Suricata IDPS and Nftables: The Mixed Mode

Giuseppe Longo

Stamus Networks

Jul 5, 2016

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMUS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

Netfilter: Intro

Netfilter

It's a framework, developed by Netfilter Organization, inside the Linux kernel that enables packet filtering, network address translation, and other packet mangling.

STAMVS NETWORKS

nftables

What's new?

- New filtering system
  - Replace {ip,ip6,arp,ebt}tables
  - New userspace tools
  - Compatibility layers
- A new language
  - Based on a grammar
  - Accessible from a library
- Netlink based communication
  - Atomic modification
  - Notification system

STAMVS NETWORKS

nft: userspace tool

New features

- Tables and chains
- Expressions
- Rules
- Sets and maps
- Dictionaries
- Contenations
- Scripting

STAMVS NETWORKS

nft: tables

Tables

- Container of chains with no specific semantic
- No predefined table configuration anymore
- Need to add a table at least

Adding tables

nft add table [<family>] <name

Examples

- nft add table ip foo
- nft add table foo
- nft add table ip6 bar

STAMUS NETWORKS

nft: chains

Chains

- No predefined chains
- Need to register base chains

Adding chains

- nft add chain [<family>] <table-name> <chain-name> { type <type> hook <hook> priority <value> policy <policy> }

Example

- nft add chain ip foo bar { type filter hook input priority 0 policy drop; }

STAMVS NETWORKS

nft: expressions

Comparison: eq, neq, gt, gte, lt, lte

- nft add rule ip foo bar tcp dport != 80

Range

- nft add rule ip foo bar tcp dport 1-1024
- nft add rule ip foo bar meta skuid 1000-1100

Prefixes

- nft add rule ip foo bar ip daddr 192.168.10.0/24
- nft add rule ip foo bar meta mark 0xffffff00/24

Flags

- nft add rule ip foo bar ct state new, established

nft: expressions (2)

Bitwise + Comparison

- nft add rule ip foo bar ct mark and 0xffff == 0x123

Set value

- nft add rule ip foo bar ct mark set 10
- nft add rule ip foo bar ct mark set meta mark

STAMVS NETWORKS

nft: rules

Counters are optional (unlike iptables)

- nft add rule ip foo bar counter

Several actions in one rule

- nft add rule ip foo bar ct state invalid log prefix "invalid: " drop

nft: sets

Sets

- Built-in generic set infrastructure that allows you to use any supported selector to build sets
- This infrastructure makes possible the representation of dictionaries and maps
- The set elements are internally represented using performance data structures such as hashtables and red-black trees

STAMVS NETWORKS

nft: set (2)

Anonymous set

- Bound to a rule, if the rule is removed, that set is released too
- They have no specific name, the kernel internally allocates an identifier
- They cannot be updated. So you cannot add and delete elements from it once it is bound to a rule

The following example shows how to create a simple set

- nft add rule ip foo bar tcp dport {22, 23} counter

This rule catches all traffic going on TCP ports 22 and 23, in case of matching the counters are updated

STAMVS NETWORKS

nft: set (3)

Named set

You can created the named sets with the following command

- nft add set ip foo whitelist { type ipv4_addr }
  - whitelist is the name of the set in this case
  - type option indicates the data type that this set stores (IPv4 addresses in this case)
  - current maximum name length is 16 characters

Fills the set

- nft add element ip foo whitelist { 192.168.0.1, 192.168.0.10 }

You can use it from the rule:

- nft add rule ip foo bar ip daddr @whitelist counter accept

The content of the set can be dynamically updated

STAMUS NETWORKS

nft: sets (4)

Supported data types

- ipv4_addr: IPv4 address
- ipv6_addr: IPv6 address
- ether_addr: Ethernet address
- inet_proto: Inet protocol type
- inet_service: Internet service (tcp port for example)
- mark: Mark type

STAMVS NETWORKS

nft: maps

Maps

- Can be used to look up for data based on some specific key that is used as input
- Internally use the generic set infrastructure

Anonymous maps

This example shows how the destination TCP port selects the destination IP address to DNAT the packet

