Automatic Recovery of CPS against Known Attacks

Dr. Muhammad Taimoor Khan

Centre of Sustainable Cyber Security

University of Greenwich, UK

DeepSec 2024, Vienna

Motivation

Modern computing systems (e.g., CPS) involve

- Computation + Physics + Chemistry + Biology + ...
- Algorithm + Logic + Control + ...
- Inter-disciplinary domains (e.g., Economics, Energy, Medicine, Law, ...)

Modern computing systems are integrating all operational domains

- systems have become hybrid and overly complex
- reliable and secure interaction among inter-disciplinary domains

The systems must be self-aware and self-organized, i.e. they should know what they are trying to do.

Modern computing system requirements

Modern computing systems should be certified s.t. they

- support audit against regulations/laws/policies ... among inter-disciplinary domains
- formally assure their reliability and security for operations
- are understandable by any machine or human
- can be verified by any machine
- are self-organized and resilient

Application of program analysis and verification is key to challenges

Context

Given a system implementation/design/model C
Show that C and its execution C_e are secure

- they are repaired/recovered, if vulnerable/compromised

Existing solutions

- Security Testing and Simulation
  - random or highly random - statistics based
  - not rigorous
  - not exhaustive
  - no definition of a reliable test or a simulation
  - shows the presence of bugs/threats and **not** their absence. **Dijkstra**
  - no formal assurance/guarantee
- Standardization Organizations, e.g. ISO
  - manual, based on testing and simulations
  - mostly empirical
  - no formal assurance

The solutions are quantitative and does not offer formal assurances

Our strategy and approach

Our strategy

- Build it right and continuously monitor

We **certify** that

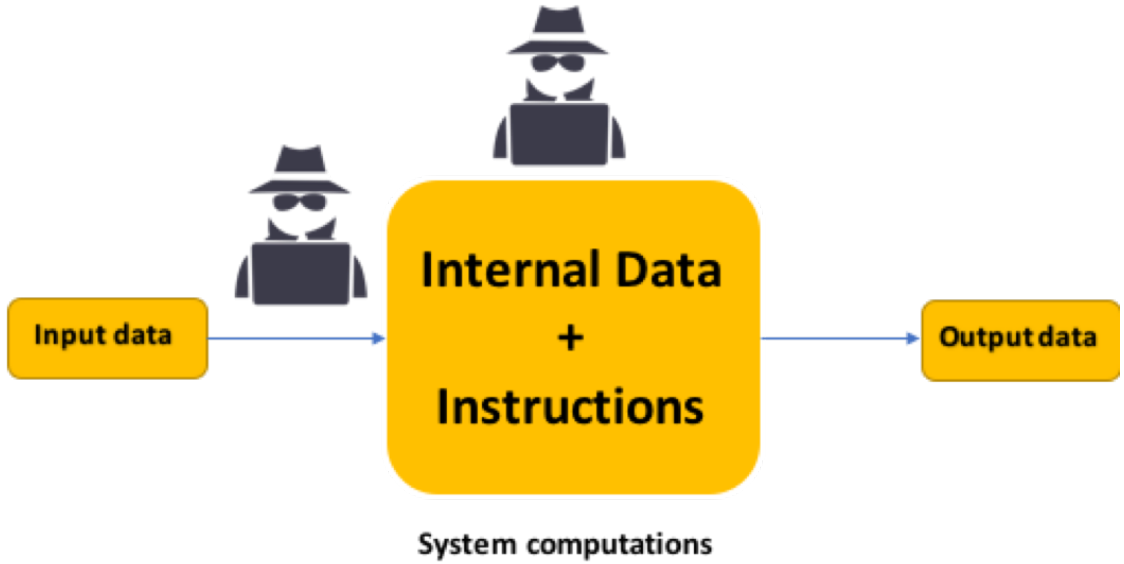
- the design of C is reliable and secure (and repaired when vulnerable)
- and its execution C_e is also reliable and secure (and recovered when compromised)

Our approach

- Specify the behavior of a system S as its
  1. functional properties, e.g. pre- and post-conditions, invariant
  2. non-functional properties, e.g. security, performance, energy
- Certify that the design C meets its specification S
  - through step-wise but sound refinements of the design
- Monitor the execution C_e through a middleware
  - ARMET: comparing the executions of specification S and implementation C_e

The system behavior

The behavior characterizes computations operating on input data



Can we assure that

- the computations are reliable and secure, e.g
  - behave as expected
  - terminate
  - are free of bugs/errors
  - cannot be compromised by any insider or external adversary
- the external-input data is reliable, e.g.
  - do we believe that we see is real?
  - data can be legal but not real - data integrity threat

