Suricata IDPS and Nftables: The Mixed Mode

Giuseppe Longo

Stamus Networks

2016 June 27

Suricata: capture modes

IDS

- PCAP: multi OS capture
- AF_PACKET: Linux high performance on vanilla kernel
- NFLOG: Netfilter on Linux

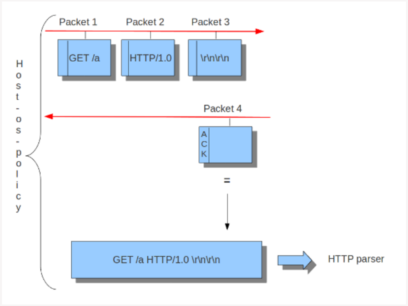
IPS

- NFQUEUE: Netfilter on Linux
- IPFW: Divert socket on FreeBSD
- AF_PACKET: Level 2 software bridge

Suricata: IDS

IDS behavior

Suricata receives traffic in chunks.



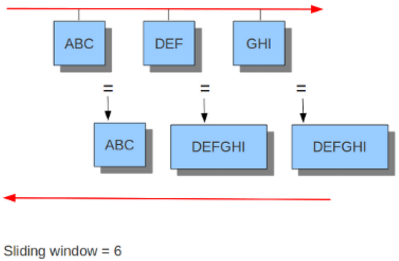
Once the ACK is sent, the chunks are reassembled, and sent to detect engine to inspect it.

Suricata: IPS

IPS behavior

- It inspects packets immediately before sending them to the receiver
- Packets are inspected using the sliding window concept
    - It inspects data as they come in until the tcp connection is closed

Suricata: IPS



Sliding window concept

- Suricata gets the first chunk and inspect it
- Then gets the second chunk, put it together with the first, and inspect it
- At the end, gets the third chunk, cut off the first one, put together second chunk with the third, and inspect it

Suricata: IPS

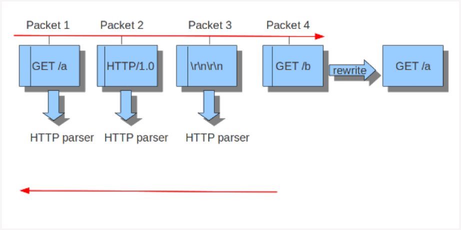
Inline mode

Normally, we analyse data once we know they have been received by the receiver, in term of TCP this means after it has been ACKed.
In IPS it does not work like this, because the data have reached the host that we protect.

Suricata: IPS

Stream in IPS

In inline mode, data is analysed before they have been ACKed.
When Suricata receives a packet, it triggers the reassembly process itself.
If the detection engine decides a drop is required, the packet containing the data itself can be dropped, not just the ACK.



As a consequence of inline mode, Suricata can drop or modify packets if stream reassembly requires it.

Suricata: administrative side

Signatures

On the administrative side, we must have signatures with a proper action in our ruleset.
An action is a property of the signature which determines what will happen when a signature matches the incoming, or outcoming, data.

Suricata: actions

Actions in IDS mode

- Pass
  - Suricata stops scanning the packet and skips to the end of all rules (only for this packet)
- Alert
  - Suricata fires up an alert for the packet matched by a signature

Suricata: actions

Actions in IPS mode

- Drop
  - If a signature containing a drop action matches a packet, this is discarded immediately and won't be sent any further
  - The receiver doesn't receive a message, risulting in a time-out connection
  - All subsequent packets of a flow are dropped
  - Suricata generates an alert for this packet
  - This only concerns the IPS mode

Suricata: actions

Actions in IPS mode

- Reject
  - This is an active rejection of the packet, both receiver and sender receive a reject packet
  - If the packet concerns TCP, it will be a reset-packet, otherwise it will be an ICMP-error packet for all other protocols
  - Suricata generates an alert too
  - In IPS mode, the packet will be dropped as in the drop action
  - Reject in IDS mode is called IDPS

