Target Generation for Internet-wide IPv6 Scanning

Austin Murdock, Frank Li, Paul Bramsen,
Zakir Durumeric, Vern Paxson

Background

IPv4 scanning - Zmap

Background

- 2^{128} addresses => 10^{30} years to scan
- 32 *nybbles* (hex characters), 8 *groups*
- 2001:0db8:0000:0001:0000:0000:22:33333
- *n* bit prefix + *m* bit subnet + 64 bit host ID
- Before - 2001:**0**db8:**000**0:**000**1:*0000:0000*:22:33333
- After - 2001:db8:0:1::22:3333

Current Strategy 1 – Use Known Patterns

Decouple where to scan from how to scan*

- *Target Generation Algorithm (TGA)*

Previous Work:

Check Simple Patterns (2::1:0:0:0:1 … 2::f:0:0:0:f) *Czyz et al.*

Known Patterns eg. “wordy” (2001::cafe:face) *RFC7707*

Current Strategy 2 – Discover Patterns

Extract patterns from “Seeds”

Seeds:

- Network Taps
- Traceroutes
- DNS
    - Reverse
    - Passive
    - Forward

Previous Work:

Recursive Algorithms *Ullrich et al.*

Machine Learning *Pawel et al.*

New Strategy – Exploit Locality

Goal: maximize number of hosts found*

Hypothesis: Seed Density ⟶ Hit Density

- Find address ranges local to seeds with high seed density
- Expand ranges to discover new addresses

Bottom up, expand from seeds to ranges

Motivation

- Allocation patterns can be tricky to leverage

  1K seeds matching a random pattern
  `prefix:subnet:<16 random nybbles>`
  16^16 possible targets

  100 seeds matching a wordy pattern
  `prefix:subnet::<word>`
  1,296 possible targets

- 2/3 of routed prefixes had less than 10 seeds

Motivation 2

- There may be different patterns in one subnet

```
2403:d000:0004:0100:0000:0000:0000:0001 Sequential
2403:d000:0004:0100:0000:0000:0000:0002
2403:d000:0004:0100:0225:90ff:fe37:358b Embedded MAC
2403:d000:0004:0100:0225:90ff:fe37:760f
2403:d000:0004:0100:0230:48ff:fe34:fe96
2403:d000:0004:0100:0000:0000:0000:café Wordy
(Actual Seeds)
```

Motivation 3

- Often networks do not allocate addresses using least significant nibbles

```
2a02:04e8:00de:1000:5b6d:0a03:0000:0001
2a02:04e8:00de:1000:5b6d:0a07:0000:0001
2a02:04e8:00de:1000:5b6d:0a08:0000:0001
2a02:04e8:00de:1000:5b6d:0a09:0000:0001
2a02:04e8:00de:1000:5b6d:0a0a:0000:0001
2a02:04e8:00de:1000:5b6d:0a0b:0000:0001
(Actual Seeds)
```

`Find dense ranges` *`not`* `dense prefixes`

Motivation 4

- Whats going on here?

```
2800:0240:0001:0021:face:b00c:0000:00a7
2800:0240:0001:0022:face:b00c:0000:00a7
2800:0240:0001:0023:face:b00c:0000:00a7
2800:0240:0001:0024:face:b00c:0000:00a7
2800:0240:0001:0026:face:b00c:0000:00a7
2800:0240:0001:0029:face:b00c:0000:00a7
2800:0240:0001:002a:face:b00c:0000:00a7
2800:0240:0001:002d:face:b00c:0000:00a7
(Actual Seeds)      |  64-bit host ID  |
Do not rely on domain knowledge
```

What *don’t* we do

- Rely on known patterns or strategies
- Reverse engineer allocation patterns
- Set algorithmic parameters
    - E.g. No notion /64 is significant,
    - no “arbitrary” thresholds

6Gen

Strategy

- Select ranges of addresses local to the seeds
- Target the most promising ranges first (high density)
- Expand these ranges to encourage discovery
- Sole parameter: “probe budget”

Generating Ranges

Create a range

```
2::a
2::b → 2::[0-f] → 2::?
```

Grow a Range

```
2::1:?
2::2:b → 2::[0-f]:[0-f] → 2::?:?
```

“Tight” vs “Loose” Ranges

```
2::3
2::5
2::9

2::[3-9] Discovery space of 4

2::? -> 2::[0-f] Discovery space of 13
```

