D4 Project

Open and collaborative network monitoring

Team CIRCL
https://www.d4-project.org/

20190207

Alexandre Dulaunoy - Sami Mokaddem

Problem statement

- CSIRTs (or private organisations) build their **own honeypot, honeynet or blackhole monitoring network**
- Designing, managing and operating such infrastructure is a tedious and resource intensive task
- **Automatic sharing** between monitoring networks from different organisations is missing
- Sensors and processing are often seen as blackbox or difficult to audit

Objective

- Based on our experience with MISP[1] where sharing played an important role, we transpose the model in D4 project
- Keeping the protocol and code base **simple and minimal**
- Allowing every organisation to **control and audit their own sensor network**
- Extending D4 or **encapsulating legacy monitoring protocols** must be as simple as possible
- Ensuring that the sensor server has **no control on the sensor** (unidirectional streaming)
- Don't force users to use dedicated sensors and allow **flexibility of sensor support** (software, hardware, virtual)

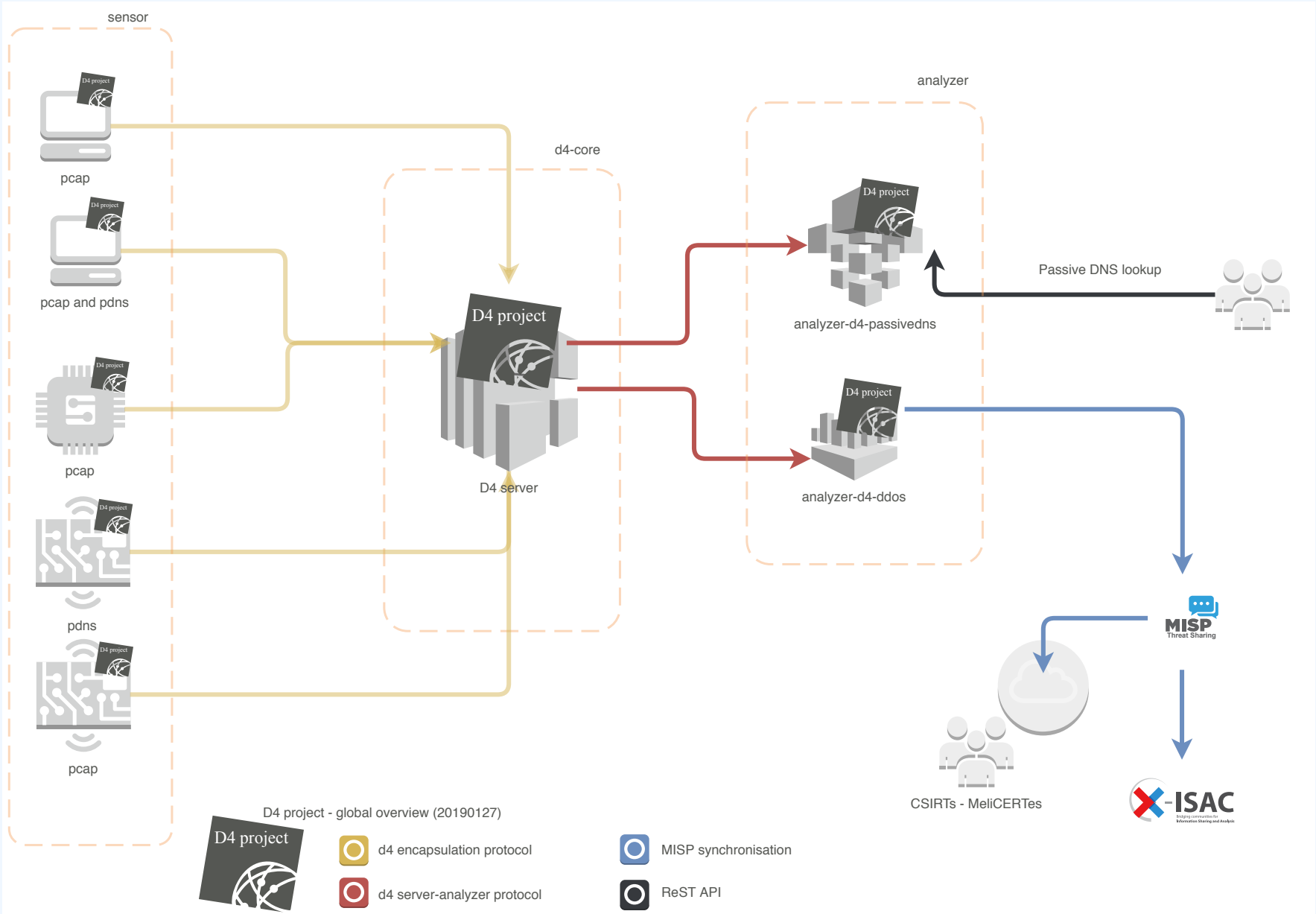
[1] https://github.com/MISP/MISP

(short) History

- D4 Project (co-funded under INEA CEF EU program) started - 1st November 2018
- D4 encapsulation protocol version 1 published - 1st December 2018
- v0.1 release of the D4 core[2] including a server and simple D4 C client - 21st January 2018
- First version of a golang D4 client[3] running on ARM, MIPS, PPC and x86 - January 2018

[2] https://www.github.com/D4-project/d4-core
[3] https://www.github.com/D4-project/d4-goclient/

