Toronto Metropolitan University

[1]

Cyber Security

CAPSTONE PROJECT

Cyber Threat Prediction Modeling

Credits

- Author: Austin Hogan
- Professor: William Zereneh
- Employer: BlackBerry - Cyber Security Systems
- Education: Toronto Metropolitan University – The Chang School & Centennial College

Project Outline

Title:	Cyber Threat Prediction Modeling
Description:	A Data Science Investigation of Honeypot Threat Data; carrying Three Objectives: Threat Prediction (Machine Learning), Enhanced Honeypot Operation (Reinforcement Learning), and Threat Actor Detection (Pattern Recognition and Classification)
Author:	Austin Hogan
Professor:	William Zereneh
Started:	Sat May 7 23:19:46 EDT 2022
Ended:	Sat July 23 10:46:42 EDT 2022
Program:	Cybersecurity, Data Protection and Digital Forensics
Topic:	Analysis of (New) Collected Honeypot Attack Data
Language:	English
Keywords:	Machine Learning, Honeypot, Reinforcement Learning, Pattern Recognition and Classification, Artificial Intelligence Algorithms

Copyright

[2]

This work is subject to Attribute-Non-Commercial-No-Derivates 4.0 International

Declaration of Authenticity

Student ID: 501145823
Student Name – Austin Hogan
Course: Certificate of Cybersecurity, Data Protection and Digital Forensics
Title of Capstone Project: Cyber Threat Prediction Modeling

I hereby declare that I am the legitimate copyright holder and author of this capstone project and that this work is authenticate and original. No portion of these works were submitted at cause of other qualifications or at any other place of learning. I hold Toronto Metropolitan University with consent to uphold this copyright and distribute said works according to the previous shown copyright policy.

Austin Hogan

Quote

“On the one hand information wants to be expensive, because it’s so valuable. The right information in the right place just changes your life. On the other hand, information wants to be free, because the cost of getting it out is getting lower and lower all the time. So, you have these two fighting against each other.”

- Stewart Brand, November 1984

(fully quoted due to misuse of this quote)

Acknowledgements

For my family, friends, colleagues, professors and others who have helped me along my journey in life.

Abstract Part 1

As Evan Francen succinctly put it, “The information security industry is broken because we don’t speak the same language.” [4] In his book *Unsecurity*, he discusses several of the major issues ailing the cyber security industry and how to fix them. While most of these issues are human issues, this capstone projects requires a technical application be made. So, despite all the differences among cybersecurity practitioners, he cites common threads which bind most to the practice. Evan Francen’s textbook answer to the question, “What is cybersecurity?” is this, “Information security is managing risks to the confidentiality, integrity, and availability of information using administrative, physical and technical controls.” [4] With those two pillars in mind, what is cybersecurity and how can cybersecurity practitioners improve communications within the field, this project intends to enhance the knowledge within cyber security using rigorous use of language, a summary of process, and how this project aims to help practitioners within our common definition of cybersecurity.

Abstract Part 2

Cybersecurity is an important field in our modern era, with human dependence on digital technology becoming ubiquitous. [5] Hence, protecting modern information transfer and storage mechanisms is a high priority of many countries, businesses, and end users. [5] Understanding how attackers develop their trade and deploy their technologies is a major objective of the Cyber Security Industry. Therein, we must find ways of gathering data on the latter; how attackers deploy their technology since how attackers develop their trade is generally for the social sciences. For large Cyber Security firms, they can often purchase data and also assess their own digital landscape to collect data on how attackers operate. Due to time and resource constraints, after considering options for gathering real-time data on malicious threat actor activity, the most effective option was single/multi node honeypot clusters. Such clusters are performant and modifiable for the data, development process, and results required.

Abstract Part 2

Therefore, this project aims to do study a portion of the threat landscape on the public internet for a brief period utilizing a less expensive, yet effective method, honeypots. This project holds several objectives. The essential objective is predictive threat modeling. The additional objectives involve utilizing further reinforcement learning, enhancing reinforcement learning with predictive threat modeling results, and developing pattern recognition to discover botnet activity. By using discoveries from threat predictions, reinforcement learning, and pattern recognition this project attends to understanding future threats, enhancing the threat data source, and detecting similar attacks leading to likely same-source attackers, sometimes called botnets. In summary, this project addresses several machine learning problem spaces within the field of Cyber Security, utilizing honeypot technology.

CONTENTS

CONTENTS

List of Tables

Table	Purpose
Table of Figures	Project Figures
Table of Scripts	Project Scripts
Table of Acronyms	Project Acronyms

Table of Figures

Table of Scripts

Script	Language	Purpose
1	bash	install tpot hive
2	bash	install tpot hive sensor
3	bash	Install tpot hive sensor part2
4	config file	environment file
5	python	github repo Hidden Markov Model

Table of Acronyms

Acronym	Expanded
TOS	Terms of Service
HMM	Hidden Markov Model
CE	Community Edition
SSH	Secure Shell Protocol
JSON	JavaScript Object Notation
TELNET	Teletype Network Protocol

Scr. 1 [install-tpot-hive.sh]

```
#!/bin/bash

source .env

useradd -s /bin/bash -m -p $(echo "$user_pass" | openssl passwd -1 -stdin) tsec
usermod -aG sudo tsec
apt install git ufw –y
git clone https://github.com/telekom-security/tpotce
cd tpotce/iso/installer/cat > tpot.conf << EOF
myCONF_TPOT_FLAVOR='HIVE’
myCONF_WEB_USER='$web_user’
myCONF_WEB_PW='$web_pass’
EOF

./install.sh --type=auto --conf=tpot.conf
ufw allow 64295/tcp

reboot
```

Scr. 2 [install-tpot-hive-sensor.sh]

```
#!/bin/bash
source .env
useradd -s /bin/bash -m -p $(echo "$user_pass" | openssl passwd -1 -stdin) tsec
usermod -aG sudo tsec
apt install git ufw –y
git clone https://github.com/telekom-security/tpotce
cd tpotce/iso/installer/
cat > tpot.conf << EOF
myCONF_TPOT_FLAVOR='HIVE_SENSOR’
myCONF_WEB_USER='$web_user’
myCONF_WEB_PW='$web_pass’
EOF
./install.sh --type=auto --conf=tpot.conf
port_tcp="80 64295 443 5555 8443 443 102 502 1025 2404 10001 44818 47808 50100 22 23 11112 21 42 135 1433 1723 1883 3306
8081 9200 25 110 143 993 995 1080 5432 5900 389 1433 1521 5432 6379 8080 9200 11211 631 9200 2575 5060”
for val in $port_tcp; do
    ufw allow ${val}/tcp
done
port_udp="5000 161 623 19 53 123 1900 69 5060”
for val in $port_udp; do
    ufw allow ${val}/udp
done
reboot
```

Scr. 3 [install-tpot-hive-sensor-part2.sh]

```
#!/bin/bash
rm .env
dps.sh
deploy.sh
```

Scr. 4 [.env]

```
web_user=
user_pass=
web_pass=
```

Introduction

In today's highly connected virtual world, every new device that comes online, creates a world in which human interdependence with electronics is ever increasing. In an ideal world with ideal users these machines would operate in perfect harmony with each other and with us as humans. However, such a world does not exist, so people in the digital information industries, such as cyber security and software development, are left to resolve the inherent issues with imperfect people using and building imperfect systems.