Uses more probes, but increases opportunity

Growing Ranges

Grow ranges incrementally to support granular budget levels

Compute change in size with Hamming distance

`2::a` Hamming distance 1
`2::b` (`2::?` is 16^1 times larger than `2::a`)

`2::1:?` Hamming distance 1
`2::2:b`

Example

2::1
2::2
2::3
2::1:1
2::1:2
2::a0
2::b1
2::c3
2::ffff
2::dddd

Seed	Closest	Dist	Range	Density
2::1	2::2 ~~2::ffff~~	1 ~~4~~	2::? ~~2::????~~	$3/16^1$ ~~$8/16^4$~~

Cost: 16

Example

2::1
2::2
2::3
2::1:1
2::1:2
2::a0
2::b1
2::c3
2::ffff
2::dddd

Output:

2::?

Seed	Closest	Dist	Range	Density
2::1	2::2 2::ffff	1 4	2::? 2::????	**$3/16^1$** –
2::1:1	2::1:2	1	2::1:?	$2/16^1$
2::a0	2::b1	2	2::??	$3/16^2$
2::ffff	2::dddd	4	2::????	$2/16^4$
...				

Example

Output:

```
2::1      2::?
2::2
2::3
2::1:1    2::1:?
2::1:2
2::a0
2::b1
2::c3
2::ffff
2::dddd
```

Seed	Closest	Dist	Range	Density
2::?	2::a0 2::1:1	1 1	2::?? 2::?:?	$6/16^2$ $5/16^2$

Example

	Output:
2::1	2::?
2::2	
2::3	
2::1:1	2::1:?
2::1:2	
2::a0	
2::b1	
2::c3	
2::ffff	
2::dddd	

Seed	Closest	Dist	Range	Density
2::?	2::a0 2::1:1	1 1	2::?? 2::?:?	**$6/16^2$** $5/16^2$
2::1:?	2::1	1	2::?:?	$5/16^2$
2::a0	2::b1	2	2::??	$3/16^2$
2::ffff	2::dddd	4	2::????	$2/16^4$
...				

Example

2::1
2::2
2::3
2::1:1
2::1:2
2::a0
2::b1
2::c3
2::ffff
2::dddd

Output:

~~2::?~~ 2::??

2::1:?

Cost: $16^2 + 16 = 272$

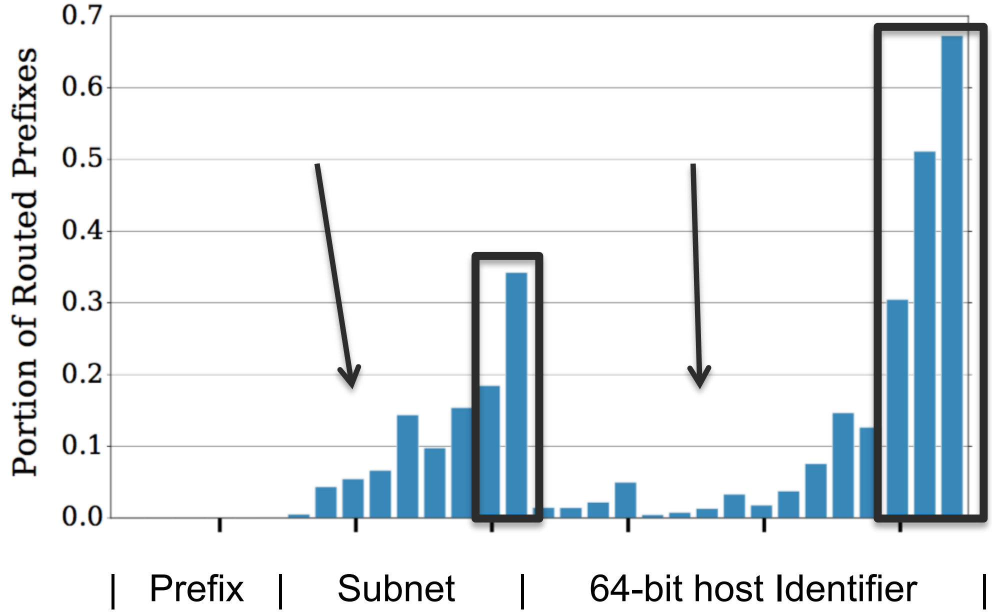
Seed	Closest	Dist	Range	Density
2::?	2::a0 2::1:1	1 1	2::?? 2::?:?	**$6/16^2$** $5/16^2$
2::1:?	2::1	1	2::?:?	$5/16^2$
~~2::a0~~	~~2::b1~~	~~2~~	~~2::??~~	~~$3/16^2$~~
2::ffff	2::dddd	4	2::????	$2/16^4$
...				

