Note: We recommend that you read the online version at https://docs.safing.io whenever possible.

Tech Overview

Here is short and very technical overview of the three components that make Safing awesome.

Portmaster

The Portmaster is an application firewall that enforces application profiles on processes.

Being an application firewall means tightly integrating with the kernel of the underlying OS (via network filter APIs or kernel modules) to gain needed information (associate network packets to processes) and control (block or deny network connections).

Application profiles describe an application's behavior in the sense of how it interacts with the Internet: Does it connect to a fixed set of domains? Is it a peer to peer application? Does it interact with the local network? Should TLS be enforced and checked? etc.

Profiles represent an application as experienced by the user, not as defined by technology - making them easy to understand and superior to other common application classification approaches. They can be either created by users themselves or - most of the times - obtained through Stamp (explained later).

Port17

Port17 brings together several state of the art network technologies and gives them a new spicy twist .
The goal of Port17 is to protect connection data as well as metadata from surveillance capitalism.

Adapted core concepts:

- To protect your privacy, we use newest proven encryption technology: a **double ratchet** [1] based protocol to provide perfect forward and backward secrecy which can change used algorithms on demand through configuration.
- The **onion-encrypted multi-hop architecture** [2] protects your identity and makes you anonymous online.
- **Zero roundtrip** connection establishment enable blazing fast connections.
- **Paid community nodes** are highly welcome to build a huge, capable and trustless network.

The new spicy twist :

- To protect network data and metadata as long as possible, Port17 selects exit nodes [3] in proximity to the destination server.
- Routes are calculated for maximum speed by default and use a minimum of 3 nodes. This behavior changes based on the active security level, providing slower, but tougher routes with more nodes.
- Exit node [3] selection can be influenced up to application/domain pairs.
- Unencrypted connections are only handled by trusted nodes run by Safing. In this sense we act as a trusted anchor until all the web is encrypted. There is no room for MITM-ing [4] nodes.
- As soon as the network has a good share of community nodes, routing will diversify routes by node ownership to further reduce needed trust on single parties (us).
- Tunnels are layer 5 [5] and up to reduce unnecessary metadata and improve speed (similar to SOCKS proxies).
- Clients always know about the full network. (will require improved method for better scaling at some point)
- Authentication is decoupled from network nodes - they only know that someone is allowed to use the network, but not who.
- Payment via cryptocurrencies allows unblockable payment access.
- Tight integration with Portmaster ensures that no data will ever leak should a tunnel break. (unlike VPNs)

A final note: You may have noted that Port17 is, in some aspects, similar to the Tor Project. The key difference is, that Port17 focuses on speed and usability, but does not claim to match Tor's level of security. We will provide a in-depth comparison in the future.

Stamp

Stamp is an online community where participants *"stamp"* (ie. tag, categorize) domains and applications (used by Portmaster for application profiles) to serve as a data source for any kind of network filter. Contributions are rewarded with reputation that gives them more influence on the platform. This reputation system will be blockchain-based in the future. Stamp is a separate project that is backed by Safing.

---

1. Double Ratchet Algorithm ↩
2. Onion routing ↩
3. final network node from which the connection is made to the destination server ↩ ↩[2]
4. Man-in-the-Middle Attack ↩
5. OSI network model ↩

User Guides

Introduction to Portmaster

> The Portmaster is currently still in Tech-Preview phase. When trying out the technology, please keep this in mind - it may be a bit rough around the edges. Also, only the base features are implemented - the UI (and this guide) hint at many other features that are not yet implemented.

Installation

In the Tech-Preview phase, we do not yet provide a full installer, as it is not yet meant to be run 24/7. When you download the installer package, there is a small script that will start the daemon, user interface and notification bar agent, if available.

Please a keep a close eye on the console output of the daemon, as it will show you what it is doing and if there are any problems.

Basics: Security Levels and Application Profiles

The two things you should know about Portmaster when testing the software are:

1. Security Levels

There are three security levels

Dynamic
Regular mode - provides additional security measures to protect your privacy, but will also try to not be in your way to help you stay focused. Use this mode in trusted networks.

