Traffic analysis with Python and DPKT

Adam Kalisz
adamkalisz.eu
adam_kalisz [at] wh2.tu-dresden.de

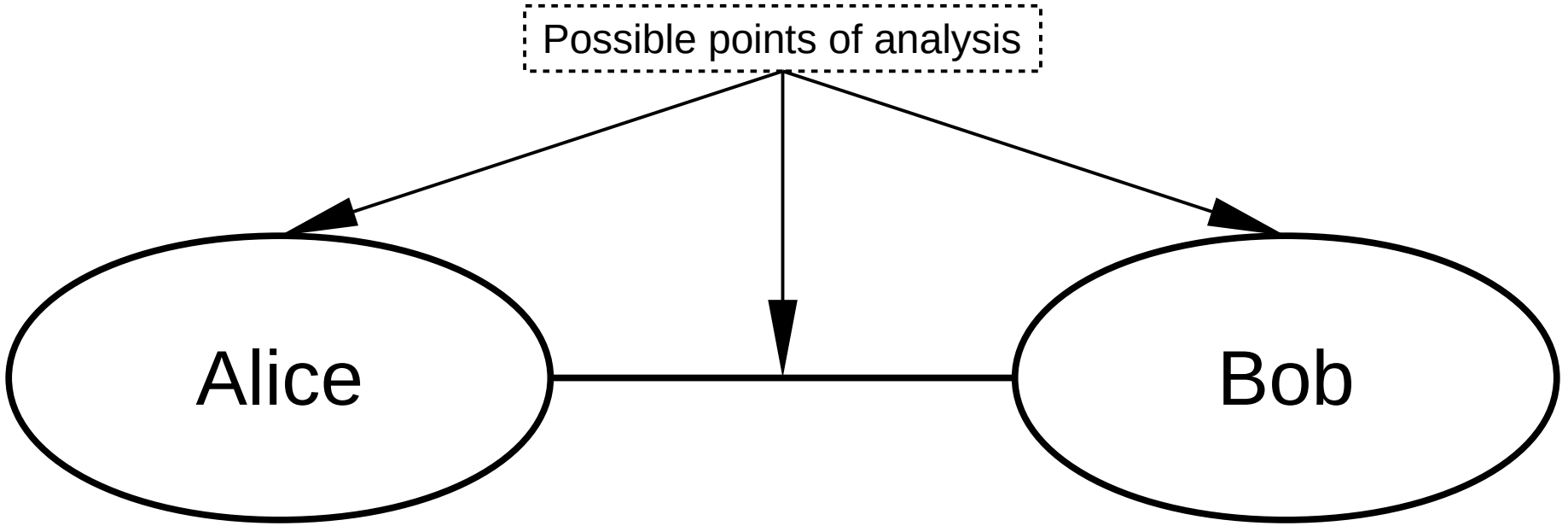
What to expect

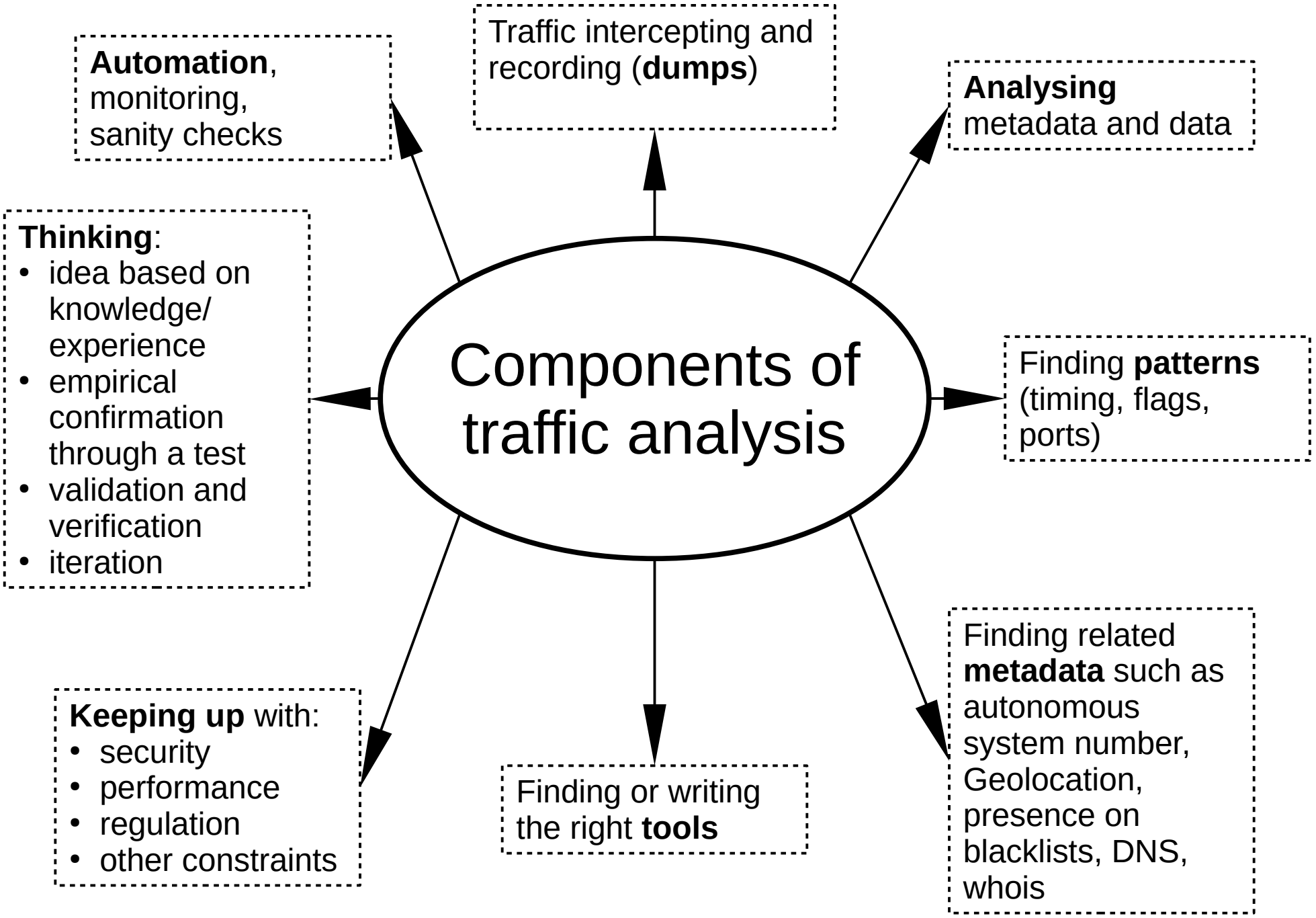
1) Why traffic analysis
2) Technical introduction to traffic analysis
3) A survey of tools, why Python/ DPKT
4) Ethics of traffic analysis
5) Examples
6) Problems, future work, alternative approaches
7) Question and Answer

Why traffic analysis

- Localizing and correcting communications-related problems
- Understanding systems (verifying, validating)
- Security anomalies (e.g. intrusion detection)
- Reverse engineering or protocol design
- Resource usage analysis („What do our users use/ need?“), accounting
- Malicous activities, such as snooping on users

Network **traffic analysis** (NTA) is the examination of data flowing over a communications channel.




Components of traffic analysis
Automation, monitoring, sanity checks
Traffic intercepting and recording (**dumps**)
Analysing metadata and data
Thinking:
• idea based on knowledge/ experience
• empirical confirmation through a test
• validation and verification
• iteration
Finding **patterns** (timing, flags, ports)
Keeping up with:
• security
• performance
• regulation
• other constraints
Finding or writing the right **tools**
Finding related **metadata** such as autonomous system number, Geolocation, presence on blacklists, DNS, whois

NTA performance with the smallest ethernet frame of 64 Bytes (Header of 18 or 22 Bytes + Payload) and overhead of 20 Bytes on the wire (Preamble, Start of Frame Delimiter and Interpacket gap), together 84 Bytes

