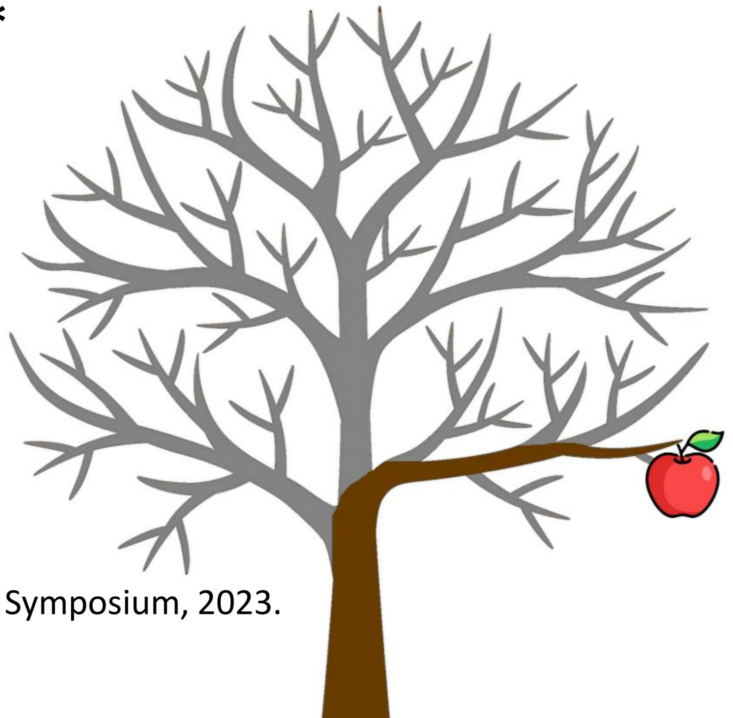
2024. 05. 30. Project Final Talk

Visualizing Fuzzing Status on Def-Use Graph and its impact

Geon Park
KAIST Programming Systems Laboratory

Directed Fuzzing

- **Fuzzing** Tests programs through randomly generated inputs.
  - e.g., Google's OSS Fuzz project, AFL
- **Directed fuzzing** aims to reach target location(s) of code
  - e.g., Examine recently changed code area, generate crashing input from bug report
- Reproduces bug 1.93 times faster than undirected fuzzing*
  - Undirected fuzzing: AFL, directed fuzzing: DAFL



* Tae Eun Kim et al., DAFL: Directed Grey-box Fuzzing guided by Data Dependency. USENIX Security Symposium, 2023.

Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.

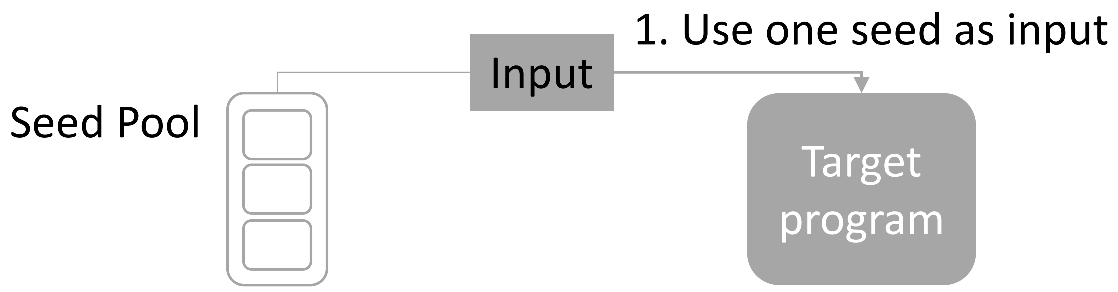
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.

Seed Pool
Target program

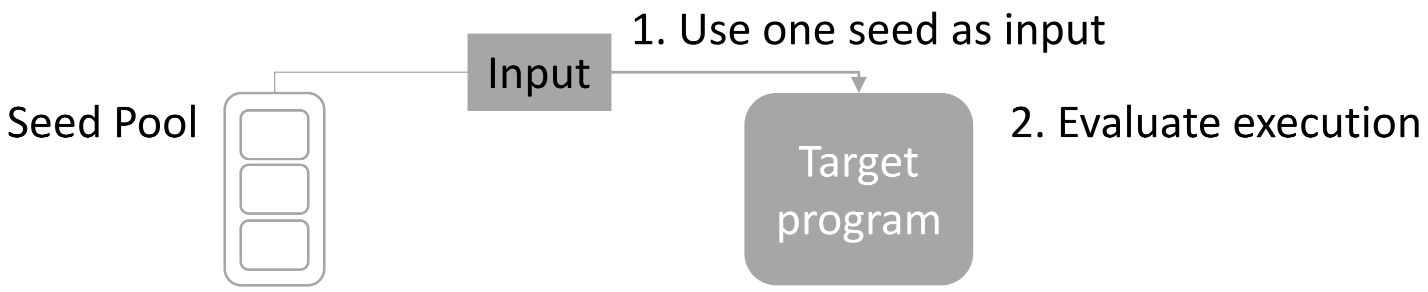
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



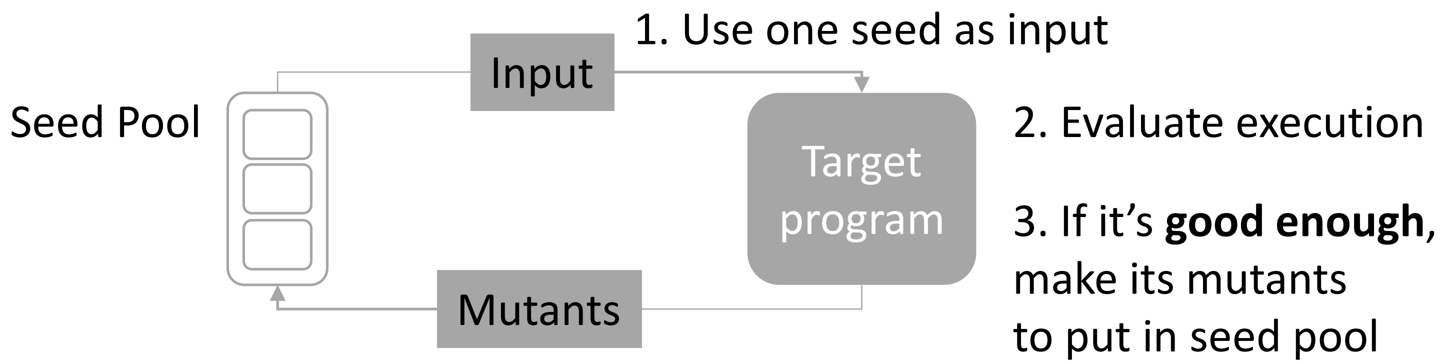
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