- nft add rule ip nat prerouting dnat tcp dport map { 80 : 192.168.1.100, 8888 : 192.168.1.101 }

This can be read as:

- if the TCP destination port is 80, then the packet is DNAT'ed to 192.168.1.100
- if the TCP destination port is 8888, then the packet is DNAT'ed to 192.168.1.101

nft: maps (2)

Named map

- nft add map nat porttoip { type inet_service: ipv4_addr }
- nft add element nat porttoip { 80 : 192.168.1.100, 8888 : 192.168.1.101 }
- nft add rule ip nat postrouting snat tcp dport map @porttoip

STAMVS NETWORKS

nft: dictionaries

Dictionaries

Also known as verdict maps, allow you to attach an action to an element

Anonymous dictionaries

This example shows how to create a tree of chains that whose traversal depends on the layer 4 protocol type:

- nft add rule ip foo bar ip protocol vmap { tcp : jump tcp-chain, udp : jump udp-chain, icmp : jump icmp-chain }

This rule-set arrangement allows you to reduce the amount of linear list inspections to classify your packets

STAMVS NETWORKS

nft: dictionaries (2)

Named dictionaries

- nft add map filter mydict { type ipv4_addr : verdict }
- nft add element filter mydict { 192.168.0.10 : drop, 192.168.0.11 : accept }
- nft add rule filter input ip saddr vmap @mydict

STAMVS NETWORKS

nft: concatenations

Contenations

Permits put two or more selectors together to perform very fast lookups by combining them with sets, dictionaries and maps.

- nft add rule ip filter input ip saddr . ip daddr . ip protocol { 1.1.1.1 . 2.2.2.2 . tcp, 1.1.1.1 . 3.3.3.3 . udp } counter accept

In this example if the packet matches the source IP address AND destination IP address AND TCP destination port,
nftables update the counter for this rule and then accepts the packet

STAMVS NETWORKS

nft: scripting

Scripting

nftables provides a native scripting environment to maintain the ruleset

```
#!/usr/sbin/nft

# we can include other rulesets
include "ipv4-nat.nft"
include "ipv6-nat.nft"

# and declare variable also
define google_dns = { 8.8.8.8, 8.8.4.4 }

add table filter
add chain filter input { type filter hook input priority 0; }
add rule filter input ip saddr $google_dns counter
```

Load the script

nft -f ruleset.nft

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

Tables and Chains

Tables

- Each table has a specific purpose and chains
- There are 5 main built-in tables in iptables
- It's not possible to add user-defined tables

Chains

- Each chain has a specific purpose and contains a ruleset that is applied on packets that traverse the chain

STAMVS NETWORKS

Tables

Filter table

- Used for filtering packets
- We can match packets and filter them in whatever way we may want
- This is the place that we actually take actions against packets
    - ACCEPT
    - DROP
    - LOG
    - REJECT
- Three built-in chains

STAMVS NETWORKS

Tables

Filter's chains

- INPUT
  - It's used on all packets that are destined for the firewall
- FORWARD
  - It's used on all non-locally generated packets that are not destined for our localhost
- OUTPUT
  - It's used for all locally generated packets

STAMVS NETWORKS

Tables

NAT table

- It's used mainly for Network Address Translation
- NATed packets get their IP addresses (or ports) altered, according to our rules
- Packets in a stream only traverse this table once
- We assume that the first packet of a stream is allowed
- The rest of the packets in the same stream are automatically NATted, Masqueraded, etc.

