Malicious context and workaround analysis of decentralized VPN: the case of Mysterium Network

Jacopo Federici

March 13, 2020

Contents

Abstract

0.1 Work structure

0.2 Outline

0.3 Goal

1 Introduction

Privacy is increasingly becoming a central topic of discussion in many different fields, as the need to protect our data is becoming critical. The massive digitalization of many aspects of our life implicitly exposes us, and our data to possible attacks. Even the simplest operations over the internet are becoming unsafe.
In the last years, we have seen security solutions integrated into popular applications, such as browsers or mobile apps, to prevent increasing attacks to users' security. For instance, in 2017, the popular browser Google Chrome, announced that it would have marked as non-secure any website using the HTTP protocol instead of HTTPS. The HTTPS protocol is the standard HTTP protocol that uses SSL/TLS certificates, an additional security layer to make the communication between clients and servers secure. Before that, connections were not protected by any encryption algorithm.
Even though the first HTTPS protocol was proposed by Netscape Navigator b in 1990, it is only in more recent years that we started assisting to a global trend to increase security levels, often achieved BY encrypting communications.
The Chrome browser initiative is part of a global trend to increase the security level, often achieved encrypting the communication: instant messages applications, internet protocols like the DNS, industrial system infrastructures, and many others.
Although the level of security of the internet has substantially increased, there is still a wide range of situations and scenarios in which the individual user browsers the internet in an unsafe way.
A solution to increase the security level is provided by the Virtual Private Network (VPN), a technical solution that allows extending a private network across a public network. It creates a virtual point-to-point connection by using a tunneling protocol over the existing network, and the tunneled traffic is encrypted. A first proposal of the VPN specification is

around 2000 [2], but only in the last years, the use of VPN became massive. The VPN data encryption feature is solving privacy problems of various nature, such as to exit a censored network, using services available or blocked in certain countries, avoiding tracking methods and other similar scenarios.
Currently, there are commercial solutions that allow users to have a private connection to the company's servers across the world, but there are many security concerns about those solutions. These companies can keep the connection logs, inspect the traffic, or sell statistical data.
Some open-source projects were, lately, born to solve these issues. Their goal is to revolutionize the idea of VPN and its uses, building a rich network of peers where two nodes can connect through a VPN. The projects provide a platform that includes all the components needed to have an efficient, fast, and easy-to-use system, from the discovery to the payment.

In this research, we analyzed the most developed project, Mysterium Network, as a case of study. As we write, Mysterium offers a working network composed of hundreds of nodes around the world, making it the best project to investigate malicious contexts and to design vulnerabilities, owns by the decentralized architecture, more than the Mysterium project itself.
We even discover how secondary aspects are pivotal to guide attacks to the system.
Then we focus on the solutions that can patch those critical aspects and provide a comparison of them with an eye on the practical perspective.

1.1 The state of the art

The world is changing at a fast pace.
First, everything is now much faster. The internet infrastructure allows us to move significant amounts of data in just a few seconds. Furthermore, our personal computer can perform operations that were impossible to execute some years ago on mainstream servers. This speed increases the computational power, and it is jeopardizing many security solutions, such as ciphers that based, and still base, their strength on the practical difficulties to solve the mathematical problems at the foundation of cipher methods.
Second, the architecture is moving back to a cloud-based infrastructure. Many workflows see, again, a thin client sending the execution step to a

cloud server that elaborates the output that is then sent back to the client. This architecture changes the surface of the attack targeting the cloud servers as the principal attack destination.
Third, some technologies are mature enough to provide an excellent alternative to traditional models — for instance, the cryptocurrency model against the bank model. Here, the changes are in the paradigm, and the reasons behind these new approaches are much more than technological.
The security noun is changing the meaning inside the people's minds, from "how thick is my bank caveau" to "how long my bank requires my new online account password"

1.2 the need for trust and privacy

https://www.geeksforgeeks.org/need-of-information-security/
https://www.hackerfactor.com/blog/index.php?/archives/868-Deanonymizing-Tor-Circui
https://www.howtogeek.com/133680/htg-explains-what-is-a-vpn/
https://www.forbes.com/sites/tjmccue/2019/06/20/why-use-a-vpn/#3f08d7105859
https://www.investopedia.com/terms/b/blockchain.asp
https://www.howtogeek.com/350322/what-is-ethereum-and-what-are-smart-contracts/

https://en.wikipedia.org/wiki/Proof_of_work
https://en.wikipedia.org/wiki/Proof_of_stake

Technology is becoming an essential tool to exercise the people's rights. There are areas of the world where these rights are not accessible, and people fight to obtain them. For example, in the People's Republic of China, the government built a firewall to censure the internet. It is called the Great Firewall Of China, abbreviated in GFW, and it is a combination of laws and technologies to control the internal domestic internet, so-called censorship. It blocks social networks, search engines, and many other services that declined the government's request to make the traffic clear and detectable.
In this context, aspects such as anonymization, become not only relevant but crucial to be preserved.
As for anonymization, a solution to obtain identity obfuscation is the VPN. It is massively used to protect data from surveillance at any level, from a public unsecured wifi connection to the country Internet Service Providers.

VPNs are provided by companies that often do not act genuinely, making the identity or the traffic detectable or, even worsts, sniffing and crafting the data for malicious actions.

The decentralized VPN completely changes the paradigm, the design of the infrastructure.
An analogy can be made with the traditional bank model: as we know, the bank is a centralized entity trusted by definition. What if the bank suddenly becomes untrusted? Cryptocurrencies spread among ourselves, the cryptocurrency owners and network participants, the trusts we put in the bank, using a blockchain as a technical solution. A similar design is used with decentralized VPN: there is no central entity to control the system, but a significant number of peers, the network, we can choose to connect.

1.3 Decentralized systems

1.3.1 dVPNs

A decentralized VPN is a network of nodes where the users can connect through a VPN to nodes without a central entity. It allows the user to create a private tunnel with another user, making encrypted all the traffic between the two points. The decentralization system, in this context, includes several steps to make the final connection happening, rather than a simple VPN connection between two peers.
First of all, the user creates an identity into the system, linked to the payment wallet, using a client application. Then, he connects to the network providing his availability to be a consumer or a provider.
The consumer user chooses the provider he prefers based on different features such as the region, quality, or cost of the node, and asks for a new connection to that server. In this step, the blockchain, based on Ethereum, allows creating a smart contract between the two participants with a predetermined amount of money. Using commercial VPN applications, such as OpenVPN or Wireguard, the connection between consumer and provider, otherwise called client and server, is established. When the connection correctly ends, the network closes the smart contract, that registers the result of the transaction in the blockchain.
We outlined the main steps from which we can think of a decentralized VPN more as a platform solution rather than a simple VPN application. Indeed,

other projects suggested the VPN is one of the features a decentralized system so composed can have. (EXPLAIN BETTER THE IDEA)

1.3.2 The business idea

As a business project, which aims to earn money, the project has a specific business plan: provide a system network where users can join as a service provider, selling their internet bandwidth, or as a service consumer, buying other's internet bandwidth. For every cryptocurrency transaction, the network applies a fee that wants, substantially, to create a market to manage.
Each couple of entities, a consumer and a provider, agree on a transaction paid using cryptocurrency in the face of the used service. This transaction is regulated by an Ethereum smart contract signed by both sides and requires a fee to be closed. This fee is not the Ethereum fee, generally called gas, but the precisely Mysterium source of earnings. For this reason, Mysterium and other projects created their coins used to buy and sell internet bandwidth between the nodes inside the network. The coin name is MYST for Mysterium Network and, as a regular cryptocurrency, it is possible to convert MYST to ETH and vice versa.
Although a node can decide to be a consumer or a provider at any moment, the two entities are different, and the type of market is called a two-sided market where bid and ask respectively grow on the other side growth. (EXPLAIN BETTER)

