Lab: Information Flow Tracking

(Week 3)

Yulei Sui

School of Computer Science and Engineering

University of New South Wales, Australia

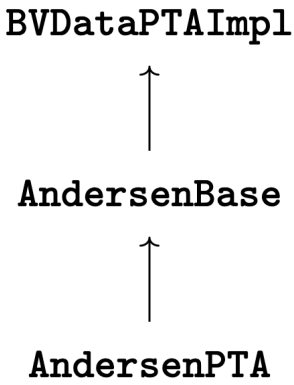
Assignment-1

- 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

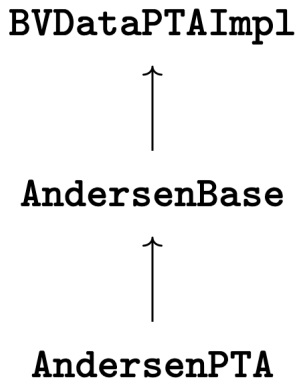
- Assignment-1 (20 points)
  - `readSrcSnkFromFile` and `reachability`: Implement a context-sensitive graph traversal on a CodeGraph (i.e., ICFG) and collect **feasible** paths from a source node to a sink node on SVF's `ICFG`.
  - `solveWorklist`: Implement **field-sensitive** Andersen's inclusion-based 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



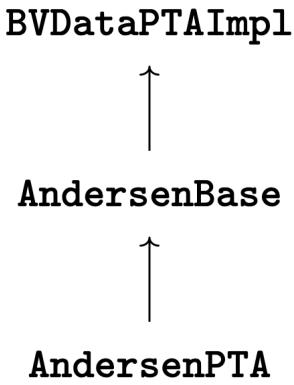
- You will be working on `AndersenPTA`'s `solveWorklist` method.

Assignment Structure



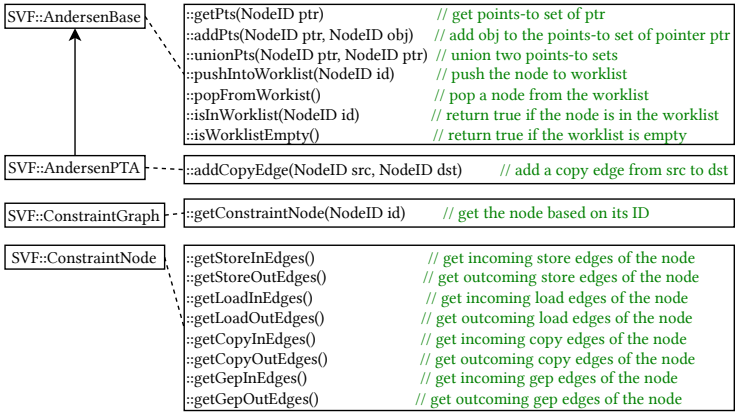
- You will be working on `AndersenPTA`'s `solveWorklist` method.
- Constraint graph is the field `consCG`.

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` and `NOALIAS` denote the expected results (oracle) that your implementation should yield (e.g., if your results failed in a `MAYALIAS` case, it means that your points-to set is incomplete).
- The `AndersenPTA::alias(NodeID, NodeID)` method is used to evaluate whether two pointers (`ConstraintNodes`/`SVFVars`) are aliases (i.e., their points-to sets intersect). You can get the ID of a `ConstraintNode`/`SVFVar` via `getId()`.
- Add `-print-pts` as an extra option for your ass1 executable when you try to print out the final points-to set of each node to validate your MAYALIAS and NOALIAS results.
- Use `-print-constraint-graph` to print out the final `ConstraintGraph` or `-dump-constraint-graph` to dump it into a dot file for viewing in VSCode.
- Use the `toString()` method in `SVFVar`, `SVFStmt`, `ConstraintNode`, or `ICFGNode` to understand the mapping from SVFIR to LLVMIR and C.

Debugging Tips (Python Version)

- `MAYALIAS` and `NOALIAS` denote the expected results (oracle) that your implementation should yield. If your result fails a `MAYALIAS` case, it typically means your points-to set is incomplete.
- Use `AndersenPTA.alias(NodeID, NodeID)` to check whether two pointers (i.e., `ConstraintNodes` or `SVFVars`) are aliases.
- 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()))
    ```
- To visualise the constraint graph, use: `AndersenPTA.consCG.dump("cons.dot")` and open it with the VSCode Graphviz plugin.
    ```

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://www.cplusplus.com/doc/tutorial/files/`
- `https://linuxhint.com/cplusplus_read_write/`
- `https://opensource.com/article/21/3/ccc-input-output`