NANOG 73 Hackathon

June 24th, 2018
DENVER, CO

Sponsored By

JUNIPER NETWORKS | Engineering Simplicity

What you should get out of this

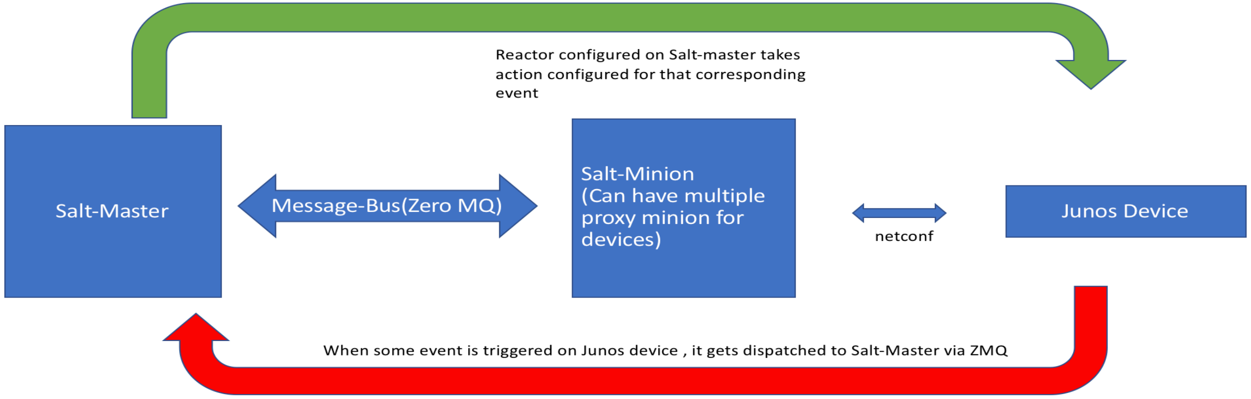
- Understanding that Automation plays a crucial part in security
- The Importance of Interoperability and Integration
- Scope of Automation for Security
- Security will require multiple parts of your organization to work together
- Understanding of DevSecOps

DevOpsqatestinfosec

- “In other words, when you hear "DevOps" today, you should probably be thinking *DevOpsQATestInfoSec*.” - Gene Kim

Implementation in the POD

- Each Pod has a Salt-Master and Salt-minion monitoring the vSRX
- Salt-Master is setup on Tools server
- Salt-Minion is setup on Trust server



Reactor configured on Salt-master takes action configured for that corresponding event

Salt-Master

Message-Bus(Zero MQ)

Salt-Minion (Can have multiple proxy minion for devices)

netconf

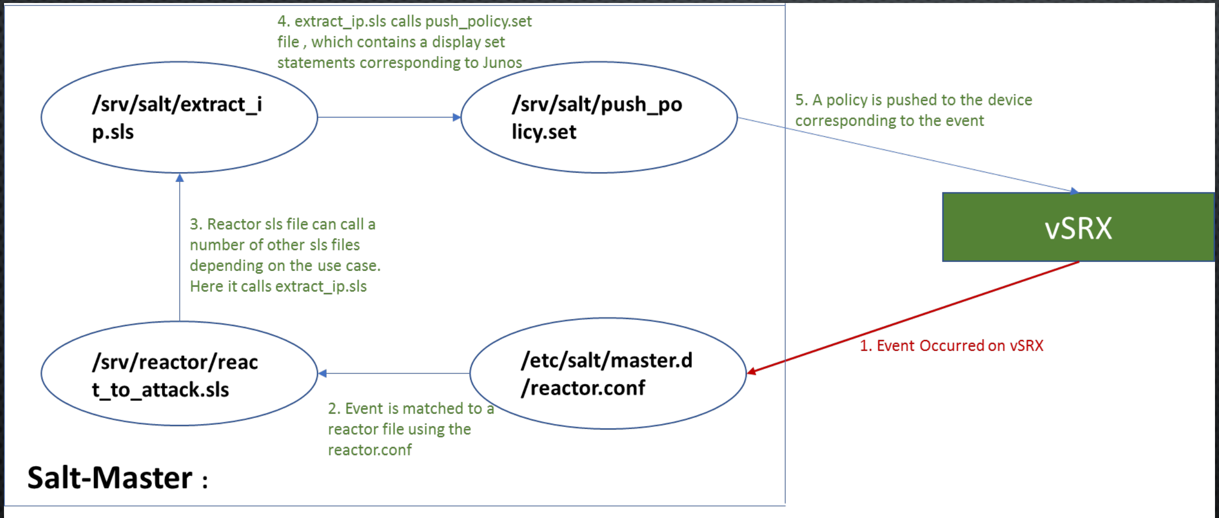
Junos Device

When some event is triggered on Junos device , it gets dispatched to Salt-Master via ZMQ

Basic Salt Reactor System

Salt-reactor

- THE MAIN PURPOSE OF THE SALT REACTOR IS TO LISTEN TO EVENTS TAKING PLACE ON THE VSRX AND REACT BASED ON THE ACTIONS ALREADY CONFIGURED VIA ANSIBLE , YAML , PYTHON SCRIPTS ALREADY CONFIGURED ON THE SALT-MASTER.
- A WORK FLOW OF **WHICH AND HOW FILES ON SALT-MASTER INTERACT** CORRESPONDING TO THE EVENT IS DESCRIBED BELOW :


4. extract_ip.sls calls push_policy.set file , which contains a display set statements corresponding to Junos
/srv/salt/extract_ip.sls
/srv/salt/push_policy.set
5. A policy is pushed to the device corresponding to the event
vSRX
3. Reactor sls file can call a number of other sls files depending on the use case. Here it calls extract_ip.sls
1. Event Occurred on vSRX
/srv/reactor/react_to_attack.sls
/etc/salt/master.d/reactor.conf
2. Event is matched to a reactor file using the reactor.conf
Salt-Master :

This diagram only show a single workflow of how salt reactor works. It is implemented in the POD assigned to the each team and the purpose of it to get participant familiar with Salt. Participants can create any number of workflows they want .

