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**
• “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**

![Diagram of Basic Salt Reactor System]

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

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!
Overview

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

1. Event Occurred on vSRX

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

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

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

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

/vsr/salt/extract_ip.sls

/vsr/salt/push_policy.set

/vsr/reactor/react_to_attack.sls

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

vSRX

Salt-Master :
"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:

```python
# extract_ip.sls

# Configuration script for Juniper appliance

pillar = {'var': 'example_value'}

# Define variables for Juniper Junos configuration

salt://push_policy.set:
  junos:
    - install_config
    - template_vars:
      host_ip: {{ pillar['var'] }}
      host_name: {{ pillar['var'] }}

# Extract IP addresses from pillar

ip1 = pillar['var'].split('->')
ip1 = ip1.split('/0')

# Filter IP addresses

if '192.168.108.' in ip1:
  break
if '10.123.198.4' in ip1:
  break
if '10.123.198.4' in ip1:
  break

# Update IP addresses

ip2 = {'ipN': 'empty', 'ipk': 'name'}

for word in pillar['var2'].split() if '->' in word:
  ip1 = word.split('->')[0]
  ip1 = ip1.replace('<', '')
  if '192.168.108.' in ip1:
    break
  if '10.123.198.4' in ip1:
    break
  if ip2.update({'ipk': ip1})
  ip1 = ip1.split('/')[0]
  if ip2.update({'ipN': ip1})
  break

# Additional configuration

if pillar['var1'] == 'RT_IDP':
  salt://push_policy.set:
    junos:
      - install_config
      - template_vars:
        host_ip: {{ pillar['var2'] }}
        host_name: {{ pillar['var2'] }}
```

```bash
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: {{ pillar['ipN'] }}
        host_name: {{ pillar['ipk'] }}
{% endif %}
```
YAML encoding to avoid render problem

```yaml
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
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

Example event

```
jnpr/syslog/Blue9_SRX/SYSTEM {
  "daemon": "RT_IDP",
  "event": "SYSTEM",
  "facility": 1,
  "hostip": "192.168.109.1",
  "hostname": "Blue9_SRX",
  "message": "IDP: at 1529884523, SIG Attack log 129.50/80 for TCP protocol 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:POSTGRES: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=0/0.0.0.0/0.0.0.0, dnat=0/0.0.0.0, rnat=0/0.0.0.0, packet-log-id: 0, alert=no, username=N/A, roles=N/A and msg-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 protocol 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:POSTGRES: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=0/0.0.0.0, dnat=0/0.0.0.0, rnat=0/0.0.0.0, packet-log-id: 0, alert=no, username=N/A, roles=N/A and msg-message:"

  "severity": 6,
  "timestamp": "2018-06-24 19:55:24"
}
```
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

```jinja
{% 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][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
Stop Tomcat Service
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!