Date 13/05/2024

Challenge 01 "Automation of cybersecurity verification for cyber physical systems"

Challenge 01 Working Group Meeting

Philippe Massonet, Coordinateur Scientifique CETIC
Guillaume NGuyen, UNamur
Martin Vivian, UCLouvain
Denis Darquennes, CETIC
Christophe Crochet, John Aoga, UCLouvain

digital wallonia 4.cyberwal

Wallonie recherche SPW

https://cyberwal.be
https://cyberexcellence.be

Agenda

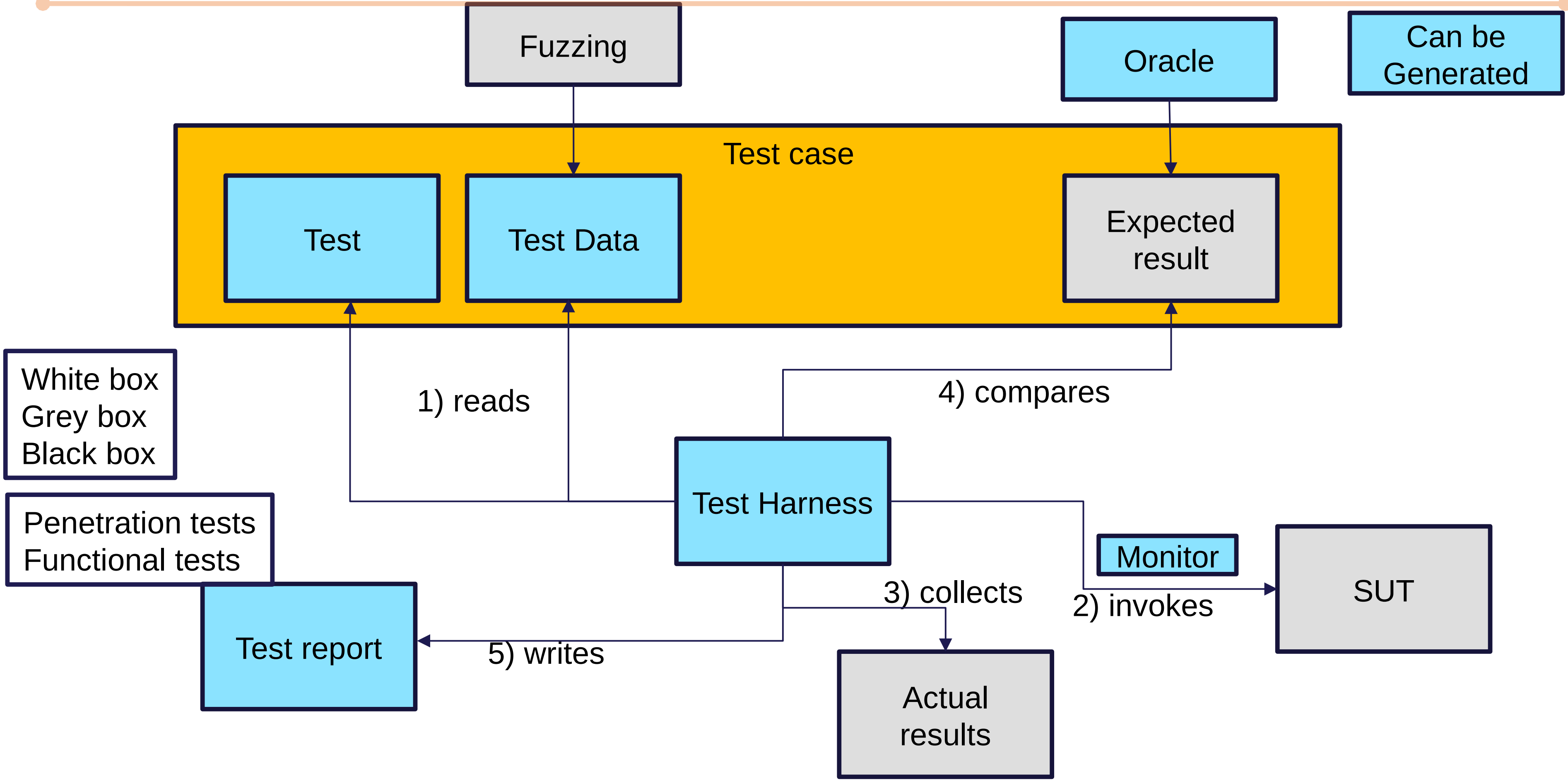
15:30-15:50	Improvements for stateful fuzzing	Martin Vivian (UCLouvain)
15:50-16:10	Identification of Cyber Physical System (CPS) & Orchestration of fuzzing testing	Guillaume Nguyen (Unamur)
16:10-16:30	Automated cybersecurity testing with genetic algorithms	Denis Darquennes, Philippe Massonet (CETIC)
16:30-16:50	Vérification of protocols via PFV (Protocol Formal Verification)	Christophe Crochet/John Aoga, UCLouvain

Challenge 01 "Automation of cybersecurity verification for cyber physical systems"

- **Summary of the Challenge:**
  - Penetration testing: still a very manual process, requires cybersecurity experts
  - Ambition: automate (partly) the creation of penetration tests to make penetration tests more accessible to companies (SMEs, large companies)
- **Research Challenges**:
  - ○ Automatic generation of functional cybersecurity tests (security architecture), use of different generation techniques (to compare) for penetration tests:
    - ○ Fuzzing techniques,
    - ○ Generation of tests by genetic mutation
    - ○ Generating tests from models

    ...
  - ○ Partial automation in the form of assistance with the creation process and definition of penetration tests.

“Test Harness pattern” - Generation



Overview of research problems

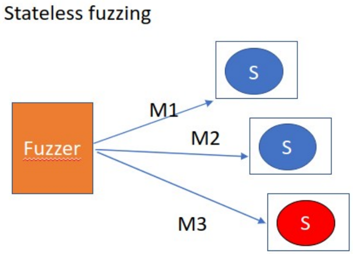
	White box	Grey box	Black box	Pen tests	Func tests
Improvements for stateful fuzzing			X	X	
Identification of Cyber Physical System (CPS) & Orchestration of fuzzing testing	X			X	X
Automated cybersecurity testing with genetic algorithms	X				X
Vérification of protocols via PFV (Protocol Formal Verification)	X				X

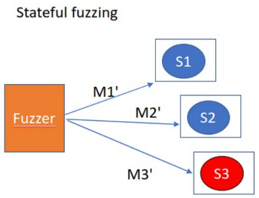
Improvements for stateful fuzzing

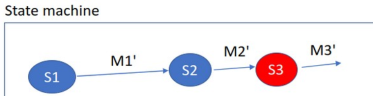
Martin Vivian, UCLouvain

Reminder

Fuzzing on State Machine

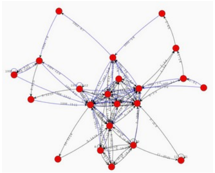






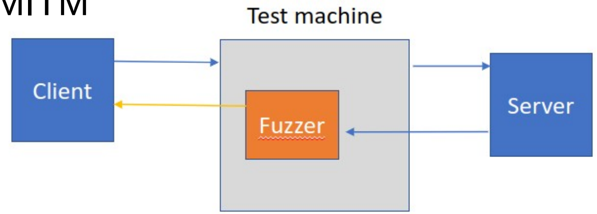
Each message have their grammar
Order of the message in this example, we must send M1' before M2' to reach S3

State machine

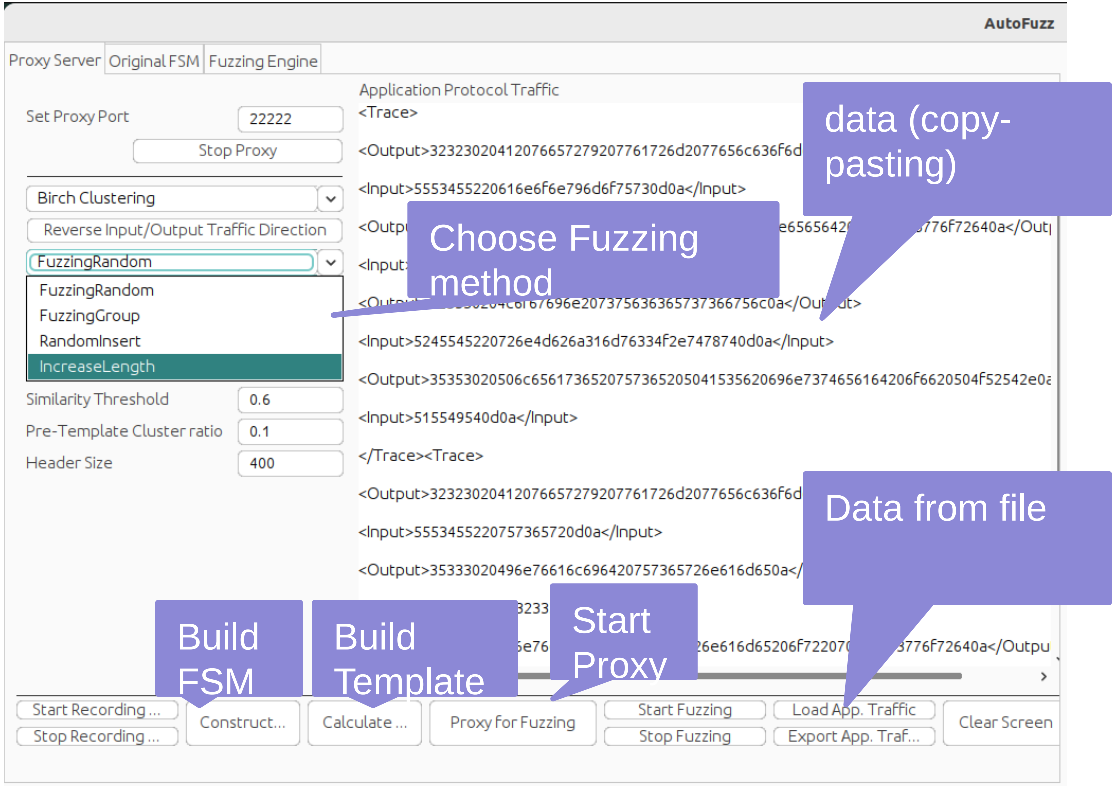


Template of message
for each state transition

MITM



Tool Introduction



Set Proxy Port: 22222

Birch Clustering

FuzzingRandom

Similarity Threshold: 0.6

Pre-Template Cluster ratio: 0.1

Header Size: 400

Application Protocol Traffic

```
<Trace>
<Output>32323020412076657279207761726d2077656c636f6d
<Input>5553455220616e6f6e796d6f75730d0a</Input>
<Outp e6565642 776f72640a</Out
<Input
<Out 4c6f67696e207375636365737366756c0a</Out ut>
<Input>5245545220726e4d626a316d76334f2e7478740d0a</Input>
<Output>35353020506c6561736520757365205041535620696e7374656164206f6620504f52542e0a
<Input>515549540d0a</Input>
</Trace><Trace>
<Output>32323020412076657279207761726d2077656c636f6d
<Input>5553455220757365720d0a</Input>
<Output>35333020496e76616c696420757365726e616d650a</
3233
e76 26e616d65206f72207 3776f72640a</Outpu
```

Data in XML format

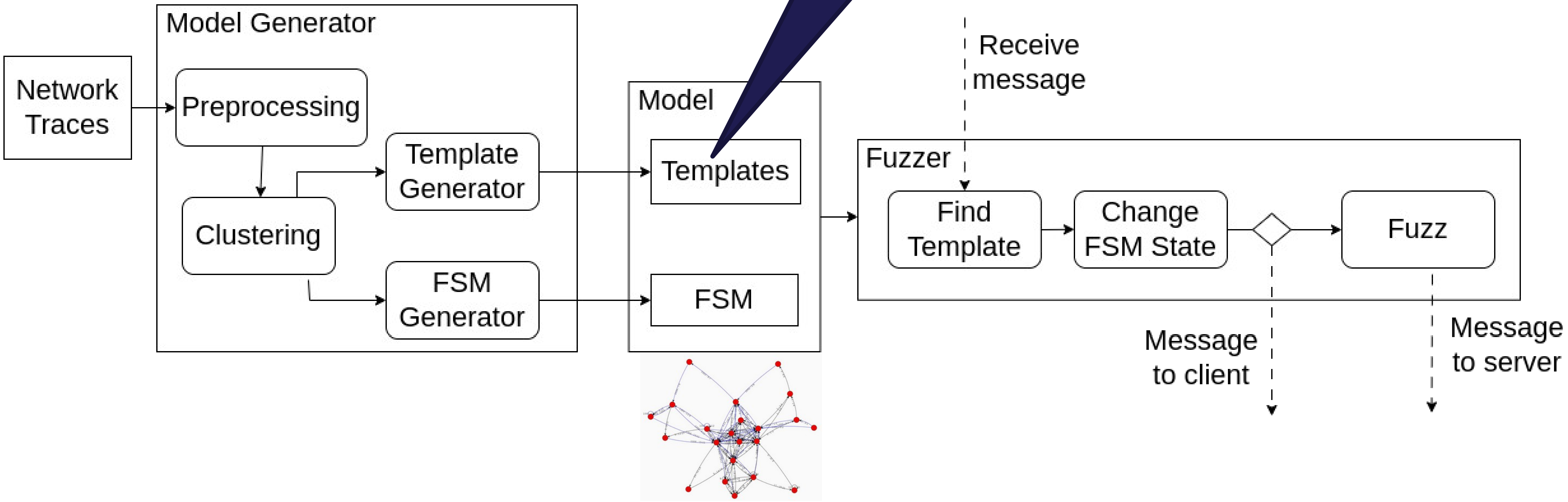
<Trace></Trace> : session
<Input></Input> : data
<Output></Output> : data

Initially start to improve Autofuzz :
https://sourceforge.net/projects/autofuzz/

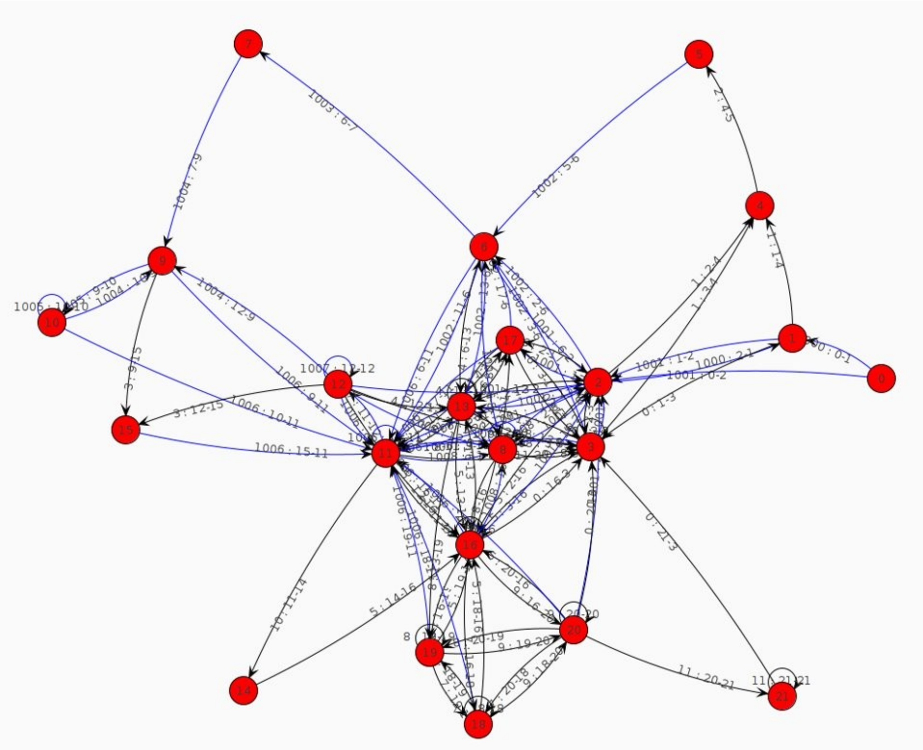
Finally keep UI and modify the rest

Process of the Tool

Template example
1:FA++—
2:FE6B++

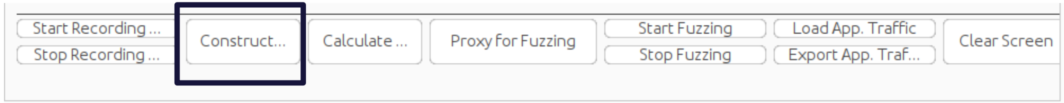


Build State Machine



Integrated tool for FSM

FlexFringe : https://github.com/tudelft-cda-lab/FlexFringe



Clustering for State Machine

Gather similar networking messages from the data to build a fsm based on this cluster

1) Preprocessing:

Find highest variable part (like crc, session id...) and don’t take in account for clustering

Example : FE+++++++A——-+++—- => “+” indicate high variablility

2) Header separation (clustering on header) :

Choose a header length for the clustering

Example : header = 4 for frame “AABBCCDD” => header part is AABB

Create separate cluster for size below

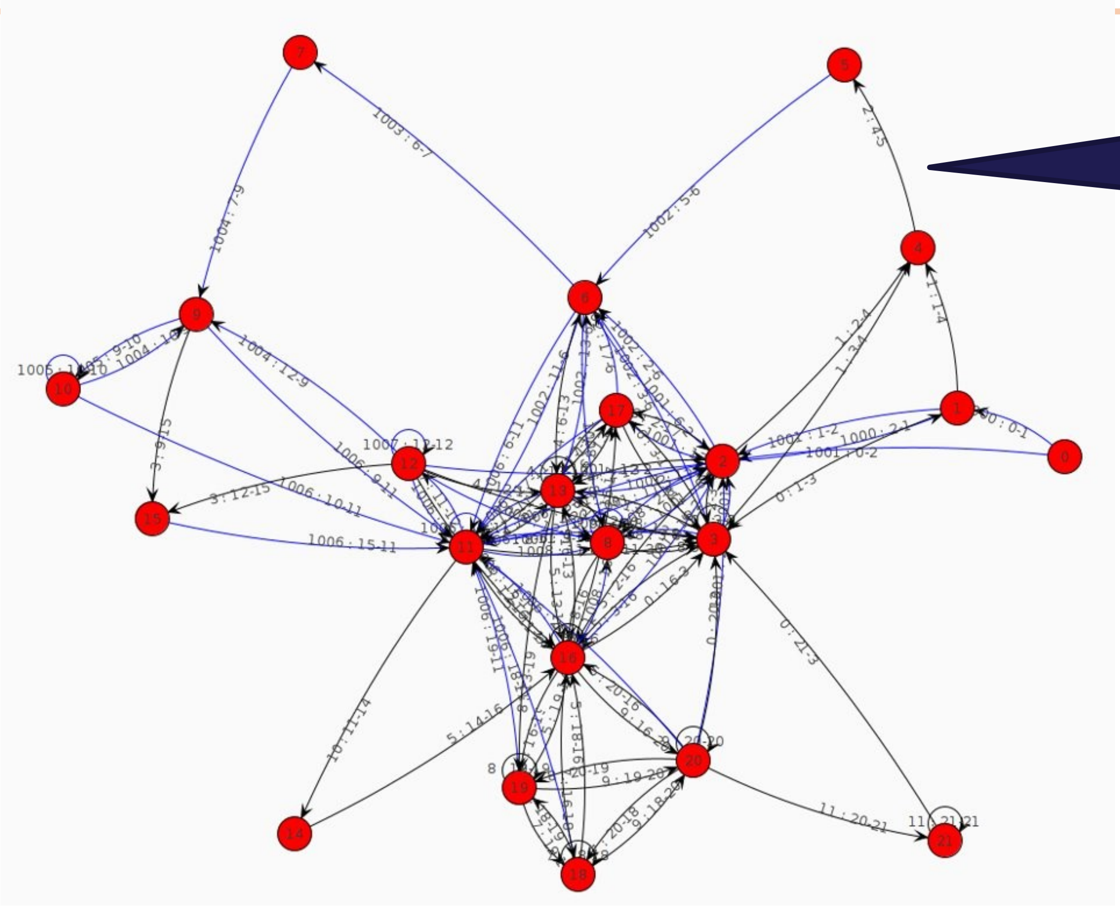
Create a separate cluster for unclusterized data

3) Algorithm used

Hiearchical Clustering algorithm :

(BirchLeaf clustering) : https://github.com/sbobek/smiling/blob/master/demo/src/main/java/smile/demo/vq/BIR

Link between State Machine and cluster



- We are at the state 4
- We receive an ouptut message that match the cluster ID 2 then we go to the state 5

Cluster Id 2 should be represented by the template : AF++-C—++
And the message received should be AFEBFCEEAA

If output are fuzz then the message will be fuzzed following the corresponding template

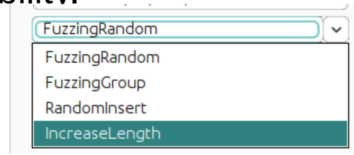
Template and Fuzzing Strategy

Template :

- Since the fuzzer is in MitM we can modify the messages by following a template
- We need before to calculate the template for each cluster
- Distinction between constant value, strong and weak variable with Xi² (distribution for each position in the frame).

Strategy :

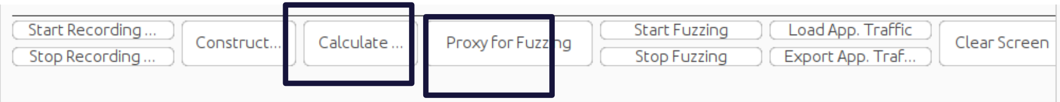
- Don’t fuzz constant, less fuzzing for highest variable and high fuzzing for weak variable. Template example: FE++++—— (red no fuzzing, yellow low probability to be fuzzed and green higher probability.
- Fuzzing Function :



- Possibility to not fuzz all frame, select states to fuzz.

