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

Jacopo Federici

February 21, 2020

Contents

0.1 Abstract

0.2 Work structure

0.3 Outline

0.4 Goal

1 State of the art

1.1 Introduction

The need for privacy is becoming more critical day after day: the massive digitalization of many aspects of our life implicitly exposes us and our data and even the simplest operations which use an internet connection are becoming unsafe.

In the last years, we have seen security solutions integrated into popular applications, such as browsers or mobile apps, following the increase of security attacks. For instance, in 2017, the popular Chrome browser announced it would mark as non-secure any website using the HTTP protocol instead of HTTPS. This move was to encourage the website to use SSL/TLS certificates and make secure the communication between client and server. It's interesting to discover the Netscape Navigator browser has made the first HTTPS version around 1990, but the official RFC publication has been released only in May 2000 [1].

Similar scenario for the VPN: abbreviation for "virtual private network" it's a technical solution to extend a private network across a public network. It creates a virtual point-to-point connection through the use of a tunneling protocol over the existing network, and the tunneled traffic is encrypted. A first VPN specification proposal 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 that only lately are beeing relevant. Those problems can be exiting 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. The company can keep the connection logs, inspect the traffic, or sell statistical data.

Some open-source projects were, lately, born to solve these problems. Their goal is to revolution the idea of VPN and its uses, building a rich network of peers where two nodes can connect through a VPN. The project provides 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 document, 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 design vulnerabilities on, 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.
The second step focuses on the solutions that can patch those critical aspects and provides a comparison of them with an eye on the practical perspective.

1.2   The world now

We are still in the middle of a large transformation. From the invention of the pc and the birth of the internet, more and more services are becoming digital; more and more people are using those services; more and more bad guys are trying to pull down those services.
But there are differences from some decades ago.
First, everything is now much faster. The internet infrastructure allows us to move significant amounts of data within a few seconds. Furthermore, our personal computer can perform operations that were impossible to execute some years ago on mainstream servers. This speed 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 base 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 destinations.
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 regard the paradigma, 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"

1.3   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-Circuits.html
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

Anonymization: in every network systems, anonymization is a tough challenge.

Even more, if privacy is a crucial point. In some countries, the VPN is massively used to protect data from surveillance at any level, from a public unsecured wifi connection to the country ISPs.

There are countries, the People's Republic of China for example, where censorship is present and where identity obfuscation is, therefore, necessary to protect yourself. In these cases, the use of a VPN is a solution, as long as a safe company provides it. With the adjective 'safe' we mean a company that does not make you or your traffic detectable or even worsts sniffs and crafts your data for malicious actions.

The decentralized VPN completely changes the paradigma, 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.

1.4 We are our trust

2 My work, steps

2.1 Decentralized VPNs

```
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\\
```

A decentralized VPN is a network of nodes where the users can connect through a VPN to nodes without a central entity. 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)

2.1.1 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, agrees 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)

2.2 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 document.

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 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.

2.3 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 the distributed database where all the network operations, the identities, and the service proposals are recorder.
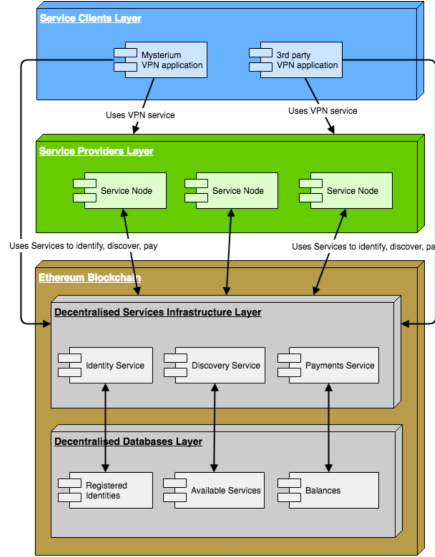
As a general workflow description, Mysterium starts by collecting all the agents' service proposals and presents them to the clients. Then it registers the client's intention to pay for the agent's service chosen by the user and creates the VPN connection between the two points. The outcome of each transaction 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 VPN services availability and to pick the most suitable VPN service.

