# Can generative AI help us better assess security solutions?

Pierre-François Gimenez
Inria researcher

JIRC
November 8th, 2024

# Who am I?

## Past positions

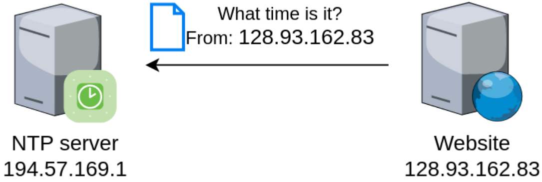
- PhD on machine learning at IRIT, Toulouse, until 2018
- Post-doc on AI/security at LAAS-CNRS, Toulouse, until 2020
- Assistant professor on AI/security at CentraleSupélec, Rennes, until 2024
- 3-month research stay at CISPA, Germany, in 2022
- Researcher on AI/security at Inria, Rennes, from 2024

## Interests

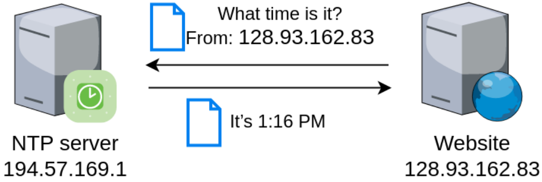
- Network intrusion detection
- Interpretable models learning
- Security data generation

# Simple denial of service attack

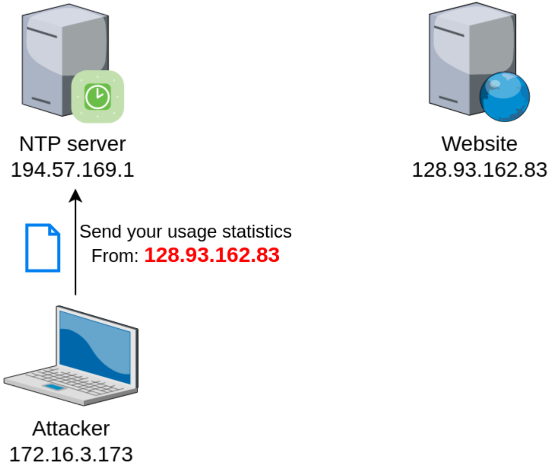
# Simple denial of service attack



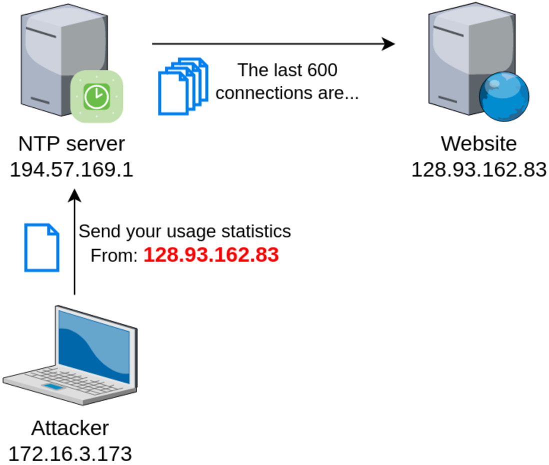
# Simple denial of service attack



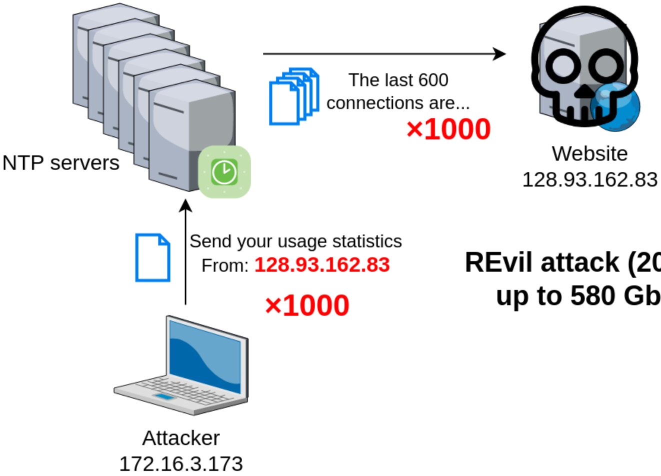
# Simple denial of service attack



# Simple denial of service attack



# Simple denial of service attack

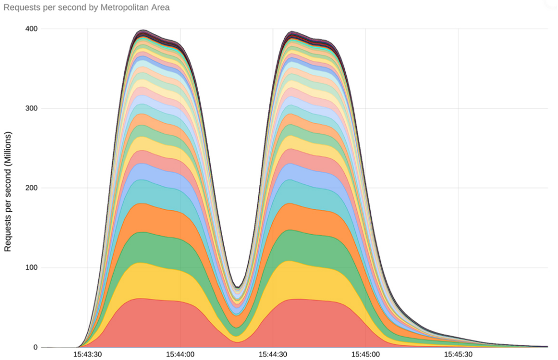



**REvil attack (2020):**
**up to 580 Gb/s**

# Introduction

## Systems are under attack

- Many untargeted, opportunistic attacks like password bruteforce
- Some targeted attacks with a huge power (e.g., DDoS attacks)
- Some very sophisticated attacks months or years in the making (SolarWinds, Stuxnet, TV5 Monde hack)




DDoS attacks against Google Cloud with 400 millions requests per second!

# Information system security

## Information system security

- Prevent the attack, detect it, and react
- Detection with **IDS**: *Intrusion Detection System*

## Detection relies on observation

- System : OS and applications logs
- Network : network communications

## Constraints

- Partial and heterogeneous observations
- Adversarial context: the attacker hides!

```
2024-05-06T23:24:16.806598+02:00
stellar-sheep sshd[16039]: Failed
password for pfg from 192.168.1.36
port 48650 ssh2
```


"ts": 1591367999.305988,
"id.orig_h": "192.168.4.76",
"id.resp_h": "192.168.4.1",
"id.resp_p": 53, "proto": "udp",
"service": "dns", "duration":
0.066851, "orig_bytes":
62, "resp_bytes": 141,
"conn_state": "SF", "orig_pkts":
2, "orig_ip_bytes": 118,
"resp_pkts": 2, "resp_ip_bytes":
197