Secure
Heightend security measures - to keep you safe in untrusted environments. It is automatically activated if you enter an unknown network, like a café's Wi-Fi, or if an attack is detected. Use this mode when you do not trust a network, or are temporarily in need of more security.

Fortress
All protective mechanisms available are activated. This will most likely cut off at least some applications from the Internet, but provides best protection technically possible. Use this mode if you think you are currently being attacked, like having clicked on a possible virus.

These Levels also influence which other features are activated. Check out the settings tab in the UI to see (planned) features are set in which security levels they should be active!

2. Application Profiles

Application Profiles are how you can control which application is allowed to connect to the Internet and how. Applications are matched by their installation path - be sure to have to path to the binary right to have a Profile applied (you can check the logs or the monitor tab in the UI).

Properties

All of the properties are explained where they are appear on settings page (press the small *i* icon), so only the most important parts are covered here:

- **Security Level:** Define the minimum Security Level (and it's configured features) that should be applied with this Profile.
- **Domain Whitelist**: Define a domain whitelist for this Profile, connections to all other domains will be denied.
- **ConnectPorts:**: Define a whitelist of remote TCP/UDP ports that applications are allowed to connect to.
- **ListenPorts:**: Define a whitelist of local TCP/UDP ports applications may listen on. Please note that the `Service` Flag needs to be set in order to allow listening at all.

Flags

Flags are an easy way to require or constraint to an application to a certain behavior.

- Executing User
  - **System:** System apps must be run by system user, else deny
  - **Admin:** Admin apps must be run by user with admin privileges, else deny
  - **User:** User apps must be run by user (identified by having an active safing UI), else deny
- Network Scope
  - **Internet:** Internet apps may connect to the Internet, if unset, all connections to the Internet are denied
  - **LocalNet:** LocalNet apps may connect to the local network (i.e. private IP address spaces), if unset, all connections to the local network are denied
- Network Destinations
  - **Strict:** Strict apps may only connect to domains that are related to themselves
  - **Gateway:** Gateway apps will connect to user-defined servers
  - **Service:** Service apps may accept incoming connections

Default Profiles

Because it is infeasible to have a separate Application Profile for every program you directly or indirectly use, you can also define a Profile for whole folders. These Profiles are called `Default` Profiles and are matched on a path prefix basis instead of an exact match basis.

User Interface

When starting the Tech-Preview version for the first time, the UI should open automatically. There you can change settings, view and edit application profiles and monitor current connections.

Inspection

There is already the option to have Portmaster check TLS validity of connections, but this module is not currently part of the Tech-Preview as it is not a core feature and needs more time for refinement. If you want to check it out, you can easily compile a version with it included by using an empty import `_ "githu...` on the Golang package in the main Golang file.

Introduction to Port17

> Port17 is currently still in Tech-Preview phase. When trying out the technology, please keep this in mind - it may be a bit rough around the edges. Also, only the base features are implemented - the UI (and this guide) hint at many other features that are not yet implemented.

Installation

In the Tech-Preview phase, we do not yet provide a full installer, as it is not yet meant to be run 24/7. When you download the installer package, there is a small script that will start the daemon, user interface and notification bar agent, if available.

Please a keep a close eye on the console output of the daemon, as it will show you what it is doing and if there are any problems.

Testing

As we do not yet provide a network for Port17 testing, you will have to create your own. This is easiest done with Docker. In the `safing-core` repo you can find a small guide and readme in the directory `port17/testing/simple` to get started.

You can then start the Portmaster with Port17 on the host and connect it to the network using the `-bootstrap` parameter to give the daemon the IP address of an initial Port17 it will connect to.

If you do not feel tech savvy enough, we would recommend to wait for the alpha release, which will also feature a ready-to-use test network with several Port17 nodes.

