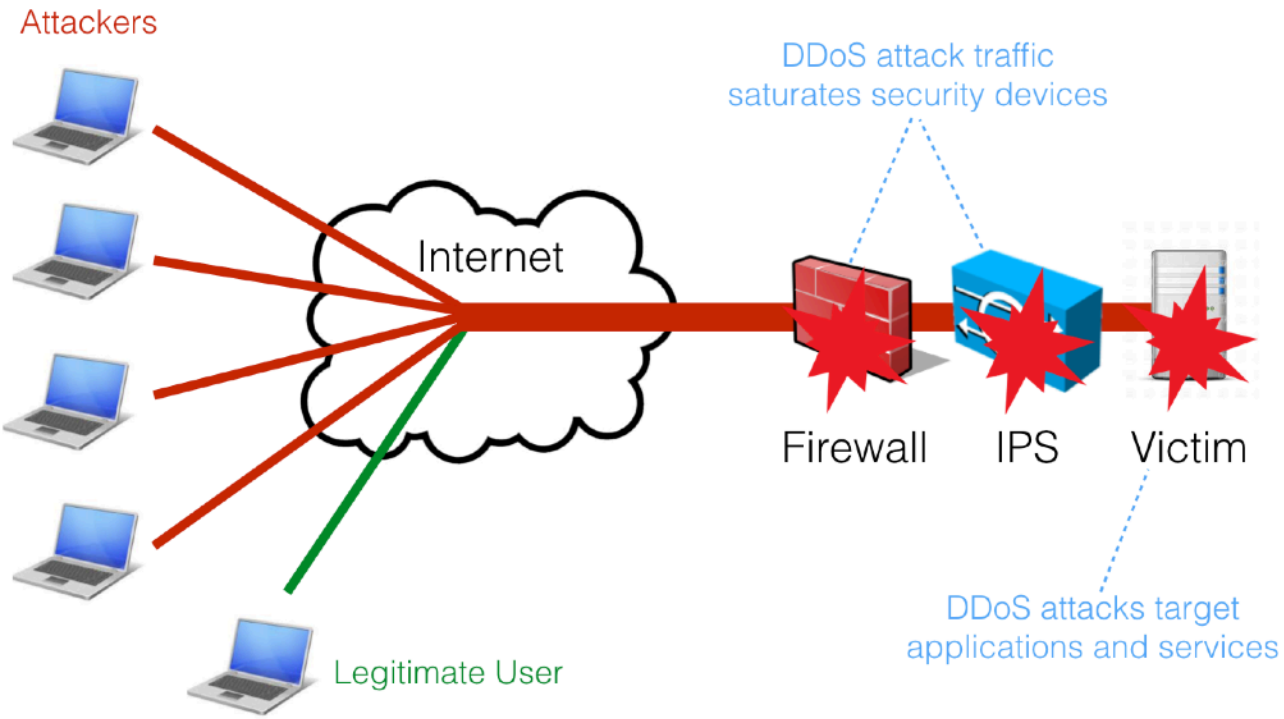
nScrub
User's Guide

Introduction

A DDoS (Distributed Denial of Service) attack is an attempt to make a service unavailable by overwhelming it with malicious traffic from multiple distributed sources.

Common routers, firewalls and IPSs are not designed to handle high volume of packets per second and are often the the first victim of the Denial of Service attacks. Specialised solutions are needed to protect the infrastructure, solutions that can not only route or bridge high packet rates but also able to filter and challenge malicious traffic.



Such dedicated solutions in the past where implemented by dedicated hardware with specially designed ASICs or FPGAs capable of handling high speed filtering.

nScrub is a software-based DDoS mitigation tool based on PF_RING and Zero Copy able to operate at 10 Gbps line rate (no matter the packet size) using commodity hardware (e.g. Intel NICs and standard servers).

nScrub is easy to configure, even for beginners and companies that never faced with DDoS until now. Affordable in terms of price, deployment, reporting. A looking glass to see the status of the network while mitigating attacks.

nScrub not only performs as good as a hardware based solution but provide a flexibility that makes it unique:

- Easy Deployment: nScrub can be implemented as bump in the wire (i.e. no BGP or traffic tunnelling necessary to deploy nScrub).
- Low Cost: very low TCO (Total Cost of Ownership).
- Modularity: extensibility for the definition of additional algorithms for traffic mitigation. They can be implemented through an API that allows developers to create their own mitigation facilities to be used in addition to those part of nScrub.

- Rich Detection: precise mitigation based on algorithms that try to determine whether the incoming traffic is real/legitimate.
- Auto-healing: automatic escalation during mitigation: when things get bad, the user has the ability to enable auto-escalation to put in place a proper reaction to threats.
- Not hardware locking: nScrub can operate with multiple hardware vendors avoiding vendor locking.

The main features include:

- Support for asymmetric (i.e. ability to mitigate only one traffic direction, from Internet to the protected network) and symmetric (i.e. mitigate from Internet to the protected network, but also forward the outbound traffic) mode. Asymmetric mode allows also to deploy nScrub as a off-ramp solution (scrubbing center) by means of BGP re-routing.
- Detection of real/legitimate traffic by means of techniques that try to detect if the sender is real or not and the session is legitimate or not.
- Ability to monitor/visualise legitimate/good ingress traffic as well as discarded/bad traffic, both sampled or not, forwarding traffic to egress interfaces to be sent to monitoring machines or IDS (e.g. snort), or to be recorded and analysed at later stage.
- Ability to configure mitigation profiles per destinations (i.e. the network to protect) so that each host of the protected network can have a custom protection policy.
- Multi protocol application support including DNS, HTTP and HTTPS