A current phenomena with the advent of the internet are individuals or their systems that attack and manipulate the digital infrastructure behind the virtual space humanity has accumulated over recent decades. The reasons social, economic, or geo-political are beyond the scope of this project. Often called hackers, they attempt to penetrate systems and exfiltrate data or make modifications that benefit the attacker almost always to the detriment of a system or system owner. One exemplary technique for identifying and managing the activities of hackers is utilizing honeypots. In this way, methods and techniques employed by hackers can be monitored, understood and classified. Also, and ideally, these observations and classifications can be prolonged and grown if the attacker, as a bot or human are unaware the penetrated system is a honeypot. Ideally the hacker will be trapped for hours collecting falsified information, studying falsified infrastructure, and falsified network traffic from a secure & monitored honeypot system.

Introduction: Purpose, Significance, Methods

Since this project has limited time and resources, the purpose will be limited to sampling a Honeypot for Markov Modeling and possibly extending out the research to see how reinforcement learning affects the models produced. Furthermore, if time and resources grant, this project aims to detect similar attackers by using multiple honeypots and controlled reinforcement learning.

The realistic aim of this project is to develop simple Markov Models of the problem space using the Cowrie Honeypot of the T-Pot Honeypot system. Since the Cowrie Honeypot classifies every attack and each attack's sequence of events using a simple action event.

To be fair and honest, similar research has been done before me, with similar end goals. Most often the unique attributes were the researcher creating their own Markov Modeling Scripts [7], using the entire Honeypot system as a study on Honeypots, or extending the project into new domains such reinforcement learning [6] or botnet identification [8]. Unfortunately, due to time constraints and resource constraints, I will be unable to attend to any significant new findings. The uniqueness of my research will rely on the fact that the honeypot data is relatively new.

My methods are rather traditional, using my own custom deployment scripts and instructions for deploying T-Pot, using common machine learning libraries written in Python for data processing and studying the outcomes of this project's research.

Introduction: Results and Conclusions

Due to the limitation of time and the Terms Of Service (TOS) of the free cloud provider trial account used in this project, my original deployment had to be backed up and shutdown within 48 hours of the notice that TOS had been broken.

This limited the ability to study reinforcement learning and botnet identification as previously planned and discussed.

Therefore, this project will focus on processing the recently collected data for the Modeling portion of this project. This results will be presented in the form of Markov Modeling diagrams and Python scripts to develop conclusions about the collected data.

Literature Review

Several papers and reports have been done on the taxonomy of honeypots in the realm of cyber security. Likewise, several papers have been written on the usage of machine learning techniques on honey pots.

Please note, due to the honey pots breaking the TOS of the chosen cloud provider Vultr, the non-primary introduction goals will not be completed.

Literature Review: Taxonomy

Server: Honeypot with a public IP address collecting data from a central branch of the internet.
Client: Honeypot with a private IP address waiting for an attacker to enter a private network.
Hybrid: Honeypot which makes use of Server and Client Honeypots to make the best use of each type.

Research vs. Production: Research honeypots are often temporary or run by educational institutions to collect data, whereas production honeypots are run in production environments to protect production systems.

Low, Medium, and High Interaction: How realistic and interactive the honeypot environment is to an attacker. High interaction comes with more risk of real attacks, yet low interaction honeypots may be detected by attackers

Virtual: Fake services, and services run on virtual resources.
Physical: Real hardware, networks, and resources.

[9]

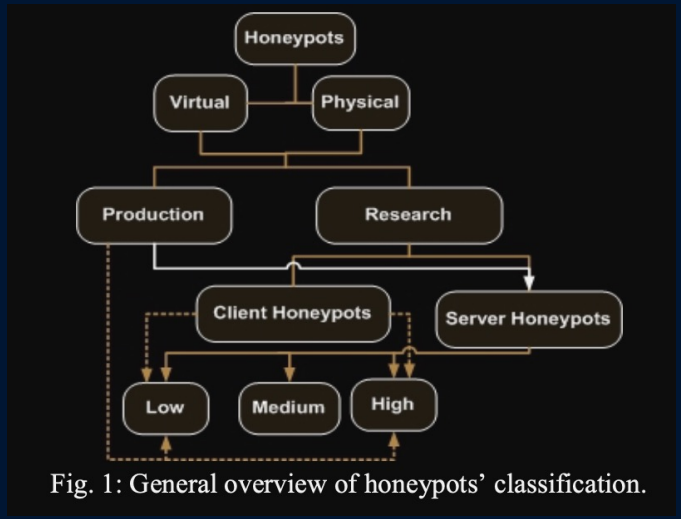


Fig. 1: General overview of honeypots' classification.

[9]

Literature Review: Taxonomy

A very detailed explanation of honeypots and their application in modern cyber security systems is described by (Maesschalck et al., 2021). Therein a discussion of the pros and cons of risk, cost, and collection procedures for various honeypot systems is formulated. Some characteristics of interaction levels are displayed in the figure. [10]

Risk: The risk of causing damage with the honeypot.
Data Capturing: The amount of information gathered.
Resource Usage: Number of resources required.
Simulation: Closeness of devices, services, or hardware.
Required Knowledge: The knowledge required to operate.
Detection: Amount of difficulty in detecting the honeypot.
Cost: Expenses to operate; manpower and resources.

Table 1. Overview of Honeypot Characteristics.

Level of Interaction	Low	Medium	High
Risk	Low	Low	High
Data Capturing	Basic	Intermediate	Comprehensive
Resource Usage	Low	Low	High
Simulation	Basic	Improved	N/A
Required Knowledge	Low	Advanced	High
Detection	Easy	Easy - Normal	Normal - Hard
Cost	Low	Low	High

[10]

Problem Definition

This project will assess regional differences between various honeypot sensors.

Each honeypot sensor, denoted “src” or source is listed.

The honeypot hive denoted “sink” or sink is the shown also.

