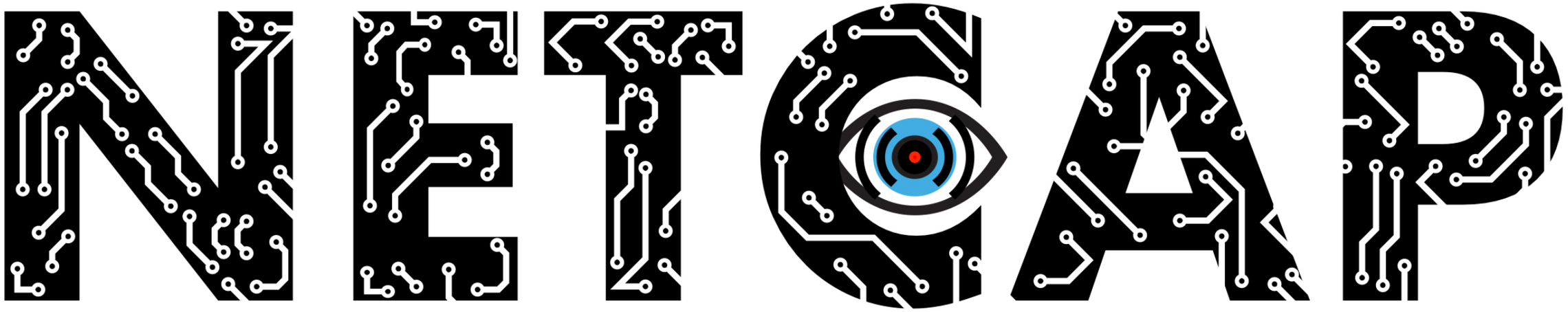


Implementation and evaluation of secure and scalable anomaly-based network intrusion detection

Philipp Mieden, HITBAMS2019

$ whoami

2019: Bachelor of Science @Ludwig Maximilian University of Munich
Security & Backend Engineer @bestbytes
Next up: Security and Network Engineering Master @University of Amsterdam

Interests:

- Network Security Monitoring & Anomaly Detection
- Machine Learning
- Programming (Golang, C / C++ / ObjC, Swift, Haskell, Python, Rust)
- Hardware & Software Security
- Reverse Engineering
- Penetration Testing

F.A.Q:
Are you moxies little brother? - Nope.
Are you a vegetarian or vegan? - Nope.
What's your favourite programming language? - Go.

Roadmap

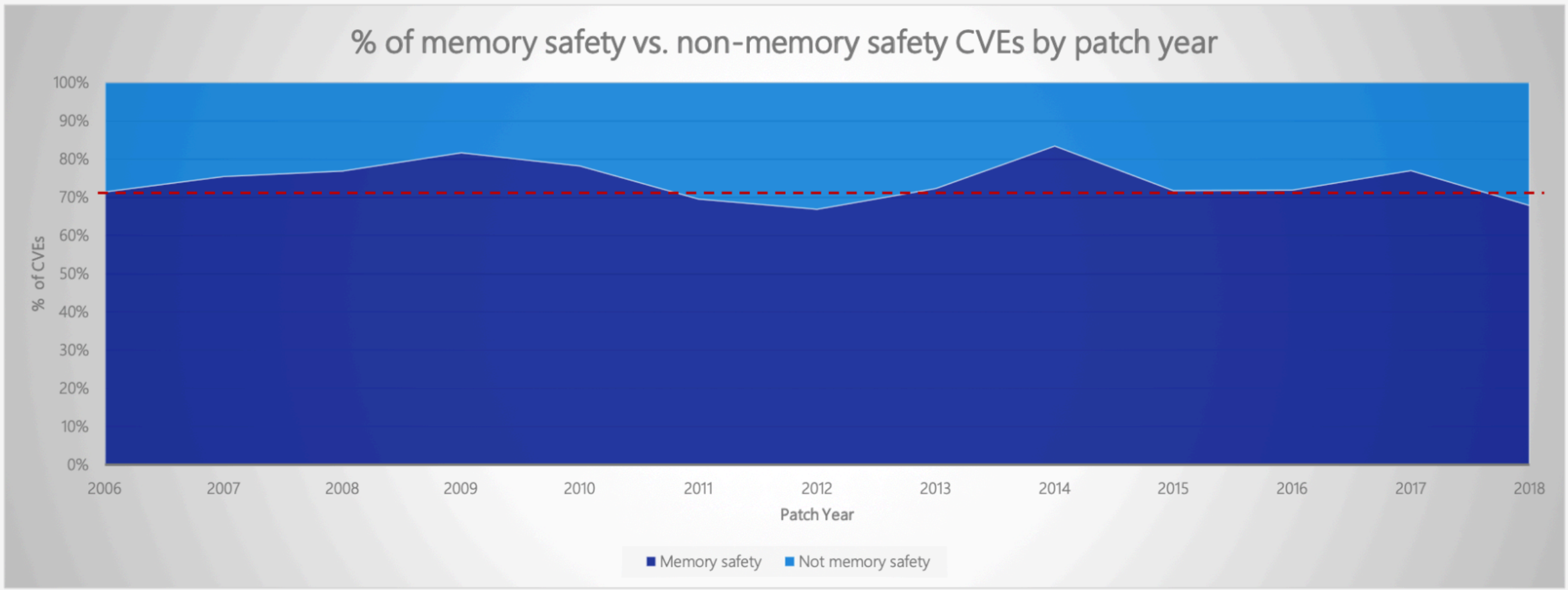
- Introduction and problem formulation
- NETCAP overview
- Thesis experiments wrap up
- What’s new

Let’s talk about a major problem of the software industry.

Memory Safety.

Memory safety issues remain dominant

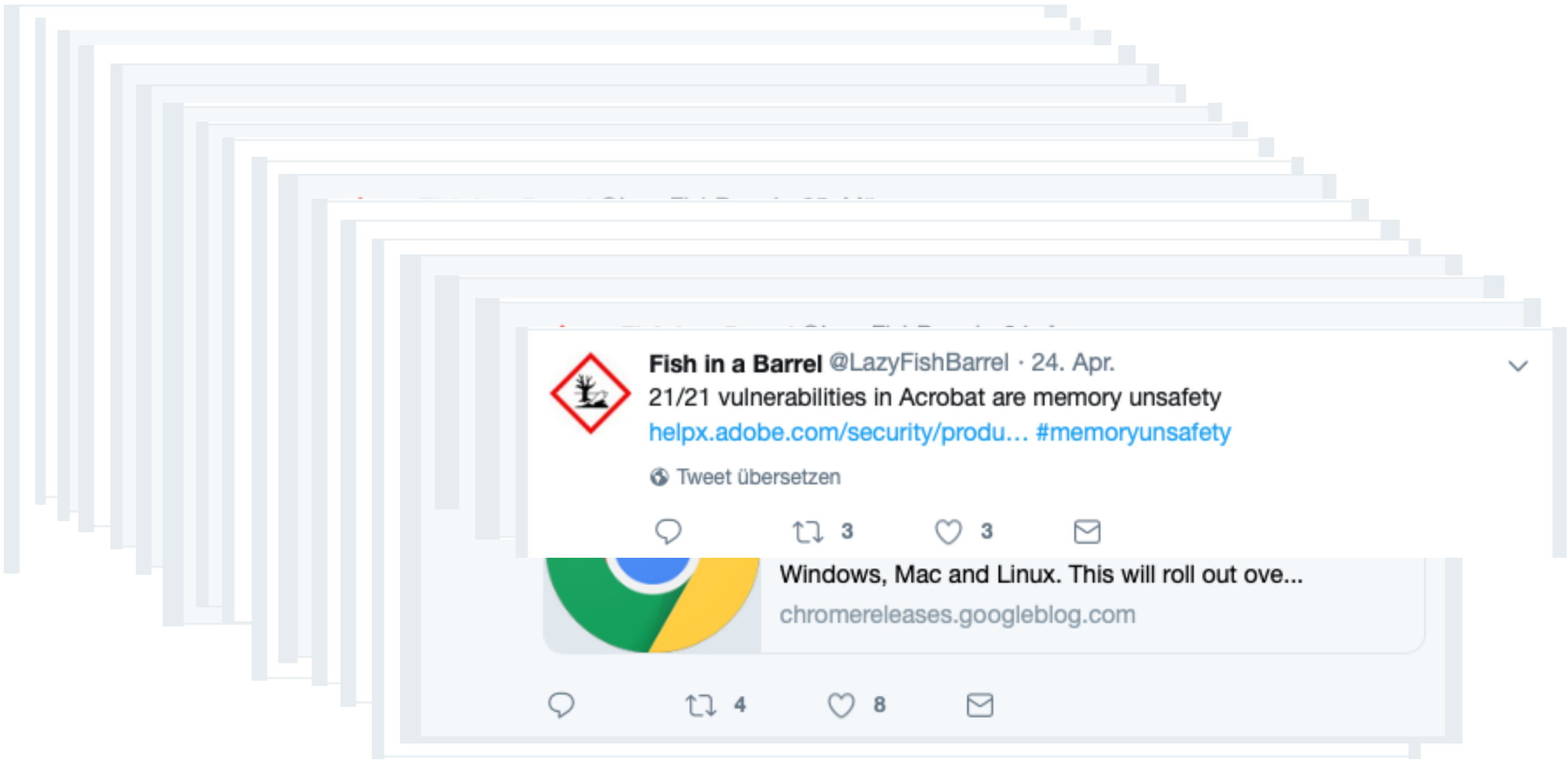
We closely study the root cause trends of vulnerabilities & search for patterns



~70% of the vulnerabilities addressed through a security update each year continue to be memory safety issues

Source: Slides from Matt Miller @ BlueHat Israel 2019

Memory corruption is an issue on every platform.



Mitigations don’t help.

They increase the cost for attacks, but do not address the root cause.

How does memory safety affect network security monitoring?

MITRE CVE results for Bro (Zeek) IDS

There are **7** CVE entries that match your search.