Docs: Core

Tinker

`</> tinker`

The Tinker is responsible for all the crypto (*cryptography*, that is) in Portmaster and Port17.

As soon as the concept and implementation are finalized, we will provide a detailed documentation here.

Docs: Portmaster

Firewall

`</> firewall` · `</> firewall/interception` · `</> portmaster` · `</> network` · `</> network/packet`

> If you not have read the Tech Overview, please start there.

The Application Firewall is responsible for interception network connections and analyzing them to only the ones that are in the interest of the user - while not bugging the user about it.

Packet Interception `</> firewall/interception` `</> network/packet`

The interception package (a seperate one for each OS) provides the firewall a stream of packet objects, which the firewall can inspect and then issue a verdict through these packet objects.

Verdicts may be:

- **Accept**: Packet is allowed to pass.
- **Block**: Packet is dropped, a TCP-Reset or ICMP `host unreachable` message is sent to the sender.
- **Drop**: Packet is dropped silently.
- **PermanentAccept**: This packet is allowed to pass, also tell the interception package to accept all future packets of this `Link`.
- **PermanentBlock**: This packet is blocked, as well as all future packets of this `Link`.
- **PermanentDrop**: This packet is dropped, as well as all future packets of this `Link`.
- **RerouteToNameserver**: Reroute this packet to the local nameserver for handling.
- **RerouteToTunnel**: Reroute this packet (and its `Link`) to the local Port17 entry point for further handling.

The *permanent* editions of verdicts were created to drastically improve performance of the portmaster, as such `Links` will be “handed over” back to the OS and will not be intercepted by the Portmaster anymore. The trade-off here is that connections cannot be killed, should the user or software change it’s mind about it later on - but this is usually not the case.

Links

`</> network#Link`

`Links` represent a physical connection between a local application and a remote server. It is defined and identified through the IP/Port pair.

Connections

`</> network#Connection`

`Connections` represent a logical connection between a local application and a Internet entity, identified by a domain. `Connections` will usually have multiple `Links` belonging to it.

The Portmaster

`</> portmaster`

The Portmaster is the component that is handed received `Connections` and `Links` as well as any intelligence data gathered to make a decision.

It always tries to make a decision on the `Connection`, which `Links` will automatically inherit. All these decisions and why they were made can easily be monitored in the UI.

Security Levels

Security Levels were created to give the user an easy and quick way to adapt to different environments and better handle the under/overblocking problem.

They are the main way to interact with the Portmaster and react to user-perceived threats.

Dynamic

This is the standard operating mode - the user is in a trusted environment and no threats have been detected. Privacy protections are activated, but may slightly tend to underblocking.

Secure

This mode is the standard mode for untrusted environments. It is automatically activated when entering an unknown network, like a café's Wi-Fi, or if an attack is detected. This mode is meant for situations where more privacy is needed and may slightly tend to overblocking.

Fortress

This is the emergency mode and should be used in highly untrusted environments, and if an attack is imminent or in progress ("You clicked on that link, didn't you…"). This mode will activate all available mechanisms to keep you safe and will undoubtedly overblock and break applications. It is meant as a small bandage until an expert can verify that everything is ok (And that the link really just showed you cat pictures).

Configuration

How these modes behave and what features are activated can be configured to some extent in the Portmaster settings. Please note, that certain features cannot be turned off in the Fortress mode in order to prevent misconfiguration and provide the same *Fortress* experience across installations - ie. you know what the Fortress mode does, even if it's not your computer you are using.

Application Profiles

`</> profiles`

Application Profiles are how you can control which application is allowed to connect to the Internet and how. Applications are matched by their installation path - be sure to have to path to the binary right to have a Profile applied (you can check the logs or the monitor tab in the UI).

