A Network Recon Bot

By Derek Ogle and Dylan Katz

Important Links

Bot Stuff: github.com/dirk37/reconbot
discord.gg/gPtpZ4

General Club Stuff: grayhats.io
Other Club Stuff: brayhats.club

Presentation Overview

1. What is Network Recon?
2. Bot Overview
3. Async IO and Coroutines
4. Command #1: Pinging a Host
5. Command #2: Geolocate a Host
6. Command #3: Accessing WHOIS
7. Command #4: Scanning a Host
8. Questions
9. Appendix

Very happy robot dancing in a field

What is Network Recon?

Man coming out of laptop with binoculars, likely performing reconnaissance

Definition

Reconnaissance

- Examining a target from the outside to get information on it

Network Reconnaissance

- Examining a network from outside to look for potential access points, vulnerabilities, and other info
    - These can include open ports, login portals, and outdated software

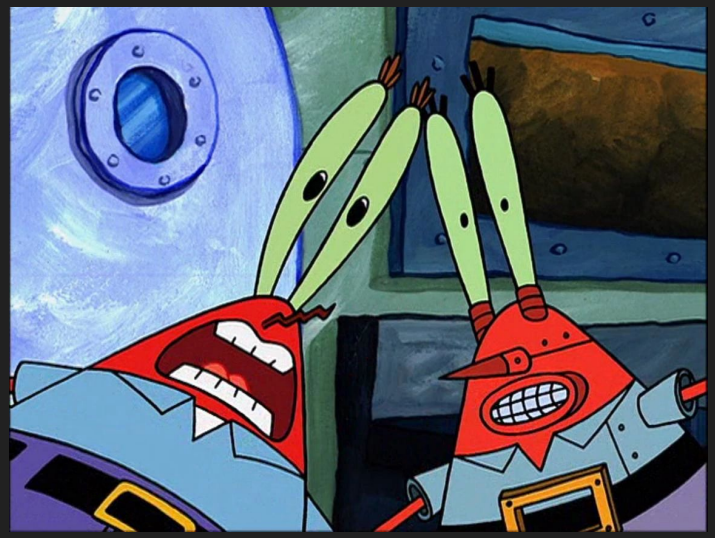
Who does Network Recon?

- Black hats (bad guys) do this for money or other reasons
- White hats (good guys) do this to find security holes before the bad guys
- Gray hats are somewhere in between

Why We Made a Discord Bot To Do This

¯_(ツ)_/¯

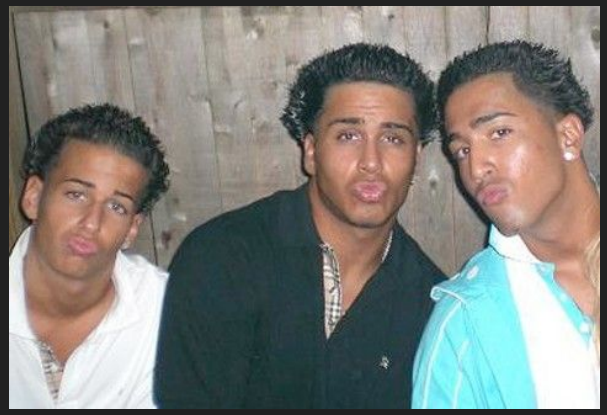
Bot Overview



- Commands and Tools
- Architecture
- Dependencies
- Hello, Discord!

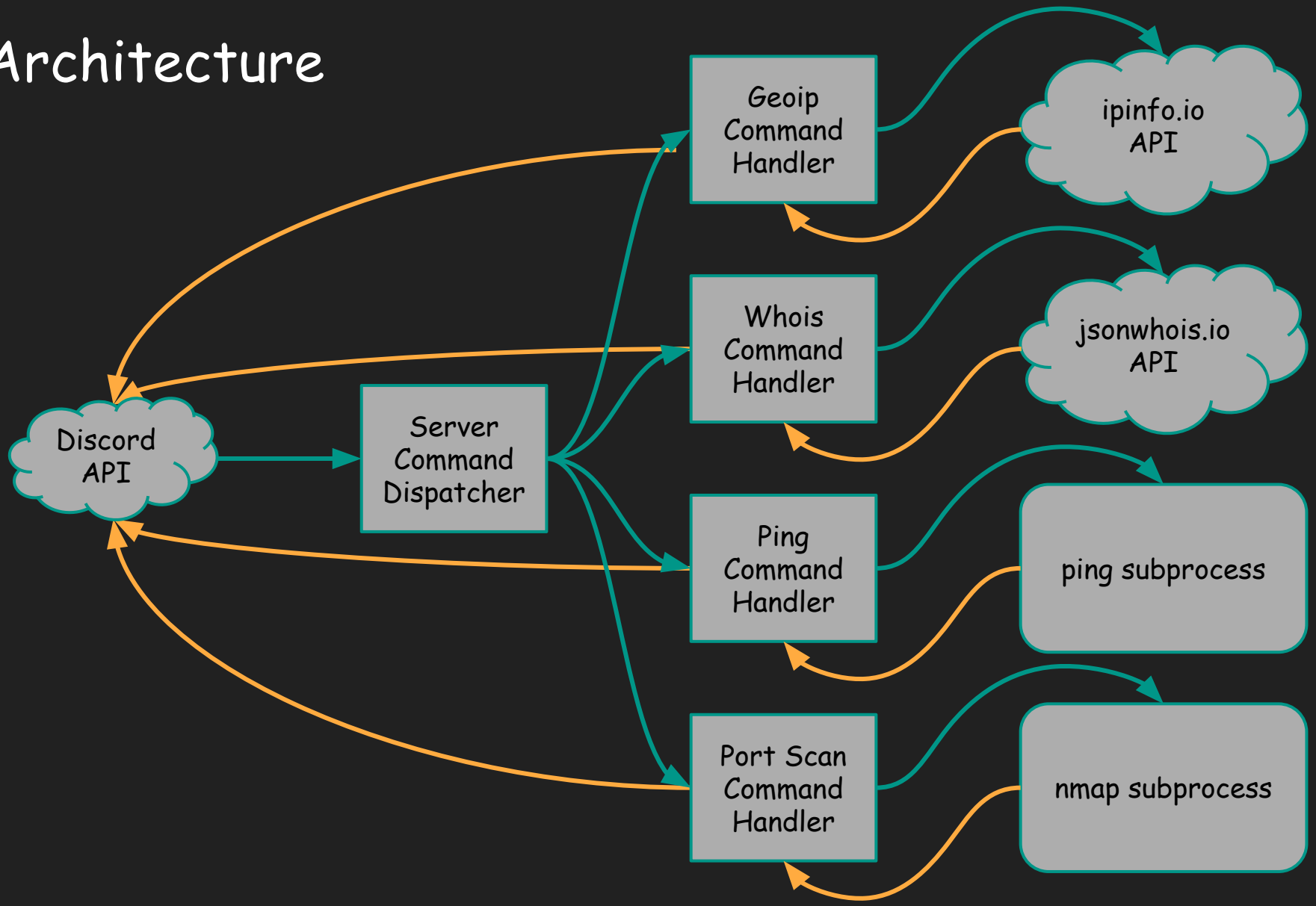
Commands and Tools / Services

Command	Tool	Service	Description
ping	ping		Pinging
geo		ipinfo.io	IP Geolocation
whois		jsonwhois.io	Whois lookup
nmap	nmap		Port Scanning

