Intelligently categorising behaviours of IoT-like devices from NetMon data

Steve Moyle
University of Oxford

UNIVERSITY OF OXFORD

Outline

- Objectives
- Background
- Identifying IoT-like devices from NetMon data
- Intelligent Behaviour Categorisation
  - Sequitur: NetMon events →Sequence extraction
  - Inductive Logic Programming: Sequences → Behavioural Rules
- Future directions

Objectives

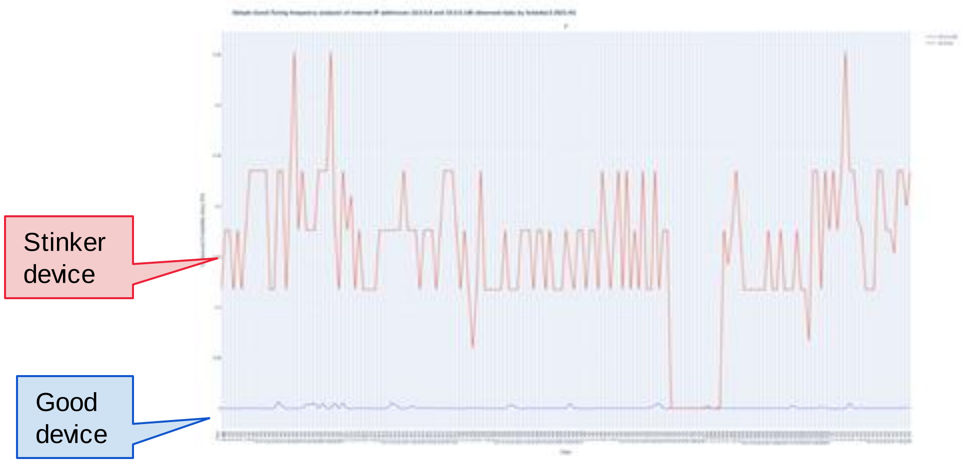
Can we identify IoT-like devices from NetMon data?

Can we categorise behaviours of IoT-like devices from NetMon data?

- Easy to defend
- Difficult to defend

Can we *auto-magically* build D3 expressions?

- D3 Static behaviour expressions*
- D3 Static behaviour inference*
- D3 Dynamic behaviour inference
  - Sequence mining from ML
  - Symbolic rule induction from ML



D3

Static behaviour expression	Static behaviour inference	Dynamic behaviour inference
Can we express static behavioural constraints	Can we infer static behaviours from monitoring	Can we infer dynamic behaviours from monitoring

Vision: ***Power intelligent gateways so they can pro-actively defend IoT devices***

*Not covered in this presentation

Background

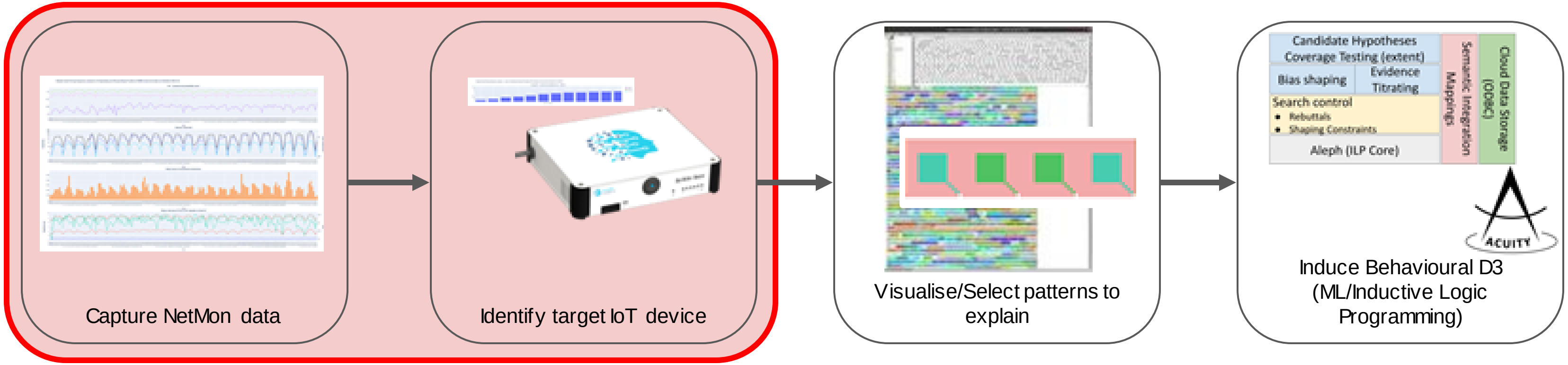
- Turing's *little* machine is **finite**, but permits an **infinite** (א) number of programs
  - This allows an infinite number of "good" programs (c.f. How many Even numbers?);
  - and an infinite number of "bad" programs (c.f. How many Odd numbers?)
  - Can we even agree on what are good and bad programs?
- Cyber defenders have a very difficult task!
  - Maybe they should focus on defending devices that are easier (those that have less volatile behaviours)