Properties `</>_profiles#Profile`

All of the properties are explained where they are appear on settings page (press the small *i* icon), here we will go through them in some more detail:

- **Name:** Name of the application.
- **Description:** Description of the application. Meant for when users discover applications they know nothing about in the monitoring tool in the UI.
- **Security Level:** Define the minimum Security Level (and it's configured features) that should be applied with this Profile.
- **Default:** Define this profile as a default Profile. See explanation in section *Default Profiles* below.
- **Framework, Find, Build, Virtual, Find parent level, Merge with parent:** These are some kind of Helper-Profiles used to rematch special applications to correct profiles. See explanation in section *Framework Profiles* below.
- **Domain Whitelist**: Define a domain whitelist for this Profile, connections to all other domains will be denied.
- **ConnectPorts:**: Define a whitelist of remote TCP/UDP ports that applications are allowed to connect to.
- **ListenPorts:**: Define a whitelist of local TCP/UDP ports applications may listen on. Please note that the `Service` Flag needs to be set in order to allow listening at all.

Flags `</>_profiles#Profile`

Flags are an easy way to require or constraint to an application to a certain behavior.

- Executing User
  - **System:** System apps must be run by system user, else deny
  - **Admin:** Admin apps must be run by user with admin privileges, else deny
  - **User:** User apps must be run by user (identified by having an active safing UI), else deny
- Network Scope
  - **Internet:** Internet apps may connect to the Internet, if unset, all connections to the Internet are denied
  - **LocalNet:** LocalNet apps may connect to the local network (i.e. private IP address spaces), if unset, all connections to the local network are denied
- Network Destinations
  - **Strict:** Strict apps may only connect to domains that are related to themselves
  - **Service:** Service apps may accept incoming connections
  - **Direct Connect:** These apps may directly connect to an IP address, without resolving DNS first. This is unusual and makes it harder to protect privacy, but may be required for P2P applications.
- Special
  - **Gateway:** Gateway apps will connect to user-defined servers. Currently not in use.
  - **Browser:** Browsers are special in that their behavior cannot really be defined. Currently not in use.

Default Profiles `</>_profiles#Profile`

Because it is infeasible to have a separate Application Profile for every program you directly or indirectly use, you can also define a Profile for whole folders. These Profiles are called `Default` Profiles and are matched on a path prefix basis instead of an exact match basis.

Framework Profiles `</>_profiles#Framework`

This system is work in progress.

Sometimes a program path may not be the real entity that is executing code. Framework Profiles provide a means to identify the real actor behind a program. For example, when a python script is executed, the program path will be python interpreter, but we actually want to match against the script that is executing, not the interpreter.

- **Framework:** Defines that this Profile is a Framework profile. The program path will be rewritten and a new match will be tried. Should the new path not produce a match, this profile will be used as a fallback.

Going *down* the process tree - eg. finding the actual script of an interpreter:

- **Find:** Regex to find match groups within the path.
- **Build:** String that uses the regex match groups to build a new path. The resulting path is checked if it exists.
- **Virtual:** Do not check if the built path exists. This is useful to construct virtual namespaces for special categories of applications, like containerized/sandboxed applications.

Going *up* the process tree, using the path of the parent process to match a profile:

- **Find parent level:** Defines the number of levels to traverse the process tree up.
- **Merge with parent:** If true, view connections of this process as a part of the identified parent process.

Advanced Inspection

`</> firewall/inspection`

In order to better secure you from attacks and ensure that connections are genuine, the Portmaster supports packet inspection.

This framework allows to easily plug in new modules that check for various kinds of network attacks or verify connections.

TLS Verification `</> firewall/inspection/tls`

This module fully verifies TLS connections to ensure that all programs use the best available versions, ciphers and options. Will block outdated and vulnerable TLS connections.

Planned

These modules are planned and will be implement at some point in time.

- ARP Attack Detection: detect and mititage ARP-based MITM attacks.
- Portscan Detection: detect portscans to block attacking host.
- Network mapper: provide a network map to users.

OS Integration

`</> firewall/interception` `</> process`

Windows

WinDivert `</> firewall/interception/windivert`

The WinDivert API and kernel driver offer a similar interface to packet interception on Windows as divert socket.

