YOU CAN'T CHEAT TIME...

Finding foes and yourself with **latency trilateration**.

Lorenzo 'Lopoc' Cococcia | DEFCON 31

INTRO | **whoami**

Lorenzo 'Lopoc' Cococcia

twitter: **@lopoc_**

GitHub: **github.com/lopoc**

INTRO | **Geolocation***

Since the dawn of time, humans have been driven to **discover new ways** of determining **<u>their own location</u>**, and the **<u>location of potential threats.</u>**

***** the process or technique of identifying the geographical location of a person or device by means of digital information processed via the internet.***

INTRO | Current Geolocation - **Limitations**

- **Target is behind a cloud provider (eg cloudFront)**

  It's like being behind a reverse proxy, IPs are almost useless and IP-based geolocation services too

- **Client won't share location!**
  But that's another story :)

DEF CON

THEORY

THEORY | **Physics 101**

Distance, Speed & Time

DISTANCE

= SPEED × TIME

DEF CON

THEORY | **Physics 101**

SPEED OF LIGHT

Nothing* can move faster than the speed of light (C)

$$C \sim 300.000.000 \text{ m/s}$$

*unless it's carrying no information (not our case)

DEF CON

THEORY | **Physics 101**

LATENCY ≠ SPEED

Latency is the **time** it takes for the first bit of information to **travel from one point to another.**

DEF CON

THEORY | **Physics 101**

LATENCY ≠ SPEED

Latency, obeys the laws physics!

Since data can’t move faster than light...