Speed	Oneway maximal packets/ second count	Time per packet (in nanoseconds)
16 Mbps (ADSL down)	$16 * 10^6 / 84 = 190476$	$84 / 16 * 10^6 = 5250$
100 Mbps (VDSL 2 down, „Fast“ Ethernet)	1190476 ~ 1 Mio	840, less than µs
1 Gbps (LAN)	11904761 ~ 12 Mio	84 ~ RAM latency
10 Gbps (Small servers)	119047619 ~ 120 Mio	8.4 ~ <L3 cache latency
40 Gbps (Cloud servers, storage)	476190476 ~ 476 Mio	2.1 ~ <L2 cache latency
100 Gbps (IXP, Core)	1190476190 ~ 1.2 Mrd	0.84 ~ <L1 cache latency, 3 CPU clock tics @3.6 GHz

Approaches to better performance:

- filters, smarter capturing
- lossy captures snaplen, sampling, flows
- less context switches/ interrupts, more affinity
- efficient datastructures and interfaces
- multi-core and low level programming
- Offload to ASIC or FPGA

Tcpdump/ Windump, Wireshark/ tshark

- Compatible tools (using pcap-files)
- Present almost everywhere
- Useful GUI with Wireshark
- Best for interactive „quick and dirty“ use

Source IP.Source Port > Destination IP.Destination Port: TCP Flags, Checksum, sequence numbers etc.

10.10.6.95.56457 > 2.60.77.74.445: Flags [.], cksum 0x579b (correct), seq 1, ack 2, win 260, length 0
10.10.6.88.65261 > 89.19.16.13.445: Flags [S], cksum 0xc8a2 (correct), seq 755949362, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65119 > 124.156.175.60.445: Flags [F.], cksum 0x1c08 (correct), seq 364, ack 357, win 260, length 0
10.10.6.95.56469 > 216.24.240.150.445: Flags [.], cksum 0xc814 (correct), seq 1714817492, ack 2556307180, win 256, length 0
10.10.6.95.56469 > 216.24.240.150.445: Flags [P.], cksum 0xbf54 (correct), seq 0:137, ack 1, win 256, length 137
10.10.6.95.56490 > 91.211.249.186.445: Flags [S], cksum 0x21c5 (correct), seq 184671723, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65261 > 89.19.16.13.445: Flags [.], cksum 0x8696 (correct), seq 755949363, ack 976250783, win 256, length 0
10.10.6.88.65261 > 89.19.16.13.445: Flags [F.], cksum 0x8695 (correct), seq 0, ack 1, win 256, length 0
10.10.6.88.65276 > 89.19.16.13.445: Flags [S], cksum 0xbf0f (correct), seq 1429717133, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65212 > 173.254.209.208.445: Flags [P.], cksum 0xcbc9 (correct), seq 373:455, ack 435, win 254, length 82
10.10.6.88.65277 > 192.3.68.90.445: Flags [S], cksum 0x85f3 (correct), seq 2332829846, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56495 > 69.175.112.224.445: Flags [S], cksum 0x68b1 (correct), seq 2268379589, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.83.60085 > 173.212.227.130.445: Flags [S], cksum 0xed24 (correct), seq 1210420127, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56472 > 2.60.77.74.445: Flags [P.], cksum 0xe65f (correct), seq 88:191, ack 132, win 260, length 103
10.10.6.88.65235 > 188.40.212.95.445: Flags [S], cksum 0x4913 (correct), seq 2940515645, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56418 > 176.48.99.254.445: Flags [F.], cksum 0xae26 (correct), seq 364, ack 283, win 259, length 0
10.10.6.95.58730 > 116.90.98.19.445: Flags [.], cksum 0xb731 (correct), seq 4255435628:4255435629, ack 1598489306, win 65392, length 1[|SMB]
10.10.6.83.60088 > 23.254.142.79.445: Flags [S], cksum 0x494c (correct), seq 3560541802, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65278 > 198.148.109.142.445: Flags [S], cksum 0x3e75 (correct), seq 4045789748, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56320 > 181.200.202.88.445: Flags [.], cksum 0x8e8b (correct), seq 0:1, ack 1, win 64240, length 1[|SMB]
10.10.6.95.56484 > 144.35.166.203.445: Flags [.], cksum 0x4dc1 (correct), seq 3224825837, ack 839999283, win 64240, length 0
10.10.6.83.60089 > 173.208.55.172.445: Flags [S], cksum 0xdcc2 (correct), seq 3229536126, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65276 > 89.19.16.13.445: Flags [.], cksum 0x509b (correct), seq 1429717134, ack 3728011266, win 256, length 0
10.10.6.88.65276 > 89.19.16.13.445: Flags [P.], cksum 0x8878 (correct), seq 0:88, ack 1, win 256, length 88
10.10.6.83.59920 > 186.65.192.112.445: Flags [F.], cksum 0xc7a0 (correct), seq 0, ack 1, win 64240, length 0
10.10.6.83.60091 > 2.49.81.188.445: Flags [S], cksum 0xeff0 (correct), seq 2605675277, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.83.60092 > 67.80.62.116.445: Flags [S], cksum 0x15d5 (correct), seq 1263890251, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56469 > 216.24.240.150.445: Flags [P.], cksum 0x5fbe (correct), seq 137:277, ack 134, win 256, length 140
10.10.6.88.65280 > 95.189.216.21.445: Flags [S], cksum 0x820e (correct), seq 1330328772, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.88.65276 > 89.19.16.13.445: Flags [P.], cksum 0x72d0 (correct), seq 88:191, ack 132, win 256, length 103
10.10.6.95.56507 > 104.161.71.144.445: Flags [S], cksum 0xaa1a (correct), seq 3680709247, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
10.10.6.95.56472 > 2.60.77.74.445: Flags [P.], cksum 0xf7ad (correct), seq 191:286, ack 205, win 260, length 95
10.10.6.88.65320 > 47.244.186.186.445: Flags [S], cksum 0xaec4 (correct), seq 1321489809, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0