Name	Description
CVE-2018-17019	In Bro through 2.5.5, there is a DoS in IRC protocol names command parsing in analyzer/protocol/irc/IRC.cc.
CVE-2018-16807	In Bro through 2.5.5, there is a memory leak potentially leading to DoS in scripts/base/protocols/krb/main.bro in the Kerberos protocol parser.
CVE-2017-1000458	Bro before Bro v2.5.2 is vulnerable to an out of bounds write in the ContentLine analyzer allowing remote attackers to cause a denial of service (crash) and possibly other exploitation.
CVE-2015-1522	analyzer/protocol/dnp3/DNP3.cc in Bro before 2.3.2 does not reject certain non-zero values of a packet length, which allows remote attackers to cause a denial of service (buffer overflow or buffer over-read) via a crafted DNP3 packet.
CVE-2015-1521	analyzer/protocol/dnp3/DNP3.cc in Bro before 2.3.2 does not properly handle zero values of a packet length, which allows remote attackers to cause a denial of service (buffer overflow or buffer over-read if NDEBUG; otherwise assertion failure) via a crafted DNP3 packet.
CVE-2007-0186	Multiple cross-site scripting (XSS) vulnerabilities in F5 FirePass SSL VPN allow remote attackers to inject arbitrary web script or HTML via (1) the xcho parameter to my.logon.php3; the (2) topblue, (3) midblue, (4) wtopblue, and certain other Custom color parameters in a per action to vdesk/admincon/index.php; the (5) h321, (6) h311, (7) h312, and certain other Front Door custom text color parameters in a per action to vdesk/admincon/index.php; the (8) ua parameter in a bro action to vdesk/admincon/index.php; the (9) app_param and (10) app_name parameters to webyfiers.php; (11) double eval functions; (12) JavaScript contained in an <FP_DO_NOT_TOUCH> element; and (13) the vhost parameter to my.activation.php. NOTE: it is possible that this candidate overlaps CVE-2006-3550.
CVE-2006-6256	Cross-site scripting (XSS) vulnerability in the file manager in admin/bro_main.php in AlternC 0.9.5 and earlier allows remote attackers to inject arbitrary web script or HTML via a folder name.

MITRE CVE results for Suricata IDS

Search Results

There are **17** CVE entries that match your search.

Name	Description
CVE-2018-6794	Suricata before 4.0.4 is prone to an HTTP detection bypass vulnerability in detect.c and stream-tcp.c. If a malicious server breaks a normal TCP flow and sends data before the 3-way handshake is complete, then the data sent by the malicious server will be accepted by web clients such as a web browser or Linux CLI utilities, but ignored by Suricata IDS signatures. This mostly affects IDS signatures for the HTTP protocol and TCP stream content; signatures for TCP packets will inspect such network traffic as usual.
CVE-2018-18956	The ProcessMimeEntity function in util-decode-mime.c in Suricata 4.x before 4.0.6 allows remote attackers to cause a denial of service (segfault and daemon crash) via crafted input to the SMTP parser, as exploited in the wild in November 2018.
CVE-2018-14568	Suricata before 4.0.5 stops TCP stream inspection upon a TCP RST from a server. This allows detection bypass because Windows TCP clients proceed with normal processing of TCP data that arrives shortly after an RST (i.e., they act as if the RST had not yet been received).
CVE-2018-10244	Suricata version 4.0.4 incorrectly handles the parsing of an EtherNet/IP PDU. A malformed PDU can cause the parsing code to read beyond the allocated data because DecodeENIPPDU in app-layer-enip-commmon.c has an integer overflow during a length check.
CVE-2018-10243	htp_parse_authorization_digest in htp_parsers.c in LibHTP 0.5.26 allows remote attackers to cause a heap-based buffer over-read via an authorization digest header.
CVE-2018-10242	Suricata version 4.0.4 incorrectly handles the parsing of the SSH banner. A malformed SSH banner can cause the parsing code to read beyond the allocated data because SSHParseBanner in app-layer-ssh.c lacks a length check.
CVE-2018-1000167	OISF suricata-update version 1.0.0a1 contains an Insecure Deserialization vulnerability in the insecure yaml.load-Function as used in the following files: config.py:136, config.py:142, sources.py:99 and sources.py:131. The "list-sources"-command is affected by this bug. that can result in Remote Code Execution(even as root if suricata-update is called by root). This attack appears to be exploitable via a specially crafted yaml-file at https://www.openinfosecfoundation.org/rules/index.yaml. This vulnerability appears to have been fixed in 1.0.0b1.
CVE-2017-7177	Suricata before 3.2.1 has an IPv4 defragmentation evasion issue caused by lack of a check for the IP protocol during fragment matching.
CVE-2017-15377	In Suricata before 4.x, it was possible to trigger lots of redundant checks on the content of crafted network traffic with a certain signature, because of DetectEngineContentInspection in detect-engine-content-inspection.c. The search engine doesn't stop when it should after no match is found; instead, it stops only upon reaching inspection-recursion-limit (3000 by default).
CVE-2016-10728	An issue was discovered in Suricata before 3.1.2. If an ICMPv4 error packet is received as the first packet on a flow in the to_client direction, it confuses the rule grouping lookup logic. The toclient inspection will then continue with the wrong rule group. This can lead to missed detection.
CVE-2015-8954	The MemcmpLowercase function in Suricata before 2.0.6 improperly excludes the first byte from comparisons, which might allow remote attackers to bypass intrusion-prevention functionality via a crafted HTTP request.
CVE-2015-0971	The DER parser in Suricata before 2.0.8 allows remote attackers to cause a denial of service (crash) via vectors related to SSL/TLS certificates.
CVE-2014-9769	pcre_jit_compile.c in PCRE 8.35 does not properly use table jumps to optimize nested alternatives, which allows remote attackers to cause a denial of service (stack memory corruption) or possibly have unspecified other impact via a crafted string, as demonstrated by packets encountered by Suricata during use of a regular expression in an Emerging Threats Open ruleset.

Several memory related fixes in latest Suricata release

April 30, 2019

by **inliniac**

in **news**, **release**

Leave a comment

Suricata 4.1.4 released

We're pleased to announce **Suricata 4.1.4**. This release fixes a number of issues found in the 4.1 branch.

Get the release here:
https://www.openinfosecfoundation.org/download/suricata-4.1.4.tar.gz

Changes

- Bug #2870: pcap logging with lz4 coverity warning
- Bug #2883: ssh: heap buffer overflow
- Bug #2884: mpls: heapbuffer overflow in file decode-mpls.c
- Bug #2887: decode-ethernet: heapbuffer overflow in file decode-ethernet.c
- Bug #2888: 4.1.3 core in HCBDCreateSpace
- Bug #2894: smb 1 create andx request does not parse the filename correctly
- Bug #2902: rust/dhcp: panic in dhcp parser
- Bug #2903: mpls: cast of misaligned data leads to undefined behavior
- Bug #2904: rust/ftp: panic in ftp parser
- Bug #2943: rust/nfs: integer underflow
- This release includes Suricata-Update 1.0.5

More memory issues
not listed in the Suricata bug tracker

```
commit 316a411b6b40365ffff382967bec8bc22f18c192
Author: Philippe Antoine <contact@catenacyber.fr>
Date:   Wed Mar 27 22:56:15 2019 +0100

    ssl : SSLProbingParser overflow fix

    Found by fuzzing
    Fixes ssl detection evasion by packet splitting

commit 666bb1b6e48b47e9fafe161ac57deae8d0fd89f0
Author: Victor Julien <victor@inliniac.net>
Date:   Mon Apr 15 14:52:38 2019 +0200

    parse/ip: fix potential oob write in ipv4 validation

    Found using AFL.
```

Mitigations in IDS solutions

Suricata: Rust for several parsers

340k Lines of C / 17k Lines of Rust

```
dreadbook:suricata alien$ git status
HEAD detached at suricata-4.1.4
nothing to commit, working tree clean
dreadbook:suricata alien$ cloc .
    1352 text files.
    1343 unique files.
     172 files ignored.

github.com/AlDanial/cloc v 1.80  T=2.86 s (415.3 files/s, 192981.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                              520          65983          51941         341645
C/C++ Header                   520           6332          15282          20393
Rust                            73           2045           2121          17642
Bourne Shell                    15           1932           1406          12055
YAML                             4            198            204           3101
m4                               3            314             23           2712
Python                          19            474            703           2265
make                            17             76             13            869
Perl                             9            103            114            868
Markdown                         4             81              0            101
Lua                              1             11             10             27
CSS                              1              2              0             25
TOML                             1              0              1              1
-------------------------------------------------------------------------------
SUM:                          1187          77551          72318         401704
-------------------------------------------------------------------------------
```

Suricata 5 beta: Mandatory use of Rust

April 30, 2019

by inliniac

in news, release

Leave a comment

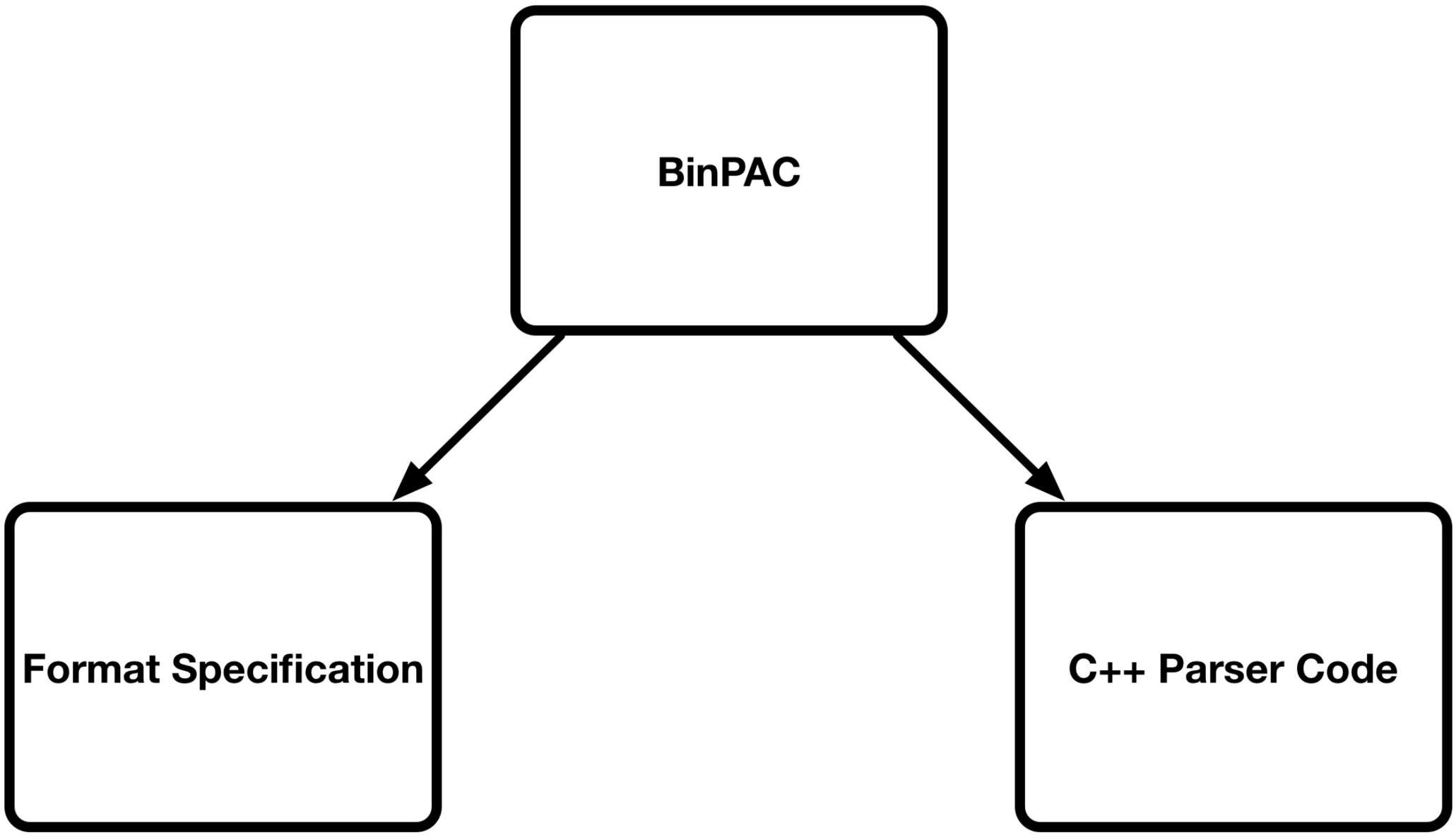
Call for testing: announcing Suricata 5.0.0-beta1

We're happy to present the first beta in the upcoming Suricata 5.0 series. In 5.0 we're making a couple of large changes.

