

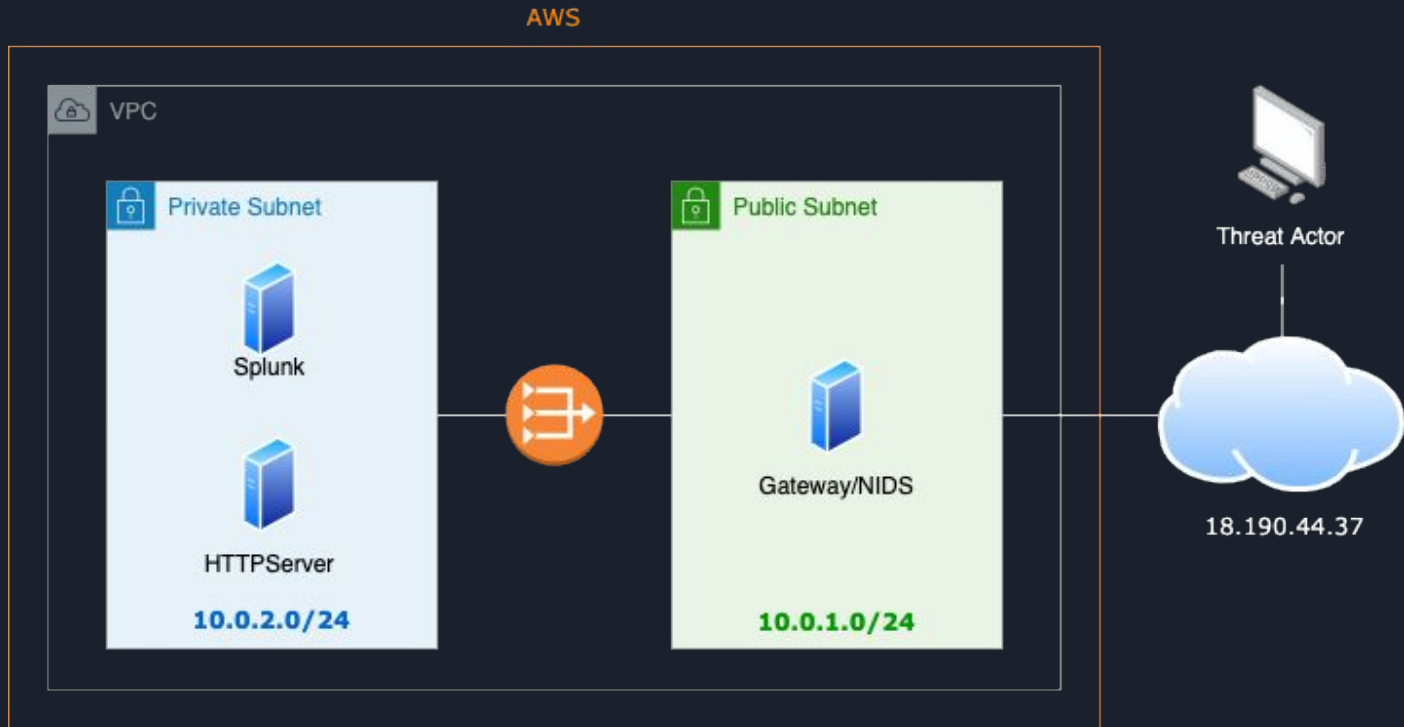


Cloud Network Intrusion Detection Simulator Presentation

CPSC 454
Fall 2021

Chris
Yu
Gage
Roberto Guerra
Louis

AWS Environment





AWS VPC Setup




On AWS (Amazon Web Service) we selected the option to create a new VPC (Virtual Private Cloud)

We then assigned the new VPC a non-routable IP range e.g. 10.0.0.0/16

Next we created a new internet gateway and attached it to the VPC

The internet gateway allows for communication between the VPC and the rest of the internet

VPC ID	State
 vpc-0940ffe6851e2d99b	 Available
Tenancy	DHCP options set
Default	dopt-54b1f53f
Default VPC	IPv4 CIDR
No	10.0.0.0/16

