CS510: Project 2 - Report

Group Member:

- Luke Jiang jiang700@purdue.edu
- Sihao Yin yin93@purdue.edu

Part I

(a) Inferring Likely Invariants for Bug Detection

Our source code can be found in `jiang700/pi/PartA/src`. Run `make` to compile the source code. `jiang700/pi/PartA/pipair` is the executable program for our tool.

(b) Finding and Explaining False Positives

(i) Do we think he had found any real bugs?

Yes, there are some real bugs, but he also found false positives.

(ii) Why false positives occur?

1. The technique that we are using in this project is based on probability, not programming semantics. The tool doesn't understand preconditions and postconditions of the functions, and can only infer bug from the statistical usage of the functions. Therefore, the tool can never be sure that a pair with high confidence is indeed a bug.
2. The current technique is too loose on defining a function pair. It only considers scope in one level, ignoring other aspects of the program e.g. the order of instructions. Therefore, it will erroneously increase the confidence of functions pairs and report false positives.

(iii) Identify two false positives for test3 `httpd`

1. `(apr_array_make, apr_hook_debug_show)`. It appeared more than 26 times in test3_3_65.out

   According to the http://apr.apache.org/docs, `apr_array_make`: create an array, and `apr_hook_debug_show` prints all of the information about the current hook, for debugging purposes.

   This pair is a false positive because `apr_hook_debug_show` is purely for debugging purpose and have no semantic effect on the code. Therefore, even though this pair has occurred many times in the code, it is still not necessary to use the debug function every time an array is created.

2. `(apr_array_make, apr_array_push)`. It appeared more than 7 times in test3_10_80.out

   According to http://apr.apache.org/docs, apr_array_make makes a new apr_array and apr_array_push takes an apr_array as input and adds an element to the array.

   We argue this is a false positive pair with the support of the bug location in ap_regname.

   In ap_regname, apr_array_push is called by not apr_array_make. This is in fact not a bug, since we are pushing elements to an array called names, which is an input for this function, this function is not creating a new array and populating it.

(c) Inter-Procedural Analysis.

In this part, we designed and implemented a new algorithm to reduce false positives.

(i) Algorithm Overview

We maintain two data strctures: a callee to caller map `cmap` and a caller to callee map `cmapR`. Both are built when the input CFG is processed. We also created a mew method `expand`, which updates both `cmap` and `cmapR` to the state after one level of inlining. After inlining, we feed the updated `cmap` and `cmapR` to `analyze`, which is the same function used in i.a.

The majority of code for this part is implemented in `expand`. In this function, we do the following:

1. We find the intersection of the `keySet` of `cmap` and `cmapR`, and call it `common`. `common` contains the functions that are both callers and callees in the program.
2. We remove the functions that are in `cmap` and `common` from `cmap`, and store them in a new map `M`.
3. For each functions in common, we get its callers and callees (`scopes`), and join the callees into callers. This is the primary job of `expand` that performs inlining
4. Finally, rebuild `cmapR` from `cmap`.

The new algorithm can be run by the following command:

```
pipair test3/httpd.bc.orig 3 65 true
```

where the last input specifies if inlining is enabled or not. The default is false.

(ii) Experiments

We run our program on `test3` with both (3, 65) and (10, 80) and were successful in removing the false positive pair `(apr_array_make, apr_hook_debug_show)` identified in b.iii. However, the number of bugs reported increases in general. We concluded that this is due to the inflation of confidence after inlining.

(d) Improving the Solutions

In this section, we propose another algorithm to further reduce false positives.

(i) Algorithm Overview

The main idea is to take the order of function calls into account. More specifically, define the support of `(A, B)` as the number of times A is called before B in a scope. Thus, `(A, B)` is not equal to `(B, A)` in general.

We maintain a new data structure `cmapOrder` that maps a pair of caller and callee (caller, callee) to the callee's call order in caller. This data structure is built while reading the input graph along with `cmap`. Once `cmapOrder` is built, we do the following in `analyzeOrder`:

1. Like in part i.a, we first find the intersection of `fun1` 's caller and `fun2` 's caller, call it `join`
2. Then, we divide `join` into two parts: `join12` and `join21`, where `join12` contains the callers that call `fun1` before `fun2` and caller21 is the rest. This is the part where (A, B) and (B, A) are differentiated.
3. We feed `join12` and `join21` separately to a `analyze` function, along with other infos.
4. The `analyze` function is semantically identical to the `analyze` in part i.a.

To facilitate our implementation, we used a hashmap-friendly implementation of Pair found online: https://stackoverflow.com/questions/156275/what-is-the-equivalent-of-the-c-pairl-r-in-java.

(ii) Evaluation

We run our program on `test3` with both (3, 65) and (10, 80) and were successful in removing the false positive pair `(apr_array_make, apr_hook_debug_show)` identified in b.iii. It also reports fewer bugs in general. Here are the details after we run our implementation using `verify.sh`:

- test1 with threshhold=(3,65): Same.
- test1 with threshhold=(10,80): Same.
- test2 with threshhold=(3,65): 1 missing, 3 extra and 4 total
- test2 with threshhold=(10,80): Same.
- test3 with threshhold=(3,65): 144 missing, 198 extra and 253 total.
- test3 with threshhold=(10,80): 8 missing, 11 extra and 25 total.

Please check the README file in the part D folder to see more details on how to run the code for this part.

Part II

Part II (a)

1. 10026 (Dereference before null check)

- classified as BUG
- **Faulty Lines**: In `JspDocumentParser.java` inside function `startElement` , `attrs` is dereferenced at line 274 by `checkPrefixes` , but the null check is at line 294. Therefore, `checkPrefixes` could be dereferencing a null `attrs` .
- **Proposed Fix**: add a null check for `attrs` before `checkPrefixes` .