Information :

Distinction beween constant value, strong ar



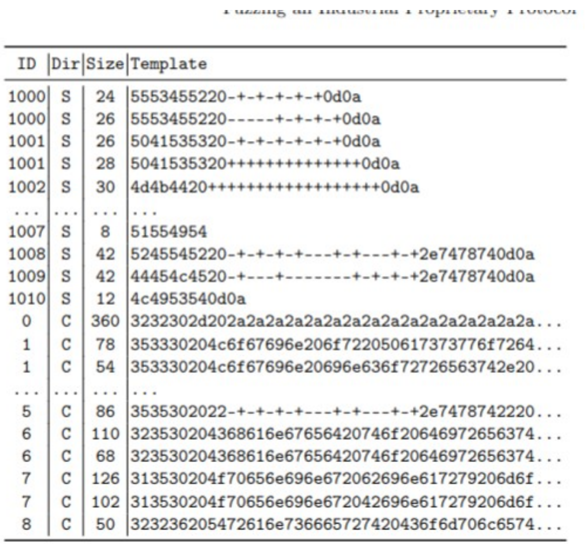
FTP case study

- First case study to test our implementation
- Test on FTP server implementation (Open and compact FTP server version 1.2)
- https://sourceforge.net/projects/open-ftpd/files/open-ftpd/

Results

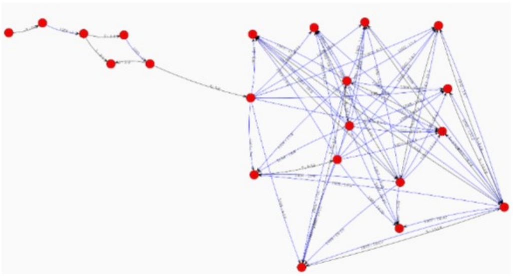
- Our tool was able to reproduce the crash from the papers

The insertion of "/r", "/n" or a space symbol in the middle of the parameters could crash whole template variable part (i.e. sending a command without a parameter) could also

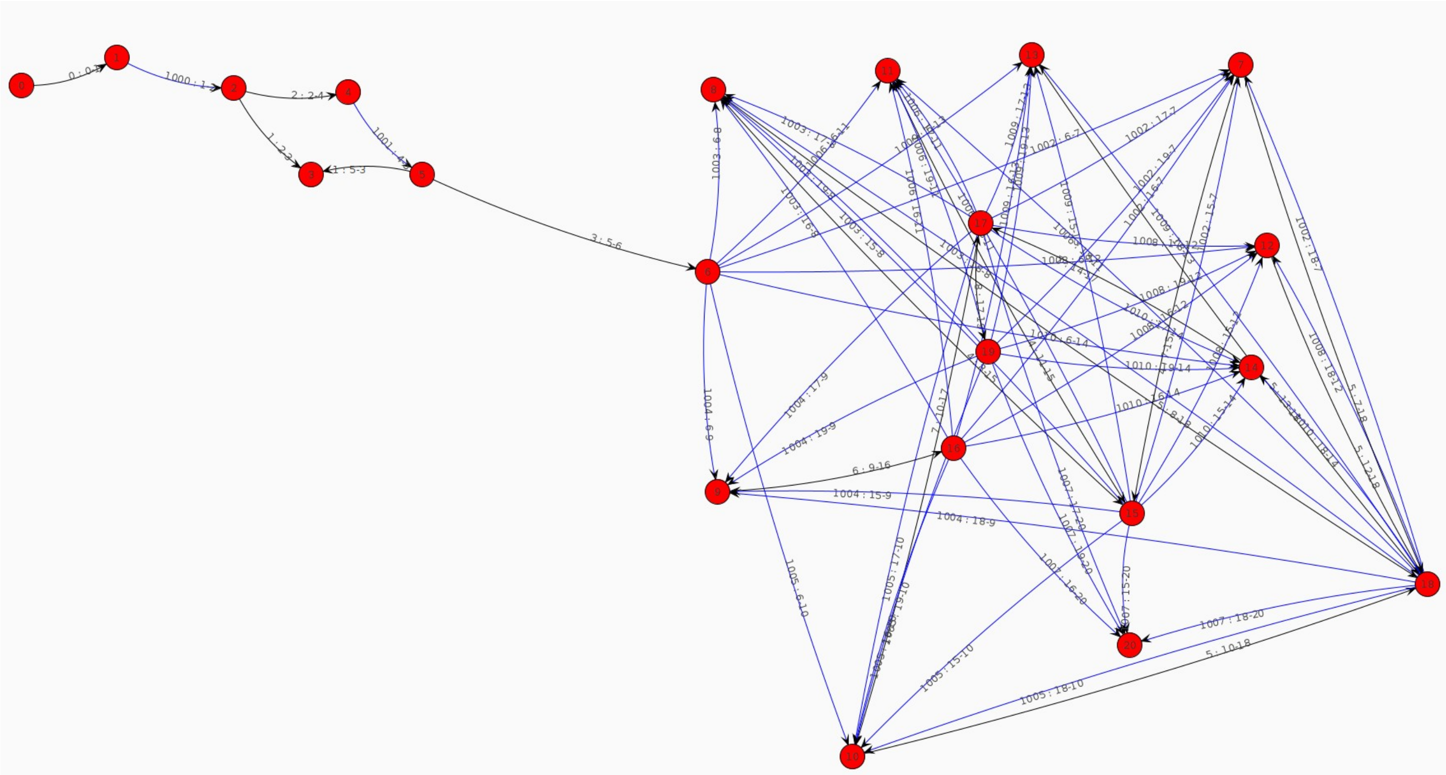


Fuzzing an Industrial Proprietary Protocol

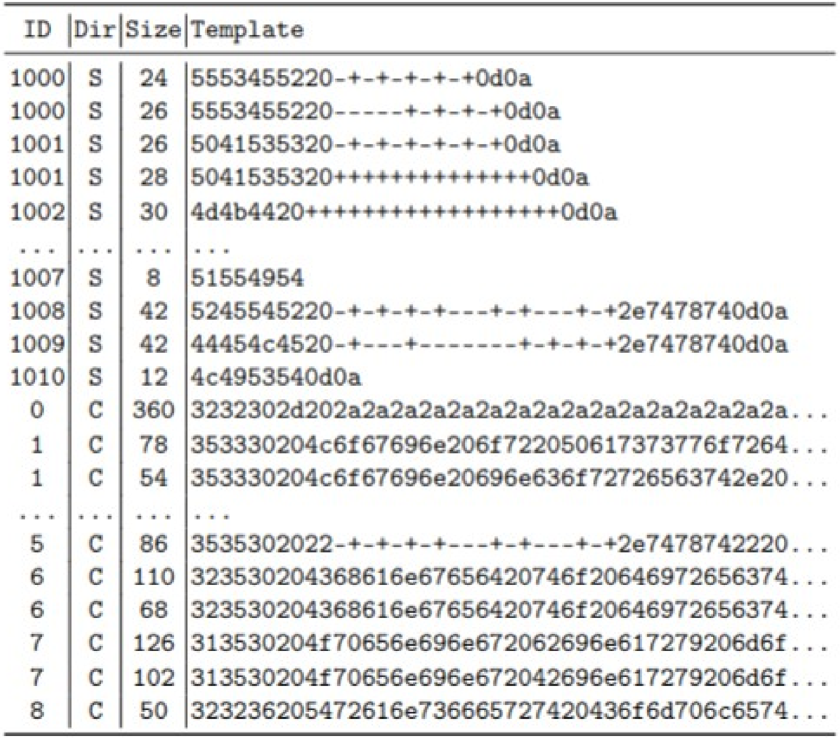
ID	Dir	Size	Template
1000	S	24	5553455220-+-+-+-+-+0d0a
1000	S	26	5553455220-----+-+-+-+0d0a
1001	S	26	5041535320-+-+-+-+-+-+0d0a
1001	S	28	5041535320+++++++++++++0d0a
1002	S	30	4d4b4420+++++++++++++++++0d0a
...	...	...	...
1007	S	8	51554954
1008	S	42	5245545220-+-+-+-+---+-+---+-+2e7478740d0a
1009	S	42	44454c4520-+---+-------+-+-+-+2e7478740d0a
1010	S	12	4c4953540d0a
0	C	360	3232302d202a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a...
1	C	78	353330204c6f67696e206f72205061737376f7264...
1	C	54	353330204c6f67696e20696e636f72726563742e20...
...	...	...	...
5	C	86	3535302022-+-+-+-+---+-+---+-+2e7478742220...
6	C	110	323530204368616e67656420746f20646972656374...
6	C	68	323530204368616e67656420746f20646972656374...
7	C	126	313530204f70656e696e672062696e617279206d6f...
7	C	102	313530204f70656e696e672042696e617279206d6f...
8	C	50	32323620547261e736665727420436f6d706c6574...



FSM : FTP


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
0 : 0-1
1000 : 1-2
2 : 2-4
1 : 2-3
1001 : 4-5
1 : 5-3
3 : 5-6
1003 : 6-8
1003 : 17-8
1006 : 6-11
1003 : 16-8
1003 : 15-8
1006 : 19-11
1006 : 16-11
1009 : 17-13
1002 : 6-7
1002 : 17-7
1002 : 19-7
1002 : 16-7
1002 : 15-7
1009 : 15-13
1008 : 17-12
1008 : 6-12
1008 : 19-12
1008 : 16-12
1010 : 6-14
1010 : 19-14
1010 : 16-14
1010 : 15-14
1010 : 18-14
1002 : 18-7
1008 : 18-12
5 : 7-18
5 : 12-18
5 : 8-18
1004 : 17-9
1004 : 19-9
1004 : 6-9
6 : 9-16
1004 : 15-9
1004 : 18-9
7 : 10-17
1005 : 6-10
1005 : 17-10
1005 : 19-10
1005 : 15-10
1005 : 18-10
1007 : 17-20
1007 : 19-20
1007 : 16-20
1007 : 15-20
1007 : 18-20
5 : 10-18

Template : FTP

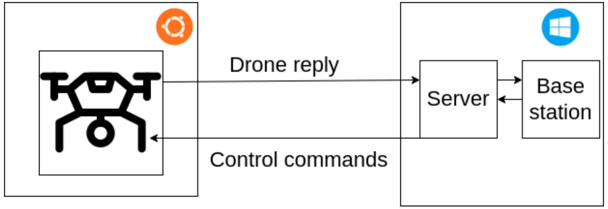


ID	Dir	Size	Template
1000	S	24	5553455220-+-+-+-+-+0d0a
1000	S	26	5553455220-----+-+-+-+0d0a
1001	S	26	5041535320-+-+-+-+-+-+0d0a
1001	S	28	5041535320++++++++++++++0d0a
1002	S	30	4d4b4420++++++++++++++++++0d0a
...	...	...	...
1007	S	8	51554954
1008	S	42	5245545220-+-+-+-+---+-+---+-+2e7478740d0a
1009	S	42	44454c4520-+---+-------+-+-+-+2e7478740d0a
1010	S	12	4c4953540d0a
0	C	360	3232302d202a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a...
1	C	78	353330204c6f67696e206f72205061737776f7264...
1	C	54	353330204c6f67696e20696e636f72726563742e20...
...	...	...	...
5	C	86	3535302022-+-+-+-+---+-+---+-+2e7478742220...
6	C	110	323530204368616e67656420746f2064697265637 4...
6	C	68	323530204368616e67656420746f20646972656374...
7	C	126	313530204f70656e696e672062696e617279206d6f...
7	C	102	313530204f70656e696e672042696e617279206d6f...
8	C	50	323236205472616e736665727420436f6d706c6574...

Application to Industrial drone Case Study

Industrial protocol caracterised by

- CRC
- Session ID
- DateTime
- Telemetric, logs packets
- Header
- Black-box
- No always tuple input-output
- No text-interpretable protocol



Results on the case study

- Relevant State machine
- Relevant template to identify cluster and reusable for fuzzing

- Tools is enough good to create separate state for the drone commands
  - For example, each time that we start the motors we go to the state 9.
  - It gives the possibility to only fuzz specific control command without modifying telemetries packet and get the good fuzzing template.

Attempt on the RTSP Protocol

- RTSP (Real Time Streaming Protocol) https://github.com/rgaufman/live555
- Protocol inside Profuzzbench : https://github.com/profuzzbench/profuzzbench
- When we launch Profuzzbench we find crash on RTSP
- But when we try to replay the frames we don't find the bugs
- Reason they fixe a session id in the code for the reproducible of results
- But that introduce a crash

Next Steps and progress in maturity of results

- Test and adapt the tool for others case studies
- Improvement the quality of the template by detecting type (string, integer...)
- Find a heuristic to know in advance the number of clusters
- Detection about the dependencies between the messages (increment)
- Possibility to correct the model during the fuzzing phases
- Compare with dynamic execution

Test-based classification framework for CPS

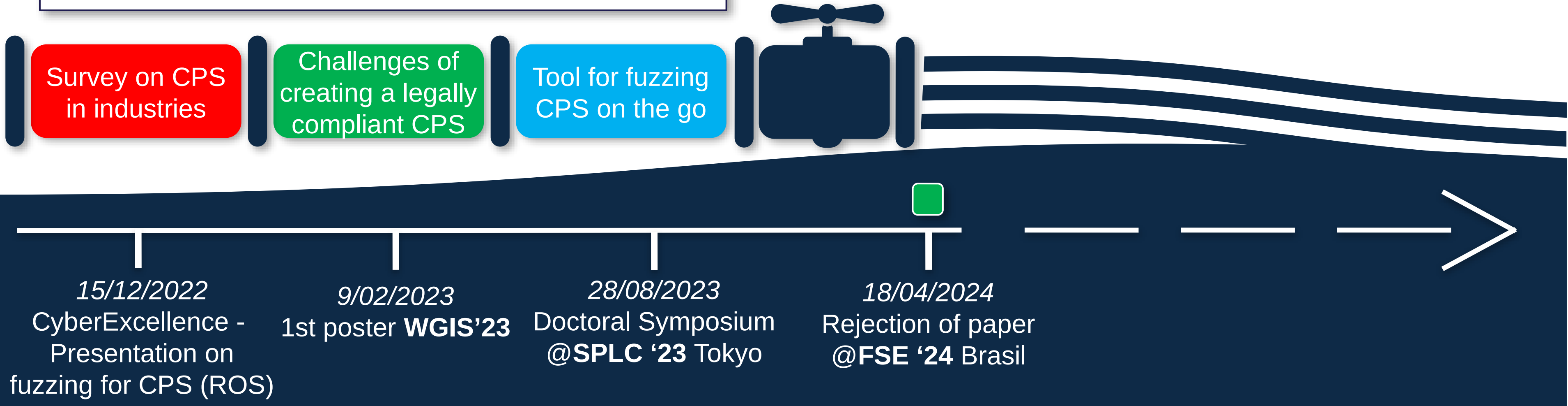
Guillaume Nguyen, UNamur

Overview

🟥 The survey is stuck due to a lack of responses from industrial actors. We hoped to get at least 25 answers and we only have 8.

🟦 We are currently designing a tool meant to be embedded in a computer which could be carried to perform onsite fuzzing. This tool would be used through a visual interface based on models and would be able to communicate on many channels.

🟩 The research intended to identify the challenges related to creating a legally compliant CPS using tests based on official EU material. However, the final contribution of the paper shifted from the creation of a matching tool for industrial actors to help them identify relevant laws and related (technical) material to a methodology critique of the current legislation landscape.



Survey on CPS in industries

Challenges of creating a legally compliant CPS

Tool for fuzzing CPS on the go

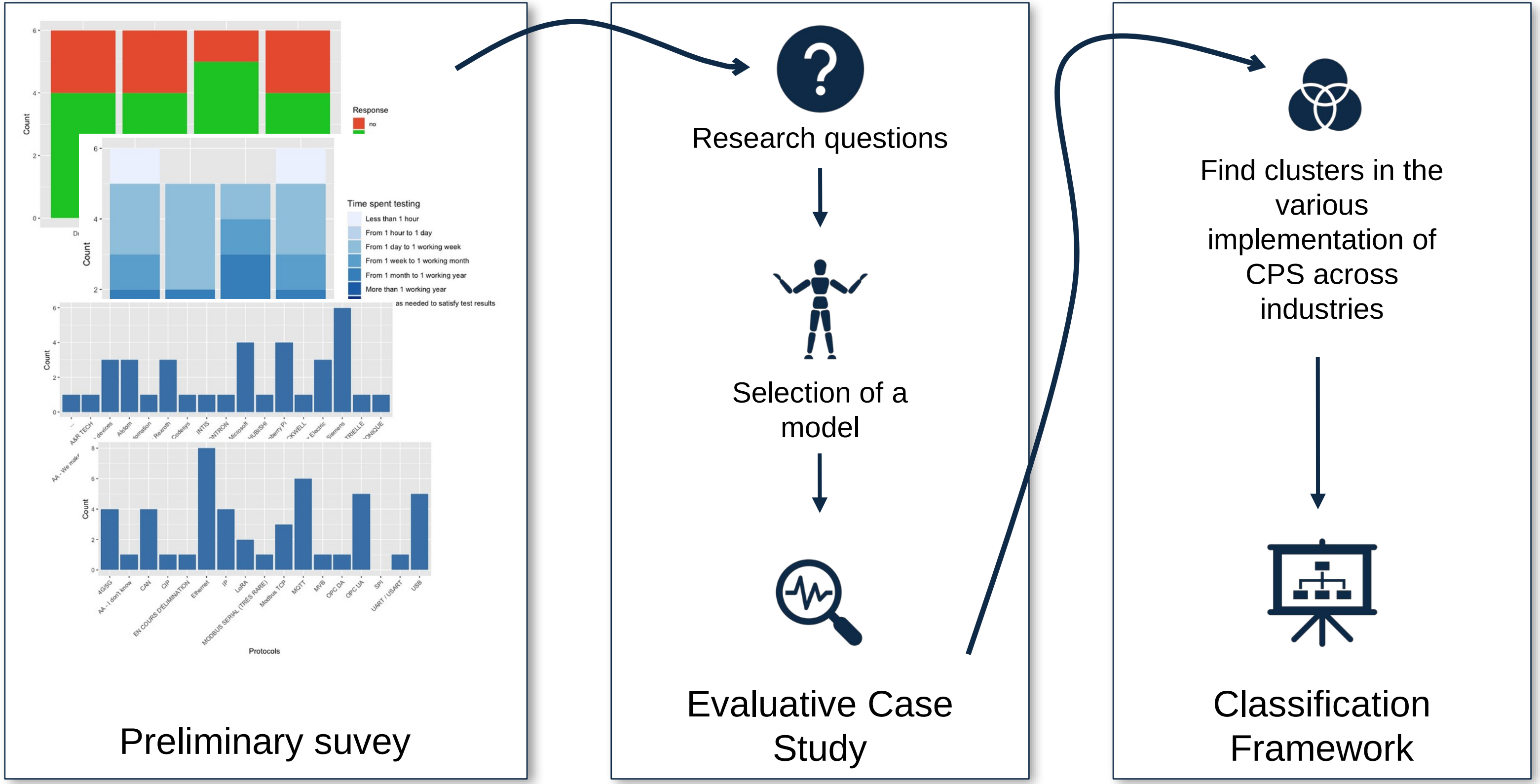
15/12/2022
CyberExcellence - Presentation on fuzzing for CPS (ROS)

9/02/2023
1st poster **WGIS'23**

28/08/2023
Doctoral Symposium @**SPLC '23** Tokyo

🟩
18/04/2024
Rejection of paper @**FSE '24** Brasil

Next steps - Survey



Preliminary suvey

Research questions

Selection of a model

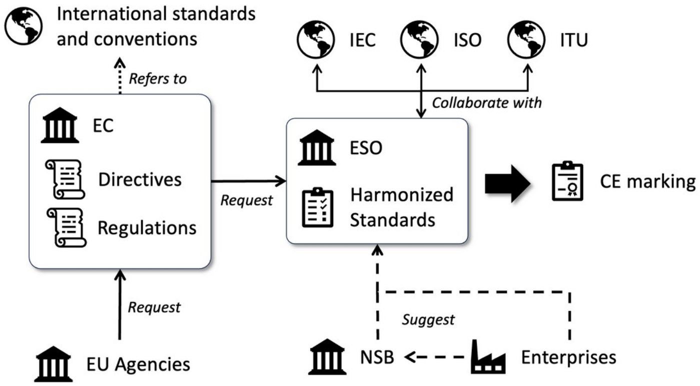
Evaluative Case Study

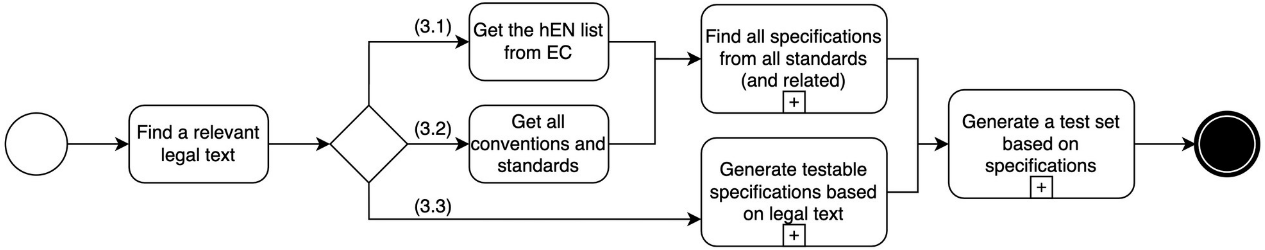
Find clusters in the various implementation of CPS across industries

Classification Framework

Next steps - Challenges of creating a legally compliant CPS

1. Finding a relevant legal text based on keywords
2. Access to cited material is not free of cost
3. Identifying the level of compliance with the legal text reached after complying with related specifications
4. Establishing the relationship to other acts based on the original one
5. Understanding technical requirements from legal texts




Find a relevant legal text
(3.1)
Get the hEN list from EC
(3.2)
Get all conventions and standards
(3.3)
Find all specifications from all standards (and related)
+
Generate testable specifications based on legal text
+
Generate a test set based on specifications
+

Automated cybersecurity testing with genetic algorithms

Denis Darquennes, Philippe Massonet

Défi 01: MUT4SEC - Test generation for CPS security with Pynguin

Groupe de travail défi 01

Denis Darquennes, Philippe Massonet, Sébastien Dupont - CETIC

digital wallonia 4.cyberwal

Wallonie recherche SPW

https://cyberwal.be
https://cyberexcellence.be

Plan

- Mut4sec - test generation for security
- The Context
- Case study: Control Center and Zone policies - infected vehicle software - **supply chain attack**
- The Pynguin test generation (white box) - how it works
- Execution of tests - spoof:
  - zone policy assignation
  - zone policy reporting
  - vehicle speed reporting
- Description of the CPS
- Next steps : the test generation for ROS

MUT4SEC - Test generation for security

Test generation is based on genetic algorithms

- filter tests for selection of most pertinent usable tests
- using the Pynguin tool

Automated test generation to highlight security vulnerabili

- in cyber physical systems (CPS) (challenge #1)
- using the ROS framework (part of the case study : the rover)

Rover case study method can be applied to other CPS (e.g.: railway systems)

