Calamary

Superstes

May 01, 2024

CONTENTS

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

CHAPTER

ONE

1- INTRO

Calamary is a squid-like proxy.

Its focus is set on **security filtering for HTTPS/TLS**.

The ruleset should be logical, transparent & easy to understand.

Features:

- support for mainstream *proxy modes*
- filtering ruleset - see *Rules*
  - ability to filter on protocol-basis
  - ability to enforce TLS (*deny any unencrypted connections*)
- TLS handling
  - certificate validation
  - peaking information without decrypting traffic
  - interception-mode with decryption
  - ability to block ECH/ESNI
- QUIC support
- detect plain HTTP and respond with generic HTTPS-redirect
- en- & disable parsing of protocols

Calamary will not:

- act as caching proxy
- act as reverse proxy
- implement edge-case workarounds for unencrypted protocols

1.1 TLS handling

Most of todays internet traffic is encrypted. Therefore a proxy needs to focus on handling it.

Calamary is striving to get the most out of TLS-peaking as TLS-interception might not be the solution for everyone.

TLS-interception has some drawbacks:

- possible legal issues
- costly processing & higher latency
- security issue if the CA gets compromised

But so does **TLS-peaking**:

- unable to filter ECH/ESNI protected traffic at all
- far less information available to filter on
  - unable to determine application-protocol

1.2 Getting Started

Getting started

1.3 Why?

Forward proxies are very useful to enforce a security-baseline in networks and a must-have for Zero-Trust environments.

Many enterprises and individuals will use proxies integrated with vendor network-firewalls or cloud-services to handle this filtering.

But some of us might like to keep control over that system.

The usage of go-based applications is easy (*single binary*) and can perform well.

1.3.1 Why not use Squid?

Squid has some limitations that make its usage more complicated than it should be.

Per example:

- intercept/transparent mode - no native solution for the DNAT restrictions

  Related errors:

  - *NF getsockopt(ORIGINAL_DST) failed*
  - *NAT/TPROXY lookup failed to locate original IPs*
  - *Forwarding loop detected*

- intercept/transparent mode - host verification - using DNS

  does hit issues with todays DNS-handling of major providers:

  - TTLs <= 1 min (*p.e. download.docker.com, debian.map.fastlydns.net*)

  Related error: *Host header forgery detected*

Squid is a good and stable software. But I get the feeling it needed to grow into more than it was designed for initially. Some behavior is inconsistent between modes and not optimized for todays IT-world.

I would much preferr a keep-it-simple approach. Even if that means that some nice-to-have features are not implemented.

1.4 How?

- Plaintext HTTP is not that common anymore.

  We are using TLS-SNI > Host-Header to resolve the target.

  Plain HTTP is unsecure by default. So we won't check for Host-Header mangling.

  The ruleset is applied 'postrouting' (*IP/Net matching*) and Host-Header domains are ignored by the ruleset.

- Whenever it is not possible to route the traffic through the proxy..

  To overcome the DNAT restriction, of losing the real target IP, there will be a *Redirector*!

- **Transparent traffic interception will be the focus**.

  Setting the environment-variables 'HTTP_PROXY', 'HTTPS_PROXY', 'http_proxy' and 'https_proxy' for all applications and HTTP-clients may be problematic/too inconsistent

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Warning: Development & writing of this documentation is still in progress!

CHAPTER

TWO

2- GETTING STARTED

2.1 Installation

Download the binary for your desired target system from the latest relase page

2.2 Run

Just execute it.

```
/usr/bin/calamary
```

Config-validation only:

```
/usr/bin/calamary -v
```

2.3 Modes

See: *Modes*

2.4 Configuration

See *Rules* for more details about defining the filter ruleset.

