

Cyber Security
Research Centre



Academic Centre of Excellence
In Cyber Security Research



in association with
National Cyber
Security Centre

Automating Threat Modelling

21st April 2022

Matthew Bradbury

Lecturer in Cyber Security

Introduction

Lecturer in Cyber Security at Lancaster University

Research focus:

- Cyber Security and Privacy
- Computing devices with limited resources (e.g., IoT)
- Context Privacy – How the actions a system takes reveals important information to an adversary





What is Academic Research?

Expectation: Write Code

```
float calculate_trust_value(edge_resource_t* edge, edge_capability_t* capability)
{
    // Get the stereotype that may inform the trust value
    edge_stereotype_t* s = NULL;
    public_key_item_t* item = keystore_find_addr(&edge->ep.ipaddr);
    if (item != NULL)
    {
        s = edge_stereotype_find(&item->cert.tags);
    }

    float trust = 0;
    float w_total = 0;
    float w, e;

    beta_dist_t temp;

    w = find_trust_weight(capability->name, TRUST_METRIC_TASK_SUBMISSION);
    beta_dist_combine(&edge->tm.task_submission, s ? &s->edge_tm.task_submission : NULL, &temp);
    e = beta_dist_expected(&temp);
    trust += w * e;
    w_total += w;

    w = find_trust_weight(capability->name, TRUST_METRIC_TASK_RESULT);
    beta_dist_combine(&edge->tm.task_result, s ? &s->edge_tm.task_result : NULL, &temp);
    e = beta_dist_expected(&temp);
    trust += w * e;
    w_total += w;

    w = find_trust_weight(capability->name, TRUST_METRIC_RESULT_QUALITY);
    e = beta_dist_expected(&capability->tm.result_quality);
    trust += w * e;
    w_total += w;
}
```

```
46 class Simulator:
47     def __init__(self, seed: int, agents: List[Agent], escls, duration: float, utility_targets: UtilityTargets, log_level: int):
48         # Initialise the PRNG and record the seed
49         self.seed = seed
50         self.rng = random.Random(self.seed)
51
52         self.agents = agents
53         for agent in self.agents:
54             agent.set_sim(self)
55
56         self.es = escls(self)
57
58         self.duration = duration
59         self.utility_targets = utility_targets
60
61         self.current_time = 0
62         self.queue = []
63
64         self.metrics = Metrics()
65
66         self.log_level = log_level
67
68     def add_event(self, event):
69         heapq.heappush(self.queue, event)
70
71     def run(self, max_start_delay: float):
72         # Add start event
73         for agent in self.agents:
74             self.add_event(AgentInit(self.rng.uniform(0, max_start_delay), agent))
75
76         while self.queue:
77             item = heapq.heappop(self.queue)
78
79             assert item.event_time >= self.current_time
80
81             # Has the simulation finished?
82             if item.event_time > self.duration:
83                 break
84
85             self.current_time = item.event_time
86
87             item.action(self)
```

Reality: Developing new theory

```

449 %
450 At this point, we wish to determine the role played by \TrustTracker in the
451 offloading problem. We now present our second main result.
452 \begin{theorem}[Necessity and Sufficiency of \TrustTracker]
453
454 Given a synchronous system  $G=(\text{RichNodes} \cup \text{ConstrainedNodes}, A)$ , it is
455 possible to solve the offloading problem if and only if each node  $c \in \text{ConstrainedNodes}$ 
456 is equipped with \TrustTracker, i.e., \OffloadingEngine and \TrustTracker are equivalent.
457
458 \end{theorem}
459 \begin{proof}
460 \begin{itemize}
461 \item \textbf{Sufficiency}: To prove that \TrustTracker is sufficient to solve
462 the offloading problem, we provide an algorithm (see \cref{alg:sufficient})
463 that uses \TrustTracker to construct \OffloadingEngine.
464 \end{itemize}
465
466 \begin{itemize}
467 \item \textbf{Necessity}: To prove the necessity of \TrustTracker to solve the
468 offloading problem, we develop an algorithm (see \cref{alg:necessary}) that
469 emulates \TrustTracker using the output of \OffloadingEngine. \qedhere
470 \end{itemize}
471 \end{proof}
472
473 The relation between \OffloadingEngine and \TrustTracker is shown in
474 \cref{fig:related-between-0-5}.
475
476 \begin{figure}[t]
477 \centering
478 \includegraphics[width=0.8\columnwidth]{Diagrams/OEandTT.pdf}
479 \caption{Architecture for sufficiency and necessity proofs.}
480 \label{fig:related-between-0-5}
481 \end{figure}
482
483 \subsubsection{Sufficiency of \TrustTracker}
484
485 \cref{alg:sufficient} shows how to transform the output of \TrustTracker (Line 6)
486 to obtain the output of \OffloadingEngine, the offloading engine, as represented
487 in \cref{fig:related-between-0-5}. Line 6 (of \cref{alg:sufficient}) represents
488 the input which \TrustTracker submits to \OffloadingEngine. At the start of every
489 synchronous round, \TrustTracker samples each resource-rich node. As it receives
490 messages from resource-rich nodes, it updates its list of bad nodes (\$BL\$) and the
491 list of unstable nodes (via the epoch vector) and outputs it to \OffloadingEngine.

```

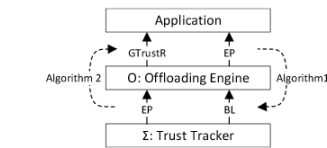


Figure 1: Architecture for sufficiency and necessity proofs.

The completeness property of the trust detector captures issues caused by untrusted nodes (i.e., bad and unstable nodes). It associates edge nodes with an epoch number to detect unstable nodes. On the other hand, the accuracy property stipulates that (eventually) good nodes are permanently trusted. We denote the trust tracker device by Σ . At this point, we wish to determine the role played by Σ in the offloading problem. We now present our second main result.

Theorem 2 (Necessity and Sufficiency of Σ). *Given a synchronous system $G = (V_R \cup V_C, A)$, it is possible to solve the offloading problem if and only if each node $c \in V_C$ is equipped with Σ , i.e., \mathcal{O} and Σ are equivalent.*

Proof.

