From MisuseCase to Analysis Data

How to use a functional toolchain for expert based AI analysis

Arlena Wellßow, OFFIS

10.05.2023

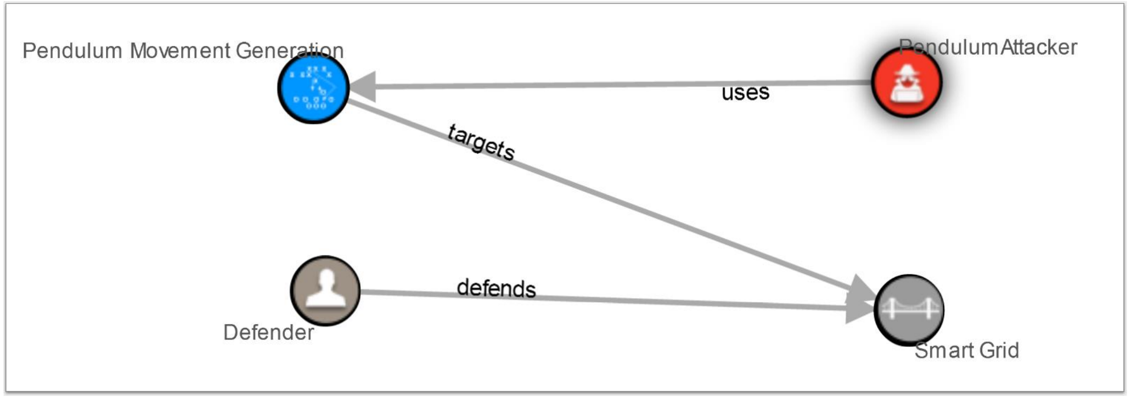
This project has received funding in the framework of the joint programming initiative ERA-Net Smart Energy Systems’ focus initiative Digital Transformation for the Energy Transition, with support from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 883973