While this works well, it's rather slow, so we are planning drastic performance improvements in the near future.

IP Helper API `</> process/iphelper`

The Windows API `IpHlpApi.dll` is used to fetch the table of current connections and get PID that belongs to the intercepted packet.

macOS

coming soon

Linux

On Linux we aim to provide two ways of OS integration:

iptables with nfqueue `</> firewall/interception/nfqueue`

Portmaster uses iptables and nfqueue to inspect and control network traffic. The nfqueue allows packets to be handed over to user space and return a verdict and set a mark on that connection.

Portmaster accepts all packets, but marks the whole connection to be accepted/dropped afterwards. This relieves Portmaster of heavy network traffic because once the fate of connection is decided, it is handed back to the kernel, never to be handed to userspace again, which is quite costly.

Here are the rules that Portmaster injects for both IPv4 and IPv6:

Chains:

```
mangle: C170
mangle: C171
filter: C17
```

Rules in own chains:

```
mangle C170 -j CONNMARK --restore-mark
mangle C170 -m mark --mark 0 -j NFQUEUE --queue-num {17040|17060} --queue-bypass

mangle C171 -j CONNMARK --restore-mark
mangle C171 -m mark --mark 0 -j NFQUEUE --queue-num {17140|17160} --queue-bypass

filter C17 -m mark --mark 0 -j DROP
filter C17 -m mark --mark 1700 -j ACCEPT
filter C17 -m mark --mark 1701 -j REJECT --reject-with {icmp-host-prohibited|icmp6-adm-prohibited}
filter C17 -m mark --mark 1702 -j DROP
filter C17 -j CONNMARK --save-mark
filter C17 -m mark --mark 1710 -j ACCEPT
filter C17 -m mark --mark 1711 -j REJECT --reject-with {icmp-host-prohibited|icmp6-adm-prohibited}
filter C17 -m mark --mark 1712 -j DROP
filter C17 -m mark --mark 1717 -j ACCEPT
```

Rules in main chains:

```
mangle OUTPUT -j C170
mangle INPUT -j C171
filter OUTPUT -j C17
filter INPUT -j C17
nat OUTPUT -p udp --dport 53 -m mark --mark 1799 -j DNAT --to {127.0.0.1:53|[::1]:53}
nat OUTPUT -m mark --mark 1717 -p {tcp|udp} -j DNAT --to-destination 127.0.0.17:1117 # for Port17
nat OUTPUT -m mark --mark 1717 -j DNAT --to-destination 127.0.0.17 # for Port17
```

Explanation of Nfqueue Numbers:

`17040` breaks up into:

- `17` is an identifier, so that you can easily spot what belongs to Portmaster/Port17
- `0` for ouput, `1` for input
- `4` for IPv4, `6` for IPv6
- `0` id for multi-threaded nfqueue (currently only one thread is used)

Explanation of Connmark Numbers:

```
1700 Accept
1701 Block
1702 Drop
1710 Permanent Accept
1711 Permanent Block
1712 Permanent Drop
1717 Reroute to Port17
1799 Reroute to nameserver (for astray DNS queries)
```

kernel module

We will provide an alternative to `iptables` by writing a kernel module to handle the needed packet interception in the future. Depending on the performance and stability of the `iptables` integration this might come sooner or later.

proc/net `</> process/proc`

In order to find out which process a packet belongs to, the proc filesystem is first parsed to find the socket id of the intercepted packet, then the process directory is search for the matching PID.

Docs: Port17

Security

> If you not have read the Tech Overview, please start there.

Our Promise

We know that every complex system can be broken, Port17 is no different. We do the best we can and we think we're doing quite good. Nevertheless we have to set limits to what extent we can and to which lengths we will go to protect you. Here we want to clearly communicate these limits:

Money
If you throw enough money at something long enough, it will eventually crumble. In our case we estimate how much an attack on Port17 costs and set a limit to what extent we promise to intervene. For now this limit is set at one million EUR (Q3 2018) - we will periodically review and update this value. Be assured that if there is a feasible solution, we will always react to a threat as fast as we can. Let's look at an example of how attack costs are estimated: If an attack requires 100 server to be run for 3 months and requires a month worth of work, we can estimate that this attack would cost about 19000€ (100 * 3 * 30€ + 10000€).