Rust

The most visible is that our Rust support is no longer optional. We're convinced that Rust is a perfect match for Suricata, and we plan to increase its footprint in our code base steadily. By making it mandatory we're able to remove parallel implementations and focus fully on making the Rust code better.

Bro / Zeek: BinPAC parser generator



Problems with BinPAC

Bro 2.5.3

Bro 2.5.4

Bro 2.5.5

Bro 2.5.5 primarily addresses security issues.

- Fix array bounds checking in BinPAC: for arrays that are fields within a record, the bounds check was based on a pointer to the start of the record rather than the start of the array field, potentially resulting in a buffer over-read.

Problem #2: Signatures

Signatures

Can only detect **known threats**

Size of signature databases is continuously growing

Existing malware can be **obfuscated** to evade signature detection

!Problem?

Let’s do it in Go!

NETCAP

Decodes network packets and generates **audit records**

Uses the **gopacket library** (~80k LoC) for decoding packets

Concurrent design: **worker pool**, each audit record written to a separate file

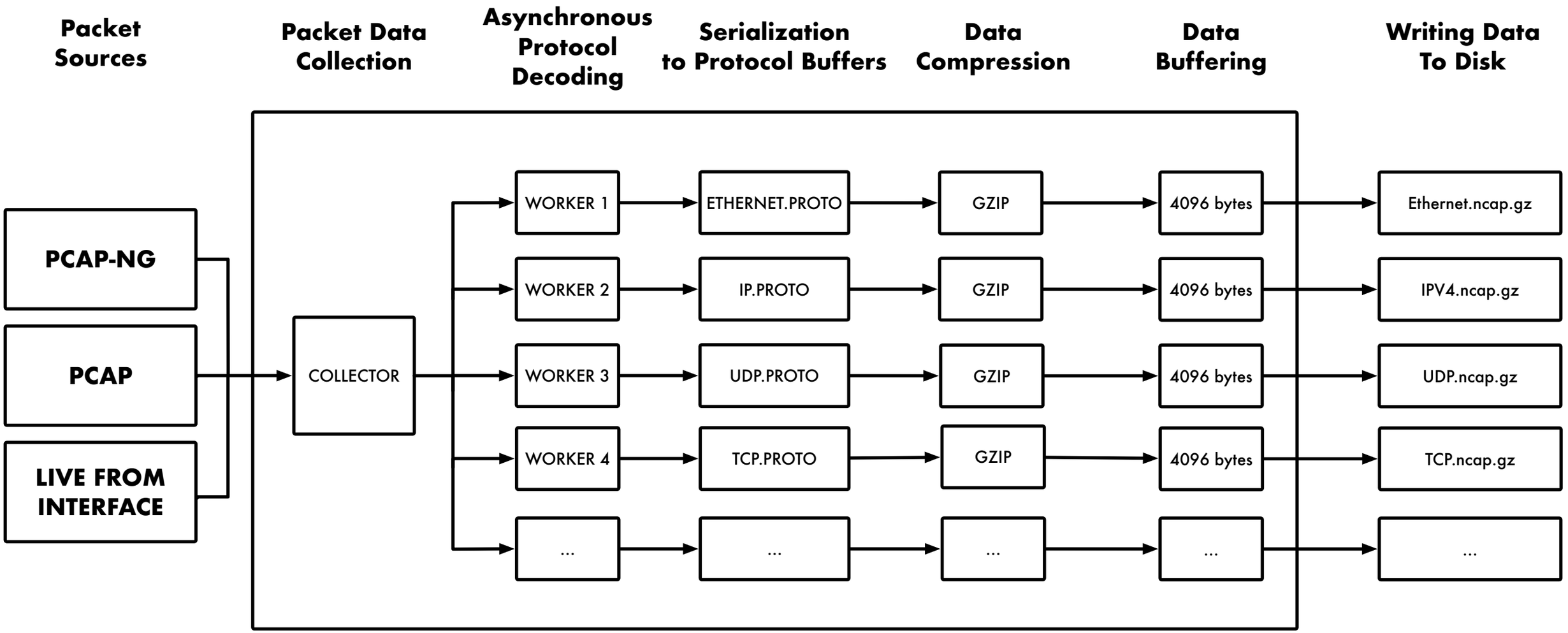
Audit record generation as **compressed protocol buffers**

Why protocol buffers?

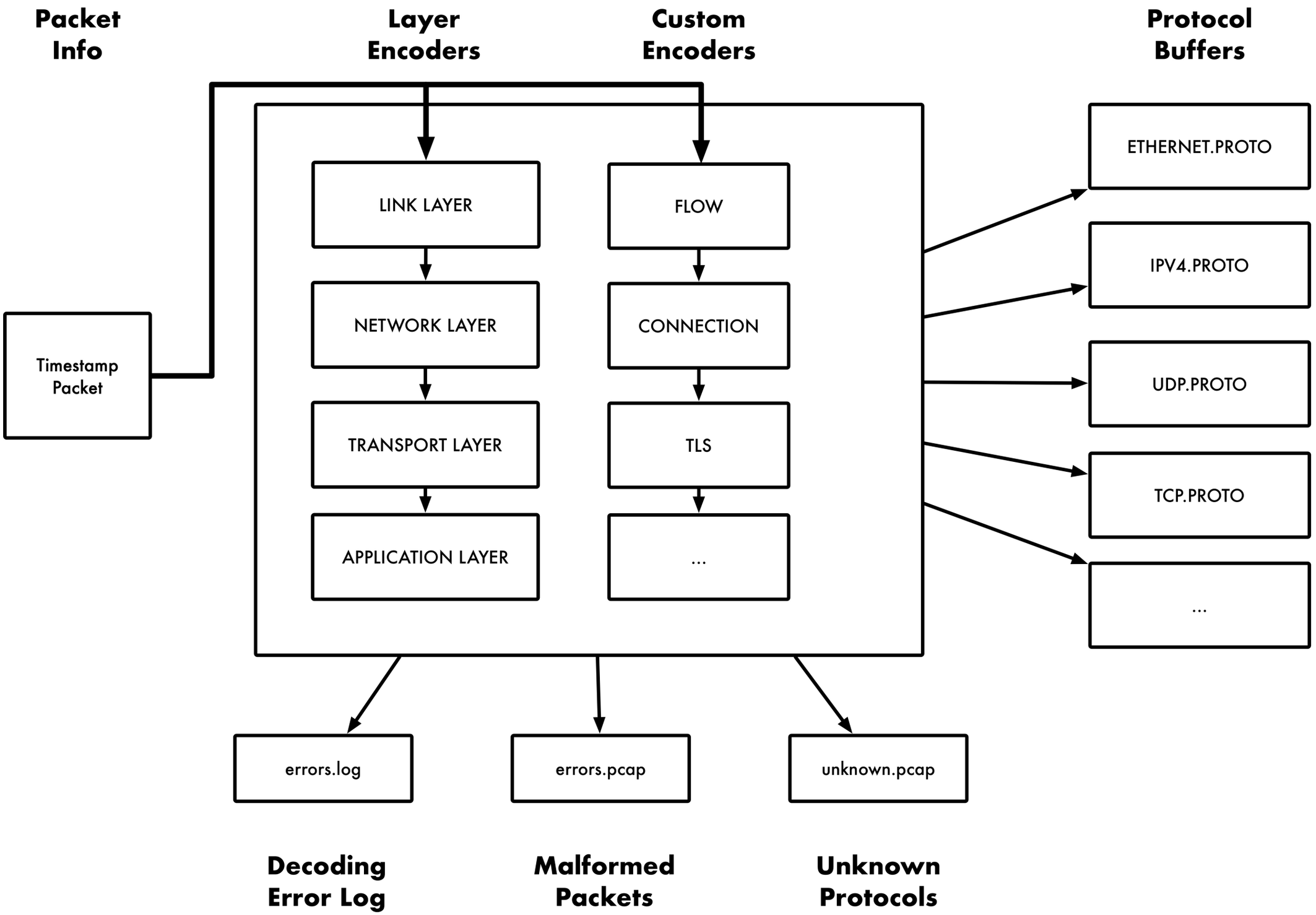
Type safe structured data - can represent complex nested structures