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

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

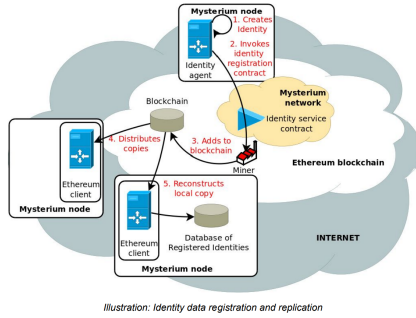Illustration: Identity data registration and replication

Figure 1: The Mysterium identity creation steps

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. To complete this operation, the nodes must pay 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.

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.

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.

8

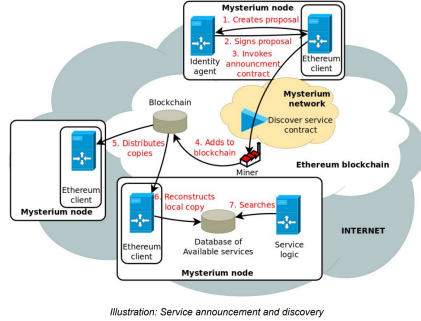*Illustration: Service announcement and discovery*

Figure 2: The Mysterium service announcement steps

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.

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 does not implement its version of VPN but 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 via RESTful API exposed in the localhost, called TequilaAPI. TequilaAPI is used by the client's application to manage the connection and for diagnostic operations. Mysterium developed a desktop client written using Electron, a technology that is a web application locally executed by NodeJs, a server backend.

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.

2.3.1 Project setup

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.
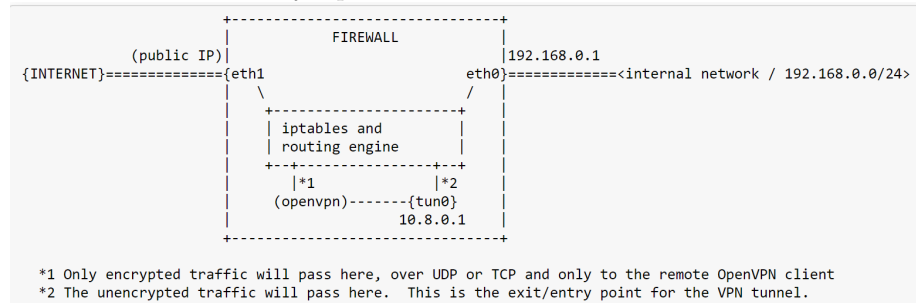
They can be of two types, TAP or TUN.

The TAP type is used to simulate a link layer device, which means the corresponding ISO model level it refers to is the number 2. 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 3. 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 'tun0' is the virtual TUN interface created by OpenVPN.

```
                 +-------------------------------+
                 |                               |
                 |            FIREWALL            |
          (public IP)|                           |192.168.0.1
    {INTERNET}=============={eth1                     eth0}=============<internal network / 192.168.0.0/24>
                 |   \                       /   |
                 |    +---------------------+    |
                 |    | iptables and        |    |
                 |    | routing engine      |    |
                 |    +--+---------------+--+    |
                 |       |*1             |*2     |
                 |    (openvpn)-------{tun0}     |
                 |                    10.8.0.1   |
                 +-------------------------------+

    *1 Only encrypted traffic will pass here, over UDP or TCP and only to the remote OpenVPN client
    *2 The unencrypted traffic will pass here.  This is the exit/entry point for the VPN tunnel.
```

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.

On the other direction, when a packet arrives at eth0 interface, "iptables" and the routing engine apply their rules to filter and masquerade and then forward it to tun0 interface. Then, 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 traffic coming from "tun0" to the public interface eth0 on a specific port. The client application listens on that port the traffic that its OpenVPN client decrypts.
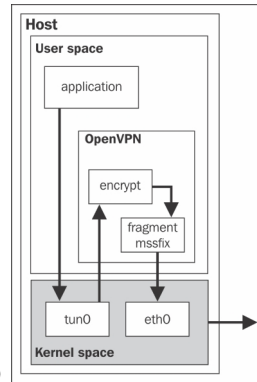
Host
User space
application
OpenVPN
encrypt
fragment
mssfix
tun0   eth0
Kernel space

