Modern Network Discovery

Modern techniques for internal asset identification

HD Moore

Introduction

HD Moore

Atredis Partners

VP R&D: Penetration testing, reverse engineering, applied research

https:// **atredis.com**

Critical Research Corporation

Founder: Building the Rumble Network Discovery platform

https:// **rumble.run**

Previous work includes Metasploit, Rapid7, BreakingPoint, Digital Defense, DoD

Why Discovery?

An accurate inventory is the **first step** of network security

- **CIS Control 1: Inventory and Control of Hardware Assets**
  - CIS Top 20 Security Controls – https://cisecurity.org/
- *“You can’t secure what you don’t know about”*

Everything builds on having an accurate asset inventory

Older techniques no longer keep up

Room for innovation!

External Asset Discovery

External inventory used to be difficult and time intensive

- Analyze Whois, ARIN, and RIPE datasets for domains and IPs
- Google searches and DNS enumeration to find web servers

Certificate Transparency is now the #1 source of visibility

- If it uses TLS, it is ***immediately*** visible on the internet
- Tons of new companies doing external inventory monitoring
- Open source tools: amass, aquatone, inetdata-ct-tail

See https://speakerdeck.com/hdm/ for related work

Internal Network Discovery

By contrast, internal network discovery has seen little innovation

IT and security tools depend on discovery, but do it poorly

- SNMP (v2/v3) enumeration of network devices
- AD/WMI enumeration of Windows assets
- SSH/Telnet enumeration of devices and servers
- Fallback to ICMP or limited nmap scans

Just *good enough* to drive product features

Internal Discovery Challenges

Discovery is complicated by SDN, virtualization, containers & cloud

Security improvements are killing network visibility

Unmanaged assets are becoming more common

BeyondCorp model upsets the discovery model

Active discovery innovation has stagnated

Active Discovery

Security risks

- Discovery tools spray your credentials across the internal network
- Easily intercepted via man-in-the-middle techniques
- Little to no validation of each endpoint before auth
- Difficult to configure low-rights discovery accounts

Ineffective

- Desktop and server hardening often breaks discovery completely
- Mobile, BYoD, and IoT require different discovery techniques
- Increasing number of systems without known credentials

Passive Discovery

New crop of security products use passive network discovery

- Companies started in OT environments, crossing over to IT
- Fancy fingerprinting based on full stack & behaviors
- Examples: Senr.io, Armis, ForeScout, Claroty, DarkTrace

An improvement from active discovery, still challenges

- Requires remote capture or tap port access to each site
- Discovery only finds things that actively communicate
- Fingerprinting limited to observed network traffic
- Difficult to inventory remote environments

Active Discovery Research

Active discovery research peaked in late 1990s/early 2000s

- Creation of Nmap, Xprobe, p0f, queso, SinFP, Satori, amap
- Nmap implemented many published techniques
- Actual use of these techniques is rare

Low-level fingerprinting is more relevant than ever

- Fingerprint cloud, virtualization, containers, and SDN

Passive fingerprinting is widely adopted

- Business and consumer routers, switches, firewalls, and access points
- DHCP fingerprinting is used for client identification

Improving Active Discovery

No credentials for Active Directory, SNMP, or SSH access

No tap port access or traffic monitoring

Accuracy should degrade incrementally

Support firewalled systems

Support mobile devices

Work across routers

Vulnerability Scanning Parallels

Get the most data from the least number of packets

Build device profiles based on information leaks

Infer information based on profile

Step through probabilities

Use fingerprint databases

Requires research!

NetBIOS Discovery

Unauthenticated UDP port 137 probes

- Status request returns name, domain, and MAC address
- Follow with a name request to obtain all IPv4 addresses
- Identifies multihomed systems & pivot points fast
- Great for PCI CDE segmentation testing

Multi-homed detection is finally being recognized as useful

- Internet-wide scans and surveying tool support
- Shodan now does this by default
- Metasploit support for years ☺

NetBIOS Discovery: Example

```
# nextnet 192.168.0.0/24 | grep nets
    {
      "host": "192.168.0.5",
      "name": "DESKTOP-D4CHFX9",
      "nets": [ "192.168.175.1", "192.168.146.1", "192.168.0.50", "169.254.209.1" ],
      "info": {
        "domain": "WORKGROUP",
        "hwaddr": "a0:36:9f:27:e7:61"
      }
    }
```

https:// github.com/hdm/nextnet

Opportunistic SNMP Discovery

Look for any device that accepts default SNMP v1/2c

- Leverage printers and other devices with low security
- Query the device ARP cache, identifying neighbors
- Dump MAC and VLAN tables of exposed switches
- Find firewalled devices in remote subnets

Limitations and benefits

- Inconsistent but incredibly valuable for further discovery
- ARP cache is limited, but also indicates active hosts

Opportunistic SNMP Discovery: ARP Cache Example

```
$ snmpwalk -v 2c -c public 192.168.30.154  .1.3.6.1.2.1.3.1.1.2
    iso.3.6.1.2.1.3.1.1.2.1.192.168.30.1      "A0 36 9F 50 77 AA"
    iso.3.6.1.2.1.3.1.1.2.1.192.168.30.130    "BC A8 A6 C0 03 01"
```

SNMP Cache + Spoofing = Remote ARP Scans

Abuse a single SNMP-enabled device in a remote subnet

Spoof traffic from its neighbors to populate the cache

- Poll the cache to enumerate the entire remote subnet
- SNMP device must reach out to each potential IP
- Use ICMP ECHO_REQUEST or TCP SYN/SYN-ACK

Blocked by rp_filter=1 / rp_forward=0 on Linux routers

Remote SNMP ARP Scan: Example

```
# nmap --send-ip -e eth0 -S snmp-host -PI target-net.0/24
# snmpwalk -v 2c -c public 192.168.30.154  .1.3.6.1.2.1.3.1.1.2
iso.3.6.1.2.1.3.1.1.2.1.192.168.30.1 = Hex-STRING: A0 36 9F 70 77 D1
iso.3.6.1.2.1.3.1.1.2.1.192.168.30.3 = Hex-STRING: 10 DA 43 A7 53 EC
iso.3.6.1.2.1.3.1.1.2.1.192.168.30.26 = Hex-STRING: 3C 28 6D 90 2F A4
iso.3.6.1.2.1.3.1.1.2.1.192.168.30.28 = Hex-STRING: 18 B4 30 18 0E 4A
iso.3.6.1.2.1.3.1.1.2.1.192.168.30.29 = Hex-STRING: 18 B4 30 D7 F5 3B
...
```

Remote ARP Scan: SNMP + UPnP NOTIFY

Abuse UPnP NOTIFY unicast requests to avoid spoofing

Send a NOTIFY with Location: http://snmp-host to all devices

- Limits discovery to UPnP devices in the subnet
- Doesn't require UPnP on the SNMP device

Send NOTIFY requests to the SNMP host with Location: http://ip

- Gets all devices, but depends on UPnP on the SNMP device
- Doesn't require UPnP on subnet devices

Thank you Martin Zeiser and Aleksandar Nikolich of TALOS!