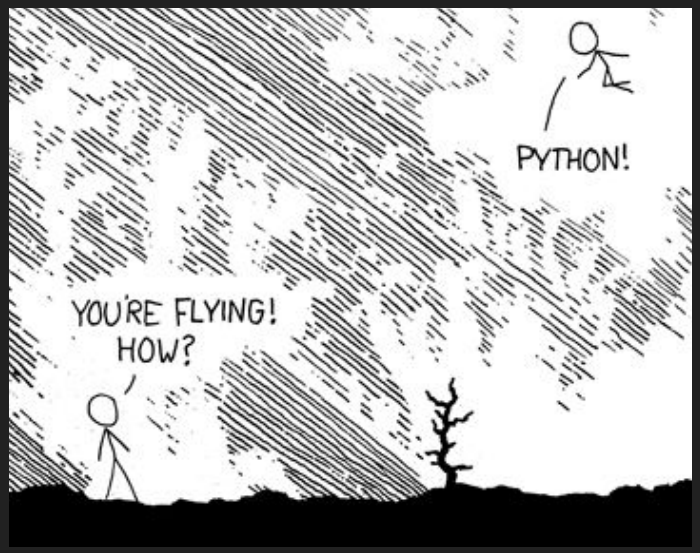


3 pack of tools

Architecture



Installing Dependencies



We made this program in Python 3.5+

From the command line, run:

```
pip3 install discord requests
```

"`pip3`" is Python's package manager, which installs additional libraries for Python that do not ship with the default install

The "`discord`" package is used to write bots for Discord

The "`requests`" package is used to make HTTP requests.

Most Basic Bot - Hello, Discord

```
import configparser
from discord.ext import commands

cfg_parser = configparser.ConfigParser()
bot = commands.Bot(command_prefix='%')
with open('config.ini') as f: cfg_parser.read_file(f)

@bot.event
async def on_ready():
    print('Logged in as {} ({})'.format(
        bot.user.name, bot.user.id))

# TODO - add your commands here

bot.run(cfg_parser['Tokens']['Discord'])
```

Hello, Discord: Configuration Parsing

*C:\Users\windows\Desktop\College\Reconbot\config.ini - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

config.ini

```
1 [Tokens]
2 discord=NDQxNDQwNjc4NzUzMzM3MzU1.DcwTWg.A_6NTFV
3 ipinfo=cc8b1d0905b2cf
4 whois=bxd0RLjHBOYuFn2loPold9WrjwKgkbox
```

length : 117 lines : 4 Ln : 2 Col : 48 Sel : 0 | 0 Unix (LF) UTF-8 INS

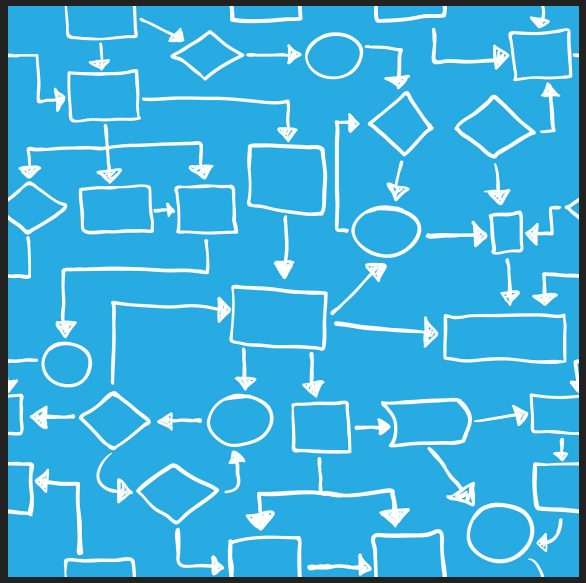
Hello, Discord: Configuration Parsing

```
import configparser
from discord.ext import commands

cfg_parser = configparser.ConfigParser()
bot = discord.Client(command_prefix='%')
with open('config.ini') as f: cfg_parser.read_file(f)

@bot.event
async def on_ready():
    print('Logged in as {} ({})'.format(
        bot.user.name, bot.user.id))

# TODO - add your commands here

bot.run(cfg_parser['Tokens']['Discord'])
```

Hello, Discord: Client Setup

```
import configparser
from discord.ext import commands

cfg_parser = configparser.ConfigParser()
bot = commands.Bot(command_prefix='%')
with open('config.ini') as f: cfg_parser.read_file(f)

@bot.event
async def on_ready():
    print('Logged in as {} ({})'.format(
        bot.user.name, bot.user.id))

# TODO - add your commands here

bot.run(cfg_parser['Tokens']['Discord'])
```

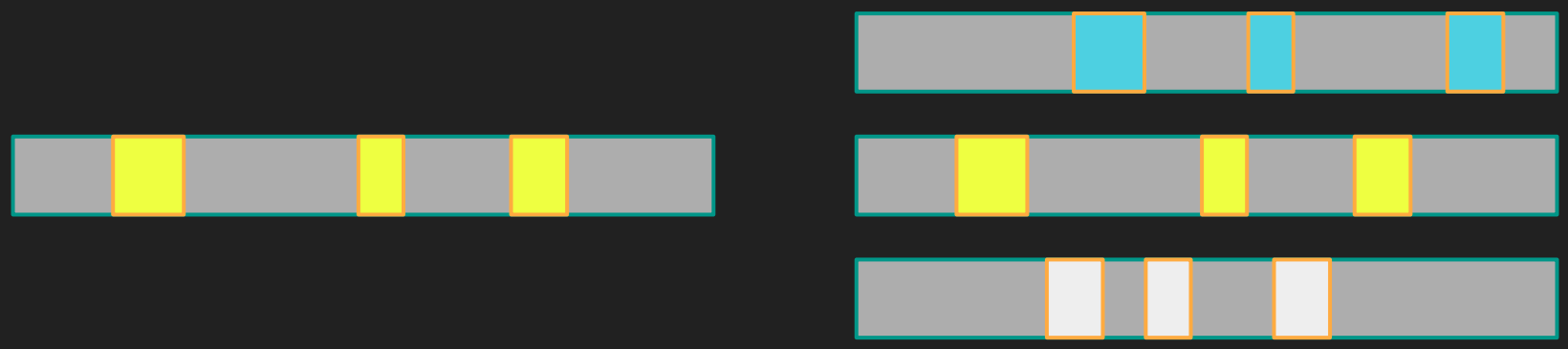
Async IO and Coroutines



- Synchronous Programming
- Asynchronous Programming
- Coroutines
- Await/Async in Python

What is Synchronous Programming?

