June 24th, 2018
DENVER, CO

Sponsored By

NANOG 73
Hackathon

Sponsored By

JUNIPER NETWORKS
Engineering Simplicity
WIRELESS INFORMATION

• **NANOG** - **This is 802.1X secured using the below user:pass (2.4GHz and 5GHz)**
• **NANOG** - **legacy** - **This is an open, unencrypted network (2.4GHz and 5GHz)**

• **The login information for the 802.1x secured network is:**
  • **User:** nanog
  • **Pass:** nanog
GENERAL LOGISTICS

- Meals & Breaks – Mineral D-G
- Bathrooms – Near Elevators
- NANOG Registration – Centennial Foyer 4PM – 6PM
- Sunday Social Event - Cervantes Concert Hall – 6:30PM - 10:30PM
  - NANOG Badge Required for Entry
AGENDA

• 8:30AM – 9:30AM Registration & Breakfast
• 9:30AM – 10:30AM Opening/Introduction, and Tutorial
• 10:30AM Break into Groups
• 12:30PM -1:30PM Lunch
• 1:30PM Resume Groups
• 3:00PM Break – Refreshments
• 6:00PM Hack Deadline, Prototype Demos, Voting
• 6:50PM Closing & Raffle Giveaway
• 7:00PM - 8:00PM Hackathon Reception
1

1. Hack-a-thon Goals

2. High-level Outline

3. Topology

4. Team Description, Game Rules, and Ideas
What to expect

- **This workshop is designed to teach you how to integrate multi-vendor tools to create a cohesive security solution to protect your business services.**
- **This is not** a workshop designed to teach you how to use a specific set of tools, and there will be help along the way to ensure you have a positive experience.

- **Stages:**
  - **Phase 1:** Familiarization with the tools – single attacker
  - **Phase 2:** More volume and more IPs using automation
  - **Phase 3:** is targeted.

*Unnecessary services or permissions will be present.*
And the winner is...

- Forensics
- How you stopped it or...
- How you would stop it
- You will vote on the best solution and presentation
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
Humans: Errors and Other bad behavior

Almost 90% of Cyber Attacks are Caused by Human Error or Behavior

Overall, the research found that about 90% of all cyber claims stemmed from some type of human error or behavior.

“The simple truth is that a data compromise is more likely to come from an employee leaving a laptop on the train than from a malicious criminal hack,” said Anthony Dagostino, the company’s head of global cyber risk.

Addressing the problem could be a simple matter of conducting training sessions that advise employees to use approved software and apply strong passwords—or applying common sense practices around technology access. For example, perhaps it’s not such a great idea to allow staff to take their laptops home on a Friday evening, should they happen to be drop by the local bar to unwind after a hard week’s work.

“90% OF ALL CYBER CLAIMS STEMMED FROM SOME TYPE OF HUMAN ERROR OR BEHAVIOR.”
THE CLOUD

THERE IS NO CLOUD.
IT’S JUST SOMEBODY ELSE’S COMPUTER.
Getting lots of calls now?
YOUR NETWORK COULD BE THE FIREWALL

End-to-end visibility districts need to secure the entire network

Instant threat intelligence and detection

Dynamically adapting policy, deployed in real-time

Enforce security everywhere

Secure Network strategy for UNTRUST/UNTRUST Model
Anatomy of an attack

Reconnaissance → Infiltrate → Expand/Pivot → Exploit
Today’s Event

- **Focus is on Defending Your Assets**
- **7-Hour Event progressing in Difficulty**
- **Blue Teams (You) must prevent Red Teams (Us) from manipulating your security posture and stopping your business services**
- **Blockers will be in place for increased difficulty and to teach you additional security measures**
- **You will select tools from a list**
- **More than one correct method**
- **We will assist you if you get stuck**
- **Red team will be partially automated as well as staffed by Juniper Networks™/NANOG**
NANOG Hackathon

Rules of Engagement, Topology and Tools Document
Blue Team Topology

- Trust
  - Security Onion
  - BlueTrustServer
  - BlueTools
  - BlueSRX (vSRX) 192.168.1xy.0/24
- DMZ
  - BlueWeb1
  - BlueWeb2
  - Sensor/Sniffer Interface 192.168.1yy.0/24
- Untrust
  - vRTR
- NAT 66.129.235.2
- Guacamole 100.123.xx.0/24