Suricata: NFQUEUE

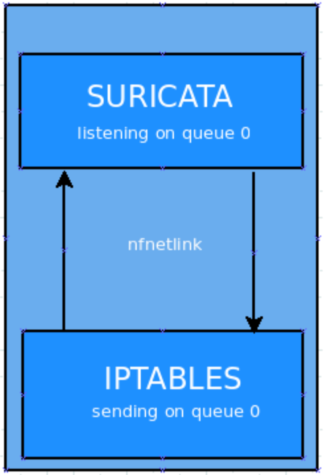
NFQUEUE

- It is used in Suricata to work in IPS mode, performing actions on the packet like DROP or ACCEPT.
- With NFQUEUE we are able to delegate the verdict on the packet to a userspace software
- The Linux kernel will ask a userspace software connected to a queue for a decision

Netfilter's rules

- nft add filter forward queue num 0
- iptables -A FORWARD -j NFQUEUE –queue-num 0

Suricata: NFQUEUE



Suricata and NFQUEUE communication

- Incoming packet matched by a rule is sent to Suricata through nfnetlink
- Suricata receives the packet and issues a verdict depending on our ruleset
- The packet is either transmitted or rejected by kernel

Suricata: NFQUEUE

NFQUEUE rule

- queue-num
  - queue number
- queue-balance
  - packet is queued by the same rules to multiple queues which are load balanced
- queue-bypass
  - packet is accepted when no software is listening to the queue
- fail-open
  - packet is accepted when queue is full
- batching verdict
  - verdict is sent to all packets

Suricata: NFQUEUE

NFQUEUE considerations

- Number of packets on a single queue is limited due to the nature of netlink communication
- Batching verdict can help but without an efficient improvement
- Starting Suricata with multiple queue could improve performance

Suricata: NFLOG

NFLOG

- It is used in Suricata to work in IDS mode, NFLOG is for LOGging
- Similar to NFQUEUE but it only sends a copy of a packet without issuing a verdict
- The communication between NFLOG and userspace software is made through netlink

Netfilter's rule

- nft add rule filter input ip log group 10
- iptables -A INPUT -j NFLOG –nflog-group 10

Group exception

Group 0 it's used by kernel

Suricata: NFLOG

NFLOG rule

- nflog-group
  - number of the netlink multicast group
- nflog-range <N>
  - number of bytes up to which the packet is copied
- nflog-threshold
  - if a packet is matched by a rule, and already N packets are in the queue, the queue is flushed to userspace
- nflog-prefix
  - string associated with every packet logged

Suricata: mixed mode

What is the mixed mode?

- It’s a feature that permits to get the traffic from different sources, giving us the possibility to choice different capture modes, as NFQUEUE and NFLOG, and mix the IPS and IDS capabilities
- The key point of mixed mode is the fact you decide on a per packet basis if handle it as IDS or IPS

Suricata: mixed mode

Motivation

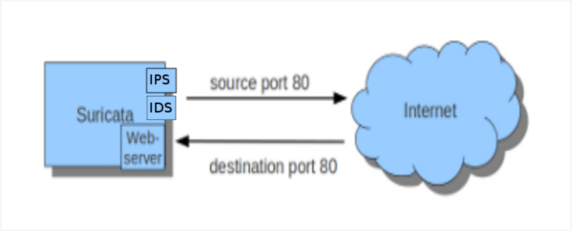
This mode gives us two advantages:

- Having a mixed environment
  - We may want to block some traffic, and inspect some
- Technical simplification
  - We could have an IPS/IDS system, as mixed mode, running many suricata instances with different configuration files

Mixed mode: usage

Scenario

- Web server on 80: can't block traffic
- Rest of traffic is less sensitive



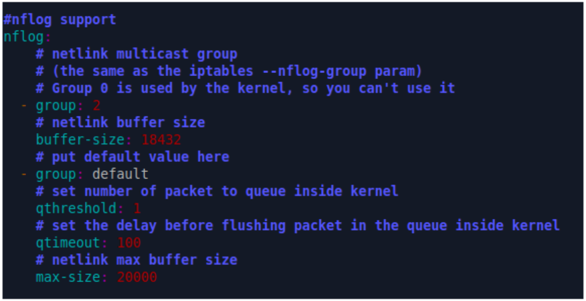
Mixed mode: usage

Netfilter ruleset

- We want to be sure not to cut off a webserver, but we want to inspect port 80
- nftables
  - nft add rule filter forward tcp dport not 80 queue num 0
  - nft add rule filter forward tcp dport 80 log group 2
- iptables
  - iptables -A FORWARD -p tcp ! –dport 80 -j NFQUEUE
  - iptables -A FORWARD -p tcp –dport 80 -j NFLOG –nflog-group 2