Hackers At Work!

ProblemGopher

Brandon Premo

Facebook

Jason Reifstenzel

Carleton University

Gabriel Nunez

Sandia National Lab

Akshat Sharma

Cisco

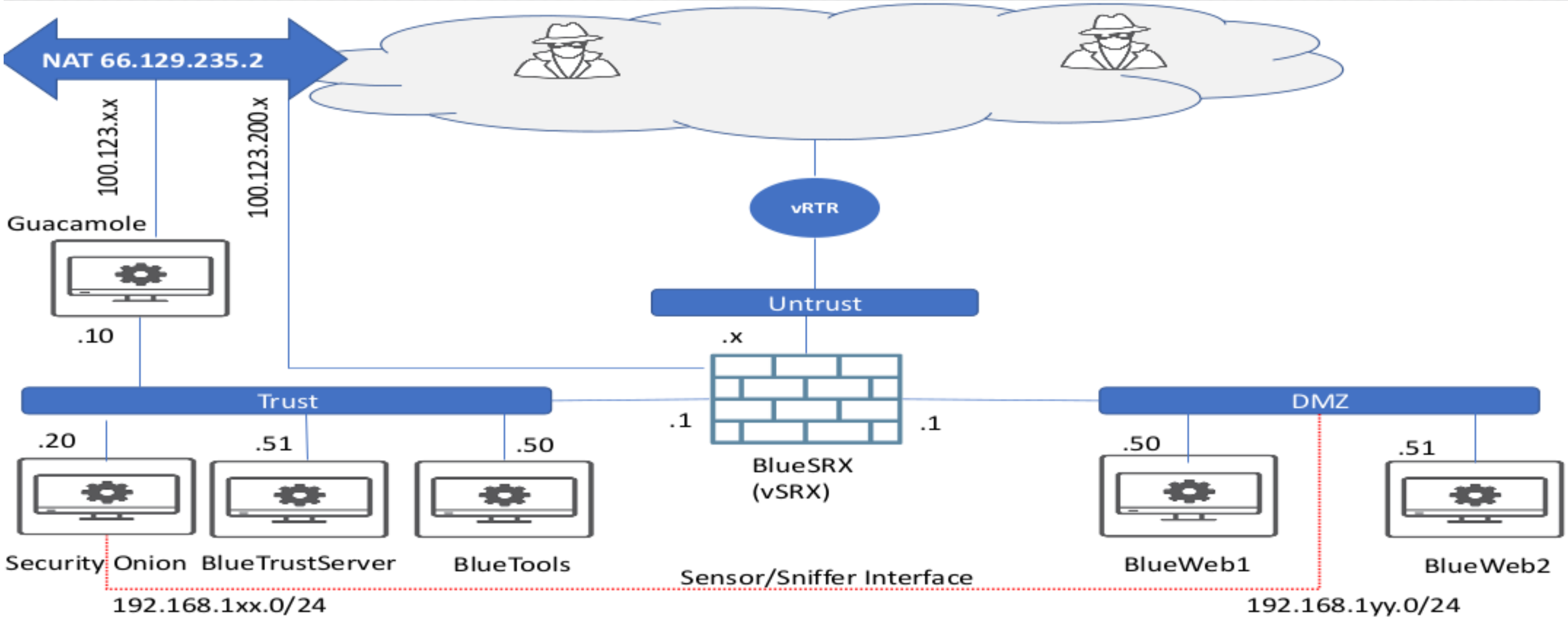
Mike Korshunov

Cisco

Overview

- Scenario Recap / Topology
- What we:
  - Saw
  - Did / Encountered as a problem
  - Would do differently

