Lab: Information Flow Tracking

(Week 3)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

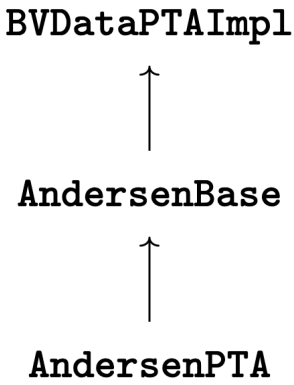
Assignment-1

- Assignment-1 (20 points)
  - `readSrcSnkFromFile` and `reachability`: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and collect **feasible** paths from a source node to a sink node on SVF's `ICFG`.
  - `solveWorklist`: Implement **field-sensitive** Andersen's inclusion-based constraint solving for points-to analysis on SVF's `ConstraintGraph`
  - `aliasCheck`: Implement taint analysis in class `ICFGTraversal`. **Checking aliases** of the two variables at source and sink. Two variables are aliases if their points-to sets have at least one overlapping element.

Assignment-1

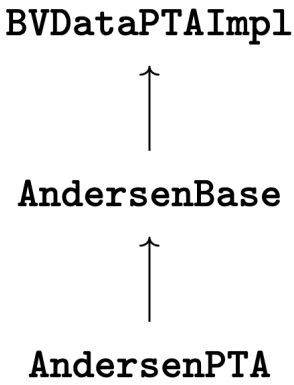
- Assignment-1 (20 points)
  - `readSrcSnkFromFile` and `reachability`: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and collect **feasible** paths from a source node to a sink node on SVF's `ICFG`.
  - `solveWorklist`: Implement **field-sensitive** Andersen's inclusion-based constraint solving for points-to analysis on SVF's `ConstraintGraph`
  - `aliasCheck`: Implement taint analysis in class `ICFGTraversal`. **Checking aliases** of the two variables at source and sink. Two variables are aliases if their points-to sets have at least one overlapping element.
  - **Specification and code template**: `https://github.com/SVF-tools/Software-Security-Analysis/wiki/Assignment-1`
  - **SVF APIs for control- and data-flow analysis** `https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-API`

Assignment Structure



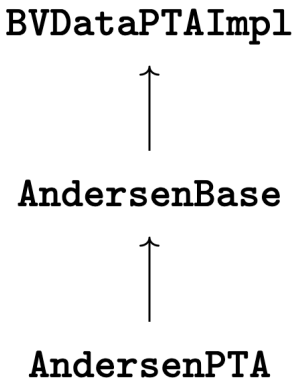
- You will be working on `AndersenPTA`'s `solveWorklist` method.

Assignment Structure



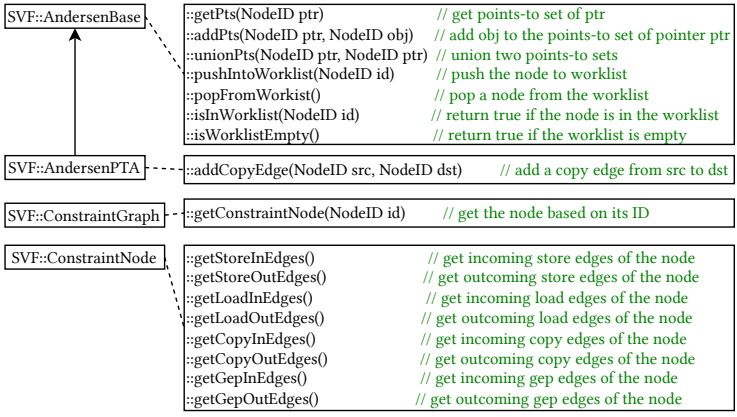
- You will be working on `AndersenPTA`'s `solveWorklist` method.
- Constraint graph is the field `consCG`.

Assignment Structure



- You will be working on `AndersenPTA`'s `solveWorklist` method.
- Constraint graph is the field `consCG`.
- More APIs about points-to operations and constraint graph are here: `https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-API`

APIs for Implementing Andersen's analysis



https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#worklist-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#points-to-set-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#alias-relations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/SVF-CPP-API#constraintgraph-constraintnode-and-constraintedge

Python APIs for Implementing Andersen's Analysis

The Python version of these APIs, provided in the `pysvf` module, retains the same names and functionalities as the C++ version.

Notes:

- The methods from `AndersenBase` are directly accessible through `AndersenPTA` in `pysvf`, as the implementation forwards all `AndersenBase` methods to `AndersenPTA`.
- `ConstraintGraph` and `ConstraintNode` are also available and behave the same as their C++ counterparts.

https://github.com/SVF-tools/Software-Security-Analysis/wiki/Pysvf-API#worklist-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/Pysvf-API#points-to-set-operations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/Pysvf-API#alias-relations
https://github.com/SVF-tools/Software-Security-Analysis/wiki/Pysvf-API#points-to-set-operations

Debugging Tips – `MAYALIAS`/`NOALIAS` (1)

- `MAYALIAS` and `NOALIAS` denote the expected results that your implementation should yield (oracle)

Debugging Tips – `MAYALIAS`/`NOALIAS` (1)

- `MAYALIAS` and `NOALIAS` denote the expected results that your implementation should yield (oracle)
- Two pointers may alias when their points-to sets intersect
- Failing a `MAYALIAS` would indicate one or both points-to set missing objects

Debugging Tips – MAYALIAS/NOALIAS (1)

- MAYALIAS and NOALIAS denote the expected results that your implementation should yield (oracle)
- Two pointers may alias when their points-to sets intersect
- Failing a MAYALIAS would indicate one or both points-to set missing objects
- Two pointers do not alias when their points-to sets are disjoint
- Failing a NOALIAS would indicate one or both points-to set contain extraneous objects

