Root-Cause Analysis through Reinforcement Learning
To
AFLRL (AMERICAN FUZZY LOP FOR REINFORCEMENT LEARNING)

By KP

Overview

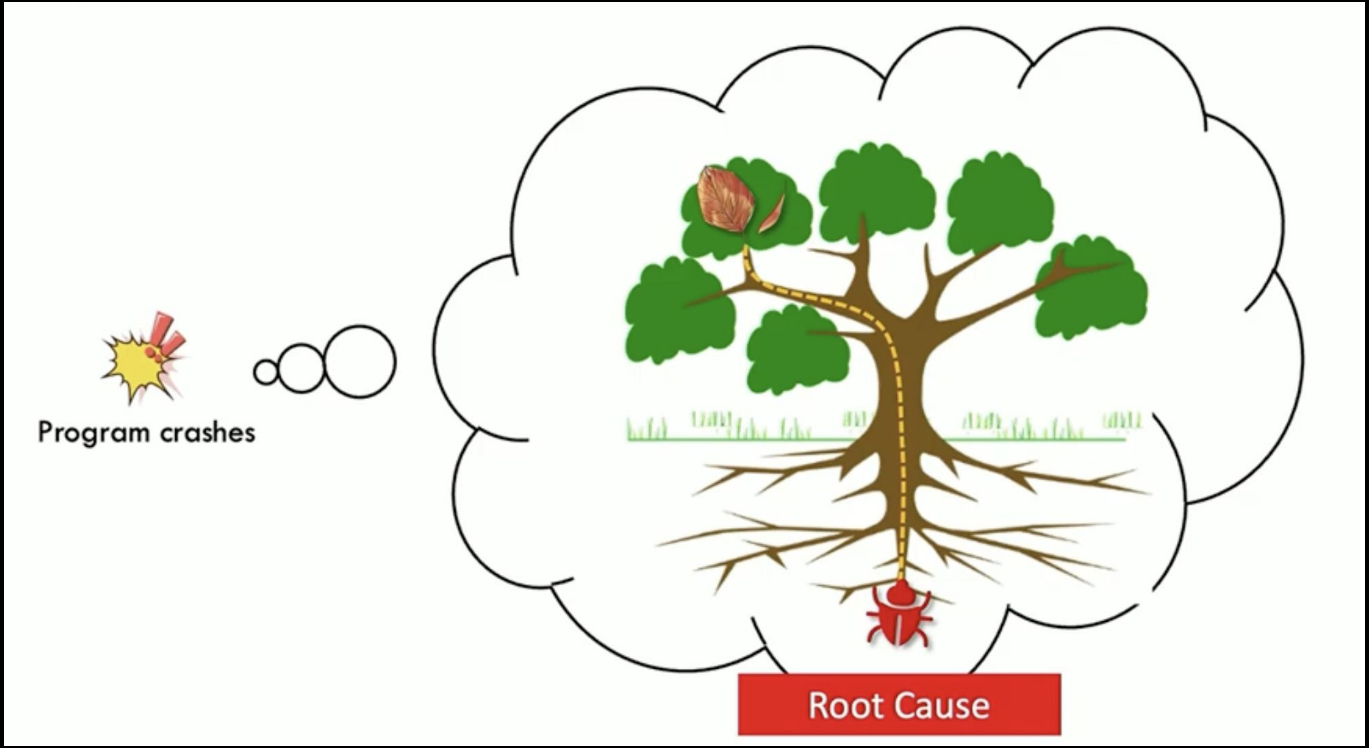
- RCA: For finding security vulnerabilities through fuzzing.
- RCA is slow, I mean very slow.
- Proper mutation for sample needed.
- Reinforcement Learning Implementation.
- Counterexamples are the key.
- Approach of using RACING increasing scalability and better than today's RCA.

Introduction

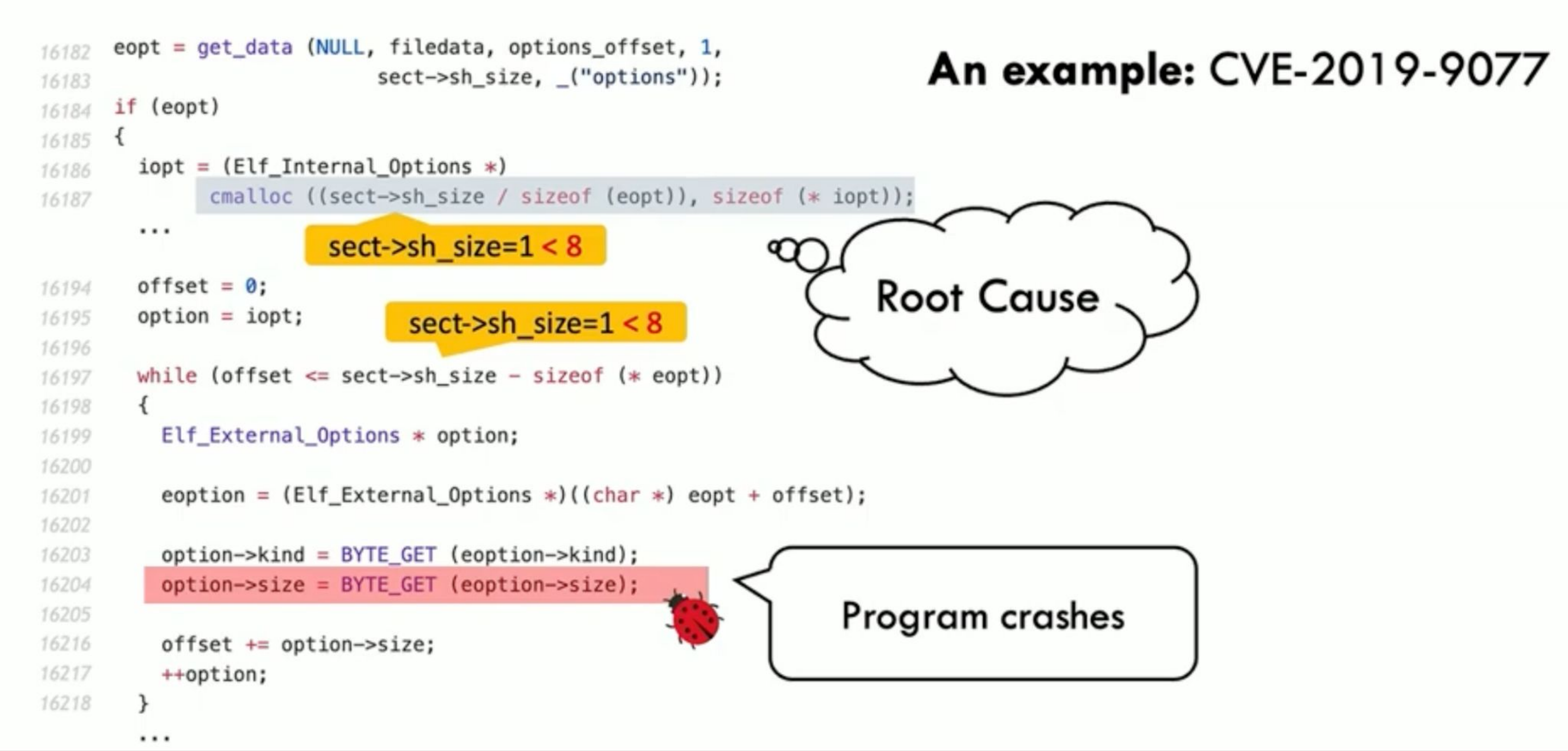
- Need for automation for faster devops cycle.
- CVE-2017-5380, CVE-2018-4145 and CVE-2022-36320.
- Try to find the crash is the key
- How to achieve that? Through mutation of test cases.
- Root cause analysis which has been done manually.