Network

1. NAT 66.129.235.2
2. Trust
   - Security Onion
   - BlueTrustServer
   - BlueTools
   - BlueSRX (vSRX) 192.168.1xy.0/24
3. DMZ
   - BlueWeb1
   - BlueWeb2
   - Sensor/Sniffer Interface 192.168.1yy.0/24
Accessing Blue POD:
https://66.129.235.2:40930/guacamole
Accessing Dashboard:
HTTP://66.129.235.2:40888/SHOWSTATUS.PHP

NANOG Hackathon Stats

Pod 1
Good IPs Blocked: 0/20
DDOS IPs blocked: 0/20
Recon IPs blocked: 0/20
Malware Blocked: 0/11

Pod 2
Good IPs Blocked: 0/20
DDOS IPs blocked: 0/20
Recon IPs blocked: 0/20
Malware Blocked: 0/11

Pod 3
Good IPs Blocked: 0/20
DDOS IPs blocked: 0/20
Recon IPs blocked: 0/20
Malware Blocked: 0/11

Pod 4
Good IPs Blocked: 0/20
DDOS IPs blocked: 0/20
Recon IPs blocked: 0/20
Malware Blocked: 0/11
## Credentials

**Accessing Blue POD:** [https://66.129.235.2:40930/guacamole](https://66.129.235.2:40930/guacamole)

<table>
<thead>
<tr>
<th>Device/App</th>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guacamole</td>
<td>PodXuser</td>
<td>Nanog18</td>
</tr>
<tr>
<td>Blue Servers (DMZ &amp; Trust)</td>
<td>auser</td>
<td>root123</td>
</tr>
<tr>
<td>SRX</td>
<td>root</td>
<td>root123</td>
</tr>
<tr>
<td>SecurityOnion</td>
<td>auser</td>
<td>root123</td>
</tr>
</tbody>
</table>
Basic SRX Policy - Stanza