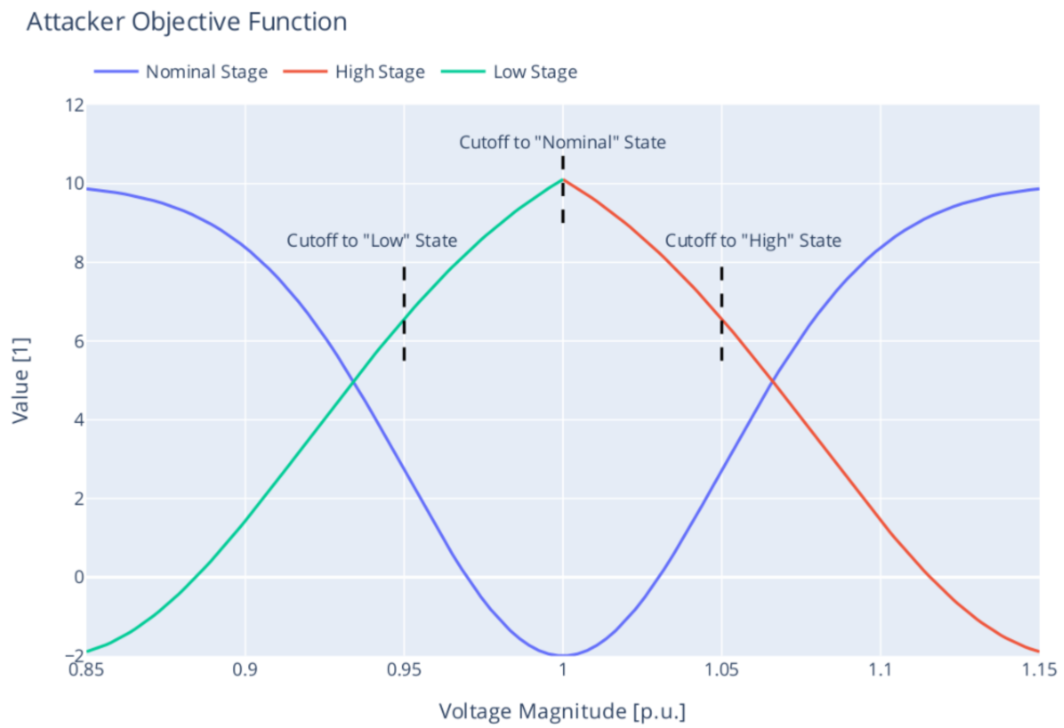
General Idea

- Build a chain from expert knowledge to analyzed knowledge base
- 6 Steps:
  1. Structure Expert Knowledge
  2. Make Knowledge Machine Readable
  3. Generate an Experiment from Knowledge
  4. Use the arsenAI / palaestrAI Toolchain
  5. Analyse the Data
  6. Re-add the Data to the Knowledgebase

Running Example

- So called „Pendulum Attack“
- Attacker uses the reaction of the defender for the attack
- Leads to an oscillating behavior





Step 1: Structure Expert Knowledge

- Done by filling out „MisuseCase“-Templates (with additional parameters or in combination with additional techniques)
- Everything a domain expert knows about the attack is structured and documented
- This document can be shared with other experts to benefit from extensive domain knowledge exchange

MisuseCase Template

Misuse Case Template

1 Description of the Misuse Case

1.1 Name of the Misuse Case

Misuse Case Identification		
ID	Area Domain(s)/ Zone(s)	Name of Misuse Case
MUC_2		Generation of pendulum movement in the voltage band

1.2 Version Management

Version Management				
Version No.	Date	Name of Author(s)	Changes	Approval Status
0.1	05.08.2022	Arlena Wellßow	First Draft	Draft

1.3 Scope and Objectives of Misuse Case

Scope and Objectives of Misuse Case	
Scope	
Objective(s)	
Related Business Case(s)	

1.4 Narrative of Misuse Case

Narrative of Misuse Case
Short Description
Complete Description

1.5 Misuse Case Conditions

Misuse Case Conditions
Assumptions
Prerequisites

Step 2: Make Knowledge Machine Readable

- Export the Domain Knowledge Documents in machine readable formats like XML
  - Possible for e.g. Word, UML

- Read XML data in a programming language of choice
  - In our case: python

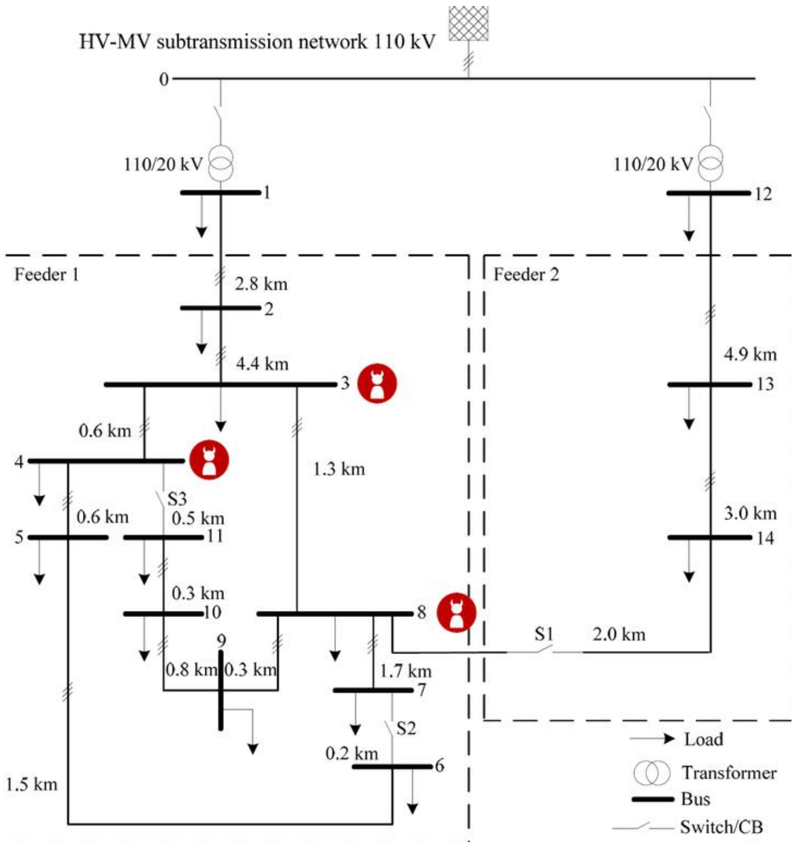
Step 3: Generate an Experiment from Knowledge