Our assurance

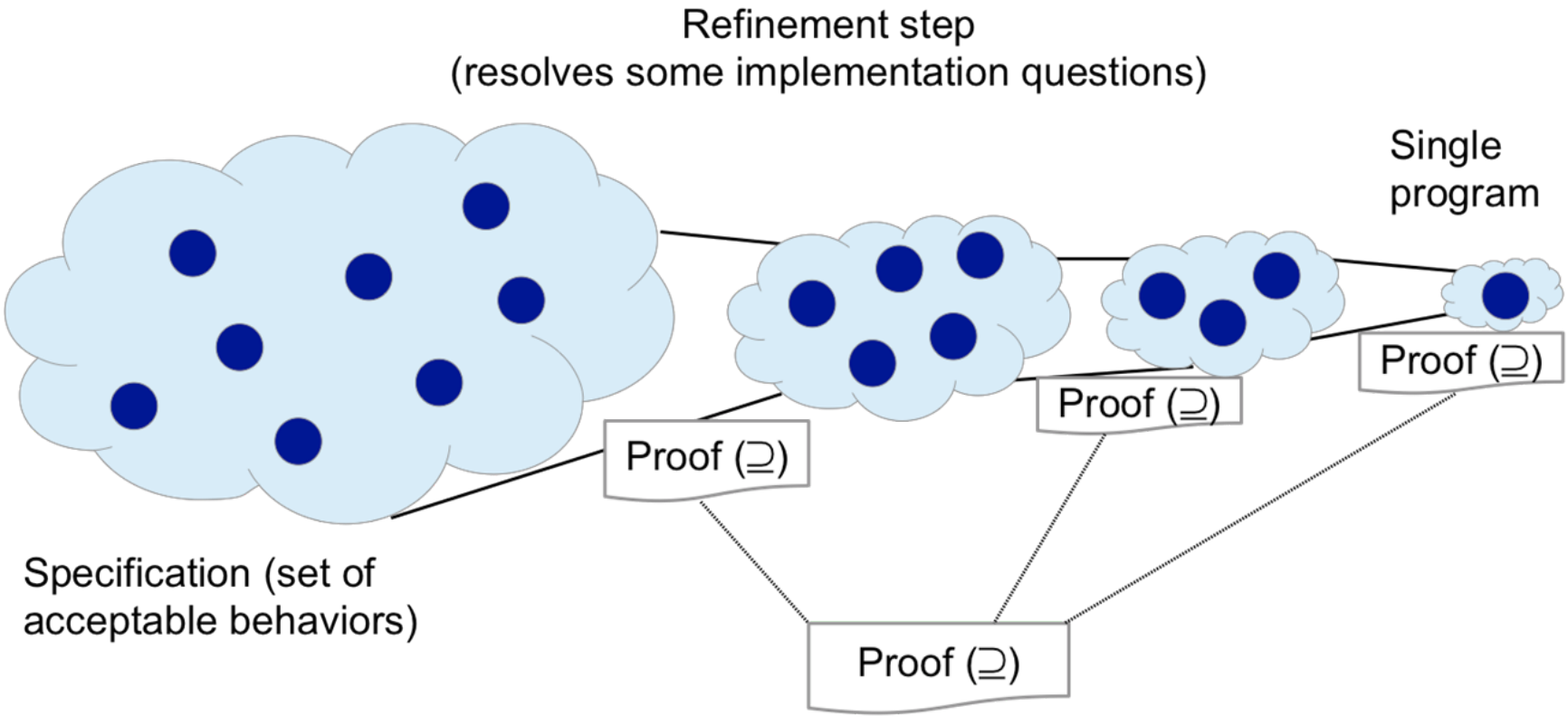
We **certify** that

- the computations design is reliable and secure
  - through stepwise refinement of the system specification
- the external-input data is free of integrity threats
  - through non-linear verification based vulnerability analysis of the system specification
- the computations execution is reliable and secure
  - through monitoring the consistency between expected and execution behaviors
  - optionally, through monitoring the identified integrity vulnerabilities

We design the computations in known environment
and execute the computations in unknown environment