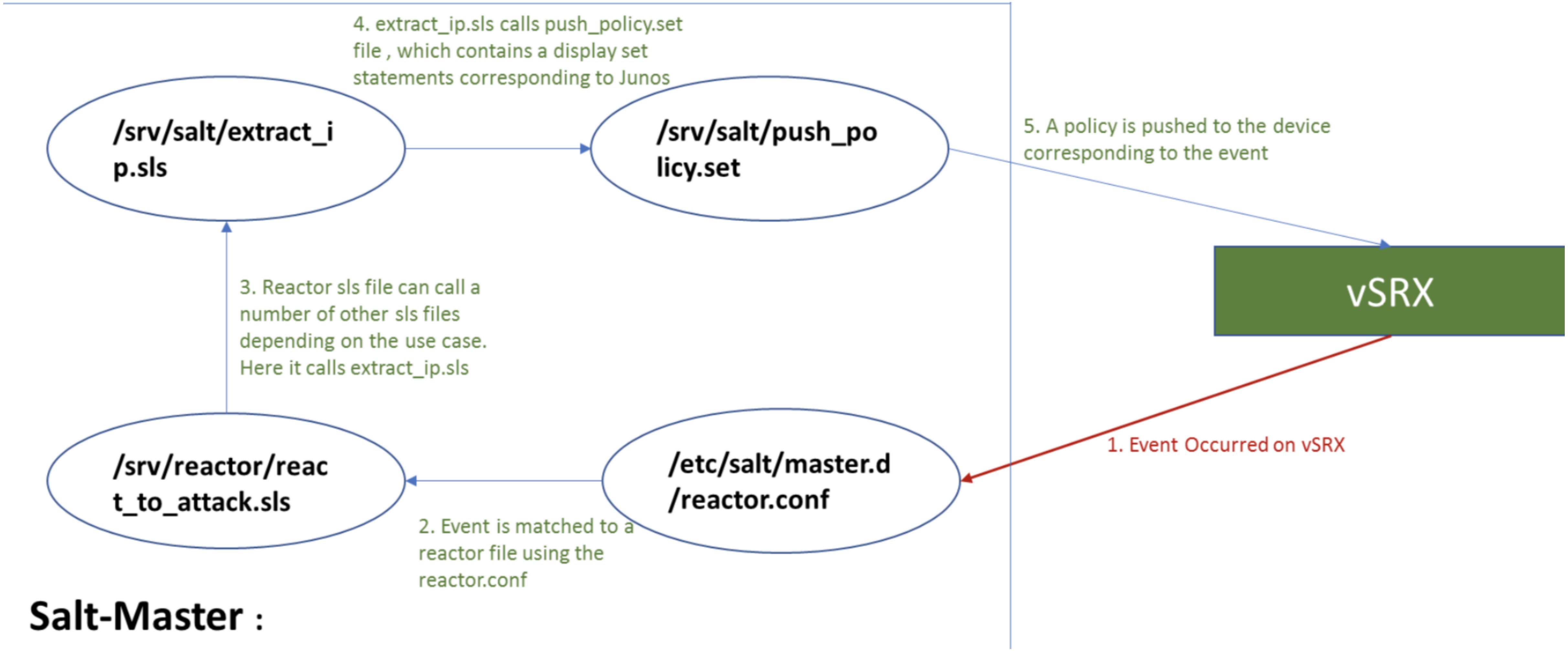
Topology


NAT 66.129.235.2
100.123.x.x
100.123.200.x
Guacamole
.10
vRTR
Untrust
.x
Trust
.1
.1
DMZ
.20
.51
.50
BlueSRX
(vSRX)
.50
.51
Security Onion
BlueTrustServer
BlueTools
Sensor/Sniffer Interface
BlueWeb1
BlueWeb2
192.168.1xx.0/24
192.168.1yy.0/24

Our initial thoughts

- Who has used salt before?
- How does this jinja thing work?
- What are we keying in on from the message bus?

What we did



4. extract_ip.sls calls push_policy.set file , which contains a display set statements corresponding to Junos

/srv/salt/extract_ip.sls

/srv/salt/push_policy.set

5. A policy is pushed to the device corresponding to the event

vSRX

3. Reactor sls file can call a number of other sls files depending on the use case. Here it calls extract_ip.sls

1. Event Occurred on vSRX

/srv/reactor/react_to_attack.sls

/etc/salt/master.d/reactor.conf

2. Event is matched to a reactor file using the reactor.conf

Salt-Master :

Parsing the message bus

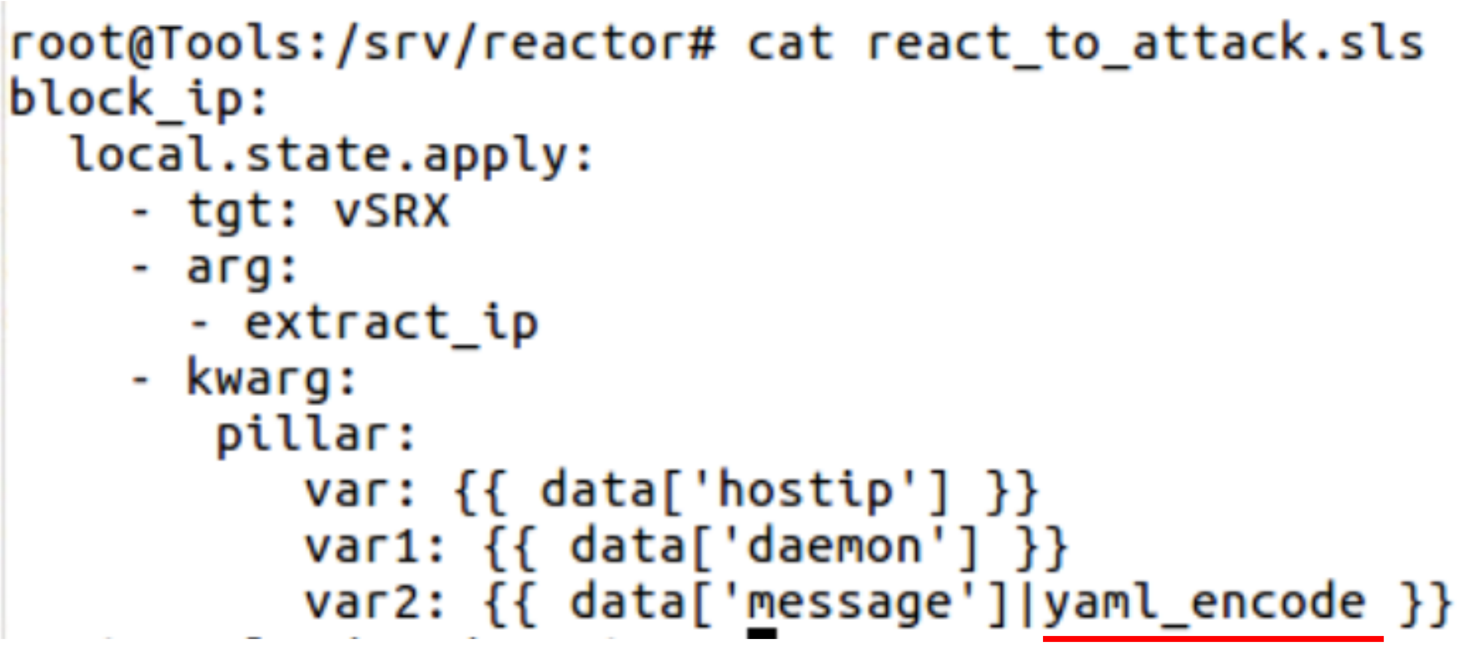
```
"jnpr/syslog/Blue8_SRX/SYSTEM": {
   "_stamp": "2018-06-24T17:29:45.277785",
   "daemon": "RT_IDP",
   "event": "SYSTEM",
   "facility": 1,
   "hostip": "192.168.108.1",
   "hostname": "Blue8_SRX",
   "message": "IDP: at 1529861385, ANOMALY Attack log <10.123.199.226/41691->192.168.128.51/21> for TCP protocol
and service FTP application FTP by rule 1 of rulebase IPS in policy NANOG. attack: id=2330, repeat=0, action=NONE,
threat-severity=HIGH, name=FTP:OVERFLOW:PASS-TOO-LONG, NAT <0.0.0.0:0->0.0.0.0:0>, time-elapsed=0, inbytes=0,
outbytes=0, inpackets=0, outpackets=0, intf:untrust:ge-0/0/0.0->dmz:ge-0/0/2.0, packet-log-id: 0, alert=no,
username=N/A, roles=N/A and misc-message -",
   "priority": 14,
   "raw": "<14>Jun 24 17:29:44 Blue8_SRX RT_IDP: IDP_ATTACK_LOG_EVENT: IDP: at 1529861385, ANOMALY Attack log
<10.123.199.226/41691->192.168.128.51/21> for TCP protocol and service FTP application FTP by rule 1 of rulebase
IPS in policy NANOG. attack: id=2330, repeat=0, action=NONE, threat-severity=HIGH, name=FTP:OVERFLOW:PASS-TOO-
LONG, NAT <0.0.0.0:0->0.0.0.0:0>, time-elapsed=0, inbytes=0, outbytes=0, inpackets=0, outpackets=0,
intf:untrust:ge-0/0/0.0->dmz:ge-0/0/2.0, packet-log-id: 0, alert=no, username=N/A, roles=N/A and misc-message -",
   "severity": 6,
   "timestamp": "2018-06-24 13:29:45"
},
```