Remote ARP Scan: SNMP + SIP INVITE

Use SIP INVITE requests to avoid spoofing

Send an INVITE with URI sip://snmp-host to all IP phones

- Limits discovery to SIP devices in the subnet
- Doesn’t require SIP on the SNMP device

Send NOTIFY requests to the SNMP host with URI sip://ip

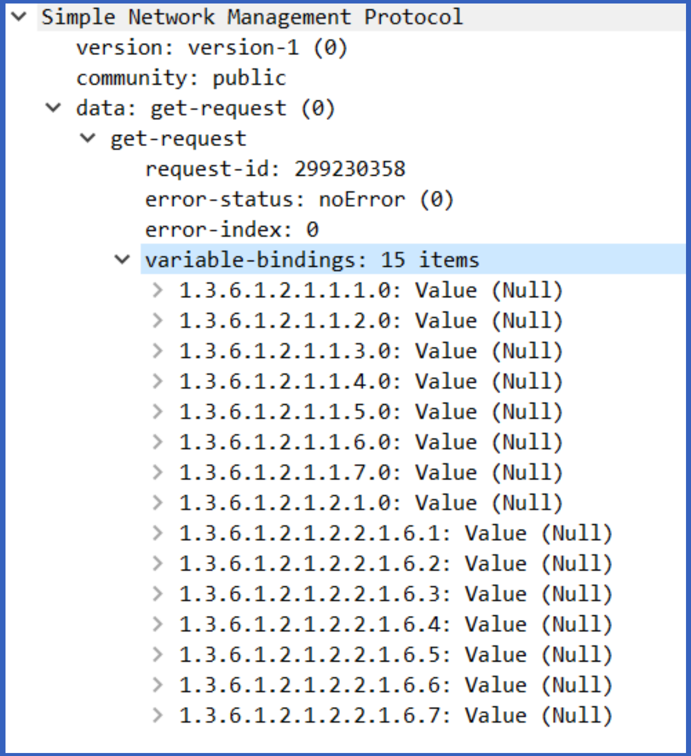
- Gets all devices, but depends on SIP on the SNMP device
- Doesn’t require SIP on subnet devices

Thank you Ricky Lawshae of Trend Micro!

SNMP v1: Stacked GET

SNMP v1 enumeration is slow, but it can be much faster

- Stack multiple queries in the same packet
- Dump sensitive OIDs all at once
- Limited by max response size



```
Simple Network Management Protocol
    version: version-1 (0)
    community: public
  data: get-request (0)
      get-request
        request-id: 299230358
        error-status: noError (0)
        error-index: 0
        variable-bindings: 15 items
          1.3.6.1.2.1.1.1.0: Value (Null)
          1.3.6.1.2.1.1.2.0: Value (Null)
          1.3.6.1.2.1.1.3.0: Value (Null)
          1.3.6.1.2.1.1.4.0: Value (Null)
          1.3.6.1.2.1.1.5.0: Value (Null)
          1.3.6.1.2.1.1.6.0: Value (Null)
          1.3.6.1.2.1.1.7.0: Value (Null)
          1.3.6.1.2.1.2.1.0: Value (Null)
          1.3.6.1.2.1.2.2.1.6.1: Value (Null)
          1.3.6.1.2.1.2.2.1.6.2: Value (Null)
          1.3.6.1.2.1.2.2.1.6.3: Value (Null)
          1.3.6.1.2.1.2.2.1.6.4: Value (Null)
          1.3.6.1.2.1.2.2.1.6.5: Value (Null)
          1.3.6.1.2.1.2.2.1.6.6: Value (Null)
          1.3.6.1.2.1.2.2.1.6.7: Value (Null)
```

SNMP v2: Stacked GETBULK

SNMP enumeration can be ludicrous speed fast with GETBULK

- Use GETBULK to query a range of OIDs with a single request
- Also stack multiple queries into the same packet request
- Enumerate all interfaces/MACs in a single shot
- Grab multiple OIDs while you are at it

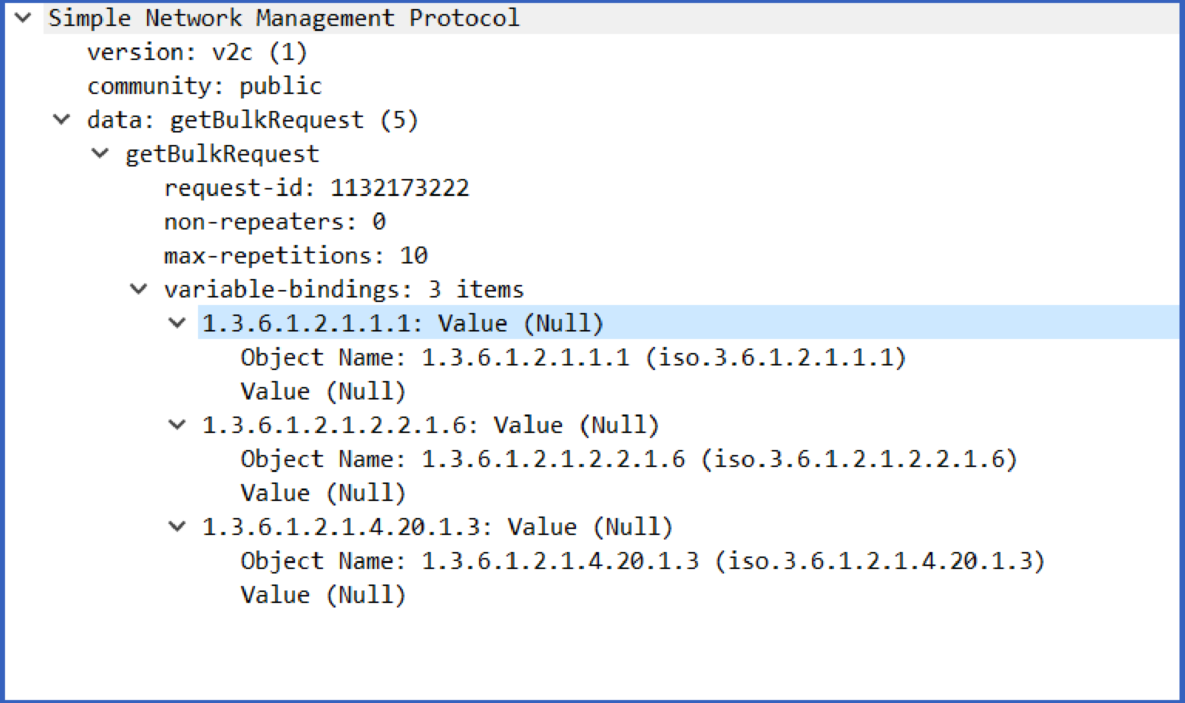
Works on most SNMP v2 servers,

- SNMP max response size may be hit, be careful
- Some implementations drop stacked queries
  - IBM iSeries are the only example so far

SNMP v2: Stacked GETBULK Example

Use stacked GETBULK to query multiple ranges at once

- Grab all the standard system OIDs
- Also all the MACs and IPs!



