A Tool to Detect IP Spoofing for System Admins

DARIUS CHITOROAGA, University College London, United Kingdom

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Do, not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Darius Chitoroaga. 2025. A Tool to Detect IP Spoofing for System Admins. *J. ACM* 37, 4, Article 111 (August 2025), 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.

2 Introduction

We aim to create a multi-tool for monitoring subnets for IP spoofing using Machine Learning techniques.

2.1 Investigation

2.1.1 Data Sources

The most important part of any ML solution is having good data. What we have been looking for is RTT and port scan data from a normally operating subnet using `nmap`.

We could also use traceroutes, maybe...

However, this only allows us to use unsupervised ML methods since we have no labelled data, only normal operations. To use supervised ML methods we would need to have accurately labelled data, which is where we run into some issues.

- Security attacks are always evolving and adapting, therefore is we have data that we 'know' is spoofed, attackers could use

different methods in the future that do not have the same signature on the network and therefore completely bypass our models.

- Further, to get data that is spoofed we would need to simulate known spoofing methods which are likely not in

use since they are known.

- To create a simulation of a normally operating subnet is not trivial and would require significant development.

Therefore, for the moment, we have decided to focus solely on unsupervised methods.

Now we run into ethical issues:

Author's Contact Information: Darius Chitoroaga, University College London, London, United Kingdom, darius.chitoroaga.22@ucl.ac.uk.

ACM 2475-1421/2025/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

- Continuous pinging and port scanning is considered unethical since it produces a fairly large load on the target

subnet, using up resources.

- These types of ping are also usually associated with cyberattacks and therefore would rightfully cause stress for

the admin of the target subnet.

Therefore, we should only ping on approved subnets:

- Home networks on a router that we own.
- Research resources that allow researchers to gather data of this type.

Or precompiled datasets, though these may be out of date.

2.1.1.1 Home Subnets

We have gathered half a days worth of pings on a home network but this is may not be very useful since there are at most 8 devices every connected to the subnet.

2.1.1.2 External Sources

PEERING A system that provides safe and easy
access for researchers and educators to the Internet's BGP routing system. However, it is against their terms of use to regularly ping using something like `nmap`.

Censys A platform that provides real-time intelligence about the whole internet.
Requires deeper dive into what kind of data we can access.

Shodan Very similar to Censys, but with more focus on IoT. Calls itself
the "Search Engine for the Internet of Everything".

Shadowserver This foundation provides free daily reports and
datasets on open services and vulnerabilities.

RIPE Atlas This service allows researchers to schedule `pings` and `traceroutes` to
volunteer devices worldwide but does not allow the use of tools like `nmap` or anything that looks like an attack or scan.

2.1.2 Wake-on-LAN

A networking standard that allows a computer to be remotely powered on or awakened from a low-power state, using a specially crafted "magic packet". This technology operates on the link layer so functions separately from IP addresses and relies on the devices' MAC address.

For WoL to work the Network Interface Controller (NIC) must remain powered on during low-power states. This feature must be enabled from the BIOS/UEFI settings on the device.

2.1.3 Detecting Devices Connected Through an Authorised Device

Tracking computers connected through another device is difficult because the "gateway" device is designed to hide them. However, `nmap` provides specific features that can detect "leakage" from hidden devices by analysing packet headers.

2.1.3.1 IP ID Sequence Analysis (`-O -v`)

This is the most reliable method for detecting multiple devices behind a single IP. Every device generates IP packets with a unique ID number. Most operating systems increment this number for every packet they send.

If multiple devices are sharing one IP connection, their IP IDs will interleave or show gaps, confusing Nmap's detection engine.

2.1.3.2 TTL Analysis

If scanning a single IP gives packets with different TTLs it is a strong indicator that there is a router forwarding traffic.

