Advanced Topics in IP Networks
Lecture 10

Lecture by Dr. Anat Bremler-Barr
Typeset by Steven Karas

2018-12-20
Last edited 20:56:47 2018-12-20

Disclaimer These notes are based on the lectures for the course Advanced Topics in IP Networks, taught by Dr. Anat Bremler-Barr at IDC Herzliyah in the fall semester of 2018/2019. Sections may be based on the lecture slides prepared by Dr. Anat Bremler-Barr.

1 Agenda

- Openflow Evolution
- Programmable Network Devices
- P4

2 P4: Programming Protocol-independent Packet Processors

Openflow started out in 2009 with a single rule table matching on 12 fields with a handful of action types. By 2013, there were 41 fields, multiple stages with heterogeneous tables, and more. Rather than a bottom up approach where each device declares how it can process data, P4 takes a top down approach. This has the user instructing the device how they want it to behave.

The current direction is for devices to have a configurable packet parser, with multiple tables of differing sizes, and generic packet-processing primitives.

Generally speaking, the benefits of this approach of data plane programmability are freedom from the underlying hardware and its limitations. There are already compilers for P4 that can attempt to target OpenFlow for simple P4 programs, although ideally the target is more flexible and general purpose. At the moment, the P4 toolchain is a vendor provided compiler with differing capabilities and external libraries.

2.1 P4 Program Structure

1. Headers (sequence and structure of fields)
2. Parsers (extract header fields)
3. Tables
4. Actions
5. Control Program

Headers are byte-aligned and define fields in terms of offset and width. Structures are unaligned. Parsers are state machines with three default states: start, accept, and reject. Tables are defined with a key to use for matching, possible actions, table size, and a default action. Table entries define the match rule for the table key, how to match, and an action to take. Actions define an action to take, and are largely defined by the target architecture.

In the next homework, we will receive a makefile that will handle the entire toolchain so we can focus only on the actual p4 program. As such, we will only need to implement small parts of p4 code.

3 Spoofed Attacks

3.1 SYN flood

3.1.1 TCP Intercept

Protects against SYN floods by using SYN Cookies before proxying "good" connections to the actual server. To actually forward the legitimate traffic to the server, there are many techniques. Proxying, running an http redirect, just reset the connection, etc.

3.1.2 Measurement of attack velocity

By measuring the replies, we can gather a statistical idea of how much traffic is being sent to a victim. However, this assumed uniform distribution of spoofed source addresses, and in combination with other effects leads to an underestimation of the actual attack velocity.

3.1.3 TTL

The attacker cannot observe the TTL at the victim, so this is a feature that can be used to classify spoofed traffic. This is a statistical measure, and there are many other caveats due to varying default values, quickly changing paths, and more.

3.1.4 Traceback

In this, we try to find the source of a flow by requesting from each network where the ingress is from, either automatically or via contacting their NOC. There was a proposal from 2000[3] to add the path to each packet, but it was too expensive in terms of data to feasibly implement.

A smarter appraoch was to use node sampling, in which each router would replace the source path with probability p. However, this depends on p being largely standard across vendors. Also, if p is too large, then it increases the amount of traffic required to observe the entire path. Conversely, if p is too small, then an attacker can subvert the analysis by spoofing the source path field as well.

Another approach is edge sampling, where we add three fields: start, end, and distance. With probability p, write self into start; otherwise, write to end and increment distance. However, this is expensive in terms of size, so instead of storing both start and end, we can simply xor the two fields. We can work backwards through a path to find the start router.

The good news is that there are several fields in IP headers that are not widely used, so we can hijack them for this purpose. The identification field in particular, which is normally used for packet fragmentation can be used for this.

4 Next Week

Next week we will discuss DDoS.

References

[1] Mark Crovella and Balachander Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

[2] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[3] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '00, pages 295–306, New York, NY, USA, 2000. ACM.

[4] George Varghese. *Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.