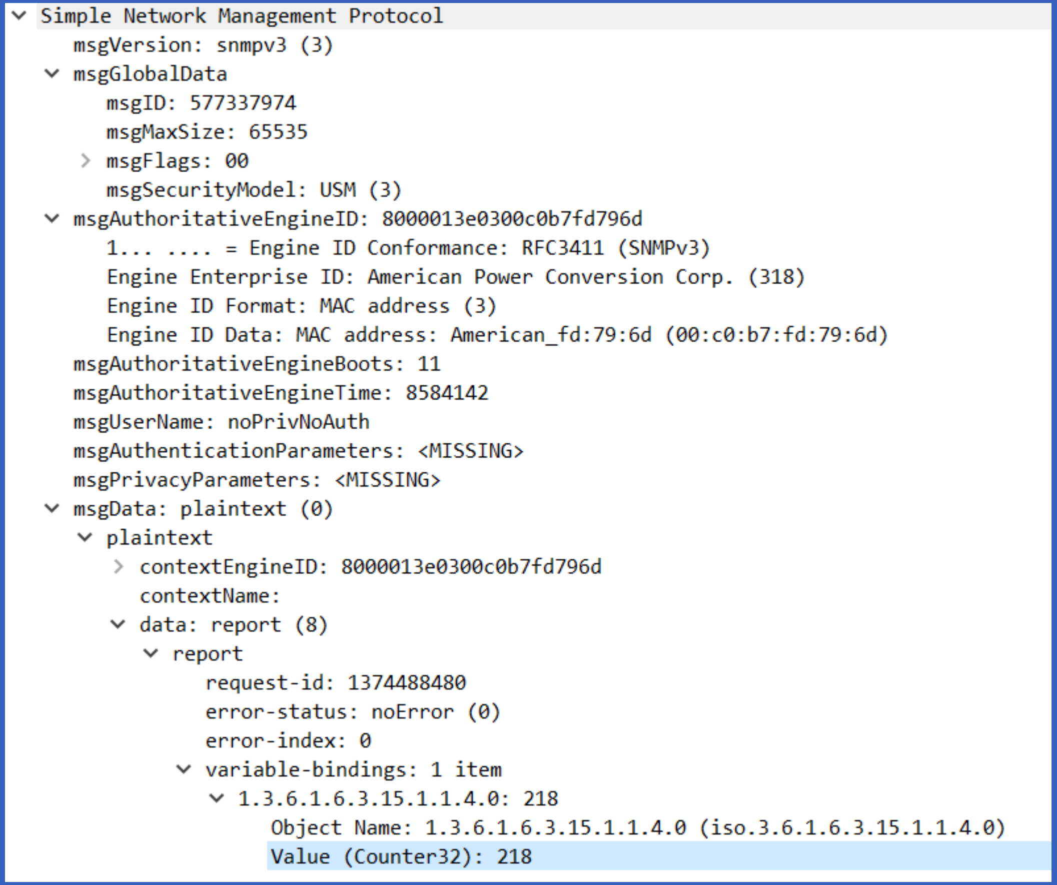
```
Simple Network Management Protocol
    version: v2c (1)
    community: public
    data: getBulkRequest (5)
        getBulkRequest
            request-id: 1132173222
            non-repeaters: 0
            max-repetitions: 10
            variable-bindings: 3 items
                1.3.6.1.2.1.1.1: Value (Null)
                    Object Name: 1.3.6.1.2.1.1.1 (iso.3.6.1.2.1.1.1)
                    Value (Null)
                1.3.6.1.2.1.2.2.1.6: Value (Null)
                    Object Name: 1.3.6.1.2.1.2.2.1.6 (iso.3.6.1.2.1.2.2.1.6)
                    Value (Null)
                1.3.6.1.2.1.4.20.1.3: Value (Null)
                    Object Name: 1.3.6.1.2.1.4.20.1.3 (iso.3.6.1.2.1.4.20.1.3)
                    Value (Null)
```

SNMP v3: Unauthenticated Discovery

SNMP v3 is often ignored, for some good reasons

- Requires a username to be guessed (brute forced)
- Requires at least 1, sometimes 2 strong passwords
- Systems vuln to NULL HMAC are getting rare
- No “default” credentials
- No fun?

SNMP v3: Unauthenticated Fields



```
v Simple Network Management Protocol
    msgVersion: snmpv3 (3)
  v msgGlobalData
      msgID: 577337974
      msgMaxSize: 65535
    > msgFlags: 00
      msgSecurityModel: USM (3)
  v msgAuthoritativeEngineID: 8000013e0300c0b7fd796d
      1... .... = Engine ID Conformance: RFC3411 (SNMPv3)
      Engine Enterprise ID: American Power Conversion Corp. (318)
      Engine ID Format: MAC address (3)
      Engine ID Data: MAC address: American_fd:79:6d (00:c0:b7:fd:79:6d)
    msgAuthoritativeEngineBoots: 11
    msgAuthoritativeEngineTime: 8584142
    msgUserName: noPrivNoAuth
    msgAuthenticationParameters: <MISSING>
    msgPrivacyParameters: <MISSING>
  v msgData: plaintext (0)
    v plaintext
      > contextEngineID: 8000013e0300c0b7fd796d
        contextName:
      v data: report (8)
        v report
            request-id: 1374488480
            error-status: noError (0)
            error-index: 0
          v variable-bindings: 1 item
            v 1.3.6.1.6.3.15.1.1.4.0: 218
                Object Name: 1.3.6.1.6.3.15.1.1.4.0 (iso.3.6.1.6.3.15.1.1.4.0)
                Value (Counter32): 218
```

msgMaxSize

EngineID

EngineBoots

EngineTime

InvalidEngineIDCount

SNMP v3: Decoding the EngineID

The EngineID encodes an EnterpriseID, sometimes much more

- EnterpriseID returns an integer associated with the vendor
- Multiple formats for the EngineID, the type field often lies
- MAC address is returned for Cisco, Juniper, tons more!
- Sometimes the MAC is present, but oddly encoded
  - `000001`**`6f002673`**`fffe`**`a90d`**`16`
  - `000007ba03`**`2c30336ea3f9`**`01`
  - `00000983`**`001ba9bc1a75`**`0001`

- Net-SNMP exposes per-host unique “random” ID
  - Use to identify unique hosts, also exposes Creation Time

Rogue Egress Detection

Identify hosts with unauthorized internet connections

- Spoof packets from an internet address to internal hosts
- Embed cookies into the payloads and packet fields
- Listen on the internet side for responses

Outlined in “Rogue Network Link Detection” paper from 2005

Identify vendor-connected systems and rogue VPN use

Original methods don’t always escape NAT gateways

Rogue Egress Detection: Example

```
root@internal # hping3 --spoof internet-host -1 internal-target
root@internal # hping3 --spoof internet-host -S -p 80 internal-target
```

```
root@internet-host # tcpdump –vn icmp or src port 80
17:35:18.666205 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 44)
rogue.external.80 > internet.2101: Flags [S.]
```

Rogue Egress Detection with UDP

Build on the original work by using UDP payloads instead

- These bypass NAT filters that drop wrong-state TCP and ICMP
- Payloads like NetBIOS also reflect hostname & IP information
- Abuse UPnP NOTIFY for HTTP-based egress detection
- Listen on the internet side for responses