***Hypothesis I* – IoT devices should have limited dynamic behaviours (i.e. low volatility)**

***Hypothesis II* – NetMon data can be used to induce descriptions of dynamic device behaviours**

UNIVERSITY OF OXFORD

Identifying IoT-like devices from NetMon data



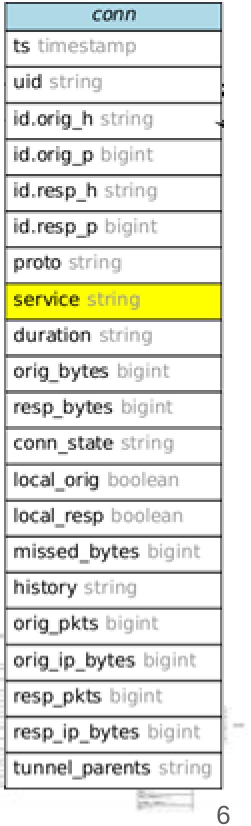
Capturing NetMon data

NetMon system

- Based on the IDS Zeek (formerly called Bro)
- Comprehensive event-relational meta-data extracted from PCAP-like captured network traffic
- *conn* table is the master table
  - Millisecond timestamps
  - Unique UID keys into *service* specific tables
- Many tables have common fields

ts	2021-01-04 14:59:39
uid	CNfInROV9fQ6nP4Ve
id.orig_h	192.168.16.39
id.orig_p	5353
id.resp_h	224.0.0.251
id.resp_p	5353
proto	udp
service	dns
...	...

NetMon device



Volatility analysis of NetMon data

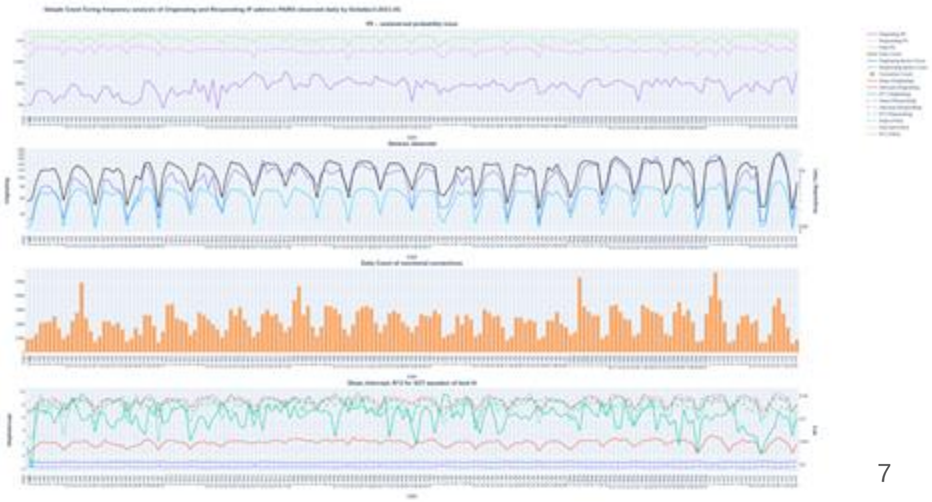
NetMon device

NetMon device deployed into a small business

- 6 month’s data acquired
- 28 to 185 active networked devices per day
- 63K to 562K logged connection events per day