Platform neutral - generate type definitions for your favourite language

NETCAP in a nutshell



NETCAP worker

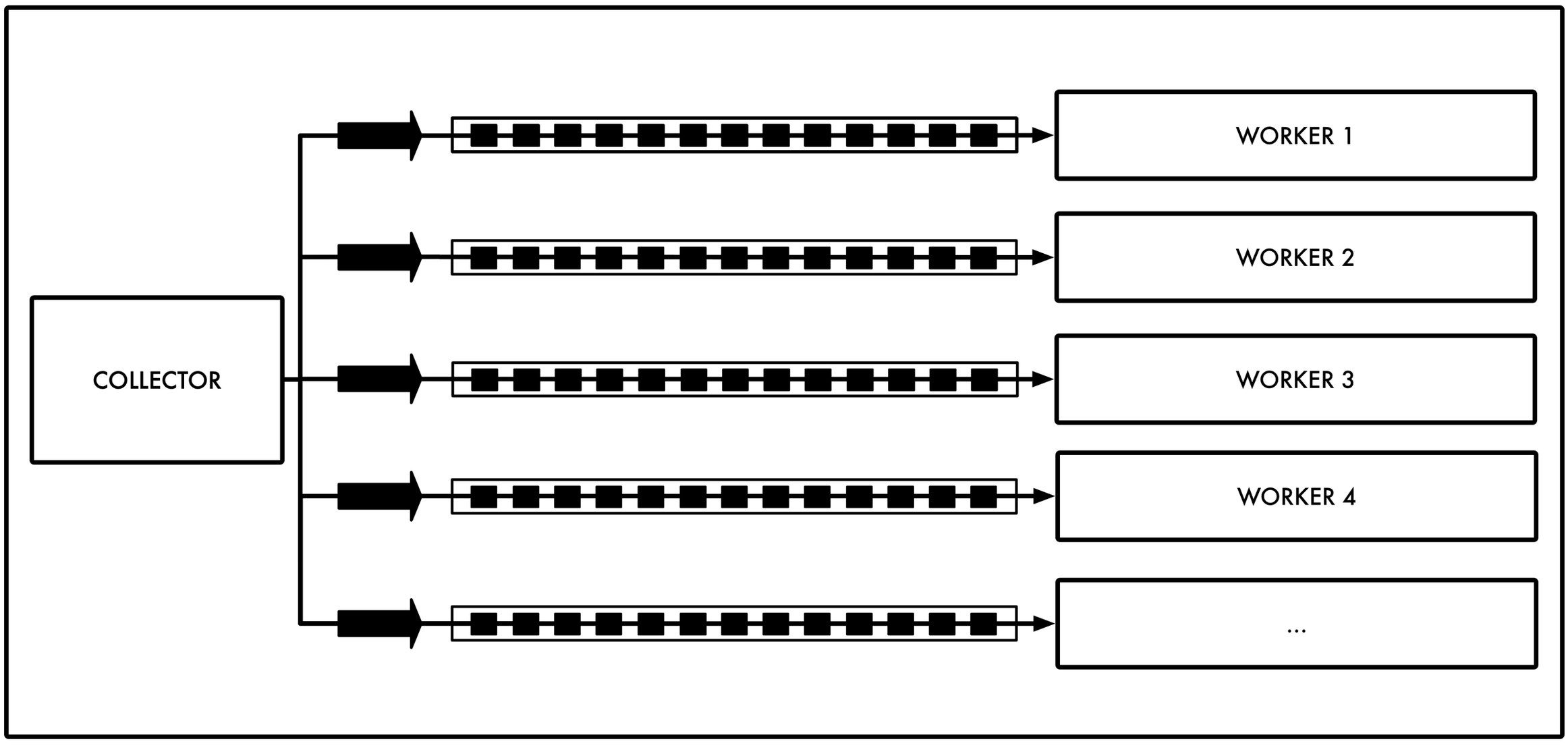


NETCAP buffered workers

Packet Data Distribution via Round Robin

Buffered Input Channel for each worker

Configurable Number of workers



Available Audit Records?

Custom:

+ **Flow (unidirectional)**
+ **Connection (bidirectional)**
+ LinkFlow (disabled by default)
+ NetworkFlow (disabled by default)
+ TransportFlow (disabled by default)
+ **TLS (Client Hello Msg + Ja3)**
+ **HTTP**

Available Audit Records?

Layers

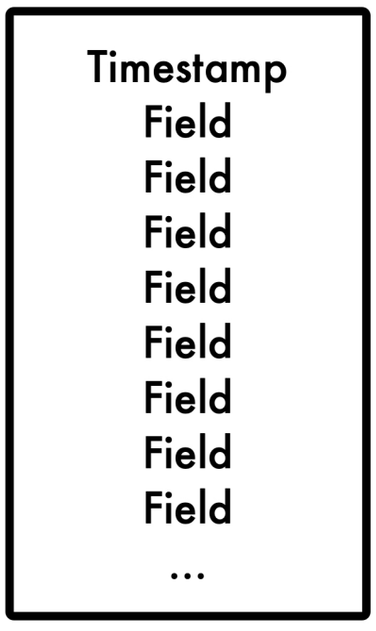
+ **TCP**
+ **UDP**
+ **IPv4**
+ **IPv6**
+ **DHCPv4**
+ **DHCPv6**
+ **ICMPv4**
+ **ICMPv6**
+ ICMPv6Echo
+ ICMPv6NeighborSolicitation
+ ICMPv6RouterSolicitation
+ **DNS**
+ **ARP**
+ **Ethernet**
+ Dot1Q
+ Dot11
+ NTP
+ **SIP**
+ **IGMP**
+ LLC
+ IPv6HopByHop
+ SCTP
+ **SNAP**
+ LinkLayerDiscovery
+ ICMPv6NeighborAdvertisement
+ ICMPv6RouterAdvertisement
+ EthernetCTP
+ EthernetCTPReply
+ LinkLayerDiscoveryInfo

v0.3.9

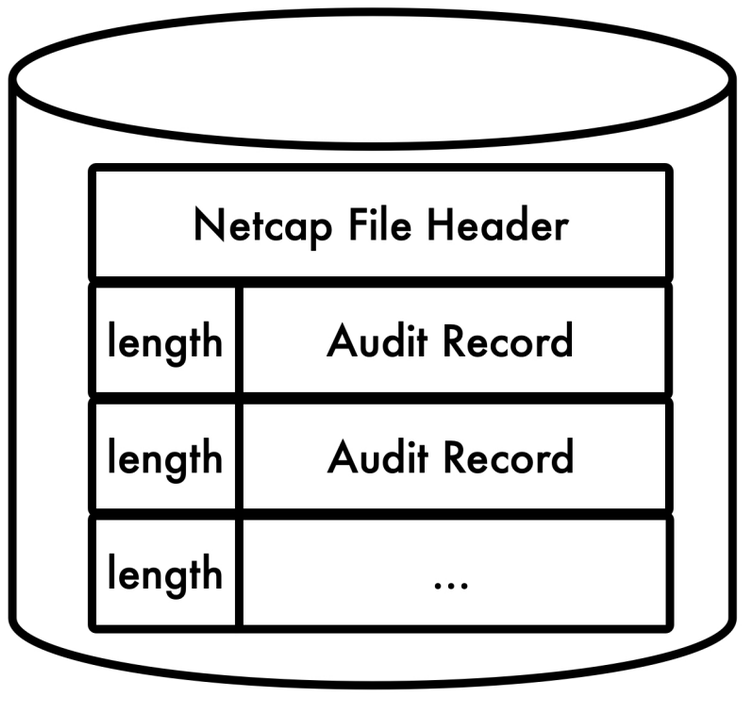
+ **OSPF**
+ BFD
+ GRE
+ FDDI
+ VRRPv2
+ EAP
+ **CiscoDiscovery**
+ NortelDiscovery
+ **IPSec**
+ Geneve
+ VXLAN
+ **USB**
+ LCM
+ MPLS
+ **ModbusTCP**

NETCAP audit records

Single Audit Record



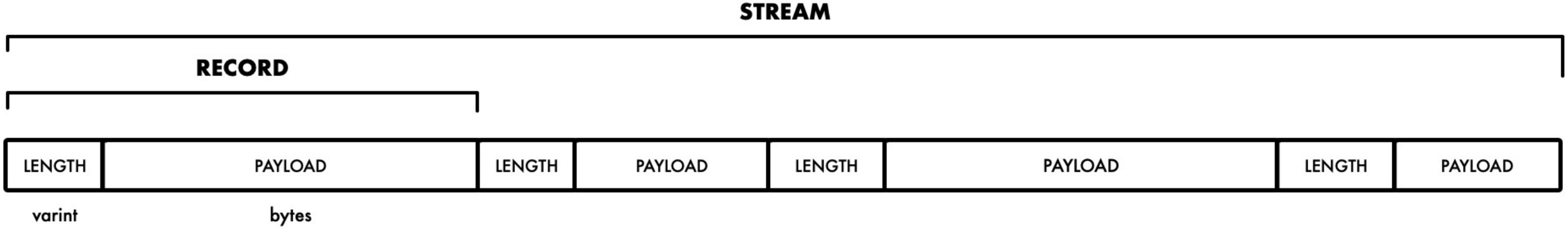
Audit Record File



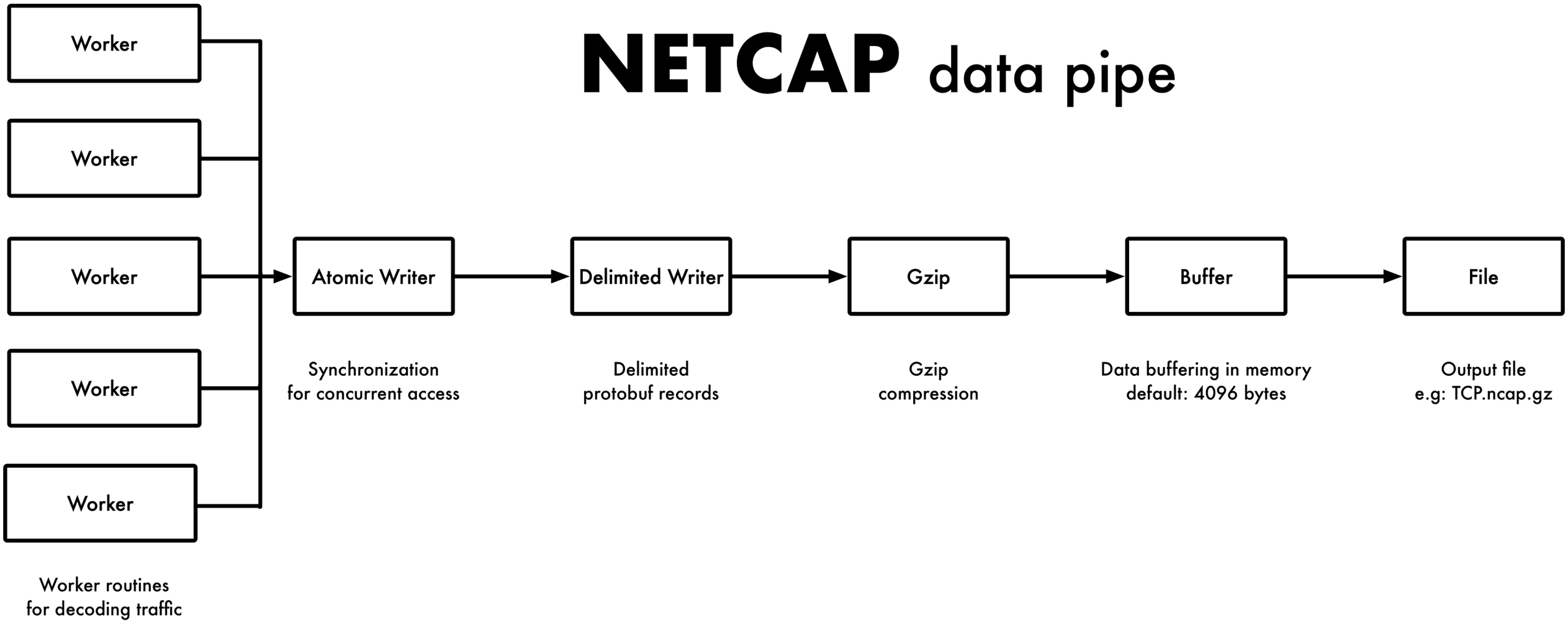
e.g:
TCP.ncap.gz (compressed)
TCP.ncap (uncompressed)