Rogue Egress Detection with UDP: Example

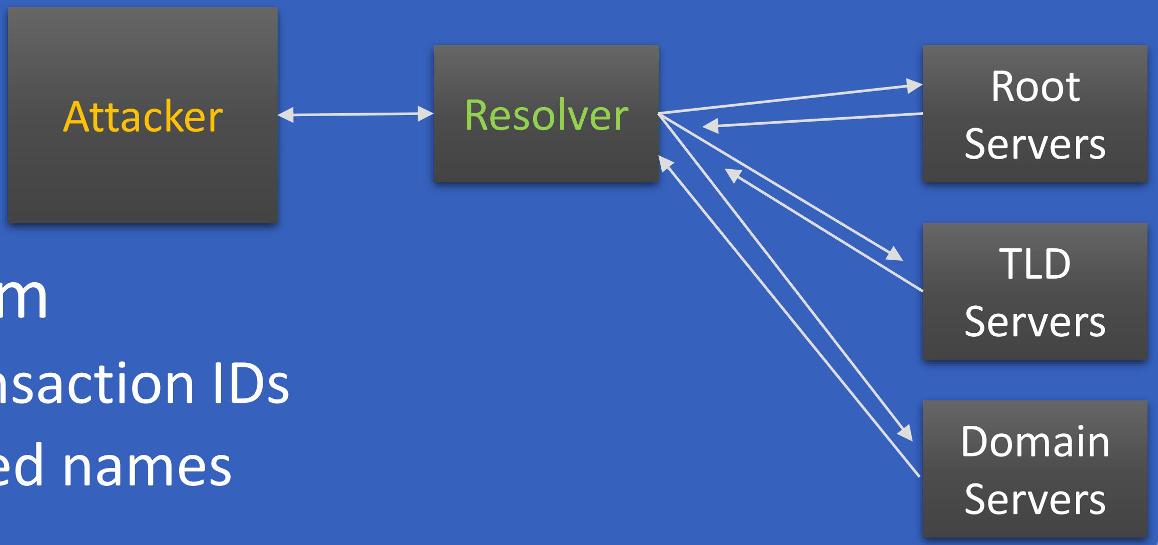
```
root@internal # zmap -S internet-host -M udp \
                -p 137 --probe-args=file:netbios_137.pkt -N 100 \
                target-net/24
```

```
root@internet-host # tcpdump –vn udp port 137
04:27:55.342360 IP (tos 0x0, ttl 112, id 30648, offset 0, flags [none],
proto UDP (17), length 185)
    egress-ip.137 > internet-host.47663: NBT UDP PACKET(137): QUERY;
POSITIVE; RESPONSE; UNICAST
```

DNS Resolvers

Resolvers traverse DNS to find answers

- At every level, find a NS record
- Resolve the NS, query the NS
- Repeat until resolved or error



Resolvers leak information upstream

- Source IP address, source port, transaction IDs
- Timing information through encoded names
- Client subnet source via EDNS0

DNS Upstream Resolver Identification

Query a custom domain on an open resolver

- Encode the source IPs into the query response
- Identify upstream DNS and misconfigurations

Determine what internal systems use what DNS settings

Also identifies weak source port + transaction ID usage

DNS Upstream Resolver Identification: Example

```
$ dig -t a <encoded>.helper.domain @internal-target


;; ANSWER SECTION:
t0db910b0bdb910b0bdb9….helper.domain. 59 IN A 74.125.115.141
```

DNS Resolvers: Ping & RTT

Encode a timestamp into a DNS request, spray it at resolvers

Receive responses and calculate the RTT from the difference

Repeat with timestamps encoded into subdomains

- 0x<timestamp>.
- 0x<timestamp>.bigdom.com
- 0x<timestamp>.subdom.bigdom.com
- 0x<timestamp>.nothere.subdom.bigdom.com

Hurray, a DNS traceroute!

DNS Resolvers: Response Addresses

DNS caching resolvers often query other resolvers

Encode the DNS destination into the Question

- 0x<timestamp>-8-8-8-8.
- 0x<timestamp>-8-8-8-8.bigdom.com

A response from 1.2.3.4 for 0x<timestamp>-8-8-8-8?

- We found a multi-homed DNS resolver!
- Useful for finding egress points

DNS Resolvers: Upstream Resolvers

DNS servers sometimes reply from other IP addresses

Sending a query to 192.168.0.1 might forward to 8.8.8.8

The server at 8.8.8.8 (anycast) may resolve from 4.4.8.8

A custom DNS server makes this easy

- Return the query source IP (and/or port+XID) in the response
- Used for weak XID and static source port detection
- Also leaks the upstream resolvers
- Used for Tor demasking too

DNS Resolvers: Analyzing Upstream

Queries to 8.8.8.8 come out of various Google IPs

- 173.194.95.135
- 173.194.101.196
- 173.194.95.141

These resolvers might pop out of more than one ISP

- Find multi-homed DNS resolvers
- Identify VPN configurations
- Detect endpoint protection
    - Zscaler, Comodo, etc

DNS Resolvers: Avoiding Filters & Caches

Clients, caches, resolvers, authoritative servers all filter requests

Caching happens everywhere, even transparent proxies

Proxies also inspect, filter, and replace parameters

Cache bust and protect queries with lame crypto

- 32-bit XOR keys for encoding addresses and timestamps
- Switch to actual block ciphers where opsec matters

DNS Resolvers: Digging Deeper

Resolvers determine what to query next by NS record

Bounce queries into private networks via dynamic NS

Leak IPv6 global addresses, poke internal systems

A custom DNS server also makes this easy

- Query <random>.192-168-0-1.mydom.com
- Return a NS for 192.168.0.1
- Monitor the timing

DNS Resolvers: Client Subnet Extension

Some caching resolvers support EDNS0 Client Subnets

- The resolver sends your source IP subnet to the final NS

```
$ nslookup target.subdom.domain.com 8.8.8.8
  you -> (“A target.subdom.domain.com”) -> 8.8.8.8
  8.8.8.8 -> (“NS domain.com”) + You/24 -> ROOT NS
  8.8.8.8 -> (“NS domain.com”) + You/24 -> .COM TLD NS
  8.8.8.8 -> (“NS domain.com”) + You/24 -> .DOMAIN.COM NS
  8.8.8.8 -> (“NS domain.com”) + You/24 -> .SUBDOM.DOMAIN.COM NS
```

- The 8.8.8.8 example is a real one
  - Google sends your IP range to some domain servers you query
  - This can be a privacy risk, other open resolvers don’t do it

DNS Resolvers: Client Subnet Example

Query a resolver for e0<random>.v1.nxdomain.us

```
$ dig -t a e00eca39330eca39330eca39330ecac6ccce6239321b9e6aa2647c05ff.v1.nxdomain.us.

e00eca39330eca39330eca39330ecac6ccce6239321b9e6aa2647c05ff.v1.nxdomain.us. 59
        IN CNAME c00eca39330ecb393316ca39330eca39330eca3933f135b10df3ca.v1.nxdomain.us.
```

CNAME response leaks the Client Subnet received from 8.8.8.8

- Lots of fancy decoding later we get

  Subnet: 136.42.153.0

  Mask: 24 bits (/24)

  Family: 1   Code: 0   Scope: 0

MAC Address Fingerprinting

Multiple research papers over the years, but fragmented focus

- Many target WiFi, Bluetooth, and mobile devices
- No general purpose MAC fingerprinting dataset