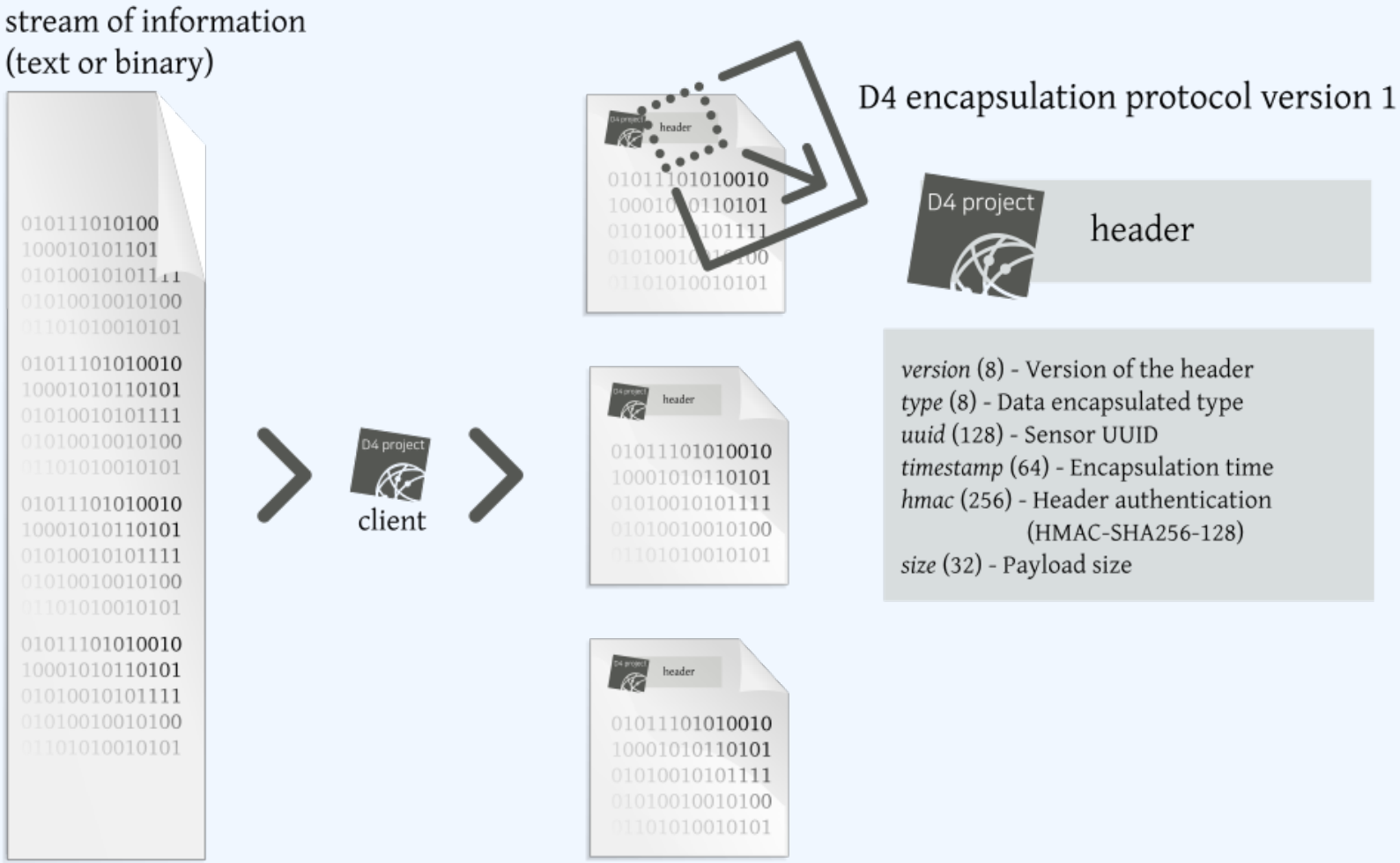
D4 Overview


sensor
pcap
pcap and pdns
pcap
pdns
pcap
d4-core
D4 project
D4 server
analyzer
D4 project
analyzer-d4-passivedns
Passive DNS lookup
D4 project
analyzer-d4-ddos
MISP
Threat Sharing
CSIRTs - MeliCERTes
ISAC
D4 project - global overview (20190127)
D4 project
d4 encapsulation protocol
d4 server-analyzer protocol
MISP synchronisation
ReST API

Roadmap (next 2 months)

- Passive DNS analyzer (alpha version released)
- Passive SSL collector and analyzer
- Backscatter DDoS traffic analyzer
- **Default server** (blackhole monitoring or Passive DNS collector) at CIRCL for organisations willing to contribute without running their own D4 server

D4 encapsulation protocol



https://www.d4-project.org

D4 Header

Name	bit size	Description
version	uint 8	Version of the header
type	uint 8	Data encapsulated type
uuid	uint 128	Sensor UUID
timestamp	uint 64	Encapsulation time
hmac	uint 256	Authentication header (HMAC-SHA-256-128)
size	uint 32	Payload size

D4 Header

Type	Description
0	Reserved
1	pcap (libpcap 2.4)
2	meta header (JSON)
3	generic log line
4	dnscap output
5	pcapng (diagnostic)
6	generic NDJSON or JSON Lines
7	generic YAF (Yet Another Flowmeter)
8	passivedns CSV stream
254	type defined by meta header (type 2)

D4 meta header

D4 header includes an easy way to **extend the protocol** (via type 2) without altering the format. Within a D4 session, the initial D4 packet(s) type 2 defines the custom headers and then the following packets with type 254 is the custom data encapsulated.

```
{
  "type": "ja3-jl",
  "encoding": "utf-8",
  "tags": [
    "tlp:white"
  ],
  "misp:org": "5b642239-4db4-4580-adf4-4ebd950d210f"
}
```

D4-core server

- D4 core server[4] is a complete server to handle clients (sensors) including the decapsulation of the D4 protocol, control of sensor registrations, management of decoding protocols and dispatching to adequate decoders/analysers.
- D4 server is written in Python 3.6 and runs on standard GNU/Linux distribution.

[4] https://github.com/D4-project/d4-core

D4 server handling

D4 server reconstructs the encapsulated stream from the D4 sensor and saves it in a Redis stream.

- Support TLS connection
- Unpack D4 header
- Verify client secret key (HMAC)
- check blocklist
- Filter by types (Only accept one connection by type-UUID - except: type 254)
- Discard incorrect data
- Save data in a Redis Stream (unique for each session)

D4 server - worker handler

After the stream is processed depending of the type using dedicated worker.