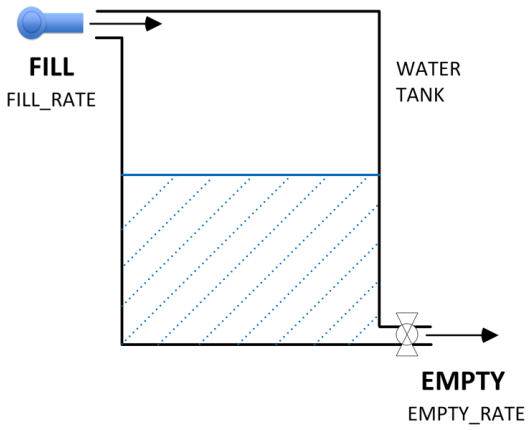
Program derivation through stepwise refinement

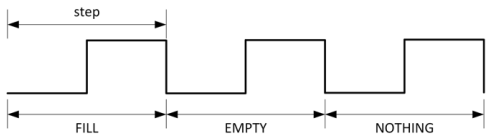
Abstract data type based declarative specification



Proof constructed and checked with Coq  , a general-purpose logic platform

Example specification

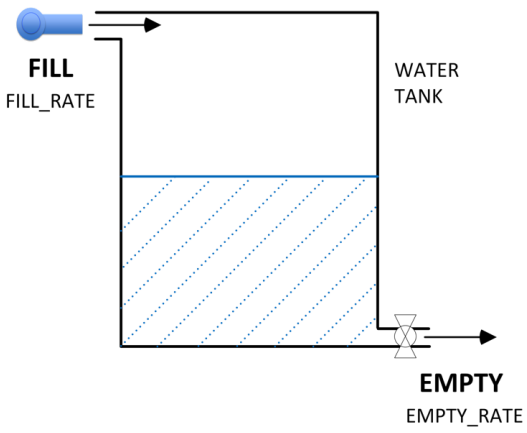


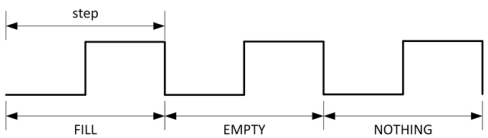



```
public enum Action { NOTHING, FILL, EMPTY }

class WaterTankSpec {
    private int water_level = 0;

    public void newSensorReading(int reading) {
        if (abs(reading - water_level) > SENSOR_ACCURACY)
            water_level = {n | True};
    }

    public Action timestep(int target_level) {
        Action act = {a | (a = FILL → water_level + FILL_RATE ≤ TANK_MAX)
                        ∧ (a = EMPTY → water_level - EMPTY_RATE ≥ 0)};
        if (act == FILL)
            water_level += FILL_RATE;
        else if (act == EMPTY)
            water_level -= EMPTY_RATE;
        return act;
    }
}
```