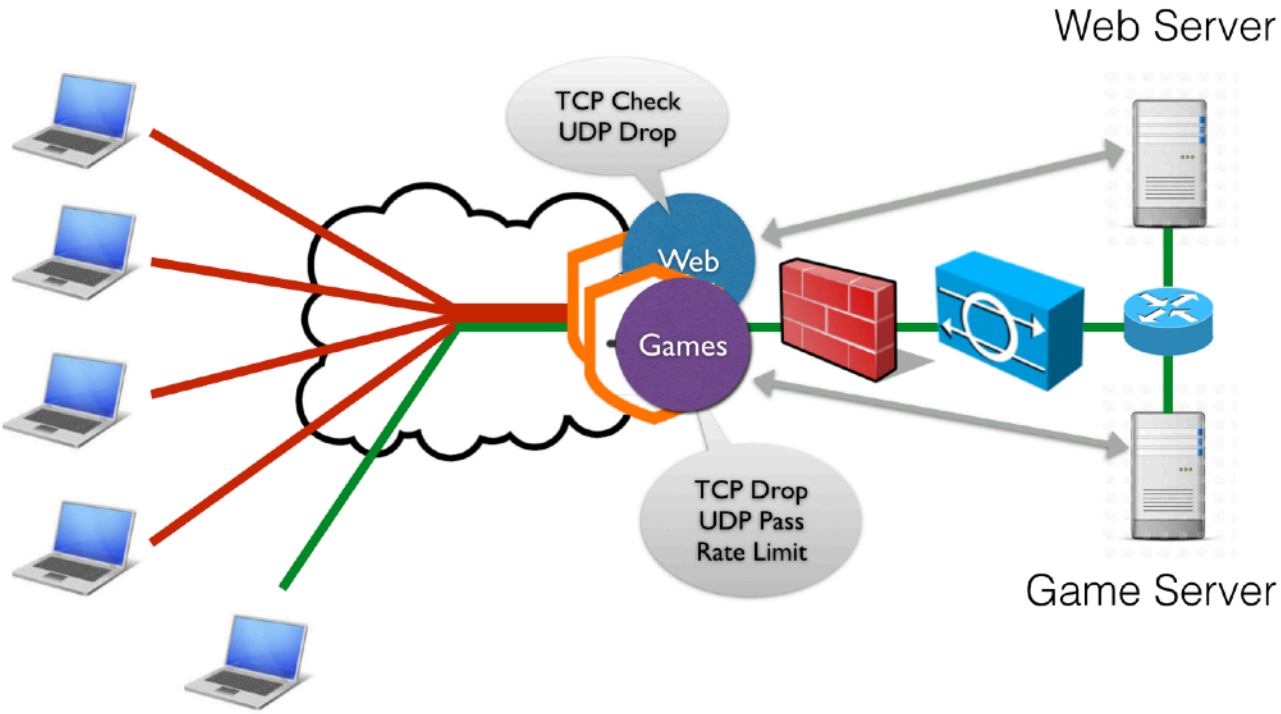
Main Features

Traffic enforcement

- TCP sessions validation
- Dynamic whitelisting with expiration on successful session check
- User-defined whitelist/blacklist/graylist of source subnets with CIDR notation
- ACL-like accept/drop policies based on UDP/TCP port, ICMP type, etc.
- Other drop policies based on IP TTL values, UDP payload size, fragments, etc.
- DNS SLIP-like checks: force TCP, etc.
- Mitigation UDP-based amplification attacks.
- Signature-based filtering (offset and string)
- HTTP filtering, based on request items name/content.
- Traffic Throttling: packets below the threshold are forwarded, otherwise they are discarded. This guarantee that unwanted traffic will have an egress rate capped to a specific value. Ability to specify the rate based on protocol and source or destination.
- Traffic checkers are implemented as plugins with a clean API, so that more checkers for specific protocols can be created.

Multi-Tenancy

- Ingress traffic is split towards several virtual mitigators, based on the destination IP address, this way it is possible to specify traffic enforcement policies per destination subnet
- Each virtual mitigator is bound to traffic enforcement profiles: default, white, black, gray. Each profile contains a traffic enforcement configuration (e.g. SYN check=yes, ICMP Drop=No) and applies to source IPs according to the lists (white/black/gray).



- Global or per-destination bypass mode

Traffic Visibility

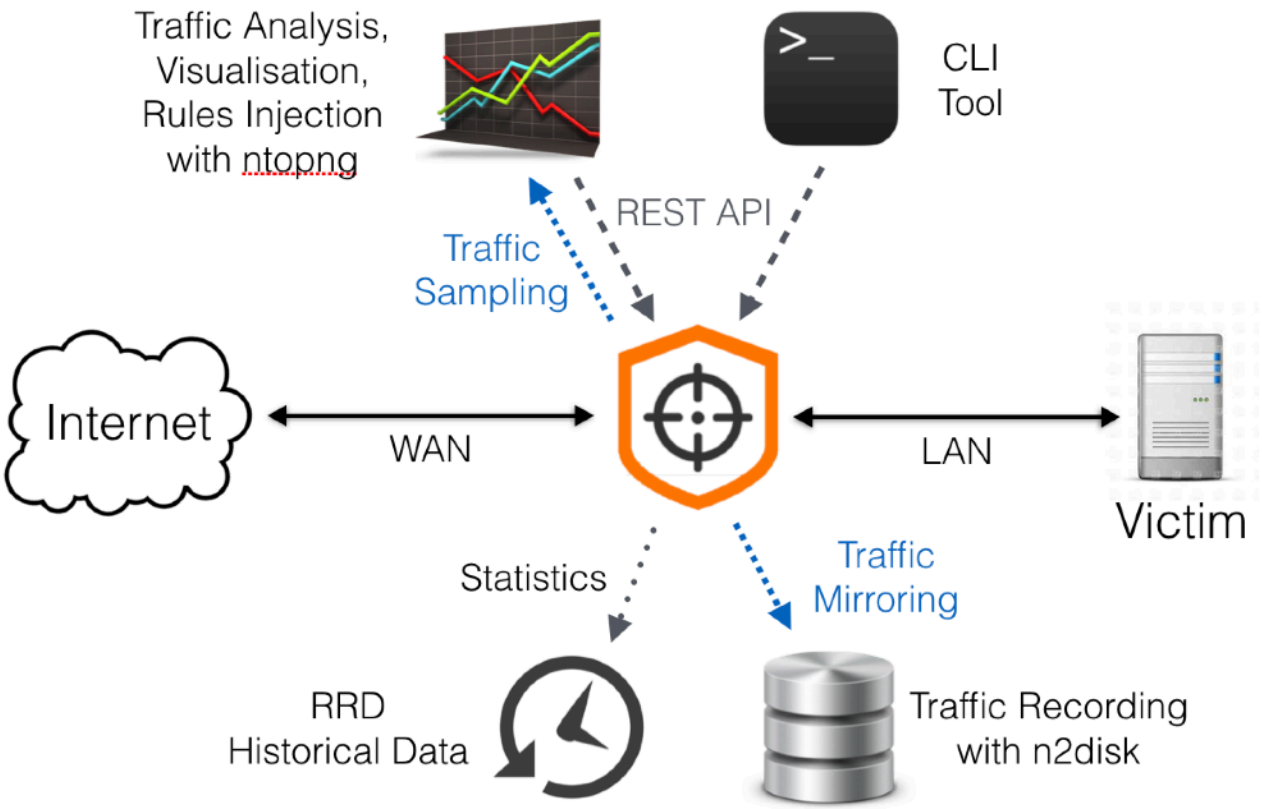
- Statistics dump to RRD for keeping an history of traffic trends.
- Ability to send sampled/full good/bad/all traffic to external virtual devices (e.g. for traffic analysis or dump).

Hw acceleration and Scalability

- Hardware bypass NIC support (Silicom): ensures that nScrub will have no impact in the infrastructure in case of hardware failure.
- Load balancing across cores using hw RSS or custom sw distribution

UI

- REST API for reconfiguring the engine on-the-fly
- CLI tool with auto-completion



Versions

The nScrub software comes in two versions, 1G and 10G. Each version comes in various flavours (S/M for 1G, L/XL/XXL for 10G) depending on speed and number of target servers that nScrub can handle.

The 10G version includes support for multithreaded packet capture and processing, with an internal architecture able to balance the load across multiple threads in a zero-copy fashion. This version can handle rates above 10 Gigabit.

Please find all nScrub flavours in the table below.

	S	**M**	**L**	**XL**	**XXL**
Speed	1 Gigabit	1 Gigabit	10 Gigabit	10 Gigabit	10 Gigabit
Servers	Up to 10	Unlimited	Up to 10	Up to 100	Unlimited

Installation

In order to run nscrub the following major steps are needed:

1. Install and configure PF_RING and ZC drivers (later in this chapter).
2. Install nScrub (later in this chapter) and configure it to run with the optimal setup (*Engine Configuration* and *Performance and Tuning* chapters).
3. Configure nScrub to mitigate traffic (*Traffic Enforcement Configuration* chapter).