- **Sufficiency:** To prove that Σ is sufficient to solve the offloading problem, we provide an algorithm (see Algorithm 1) that uses Σ to construct \mathcal{O} .
- **Necessity:** To prove the necessity of Σ to solve the offloading problem, we develop an algorithm (see Algorithm 2) that emulates Σ using the output of \mathcal{O} . \square

The relation between \mathcal{O} and Σ is shown in Figure 1.

A. Sufficiency of Σ

Algorithm 1 shows how to transform the output of Σ (Line 6) to obtain the output of \mathcal{O} , the offloading engine, as represented in Figure 1. Line 6 (of Algorithm 1) represents the input which Σ submits to \mathcal{O} . At the start of every synchronous round, Σ samples each resource-rich node. As it receives messages from resource-rich nodes, it updates its list of bad nodes (BL) and the list of unstable nodes (via the epoch vector) and outputs it to \mathcal{O} . We assume a threshold σ to identify unstable nodes. When \mathcal{O} is ready to select an edge node for offloading, Σ outputs the list $ChosenP$ of trusted nodes (those that are good

Algorithm 1 Sufficiency: Σ sufficient to solve \mathcal{O}

▷ Output of Σ to obtain offloading engine

```

1: function INIT
2:   START(off,  $\delta$ )
3:    $Ep \leftarrow V_R \times \{0\}$ 
4:   ▷ Trusted nodes in one round
5:    $trustR \leftarrow V_R$ 
6:    $GtrustR \leftarrow V_R$ 
7:   event EVALUATED( $BL, E$ ) → ▷  $BL$ : bad list,  $E$ : Epoch
8:    $Ep, trustR \leftarrow E, V_R$ 
9:    $UR \leftarrow \{u \mid u \in V_R \wedge E(u) \geq \sigma\}$ 
10:   $trustR \leftarrow trustR \setminus UR$ 
11:   $GtrustR \leftarrow GtrustR \cap trustR$ 
12:  timeout (off) →
13:  if  $GtrustR \neq \emptyset$  then
14:    return ( $GtrustR, Ep$ )
15:  else
16:    START(off,  $\delta$ )
17:     $GtrustR \leftarrow trustR$ 

```

Algorithm 2 Necessity: Σ necessary to solve \mathcal{O}

▷ Output of offloading engine to obtain Σ

```

1: function OUTPUT( $T_r, E$ )
2:    $UR \leftarrow \{u \mid u \in V_R \wedge E(u) \geq \sigma\}$ 
3:    $BL \leftarrow V_R \setminus (UR \cup T_r)$ 
4:   return ( $BL, E$ )

```

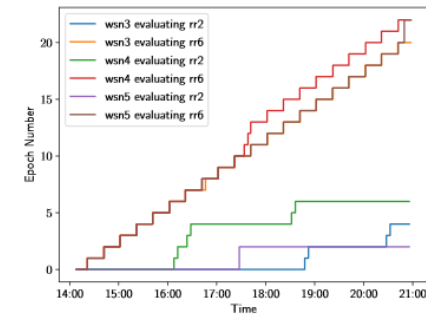
VI. IMPLEMENTING Σ AND PROBABILISTIC OFFLOADING

The use of Σ makes it possible to solve the offloading problem, however, it still needs to be understood if it is possible to implement Σ in a synchronous system. Unfortunately, the answer is negative. Our third major result is thus:

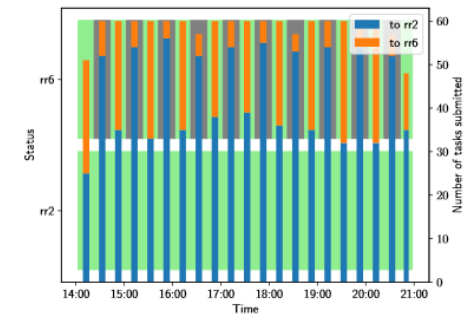
Theorem 3 (Impossibility of implementing Σ). *Given a synchronous system, it is impossible to implement Σ .*

Proof. We assume an algorithm A exists for Σ and then show a contradiction.

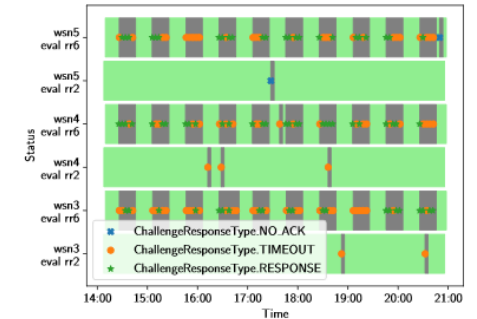
Consider a failure pattern F_0 where there are no failures and consider a computation $E = (T, \mathcal{O}, A, A)$ of Σ which



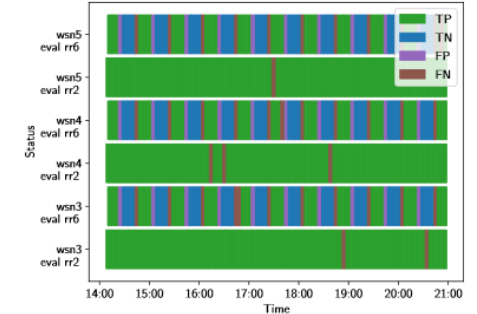
(a) Evolution of the Epoch number over time.



(c) The true status of resource-rich nodes and the number of tasks submitted to them in a time window where their behaviour was stable.



