How to train your Meerkat(s)
A journey from stock to specialization

Sascha Steinbiss, Robert Haist
DCSO Deutsche Cyber-Sicherheitsorganisation GmbH
EUREF-Campus 22
10829 Berlin
http://www.dcso.de
info@dcso.de
About DCSO

- Managed (IT-)Security Service ("MSS") Provider
- Founded and advised by German DAX 30 companies and scientific institutions
- CyberDefence Services
  - Incident Response
  - Threat Intelligence
  - Network Security Monitoring ("TDH")
- Focus on advanced attack detection, mitigation and attacker profiling
We started using Suricata in Incident Response context way before DCSO was founded.

- We needed to build a sensor for our MSS to track attackers in high volume networks.

- We don’t sell sensors — they are our tool of choice.

- We mostly need metadata parsers for $ALL\text{-}THE\text{-}THINGS$
MSS challenges

- Highly heterogenous customer networks
  - each deployment is unique
  - overlapping IP ranges, same traffic seen by different sensors, …

- Limited control of traffic acquisition (SPAN vs. tap)
  - mismatch: switch ↔ sensor capabilities
  - asymmetric routing
  - …

- Customer networks change without prior notification
  - “Where’s my traffic!?"
  - “Huh... am I supposed to see anything?”

Let’s have a look at some of our challenges and how we solved them.
Challenge: Traffic Acquisition

If your sensor is in a remote desert you don’t play with the kernel.

- We have all been there (probably)
- AF_PACKET vs. PF_RING vs. Capture Card
- We had PF_RING in production. Although the very helpful support of ntop we had kernel panics and reliability issues with (remote) updates and deployment.
- We went back to AF_PACKET and plan to adopt AF_XDP early with out Intel X710 cards.
Challenge: Mass deployment

Uh — we need HOW many sensors out by WHEN?! 

- Deployments are unique → so are Suricata configurations
- Scaling sensor roll-outs requires standardization and automation
  - well-defined + parallelizable: Debian + Ansible
  - unattended provisioning via preseeding
  - only high-level site-specific network configuration required
- Move variation into hardware and ship pre-configured Suricata
  - baseline later when traffic is seen
- One sensor model to rule them all
Challenge: Data flows

It's all fun and games until you hit ≈15 Gbit/s (up to 50k events/s)

Sensor side
- ELK: obvious start, but does not scale on a single machine
- Same for other local (persistent) storage engines (MongoDB, PostgreSQL, ...)
- Abandoned local storage, focus on local processing and forwarding results
- Tried Redis, fell in love and built a processing chain around it (more in a minute)

Backend side
- Apache NiFi: message broker for events in the back-end lead to...
- Java stack traces everywhere
- Tried our luck with RabbitMQ and never looked back
- Customer sites or some of their remote locations might have slow upstream, so we need to be picky about what to send home
Challenge: Sensor side vs. backend processing

We can’t store everything on the sensor && we can’t send everything home.

Produce and polish data at source

- Need a way to operationalize event selection/enrichment/aggregation
- FEVER orchestrates parallel processing handlers
- Handlers subscribe to event types, communicate with backend
- Structure outgoing data as desired, compress as needed

Get data where they need to be

- Backend consumers can work at scale: more workers, more space, more everything
- Multiple components will eventually require the same data
- Promote less monolithic consumers by providing common sinks and sources

8/22 November 15, 2018 Sascha Steinbiss, Robert Haist
Building passive enrichment capabilities using metadata

“You know the technologies you intended to use in that network. We know the technologies that are actually in use in that network. Subtle difference.” — Rob Joyce

Network enumeration
- Support analysts by annotating seen assets with tags: internal, external, proxy, $service, ...
- Tags can reliably be assigned via aggregated flows $\langle ip_s, ip_d, port_d \rangle$ and HTTP_HOST per request
- Broaden view to netranges
- Augment with customer-provided metadata (e.g. location)

Passive DNS
- Aggregate DNS answers by $\langle rrname, rrtype, rdata, sensorID \rangle$
- Submit tuples + counts per time period
- Provide unified view of observations via GraphQL interface
- Data model conforms to COF
- Result: free server software *balboa*
- Supports third party collectors/transport
Challenge: Sensor baselining

It’s hard to manage sensor specific variables (e.g. $HOME_NET, $PROXY_SERVER, $DOMAIN_CONTROLLER) if you have a lot of sensors!