DeepMAC is one exception, but still early

- Over 20 years of OUI registry history
- Small number of fingerprints

MDNS aka Bonjour aka avahid

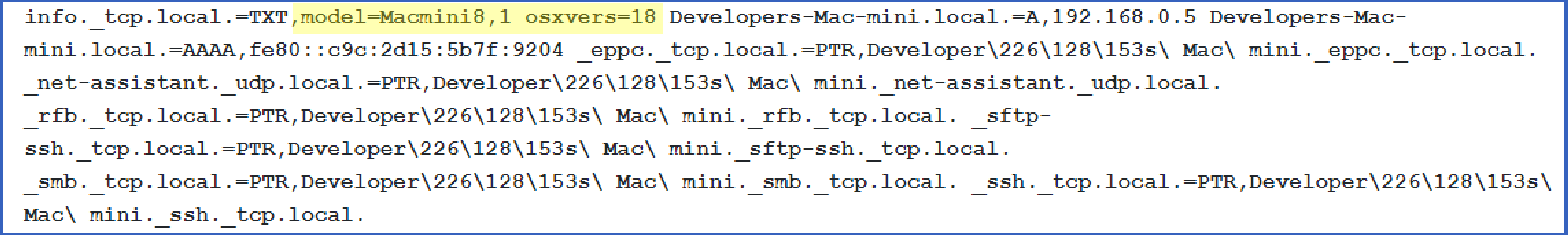
Often overlooked, provides valuable data without auth

- Precise Mac OS X version on local subnets
- MAC address via specific response fields
- List of IPv4 and IPv6 addresses
- Hostname and domain
- Available services

MDNS: Local Examples

Exact macOS version and hardware identification

Contributed via https://github.com/rapid7/recog/



```
info._tcp.local.=TXT,model=Macmini8,1 osxvers=18 Developers-Mac-mini.local.=A,192.168.0.5 Developers-Mac-
mini.local.=AAAA,fe80::c9c:2d15:5b7f:9204 _eppc._tcp.local.=PTR,Developer\226\128\153s\ Mac\ mini._eppc._tcp.local.
_net-assistant._udp.local.=PTR,Developer\226\128\153s\ Mac\ mini._net-assistant._udp.local.
_rfb._tcp.local.=PTR,Developer\226\128\153s\ Mac\ mini._rfb._tcp.local. _sftp-
ssh._tcp.local.=PTR,Developer\226\128\153s\ Mac\ mini._sftp-ssh._tcp.local.
_smb._tcp.local.=PTR,Developer\226\128\153s\ Mac\ mini._smb._tcp.local. _ssh._tcp.local.=PTR,Developer\226\128\153s\
Mac\ mini._ssh._tcp.local.
```

MDNS: Remote Examples

macOS remote desktop	net-assistant/udp
macOS with iOS sync	companion-link/tcp
IP cameras	psia/tcp
Android phones	38.86.168.192.in-addr.arpa.=PTR,Android-3.local.
Printer w/MAC	Samsung\ SCX-472x\ Series\ \(SEC00159385EBCA\)._http._tcp.local.
Sonos w/MAC	131.0.168.192.in-addr.arpa.=PTR,Sonos-7828CBB27258.local

TLS: Data Mining Certificate Fields

TLS certificates often expose sensitive device information

- Fortinet products expose serial number in the TLS certificate
- SonicWall expose internal IP addresses
- Dell iDRAC exposes Service Tags
- Lots of MAC address exposure
- Easy fingerprinting!

TLS: Data Mining Certificate Fields: Example

CN=UBNT-**F0:9F:C2:E0:81:06**,OU=Technical Support,O=Ubiquiti Networks Inc

CN=synology.com,O=Synology Inc.,L=Taipei,C=TW

CN=4Y3ZXB FA8FCA80803D,OU=Cast,O=Google Inc

CN=561c77aa13…3c8f3034eefe2,O=Palo Alto Networks

CN=Chromecast ICA 11 (Video Assist),OU=Cast,O=Google Inc

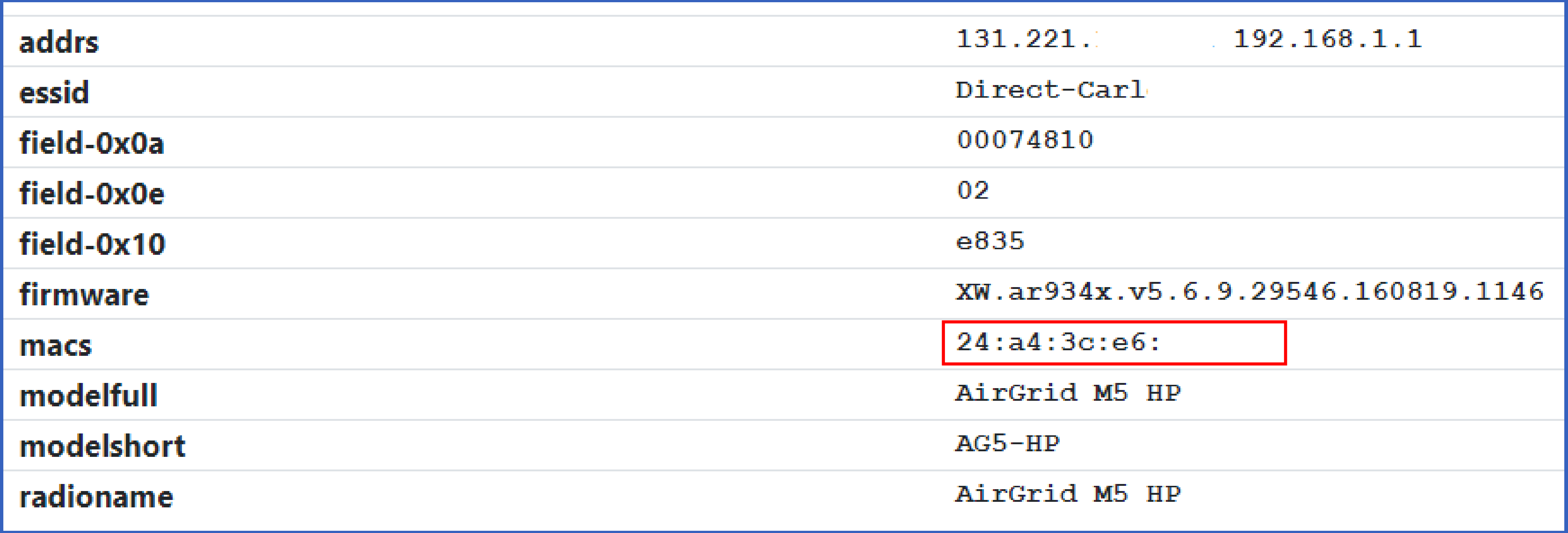
Manufacturer Discovery Protocols

Hundreds of different protocols across the vendors

- Super useful, but poorly documented
- Synology, Ubiquiti, Netgear, ASUS, Brother, etc
- Sometimes broadcast only

Manufacturer Discovery Protocols: Example

Ubiquiti Discovery Protocol



addrs	131.221. 192.168.1.1
essid	Direct-Carl
field-0x0a	00074810
field-0x0e	02
field-0x10	e835
firmware	XW.ar934x.v5.6.9.29546.160819.1146
macs	24:a4:3c:e6:
modelfull	AirGrid M5 HP
modelshort	AG5-HP
radioname	AirGrid M5 HP

MAC Addresses Beyond Layer 2

- SNMP v3 No Auth + SNMP v1/2c Community
- MDNS via name and EUI-64 IPv6
- Device-specific protocols
- SSDP via multiple fields
- DHCP hostname + PTRs
- Device HTML pages
- NetBIOS replies
- TLS certificates

MAC Address Information

MAC addresses tell you the vendor, but also much more!