- Instructions run sequentially, one directly after another
- To read over a network, you block, and cannot read from other locations at the same time
- Time spent blocking leaves unused CPU cycles
- Can be fixed with threads, but at a cost of overhead and complexity



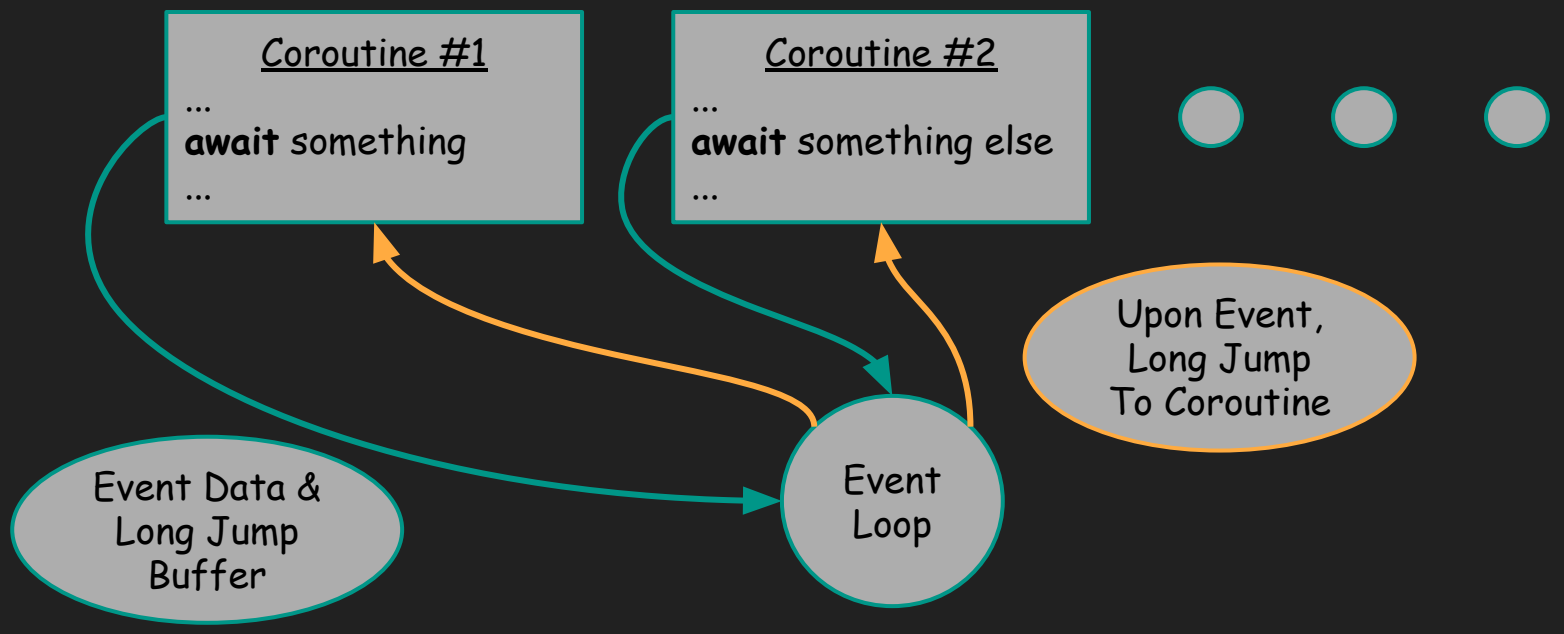
What is Asynchronous Programming?

- Tell the OS that we have a set of events that we want to wait on
- Get back any events that finish in the order that they finish
- Loop over this process creates an "event loop"
- Handle them with callback functions
- Significantly better performance and lower resource usage than threads

- Destroys flow control of the program (see: callback hell with Node.js)
- Often requires global variables to manage state
- Implemented differently on every OS
  - epoll(7) on linux, kqueue(2) on macOS/*BSD, IOCP on Windows/Solaris
  - poll(2) and select(2) are more portable but require linear lookups and are slow

What are Coroutines?

- Light-weight threads in user space using platform specific APIs
- Give the flow control and readability of synchronous programming
- Give the performance and scalability of asynchronous programming
- Often a language/runtime feature (e.g. Go, .Net Fibres, Python 3.5+, etc.)



What do Async/Await do?

- Designate that a function is a coroutine with the **async** keyword
- Call a coroutine with the **await** keyword
- A coroutine can only be called from another coroutine
- Python libraries (e.g. aiohttp, discord.py) generally start the environment for you
- If your library does not do this, then you may do it manually with:

```
loop = asyncio.get_event_loop()
loop.run_until_complete(my_coroutine)
```

Pinging a Host

Command #1

Steve Harvey - **Host** of Family Feud

- What is ping?
- Time to live
- Pinging from Python
  - os.popen
  - subprocess.run
  - asyncio.create_subprocess

What is ping?

- Utility that sees if a host (computer or server) is online
  - Sends several ICMP packets and waits for a response
  - Measures response time and rate
  - Displays results

Sample usage on Linux:

```
~> ping -c 3 google.com
PING google.com (172.217.3.206): 56 data bytes
64 bytes from 172.217.3.206: icmp_seq=0 ttl=54 time=36.727 ms
64 bytes from 172.217.3.206: icmp_seq=1 ttl=54 time=37.000 ms
64 bytes from 172.217.3.206: icmp_seq=2 ttl=54 time=33.598 ms

--- google.com ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 33.598/35.775/37.000/1.543 ms
~>
```

Ping - Time to Live

- Maximum number of routers packet will pass through before being dropped
  - Used to prevent loops in the network
- Each router decrements the number till its zero
- TTL is set by computer which was pinged

Why is this important?

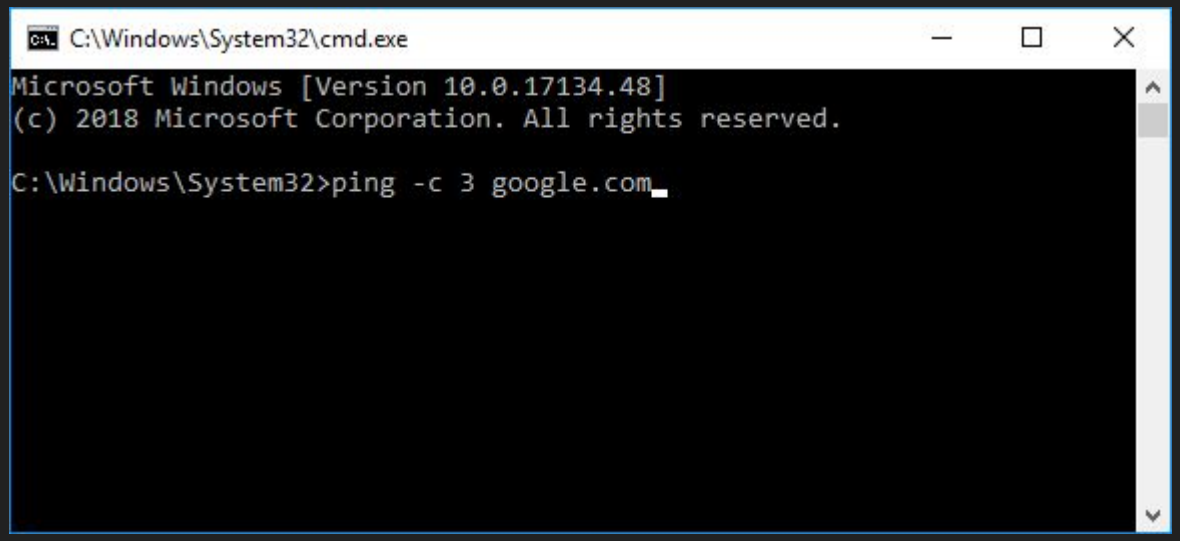
- TTL can be used to guess the OS its running
  - Most windows systems have a default TTL of 128
  - Standard Linux is 255
  - Macs are 64