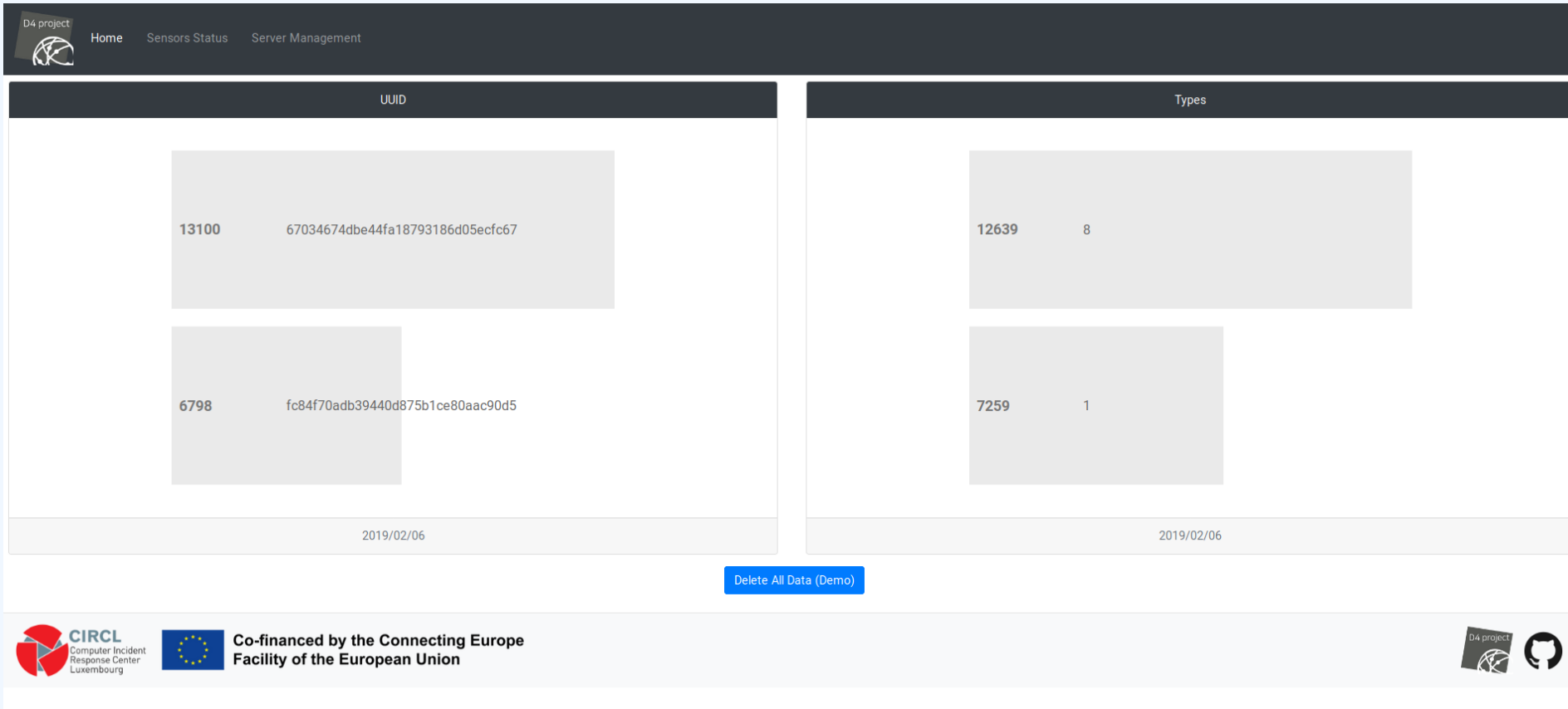
- Worker Manager (one by type)
  - Check if a new session is created and valid data are saved in a Redis stream
  - Launch a new Worker for each session
- Worker
  - Get data from a stream
  - Reconstruct data
  - Save data on disk (with file rotation)
  - Save data in Redis. Create a queue for D4 Analyzer(s)