Using Markov Modeling it will be discovered which honeypot sensor creates the best Markov Models and compare some of the data collected by region/sensor and their graphs.

```
# sink
hive_base.city = "Toronto"
hive_base.country = "Canada"
hive_base.code_name = "realtatami"
hive_base.ip = "99.251.10.123"
# src 0
hive_sensors[0].city = "Toronto"
hive_sensors[0].country = "Canada"
hive_sensors[0].code_name = "rulingcommittee"
hive_sensors[0].ip = "216.238.83.8"
# src 1
hive_sensors[1].city = "Seoul"
hive_sensors[1].country = "Korea"
hive_sensors[1].code_name = "attractivescow"
hive_sensors[1].ip = "158.247.238.143"
# src 2
hive_sensors[2].city = "Melbourne"
hive_sensors[2].country = "Austrailia"
hive_sensors[2].code_name = "deadinlay"
hive_sensors[2].ip = "67.219.98.203"
# src 3
hive_sensors[3].city = "Stockholm"
hive_sensors[3].country = "Sweden"
hive_sensors[3].code_name = "longtermresult"
hive_sensors[3].ip = "65.20.115.181"
# src 4
hive_sensors[4].city = "Mumbai"
hive_sensors[4].country = "India"
hive_sensors[4].code_name = "sadproduce"
hive_sensors[4].ip = "65.20.68.106"
# src 5
hive_sensors[5].city = "Los Angeles"
hive_sensors[5].country = "United States"
hive_sensors[5].code_name = "resonantnorse"
hive_sensors[5].ip = "149.248.9.4"
```

Fig 1. [honeypot regions]

Research Methodology

By assessing regional differences and similarities between honeypot sensors and their respective Markov Models, this research intends to assess the ability of predicting future attacks by region.

Using Scr. 1 [install-tpot-hive.sh] a honeypot hive will be installed on a server to collect data streams from various honeypot hive sensors as a sink, and Scr. 2 [install-tpot-hive-sensor.sh] will be installed on six servers to collect data from various attackers and forward the data to the honeypot hive as regional sources. T-Pot CE Honeypot makes use of SSH tunneling to complete this data transfer task continuously, after the initial install and setup.

Scr. 3 [install-tpot-hive-sensor-part2.sh] will be run after the initial install completes on each of the six servers to make sure the hive sensor is running correctly.

The 6 honeypots were active from approximately May 27th 2022 at 11am until June 1st 2022 at 11am.

Research Methodology
Data Collection

This honey pot would be considered a low interaction honeypot, a virtual research server as the individual honeypots run inside Docker and the interactions are limited to software versions of real tools.

A good example is the Cowrie honeypot which mimics SSH interaction and others.

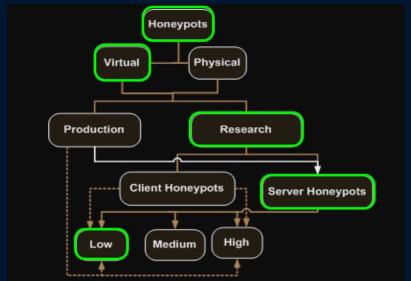


Fig. 1: General overview of honeypots' classification.

[9]

Table 1. Overview of Honeypot Characteristics.

Level of Interaction	Low	Medium	High
Risk	Low	Low	High
Data Capturing	Basic	Intermediate	Comprehensive
Resource Usage	Low	Low	High
Simulation	Basic	Improved	N/A
Required Knowledge	Low	Advanced	High
Detection	Easy	Easy - Normal	Normal - Hard
Cost	Low	Low	High

[10]

Research Methodology
Data Analysis

T-Pot CE honeypot collects several data streams from several different honeypots in a low-medium interaction configuration.
The data will be divided regionally by the location of the Honey Pot Sensor.
The data will be scored using hmmlearn [11] python library for Markov Models.

The below excerpt is from the cowrie honeypot logs timestamped at, May 31, 2022 @ 19:59:59.647 that gives some insight to how the data is made and collected.

This informational example is exhaustive so to share with readers potential information that could be used from T-Pot CE for future attack predictive analysis.

However, following the exhaustive JSON data excerpt, a description of the essential requirements for creating Hidden Markov Models is made.

Research Methodology
Data Analysis


```
{
  "_index": "logstash-2022.05.31",
  "_id": "NqiPHIEBWDV5yY3E4GiK",
  "_version": 1,
  "_score": null,
  "fields": {
    "geoip_ext.country_name.keyword": [
      "United States"
    ],
    "geoip_ext.region_code": [
      "CA"
    ],
    "geoip.region_name.keyword": [
      "Tocantins"
    ],
    "geoip.country_code2.keyword": [
      "BR"
    ],
    "geoip_ext.country_code2.keyword": [
      "US"
    ],
    "eventid.keyword": [
      "cowrie.session.connect"
    ],
    "type": [
      "Cowrie"
    ],
    "path": [
      "/data/cowrie/log/cowrie.json"
    ],
    "headers.accept_encoding.keyword": [
      "gzip,deflate"
    ],
    "sensor.keyword": [
      "62185d46eced"
    ],
    "geoip_ext.continent_code.keyword": [
      "NA"
    ],
    "geoip.region_code.keyword": [
      "TO"
    ],
    "headers.content_length.keyword": [
      "5832"
    ],
    "geoip_ext.as_org": [
      "Choopa, LLC"
    ],
    "headers.http_version": [
      "HTTP/1.1"
    ],
    "geoip.city_name.keyword": [
      "Peixe"
    ],
    "session.keyword": [
      "85027aebbe03"
    ],
    "headers.http_version.keyword": [
      "HTTP/1.1"
    ],
    "geoip_ext.city_name": [
      "Los Angeles"
    ],
```

Research Methodology
Data Analysis


```
"t-pot_ip_int.keyword": [
  "149.248.9.4"
],
"headers.content_type": [
  "application/json"
],
"t-pot_ip_ext.keyword": [
  "149.248.9.4"
],
"dest_ip": [
  "172.25.0.2"
],
"geoip.latitude": [
  -12
],
"geoip_ext.ip": [
  "149.248.9.4"
],
"geoip_ext.location": [
  {
    "coordinates": [
      -118.2606,
      34.0729
    ],
    "type": "Point"
  }
],
"geoip.postal_code.keyword": [
  "77460"
],
"geoip.region_code": [
  "TO"
],
"geoip.as_org": [
  "Raque alves da Siva Me"
],
"geoip.ip": [
  "170.83.157.5"
],
"headers.http_host.keyword": [
  "127.0.0.1:64305"
],
"t-pot_ip_ext": [
  "149.248.9.4"
],
"geoip_ext.country_code3.keyword": [
  "US"
],
"headers.http_host": [
  "127.0.0.1:64305"
],
"geoip.country_name": [
  "Brazil"
],
"timestamp": [
  "2022-05-31T23:59:59.647Z"
],
"geoip.asn": [
  266483
],
```

Research Methodology
Data Analysis


```
"headers.connection.keyword": [
  "Keep-Alive"
],
"geoip.city_name": [
  "Peixe"
],
"geoip_ext.timezone.keyword": [
  "America/Los_Angeles"
],
"geoip.as_org.keyword": [
  "Raque alves da Siva Me"
],
"headers.accept_encoding": [
  "gzip,deflate"
],
"t-pot_hostname.keyword": [
  "resonantnorse"
],
"geoip_ext.postal_code": [
  "90012"
],
"@timestamp": [
  "2022-05-31T23:59:59.647Z"
],
"headers.request_path.keyword": [
  "/"
],
"geoip_ext.timezone": [
  "America/Los_Angeles"
],
"path.keyword": [
  "/data/cowrie/log/cowrie.json"
],
"headers.connection": [
  "Keep-Alive"
],
"geoip.timezone": [
  "America/Araguaina"
],
"geoip_ext.asn": [
  20473
],
"geoip_ext.longitude": [
  -118.25
],
"t-pot_ip_int": [
  "149.248.9.4"
],
"headers.request_path": [
  "/"
],
"geoip.country_name.keyword": [
  "Brazil"
],
"geoip_ext.region_name.keyword": [
  "California"
],
"protocol": [
  "ssh"
],
"type.keyword": [
  "Cowrie"
],
```