Flow / Connection	Example
Timestamp First Seen (seconds.micro)	1499257434.003136
Link Layer Protocol	Ethernet
Network Layer Protocol	IPv4
Transport Layer Protocol	TCP
Application Layer Protocol	HTTP
Source Mac Address	00:0c:28:9f:16:1e
Destination Mac Address	00:0c:28:c9:60:ce
SrcIP	173.15.11.103
SrcPort	1873
DstIP	95.100.238.75
DstPort	80
Size in bytes	922
Number of Packets	6
Timestamp Last Seen (seconds.micro)	1499257551.553088
Duration (nanoseconds)	117549952000

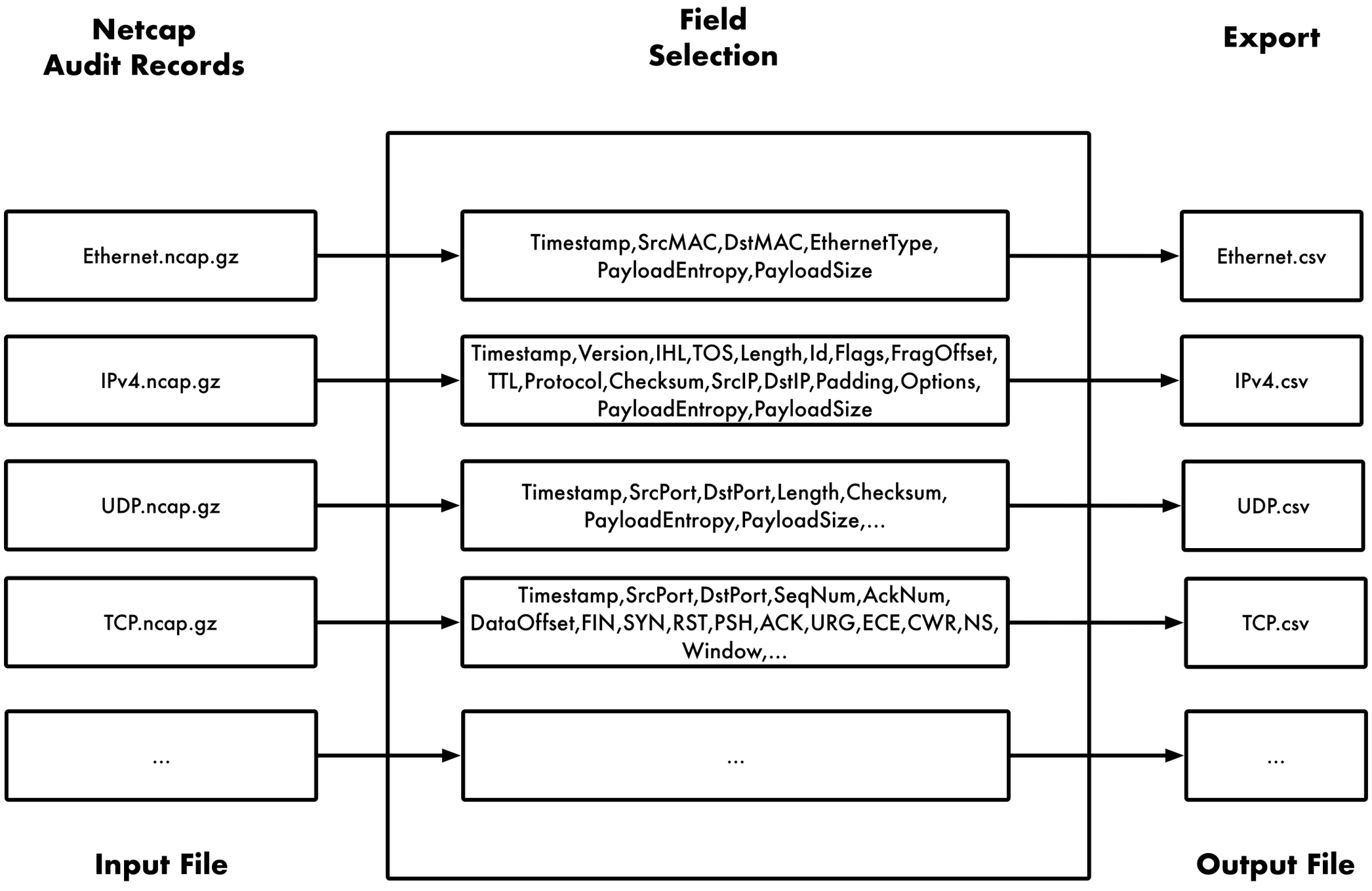
Format on disk: Length delimited audit records



NETCAP data pipe



NETCAP filtering & csv export

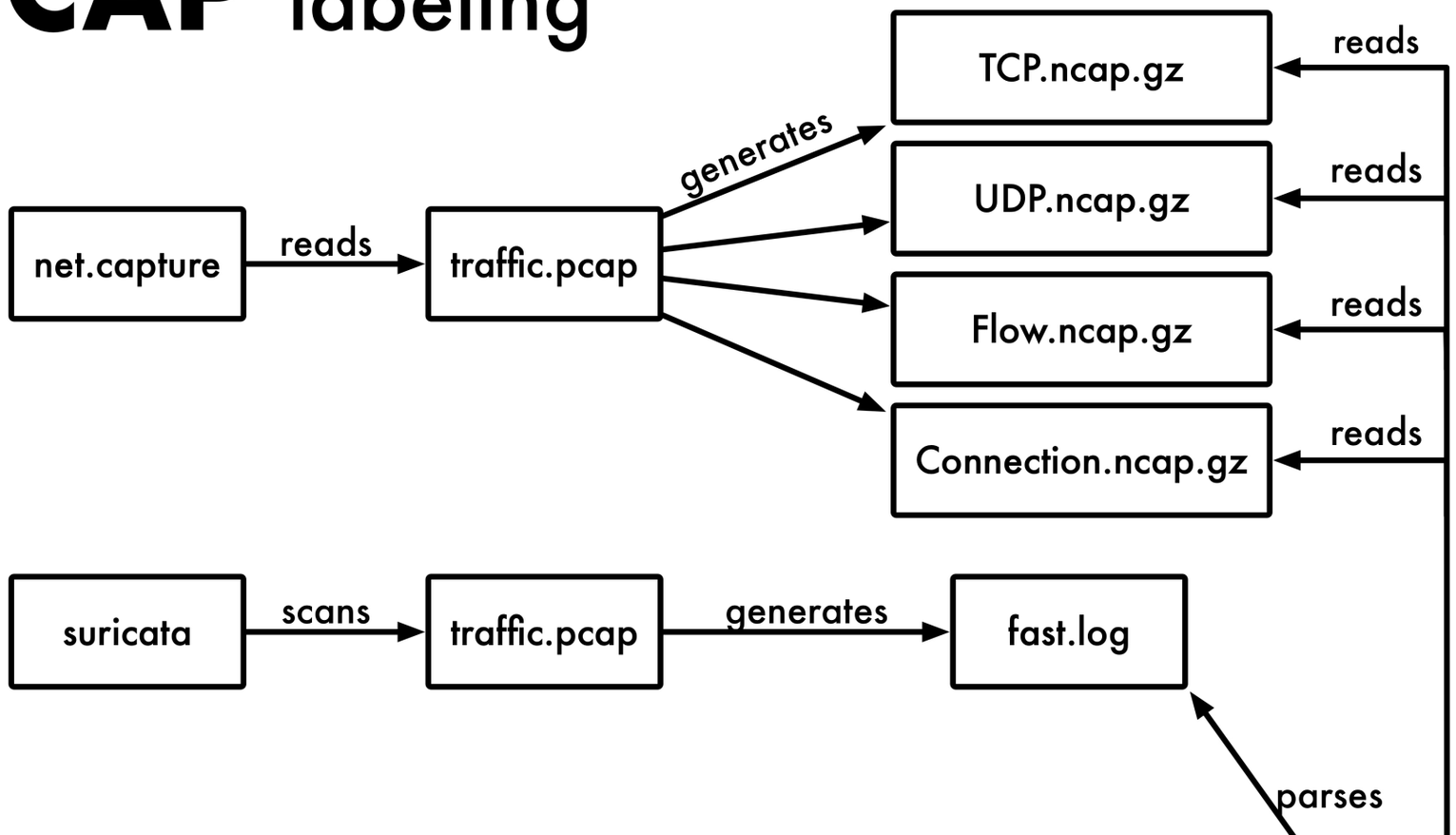



```
$ net.capture -r TCP.ncap.gz    -select Timestamp,SrcPort,DstPort,SeqNum,Window,ACK,SYN,RST   > TCP.csv
```

NETCAP labeling

Phase 1:

Data generation
with net.capture

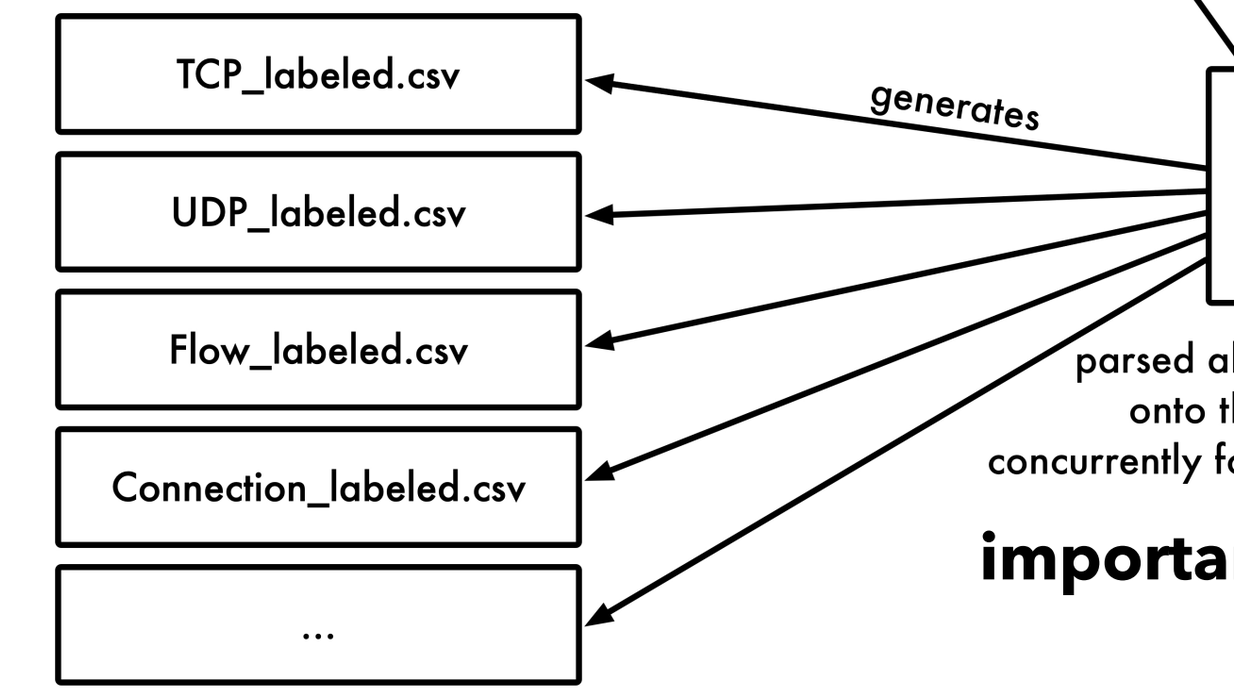


Phase 2:

Label extraction
with suricata

Phase 3:

Mapping alerts
with net.capture

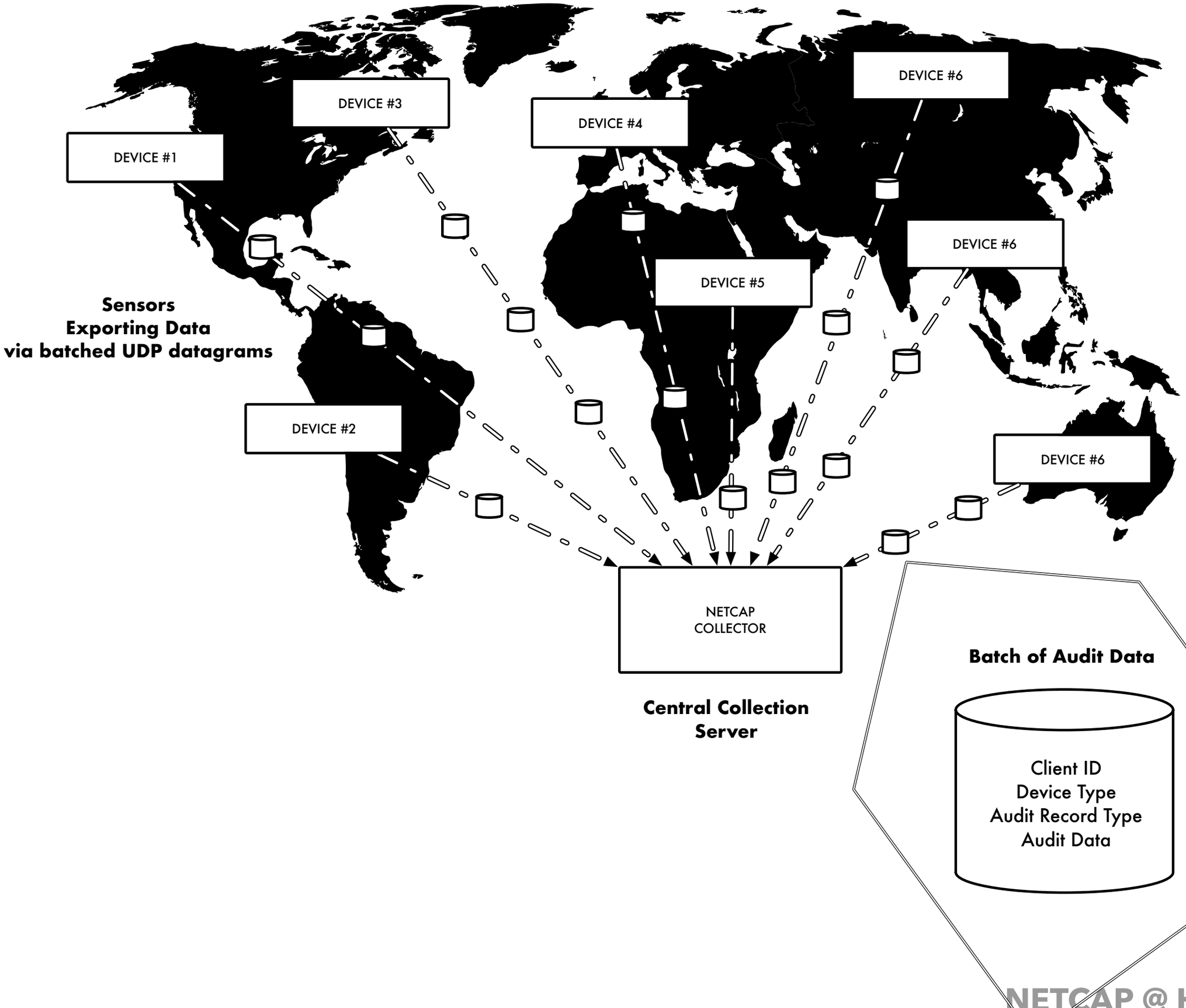


final data in CSV format
with mapped alerts for each record

parsed alerts will be mapped
onto the collected data
concurrently for each audit record type

important: granularity

NETCAP Sensors



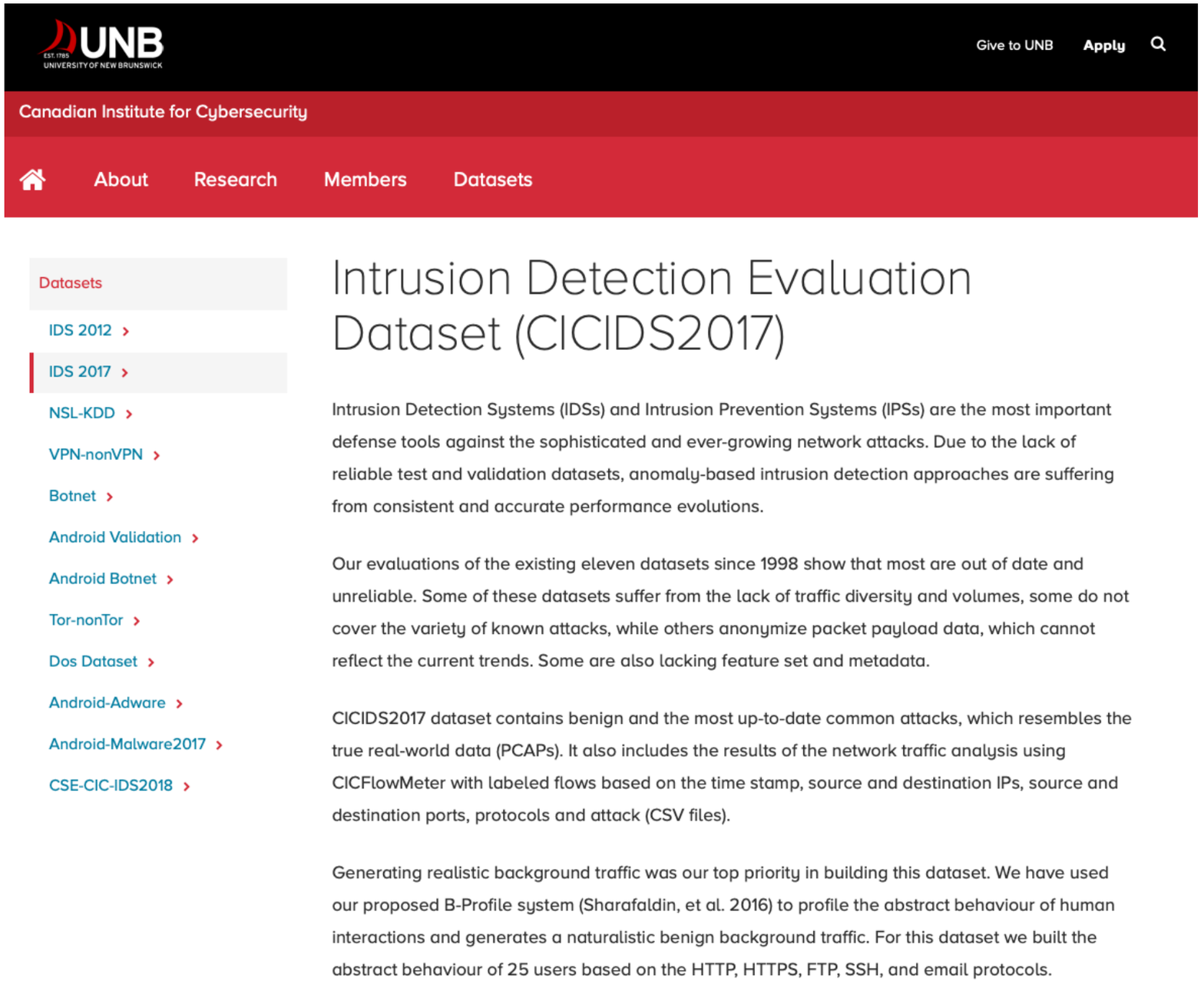
Use Cases?

Monitor honeypots

Forensic Analysis

Research! :) - GPLv3 license

Classification of malicious behaviour with NETCAP and a Deep Neural Network with Tensorflow



CICIDS2017 Dataset:

Up to date, 5 days of traffic

Well documented

~50GB **original PCAPs**

Monday: Normal Traffic

Tuesday: Brute Force

Wednesday: DoS

Thursday: Web Attacks

Friday: Botnet Traffic

Experiment takeaways

Encoding strategies are vital for performance

High detection accuracy (95-99.9%) can be achieved with a **handful of extracted features** (Flow / Connection Durations, Payload Size and Entropy)

Different approaches to labelling can be used to **increase value for analysts**

High accuracy for **protocol specific approach**

What’s new

Protobuf Serialisation Performance

with golang code generator:

```
$ go test -bench=. -v ./types
=== RUN   TestMarshal
--- PASS: TestMarshal (0.00s)
goos: darwin
goarch: amd64
pkg: github.com/dreadl0ck/netcap/types
BenchmarkMarshal-12        10000000         184 ns/op        64 B/op        1 allocs/op
BenchmarkUnmarshal-12      10000000         160 ns/op        40 B/op        2 allocs/op
PASS
ok      github.com/dreadl0ck/netcap/types   3.830s
```

with gogo code generator:

```
$ go test -bench=. -v ./types
=== RUN   TestMarshal
--- PASS: TestMarshal (0.00s)
goos: darwin
goarch: amd64
pkg: github.com/dreadl0ck/netcap/types
BenchmarkMarshal-12        20000000         89.1 ns/op       64 B/op        1 allocs/op
BenchmarkUnmarshal-12      20000000          110 ns/op       40 B/op        2 allocs/op
PASS
ok      github.com/dreadl0ck/netcap/types   4.215s
```

Payload Capture

Payload capture

It is now possible to capture payload data for the following protocols: TCP, UDP, ModbusTCP, USB

This can be enabled with the **-payload** flag:

```
net.capture -r traffic.pcap -payload
```

Also available for live capture:

```
net.capture -iface en0 -payload
```

USB Decoding

USB decoding

USB live capture is now possible, currently the following Audit Records exist: USB and USBRequestBlockSetup.