(BID AND ASK GRAPHIC DEPENDING ON THE FEES)

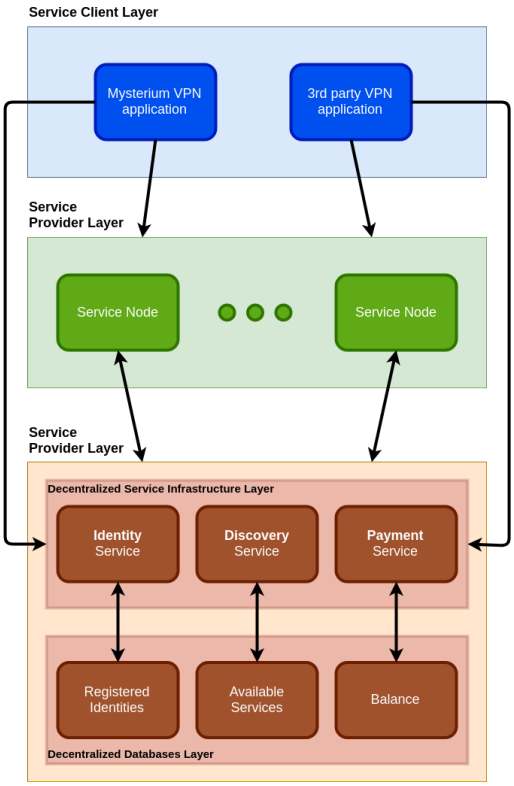
```
concetti introduttivi
- blockchain in linea di massima\\
- ethereum in linea di massima e descrizione degli smart contracts\\
- proof of work, proof of stake\\
- reputazione in una net (Eigentrust)\\
- - cercare applicazione pratica di Eigentrust\\
- descrizione di BFT\\
```

2 Mysterium Network

The main open-source projects which aim to create a decentralized VPN are Mysterium Network, Privatix, Substratum, and Sentinel. We explored all of them, and we can safely say the most advanced project is Mysterium Network, at the time we are writing this research.
It is the most completed in terms of development and design. Furthermore, it's 100% open source, that for this kind of project is a requirement which conditions the users' adoption and the market diffusion.
Therefore, we decided to use the Mysterium Network as a test case to analyze the malicious context the project is exposed to and propose our proof of concepts as solutions.

2.1 Architecture

Mysterium Network is an open-source project. Its main goal is to build a decentralized infrastructure made up of layered VPN protocols, blockchain, and smart contracts.
As described in figure X, we can distinguish between a service client block and a service provider block. Both of them lean on the Ethereum blockchain block, which is a distributed database where all the network operations, the identities, and the service proposals are recorder.



As a general workflow description, Mysterium starts by collecting all the agents' service proposals and presents them to the clients. Then it registers the client's intention to pay for the agent's service chosen by the user and creates the VPN connection between the two points. The outcome of each transaction is saved in the blockchain.

To implement all the functionalities just said, Mysterium has four core components inside its architecture:

- **Ethereum Blockchain** allows running decentralized code with smart contracts, enabling reliable services and payment handling.
- **Identity service and database of registered identities** ensure the proper identity acknowledgment between client and service provider.
- **Discovery service and database of available services** provide means to announce the availability of VPN services and to pick the most suitable VPN service.

- **Payment service and database of balances** allow secure promise-based micropayments for services.

2.1.1 Identity Service

The **identity service** manages the network identities that every node needs. A single node can create one identity or more, and each of them consists of a public and a private key. The last 20 bytes of the hash function, applied on the public key, is the unique id used to identify the node into the network. To publish a new identity so that other network members can reach it, Mysterium uses the smart contract mechanism. The private and public keys are used to validate the signature for the communication between the nodes.
The blockchain mining process appends the public key and the unique id to the Ethereum blockchain after the node pays a fee to the network. This amount of money has the purpose of giving value to the identity. In such a way, to abandon the identity to create a new one is unattractive, because it is expensive, but not impossible.
We will discuss those aspects later, but we can surely say this is the first barrier to block in mass identity creation.
Furthermore, the user benefits in using the same identity because it is possible to rebuild his transaction history and his balance from the Ethereum blockchain, making the user more predictable and thus trustworthy for service providers.

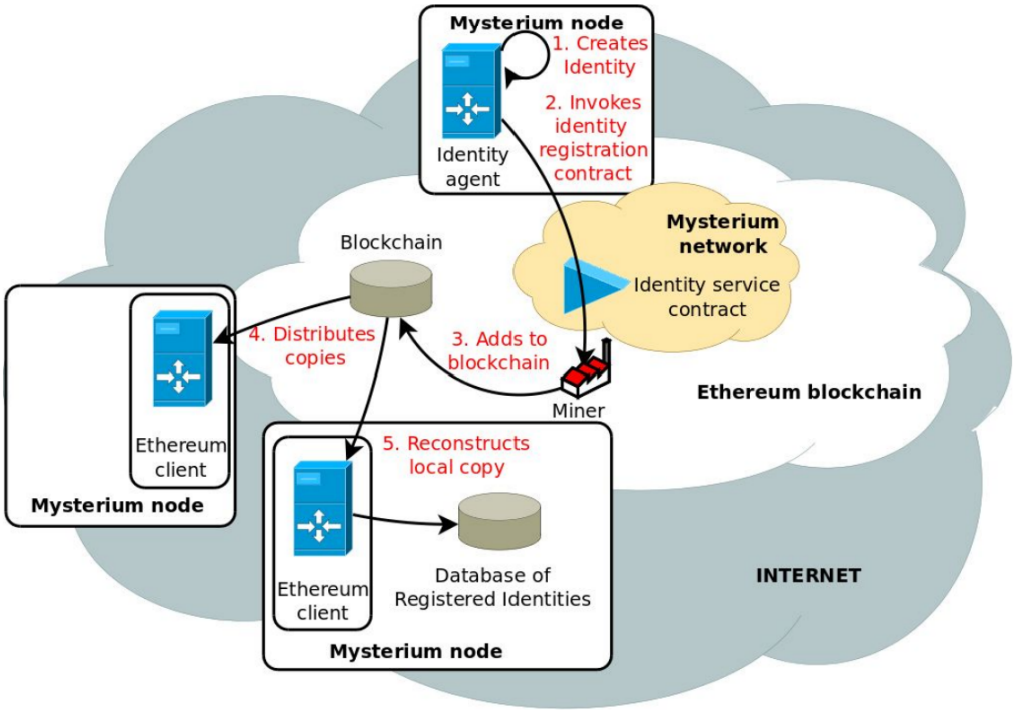


Illustration: Identity data registration and replication

2.1.2 Service Discovery

The **service discovery** provides the list of service proposals available in the network. The service posts its service offer characterized by some service parameters: those parameters can be the VPN application the provider supports, the provider's connection IP address and port, the provider's server location, or the bandwidth per session. Then, the miners append the proposal to the blockchain, making it publicly available for everyone.