SLow SLow Slowwwww.......

RCA



Example



An example: CVE-2019-9077

```
eopt = get_data (NULL, filedata, options_offset, 1,
                      sect->sh_size, _("options"));
if (eopt)
{
  iopt = (Elf_Internal_Options *)
      cmalloc ((sect->sh_size / sizeof (eopt)), sizeof (* iopt));
  ...

  offset = 0;
  option = iopt;

  while (offset <= sect->sh_size - sizeof (* eopt))
  {
    Elf_External_Options * option;

    eoption = (Elf_External_Options *)((char *) eopt + offset);

    option->kind = BYTE_GET (eoption->kind);
    option->size = BYTE_GET (eoption->size);

    offset += option->size;
    ++option;
  }
  ...
```

sect->sh_size=1 < 8

sect->sh_size=1 < 8

Root Cause

Program crashes

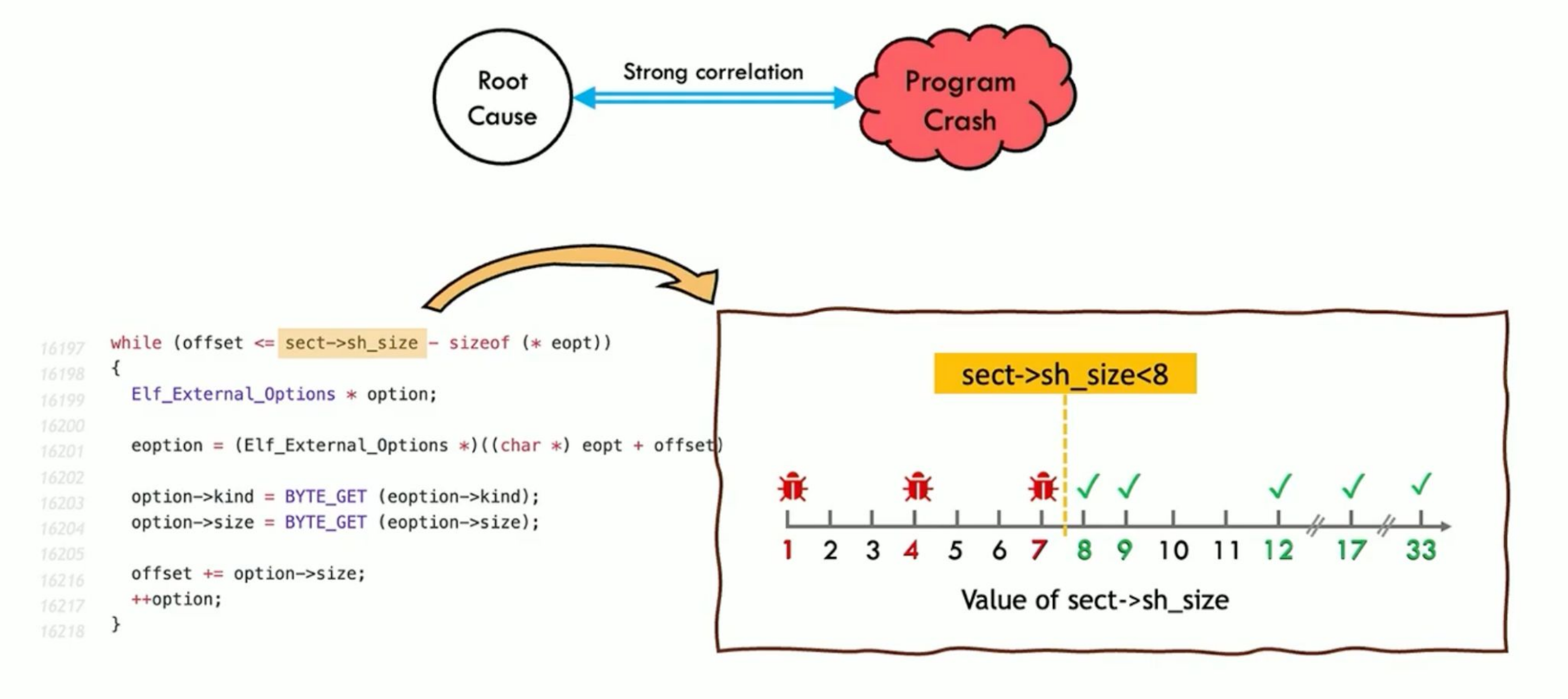
Automated RCA Challenges

- Having a pipeline with bucket of analysis tools.
- They rely on rules and types for finding bug.
- Lead to type-confusion flaws.

Spectrum Based Fault Localization (SFL)