latency cannot be arbitrarily low

THEORY | **Math 101**

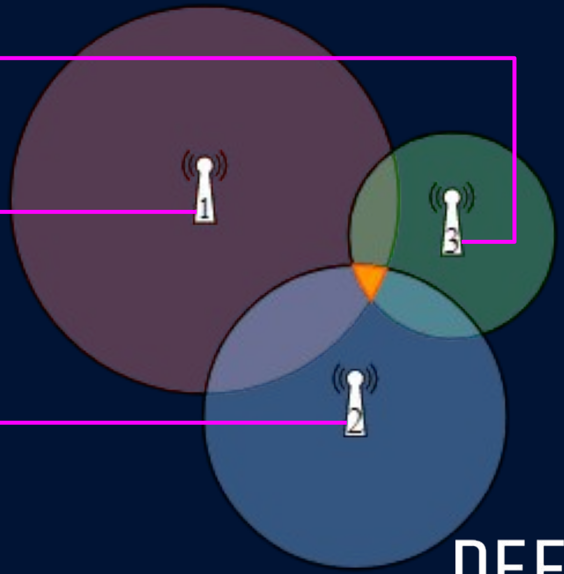
TRILATERATION RECIPE

Points

Distances

and

Target



DEF CON

THEORY | **Math 101**

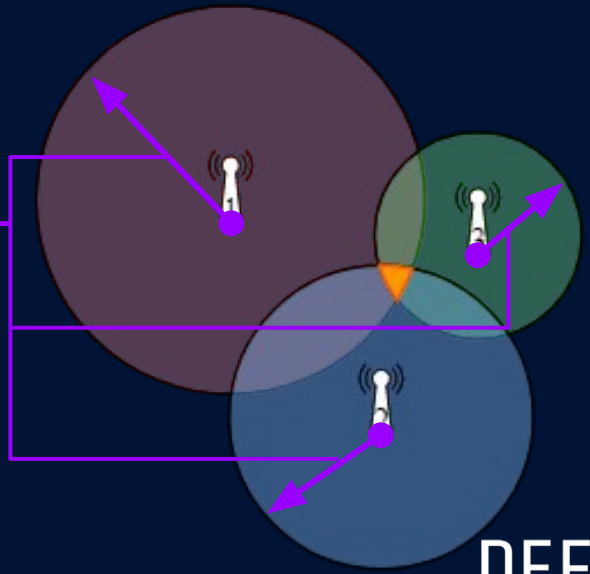
TRILATERATION RECIPE

Points

Distances

and

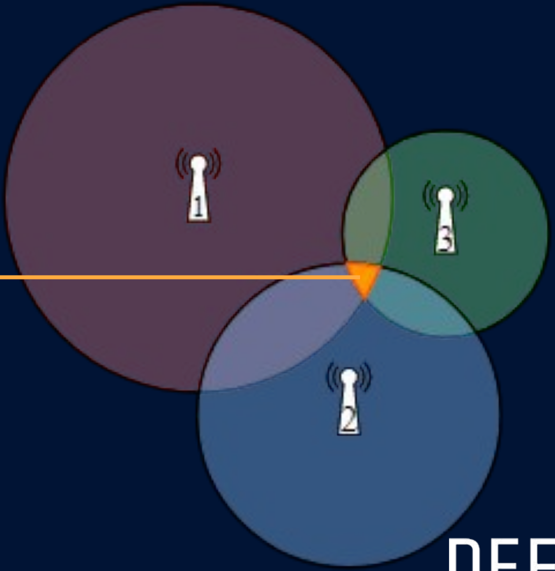
Target



THEORY | **Math 101**

TRILATERATION RECIPE

Points
Distances
and
Target

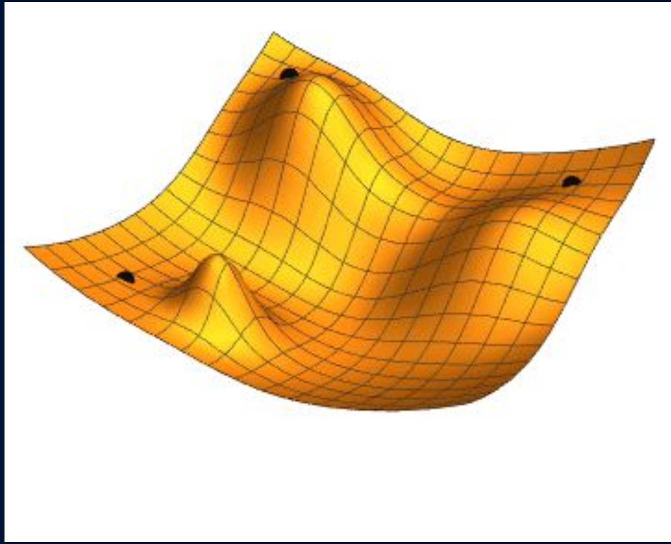


DEF CON

THEORY | **Math 101**

Optimizer.Minimizer*

Is a function that minimizes a scalar function of one or more variables.



DEF CON

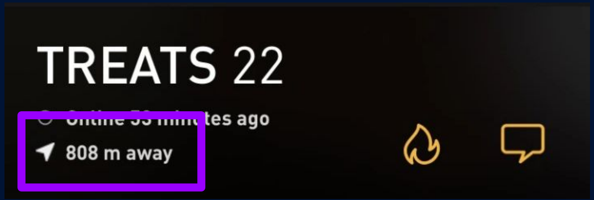
EXAMPLES | **STALKERS**

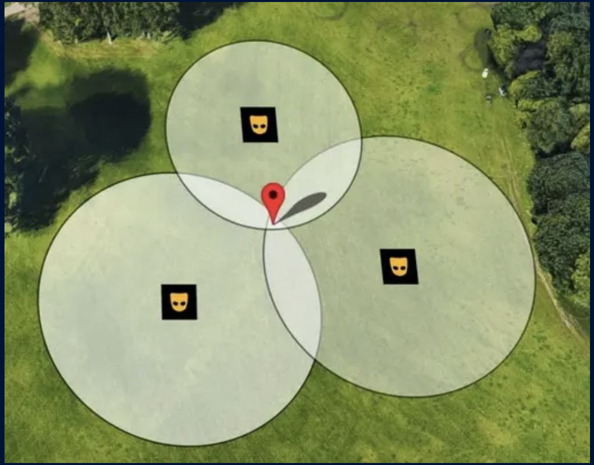

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY MAY 20, 2016 7:00 AM

Gay Dating Apps Promise Privacy, But Leak Your Exact Location

Researchers in Kyoto demonstrate for WIRED how they can precisely track the locations of people using Grindr, Hornet, and Jack'd despite features meant to hide them.





N. P. Hoang, Y. Asano and M. Yoshikawa, "Your neighbors are my spies: Location and other privacy concerns in GLBT-focused location-based dating applications," 2017 19th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea (South), 2017, pp. 851-860, doi: 10.23919/ICACT.2017.7890236.