2.1.3.3 The `ipidseq` Script

Nmap has a dedicated script specifically designed to classify the IP ID generation method, which helps in identifying these "zombie" hosts or NATed environments

2.1.3.4 Inconsistent OS Fingerprints

When you run OS detection against a NATed IP, Nmap receives responses from potentially different devices on different ports. Port 80 might be forwarded to a Linux server (TTL 64), while Port 3389 is forwarded to a Windows box (TTL 128).

2.1.4 Fingerprinting

2.1.4.1 Decide invariants

Stable identity features that shouldn't change much

- OS family + version (Linux 5.10)
- Device type hint (rounter/gateway/printer)
- Vendor from MAC address (Technicolor Delivery Technologies Belgium NV)
- Typical open ports
- Typical services per port (nginx on 80/443/8080)

Volatile features

- Uptime
- Exact TTL
- Ephemeral ports (49152)
- Number of closed ports

Should not let volatile features dominate embeddings

2.1.4.2 Normalise data into a canonical representation

Small models need focused data to be useful. Should use a normalised schema such as:

```
{
  "os": "linux",
  "os_version": "4.14",
  "distribution": "openwrt",
  "device_vendor": "technicolor",
  "open_ports": [53,80,139,443,445,631,5000,6699,8080,9000,49152],
  "services": {
    "53": "dns",
    "80": "http-nginx",
    "443": "https-nginx",
    "8080": "http-nginx",
    "139": "netbios",
    "445": "smb",
    "631": "ipp",
    "5000": "upnp"
  },
  "web_stack": ["nginx"],
  "smb_exposed": true,
  "printer_protocol": true,
```

```
  "upnp_exposed": true
}
```

2.1.4.3 Create Multiple Embeddings

Use separate embeddings for different semantic groups. This allows us to explain similarity/difference later. Also means models don't get overloaded.

Group	Purpose
OS embedding	Device class similarity
Port-set embedding	Network posture
Service-stack embedding	Application exposure

OS `Operating system: OpenWrt 19.07, Linux kernel 4.14, embedded router`
Ports `Open TCP ports: 53, 80, 139, 443, 445, 631, 5000, 6699, 8080, 9000, 49152`
Services `Services detected: DNS, nginx HTTP, nginx HTTPS, SMB, NetBIOS, IPP printing, UPnP`
Should test which parts should be paired, since could be useful to pair ports and the services running on them.

2.1.4.4 Fingerprinting using similarity

Networking data is often varied and hosts can change values for any reason, so classifying hosts is unlikely to be successful. Therefore, we rely on cosine similarity between embeddings. The similarities between different embedding classes can have different weights (OS > ports > services).

2.1.4.5 Catching anomalies

Use **delta-based scoring** based on an initial baseline for some host. i.e. for a certain host the baseline may be:

- Ports: {53,80,443,139,445,631,5000,8080}
- Services: nginx, SMB, IPP
- OS: OpenWrt

Structural Anomalies

- New port appears
- Service mismatch (`ssh` on 8080)
- HTTP server changes from nginx to Apache

Semantic Anomalies

- Port profile embedding drifts from baseline
- Service description no longer matches "router-like" (based on device hint)
- OS embedding changes family

2.1.4.6 Architecture (Pipeline)

Nmap ⟶ Normalise ⟶ Feature Groups ⟶ Embeddings ⟶
Similarity vs baseline ⟶ Drift / Anomaly score ⟶ LLM takes action / summary

3 More Ideas

3.1 Using `tcpdump`:

- TTL variance per source IP
- TCP handshake anomalies
- Inconsisten IP ID behaviour

Spoofing indicators:
- Same source IP ⟶ different TTLs
- SYNs that never complete handshakes
- No response to challenge ACKs

Would probably need to heavily filer the packets that make it to our model.

3.2 Using `traceroute`:

- Path stability checks
- Comparing route to claimed source ASN

Indicators:
- Traceroute path doesn't make sense for source IPs ASN.
- Same IP ⟶ different paths in short time.

3.3 Using `hping3`:

References

D Artifact Appendix

In this section we show how to reproduce our findings.