Automated configuration Appliance:

```
auser4Tools:/srv/salt$ cat extract_ip.sls
{% set ip = pillar['var'] %}
{% set ip2 = {'ipN' : 'empty','ipk': 'name'} %}

{% for word in pillar['var2'].split() if "->" in
word %}
  {% set ip1 = word.split('->')[0] %}
  {% set ip1 = ip1|replace("<","") %}

  {% if '192.168.108.' in ip1 %}
      {% break %}
  {% endif %}

  {% if '10.123.198.4' in ip1 %}
      {% break %}
  {% endif %}
 {% if ip2.update({'ipk' : ip1 }) %} {% endif %}
 {% set ip1 = ip1.split('/')[0] %}
 {% if ip2.update({'ipN' : ip1 }) %} {% endif %}
 {% break %}

{% endfor %}
```

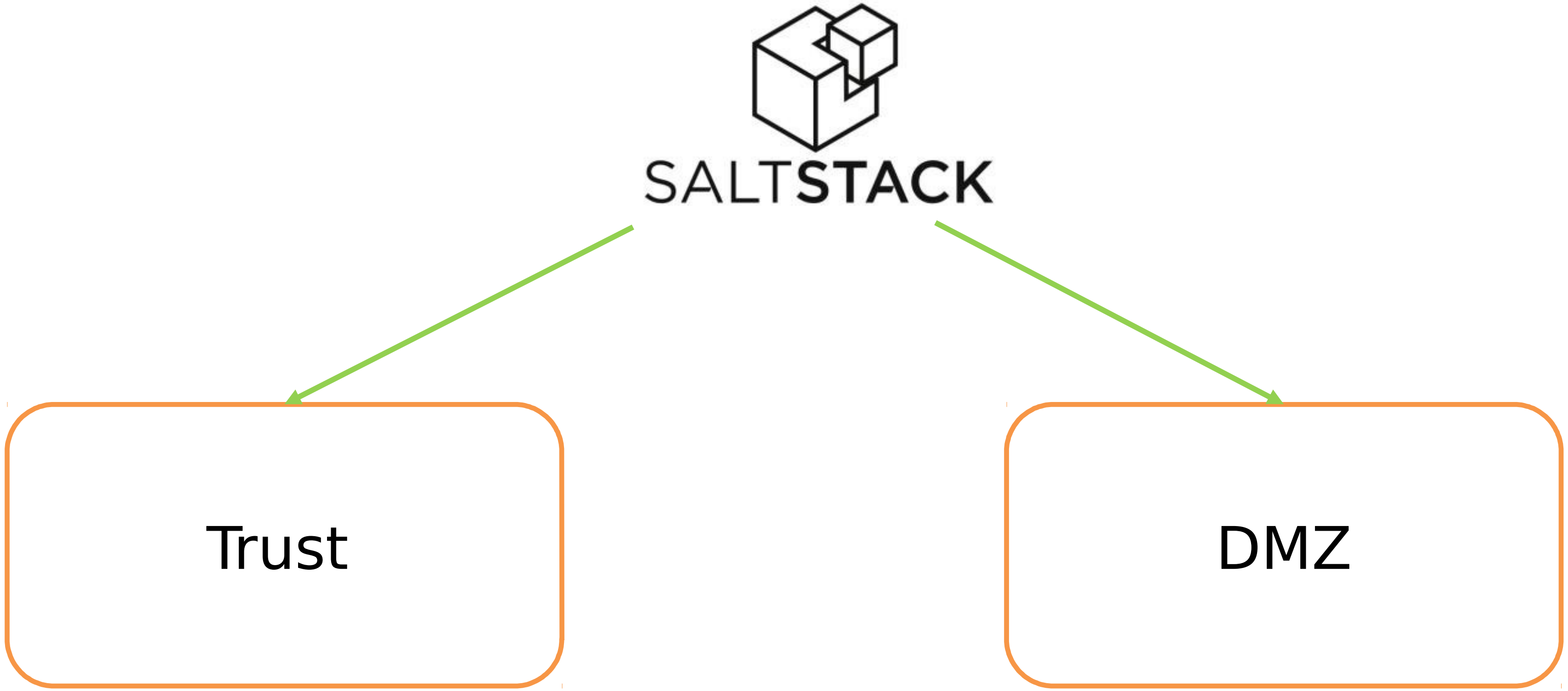
```
{% if pillar[‘var1’] == ‘RT_IDP’ %}
  salt://push_policy.set :
           junos:
               -  install_config
               -  template_vars:
                      host_ip:  {{ ip2[‘ipN’] }}
                      host_name:
{{ ip2[‘ipk’] }}
{% endif  %}
```