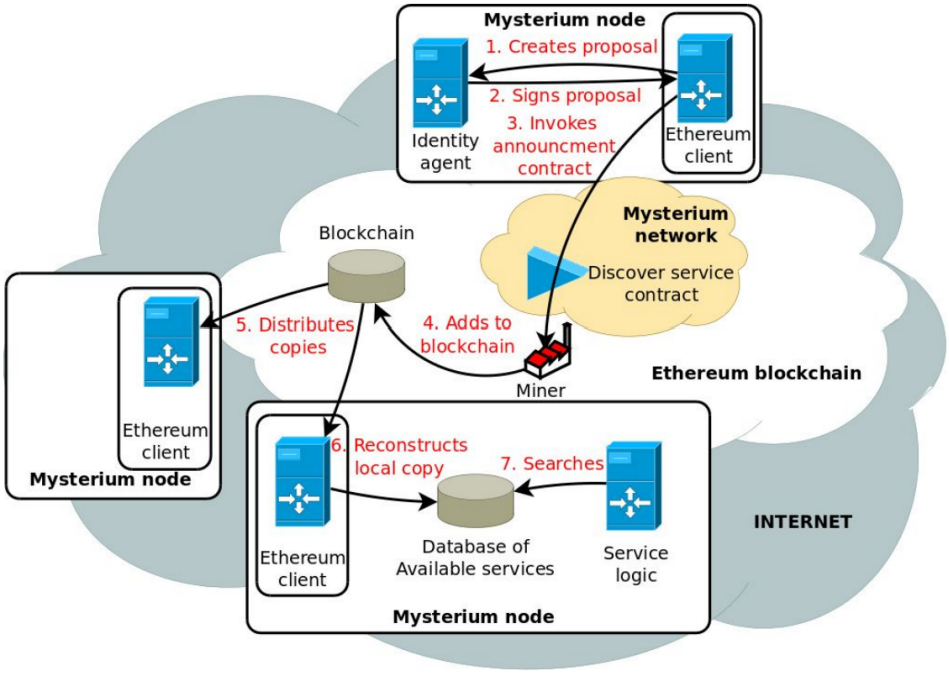


Illustration: Service announcement and discovery

2.1.3 Payments Handling

Ethereum blockchain holds all the mechanisms for the **payments handling**. Therefore everyone needs an account to operate in the Mysterium Network. To advance a request, the consumer has to deposit currency in his wallet. Then, he can ask for a connection to a provider who has previously published a service proposal. Mysterium describes this step as a promise creation because the client effectively promises he will pay when the service ends. Positively clearing a promise is the everybody's best interest, the producer and the consumer, since a canceled promise marks the consumer's history hitting his reputation for future connections.
From what we have described so far, it seems the consumer can decide to avoid paying for the used service: technically, it is possible, but realistically there are some considerations to be made.
First of all, as previously described, identity creation takes money to be completed, and for any new connection, the user's balance has to be

positive. In any case, the blockchain records any connection result so that everybody can rebuild the user's history at any moment. Therefore, it is possible to deduct the user's reputation based on its history.
Using this reputation, the provider can decide to block his service to all users that have a low reputation caused by an unpaid precedent. On the other hand, other providers allow new users' connections with a low reputation but at a high price, accepting the risk of having an unpaid transaction.

2.2 Components

The core application is called **node**. It's written in GoLang, a new program language made by Google that is having a high adoption in the last years. The node creates a local webserver to connect to the network, manages the cryptocurrency wallet, and initiates the VPN connection as a client or manages the incoming connection as a server. Mysterium leans on famous VPN applications, which are OpenVPN and Wireguard.
The node can be run in two modes, as a client or as a server, and it is managed via command-line interface or RESTful API, called TequilaAPI, exposed in the localhost.

The main functionalities of TequilaAPI concern:

- the authentication,
- the connection,
- the identities,
- the location,
- the proposals,
- the service operations in provider mode
- other minor features

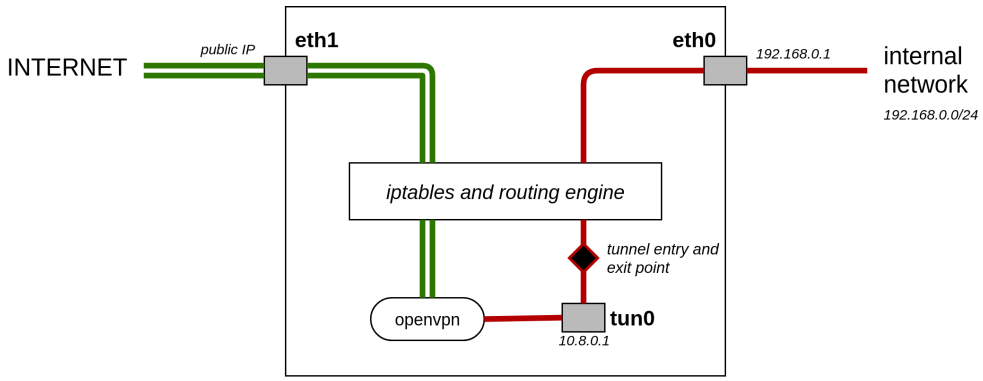
TequilaAPI is used by the client's application to manage the connection and for diagnostic operations.
Mysterium developed a desktop client written using Electron, which uses the TequilaAPIs.

The node is also available as a ready-to-use Docker image and published on the Docker Hub.
On the mobile side, Mysterium is present with its multiplatform app that works only as a client.

(MORE ON MOBILE)

2.2.1 Technical description

We have chosen the Linux platform and OpenVPN as the VPN application, but similar considerations can be made with different configurations.
When the Mysterium application starts, it calls OpenVPN, which sets up its network environment. OpenVPN creates a virtual network device used to encrypt and decrypt the traffic. These devices reside in the kernel and are entirely virtual, which means there is no hardware on their back.
They can be of two types, TAP or TUN.
The TAP type is used to simulate a link layer device, which means the corresponding ISO model level it refers to is number two. Therefore, a TAP interface handles the transmission of the data frames between two nodes on the same physical layer. The behavior is equal to a bridge device: to connect two separates networks as if they were one.
On the other hand, the TUN type is, in the ISO model, one layer above, working at level three. A TUN interface handles packets, such as Internet Protocol packets. The main functions include port forwarding, host addressing, and traffic control, for example. The behavior is, in this case, equal to a router device: to connect two separates networks allowing their communication but keeping them independent.
Usually, the TUN type is the most used, since the TAP type brings some limitations such as it cannot be used with Android or iOS devices, causes higher overhead on the VPN tunnel, and scales poorly.
The following schema describes an example of a scenario where the network has two ethernet cards, one public and one internal.
The virtual TUN interface created by OpenVPN is `tun0`, the public interface is `eth0`, and the internal interface is `eth1`.



When a packet arrives at the `eth1` interface, OpenVPN picks it up, decrypts, and sends the data to the`tun0` interface. The `iptables` and the routing engine filter and masquerade the packet and send it to`eth0`, where it becomes available to the final receiver.
In the other direction, when the packet arrives at `eth0` interface, `iptables` and the routing engine apply their rules to filter and masquerade, and then they forward it to `tun0` interface. Thus, OpenVPN picks it up, encrypts the data, and sends it to the public `eth1` interface.
In our scenario, the network configuration is slightly different and does not include an internal interface network. As the schema below describes, OpenVPN sends the traffic coming from `tun0` to the public interface`eth0` on a specific port. The client application listens on that port, and the OpenVPN client decrypts the received data.

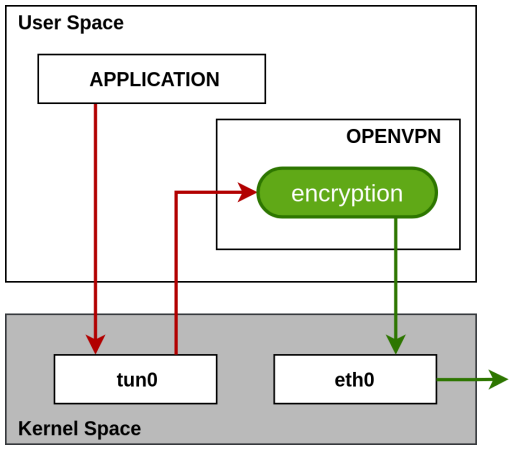


IMAGE TO EDIT AS BEFORE DESCRIBED

2.2.2 Start a node

There are two ways to start a Mysterium node. The first one is using the Docker image: Docker is a platform that uses the OS virtualization to deliver software in packages called containers. The image contains all the specifics that the application requires to be executed.
From the software image, Docker creates as many containers as we instantiate, and they are the virtualized software we will use. We can think of the software image as the house project and the software container as the real house construction. As the house construction can have personalizations, either the container is set up passing parameters to customize the network, the storage, or other settings such as the resources allocated by Docker to the single container.