To capture USB traffic live on macOS, install wireshark and bring up the USB interface:

```
sudo ifconfig XHC20 up
```

Now attach netcap and set baselayer to USB:

```
net.capture -iface XHC20 -base usb
```

To read offline USB traffic from a PCAP file use:

```
net.capture -r usb.pcap -base usb
```

Configurable CSV Output

Configurable separators for CSV structures

The separator characters for structs in CSV output mode are now configurable via commandline flags.

Default is '(' for opening, '-' as separator for values and ')' for closing.

```
type Message struct {
    string Text
    bool   Secret
    int    MagicNumber
}
```

would appear in CSV like:

```
(Text-Secret-MagicNumber)
```

with the concrete field values:

```
(Hi-true-42)
```

Restructured Interface

Commandline Tools

The commandline tools have been restructured and the framework now consists of:

- net.**capture** (capture audit records live or from dumpfiles)
- net.**dump** (dump with audit records in various formats)
- net.**label** (tool for creating labeled CSV datasets from netcap data)
- net.**collect** (collection server for distributed collection)
- net.**agent** (sensor agent for distributed collection)
- net.**proxy** (http reverse proxy for capturing traffic from web services)
- net.**util** (utility tool for validating audit records and converting timestamps)
- net.**export** (exporter for prometheus metrics)

Golang Library Improvements:

netcap.Writer

The netcap library now exposes a data structure for writing audit records to disk.

Check out the GoDocs: https://godoc.org/github.com/dreadl0ck/netcap

```
type Writer
    func NewWriter(name string, buffer, compress, csv bool, out string, writeChan bool) *Writer
    func (w *Writer) Close() (name string, size int64)
    func (w *Writer) GetChan() <-chan []byte
    func (w *Writer) Write(msg proto.Message) error
    func (w *Writer) WriteCSV(msg proto.Message) (int, error)
    func (w *Writer) WriteCSVHeader(msg proto.Message) (int, error)
    func (w *Writer) WriteHeader(t types.Type, source string, version string, includesPayloads bool) error
    func (w *Writer) WriteProto(msg proto.Message) error
```

Configurable gopacket.DecodeOptions

Gopackets DecodeOptions are now configurable via commandline, three options exist:

- lazy (gopacket.Lazy)
- default (gopacket.Default)
- nocopy (gopacket.NoCopy)

By default, netcap uses the the lazy decoding option.

Protobuf type definitions in each release

Precompiled type definitions for:

+ Go
+ C++
+ Java
+ Rust
+ CSharp
+ JS
+ Python
+ Swift

Python Support

Retrieving the audit records as pandas dataframe:

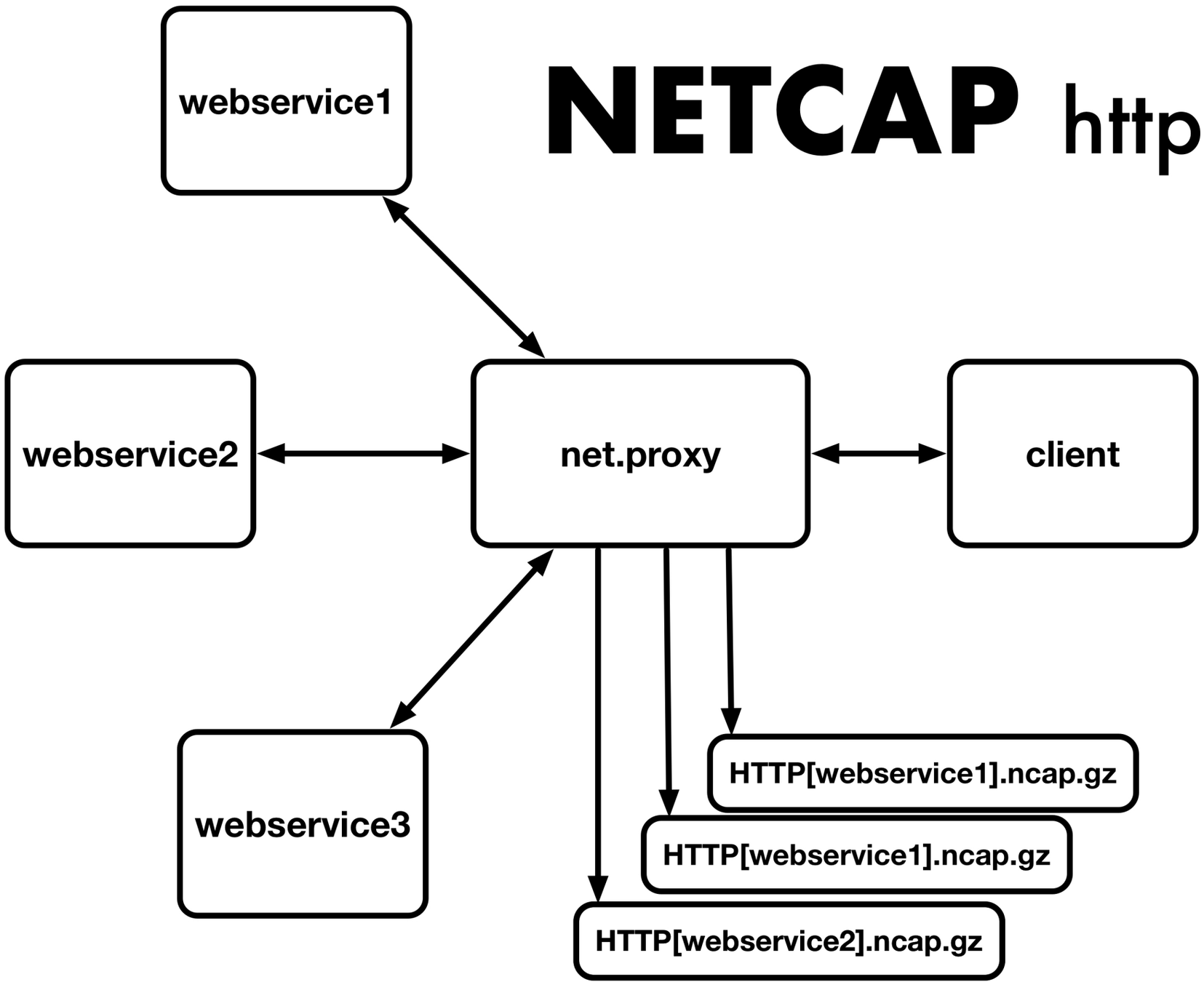
```python
#!/usr/bin/python

import pynetcap as nc

reader = nc.NCReader('pcaps/HTTP.ncap.gz')

reader.read(dataframe=True)
print("[INFO] completed reading the audit record file:", reader.filepath)
print("DATAFRAME:")
print(reader.df)
```

A proxy for web services



Enhanced HTTP audit records

Using the http tracing functionality from the go standard library,
several interesting time deltas have been added to the HTTP audit record type.

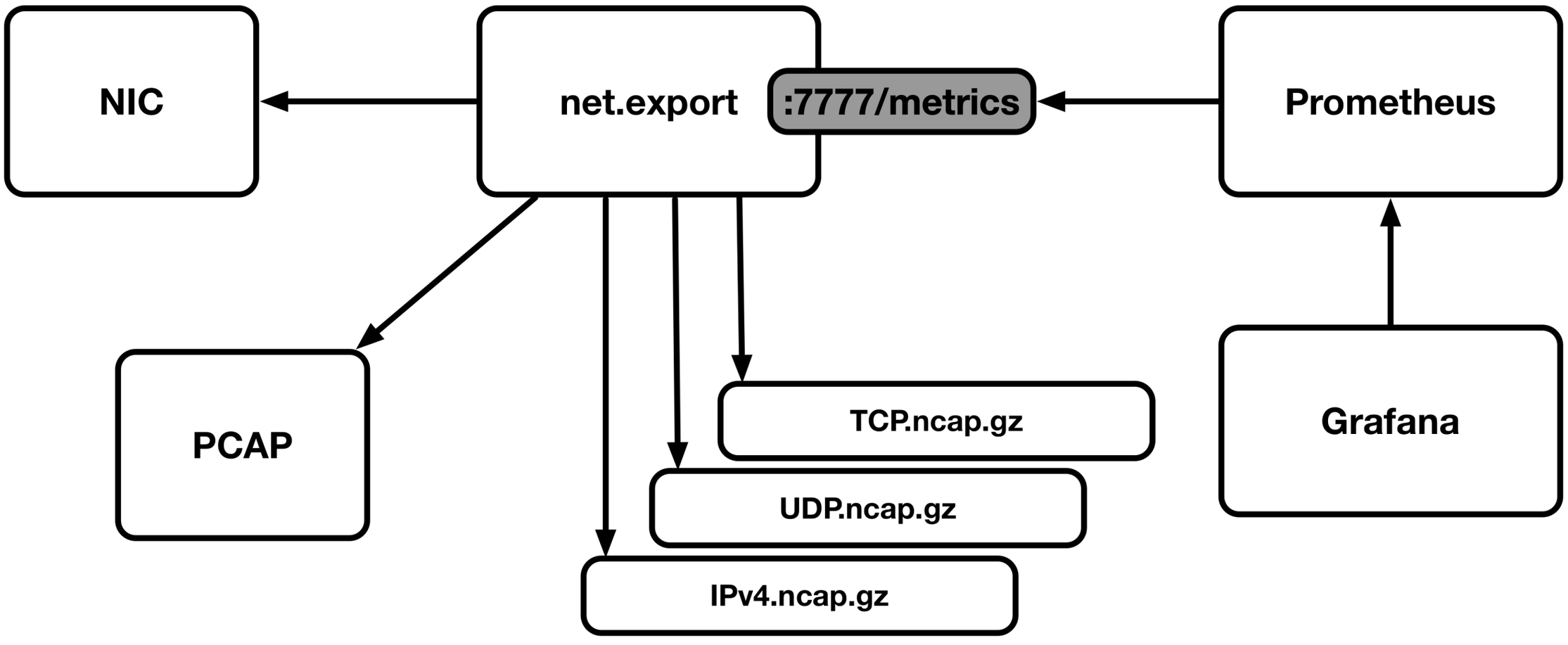
```
// Time Deltas (Nanoseconds)
// currently only available when using the HTTP proxy with tracing enabled.
int64  DoneAfter              = 20;
int64  DNSDoneAfter           = 21;
int64  FirstByteAfter         = 22;
int64  TLSDoneAfter           = 23;
```