STAMVS NETWORKS

Tables

NAT's chains

- PREROUTING
  - It's used to alter packets as soon as they get into the firewall
- OUTPUT
  - It's used for altering locally generated packets before they get to the routing decision
- POSTROUTING
  - It's used to alter packets just as they are about to leave the firewall

STAMVS NETWORKS

Tables

Mangle table

- This table is used mainly for mangling packets
- Among other things, we can change the content of different packets and some of their headers
- Examples
  - TTL
  - ToS
  - Mark

STAMVS NETWORKS

Tables

Mangle's chains

- PREROUTING
  - it's used for altering packets just as they enter the firewall and before they hit the routing decision
- POSTROUTING
  - it's used to mangle packets just after all routing decisions have been made
- INPUT
  - it's used to alter packets after they have been routed to the localhost itself, but before the userspace software sees the data
- FORWARD
  - it's used to mangle packets after they have hit the first routing decision, but before they actually hit the last routing decision
- OUTPUT
  - it's used for altering locallty generated packets after they enter the routing decision

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMUS NETWORKS

Rule

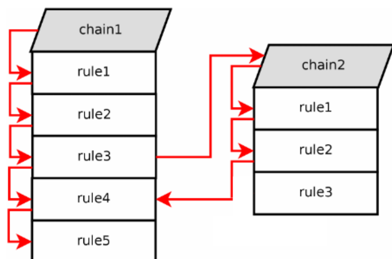
Rule

A rule is a set of criteria with a target that specify the action to take

Target

- ACCEPT
  - the packet is accepted (it's sent to the destination)
- DROP
  - the packet is dropped (it's not sent to the destination)
- User-defined chain
  - another ruleset is executed
- RETURN
  - stops executing the next set of rules in the current chain for this packet.
  - The control will be returned to the calling chain

The RETURN target

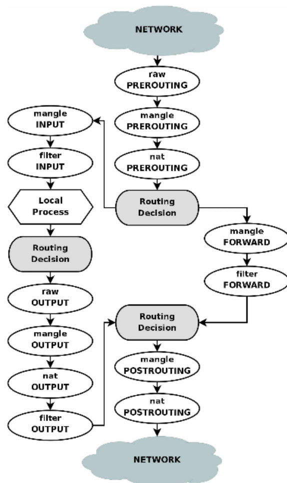


Return target

- A packet traverses chain1
- When rule3 matches the packet, it is sent to chain 2
- The packet traverses chain2 until is matched by rule2
- At this point, packet returns to chain1 and rule3 is not tested

STAMUS NETWORKS

Packet Path



1. Netfilter
   - Nftables
   - Tables and chains
   - Rules
2. Suricata
   - Intro
   - IDS / IPS
   - Signatures
   - NFQUEUE
   - NFLOG
3. Mixed Mode
   - Introduction
   - Usage
   - Ninja usage
4. Conclusion

STAMVS NETWORKS

1. Netfilter
   - Nftables
   - Tables and chains
   - Rules
2. Suricata
   - Intro
   - IDS / IPS
   - Signatures
   - NFQUEUE
   - NFLOG
3. Mixed Mode
   - Introduction
   - Usage
   - Ninja usage
4. Conclusion

STAMVS NETWORKS

Intro

About Suricata

- OpenSource (GPLv2) backed by OISF
- Cross-platform support (primarily Linux and BSD)
- Stable versions 3.1 and 3.0.2
- Multi-threading and High Performance
- Protocol detection, file extraction, lua scripting
- Many supported output formats like Eve/Json
- Hardware Acceleration
- Reading PCAPs
- EmergingThreats ruleset support
- Support via IRC, Mailinglist, Redmine

STAMVS NETWORKS

About OISF

Open Information Security Foundation

- Non-profit foundation
- Support for community-driven technology like Suricata and libhtp
- Funding comes from donations
- Organizations can become Consortium members
- Organizes SuriCon and Trainings (User and Developer)

STAMVS NETWORKS

1. Netfilter
   - Nftables
   - Tables and chains
   - Rules
2. Suricata
   - Intro
   - IDS / IPS
   - Signatures
   - NFQUEUE
   - NFLOG
3. Mixed Mode
   - Introduction
   - Usage
   - Ninja usage
4. Conclusion

STAMVS NETWORKS

Suricata: capture modes

IDS

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

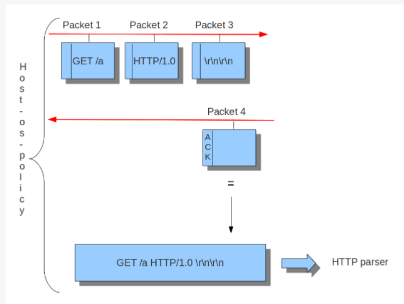
IPS

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

STAMVS NETWORKS

IDS behavior

Suricata receives traffic in chunks.