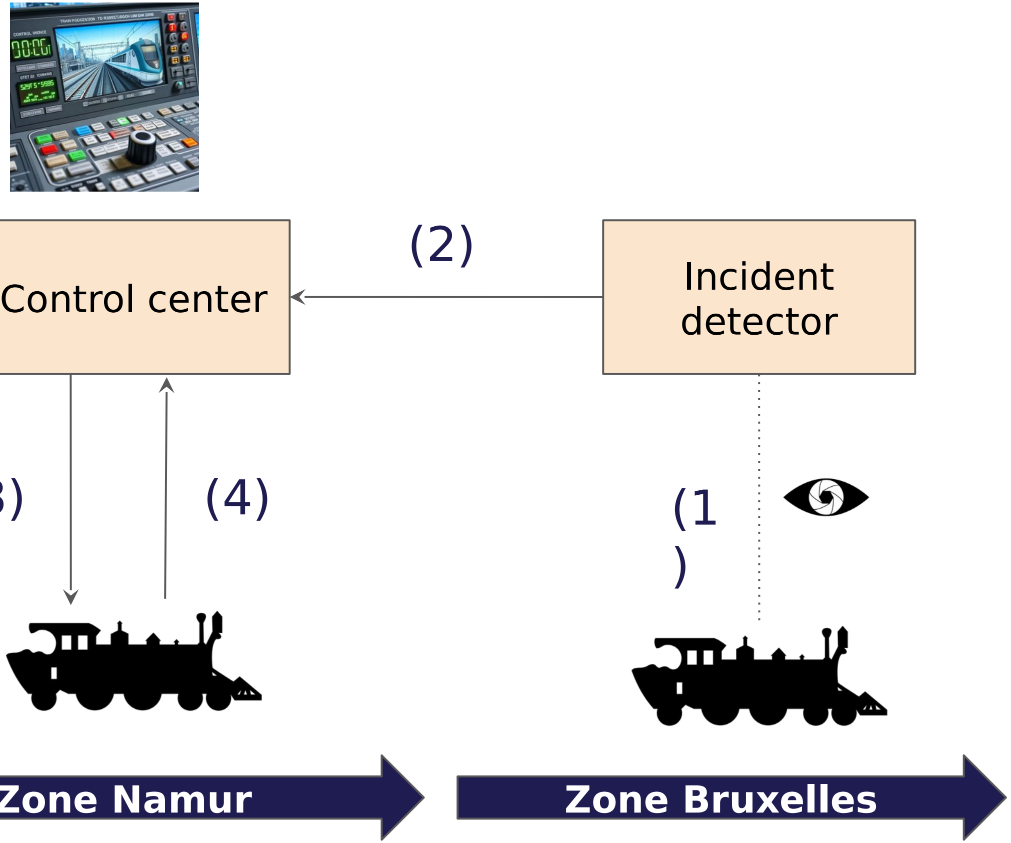
Context

Protecting railway systems

Control center supervision process:

1. Incident detected
2. Incident position
3. Adapt speed profile
4. Monitor speed / distance

→ **Add Assertions: check integrity** of vehicle controls (policy, speed)



Control center

Incident detector

(2)

(3)

(4)

(1
)

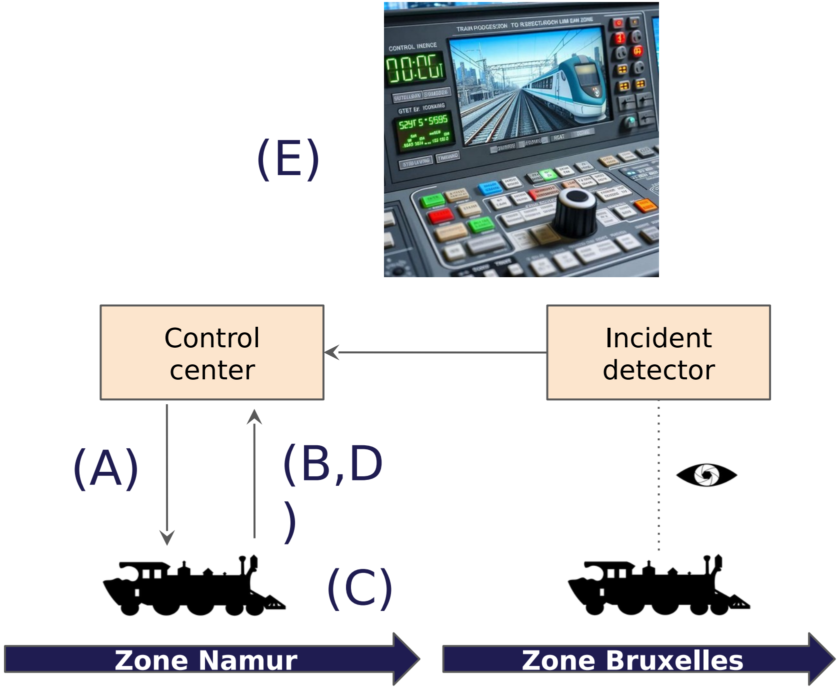
Zone Namur

Zone Bruxelles

Control Center and Zone policies

Integrity tests on threats:

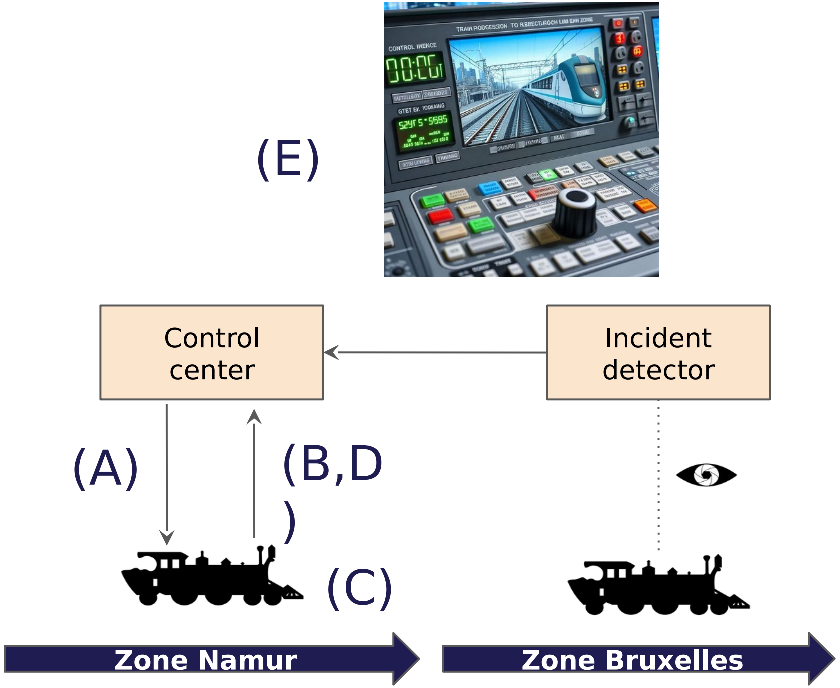
- (A) Integrity: zone policy sent is the one received
- (B) Zone policy is respected
- (C) Integrity: monitored data corresponds to real data
- (D) Integrity: monitoring data sent is the same that is received
- (E) Monitoring data displayed is the same



Control Center and Zone policies

Integrity tests on threats:

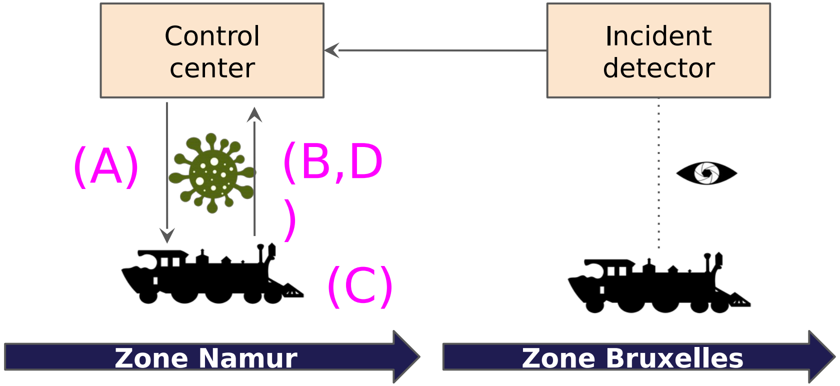
- (A) Integrity: zone policy sent is the one received => **attack on registered zone_policy**
- (B) Zone policy is respected => **attack on communication of speed_policy**
- (C) Integrity: monitored data corresponds to real data => **attack on effective speed**
- (D) Integrity: monitoring data sent is the same that is received => **equivalent to (B)**
- (E) Monitoring data displayed



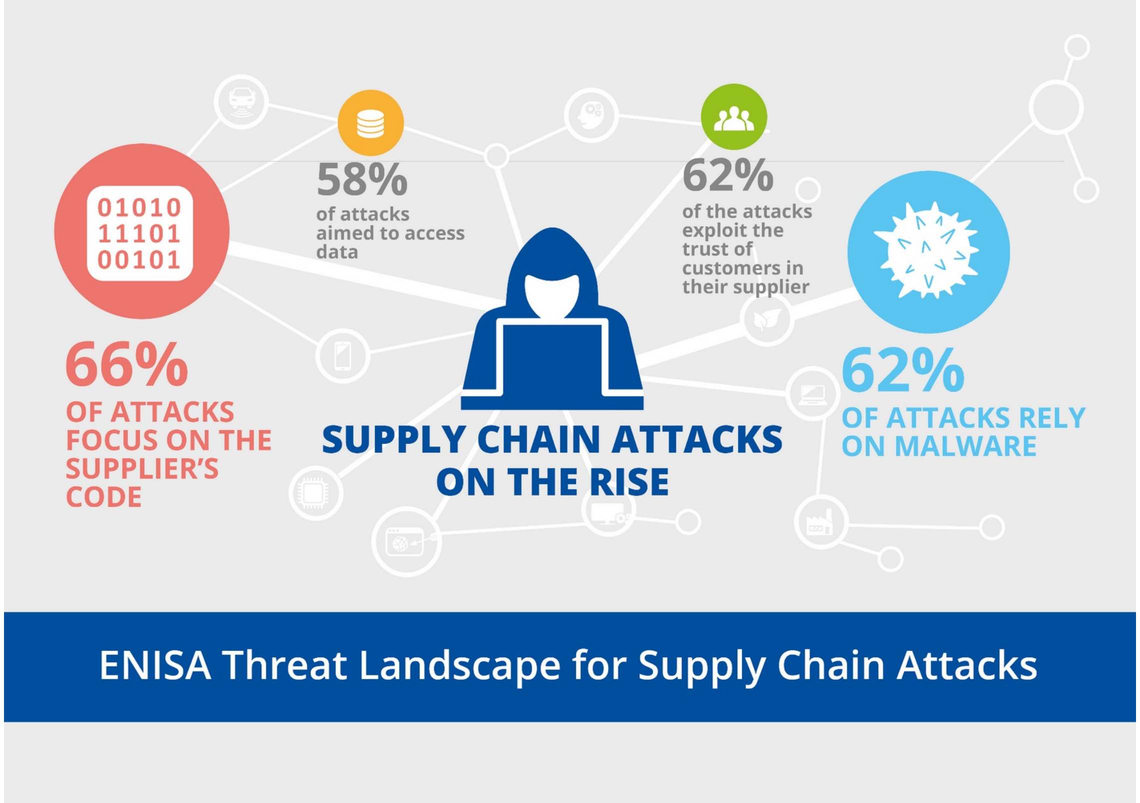
Attack: infected vehicle software

Protecting railway systems against

- Infection through **supply chain attack**
  - e.g. Usage of untrustworthy 3rd party software
- Generating three attacks on the train:
  - (A) spoof zone_policy assignation
  - (B,D) spoof zone_policy reporting
  - (C) spoof vehicle_speed reporting



Supply chain attacks - #1 threat in 2030



58% of attacks aimed to access data

62% of the attacks exploit the trust of customers in their supplier

66% OF ATTACKS FOCUS ON THE SUPPLIER'S CODE

SUPPLY CHAIN ATTACKS ON THE RISE

62% OF ATTACKS RELY ON MALWARE

ENISA Threat Landscape for Supply Chain Attacks

ENISA - Threat Landscape for Supply Chain Attacks (2021)

European Cyber Resilience Act – European Parliament briefing (2022)

Using 3rd parties … When things go wrong…

SolarWinds Supply Chain Attack (2020)

SolarWinds, a company that provides IT management and monitoring software, suffered a cyberattack where attackers compromised its software development process.

The attackers inserted a backdoor into SolarWinds' Orion software during the development phase. This compromised software was then distributed to SolarWinds' customers, including government agencies, critical infrastructure entities, and businesses in various countries. Attackers were stayed undetected for at least 6 months, and maybe up to 14 months

Pynguin - Automated Unit Test Generation

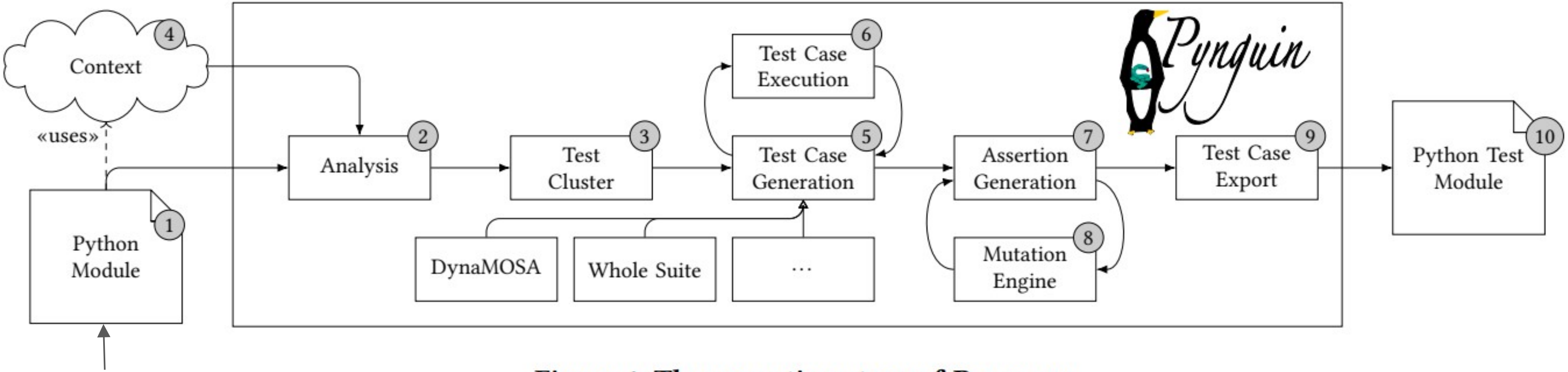


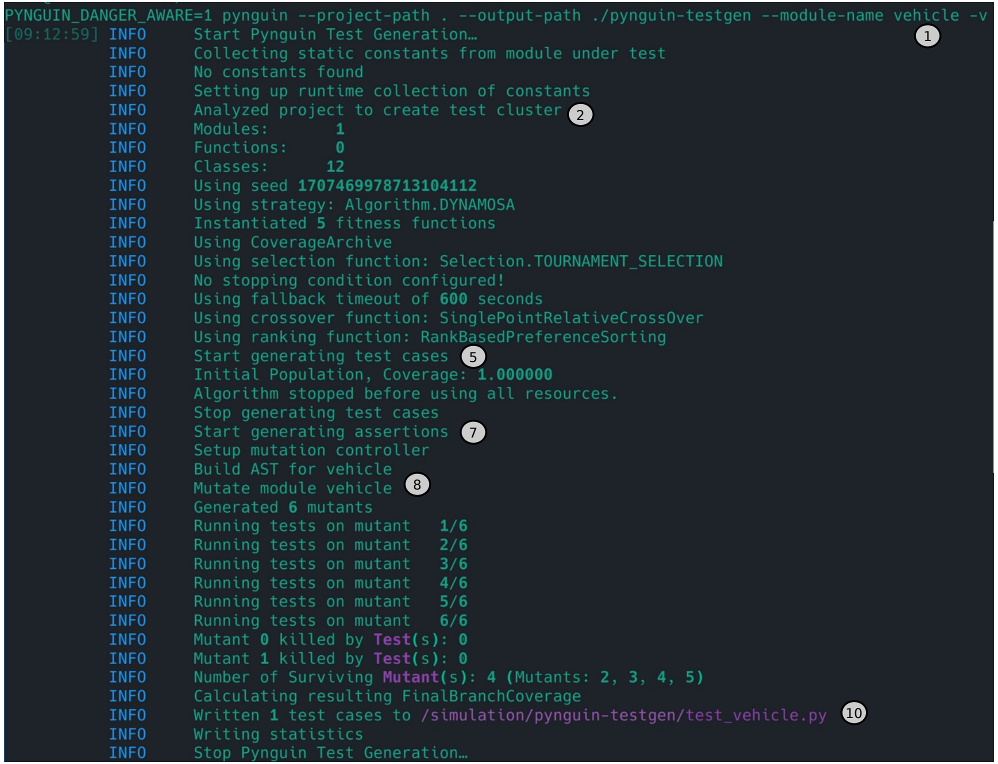
Figure 1: The execution steps of Pynguin.

Add (integrity) assertions

Tester

S. Lukasczyk and G. Fraser, "Pynguin: Automated Unit Test Generation for Python," 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Pittsburgh, PA, USA, 2022, pp. 168-172, doi: 10.1145/3510454.3516829.

Test generation - Run Pynguin on Vehicle



```
PYNGUIN_DANGER_AWARE=1 pynguin --project-path . --output-path ./pynguin-testgen --module-name vehicle -v
[09:12:59] INFO     Start Pynguin Test Generation…                                      ①
           INFO     Collecting static constants from module under test
           INFO     No constants found
           INFO     Setting up runtime collection of constants
           INFO     Analyzed project to create test cluster ②
           INFO     Modules:       1
           INFO     Functions:     0
           INFO     Classes:      12
           INFO     Using seed 1707469978713104112
           INFO     Using strategy: Algorithm.DYNAMOSA
           INFO     Instantiated 5 fitness functions
           INFO     Using CoverageArchive
           INFO     Using selection function: Selection.TOURNAMENT_SELECTION
           INFO     No stopping condition configured!
           INFO     Using fallback timeout of 600 seconds
           INFO     Using crossover function: SinglePointRelativeCrossOver
           INFO     Using ranking function: RankBasedPreferenceSorting
           INFO     Start generating test cases ⑤
           INFO     Initial Population, Coverage: 1.000000
           INFO     Algorithm stopped before using all resources.
           INFO     Stop generating test cases
           INFO     Start generating assertions ⑦
           INFO     Setup mutation controller
           INFO     Build AST for vehicle
           INFO     Mutate module vehicle ⑧
           INFO     Generated 6 mutants
           INFO     Running tests on mutant   1/6
           INFO     Running tests on mutant   2/6
           INFO     Running tests on mutant   3/6
           INFO     Running tests on mutant   4/6
           INFO     Running tests on mutant   5/6
           INFO     Running tests on mutant   6/6
           INFO     Mutant 0 killed by Test(s): 0
           INFO     Mutant 1 killed by Test(s): 0
           INFO     Number of Surviving Mutant(s): 4 (Mutants: 2, 3, 4, 5)
           INFO     Calculating resulting FinalBranchCoverage
           INFO     Written 1 test cases to /simulation/pynguin-testgen/test_vehicle.py ⑩
           INFO     Writing statistics
           INFO     Stop Pynguin Test Generation…
```

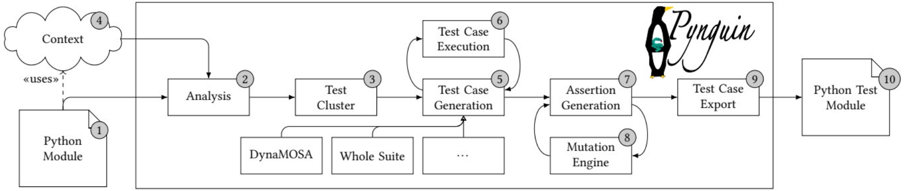
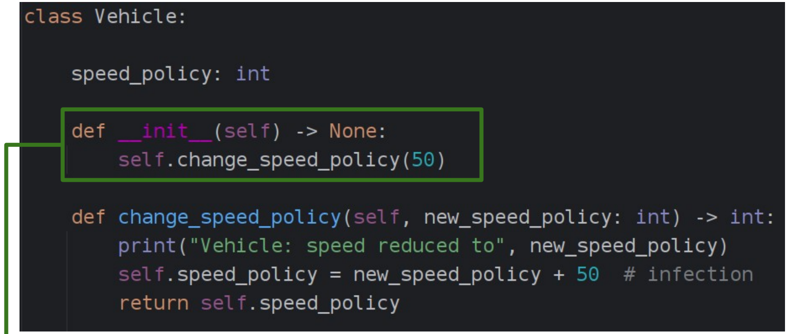
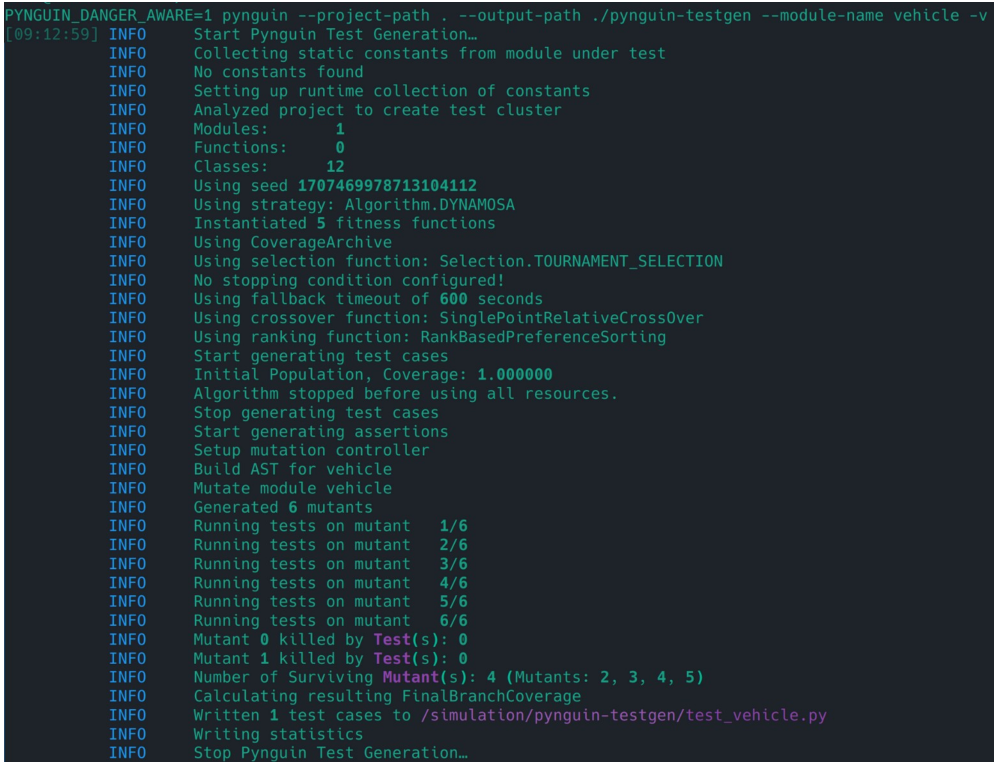


Figure 1: The execution steps of Pynguin.

