Game-theoretic APT defense: Overview of potentially problematic R code & behavior

Based on:
R-Code implementation:
https://github.com/jku-lit-scsl/ComputersAndSecurity_RoboticsCaseStudies_Cut-The-Rope
Title: Game-theoretic APT defense: An experimental study on robotics
Presented by: Beniamin Jablonski
Institute: Johannes Kepler Universität Linz
Date: 26.02.2025

Part 1: weight_value = 1 in ctr-core_1.R

- In Ctr-Core we have:

```
attack_graph <- set_edge_attr(attack_graph,
                              name = "weight",
                              index = get.edge.ids(attack_graph, edgelist),
                              value = 1)   # the virtual start is no obstacle towards the real entry point
```

- Which makes the claim that by setting the weight value of a new edge to 1 it makes it trivial to travel

Part 1: hardness_value = 1 in experiment_3.R & experiment_4.R

At the same time experiments 3 & 4 argue that a <u>**hardness value**</u> **of 1** makes takes the edge trivial to travel

```
# Handle NA values
hardness[is.na(hardness)] <- 1
# fix missing hardness values: if we know nothing,
# we consider the edge easy (trivial) to traverse
```

Can both be true?

How does the weight value of a given edge relate to hardness?

Part 1: The Mir100 Graph

We can find the answer in the “attack_graph_MIR100.R” file

There we can see that the hardness values (here called “edgeProbs”) are manually stored in the attack graph as “edge_probabilities” which later will be extracted as “hardness” values.

```
edgeProbs <- c(0.111265, 0.111265, 0.47287625, 0.47287625, 0.47287625,  0.3449215,
0.47287625, 1, 0.3449215, 0.47287625, 1, 1, 1, 0.47287625, 0.47287625, 0.47287625,
0.47287625, 0.47287625, 0.47287625, 0.47287625, 0.3449215,0.3449215, 0.3449215, 1)
```

```
attack_graph <- set_edge_attr(attack_graph,
                              name = "edge_probabilities",
                              index = E(attack_graph),
                              value = edgeProbs)
```

Later in experiments 3 & 4 file:

```
hardness <- edge_attr(attack_graph, "edge_probabilities", E(attack_graph, path=route))
hardness[is.na(hardness)] <- 1 # fix missing hardness values: if we know nothing, we consider
```

Part 1: The Mir100 Graph - weight is defined via edgeProbs!

At the same time we clearly can see how the weight values have been calculated!

```
attack_graph <- set_edge_attr(attack_graph,
                              name = "weight",
                              index = E(attack_graph),
                              value = -log(edgeProbs))
```

weight = -log(edge_probability)

Or conversely: edge_probability = exp(-weight)

Part 1: Conclusion & Fix

```
attack_graph <- set_edge_attr(attack_graph,
                              name = "weight",
                              index = get.edge.ids(attack_graph, edgelist),
                              value = 1)   # the virtual start is no obstacle towa
```

0

Conclusion:

- The way the graph has been defined we now know that
- hardness = edge_probability = exp(-weight)
- Thus setting the weight value to 1 in the ctr-core_1.R file was most likely an oversight
- Correct would have been to set it to 0 instead

Proposed fix:

- As long as we stick with this specific definition for the weights (weight = -log(edge_probability)), it only makes sense to set all default weight values to 0 for this graph because:
- Hardness = exp(-0) = 1

Alternatively one could modify randomSteps to always calculate hardness from weights directly:

- hardness <- **exp**(-edge_attr(attack_graph, "weight", E(attack_graph, path=route)))

Part 2: Which hardness/weight value does R use for later calculations **if multiple parallel edges with different weight values are present?**

Part 2: Which hardness/weight value does R use for later calculations if multiple edges with different weight values are present?

Mir100 Graph has 3 such Attack paths:

```
Path 3: [0, 3, 6, 8, 'c(12,13,14,16)']
  Parallel edges detected between 8 -> c(12,13,14,16):
    Edge key=0, weight=0.0
    Edge key=1, weight=0.7489220813074156
    *** These parallel edges have DIFFERENT weights ***
Path 5: [0, 3, 8, 'c(12,13,14,16)']
  Parallel edges detected between 8 -> c(12,13,14,16):
    Edge key=0, weight=0.0
    Edge key=1, weight=0.7489220813074156
    *** These parallel edges have DIFFERENT weights ***
Path 8: [0, 2, 11, 'c(12,13,14,16)']
  Parallel edges detected between 11 -> c(12,13,14,16):
    Edge key=0, weight=1.064439873679208
    Edge key=1, weight=0.7489220813074156
    Edge key=2, weight=0.0
    *** These parallel edges have DIFFERENT weights ***
```

- Since each of these paths has parallel edges with different weights….which of these does R use to calculate the entries in the Payoff matrix?

```
Payoff Matrix (probability of reaching target):
Row  1: 0.039406 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  2: 0.000000 0.000000 0.195410 0.000000 0.122740 0.000000 0.058041 0.122740 0.042335 0.000000 0.028743
Row  3: 0.044279 0.118219 0.083333 0.136853 0.122740 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  4: 0.044279 0.078813 0.000000 0.078813 0.000000 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  5: 0.044279 0.039406 0.195410 0.039406 0.122740 0.039406 0.058041 0.122740 0.042335 0.039406 0.028743
Row  6: 0.044279 0.171217 0.195410 0.136853 0.122740 0.078813 0.000000 0.122740 0.042335 0.092405 0.028743
Row  7: 0.044279 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.000000 0.042335 0.092405 0.028743
Row  8: 0.044279 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.122740 0.000000 0.092405 0.028743
```