Debugging Tips – MAYALIAS/NOALIAS (2)

```
extern void MAYALIAS(void* p, void* q);  // NECESSARY!
int global;
int *p_global;
void foo() { p_global = &global; }
int main() {
    int *p_local;
    p_local = &global;
    foo();
    MAYALIAS(p_local, p_global);  // pts(p_local) must intersect with
    return 0;                     // pts(p_global) or we crash!
}
```

Debugging Tips (C++) – Inspecting the Constraint Graph (1)

`-print-constraint-graph` will print the final constraint graph (i.e., after analysis)

```
./build/bin/ass1 -pta -print-constraint-graph \
Assignment-1/Tests/testcases/pta/test1.ll

...
15 -- Addr --> 14
21 -- Addr --> 20
0 -- Copy --> 1
7 -- Load --> 18
0 -- Store --> 7
...
```

Debugging Tips (C++) – Inspecting the Constraint Graph (2)

`-dump-constraint-graph` will produce two constraint graphs in DOT format
`consCG_initial.dot`: before analysis
`consCG_final.dot`: after analysis

```
./build/bin/ass1 -pta -dump-constraint-graph \
Assignment-1/Tests/testcases/pta/test1.ll
```

Debugging Tips (C++) – Inspecting the ICFG

`-dump-icfg` will produce the ICFG in DOT format as `icfg_initial.dot`

```
./build/bin/ass1 -icfg -dump-icfg \
Assignment-1/Tests/testcases/pta/test1.ll
```

Debugging Tips (C++) – Inspecting Points-To Sets (1)

- Use the `.toString()` method of an `SVFVar`, `SVFStmt`, `ConstraintNode`, or `ICFGNode` to understand the mapping from SVFIR to LLVMIR and C
- Use `.toString()` in conjunction with `.getId()` to understand the mapping to node IDs
- `AndersenPTA::alias(NodeID, NodeID)` returns whether two pointers (`ConstraintNode`s/`SVFVar`s; use `.getId()`) are aliases
- You can print points-to sets coming from `getPts()` by iterating over them; they are just sets of numbers

Debugging Tips (C++) – Inspecting Points-To Sets (2)

- `-print-pts` shows the final points-to set of every node after the analysis

```
./build/bin/ass1 -pta -print-pts Assignment-1/Tests/testcases/pta/test1.ll

NodeID 0  PointsTo: { 5 }

NodeID 4  PointsTo: {empty}

NodeID 7  PointsTo: {empty}

...
```

C++ File Reading

Implement method `readSrcSnkFormFile` in `Assignment-1.cpp` to parse the two lines from `SrcSnk.txt` in the form of

```
source -> { source src set getname update getchar tgetstr }
sink -> { sink mysql_query system require chmod broadcast }
```

Please refer to the following links (among many others) for C++ file reading:

- `https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm`
- `https://opensource.com/article/21/3/ccc-input-output`
- `https://www.cplusplus.com/doc/tutorial/files/`
- `https://learn.microsoft.com/cpp/standard-library/input-streams`

Debugging Tips (Python) – Graph generation

The command-line flags used in the C++ examples are the same in Python:

- `-print-constraint-graph`: Prints constraint graph **after analysis**
- `-print-pts`: Prints points-to set of every node **after analysis**
- `-dump-constraint-graph`: Produces constraint graphs before/after analysis in DOT format
- `-dump-icfg`: Produces ICFG in DOT format

Debugging Tips (Python) – Graph generation

The command-line flags used in the C++ examples are the same in Python:

- `-print-constraint-graph`: Prints constraint graph **after analysis**
- `-print-pts`: Prints points-to set of every node **after analysis**
- `-dump-constraint-graph`: Produces constraint graphs before/after analysis in DOT format
- `-dump-icfg`: Produces ICFG in DOT format

Remark

Of the flags above, only `-dump-icfg` works when running `-icfg` tests.

Debugging Tips (Python) – Graph generation

- The command is slightly different:

```
python ./Assignment-1/Python/test.py -pta -print-pts \
    ./Assignment-1/Tests/testcases/pta/test1.ll

  NodeID 0  PointsTo: { 5 }

  NodeID 4  PointsTo: {empty}

  NodeID 7  PointsTo: {empty}

  ...
```

Debugging Tips (Python) – Graph generation

- All graphs can also be generated programmatically using:

```
AndersenPTA.pag.dump("pag.dot")
AndersenPTA.consCG.dump("cons.dot")
ICFGTraversal.icfg.dump("icfg.dot")
```

Debugging Tips (Python)

- Use `AndersenPTA.alias(NodeID, NodeID)` to check whether two pointers (i.e., `ConstraintNodes` or `SVFVars`) are aliases.

```
# Assuming `ander` is some `AndersenPTA`
node1 = get_node()
node2 = get_node()
is_alias = ander.alias(node1.getId(), node2.getId())
```

Debugging Tips (Python)

- Use `AndersenPTA.alias(NodeID, NodeID)` to check whether two pointers (i.e., `ConstraintNodes` or `SVFVars`) are aliases.

```
# Assuming `ander` is some `AndersenPTA`
node1 = get_node()
node2 = get_node()
is_alias = ander.alias(node1.getId(), node2.getId())
```

Remark

In Python, `NodeID`s are just `int`s, they are not their own type.

Debugging Tips (Python)

- In Python, you can inspect the points-to sets directly in code:

```
for node in AndersenPTA.consCG.getNodes():
    print(node.getId(), AndersenPTA.getPts(node.getId()))
```