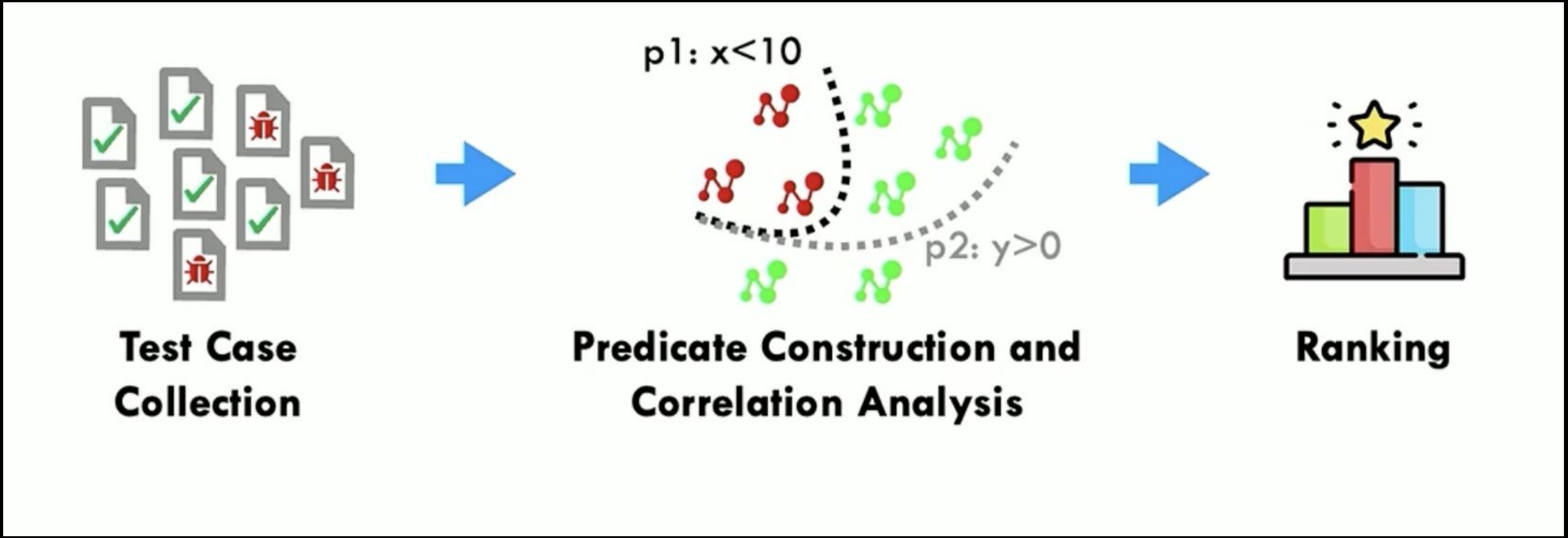
- Finding bug without explicit data dependency.
- They outperform traditional automated RCA.
- They try to correlate crash and root cause through statistics.
- Ranking highly correlated test case and find the crash.
- Takes lot of time 12 hours or even week.

Statistical RCA



```
while (offset <= sect->sh_size - sizeof (* eopt))
{
  Elf_External_Options * option;

  eoption = (Elf_External_Options *)((char *) eopt + offset)

  option->kind = BYTE_GET (eoption->kind);
  option->size = BYTE_GET (eoption->size);

  offset += option->size;
  ++option;
}
```

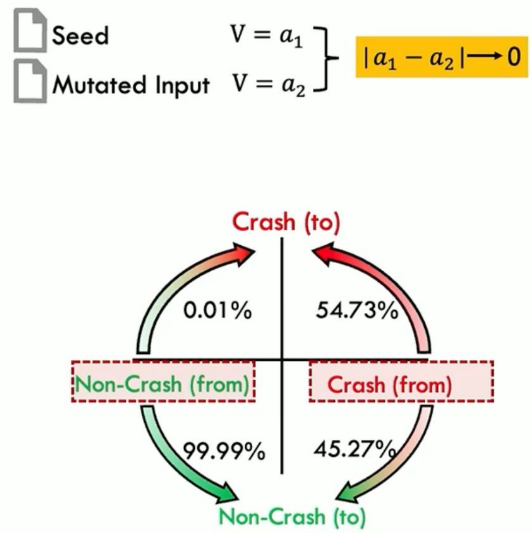
General Workflow of Statistical RCA

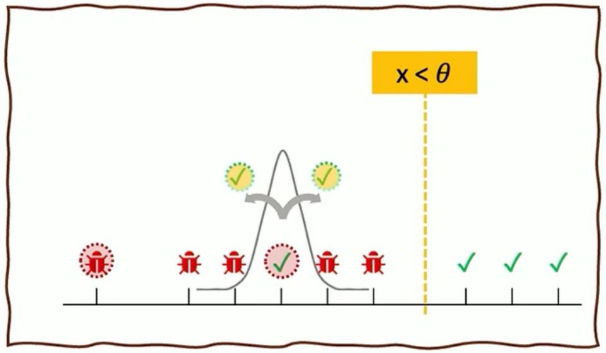


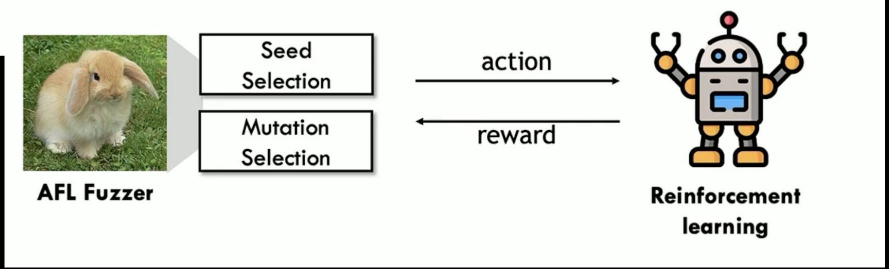
Racing

What is Racing? Root-cAuse-analysis on Counter-examples based reinforcement-learnING.....

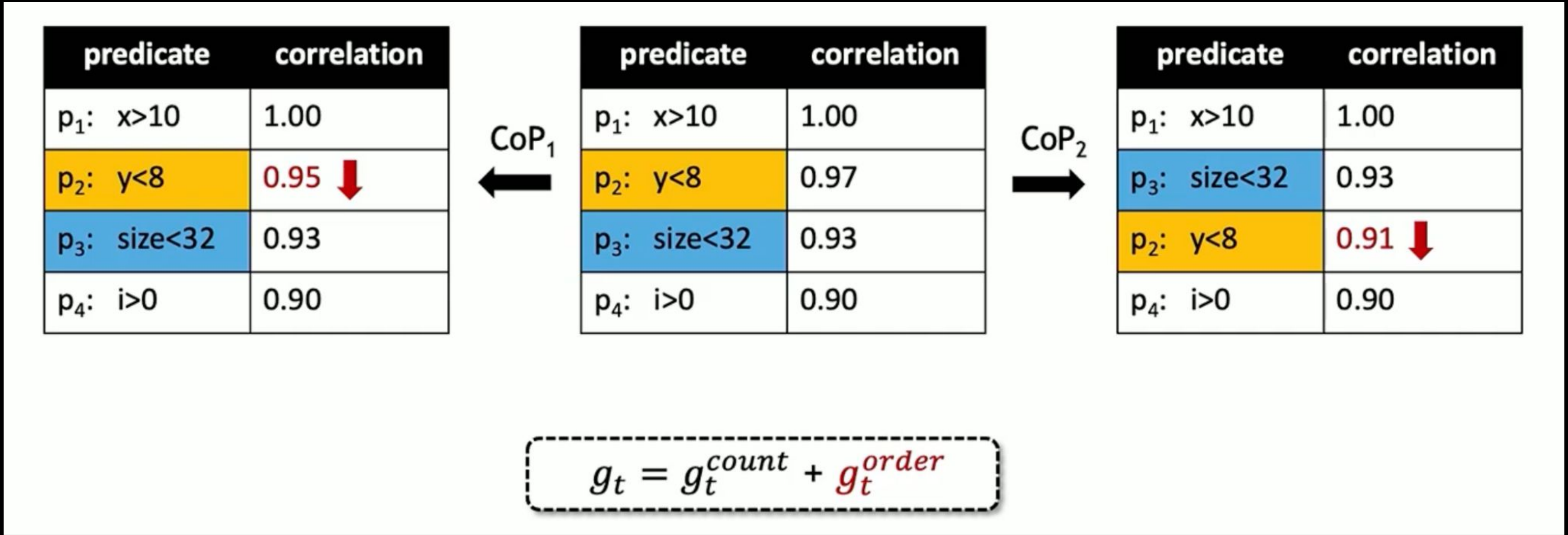
How to mutate test case?