Research Methodology
Data Analysis


```
"host": [
  "172.18.0.1"
],
"geoip.longitude": [
  -48.53125
],
"host.keyword": [
  "172.18.0.1"
],
"dest_port": [
  22
],
"geoip_ext.region_code.keyword": [
  "CA"
],
"t-pot_hostname": [
  "resonantnorse"
],
"geoip_ext.country_name": [
  "United States"
],
"geoip_ext.region_name": [
  "California"
],
"geoip.region_name": [
  "Tocantins"
],
"@version.keyword": [
  "1"
],
"geoip.continent_code.keyword": [
  "SA"
],
"geoip_ext.continent_code": [
  "NA"
],
"dest_ip.keyword": [
  "172.25.0.2"
],
"headers.content_type.keyword": [
  "application/json"
],
"geoip_ext.postal_code.keyword": [
  "90012"
],
"sensor": [
  "62185d46eced"
],
"protocol.keyword": [
  "ssh"
],
"geoip.continent_code": [
  "SA"
],
"eventid": [
  "cowrie.session.connect"
],
"session": [
  "85027aebbe03"
],
"geoip.country_code3.keyword": [
  "BR"
],
```

Research Methodology
Data Analysis


```
"geoip_ext.dma_code": [
  803
],
"geoip_ext.country_code3": [
  "US"
],
"geoip_ext.country_code2": [
  "US"
],
"src_ip": [
  "170.83.157.5"
],
"geoip.country_code3": [
  "BR"
],
"geoip.location": [
  {
    "coordinates": [
      -48.5328,
      -11.9961
    ],
    "type": "Point"
  }
],
"geoip.country_code2": [
  "BR"
],
"@version": [
  "1"
],
"src_ip.keyword": [
  "170.83.157.5"
],
"geoip_ext.latitude": [
  34.0625
],
"headers.http_user_agent.keyword": [
  "Manticore 0.8.0"
],
"message": [
  "New connection: 170.83.157.5:59812 (
172.25.0.2:22) [session: 85027aebbe03]"
],
"geoip_ext.as_org.keyword": [
  "Choopa, LLC"
],
"headers.http_user_agent": [
  "Manticore 0.8.0"
],
"geoip.postal_code": [
  "77460"
],
"src_port": [
  59812
],
"headers.content_length": [
  "5832"
],
```

Research Methodology
Data Analysis


```
  "highlight": {
    "type.keyword": [
      "@kibana-highlighted-field@Cowrie@/kibana-highlighted-field@"
    ]
  },
  "sort": [
    1654041599647
  ]
}
```


As you can see from the previous example log output, the data collection for cowrie and T-Pot CE is extremely verbose and rich with datapoints.
To make use of the data however, one only needs the group honeypots by honeypot sensor region, then group by telnet/ssh session id, then sort by timestamp, and finally order sequences by “event id” for processing within a Markov Modeling tool.

Research Methodology
Data Analysis
Attacks By Source Region

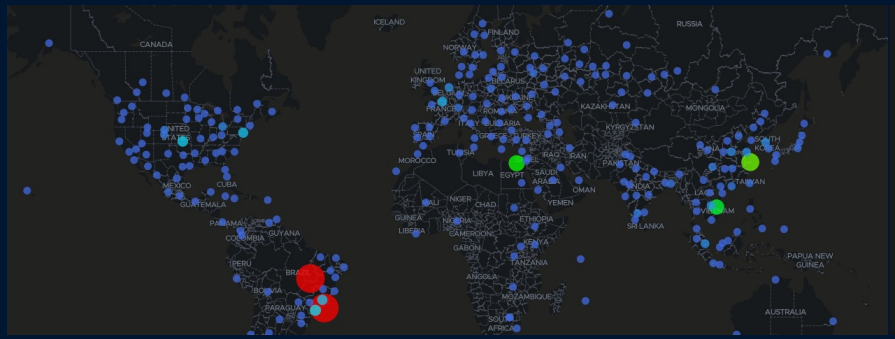


Fig 3. [kibana – by country - map]

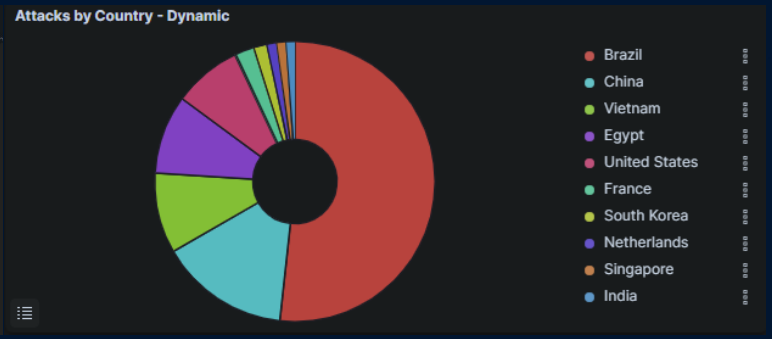


Fig 4. [kibana – by country - dynamic]

Research Methodology
Data Analysis
Attacks By Source Region

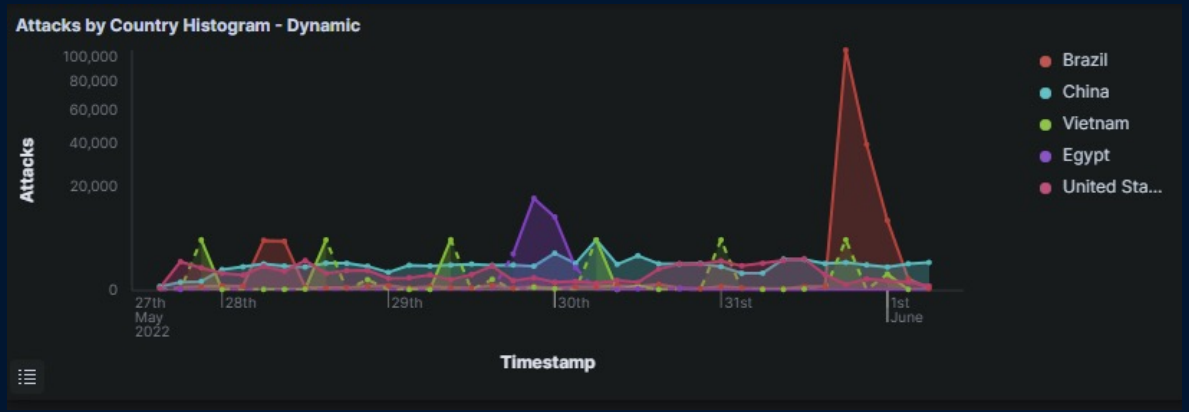


Fig 5. [kibana – by country - histogram]