Pinging from Python

Using os.popen is equivalent to typing the command into a terminal. For example:

```
import os
os.popen("ping -c 3 google.com")
```

is the same as:



Pinging from Python

- Version #1: Pinging using os.popen

```
import os

@bot.command()
async def ping(host):
    await bot.say('Pinging {}'.format(host))
    res = os.popen('ping -{} 3 {}'.format('c' if os.name ==
        'posix' else 'n', host))

    if res.returncode:
        await bot.say('Error: host {} not up'.format(host))
    else:
        await bot.say('```{}```'.format(res.read()))
```

Problems

- This has a security hole you could drive a truck through

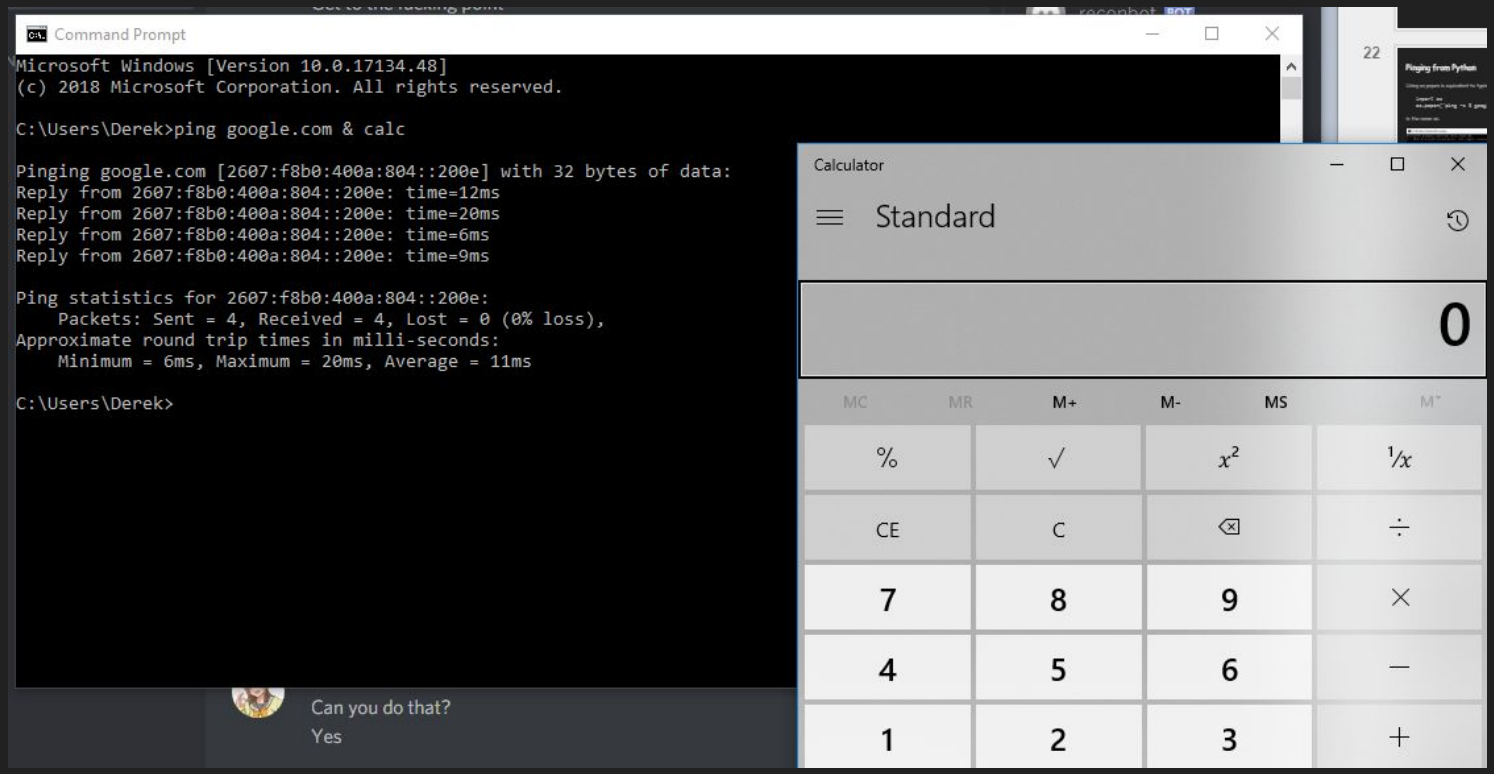
```
await ping('localhost & touch ~/HACKED')
```

- Because it runs output directly inside a shell, we can use shell features
- The & operator tells the shell to fork into the background
- What comes next runs as a different command entirely
- Solution: use **subprocess** instead

Problems

```
import os
os.popen("ping google.com & calc")
```

is the same as:



Pinging from Python

- Version #2: Pinging using **subprocess**

```
import os
import subprocess

@bot.command()
async def ping(host):
    await bot.say('Pinging {}'.format(host))
    args = ['ping', '-c' if os.name=='posix' else '-n',3, host]
    res = subprocess.run(args, stdout=subprocess.PIPE)

    if res.returncode:
        await bot.say('Error: host {} not up'.format(host))
    else:
        await bot.say('```{}```'.format(
            res.stdout.decode('utf-8')))
```

Improvements with Part 2

- Subprocess only runs the process specified so command injection is no longer possible

Problems with Part 2

- subprocess.run(args) is a blocking operation
  - This means that while the pinging is happening, the bot will hang
  - Solution: use **asyncio** subprocess instead of the default **subprocess**

Pinging from Python

- Version #3: Pinging using **asyncio**

```
import os
import asyncio

@bot.command()
async def ping(host):
    await bot.say('Pinging {}'.format(host))
    args = ['ping', '-c' if os.name=='posix' else '-n',3, host]
    proc = await asyncio.create_subprocess_exec(*args,
        stdout=subprocess.PIPE)
    (data, _) = await proc.communicate()
    if res.returncode:
        await bot.say('Error: host {} not up'.format(host))
    else:
        await bot.say('```{}```'.format(data.decode('utf-8'))
```

Moderately Important Detail

- This operation doesn't work with default config on windows
- You need a few extra lines of code to check for windows systems and use a different config:

```
import sys
import asyncio

if sys.platform == 'win32':
    asyncio.set_event_loop(asyncio.ProactorEventLoop())
```

This should now work on windows!

Ping Integration Complete!

