Introduction

FastNetMon Community is a software package for detecting DDoS attacks and their subsequent mitigation.

This tool will help you to maximize the availability of your network and eliminate any outages due to DoS / DDoS attacks.

Installation

Software environment requirements

The project is completely free and open source.

It's available for the following platforms:

- Linux
- FreeBSD
- MacOS

I would like to point out that the recommended platform is Linux and we provide official binary packages only for Linux.

Binary packages are available for the following Linux distributions for 64-bit platforms only (amd64 / x86_64 and ARM64):

Ubuntu LTS Debian RHEL based distros (CentOS, AlmaLinux, RockyLinux)

Installation from packages is the preferred installation option. Please use our official installer tool for it: https://fastnetmon.com/install/

You may try compiling it on your own and you can find instructions for it here: https://github.com/pavel-odintsov/fastnetmon/blob/master/.circleci/config.yml#L223C1-L224C1

Hardware Requirements

Product can be installed both in a virtual environment and on physical servers.

The sFlow, NetFlow, Mirror AF_PACKET modes do not impose any special hardware requirements and they work fine inside Docker environments, inside Linux containers (Docker) and any virtualisation system.

Mirror XDP (Linux) Mirror Netmap (FreeBSD only) will work well only on physical server or virtual machine. For best performance we recommend running them on physical server or inside virtual machine with network card plugged via PCI-E Passthrough. These modes require significant amount of CPU resources and you can use rule of thumb and reserve 4 CPU cores per 10G interface.

Initial Configuration

After installing the project from a binary package, you need to perform a basic configuration, which includes two steps:

- Choosing a plugin for traffic capture (for a list of available plugins, please see the "Traffic Capture Subsystem" section)
- Adding own networks (required to identify our traffic from the stream and determine its direction)

We leave the configuration of the intrusion detection subsystem for later, because it requires more detailed description.

To select a traffic capture plugin, open the /etc/fastnetmon.conf file with a text editor and change the value from "off" to "on" for the option you like.

To add your own subnets, you need to add them to the /etc/networks_list file in CIDR (10.0.0.0/24 or cafe::/48) format, one network per line.

In addition, be sure to disable the ban using the enable_ban parameter by setting its value to "off" to prevent any destructive actions.

After completing the basic configuration, you can launch the product using the method familiar to your distribution.

To apply configuration changes you need to restart daemon: sudo systemctl restart fastnetmon

In case of any problems, please check log file, /var/log/fastnetmon.log.

After that, you can run the fastnetmon_client client monitoring tool and verify that the product is seeing the traffic.

General project architecture

The product is a multi-threaded application written in C++ and running in user space.

From an architectural point of view, the project consists of the following parts:

- Traffic capture subsystem
- Analysis Subsystem
- Visualization subsystem
- Action Subsystem

Traffic capture subsystem

Since the main working material for the program is traffic, we need to capture it first.

The product supports traffic capturing in the following ways:

- NetFlow v5, v9
- IPFIX
- sFlow v5
- From mirror ports (mirror, SPAN)

A general comparison table for all traffic capture modes can be found here: https://fastnetmon.com/docs/capture_backends/

In some cases, it may require disabling traffic processing in one direction or another. This can be done using the process_incoming_traffic and process_outgoing_traffic configuration options.

The flow tracking mode in which information about the number of flows per second is enabled by default. It can be controlled using the flag: enable_connection_tracking

This mode significantly increases the consumption of processor resources and reduces the amount of traffic that can be processed by the product.

Capture mode from mirror ports

In this mode, traffic is captured from the mirror ports of the switch, router or other device.

All information about the transmitted traffic is captured (including packet headers and payload).

This mode is the most resource-intensive, since each packet is analyzed and requires a fast CPU with a large number of logical cores.

Three options for capturing from mirror ports are supported, their names in the configuration file are as follows:

- pcap
- Mirror AF PACKET
- Mirror AF XDP
- Mirror Netmap

Their sharing is prohibited. Using the interfaces parameter, you can specify one or more network interfaces that will be monitored using the selected mode. Here you can also specify multiple interfaces separated by commas, all capture modules support this feature (except pcap).

The pcap capture mode is extremely slow and cannot be used at speeds over 30-50 megabits. It consumes a very large amount of CPU resources and loses packets in case of high load. This mode is not intended for industrial use in production. Its feature is that the interface is not disconnected from the operating system and can be used by other applications.

The mirror_netmap mode is implemented using the Netmap driver which is supported natively only on FreeBSD platforms offers performance comparable to AF XDP on Linux.

In the case of using the mirror and mirror_netmap modes, it is possible to use sampled port capture modes (when not all traffic is duplicated, but a certain part of it, for example, 1/256). In this case, the program needs to inform the program of the sampling frequency selected on the side of the network device using the following parameters: netmap_sampling_ratio (for mirror_netmap) and pfring_sampling_ratio for the mirror mode.