PRACTICE

DEF CON

PRACTICE | **Latency measurement**

Our first latency

```
> ping fakedomain.foo
```

```
PING www.fakedomain.foo (12.34.56.78): 56 data bytes
64 bytes from 12.34.56.78: icmp_seq=0 ttl=116 time=65.148 ms
...
3 packets transmitted, 3 packets received, 0.0% packet loss

round-trip min/avg/max/stddev = 55.570/59.237/65.148/4.220 ms
```

PRACTICE | **Latency measurement**

```
round-trip min/avg/max/stddev = 55.570/59.237/65.148/4.220 ms
```

round-trip means 2 times the distance (D)
REMEMBER: **data cannot move faster than light (C)**

So the maximum distance is...

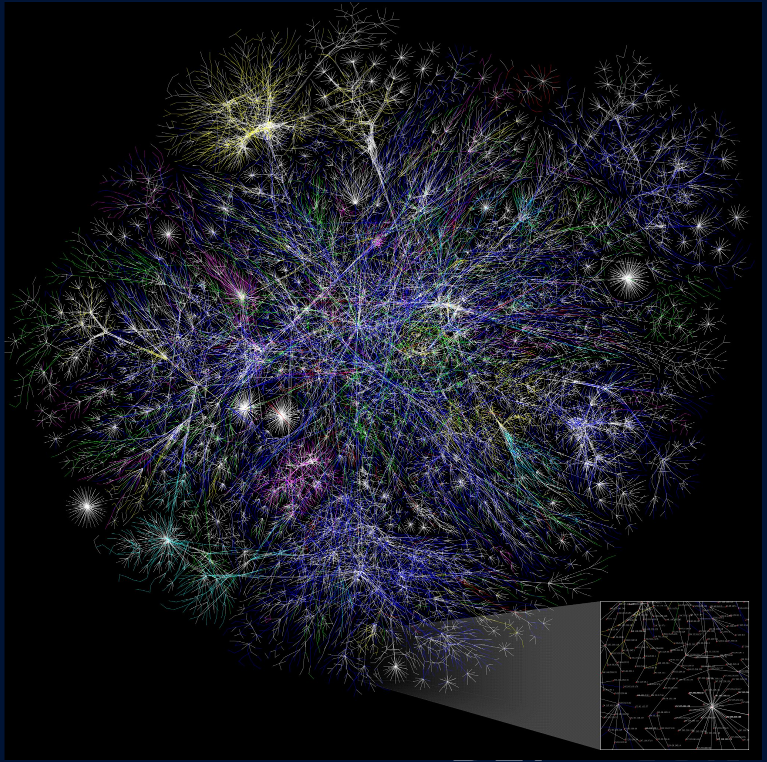
```
2D = 55.57e-3 * C
 D = 55.57e-3 * 300e3 / 2 = 8335 Km
```

Target at D > 8335! -> Physically impossible

PRACTICE | **Limitations of “speed x time” model**

Yes, we can infer distances from latencies, and setting boundaries too, but we cannot use linear equations...

internet is not linear...



PRACTICE | **ML time**

We need a better function that, given latency, returns the expected distance

Some simple ML helps

Simple physics:

```
dist = speed x time
```

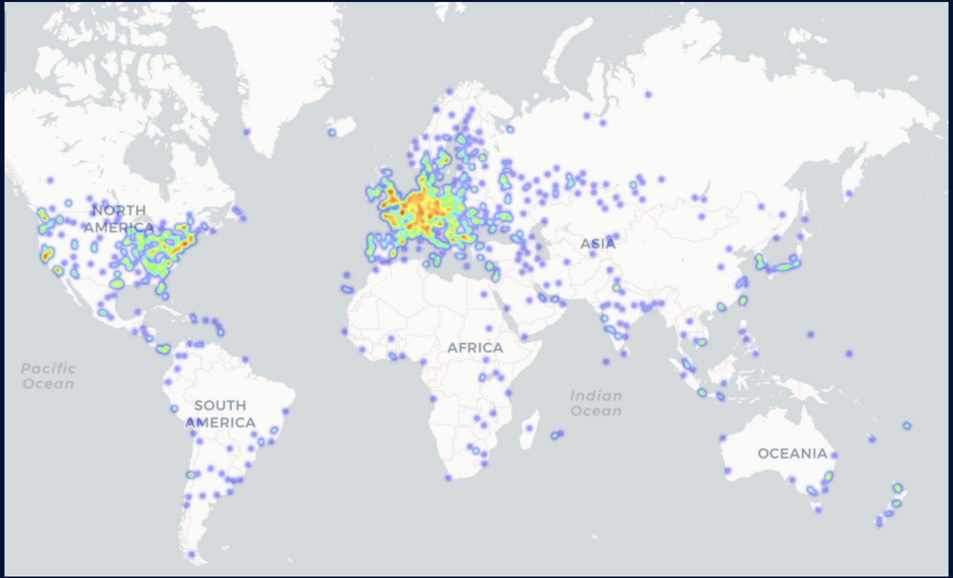
ML:

```
dist = ml_model([time, …])
```

DEF CON

PRACTICE | **ML time**

Obtain as many latency measures as possible



This map shows all locations' latencies I measured

ca. 39k measures*

*[src_country, latency, distance]