Lets Try It

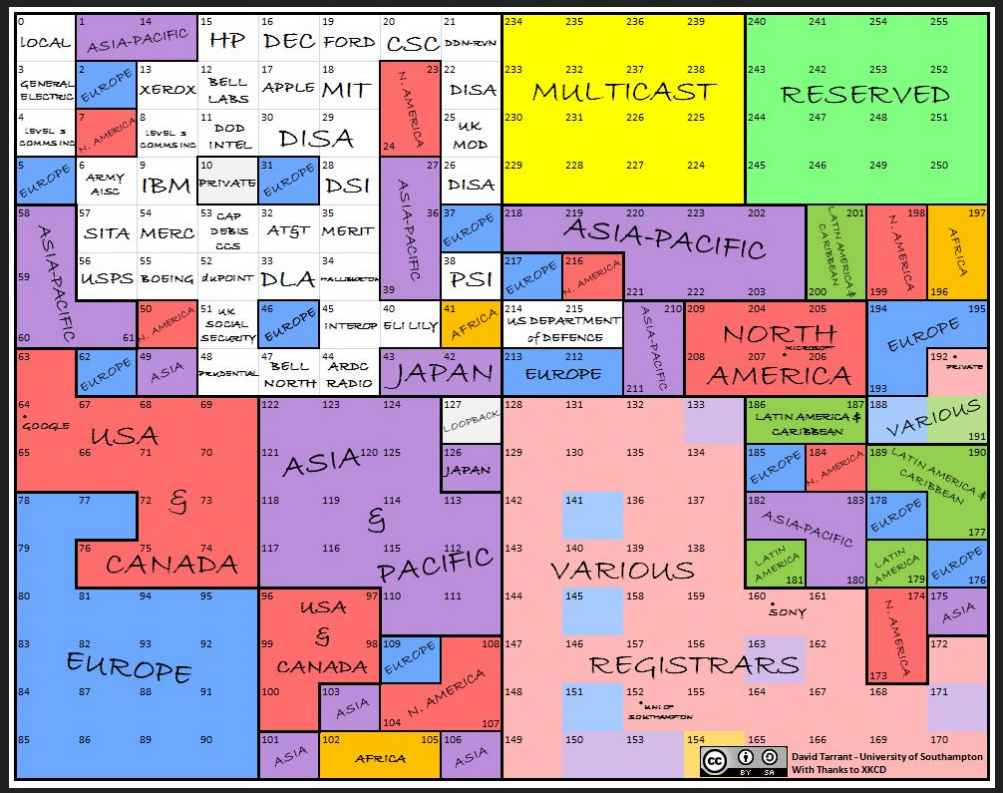
IP Geolocation

Command #2

- What is IP Geolocation
- What is ipinfo.io
- DNS from Python
- Geolocation from Python
  - requests
  - aiohttp.ClientSession

What is IP Geolocation?

- Database of IP addresses and locations
- You give it an IP address, and it gives you coordinates
  - Location is approximate and usually points to ISP buildings
  - Different countries have different IP blocks



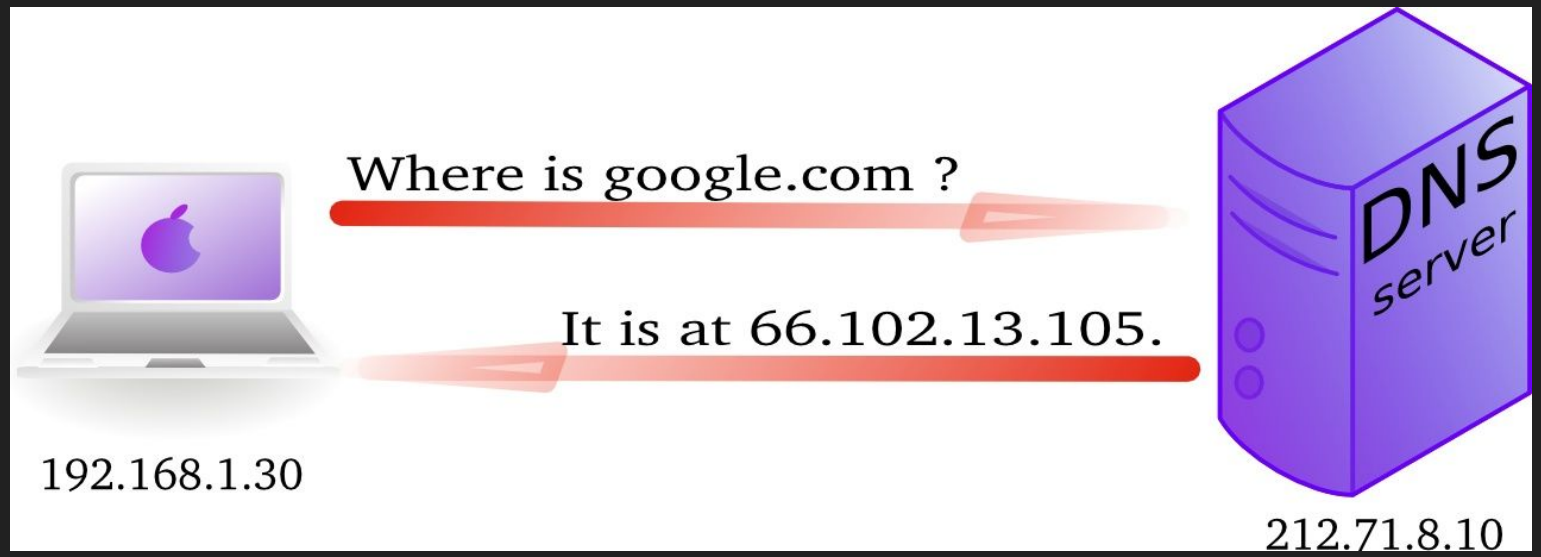
What is ipinfo.io?

- Online service that makes the geolocation database accessible
- Accessible over a REST API and returns result as a JSON string
- Usable from cURL:

```
~> curl http://ipinfo.io/216.239.36.21?token=daslje93ffj
{
  "ip": "216.239.36.21",
  "hostname": "any-in-2415.1e100.net",
  "city": "Emeryville",
  "region": "California",
  "country": "US",
  "loc": "37.8342,-122.2900",
  "postal": "94608",
  "org": "AS15169 Google LLC"
}
~>
```

DNS from Python

- Our bot is designed to work on domain names such as google.com
- IP geolocation does not work with domain names
- Luckily, domain names are just shorthand for an ip address
- DNS (Domain Name Service) translates domain names to ip address
- Before our bot can use ip geolocation, we must use DNS to get the ip from the domain name