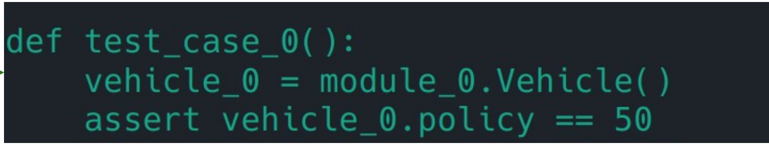
Test generation - Run Pynguin on Vehicle



```
class Vehicle:

    speed_policy: int

    def __init__(self) -> None:
        self.change_speed_policy(50)

    def change_speed_policy(self, new_speed_policy: int) -> int:
        print("Vehicle: speed reduced to", new_speed_policy)
        self.speed_policy = new_speed_policy + 50  # infection
        return self.speed_policy
```



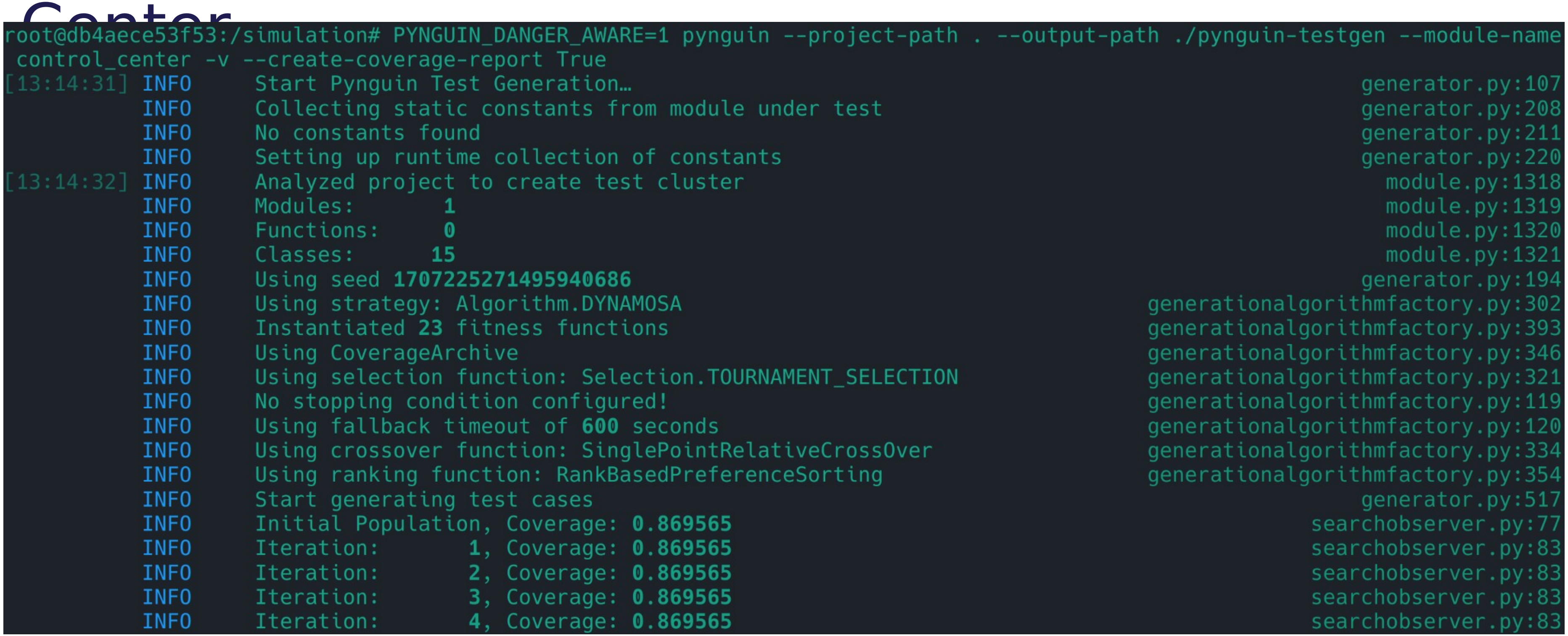
```
PYNGUIN_DANGER_AWARE=1 pynguin --project-path . --output-path ./pynguin-testgen --module-name vehicle -v
[09:12:59] INFO     Start Pynguin Test Generation…
           INFO     Collecting static constants from module under test
           INFO     No constants found
           INFO     Setting up runtime collection of constants
           INFO     Analyzed project to create test cluster
           INFO     Modules:       1
           INFO     Functions:     0
           INFO     Classes:      12
           INFO     Using seed 1707469978713104112
           INFO     Using strategy: Algorithm.DYNAMOSA
           INFO     Instantiated 5 fitness functions
           INFO     Using CoverageArchive
           INFO     Using selection function: Selection.TOURNAMENT_SELECTION
           INFO     No stopping condition configured!
           INFO     Using fallback timeout of 600 seconds
           INFO     Using crossover function: SinglePointRelativeCrossOver
           INFO     Using ranking function: RankBasedPreferenceSorting
           INFO     Start generating test cases
           INFO     Initial Population, Coverage: 1.000000
           INFO     Algorithm stopped before using all resources.
           INFO     Stop generating test cases
           INFO     Start generating assertions
           INFO     Setup mutation controller
           INFO     Build AST for vehicle
           INFO     Mutate module vehicle
           INFO     Generated 6 mutants
           INFO     Running tests on mutant   1/6
           INFO     Running tests on mutant   2/6
           INFO     Running tests on mutant   3/6
           INFO     Running tests on mutant   4/6
           INFO     Running tests on mutant   5/6
           INFO     Running tests on mutant   6/6
           INFO     Mutant 0 killed by Test(s): 0
           INFO     Mutant 1 killed by Test(s): 0
           INFO     Number of Surviving Mutant(s): 4 (Mutants: 2, 3, 4, 5)
           INFO     Calculating resulting FinalBranchCoverage
           INFO     Written 1 test cases to /simulation/pynguin-testgen/test_vehicle.py
           INFO     Writing statistics
           INFO     Stop Pynguin Test Generation…
```



```
def test_case_0():
    vehicle_0 = module_0.Vehicle()
    assert vehicle_0.policy == 50
```

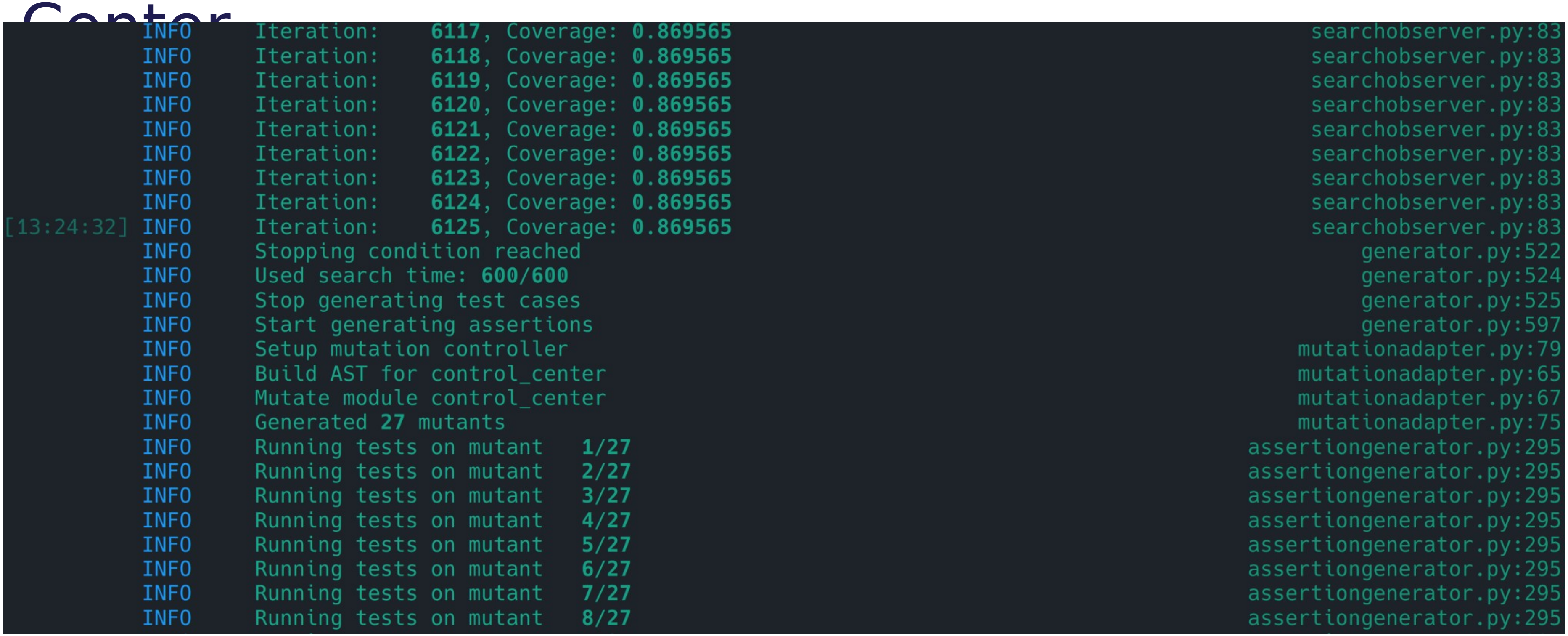
generated by Pynguin

Test generation - Run Pynguin on Control Center



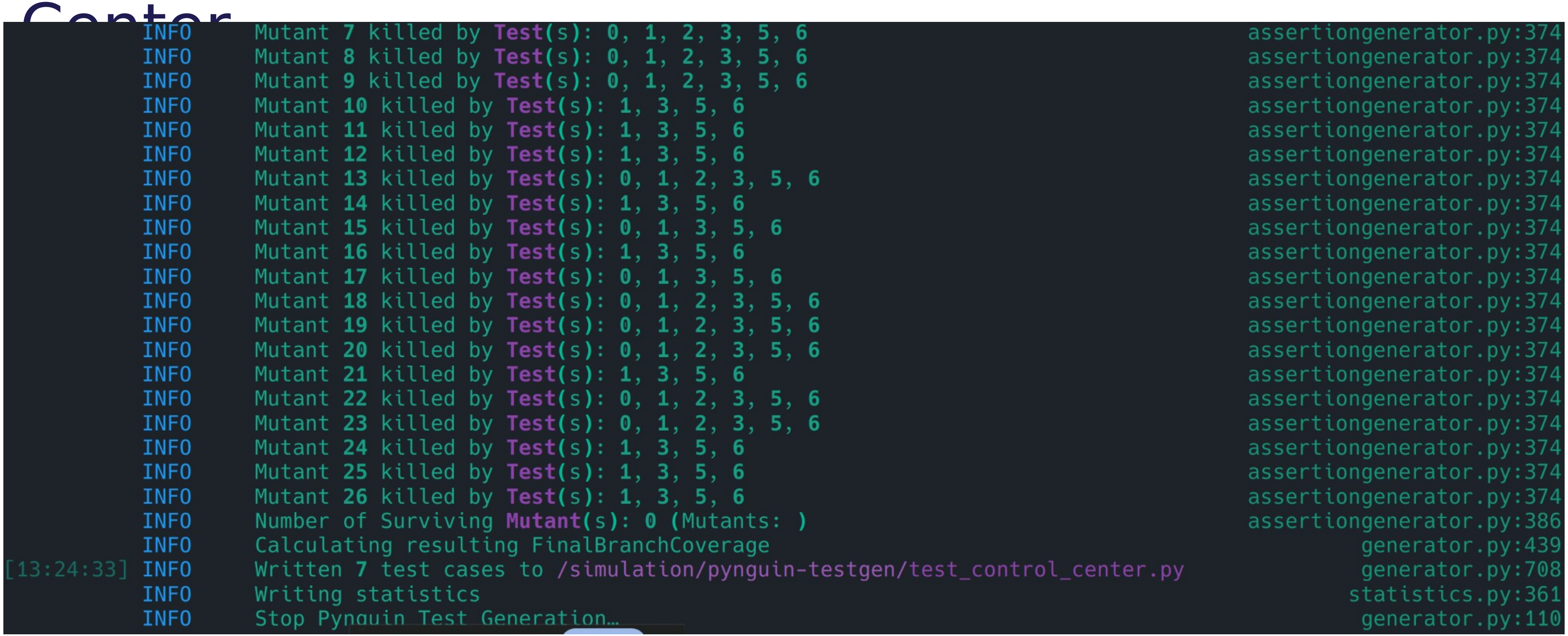
```
root@db4aece53f53:/simulation# PYNGUIN_DANGER_AWARE=1 pynguin --project-path . --output-path ./pynguin-testgen --module-name
 control_center -v --create-coverage-report True
[13:14:31] INFO     Start Pynguin Test Generation…                                                      generator.py:107
           INFO     Collecting static constants from module under test                                  generator.py:208
           INFO     No constants found                                                                  generator.py:211
           INFO     Setting up runtime collection of constants                                          generator.py:220
[13:14:32] INFO     Analyzed project to create test cluster                                               module.py:1318
           INFO     Modules:       1                                                                      module.py:1319
           INFO     Functions:     0                                                                      module.py:1320
           INFO     Classes:      15                                                                      module.py:1321
           INFO     Using seed 1707225271495940686                                                      generator.py:194
           INFO     Using strategy: Algorithm.DYNAMOSA                                generationalgorithmfactory.py:302
           INFO     Instantiated 23 fitness functions                                 generationalgorithmfactory.py:393
           INFO     Using CoverageArchive                                             generationalgorithmfactory.py:346
           INFO     Using selection function: Selection.TOURNAMENT_SELECTION          generationalgorithmfactory.py:321
           INFO     No stopping condition configured!                                 generationalgorithmfactory.py:119
           INFO     Using fallback timeout of 600 seconds                             generationalgorithmfactory.py:120
           INFO     Using crossover function: SinglePointRelativeCrossOver            generationalgorithmfactory.py:334
           INFO     Using ranking function: RankBasedPreferenceSorting                generationalgorithmfactory.py:354
           INFO     Start generating test cases                                                         generator.py:517
           INFO     Initial Population, Coverage: 0.869565                                          searchobserver.py:77
           INFO     Iteration:         1, Coverage: 0.869565                                          searchobserver.py:83
           INFO     Iteration:         2, Coverage: 0.869565                                          searchobserver.py:83
           INFO     Iteration:         3, Coverage: 0.869565                                          searchobserver.py:83
           INFO     Iteration:         4, Coverage: 0.869565                                          searchobserver.py:83
```

Test generation - Run Pynguin on Control Center



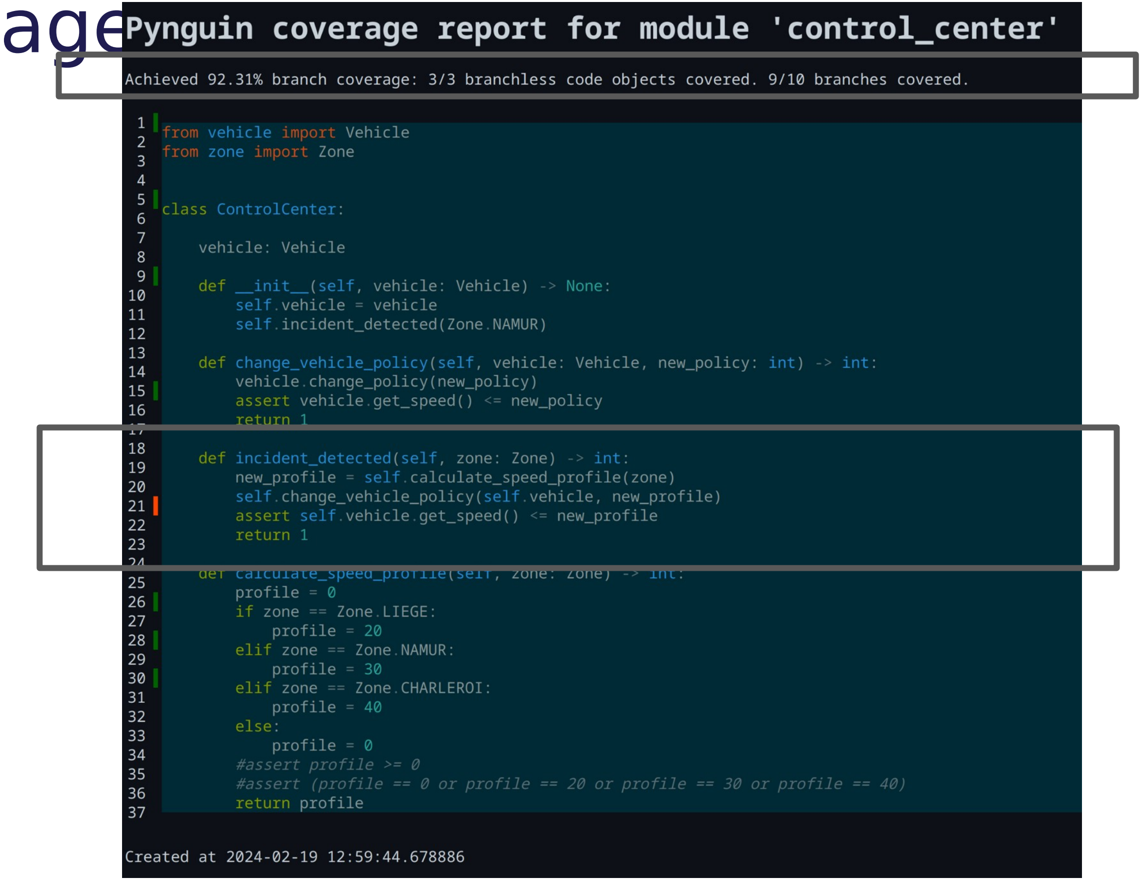
```
           INFO     Iteration:    6117, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6118, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6119, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6120, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6121, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6122, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6123, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Iteration:    6124, Coverage: 0.869565                                                      searchobserver.py:83
[13:24:32] INFO     Iteration:    6125, Coverage: 0.869565                                                      searchobserver.py:83
           INFO     Stopping condition reached                                                                      generator.py:522
           INFO     Used search time: 600/600                                                                       generator.py:524
           INFO     Stop generating test cases                                                                      generator.py:525
           INFO     Start generating assertions                                                                     generator.py:597
           INFO     Setup mutation controller                                                                   mutationadapter.py:79
           INFO     Build AST for control_center                                                                mutationadapter.py:65
           INFO     Mutate module control_center                                                                mutationadapter.py:67
           INFO     Generated 27 mutants                                                                        mutationadapter.py:75
           INFO     Running tests on mutant   1/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   2/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   3/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   4/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   5/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   6/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   7/27                                                          assertiongenerator.py:295
           INFO     Running tests on mutant   8/27                                                          assertiongenerator.py:295
```

Test generation - Run Pynguin on Control Center



```
           INFO     Mutant 7 killed by Test(s): 0, 1, 2, 3, 5, 6                         assertiongenerator.py:374
           INFO     Mutant 8 killed by Test(s): 0, 1, 2, 3, 5, 6                         assertiongenerator.py:374
           INFO     Mutant 9 killed by Test(s): 0, 1, 2, 3, 5, 6                         assertiongenerator.py:374
           INFO     Mutant 10 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 11 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 12 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 13 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 14 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 15 killed by Test(s): 0, 1, 3, 5, 6                           assertiongenerator.py:374
           INFO     Mutant 16 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 17 killed by Test(s): 0, 1, 3, 5, 6                           assertiongenerator.py:374
           INFO     Mutant 18 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 19 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 20 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 21 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 22 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 23 killed by Test(s): 0, 1, 2, 3, 5, 6                        assertiongenerator.py:374
           INFO     Mutant 24 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 25 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Mutant 26 killed by Test(s): 1, 3, 5, 6                              assertiongenerator.py:374
           INFO     Number of Surviving Mutant(s): 0 (Mutants: )                         assertiongenerator.py:386
           INFO     Calculating resulting FinalBranchCoverage                                     generator.py:439
[13:24:33] INFO     Written 7 test cases to /simulation/pynguin-testgen/test_control_center.py    generator.py:708
           INFO     Writing statistics                                                           statistics.py:361
           INFO     Stop Pynguin Test Generation…                                                 generator.py:110
```

Test generation - Pynguin output - Coverage



Pynguin coverage report for module 'control_center'

Achieved 92.31% branch coverage: 3/3 branchless code objects covered. 9/10 branches covered.

```
from vehicle import Vehicle
from zone import Zone


class ControlCenter:

    vehicle: Vehicle

    def __init__(self, vehicle: Vehicle) -> None:
        self.vehicle = vehicle
        self.incident_detected(Zone.NAMUR)

    def change_vehicle_policy(self, vehicle: Vehicle, new_policy: int) -> int:
        vehicle.change_policy(new_policy)
        assert vehicle.get_speed() <= new_policy
        return 1

    def incident_detected(self, zone: Zone) -> int:
        new_profile = self.calculate_speed_profile(zone)
        self.change_vehicle_policy(self.vehicle, new_profile)
        assert self.vehicle.get_speed() <= new_profile
        return 1

    def calculate_speed_profile(self, zone: Zone) -> int:
        profile = 0
        if zone == Zone.LIEGE:
            profile = 20
        elif zone == Zone.NAMUR:
            profile = 30
        elif zone == Zone.CHARLEROI:
            profile = 40
        else:
            profile = 0
        #assert profile >= 0
        #assert (profile == 0 or profile == 20 or profile == 30 or profile == 40)
        return profile
```

Created at 2024-02-19 12:59:44.678886

Branch coverage

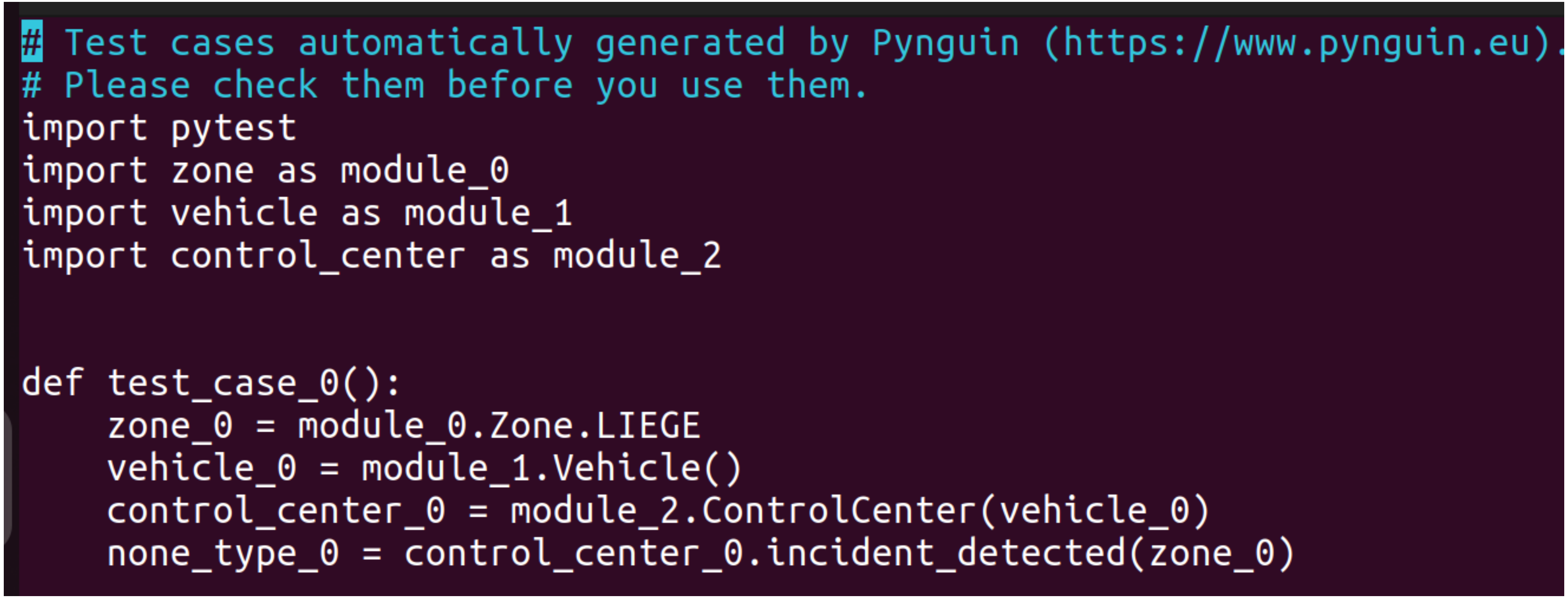
assertion added by tester

vertical orange line: not covered by any generated test

vertical green line: covered, test generated for it

Test generation - Pynguin output

Example list of 5 generated tests - tests usable … or not !