Research Methodology
Data Analysis
Attacks By Source IP Reputation

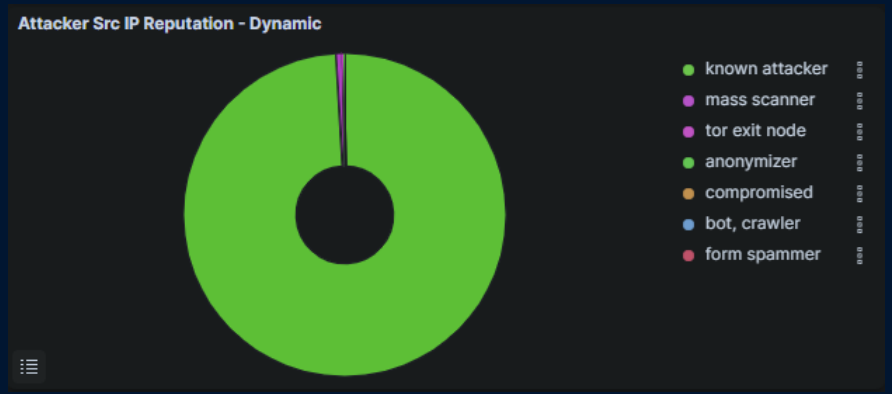


Fig 6. [kibana – by ip reputation]

Research Methodology
Data Analysis
Attacks by Command Input

Cowrie Input - Top 10

Command Line Input	Count
shell	752
system	582
sh	371
enable	286
while read i	250
dd bs=52 count=1 if=.s || cat .s || while read i; do echo $i; done < .s	249
rm .s; exit	227
ls /	85
linuxshell	72
config terminal	52

Fig 6. [kibana – by ip reputation]

Research Methodology
Data Analysis
Attacks by Destination Honeypot

As shown the most attacked honeypots in ordered was:

1. Stockholm, Sweden
2. Los Angeles, United States
3. Melbourne, Australia
4. Mumbai, India
5. Toronto, Canada
6. Seoul, Korea

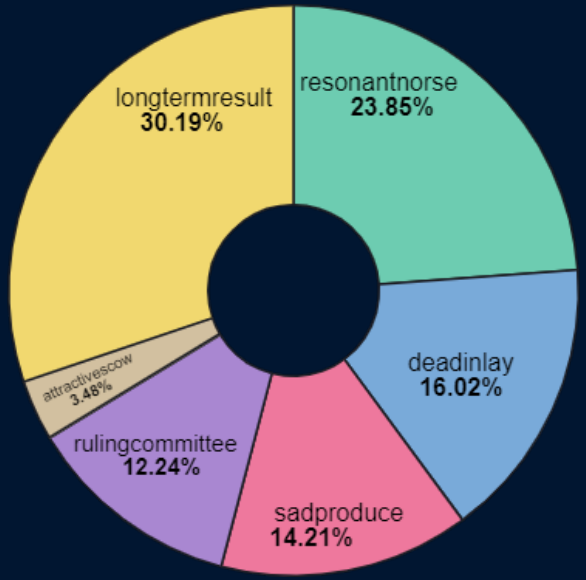


Fig 8. [kibana – by honeypot hostname]

```
# sink
hive_base.city = "Toronto"
hive_base.country = "Canada"
hive_base.code_name = "realtatami"
hive_base.ip = "99.251.10.123"
# src 0
hive_sensors[0].city = "Toronto"
hive_sensors[0].country = "Canada"
hive_sensors[0].code_name = "rulingcommittee"
hive_sensors[0].ip = "216.238.83.8"
# src 1
hive_sensors[1].city = "Seoul"
hive_sensors[1].country = "Korea"
hive_sensors[1].code_name = "attractivescow"
hive_sensors[1].ip = "158.247.238.143"
# src 2
hive_sensors[2].city = "Melbourne"
hive_sensors[2].country = "Austrailia"
hive_sensors[2].code_name = "deadinlay"
hive_sensors[2].ip = "67.219.98.203"
# src 3
hive_sensors[3].city = "Stockholm"
hive_sensors[3].country = "Sweden"
hive_sensors[3].code_name = "longtermresult"
hive_sensors[3].ip = "65.20.115.181"
# src 4
hive_sensors[4].city = "Mumbai"
hive_sensors[4].country = "India"
hive_sensors[4].code_name = "sadproduce"
hive_sensors[4].ip = "65.20.68.106"
# src 5
hive_sensors[5].city = "Los Angeles"
hive_sensors[5].country = "United States"
hive_sensors[5].code_name = "resonantnorse"
hive_sensors[5].ip = "149.248.9.4"
```

Fig 1. [honeypot regions]

Research Methodology
Data Analysis
Attacks Password Selection

Top 20 values of password.keyword	Count of records
123456	671
admin	485
password	371
12345	335
1234	301
root	250
(empty)	230
user	149
1111	118
888888	114
1111111	108
ubnt	106
123	100
666666	98
raspberry	96
54321	95
default	91
pass	90
7ujMko0admin	86
0	85
Other	24,764

Fig 9. [kibana – attack passwords]

Research Methodology
Cowrie Honeypot

Cowrie is a medium-low interaction SSH and Telnet honeypot. It offers many features and emulations of the actual system.

Cowrie honeypot allows for 18 possible states wherein Hidden Markov Models sequences can be derived.

Using hmmlearn a python library tool and similar hidden Markov modeling tools, the various regional state sequences will be graphed and scored.

Tools used will include: Visual Studio Code, jupyter-notebook, hmmlearn and others described later in greater detail.

```
Number of unique event ids 18
00 cowrie.client.kex
01 cowrie.client.size
02 cowrie.client.var
03 cowrie.client.version
04 cowrie.command.failed
05 cowrie.command.input
06 cowrie.command.success
07 cowrie.direct-tcpip.data
08 cowrie.direct-tcpip.request
09 cowrie.log.closed
10 cowrie.login.failed
11 cowrie.login.success
12 cowrie.session.closed
13 cowrie.session.connect
14 cowrie.session.file_download
15 cowrie.session.file_download.failed
16 cowrie.session.file_upload
17 cowrie.session.params
```

Fig 2. [cowrie event ids]

Experimentation

Using the state machine that is prebuilt into Cowrie honeypots on T-Pot CE Honeypot we can easily derive sequences from the given data.

Later these sequences can be put into Hidden Markov Models.