Internet gateway ID	State
 igw-057f778c70278a33a	 Attached
VPC ID	Owner
vpc-0940ffe6851e2d99b my-VPC	 691735480723



AWS VPC Setup - Public and Private Subnets

We created public and private subnets within the VPC

Then we assigned different IP ranges to each subnet, 10.0.1.0/24 for the public subnet and 10.0.2.0/24 for the private subnet

The public subnet connects to the internet gateway while the private subnet does not

Name ▾	Subnet ID ▾	State ▾	VPC ▾	IPv4 CIDR
my-Private1	subnet-042f3b5d2092bf53c	✔ Available	vpc-0940ffe6851e2d99b my-...	10.0.2.0/24
my-Public1	subnet-0ebeed31a3984580c	✔ Available	vpc-0940ffe6851e2d99b my-...	10.0.1.0/24

AWS VPC Setup - Route Tables

We created a route table that has the internet gateway as a target and 0.0.0.0/0 as a destination, then associated this route table with the public subnet

<input type="checkbox"/>	Name	Route table ID	Explicit subnet associat...
<input checked="" type="checkbox"/>	public-RT	rtb-0bb89e4cb66057ce1	subnet-0ebeed31a3984...
<input type="checkbox"/>	private-RT	rtb-0719b18892cce4800	subnet-042f3b5d2092b...

rtb-0bb89e4cb66057ce1 / public-RT

Details Routes Subnet associations Edge associations Route propagation

Routes (2)

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-057f778c70278a33a

We also created another route table without the igw (Internet Gateway) as a target and associated this table with the private subnet

<input type="checkbox"/>	Name	Route table ID	Explicit subnet associat...
<input type="checkbox"/>	public-RT	rtb-0bb89e4cb66057ce1	subnet-0ebeed31a3984...
<input checked="" type="checkbox"/>	private-RT	rtb-0719b18892cce4800	subnet-042f3b5d2092b...

rtb-0719b18892cce4800 / private-RT

Details Routes Subnet associations Edge associations Route propagation

Routes (2)

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	eni-04433d5fcc280f658

AWS VPC Setup - Security Groups

We specified allowed inbound and outbound traffic to and from the public subnet instance

Details

Security group name default	Security group ID sg-008ab3b127b75bfb6	Description default VPC security group	VPC ID vpc-0940ffe6851e2d9
--------------------------------	---	---	-------------------------------

Inbound rules (2) Manage tags Edit inbound rules

Filter security group rules

Type	Protocol	Port range	Source
All traffic	All	All	sg-008ab3b127b75bfb6
All traffic	All	All	0.0.0.0/0

Outbound rules (1/1) Manage tags Edit outbound rules

Filter security group rules

Type	Protocol	Port range	Destination
All traffic	All	All	0.0.0.0/0

We also specified allowed inbound and outbound traffic to and from private subnet instance

Security group name launch-wizard-4	Security group ID sg-09c7ce225a0dca900	Description launch-wizard-4 created 2021-10-19T23:58:58.656-07:00	VPC ID vpc-0940ffe6851e2d9
--	---	--	-------------------------------

Inbound rules (1/1) Manage tags Edit inbound rules

Filter security group rules

Type	Protocol	Port range	Source
SSH	TCP	22	0.0.0.0/0

Outbound rules (1/1) Manage tags Edit outbound rules

Filter security group rules

Type	Protocol	Port range	Destination
All traffic	All	All	0.0.0.0/0

AWS VPC Setup - EC2 Instance

We created an EC2 instance for the public subnet

Instance ID i-087447695cde9aa0f (CPSC454-NIDS)	Public IPv4 address 18.190.44.37 open address	Private IPv4 addresses 10.0.1.51
IPv6 address -	Instance state Running	Public IPv4 DNS -
Private IPv4 DNS ip-10-0-1-51.us-east-2.compute.internal	Instance type t2.micro	Elastic IP addresses 18.190.44.37 [Public IP]
VPC ID vpc-0940ffe6851e2d99b (my-VPC)	Security details	
Subnet ID subnet-0ebeed31a3984580c (my-Public1)	IAM Role -	
	Security groups sg-008ab3b127b75bfb6 (default)	