- Use the machine readable domain knowledge to export every experiment relevant information
- E.g. objectives of the agents, name of the attack, KPIs, ...
- Analyse XML data and write the important information to a .yaml experiment file

```yaml
%YAML 1.2
 # The Classic ARL reference experiment

---
uid: Cost Based Experiment
seed: 20222811  # Seed used to initialize the random number generator
version: 3.4.1  # Target palaestrAI version
output: palaestrai-runfiles  # Directory where the run files will be saved
repetitions: 1  # How often each run will be repeated. Each repetition is a separate file
max_runs: 10  # Maximum number of runs (excluding repetitions) created by the DoE generator
definitions:  # In this section all the components will be defined
  ###########################################
  # The environment' section
  environments:
    midasmv_tar_ms:
      environment:
        name: palaestrai_mosaik:MosaikEnvironment
        uid: midas_powergrid
        params:
          module: midas.tools.palaestrai:Descriptor  # Always the same if you use MIDAS
          description_func: describe  # Always the same if you use MIDAS
          instance_func: get_world  # Always the same if you use MIDAS
          arl_sync_freq: &step_size 900 # &step_size 1 means to set variable step_size=1
          end: &end 9001 # One step extra because one step gets "lost"
          silence_missing_input_connections_warning: True

          params:  # Parameters that are passed to description_func and
                   # instance_func, Part that is being sent to MIDAS directly
            name: carl_cigre_ts
            config: midas-scenarios/classic-arl.yml
            end: *end
            step_size: *step_size # &step_size put here from above with value 1
            start_date: 2020-05-01 00:00:00+0100 # Use ISO datestring or magic keyword 'random'
            mosaik_params: {addr: [127.0.0.1, 56781]}
            store_params: #everything in here overwrites what is written in classic-arl.yml in /midas-scenarios/
              buffer_size: 1000 # Every x steps, intermediate results are dumped to the midas store
              keep_old_files: true
              filename: retro_psi_midas.hdf5
      reward:
        name: midas.tools.palaestrai.rewards:RetroPsiReward
  ###########################################
  # The agents' section
```



[1]

Step 4: Use the ArsenAI / PalaestrAI toolchain

- Read in YAML to arsenAI
- Then run palaestrAI

palaestrAI: General Toolchain

arsenAI

- CPS-AO (Experiment YAML)
- DoE Sampling

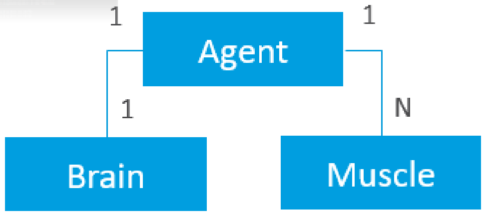
Experiment Run Documents

palaestrAI

- Store
- environments
- CPS-EE
- Agent(s)