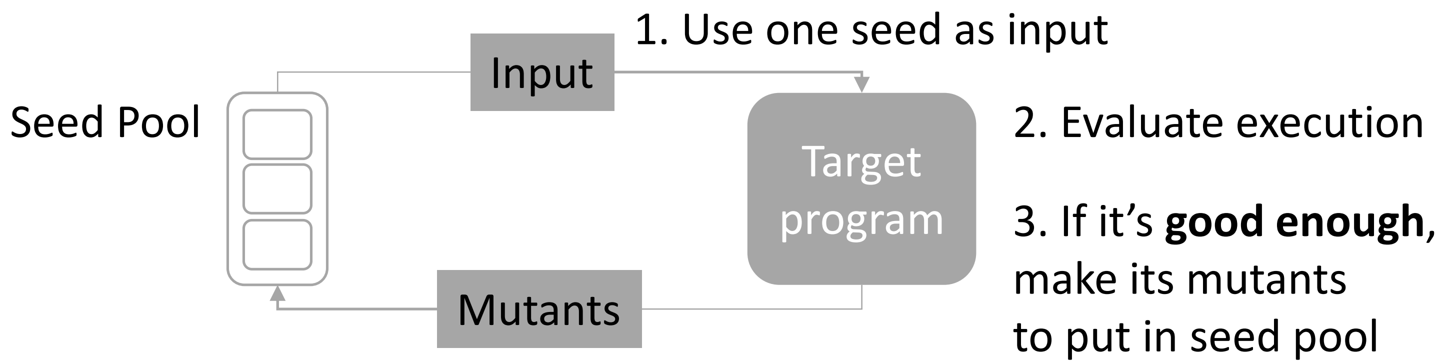
Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



Developing Fuzzing

- Performance varies based on how fuzzing guidance is given and used.



- What is the criteria for the execution to be **good enough**?

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print(‘0’)
end if
if (flag == 1) then
   print(‘1’)
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print('0')
end if
if (flag == 1) then
   print('1')
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print('0')
end if
if (flag == 1) then
   print('1')
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print('0')
end if
if (flag == 1) then
   print('1')
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print('0')
end if
if (flag == 1) then
   print('1')
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important

```
def getSize(width, height, some_data):
if (some_data) then
   doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

```
def doSomething():
if (flag == 0) then
   print('0')
end if
if (flag == 1) then
   print('1')
end if
…
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important

```
def getSize(width, height, some_data):
if (some_data) then
  doSomething() // 1000 LoC
end if
print("Size is", width × height)
```

Developing Fuzzing

- For undirected **fuzzing**, increasing code coverage is considered important
- For **directed fuzzing**, data-flow analysis is important
  - Data-flow analysis is for what part of code effects data used in target line
  - In example, doSomething() does not effect data for line 5
  - Increasing code coverage of doSomething() does not affect performance

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething() // 1000 LoC
4:  end if
5:  print("Size is", width × height)
```

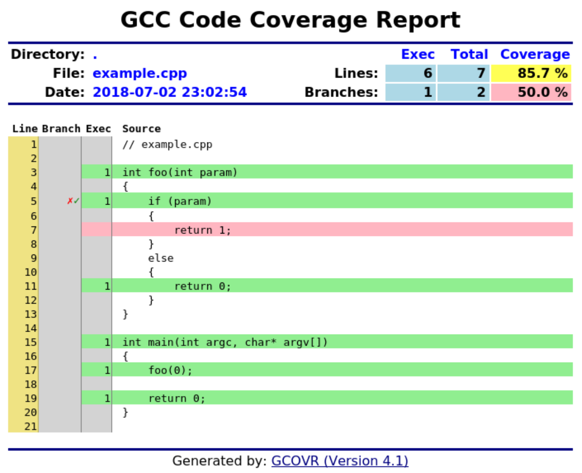
width: defined at 1, used at 5

height: defined at 1, used at 5

some_data: defined at 1, used at 2

Challenge 1

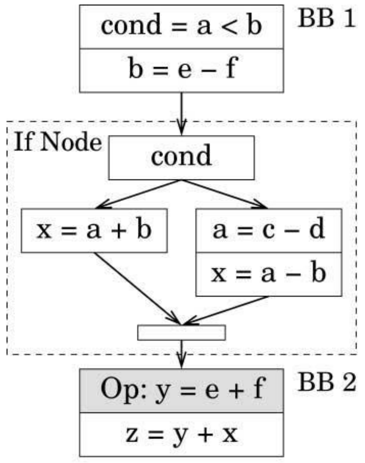
	Existing coverage inspectors	Need for directed fuzzing inspector
Based on	lines	basic blocks
Purpose	see where / how much are (un)covered	see how far the fuzzing reached toward target



GCC Code Coverage Report

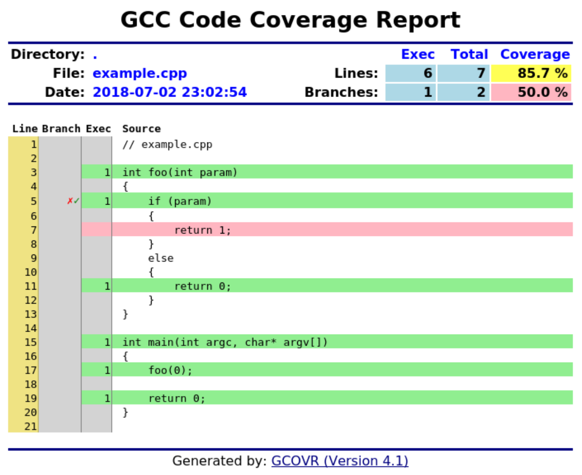
```
Directory: .                                   Exec  Total  Coverage
     File: example.cpp               Lines:       6      7    85.7 %
     Date: 2018-07-02 23:02:54    Branches:       1      2    50.0 %