Volatility measurements based on the Simple Good-Turing (SGT) frequency estimator

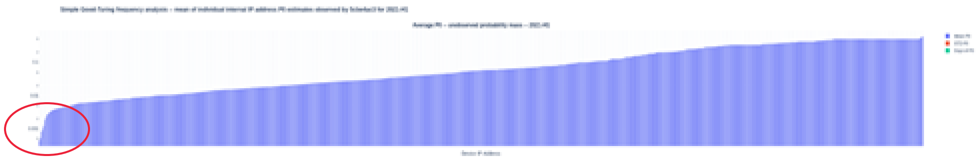
- SGT Volatility contains components separate from aggregate rates

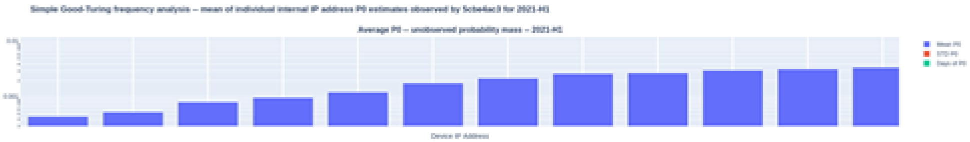


Identifying IoT-like devices from NetMon data

Can we identify an IoT-like device from *Volatility?*

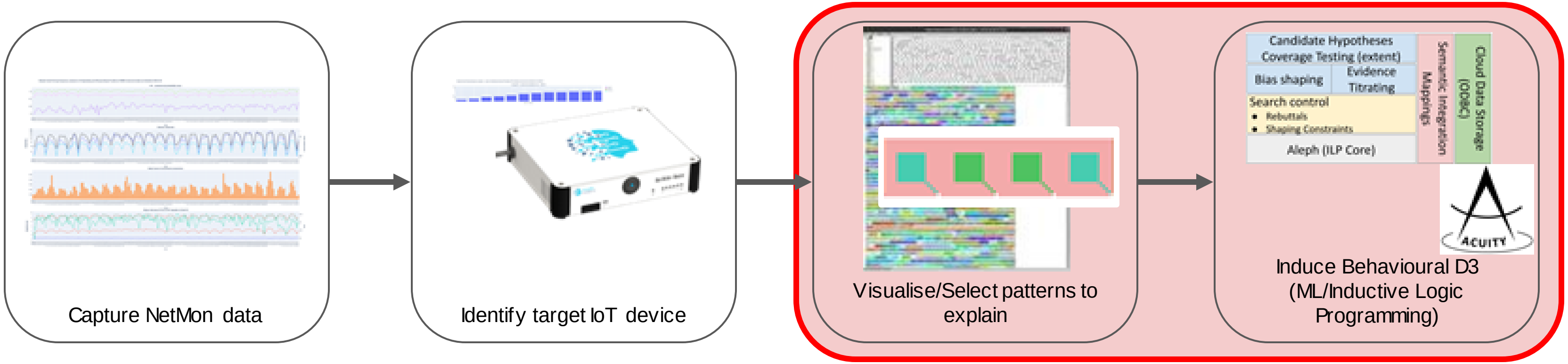
- Rank all 185 devices observed over 6 months by **mean** volatility (log scale)
- Gather information available on lowest ranked devices
- Best efforts identification of device type
- Definitely identified the one IoT device known to exist in the network – *the NetMon device itself*
- Strong support for having identified other devices (Routers, printers, …)





Low Volatility IOT device

Characterizing the behaviour of an IoT device from NetMon data



Extracting sequence patterns from NetMon data

- Use Zeek’s **conn**ection table to build **tokens** for each connection event
- Generate unique MD5-based 9 digit numeric **token_uid**
- (… some other raw information in a CSV file)
- (initially) ignore timestamps, and collect the sequence of **token_uid**s into a single **training** file -- with one token_uid per line
- Run ***Sequitur*** on the **training file**
- Parse Sequitur’s **hierarchical grammar**
- Visualise / explore the training file and the original training file in the context of the hierarchical grammar and the extracted repeating subsequences. (next slide)