Mixed mode: usage

Suricata configuration



```
#nflog support
nflog:
    # netlink multicast group
    # (the same as the iptables --nflog-group param)
    # Group 0 is used by the kernel, so you can't use it
  - group: 2
    # netlink buffer size
    buffer-size: 18432
    # put default value here
  - group: default
    # set number of packet to queue inside kernel
    qthreshold: 1
    # set the delay before flushing packet in the queue inside kernel
    qtimeout: 100
    # netlink max buffer size
    max-size: 20000
```

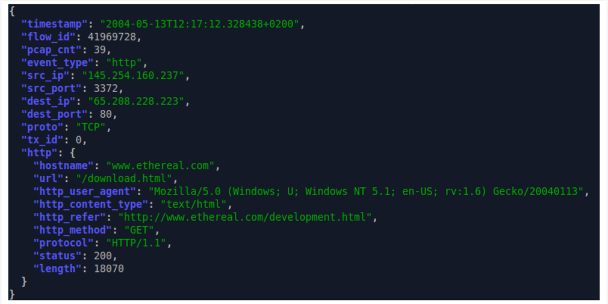
Suricata in mixed mode

suricata -c suricata.yaml -q 0 –nflog -v

Mixed mode: usage

Scenario 2

- This time we want to send all traffic of a suspicious IP from IDS to IPS
- Let's suppose that we notice a suspiscious IP in the eve log file and we want to block it



```
{
  "timestamp": "2004-05-13T12:17:12.328438+0200",
  "flow_id": 41969728,
  "pcap_cnt": 39,
  "event_type": "http",
  "src_ip": "145.254.160.237",
  "src_port": 3372,
  "dest_ip": "65.208.228.223",
  "dest_port": 80,
  "proto": "TCP",
  "tx_id": 0,
  "http": {
    "hostname": "www.ethereal.com",
    "url": "/download.html",
    "http_user_agent": "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113",
    "http_content_type": "text/html",
    "http_refer": "http://www.ethereal.com/development.html",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 18070
  }
}
```

Mixed mode: usage

Solution

- We should add a rule to block the incoming traffic from this IP:
  - nft add rule filter input ip saddr 145.254.160.237
- This solution is not very performing because if we want to block another IP address we need to add another identical rule
  - rules duplication

Mixed mode: usage

Solution improvement

Build a set containing all suspiscious IPs and block all incoming traffic from them.

nftables way

- nft add set filter suspisciousips {type ipv4_addr }
- nft add element filter suspisciousips {145.254.160.137}
- nft add rule filter input ip saddr @suspisciousips queue 0

iptables way

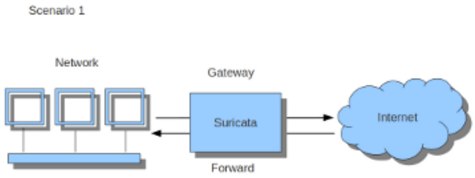
- ipset create suspisciousips
- ipset add suspisciousips 145.254.160.237
- iptables -A FORWARD -m set –set suspisciousips -j NFQUEUE –queue-num 0

STAMUS NETWORKS

Mixed mode: ninja usage

Scenario

We are using Suricata on a gateway that inspects all incoming traffic, and in particular we want to block all SSH connections from fake SSH agents.