With the following command, we pull the image of the Mysterium node to the Docker platform:

```
1  docker pull mysteriumnetwork/myst
```

Then, we create the container using:

```
1  sudo docker run --cap-add NET_ADMIN --net host --
     name myst -d mysteriumnetwork/myst service --
     agreed-terms-and-conditions --openvpn.port 1194
```

The parameters specify, from left to right:

1. to add the network capability to the operative system, which is Linux,
2. to connect the container to the net,
3. to name the container `myst`,
4. to run the container in background mode and detach it from the shell.

The remaining values are passed to the Mysterium software. The `service` value means Mysterium node starts as a node, thus a service provider, instead of a client, the next one is to agree on the terms and conditions, and the last one is to set up the OpenVPN port.
When the container is up and running, we can check the logs using:

```
1  sudo docker logs -f myst
```

We can find relevant details reading the log, such as the identity registration and the node proposal:

```
1  ...
2  [Mysterium.api] Identity registered: 0
      x4cd126119cd14e38c90e34dd8b6e0e2174b71123
3  [Mysterium.api] Proposal pinged for node: 0
      x4cd126119cd14e38c90e34dd8b6e0e2174b71123
4  ...
```

The identity registration is the unique ID assigned from the network to the node. With this ID, it is possible to discover it, its connection details, and transaction history on the Mysterium Network website.
The second way to install the Mysterium node is using the Debian package. To manage its execution, we use the `systemctl`, a management tool for controlling the init system:
We run

```
1  sudo systemctl daemon-reload
```

to reload the daemon after the changes to the node settings, and

```
1  sudo systemctl restart mysterium-node
```

to reload the node.
If the node is up and running, it exposes a web interface on the localhost available at port 4449, where it is possible to control the service status and see connection statistics.

2.2.3 Connect to a node

When the node works, the simplest way is to use the mobile or desktop client applications that provide a graphical interface to choose the server node and establish the connection. In a few seconds, the Mysterium network creates the connection record in the blockchain, initializes the connection and, then, the VPN takes place.
Another way to connect to the service provider is using the command-line interface the node as a client exposes. Referring to the node installed by the Debian package, the command to run the node and control it is:

```
1  myst cli
```

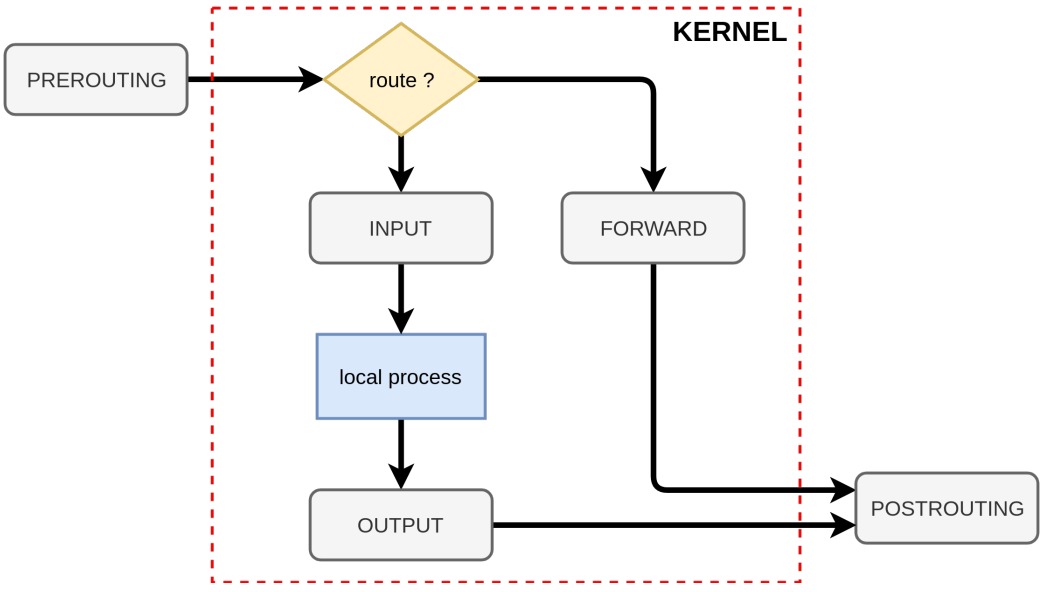
We then enter in a view where we can perform the operations on the node that regard the identity management, the connection creation, and the status diagnostic.

2.2.4 Iptables

The `iptables` is a framework to manage the network decisions. It is based on the Netfilter kernel component, which exposes a set of hooks that are triggered at specific points of the network stack. Every packet that enters the networking system will trigger these hooks as it proceeds through the stack so all the programs that register with these hooks can interact with the traffic.

Five different hooks correspond to different points of the stack:

- `NF_IP_PRE_ROUTING`
- `NF_IP_LOCAL_IN`
- `NF_IP_FORWARD`
- `NF_IP_LOCAL_OUT`
- `NF_IP_POST_ROUTING`



Iptables is mainly used as a firewall which organizes its rule in tables and chains. The tables classify the rules according to the type of decision that regards the packet.
The available tables are the filter table, the NAT table, the Mangle table, the Raw table, and the Security table.
The chains correspond to the available hooks, therefore iptbales has the **prerouting**, **input**, **forward**, **output**, and **postrouting** chains.
The following table explains the triggering logic and execution precedence.

Chains **Tables**	**PREROUTING**	**INPUT**	**FORWARD**	**OUTPUT**	**POSTROUTING**
ruoting decision				X	
raw	X			X	
connection tracking enabled	X			X	
mangle	X	X	X	X	X
DNAT (NAT)	X			X	
routing decision	X			X	
filter		X	X	X	
security		X	X	X	
SNAT (NAT)		X			X

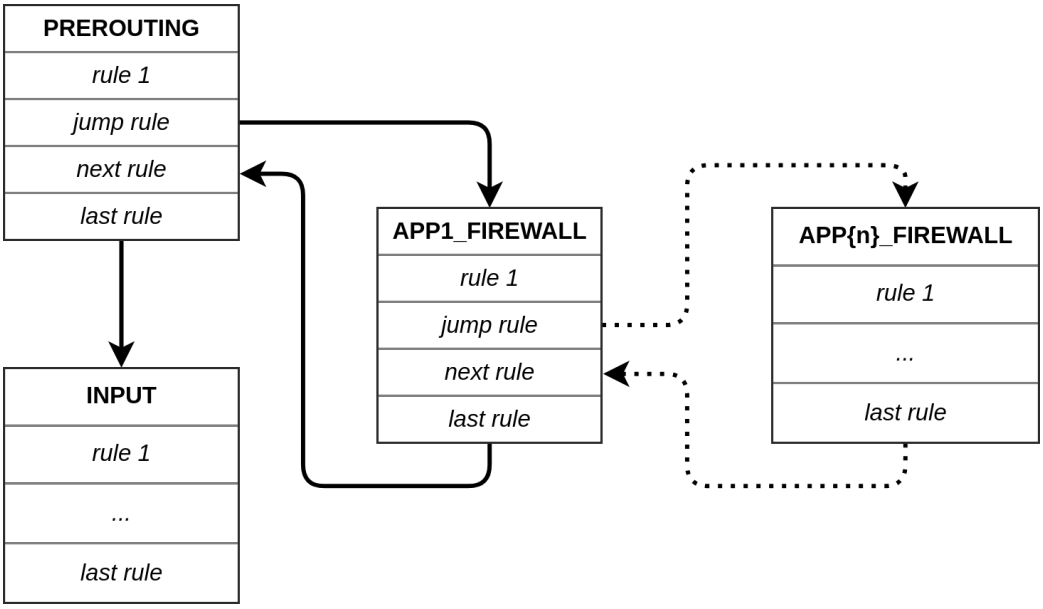
The packet triggers the hooks from the first row going down to the bottom of the table. It enters the chain depending on the routing decisions that are made based on its direction, incoming or outgoing, and whether the filtering rule accepts the packet.