Line Branch Exec  Source
   1              // example.cpp
   2
   3           1  int foo(int param)
   4              {
   5      ✗✓   1      if (param)
   6                  {
   7                      return 1;
   8                  }
   9                  else
  10                  {
  11           1          return 0;
  12                  }
  13              }
  14
  15           1  int main(int argc, char* argv[])
  16              {
  17           1      foo(0);
  18
  19           1      return 0;
  20              }
  21
```

Generated by: GCOVR (Version 4.1)



Challenge 1

	Existing coverage inspectors	Need for directed fuzzing inspector
Based on	lines	basic blocks
Purpose	see where / how much are (un)covered	see how far the fuzzing reached toward target



```
                     GCC Code Coverage Report

Directory: .                                 Exec   Total  Coverage
     File: example.cpp               Lines:     6       7    85.7 %
     Date: 2018-07-02 23:02:54    Branches:     1       2    50.0 %

Line Branch Exec  Source
   1              // example.cpp
   2
   3            1 int foo(int param)
   4              {
   5        ✗✓  1     if (param)
   6                  {
   7                      return 1;
   8                  }
   9                  else
  10                  {
  11            1         return 0;
  12                  }
  13              }
  14
  15            1 int main(int argc, char* argv[])
  16              {
  17            1     foo(0);
  18
  19            1     return 0;
  20              }
  21

                 Generated by: GCOVR (Version 4.1)
```

no basic block coverage visualizer

Challenge 2

- Key aspect of directed fuzzing is **data-flow coverage**

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