Prerequisites:

- PF_RING for packet capture acceleration
- redis-server for storing configuration and system state

Please note that installation packages are provided for some Linux distributions including Ubuntu, Debian and CentOS at http://packages.ntop.org/. Please follow the steps on the website for configuring the repository and install at least the *pfring, pfring-drivers-zc-dkms* and *nscrub*.

PF_RING

After installing *pfring* and *pfring-drivers-zc-dkms*, it is possible to use the init script under /etc/init.d/pf_ring to automate the kernel module and drivers loading as explained in the *PF_RING User's Guide* at http://www.ntop.org/guides/pf_ring/get_started/packages_installation.html

For the impatient, this is an example of a minimal configuration for a dual-port ixgbe card (replace ixgbe with your driver model) on a uniprocessor:

```
# touch /etc/pf_ring/pf_ring.start
# mkdir -p /etc/pf_ring/zc/ixgbe
# touch /etc/pf_ring/zc/ixgbe/ixgbe.start
# echo "RSS=0,0" > /etc/pf_ring/zc/ixgbe/ixgbe.conf
# echo "node=0 hugepagenumber=1024" > /etc/pf_ring/hugepages.conf
```

Please note that RSS=0,0 will create as many interface queues as the number of cores, however since most Intel cards support up to 16 RSS queues for load balancing, in case of CPUs with many cores it's a good practice to force it to 16 using RSS=16,16.

Please also note that the number of hugepages requires for memory allocation depends on the number of interface queues (and auxiliary queues if any) and the MTU. For instance if you configure 12 RSS queues with an MTU of 1500 bytes, you probably need up to 2048 hugepages, if you configure 16 RSS queues a common configuration is 4096 hugepages.

In order to load pf_ring, if your system is using systemd please run:

```
# systemctl start pf_ring
```

Otherwise please use the init.d script:

```
# /etc/init.d/pf_ring start
```

nScrub

The nscrub installation is pretty straightforward, just install the *nscrub* package from the repository, then move to the next chapters.

License Setup

nScrub needs a license in order to operate permanently (i.e. not in demo mode). In order to obtain a license, you need to go to the web shop and order it using the system ID and nScrub version.

Identity version and system ID:

```
# nscrub -V
nscrub 1.0 [SystemID: 1234567890]
```

Get a license and install it:

```
# echo <LICENSE> > /etc/nscrub.license
```

You can now start your licensed nScrub copy. It is possible to check license status with:

```
# nscrub --check-license
```

and maintenance expiration with:

```
# nscrub --check-maintenance
```

Please note that in order to run nScrub on top of PF_RING ZC, you also need ZC licenses, one for each interface you are going to use. In order to obtain ZC licenses, you need to go to the web shop and order them using the MAC addresses as reported by:

```
# ifconfig <interface>
```

the driver model as reported by:

```
# ethtool -i <interface> | grep driver
```

and the PF_RING version as reported by:

```
 # zcount -h | grep PFRING_ZC
```

Engine Configuration

The engine can be configured using the REST API or the CLI tool *nscrub-cli* with auto-completion support (note this is also using the REST API as backend).
In order to use the API over HTTPs you need to create a SSL certificate:

```
# openssl req -new -x509 -sha1 -extensions v3_ca -nodes -days 365 -out cert.pem
# cat privkey.pem cert.pem > /usr/share/nscrub/ssl/ssl-cert.pem
# rm -f privkey.pem cert.em
```

In order to run nScrub using the init.d script, /etc/nscrub/nscrub.start should be created and the configuration file should be placed under /etc/nscrub/nscrub.conf. Example:

```
# touch /etc/nscrub/nscrub.start
# cat /etc/nscrub/nscrub.conf
--wan-interface=zc:eth1
--lan-interface=zc:eth2
--log-path=/var/tmp/nscrub.log
--pid-path=/var/tmp/nscrub.pid
--daemon
```

If your system is using systemd, the application can be run with:
In order to load pf_ring, if you are using systemd please run:

```
# systemctl start nscrub
```

Otherwise please use the init.d script:

```
# /etc/init.d/nscrub start
```

The following options can be specified in the configuration file. For default values please refer to the application help (`--help|-h` option).

Basic Settings

```
[--wan-interface|-i] <device>             | First device name (internet)
[--lan-interface|-o] <device>             | Second device name (local network)
[--asymmetric|-A]                         | Asymmetric routing (wan to lan
                                          | traffic only)
[--sw-distribution|-w]                    | Use SW Distribution + RSS TX Queues
[--active-wait|-a]                        | Active packet wait
```

Host Connectivity/Routing

```
[--connect-host|-X]                       | Let the host be reachable on the
                                          | wan/lan interfaces
[--routing|-x]                            | Act as a router (usually used with
                                          | BGP traffic diversion)
```

CPU Affinity

```
[--balancer-affinity|-r] <id>:<id>        | Bind packet distribution threads to
                                          | core ids (-w only)
[--thread-affinity|-g] <id>[:<id>[..]]    | Bind processing threads to core id
[--time-source-affinity|-T] <id>          | Bind time-source thread to core id
```

Traffic Monitoring

```
[--aux-queues|-O] <num>                    | Enable <num> sets of auxiliary
                                           | egress queues for packet sampling
                                           | (traffic analysis/dumping)
[--event-script-dir|-Q] <dir>              | Event scripts directory
```

Advanced Settings

```
[--cluster-id|-c] <id>                     | ZC cluster ID
[--ht-idle-timeout|-I] <usec>              | Flow idle timeout
[--dyn-white-list-idle-timeout|-e] <usec> | Auto-whitelist idle timeout
[Default: 3600000000]
[--gre-decapsulation|-E]                   | Decapsulate GRE traffic
[--redis|-D] <host[:port][@db-id]>         | Redis DB host[:port][@database id]
```

REST Server

```
[--http-address|-G] <address>              | IP address for REST server socket
                                           | binding
[--http-port|-H] <port>                    | HTTP port for REST server
[--https-port|-s] <port>                   | HTTPs port for REST server
[--docs-root|-R] <dir>                     | Docs root directory
```

Logging

```
[--log-path|-l] <path>                     | Log file path
[--stats-log-path|-y] <path>               | Stats log file path
[--debug-level|-t] <level>                 | Trace level
```

Service

```
[--daemon|-d]                              | Run as a daemon
[--pid-path|-p] <path>                     | PID file path
[--user] <sys user>                        | Run with the specified user
```