The chain traversal order is the following:
Incoming packets destined for the local system:
PREROUTING → INPUT

Incoming packets destined to another host:
PREROUTING → FORWARD → POSTROUTING

Locally generated packets:
OUTPUT → POSTROUTING

Every chain is a list of rules. Every packet is checked against each rule and, if the matching component checks, iptables executes an action.
The rule can match the packet depending on the protocol type, the source or destination address, or packet, among other criteria.
The action called target, which is triggered if the rule matches, can be of terminating or non-terminating type.
The terminating type target causes the packet to exit the chain after a verdict that can be to allow or to drop the packet. The non-terminating type target causes the packet to continue its evaluation in the chain. There can be any number of non-terminating targets before a terminating one.
A very used non-terminating target is the jump action: a rule can decide to jump a packet into another chain, and when that chain finishes the rules check, the execution comes back to the previous chain. This way is useful to organize the rules in chains based on their purposes.



With `iptables`, it is possible to build firewalls and control systems to manage the computer network efficiently.

2.3 Design vulnerabilities

Mysterium is a complex system that integrates components of different types. The distinct nature of these components and their conjunction opens

to new vulnerabilities scenarios where the focus is on the system overall and how the elements dialog each other.
In the following chapter, we provide considerations on the Mysterium Network system concerning its components and their integration. In order to do this, we had to assumed some components behaviours and their impact on the system because Mysterium does not implement them yet, at this time. Indeed, Mysterium project is under development, which plans to have a fully decentralized network in 2021.
We classify the several aspects that, on a project like Mysterium, must be considered to evaluate its design vulnerabilities based on:

- the position in the network: starting, middle or endpoint point;
- the step of the flow: the service proposals, the promise issuing, the transaction closing, and other steps;
- the operations on the passing data: sniffing, filtering or crafting the data;
- side channels attack and other particular scenarios.

2.3.1 The position in the network

In the simplest configuration, the network includes a starting point that is the service consumer node and an endpoint, which is the service provider node. The VPN tunnel connects the two nodes and encrypts their traffic. The service node, as previously described, is the consumer's exit point to the internet.
We consider secure the traffic passing through the VPN tunnel because it is encrypted, not secure otherwise, that is from the provider to the internet.
SIMPLE NETWORK CONFIGURATION WITH AN EXIT NODE
The exit nodes connect two areas with two different security levels: for this reason, it is sensitive by default and without specific precautions can be the weak link of the system.
The Tor network has a similar problem. Tor stands for The Onion Router, and it is a network of peers where the user's encrypted traffic is sent into, and it bounces multiple times before getting an exit node from where, finally, it reaches the external resource.
The traffic is encrypted for each bouncing, except the last one, where it must be in clear to connect to the internet resource. The end node's role is

to decouple the encrypted network from clear internet, and this makes it so valuable. If an exit node is malicious, it can potentially understand all the data passing through it and going to the internet resource.
Dan Egerstad is a Swedish computer security consultant. He intercepted thousands of diplomatic emails setting up malicious exit nodes around the world and sniffing the connection between them and the clear internet [4].
To discover malicious nodes in the Tor network, another Swedish researcher built up a website with a login and a system to count the login attempts. Then, he used multiple Tor paths with different exit nodes to access the page. The system counted more login attempts that the researcher have done: this means some exit nodes sniffed the connections, caught the username and password, and tried to login by themselves [5].
In both cases, the use of HTTP instead of HTTPS, or of other unencrypted protocols, has been necessary, but the research aim was to demonstrate the exit nodes' critical role because there still are many protocols and applications that do not use encryption for their traffic.
The conclusion is that Tor's end nodes are the weak link of the system when the users use it as an end-to-end encryption network, rather than an anonymization network, as well as Mysterium Network end nodes.
For this reason, many of the conceptual vulnerabilities held by the exit node of Tor can be applied to a Mysterium Network exit node. But we want to point out a difference between Tor and Mysterium networks: the type of target users of the two projects.
The Tor's users use it to accomplish specific online activities where the identity obfuscation is mandatory. For the rest of the actions, Tor is not efficient, can be blocked by ISP and, in some countries, is even banned.
The Mysterium Network users, instead, use it for a broader range of online activities, and we can assume the use with a persistent connection to the VPN network with almost no impact on the user's system. In this direction, the implementation of the mobile app is crucial: smartphones are becoming the first device from where people access the internet and the capability to have a device connected all the time with the VPN network provides a security level not yet implemented for consumer uses.
This scenario, however, introduces move vulnerabilities simply because the network is used for more activities, and many protocols and applications still use unencrypted communication.
Figure X describes the network configuration of a Mysterium end node. The provider manages all the traffic from the VPN tunnel, both incoming

and outgoing and because it has not the VPN encryption, he can potentially perform all the manipulation he wants. We will discuss further this scenario analyzing the technical solutions and providing a practical example to exploit in chapter X.
If we ask someone to look at a photo for us and tell us to describe its content, he must, of course, understand the meaning, can change a little detail, or even tells someone else the content of the picture.
An improved configuration implies one or more middle points between the consumer and the provider. Those points are called hops, and they act similarly to the provider node since they forward the traffic.
Mysterium did not implement those hops in the network yet. Therefore we make a hypothesis on its possible implementation.
As well as for the provider nodes, we can suppose there is a distributed list of the forwarder nodes. When the consumer chooses the provider, and they agree to issue a payment promise, the network picks a node from that list to place it as the hop. The middle node receives a new VPN connection from the consumer and creates a new VPN connection to the provider. He technically builds and manages two different VPN tunnels: it takes encrypted data from one side, decrypts them, and encrypts again to send on the other one.
The scenario is similar to the end node because the middle node holds unencrypted data in the forwarding action.
The following picture well describes the situation with even more than one hop. We would have a segmented connection, called multihop connection, between the ends, where all the hops can perform unwanted operations on the traffic.
IMAGE OF MULTIHOP CONNECTION.
Mandatory would be an additional encryption layer end to end, over the VPN encryption between each node, that the consumer and provider have to manage. This second encryption solves the multihop data exposing because the middle nodes would manage only data encrypted by the second layer. The cons are that it increases the computational load.
The vulnerabilities, in this scenario, are restricted to all the operations that concern the traffic shape, with no abilities to understand it. These actions hit the quality of service, a class of attacks that aim to lower or even block the service.
Many traffic shapes operations can be performed on a connection: the simplest and the fastest is on the traffic speed, and it does not need to

understand the content at all.
A demonstration of how a middle node can slow down the data speed that it forwards, is using a Linux package called `tc` [3]. It stands for Traffic Control, and it is created and used for service quality analysis.
With the following command, we instruct the network manager to apply a delay of 500 milliseconds for every packet sent to the interface `eth0`.

```
1   sudo tc qdisc add dev eth0 root netem delay 500ms
```

Here is what each option means:

- `qdisc`: modify the scheduler (also known as queuing discipline)
- `add`: add a new rule
- `dev eth0`: rules will be applied on device`eth0`
- `root`: modify the outbound traffic scheduler (aka known as the egress qdisc)
- `netem`: use the network emulator to emulate a WAN property
- `delay`: the network property that is modified
- `500ms`: introduce delay of 500 ms

With the following command, we delete the delay rule.

```
1   sudo tc qdisc del dev eth0 root netem
```

In this simple way, the traffic speed between the consumer and the provider has a poor quality because one or more middle points are performing this slowing down the operation. From the consumer side, the poor connection is a quality agreement breach, and it affects the service provider's reputation. Since the provider can detect the slow speed, he can suppose it is a packet path issue and can propose to the network to change one or more middle nodes.