# Two categories of detectors

Signature-based detection

Anomaly detection

**Date :** 2024-04-25 10:24:52+02:00
**IP source :** 194.57.169.1
**IP destination :** 128.93.162.83

**Signature :** alert udp any any -> any 123 (content:"|00 02 2A|"; offset:1; depth:3; byte_test:1,!&,128,0; byte_test:1,&,4,0; byte_test:1,&,2,0; byte_test:1,&,1,0; threshold: type both, track by_dst,count 2, seconds 60);

**Tentative d'attaque via NTP !**

**Date :** 2024-04-25 10:24:52+02:00
**IP source :** 194.57.169.1
**IP destination :** 128.93.162.83

**Score d'anomalie : 7,6**

Signatures database

+ quick, clear
– regular updates, only documented attacks

Normal behavior model

+ can detect undocumented attacks
– false positives, **no alert description**

# Two categories of detectors

**Signature-based detection**

Date : 2024-04-25 10:24:52+02:00
IP source : 194.57.169.1
IP destination : 128.93.162.83

**Signature :** alert udp any any -> any 123 (content:"|00 02 2A|"; offset:1; depth:3; byte_test:1,!&,128,0; byte_test:1,&,4,0; byte_test:1,&,2,0; byte_test:1,&,1,0; threshold: type both, track by_dst,count 2, seconds 60);

**Tentative d'attaque via NTP !**

Signatures database

- + quick, clear
- – regular updates, only documented attacks

**Anomaly detection**

**Date :** 2024-04-25 10:24:52+02:00
**IP source :** 194.57.169.1
**IP destination :** 128.93.162.83

**Score d'anomalie : 7,6**

Normal behavior model

- + can detect undocumented attacks
- – false positives, **no alert description**

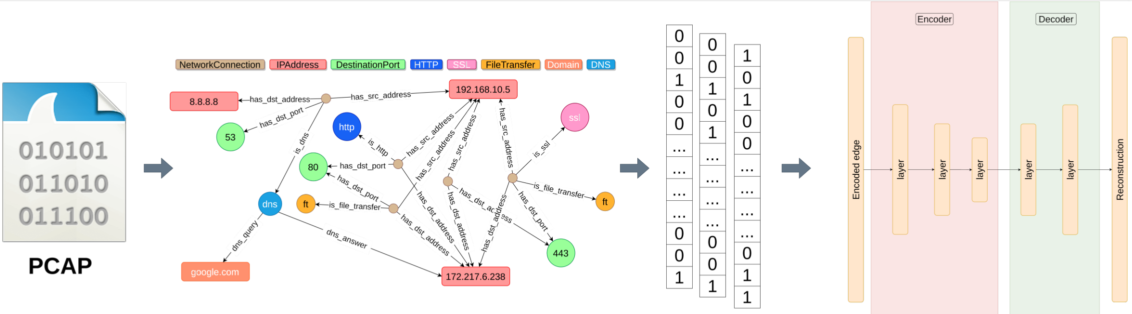
# Outline

# Intrusion detection

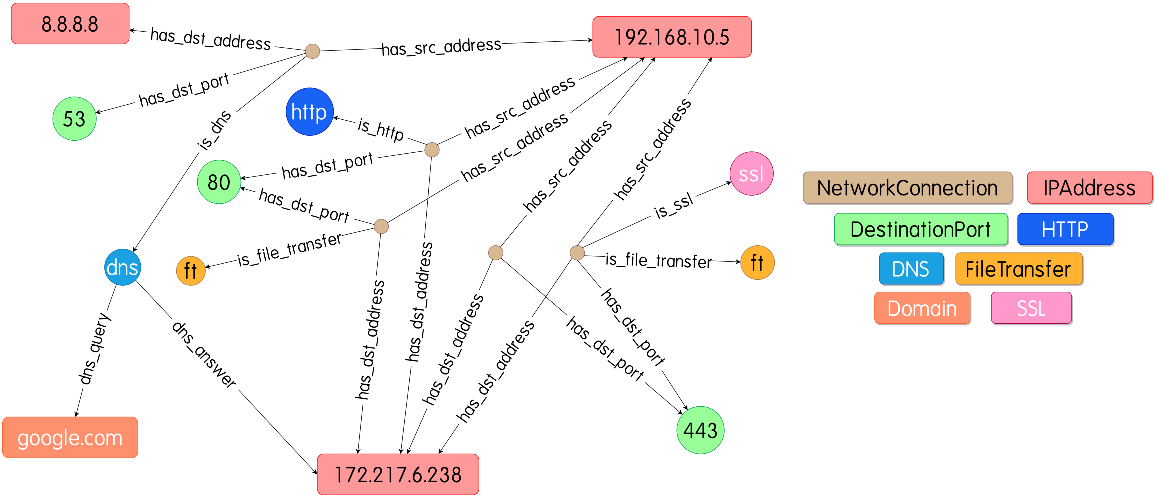
# Overview of our approach Sec2graph

## Structure of our approach

- Probes capture the network data
- These data are merged into a graph structure
- The graph is transformed into a format usable with a deep learning model
- The model affects an anomaly score to each data point



# Security objects graph example



# Security objects graph

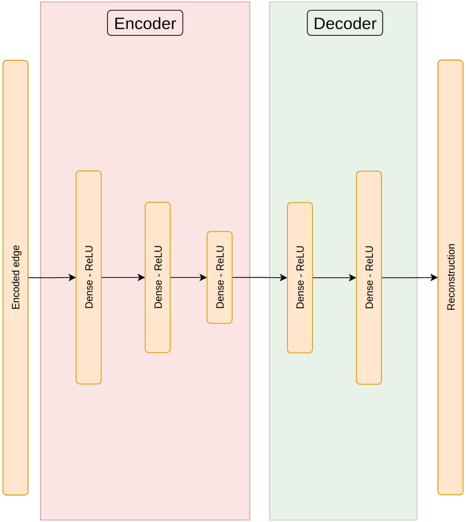
## Nodes

- Each node type corresponds to a "security object":
  - protocols: DNS, SSH, DCERPC, SNMP, FTP, DHCP, HTTP, SMTP
  - network data: port, MAC address, IP address, network connection, URI, domain
  - and others
- Nodes contain a set of attributes related to these objects

## Edges

- Edges are typed and oriented
- They do not contain attributes
- An edge between two nodes means that these two nodes are found within the same event