```
width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2
```

data-flow

Challenge 2

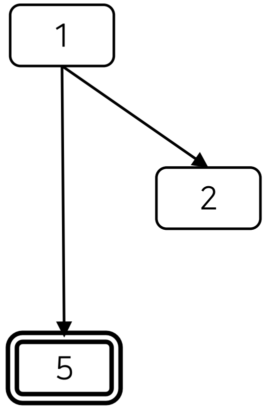
- Key aspect of directed fuzzing is **data-flow coverage**
- Visible into **def-use graph**

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Challenge 2

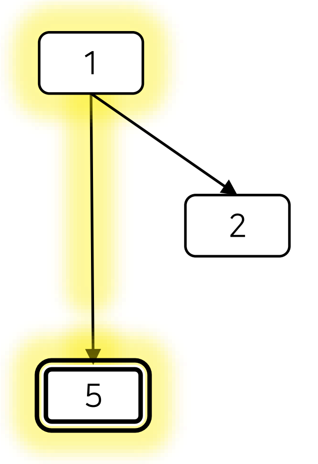
- Key aspect of directed fuzzing is **data-flow coverage**
- Visible into **def-use graph**

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Challenge 2

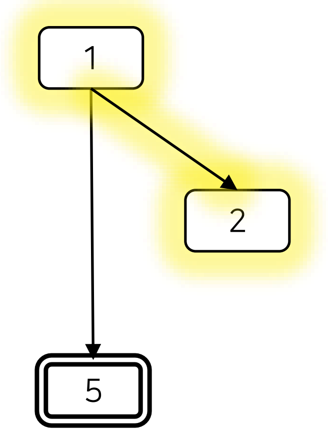
- Key aspect of directed fuzzing is **data-flow coverage**
- Visible into **def-use graph**

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:      doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



def-use graph

Challenge 2

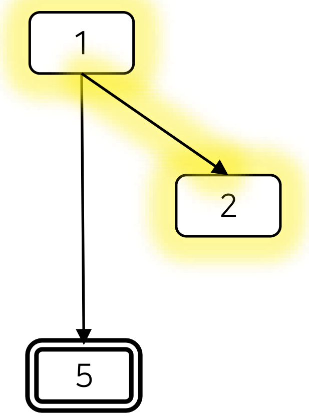
- Key aspect of directed fuzzing is **data-flow coverage**
- Visible into **def-use graph**
- Manually traversing def-use graph alongside heat-map boosts understanding
- Not yet in the world

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

code

width: defined at 1, used at 5
height: defined at 1, used at 5
some_data: defined at 1, used at 2

data-flow



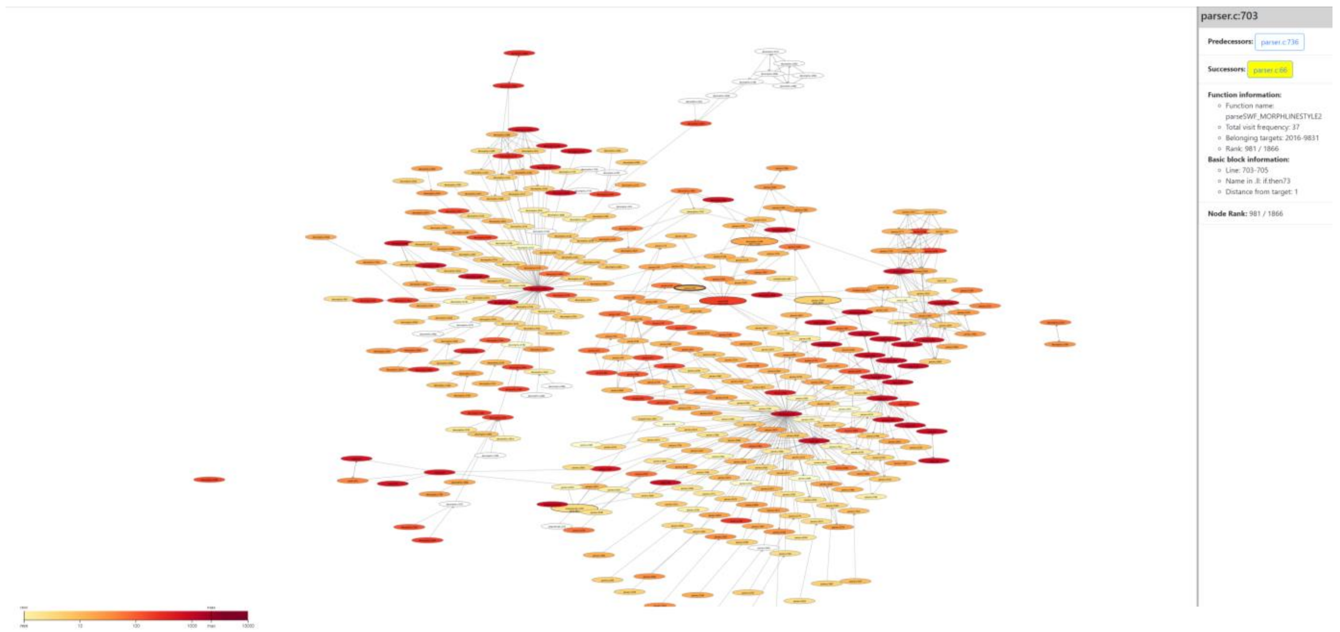
def-use graph

Our Solution

- TopViz: Def-use graph visualizer for directed fuzzing
- Tackle Challenge 1 (No basic block coverage visualizer) by a **basic-block heatmap**