Version and License

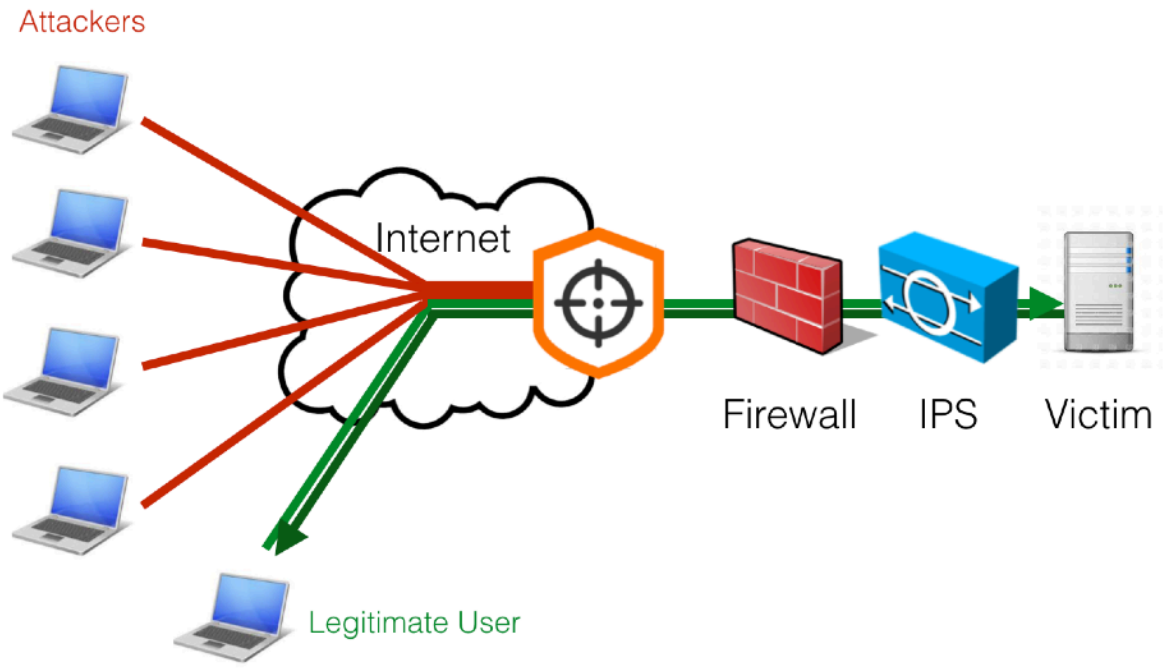
```
[--version|-V]                             | Print version
[--system-id|-Y]                           | Print system identifier
[--check-license|-C]                       | Checks if license is valid
[--check-maintenance|-M]                   | Checks maintenance expiration
```

Working Modes

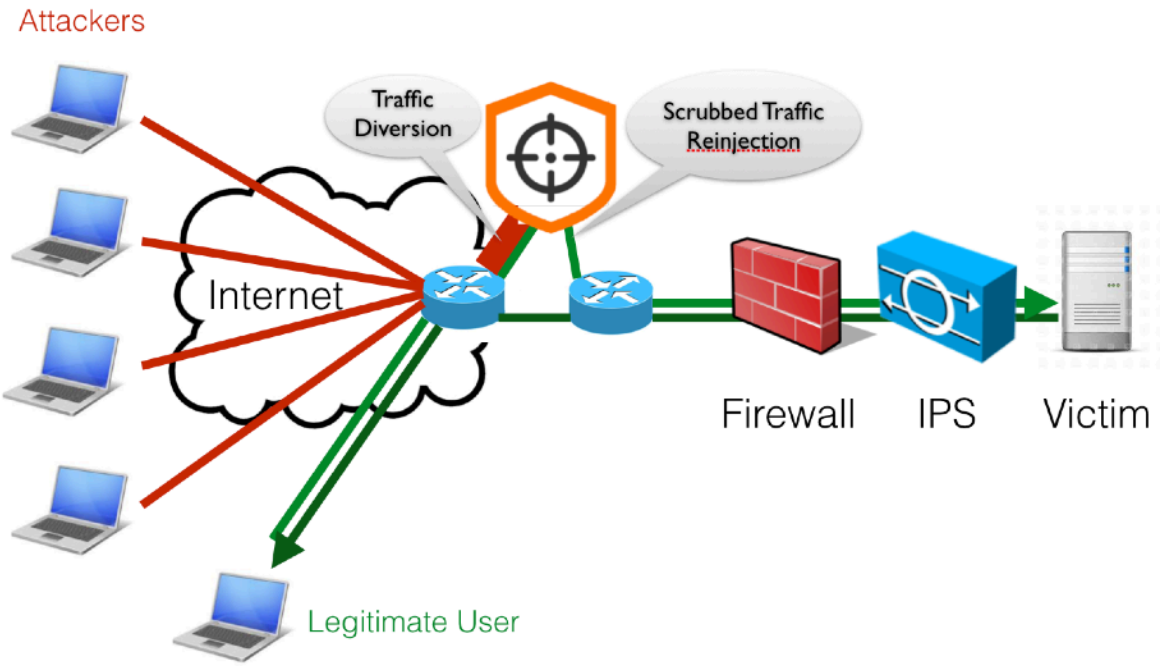
The software can run in transparent bridge or routing mode, in symmetric or asymmetric mode.

Transparent Bridge Mode

When running the software as transparent bridge (default), an interface pair must be specified for traffic bridging. This mode can work both in bidirectional mode, inspecting traffic in both directions:



or in asymmetric mode, specifying the `--asymmetric-routing|-A` option (inbound only), when used for instance with traffic diversion techniques:



Routing Mode

When running the software in routing mode (specifying the `–routing|-x` option), one or two interfaces can be used. Also in this configuration both symmetric and asymmetric mode can be used.
In this mode a routing table must be configured using the API or the CLI tool. In order to learn how to use the API read the *Traffic Enforcement Configuration* section below and the API documentation.

Performance and Tuning

This section covers a few key aspects for running the application at line-rate.

Hardware Configuration

First of all you should figure out what is the actual hardware configuration. Please make sure that:

1. Memory is properly installed.

Please check on ark.intel.com the number of memory channels supported by your CPU, and make sure there is a memory module installed for each channel. If you do not have physical access to the machine, or you just want to double-check everything is properly configured, you can use dmidecode. In the example below the machine has 4 memory channels, and there is a memory module installed for each channel.

```
# dmidecode | grep "Configured Clock Speed"
      Configured Clock Speed: 1600 MHz
      Configured Clock Speed: 1600 MHz
      Configured Clock Speed: 1600 MHz
      Configured Clock Speed: 1600 MHz
```

2. NICs are properly installed.

Please check the cards you are using are installed on PCIe slots with enough lanes and speed for supporting the bandwidth you expect. You can check this from dmesg, or doing some math using the PCIe speed based on version and the number of lanes reported on the motherboard.

```
$ dmesg | grep Speed
[    3.075885] ixgbe 0000:01:00.0: (Speed:5.0GT/s, Width: x8, Encoding Loss:20%)
[    3.314675] ixgbe 0000:01:00.1: (Speed:5.0GT/s, Width: x8, Encoding Loss:20%)
```