PRACTICE | **ML time**

Fit a SVR* model on the data

*Support Vector Regression

Features:

- measurement country_code
- latency_time

Output:

- Distance !!!

Distance Prediction SVR

```
from sklearn import svm
```

```
X = list(x_train)
y = list(y_train.transpose()[0])
```

```
svr = svm.SVR(C=100, epsilon=0.005)
svr.fit(X, y)
```

Sorry mates, gotta run, this is not a ML speech :(

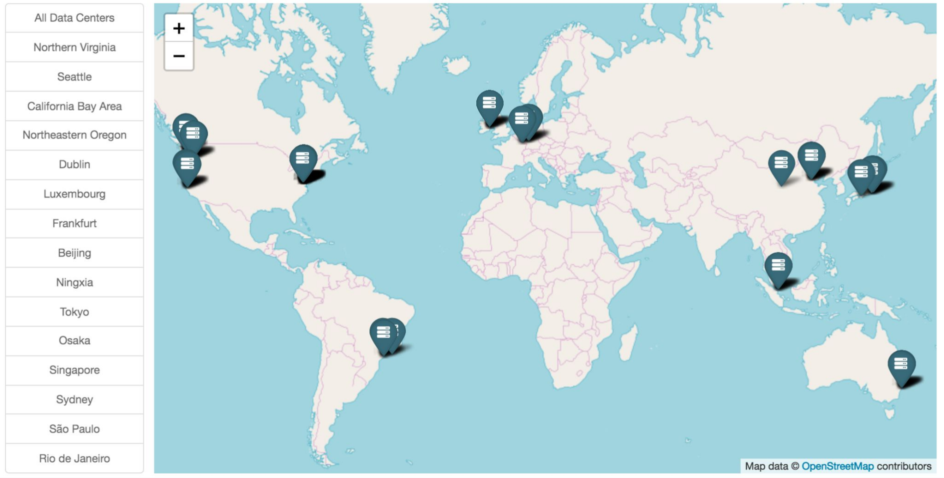
DEF CON

PRACTICE | **Building the infra**

OSINT will help knowing where data centers are...

remember:

we need **POINTS Locations**




```
regions = [
            'ap-northeast-1',
            'eu-north-1',
            'us-west-2',
            'eu-south-1',
            'eu-central-1',
            'us-east-1',
            'us-east-2',
            'ap-northeast-2',
            'eu-west-3',
            'ap-south-1',
            'ca-central-1',
            ]
```

PRACTICE | **Building the infra**

A few lines of code for latency measure (HTTP and HTTPS) <u>no ping anymore!</u>

then deploy it in each point (data center)

```python
def timeprobe(target_url, ntimes=3):
    data = []
    for i in range(ntimes):
        res = subprocess.run(
            [
                "curl",
                "-s",
                "-I",
                "-o",
                "/dev/null",
                "-w",
                "%{time_pretransfer},%{time_starttransfer}",
                target_url,
            ],
            capture_output=True,
            shell=False,
        )
        time_pretransfer, time_starttransfer = eval(res.stdout.decode())
        data.append(time_starttransfer - time_pretransfer)
    return min(data)
```

PRACTICE | **Building the infra**

A few lines of code for latency measurement (HTTP and HTTPS) no ping anymore!

then deploy it in each point (data center)

```
def timeprobe(target_url, ntimes=3):
    data = []
    for i in range(ntimes):
        res = subprocess.run(
            [
                "curl",
                "-s",
                "-I",
                "-o",
                "/dev/null",
                "-w",
                "%{time_pretransfer},%{time_starttransfer}",
                target_url,
            ],
            capture_output=True,
            shell=False,
        )
        time_pretransfer, time_starttransfer = eval(res.stdout.decode())
        data.append(time_starttransfer - time_pretransfer)
    return min(data)
```

PRACTICE | **Do magic!**

Run code against the **target url** and collect latencies, one per known point.