- Tackle Challenge 2 (No effective def-use graph visualizer) by an **ample-info visualizer**
- Integrate that into a def-use graph visualizer with heatmap information

Features

- **basic-block heatmap**
- **ample-info visualizer**
  - Express targets
  - Express road to target in yellow
- **good for fuzzing developer (me)**
  - Track .c code
  - Grasp block’s purpose by name



parser.c:703

Predecessors: parser.c:736

Successors: parser.c:66

Function information:
- Function name: parseSWF_MORPHLINESTYLE2
- Total visit frequency: 37
- Belonging targets: 2016-9831
- Rank: 981 / 1866

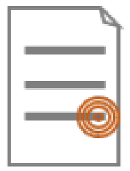
Basic block information:
- Line: 703-705
- Name in .ll: if.then73
- Distance from target: 1

Node Rank: 981 / 1866

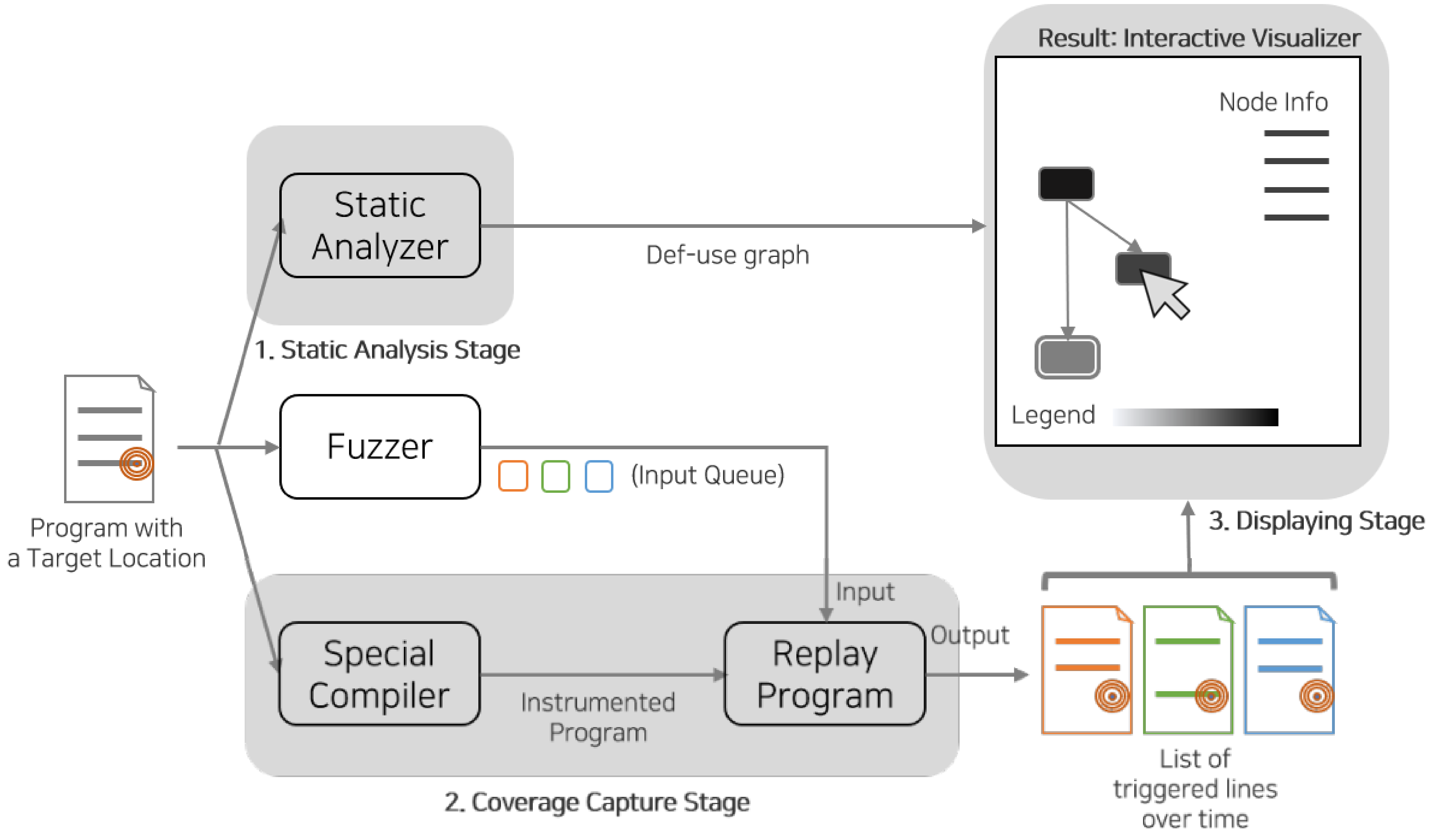
http://elvis08.kaist.ac.kr/gun/dug/swftophp-4.7-topuzz/Topuzz/full-time/0/

Visualizer Architecture and methods

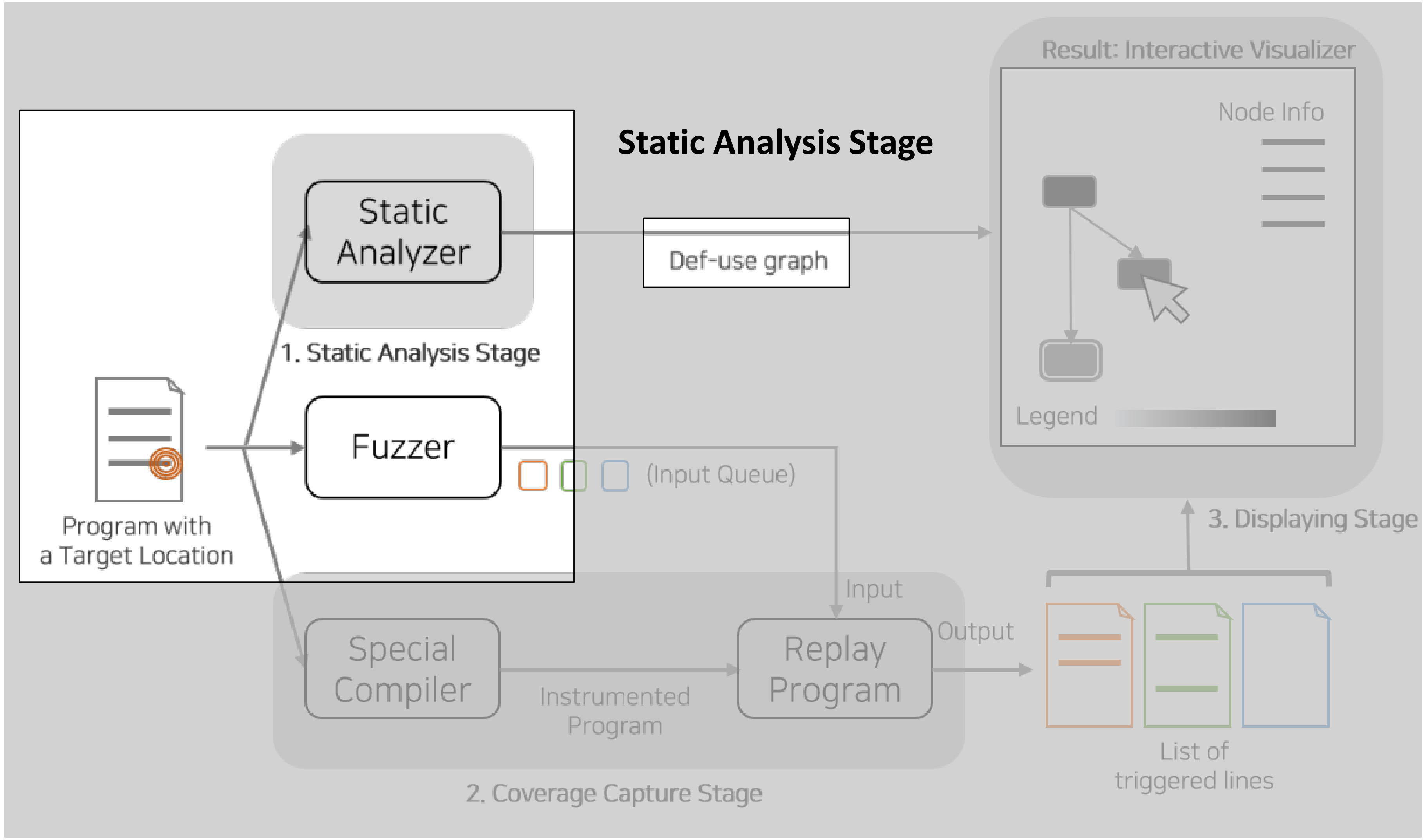
Visualizer Architecture and methods



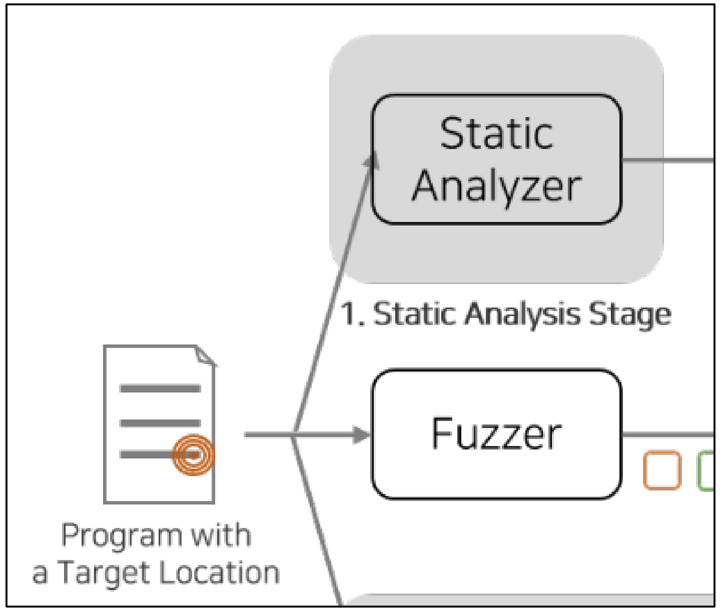
Visualizer Architecture and methods



Visualizer Architecture and methods

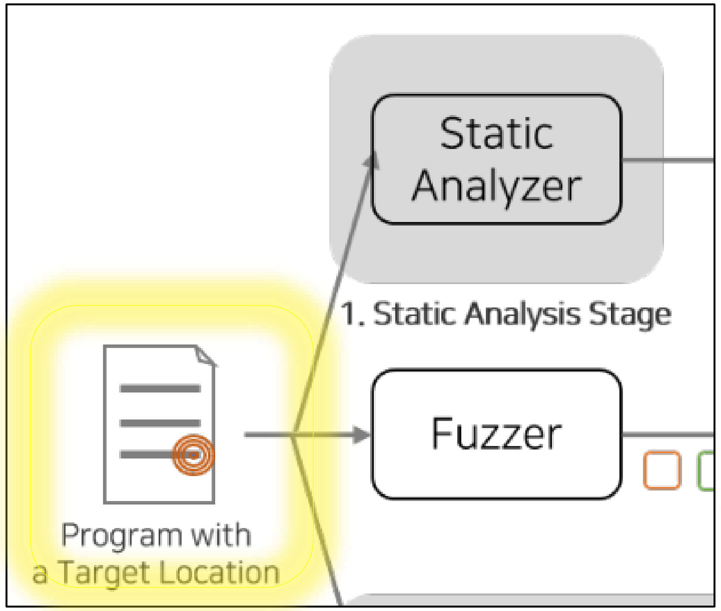


Visualizer Architecture and methods



Static Analysis Stage

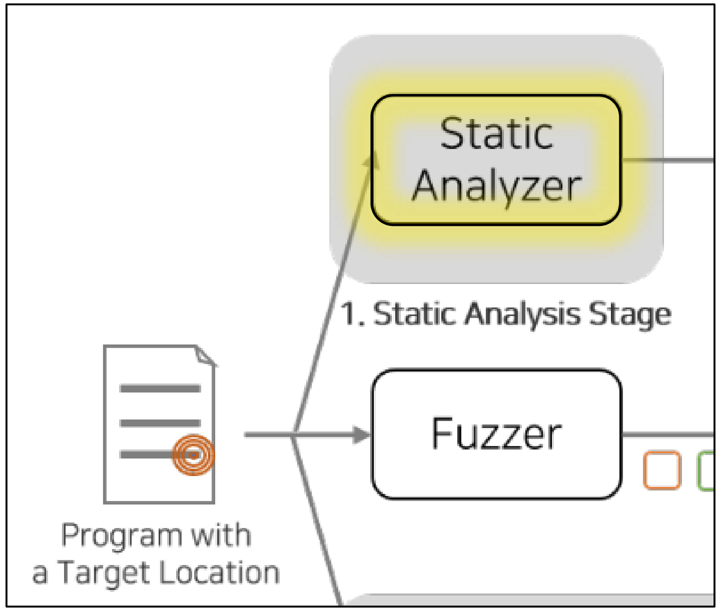
Visualizer Architecture and methods



Static Analysis Stage