YAML encoding to avoid render problem



```
root@Tools:/srv/reactor# cat react_to_attack.sls
block_ip:
  local.state.apply:
    - tgt: vSRX
    - arg:
      - extract_ip
    - kwarg:
       pillar:
          var: {{ data['hostip'] }}
          var1: {{ data['daemon'] }}
          var2: {{ data['message']|yaml_encode }}
```

Next time:
Policy propagation to...



SALTSTACK

Trust

DMZ

Conclusion

- A++ would hack again
- Thanks to NANOG and Juniper

NANOG 73 Hackathon

Benedikt Rudolph - DECIX
Flavio Castro – Paypal
Shraddha Tekawade - Oracle (OCI)
Aaron Ashley - Oracle (OCI)
Andrew Warren - Oracle (OCI)
Syed W Ahmed - Oracle (OCI)

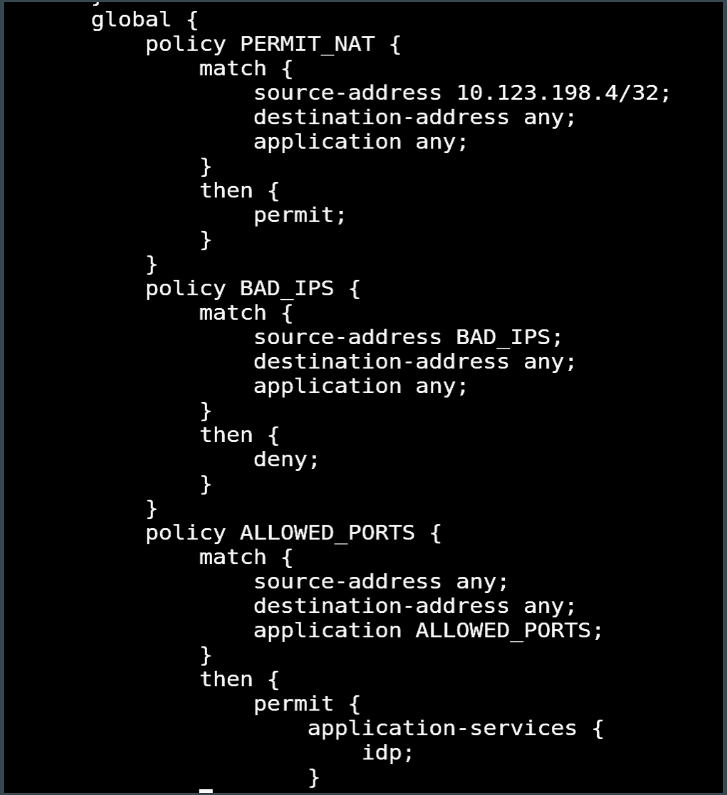
•••

Forensics – Where is the attack?

- Syn-Floods: noticed in Syslog / Kibana
- Ping floods: detected via security-onion in squert
- Service vulnerability: Detected via security-onion logs in squert
- Whitelisting public services from DMZ (global policy)
    - Prevents blocking good traffic by accident
- Went through all services on web1/2
    - Secured FTP
    - Patch Servers – more details later on that

SRX Implementation

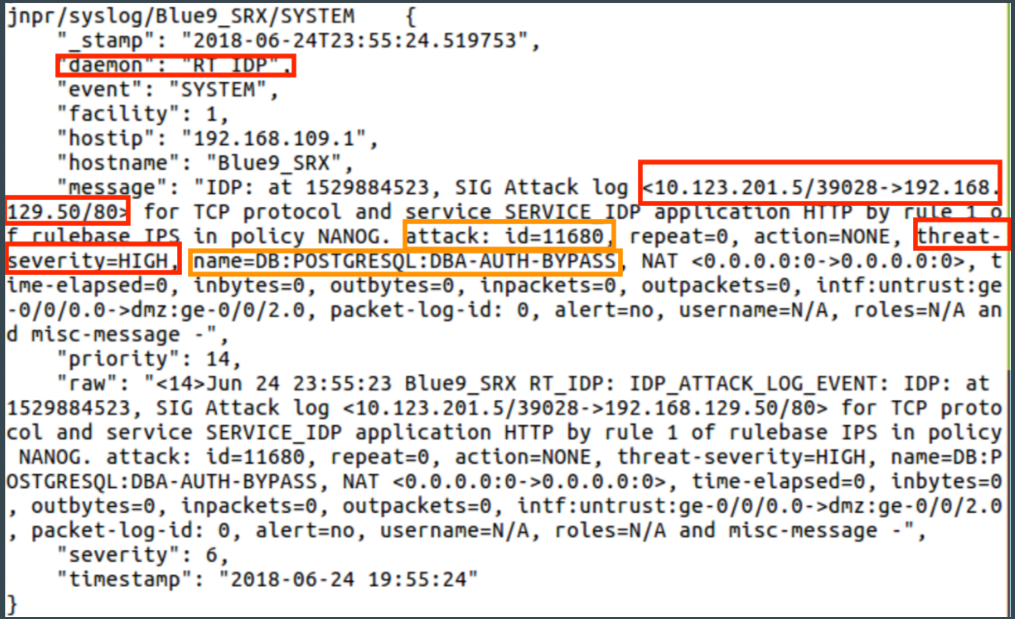
- Created policies that matched communication requirements
- Provided lockout protection
- Too many bad IP's to enter manually
- Support for automation by using an address-set



```
    global {
        policy PERMIT_NAT {
            match {
                source-address 10.123.198.4/32;
                destination-address any;
                application any;
            }
            then {
                permit;
            }
        }
        policy BAD_IPS {
            match {
                source-address BAD_IPS;
                destination-address any;
                application any;
            }
            then {
                deny;
            }
        }
        policy ALLOWED_PORTS {
            match {
                source-address any;
                destination-address any;
                application ALLOWED_PORTS;
            }
            then {
                permit {
                    application-services {
                        idp;
                    }
```