Evaluation

1. ~3M DNS AAAA seeds from Rapid7
   - ~ 8K routes prefixes*
   - ~ 7K ASes
2. Run 6Gen on each routed prefix (1M budget per prefix)
3. Convert list of target ranges to addresses (~6B targets)**
4. Probe addresses on tcp/80 (SYN scan)

Post talk note: *In the talk I mentioned that this total is for prefixes with 2 or more seeds.
In the paper we do not remove prefixes with one seed and report this number as 10,038.
**I mention in the talk that this total is less than 8B because 6Gen does not always generate 1M targets.

Where are the dynamic nybbles?



Evaluation

~55 million responses from ~6B probes

- ~30 Million from Akamai
- ~20 Million from Amazon

Encounter large blocks of responsive addresses
- E.g., Akamai has “active” /56s

How can we quickly detect large active regions?

Randomly probe each /96 -> 2^{32} possible addresses

Filter removed { 10.0 M / 10.2 M } /96s from 138 ASes

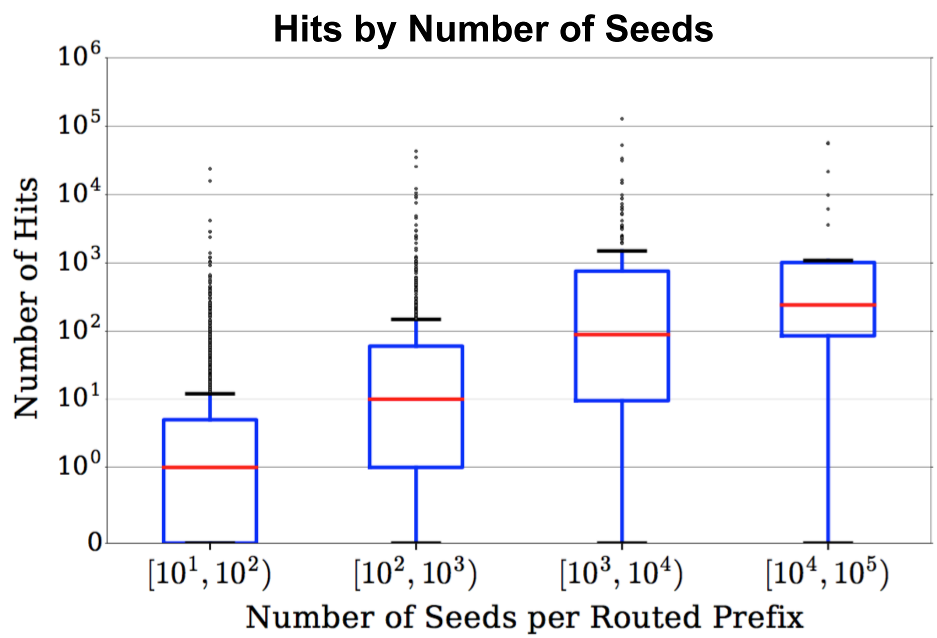
Manually removed two additional ASes (after /96 filtering)

/96 filter + manual inspection removed 98% of hits

Filtered Results

~ 1M new (non-seed) responses
- ~ 3K routed prefixes
- ~ 2K Ases

<u>New addresses for ~40% of prefixes*</u>



Post talk note: *In the talk I mentioned that this percentage is for prefixes with 2 or more seeds. In the paper we do not exclude prefixes with one seed and report this metric as 28%.

Future Work

Better detection of “active” blocks

Adaptive Scanning

- Density validation
- Pattern recognition for ranges

Thank You

Austin Murdock
austinmurdock@berkeley.edu
@austinkarch