The rendezvous server is a public service that has as main objective the list of available clients and, for each customer, all services running for establishing PSWs.
The information here housed is provided by each broker as soon as it receives the registration of a service from a client.
After that, that information is available publicly so all clients can access brokers and services of each through web interface of its broker.



Chat App

Chat app deployed is built under 2 threads: a receiver thread and a sender thread. On start, the application will automatically register the receiver thread so that it can be connected from outside.

So, on start up, app chooses a port where receiver is listening and asks for an IP and Port to connect. After that, the other client terminal can access that information and connect. A simple two way connection is set over TCP without encryption. This last clause is guaranteed by SSH tunnel.

Figure 5 - Chat application example

Used libraries / modules / components

Firewall

```
#!/bin/bash

echo 1 > /proc/sys/net/ipv4/ip_forward
echo Done
```

```
# Generated by iptables-save v1.4.21 on Tue May 24 23:22:50 2016
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [11:1088]
:GATE1 - [0:0]
:KNOCKING - [0:0]
:PASSED - [0:0]
-A INPUT -p icmp -m icmp --icmp-type 0 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8000 -j ACCEPT
-A INPUT -j KNOCKING

-A GATE1 -p tcp --dport 1111 -m recent --set --name AUTH1 --mask 255.255.255.255
--rsource -j DROP

-A GATE1 -m recent --remove --name AUTH1 --mask 255.255.255.255 --rsource
-A GATE1 -p tcp --dport 2222 -m recent --set --name AUTH2 --mask 255.255.255.255
--rsource -j DROP

-A GATE1 -m recent --remove --name AUTH2 --mask 255.255.255.255 --rsource
-A GATE1 -p tcp --dport 3333 -m recent --set --name AUTH3 --mask 255.255.255.255
--rsource -j DROP

-A GATE1 -j DROP

-A KNOCKING -m recent --rcheck --seconds 30 --name AUTH3 --mask 255.255.255.255
--rsource -j PASSED
-A KNOCKING -m recent --rcheck --seconds 10 --name AUTH2 --mask 255.255.255.255
--rsource -j GATE1
-A KNOCKING -m recent --rcheck --seconds 10 --name AUTH1 --mask 255.255.255.255
--rsource -j GATE1
-A KNOCKING -j GATE1
-A PASSED -m recent --remove --name AUTH3 --mask 255.255.255.255 --rsource
-A PASSED -p tcp -m tcp --dport 22 -j ACCEPT
-A PASSED -j GATE1
COMMIT
# Completed on Tue May 24 23:22:50 2016
```

Port-knocking

```
#!/usr/bin/env python

import argparse
import socket
import sys


parser = argparse.ArgumentParser()
parser.add_argument('host', metavar='HOST', type=str,
                    help='Hostname to knock at')
parser.add_argument('ports', metavar='PORT', type=int, nargs='+',
                    help='Port(s) to use, in order specified')
parser.add_argument('-t', '--timeout', type=int,
                    help='Timeout for connection attempt (seconds), default 10')
parser.add_argument('-v', '--verbose', action="store_true",
                    help='Show detailed information')

parser.set_defaults(timeout=10)
args = parser.parse_args()

TCP_IP = args.host
TIMEOUT = args.timeout
VERBOSE = args.verbose

ports_failed = []

for TCP_PORT in args.ports:
    if VERBOSE:
        sys.stdout.write("Knocking on port ")
        sys.stdout.write('{0: <10}'.format(str(TCP_PORT) + '...'))
        sys.stdout.flush()

    sock_msg, sock_ok = None, True

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(TIMEOUT)
        s.connect((TCP_IP, TCP_PORT))
        sock_msg = "open"
        s.close()
    # timeouts are knocks, too
    except socket.timeout, e:
        sock_msg = "no answer"
    except socket.error, e:
        ports_failed.append(TCP_PORT)
        sock_msg = "%s" % e
        sock_ok = False

    if VERBOSE:
        if sock_ok:
            sys.stdout.write("OK")
        else:
            sys.stdout.write("FAILED")
        if sock_msg:
            sys.stdout.write(" (%s)" % sock_msg)
        sys.stdout.write("\n")
        sys.stdout.flush()

if len(ports_failed):
    s_ports = ", ".join([str(p) for p in ports_failed])
```

```
    print "\nFailed ports: %s" % s_ports
    sys.exit(1)
else:
    sys.exit(0)
```

Broker & Rendez-Vous

Broker and Rendez-Vous services were developed using Django and its API is up and running using REST Framework

SSH Tunnel

To perform encrypted communications, a module was developed and configured using socks.

Configurations to deploy the project

Deploy Broker and Rendez-vous Servers (they are in different hosts each):

- # apt-get install virtualenv python-pip python-dev
- $ git clone https://code.ua.pt/git/sar-1516-proj-g1
- $ cd sar-1516-proj-g1
- $ virtualenv ~/sar-venv
- $ source ~/sar-venv/bin/activate
- $ pip install -r requirements.txt

Deploy Broker Servers:

- $ source ~/sar-venv/bin/activate
- $ cd sar-1516-proj-g1/Broker
- $ python manage.py makemigrations
- $ python manage.py migrate
- $ python manage.py runserver 0.0.0.0:8000

Deploy Rendez-vous Server:

- $ source ~/sar-venv/bin/activate
- $ cd sar-1516-proj-g1/rendezvous
- $ python manage.py makemigrations
- $ python manage.py migrate
- $ python manage.py runserver 0.0.0.0:9000

Deploy firewall

- $ cd sar-1516-proj-g1/portknock
- # iptables-restore iptables.rules

References

- http://tutorial.djangogirls.org/en/django_start_project/index.html
- https://www.digitalocean.com/community/tutorials/how-to-configure-port-knocking-using-only-iptables-on-an-ubuntu-vps
- http://www.microhowto.info/howto/implement_port_knocking_using_iptables.html
- https://wiki.archlinux.org/index.php/Port_knocking#Port_knocking_with_iptables_only
- http://www.securitygeneration.com/wp-content/uploads/2010/05/An-Analysis-of-Port-Knocking-and-Single-Packet-Authorization-Sebastien-Jeanquier.pdf
- http://www.sans.org/reading-room/whitepapers/sysadmin/port-knocking-basics-1634