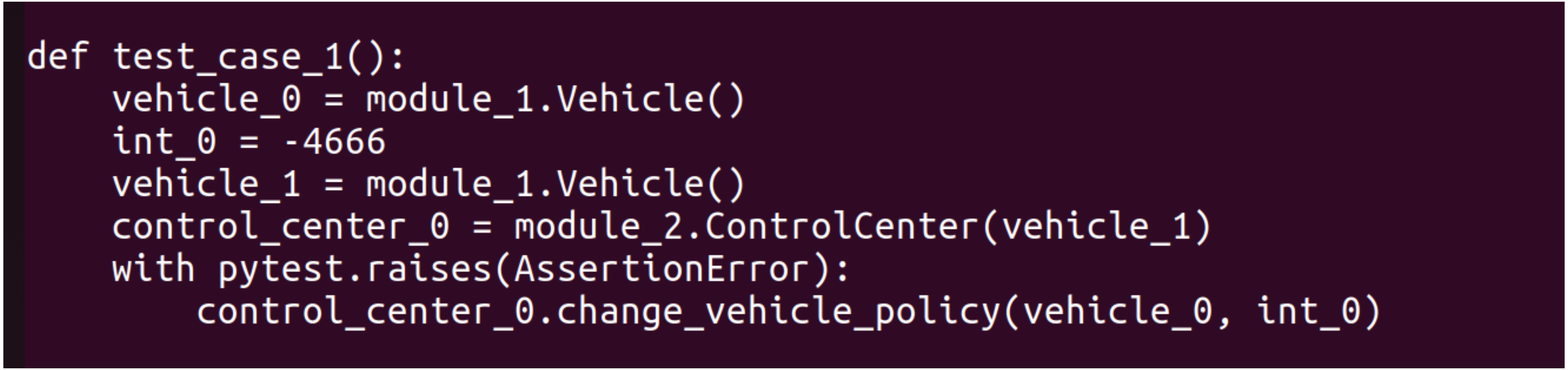


```
# Test cases automatically generated by Pynguin (https://www.pynguin.eu).
# Please check them before you use them.
import pytest
import zone as module_0
import vehicle as module_1
import control_center as module_2


def test_case_0():
    zone_0 = module_0.Zone.LIEGE
    vehicle_0 = module_1.Vehicle()
    control_center_0 = module_2.ControlCenter(vehicle_0)
    none_type_0 = control_center_0.incident_detected(zone_0)
```

Test not useful because there is no assertion generated

Test generation - Pynguin output

List of generated tests

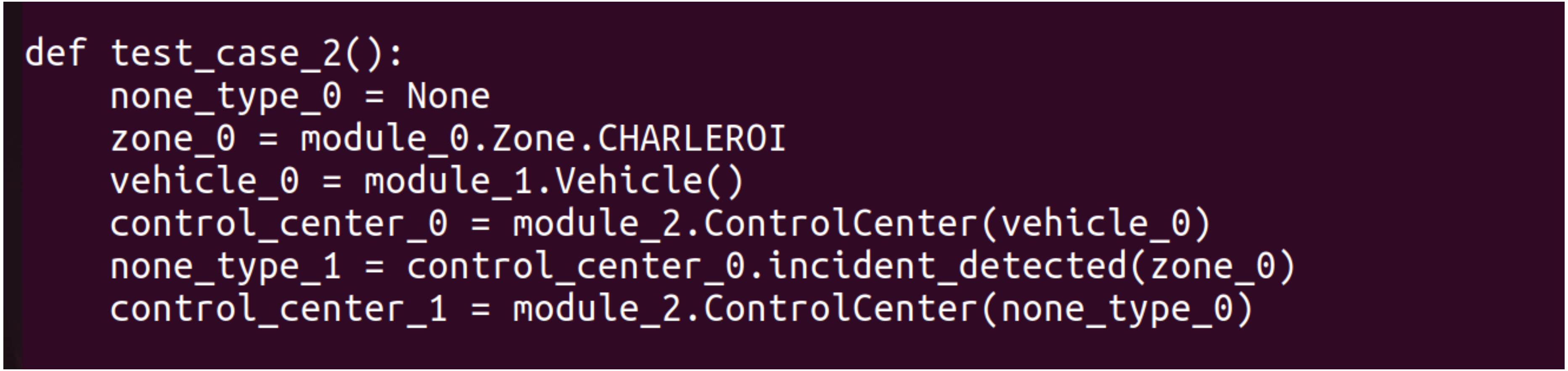


```
def test_case_1():
    vehicle_0 = module_1.Vehicle()
    int_0 = -4666
    vehicle_1 = module_1.Vehicle()
    control_center_0 = module_2.ControlCenter(vehicle_1)
    with pytest.raises(AssertionError):
        control_center_0.change_vehicle_policy(vehicle_0, int_0)
```

Test useful but vehicle 1 line can be deleted

Test generation - Pynguin output

List of generated tests

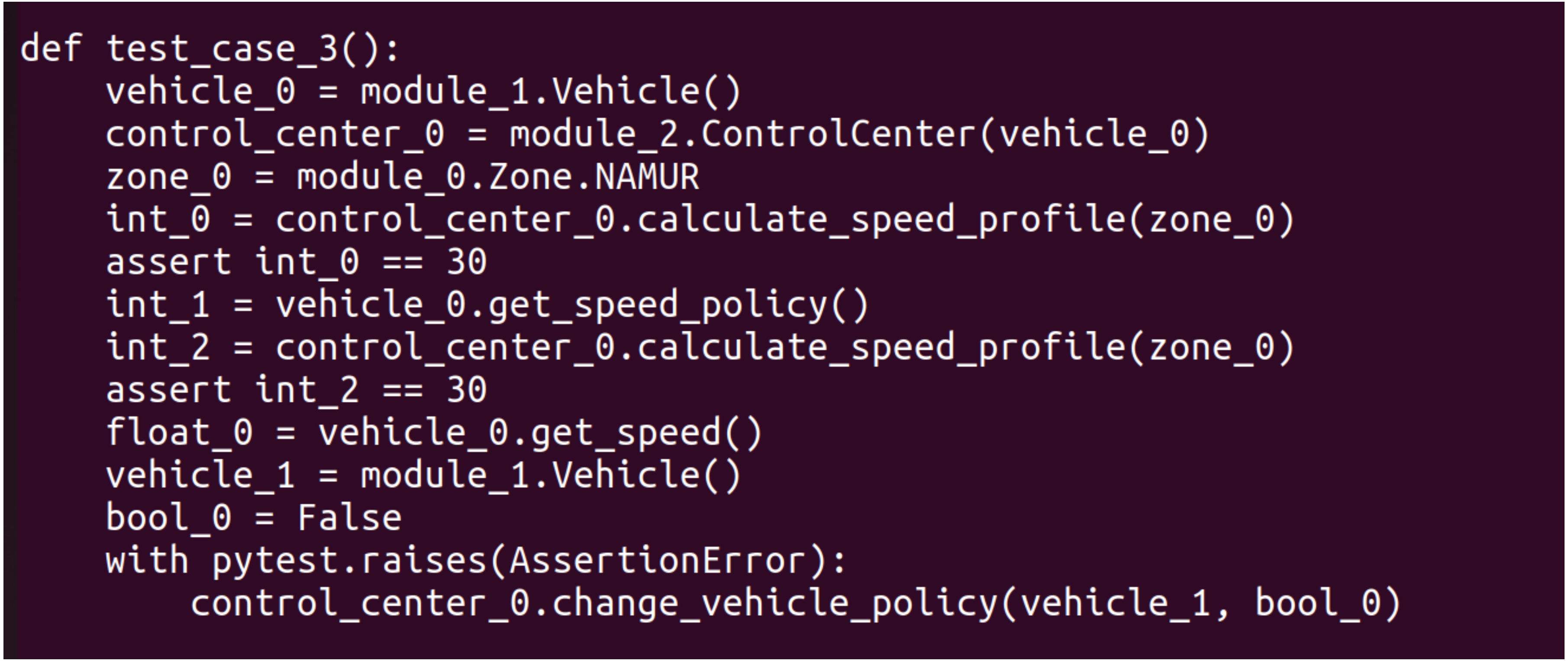


```
def test_case_2():
    none_type_0 = None
    zone_0 = module_0.Zone.CHARLEROI
    vehicle_0 = module_1.Vehicle()
    control_center_0 = module_2.ControlCenter(vehicle_0)
    none_type_1 = control_center_0.incident_detected(zone_0)
    control_center_1 = module_2.ControlCenter(none_type_0)
```

Test not useful because the incident that could be tested (zone charleroi) is not referenced

Test generation - Pynguin output

List of generated tests

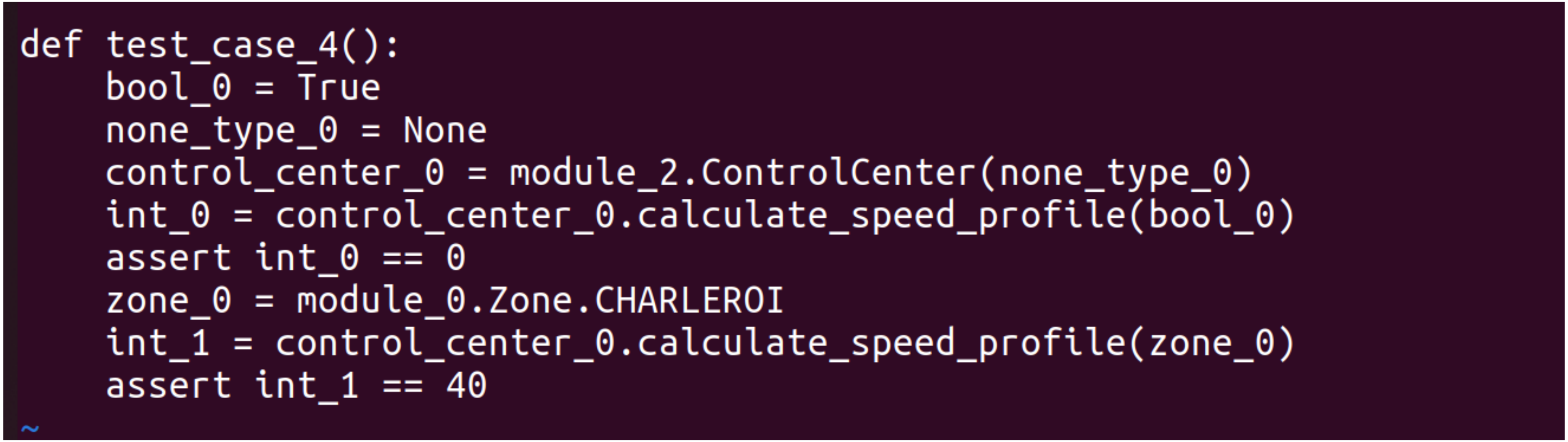


```
def test_case_3():
    vehicle_0 = module_1.Vehicle()
    control_center_0 = module_2.ControlCenter(vehicle_0)
    zone_0 = module_0.Zone.NAMUR
    int_0 = control_center_0.calculate_speed_profile(zone_0)
    assert int_0 == 30
    int_1 = vehicle_0.get_speed_policy()
    int_2 = control_center_0.calculate_speed_profile(zone_0)
    assert int_2 == 30
    float_0 = vehicle_0.get_speed()
    vehicle_1 = module_1.Vehicle()
    bool_0 = False
    with pytest.raises(AssertionError):
        control_center_0.change_vehicle_policy(vehicle_1, bool_0)
```

Test useful but tests two times the same speed (30). Could be simplified.
4 last lines are not useful and could be deleted. Code level reasoning.

Test generation - Pynguin output

List of generated tests

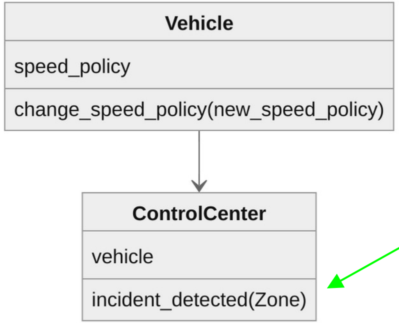


```
def test_case_4():
    bool_0 = True
    none_type_0 = None
    control_center_0 = module_2.ControlCenter(none_type_0)
    int_0 = control_center_0.calculate_speed_profile(bool_0)
    assert int_0 == 0
    zone_0 = module_0.Zone.CHARLEROI
    int_1 = control_center_0.calculate_speed_profile(zone_0)
    assert int_1 == 40
~
```

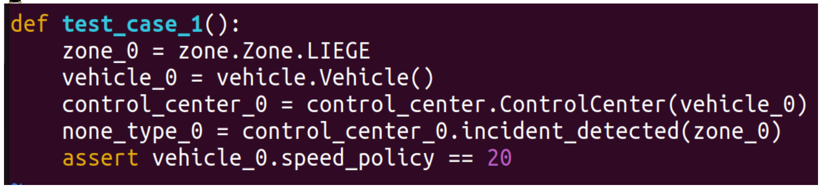
Test useful but some lines are not useful.

Execution of tests - vehicle non infected

Apply following test to control center code:

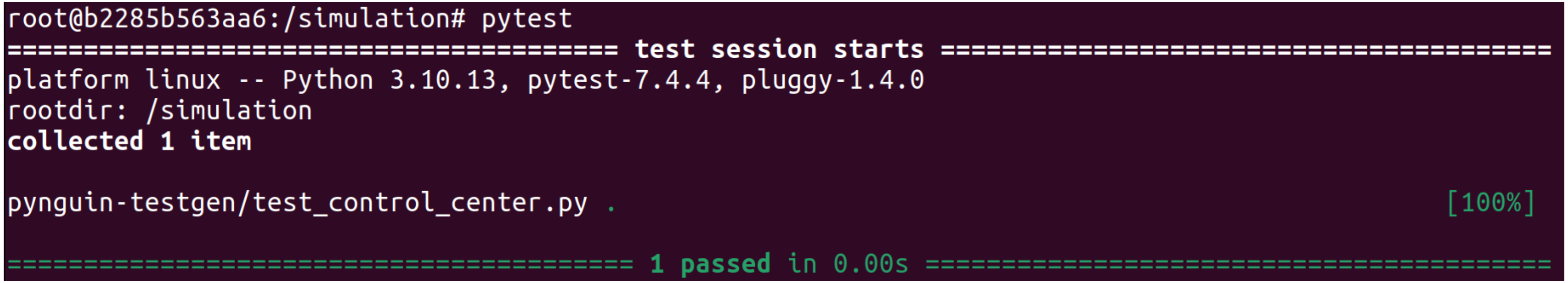


Pynguin



```
def test_case_1():
    zone_0 = zone.Zone.LIEGE
    vehicle_0 = vehicle.Vehicle()
    control_center_0 = control_center.ControlCenter(vehicle_0)
    none_type_0 = control_center_0.incident_detected(zone_0)
    assert vehicle_0.speed_policy == 20
```

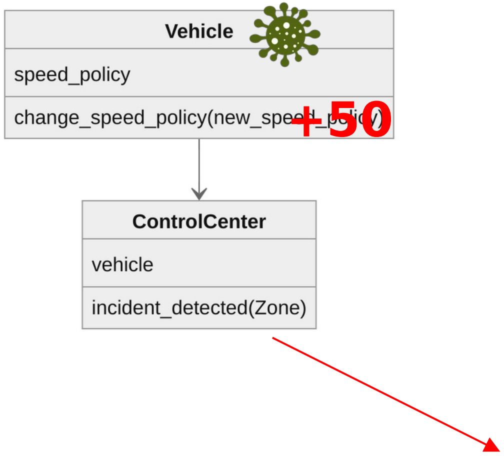
pytest

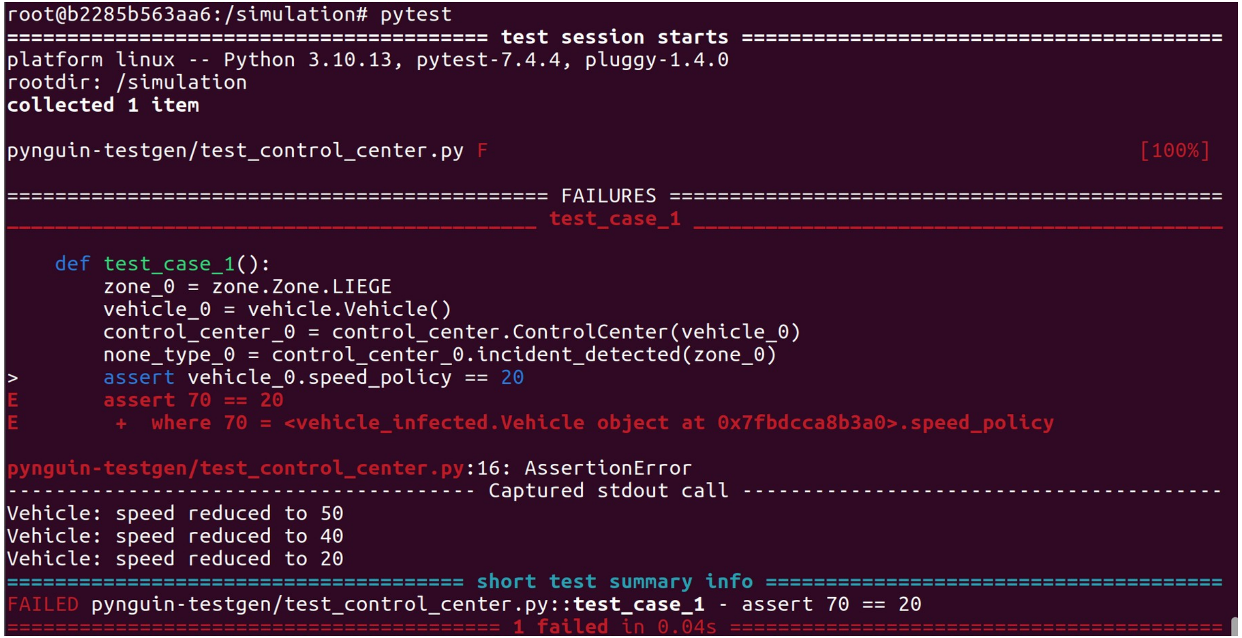


```
root@b2285b563aa6:/simulation# pytest
==================================== test session starts ====================================
platform linux -- Python 3.10.13, pytest-7.4.4, pluggy-1.4.0
rootdir: /simulation
collected 1 item

pynguin-testgen/test_control_center.py .                                              [100%]

===================================== 1 passed in 0.00s =====================================
```

Execution of tests - vehicle infected

Apply following test to control_center code:



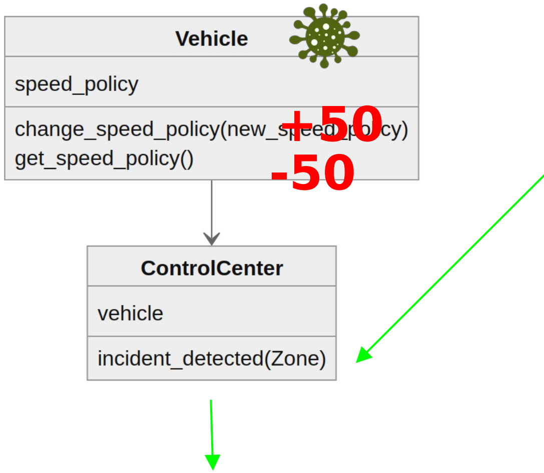


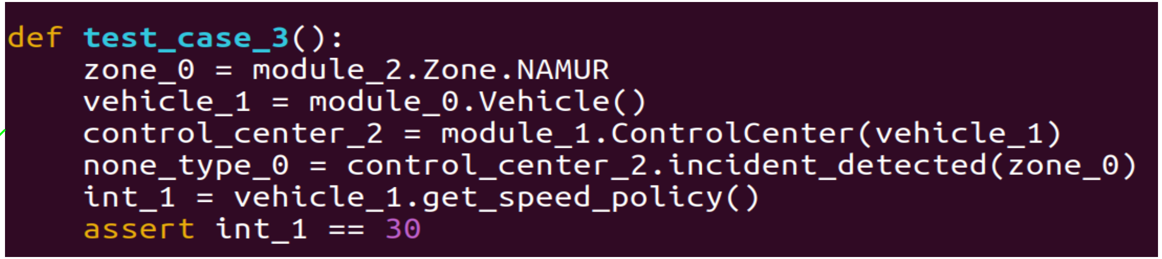
```
root@b2285b563aa6:/simulation# pytest
======================================== test session starts =========================================
platform linux -- Python 3.10.13, pytest-7.4.4, pluggy-1.4.0
rootdir: /simulation
collected 1 item

pynguin-testgen/test_control_center.py F                                                        [100%]

============================================= FAILURES ==============================================
____________________________________________ test_case_1 ____________________________________________

    def test_case_1():
        zone_0 = zone.Zone.LIEGE
        vehicle_0 = vehicle.Vehicle()
        control_center_0 = control_center.ControlCenter(vehicle_0)
        none_type_0 = control_center_0.incident_detected(zone_0)
>       assert vehicle_0.speed_policy == 20
E       assert 70 == 20
E        +  where 70 = <vehicle_infected.Vehicle object at 0x7fbdcca8b3a0>.speed_policy

pynguin-testgen/test_control_center.py:16: AssertionError
--------------------------------------- Captured stdout call ----------------------------------------
Vehicle: speed reduced to 50
Vehicle: speed reduced to 40
Vehicle: speed reduced to 20
====================================== short test summary info ======================================
FAILED pynguin-testgen/test_control_center.py::test_case_1 - assert 70 == 20
========================================= 1 failed in 0.04s ==========================================
```

Infected vehicle software

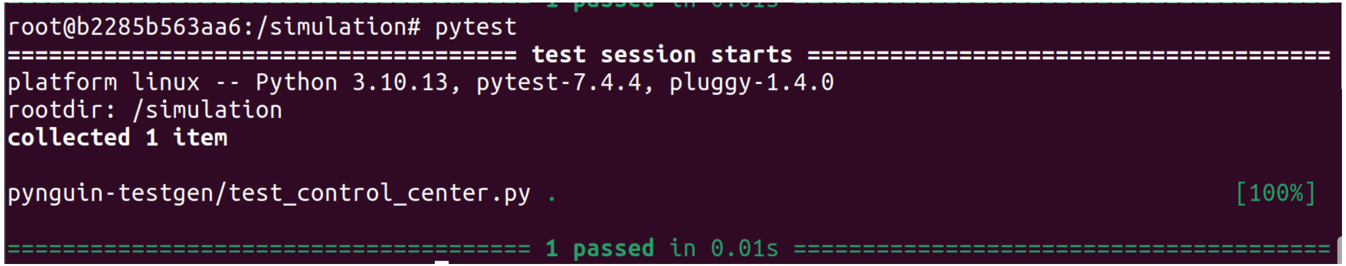
Intermediate attack:

- magnify the vehicle speed policy change,
- **spoof the speed policy readings for the control center.**





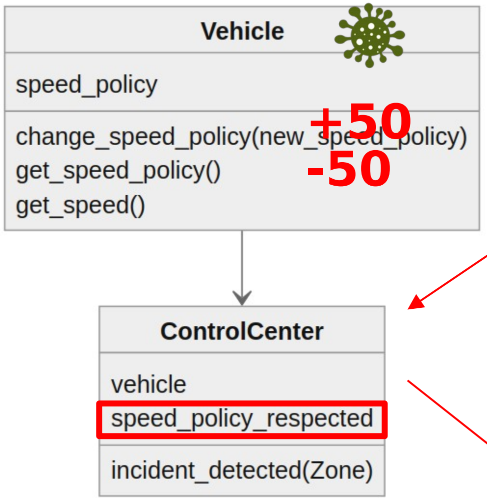
```python
def test_case_3():
    zone_0 = module_2.Zone.NAMUR
    vehicle_1 = module_0.Vehicle()
    control_center_2 = module_1.ControlCenter(vehicle_1)
    none_type_0 = control_center_2.incident_detected(zone_0)
    int_1 = vehicle_1.get_speed_policy()
    assert int_1 == 30
```

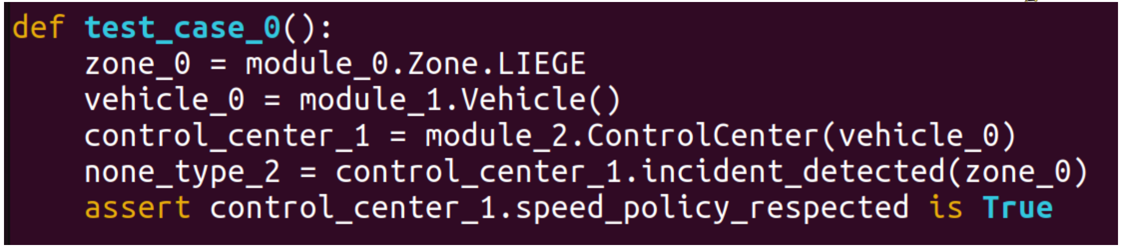


```
===================================== 1 passed in 0.01s ======================================
root@b2285b563aa6:/simulation# pytest
==================================== test session starts =====================================
platform linux -- Python 3.10.13, pytest-7.4.4, pluggy-1.4.0
rootdir: /simulation
collected 1 item

pynguin-testgen/test_control_center.py .                                                [100%]

===================================== 1 passed in 0.01s ======================================
```

Execution of tests - vehicle infected

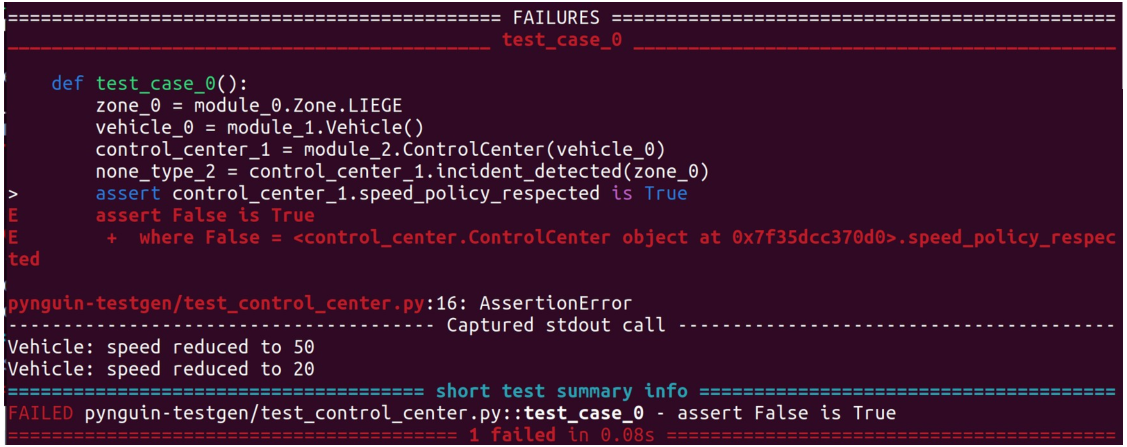
Apply following test to control center code:





```
def test_case_0():
    zone_0 = module_0.Zone.LIEGE
    vehicle_0 = module_1.Vehicle()
    control_center_1 = module_2.ControlCenter(vehicle_0)
    none_type_2 = control_center_1.incident_detected(zone_0)
    assert control_center_1.speed_policy_respected is True
```

speed_policy_respected = self.vehicle.get_speed() <= new_speed_policy

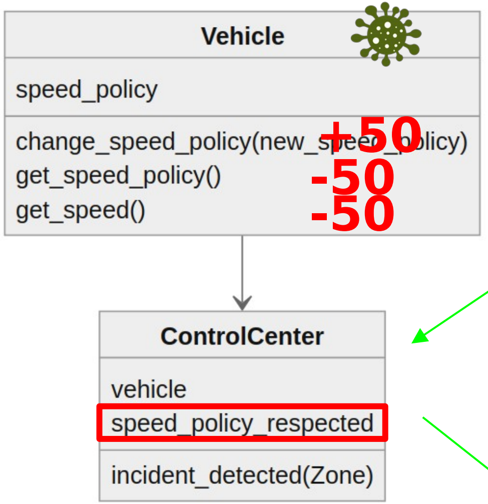


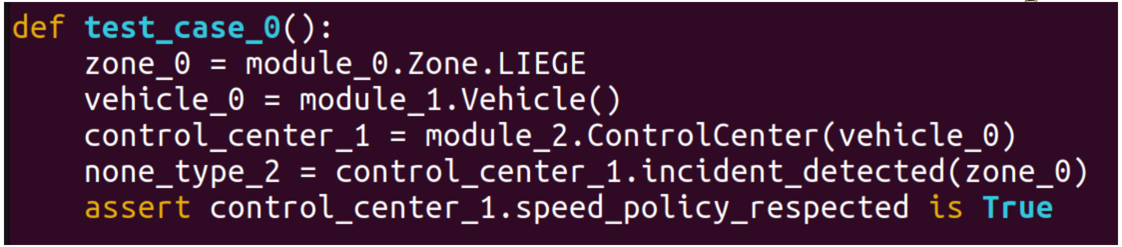
```
=========================================== FAILURES ===========================================
__________________________________________ test_case_0 _________________________________________

    def test_case_0():
        zone_0 = module_0.Zone.LIEGE
        vehicle_0 = module_1.Vehicle()
        control_center_1 = module_2.ControlCenter(vehicle_0)
        none_type_2 = control_center_1.incident_detected(zone_0)
>       assert control_center_1.speed_policy_respected is True
E       assert False is True
E        +  where False = <control_center.ControlCenter object at 0x7f35dcc370d0>.speed_policy_respec
ted

pynguin-testgen/test_control_center.py:16: AssertionError
--------------------------------------- Captured stdout call ---------------------------------------
Vehicle: speed reduced to 50
Vehicle: speed reduced to 20
==================================== short test summary info =====================================
FAILED pynguin-testgen/test_control_center.py::test_case_0 - assert False is True
======================================= 1 failed in 0.08s ========================================
```

Execution of tests - vehicle infected

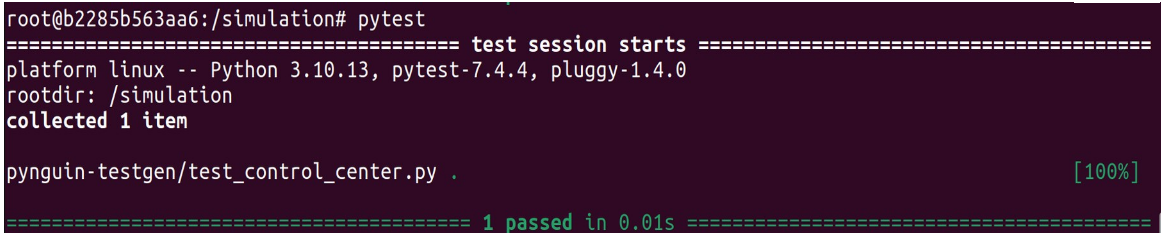
Advanced attack:

- magnify the vehicle speed policy change,
- spoof the speed policy readings for the control center.
- **spoof speed readings for the control center**





```python
def test_case_0():
    zone_0 = module_0.Zone.LIEGE
    vehicle_0 = module_1.Vehicle()
    control_center_1 = module_2.ControlCenter(vehicle_0)
    none_type_2 = control_center_1.incident_detected(zone_0)
    assert control_center_1.speed_policy_respected is True
```

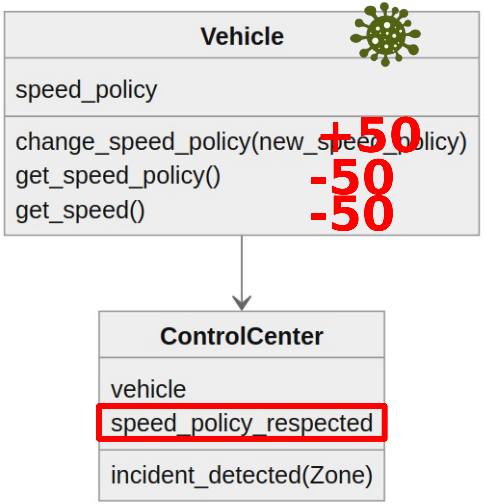


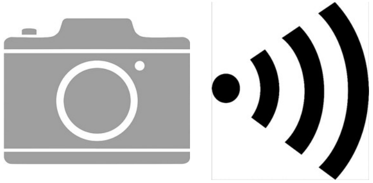
```
root@b2285b563aa6:/simulation# pytest
========================================= test session starts =========================================
platform linux -- Python 3.10.13, pytest-7.4.4, pluggy-1.4.0
rootdir: /simulation
collected 1 item

pynguin-testgen/test_control_center.py .                                                        [100%]

========================================== 1 passed in 0.01s ==========================================
```

Execution of tests - external observer

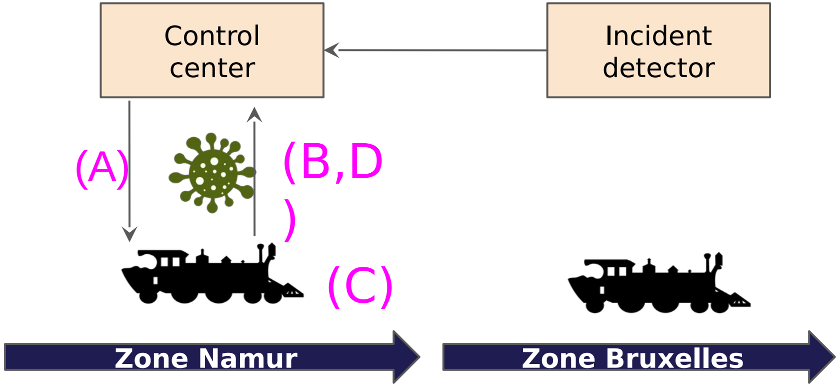
Advanced attack:

- magnify the vehicle speed policy change,
- spoof the speed policy readings for the control center.
- **spoof speed readings for the control center**





External observer
(speed camera)



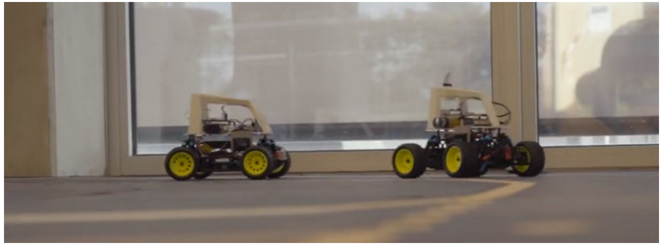
Execution of tests - summary

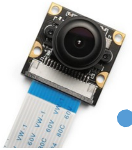
- user express security invariants (properties)
  - security policies are implemented through security invariants
- pynguin generates assertions to verify they are respected - or not
- with corresponding generated tests,
  - on non infected code => test successful
  - on infected code => test failure

Test report

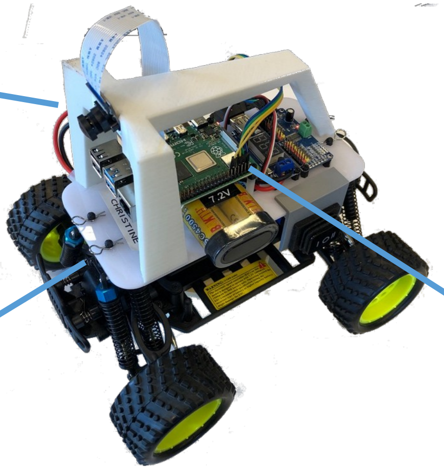
Test / Assertion	**A** Integrity: zone policy sent is the one received	**B** Zone policy is respected	**C** Integrity: monitored data corresponds to real data	**D** Integrity: monitoring data sent is the same that is received	**(E)** Monitoring data displayed is the same as received data
Test 1	X				Out of scope
Test 2		X		X	
…			External observer		

Cyber Lab - Cyber Physical Systems (CPS)





Wide Lens Camera
Tracking



Donkey Car
Chassis

Ultrasonic Distance
Sensor

RaspBerry Pi /
Jetson Nano
Brain

Next steps - Test generation for ROS

ROS 2™

https://www.ros.org/

ROS® industrial

ROS-INDUSTRIAL

ROS-Industrial is an open-source project that extends the advanced capabilities of ROS to manufacturing automation and robotics.
https://rosindustrial.org

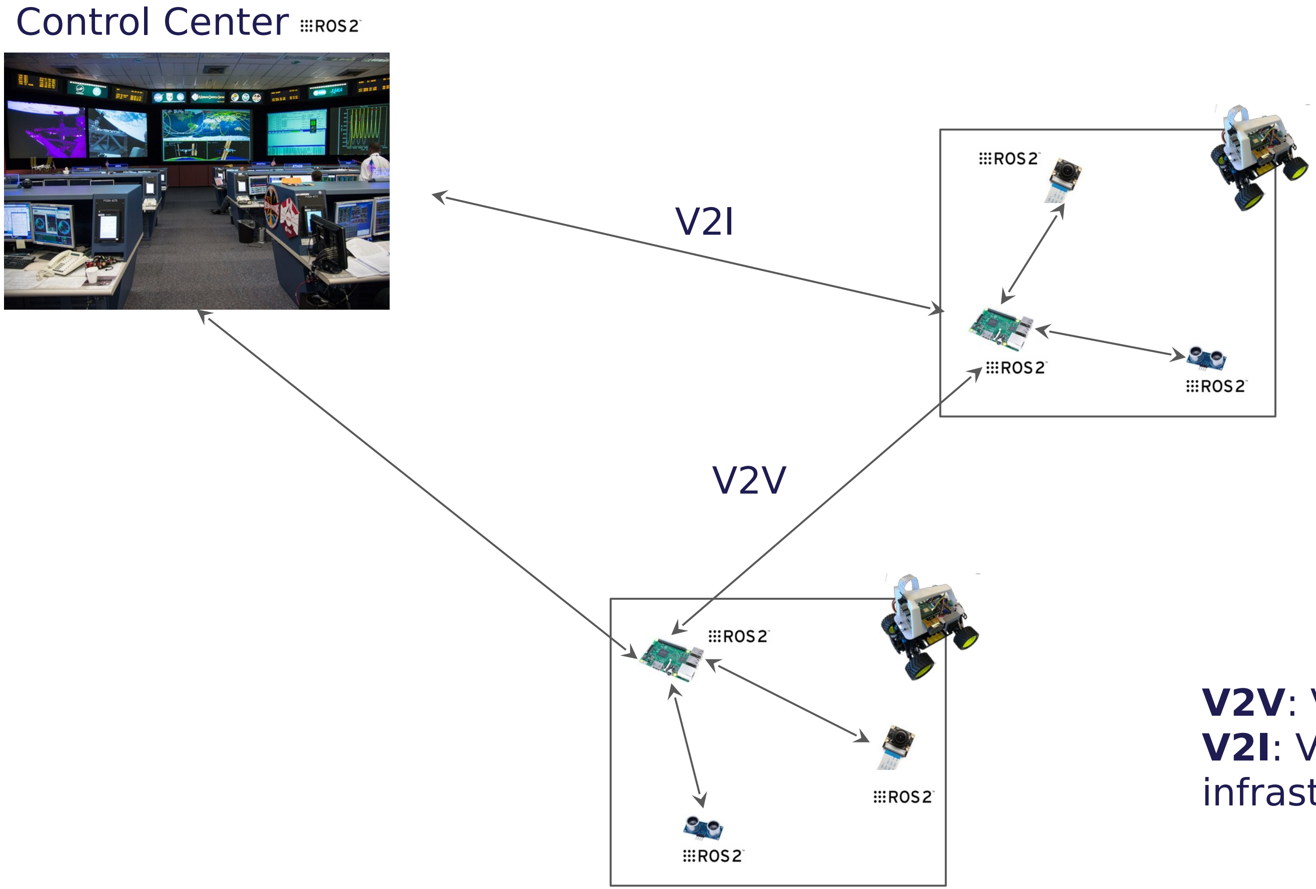
NASA VIPER

Prospecting for lunar resources in permanently shadowed regions of the lunar south pole

- **ROS** used in ground software systems
- **Gazebo** simulation used in mission development, testing, planning, operator training, etc.
- Other open source software
  - cFS/ROS bridge
  - Yamcs
  - OpenMCT
- NASA requires software used in **flight missions** to be space qualified

An open-source **space robotics framework** for developing flight-quality robotics and autonomous space systems
https://space.ros.org/

Cyber Lab - Cyber Physical Systems (CPS)