The default config path is `/etc/calamary/config.yml`

Basic config example:

```yaml
---

service:
  listen:
    - mode: 'transparent'
      ip4: ['127.0.0.1']  # is default
      ip6: ['::1']  # is default
      port: 4128
      tcp: true
```

(continues on next page)

(continued from previous page)

```
    udp: false  # not yet implemented
    tproxy: false

certs:
  serverPublic: '/tmp/calamary.crt'
  serverPrivate: '/tmp/calamary.key'
  interceptPublic: '/tmp/calamary.subca.crt'
  interceptPrivate: '/tmp/calamary.subca.key'

dnsNameservers: ['1.1.1.1', '8.8.8.8']
debug: false
timeout:  # ms
  connect: 2000
  process: 1000
  idle: 30000

output:
  retries: 1  # connect-retries

metrics:
  enabled: false
  port: 9512

vars:
  - name: 'net_private'
    value: ['192.168.0.0/16', '172.16.0.0/12', '10.0.0.0/8']
  - name: 'svc_http'
    value: [80, 443]

rules:
  - match:
      dest: '192.168.100.0/24'
    action: 'drop'

  - match:
      port: ['!443', '!80']
    action: 'drop'

  - match:
      src: '$net_private'
      dest: '$net_private'
      port: '$svc_http'
      protoL4: 'tcp'
    action: 'accept'

  - match:
      dest: '!$net_private'
      port: 443
      protoL4: 'tcp'
    action: 'accept'
```

2.5 Redirect traffic

See *Redirect traffic*

2.5.1 Systemd service

Example systemd service to run Calamary:

```
[Unit]
Description=Calamary Forward Proxy
Documentation=https://docs.calamary.net
Documentation=https://github.com/superstes/calamary
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
Environment=CONFIG=/etc/calamary/config.yml

# validate before start/restart
ExecStartPre=/usr/bin/calamary -f $CONFIG -v
ExecStart=/usr/bin/calamary -f $CONFIG

# validate before reload
ExecReload=/usr/bin/calamary -f $CONFIG -v
ExecReload=/bin/kill -HUP $MAINPID

User=proxy
Group=proxy
Restart=on-failure
RestartSec=5s

StandardOutput=journal
StandardError=journal
SyslogIdentifier=calamary

[Install]
WantedBy=multi-user.target
```

2.6 Build from sources

Download and 'install' Golang 1.21 to build the binary from sources: Golang download

```
git clone https://github.com/superstes/calamary
cd calamary/lib
go mod download
cd main/
go build -o calamary
```

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Warning: Development & writing of this documentation is still in progress!

CHAPTER

THREE

DEBUG

If you encounter any unexpected behaviour you might want to enable the debug mode in your config-file to let Calamary produce more verbose logs.

3.1 Examples

3.1.1 Tranparent Mode - Plaintext HTTP

```
2023-10-02 00:10:30 | DEBUG | service | Accept: tcp://127.0.0.1:4128
2023-10-02 00:10:30 | DEBUG | parse | 192.168.11.104 => ? | Parsing TCP connection
2023-10-02 00:10:30 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:80 | Processing␣
↪TCP
2023-10-02 00:10:30 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:80 | TLS␣
↪information: IsTls=false, TlsVersion=0, TlsSni=
2023-10-02 00:10:30 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:80 | Packet␣
↪L5Proto: HTTP | TLS: none
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 0 -␣
↪DNet: [192.168.100.0/24] vs 135.181.170.219
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 1 - !
↪DPort: [443 80] vs 80
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 2 -␣
↪Proto L4: [10] vs 10
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 2 -␣
↪SNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 192.168.11.104
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 2 -␣
↪DNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 135.181.170.219
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 3 -␣
↪Proto L4: [10] vs 10
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 3 - !
↪DNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 135.181.170.219
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 3 -␣
↪DPort: [443 80] vs 80
2023-10-02 00:10:30 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:80 | Rule 3 -␣
↪Applying action 'accept'
2023-10-02 00:10:30 | INFO | forward | 192.168.11.104 => 135.181.170.219:80 | Accept
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | Connection␣
↪established
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | Forwarding
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | 5 bytes sent
```

(continues on next page)

(continued from previous page)

```
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | 76 bytes sent
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | 351 bytes␣
↪received
2023-10-02 00:10:30 | DEBUG | send | 192.168.11.104 => 135.181.170.219:80 | Closed
```