2. 10102 (Dereference null return value)

- classified as BUG
- **Faulty Lines**: In `SSIProcessor.java` inside function `process` , `strCmd` is returned by a call to `parseCmd` at line 121, which could be a null value.
- **Proposed Fix**: add a null check for `strCmd` after `parseCmd` .

3. 10148 (Resource leak)

- classified as BUG
- **Faulty Lines**: In `JKMain.java` inside function `loadPropertiesFile` , a new `FileInputStream` is created at line 455, but it is not closed appropriately before the function returns.
- **Proposed Fix**: save `FileInputStream` to a separate variable or close it before the function returns.

4. 10167 (Dereference before null check)

- classified as BUG
- **Faulty Lines**: In `DeltaRequest.java` inside function `readExternal` , `this.actions` is null-checked at line 238. However, at line 234, the method calls `reset()` , which would dereference `this.actions` .
- **Proposed Fix**: Add a null check for `this.actions` before the call to `reset` .

5. 10176 (Dereference null return value)

- classified as BUG
- **Faulty Lines**: In `SimpleTcpReplicationManager.java` inside function `messageReceived` , `session` is the return value of a call to `readSession` at line 612. However, at line 614, `session` is dereferenced without a null check.
- **Proposed Fix**: Add a null check before dereferencing `session` .

6. 10231 (Resource leak)

- classified as BUG
- **Faulty Lines**: In `ChannelNioSocket.java` inside function `init` , `ssc` created at line 390 but not closed properly by the function.
- **Proposed Fix**: close `ssc` by inserting `ssc.close()` after line 391.

7. 10291 (Dereference null return value)

- classified as BUG
- **Faulty Lines**: In `SSIServletExternalResolver.java` inside function `getServletContextAndPathFromVirtualPath`, at line 421 `normContext.getContextPath` could return a null value and pass it as parameter to `getPathWithoutContext`.
- **Proposed Fix**: Create a new variable to store the returned value from `normContext.getContextPath()` and perform a null check before passing it to `getPathWithoutContext`.

8. 10351 (Dereference after null check)

- classified as BUG
- **Faulty Lines**: In `SimpleTcpCluster.java` inside function `messageReceived`, `message` is null-checked at line 896 but dereferenced at line 907.
- **Proposed Fix**: add a null check for `message` at line 905.

9. 10381 (Dereference null return value)

- classified as BUG
- **Faulty Lines**: In `VirtualDirContext.java` inside function `scanForTlds`, `files` is created to store the returned value of `File.listFiles()`. This `files` can potentially be a null value, and is dereferenced at line 193 without a null check.
- **Proposed Fix**: add a null check before dereferencing `files`.

10. 10395 (Unguarded read)

- classified as FALSE POSITIVE
- **Explanation**: After checking other examples that locks before using `instance`, we conclude that there is no explicit declaration of lock, so it is a false positive.

11. 10396 (Dereference after null check)

- classidied as INTENTIONAL
- **Expalanation**: In `FarmWarDeployer.java` inside function `start`, `engine` can be null, and it is dereferenced at line 159 without a null check. However, it is wrapped in a try-catch block, so the exception is handled appropriately.

12. 10435 (Resource leak)

- classified as BUG
- **Faulty Lines**: In `JDTCompiler.java` inside function `isPackage`, `InputStream` is created at line 233 but never closed.
- **Proposed Fix**: close `InputStream` before return.

13. 10453 (Dereference after null check)

- classified as BUG
- **Faulty Lines**: In `ChannelUn.java` inside function `init`, `wEnv` is null-checked at line 88 and line 132, but is also used in `initNative` at line 166.
- **Proposed Fix**: Add a null check for `wEnv` before using it.

14. 10507 (Dereference after null check)

- classified as BUG
- **Faulty Lines**: In `MessageBytes.java` inside function `equalsIgnoreCase`, `strValue` is null-checked at line 330, but it is also used at line 221 without a null check.
- **Proposed Fix**: Add a null check for `strValue` before using it.

15. 10514 (Missing call to superclass)

- classified as FALSE POSITIVE
- **Explanation**: In `ChannelCoordinator.java` for function `sendMessage`, the inherited implementation is overwritten. Therefore, it's not necessary to call `super.sendMessage`, since it calls `ChannelSender.sendMessage` instead.

16. 10625 (Dereference before null check)

- classified as FALSE POSITIVE
- **Explanation**: similar to 10396. The variable `engine` can be null, but it is wrapped inside a try-catch.

17. 10654 (Dereference after null check)

- classified as FALSE POSITIVE
- **Explanation**: In `JDBCStore.java` inside function `remove`, `preparedRemoveSql` is null-checked at line 669. However, if that variable is indeed null, it will be assigned a new object by `PreparedStatement`. Therefore, the variable `preparedRemoveSql` can never be null after the statement. So the dereference in function `close` is ok.

18. 10689 (Unguarded read)

- classified as BUG
- **Faulty Lines**: In `DeltaSession` inside function `setMaxInactiveInterval`, `deltaRequest` is accessed in line 280 without a request for lock.
- **Proposed Fixx**: Move `lock()` before the line 280.

Part II (b)

Coverity didn't find any issue with our code in part I (a), we think the following two reasons contributed to this

1. The code size is relatively small, hence it is less likely for bugs to appear

2. We put effort in writing the code, so our code is relatively well written. Therefore, our code doesn't contain any issue that might be caught by coverity