compute distances with the pretrained SVR (latency and country from where the measure has been made)

```python
def timeprobe(target_url, ntimes=3):
    data = []
    for i in range(ntimes):
        res = subprocess.run(
            [
                "curl",
                "-s",
                "-I",
                "-o",
                "/dev/null",
                "-w",
                "%{time_pretransfer},%{time_starttransfer}",
                target_url,
            ],
            capture_output=True,
            shell=False,
        )
        time_pretransfer, time_starttransfer = eval(res.stdout.decode())
        data.append(time_starttransfer - time_pretransfer)
    return min(data)
```

PRACTICE | **Do magic!**

Run code against the **target url** and collect latencies, one per known point.

compute distances with the pretrained SVR (latency and country from where the measure has been made)

```python
def timeprobe(target_url, ntimes=3):
    data = []
    for i in range(ntimes):
        res = subprocess.run(
            [
                "curl",
                "-s",
                "-I",
                "-o",
                "/dev/null",
                "-w",
                "%{time_pretransfer},%{time_starttransfer}",
                target_url,
            ],
            capture_output=True,
            shell=False,
        )
        time_pretransfer, time_starttransfer = eval(res.stdout.decode())
        data.append(time_starttransfer - time_pretransfer)
    return min(data)
```

PRACTICE | **Do magic!**

Now we have 2 arrays and an unknown place:

- **Points** , known points (data-centers coordinates)
- **Dists** , not very accurate distance (computed with our ML model)
- **X** , the unknown target location

DEF CON

PRACTICE | **Do magic!**

Minimize the following equation and find X*

* in the form of [lat, lon]

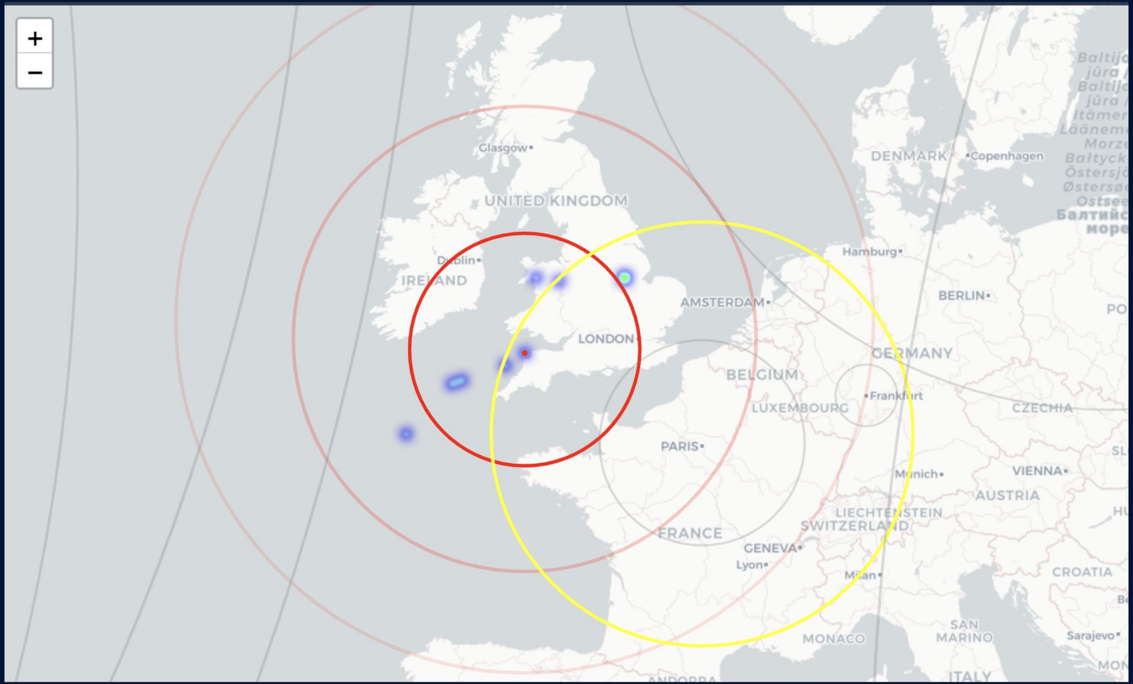
$$X = \min_{x} \sum_{i=1}^{n} |Dists_i - geo\_distance(x, Points_i)|$$

$$x \in [-90, 90] \times [-180, 180]$$

```
minimize(abs(Dists - geo_dist(X, Points)))
```