Prometheus Metrics

- NETCAP related metrics (Protocols, Decoding errors etc)
- Go runtime related metrics (Number of goroutines, memory usage etc)
- Audit record related metrics (Field values and custom metrics)

Prometheus metrics

NETCAP metrics

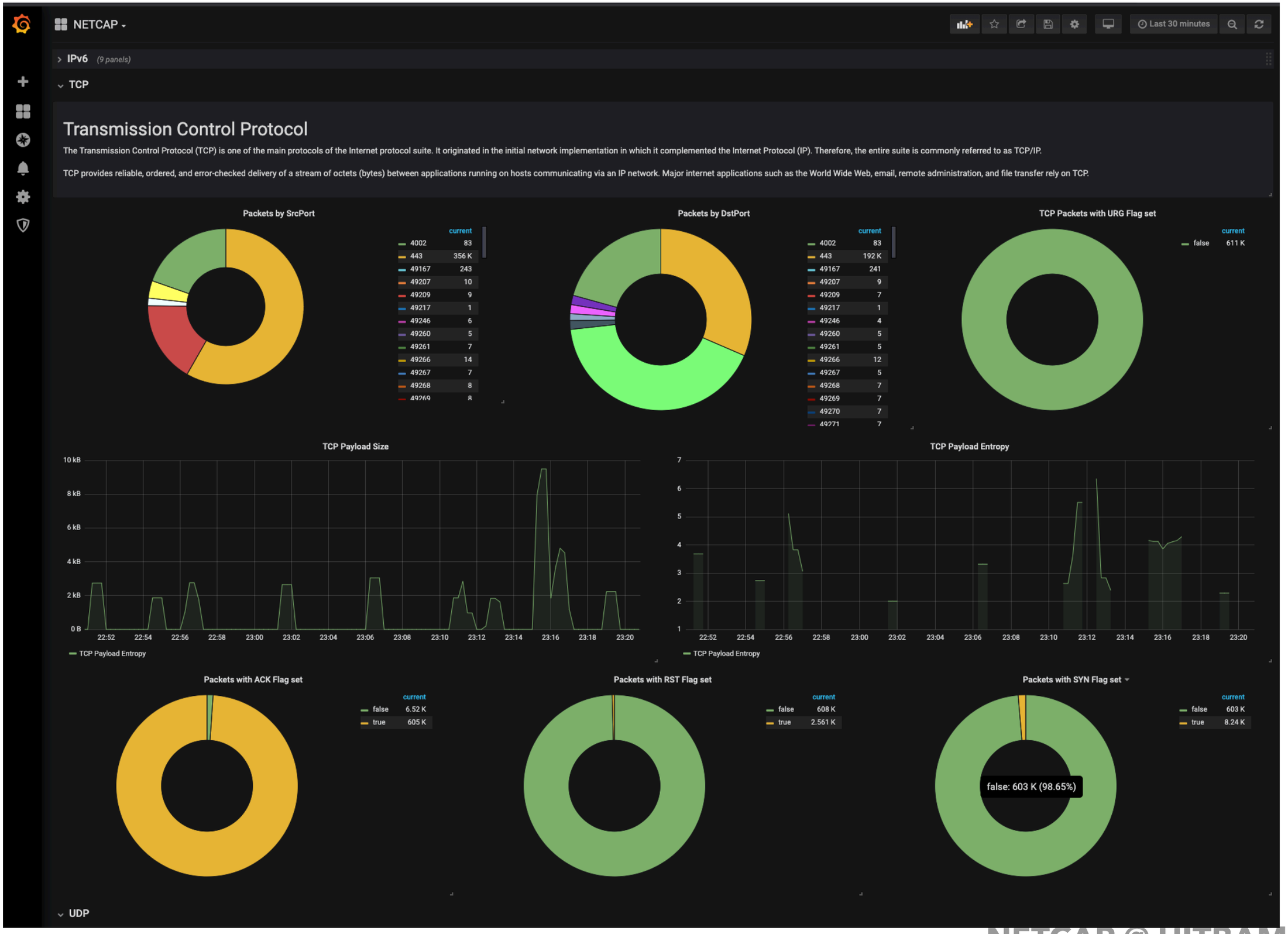


v0.4

Overview Dashboard

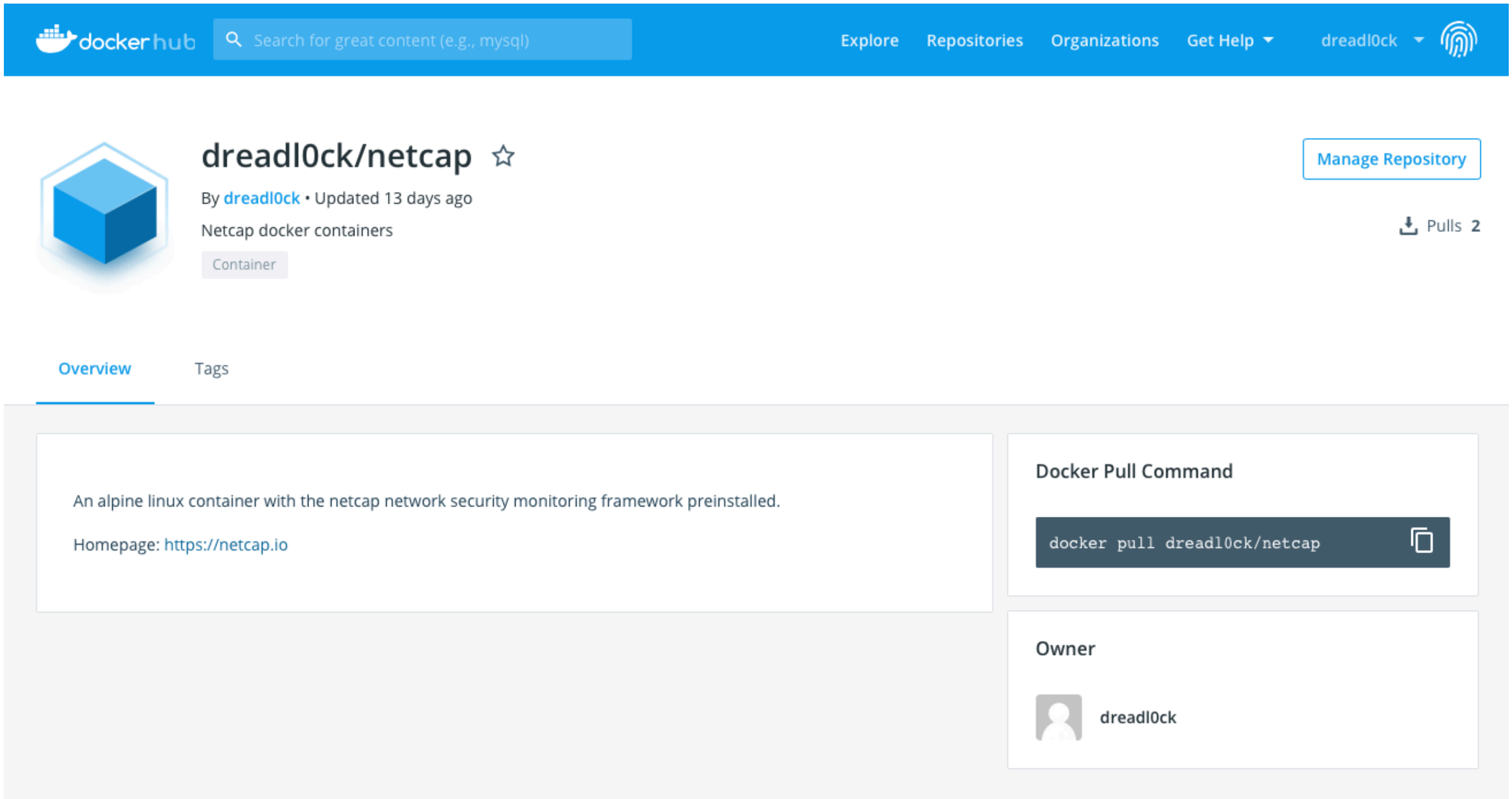
HTTP Dashboard

TCP Dashboard



v0.4

Alpine Linux Docker Image



Website

13:47 Wed 8. May

78 %

netcap.io

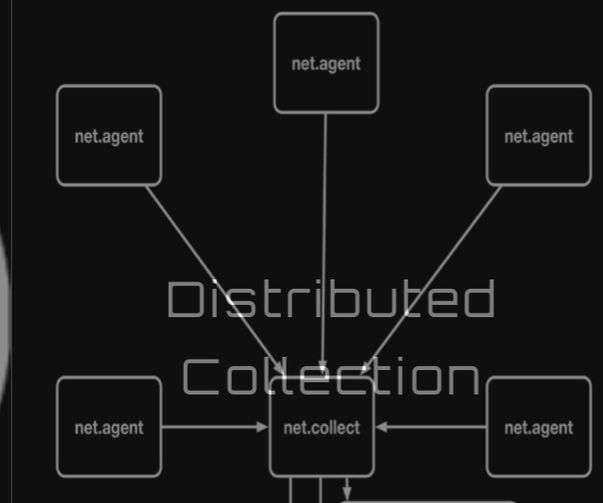
NETCAP Features Blog Download Docs

Concurrent Design

Commandline Interface

USB Capture

Open Source



Distributed Collection

PCAP Support

Live Capture

v0.4

Documentation

docs.netcap.io

Overview

A brief overview

The *Netcap* (NETwork CAPture) framework efficiently converts a stream of network packets into highly accessible type-safe structured data that represent specific protocols or custom abstractions. These audit records can be stored on disk or exchanged over the network, and are well suited as a data source for machine learning algorithms. Since parsing of untrusted input can be dangerous and network data is potentially malicious, implementation was performed in a programming language that provides a garbage collected memory safe runtime.

It was developed for a series of experiments in my bachelor thesis: *Implementation and evaluation of secure and scalable anomaly-based network intrusion detection*. Currently, the thesis serves as documentation until the wiki is ready, it is included at the root of this repository (file: mied18.pdf). Slides from my presentation at the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities are available on researchgate.

Future Development

- **YARA** support for labelling
- **benchmarks** & performance optimizations
- **Deep Packet Inspection Module** that looks for certain patterns in the payload to identify the application layer
- implement **IPv6 stream reassembly**
- implement an **interface for application layer decoders that require stream reassembly**

Questions?

dreadl0ck@protonmail.ch