3.1.2 Tranparent Mode - HTTPS

```
2023-10-02 00:13:20 | DEBUG | service | Accept: tcp://127.0.0.1:4128
2023-10-02 00:13:20 | DEBUG | parse | 192.168.11.104 => ? | Parsing TCP connection
2023-10-02 00:13:20 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:443 | Processing␣
↪TCP
2023-10-02 00:13:20 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:443 | TLS␣
↪information: IsTls=true, TlsVersion=771, TlsSni=superstes.eu
2023-10-02 00:13:20 | DEBUG | parse | 192.168.11.104 => 135.181.170.219:443 | Packet␣
↪L5Proto: TLS | TLS: 1.2
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 0 -␣
↪DNet: [192.168.100.0/24] vs 135.181.170.219
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 1 - !
↪DPort: [443 80] vs 443
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 2 -␣
↪Proto L4: [10] vs 10
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 2 -␣
↪SNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 192.168.11.104
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 2 -␣
↪DNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 135.181.170.219
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 3 -␣
↪Proto L4: [10] vs 10
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 3 - !
↪DNet: [192.168.0.0/16 172.16.0.0/12 10.0.0.0/8] vs 135.181.170.219
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 3 -␣
↪DPort: [443 80] vs 443
2023-10-02 00:13:20 | DEBUG | filter | 192.168.11.104 => 135.181.170.219:443 | Rule 3 -␣
↪Applying action 'accept'
2023-10-02 00:13:20 | INFO | forward | 192.168.11.104 => 135.181.170.219:443 | Accept
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | Connection␣
↪established
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | Forwarding
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | 517 bytes␣
↪sent
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | 4096 bytes␣
↪received
...
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | 842 bytes␣
↪sent
2023-10-02 00:13:20 | DEBUG | send | 192.168.11.104 => 135.181.170.219:443 | Closed
```

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

CHAPTER

FOUR

METRICS

Calamary provides a built-in Prometheus exporter.

You can enable it using the config-file:

```
---

service:
  ...

  metrics:
    enabled: false
    port: 9512

...
```

4.1 Test

```
curl http://localhost:9512/metrics
```

4.2 Metric items

```
# traffic
calamary_bytes_rcv 12707
calamary_bytes_sent 1760
calamary_current_connections 1
calamary_req_protoL3{protocol="IPv4"} 3
calamary_req_protoL5{protocol="HTTP"} 1
calamary_req_protoL5{protocol="TLS"} 2
calamary_req_tcp 3
calamary_req_tls_version{version="1.2"} 2
calamary_req_tls_version{version="none"} 1

# rules
calamary_rule_actions{action="accept"} 3
calamary_rule_actions{action="deny"} 1
calamary_rule_hits{ruleId="0"} 3
```

(continues on next page)

(continued from previous page)

```
calamary_rule_hits{ruleId="1"} 3
calamary_rule_hits{ruleId="2"} 3
calamary_rule_hits{ruleId="3"} 3
calamary_rule_matches{ruleId="3"} 3
```

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Warning: Development & writing of this documentation is still in progress!

CHAPTER

FIVE

MODES

5.1 Transparent

5.1.1 Info

State: Implemented/Testing

Calamary focuses on transparent traffic interception.

You will have to redirect the traffic: *Redirect*

This mode will work for TCP & UDP.

5.1.2 Behavior

DNAT - TCP (plaintext)

Server

```
2023-10-01 23:43:01 | INFO | 192.168.11.104 => 135.181.170.219:80 | Accept
```

Client

```
curl http://superstes.eu -v
*   Trying 135.181.170.219:80...
* Connected to superstes.eu (135.181.170.219) port 80 (#0)
> GET / HTTP/1.1
> Host: superstes.eu
...
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host superstes.eu left intact
```

DNAT - TLS

Server

```
2023-10-01 23:43:09 | INFO | 192.168.11.104 => 135.181.170.219:443 | Accept
```

Client

```
host@calamary$ curl https://superstes.eu -v

*   Trying 135.181.170.219:443...
* Connected to superstes.eu (135.181.170.219) port 443 (#0)
...
< HTTP/2 302
< server: nginx
...
<
<html>
<head><title>302 Found</title></head>
<body>
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host superstes.eu left intact
```

5.2 HTTP Proxy

5.2.1 Info

State: Implemented/Testing

You can also choose to let Calamary act as a HTTP/S proxy.

One commonly uses this feature if only some applications should send their traffic over the proxy.

This mode only supports TCP.

Note: Calamary uses TLS-SNI > Host-Header to find its actual target host. It will also check all IPs (IPv6 > IPv4) that are returned by the DNS query for their reachability, before establishing a connection.