Automated Event Processing

- On Salt-master, processed syslog messages from SRX.
- Parsed messages from RT_IDP daemon
- Added addresses to the BAD_IPS address-set



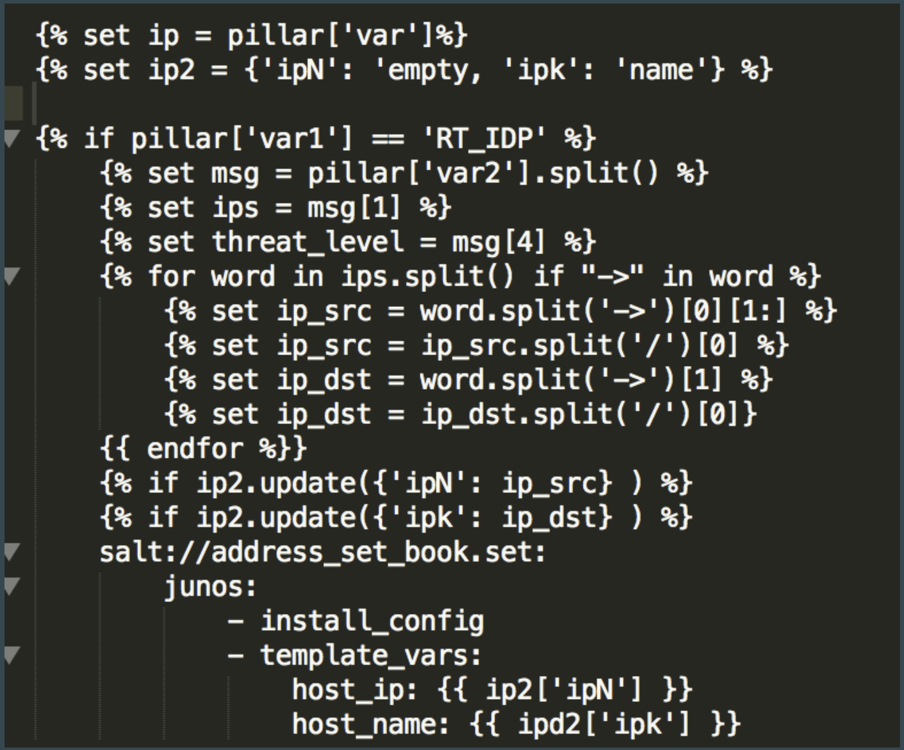
```
jnpr/syslog/Blue9_SRX/SYSTEM    {
    "_stamp": "2018-06-24T23:55:24.519753",
    "daemon": "RT_IDP",
    "event": "SYSTEM",
    "facility": 1,
    "hostip": "192.168.109.1",
    "hostname": "Blue9_SRX",
    "message": "IDP: at 1529884523, SIG Attack log <10.123.201.5/39028->192.168.
129.50/80> for TCP protocol and service SERVICE_IDP application HTTP by rule 1 o
f rulebase IPS in policy NANOG. attack: id=11680, repeat=0, action=NONE, threat-
severity=HIGH, name=DB:POSTGRESQL:DBA-AUTH-BYPASS, NAT <0.0.0.0:0->0.0.0.0:0>, t
ime-elapsed=0, inbytes=0, outbytes=0, inpackets=0, outpackets=0, intf:untrust:ge
-0/0/0.0->dmz:ge-0/0/2.0, packet-log-id: 0, alert=no, username=N/A, roles=N/A an
d misc-message -",
    "priority": 14,
    "raw": "<14>Jun 24 23:55:23 Blue9_SRX RT_IDP: IDP_ATTACK_LOG_EVENT: IDP: at
1529884523, SIG Attack log <10.123.201.5/39028->192.168.129.50/80> for TCP proto
col and service SERVICE_IDP application HTTP by rule 1 of rulebase IPS in policy
 NANOG. attack: id=11680, repeat=0, action=NONE, threat-severity=HIGH, name=DB:P
OSTGRESQL:DBA-AUTH-BYPASS, NAT <0.0.0.0:0->0.0.0.0:0>, time-elapsed=0, inbytes=0
, outbytes=0, inpackets=0, outpackets=0, intf:untrust:ge-0/0/0.0->dmz:ge-0/0/2.0
, packet-log-id: 0, alert=no, username=N/A, roles=N/A and misc-message -",
    "severity": 6,
    "timestamp": "2018-06-24 19:55:24"
}
```