Experimentation code can be found at the following link:

https://github.com/a93h/tpot-ce-cowrie-project/blob/main/HMM.ipynb

The code walkthrough will occur after this slide.

```
Number of unique event ids 18
00 cowrie.client.kex
01 cowrie.client.size
02 cowrie.client.var
03 cowrie.client.version
04 cowrie.command.failed
05 cowrie.command.input
06 cowrie.command.success
07 cowrie.direct-tcpip.data
08 cowrie.direct-tcpip.request
09 cowrie.log.closed
10 cowrie.login.failed
11 cowrie.login.success
12 cowrie.session.closed
13 cowrie.session.connect
14 cowrie.session.file_download
15 cowrie.session.file_download.failed
16 cowrie.session.file_upload
17 cowrie.session.params
```

Fig 2. [cowrie event ids]

Markov Modeling: imports

Here several essential python libraries are imported after being installed using the python command line tool pip.

pip install –r requirements.txt

The randomness generators are also seeded to make the attempts repeatable.

```
import numpy
numpy.random.seed(42)
print(numpy.__path__)

%matplotlib inline
import pandas as pd
import random
import numpy
import matplotlib.pyplot as plt
import pomegranate
import seaborn; seaborn.set_style('whitegrid')
import traceback

from elasticsearch import Elasticsearch
from elasticsearch_dsl import Search
from hmmviz import TransGraph
from hmmlearn import hmm
from pprint import pprint

random.seed(0)
numpy.random.seed(0)
numpy.set_printoptions(suppress=True)

%load_ext watermark
%watermark -m -n -p numpy,scipy,pomegranate,pandas,hmmlearn
```

Scr. 5 [github repo – HMM.py]

Markov Modeling: HiveInfo Class

A HiveInfo class is created to store and easily manipulate each of the hive's various attributes.

In [20]:
```
# Hive Information Class
class HiveInfo:
    def __init__(self) -> None:
        self.city = None,
        self.country = None
        self.code_name = None
        self.ip = None
```

In [21]:
```
# populate hive information
hive_base = HiveInfo()
hive_sensors = [HiveInfo(), HiveInfo(), HiveInfo(), HiveInfo(), HiveInfo(), HiveInfo()]
# sink
hive_base.city = "Toronto"
hive_base.country = "Canada"
hive_base.code_name = "realtatami"
hive_base.ip = "99.251.10.123"
# src 0
hive_sensors[0].city = "Toronto"
hive_sensors[0].country = "Canada"
hive_sensors[0].code_name = "rulingcommittee"
hive_sensors[0].ip = "216.238.83.8"
# src 1
hive_sensors[1].city = "Seoul"
hive_sensors[1].country = "Korea"
hive_sensors[1].code_name = "attractivescow"
hive_sensors[1].ip = "158.247.238.143"
# src 2
hive_sensors[2].city = "Melbourne"
hive_sensors[2].country = "Austrailia"
hive_sensors[2].code_name = "deadinlay"
hive_sensors[2].ip = "67.219.98.203"
# src 3
hive_sensors[3].city = "Stockholm"
hive_sensors[3].country = "Sweden"
hive_sensors[3].code_name = "longtermresult"
hive_sensors[3].ip = "65.20.115.181"
# src 4
hive_sensors[4].city = "Mumbai"
hive_sensors[4].country = "India"
hive_sensors[4].code_name = "sadproduce"
hive_sensors[4].ip = "65.20.68.106"
# src 5
hive_sensors[5].city = "Los Angeles"
hive_sensors[5].country = "United States"
hive_sensors[5].code_name = "resonantnorse"
hive_sensors[5].ip = "149.248.9.4"
```

Scr. 5 [github repo – HMM.py]

Markov Modeling: HiveInfo Class

Each of the data for each hive sensor is collected via hostname and a simple Elasicsearch query language, commonly used in Kibana.
Data-frame info, is printed.

In [22]:
```
def query_es_return_dataframe(index, query): # query es instance and return dataframe
    search_context = Search(using=es, index=index)
    s = search_context.query("query_string", query=query)
    response = s.execute()
    if response.success():
        df = pd.DataFrame((d.to_dict() for d in s.scan()))
    return df
```

In [23]:
```
# connect with local T-Pot instance, contains backup from honeypot cluster
es = Elasticsearch("http://localhost:9200")
indice_search_tag = "logstash-20*"
```

In [24]:
```
dfs = []
for hive_sensor in hive_sensors:
    print(pprint(vars(hive_sensor)))
    df = query_es_return_dataframe(indice_search_tag, f"type.keyword: \"Cowrie\" AND t-pot_hostname.keyword: \"{hive_sensor.code_name}\"")
    # show  dataframe
    df.info()
    # collect data frame
    dfs.append(df)
# merge all useful dataframes
super_df = pd.concat(dfs)
```

Scr. 5 [github repo – HMM.py]

Markov Modeling: Process Dataframe Function

The following excerpt completes the following.
1. Maps the event ids from strings to integers for state transitions.
2. Prints the unique event ids.
3. Sorts the data frame by timestamp
4. Groups the data frame by session and aggregates the groups by event id as a list.
5. Calculates the number of interactions per session.
6. Begins the transition generation for graphing the Markov Chains.

```
# sort unique event ids and drop unused values
event_ids = df['eventid'].unique()
unique_event_ids = []
for ei in event_ids:
    if isinstance(ei, str):
        unique_event_ids.append(ei)
unique_event_ids.sort()
print(f"Number of unique event ids {len(unique_event_ids)}")
# pretty print unique event ids
event_id_from_int = {}
int_from_event_id = {}
for i, unique_event_id in enumerate(unique_event_ids):
    print("{:02d} {}".format(i, unique_event_id))
    event_id_from_int[i] = unique_event_id
    int_from_event_id[unique_event_id] = i
# resolve timestamp to real date
df['@timestamp'] = pd.to_datetime(df['@timestamp'])
# sort by date
df = df.sort_values(['@timestamp'])
# verify dates are sorted
df['@timestamp']
# group dataframe by session and aggregate eventids
df_grouped = df.groupby("session")['eventid'].agg(list)
# checking one example to verify sort and command order makes sense
df_grouped[1]
# calculate and show number of interactions per session
number_of_interactions_per_session = len(df) / len(df_grouped)
print("Average Number of Interactions per session {}".format(number_of_interactions_per_session))
# calculate markov probabalities
# note total transitions
total_transitions = 0
# transition dictionary collection
transition_dict = {}
# transition matrix (N X N) where N is the number of states
for event_id in unique_event_ids:
    transition_dict[event_id] = [0] * len(unique_event_ids)
# prepate data for fit
```

Scr. 5 [github repo – HMM.py]

Markov Modeling: Process Dataframe Function

The following excerpt completes the following.

1. Initializes three lists used later for modeling in hmmlearn and pomegranate python Markov Modeling libraries.
2. Builds data in formats readily usable by hmmlearn, pomegranate and hmmviz.
3. Plots the hmmviz TransGraph or transition graph for the given data.
4. Prepares the hmmlearn model of the transition sequences.
5. Returns a Tuple of different types of data to be used in other parts of the code.