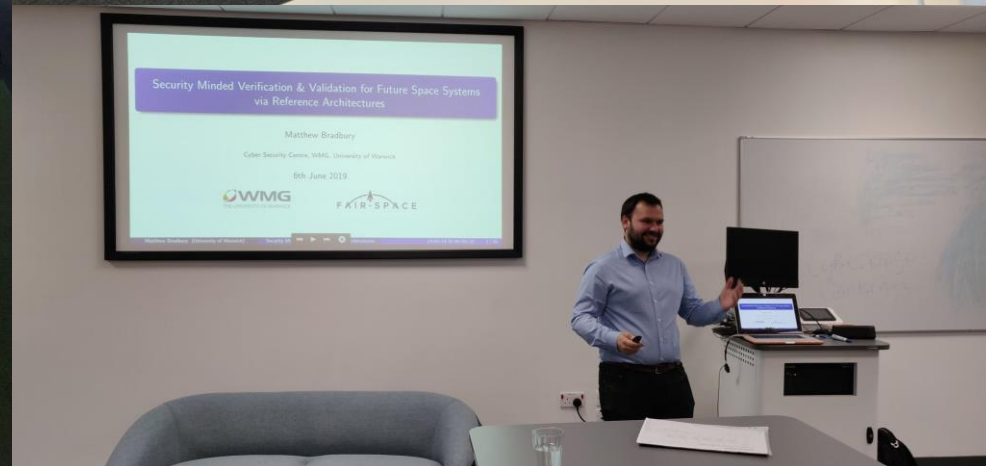
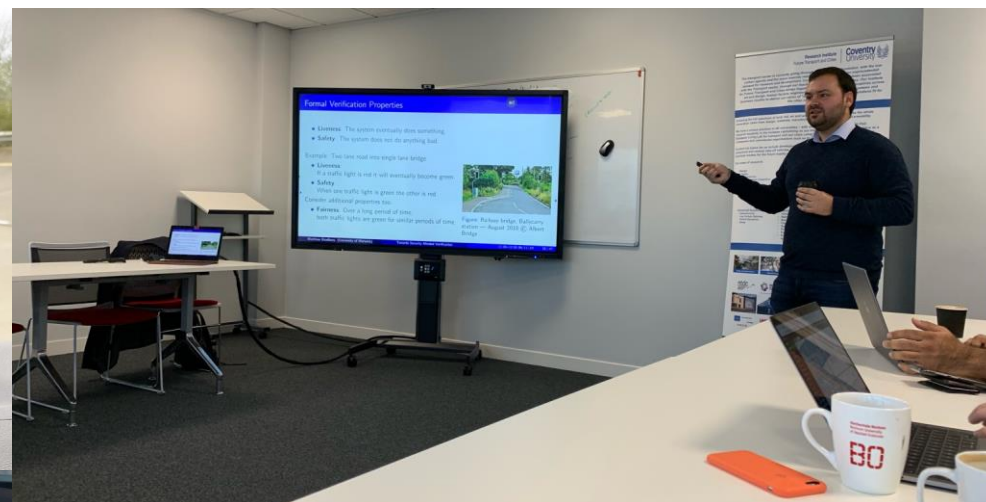
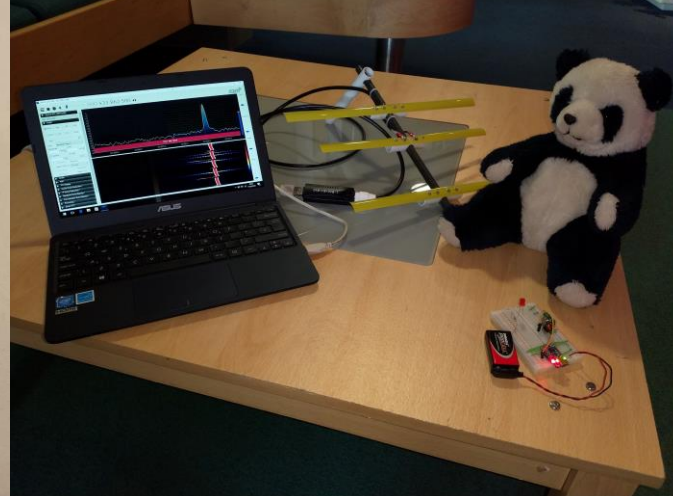
(b) Times at which resource-constrained nodes trusted resource-rich nodes. Events that led to loss of trust are indicated.



(d) Was the trust correctly evaluated? TP = trusted when good, TN = untrusted when bad, FP = trusted when bad, FN = untrusted when good.

Figure 6: Results for when resource-rich node 2 is good and 6 is unstable.

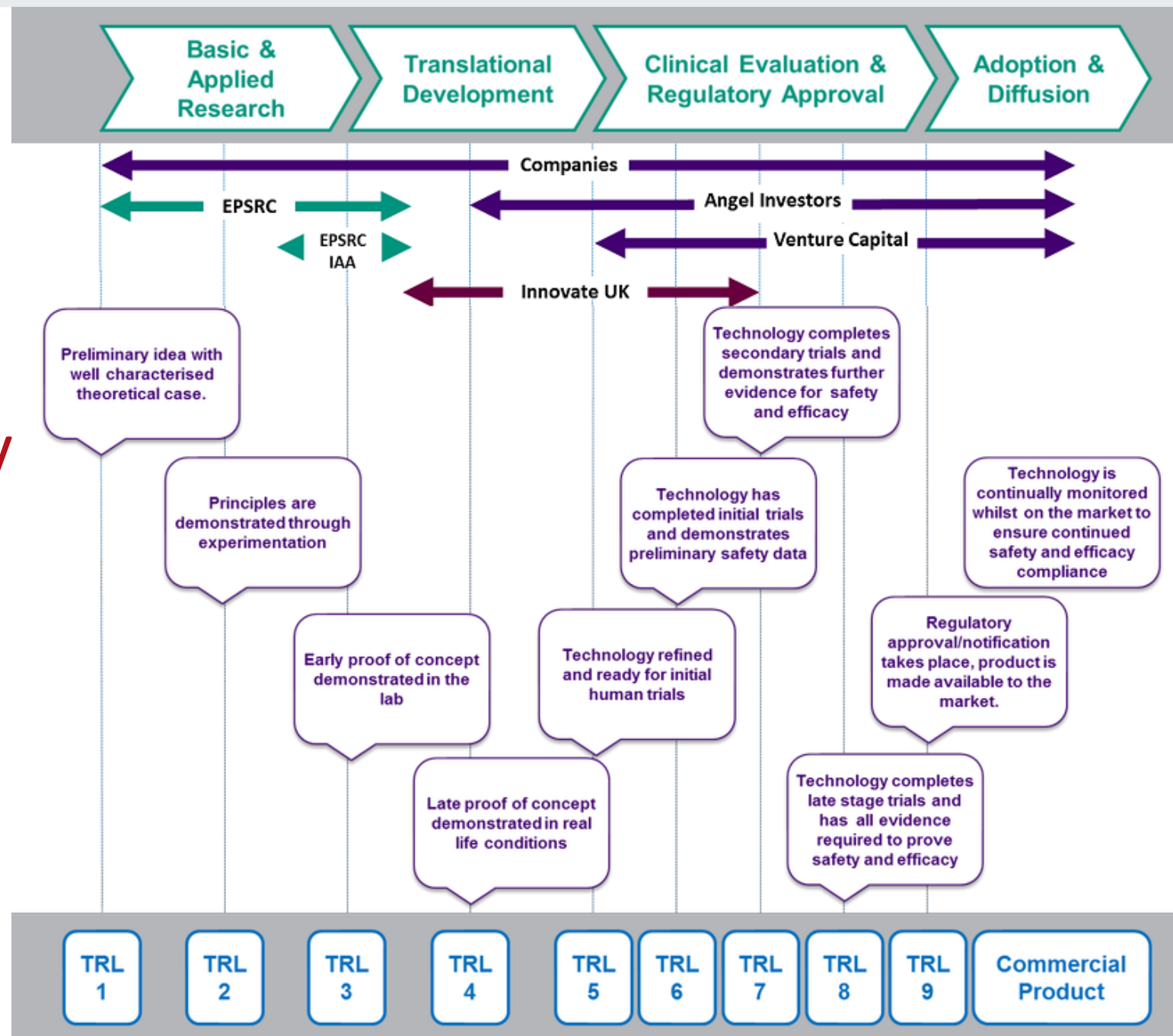
Reality: Experimentation, Outreach, ...