2.3.2 The flow steps

The first and easiest idea to implement to use the system maliciously has already been presented before: the attacker is the consumer, he creates an account in the Mysterium Network, has a positive balance to the

cryptocurrency wallet, and look for a service proposal. They agree on the service cost and issue a payment promise. If the user does not respect the promise... TODO

2.3.3 Operations on the passing data

In this section we analyze the different operation an end node can perform, based on the fact it manages traffic outside the VPN tunnel.
The traffic outside the tunnel is not encrypted by the VPN, but this does not necessarily mean it is not encrypted by itself. Some protocols encrypt data by design. Others do not encrypt or have different structure levels where the lowest have no encryption at all and can be modified.
Based on this consideration, the operations that a node can potentially do on the passing data can be grouped in sniffing, filtering, and crafting.
The first step to manipulate the data is to catch it, and we want to show an example of how-to-do. After starting a Mysterium Node on our test server and then establish the connection to it using a Mysterium client, we use `tcpdump` to dump the traffic of the network. For the example purpose, we get an online resource hosted on a different server and available using the HTTP protocol, so that the dumped traffic is in clear.
Our example provides an HTTP request to the host www.jafed.xyz with the method GET to obtain a file named test.txt with the content the string `cool, you got me sir!`.
The request is executed using curl, a command-line tool for transferring data using various protocols.

```
1  curl www.jafed.xyz/test.txt
```

With the following command, instead, we capture the request specifying the output format defined by `-v -n -l`, then we pipe it to the `egrep` command to parse the string and return an easy-to-read text:

```
1  sudo tcpdump -v -n -l | egrep -i "POST /|GET /|
     Host:"
```

A slice of output is

```
1  GET /test.txt HTTP/1.1
2        Host: www.jafed.xyz
```

that shows we caught the HTTP request.

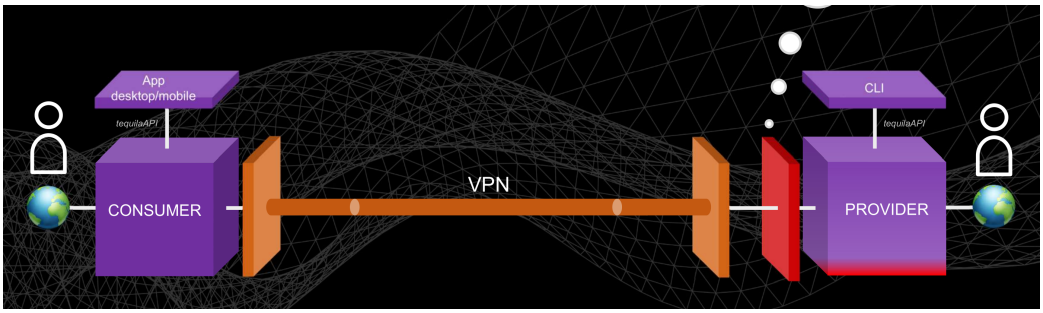
We use Wireshark to analyze the content, a powerful graphical application for network usages. It supports the pcapng file extension that tcpdump uses to save the data.
We print the captured result filtered of the packets of our interest:

```
1  GET /test.txt HTTP/1.1
2  Host: www.jafed.xyz
3  User-Agent: curl/7.58.0
4  Accept: */*
5
6  HTTP/1.1 200 OK
7  Date: Sun, 09 Feb 2020 16:59:45 GMT
8  Server: Apache
9  Upgrade: h2,h2c
10 Connection: Upgrade
11 Last-Modified: Thu, 06 Feb 2020 21:49:33 GMT
12 ETag: "1ae1c77-15-59def3f9b28fd"
13 Accept-Ranges: bytes
14 Content-Length: 21
15 Content-Type: text/plain
16
17 cool, you got me sir!
```

2.3.4 Side channels attack and particular scenarios

In this section, we discuss the possible scenarios that do not directly regard attacks on specific components, but more on how these harmless operations in Mysterium Network lead to vulnerabilities.
In Mysterium Network, the billing method is per the amount of transferred data or per time. Considering the first method, if the exit node is able to higher the volume of the traffic to the consumer, it means the exit node earns money without providing an equal service. A well-designed attack can perform the traffic shaping hiding the increase of traffic volume with network inefficiency.
In the "tcp/udp" subsection in the Craftberry tool description, we implemented an attack that uses this technic to make multiple copies of the packets using tcp and udp protocols with no content understanding. A more refined attack would be modifying the packet specifically for the

protocol, forcing the error in the communication so that the protocol corrects it. In this document, we did not focus on those particular cases which are deeply protocol dependent, but the modification aims to increase the transmitted data because the bill is calculated on that.
VLAN tag attack when the client is into an enterprise network. The attack crosses the firewall of the infrastructure.



3 Craftberry

3.1 The idea

Craftberry is a proof of concept tool to demonstrate how a node can sniff, filter and craft the outgoing traffic from an exit node of the Mysterium Network.
It is an application that runs via the command-line interface. It catches the incoming or outgoing packets in line with the packet flow, and it executes different actions accurately implemented to cover many attack and defense cases.
The software is entirely written in C and C++. It uses the `nfqueue` extension to move the packet from the kernel space to the user space, where it is crafted and sent back to the kernel space to continue the network traversal. The application uses the PcapPlusPlus library to parse the packet, build its layers, and modify the content.
The application sets up the iptables rules to redirect the packets. These rules follow the Craftberry action passed via CLI.

We propose part of the output provided by the application when the help command is used:

- `-a`: Use the specified action

- `-i`: Use the specified interface,tun0 by default. Can be interface name (e.g. eth0) or interface IPv4 address
- `-t`: Use the specified timeout in seconds, if not defined it runs until some external signals stop the execution (e.g. ctrl+c)
- `-l`: Write all the crafted and generated traffic into a `pcapng file` having name passed by parameter or, if the parameter is equal to 'default', the name is `out_<epoch_ms>.pcapng`
- `-d`: Direction filtering by and perform the crafting IN, OUT, default = IN
- `-v`: Shows verbose debug application logs
- `-h`: Displays this help message and exits

The parameter set by `-a` defines the action Craftberry has to perform when it catches a packet. We divide the operations into two categories: attack or defense. The attack operations are the most interesting for this research, but we also implemented a defense solution that protects the passing traffic encrypting the packet content.
The attacks differ on which ISO-OSI layer the packet is crafted.

IMAGE OF THE ISO OSI LAYERS MODEL

The attack actions categorized by layer are:

- Layer 2: VLanDoubleTagging
- Layer 3: IP, icmp
- Layer 4: TCP/UDP
- Layer 5 and above: HTTP, DNS, NTP

side attacks:
- auditin/collezione dei dati sniffati, inviati ad un server esterno per collezionamento, statistiche, vendita a terzi. - trojan all'interno di craftberry, per botnet. (approfondire aspetto che se il sw di attacco diventa popolare, può essere lui stesso veicolo di malware, quindi attacco l' attaccante)

The defense action is at layer 4. It encrypts the payload of the packets with `ChaCha20` algorithm, a `Salsa20` modification published in 2008. It is a particularly efficient stream cipher.

3.2 `nfqueue` extension

NFQUEUE is a kernel and user mode module for managing network packets in iptables. It allows writing `Netfilter` target modules in userspace. This module is an application that is called by `nfqueue` hooks. In the program, the hooks are bounded to a function that is called when the iptables match a rule.
The function gets the queue details and the packet payload, it can craft the packet, and then send a verdict to the iptables module.