The device age can be determined by the allocation date

- Date provides a minimum manufacturer on date
- Easy CSV file at https://github.com/hdm/mac-ages
- Full JSON at https://github.com/hdm/deepmac-tracker

OUI is shared between Bluetooth, WiFi, and Ethernet

Match specific device versions based on sub-blocks*

Multihomed Devices

Many systems are connected to more than one network

- Executive laptop on WiFi and corporate Ethernet LAN
- Desktop also connected to a VPC via OpenVPN
- Switch responsible for segmenting VLANs

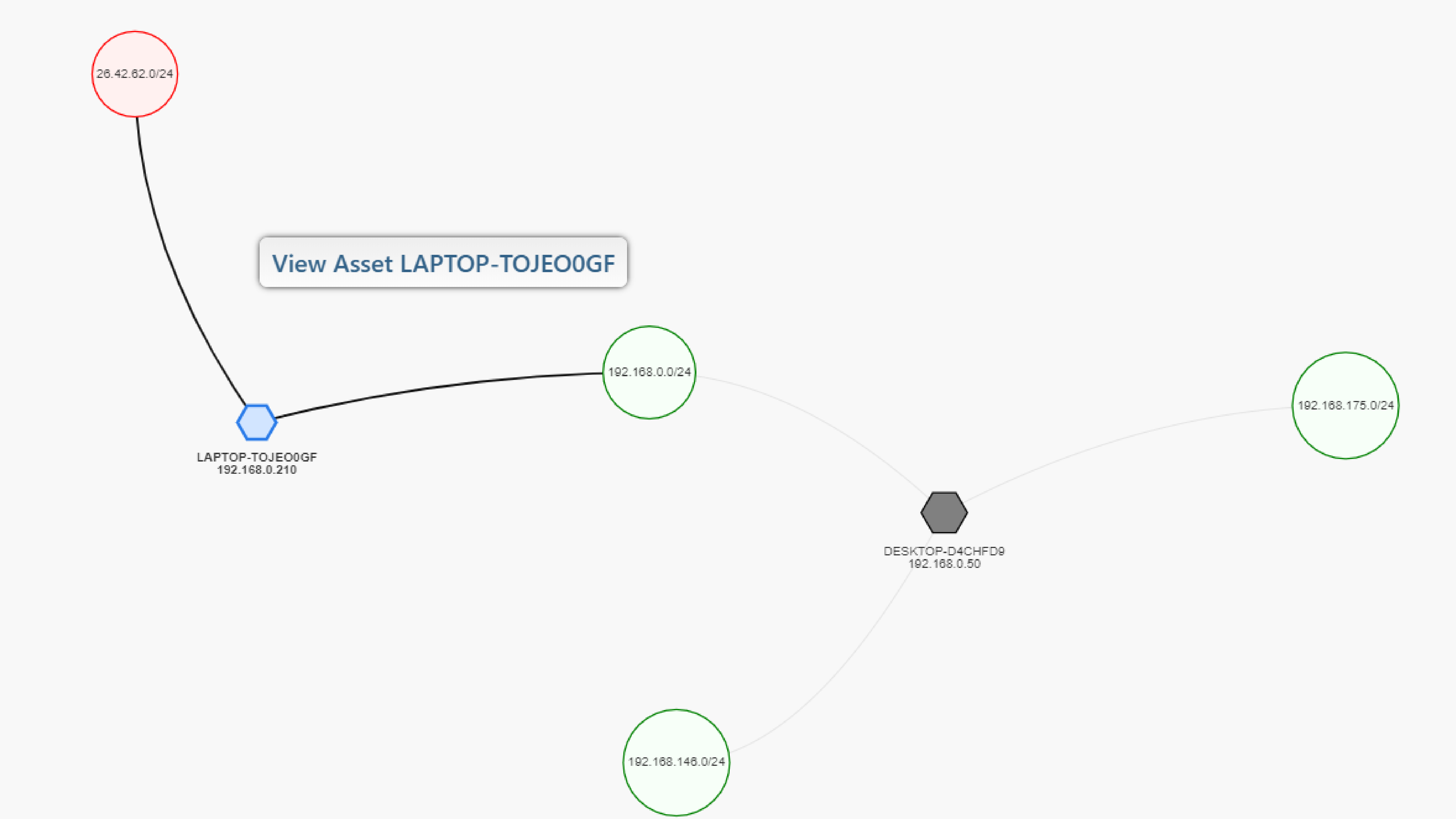
Sometimes these networks are sensitive

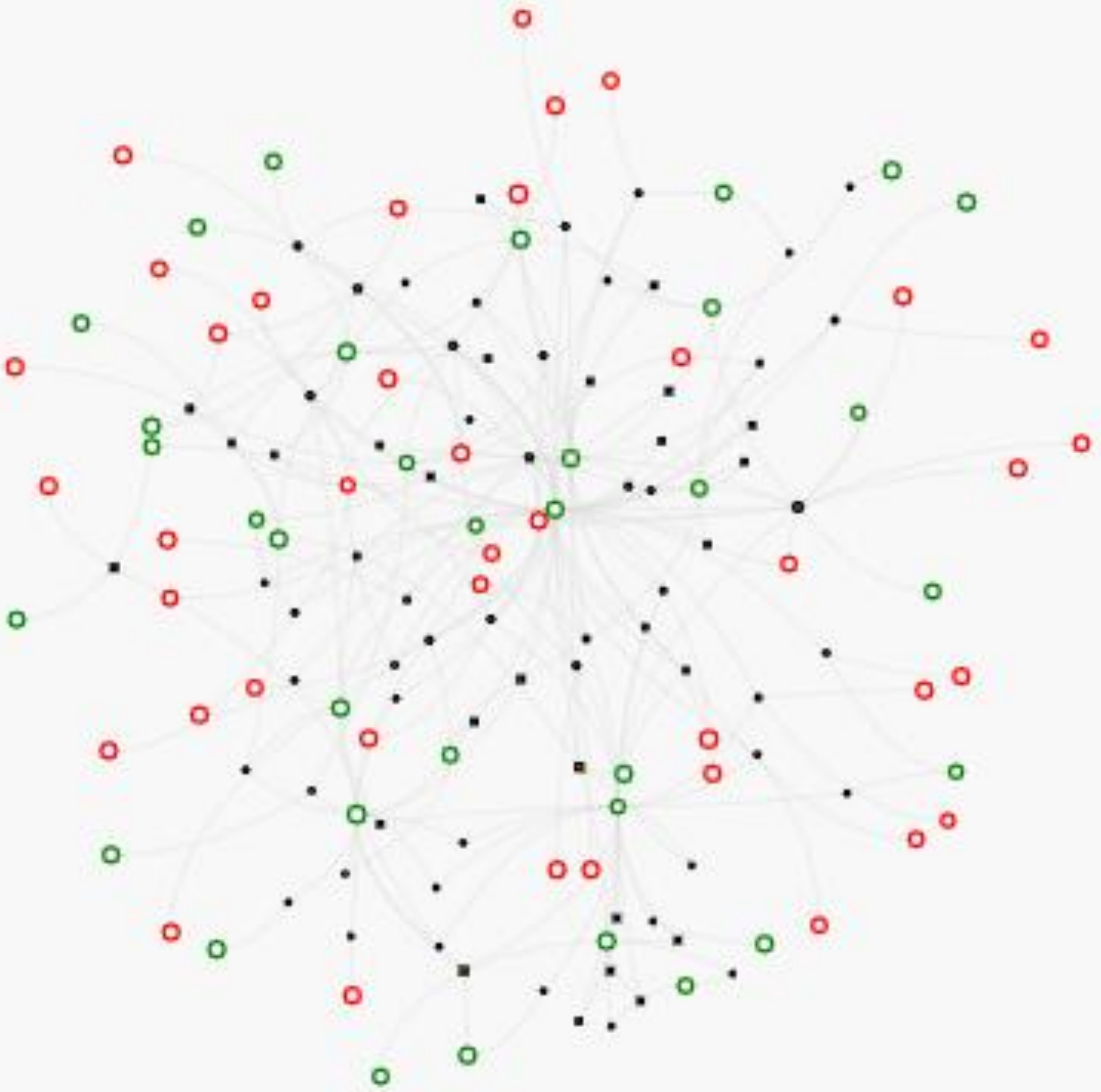
- Low security (guest WiFi) vs high security (LAN)
- PCI cardholder data environment
- Red/Black networks