D4 server - management interface

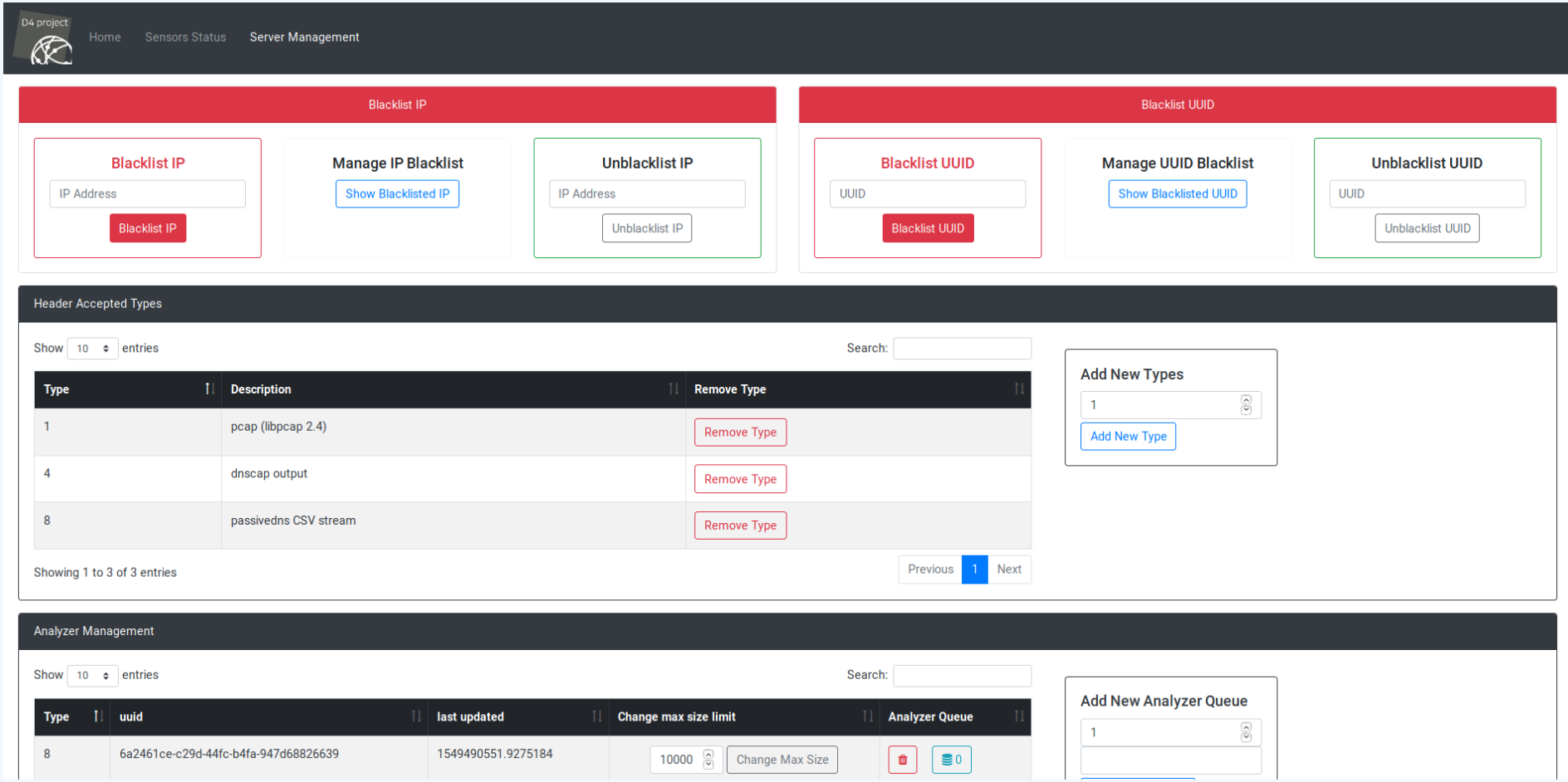
The D4 server provides a web interface to manage D4 sensors, sessions and analyzer.