FLOW OF THE PACKET (CATCHING, CRAFTING AND THEN VERDICTING IT)

3.3 PcapPlusPlus

It is a multiplatform C++ library for capturing, parsing and crafting of network packets. As major features, there are that it is designed to be efficient, powerful, and easy to use. It is a wrapper for the most popular packet processing engines making easy decoding and forging a large variety of network protocols.

3.4 Software Architecture

Craftberry has an architecture composed of four components: the first one is the `main` function that sets up the `iptables` rules, register the callback function to get the captured packet, and manages the `cli` parameters. The second one is the callback function, the core of the application, that takes the captured packets and calls the corresponding action to execute on it. The third component includes all the methods that implement the attack or defense mechanisms. All of them belong to the `Action` namespace. The fourth component is a class, called `Configuration`, which includes the functions to collect statistics on the elaborated data, and to manage the `tun0` network interface. It also implements auxiliary functions for the application.

3.4.1 The `main` function

We outline the principal sections of the `main` function:
The first block of code concerns the registration of the function for external signals, like `ctrl+c`. Craftberry can run with a timeout, if the value is passed via CLI, or in infinite mode. In both cases, the execution can be stopped using the external `ctrl+c` signal.

```
1  struct sigaction sigIntHandler;
2  sigIntHandler.sa_handler = ctrl_c; \\the callback
     function when the signal is emitted
3  sigemptyset(&sigIntHandler.sa_mask);
4  sigIntHandler.sa_flags = 0;
5  sigaction(SIGINT, &sigIntHandler, NULL);
```

The application manages the command-line input parameters using the `getopt_long` internal function. The parameters are specified in the `help` function output. The input parameters are validated and used to initialize the `Configuration` class.

The next step is to register the callback function to the `nfqueue` when the `iptables` rule is matched. To do so, we need to add the rule to the `iptables` list. This operation is done by the `makeIptableCmd` function described later in the `Configuration` class.

The main function ends with the infinite loop that reads a buffer pointed by the handler calculated using the `nfqueue` initializing function. When the packet matches the `iptables` rule, it is moved into the user space memory. Craftberry continuously reads the memory and, when it is not empty, starts the procedure to parse and elaborate the packet.
If the execution has a timeout, we use a separate thread to terminate Craftberry when the timeout is expired. At that time, it is called a procedure to destroy the `nfqueue` queue, print an execution summary, free the resources, and delete the `iptables` rules.

3.4.2 The callback function

The callback function is the core of the application. It has the following signature:

```
1  static int callback(struct nfq_q_handle *qh,
       struct nfgenmsg *nfmsg, struct nfq_data *nfa,
       void *data);
```

- `qh`: pointer to the `nfqueue` internal handle for the queue;
- `nfmsg`: pointer to the Netfilter address family dependent message;
- `nfa`: pointer to the Netfilter captured packet data;
- `data`: pointer to a user-defined memory to pass context values;

The most important parameter, for our purposes, is `nfa` from which we extract the message packet header and payload.

```
1  struct nfqnl_msg_packet_hdr *ph =
       nfq_get_msg_packet_hdr(nfa);
2  unsigned char *rawData = nullptr;
3  int len = nfq_get_payload(nfa, &rawData);
4  struct pkt_buff *pkBuff = pktb_alloc(AF_INET,
       rawData, len, 0x1000);
5  CHECK(ph == nullptr, "Issue while packet header");
6  CHECK(len < 0, "Can't get payload data");
7  struct timeval timestamp;
8  nfq_get_timestamp(nfa, &timestamp);
```

Note: we use the MACRO `CHECK` to ensure the return values are valid.

The `pkBuff` variable contains the packet raw data from where we build the PcapPlusPlus `RawPacket`. From the `RawPacket`, the library is capable of parsing the packet and building all the layers.

```
1  pcpp::RawPacket *inPacketRaw = new pcpp::RawPacket
       (pktb_data(pkBuff), pktb_len(pkBuff), timestamp
       , false, pcpp::LINKTYPE_RAW);
2  pcpp::Packet *inPacket = new pcpp::Packet(
       inPacketRaw);
```

```
3   //Print all packet layers for inspection
4   pcpp::Layer *L = p->getFirstLayer();
5   while (L != nullptr) {
6    cout << "\t\t| LEV: " << L->getOsiModelLayer() <<
         " => " << L->toString() << endl;
7    L = L->getNextLayer();
8   }
```

The parsed packet is the input for the single operation that the Craftberry user wants to perform.
Each operation is selected matching the action, specified in the passed parameter, and the protocol, recognized in the packet.

The pattern is:

```
1   if (conf->method.compare("HTTPBLOCK") == 0 &&
       Action::HTTP::isProtocol(inPacket)) {
2    //do something with the packet
3    return verdict_accept(qh, ntohl(ph->packet_id),
        inPacket);
4   }
```

where, for instance, the passed action is `HTTPBLOCK` and `inPacket` must contain a HTTP layer.
Every method must end with a call to a `verdict` function that decrees if the packet is accepted or dropped.
For the majority of our action implementations, the verdict is to accept the packet that has been modified. Before calling the function to accept the verdict, we collect statistic data and save the packet into a log file. The log file contains all the packets that Craftberry has captured, in case crafted, and accepted to continue the network stack traversal.

3.4.3 The action classes

The action classes contain the implementation of the attack or defense mechanisms. As said, they belong to the `Action` namespace and have a

similar common structure. All of them share the `isProtocol` method used to identify if the action can be performed on the packet.
The single method that executes the attack or defense mechanism takes in input a `Packet` variable, thus a parsed packet. The advantage of using the PcapPlusPlus library is that it exposes functions to craft the packet easily.

We describe all the action methods and the relative attack or defense idea in the Action chapter.

3.4.4 The `Configuration` class

It is a `singleton` class that contains all the application setup, collect data, and exposes functions used by all the other application components, thus, it is a shared instance

An important function is to open and manage the `tun0` network interface using the PcapPlusPlus device management methods. Then it implements the `sendPkt` function to send a packet to the destination, the `tun0` interface: for some actions this function is needed to complete the attack.

```
1  bool sendPkt(RawPacket *p) {
2   if (p->getRawDataLen() > conf->devTun0->getMtu())
        {
3    conf->dropped.packets++;
4    conf->dropped.bytes += p->getRawDataLen();
5    return false;
6   }
7   if (conf->devTun0->sendPacket(*p)) {
8    conf->created.packets++;
9    conf->created.bytes += p->getRawDataLen();
10   conf->devLogFile->writePacket(*p);
11   return true;
12  }
13  cout << "something strange just happened" << endl
       ;
14  return false;
15 };
```

The function also collects data for the statistics.

The `Configuration` class also contains the `makeIptableCmd` function to add or remove the rule to the `iptables` list. It composes the bash command string and then calls the `system()`function to execute the command. The function builds the string depending on the adding deleting action passed by the function caller.

```
1  void makeIptableCmd(bool isDeleting) {
2   string protocol = "";
3   if (conf->method.compare("BEQUITE") == 0) {
4    protocol = "all";
5   } else if (conf->method.compare("TCPMULTIPLY") ==
        0 || conf->method.compare("IPV4") == 0) {
6    protocol = "tcp";
7   } else ...
8   string dir = std::to_string(static_cast<std::
       underlying_type<Direction>::type>(conf->
       direction));
9   string cmd = ("sudo iptables -t filter "s +
10      (!isDeleting ? "-I "s : "-D "s) +
11      (conf->direction == 0 ? "INPUT "s : "OUTPUT "s
           ) +
12      "-p "s + protocol + " -j NFQUEUE "s +
13      " --queue-num "s + dir);
14  cout << "IPTABLE RULE: " << endl
15   << "\t$: " << cmd << endl;
16  system(("\n#/bin/bash\n\n"s + cmd).c_str());
17 };
```

In a scenario where Craftberry is executed two times, once capturing the incoming traffic and then capturing the outgoing traffic, we use the `--queue-num nfqueue` parameter to bind the queue with the Craftberry execution. Therefore, the queue number is defined by the packet capturing direction.