All about Correlation



predicate	correlation
p_1: x>10	1.00
p_2: y<8	0.95 ↓
p_3: size<32	0.93
p_4: i>0	0.90

$\xleftarrow{CoP_1}$

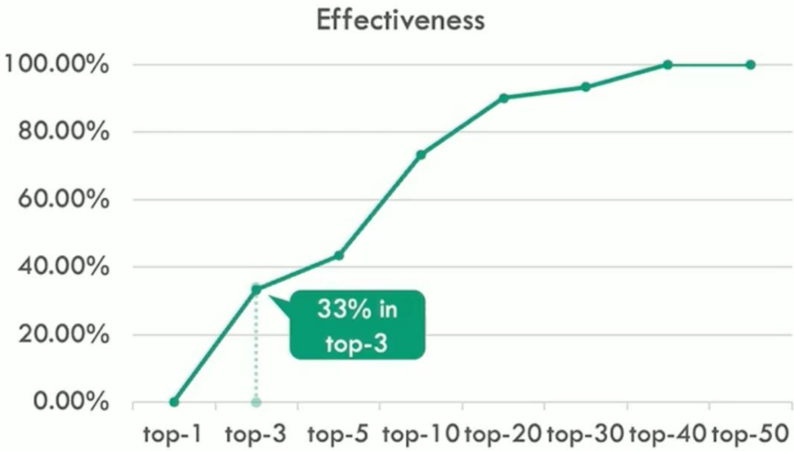
predicate	correlation
p_1: x>10	1.00
p_2: y<8	0.97
p_3: size<32	0.93
p_4: i>0	0.90

$\xrightarrow{CoP_2}$

predicate	correlation
p_1: x>10	1.00
p_3: size<32	0.93
p_2: y<8	0.91 ↓
p_4: i>0	0.90