Example event

Log Event Processing

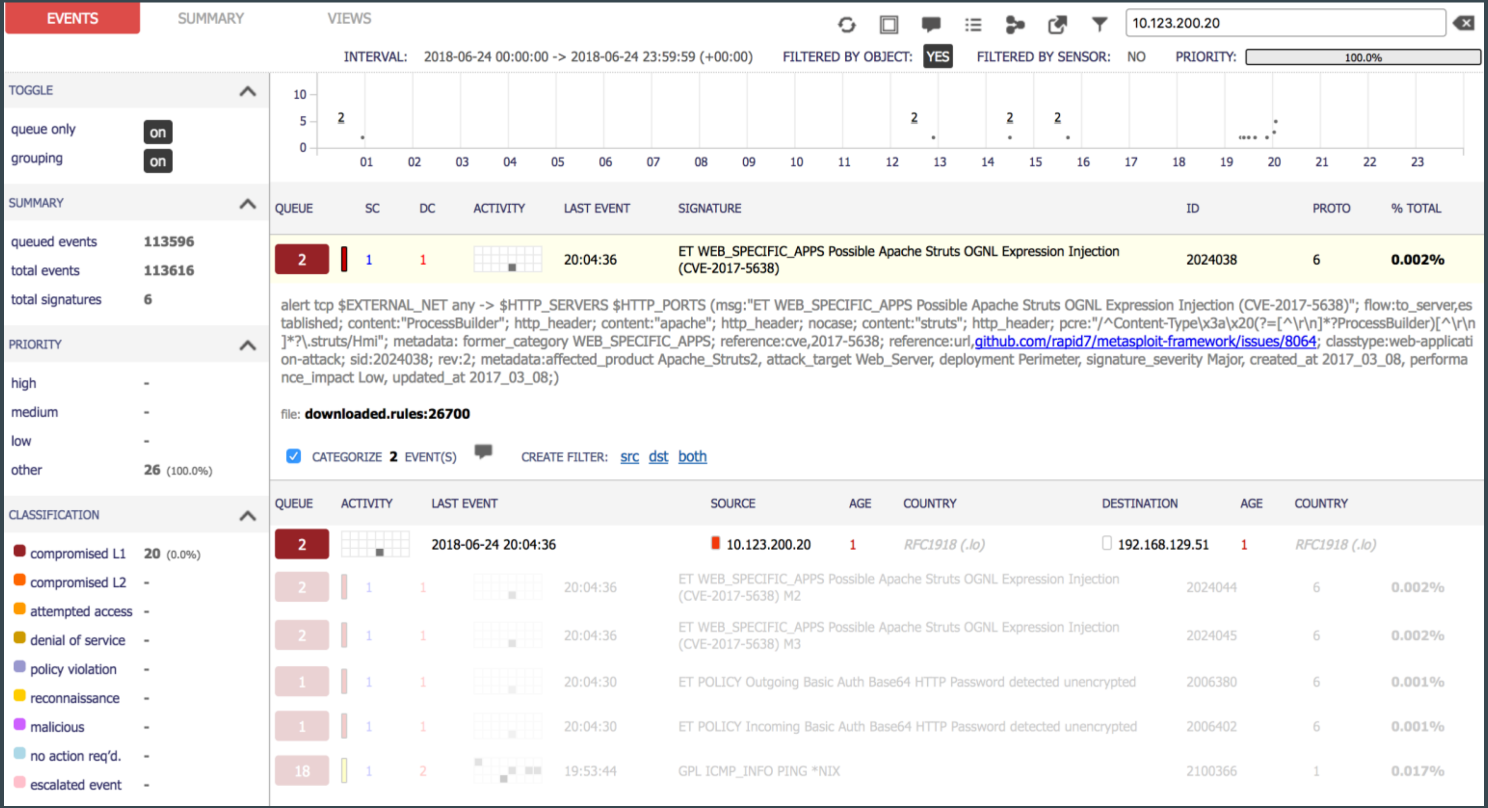
- Input data patterns were learned on the go
- Had multiple iteration on parsing correct src and dest and then take actions.
- At one point we blocked NAT and WEB1 and WEB2 Ips.

Final jinja template



```
{% set ip = pillar['var']%}
{% set ip2 = {'ipN': 'empty, 'ipk': 'name'} %}

{% if pillar['var1'] == 'RT_IDP' %}
    {% set msg = pillar['var2'].split() %}
    {% set ips = msg[1] %}
    {% set threat_level = msg[4] %}
    {% for word in ips.split() if "->" in word %}
        {% set ip_src = word.split('->')[0][1:] %}
        {% set ip_src = ip_src.split('/')[0] %}
        {% set ip_dst = word.split('->')[1] %}
        {% set ip_dst = ip_dst.split('/')[0]}
    {{ endfor %}}
    {% if ip2.update({'ipN': ip_src} ) %}
    {% if ip2.update({'ipk': ip_dst} ) %}
    salt://address_set_book.set:
        junos:
            - install_config
            - template_vars:
                host_ip: {{ ip2['ipN'] }}
                host_name: {{ ipd2['ipk'] }}
```

Uncovering Targeted Attacks with Squert



10.123.200.20

INTERVAL: 2018-06-24 00:00:00 -> 2018-06-24 23:59:59 (+00:00) FILTERED BY OBJECT: YES FILTERED BY SENSOR: NO PRIORITY: 100.0%

TOGGLE

queue only	on
grouping	on

SUMMARY

queued events	113596
total events	113616
total signatures	6

PRIORITY

high	-
medium	-
low	-
other	26 (100.0%)

CLASSIFICATION

compromised L1	20 (0.0%)
compromised L2	-
attempted access	-
denial of service	-
policy violation	-
reconnaissance	-
malicious	-
no action req'd.	-
escalated event	-

QUEUE	SC	DC	ACTIVITY	LAST EVENT	SIGNATURE	ID	PROTO	% TOTAL
2	1	1		20:04:36	ET WEB_SPECIFIC_APPS Possible Apache Struts OGNL Expression Injection (CVE-2017-5638)	2024038	6	0.002%