```
def getSize(width, height, some_data):
if (some_data) then
  doSomething()
end if
print("Size is", width × height)
```

Visualizer Architecture and methods

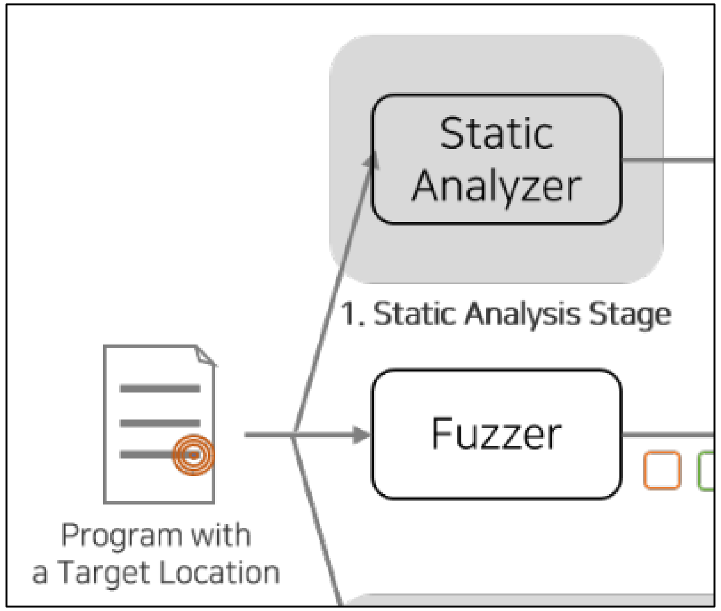


Static Analysis Stage

Def-use graph