DNS from Python

- Version #1: Using socket.gethostbyname

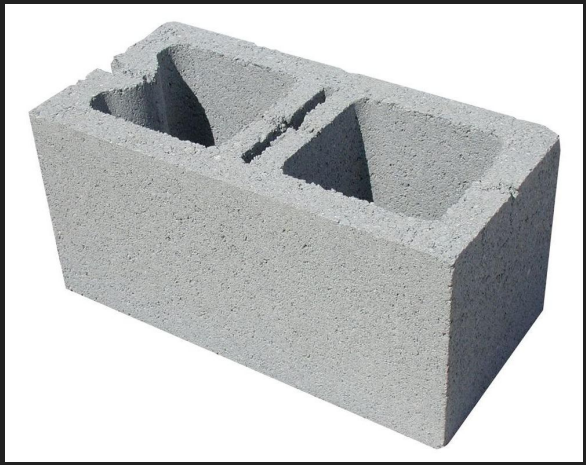
```
import socket

async def dnsquery(host):
    ipa = socket.gethostbyname(host)
    await chan.send_message('IP Addr of `{}` is `{}`'.format(
        host, ipa))
    return ipa

@bot.command()
async def hostresolve(host):
    await dnsquery(host)
```

A Note on Performance

- socket.gethostbyname is blocking
  - We should use this asynchronously

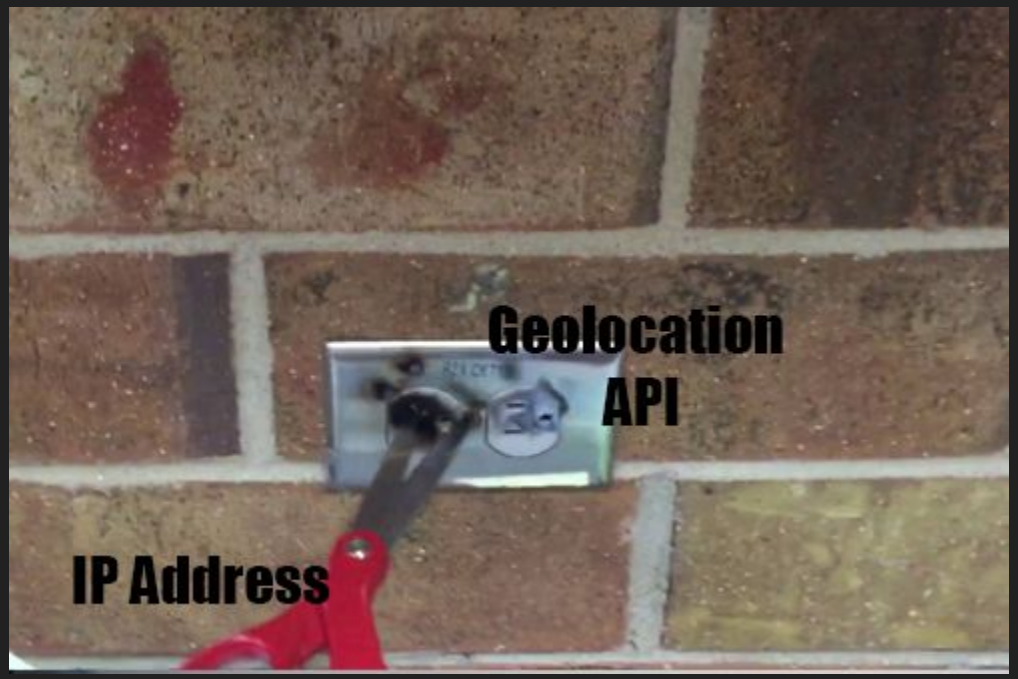
However:

- Because of the way that Python's event loops work on Windows, this cannot be fixed on Windows
- This is because DNS is a UDP protocol, and the Python event loop that supports subprocesses does not support UDP



What was the point of all this again?

- Geolocation only works with ip addresses
- We just used DNS to get the IP address from the domain name
- Now that we have the IP, we can plug it in to our geolocation API



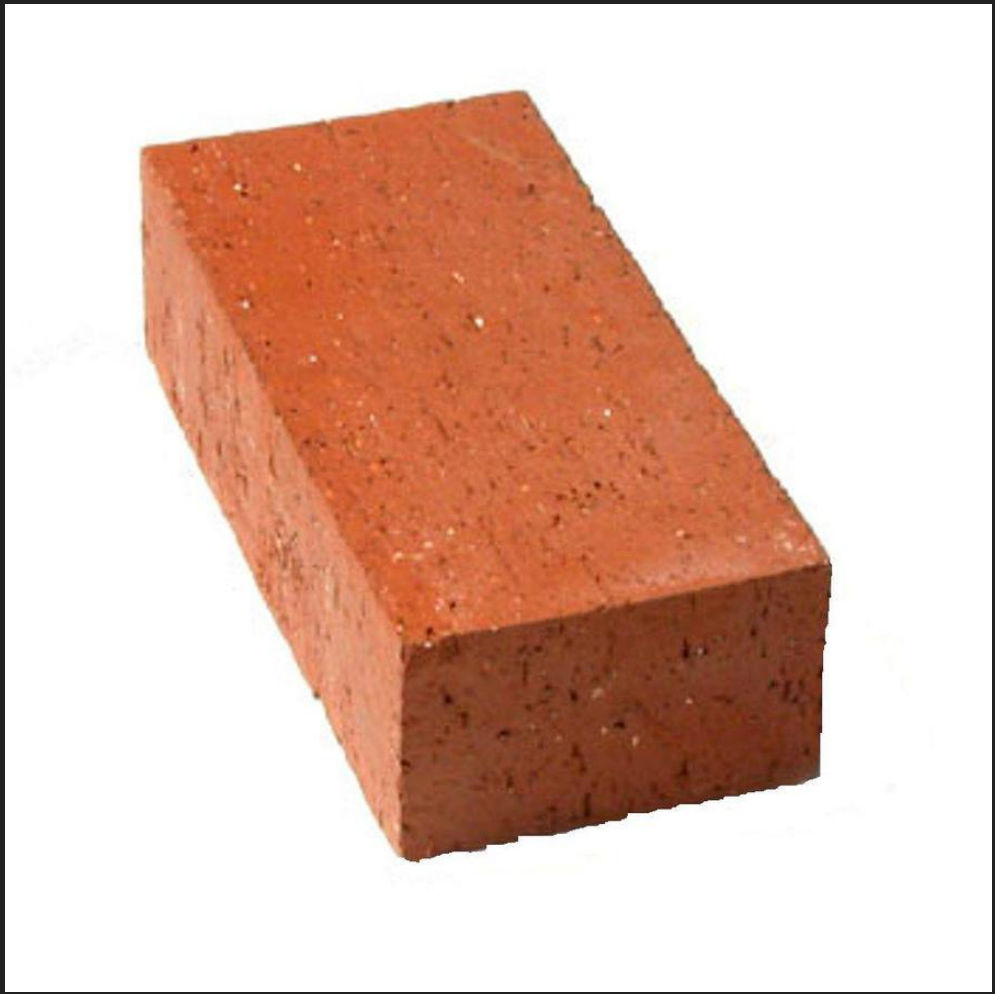
Geolocation from Python

- Version #1: Using requests

```
import json
import requests

@bot.command()
async def geolocate(host):
    ipa = await dnsquery(host)

    req = requests.get('http://ipinfo.io/{}?token={}'.format(
        ipa, cfg_parser['Tokens']['Ipinfo']))
    res = json.loads(req.text)

    await bot.say('https://maps.google.com?q={}'.format(
        res['loc']))
```

Problems

- requests.get is blocking
- Solution: use aiohttp



Geolocation from Python

- Version #2: Using aiohttp.ClientSession.get

```
import json
import aiohttp
HTTP_SESSION = aiohttp.ClientSession(
    skip_auto_headers=['User-Agent'])

@bot.command()
async def geolocate(host):
    ipa = await dnsquery(host)
    req = await HTTP_SESSION.get('http://ipinfo.io/{}'+
        '?token={}'.format(ipa, cfg_parser['Tokens']['Ipinfo']))
    txt = await req.text()
    res = json.loads(txt)
    await bot.say('https://maps.google.com?q={}'.format(
        res['loc']))
```

ipinfo.io Integration Complete!