# Anomaly detection: Autoencoder (AE)




### Autoencoder

An autoencoder is a deep learning architecture with a bowtie shape

### Learning

Minimisation of the reconstruction error between the input vector and its reconstructed version

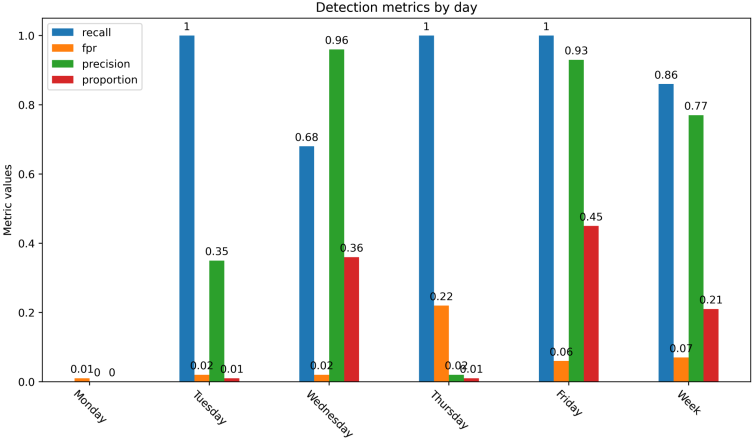
### Detection

Raise an alert when the reconstruction error is above a threshold

# Performances on CIC-IDS2017

## Performances

Recall is mostly good but we have a very high false positive (22%!) on Thursday



# Alert explaination

# How to explain the predictions?

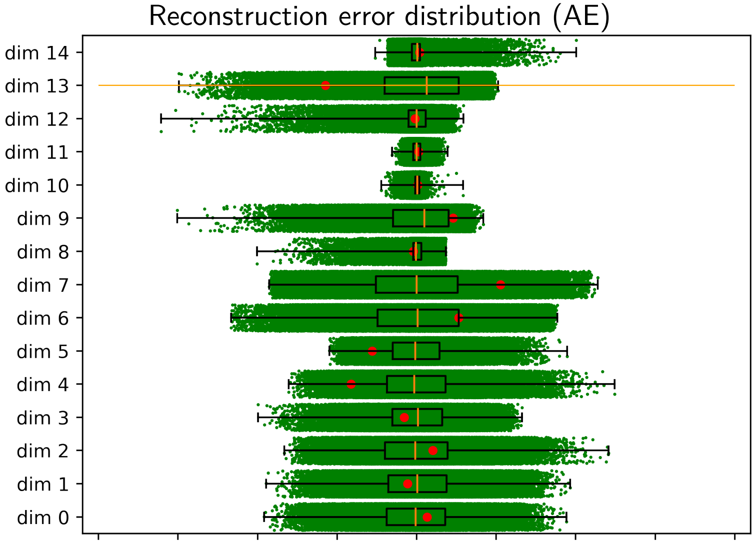
## The issue

- Explanations could help us understand the false positives
- There exists a lot of explanation techniques... (LIME, salient maps, counterfactual explanation...)
- ...but little work on explanations for unsupervised learning!

## First, naive approach

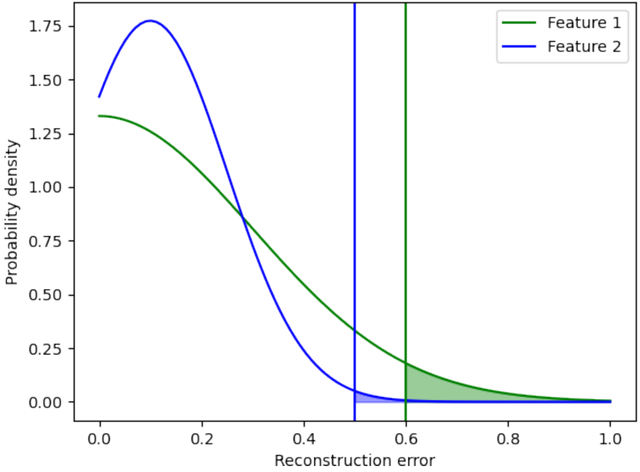
- We can compute the contribution of each feature to the global reconstruction error
- However, we found out this idea does not produce satisfactory explanations:
  - Some features are always difficult to reconstruct because of their high variance
  - Some features are always very faithfully reconstructed, and even a small reconstruction error may reveal an anomaly

# What it looks like



# Limitations

Comparison of the reconstruction errors of two dimensions




## Key Idea

The highest reconstruction error is not always an indication of the most abnormal dimension.

## Our approach

This area is called the p-value:

$$p_i = \frac{\#\{r_i \geq e_i\}}{\#\{r_i\}}$$

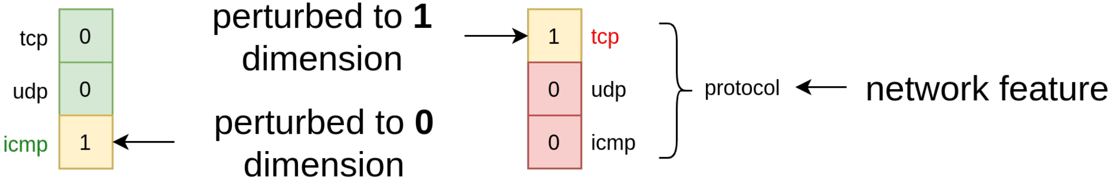
# Experimental protocol

## Protocol

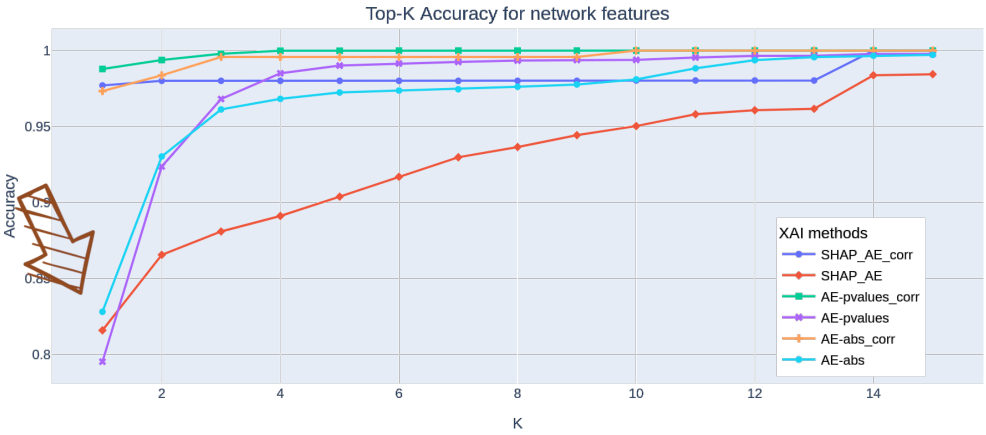
- Inject noise in a known network characteristic of vectors
- Assess ability of XAI methods to find the noisy network characteristic