Specification with a set of behaviors

Example code

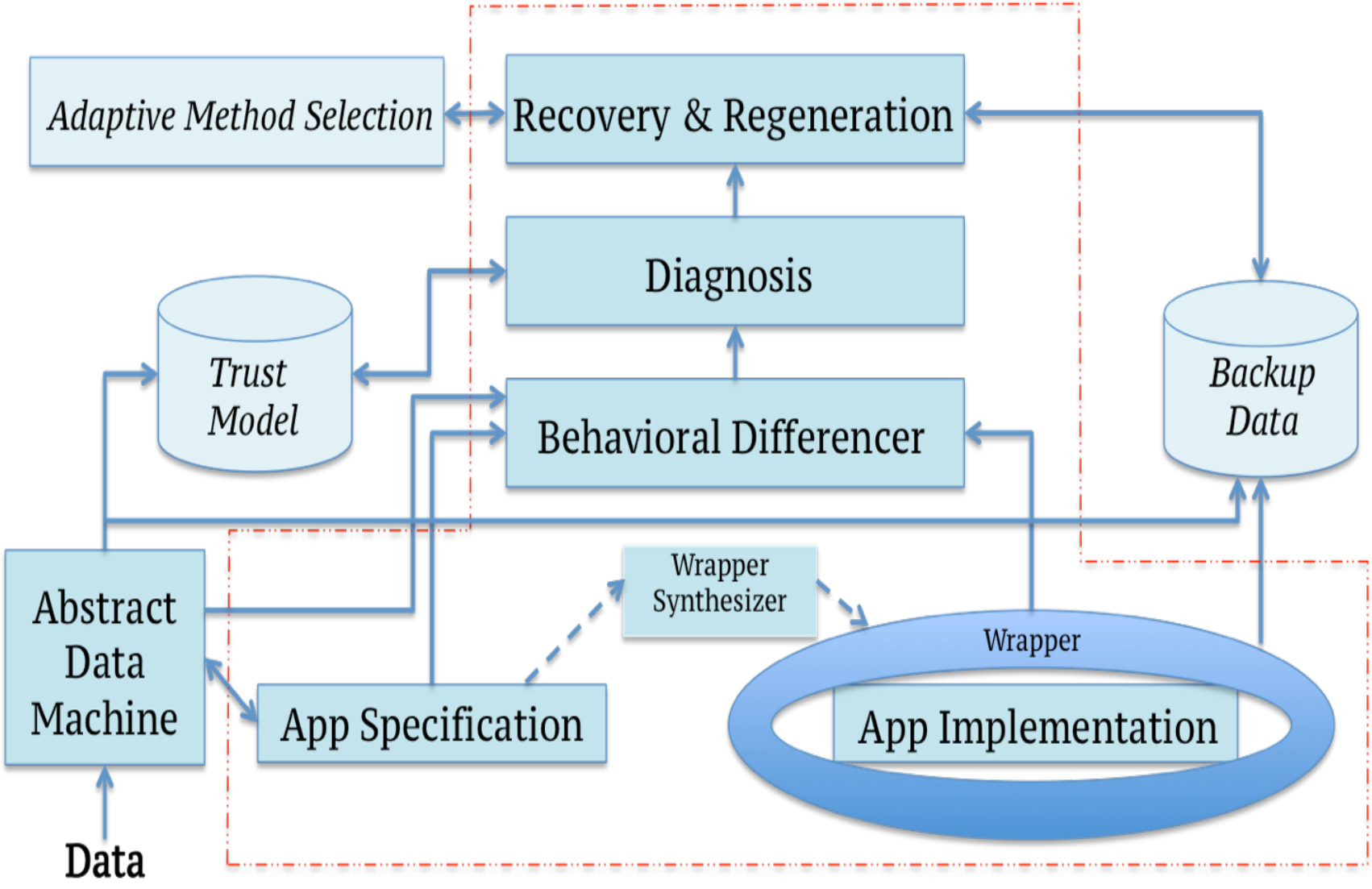




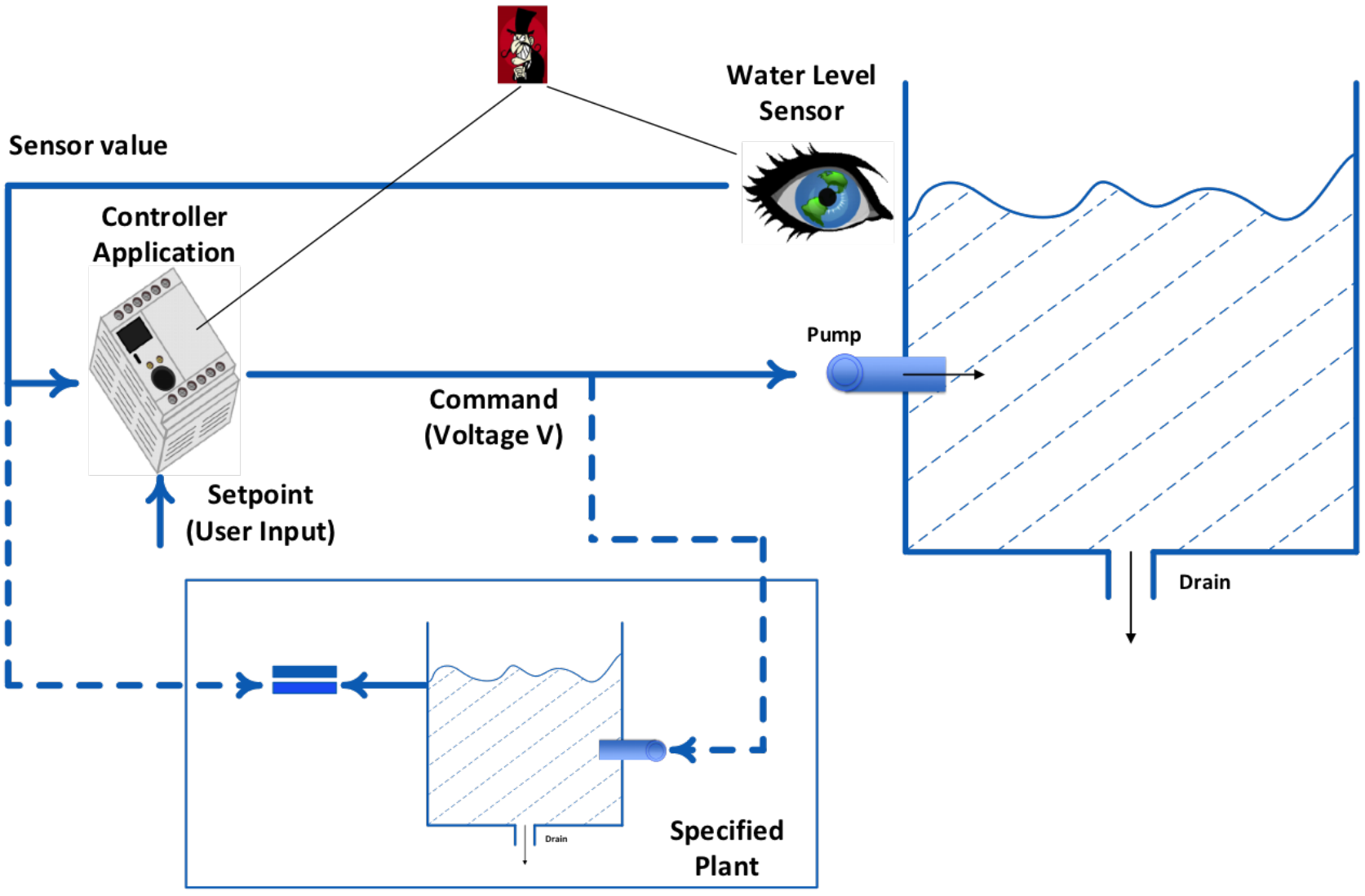

```
public enum Action { NOTHING, FILL, EMPTY }

class WaterTank {
    private int water_estimate = 0;

    public void newSensorReading(int reading) {
        water_estimate = reading;
    }

    public Action timestep(int target_level) {
        if (water_estimate < target_level
        && water_estimate + SENSOR_ACCURACY + FILL_RATE < TANK_MAX)
            water_estimate += FILL_RATE; return FILL;
        } else if (water_estimate > target_level
            && water_estimate - SENSOR_ACCURACY - EMPTY_RATE >= 0)
            water_estimate -= EMPTY_RATE; return EMPTY;
        } else
            return NOTHING;
    }
}
```