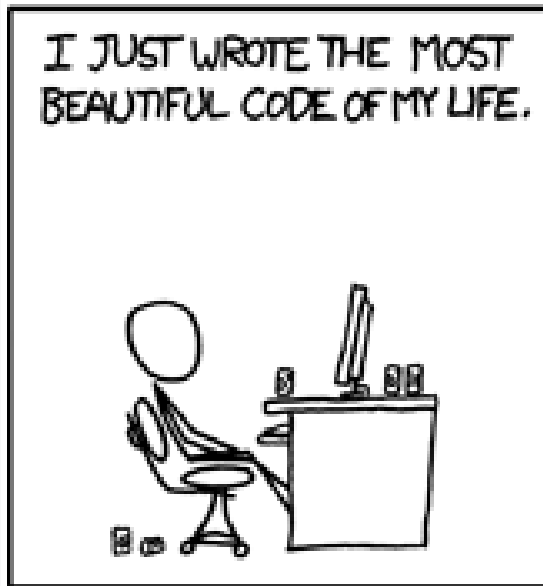
Identify and address **novel** problems

- Novel – a **new** problem, a **new** and **better** solution
- We tend not to focus on engineering activities
- Our work may incorporate building a system, but also:
 - Quantification of security / privacy / performance
 - Formal modelling to prove security / privacy of a system
 - Experimentation and analysis on a system

Technology Readiness Levels

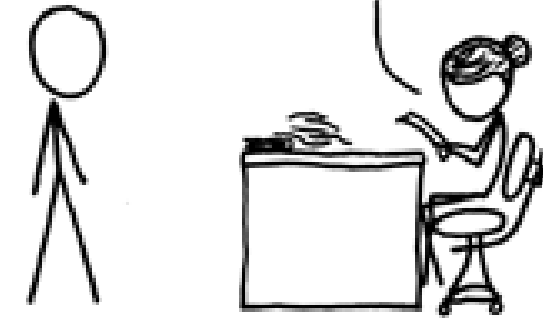


Academia vs. Business



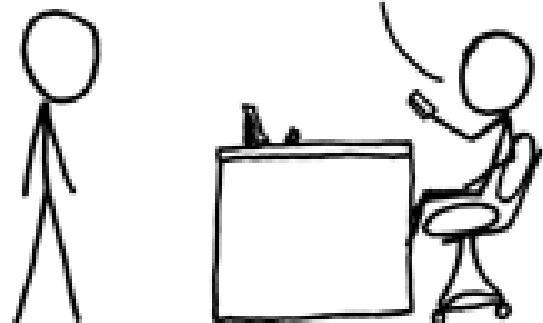
ACADEMIA:

MY GOD ... THIS WILL MEAN A HALF-DOZEN PAPERS, A THESIS OR TWO, AND A PARAGRAPH IN EVERY TEXTBOOK ON QUEUING THEORY!



BUSINESS:

YOU GOT THE PROGRAM TO STOP JAMMING UP? GREAT. WHILE YOU'RE FIXING STUFF, CAN YOU GET OUTLOOK TO SYNC WITH OUR NEW PHONES?



<https://xkcd.com/664/>

© Randall Munroe

Engaging with Academia

- Student Projects (Both UG and PG)
- Summer internships
- Knowledge Transfer Partnerships & Other Innovate UK Funded Initiatives
- Consultancy
- Contract / Collaborative Research
- CASE Studentship Sponsorship
- PhD students
- & More