Experiment with AE-abs (intuitive method), SHAP_AE (state of the art), AE-pvalues (our method)

## Example of noise insertion in the protocol characteristic



# Benchmark results




## Top-K accuracy

Proportion of samples for which the right explanation is among the Top-K explanations. But sometimes several explanations are correct. . .

# Several correct explanations

$$1 + 1 = 0$$

**Where is the error?**

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

## Where is the error?

We can all agree there is an error. But where do you think it is?

- $0$ should be $2$
- $+$ should be $-$
- $1$ should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

### Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

### Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

## Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

### Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

$$1 + 1 = 0$$

### Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Several correct explanations

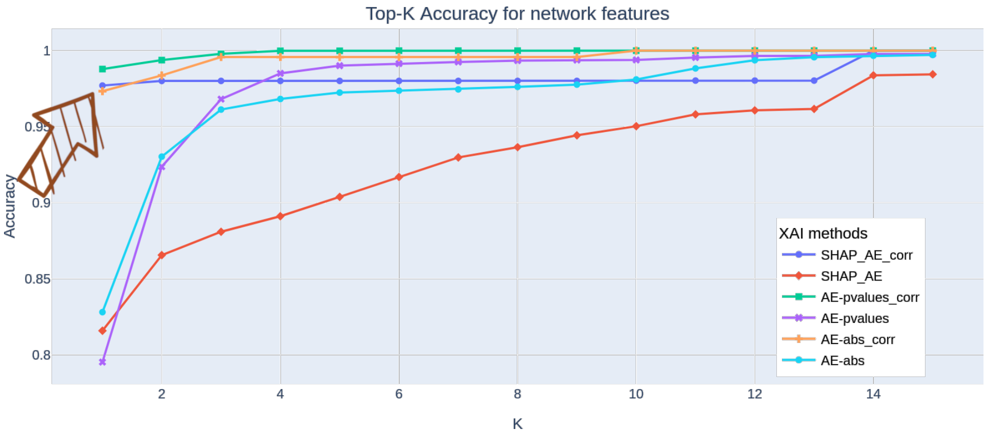
$$1 + 1 = 0$$

### Where is the error?

We can all agree there is an error. But where do you think it is?

- 0 should be 2
- $+$ should be $-$
- 1 should be $-1$
- $=$ should be $>$
- "(mod 2)" is missing
- "is false" is missing

# Benchmark results

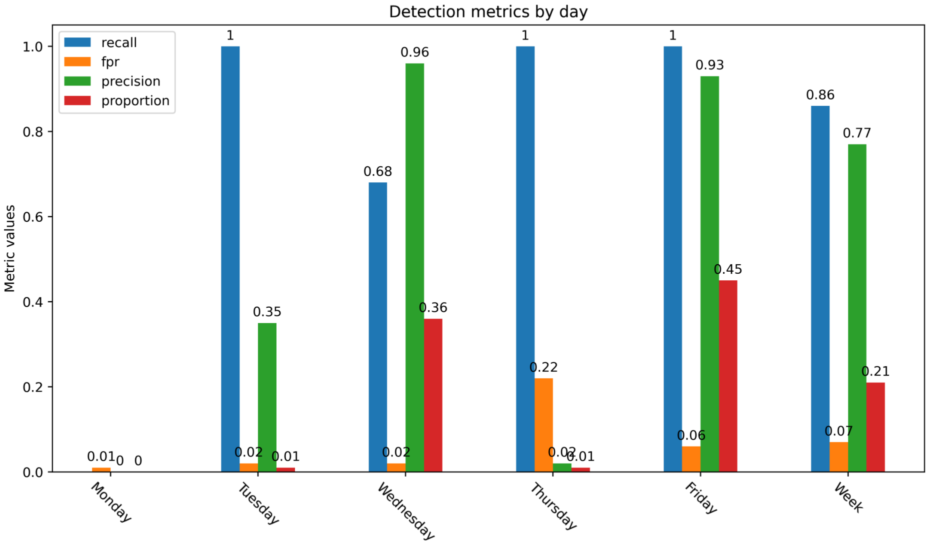



## A more realistic evaluation

Evaluation modification: accepting correlated features as correct explanations

# Data quality in security

# Remember that?...



# What is the issue with CIC-IDS2017?

## Not only one. . .

- Labeling issue: CIC-IDS2017 has a scan attack on Thursday that is not corrected labeled. About 70,000 flows of scan are labeled as "benign"!
- Duplication issue: probably due to a badly configured probe, on average 500,000 packets are duplicated per day. It caused the CSV files to contain bad data
- Shortcut learning possible: the tools use their default user agent
- And a few minors issues

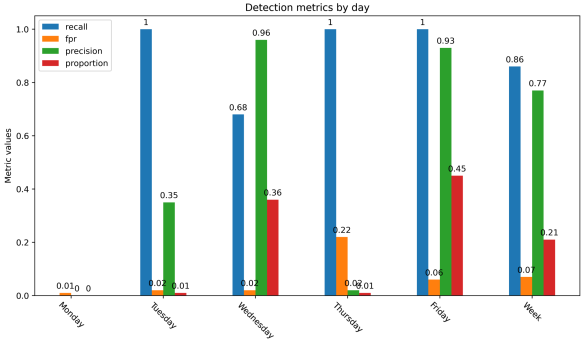
Corrected CIC-IDS2017 files: `https://gitlab.inria.fr/mlanvin/crisis2022`