We also created an EC2 instance for the private subnet

Instance ID i-0e3c29eb28af485bf (CPSC454-Private1)	Public IPv4 address -	Private IPv4 addresses 10.0.2.17
IPv6 address -	Instance state Stopped	Public IPv4 DNS -
Private IPv4 DNS ip-10-0-2-17.us-east-2.compute.internal	Instance type t2.micro	Elastic IP addresses -
VPC ID vpc-0940ffe6851e2d99b (my-VPC)	Security details	
Subnet ID subnet-042f3b5d2092bf53c (my-Private1)	IAM Role -	
	Security groups sg-09c7ce225a0dca900 (launch-wizard-4)	



AWS VPC Setup - NIDS (Snort)

In the ECS2 instance for the public subnet we made a temp directory

```
mkdir ~/snort_src && cd ~/snort_src
```

Then we downloaded and installed the Data Acquisition library (DAQ) used by Snort

Finally, we downloaded and installed Snort

```
wget https://www.snort.org/downloads/snort/snort-2.9.16.tar.gz
```

```
./configure --enable-sourcefire && make && sudo make install
```


Snort Configuration

We modified the snort.conf file with: **sudo**

snort -T -c /etc/snort/snort.conf

We then validated by testing our

configuration as shown in Fig 5: **sudo snort**

-T -c /etc/snort/snort.conf

```
-- Initialization Complete --

-*> Snort! <*-
Version 2.9.10.1 GDE (Build 1005)
By Martin Roesch & The Snort Team: http://www.snort.org/contact@team
Copyright (C) 2014-2021 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.9.1 (with TPACKET_V3)
Using PCRE version: 8.39 2016-06-14
Using ZLIB version: 1.2.11

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 3.2 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_OAF Version 1.1 <Build 1>
Preprocessor Object: SF_S7COPPPPLUS Version 1.0 <Build 1>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_MQOOLUS Version 1.1 <Build 1>
Preprocessor Object: SF_SOF Version 1.1 <Build 1>
Preprocessor Object: SF_DCEMPC2 Version 1.0 <Build 1>
Preprocessor Object: apid Version 1.1 <Build 5>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_INAP Version 1.0 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_SMLPP Version 1.1 <Build 4>

Total snort Fixed Memory Cost - MaxRss:56312
Snort successfully validated the configuration!
Snort exiting
```



Snort Rules

```
alert tcp any any -> any 22 (msg:"SSH detected"; threshold: type both, track by_src, count 5, seconds 30; sid:001; rev:001;)
alert tcp any any -> any 23 (msg:"Possible NMAP scan detected";threshold: type both, track by_src, count 1, seconds 30; sid:002; rev:001;)
alert tcp any any -> any 22 (msg:"SSH Brute Force Detected"; threshold: type threshold, track by_src, count 30, seconds 5; sid:003; rev:001;)
alert tcp any any -> any 8000 (msg:"Splunk Access"; threshold: type both, track by_src, count 1, seconds 60; sid:004; rev:001;)
alert icmp any any -> any 8080 (msg:"ICMP flood"; sid:005; rev:1; classtype:icmp-event; threshold: type both, track by_dst, count 500, seconds 3;)
alert tcp any any -> any 8080 (msg:"Possible DoS Attack Type: SYN flood"; flow:stateless; sid:006; threshold: type both, track by_dst, count 20, seconds 10;)
```

Snort utilizes a series of rules that help define malicious network activity and uses those rules to find packets that match against them and generates alerts for users.