Let's Try It

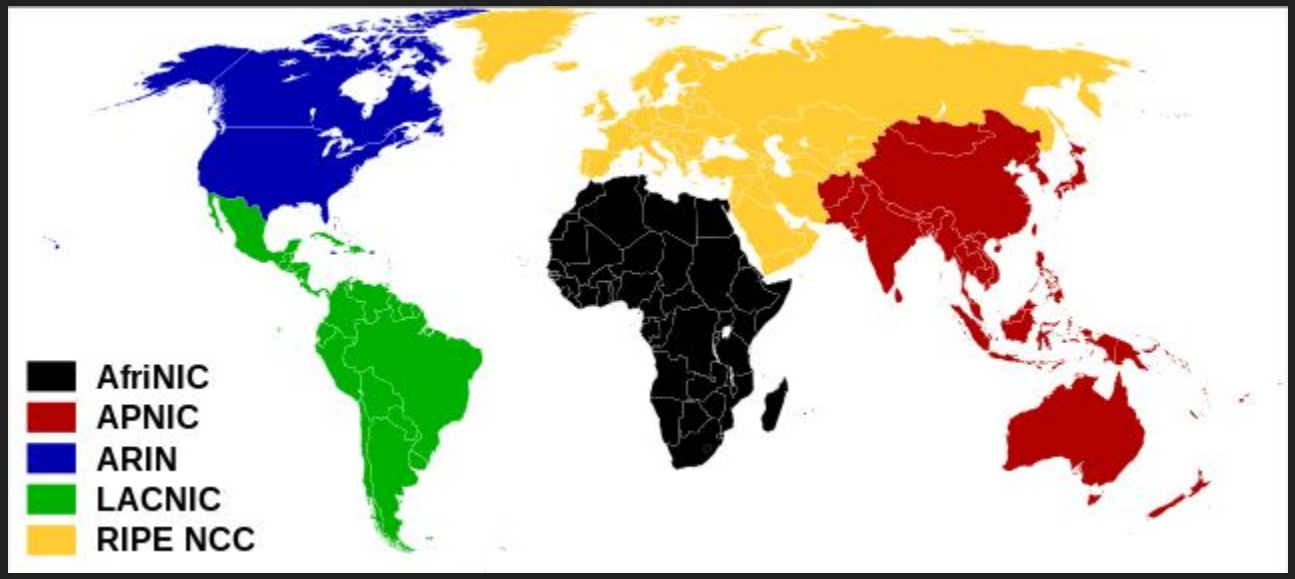
Accessing WHOIS

Command #3

- What is WHOIS
- Mistakes with WHOIS
- jsonwhois.io API
- WHOIS from Python

What is WHOIS?

- WHOIS is a database of who owns which domain names
- Each TLD (Top Level Domain - .com, .org, etc) manages their own whois database



Whois

- Whois contains info on the owner of a domain name
  - This include name, address, phone number, email, and other contact info
- Useful for determining who owns a domain
  - Unless they used private registration
- Private registration
  - The domain registrar puts their details in instead of yours
  - Masks your identity
  - Prevents spam (lots of spammers scrape whois for contact info)
  - Unless you want hundreds of cancerous emails, use private registration
  - I didn't use private registration once….

What happens when you don't use private registration

Low Cost ($) Design and Development Service! Inbox x

Devi <gomtideviseo.web@gmail.com> Feb 20

to derek

Hi ,

I wrote you an email in regards to **Website Design/Mobile APPs Development and Re-design in Multi-Language**. I would be really interested to present you with my approach and proposal in this regard. Our technology firm is based in New Jersey, US having a team of 150+ IT experts with experience of delivering over 2000+ projects into Website & Custom Mobile App Development for iOS/Android & Windows platforms at very reasonable price.
If you are interested in our services then I can send you our Company profile/portfolio/ more details/action for your website with affordable quotation?
I'm looking forward to your reply.

Regards,
Devi!!

Note: This Email ID used for Bulk Mailing only after your reply one of our company's representatives will get back to you shortly via E-mail or call.

Web design and Development !! Inbox x

Kiran Sukla <kiransukla45@gmail.com> Mar 5

to derek

Hi The,
Hope this mail fined you well.

I went to your website and I found that it could be more attractive and more creative for the better.
Due to this reason your website may not be putting impact over the users as you are wishing. I can help you by sharing some ideas (if you are interested and allow me to do so)
regarding the enhancement to the website design so it could add to the overall look and feel to your website to enable it catch more eyes online.

We are creative website design company based offering quality with skill work force having ample of experience in website designing and Re-designing work. With over 8 years of web design experience, our team dedicated designers can help take your web project to the next level.

Let me take our opportunity to introduce our services:-

1.Website design development
2.Web application development
3.New website creation
4.Website Re-designing development
5.E-commerce website development
6.CMS web design

Do let me know if you are interested and I would be happy to share our methodologies, past work details and client testimonials and prices.
I look forward to your positive mail

Thanks and Regards
Kiran Sukla
Business management development

Note: - It is not an automated e-mail, we keep on sending out these emails to all those people whom we find eligible of using our services. To unsubscribe from future e-mail (i.e.to ensure that we do not contact you again for this matter), please send me blank e-mail to it with 'No' as subject.

Whois API

- The official whois gets angry if you try and make requests programmatically
- Must use an API which has somehow acquired a copy of the official whois data
- We're using jsonwhois.io
- Once you sign up and get an api key, requests can be made with a get

https://api.jsonwhois.io/whois/domain?key=APIKEY&domain=DOMAIN

- Returns json containing the info

```
{
    "result": {
        "changed": "2017-09-03 20:32:00",
        "contacts": {
            "admin": [
                {
                    "address": null,
                    "changed": null,
                    "city": null,
                    "country": null,
                    "created": null,
                    "email": null,
                    "fax": null,
                    "handle": null,
                    "name": "WhoisGuard Protected",
                    "organization": "WhoisGuard, Inc.",
                    "phone": null,
                    "state": null,
                    "type": null,
                    "zipcode": null
                }
            ]
        },
        "created": "2017-07-05 18:34:43",
        "dnssec": null,
        "expires": "2018-07-05 18:34:43",
        "name": "gggg",
        "nameservers": [
            "DNS1.REGISTRAR-SERVERS.COM",
            "DNS2.REGISTRAR-SERVERS.COM"
        ],
        "registered": true,
        "registrar": {
            "email": "abuse@namecheap.com",
            "id": "1068",
            "name": "NameCheap, Inc",
            "url": null
        },
        "status": "clientTransferProhibited https://icann.org/epp#clientTransferProhibited"
    }
}
```


This is what the JSON return from the API looks like

WHOIS from Python

- Version #1: With aiohttp

```
@bot.command()
async def whois(name):
    await bot.say('Running WHOIS against `{}`'.format(name))

    req = await HTTP_SESSION.get('https://api.jsonwhois.io/'+
        'whois/domain?key={}&domain={}'.format(
        cfg_parser['Tokens']['Whois'], name))
    txt = await req.text()
    jdata = json.loads(txt)

    await bot.say('```{}```'.format(json.dumps(
        jdata, sort_keys=True, indent=4)))
```

jsonwhois.io Integration Complete!