## Why wasn't it found before?

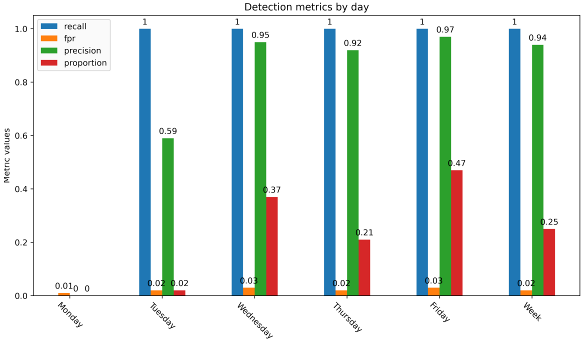
Turns out that the missing attack has duplicated packets, so its csv files didn't look like the other scan attacks. Consequence: supervised methods miss this unlabeled attack

These results make us confident in the usefulness of our explanation method

# Updated results on CIC-IDS2017




Before CIC-IDS2017 correction




After CIC-IDS2017 correction

# Flawed datasets

## Public dataset

- Most IDS research relies on public dataset
- It allows for reproducible results and comparison between methods
- A few datasets are popular: NSL-KDD, CIC-IDS-2017/2018, and a few others

## Criticisms

We are not the only ones finding issues in datasets

- NSL-KDD is still used but obsolete
- 4 articles have been published on issues on CIC-IDS-2017 alone
- Other datasets are also criticized

Common issues: unrealistic testbed, duration too low, badly configured tool and probe

# Alternatives

## Real data

- Difficult to obtain/share due to confidentiality and privacy reasons
- Typically not labeled

## Our own testbed

- Ongoing work at PIRAT
- Based on the SOCBED framework
- Slow: we need one month to generate one month of data

## Data generation with AI

- Could be much faster than testbed
- Is AI mature enough? How to explain the generation process and to evaluate the data?

My research project: **use AI to generate data**

# Network data generation

# **FosR**: Forger of security Recordings

## Goals

- Generation of network (pcap files) and system data (logs)
- Consistency between network and system
- In-depth data quality evaluation
- Minimal expert's input
- Explainable models

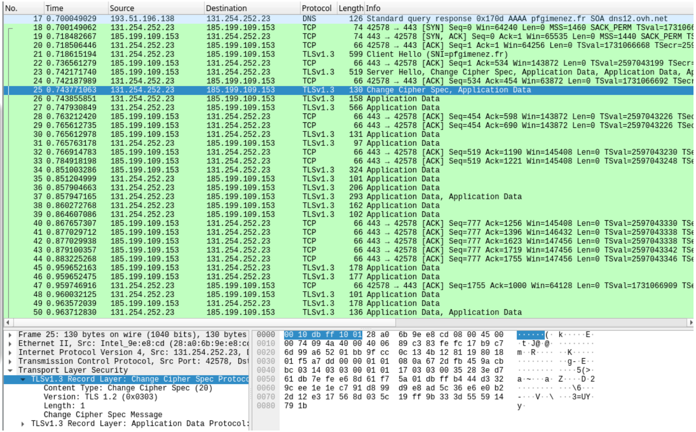
## Ongoing work: pipeline prototype

- We focus on benign network data
- Input data: pcap file
- Output data: a pcap file statistically similar to the input data

# Network data example

## Network data

- Raw data consist of packets, regrouped in conversation
- Cybersecurity analysis typically rely on network flow records that describe conversations statistically




```
ts,proto,src_ip,dst_ip,dst_port,fwd_packets,bwd_packets,fwd_bytes,bwd_bytes
1730800143,TCP,131.254.252.23,216.58.213.78,443,33,41,5988,1950
```

# Generation evaluation

## Difficulties

- No standard metrics
- Evaluation is often partial

## Proposition

A set of metrics evaluating:

| | | |
|---|---|---|
| Realism | : | do the generated data belong to the actual distribution? |
| Diversity | : | can we generate the whole variety of behavior present in the distribution? |
| Novelty | : | can we generate data not present in the train set? |
| Conformity | : | are the generated data compliant with technical specifications? |

We do not evaluate privacy for the moment: we assume the training data do not contain any personal information

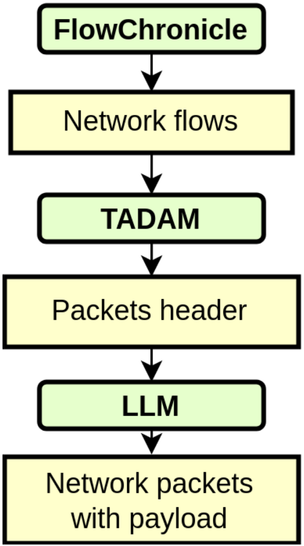
# Network generation pipeline

## SecGen

- A collaboration with researchers from CISPA (Germany)
- Goal: complete a network generation pipeline
- Intermediary step: network flows

## Two joint works

- FlowChronicle: a network flow generator
- TADAM: a probabilistic timed automata learner for packet header generation



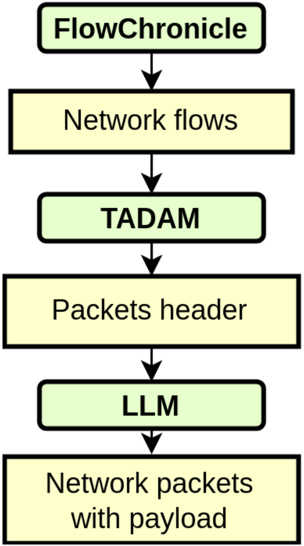
# Network generation pipeline

## SecGen

- A collaboration with researchers from CISPA (Germany)
- Goal: complete a network generation pipeline
- Intermediary step: network flows

## Two joint works

- FlowChronicle: a network flow generator
- TADAM: a probabilistic timed automata learner for packet header generation



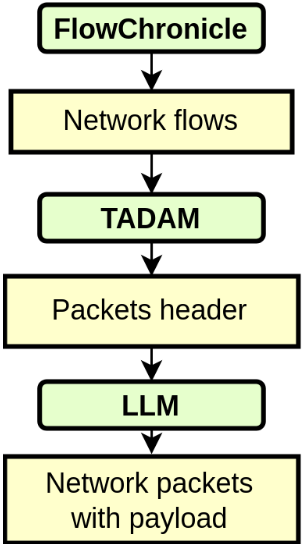
# Network generation pipeline

## SecGen

- A collaboration with researchers from CISPA (Germany)
- Goal: complete a network generation pipeline
- Intermediary step: network flows