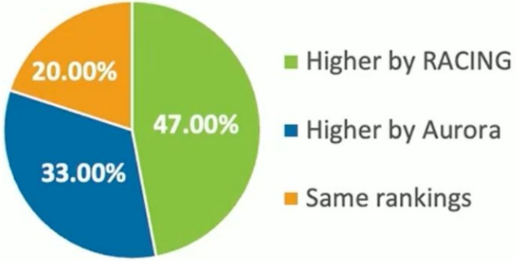
$$g_t = g_t^{count} + g_t^{order}$$

Existing Results

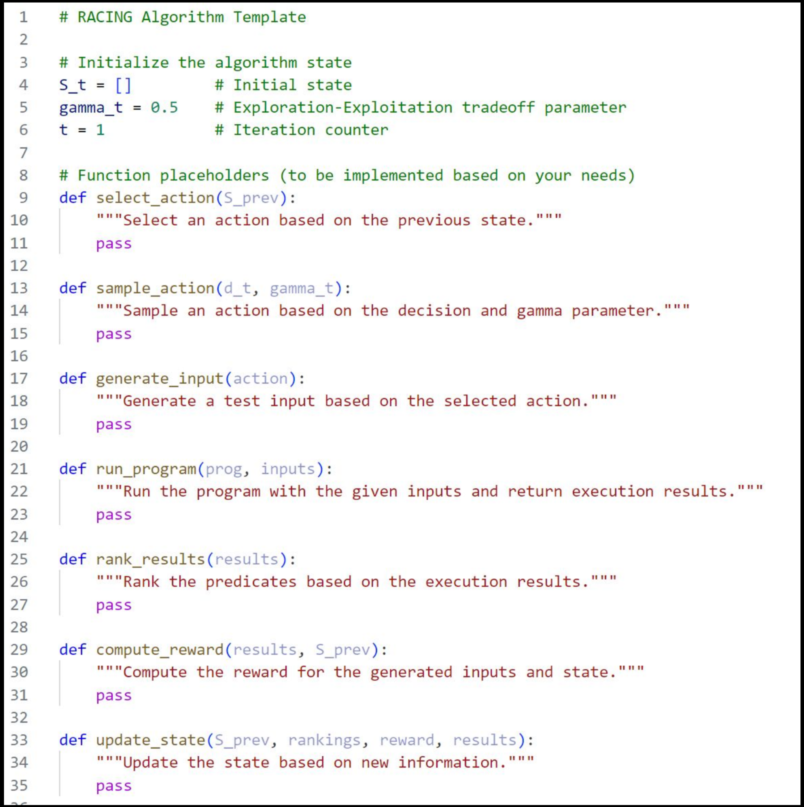


Effectiveness

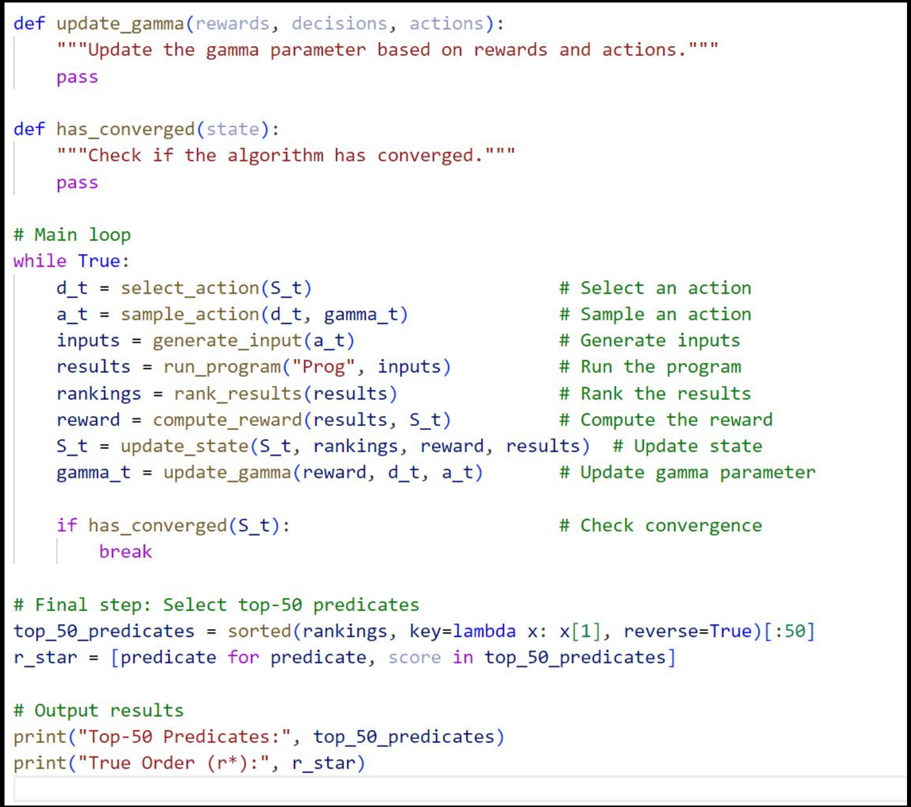
33% in top-3



Algorithm code

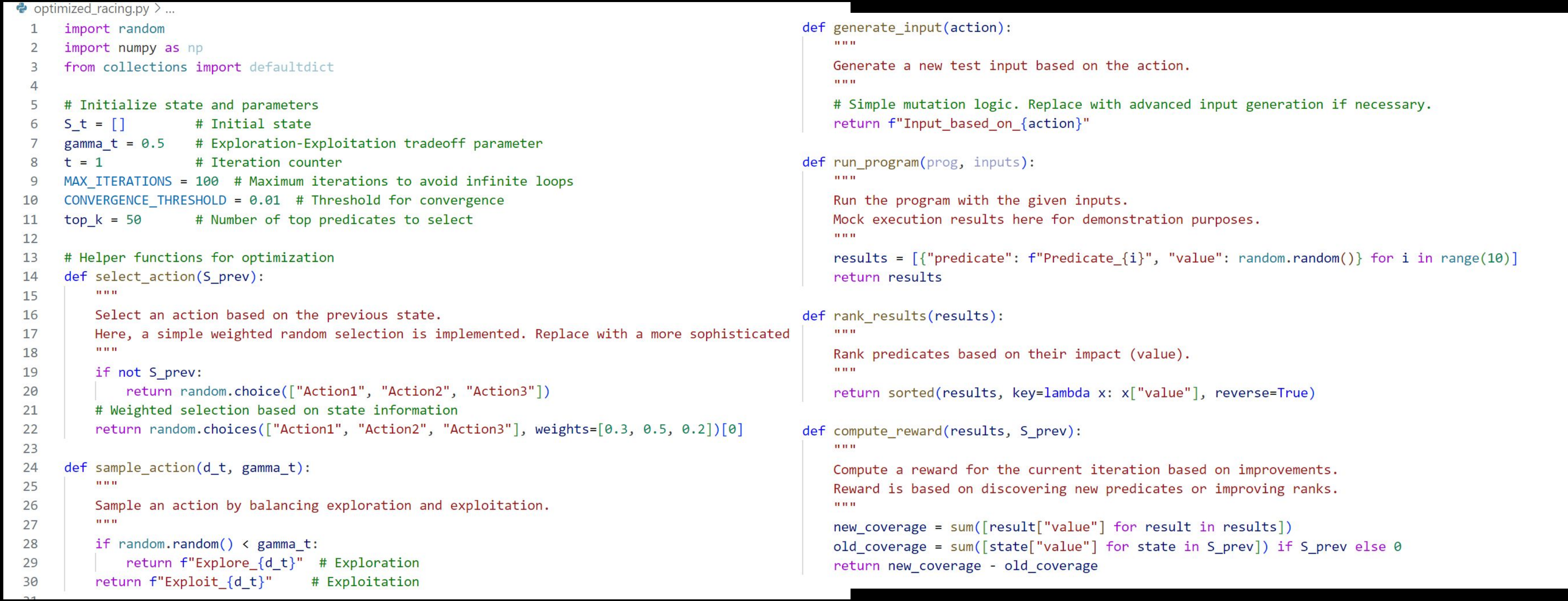


```python
# RACING Algorithm Template

# Initialize the algorithm state
S_t = []          # Initial state
gamma_t = 0.5     # Exploration-Exploitation tradeoff parameter
t = 1             # Iteration counter

# Function placeholders (to be implemented based on your needs)
def select_action(S_prev):
    """Select an action based on the previous state."""
    pass

def sample_action(d_t, gamma_t):
    """Sample an action based on the decision and gamma parameter."""
    pass

def generate_input(action):
    """Generate a test input based on the selected action."""
    pass

def run_program(prog, inputs):
    """Run the program with the given inputs and return execution results."""
    pass

def rank_results(results):
    """Rank the predicates based on the execution results."""
    pass

def compute_reward(results, S_prev):
    """Compute the reward for the generated inputs and state."""
    pass

def update_state(S_prev, rankings, reward, results):
    """Update the state based on new information."""
    pass
```



```python
def update_gamma(rewards, decisions, actions):
    """Update the gamma parameter based on rewards and actions."""
    pass

def has_converged(state):
    """Check if the algorithm has converged."""
    pass

# Main loop
while True:
    d_t = select_action(S_t)                              # Select an action
    a_t = sample_action(d_t, gamma_t)                     # Sample an action
    inputs = generate_input(a_t)                          # Generate inputs
    results = run_program("Prog", inputs)                 # Run the program
    rankings = rank_results(results)                      # Rank the results
    reward = compute_reward(results, S_t)                 # Compute the reward
    S_t = update_state(S_t, rankings, reward, results)  # Update state
    gamma_t = update_gamma(reward, d_t, a_t)              # Update gamma parameter

    if has_converged(S_t):                                # Check convergence
        break

# Final step: Select top-50 predicates
top_50_predicates = sorted(rankings, key=lambda x: x[1], reverse=True)[:50]
r_star = [predicate for predicate, score in top_50_predicates]

# Output results
print("Top-50 Predicates:", top_50_predicates)
print("True Order (r*):", r_star)
```