5.2.2 Behavior

HTTP

Server

```
2023-10-01 23:40:34 | INFO | 127.0.0.1 => 135.181.170.219:80 | Accept
```

Client

```
host@calamary$ http_proxy=http://localhost:4130 curl http://superstes.eu -v

* Uses proxy env variable http_proxy == 'http://localhost:4130'
*   Trying 127.0.0.1:4130...
* Connected to (nil) (127.0.0.1) port 4130 (#0)
> GET http://superstes.eu/ HTTP/1.1
> Host: superstes.eu
> User-Agent: curl/7.81.0
> Accept: */*
> Proxy-Connection: Keep-Alive
>
...
<
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host (nil) left intact
```

HTTPS

Server

```
2023-10-01 23:40:43 | INFO | 127.0.0.1 => 135.181.170.219:443 | Accept
```

Client

```
host@calamary$ https_proxy=http://localhost:4130 curl https://superstes.eu -v

* Uses proxy env variable https_proxy == 'http://localhost:4130'
*   Trying 127.0.0.1:4130...
* Connected to (nil) (127.0.0.1) port 4130 (#0)
* allocate connect buffer!
* Establish HTTP proxy tunnel to superstes.eu:443
> CONNECT superstes.eu:443 HTTP/1.1
> Host: superstes.eu:443
> User-Agent: curl/7.81.0
> Proxy-Connection: Keep-Alive
>
< HTTP/1.1 200 OK
< Content-Length: 0
* Ignoring Content-Length in CONNECT 200 response
<
* Proxy replied 200 to CONNECT request
* CONNECT phase completed!
...
> GET / HTTP/2
> Host: superstes.eu
> user-agent: curl/7.81.0
```

(continues on next page)

(continued from previous page)

```
> accept: */*
>
...
< HTTP/2 302
< server: nginx
...
<
* TLSv1.2 (IN), TLS header, Supplemental data (23):
<html>
<head><title>302 Found</title></head>
<body>
<center><h1>302 Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
* Connection #0 to host (nil) left intact
```

5.3 HTTPS Proxy

Has the same behavior like 'HTTP Proxy' but the transport from client to proxy is also encrypted.

5.3.1 Behavior

tbd

5.4 Proxy Protocol

5.4.1 Info

State: in development

You can use the proxy-protcol mode if you want to send traffic from remote systems over the proxy.

The commonly used proxy-protocol preserves the original source- & destination while minimizing overhead.

5.4.2 Behavior

tbd

5.5 SOCKS5

5.5.1 Info

State: not implemented

Like HTTP/S proxy, but it works for UDP as well.

5.5.2 Behavior

tbd

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Warning: Development & writing of this documentation is still in progress!

CHAPTER

SIX

REDIRECT TRAFFIC

6.1 Basics

You may want/need to redirect traffic to the proxy's listeners for some use-case.

This is essential for using the `transparent` mode.

For modes like `proxyproto`, `http`, `https` or `socks5` this is not necessary. (*but it's also possible using the Redirector*)

You will have to choose between using **DNAT** and **TPROXY** to redirect the traffic on firewall-level.

TProxy has the benefit that it won't modify the packets destination. This makes processing the traffic easier and can be benefitial in regards to performance.

But it also has the drawback that traffic that originates from the proxy-server (*netfilter hook - output*) will have to be looped-back.

I personally like to use TProxy for filtering input/forward- and DNAT for output-traffic.

> **Warning:** The config-examples below may not be complete! If you find issues with them - please open an issue

6.2 NFTables

Read more about NFTables here: wiki.superstes.eu - NFTables

6.2.1 DNAT

```
define PROXY_PORT=4128
define PROXY_UID=13  # user-id of the proxy-user; anti-loop
define PROXY_PORTS={ 80, 443, 587, 25, 53, 853, 123 }

table inet default {
    chain prerouting_dnat {
        type nat hook prerouting priority dstnat; policy accept;

        # redirect traffic from outside
        meta l4proto tcp redirect to $PROXY_PORT
        # redirect to = equivalent to 'dnat to 127.0.0.1/::1'
    }
```

(continues on next page)