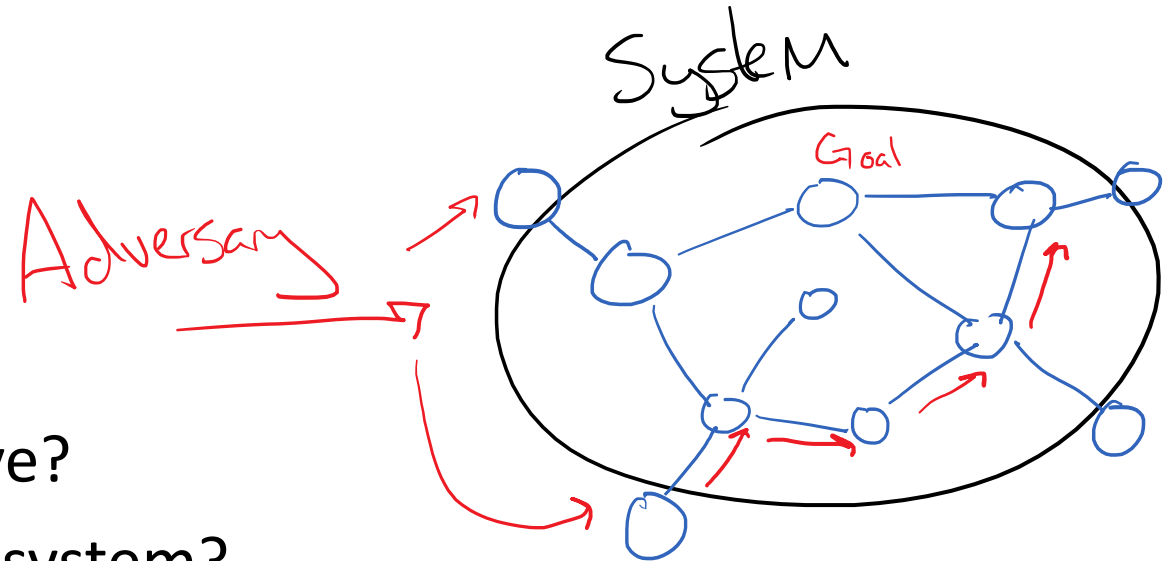


Automating Threat Modelling

What is Threat Modelling

You have a system:

- How can an adversary attack it?
- What goals might the adversary have?
- What path does it take through the system?
- How does it compromise different components of your system?



Threat Modelling tends to be Manual

Define the system

Do you know all components in your system?



What is your adversary, what are their capabilities, TTPs, ...



Identify how each component can be attacked

How do you know the component can be attacked in this way?

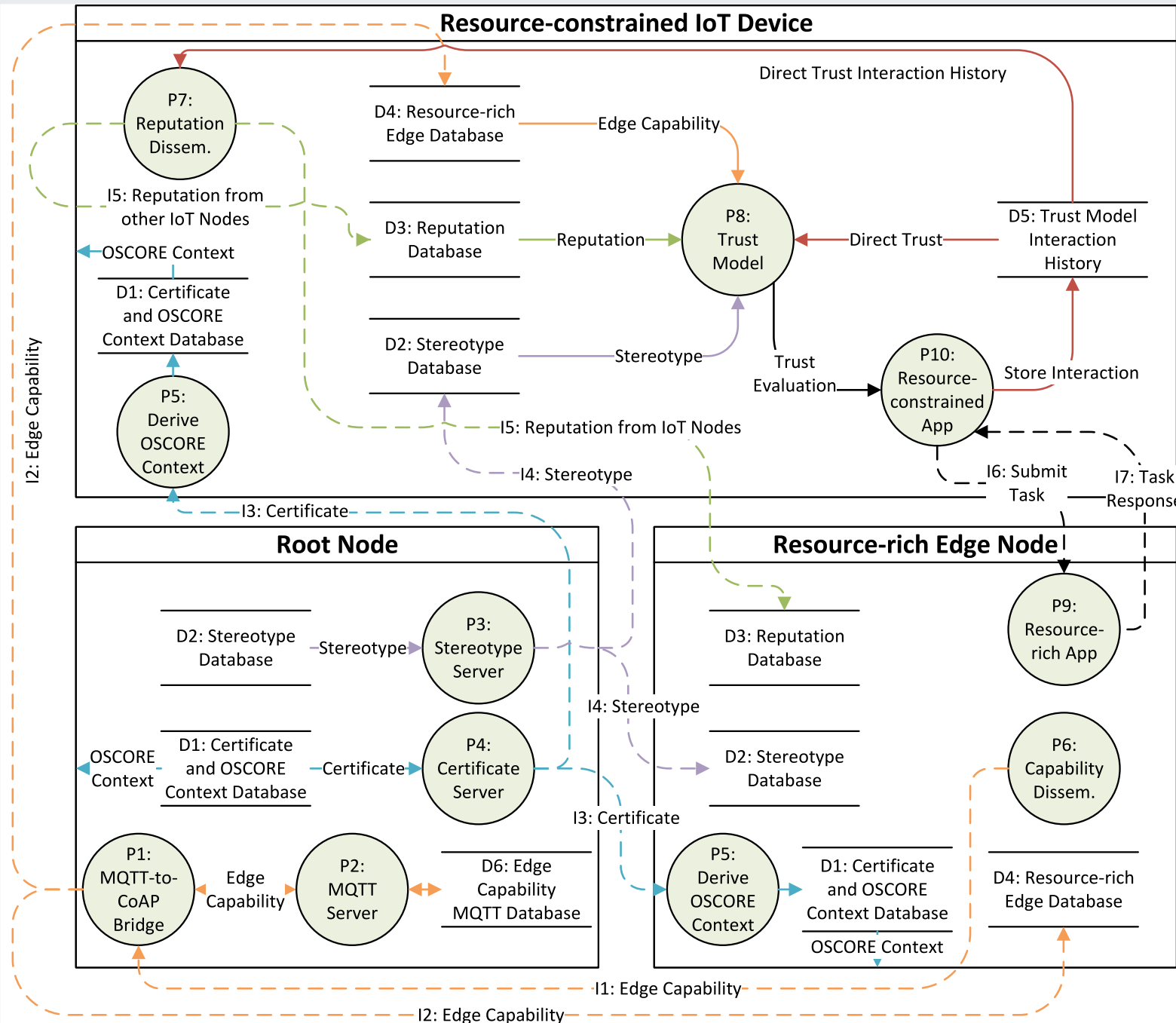


What path does the adversary take through the system to reach its goal?