eroot@blue1_SRX# show security policies
from-zone untrust to-zone trust {
    policy Salt-block {
        match {
            source-address [ test2 test3 10.123.197.11/30888 10.123.197.11/6785 10.123.198.231/56103 ];
            destination-address any;
            application any;
        }
        then {
            permit;
        }
    }
} global {
    policy PERMIT_ANY {
        match {
            source-address any;
            destination-address any;
            application any;
        }
        then {
            permit {
                application-services {
                    idp;
                }
                log {
                    session-init;
                    session-close;
                }
            }
        }
    }
}
Basic SRX Policy – Display Set

```
[edit]
root@Blue1_SRX# show security policies | display set
set security policies from-zone untrust to-zone trust policy Salt-block match source-address test2
set security policies from-zone untrust to-zone trust policy Salt-block match source-address test3
set security policies from-zone untrust to-zone trust policy Salt-block match source-address 10.123.197.11/30886
set security policies from-zone untrust to-zone trust policy Salt-block match source-address 10.123.197.11/6785
set security policies from-zone untrust to-zone trust policy Salt-block match source-address 10.123.198.231/56103
set security policies from-zone untrust to-zone trust policy Salt-block match destination-address any
set security policies from-zone untrust to-zone trust policy Salt-block match application any
set security policies from-zone untrust to-zone trust policy Salt-block then permit
set security policies global policy PERMIT_ANY match source-address any
set security policies global policy PERMIT_ANY match destination-address any
set security policies global policy PERMIT_ANY match application any
set security policies global policy PERMIT_ANY then permit application-services idp
set security policies global policy PERMIT_ANY then log session-init
set security policies global policy PERMIT_ANY then log session-close

[edit]
root@Blue1_SRX#```
<table>
<thead>
<tr>
<th>From Zone</th>
<th>To Zone</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>global</td>
<td>global</td>
<td>ping, traceroute</td>
</tr>
<tr>
<td>untrust</td>
<td>dmz</td>
<td>http, https, alt_http (8000 – 9000), ftp, scp</td>
</tr>
<tr>
<td>dmz</td>
<td>trust</td>
<td>http, https, alt_http (8000 – 9000), ftp, scp</td>
</tr>
<tr>
<td>Trust</td>
<td>Untrust</td>
<td>internet access – play videos, facebook, etc</td>
</tr>
</tbody>
</table>
Blue Team Tools:

1. Security Onion
2. Salt-Stack with J-EDI

(These two tools are pre-installed and setup on the Blue Team POD, participants are welcome to install any other tools of their choice)
Introduction to the tool

• **Security Onion** is a free and open source Linux distribution for intrusion detection, enterprise security monitoring, and log management. It includes Elasticsearch, Logstash, Kibana, Snort, Suricata, Bro, OSSEC, Sguil, Squert, NetworkMiner, and many other security tools.

• **Accessing the Tool:**
  • Security Onion’s Kibana user interface can be accessed from the POD by browsing to the URL [HTTPS://192.168.XXX.20/app/kibana](https://192.168.XXX.20/app/kibana)

• **Credentials:**
  • Username: AUSER
  • Password: root123
Security onion dashboard

Security Onions UI provides easy access to the alert, host, session, transaction data. Participant can navigate using this dashboard **dashboard** and get all sort of alert metrics.
Security onion dashboard

Log Types:
- RT_FLOW: 447,411
- ossec_archive: 40,911
- bro_conn: 18,198
- ORION: 13,173
- win32: 4,952
- ossec: 886
- rsync: 830
- systemd: 166
- RT_IDE: 80
- bro_dns: 66

Notices - Log Count: 4
NIDS - Alert Count: 4,952
OSSEC Alerts - Log Count: 886
Security onion dashboard
Security onion dashboard

NIDS alert (continued): More stats can be found at the bottom of the page to dig deeper into one log
SQUERT (can be accessed from the left navigation pane of Kibana Dashboard)
<table>
<thead>
<tr>
<th>COUNT</th>
<th>% TOTAL</th>
<th>MSE</th>
<th>MST</th>
<th>SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7996</td>
<td>12.72%</td>
<td>1</td>
<td>2</td>
<td>GET WEB_SERVER Script tag in URI Possible Cross Site Scripting attempt</td>
</tr>
<tr>
<td>7672</td>
<td>12.28%</td>
<td>0</td>
<td>2</td>
<td>GET X-SERVINFO PHP injection attack (mysql)</td>
</tr>
<tr>
<td>6124</td>
<td>9.98%</td>
<td>1</td>
<td>2</td>
<td>GET WEB_SERVER: Crip System Stack (mbx)</td>
</tr>
<tr>
<td>8612</td>
<td>7.96%</td>
<td>1</td>
<td>2</td>
<td>GET 10X Executable Downloader from dotted quad host</td>
</tr>
<tr>
<td>3676</td>
<td>0.78%</td>
<td>1</td>
<td>2</td>
<td>GPL EXPL0T: jaf access</td>
</tr>
<tr>
<td>2517</td>
<td>0.43%</td>
<td>1</td>
<td>2</td>
<td>GET WEB SERVER Possible CGI=2014-427, attempt in header</td>
</tr>
<tr>
<td>2042</td>
<td>0.34%</td>
<td>1</td>
<td>2</td>
<td>GET WEB SERVER Possible CGI=2014-427, attempt</td>
</tr>
<tr>
<td>1929</td>
<td>0.31%</td>
<td>1</td>
<td>2</td>
<td>GET SCAI Traced Auth Bude Force attempt (torcall)</td>
</tr>
<tr>
<td>1738</td>
<td>0.29%</td>
<td>1</td>
<td>2</td>
<td>GPL EXPL0T accesses access</td>
</tr>
<tr>
<td>1228</td>
<td>0.20%</td>
<td>1</td>
<td>2</td>
<td>GPL WEB_SERVER adobe.exe access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>% TOTAL</td>
<td>MSE</td>
<td>MST</td>
<td>SIGNATURE</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-----</td>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td>5326</td>
<td>89.22%</td>
<td>112</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>3.08%</td>
<td>1</td>
<td>2</td>
<td>192.12.202.1</td>
</tr>
<tr>
<td>1369</td>
<td>2.44%</td>
<td>1</td>
<td>2</td>
<td>192.12.202.1</td>
</tr>
<tr>
<td>108</td>
<td>0.20%</td>
<td>6</td>
<td>6</td>
<td>192.168.121.3</td>
</tr>
<tr>
<td>904</td>
<td>0.49%</td>
<td>1</td>
<td>2</td>
<td>192.168.121.5</td>
</tr>
<tr>
<td>195</td>
<td>0.28%</td>
<td>1</td>
<td>2</td>
<td>192.14.151.10</td>
</tr>
<tr>
<td>160</td>
<td>0.16%</td>
<td>1</td>
<td>2</td>
<td>192.168.121.3</td>
</tr>
<tr>
<td>99</td>
<td>0.16%</td>
<td>1</td>
<td>2</td>
<td>192.168.121.6</td>
</tr>
<tr>
<td>94</td>
<td>0.15%</td>
<td>1</td>
<td>2</td>
<td>192.168.121.16</td>
</tr>
<tr>
<td>33</td>
<td>0.06%</td>
<td>10</td>
<td>1</td>
<td>0.0.0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COUNT</th>
<th>% TOTAL</th>
<th>MSE</th>
<th>MST</th>
<th>SIGNATURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42.51%</td>
<td>6</td>
<td>1</td>
<td>UNITED STATES (us)</td>
</tr>
<tr>
<td>0</td>
<td>100.00%</td>
<td>2</td>
<td>1</td>
<td>UNITED KINGDOM (gb)</td>
</tr>
</tbody>
</table>
Links to browse to learn more about Security onion and Squert:


Salt-Stack with J-EDI
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
Basic Junos commands and template

- `salt vsrx junos.cli "show version"`

You can replace “show version” in the above command with Junos commands like “show interfaces”, “show configuration” and many other configuration & operational mode commands and get the results on Tools server.

- You can also configure the device from tools server using the “Junos set commands”. All you need to do is create a `commands.set` file and use it like this: `tools~# salt 'device_name' junos.install_config 'path to .set file'`
- A sample of the “.set” is present in `/srv/salt/push_policy.set` location on the tools server. This file is being used in a reactor (*described later in this doc*).
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:**