<action> <protocol> <source ip> <source port> -> <destination ip> <destination port> (rule options)



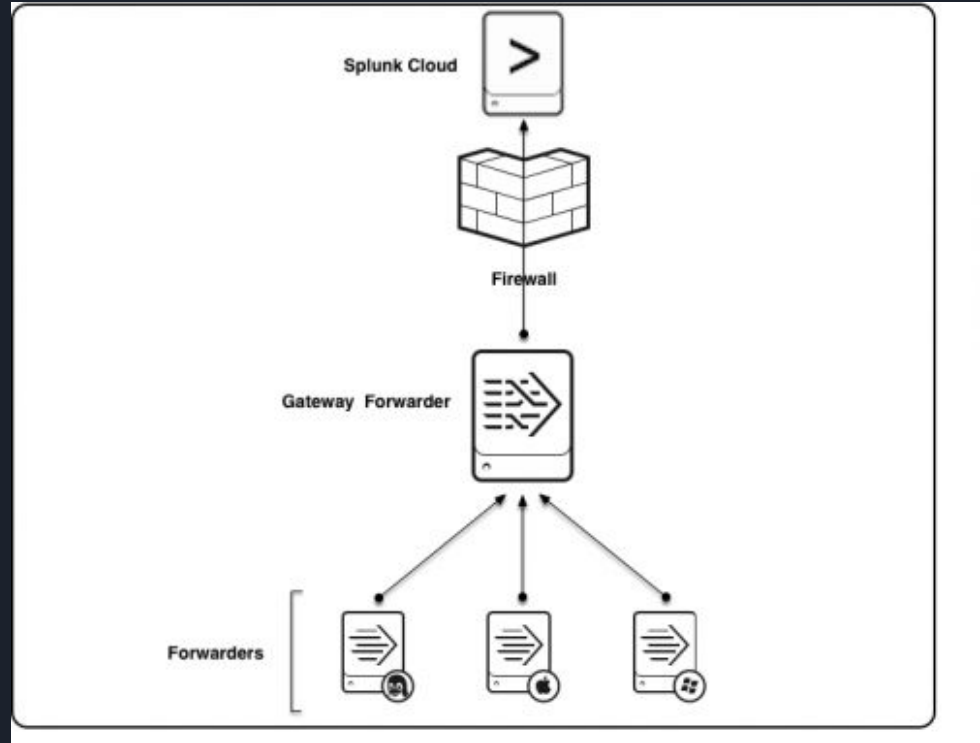
Splunk (SIEM)

We have included a Security Information and Event Management system, called *Splunk*, to analyze traffic in and out of the cloud. Splunk is also able to capture, index, and correlate real time data into a searchable container.