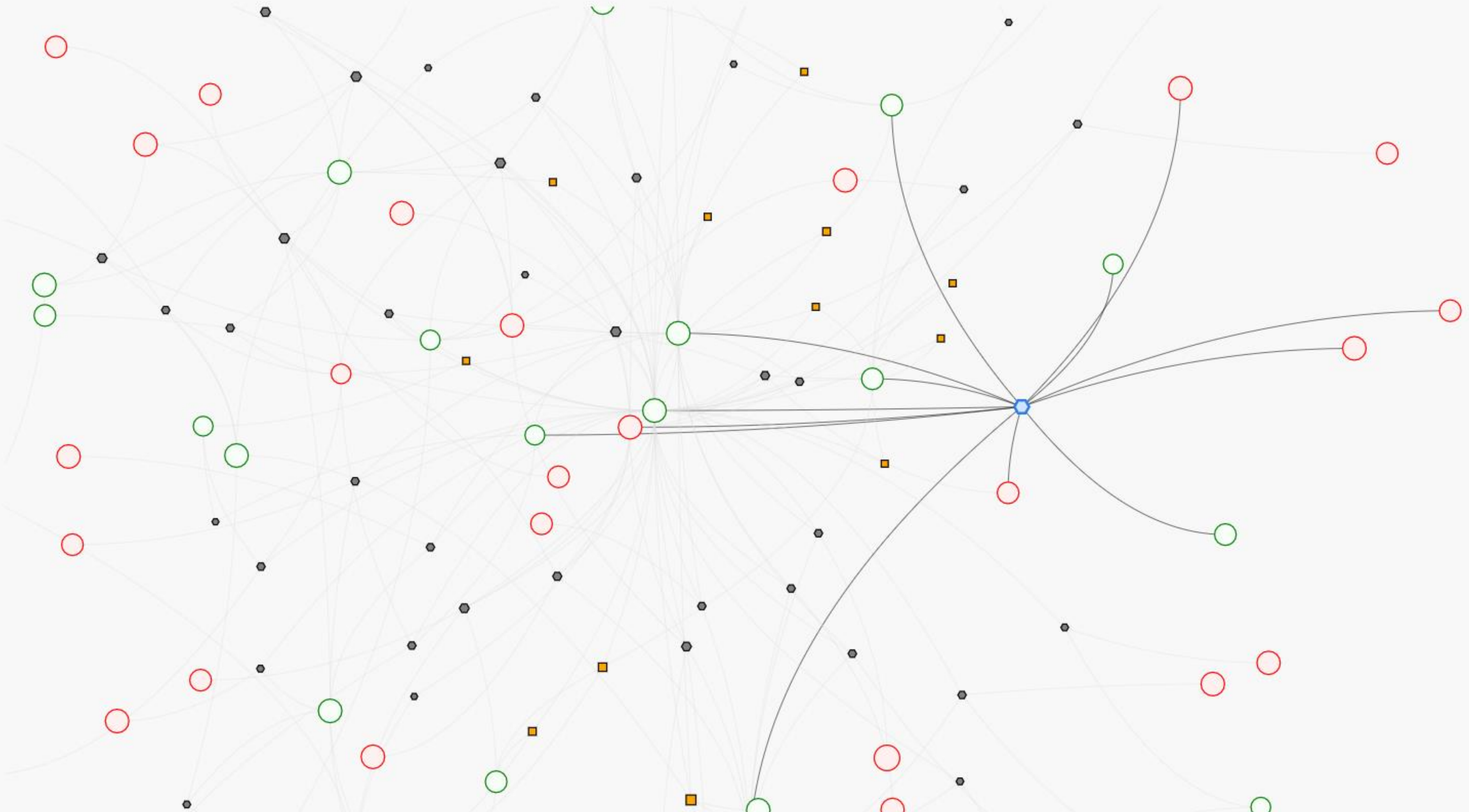
Multihomed Devices

Network responses expose secondary interface addresses

- UPnP, MDNS, NetBIOS, SNMP, and other protocols leak IPs
- Use unique probe IDs to correlate the responses
- Find devices bridging sensitive networks
- Identify devices with VPN connections
- Bypass network segmentation


26.42.62.0/24
View Asset LAPTOP-TOJEO0GF
192.168.0.0/24
LAPTOP-TOJEO0GF
192.168.0.210
192.168.175.0/24
DESKTOP-D4CHFD9
192.168.0.50
192.168.146.0/24





Uniqueness

What represents a unique system on the network?

- A physical device, a cluster, an IP address?
- A TLS certificate or DNS hostname?
- Uniqueness detection is an art

Given a wide enough view, how many unique things?

- A core switch may have dozens of interfaces
- A route may connect multiple networks
- A cluster may include many devices

Uniqueness

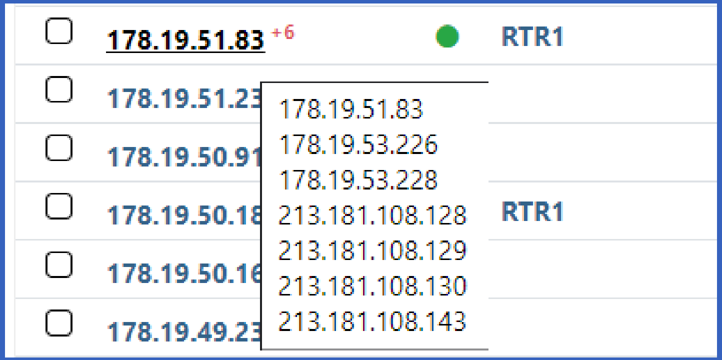
Attributes that define a unique system on the network

- MAC addresses assigned by a vendor (link-level unique)
- Random things that should never be predictable
- Shared attributes