Capture mode with NetFlow

In this mode, we receive telemetry information about traffic. By default, it is received on port 2055 of the UDP protocol on all interfaces. The mode is enabled by the netflow = on parameter.

It is worth noting that this mode does not provide such a high attack detection rate as sFlow and mirror, since by its nature this protocol contains a timeout after which traffic data is transmitted to the collector.

From the second device that generates NetFlow, you must set the minimum possible values for active flow timeout and inactive flow timeout that do not cause performance degradation (consult your vendor's documentation).

The delay in detecting an attack in the case of NetFlow/IPFIX is increased by the greater of these two values (active flow timeout and inactive flow timeout)

Note that NetFlow / IPFIX can also be sampled. For NetFlow 5, 9 and IPFIX we support retrieving sample rate information directly from traffic telemetry. In some cases this logic may not work and you can specify the sample rate in the configuration file using the netflow_sampling_ratio parameter.

In some cases, you may need to change the port where NetFlow is received, this can be done by editing the netflow_port variable. You can also specify multiple ports, separated by commas, to run multiple collectors (useful when NetFlow collects information from different devices on different ports).

To set a specific host on which FastNetMon should receive traffic, you can use the netflow_host variable, in which you can explicitly specify the IP on which traffic should be listened to, or use 0.0.0.0 to listen on all interfaces.

Capture mode with sFlow

This mode is enabled by the parameter sflow = on.

In this mode, the network device performs traffic sampling, without any aggregation.

When choosing a traffic sampling rate you need to consider total bandwidth on devices and select sampling rate accordingly.

Bandwidth	Sampling rate
100 Mbit	500
1 Gbit	1000
10 Gbit	2000
40 Gbit	4000

Bandwidth	Sampling rate
100 Gbit	10000

This mode is very accurate (at a reasonable sampling rate) along with the mirror, provides a high speed of attack detection.

Analysis Subsystem

Detector Configuration

The enable_ban parameter is used to globally enable/disable any reaction of the product to attacks.

For each node, it is possible to specify which traffic parameters can be used to detect an attack:

- ban_for_pps - block if the specified number of pps to / from the host is exceeded
- ban_for_bandwidth - block if the specified number of megabits / second to / from the specified host is exceeded
- ban_for_flows - blocking if the specified number of flows/second to/from the host is exceeded

Each of the modes can be turned on or off depending on your needs. For each of these traffic parameters, you can specify a specific value, after which an attack will be detected:

- threshold_pps
- threshold_mbps
- threshold_flows

In addition, if you need to completely disable attack detection for a given host, then a white list of networks can be used for this. It is located in the /etc/networks_whitelist file and is in CIDR format, one network per line.

According to the list of these networks, if an attack is detected, a long prefix match is performed, and if the host on which the attack was detected is in the white list, no action is taken.

But be careful with this option. It is better to use an elevated threshold for a given host or network then disabling detection completely.

When an attack is detected, host is blocked for a specified time, it can be set through the ban_time parameter.

After this time, it will be blocked if the attack has stopped. It is possible to achieve an unban even if the attack is still in progress, this can be achieved using the unban_only_if_attack_finished configuration parameter. When checking the attack for activity, the speed is checked against the powder values and if they are still exceeded, the unlock is not carried out and will be unbanned on the next attempt.

The unlock thread runs by default every 60 seconds. But if you set the lock time to less than 60 seconds, then the unlock thread runs every lock_time/2 seconds.

Host groups mechanism

If you want to set different traffic thresholds for different networks, then you can use the functionality of so called hostgroups.

Using the following syntax, you can declare a new hostgroup named my_hosts:

```
hostgroup = my_hosts:10.10.10.221/32,192.168.1.0/24
```

I draw your attention to the fact that the prefixes that you list here must be explicitly listed in the /etc/networks_list file, otherwise nothing will work

After creating a group, you can specify your own threshold settings for it. Different from those set globally with the only difference that before the variable name the node group name is added:

```
my_hosts_enable_ban = no
my_hosts_ban_for_pps = no
my_hosts_ban_for_bandwidth = no
my_hosts_ban_for_flows = no
my_hosts_threshold_pps = 20000
my_hosts_threshold_mbps = 1000
my_hosts_threshold_flows = 3500
```

Logic for traffic speed calculations

For each host from networks listed in /etc/networks_list we allocated multiple counters (packets per second, bits per second, flows per second) in incoming and outgoing directions The speed is calculated every second for all hosts in your network for all mentioned traffic types.

After finishing speed calculation FastNetMon checks all thresholds and blocks host which exceeds threshold.

Visualization subsystem

This subsystem includes all plug-ins that provide visualization of traffic in one form or another.

Currently supported:

- Command Line Interface fastnetmon_client
- API management tool: fastnetmon_api_client
- Graphite
- InfluxDB

FastNetMon client

FastNetMon client displays 7 (the specific value is set using the max_ips_in_list parameter) traffic consumers in the incoming and outgoing direction.

Sorting is carried out by default by the number of packets, this can be changed using the sort_parameter configuration parameter (it can take the values packets, bytes, flows).