In case of NUMA systems (i.e. multiple CPUs), please make sure the card is installed on the right PCIe slot, directly connected to the CPU you are going to use for traffic processing. You can use lstopo for this. In the example below there are 2 NICs, one is installed on NUMA node 0 (eth0 and eth1) and one is installed on NUMA node 1 (eth1 and eth2).

```
# lstopo
Machine (63GB)
  NUMANode L#0 (P#0 31GB)
    Socket L#0 + L3 L#0 (20MB)
      L2 L#0 (256KB) + L1d L#0 (32KB) + L1i L#0 (32KB) + Core L#0
        PU L#0 (P#0)
        PU L#1 (P#16)
      L2 L#1 (256KB) + L1d L#1 (32KB) + L1i L#1 (32KB) + Core L#1
        PU L#2 (P#2)
        PU L#3 (P#18)
      L2 L#2 (256KB) + L1d L#2 (32KB) + L1i L#2 (32KB) + Core L#2
        PU L#4 (P#4)
        PU L#5 (P#20)
      L2 L#3 (256KB) + L1d L#3 (32KB) + L1i L#3 (32KB) + Core L#3
        PU L#6 (P#6)
```

```
      PU L#7 (P#22)
    L2 L#4 (256KB) + L1d L#4 (32KB) + L1i L#4 (32KB) + Core L#4
      PU L#8 (P#8)
      PU L#9 (P#24)
    L2 L#5 (256KB) + L1d L#5 (32KB) + L1i L#5 (32KB) + Core L#5
      PU L#10 (P#10)
      PU L#11 (P#26)
    L2 L#6 (256KB) + L1d L#6 (32KB) + L1i L#6 (32KB) + Core L#6
      PU L#12 (P#12)
      PU L#13 (P#28)
    L2 L#7 (256KB) + L1d L#7 (32KB) + L1i L#7 (32KB) + Core L#7
      PU L#14 (P#14)
      PU L#15 (P#30)
  HostBridge L#0
    PCIBridge
      PCI 8086:154d
        Net L#1 "eth0"
      PCI 8086:154d
        Net L#2 "eth1"
NUMANode L#1 (P#1 31GB)
  Socket L#1 + L3 L#1 (20MB)
    L2 L#8 (256KB) + L1d L#8 (32KB) + L1i L#8 (32KB) + Core L#8
      PU L#16 (P#1)
      PU L#17 (P#17)
    L2 L#9 (256KB) + L1d L#9 (32KB) + L1i L#9 (32KB) + Core L#9
      PU L#18 (P#3)
      PU L#19 (P#19)
    L2 L#10 (256KB) + L1d L#10 (32KB) + L1i L#10 (32KB) + Core L#10
      PU L#20 (P#5)
      PU L#21 (P#21)
    L2 L#11 (256KB) + L1d L#11 (32KB) + L1i L#11 (32KB) + Core L#11
      PU L#22 (P#7)
      PU L#23 (P#23)
    L2 L#12 (256KB) + L1d L#12 (32KB) + L1i L#12 (32KB) + Core L#12
      PU L#24 (P#9)
      PU L#25 (P#25)
    L2 L#13 (256KB) + L1d L#13 (32KB) + L1i L#13 (32KB) + Core L#13
      PU L#26 (P#11)
      PU L#27 (P#27)
    L2 L#14 (256KB) + L1d L#14 (32KB) + L1i L#14 (32KB) + Core L#14
      PU L#28 (P#13)
      PU L#29 (P#29)
    L2 L#15 (256KB) + L1d L#15 (32KB) + L1i L#15 (32KB) + Core L#15
      PU L#30 (P#15)
      PU L#31 (P#31)
  HostBridge L#8
    PCIBridge
      PCI 8086:154d
        Net L#7 "eth2"
      PCI 8086:154d
        Net L#8 "eth3"
```

In case of NUMA system it is important to tell the driver what is the CPU we are going to use for packet processing for each network interface, in order to setup resources in the right place in memory and improve memory locality. In order to do this, please refer to the *Installation - PF_RING* section where we configured the driver using:

```
# echo "RSS=0,0" > /etc/pf_ring/zc/ixgbe/ixgbe.conf
```

In order to specify the CPU affinity it is possible to use the numa_cpu_affinity parameter, passing core IDs for memory allocation, one per interface (4 interfaces in this example):

```
# echo "RSS=0,0,0,0 numa_cpu_affinity=0,0,8,8" > /etc/pf_ring/zc/ixgbe/
ixgbe.conf
```

Where 0 is a core ID on the first CPU, 8 is a core ID on the second CPU.

Please remember to reload PF_RING using the init.d script after changing the configuration.

Load Balancing

Load balancing is important for distributing workload across multiple CPU cores. The application supports traffic distribution using hardware RSS support or using zero-copy software distribution based on ZC.

Hardware Traffic Distribution (RSS)

By default the application expects that hw RSS has been configured, spawning as many processing threads as the number of RSS queues. In the *Installation - PF_RING* section we configured the driver with the "auto" RSS mode (as many RSS queues as the number of CPU cores) using:

```
# echo "RSS=0,0" > /etc/pf_ring/zc/ixgbe/ixgbe.conf
```

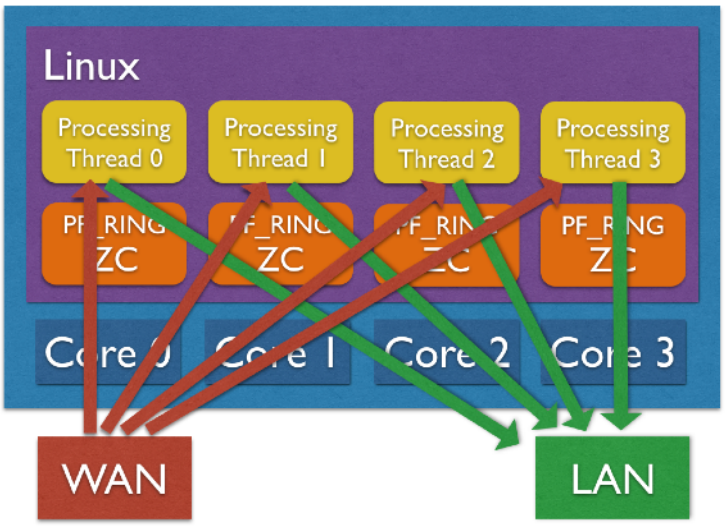
It is also possible to specify the number or queues, passing the number of queues per interface (2 interfaces, 4 queues per interface in this example) to the RSS parameter:

```
# echo "RSS=4,4" > /etc/pf_ring/zc/ixgbe/ixgbe.conf
```

Please remember to reload PF_RING using the init.d script after changing the configuration.

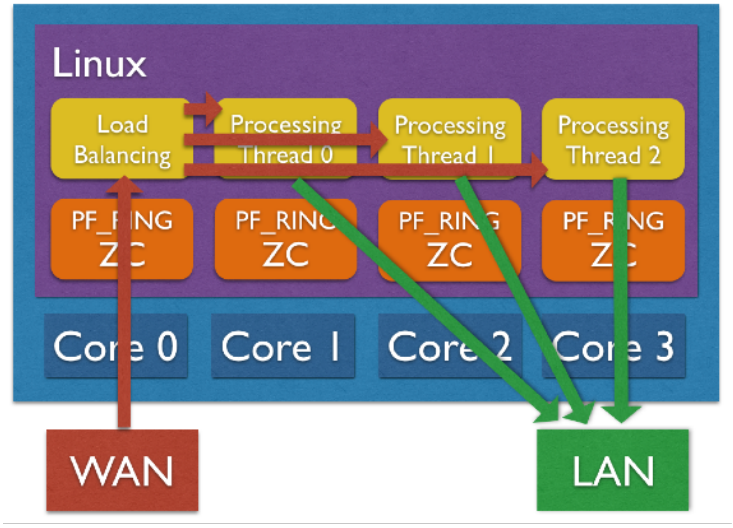
It is possible to check the current number of RSS queues for each interface from the informations reported by PF_RING under /proc:

```
$ cat /proc/net/pf_ring/dev/eth1/info | grep Queues
Max # TX Queues:   4
# Used RX Queues:  4
```



Software Traffic Distribution (ZC)

Another option for distributing traffic across cores is using zero-copy software distribution based on ZC. This mode provides more flexibility in traffic distribution as hash is computed in software and it can be applied to any packet header or payload. Please note RSS is still needed for lock-free zero-copy transmission as depicted in the picture below. In order to enable software distribution the `--sw-distribution|-w` option should be specified into the nscrub configuration file.