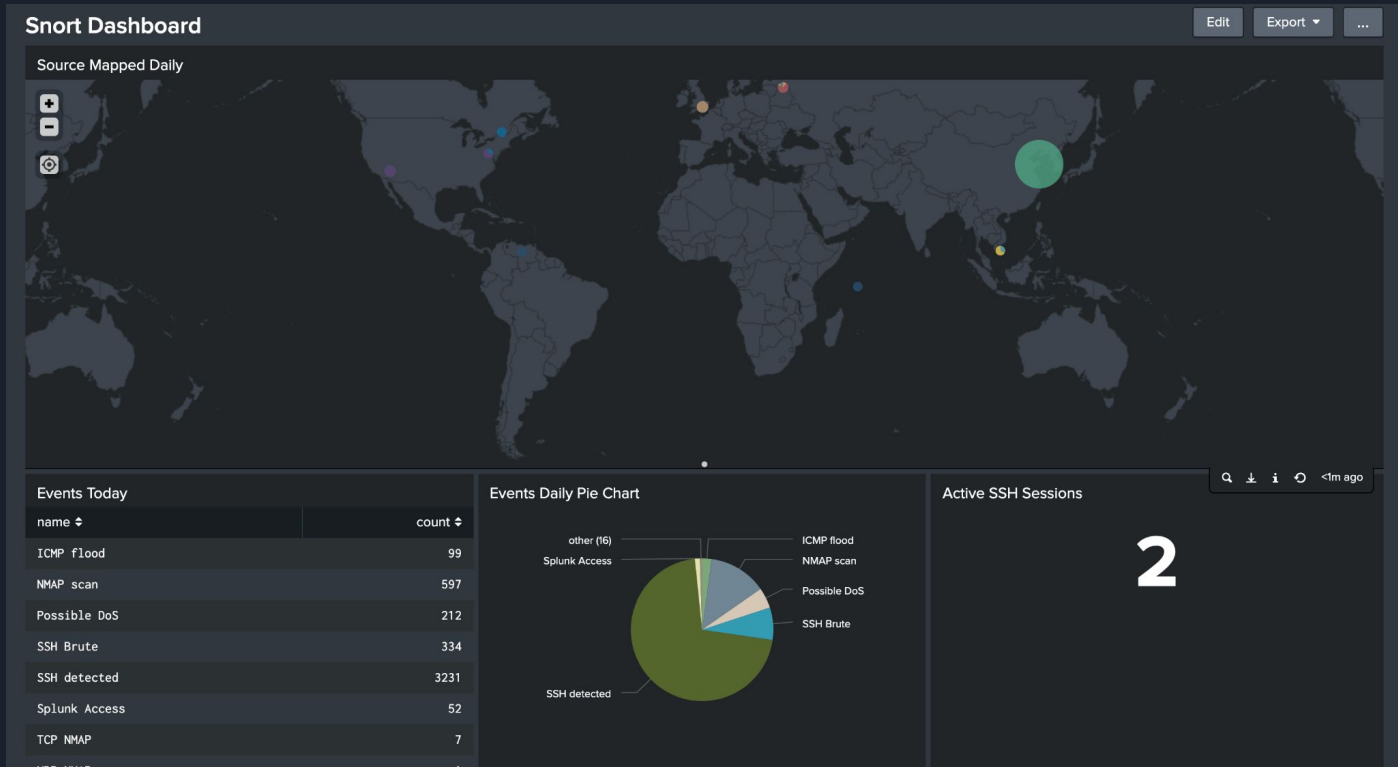


Splunk (cont.)

We have also implemented a Splunk Forwarder, which can forward Snort log files to a port of our Splunk instance and create dashboards on Splunk.



SIEM Dashboard



SIEM Dashboard (cont.)

Time	Event
12/6/21 11:31:49.862 PM	<p>[**] [1:1:1] SSH detected [**] [Priority: 0] 12/06-23:31:49.862105 218.92.0.204:29607 -> 10.0.1.51:22 TCP TTL:30 TOS:0x0 ID:18617 IpLen:20 DgmLen:116 DF ***AP*** Seq: 0xAC329FF2 Ack: 0xF6B87E1C Win: 0x106 TcpLen: 32 Show all 6 lines host = ip-10-0-1-51 source = /var/log/snort/alert.full sourcetype = snort_alert_full</p>

indexed snort logs on Splunk

Search String

```
index="snort" source="/var/log/snort/alert.full" "SSH detected" | dedup src_ip |  
stats count by src_ip | eventstats sum(count) as total | table total
```

creating unique searches

Active SSH Sessions

2

dashboard display of SSH sessions

Intrusion Detection Simulations

Our project includes scripts that contains various network tools to detect and prevent the following attacks:

- SSH Brute Force attacks (Hydra)
- DDoS attacks (hping3)
 - ICMP Flooding
 - SYN Flooding
- Nmap Scan Detection
 - TCP Scan
 - UDP Scan

```
import time, subprocess

while True:

    #TCP NMAP scan
    a = subprocess.Popen('nmap -sC -sV -Pn 18.190.44.37', shell=True)
    time.sleep(10)

    #UDP NMAP scan
    b = subprocess.Popen('nmap -sU 18.190.44.37', shell=True)
    time.sleep(10)

    #SSH Brute Force
    c = subprocess.Popen('hydra -l user -P /usr/share/wordlists/dirb/others/best110.txt ssh://18.190.44.37', shell=True)
    time.sleep(10)

    time.sleep(600)
```

script running nmap scans and hydra in a loop



Intrusion Detection Simulations (cont.)

```
import os
import signal
import time
import subprocess

while True:
    #ICMP Flooding
    icm = subprocess.Popen("hping3 -1 -p 8080 --flood 18.190.44.37", shell=True)
    time.sleep(1)

    #SYN Flooding
    syn = subprocess.Popen("hping3 -S --flood -p 8080 18.190.44.37", shell=True)
    time.sleep(1)

    os.killpg(os.getpgid(icm.pid), signal.SIGTERM)
    os.killpg(os.getpgid(syn.pid), signal.SIGTERM)
```

script running hping3 commands on demand