This material is based upon work supported by the Defense
Advanced Research Projects Agency (DARPA) under Contract No.
HR001119C0102. The opinions, findings, and conclusions stated
herein are those of the authors and do not necessarily reflect
those of DARPA.

Finding horizontal scans

Introduction

A *horizontal scan* is a scan, by a single source, that probes a subnet for a single protocol/destination port. A *complete horizontal scan* is a horizontal scan that probes every address in the subnet at least once within a given period of time. An *efficient horizontal scan* is a complete horizontal scan that probes every address in the subnet exactly once within a given period of time.

There are cases where multiple sources collaborate by each doing incomplete horizontal scans which, when combined, are complete (and potentially efficient), but the tools in this directory focus on the horizontal scans performed by a single source.

This directory contains two programs: `find-fast-runs`, which detects scans and tabulates information about them, and `find-horiz-scans.sh`, which manages the task of running `find-fast-runs`.

find-horiz-scans.sh

Finding scans in a large data set can consume a large amount of computation and RAM. Every packet might be part of a scan, and so a lot of memory would be required to keep track of all the potential candidate scans in progress. The purpose of `find-horiz-scans.sh` is to invoke `find-fast-runs` on only the data from each destination subnet, so that each instance if `find-fast-runs` only needs to consider a fraction of the input. For example, imagine that our telescope is monitoring a /20 subnet, and we want to find horizontal scans of each of the 16 /24 subnets within that /20. We can use `find-horiz-scans.sh` to search for scans in each /24 individually, instead of trying to do all 16 at once. If you are resource-constrained, this can be extremely useful.

The output of `find-horiz-scan.sh` is stored in a directory named `$DATANAME-output`, with subdirectories named `out-PREFIXLEN-MAXELAPSED` where PREFIXLEN is the length of the prefix for the subnet (i.e. 24 for a /24, or 28 for a /28) and MAXELAPSED is the maximum elapsed time allowed for a scan to complete (measured in seconds). Within the subdirectories, the output for each destination subnet is stored in a gzip'd file named `BASE.txt.gz` (where BASE is the base address of the subnet).

find-fast-runs

The `find-fast-runs` is a program that reads CSV input (in the format created by `cpcap2csv`), finds horizontal scans, and prints out information about them. (By default, it only searches for complete horizontal scans, but it can also detect incomplete horizontal scans.)

The output from `find-fast-runs` writes seven lines of output for each scan that it discovers. The first token of each line defines the format and type of data on each line: `src`, `OFFS`, `TTLS`, `SPORTS`, `ISNS`, `IPIDS`, and `TIMES`.

The `src` line is always the first line for each scan. The `src` line consists of pairs of name/value tokens, for the following names:

- *src* - the source IP address of the scanner, as a decimal integer
- *dst* - the address of the start of the scanned subnet, as a decimal integer
- *proto* - the IP protocol, as a decimal integer
- *dport* - the destination port, as a decimal integer
- *cov* - the count of destination addresses covered by the scan, as a decimal integer
- *cnt* - the packet count, as a decimal integer
- *inv* - the number of inversions between an in-order scan and the order observed (see below), as a decimal integer

- *start* - the timestamp of the first packet in the scan, as a floating point number (in seconds).
- *elapsed* - the elapsed time between the first and last packet in the scan, as a floating point number (in seconds).

For example, a src line might look like.

```
src 167968768 dst 167772160 proto 17 dport 5060 cov 256 cnt 256
inv 1631 start 1696186687.484759 elapsed 0.172546
```

The remaining six lines each start with the given token (`OFFS`, etc), followed by a left square bracket, and then a comma-separated list of values, terminated with a right square bracket. Each value in the list gives the value of the corresponding value from the packet:

- *OFFS* - the offset of the destination address of the packet within the subnet
- *TTLS* - the value of the TTL field in the IP header of the packet
- *SPORTS* - the source port (aka the ephemeral port) of the packet
- *ISNS* - the initial sequence number of the packet (for TCP packets only – for other protocols, the values are always zero)
- *IPIDS* - the IP packet identifier number in the IP header of the packet
- *TIMES* - the elapsed time between the arrival of the first packet in the scan and the packet, rounded to the nearest millisecond

Counting inversions

The `inv` field in the `src` line gives the count of *inversions* there are in the order of the scan, which is the total distance that each element in the OFFS array would need to move in order to sort the OFFS in ascending order. The easiest way to compute the inversion count of a list is to do a bubblesort of an array and count the number of times that a pair of adjacent elements are swapped – and this is exactly what `find-fast-runs` does.

The inversion count provides a metric for how "out of order" the scan is with respect to the order of addresses in the subnet. A scan that visits each destination in ascending order will have an inversion count of zero, and a scan that visits each destination in descending order will have an inversion count of N(N-1)/2, where N is the size of the subnet.

A random scan, which visits the destinations in a random permutation of the destination addresses, will have an inversion count with an expected value of N(N-1)/4. Note, however, that the number of inversions *does not* indicate whether the scan order is truly random, because it is easy to invent scan orders with the same number of inversions as a truly random order, but which have a simple and completely non-random construction.

We can use the inversion count to determine whether the scan is *almost* in order (i.e. the inversion count is low), or whether it is intentionally out-of-order (i.e. the inversion count is near the expected count for a random permutation).

For example, we see many fast scans where an entire subnet is scanned within a fraction of a second. Many of these have low (but non-zero) inversion counts. Given the natural packet reordering that can happen to packets sent in a burst, we can infer that it is likely that the packets were originally sent in order, and the low inversion count is the effect of inadvertent packet reordering.