hARL

- rDQN, DDPG, TD3, PPO, SAC, … MADDPG



ARL

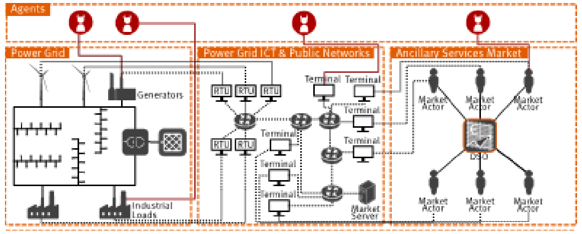


Image done by E. Veith in the project context

arsenAI Experiment Definition

- arsenAI is the Experiment Design part of proper experimentation
  - Defines Parameters and Factors
  - Does a space-filling design sampling to construct concrete experiment runs
  - Nomenclature: Experiment => arsenAI (sampling) => Experiment Run
- All experiment runs are reproducible blue-prints for actual runs
- Experiment and Experiment Run documents are stored in the database
- Experiments plug agents into environments (all are loadable modules)
- Format: YAML

```yaml
###########################################
# The agents' section
agents:
  Attacker:
    name: Attacker
    brain:
      name: harl.ppo.brain:PPOBrain
      params:
        max_timesteps_per_episode: 64
    muscle:
      name: harl.ppo.muscle:PPOMuscle
      params: {}
    objective:
      name: psi_objectives.voltage_attacker_objective:VoltageBandViolationPendulum
      params: { }
  Defender:
    name: Defender
    brain:
      name: harl.ppo.brain:PPOBrain
      params:
        max_timesteps_per_episode: 64
    muscle:
      name: harl.ppo.muscle:PPOMuscle
      params: {}
    objective:
      name: psi_objectives.voltage_defender_objective:VoltageDefenderObjective
      params:
        beta: 2
###########################################
# The sensors' section
sensors:
  all_sensors:
    midas_powergrid:
      - midas_powergrid.Powergrid-0.0-bus-1.vm_pu
      - midas_powergrid.Powergrid-0.0-bus-1.va_degree
      - midas_powergrid.Powergrid-0.0-bus-10.vm_pu
      - midas_powergrid.Powergrid-0.0-bus-10.va_degree
      - midas_powergrid.Powergrid-0.0-bus-11.vm_pu
```

Anatomy of an Experiment Definition

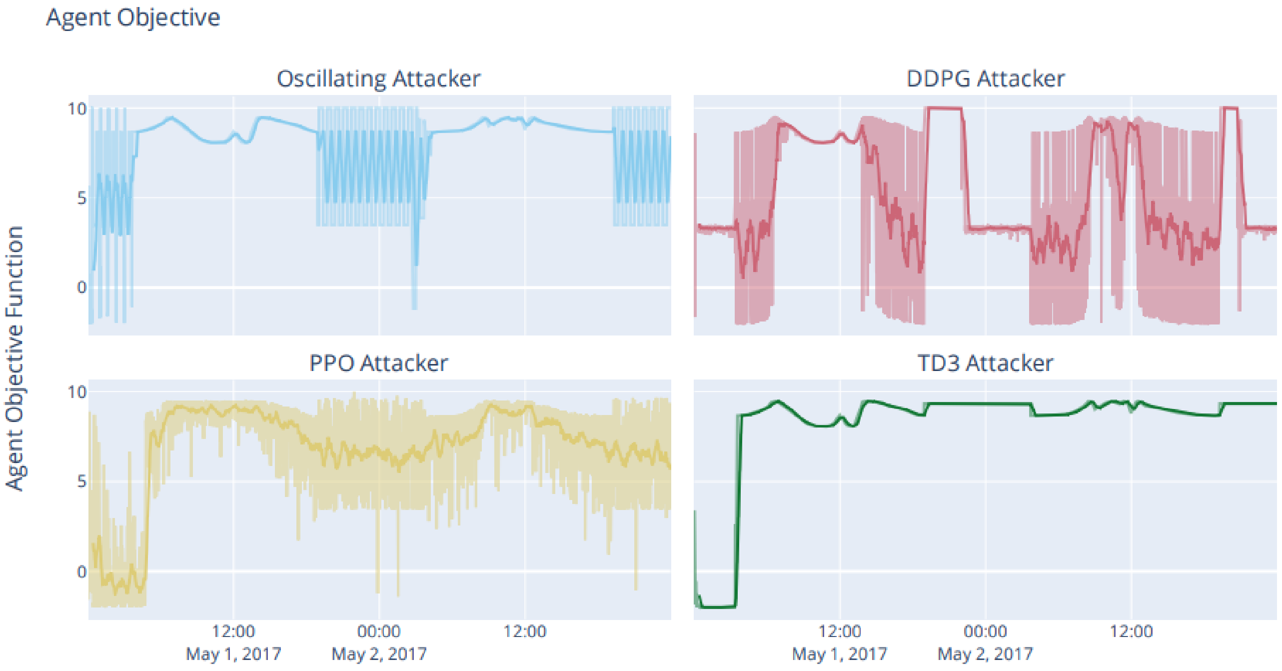
- Experiment definition documents consist of
  - Definitions of agents, sensor/actuator lists, environments, and halting conditions (parameters)
  - Definition of experiment phases (factors)
- arsenAI run: Sampling
  - Chooses optimal space-filling design according to the number of factors and max samples desired
  - 1 Experiment definition can yield 100, 200, … 1000, … experiment runs
- palaestrAI executes experiment runs phase-by-phase
  - E.g., train agent, test agent (or any other combination of agents, sensor/actuator assignments, environments, halting conditions, etc you can think of)
  - palaestrAI re-loads states of agents from previous phases, if wanted