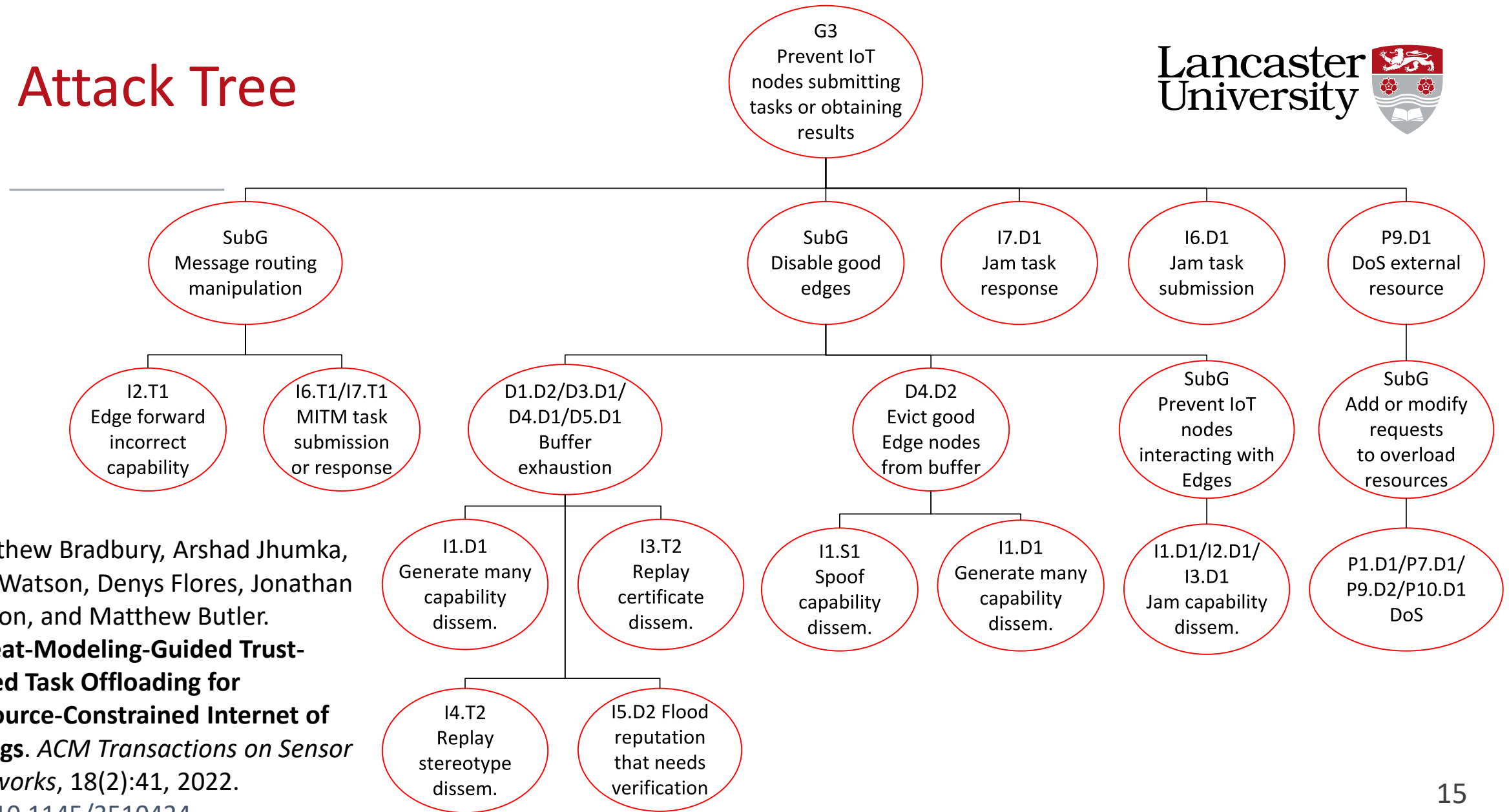
How do you know what an adversary is capable of after a compromise?

Data Flow Diagram

- Resource-constrained IoT devices offloading tasks to resource-rich Edge nodes
- What data is used to assess a measure of *behavioural trust* in an Edge node performing a task?



Attack Tree



Matthew Bradbury, Arshad Jhumka, Tim Watson, Denys Flores, Jonathan Burton, and Matthew Butler.

Threat-Modeling-Guided Trust-Based Task Offloading for Resource-Constrained Internet of Things. *ACM Transactions on Sensor Networks*, 18(2):41, 2022.

[doi:10.1145/3510424](https://doi.org/10.1145/3510424).

Existing Automatic Tools

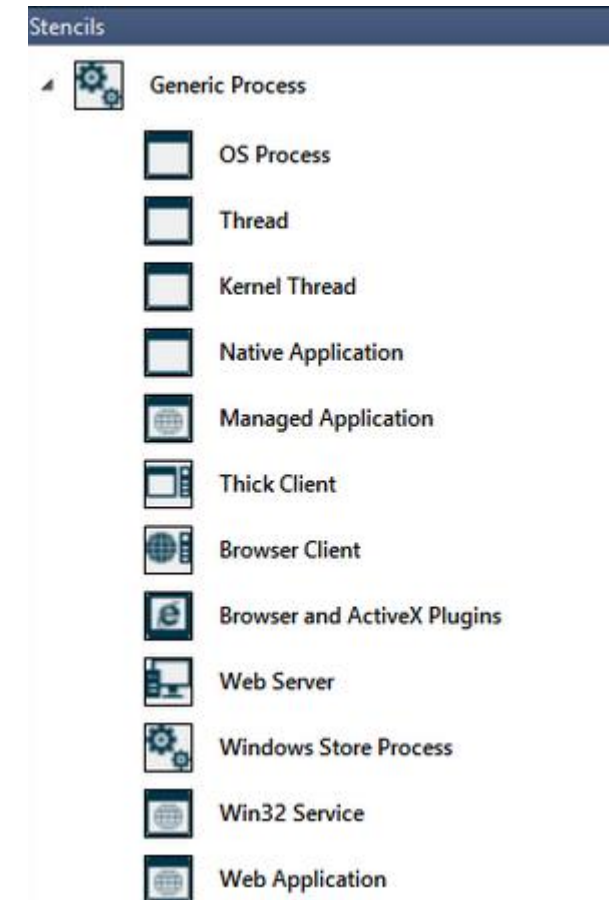
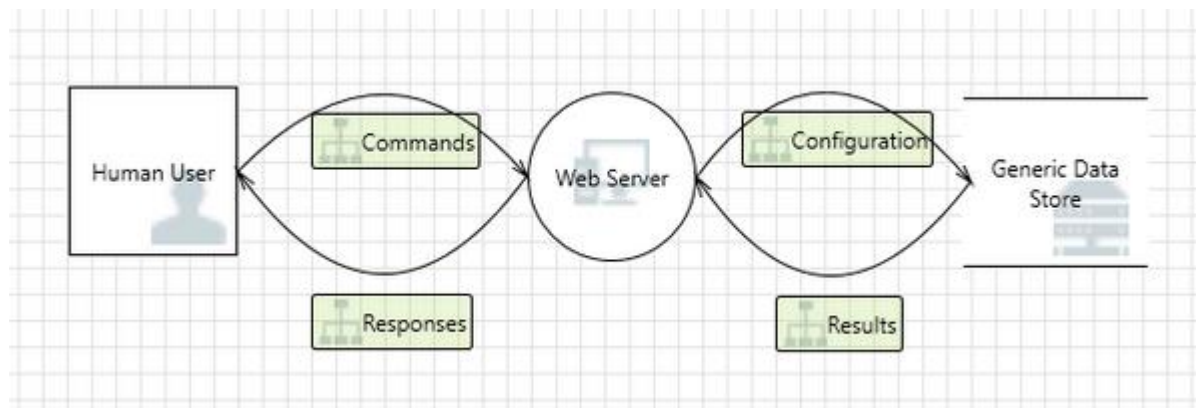
- Automatic threat modelling tools exist!
- Typically aim to solve the problem of:
“Identify how each component can be attacked”

```
"SID": "INP02",
"target": [
  "Process"
],
"description": "Overflow Buffers",
"details": "Buffer Overflow attacks target improper or missing bounds checking on buffer operations, typically triggered by input injected by an adversary. As a consequence, an adversary is able to write past the boundaries of allocated buffer regions in memory, causing a program crash or potentially redirection of execution as per the adversaries' choice.",
"Likelihood Of Attack": "High",
"severity": "Very High",
```