Single program with one behavior

ARMET - monitoring and resilience



Demo


Water Level
Sensor
Sensor value
Controller
Application
Pump
Command
(Voltage V)
Setpoint
(User Input)
Drain
Drain
Specified
Plant

ARMET - reliable, secure and resilient software

- Self-aware system
  - through specification and dependency-directed reasoning
- System is allowed to only behave legally
  - specify legal behavior of the system (predictions)
  - observe runtime behavior of the system (observations)
  - continuous monitoring of prediction/observation consistency
  - IF inconsistency, THEN diagnosis
  - recovery (safe state from alternate, reliable resources)
- Detection of known and unknown errors and attacks
  - as inconsistency between observations and predictions
- System adaptability to evolving constraints, standards
  - specify policies as legal behavior and monitor behavioral consistency
- ARMET is sound and complete
  - whenever there is an attack or error, it alarms
  - whenever it alarms, there is an attack or error

Automated vulnerability analysis for data integrity

Vulnerability analysis in non-linear domain is NP hard

- requires solving logical combination of non-linear constraints
- non-convex and non-smooth optimization
  - multiple feasible regions
  - multiple locally optimal points within each region

δ-complete decision procedure solves the problem, e.g. dReal

Given: a model of system (may include non-linear constraints) and a set of properties with an error δ

Ask: are there any values that satisfy the model but are not real?

1. No (unsat)
   - there are no such values
   - you are safe
2. Yes (δ-sat)
   - there are such values (returns set of values)
   - values are vulnerabilities/attack vectors for the model
   - remove such vulnerabilities from the design
     - refinine/strengthen the model/constraints
     - until unsat

More results

We have also applied the developed techniques to automatically

- repair applications against time vulnerabilities
- recovered e-commerce applications against attacks

Automatic Repair of Applications (1)

The notion of time is used by networked systems as

- they wait for a specific time to synchronise

Time is represented as integer values in modern programming languages and

- can have negative values
- suffers from overflow problem

Therefore, we have developed a technique that

- performs an abstract analysis over the time domain of a program (using Time Type System)
  - to detect the problematic time expressions
- exploits binary representation of machine numbers
  - to automatically repair the problematic expressions

Automatic Repair of Applications (2)