Scr. 5 [github repo – HMM.py]

```
# prepate data for fit
np_fit_data = []
np_fit_data_hmm = []
np_fit_data_lengths = []
# iterate over grouped session and state
# populate transition dictionary
for session, states in df_grouped.items():
    for i in range(len(states) - 1):
        transition_dict[states[i]][int_from_event_id[states[i+1]]] += 1
    # prepare data for fit
    states_as_ints = []
    states_as_ints_hmm = []
    for state in states:
        states_as_ints.append(int_from_event_id[state])
        states_as_ints_hmm.append([int_from_event_id[state]])
    np_fit_data.append(states_as_ints)
    np_fit_data_hmm += states_as_ints_hmm
np_fit_data_lengths = list(map(len, np_fit_data))

# put transition data into dataframe for makov visualization
transition_df = pd.DataFrame(columns=unique_event_ids, index=unique_event_ids)
for row_event_id, col_event_int in transition_dict.items():
    total = sum(col_event_int)
    transition_df.loc[row_event_id] = [(v / total if total > 0 else 0) for v in col_event_int]
    # create graph of system
print(transition_df)
graph = TransGraph(transition_df)

fig = plt.figure(1*idx, figsize=(75, 75))

graph.draw(
    nodefontsize=25,
)
# graphical representation of markov chains and their strength
plt.show()
# print(np_fit_data)
print(f"Number of sequences: {len(np_fit_data)}")
# pomegranate models
model_pom = None
# model_pom = pomegranate.HiddenMarkovModel.from_samples(pomegranate.DiscreteDistribution, n_components=len(unique_event_ids), X=np_fit_data)
# model_pom.bake(verbose=True)
# plt.figure(2*idx , figsize=(100, 100))
# # a plot of the state machine and the probabilities
# model_pom.plot()
# hmm learn models
model_hmm = hmm.MultinomialHMM(n_components=len(unique_event_ids), verbose=True, n_iter=20).fit(np_fit_data_hmm, np_fit_data_lengths)
fig, ax = plt.subplots()
ax.imshow(model_hmm.transmat_, aspect='auto', cmap='spring')
ax.set_title('Transition Matrix')
ax.set_xlabel('State To')
ax.set_ylabel('State From')
return (model_pom, model_hmm, np_fit_data_hmm, np_fit_data_lengths)
```

Markov Modeling: Process Dataframe Function

The following excerpt completes the following.

1. Initializes three lists used later for modeling in hmmlearn and pomegranate python Markov Modeling libraries.
2. Builds data in formats readily usable by hmmlearn, pomegranate and hmmviz.
3. Plots the hmmviz TransGraph or transition graph for the given data.
4. Prepares the hmmlearn model of the transition sequences.
5. Returns a Tuple of different types of data to be used in other parts of the code.

Scr. 5 [github repo – HMM.py]

```
# prepate data for fit
np_fit_data = []
np_fit_data_hmm = []
np_fit_data_lengths = []
# iterate over grouped session and state
# populate transition dictionary
for session, states in df_grouped.items():
    for i in range(len(states) - 1):
        transition_dict[states[i]][int_from_event_id[states[i+1]]] += 1
    # prepare data for fit
    states_as_ints = []
    states_as_ints_hmm = []
    for state in states:
        states_as_ints.append(int_from_event_id[state])
        states_as_ints_hmm.append([int_from_event_id[state]])
    np_fit_data.append(states_as_ints)
    np_fit_data_hmm += states_as_ints_hmm
np_fit_data_lengths = list(map(len, np_fit_data))

# put transition data into dataframe for makov visualization
transition_df = pd.DataFrame(columns=unique_event_ids, index=unique_event_ids)
for row_event_id, col_event_int in transition_dict.items():
    total = sum(col_event_int)
    transition_df.loc[row_event_id] = [(v / total if total > 0 else 0) for v in col_event_int]
    # create graph of system
print(transition_df)
graph = TransGraph(transition_df)

fig = plt.figure(1*idx, figsize=(75, 75))

graph.draw(
    nodefontsize=25,
)
# graphical representation of markov chains and their strength
plt.show()
# print(np_fit_data)
print(f"Number of sequences: {len(np_fit_data)}")
# pomegranate models
model_pom = None
# model_pom = pomegranate.HiddenMarkovModel.from_samples(pomegranate.DiscreteDistribution, n_components=len(unique_event_ids), X=np_fit_data)
# model_pom.bake(verbose=True)
# plt.figure(2*idx , figsize=(100, 100))
# # a plot of the state machine and the probabilities
# model_pom.plot()
# hmm learn models
model_hmm = hmm.MultinomialHMM(n_components=len(unique_event_ids), verbose=True, n_iter=20).fit(np_fit_data_hmm, np_fit_data_lengths)
fig, ax = plt.subplots()
ax.imshow(model_hmm.transmat_, aspect='auto', cmap='spring')
ax.set_title('Transition Matrix')
ax.set_xlabel('State To')
ax.set_ylabel('State From')
return (model_pom, model_hmm, np_fit_data_hmm, np_fit_data_lengths)
```

Markov Modeling: Model Loop

The shown model loop produces usable Hidden Markov Models using Discrete Distributions.

It also calls the previous slide's function to process the given data frame and corresponding hive sensor.

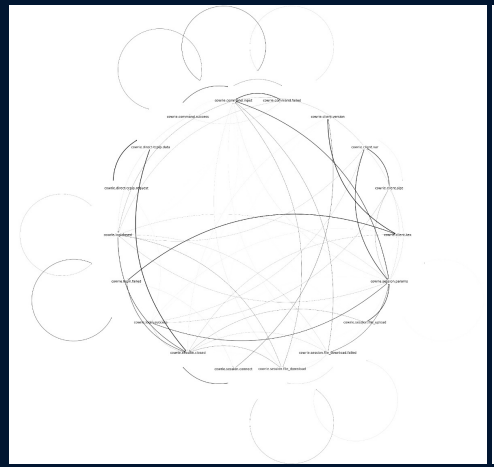
```
my_models = []
for idx, (df, hive_sensor) in enumerate(zip(dfs, hive_sensors)):
    pprint(vars(hive_sensor))
    print(f"{df.info()}")
    my_models.append(process_dataframe(idx + 1, df))
```

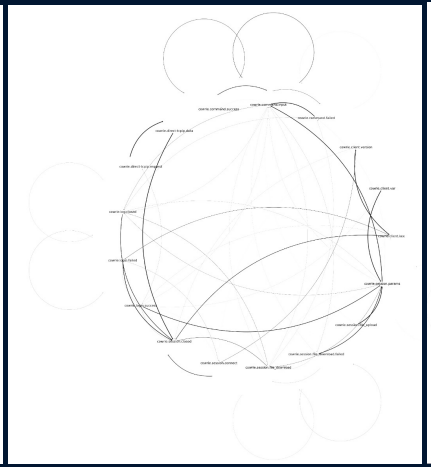
Scr. 5 [github repo – HMM.py]