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

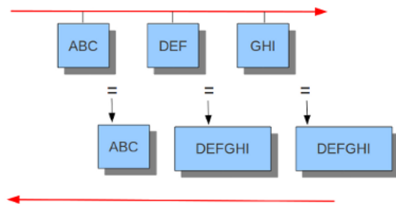
Suricata: IPS

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

STAMVS NETWORKS

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

STAMUS NETWORKS

Suricata: IPS

Inline mode

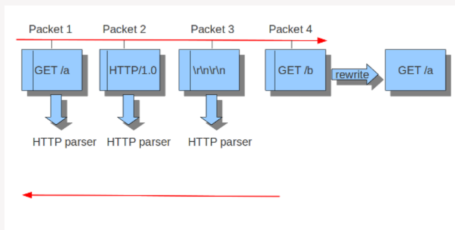
Normally, we analyse data once we know they have been received by the receiver, in term of TCP this means after it has been ACKed.
In IPS it does not work like this, because the data have reached the host that we protect.

STAMVS NETWORKS

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.

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

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.

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMUS NETWORKS

Suricata: NFQUEUE

NFQUEUE

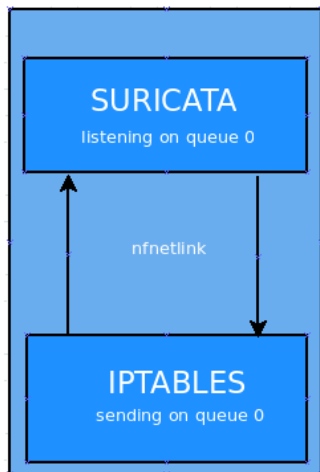
- It is used in Suricata to work in IPS mode, performing actions on the packet like DROP or ACCEPT.
- With NFQUEUE we are able to delegate the verdict on the packet to a userspace software
- The Linux kernel will ask a userspace software connected to a queue for a decision

Netfilter's rules

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

STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMUS NETWORKS

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

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

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

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

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, like 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

STAMVS NETWORKS

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

STAMVS NETWORKS

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

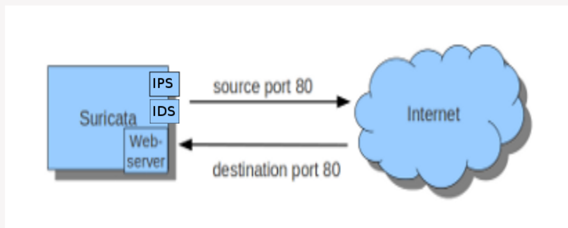
4 Conclusion

STAMVS NETWORKS

Mixed mode: usage

Scenario

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



STAMVS NETWORKS

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

STAMVS NETWORKS

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

STAMVS NETWORKS

Mixed mode: usage

Scenario 2

- This time we want to send all traffic of an IP address 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 queue 0
- This solution is not very performing because if we want to block another IP address we need to add another identical rule
  - rules duplication

STAMVS NETWORKS

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

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

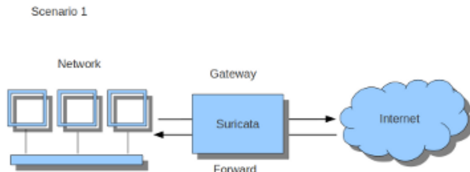
4 Conclusion

STAMVS 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

STAMVS NETWORKS

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 any any -> $SSH_SERVER any (msg:"Unexpected ssh connection"; sid:1234; rev:1234;)
- alert icmp any 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
}
```

1 Netfilter
- Nftables
- Tables and chains
- Rules

2 Suricata
- Intro
- IDS / IPS
- Signatures
- NFQUEUE
- NFLOG

3 Mixed Mode
- Introduction
- Usage
- Ninja usage

4 Conclusion

STAMVS NETWORKS

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`