We have validated the approach

- on 20 open source (Java) projects
- repairing 246 errors

Name	# Hash	# Classes	# Methods	# Tests	SLOC	Time [s]	# Time Var	# Typed	# DoT	# Patches
Activemq	ccf56875b	5,100	44,072	20,450 (30.41%)	421,839	473.63	31,094	31,071	23	23
Activiti	917246113	2,103	15,381	3,968 (60.68%)	139,672	203.48	11,266	11,266	0	0
Airavata	391843a00	9,320	70,875	162 (8.25%)	711,587	1,146.17	131,439	131,439	0	1
Alluxio	2bf790f505	3,364	24,975	4,270 (45.85%)	233,897	223.00	36,854	36,833	21	8
Atmosphere	d51726dcc	500	4,101	504 (55.16%)	35,843	35.86	4,074	4,072	2	0
Aws-sdk-java	5984638d0b	27,208	205,432	2,586 (57.33%)	1,795,234	28,091.57	186,336	186,335	1	9
Beam	3b03106b55	3,844	21,404	8,930 (66.64%)	210,960	201.79	19,548	19,543	5	1
Camel	497fa7760e1	20,024	116,080	47,704 (40.68%)	1,065,292	4,696.54	77,766	77,760	6	6
Elastic-job	dbb79ef4	611	2,497	1,842 (87.84%)	26,418	19.17	1,942	1,941	1	0
Flume	7d3396f2	995	6,705	2,288 (48.00%)	85,750	52.29	9,086	9,075	11	8
Hadoop	128dd91e100	12,597	100,635	10,830 (51.16%)	1,267,414	1,955.95	121,859	121,715	144	35
Hazelcast	02e3fbf737	7,663	59,294	11,035 (76.57%)	649,789	736.54	37,383	37,355	28	1
Hbase	44f8abd5c6	9,535	128,928	4,614 (34.11%)	1,201,149	1,995.11	248,935	248,895	40	51
Jetty	65528f76c5	3,781	25,554	12,742 (45.65%)	342,602	301.51	26,568	26,559	9	8
Kafka	c74acb24e	1,896	14,007	9,331 (71.87%)	149,644	119.29	16,440	16,429	11	19
Lens	cdd7b099	1,036	8,114	2,432 (53.83%)	99,523	75.88	10,622	10,613	9	9
Nanohttpd	f1cb85c	124	716	478 (75.25%)	7,532	4.25	742	742	0	0
Neo4j	a41464ed3ba	9,158	61,407	150,648 (53.61%)	680,986	770.02	43,814	43,804	10	18
Sling	73fe13fa28	6,022	38,049	7,078 (35.80%)	433,384	1,046.44	50,845	50,834	11	49
Twitter4j	cf6afc3e	418	4,642	724 (40.00%)	32,436	41.63	2,985	2,984	1	0
SUM	**-**	**125,299**	**952,868**	**302,616 (51.93%)**	**9,590,951**	**42,190.12**	**1,069,598**	**1,069,265**	**333**	**246**

Automatic Recovery of Applications (1)

Software Engineers trust

- the programming languages in which they develop applications that but actually suffer from various vulnerabilities that are exploited to launch real attacks at run-time
  - e.g., web applications suffer from Injection - SQL, LDAP, Command Parameter, XPath, NoSQL, Log4shell, ...

Therefore, we have developed a technique that

- models the vulnerabilities/threats in a specification language
  - to detect the threats at run-time
- exploits semantics of the attacks and the programming language
  - to automatically recover the threats at run-time in
    - full-mode - recovers the system state
    - partial-mode - recovers the operation state
    - basic-mode - recovers the user state

Automatic Recovery of Applications (2)

We have validated the approach

- in real-world use case scenarios in the frame of our project ENSURESEC
- recovering most of the vulnerabilities, e.g., SQL injection, malicious deliveries of items, malicious sellers, to name a few

Demo

Team

- Muhammad Taimoor Khan
  - Operational Lead, ACE-CSR, University of Greenwich, UK
- Dimitrios Serpanos
  - Director, Industrial Systems Institute, Uni. of Patras, Greece
- Howard Shrobe
  - Director CyberSecurityInitiative, MIT CSAIL, USA
  - Program Manager (Machine Common Sense), DARPA, USA
- Martin Pinzger
  - Professor of Software Engineering, Alpen-Adria University, Austria
- Giovanni Liva
  - Team Captain, Dynatrace, Austria
- and all collaborators

Thanks

DEEPSEC
IN-DEPTH SECURITY