- Get Sensors status, errors and statistics
- Get all connected sensors
- Manage Sensors (stream size limit, secret key, ...)
- Manage Accepted types
- UUID/IP blocklist
- Create Analyzer Queues

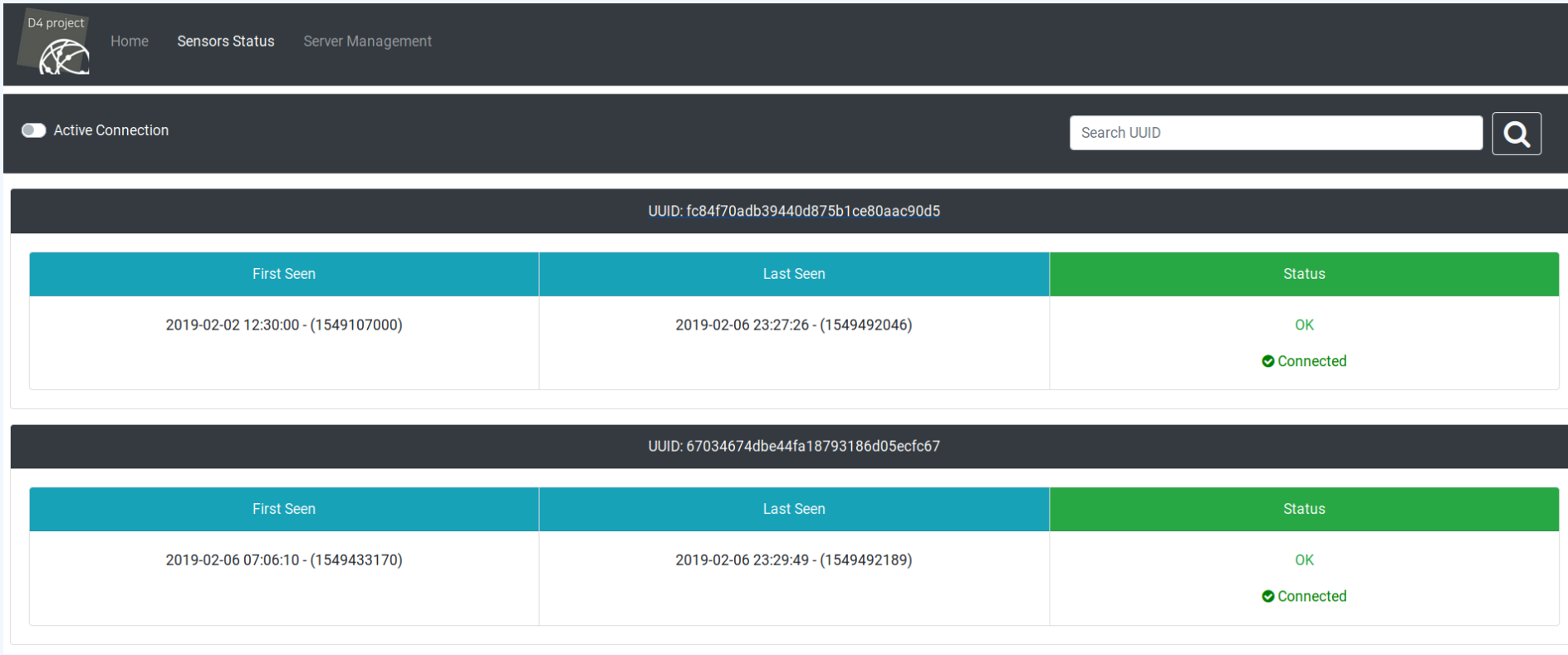
D4 server - main interface



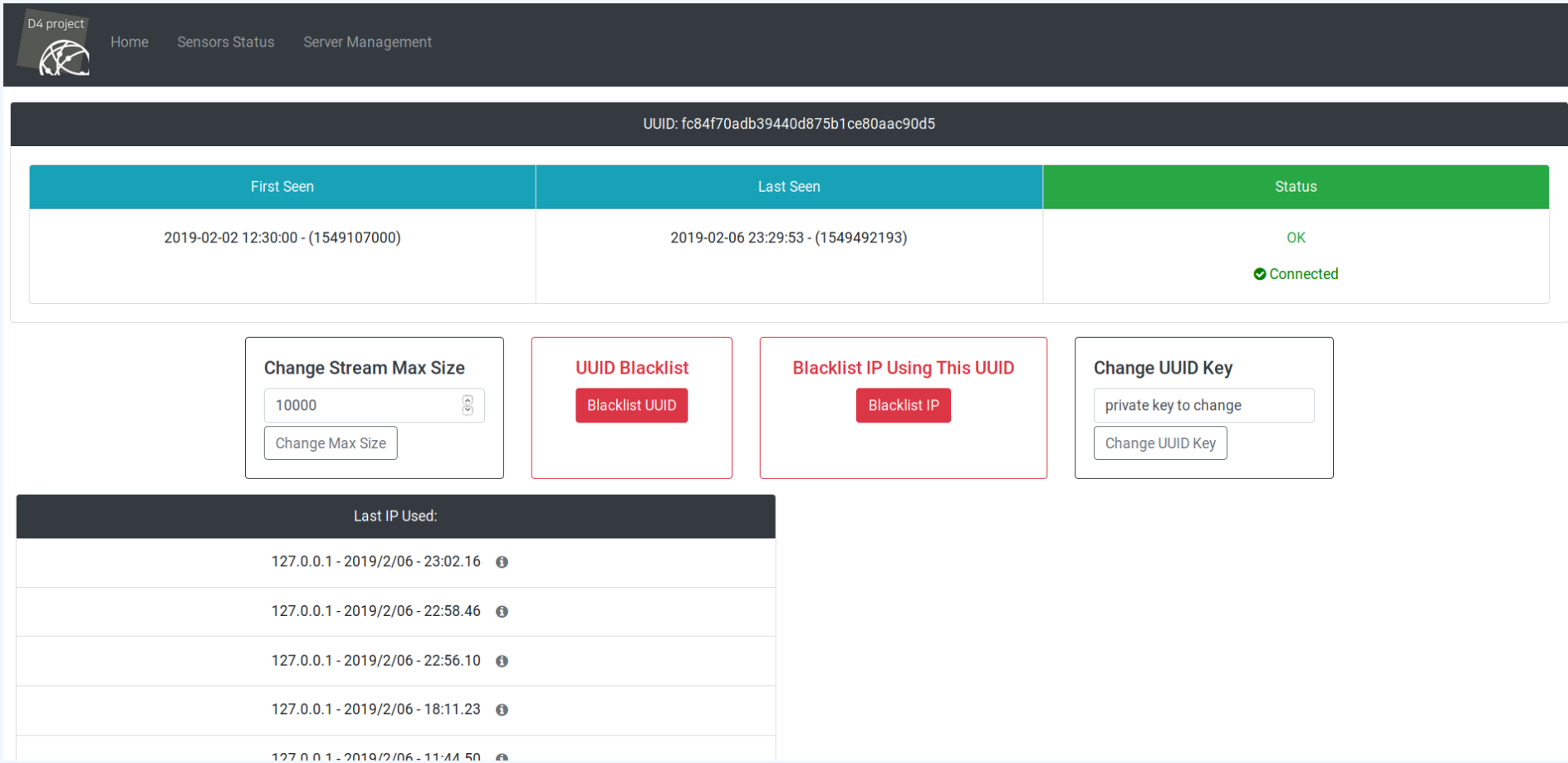
D4 server - server management



D4 server - sensor overview



D4 server - sensor management



Use-case: migrating a legacy network capture model into a D4 network sensor

Remote network capture

CIRCL operated honeybot for multiple years using a simple model of remote network capture.

Definition (Principle)

- KISS (Keep it simple stupid) - Unix-like
- Linux & OpenBSD operating systems

Sensor

```
tcpdump -l -s 65535 -n -i vr0 -w - '( not port
   $PORT and not host $HOST )' | socat - OPENSSL
   -CONNECT:$COLLECTOR:$PORT,cert=/etc/openssl/
   client.pem,cafile=/etc/openssl/ca.crt,verify=1
```

Remote network capture

Limitations

- Scalability $\rightarrow$ one port per client
- Identification and registration of the client
- Integrity of the data

Multiplexing streams in D4

- Inspired by the unix command `tee`
- Read from standard input
- Add the d4 header
- Write it on standard output