3.5 Actions

3.5.1 VLan double tagging

use the double tagging attack method on cisco routers

3.5.2 IP

The IP packet contains information on the source and the destination IP address. The goal of this attack is to change the destination address to redirect the packet somewhere else.
When the packet is captured from the callback, the `changeDst` function is called passing the parsed packet. Then, we extract the IPv4 layer from the stack and change the destination Ip using the PcapPlusPlus methods.
The code is the following:

```
1  void changeDst(Packet *inPacket) {
2   IPv4Layer *ipv4 = inPacket->getLayerOfType<pcpp::
       IPv4Layer>();
3   ipv4->setDstIpAddress(IPv4Address(newIp));
4  }
```

Where `newIp` is a class field that contains the new destination IP the attack aims to set. It is set during the action creation.

The scenarios that can use this action are where the attacker wants to redirect all the traffic to another server, often called Command and Control, to perform other activities on the data.

3.5.3 ICMP

ICMP, an abbreviation for Internet Control Message Protocol, is a protocol used for network messages and diagnostic operations. The action on this level has been implemented for test purposes and does not have any particular attack model. By the way, after capturing the ICMP packet, we are able to change its values for the request and the reply messages.

3.5.4 TCP/UDP

The context of the attack supposes that communication has supplementary encryption, over the VPN encrypted layer, provided by end-to-end encryption between the Mysterium client and the resource server the user is communicating with, and not the exit node. Therefore, the latter cannot perform any content understanding, and can do only actions on the entire packet, as described in the Design vulnerabilities chapter.
The attack accomplishes a simple copy of the packet: when the provider receives packets from the internet, it sends `n` times a copy of the captured packet to the consumer.
The implementation is as simple as significant: with this method, with no content understanding, the passing data are more than the necessary, and the service consumer is affected.

We have implemented a demonstrative example of this attack using the protocols TCP and UDP.

The code in the main function is the following:

```
1  if (conf->method.compare("ICMPMULTIPLY") == 0 &&
       Action::Icmp::isProto(inPacket)) {
2   Action::Tcp *action = new Action::Tcp(n);
3   action->multiply(inPacket->getRawPacket());
4   return verdict_accept(qh, ntohl(ph->packet_id),
       inPacket);
5  }
```

The multiply method is implemented as follows:

```
1  void multiply(RawPacket *inPacket) {
2   for (int i = 0; i < this->n; i++) {
3    sendPkt(inPacket);
4   }
5  }
```

The `n` parameter is a TCP class field.
For each received TCP packet, the method send `n` copies of it to the network.

3.5.5 DNS robber

The DNS is a network protocol necessary to resolve a domain name to the correspondent address. Unfortunately, it is unencrypted and susceptible to attacks of a different nature [**?**].
The attack idea is to capture a DNS request and response and perform DNS hijacking [**?**]. The hijacking can be done in different ways: capturing the DNS request or capturing the DNS response.
The request contains the name server the machine wants to resolve to get the corresponding IP address necessary to begin the communication. If we catch the request packet, we can change the name server with another malicious nameserver under our control. The DNS resolver reply to the consumer with our machine IP that returns our web page, if the query is HTTP. This attack can be used for phishing, the attempts to obtain sensitive information by disguising oneself as a trustworthy entity, or for collecting data.
The response packet modification has a similar goal

3.5.6 HTTP

Hypertext Transfer Protocol is an application-layer protocol for transmitting hypermedia documents, and it is the standard in the client-server architectures. The HTTP protocol is stateless, meaning that the server does not keep any state between two requests.
This protocol is perfect for implementing an attack and for demonstrating how a node can maliciously and heavily modify the content of a packet. Our goal is to intercept the response of the HTTP answer, detect if the payload is an image, and then substitute it with another one. We need to clarify that the HTTP response content can be split over many TCP packets, if it is larger than the maximum TCP packet size. Craftberry captures TCP packets but does not perform TCP reassembly. Therefore, for our demonstration purpose, it can operate the image substitution only for images contained in a single TCP frame.
A little variant of this attack append the original name server and final resource, also called URL, as `GET` parameter to another name

- http://linux-training.be/networking/ch14.html

`http://unixwiz.net/techtips/gnu-c-attributes.html`

https://gcc.gnu.org/onlinedocs/gcc-4.0.2/gcc/Type-Attributes.html
https://groups.google.com/forum/#!msg/pcapplusplus-support/e7rN93LfTSg/MFnVEKCNCAA
https://byt3bl33d3r.github.io/using-nfqueue-with-python-the-right-way.html

3.5.7 Cypher

3.6 Tests

???

3.7 Design solutions for malicious context

3.7.1 comparing data

invia la richiesta a due nodi e le confronta (essenziale possibilità di connessione a due nodi in contemporanea: due o più container docker mappati su porte diverse con uno script che invia la richiesta a più container e confronta le risposte)

3.7.2 comparing hashes

come precedente ma con la gestione degli hash invece che di tutto il dato

4 Conclusions

4.1 Future development

4.2 Considerations

The use of Mysterium Network like an end-to-end encryption tool: https://nakedsecurity.sophos.com/2015/06/25/can-you-trust-tors-exit-nodes/ Here there is a helpful consideration of how critical is an exit point for TOR.

References

[1] HTTP Over TLS - RFC https://tools.ietf.org/html/rfc2818

[2] A Framework for IP Based Virtual Private Networks - RFC https://tools.ietf.org/html/rfc2764

[3] Traffic Control https://jvns.ca/blog/2017/04/01/slow-down-your-internet-with-tc/

[4] exit nodes of TOR network https://www.wired.com/2007/09/rogue-nodes-turn-tor-anonymizer-into-eavesdroppers-paradise/?currentPage=all

[5] malicious TOR exit nodes https://www.makeuseof.com/tag/priority-wretched-hive-scum-villainy-5-ways-stay-safe-bad-tor-exit-nodes/

[6] DNS attack patterns https://www.fireeye.com/blog/threat-research/2019/01/global-dns-hijacking-campaign-dns-record-manipulation-at-scale.html

[7] DNS hijacking https://securitytrails.com/blog/dns-hijacking

[8] libnetfilter queue https://home.regit.org/netfilter-en/using-nfqueue-and-libnetfilter$_{q}ueue$/

[9] todo https://buffered.com/glossary/tun-tap/

[10] todo https://netfilter.org/projects/libnetfilter$_{q}ueue$/$doxygen$/$html$/$group_{nfq_{verd.html}}$

[11] todo http://www.cs.unibo.it/ ghini/didattica/sistemimobili/IPtables$_{N}etfilter_{A}ddDelay$/$packet_{e}ngine.c$

[12] todo https://byt3bl33d3r.github.io/using-nfqueue-with-python-the-right-way.html

[13] todo https://groups.google.com/forum/!msg/pcapplusplus-support/e7rN93LfTSg/MFnVEKCNCAAJ

[14] todo https://subscription.packtpub.com/book/networking$_{a}nd_{s}ervers$/9781783553136/1/$ch01lvl1sec12$/$internals$

[15] todo https://mysterium.network/whitepaper.pdf

[16] todo https://pcapplusplus.github.io/docs/features

[17] todo https://www.apriorit.com/dev-blog/598-linux-mitm-nfqueue

[18] todo https://stackoverflow.com/questions/27293924/change-tcp-payload-with-nfqueue-scapy

[19] todo https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/shining-a-light-on-the-risks-of-holavpn-and-luminati