Legal
We will not break any laws. Our systems are designed to collect as little personal information as possible and we continually evaluate concepts to further decentralize our control over the Port17 network. Should we ever be forced to break the system or collect data on you, Safing (the company) will *hara-kiri*.

Privacy

To protect your privacy, we use a double ratchet based protocol to provide *perfect forward and backward* secrecy. Used algorithms can be easily changed to address new threats and trust issues. If you want to dig deeper, check out our crypto libarary: Tinker

Your private data is perfectly safe within Port17.

Anonymity

In order to anonymize connections, they are routed through multiple network nodes, encrypting/decrypting your data at every node. This proven method, known as onion-routing, is the state-of-the-art for anonymity networks - which has well known weaknesses.

There are three possible attacks to break this anonymity:

Rogue Nodes

The obvious threat to anonymity are nodes that are compromised or silently collude. These nodes then extract session data and if a route is chosen that consists *entirely* of such nodes, the attacker can link the source to the destination.

The best way to tackle this problem is have a great community which hosts a lot of nodes. Node owners are encouraged to mark their nodes as a group. Clients will then aim to diversify node ownership when calculating routes through the network.

At least for the first phase of operation, we will have to ability to *moderate* community nodes so that we can blacklist nodes that do not act in the interest of the community. We are looking into options how to decentralize this control over the network.

Traffic Analysis

One major concern with all anonymity networks is traffic analysis. With enough network visibility, one can easily find out where a connection really goes to and comes from, without even interacting with the network itself.

To provide the best possible anonymity, we studied the principles of *anti traffic analysis* described by *Gordon Welchman* in his book *The Hut Six Story*.

Here we compare these principles with Port17.

| # | Principle | Port17 | |:–|

Port17 Architecture

The Mesh Network

Port17 employs a static - but dynamically created - mesh network. Clients may only move within existing connections and will never trigger a new layer 4 connection to be established.

Bootstrapping `</>_port17/manager`

When a Port17 Node comes online for the first time, it needs to bootstrap itself to the network.

Before doing this the node initializes itself:

- it generates its identity (a private/public key pair)
- it configures its `Bottle`

To join the network, the node then downloads a couple of `Bottles` from `https://bootstrap.safing.io/bootstrap-nodes.json`.

> Note: until the internal Port17 update system is in place, bootstrap.safing.io will also be used for updates.

These Bottles are then used for an initial connection to the network.

When a node (or client) has been offline for over a day, the `Bottle` of the node they want to connect to may not have any valid ephemeral keys left, so they will have to send a `Seagull` to fetch a new `Bottle` before connecting.

When finally connected to the first node, the system will request that the nodes sends all `Bottles` in store to get the internal network map up to date. These new `Bottles` are all handled as explained in the `Bottle` section.

Tunneling

Port17 tunneling is done on layer 5 in the network layer model. This means that layer 4 (eg. TCP, UDP, …) is terminated locally and *cannot* leak any information, such as your IP address. Everything above, including TLS, is routed through the Port17 network without mangling.

New connections are always first reviewed by Portmaster to determine if and how it should be handled by Port17. Connections are then redirected to a local port, where Port17 awaits new connections. This redirection is done either by DNS or - if connecting to an IP address directly - diverted Portmaster.

Upon accepting a new connection, Port17 matches it to the information received by Portmaster and sets possible custom options. The `port17/navigator` is then asked to calculate a route which is then built and the connection is then forwarded.

Bottles (Gossip) `</>_port17/bottle`

Nodes exchange information about themselves by passing `Bottles` around. These `Bottles` are both sent to all connected nodes and are multicasted to the local network.

`Bottles` appear in two flavours: `public` and `local`.