CPU Affinity

CPU affinity is very important for equally distributing the load across cores. There are a few options for setting the core affinity for each thread used by the application.

Time-Source Thread

In order to have a precise low-cost time source, a dedicated thread is used. It is possible to bind this thread to a core using the `--time-source-affinity|-T` option.

Processing Threads

Using the load balancing support, both hardware RSS or software distribution, a processing thread is spawned for each RSS queues. It is possible to bind each thread to a core by listing core IDs in the nscrub configuration file using the `--thread-affinity|-g` option (colon-separated list). Example (4 RSS queues):

```
# cat /etc/nscrub/nscrub.conf
--wan-interface=zc:eth1
--lan-interface=zc:eth2
--time-source-affinity=0
--thread-affinity=1:2:3:4
--log-path=/var/tmp/nscrub.log
--pid-path=/var/tmp/nscrub.pid
--daemon
```

Load Balancer Thread (Software Traffic Distribution only)

Using software distribution (`--sw-distribution|-w`) in place of hw RSS, an additional balancer thread is used for computing the packet hash and steer the packet to the right processing thread. It is possible to set the core affinity for this thread using the `--balancer-affinity|-r` option. Example:

```
# cat /etc/nscrub/nscrub.conf
--wan-interface=zc:eth1
--lan-interface=zc:eth2
--time-source-affinity=0
--balancer-affinity=1
--thread-affinity=2:3:4:5
--log-path=/var/tmp/nscrub.log
--pid-path=/var/tmp/nscrub.pid
--daemon
```

Traffic Enforcement Configuration

Once the application is up and running, it's time to configure it for enabling traffic mitigation. This means we need to create virtual scrubbers (objects containing protection policies based on target), each virtual scrubber inspects the traffic matching one, or more, destination subnets. Each virtual scrubber, identified by a target ID, has its own traffic enforcement profiles that can be configured/changed/inspected at runtime using the API.

This sections provides some basic knowledge for configuring the engine using the REST API. Please refer to the API documentation for the full API specifications.

Note that although this section covers the configuration using the REST API, a few command line tools (see Appendix A) are also available to ease the configuration, including:

- *nscrub-cli,* implementing a console with autocompletion with all the functionalities implemented by the REST API.
- *nscrub-add,* a wizard tool for creating new victims with a basic configuration (further customisations are usually needed using *nscrub-cli* or the API).
- *nscrub-export, a* tool for dumping the current configuration for a specific victim.

Default credentials for configuring nscrub:

Username: admin
Password: admin
HTTP port: 8880
HTTPs port: 4443
Socket binding: localhost

Victims definition

Victims can be dynamically added, removed and configured at runtime.

Read active victims using the REST API:

```
# curl -u <user>:<password> http://<host>:<port>/targets?action=list
```

or do the same using the command line tool:

```
$ nscrub-cli
localhost:8880> list targets
```

Add a new victim:

```
# curl -u <user>:<password> http://<host>:<port>/targets?
action=add\&target_id=<victim name>\&subnet=<subnet (CIDR notation)>
```

Each victim is bound to a few profiles: default, black, white, gray. The default profile applies to all unknown sources, while the other profiles apply to the corresponding lists of source IPs (attackers). In essence black, white and gray are just placeholders for defining different traffic enforcement policies based on source IP "colour". The white profile is a

special profile, in fact source IPs recognised as real and legitimate are automatically added to this special list.

It is a common practice to set the "drop all" policy to the black profile:

```
# curl -u <user>:<password> http://<host>:<port>/profile/all/drop?
target_id=<victim name>\&profile=black\&action=enable
```

It is also a common practice to set the "accept all" policy to the white profile:

```
# curl -u <user>:<password> http://<host>:<port>/profile/all/accept?
target_id=<victim name>\&profile=white\&action=enable
```

The gray profile is usually used for applying special policies to "special" IPs. For instance it is a common practice to set the "default" policy to "drop" and then specify more specific policies to let specific traffic types through.

```
# curl -u <user>:<password> http://<host>:<port>/profile/default?
target_id=<victim name>\&profile=gray\&action=update\&value=drop
```

The default profile is where the real traffic enforcement policies go, for checking unknown traffic. For instance it is also a common practice to set the default policy to drop:

```
# curl -u <user>:<password> http://<host>:<port>/profile/default?
target_id=<victim name>\&profile=default\&action=update\&value=drop
```

Accept ICMP:

```
# curl -u <user>:<password> http://<host>:<port>/profile/icmp/accept?
target_id=<victim name>\&profile=default\&action=enable
```

Drop UDP:

```
# curl -u <user>:<password> http://<host>:<port>/profile/udp/drop?
target_id=<victim name>\&profile=default\&action=enable
```

Accept UDP port 53 (DNS):

```
# curl -u <user>:<password> http://<host>:<port>/profile/udp/src/53/accept?
target_id=<victim name>\&profile=default\&action=enable
```

Check TCP traffic:

```
# curl -u <user>:<password> http://<host>:<port>/profile/tcp/syn/check_method?
target_id=<victim name>\&profile=default\&action=update\&value=rfc
```

It is also possible to set a rate limiter (in this example per source) to set a threshold to the traffic rate.

```
# curl -u <user>:<password> http://<host>:<port>/profile/rate?target_id=<victim
name>\&profile={black, white, gray, default}[\&action=update\&value=<pkts/s>]
```

Many more policies are available, please refer to the full API documentation.

Please note all the settings can also be read, omitting the action (and value) parameter.

In order to temporarily disable traffic checks, it is possible to put the system in bypass state, both globally:

```
# curl -u <user>:<password> http://<host>:<port>/bypass?[action={enable,
disable}]
```

or per victim:

```
# curl -u <user>:<password> http://<host>:<port>/profile/bypass?
target_id=<victim name>\&profile=default[\&action={enable, disable}]
```

Traffic Monitoring and Statistics

This section shows the options for having visibility of the traffic under mitigation.

Statistics

The application provides global and per-victim statistics.