## Two joint works

- FlowChronicle: a network flow generator
- TADAM: a probabilistic timed automata learner for packet header generation



# FlowChronicle

## Pattern language

- Hybrid approach: pattern detection and statistical modeling
- Pattern detection: find temporal patterns of flows
  - DNS query then HTTP(S)
  - IMAP request then HTTP(S)
- The values that are not fixed are modelized with a Bayesian network
- These patterns are self-explanatory:
  - they can be verified by an expert
  - they can also be added manually
- This work has just been accepted for publication

# FlowChronicle

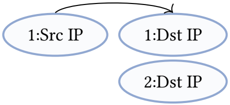
## Model — Pattern and Bayesian Network:

$\epsilon$ :

$$\begin{bmatrix} \beta & \beta & \beta \end{bmatrix}$$


p :

$$\begin{bmatrix} \beta_A & \beta & 993 \\ A & \beta & 80 \end{bmatrix}$$




q :

$$\begin{bmatrix} \beta_A & 8.8.8.8 & 52 \\ A & \beta_B & 443 \\ B & \beta & 3306 \end{bmatrix}$$

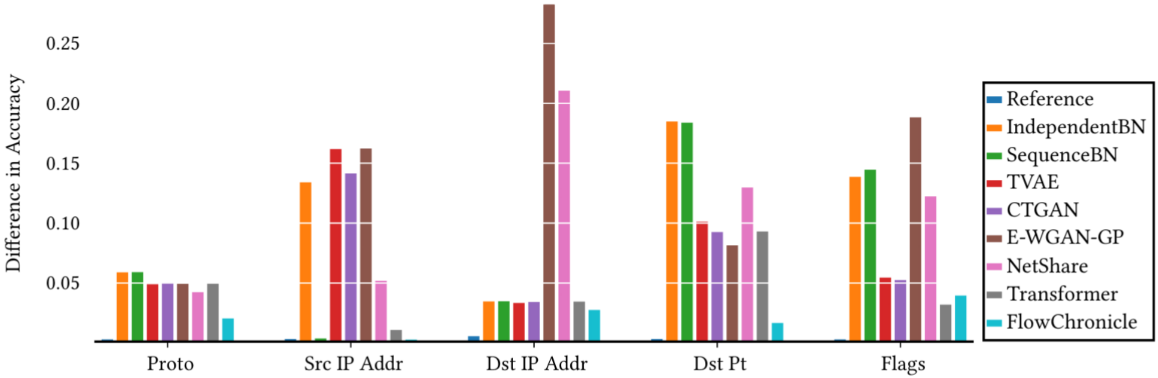

## Data and Pattern Windows:

| Time | Src IP | Dst IP | Port |
|---|---|---|---|
| 12 | 134.96.235.78 | 142.251.36.5 | 993 |
| 56 | 134.96.235.129 | 8.8.8.8 | 52 |
| 89 | 134.96.235.78 | 212.21.165.114 | 80 |
| 113 | 134.96.235.129 | 198.95.26.96 | 443 |
| 145 | 198.95.26.96 | 198.95.28.30 | 3306 |
| 156 | 134.96.235.78 | 134.96.234.5 | 21 |
| 178 | 134.96.235.36 | 185.15.59.224 | 993 |
| 206 | 134.96.235.36 | 128.93.162.83 | 80 |

# FlowChronicle: non-temporal generation quality

| | **Density** | **CMD** | **PCD** | **EMD** | **JSD** | **Coverage** | **DKC** | **MD** | **Rank** |
|---|---|---|---|---|---|---|---|---|---|
| | *Real.* ↑ | *Real.* ↓ | *Real.* ↓ | *Real./Div.* ↓ | *Real./Div.* ↓ | *Div.* ↑ | *Comp.* ↓ | *Nov.* = | *Average Ranking* |
| **Reference** | (0.69) | (0.06) | (1.38) | (0.00) | (0.15) | (0.59) | (0.00) | (6.71) | - |
| **IndependentBN** | 7 (0.24) | 5 (0.22) | 6 (2.74) | 8 (0.11) | 4 (0.27) | 4 (0.38) | 4 (0.05) | 4 (5.47) | 5.25 |
| **SequenceBN** | 6 (0.30) | 2 (0.13) | 5 (2.18) | 7 (0.08) | 3 (0.21) | 3 (0.44) | 2 (0.02) | 3 (5.51) | 3.875 |
| **TVAE** | 3 (0.49) | 4 (0.18) | 3 (1.84) | 2 (0.01) | 5 (0.30) | 5 (0.33) | 6 (0.07) | 5 (5.17) | 4.125 |
| **CTGAN** | 2 (0.56) | 3 (0.15) | 2 (1.60) | 3 (0.01) | 2 (0.15) | 2 (0.51) | 8 (0.11) | 2 (5.70) | 3.0 |
| **E-WGAN-GP** | 8 (0.02) | 7 (0.34) | 8 (3.63) | 5 (0.02) | 7 (0.38) | 8 (0.02) | 7 (0.07) | 6 (4.66) | 7.0 |
| **NetShare** | 5 (0.32) | 6 (0.28) | **1 (1.47)** | 6 (0.03) | 6 (0.36) | 6 (0.22) | 5 (0.05) | 7 (3.82) | 5.25 |
| **Transformer** | **1 (0.62)** | 8 (0.78) | 7 (3.62) | **1 (0.00)** | 8 (0.55) | 7 (0.03) | 3 (0.05) | 8 (3.75) | 5.375 |
| **FlowChronicle** | 4 (0.41) | **1 (0.03)** | 4 (2.06) | 4 (0.02) | **1 (0.10)** | **1 (0.59)** | **1 (0.02)** | **1 (5.87)** | **2.125** |

# FlowChronicle: temporal generation quality



# Data generated with FlowChronicle

## Output of FlowChronicle

- FlowChronicle outputs network flow records, e.g:

```
ts,proto,src_ip,dst_ip,dst_port,fwd_packets,bwd_packets,fwd_bytes,bwd_bytes
1730800143,TCP,131.254.252.23,216.58.213.78,443,33,41,5988,1950
```

- But in the end, we want to generate packets!