`public Bottles` represent nodes on the public backbone. Most of them advertise a public IPv4 and/or IPv6 address. There are also nodes, which do not propagate an IP address. Clients only use these as intermediary nodes.

If a `public Bottle` is received it is handled like this:

- `port17/manager` Bottle is received
- `port17/bottle` Bottle is parsed, signature and validity is checked
- `port17/bottlerack` Bottle is compared to stored version, continue if new or changed.
- `port17/manager` verify new/changed advertised IP addresses
  - if verification failed, forward bottle as distrusted.

- `port17/manager` forward bottle to all connected nodes and clients, as well as locally

`local Bottles` represent nodes in the local network.

If a `local Bottle` is received it is handled like this:

- `port17/manager` The bottle is parsed.
- `port17/bottle` Signature is checked. If invalid, abort.
- `port17/manager` If the bottle does not advertise any IP addresses,
- `port17/bottlerack` Bottle is compared to stored version, continue if new or changed.
- `port17/manager` verify new/changed advertised IP addresses
  - if verification failed, forward bottle as distrusted.
- `port17/manager` if PortName changed, verify if name is unique, otherwise, abort.
- `port17/manager` forward bottle to all connected nodes and clients, as well as locally

Routing

`</> port17/navigator` `</> port17/manager`

Routes are entirely chosen by the clients:

- Exit nodes are chosen in proximity to the destination in order to protect the connection as long as possible.
- Node ownership is diversified.
- The fastest route with at least 3 hops is chosen.

In the future, users will have multiple options to influence how routes through the network are calculated:

- Exclude nodes by name or group
- Exclude areas by AS-Number, Country or IP range

  This product includes GeoLite2 data created by MaxMind, available from http://www.maxmind.com.

Port17 Network Stack

`</> port17`

Overview

``` A Port17 Node

```
+
```

# OS Integration

## Cross-platform

## Windows

### WinDivert `</> firewall/interception/windivert`

If DNS is resolved for a connection, Portmaster replies with an IP address in the `127.17/16` range. This enables the Portmaster to distinguish between domains even if they resolve to the same IP address. These connections are then rerouted to the Port17 entry point ( `127.0.0.17:1117` ) by `NAT` ing and reinjecting them.

If connecting to an IP address directly, the Portmaster directly `NAT` s to `127.0.0.17:1117` .

While this works well, it’s rather slow, so we are planning drastic performance improvements in the near future.
```

macOS

coming soon

Linux

On Linux we aim to provide two ways of OS integration:

iptables `</> firewall/interception/nfqueue`

If DNS is resolved for a connection, Portmaster replies with an IP address in the `127.17/16` range. This enables the Portmaster to distinguish between domains even if they resolve to the same IP address. These connections are then rerouted to the Port17 entry point (`127.0.0.17:1117`) by marking them with `1717` .

If connecting to an IP address directly, the Portmaster marks the connection with `1717` and the iptables rules below will redirect the connection.

Three additonal rules are added to the iptables main chains:

```
nat OUTPUT -m mark --mark 1717 -p {tcp|udp} -j DNAT --to-destination 127.0.0.17:1117
nat OUTPUT -m mark --mark 1717 -j DNAT --to-destination 127.0.0.17
```

kernel module

We will provide an alternative to `iptables` by writing a kernel module to handle the needed packet interception in the future. Depending on the performance and stability of the `iptables` integration this might come sooner or later.

FAQ

What is your business model?

Check out how we *business* and where our money comes from.

Where is the specification?

This site (docs.safing.io) describes how Safing works and is the authoritative documentation/specification - if the source code deviates, it's a bug. We do not go into very fine technical detail here because we do not want to replicate information already provided by the great GoDoc tool. We will refer to documentation within GoDoc where appropriate.

Ask a question

Depending on your question we would ask you to use the following channels:

- Usage of Safing in your Environment: Safing Community
- Question about how Safing works (ie. something is not clearly explained on this site): Raise an issue on Github