Global statistics are available through the logs:

```
# tail -f  /var/tmp/nscrub.log
```

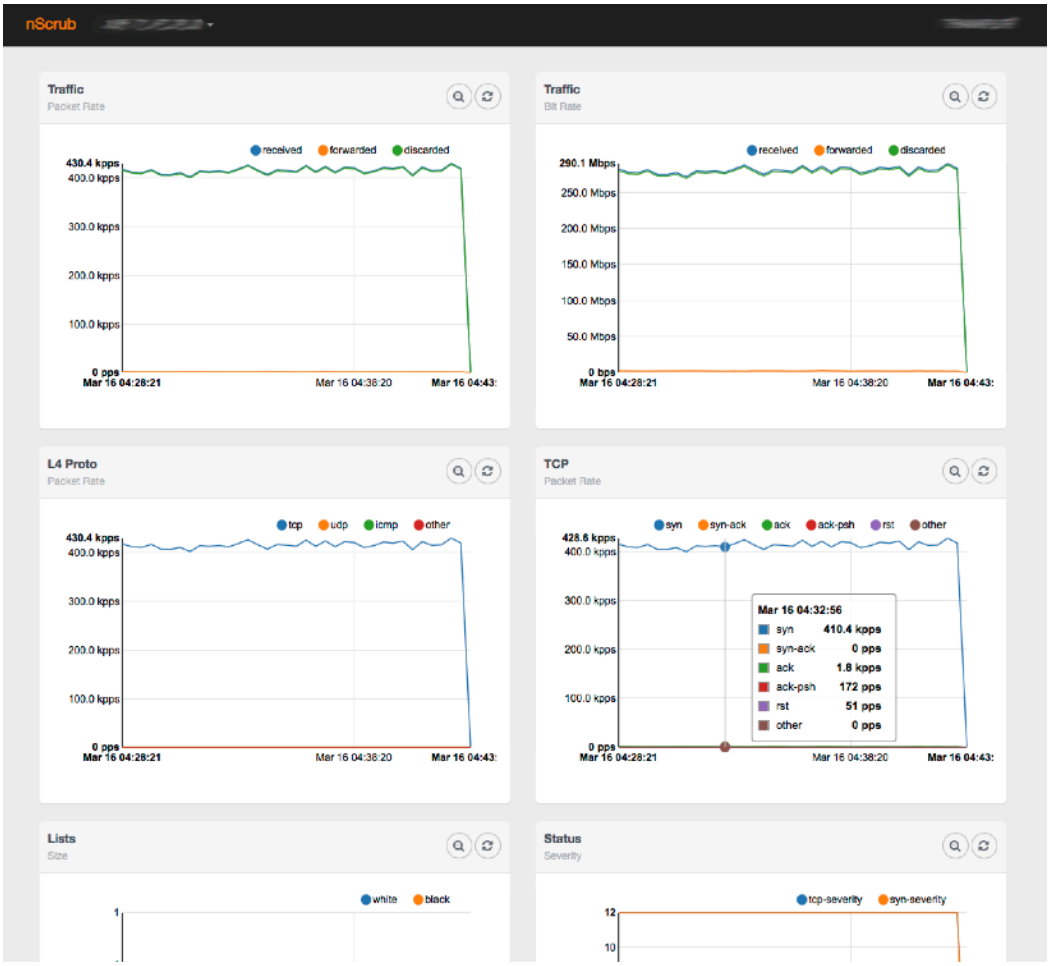
or through the REST API:

```
# curl -u <user>:<password> http://<host>:<port>/stats
```

Per-victim stats are available through the REST API:

```
# curl -u <user>:<password> http://<host>:<port>/targets?
action=stats\&target_id=<target id>
```

Historical statistics from RRDs, both global and per-victim, are also available using a browser connecting to:



```
http://<host>:<port>/monitor.html
```

Traffic Monitoring

The software has the ability to export some traffic to software queues for attaching external applications, for traffic analysis or recording:

In order to enable auxiliary software queues for traffic monitoring, the `--aux-queues|-O` option has to be added to the nScrub configuration file, specifying the number of queues to allocate, which matches the maximum number of concurrent application that will monitor the traffic (queues are single consumer).

```
--aux-queues 1
```

Please note that in order to avoid locking mechanisms, a queue per thread is created (we call it "queue set"), for each consumer application.

Once auxiliary queues have been enabled, it is possible to configure them for exporting sampled (or all) traffic (good, discarded, or all) to external applications through the API:

```
# curl -u <user>:<password> http://<host>:<port>/mirror/<queue set id>/type[?
action=update\&value={forwarded, discarded, all}]
# curl -u <user>:<password> http://<host>:<port>/mirror/<queue set id>/
sampling[?action=update\&value=<sampling rate (0 for no traffic)>]
```

Example:

```
# curl -u <user>:<password> http://<host>:<port>/mirror/0/type?
action=update\&value=all
# curl -u <user>:<password> http://<host>:<port>/mirror/0/sampling?
action=update\&value=100
```

It is possible for instance to use tcpdump for monitoring the traffic running one instance per processing thread, specifying the cluster ID and thread/queue ID in the interface name:

```
# tcpdump -Q in -ni zc:99@0
# tcpdump -Q in -ni zc:99@1
# tcpdump -Q in -ni zc:99@2
# tcpdump -Q in -ni zc:99@3
```

Alternatively, it is possible to aggregate all traffic to a single queue and run a single tcpdump instance:

```
# zbalance_ipc -i zc:99@0,zc:99@1,zc:99@2,zc:99@3 -c 100 -n 1

# tcpdump -Q in -ni zc:100@0
```

In the same way it is possible to analyse the traffic using ntopng:

```
# ntopng -i zc:99@0 -i zc:99@1 -i zc:99@2 -i zc:99@3
```

Appendix A - Command line tools

nscrub-cli is the command line tool for configuring nScrub using autocompletion, instead of using the REST API manually.

You can run nscrub-cli to get access to the nscrub configuration, without any option or passing user/password and host/port if you customised the nscrub configuration.

```
$ nscrub-cli -h
nscrub-cli [-v] [-h] [-a <user>:<password>] [-c <host>:<port>]
```

Run nscrub-cli to get a console, then type "help" to get a full list of supported commands. Please refer to the REST API for a description.

A wizard tool *nscrub-add* is also available for creating a new target with a basic configuration, providing the target ID, subnet and service type.

```
nscrub-add [options] <target id> <subnet> <target type>
```

Example:

```
$ nscrub-add WEBSERVER 192.168.1.1/32 web
```

Please note that further configurations are usually needed to create an optimal configuration based on the specific target, it is possible to use *nscrub-cli* for that.

nscrub-cli also allows you to pass a configuration file containing *nscrub-cli* commands using pipe. Example:

```
$ cat nscrub.conf
targets add TEST 192.168.1.2/32
targets add TEST 192.168.1.3/32
$ cat nscrub.conf | nscrub-cli
```

