SF-TAP: Scalable and Flexible Traffic Analysis Platform Running on Commodity Hardware

Yuuki Takano, Ryosuke Miura, Shingo Yasuda Kunio Akashi, Tomoya Inoue

NICT, JAIST
(Japan)

in USENIX LISA 2015
Table of Contents

1. Motivation
2. Related Work
3. Design of SF-TAP
4. Implementation of SF-TAP
5. Performance Evaluation
6. Conclusion
Motivation (1)

- Programmable application level traffic analyzer
  - We want …
  - to write traffic analyzers in any languages such as Python, Ruby, C++, for many purposes (IDS/IPS, forensic, machine learning).
- **not** to write codes handling TCP stream reconstruction (quite complex).
- modularity for many application protocols.
Motivation (2)

- High speed application level traffic analyzer
  - We want …
    - to handle high bandwidth traffic.
    - to handle high connections per second.
    - horizontal and CPU core scalable analyzer.
Motivation (3)

• Running on Commodity Hardware

• We want ...

  • open source software.

• not to use expensive appliances.
Related Work

SF-TAP

+ modularity and scalability

GASPP [USENIX ATC 2014]
SCAP [IMC 2012]
libnids
(flow oriented analyzer)
(low level traffic capture)

l7-filter
nDPI
libprotobuf
(application traffic detector)

DPDK netmap [USENIX ATC 2012]
pcap BPF [USENIX ATC 1993]
High-level Architecture of SF-TAP
Design Principle (1)

- Flow Abstraction
  - abstract flows by application level protocols
  - provide flow abstraction interfaces like /dev, /proc or BPF
  - for multiple programming languages
- Modular Architecture
  - separate analyzing and capturing logic
  - easily replace analyzing logic
Design Principle (2)

• Horizontal Scalable
  • analyzing logic tends to require many computer resources
  • volume effect should solve the problem
• CPU Core Scalable
  • both analyzing and capturing logic should be core scalable for efficiency
Design of SF-TAP (1)

**defined 4 planes**

- **Analyzer Plane**
  - application level analyzers
  - Forensic, IDS/IPS, etc...
  - (users of SF-TAP implements here)

- **Abtractor Plane**
  - flow abstraction
  - (we implemented)

- **Separator Plane**
  - flow separation
  - (we implemented)

- **Capturer Plane**
  - traffic capturing
  - (ordinary tech.)

---

**SF-TAP Cell**

**Abstractor Plane**
- Flow Abstractor
  - TCP and UDP Handler
  - Flow Classifier
  - Flow Identifier
  - IP Packet Defragmenter

**Analyzer Plane**
- Application Protocol Analyzer
  - TLS Analyzer
  - HTTP Analyzer
  - etc...

**SF-TAP Cell Incubator**
- Flow Separator
  - IP Fragment Handler

**Capturer Plane**
- Packet Forwarder

**Separator Plane**
- L2 Bridge

**L3/L7 Sniffer**
- mirroring traffic

**SSL Proxy**
- etc...
Design of SF-TAP (2)
SF-TAP Cell Incubator

Flow Separator
separate flows to multiple Ifs

IP Fragment Handler
handle fragmented packets

Packet Forwarder
layer 2 bridge
layer 2 frame capture
Design of SF-TAP (3)
SF-TAP Flow Abstractor

TCP and UDP Handler
reconstruct TCP flows
nothing to do for UDP

Flow Identifier
identify flows by IP and port

IP Packet Defragmenter
defragment IP packets if needed
Design of SF-TAP (4)
SF-TAP Flow Abstractor

Flow Classifier
classify flows by regular expressions
output to abstraction IFs
Implementation

- SF-TAP cell incubator
  - C++11
  - it uses netmap, available on FreeBSD
- SF-TAP flow abstractor
  - C++11
  - it uses pcap or netmap *(updated from the paper)*
  - available on Linux, *BSD, and MacOS
- Source Code
  - https://github.com/SF-TAP
- License
  - 3-clauses BSD
Performance Evaluation (1)

Figure 7: CPU Load of HTTP Analyzer and Flow Abstractor

Figure 8: Total Memory Usage of HTTP Analyzer

Figure 9: Packet Drop against CPS

5.3 Performance Evaluation of the Cell Incubator

In the experiments involving the cell incubator, we used a PC with DDR3 16 GB Memory and an Intel Xeon E5-2470 v2 processor (10 cores, 2.4 GHz, 25 MB cache) and FreeBSD 10.1. The computer was equipped with four Intel quad-port 1 GbE NICs and an Intel dual-port 10 GbE NIC. We generated network traffic consisting of short packets (i.e., 64-byte L2 frames) on the 10 GbE lines for our evaluations. The cell incubator separated traffic based on the flows, with the separated flows forwarded to the twelve 1 GbE lines. Figure 13 shows our experimental network.

We conducted our experiments using three patterns:

1. the cell incubator worked in the mirroring mode using port mirroring on the L2 switch; in other words, it captured packets at \( \alpha \) and forwarded packets to \( \gamma \);
2. the cell incubator worked in the inline mode but did not forward packets to 1 GbE NICs, instead only \( \alpha \) to \( \beta \); and
3. the cell incubator worked in the inline mode, capturing packets at \( \alpha \) and forwarding to both \( \beta \) and \( \gamma \).

Table 14 shows the performance of the cell incubator. For pattern (1), i.e., the mirroring mode, the cell incubator could manage packets up to 12.49 Mpps. For pattern (2), i.e., the cell incubator working as an L2 bridge, it could forward packets up to 11.60 Mpps. For pattern (3), i.e., forwarding packets to \( \beta \) and \( \gamma \), the cell incubator could forward packets to \( \beta \) and \( \gamma \) up to 11.44 Mpps. The performance of the inline mode was poorer than that of the mirroring mode because packets were forwarded to two NICs when using the inline mode. However, the inline mode is more suitable for specific purposes such as IDS/IPS because the same packets are dropped at \( \beta \) and \( \gamma \). In other words, all transmitted packets can be captured when using the inline mode.

Table 15 shows the CPU load averages of the cell incubator when in the inline mode and forwarding 64-byte frames. At 5.95 and 10.42 Mpps, packets were not dropped when forwarding. At approximately 10.42 Mpps, the upper limit of dropless forwarding was reached. This indicates that several CPUs were used for forwarding, but the 15th CPU's resources were especially consumed.

8

packet drop against connections per second

15
Performance Evaluation (1)

Forwarding performance of SF-TAP cell incubator
Other Features

• L7 Loopback interface for encapsulated flows

• Load balancing mechanism for application protocol analysers

• Separating and mirroring modes of SF-TAP cell incubator

• See more details in our paper
Conclusion

- We proposed SF-TAP for application level traffic analysis.

- SF-TAP has following features.
  - flow abstraction
  - running on commodity hardware
  - modularity
  - scalability

- We showed SF-TAP has achieved high performance in our experiments.