- Token from conn fields
  <id.orig_h>_<id.resp_h>_<id.resp_p>_<proto>_<service>

"10.0.0.145_10.0.0.137_53_udp_dns"

- token_uid: 271311686

- CSV file (direct from AWS Athena SQL)

```
"Timestamp","token","token_uid","year","month","day","conn_uid"
"2021-01-01 00:00:03.135","10.0.0.145_169.254.169.254_80_tcp_-","350344446","2021","1","1","CCsBTv1Te0if5EF777"
"2021-01-01 00:00:03.230","10.0.0.145_10.0.0.137_53_udp_dns","271311686","2021","1","1","CJTneEzIgNIvQyjJ2"
   :
"2021-01-01 00:00:04.029","10.0.0.145_10.0.0.137_53_udp_dns","271311686","2021","1","1","CepU6O1S80ZPwXFQA7"
   :
```

- Sequitur input file (training file)

```
350344446
271311686
   :
271311686
   :
```

Sequitur: NetMon events →Sequence extraction

Visualisation

Source data

- NetMon from single day 2021-01-01; where IP address 10.0.0.145 is mentioned (a brAIn-box)
- 3232 events (forming tokens/token_uids)

Hierarchical grammar

- [<9 digit integer>] is the raw token_uid
- Integer without [] refers to sequitur's rule id
- 366 rules (and the start sequence rule 0)

File explorer style Hierarchical grammar navigator

- Folders represent rules
- Files represent tokens

Visual representation of training sequence of tokens

- Background rectangles represent rules Each rule is a unique colour NB: White indicates that the token is not part of any rule
- Inside squares represent tokens Each token is a unique colour

Source training CSV file

Grammar node detail Shows root - raw sequitur grammar

50th token

100th token

Black underline represents the first token of a new hour

Tool-tip displays more information on particular token/rule

2400th token

Inductive Logic Programming: Sequences → Behavioural Rules

- Inductive Logic Programming is a form of machine learning which uses logic to describe the concepts
- From a family of computer programs that follow the Specific (observations) → General (laws) idea
- ILP uses symbolic logic as a representation language
- It is the study of constructing plausible hypotheses in logic from specific observations
- Draws on results from
  - Logic Programming
  - Statistics
  - Design of algorithms

UNIVERSITY OF OXFORD

ILP: An illustration

(Deductive) LP

x is the grandfather of **y**
if
x is the father of some **z**
and
z is the parent of **y**

Henry is the father of **Jane**
Jane is the parent of **John**
Jane is the parent of **Rob**

Henry is the grandfather of **John**
Henry is the grandfather of **Rob**

(Inductive) LP

Examples

Henry is the grandfather of **John**
Henry is the grandfather of **Rob**

Background

Henry is the father of **Jane**
Jane is the parent of **John**
Jane is the parent of **Rob**

Hypothesis

x is the grandfather of **y** if
x is the father of some **z** and
z is the parent of **y**

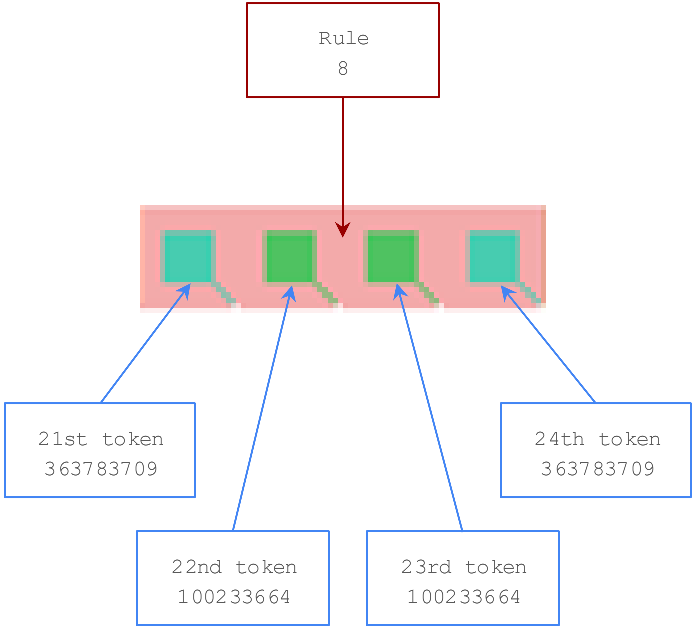
Feeding the ILP the Sequence data

Background “Knowledge”