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"ET WEB_SPECIFIC_APPS Possible Apache Struts OGNL Expression Injection (CVE-2017-5638)"; flow:to_server,established; content:"ProcessBuilder"; http_header; content:"apache"; http_header; nocase; content:"struts"; http_header; pcre:"/^Content-Type\x3a\x20(?=[^\r\n]*?ProcessBuilder)[^\r\n]*?\.struts/Hmi"; metadata: former_category WEB_SPECIFIC_APPS; reference:cve,2017-5638; reference:url,github.com/rapid7/metasploit-framework/issues/8064; classtype:web-application-attack; sid:2024038; rev:2; metadata:affected_product Apache_Struts2, attack_target Web_Server, deployment Perimeter, signature_severity Major, created_at 2017_03_08, performance_impact Low, updated_at 2017_03_08;)
```

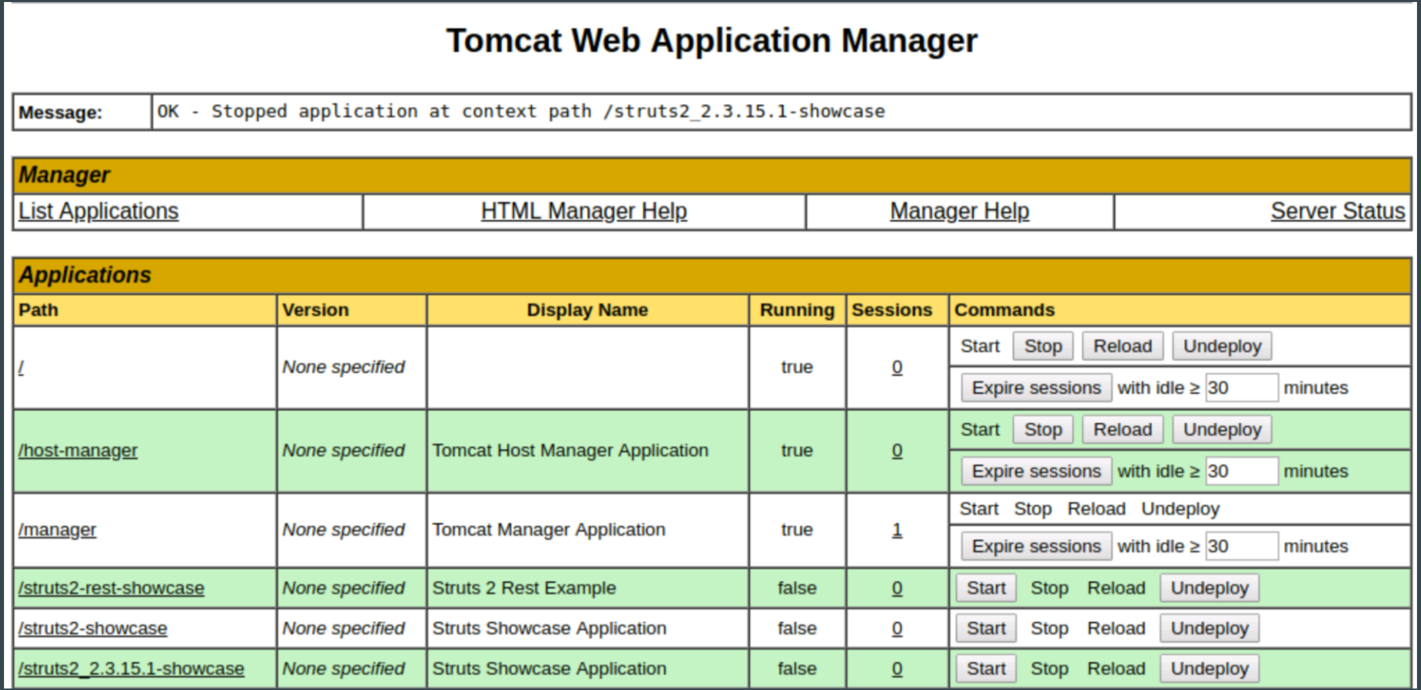
file: **downloaded.rules:26700**

☑ CATEGORIZE **2** EVENT(S) CREATE FILTER: src dst both

QUEUE	ACTIVITY	LAST EVENT	SOURCE	AGE	COUNTRY	DESTINATION	AGE	COUNTRY
2		2018-06-24 20:04:36	10.123.200.20	1	RFC1918 (.lo)	192.168.129.51	1	RFC1918 (.lo)

QUEUE	SC	DC	ACTIVITY	LAST EVENT	SIGNATURE	ID	PROTO	% TOTAL
2	1	1		20:04:36	ET WEB_SPECIFIC_APPS Possible Apache Struts OGNL Expression Injection (CVE-2017-5638) M2	2024044	6	0.002%
2	1	1		20:04:36	ET WEB_SPECIFIC_APPS Possible Apache Struts OGNL Expression Injection (CVE-2017-5638) M3	2024045	6	0.002%
1	1	1		20:04:30	ET POLICY Outgoing Basic Auth Base64 HTTP Password detected unencrypted	2006380	6	0.001%
1	1	1		20:04:30	ET POLICY Incoming Basic Auth Base64 HTTP Password detected unencrypted	2006402	6	0.001%
18	1	2		19:53:44	GPL ICMP_INFO PING *NIX	2100366	1	0.017%

Stop Tomcat Service



Tomcat Web Application Manager

Message:	`OK - Stopped application at context path /struts2_2.3.15.1-showcase`

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	*None specified*		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	*None specified*	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	*None specified*	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/struts2-rest-showcase	*None specified*	Struts 2 Rest Example	false	0	Start Stop Reload Undeploy
/struts2-showcase	*None specified*	Struts Showcase Application	false	0	Start Stop Reload Undeploy
/struts2_2.3.15.1-showcase	*None specified*	Struts Showcase Application	false	0	Start Stop Reload Undeploy

Saltstack experience

- Saltstack is very hard to diagnose / debug*
- Fixed parsing but then broke automated policy push due to a syntax error, which was fixed later.
- Pushing the policy is easy
- Frequency of attack events reduced software testing speed
- Only received logs from SRX IDP no security onion messages
- Saltstack event log structure differed from raw and kibana logs

*This could be from lack of experience with tool.

Future enhancements

- Test driven development would’ve been nice
- Block on threat level in log
- Process security-onion logs and automate actions based on that as well
- Use better parsing to include src/dst ports
- Make firewall rules zone based

Thank you!

JUNIPER NETWORKS | Engineering Simplicity