Using D4 native client

```
tcpdump -n -s0 -w - | ./d4 -c ./conf | socat -
   OPENSSL-CONNECT:$D4-SERVER-IP-ADDRESS:$PORT,
   verify=1
```

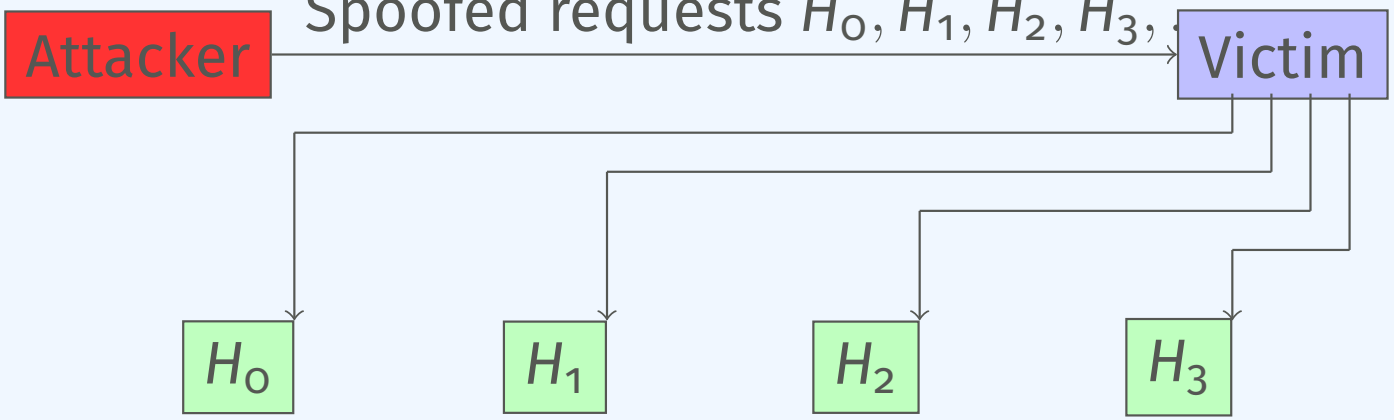
Configuration directory

Parameter	Explanation
type	see D4 Header slide
source	standard input
key	HMAC key
uuid	Identifier of the sensor
version	version of the sensor
destination	standard output
snaplen	length of data being read & written

Use-case: D4 analyzer to detect DDoS attacks in backscatter traffic

Observing SYN floods attacks in backscatter traffic

Attack description



Connections
H_0
H_1
H_2
H_3

What can be derived from backscatter traffic?

- External point of view on ongoing denial of service attacks
- Confirm if there is a DDoS attack
- Recover time line of attacked targets
- Confirm which services are a target (DNS, webserver, . . .)
- Infrastructure changes or updates
- Assess the state of an infrastructure under denial of service attack
  - Detect failure/addition of intermediate network equipments, firewalls, proxy servers etc
  - Detect DDoS mitigation devices or services
- Create probabilistic models of denial of service attacks

Confirm if there is/was a DDoS attack

Problem

- Distinguish between compromised infrastructure and backscatter
- Look at TCP flags → filter out single SYN flags
- Focus on ACK, SYN/ACK, ...
- Do not limit to SYN/ACK or ACK → ECE (ECN Echo)[5]

```
tshark -n -r capture-20170916110006.cap.gz -T
   fields -e frame.time_epoch -e ip.src -e tcp.
   flags
1505552542.807286000 x.45.177.71 0x00000010
1505552547.514922000 x.45.177.71 0x00000010
```

[5] https://tools.ietf.org/html/rfc3168

Passive Identification of Backscatter (WiP)

```
./pibs -b -r pcap_file.cap
```

Early version is available of PIBS[6] with a focus on TCP traffic.

Options	Explanations
-r	read pcap file
-b	display IPs under DDoS on standard output

Dependencies
libwiretap-dev
libhiredis-dev
libwsutil-dev

[6] https://github.com/D4-project/analyzer-d4-pibs