IMAGE TO EDIT AS BEFORE DESCRIBED
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 substantially 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, then Iptbales has the pre-routing, input, forward, output, and post-routing chains. The following table explains the triggering logic and execution precedence.
IPTABLES TABLE
The packet triggers the hooks from the first row going down to the bottom of the row. It enters the chain depending on the routing decisions that are made, 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

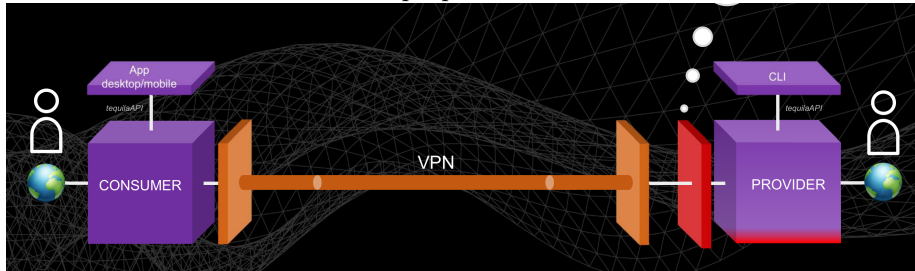| Chains / Tables	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
ruoting decision				•	
raw	•			•	
connection tracking enabled	•			•	
mangle	•	•	•	•	•
DNAT (NAT)	•			•	
routing decision	•			•	
filter		•	•	•	
security		•	•	•	
SNAT (NAT)		•			•

- 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, an action is executed.

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 rule check, the execution is back to the previous chain. This is a useful way to organize the rules in chains based on their purposes.

2.4   Basic sniffing to catch content (Wireshark, tcpdump)

Note:

- https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture

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

2.5   Craftberry, a demo tool

useful tips

```
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/MFnVEKCNCAAJ
https://byt3bl33d3r.github.io/using-nfqueue-with-python-the-right-way.html
```

2.5.1   liv 2: arp, vlan tagging

2.5.2   liv 3: ip, icmp

2.5.3   liv 4: tcp/udp

2.5.4   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)

- copia x volte dei pacchetti (TCP/UDP)

- gonfiamento dei pacchetti nei campi che poi verranno corretti dai check (TCP/UDP)

- modifica del payload se in chiaro (TCP/UDP)

- azione positiva sul payload (UDP), ad esempio cifratura chacha20

2.5.5   liv 5+: http/imap/dns/ntp/etc

analisi e modifica dei contenuti a livello del singolo protocollo (ad esempio sostituzione di tutte le immagini con una immagine di default)

2.6   Design solutions for malicious context

2.6.1   comparing

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)

2.6.2   comparing hashes

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

3 Design integration with reputation systems

3.1 something here

4 Conclusions

4.1 Tests

4.2 Future development

4.3 Considerations

problemi individuati

1. valutazione della reputazione dell'agent da parte del client

2. valutazione della reputazione dell'agent da parte della Net

soluzioni proposta Proposte per la valutazione della reputazione dell'agent da parte del client

Le seguenti proposte sono metodi di verifica del corretto comportamento ed hanno come scopo finale la valutazione della reputazione degli agent

- Dati civetta: modifiche al client che con scadenze richiede una o più risorse dal valore noto e verifica che siano integre. Le scadenze possono essere regolari/random/all'inizio frequenti/dipendenti dalla reputazione dell'agent. È necessario avere delle risorse distribuite e disponibili: potrebbero essere i nodi stessi della rete (altri agent).

- Dati duplicati: modifiche al client che implementa la possibilità di connessione con più agent. Dopo la richiesta il client compara i dati ottenuti dalle due fonti

  - Hash *: (soluzione aggiuntiva) I nodi sono generalmente in posizioni migliori dei client in termini di velocità. L'idea è quella di far generare un hash dei dati che il client richiede ad un altro nodo fuori dal servizio primario, così da comparare gli hash e non tutto il dato. Inoltre, se la richiesta la faccio a molti più nodi posso intrinsicamente verificare quali modificano i dati nel network.

- Applicazione di modelli di reputazione (Eigentrust): attualmente non studiato

References

[1] https://tools.ietf.org/html/rfc2818

[2] https://tools.ietf.org/html/rfc2764

[3] manually managing references, `https://en.wikibooks.org/wiki/LaTeX/Manually_Managing_References` 2016