- 366 Sequence ‘rules’
- Underlying network metadata
- Raw Source data - Recorded 2021-01-01; where IP address 10.0.0.145 is mentioned (a brAIn-box)
- 3232 events (forming tokens/token_uids)

Examples from the hierarchical sequence grammar to be explained

- Rule 8 occurs 18 times at event positions (0=midnight) [21,71,122,177,228,295,346,402,960,1118, 1274,1533,1835,2000,2153,2452,2743,3051]



ILPing Sequence 'Rule' 8 – 55 logical elements

```
Sequence A is interesting if:
   A has B events,
   A has 4 events,
   A has start uuid C,
   A starts at position D,
   event C precedes event E,
   event C has port service F,
   event C has port service dns,
   event C has service G,
   event C has service dns,
   event C has dest name H,
   event C has dest name name_not_found,
   event E precedes event I,
   event E has port service J,
   event E has port service ssh,
   event E has service K,
   event E has service -,
   event E has dest name L,
   event E has dest name ec2-3-209-99-
56.compute-1.amazonaws.com,
   event E has dest suffix M,
   event E has dest suffix amazonaws.com,
        :
```

```
       :
     <SNIP>
       :
   event P precedes event S,
   event P has port service Q,
   event P has port service https,
   event P has service K,
   event P has service -,
   event P has dest name R,
   event P has dest name s3-ap-southeast-
2-w.amazonaws.com,
   event P has dest suffix M,
   event P has dest suffix amazonaws.com.
```

Procedure

1) Specifically explain only example 8
2) Search through a space of logically more **general** alternatives for a more succinct rule that is "good" that does not unnecessarily cover other sequence rules

Understanding Generalisation of Sequence ‘Rule’ 8

The second event in the sequence is a SSH connection to a consistent AWS host.

How well does it fit the data?

The Extent of proposed hypothesis is that it covers (explains) Sequence Rule 8, but also sequence rules 193 and 318. On inspection, it seems that sequence rules 193 and 318 are spurious extensions of sequence rule 8 generated by `sequitur`.

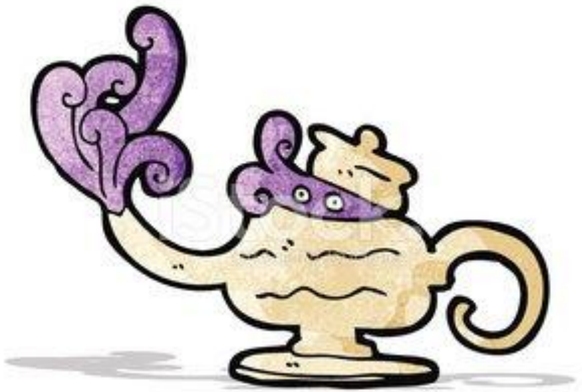
- Curious temporal pattern
  - The timestamps of all events that are covered by the hypothesis occur on average at 10.02 minutes past the hour (SD=0.005 minutes)
  - Maybe this device is ***beaconing*** using SSH

Hypothesis

```
Sequence A is interesting if:
   A has start uuid B and
   event B precedes event C and
   event C has port service ssh and
   event C has dest name ec2-3-209-99-
56.compute-1.amazonaws.com.
```

What happens next?