Step 5: Analyze the Data

- Check generated data
- Evaluate the learned patterns of the attacking agent
- Future Work: Read out explainable moves from the agent for new strategies


Oscillating Attacker Setpoints
Victim Buses Voltage Ranges
DDPG Attacker Setpoints
Victim Buses Voltage Ranges
PPO Attacker Setpoints
Victim Buses Voltage Ranges
TD3 Attacker Setpoints
Victim Buses Voltage Ranges
00:10 May 1, 2017
00:20
00:30
00:40
03:15 May 2, 2017
03:20
03:30
03:40
03:50
04:00
PV 0 Q(t) Setpoint
PV 1 Q(t) Setpoint
PV 5 Q(t) Setpoint
V(t) Baseline Normal Operation
V(t) Range During Attack
V(t) Baseline Normal Operation
V(t) Range During Attack

[1]


Agent Objective
Agent Objective Function
Oscillating Attacker
DDPG Attacker
PPO Attacker
TD3 Attacker
12:00 May 1, 2017
00:00 May 2, 2017
12:00

[1]

Step 6: Re-add the Data to the Knowledgebase

- At a later stage of the project
- Read in structured analysis data to e.g. STIX dataformat
- A TAXII-Server was already established and filled with already (in STIX-format) existing data

First Breakthrough [1]

- First five steps were absolved by a prototyped workflow
- A already known attack was taken and written to the structured MUC-template
- First information was read into an experiment file
- General data was analysed to check the behaviour of the attacking agent

Lookout

- Projectscope:
  - Trajectories
  - Re-add data to a Knowledgebase

- In general:
  - Explainable Agent Actions
  - „Knowledge in the Loop“
  - Fixpoint Iteration for new Strategies

- And of course: Train a defending agent to handle the attacks

Any Questions?

Feel free to send a mail to

arlena.wellssow@offis.de

for any later questions, further regards or scenario ideas.

[1] Eric Veith, Arlena Wellßow, and Mathias Uslar. 2023. Learning New Attack Vectors from Misuse Cases with Deep Reinforcement Learning. Frontiersin Energy Research 11 (2023), 157.

Disclaimer

The content and views expressed in this material are those of the authors and do not necessarily reflect the views or opinion of the ERA-Net SES initiative. Any reference given does not necessarily imply the endorsement by ERA-Net SES.

About ERA-Net Smart Energy Systems | www.eranet-smartenergysystems.eu

The transnational joint programming platform (JPP) ERA-Net SES unites 30 funding partners from European and associated countries. It functions as a network of owners and managers of national and regional public funding programs in the field of research, technical development and demonstration. It provides a sustainable and service-oriented joint programming platform to finance transnational RDD projects, developing technologies and solutions in thematic areas like smart power grids, integrated regional and local energy systems, heating and cooling networks, digital energy and smart services, etc.