## Next intermediary step

- Before generating complete packets, we propose to first generate an intermediate representation
- More precisely, we generate for each packet a tuple with:
  - the direction (forward or backward)
  - the TCP flags
  - the size of the payload
  - the time since the last packet (i.e., the inter-arrival time)

# TADAM

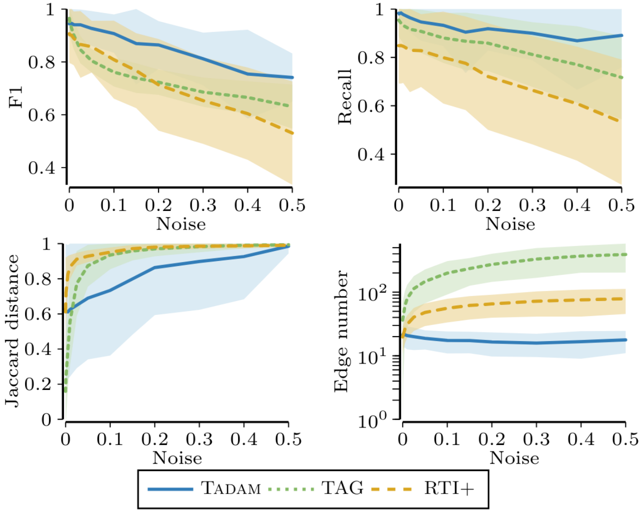
## Learning

- Network protocols typically rely on finite state automata
- We propose to learn probabilistic timed automata to capture packet header sequences
- Existing automata learners from observations cannot handle noisy data
- We propose TADAM: a robust timed automata learner
- Two main contributions:
  - A compression-based score to avoid overfitting
  - An explicit modelization of the noise

## Experimental results

- TADAM is far more robust to noise
- TADAM learns smaller models
- TADAM has better performance on real-world classification and anomaly detection tasks

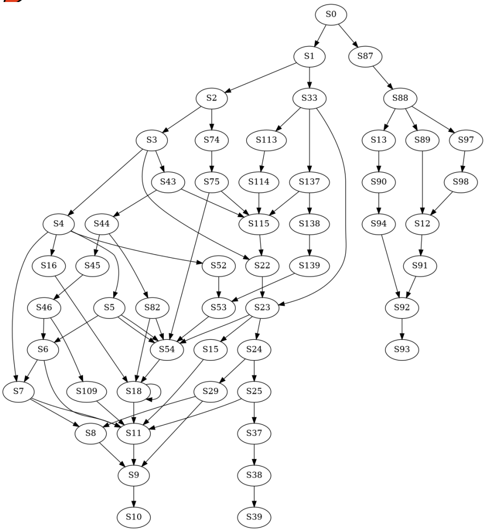
# TADAM: experiments




| Learner | AU-ROC | TPR | FPR | F1 |
|---|---|---|---|---|
| TADAM | **0.982** | 0.998 | **0.025** | **0.705** |
| TAG | 0.891 | **1** | 0.142 | 0.298 |
| RTI+ | 0.790 | **1** | 0.292 | 0.171 |
| HMM | 0.608 | 0.640 | 0.085 | 0.288 |

Table 3: *Anomaly detection performance on HDFS_v1 dataset.* We report the TPR, FPR and F1-score for the threshold maximizing TPR-FPR.

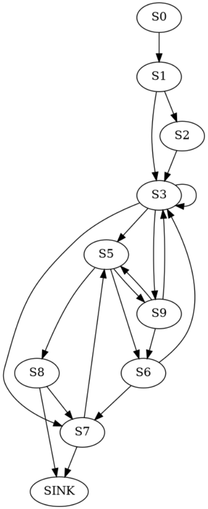
# Example: Kerberos protocol




TAG

## And for network protocols?

- We limit the observations to some data: TCP flags, direction, size and inter-arrival time
- In particular, we do not look at the payload, so no perspective on the semantics of the message
- In practice, it's not easy to interpret them

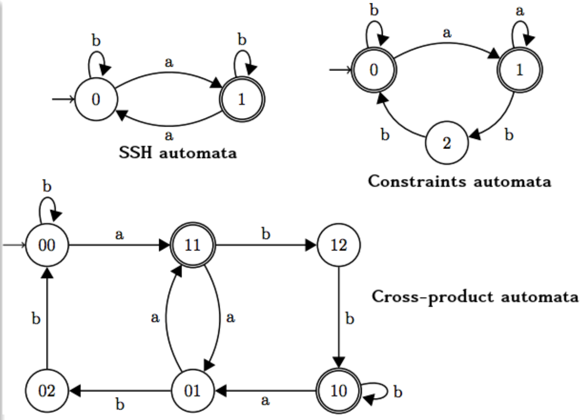



TADAM

# Header generation

## Generation from automata

- With a probabilistic automata, we can easily sample packet headers sequences
- But generation must be parameterized according to a network flow record!
- For example: total size = 5200 bytes, 5 forward packets, 8 backward packets
- This can be done easily by representing the constraints by an automaton and computing the intersection between the languages of the protocol automaton and the constraints automaton



# Payload generation

## From headers sequence to packets

- Intrusion detection system typically do not inspect the payload, so its realism is not our highest priority
- Most data can be filled automatically (ACK number, checksum, etc.)
- Some payloads are encrypted, so we can generate random data that are indistinguishable
- For plain-text payloads, we propose to replay them or to use LLMs
- We did some preliminary experiments with GPT-4 to generate realistic payloads, but conditioning the generation is not reliable and it is slow

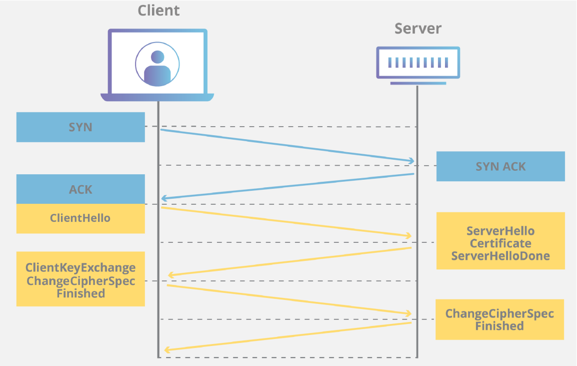
# Payload generation: example

## Example: TLS handshake generation

It must be:

- Consistent with the packet size generated by TADAM: the length of packet is highly influenced by the signature length of the cipher suite
- Consistent with the protocol:
  - The server name should be consistent in ClientHello and ServerHello
  - The cipher used in ServerHello should be available in ClientHello
  - Different OS use different ciphers

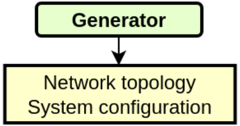
Not an easy task for LLMs!