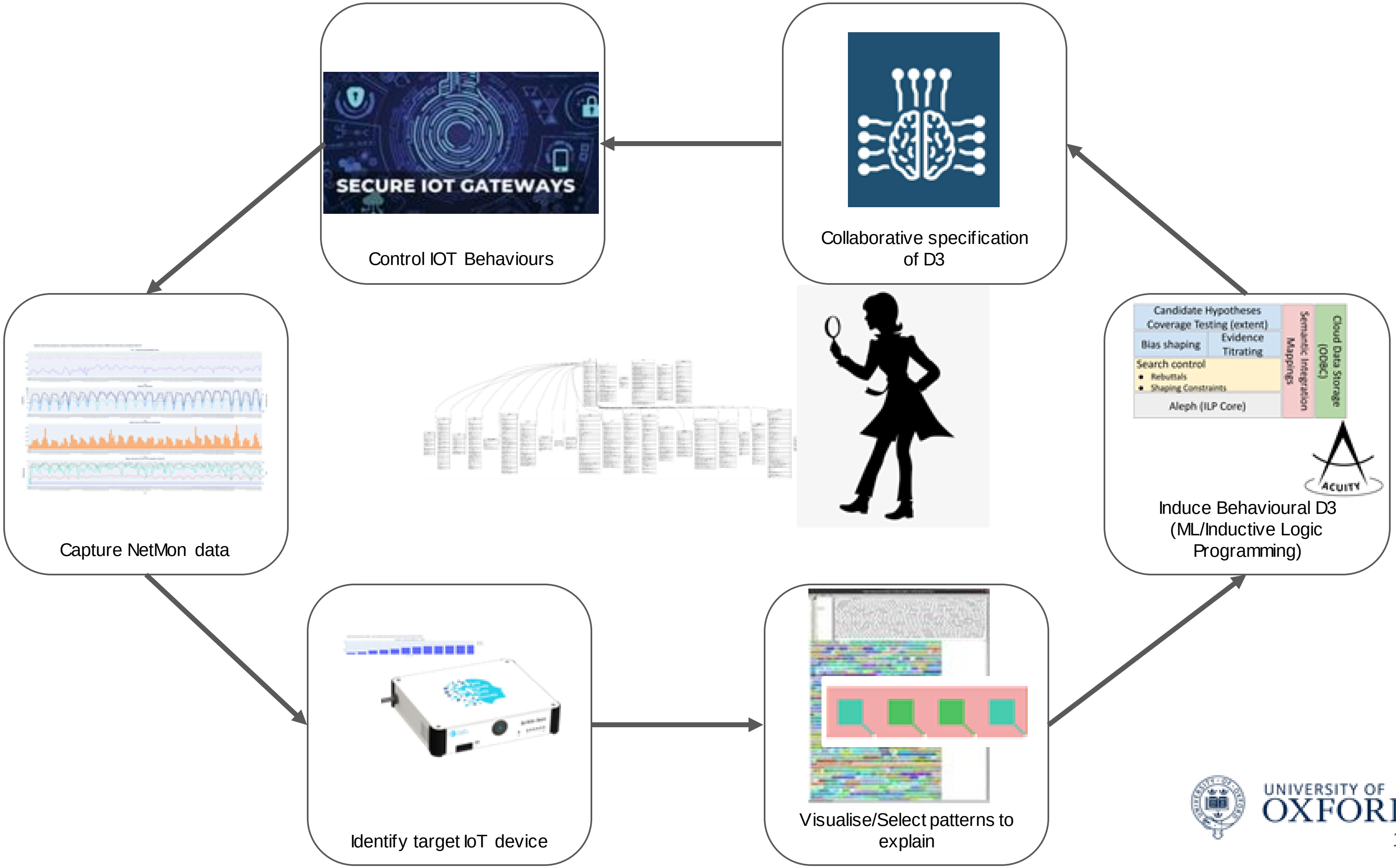
- Turn the focus to another unexplained sequence rule, to categorise other behaviours of the device
- Designate behaviours as expected (good) and unexpected (bad)
- Use the ILP generated rules as triggers for detecting unexpected behaviours
- Generate D3 statements that can be enforced by the IoT gateway



UNIVERSITY OF OXFORD

Future work

- Additional ILP background knowledge
  - Incorporate beaconing determination into potential hypotheses
  - Delve into the full NetMon relational data model
  - Add other Security Analytics data
  - Add Cyber Forensics capabilities
- Integrate the ILP with the Graphical prototype
  - Show explained sequences faded, whilst highlighting yet to be explained events
- Generation of D3 statements
- Automate the process further



UNIVERSITY OF OXFORD