Part 2: Inconsistency in Experiment 3

- Original Payoff Matrix

- With hardness = exp(-min_weight) we see a discrepancy for attack paths 3 & 5 (blue)

- With hardness = exp(-max_weight) we see a discrepancy for attack path 8 (yellow)

Pay-off matrix in R:

```
Payoff Matrix (probability of reaching target):
Row  1: 0.039199 0.170457 0.091827 0.136217 0.057858 0.105744 0.057858 0.121826 0.042338 0.091950 0.028909
Row  2: 0.000553 0.002182 0.091827 0.001719 0.057858 0.001269 0.057858 0.121826 0.042338 0.001147 0.028909
Row  3: 0.044047 0.117656 0.039444 0.136217 0.057858 0.105744 0.057858 0.121826 0.042338 0.091950 0.028909
Row  4: 0.044047 0.078418 0.000518 0.078418 0.000518 0.105744 0.057858 0.121826 0.042338 0.091950 0.028909
Row  5: 0.044047 0.039199 0.091827 0.039199 0.057858 0.039199 0.057858 0.121826 0.042338 0.039199 0.028909
Row  6: 0.044047 0.170457 0.091827 0.136217 0.057858 0.078418 0.000518 0.121826 0.042338 0.091950 0.028909
Row  7: 0.044047 0.170457 0.091827 0.136217 0.057858 0.105744 0.057858 0.000518 0.042338 0.091950 0.028909
Row  8: 0.044047 0.170457 0.091827 0.136217 0.057858 0.105744 0.057858 0.121826 0.000518 0.091950 0.028909
```

Pay-off Matrix in Python with: hardness = exp(-min_weight)

```
Payoff Matrix (probability of reaching target):
Row  1: 0.039406 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  2: 0.000000 0.000000 0.195410 0.000000 0.122740 0.000000 0.058041 0.122740 0.042335 0.000000 0.028743
Row  3: 0.044279 0.118219 0.083333 0.136853 0.122740 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  4: 0.044279 0.078813 0.000000 0.078813 0.000000 0.106259 0.058041 0.122740 0.042335 0.092405 0.028743
Row  5: 0.044279 0.039406 0.195410 0.039406 0.122740 0.039406 0.058041 0.122740 0.042335 0.039406 0.028743
Row  6: 0.044279 0.171217 0.195410 0.136853 0.122740 0.078813 0.000000 0.122740 0.042335 0.092405 0.028743
Row  7: 0.044279 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.000000 0.042335 0.092405 0.028743
Row  8: 0.044279 0.171217 0.195410 0.136853 0.122740 0.106259 0.058041 0.122740 0.000000 0.092405 0.028743
```

Pay-off Matrix in Python with hardness = exp(-max_weight)

```
Payoff Matrix (probability of reaching target):
Row  1: 0.039406 0.171217 0.092405 0.136853 0.058041 0.106259 0.058041 0.042335 0.042335 0.092405 0.028743
Row  2: 0.000000 0.000000 0.092405 0.000000 0.058041 0.000000 0.058041 0.042335 0.042335 0.000000 0.028743
Row  3: 0.044279 0.118219 0.039406 0.136853 0.058041 0.106259 0.058041 0.042335 0.042335 0.092405 0.028743
Row  4: 0.044279 0.078813 0.000000 0.078813 0.000000 0.106259 0.058041 0.042335 0.042335 0.092405 0.028743
Row  5: 0.044279 0.039406 0.092405 0.039406 0.058041 0.039406 0.058041 0.042335 0.042335 0.039406 0.028743
Row  6: 0.044279 0.171217 0.092405 0.136853 0.058041 0.078813 0.000000 0.042335 0.042335 0.092405 0.028743
Row  7: 0.044279 0.171217 0.092405 0.136853 0.058041 0.106259 0.058041 0.000000 0.042335 0.092405 0.028743
Row  8: 0.044279 0.171217 0.092405 0.136853 0.058041 0.106259 0.058041 0.042335 0.000000 0.092405 0.028743
```

Part 2: Where does this discrepancy come from?

The main suspect: E(....) method in R

```
# steps determined by hardness to exploit
randomSteps <- function(route, attackRate = NULL, defenseRate = NULL) {
  hardness <- edge_attr(attack_graph, "edge_probabilities", E(attack_graph, path=route))
```

Documentation: When using E(....) with path parameter on graphs with parallel edges, **it arbitrarily selects one edge instead of returning all matching edges, causing inconsistent behavior in the randomSteps function.**

Source: https://igraph.org/r/doc/E.html (see under Path Argument)

Part 2: Conclusion & Proposed Solution

- Conclusion:
  - The E(...) method can not be trusted for our graphs with parallel edges

- Solution
  - Since a lower weight translates to a higher probability of success (remember hardness = exp(-w)) it makes sense that in the presence of multiple parallel edges the attacker would pick the **edge with the lowest weight.**
  - **Thus hardness = exp(-min_weight) in python implementation**

```
# Yes hardness of 1 means path is trivial, hardness 0 means path is impossible
hardness.append(np.exp(-min_weight))
```