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.

Preprocessing fragments

Introduction

The Internet protocol version 4 allows packets to be fragmented. This can happen at the source, if the packet is too large for the underlying transport, or it can be done by the network itself if the packet is too large to traverse one of the hops along its path to the destination.

Each fragment includes information about how the fragments can be reassembled into the original packet. This information has is contained in two fields: the first is the “more fragments” bit, which indicates that the current packet is a fragment and is followed by additional fragments, and a fragment offset field, which provides the offset into the original packet that the IP payload of this fragment represents.

The motivation for fragmentation is that a packet may need to be too large to fit in a single transport-layer unit. Most contemporary transport layers have transport units that are large enough for most purposes, so fragments are somewhat uncommon – although there are a few protocols that require packets that are too large for common transport layers like Ethernet or wifi.

We expect fragments to be unusual in IBR, because most IBR consists of simple probes designed to provoke a response, and these packets are typically small, with no need for fragmentation on any common transport layer. During out initial examination of the IBR arriving at our telescope, fragments were infrequent. Hours, or even several days, would go by without seeing a single fragment, as expected.

In the data from our telescope, more 60% of the hours do not have any fragments at all, and almost 90% of the hours have 10 or fewer. A small number of hours, however, have a few thousand fragments, and a handful of hours have hundreds of thousands of fragments.

Some networks disallow fragments, except in specific situations. Fragments have been used in the past as ways to circumvent DPI or other security measures. It is possible to construct fragments whose assembly is ambiguous – the fragments overlap and the choice of which values to use and which to discard may be different for the DPI appliance and the destination host, allowing an adversary to send a message that cannot be detected by the DPI. Therefore we are interested in scanners that send fragments, because they might be attempting to find networks that permit fragments, particularly fragments that are unusually small, or which have other unusual properties.

Processing steps

The `fragment-tasks.sh` script finds fragments in the pcap files, and saves them to fragment-only pcap files, and creates CSV files with descriptions of each fragment. The last step of `fragment-tasks.sh` is to use `test-assembly.py` to detect whether the fragments can be completely, correctly, and unambiguously reassembled.

Like most of the other scripts in this package, `fragment-tasks.sh` takes its parameters from a parameter file, environment variables, or values on its commandline. See `../../params/example.params` for a description of the parameters. The required parameters are `DATANAME`, `PCAPDIR`, `FRAG_PCAPOUTDIR`, `FRAG_CSVOUTDIR`, and `FRAG_RESULTDIR`.

The pcaps for the fragmented packets are stored in `$FRAG_PCAPOUTDIR`, and the corresponding CSV files are stored in `$FRAG_CSVOUTDIR`. Note that the CSV files are in a different format than the CSV files created by other tools (because fragments have different information in their headers). See `../../C/show-frags.c` for more information.

The `$FRAG_RESULTDIR` contains information about the counts of fragments per

hour, the busiest sources and destinations for fragments, and the results of attempting to reassemble the packets.