A small dump of a capture of “WannaCrypt“ malware in action
https://www.endgame.com/blog/technical-blog/wcrywanacry-ransomware-technical-analysis

Programming language + library

- Even more flexible
- Great for own tools
- Performance/ reliability?
- Packet generation
- Analysis automation

Tool examples

- Python/ Scapy
- Gopacket
- Perl Net::Pcap and others probably
- Lua + Wireshark
- C + libpcap
- Python/ DPKT

Ethics considerations

- IP addresses Personally Identifiable Information (PII)
- Be considerate of peoples privacy
- Don‘t disturb normal traffic with packet generation on live networks
- Don‘t be too confident about software anybody has written (general SW engineering advice)

Python/ DPKT

- A bit harder than Scapy
- Much better performance
- Documentation lacking

Examples

```
import dpkt
import socket
def inet_to_str(inet):
        try:
            return socket.inet_ntop(socket.AF_INET, inet)
        except ValueError:
            return socket.inet_ntop(socket.AF_INET6, inet)

with open(capture_name, 'rb') as dump:
    for time_stamp, payload in dpkt.pcap.Reader(dump):
        eth = dpkt.ethernet.Ethernet(payload)
        print(inet_to_str(eth.data.src))
```

```
#!/usr/bin/env python3

import dpkt
import socket

def inet_to_str(inet):
    try:
        return socket.inet_ntop(socket.AF_INET, inet)
    except ValueError:
        return socket.inet_ntop(socket.AF_INET6, inet)

with open('./testdump.pcap', 'rb') as dump:
    for time_stamp, payload in dpkt.pcap.Reader(dump):
        eth = dpkt.ethernet.Ethernet(payload)
        print("{:15}\t {:5}\t {:15}\t {:5}".format(
            inet_to_str(eth.data.src),
            str(eth.data.data.sport),
            inet_to_str(eth.data.dst),
            str(eth.data.data.dport)))
```

Problems

- Still a bit slow
- Need to write code
- DPKT community quite small

Alternative approaches

- pmacct
- Netflow or sFlow
- In-kernel module written in BPF

Thank you for your attention