Mostly unique attributes

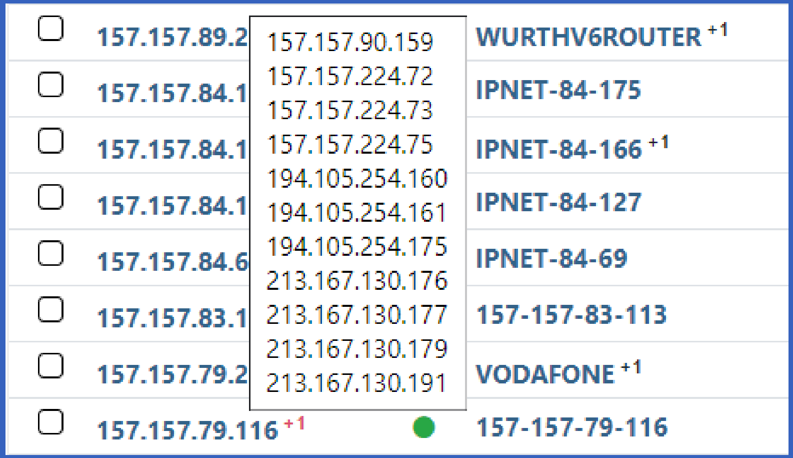
- Non-blacklisted MAC addresses
- NTLMSSP DNSComputerName
- Remote Desktop TLS Certificate
- SNMP “random” IDs

Uniqueness: Examples (All of Iceland)



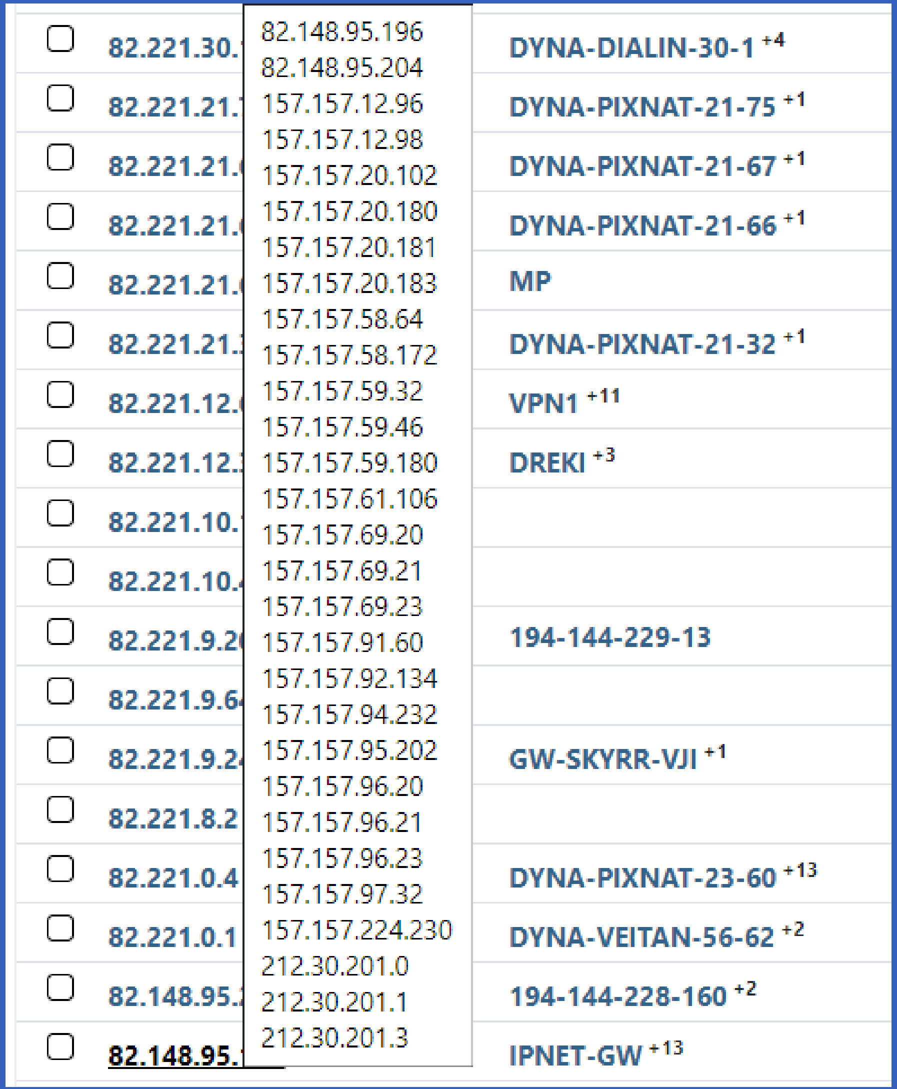
☐	178.19.51.83 $^{+6}$	●	RTR1
☐	178.19.51.23		
☐	178.19.50.91		
☐	178.19.50.18		RTR1
☐	178.19.50.16		
☐	178.19.49.23		

178.19.51.83
178.19.53.226
178.19.53.228
213.181.108.128
213.181.108.129
213.181.108.130
213.181.108.143



☐	157.157.89.2		WURTHV6ROUTER $^{+1}$
☐	157.157.84.1		IPNET-84-175
☐	157.157.84.1		IPNET-84-166 $^{+1}$
☐	157.157.84.1		IPNET-84-127
☐	157.157.84.6		IPNET-84-69
☐	157.157.83.1		157-157-83-113
☐	157.157.79.2		VODAFONE $^{+1}$
☐	157.157.79.116 $^{+1}$	●	157-157-79-116

157.157.90.159
157.157.224.72
157.157.224.73
157.157.224.75
194.105.254.160
194.105.254.161
194.105.254.175
213.167.130.176
213.167.130.177
213.167.130.179
213.167.130.191



☐	82.221.30.	DYNA-DIALIN-30-1 $^{+4}$
☐	82.221.21.	DYNA-PIXNAT-21-75 $^{+1}$
☐	82.221.21.	DYNA-PIXNAT-21-67 $^{+1}$
☐	82.221.21.	DYNA-PIXNAT-21-66 $^{+1}$
☐	82.221.21.	MP
☐	82.221.21.	DYNA-PIXNAT-21-32 $^{+1}$
☐	82.221.12.	VPN1 $^{+11}$
☐	82.221.12.	DREKI $^{+3}$
☐	82.221.10.	
☐	82.221.10.	
☐	82.221.9.2	194-144-229-13
☐	82.221.9.6	
☐	82.221.9.2	GW-SKYRR-VJI $^{+1}$
☐	82.221.8.2	
☐	82.221.0.4	DYNA-PIXNAT-23-60 $^{+13}$
☐	82.221.0.1	DYNA-VEITAN-56-62 $^{+2}$
☐	82.148.95.	194-144-228-160 $^{+2}$
☐	82.148.95.	IPNET-GW $^{+13}$

82.148.95.196
82.148.95.204
157.157.12.96
157.157.12.98
157.157.20.102
157.157.20.180
157.157.20.181
157.157.20.183
157.157.58.64
157.157.58.172
157.157.59.32
157.157.59.46
157.157.59.180
157.157.61.106
157.157.69.20
157.157.69.21
157.157.69.23
157.157.91.60
157.157.92.134
157.157.94.232
157.157.95.202
157.157.96.20
157.157.96.21
157.157.96.23
157.157.97.32
157.157.224.230
212.30.201.0
212.30.201.1
212.30.201.3

Next Steps

Discovery is not dead! Let’s continue pushing research

Create fingerprints from real-world scan data

Open source fingerprint collaboration

• Recog	https:// github.com/rapid7/recog
• DeepMAC	http:// deepmac.org/

Tools from work on Rumble Network Discovery

- https:// github.com/hdm/rumble-tools

Q & A

Email	x [at] hdm.io
Twitter	@hdmoore
Personal	https://hdm.io

Work

Atredis Partners	@	https:// atredis.com
Rumble Discovery	@	https:// rumble.run