<https://github.com/izar/pytm/blob/master/pytm/threatlib/threats.json>

Existing Automatic Tools

- Typically these tools have pre-defined components
- You can define that these components interact
- The tool then produces threats against individual components



The Problem

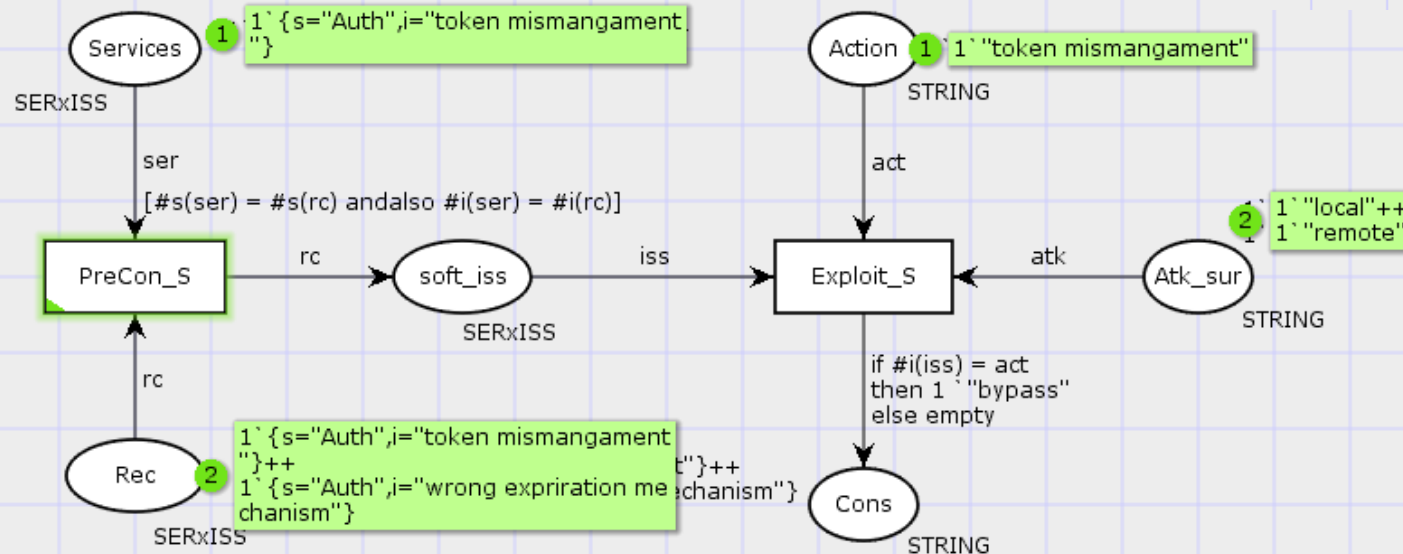
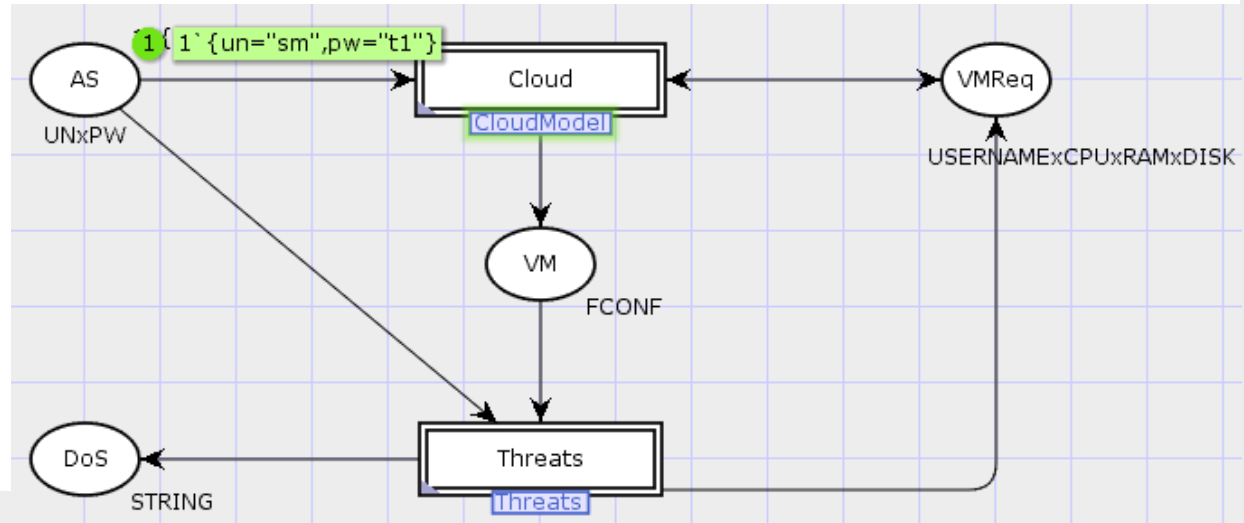
- How do we answer:
“What path does the adversary take through the system to reach its goal?”
- Challenging to do realistically with tools targeting high-level and low-level abstractions
- Do these tools tell us useful information?
- Do they just tell us about known attack vectors instead of discovering new ones?