Network Mapping

Command #4

- What is NMAP
- NMAP from Python

NMAP

- Nmap is a tool for network reconnaissance
  - Specifically finding, identifying, and scanning machines
- It has a satanically large variety of options, in this bot we're just using the default option
  - The default option does a basic port scan of the most common ports
  - This can reveal other services running on the server besides the website

NMAP Example

C:\Windows\System32\cmd.exe

```
C:\Windows\System32>nmap zsport.com

Starting Nmap 7.60 ( https://nmap.org ) at 2018-05-23 16:43 Pacific Daylight Time
Nmap scan report for zsport.com (206.188.192.217)
Host is up (0.074s latency).
rDNS record for 206.188.192.217: vux.netsolhost.com
Not shown: 996 closed ports
PORT    STATE SERVICE
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 3.50 seconds

C:\Windows\System32>
```

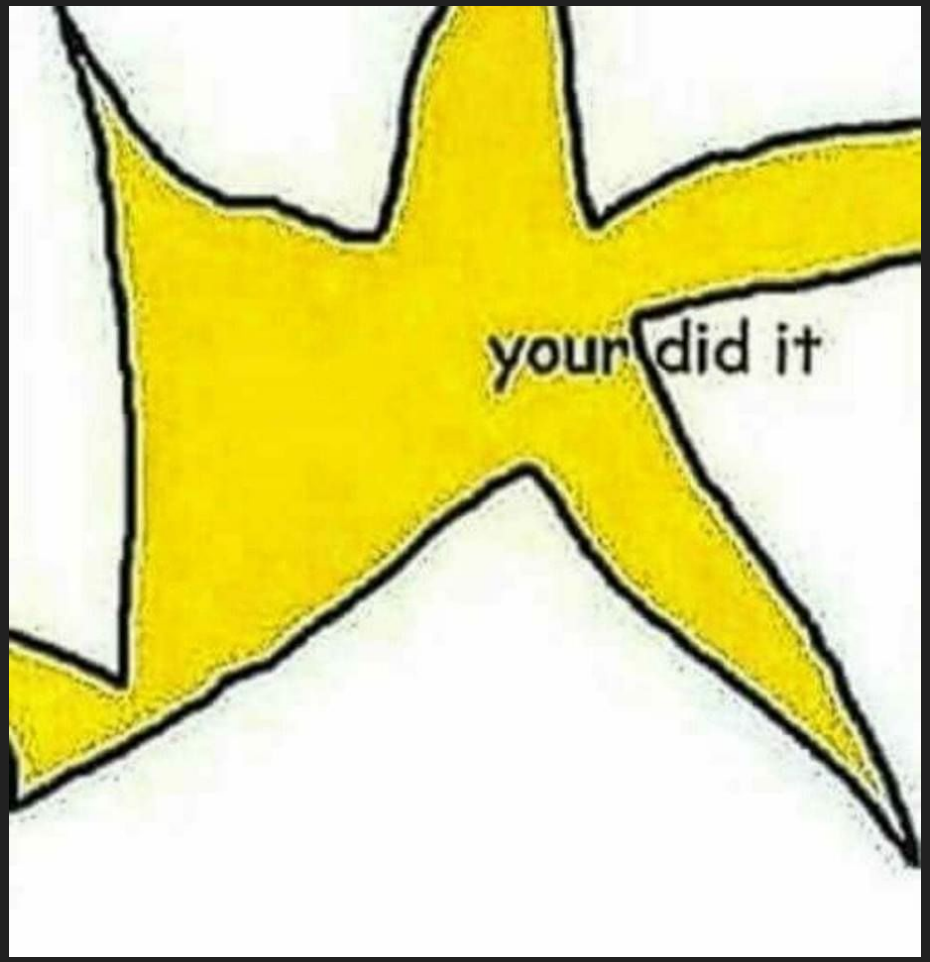
This web server is also running ssh and ftp

NMAP from Python

- Version #1: Using asyncio.create_subprocess_exec

```
@bot.command()
async def nmap(name):
    await bot.say('Running NMAP against `{}`'.format(name))
    args = ['nmap', name]

    proc = await asyncio.create_subprocess_exec(*args,
        stdout=asyncio.subprocess.PIPE)
    (data, _) = await proc.communicate()

    if proc.returncode:
        await bot.say('Error: Could not find host')
    else:
        await bot.say('```{}```'.format(data.decode('utf-8')))
```

Nmap Integration Complete!



Full Bot Demo

Questions?

Appendix

- Asynchronous DNS Lookups
- Global Installation
- Starting on Boot

Asynchronous DNS Lookups

- As mentioned earlier, gethostbyname is blocking
- This can be solved on non-Windows systems with the aiodns package

```
~> pip3 install aiodns
```

- dnsquery() Version #2

```
import aiodns
DNS_RESOLVER = aiodns.DNSResolver()
async def dnsquery(host):
    try:
        hosts = await DNS_RESOLVER.query(host, 'A')
        await bot.say('Ip Addr for `{}` is `{}`'.format(host,
            hosts[0].host)
        return hosts[0].host
    except aiodns.error.DNSError as err:
        await bot.say('Error: {}'.format(err.args[1]))
```

Asynchronous DNS Lookups

- This does not properly handle raw IP addresses on its own
  - i.e. await hostresolve(chan, '8.8.8.8') fails, when it did not beforehand
- You can solve this issue using the **ipaddress** library:

```
import ipaddress

…
    try:
        ipaddress.ip_address(host)
        await bot.say('`{}` is a valid IP Address'.format(host))
        return host
    catch ValueError:
        pass
…
```

Installing Globally

- Test your script on your local machine
- You can install your script globally (if just one file) by adding to the top:

```
#!/usr/bin/env python3
```

- You can then run:

```
~> chmod +x my_bot.py
~> sudo cp my_bot.py /usr/local/bin/
~> my_bot.py
```

- This will allow you to start up your script from anywhere on your system!

Starting On Boot With Systemd

- Ideally, you want your service to run in the background
- This step is platform specific, but most Linux systems use systemd
- Create the following init file:


```
[Unit]
Description=Network Recon Discord Bot

[Service]
ExecStart=my_bot.py

[Install]
WantedBy=multi-user.target
```

- Finish this off by running:

```
~> sudo systemctl enable my_bot.service
~> sudo systemctl start my_bot.service
```