Optimization in a code

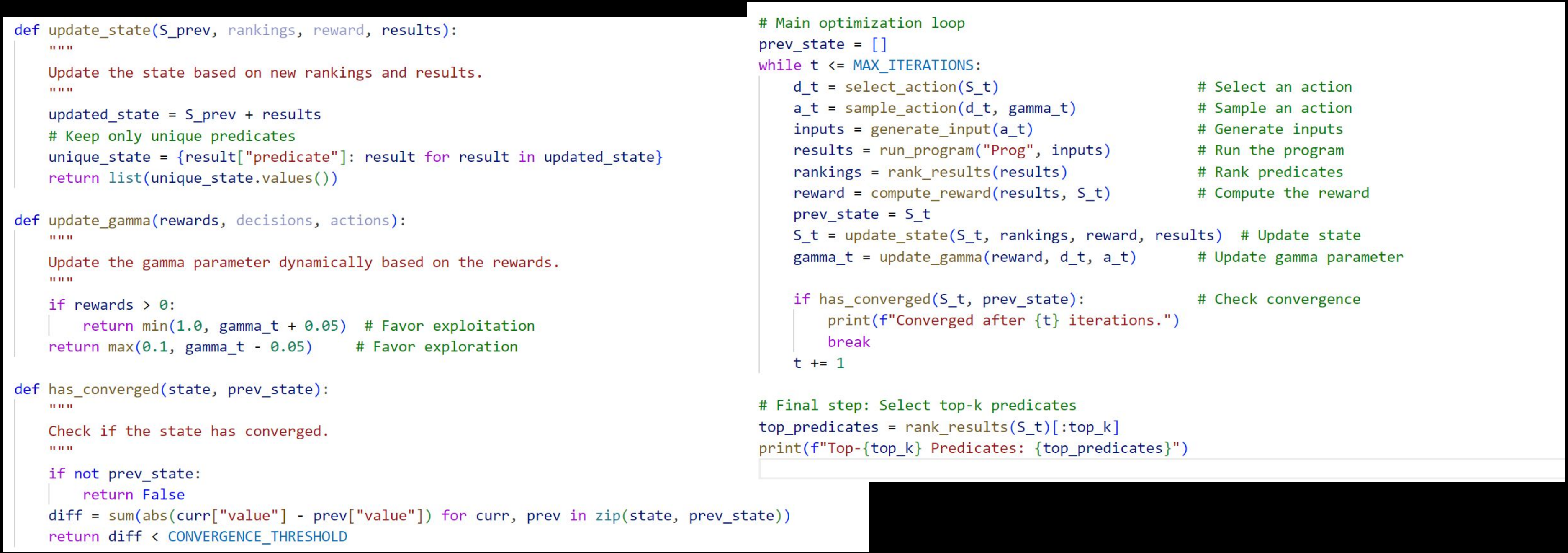


optimized_racing.py > ...

```python
import random
import numpy as np
from collections import defaultdict

# Initialize state and parameters
S_t = []         # Initial state
gamma_t = 0.5    # Exploration-Exploitation tradeoff parameter
t = 1            # Iteration counter
MAX_ITERATIONS = 100  # Maximum iterations to avoid infinite loops
CONVERGENCE_THRESHOLD = 0.01  # Threshold for convergence
top_k = 50       # Number of top predicates to select

# Helper functions for optimization
def select_action(S_prev):
    """
    Select an action based on the previous state.
    Here, a simple weighted random selection is implemented. Replace with a more sophisticated
    """
    if not S_prev:
        return random.choice(["Action1", "Action2", "Action3"])
    # Weighted selection based on state information
    return random.choices(["Action1", "Action2", "Action3"], weights=[0.3, 0.5, 0.2])[0]

def sample_action(d_t, gamma_t):
    """
    Sample an action by balancing exploration and exploitation.
    """
    if random.random() < gamma_t:
        return f"Explore_{d_t}"  # Exploration
    return f"Exploit_{d_t}"     # Exploitation
```

```python
def generate_input(action):
    """
    Generate a new test input based on the action.
    """
    # Simple mutation logic. Replace with advanced input generation if necessary.
    return f"Input_based_on_{action}"

def run_program(prog, inputs):
    """
    Run the program with the given inputs.
    Mock execution results here for demonstration purposes.
    """
    results = [{"predicate": f"Predicate_{i}", "value": random.random()} for i in range(10)]
    return results

def rank_results(results):
    """
    Rank predicates based on their impact (value).
    """
    return sorted(results, key=lambda x: x["value"], reverse=True)

def compute_reward(results, S_prev):
    """
    Compute a reward for the current iteration based on improvements.
    Reward is based on discovering new predicates or improving ranks.
    """
    new_coverage = sum([result["value"] for result in results])
    old_coverage = sum([state["value"] for state in S_prev]) if S_prev else 0
    return new_coverage - old_coverage
```

Continuation

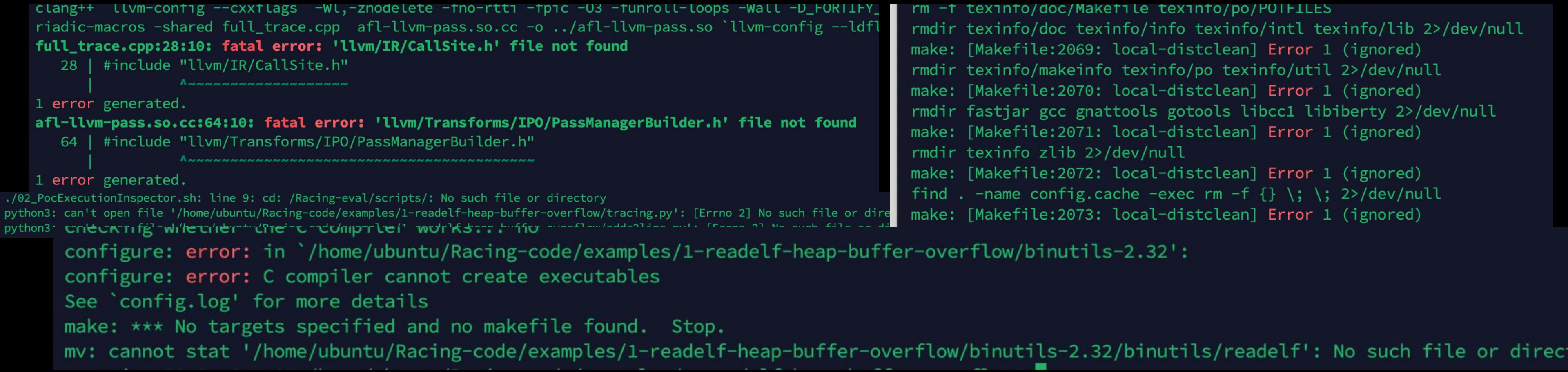


```
def update_state(S_prev, rankings, reward, results):
    """
    Update the state based on new rankings and results.
    """
    updated_state = S_prev + results
    # Keep only unique predicates
    unique_state = {result["predicate"]: result for result in updated_state}
    return list(unique_state.values())

def update_gamma(rewards, decisions, actions):
    """
    Update the gamma parameter dynamically based on the rewards.
    """
    if rewards > 0:
        return min(1.0, gamma_t + 0.05)  # Favor exploitation
    return max(0.1, gamma_t - 0.05)     # Favor exploration

def has_converged(state, prev_state):
    """
    Check if the state has converged.
    """
    if not prev_state:
        return False
    diff = sum(abs(curr["value"] - prev["value"]) for curr, prev in zip(state, prev_state))
    return diff < CONVERGENCE_THRESHOLD
```

```
# Main optimization loop
prev_state = []
while t <= MAX_ITERATIONS:
    d_t = select_action(S_t)                       # Select an action
    a_t = sample_action(d_t, gamma_t)              # Sample an action
    inputs = generate_input(a_t)                   # Generate inputs
    results = run_program("Prog", inputs)          # Run the program
    rankings = rank_results(results)               # Rank predicates
    reward = compute_reward(results, S_t)          # Compute the reward
    prev_state = S_t
    S_t = update_state(S_t, rankings, reward, results)  # Update state
    gamma_t = update_gamma(reward, d_t, a_t)       # Update gamma parameter

    if has_converged(S_t, prev_state):             # Check convergence
        print(f"Converged after {t} iterations.")
        break
    t += 1

# Final step: Select top-k predicates
top_predicates = rank_results(S_t)[:top_k]
print(f"Top-{top_k} Predicates: {top_predicates}")
```