- Default settings → lot of rules not firing
- Customer-provided network info very diverse in {correct, complete} -ness
- Manual XLS/CSV/XML/... wrangling error-prone and not scalable
- Solution: automatically generate YAML with vars from
  - $HOME_NET → RFC1918/5735 + ranges tagged as ‘internal’
  - $PROXY_SERVERS → hosts tagged with ‘proxy’ tag
  - $DOMAIN_CONTROLLERS → hosts tagged with DC tag
  - $*_SERVERS → hosts tagged with specific protocol tag
- Additionally: (de-)activate different rulesets and parsers based on observed traffic
Challenge: No control of network input

Without control of traffic sources, ensuring visibility at customer sites is like herding cats.

- Physical sensor installation done by customers/contractors/…
- Highly diverse levels of knowledge, expertise and/or authority
- Network changes are rarely communicated
  - Changed firewall rules (Sensor cut off from backend)
  - SPAN port is disabled/repurposed (complete LoV)
  - Connection of additional monitoring interfaces (partial LoV)
  - Adjustment of traffic volume (possible LoV)
- **slinkwatch**: dynamic configuration and maintenance of monitoring interfaces
  - auto-detection of interface link and traffic change (for RX only)
  - updates interface entries in Suricata’s config YAML, service restart via systemd/init
  - dynamic allocation of threads per interface
Challenge: Performance monitoring

Sensor 1

SURICATA → telegraf
FEVER

DCSO backend

kapacitor

influxdb

RabbitMQ

Grafana

chronograf

Sensor n

SURICATA → telegraf
FEVER

13/22 November 15, 2018 Sascha Steinbiss, Robert Haist
Challenge: Performance monitoring (basic stats)

Plain telegraf
Challenge: Performance monitoring (traffic)

FEVER

Patched telegraf
Challenge: Performance monitoring (Suricata internals)
Challenge: Performance monitoring (ethtool)

NIC statistics
- measure and correlate NIC level stats with Suricata performance indicators
- useful in debugging potential issues
- enabled by ethflux tool exposing all ethtool values on interface(s)
Challenge: Performance monitoring (overview)
This PR adds a new input plugin for the runtime statistics of the Suricata IDS/IPS system (https://suricata-ids.org). It provides a socket for an appropriately configured Suricata instance to write its data into, and takes care of parsing the received JSON into measurements and fields.


Unit tests are present with a coverage of 97.8%, no race conditions were detected by `go test -cover -race -v ./plugins/inputs/suricata`.

**Required for all PRs:**

- Signed CLA.
- Associated README.md updated.
- Has appropriate unit tests.
Lessons Learned

- Automate, automate, automate (Ansible)
- Be the master of your own house (only depend on own infra in isolated VPN)
- Message-queue all the things (RabbitMQ allows easy access in back-end)
- Monitor en detail (see non-obvious things failing ASAP)
- Build your own specialized components if off-the-shelf stuff does not work
- Stay on the beaten path if requirements aren’t too special (AF_PACKET vs. PF_RING, ...)
- Only stream-process stuff on the sensor, no storage/lookup
- Open source is your/our friend
Open source releases

**fever** fast, extensible, versatile event router for Suricata’s EVE-JSON format
https://github.com/DCSO/fever · BSD-3-clause

**balboa** server for indexing and querying passive DNS observations
https://github.com/DCSO/balboa · BSD-3-clause

**slinkwatch** automatic enumeration and maintenance of Suricata monitoring interfaces
https://github.com/DCSO/slinkwatch · GPLv2

**ethflux** InfluxDB data gatherer for ethtool-style network interface information
https://github.com/DCSO/ethflux · BSD-3-clause

**bloom** highly efficient Bloom filter library and command line tool written in Go
https://github.com/DCSO/bloom · BSD-3-clause

Available in Debian buster (and stretch-backports) soon.
Questions?
Talk to us!

@ssatta
@RobertHaist