```
def getSize(width, height, some_data):
if (some_data) then
  doSomething()
end if
print("Size is", width × height)
```

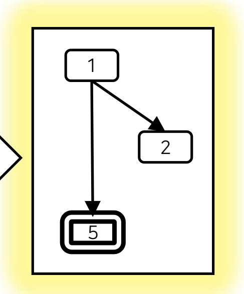
Visualizer Architecture and methods



Static Analysis Stage

Def-use graph

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```



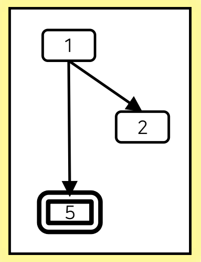
Visualizer Architecture and methods

- **How?**

Program

```
1:  def getSize(width, height, some_data):
2:  if (some_data) then
3:     doSomething()
4:  end if
5:  print("Size is", width × height)
```

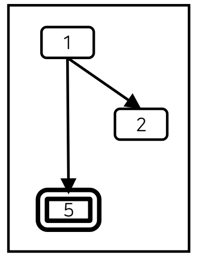
Basic-block-based graph



Visualizer Architecture and methods

- **How?**

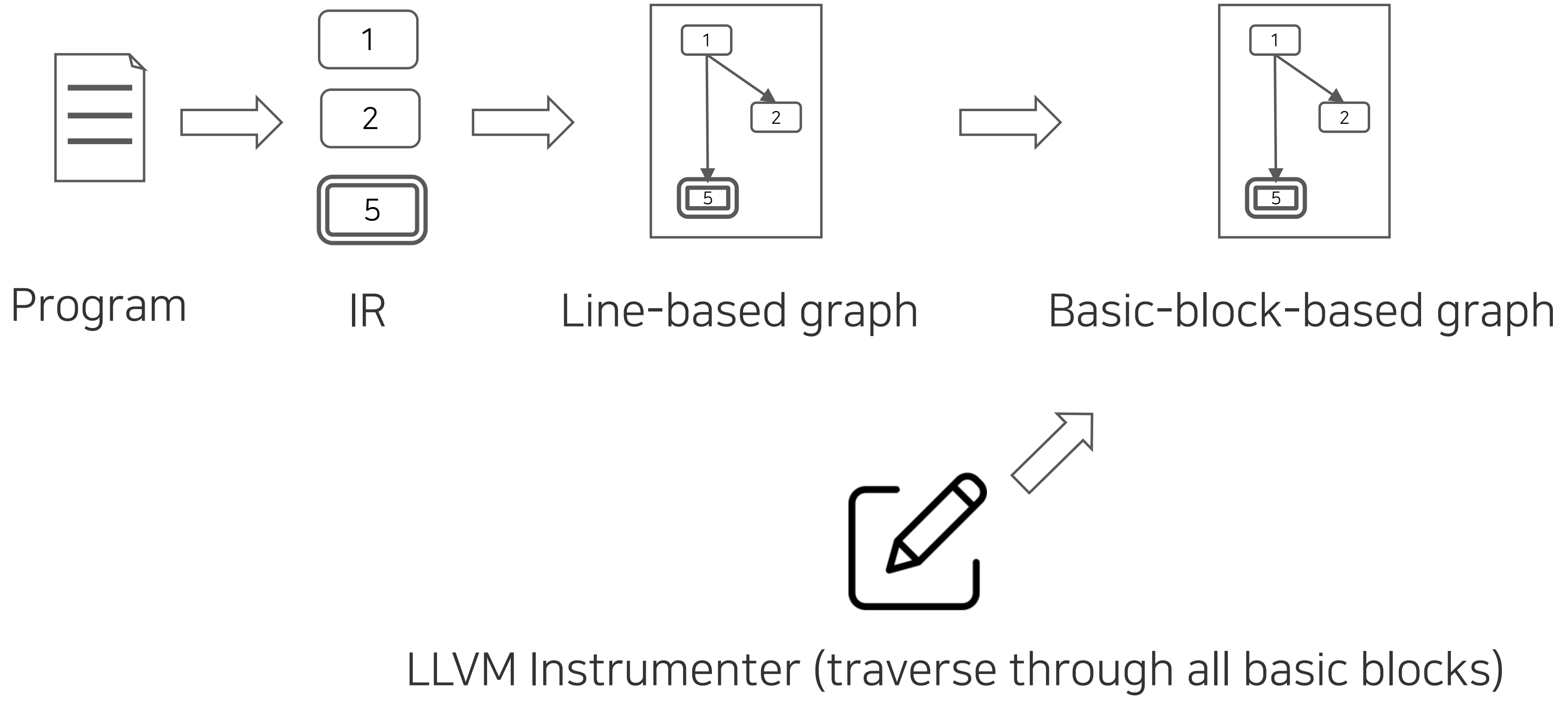
Program



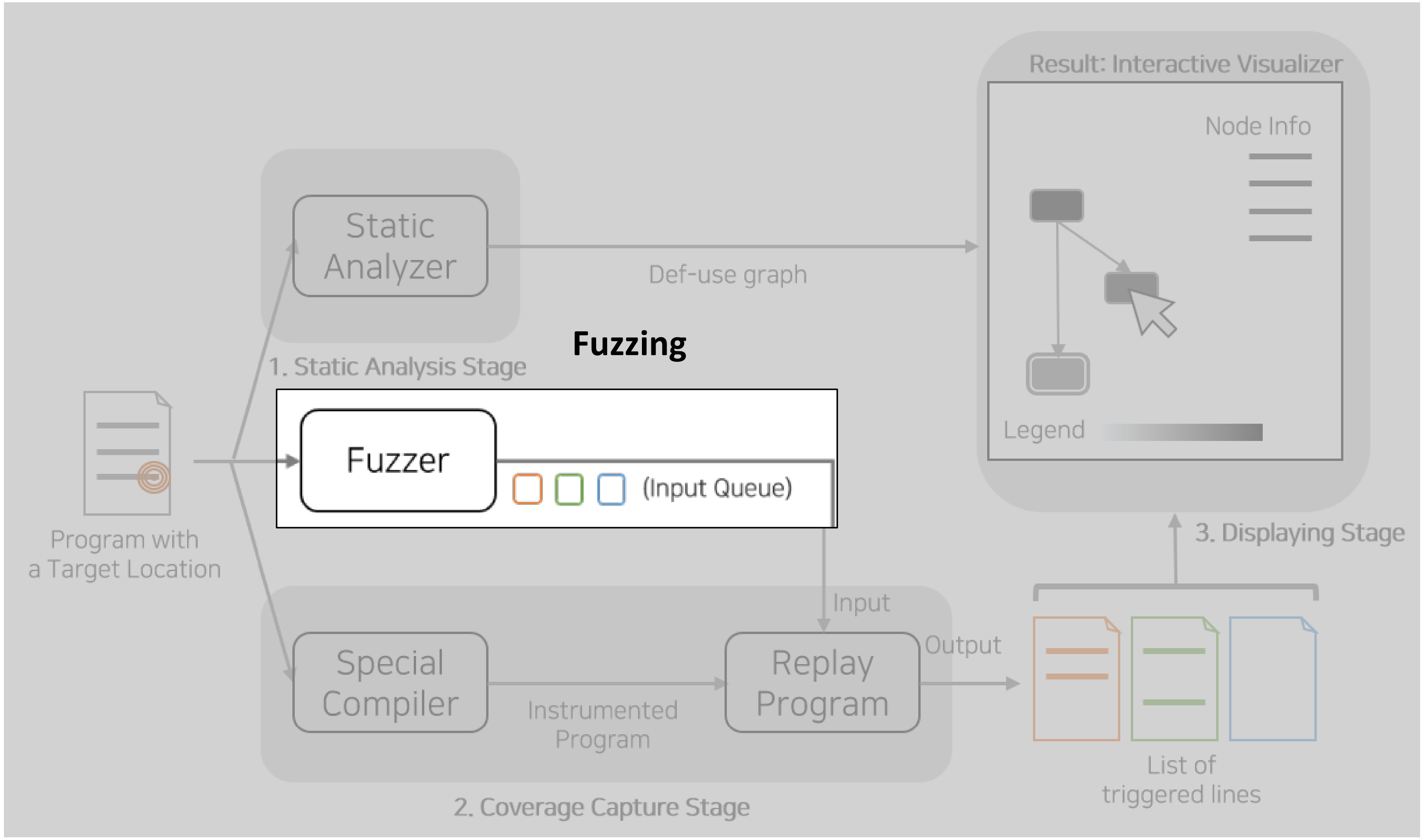
Basic-block-based graph

Visualizer Architecture and methods

- **How?**

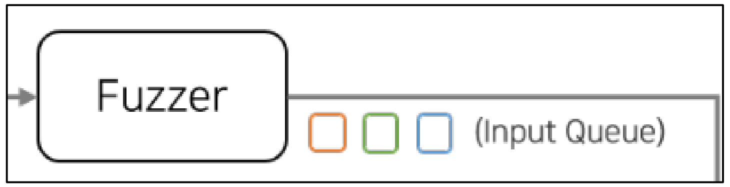


Visualizer Architecture and methods



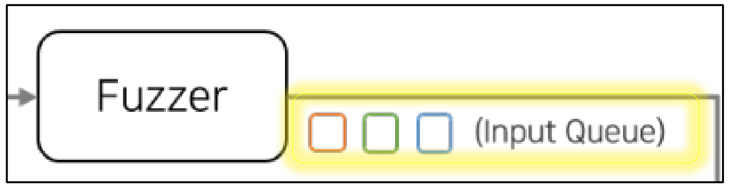
Visualizer Architecture and methods

Fuzzing



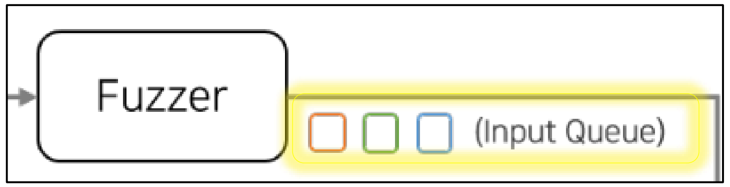
Visualizer Architecture and methods

Fuzzing



Visualizer Architecture and methods

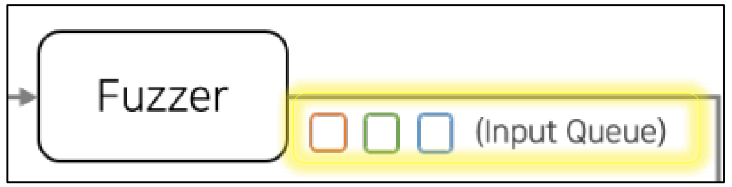
Fuzzing



compatible with any fuzzing tools

Visualizer Architecture and methods

Fuzzing



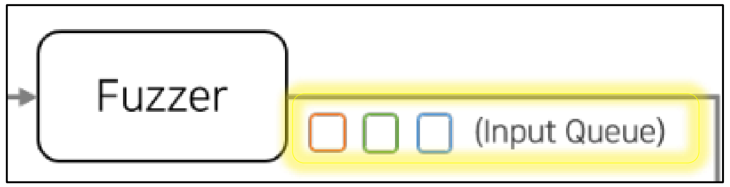
compatible with any fuzzing tools

easily usable standalone module

Visualizer Architecture and methods

- **Why?**
- We snatch the input queue to reproduce coverage in a separate binary

