Generative AI for assessing network intrusion detection systems

Pierre-François Gimenez
Researcher at Inria

Inria/AISI Workshop
November 6th, 2024

The issue of data in security

Why do we need data?

- For evaluating security measures, most notably detection
- For using machine learning in cybersecurity

Current state of datasets

- Public datasets are typically run in testbed with no real users
- They can suffer from mislabelling, network and attack configurations errors, etc.
- We cannot access private data due to confidentiality and privacy reasons

⇒ we cannot confidently evaluate anomaly-based detection because of the dubious quality and the lack of realistic users

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

Security data generation

Goals

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

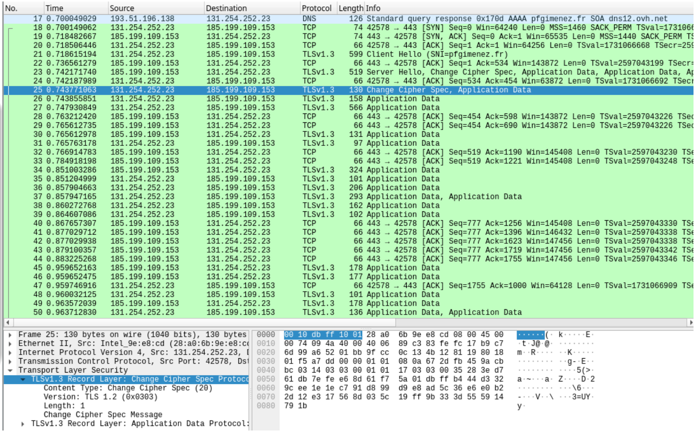
Ongoing work: pipeline prototype

- 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
```

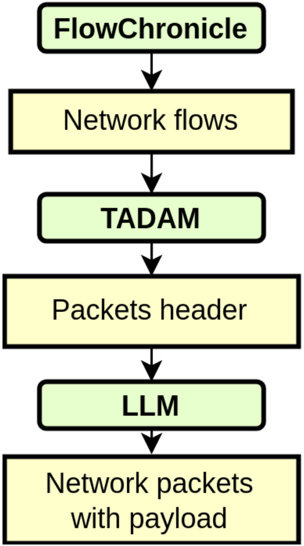
Approach

State of the part

- Several approaches have been tried to generate network flow records or pcap files: VAE, GAN, LLMs
- The results are not very good:
  - A significant portion of generated data do not comply with network protocols
  - Generated data do not reflect the diversity of the original data

Our approach: a three-step generation

- FlowChronicle (published): a network flow generator
- TADAM (under review): a packet header generator
- Starting work with LLMs: full packet generator



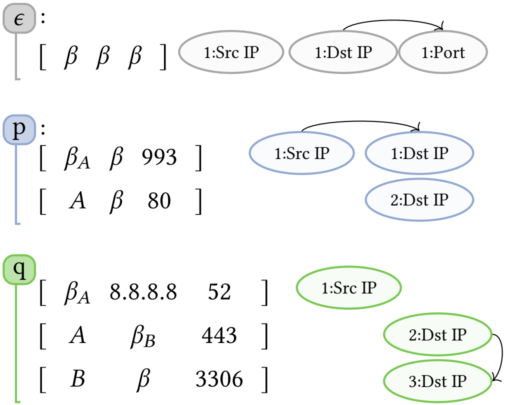
FlowChronicle

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)
- Some values are fixed in the patterns
- 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 was published recently

FlowChronicle

Model — Pattern and Bayesian Network:



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: 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**

TADAM

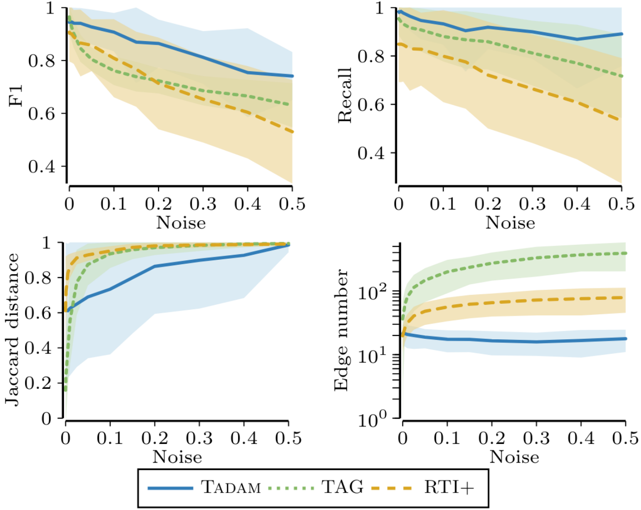
Learning

- 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.

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 protocol automaton and the constraints automaton

Payload generation

From headers sequence to packets

- 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

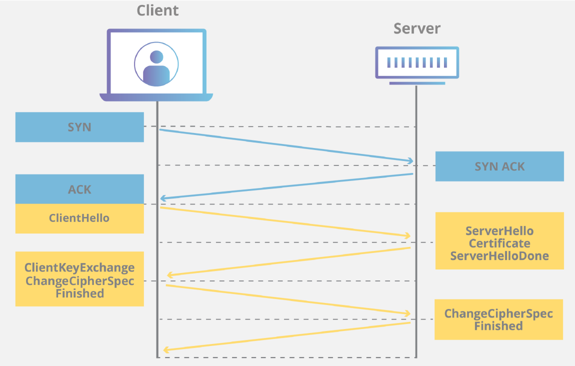
Payload generation: example

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

LLMs for 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

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

The need of data

- Good quality data is of utmost importance for security system evaluation
- One way to achieve such 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 is certainly a key to generating actual data, i.e., packet payload and logs
  $\rightarrow$ conditioning their generation remains a challenge