Unexpected Turn

- The original code base from authors were not producing proper results.



```
clang++ `llvm-config --cxxflags` -Wl,-znodelete -fno-rtti -fpic -O3 -funroll-loops -Wall -D_FORTIFY_
riadic-macros -shared full_trace.cpp  afl-llvm-pass.so.cc -o ../afl-llvm-pass.so `llvm-config --ldfl
full_trace.cpp:28:10: fatal error: 'llvm/IR/CallSite.h' file not found
   28 | #include "llvm/IR/CallSite.h"
      |          ^~~~~~~~~~~~~~~~~~~~
1 error generated.
afl-llvm-pass.so.cc:64:10: fatal error: 'llvm/Transforms/IPO/PassManagerBuilder.h' file not found
   64 | #include "llvm/Transforms/IPO/PassManagerBuilder.h"
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 error generated.
./02_PocExecutionInspector.sh: line 9: cd: /Racing-eval/scripts/: No such file or directory
python3: can't open file '/home/ubuntu/Racing-code/examples/1-readelf-heap-buffer-overflow/tracing.py': [Errno 2] No such file or dire
python3: can't open file '/home/ubuntu/Racing-code/examples/1-readelf-heap-buffer-overflow/addr2line.py': [Errno 2] No such file or di
```

```
rm -f texinfo/doc/Makefile texinfo/po/POTFILES
rmdir texinfo/doc texinfo/info texinfo/intl texinfo/lib 2>/dev/null
make: [Makefile:2069: local-distclean] Error 1 (ignored)
rmdir texinfo/makeinfo texinfo/po texinfo/util 2>/dev/null
make: [Makefile:2070: local-distclean] Error 1 (ignored)
rmdir fastjar gcc gnattools gotools libcc1 libiberty 2>/dev/null
make: [Makefile:2071: local-distclean] Error 1 (ignored)
rmdir texinfo zlib 2>/dev/null
make: [Makefile:2072: local-distclean] Error 1 (ignored)
find . -name config.cache -exec rm -f {} \; \; 2>/dev/null
make: [Makefile:2073: local-distclean] Error 1 (ignored)
```

```
checking whether the C compiler works... no
configure: error: in `/home/ubuntu/Racing-code/examples/1-readelf-heap-buffer-overflow/binutils-2.32':
configure: error: C compiler cannot create executables
See `config.log' for more details
make: *** No targets specified and no makefile found.  Stop.
mv: cannot stat '/home/ubuntu/Racing-code/examples/1-readelf-heap-buffer-overflow/binutils-2.32/binutils/readelf': No such file or direc
```

- Because of these errors and file issues, I changed the direction of my project.

Building AFLRL

New Focus:

- **"AFLRL"**: Training reinforcement learning models using AFL (American Fuzzy Lop).
- Build datasets for RL model training.
- Enhance AFL’s capabilities by adding vulnerable code paths to generate robust training data.

Updated Objective

Dataset Creation:

- Generate meaningful datasets for training RL models using AFL.

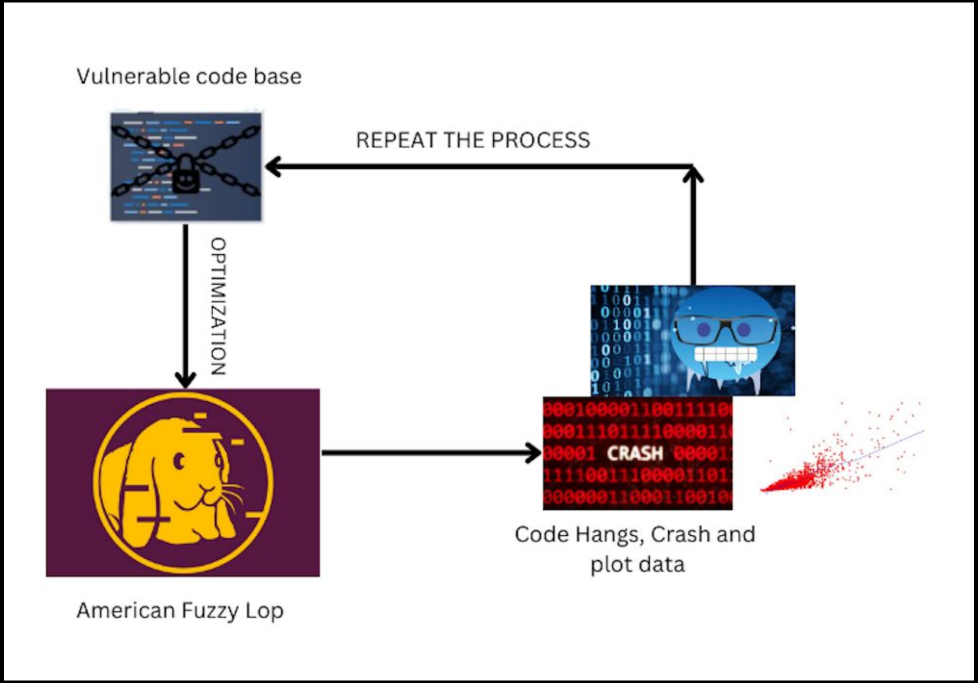
Optimization:

- Focus on AFL’s features to produce proper outputs for RL training.