Please refer to the *nscrub-cli* help for an updated list of commands like in the example below.

```
$ nscrub-cli
localhost:8880> help
status
stats
hostname [NAME]
users
useradd NAME GROUP PASSWORD
usermod NAME GROUP [PASSWORD]
userdel NAME
bypass [enable|disable]
mirror ID type [forwarded|discarded|all]
mirror ID sampling [RATE]
route
route add SUBNET gw IP
route del SUBNET
vlan id SRC-VLAN-ID reforge [DST-VLAN-ID]
list targets
add target ID SUBNET
del target ID SUBNET
purge target ID
```

```
target ID desc [DESCRIPTION]
target ID vlan reforge [DST-VLAN-ID]
target ID type [web|dns|game|isp]
target ID stats
target ID attackers show LISTNAME|dynamic|* white|gray|black|*
target ID attackers add LISTNAME SUBNET white|gray|black
target ID attackers del LISTNAME SUBNET
target ID attackers purge LISTNAME|dynamic
target ID attackers showlists
target ID attackers search LISTNAME|dynamic|* white|gray|black|* SUBNET
target ID attackers dynamic autopurging [enable|disable]
target ID attackers dynamic expiration [SEC]
target ID profile white|gray|black|default bypass [enable|disable]
target ID profile white|gray|black|default default [drop|pass]
target ID profile white|gray|black|default rate src [PPS]
target ID profile white|gray|black|default rate dst [PPS]
target ID profile white|gray|black|default all drop [enable|disable]
target ID profile white|gray|black|default all accept [enable|disable]
target ID profile white|gray|black|default ip
target ID profile white|gray|black|default ip ttl NUM drop [enable|disable]
target ID profile white|gray|black|default ip ttl NUM accept [enable|disable]
target ID profile white|gray|black|default udp
target ID profile white|gray|black|default udp drop [enable|disable]
target ID profile white|gray|black|default udp accept [enable|disable]
target ID profile white|gray|black|default udp src NUM drop [enable|disable]
target ID profile white|gray|black|default udp src NUM accept [enable|disable]
target ID profile white|gray|black|default udp dst NUM drop [enable|disable]
target ID profile white|gray|black|default udp dst NUM accept [enable|disable]
target ID profile white|gray|black|default udp fragment drop [enable|disable]
target ID profile white|gray|black|default udp fragment payload min_len [BYTES]
target ID profile white|gray|black|default udp fragment payload max_len [BYTES]
target ID profile white|gray|black|default udp checksum0 drop [enable|disable]
target ID profile white|gray|black|default udp payload min_len [BYTES]
target ID profile white|gray|black|default udp payload max_len [BYTES]
target ID profile white|gray|black|default udp rate src [PPS]
target ID profile white|gray|black|default udp rate dst [PPS]
target ID profile white|gray|black|default tcp
target ID profile white|gray|black|default tcp drop [enable|disable]
target ID profile white|gray|black|default tcp accept [enable|disable]
target ID profile white|gray|black|default tcp src NUM drop [enable|disable]
target ID profile white|gray|black|default tcp src NUM accept [enable|disable]
target ID profile white|gray|black|default tcp dst NUM drop [enable|disable]
target ID profile white|gray|black|default tcp dst NUM accept [enable|disable]
target ID profile white|gray|black|default tcp syn check_method [rfc|proxy|
bypass]
target ID profile white|gray|black|default tcp syn rfc threshold [PPS]
target ID profile white|gray|black|default tcp syn wl_session_only [enable|
disable]
target ID profile white|gray|black|default tcp syn rate src [PPS]
target ID profile white|gray|black|default tcp syn rate dst [PPS]
target ID profile white|gray|black|default tcp syn auto_blacklist [enable|
disable]
target ID profile white|gray|black|default tcp synack rate src [PPS]
target ID profile white|gray|black|default tcp synack rate dst [PPS]
target ID profile white|gray|black|default tcp synack wl_session [enable|
disable]
target ID profile white|gray|black|default tcp threshold [MBITPS]
target ID profile white|gray|black|default icmp
target ID profile white|gray|black|default icmp drop [enable|disable]
target ID profile white|gray|black|default icmp accept [enable|disable]
target ID profile white|gray|black|default icmp type NUM drop [enable|disable]
target ID profile white|gray|black|default icmp type NUM accept [enable|disable]
target ID profile white|gray|black|default gre
target ID profile white|gray|black|default gre drop [enable|disable]
target ID profile white|gray|black|default gre accept [enable|disable]
target ID profile white|gray|black|default dns
```

```
target ID profile white|gray|black|default dns request check_method [forcetcp|
default]
target ID profile white|gray|black|default dns request rate src [PPS]
target ID profile white|gray|black|default dns request rate transaction_id [PPS]
target ID profile white|gray|black|default dns request threshold [PPS]
target ID profile white|gray|black|default dns request type NUM drop [enable|
disable]
target ID profile white|gray|black|default pattern
target ID profile white|gray|black|default pattern NUM drop [{hex, string},
[payload+]{OFFSET, any},VALUE]
target ID profile white|gray|black|default http request field
target ID profile white|gray|black|default http request field NUM drop
[LABEL,VALUE]
help
quit
```

Appendix B - Utilities and Testing

This section provides a few examples for checking the mitigation and bridging throughput.

SYN Mitigation

In order to test SYN mitigation you can use one of the SYN flood generators available on internet, such as Juno:

```
# wget http://packetstorm.wowhacker.com/DoS/juno.c
# gcc -O2 juno.c -o juno
```

Throughput

In order to test the bridging throughput you can use iperf:

Target machine:

```
# iperf -s --bind 192.168.1.254
```

Client machine:

```
# iperf -c 192.168.1.254
```

Appendix C - Hugepages

This section describes how to enable hugepages into your system, a mandatory step for running nscrub with ZC. Please note that hugepages are already loaded by the PF_RING init.d script, this section aims to describe how hugepages work in order to provide the knowledge needed for configuring and checking the hugepages status.

Linux typically uses memory pages of 4 KBytes, but provides an explicit interface to allocate pages with bigger size called hugepages. It is up to developers/administrators to decide when they have to be used.

Hugepages advantages:

- Large amounts of physical memory can be reserved for memory allocation, that otherwise would fail especially when physically contiguous memory is required.
- Reduced overhead: as the TLB (Translation Lookaside Buffer) contains per-page virtual to physical address mappings, using a large amount of memory with the default page size leads to processing overhead for managing the TLB entries.

The default hugepage size is usually 2 MBytes. The hugepage size can be found in /proc/meminfo:

```
# cat /proc/meminfo | grep Hugepagesize
  Hugepagesize: 2048 kB
```

Hugepages can be dinamically reserved with:

```
# echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

The above pages are allocated by the system without node affinity. If you want to force allocation on a specific NUMA node you have to do:

```
# echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
nr_hugepages
# echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/
nr_hugepages
```

It is possible to change the default hugepages size and reserve large amounts of memory at boot time using the following kernel options:

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

If this commands returns a non-empty string, 2MB pages are supported:

```
# cat /proc/cpuinfo | grep pse
```

If this commands returns a non-empty string, 1GB pages are supported:

```
# cat /proc/cpuinfo | grep pdpe1gb
```

In order to make the hugepages available for use, Linux provides a RAM-based filesystem called "hugetlbfs" that have to be mount'ed with:

```
# mount -t hugetlbfs none /dev/hugepages
```

With no options the default hugepage size is used. To use a different size it is possible to specify the "pagesize=" option. In order to control the maximum amount of memory bound to a mount point it is possible to specify the "size=" option (size is rounded down to the nearest hugepage size). Example:

```
# mount -t hugetlbfs -o pagesize=1G,size=2G none /dev/hugepages
```

It is possible to see what pages are currently in use using the following command:

```
# cat /sys/devices/system/node/node*/meminfo | grep Huge
Node 0 HugePages_Total:  1024
Node 0 HugePages_Free:   1024
Node 0 HugePages_Surp:      0
```

Appendix D - EULA

Licensee's use of this software is conditioned upon acceptance of the terms specified in https://svn.ntop.org/svn/ntop/trunk/legal/LicenseAgreement/Reseller/EULA.txt