(continued from previous page)

```
    chain output_dnat {
        type nat hook output priority -100; policy accept;

        # redirect traffic from this host
        meta l4proto tcp meta skuid != $PROXY_UID redirect to $PROXY_PORT
    }

    chain input {
        type filter hook output priority 0; policy drop;

        # allow traffic to proxy
        meta l4proto tcp dport $PROXY_PORT ip daddr 127.0.0.1 accept comment "Allow␣
↪Network to proxy"
        meta l4proto tcp dport $PROXY_PORT ip6 daddr ::1 accept comment "Allow Network␣
↪to proxy"
    }

    chain output {
        type filter hook output priority 0; policy drop;

        # allow traffic to proxy
        meta l4proto tcp dport $PROXY_PORT ip daddr 127.0.0.1 accept comment "Allow␣
↪localhost to proxy"
        meta l4proto tcp dport $PROXY_PORT ip6 daddr ::1 accept comment "Allow localhost␣
↪to proxy"

        # optional logging
        meta l4proto tcp dport $PROXY_PORTS meta skuid $PROXY_UID ct state new log␣
↪prefix "NFTables Proxy outgoing "
        # allow traffic from proxy
        meta l4proto tcp dport $PROXY_PORTS meta skuid $PROXY_UID accept comment "Allow␣
↪proxy traffic"
    }
}
```

6.2.2 TProxy

Full TProxy example: gist.github.com/superstes - TProxy NFTables

You might need to enable some nftables kernel modules: Kernel docs - NFTables extensions

6.3 IPTables

6.3.1 DNAT

```
PROXY_PORT=4128
PROXY_UID=13  # user-id of the proxy-user; anti-loop

# redirect traffic from outside
iptables -t nat -A PREROUTING -p tcp -j REDIRECT --to-destination --to-port "$PROXY_PORT"
# redirect traffic from localhost
iptables -t nat -A OUTPUT -p tcp -m owner ! --uid-owner "$PROXY_UID" -j REDIRECT --to-
↪port "$PROXY_PORT"

# allow traffic to proxy
#   iptable -A INPUT -p tcp -d 127.0.0.1 --dport "$PROXY_PORT" -j ACCEPT
#   iptable -A OUTPUT -p tcp -d 127.0.0.1 --dport "$PROXY_PORT" -j ACCEPT

# optional logging
iptables -A OUTPUT -p tcp -m conntrack --ctstate NEW -j LOG -log-prefix "IPTables Proxy␣
↪outgoing "
# allow traffic from proxy
iptable -A OUTPUT -p tcp -m owner ! --uid-owner "$PROXY_UID" -j ACCEPT
```

6.3.2 TProxy

Full TProxy example: gist.github.com/superstes - TProxy IPTables

You might need to enable some iptables kernel modules: Kernel docs - IPTables extensions

6.4 TProxy

To run Calamary as TPROXY target - you will have to set CAP_NET_RAW:

```
bind to any address for transparent proxying
```

You can add it like this:

```
setcap cap_net_raw=+ep /usr/bin/calamary

# make sure only wanted users can execute the binary!
chown root:proxy /usr/bin/calamary
chmod 750 /usr/bin/calamary
```

Read more about TPROXY here:

- wiki.superstes.eu - NFTables - TProxy
- kernel docs

6.4.1 Output loopback

You will have to configure a loopback route if you want to proxy 'output' traffic:

```
echo "200 proxy_loopback" > /etc/iproute2/rt_tables.d/proxy.conf

# These need to be configured persistent: (maybe use an interface up-hook)
ip rule add fwmark 200 table proxy_loopback
ip -6 rule add fwmark 200 table proxy_loopback
ip route add local 0.0.0.0/0 dev lo table proxy_loopback
ip -6 route add local ::/0 dev lo table proxy_loopback

# can be checked using:
ip rule list
ip -6 rule list
ip -d route show table all

# you might need to set a sysctl:
sysctl -w net.ipv4.conf.all.route_localnet=1

# you might want to block 127.0.0.1 on non loopback interfaces if you enable it:
iptables -t raw -A PREROUTING ! -i lo -d 127.0.0.0/8 -j DROP
iptables -t mangle -A POSTROUTING ! -o lo -s 127.0.0.0/8 -j DROP
```