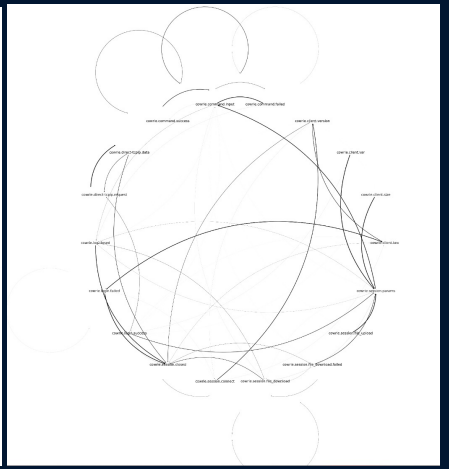
Markov Modeling: hmmviz Transition Strengths

As you can see there are difference but also many similarities between various regions. You can observe the following link for more details and transition heat maps.

https://github.com/a93h/tpot-ce-cowrie-project/blob/main/HMM.ipynb







Markov Modeling: Final Scores

This last for loop does the following:

1. Prints the hive sensor information.
2. Prints the number of sequences.
3. Prints the score of the model.

```
for my_model, hive_sensor in zip(my_models, hive_sensors):
    pprint(vars(hive_sensor))
    model_pom = my_model[0]
    model_hmm = my_model[1]
    np_fit_data_hmm = my_model[2]
    np_fit_data_lengths = my_model[3]
    print(f"Number of Sequences: {len(np_fit_data_lengths)}")
    try:
        # print(model_hmm.transmat_)
        score = model_hmm.score(X=np_fit_data_hmm, lengths=np_fit_data_lengths)
        print(score)
        # print(model_hmm.predict(numpy.array([[13], [3], [9]]), lengths=[3]))
    except Exception as e:
        print(f"hmmlearn model failed to build {e}")
        traceback.print_exc()
    try:
        # print(model_pom.probability([[13, 3, 9]]))
        pass
    except Exception as e:
        print(f"promegranate model failed to build {e}")
        traceback.print_exc()
```

Scr. 5 [github repo – HMM.py]

Markov Modeling: Final Scores

{'city': 'Toronto',
 'code_name': 'rulingcommittee',
 'country': 'Canada',
 'ip': '216.238.83.8'}
12131
-21514.36465916146
{'city': 'Seoul',
 'code_name': 'attractivescow',
 'country': 'Korea',
 'ip': '158.247.238.143'}
3973
-6845.826437526963
{'city': 'Melbourne',
 'code_name': 'deadinlay',
 'country': 'Austrailia',
 'ip': '67.219.98.203'}
14213
-23592.781711660155

{'city': 'Stockholm',
 'code_name': 'longtermresult',
 'country': 'Sweden',
 'ip': '65.20.115.181'}
32788
-33832.91060668411
{'city': 'Mumbai',
 'code_name': 'sadproduce',
 'country': 'India',
 'ip': '65.20.68.106'}
11435
-19231.69921274217
{'city': 'Los Angeles',
 'code_name': 'resonantnorse',
 'country': 'United States',
 'ip': '149.248.9.4'}
27039
-16527.209799049913

As you can see the best score with the most ideal log probability is Seoul Korea. The most predictable data was fount there although the number of sequences was much lower there.
The least predictable and region with the worst score was Stockholm Sweden, and on the opposite spectrum received the most amount of traffic. Although, we can see this is a not a true correlation.

Seoul	3973	-6845.826437526963
Los Angeles	27039	-16527.209799049913
Mumbai	11435	-19231.69921274217
Toronto	12131	-21514.36465916146
Melbourne	14213	-23592.781711660155
Stockholm	32788	-33832.91060668411

Conclusions and Future Work

In future and if time permits, I will use this knowledge to delve further into honeypots to identify threat actors through a mixture of more realistic honeypots, reinforcement learning and data analysis via machine learning.

For now, I realize that simple data training using Markov Models suffices for simple data analysis and basic threat prediction but will be insufficient in identifying genuine threat sources.

References

[1] Toronto Metropolitan University. (2022, May 8). *Toronto Metropolitan University*. Wikipedia. Retrieved May 14, 2022, from https://en.wikipedia.org/wiki/Toronto_Metropolitan_University#/media/File:TMU_logo.svg

[2] Spremić, M., & Šimunic, A. (2018, July 4). *Cyber Security Challenges in Digital Economy*. International Association of Engineers. Retrieved May 14, 2022, from http://www.iaeng.org/publication/WCE2018/WCE2018_pp341-346.pdf

[3] quoteresearch, A. (2020, December 16). *Information wants to be expensive. information wants to be free*. Quote Investigator. Retrieved May 14, 2022, from https://quoteinvestigator.com/2018/03/09/info/

[4] Francen, E. (2019). *Unsecurity: Information Security is failing. breaches are epidemic. how can we fix this broken industry?* Beaver's Pond Press, Inc.

[5] Spremić, M., & Šimunic, A. (2018, July 4). *Cyber Security Challenges in Digital Economy*. International Association of Engineers. Retrieved May 14, 2022, from http://www.iaeng.org/publication/WCE2018/WCE2018_pp341-346.pdf

[6] Ferrer, O. N. (2021, January 3). *Analysis of reinforcement learning techniques applied to honeypot ... - UOC*. Universitat Oberta de Catalunya. Retrieved June 25, 2022, from http://openaccess.uoc.edu/webapps/o2/bitstream/10609/126948/7/onavarrofeTFM021memory.pdf

References

[7] Rade, R. (2018, January 8). *Predictive threat intelligence*. Rahul Rade. Retrieved June 25, 2022, from https://imrahulr.github.io/machine%20learning/big%20data/kafka/cyber%20security/hmm/coe-cnds/2018/01/08/a-threat-intelligence.html

[8] Lee, S., Abdullah, A., Jhanjhi, N., Kok, S., ZA, A., N, Z., A, A., S-G, B., C, M., C, L., G, B., S, M., K, R., K, C., V, G., E, C., B, C., J, W., L, S., ... C, K. (2021, January 25). *Classification of botnet attacks in IOT smart factory using Honeypot combined with machine learning*. PeerJ Computer Science. Retrieved June 25, 2022, from https://peerj.com/articles/cs-350/

[9] Mohammadzadeh, H., Mansoori, M., & Honarbakhsh, R. (2011). *Taxonomy of hybrid honeypots - IPCSIT*. IPCSIT. Retrieved July 10, 2022, from http://ipcsit.com/vol11/27-ICNEE2011-N30007.pdf

[10] Maesschalck, S., Giotsas, V., Green, B., & Race, N. (2021, December 30). *Don't get Stung, cover your ICS in honey: How do honeypots fit within industrial control system security*. Computers & Security. Retrieved July 10, 2022, from https://www.sciencedirect.com/science/article/pii/S0167404821004211

[11] Lebedev, S. (2014, March 23). *HMMLEARN/hmmlearn: Hidden markov models in python, with scikit-learn like API*. GitHub. Retrieved July 16, 2022, from https://github.com/hmmlearn/hmmlearn

[12] Hogan, A. (2022, July 16). *A93H/tpot-CE-cowrie-project: Simple tpot CE cowrie honeypot project*. GitHub. Retrieved July 16, 2022, from https://github.com/a93h/tpot-ce-cowrie-project