Control Center ROS2

V2I

V2V

ROS2

ROS2

ROS2

ROS2

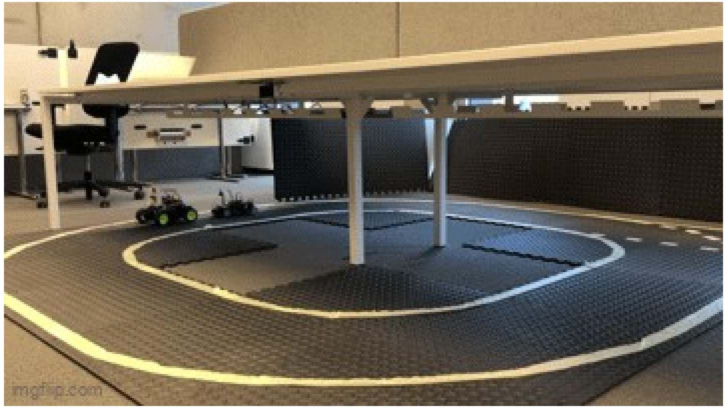
ROS2

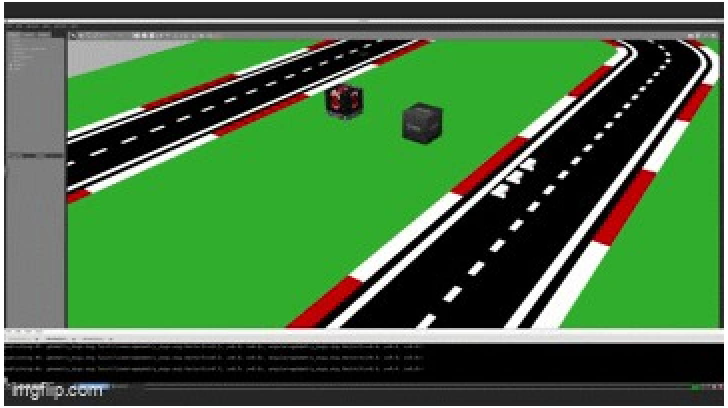
ROS2

V2V: Vehicle to vehicle
V2I: Vehicle to infrastructure

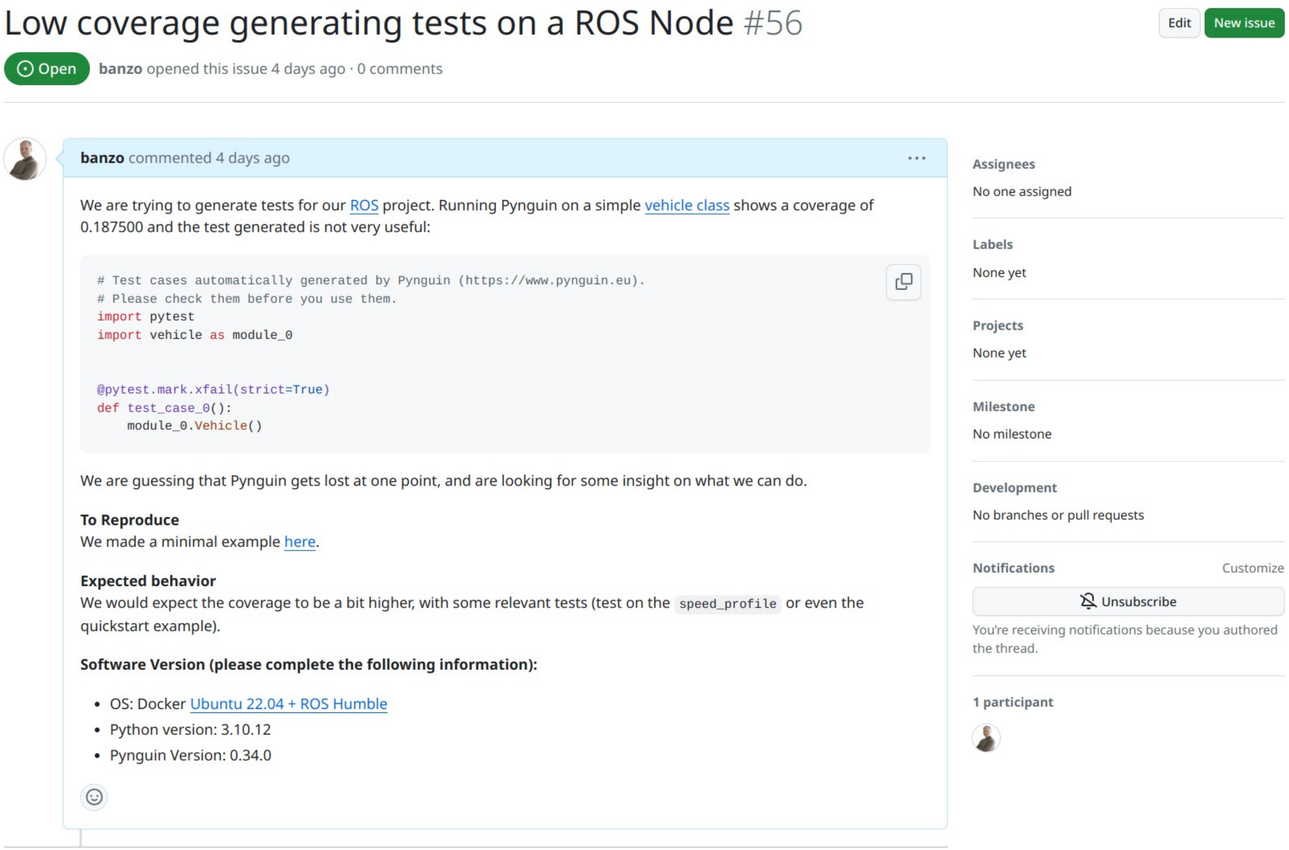
Cyber Lab - Cyber Physical Systems (CPS)

```
Position:[0, 0] Delta:[0, 0]
Command:[0.00, 0.90] Delta_ts:0.1004791259765625
Final Command:[353.0, 399.0]
```





Next steps - Test generation for ROS



Low coverage generating tests on a ROS Node #56

banzo opened this issue 4 days ago · 0 comments

banzo commented 4 days ago

We are trying to generate tests for our ROS project. Running Pynguin on a simple vehicle class shows a coverage of 0.187500 and the test generated is not very useful:

```
# Test cases automatically generated by Pynguin (https://www.pynguin.eu).
# Please check them before you use them.
import pytest
import vehicle as module_0


@pytest.mark.xfail(strict=True)
def test_case_0():
    module_0.Vehicle()
```

We are guessing that Pynguin gets lost at one point, and are looking for some insight on what we can do.

To Reproduce
We made a minimal example here.

Expected behavior
We would expect the coverage to be a bit higher, with some relevant tests (test on the `speed_profile` or even the quickstart example).

Software Version (please complete the following information):

- OS: Docker Ubuntu 22.04 + ROS Humble
- Python version: 3.10.12
- Pynguin Version: 0.34.0

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
No branches or pull requests

Notifications
Customize

You're receiving notifications because you authored the thread.

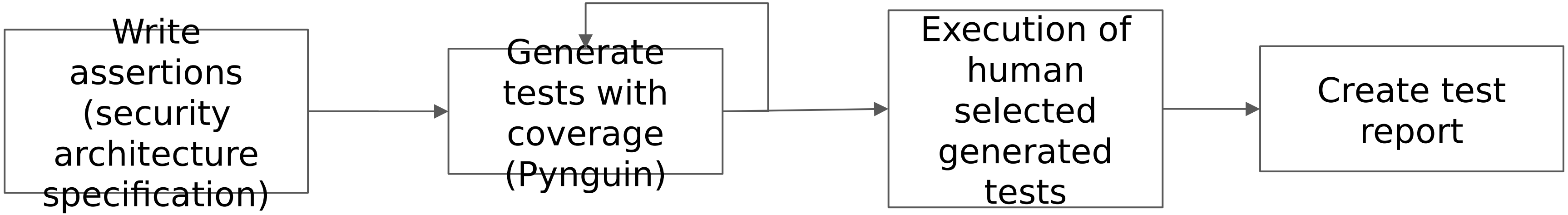
1 participant

https://github.com/se2p/pynguin/issues/56

Conclusions and next steps

- Generation of integration tests for the use case
- Generation of security tests based on control variables introduced inside code

- Problem with ROS for test generation
- For a same coverage level, generated tests are not similar
- Implement assertions in place of variables
- Generate tests for all assertions ? - Does it generate the right tests ? Are there missing tests, and able to discover vulnerabilities ? What is the coverage level ?
- incorporate (how?) a fuzzer iot obtain more tests ?



Write assertions (security architecture specification)

Generate tests with coverage (Pynguin)

Execution of human selected generated tests

Create test report

Further reading

- MITRE - DELIVER UNCOMPROMISED: SECURING CRITICAL SOFTWARE SUPPLY CHAINS PROPOSAL TO ESTABLISH AN END-TO-END FRAMEWORK FOR SOFTWARE SUPPLY CHAIN INTEGRITY (2021)
- ENISA - Good Practices for Supply Chain Cybersecurity
- ROS Robotics Companies list

PFV – Protocol Formal Validation

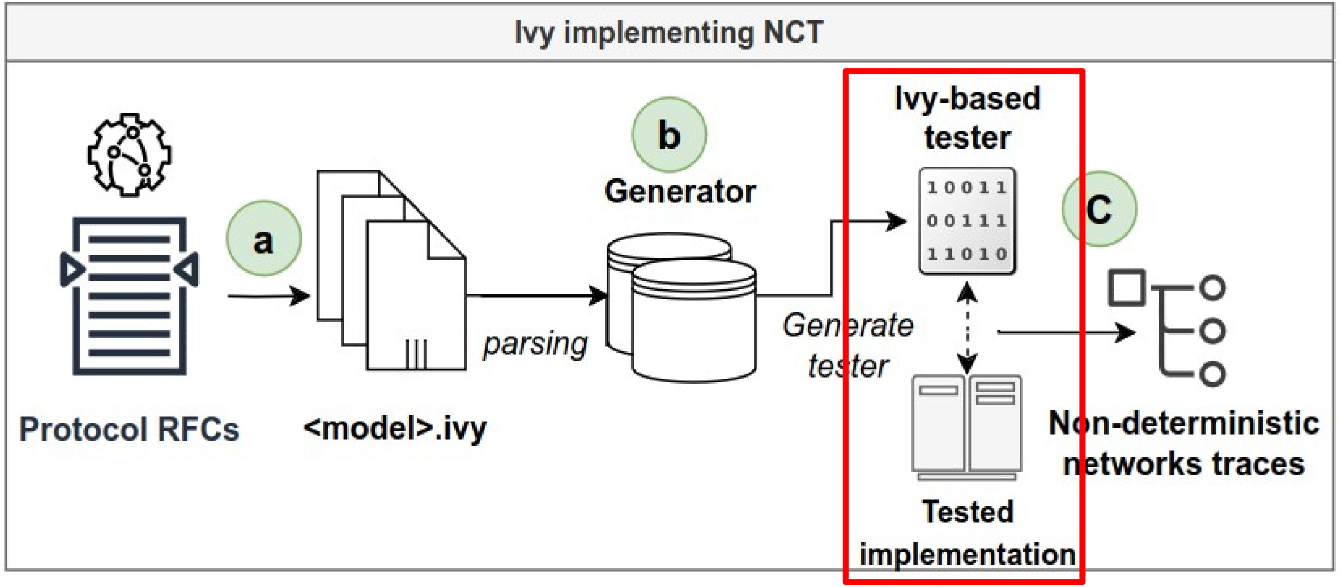
By Christophe Crochet & John Aoga & Axel Legay

Plan

1. **Network Simulator-centric Compositional Testing (NSCT)**
2. IDS Validation
3. Conclusion

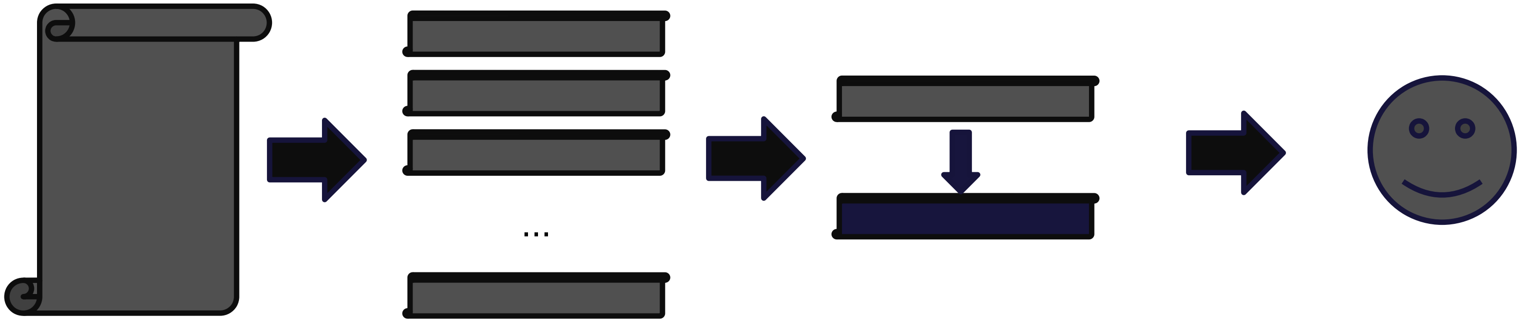
Network centric Compositional Testing (NCT)

- Extension of *Network-centric Compositional Testing* (NCT)
  - *by Kenneth McMillan*



Network centric Compositional Testing (NCT)

- Extension of *Network-centric Compositional Testing* (NCT)
  - *by Kenneth McMillan*



RFC9000

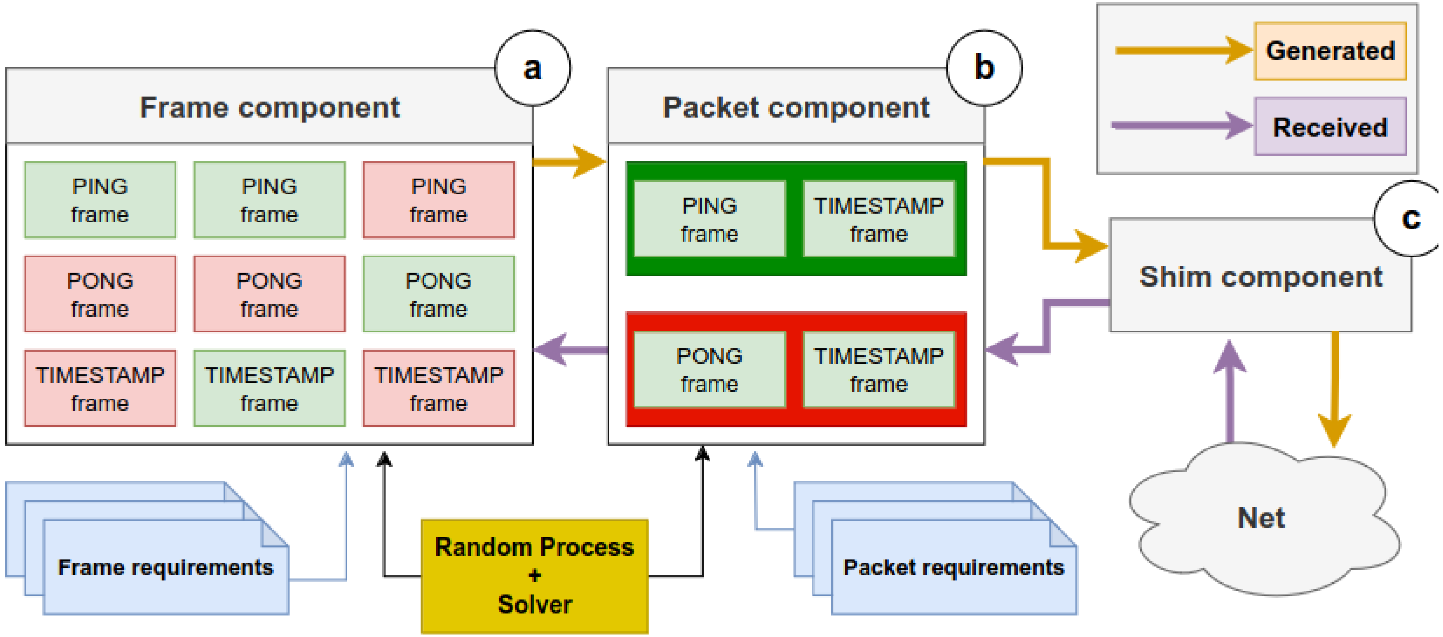
Set of requirements = **generic QUIC formal specification**

Refinement of some requirements = tests e.g only allow generation of X frame

Ivy **QUIC implementation** (server or client)

Network centric Compositional Testing (NCT)

- Random Process



Network centric Compositional Testing (NCT)

- Testing - Previous Works

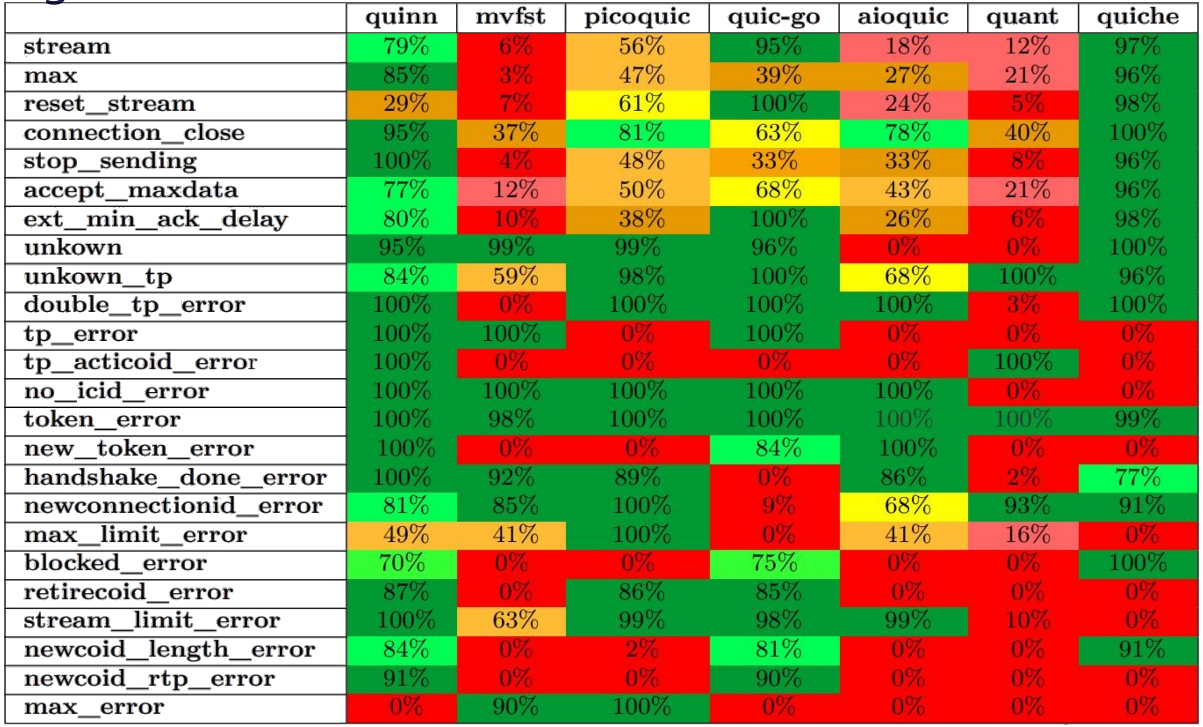
1. Violation of the specification
2. Feature not implemented
3. Internal errors and crashes
4. Problem in the draft

35 main errors developed

Network centric Compositional Testing (NCT)

Server

- Testing - Previous Works

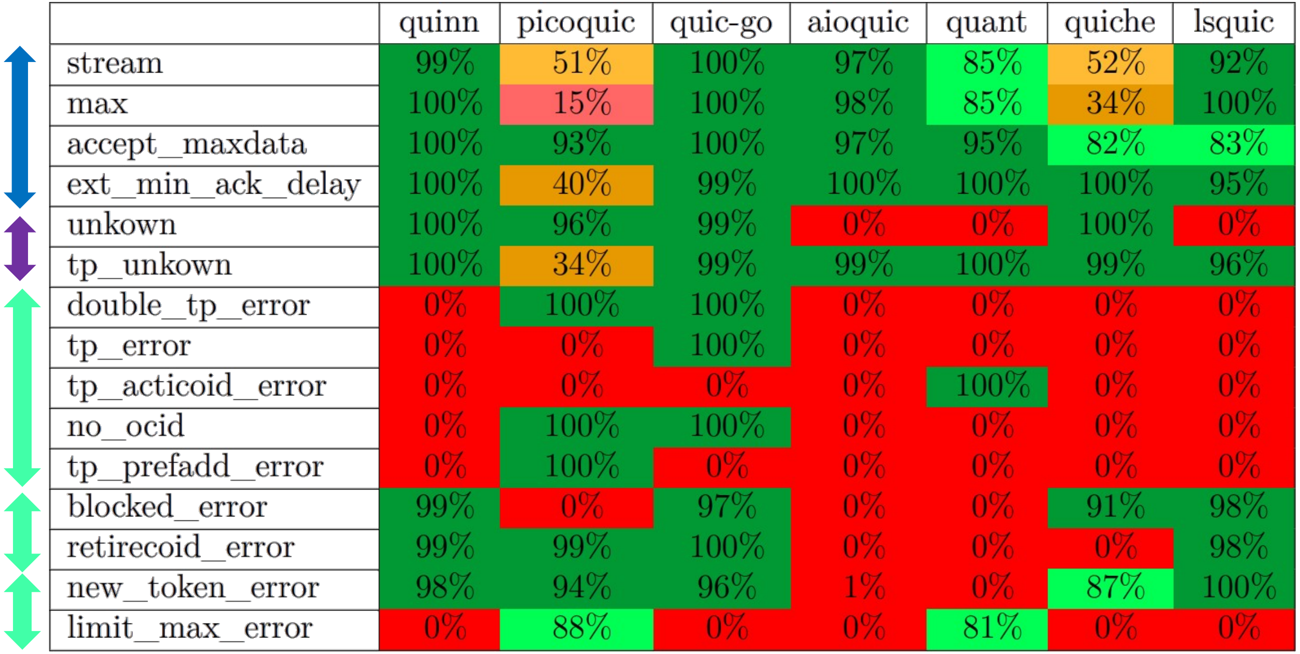


		quinn	mvfst	picoquic	quic-go	aioquic	quant	quiche
Generic	stream	79%	6%	56%	95%	18%	12%	97%
	max	85%	3%	47%	39%	27%	21%	96%
	reset_stream	29%	7%	61%	100%	24%	5%	98%
	connection_close	95%	37%	81%	63%	78%	40%	100%
	stop_sending	100%	4%	48%	33%	33%	8%	96%
	accept_maxdata	77%	12%	50%	68%	43%	21%	96%
	ext_min_ack_delay	80%	10%	38%	100%	26%	6%	98%
Unknown	unkown	95%	99%	99%	96%	0%	0%	100%
	unkown_tp	84%	59%	98%	100%	68%	100%	96%
Transport parameter errors	double_tp_error	100%	0%	100%	100%	100%	3%	100%
	tp_error	100%	100%	0%	100%	0%	0%	0%
	tp_acticoid_error	100%	0%	0%	0%	0%	100%	0%
	no_icid_error	100%	100%	100%	100%	100%	0%	0%
Violation of the draft	token_error	100%	98%	100%	100%	100%	100%	99%
	new_token_error	100%	0%	0%	84%	100%	0%	0%
	handshake_done_error	100%	92%	89%	0%	86%	2%	77%
	newconnectionid_error	81%	85%	100%	9%	68%	93%	91%
Invalid field	max_limit_error	49%	41%	100%	0%	41%	16%	0%
	blocked_error	70%	0%	0%	75%	0%	0%	100%
	retirecoid_error	87%	0%	86%	85%	0%	0%	0%
	stream_limit_error	100%	63%	99%	98%	99%	10%	0%
	newcoid_length_error	84%	0%	2%	81%	0%	0%	91%
	newcoid_rtp_error	91%	0%	0%	90%	0%	0%	0%
	max_error	0%	90%	100%	0%	0%	0%	0%