```
Salt-Master:

/srv/salt/extract_ip.sls

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

/srv/reac sor/reat to动力电池.sls

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

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

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

/srv/salt/push_policy.sls

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

vSRX
```

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

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

```bash
root@Tools:~# cat /etc/salt/master.d/reactor.conf
# Example reactor configurations below
reactor:
  - 'salt/engines/hook/gitlab':
    - /srv/reactor/gitlab_example.sls
    - /srv/reactor/proxy_inventory.sls
  - 'jnpr/syslog/Blue1_SRX/*':
    - /srv/reactor/react_to_attack.sls
```

The expression is matching the events published by vSRX on message bus to a `.sls` file. It is a regular expression used for matching and whenever an event is published starting with “jnpr/syslog/Blue_1SRX/” corresponding `.sls` file is triggered.
Salt: Files, their purpose and details

2. /srv/reactor/react_to_attack.sls

    root@Tools:~# cat /srv/reactor/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'] }}

hostip, daemon, and message are the attributes that belong to the message log corresponding to the event published on the message bus. Here we are assigning them to separate variables so that this data can be utilized in next scripts to react to event and push policies.

The event log is received in JSON format on the salt master bus. You can grab any data from it in this file using the “key” and later use that in your scripts.
Salt: Files, their purpose and details

3. /srv/salt/extract_ip.sls:

As we don’t want to push policy corresponding to all the events that we receive, this condition ‘UT_FLOW’ makes this reactor to push policy corresponding to UTM events only. Participants can put several other match criteria here matching to their event log, to push policy in response to specific events.
Salt: Files, their purpose and details

4. /srv/salt/push_policy.set

These are not the part of Junos syntax. These are template variables that we getting from the extract_ip.sls and using to push a policy corresponding to the IP address filtered from the log.
Other Useful resources to learn about Salt

**JUNOS salt module:**

[https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html](https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html)

**GitHub Example:**

Rules of engagement

You must not block legitimate customer traffic.

You must ensure that the following services are reachable from the internet.
- Ping, HTTP (80 & 8080), HTTPS, FTP

You can use IPS but due to customer SLA penalties and previous issues, you cannot use IPS to block inline.
Thank you!