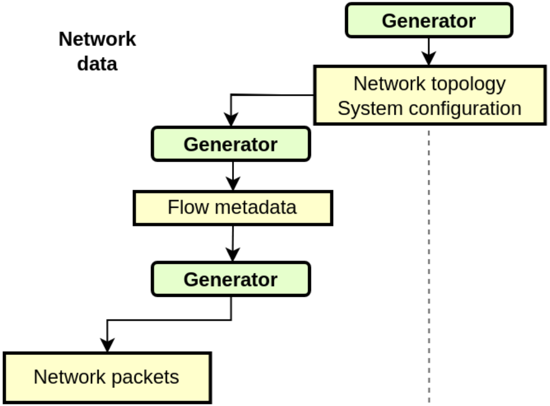

The four packets (in yellow) of a TLS handshake

# Future works: system data generation

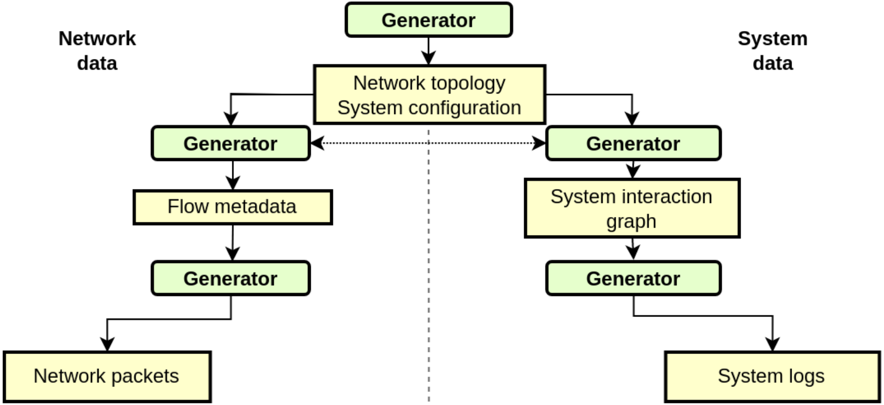
# The future of FosR



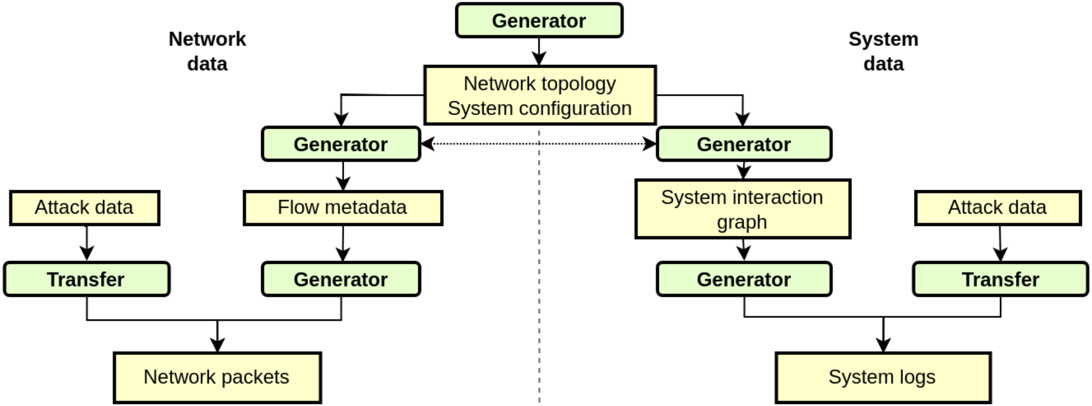
# The future of FosR



# The future of FosR



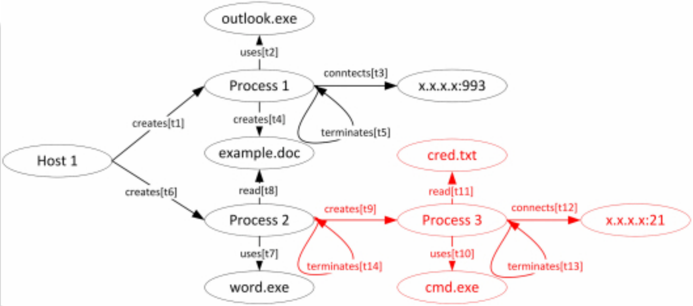
# The future of FosR



# System data generation

## System data generation

- Our next goal will be to generate system data, i.e., logs
- We propose to proceed with a two-step approach:
  - generate a provenance graph (graph of interactions between system entities and resources)
  - generate logs from such interactions




A provenance graph

# System data generation

## Provenance graph

- Provenance graph are knowledge graphs
- Nodes and edges are typed
- Each edge correspond to a system event and is associated to a timestamp

## Provenance graph generation

Possible approaches:

- pattern mining in graphs
- edge time series
- deep learning?
- probably more?

# Log generation

## Log parsing and generation

- Log parsing is notoriously complex
- Each application has its own semi-structured format, and it tends to change
- Log parsing and generation could be a perfect application for LLMs
- On top of well-known formats that could be directly generated, more obscure formats could be learned with few-shot learning or fine tuning

# Conclusion

# Conclusion

## The need of data

- Good quality data is of utmost importance for security system evaluation
- But public datasets have issues and errors
- One way to achieve good quality is through generative AI

## Current and future work

- "Classical" AI can yield better quality generation for low-dimension feature spaces, on top of being explainable
  $\rightarrow$ adapted to intermediate data structure generation
- LLMs may certainly be a key to generating actual data, i.e., packet payload and logs
  $\rightarrow$ conditioning their generation remains a challenge