Fuzzing



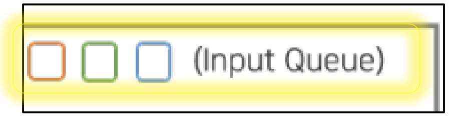
compatible with any fuzzing tools

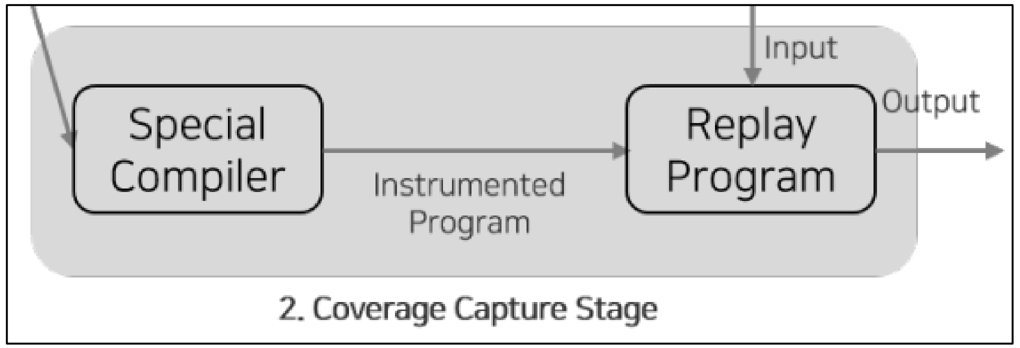
easily usable standalone module

Visualizer Architecture and methods

- **Why?**
- We snatch the input queue to reproduce coverage in a separate binary

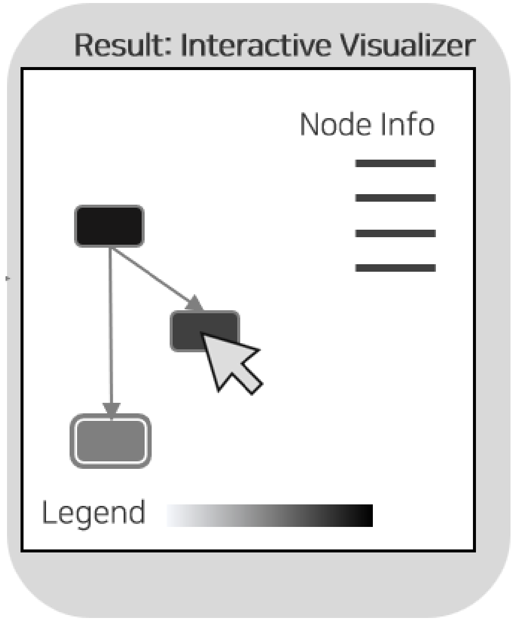
2. Coverage Capture Stage





Visualizer Architecture and methods

- **3. Visualizing**
- Our interest
  - How far was the node from any target?
  - Did lagging target node get less visit?
  - How did the fuzzing setting change basic block coverage?
- Visualization results
  - Gives distance and routes to target
  - Easily understandable



Result

- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.
- TopViz is compatible with..
  - **any** fuzzing tools.
  - **any** target programs with open source code.

Future works

- **Time bar to see hourly status**
- Visualize time axis to help establish dynamic heuristic

- **Chart to compare basic blocks hit-rate per fuzzing setting**
- Easy to know the contribution of new-setted fuzzing

Conclusion

- This research builds a brand-new visualization tool, **TopViz**.
- TopViz **helps developing** directed fuzzing by..
  - tracking data-flow to target location.
  - **clear**, interactive frequency chart and parent/child buttons.
- TopViz is **first** to aid directed fuzzing research.
- TopViz is compatible with..
  - **any** fuzzing tools.
  - **any** target programs with open source code.