Network centric Compositional Testing (NCT)

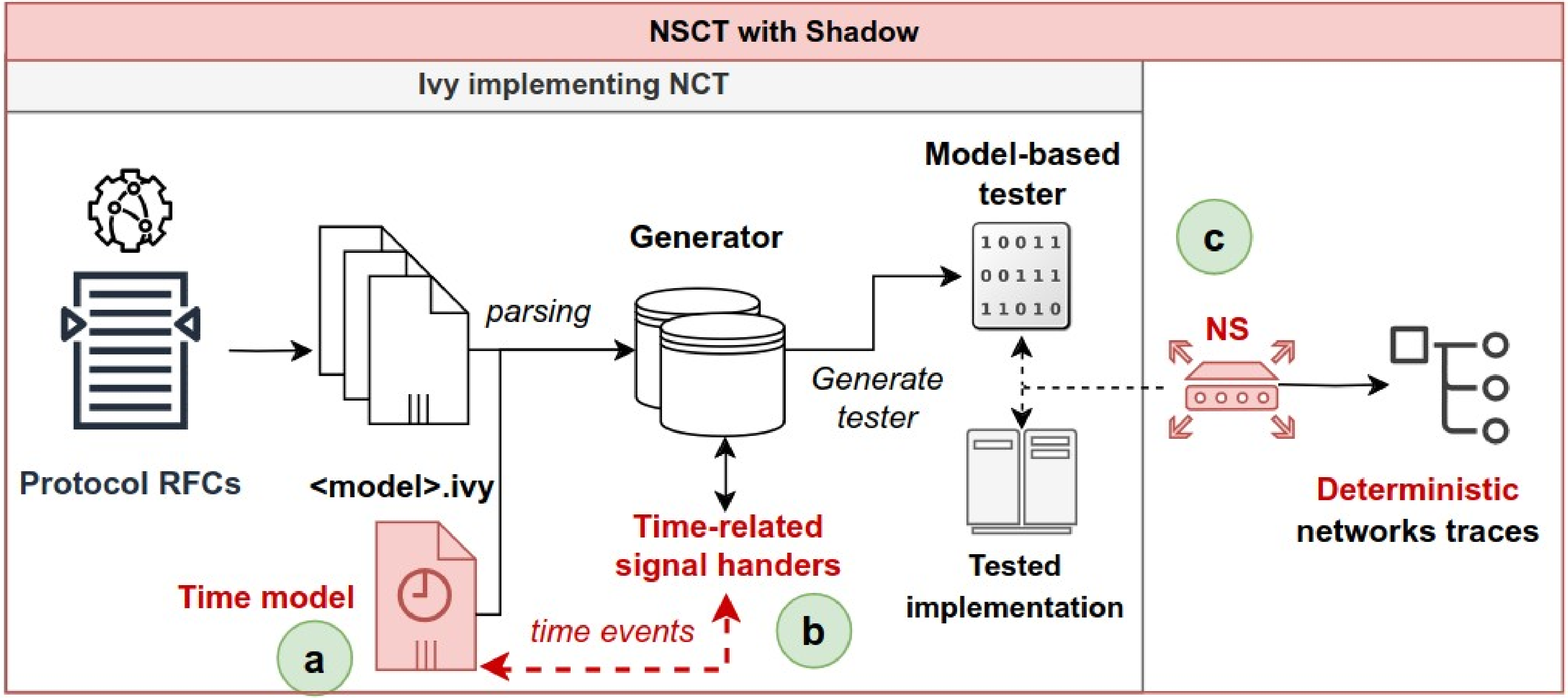
Client
No migration

- Testing - Previous Works



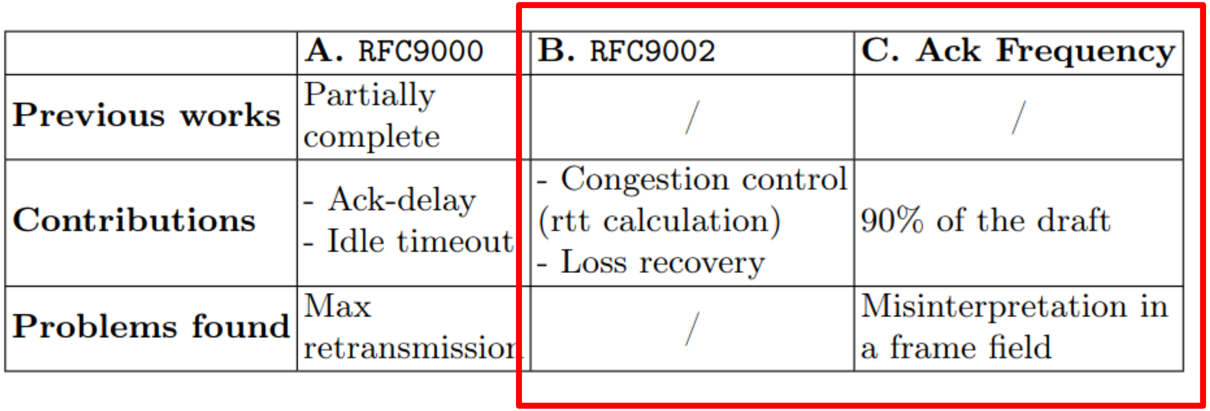
	quinn	picoquic	quic-go	aioquic	quant	quiche	lsquic
stream	99%	51%	100%	97%	85%	52%	92%
max	100%	15%	100%	98%	85%	34%	100%
accept_maxdata	100%	93%	100%	97%	95%	82%	83%
ext_min_ack_delay	100%	40%	99%	100%	100%	100%	95%
unkown	100%	96%	99%	0%	0%	100%	0%
tp_unkown	100%	34%	99%	99%	100%	99%	96%
double_tp_error	0%	100%	100%	0%	0%	0%	0%
tp_error	0%	0%	100%	0%	0%	0%	0%
tp_acticoid_error	0%	0%	0%	0%	100%	0%	0%
no_ocid	0%	100%	100%	0%	0%	0%	0%
tp_prefadd_error	0%	100%	0%	0%	0%	0%	0%
blocked_error	99%	0%	97%	0%	0%	91%	98%
retirecoid_error	99%	99%	100%	0%	0%	0%	98%
new_token_error	98%	94%	96%	1%	0%	87%	100%
limit_max_error	0%	88%	0%	0%	81%	0%	0%

Network Simulator-centric Compositional Testing (NSCT)



Network Simulator-centric Compositional Testing (NSCT)

- Testing - Previous Works

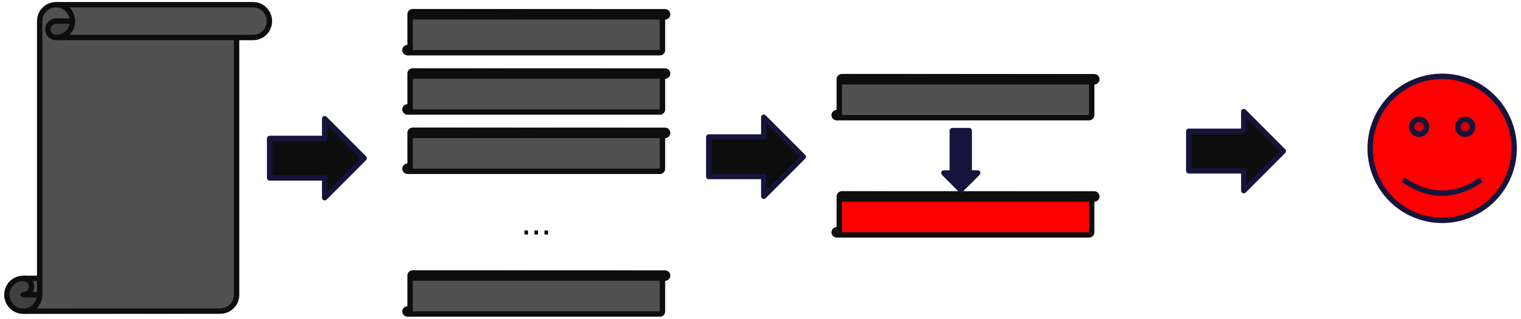


	A. RFC9000	B. RFC9002	C. Ack Frequency
Previous works	Partially complete	/	/
Contributions	- Ack-delay - Idle timeout	- Congestion control (rtt calculation) - Loss recovery	90% of the draft
Problems found	Max retransmission	/	Misinterpretation in a frame field

Table 1: Summary of contributions to Ivy model and problems found in *picoquic*

Network centric Compositional Testing (NCT)

- Attack models



RFC9000

Set of requirements = generic QUIC formal specification

Illegal/Malicious requirements addition/modification

Ivy QUIC attacker

Network centric Compositional Testing (NCT)

- Attack models - Previous Works:
  - **Man In the Middle:**
    - lsquic vulnerable with version negociation attack
      1. lsquic start the handshake with version 0xff000022 (draft-34)
      2. then we propose the 0xff00001d version (draft-29).
      3. It responds us by resending an Initial packet with incorrect checksum.
  - **DoS - Packet/frame manipulation:**
    - NEW_CONNECTION_ID frame - quant
    - Malicious QUIC frame injection - picoquic

Network centric Compositional Testing (NCT)

- Att
  - **Ma**
    - [redacted]

(draft-34)

correct

Paper in Preparation
+ Timing attacks

- **DoS - Packet/frame manipulation:**
  - NEW_CONNECTION_ID frame - quant
  - Malicious QUIC frame injection - picoquic

Network Simulator-centric Compositional Testing (NSCT)

- Summary:
  - **NCT**:
    - Model-Based Formal Specification Adversarial testing (*Black Box Endpoint*)
    - Component Based
    - Randomized Process + Non-Deterministic
    - Efficient to find errors in implementation and ambiguity in specification
    - Efficient to find vulnerabilities in implementation

Network Simulator-centric Compositional Testing (NSCT)

- Summary:
  - **NSCT**:
    - Model-Based Formal Specification Adversarial testing in NS (***Grey Box*** *Endpoint*)
    - Component Based
    - Randomized Process + **Deterministic + Reproducible + online debugging**
    - **Enable Timing based attacks**
    - *~ Might need implementation of syscalls*

Plan of the Presentation

1. Network Simulator-centric Compositional Testing (NSCT)
2. **IDS Validation**
3. Conclusion

https://dl.acm.org/doi/pdf/10.1145/3359986.3361208

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8606252

IDS Validation

Formal APT Model

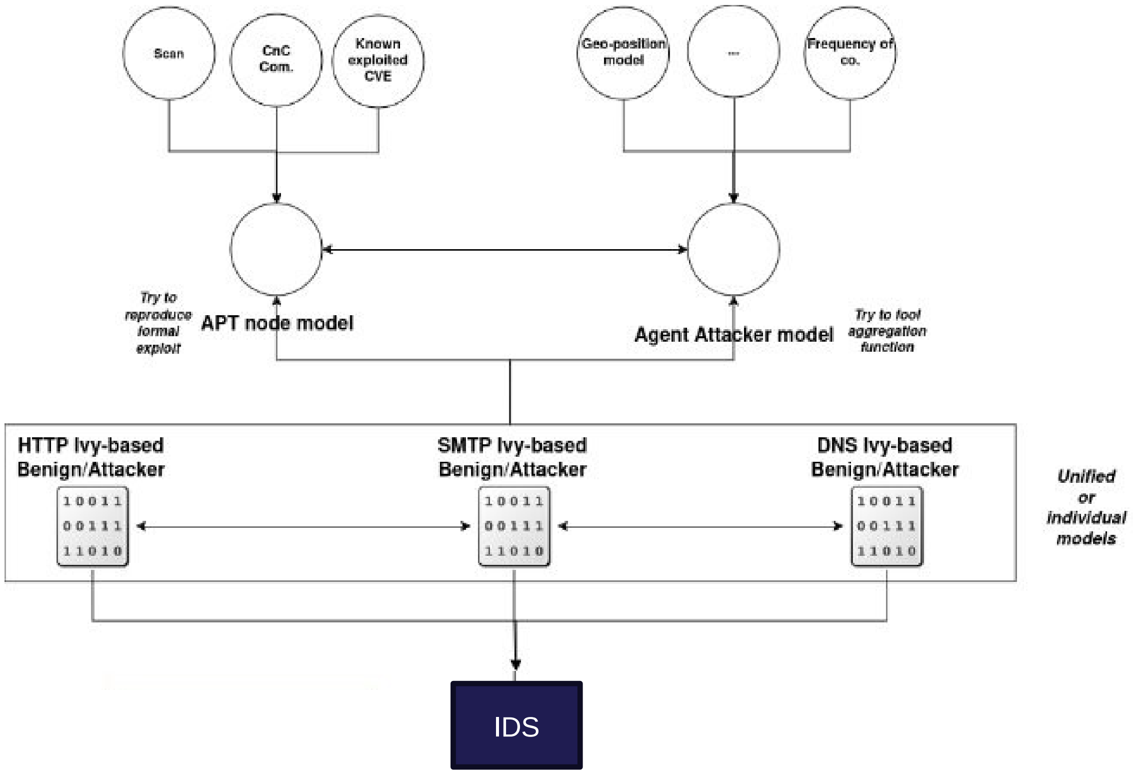
- APT = Advanced Persistent Threat
  - Infiltration
  - Escalation and Lateral Movement
  - Exfiltration
  - APT Attack Tree (*for multiple RFCs – Attacks: HTTP, FTP, ...*)
  - **Formal APT Attack Tree Nodes/Components !**
    - Web based nodes only (no usb, social engineering, ...)
    - Formal Attack "API"

https://dl.acm.org/doi/pdf/10.1145/3359986.3361208

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8606252

IDS Validation

Formal APT Model

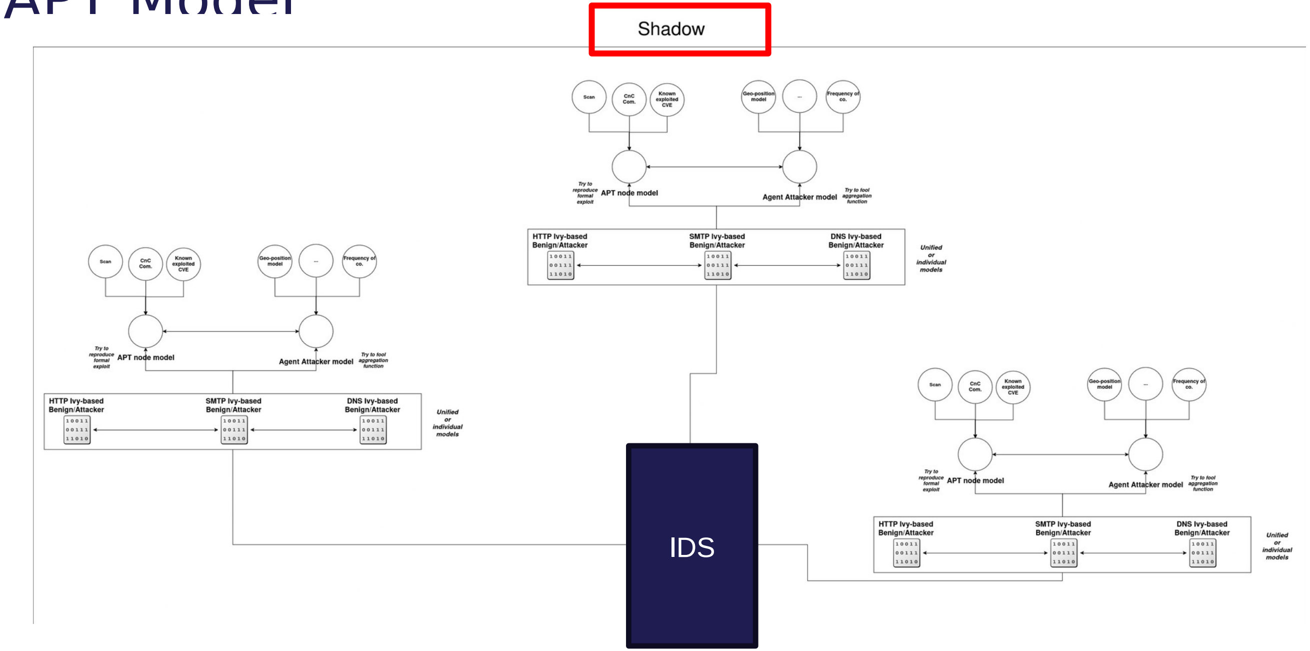

Scan
CnC Com.
Known exploited CVE
Geo-position model
...
Frequency of co.
Try to reproduce formal exploit
APT node model
Agent Attacker model
Try to fool aggregation function
HTTP Ivy-based Benign/Attacker
SMTP Ivy-based Benign/Attacker
DNS Ivy-based Benign/Attacker
10011
00111
11010
Unified or individual models
IDS

https://dl.acm.org/doi/pdf/10.1145/3359986.3361208

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8606252

IDS Validation

Formal APT Model


Shadow
Scan
CnC Com.
Known exploited CVE
Geo-position model
...
Frequency of co.
Try to reproduce formal exploit
APT node model
Agent Attacker model
Try to fool aggregation function
HTTP Ivy-based Benign/Attacker
SMTP Ivy-based Benign/Attacker
DNS Ivy-based Benign/Attacker
10011
00111
11010
Unified or individual models
Scan
CnC Com.
Known exploited CVE
Geo-position model
...
Frequency of co.
Try to reproduce formal exploit
APT node model
Agent Attacker model
Try to fool aggregation function
HTTP Ivy-based Benign/Attacker
SMTP Ivy-based Benign/Attacker
DNS Ivy-based Benign/Attacker
10011
00111
11010
Unified or individual models
Scan
CnC Com.
Known exploited CVE
Geo-position model
...
Frequency of co.
Try to reproduce formal exploit
APT node model
Agent Attacker model
Try to fool aggregation function
HTTP Ivy-based Benign/Attacker
SMTP Ivy-based Benign/Attacker
DNS Ivy-based Benign/Attacker
10011
00111
11010
Unified or individual models
IDS

IDS Validation

Formal APT Model - *NSCT*

Phantom = Extension of Shadow

- 60 Tor networks using Tor v0.4.5.9
- Blade server cluster in which each blade contained identical hardware:
  - 1.25 TiB of RAM and
  - 4×8 core Intel Xeon E5-4627v2 CPUs (without hyper-threading support) running at 3.30 GHz.

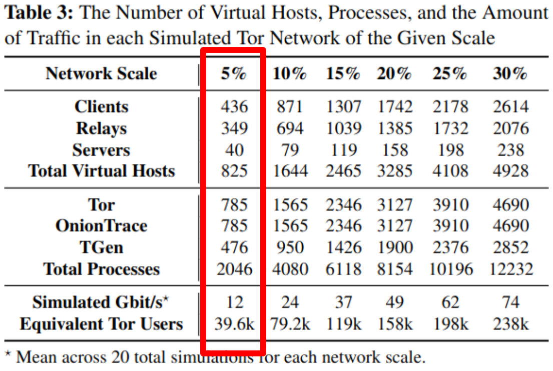
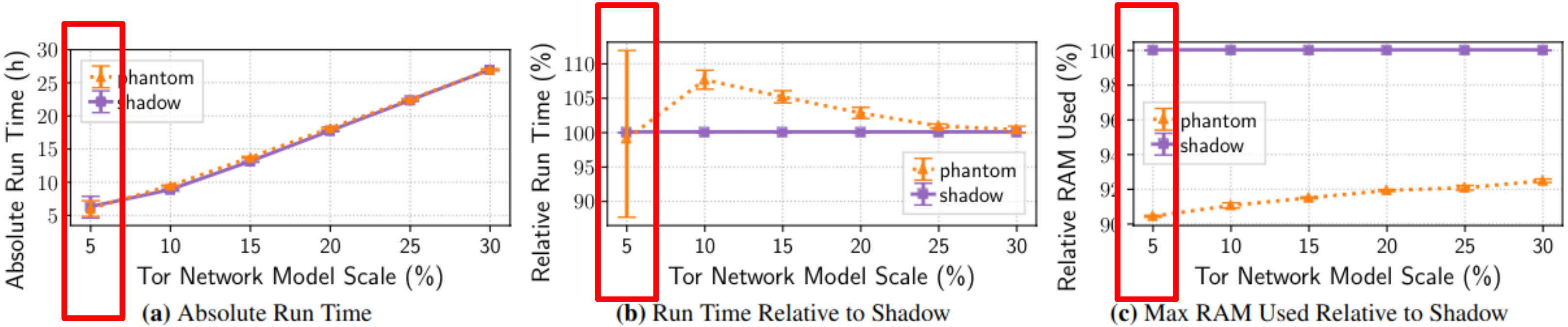


Table 3: The Number of Virtual Hosts, Processes, and the Amount of Traffic in each Simulated Tor Network of the Given Scale

Network Scale	5%	10%	15%	20%	25%	30%
Clients	436	871	1307	1742	2178	2614
Relays	349	694	1039	1385	1732	2076
Servers	40	79	119	158	198	238
Total Virtual Hosts	825	1644	2465	3285	4108	4928
Tor	785	1565	2346	3127	3910	4690
OnionTrace	785	1565	2346	3127	3910	4690
TGen	476	950	1426	1900	2376	2852
Total Processes	2046	4080	6118	8154	10196	12232
Simulated Gbit/s*	12	24	37	49	62	74
Equivalent Tor Users	39.6k	79.2k	119k	158k	198k	238k

* Mean across 20 total simulations for each network scale.



(a) Absolute Run Time

(b) Run Time Relative to Shadow

(c) Max RAM Used Relative to Shadow

Figure 23: The time and memory required to complete each Tor network simulation in Phantom (using `seccomp` interception) and in Shadow's uni-process design as the network model scale increases. (b) and (c) show performance relative to Shadow's baseline.

Plan of the Presentation

1. Network Simulator-centric Compositional Testing (NSCT)
2. IDS Validation
3. **Conclusion**

Conclusion

- NCT/NSCT can find bugs and model attacks
  - Probably lower cost
- Leverage LLM for automating attacks and model creation
- GUI

Planning réunion de groupe de travail par Défi

Date	Description
23/01/2023	First meeting of the working group
29/09/2023	Présentation des research results and discussion on demonstrators
13/05/2024	Présentation of démonstrateurs
*/11/2024	Présentation of more mature demonstrators

Who participates:

- Companies interested in the challenge
- Challenge Manager
- Researchers contributing to the challenge
- WSL
- Réseau Lieu

Thank you for your attention