Start with random values of X and plot the results

PRACTICE | **Do magic!**



Run the minimizer with **different initial values of X** and collect different suitable points.

plot a **heatmap**, perform clustering or anything you like

CYBER STUFF

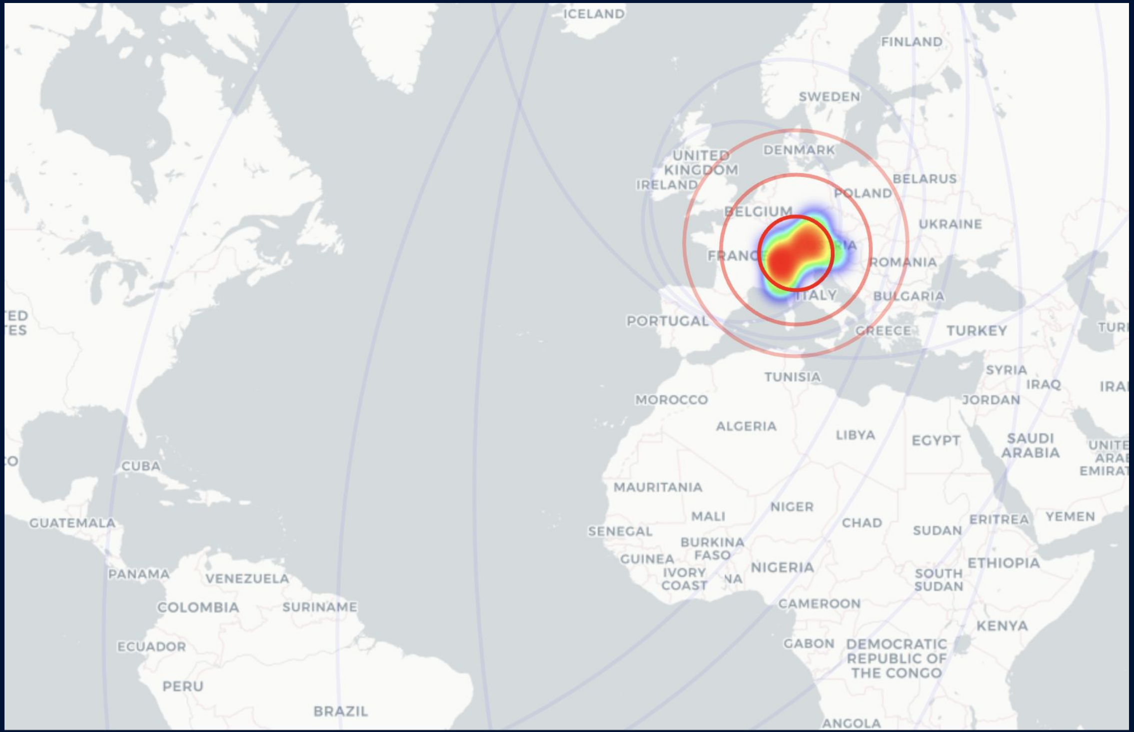
CYBER STUFF | **Defensive**

C2s behind CDNs (eg Cloudfront) are common

Can we tell if it is in US, Russia, China etc… or at least, **can we exclude some large parts of the world?**
Usually no, unless we ask the cloud provider*

*good luck!

CYBER STUFF | **Defensive**



Not precise at all :(
The smallest red circle is centered somewhere in Swiss.

the target is actually located in Rome behind cloudfront !!!

600km error, but better than nothing!!!

CYBER STUFF | **Offensive**

Sandbox Detection

Let's embed our model inside a malware.
It'll measure latencies against N embedded and geographically distributed hosts (using target_country as src_country)

In the presence of a fakenet, we would have almost the same latency (few ms) per each host!

IMPOSSIBLE SOLUTION -> it's a sandbox (maybe)

CYBER STUFF | **Offensive**

Sandbox Detection

They do not overlap!

Maybe a custom or random latency setting on Fakenet* would be great… maybe.

Malware Self-Geolocation

Similar to sandbox detection, the malware could be aware of which part of the world it is running!

Do the same as done with the sandbox, compare the result with the the one you expect...

Is it perfect, is it bullet-proof?
Absolutely no, but promising !

DEMO

Looking beyond the CDNs curtain

Remember, latency is neither the speed nor the bandwidth, and, **you cannot cheat time.**

DEF CON