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

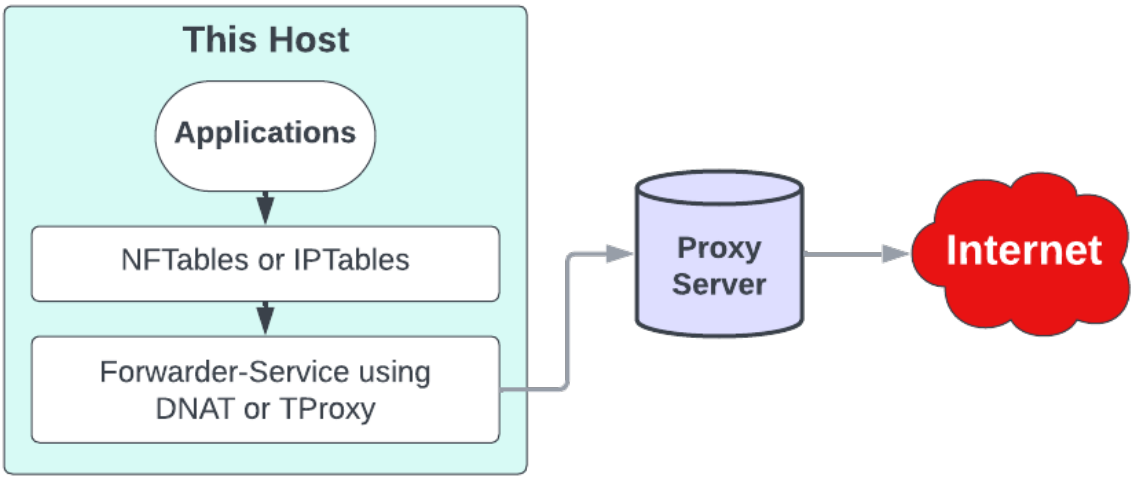
CHAPTER

SEVEN

REDIRECTOR

The redirector will be a smaller version of Calamary. (*without filtering*)

It can be used to forward traffic from remote locations to your proxy servers.



Per example - this might be useful if you have:

- Distributed systems
  - Cloud servers that are only connected to public WAN and should send all their outbound traffic over your proxy
  - Simple/dumb firewalls/routers that should send the networks outbound traffic over your proxy

As it utilizes the commonly used proxy-protocol to redirect the traffic, it might also be useful in combination with other proxies.

Tip: Check out the Repository on GitHub

Warning: **WARNING**: This Project is still in early development

Warning: Development & writing of this documentation is still in progress!

CHAPTER

EIGHT

RULES

You can define a list of rules that calamary will apply to the traffic passing through it.

Rules basically consist of a `match` and an `action`!

```
rules:
  - match:
      dest: '192.168.100.0/24'
    action: 'drop'
```

8.1 Matches

Multiple matches can be defined in a single rule.

```
- match:
    src: 'IP OR NET+CIDR'
    dest: 'IP OR NET+CIDR'
    port: 'NUMBER'  # destination ports
    sport: 'NUMBER'  # source ports
    protoL3: 'ip4/ivp4/ip6/ip6'
    protoL4: 'tcp/udp'  # others might be supported later on
    protoL5: 'tls/http/dns/ntp'  # others might be supported later on
    dns: 'DOMAIN'  # domain/TLS-SNI to match
    encrypted: 'true/false/yes/no'  # match TLS traffic
```

The value of matches is **case-insensitive** by default.

NOTE: The HTTP host-header domain is not compared if `dns` is used - as it can be modified easily.

You can define **multiple values** for each match.

Matches can also be **negated** by using the `!` prefix:

```
rules:
  - match:
      port: ['!80', '!443', '!587']
    action: 'drop'

  - match:
      dest: '!192.168.0.0/16'
      port: 443
```

(continues on next page)

(continued from previous page)

```
      protoL4: 'tcp'
    action: 'accept'
```

Packets that don't match any `accept` rule will be **dropped by default**.

8.2 Actions

Available actions include:

- 'accept' (*alias: 'allow'*)
- 'deny' (*alias: 'drop'*)

Other actions like 'limit' will be implemented later on.

8.3 Variables

Calamary enables you to define variables that can be used inside your ruleset.

```
vars:
  - name: 'net_private'
    value: ['192.168.0.0/16', '172.16.0.0/12', '10.0.0.0/8']
  - name: 'svc_http'
    value: [80, 443]
```

Variables are referenced using the $ prefix.

Whenever you use a variable, you can also negate it like any other value:

```
rules:
  - match:
      src: '$net_private'
      dest: '!$net_private'
    action: 'accept'
```