Solution

- Suricata detects an SSH connection and log it to EVE log file
- Add the suspiscious IP to the set

Mixed mode: ninja usage

Deny On Monitoring

- Written by Eric Leblond
- Implements a solution similar fail2ban
- It parses the Suricata EVE log file searching for SSH events
- if the client version is suspiscious, it adds the host to a blacklist by using nftables or ipset
  - suspiscious: client version != libssh

Consequence

Suricata will act as IPS on incoming connection from the suspiscious IPs detected by DOM

Rulesets

Netfilter ruleset (nftables)

- nft add set filter suspisciousips {type ipv4_addr}
- nft add rule filter input ip saddr @suspisciousips queue 0
- nft add rule filter input log group 2

Netfilter ruleset (iptables)

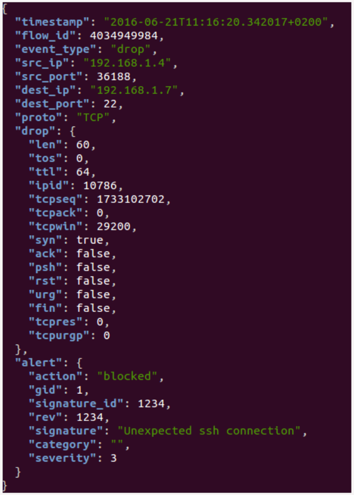
- iptables -A INPUT -m set –set suspisciousips -j NFQUEUE –queue-num 0
- iptables -A INPUT -j NFLOG –nflog-group 2

Suricata ruleset

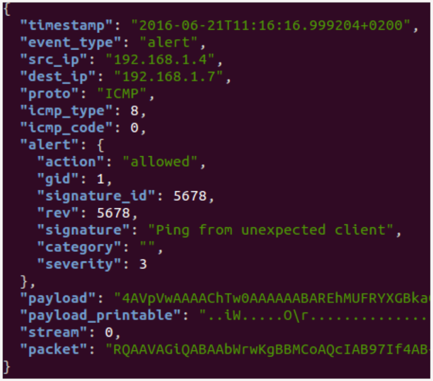
- drop tcp 192.168.1.4 any -> $SSH_SERVER any (msg:"Unexpected ssh connection"; sid:1234; rev:1234;)
- alert icmp 192.168.1.4 any -> $SSH_SERVER any (msg:"Ping from unexpected client"; sid:5678; rev:5678;)

Results

Log examples



```
{
  "timestamp": "2016-06-21T11:16:20.342017+0200",
  "flow_id": 4034949984,
  "event_type": "drop",
  "src_ip": "192.168.1.4",
  "src_port": 36188,
  "dest_ip": "192.168.1.7",
  "dest_port": 22,
  "proto": "TCP",
  "drop": {
    "len": 60,
    "tos": 0,
    "ttl": 64,
    "ipid": 10786,
    "tcpseq": 1733102702,
    "tcpack": 0,
    "tcpwin": 29200,
    "syn": true,
    "ack": false,
    "psh": false,
    "rst": false,
    "urg": false,
    "fin": false,
    "tcpres": 0,
    "tcpurgp": 0
  },
  "alert": {
    "action": "blocked",
    "gid": 1,
    "signature_id": 1234,
    "rev": 1234,
    "signature": "Unexpected ssh connection",
    "category": "",
    "severity": 3
  }
}
```



```
{
  "timestamp": "2016-06-21T11:16:16.999204+0200",
  "event_type": "alert",
  "src_ip": "192.168.1.4",
  "dest_ip": "192.168.1.7",
  "proto": "ICMP",
  "icmp_type": 8,
  "icmp_code": 0,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 5678,
    "rev": 5678,
    "signature": "Ping from unexpected client",
    "category": "",
    "severity": 3
  },
  "payload": "4AVpVwAAAAChTw0AAAAAABAREhMUFRYXGBka
  "payload_printable": "..iW.....O\r.............
  "stream": 0,
  "packet": "RQAAVAGiQABAAbWrwKgBBMCoAQcIAB97If4AB
}
```

Question ?

Mixed mode

- Code not merged yet
- It still requres some testing
- Feedback is appreciated

More information

- Suricata: `http://www.suricata-ids.org/`
- Netfilter: `http://www.netfilter.org/`
- Stamus Networks: `https://www.stamus-networks.com/`

Contact me

- Mail: glongo@stamus-networks.com
- Twitter: @theglongo
- `https://www.stamus-networks.com`