In addition to information about incoming and outgoing traffic, a number of fields are displayed:

- Internal traffic - traffic between our nodes
- Other traffic - traffic that we could not identify as belonging to our networks
- Total amount of IPv6 packets - total number of fixed IPv6 packets
- Total amount of not processed packets - the total number of packets that, for one reason or another, were not recognized correctly (arp and other service traffic that does not use the IP protocol)

Graphite

If this plugin is activated with the graphite = on parameter, each time the function for recalculating the traffic speed is called, all nodes with off-road traffic speed will be uploaded to Graphite.

In addition, if the enable_subnet_counters parameter was enabled in the configuration, then Graphite will also send information about the traffic of all subnets specified in the /etc/networks_list file.

You can add your own prefix to all field names that are sent to Graphite using the graphite_prefix configuration option.

You can set the port and host (in the form of an IP address only) using the corresponding graphite_host and graphite_port parameters.

To set up a Graphite system, you can use this guide:
https://fastnetmon.com/docs/graphite_integration/

InfluxDB

This database has support for the Graphite telemetry protocol, so InfluxDB support is configured identically to Graphite protocol support.

A complete InfluxDB system configuration instruction can be found here:
https://fastnetmon.com/docs/influxdb_integration/

Action subsystem

It starts work when attack was detected.

It includes several components in itself:

- Notify script
- Attack sample collection
- Subsystem of logging
- Redis
- BGP Unicast announcement

Notify script

Using the notify_script_path configuration parameter, you can set the absolute path to the script that will be called in the following cases:

- Blocking a node after committing a "ban" attack
- Unbanning a node after the end of the "unban" attack

In the case of calling the ban and details actions, we pass a lot of useful information about the attack by passing on stdin to the script. I would like to draw your attention to the fact that due to the peculiarities of the Linux OS family, you must accept this information in the script, otherwise the entire program will end with an error.

The following command line parameters are passed to this script:

- IP of host which is under attack (incoming attack) or source of attack (outgoing attack)
- Attack direction: incoming or outgoing
- Attack bandwidth in packets per second
- Attack action: ban or unban

An example of a notify script can be found here: https://raw.githubusercontent.com/pavel-odintsov/fastnetmon/master/src/notify_about_attack.sh

Logging subsystem

In some cases, it is convenient to send all messages added by the product to log files to a remote or local syslog server.

Activation of logging to the local syslog can be achieved with the logging:local_syslog_logging parameter.

If you want to send log files to a remote server, you need to activate the logging:remote_syslog_logging parameter and specify the address and port of the log server via the logging:remote_syslog_server, logging:remote_syslog_port parameters.

Collecting attack fingerprints

If an attack is detected in the /var/log/fastnetmon_attacks directory, a file with a name in the format "IP_28_07_15_10:08:17" will be created, in which information about the attack strength, protocols, its name will be saved, and a packet dump and a flow dump will be saved (if flow tracking is enabled).

In addition to this mode, it is possible to completely capture the attack traffic and then save it in pcap format, this can be achieved with the collect_attack_pcap_dumps parameter. In this case, next to the file from the previous paragraph, a file with the same name, but with the .pcap extension, will be created, which can be read by Wireshark and Tcpdump utilization in order to analyze the attack in detail.

Redis

This action is enabled by the redis_enabled parameter, and the redis_host and redis_port parameters set the IP address of the host running Redis. Since all keys are stored in one place, if you run multiple application instances, you may need to separate their data within the Redis database, for this you can use the redix_prefix key.

Redis saves information about an attack as soon as it detected, it is stored in a key called IP_information. If the flow tracking mode is enabled, then all information about all flows relative to the attacked host will be placed in the IP_flow_dump key.

After collecting the attack fingerprint, this fingerprint will be stored at IP_packets_dump.

Please note that the value of the _information key is stored in JSON format, which is convenient for processing from scripts or external applications. All other data is stored in plain text format.

BGP

In this option, we can raise the given announcement when an attack is detected on our host and lower it when the attack has stopped. Usually this option is called RTBH, BGP Blackhole.

To use this option, we need an external BGP daemon, which can be configured according to the instructions: https://fastnetmon.com/docs/exabgp_integration/, after installation you must establish a BGP session between it and your network equipment.

After the daemon is configured and running, we need to enable BGP support directly in the program. This is done by activating the exabgp = on flag, you also need to specify the path to the socket API through the exabgp_command_pipe parameter.

In addition, we need to specify the next hop (this information should be given to you by a network engineer), which will be written in announcements through the parameter: exabgp_next_hop.

In addition, we can BGP Community, which will be added to the announcement, this can be done using the exabgp_community parameter, it looks like “65001:666”, if you need to install several communities, then use this syntax: “[65001:666 65001:777]”. Square brackets are required

We can announce both the host itself, on which the attack was recorded (exabgp_announce_host = on), and the network to which it belongs (exabgp_announce_whole_subnet = on).