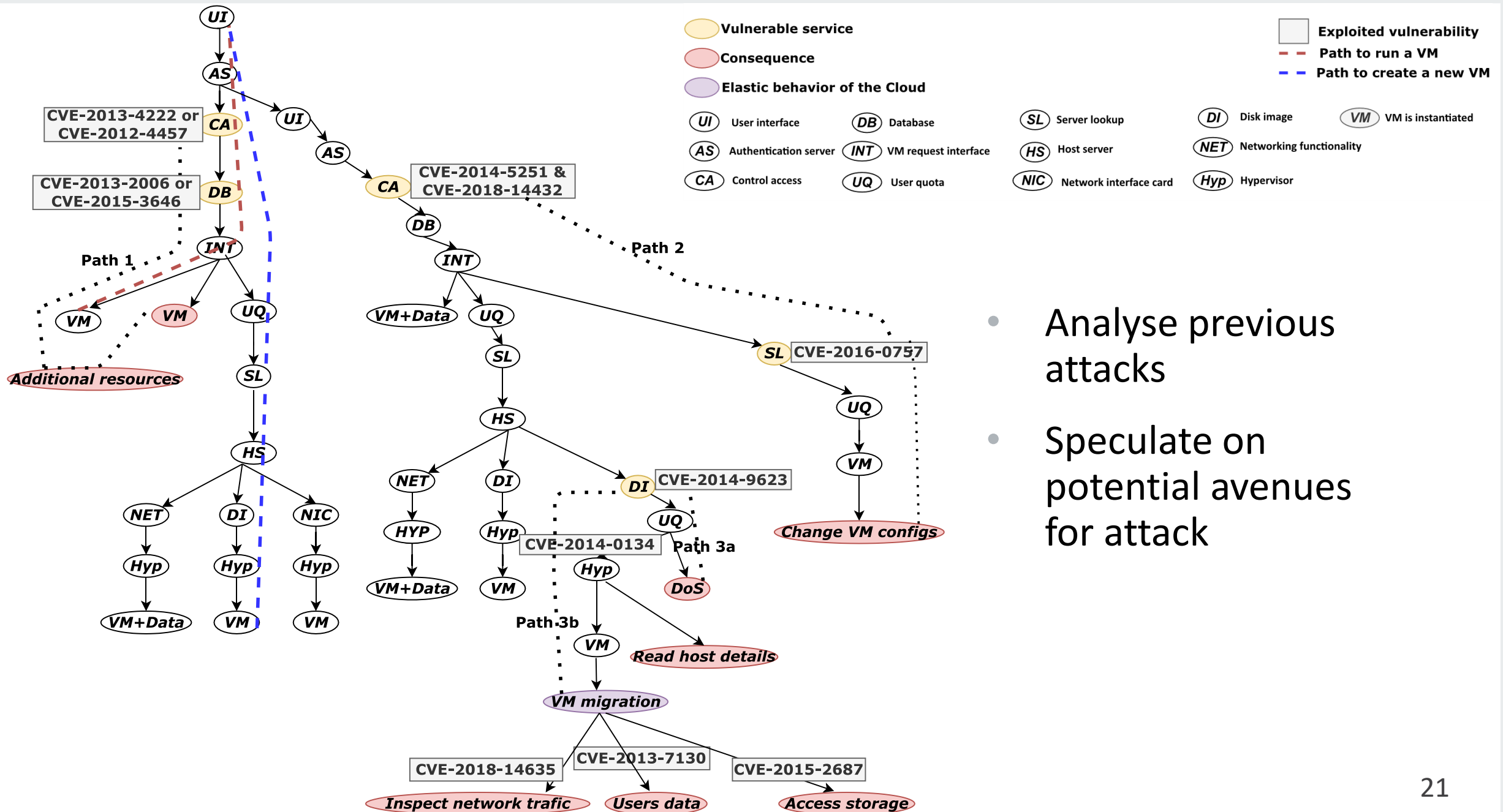
Scope for Academic Research

- Research questions:
 - How do we describe a system at the appropriate level?
 - How do we model adversary capability escalation?
 - Can we describe past attacks?
 - Can we speculate on future possible attacks?
- Make the model arbitrary – suitable for general systems

Some preliminary work

Model the system and adversary using Petri nets





- Analyse previous attacks
- Speculate on potential avenues for attack

More to be done

- Research questions:
 - How do we describe a system at the appropriate level? ✓
 - How do we model adversary capability escalation? Not yet
 - Can we describe past attacks? ✓ (partially)
 - Can we speculate on future possible attacks? ✓ (partially)
- The cost of creating these models is high - can we automate this process?
- How can cyber physical systems be considered?

References

- <https://www.oreilly.com/library/view/threat-modeling/9781492056546/ch04.html>
- <https://github.com/threatspec/threatspec>
- <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats>
- <https://github.com/izar/pytm>
- <https://owasp.org/www-project-threat-dragon/>

Thank you for watching!



Dr. Matthew Bradbury

Matthew Bradbury is a Lecturer conducting research into security, context privacy and trust assessment in resource-constrained and distributed systems.

Get in contact via:

<https://lancaster.ac.uk/cybersecurity>

<https://mbradbury.github.io>

m.s.bradbury@lancaster.ac.uk