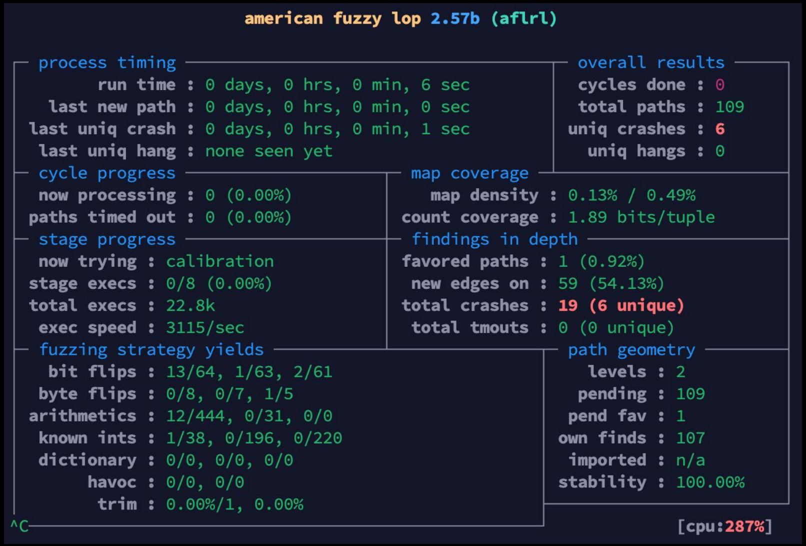
Outcome:

- Improve path exploration and crash detection efficiency in fuzz testing.

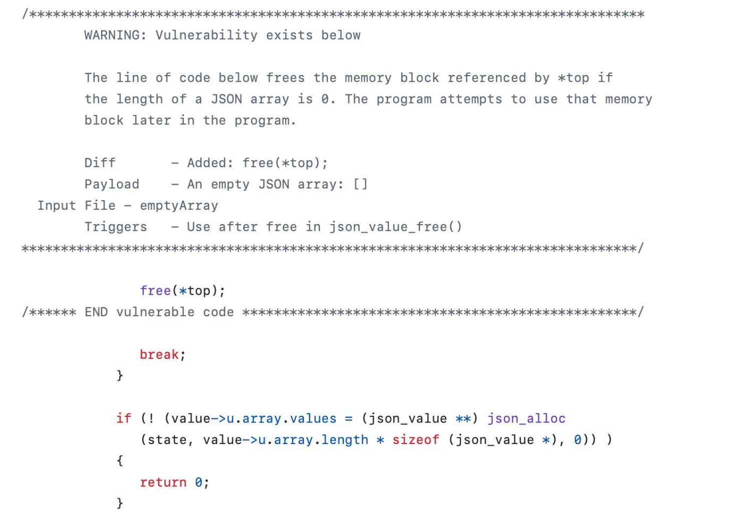
Flowchart


Vulnerable code base
REPEAT THE PROCESS
OPTIMIZATION
American Fuzzy Lop
CRASH
Code Hangs, Crash and plot data

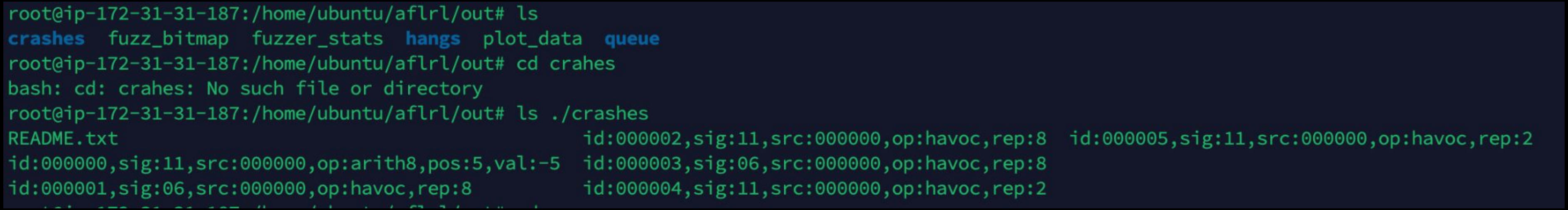
OUTPUT



```
                  american fuzzy lop 2.57b (aflrl)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 0 min, 6 sec        │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec        │  total paths : 109    │
│ last uniq crash : 0 days, 0 hrs, 0 min, 1 sec        │ uniq crashes : 6      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 0 (0.00%)         │    map density : 0.13% / 0.49%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.89 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : calibration           │ favored paths : 1 (0.92%)              │
│ stage execs : 0/8 (0.00%)           │  new edges on : 59 (54.13%)            │
│ total execs : 22.8k                 │ total crashes : 19 (6 unique)          │
│  exec speed : 3115/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 13/64, 1/63, 2/61                     │    levels : 2          │
│  byte flips : 0/8, 0/7, 1/5                         │   pending : 109        │
│ arithmetics : 12/444, 0/31, 0/0                     │  pend fav : 1          │
│  known ints : 1/38, 0/196, 0/220                    │ own finds : 107        │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 0/0, 0/0                              │ stability : 100.00%    │
│        trim : 0.00%/1, 0.00%                        ├────────────────────────┘
^C────────────────────────────────────────────────────┘          [cpu:287%]
```



```
/*******************************************************************************
        WARNING: Vulnerability exists below

        The line of code below frees the memory block referenced by *top if
        the length of a JSON array is 0. The program attempts to use that memory
        block later in the program.

        Diff       - Added: free(*top);
        Payload    - An empty JSON array: []
  Input File - emptyArray
        Triggers   - Use after free in json_value_free()
*******************************************************************************/

                  free(*top);
/****** END vulnerable code ***************************************************/

                  break;
              }

              if (! (value->u.array.values = (json_value **) json_alloc
                  (state, value->u.array.length * sizeof (json_value *), 0)) )
              {
                  return 0;
              }
```



```
root@ip-172-31-31-187:/home/ubuntu/aflrl/out# ls
crashes  fuzz_bitmap  fuzzer_stats  hangs  plot_data  queue
root@ip-172-31-31-187:/home/ubuntu/aflrl/out# cd crahes
bash: cd: crahes: No such file or directory
root@ip-172-31-31-187:/home/ubuntu/aflrl/out# ls ./crashes
README.txt                                  id:000002,sig:11,src:000000,op:havoc,rep:8  id:000005,sig:11,src:000000,op:havoc,rep:2
id:000000,sig:11,src:000000,op:arith8,pos:5,val:-5  id:000003,sig:06,src:000000,op:havoc,rep:8
id:000001,sig:06,src:000000,op:havoc,rep:8  id:000004,sig:11,src:000000,op:havoc,rep:2
```

Challenges and Future work

- Environmental and version mismatches
- Transition from RACING to AFLRL required some work.

Future work

- Train RL with generated crashes.
- Optimize AFL for better vulnerability detection and RL integration.

Thank You