Testing Final Project

SE 465 - 001

Jessica Hanta - 20506674 - jahanta@uwaterloo.ca
John Salaveria - 20511560 - jjsalave@uwaterloo.ca

Part I

Section B

For a tool that detects bugs based on identifying commonly repeating function pairs, false positives can arise due to coincidence that functions are commonly called together while being independent of each other. False positive function pairings identify functions that are often found within the same scopes. However, calling a function from the false positive pair pair does not require the other function to be called possibly since the two functions do not share a similar responsibility.

We can examine the two following pairs that result as false positives from our bug detection tool output:

- `(apr_array_make, apr_array_push)`
- `(apr_array_push,apr_hook_debug_show)`

For the first function pair , our tool generated following errors for `apr_array_make` and `apr_array_push`:

```
bug: apr_array_make in ap_init_virtual_host, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in ap_make_method_list, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in apr_xml_parser_create, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in create_core_dir_config, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in create_core_server_config, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 86.96%
bug: apr_array_make in prep_walk_cache, pair: (apr_array_make, apr_array_push),
support: 40, confidence: 86.96%
bug: apr_array_push in ap_add_file_conf, pair: (apr_array_make, apr_array_push),
support: 40, confidence: 80.00%
bug: apr_array_push in ap_add_per_dir_conf, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_add_per_url_conf, pair: (apr_array_make,
apr_array_push), support: 40, confidence: 80.00%
```

In the header file `apr_tables.h`, the behaviour of `apr_array_make` is described in the comments as creating an array, and `apr_array_push` returns an address in the array that can be assigned a value. Although these two functions are both operations that are associated with arrays, they are not dependent since their functionalities are relevant to separate concerns. Unlike `alloc()` and `free()` which rely on being called as pairs to manage memory, making an array and pushing elements into an array are not inverse operations that are directly dependent.

The second function pair of `apr_array_push` and `apr_hook_debug_show` are reported as bugs for the following scopes from the bug detection tool:

```
apr_hook_debug_show in ap_hook_default_port, pair: (apr_array_push,
apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_http_scheme, pair: (apr_array_push,
apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_post_read_request, pair: (apr_array_push,
apr_hook_debug_show), support: 28, confidence: 87.50%
bug: apr_hook_debug_show in ap_hook_log_transaction, pair: (apr_array_push,
apr_hook_debug_show), support: 28, confidence: 87.50%
```

Similar to the previous function pair, `apr_array_push` and `apr_hook_debug_show` are examples of a false positive pairing reported by the tool since these two functions are independent and don't share similar responsibilities. `apr_array_push` is a function that deals with inserting new elements into an array, and `apr_hook_debug_show` is described as a debugging method that prints information with regards to the hook parameter passed in.

Section C

We took a dynamic programming approach in our algorithm. We preprocess all of the call sites and determine what nodes are reachable from it with the appropriate depth. We store the reachable nodes in a HashMap to guarantee unique values. Each node has the reachable nodes map. We use the preprocessed data to determine all of the reachable nodes from a given root node. We iterate through all of a root node's children and the reachable nodes from the root node's children. We add a child's reachable node to the root's children if it is not already contained in the root's children map. We also increment that node's support count if it is not already contained in the map. This guarantees uniqueness in the root's children and that a node will not have the support incremented twice. The nodes are passed to the compute pi pair method with the updated child nodes and support values.

Pseudo Code

```
void analysis
    //preprocessing - determines reachable nodes from all
    //callsite nodes given the depth level
    for all CallSite nodes n
        expand(n, n, depth)
    //preprocessing complete - look at a nodes child nodes
    //reachable nodes and add the ones that aren't already in
    //the current node's set of children
    for all CallSite nodes n
        for all nodes c in n.children
            for all nodes r in c.reachable
                if r is not in n.children
```

```
                                    add r to n.children
                                    increment r.support
void expand (Node root, Node curr, int depth)
      //we want to expand until depth is 0
      if depth = 0
            return
      //add the curr node to the root's reachable nodes
      root.addReachable(curr)
      for all children c in curr.children
            //we only want to expand on callsites
            if c is a call site
                  expand(root, c, depth-1)
```

Analysis:

We conducted a simple experiment using the provided Test 2 file and a depth level of 1. We conducted 3 other tests using the provided Test 3 file with varying depth levels. We found that with inter procedural analysis, more bugs are reported. The exact results of our experiment are in the table below:

Test Case	Depth Level	# of Original Analysis Errors	# of Inter Procedural Analysis Errors
Test2_3_65	1	4	11
Test3_3_65	1	205	12307
Test3_3_65	2	205	32882
Test3_3_65	3	205	62579

The results of our experiment make sense. Inter procedural analysis requires the expansion of functions. A given root function will contain more than the child functions; it contains a child's child functions as well. With inter procedural analysis, function and pair supports increase because function calls can be part of scopes where they are not explicitly called from. The increase of function and pair supports will result in an increased pair confidence value. More pairs will meet the threshold based on the confidence level, which is why more bugs are reported.

Our solution does better bug reporting than the original algorithm, as demonstrated with the Test2_3_65 output. We report 11 bugs, whereas the original algorithm only reports 3.

Original Bugs:

```
bug: B in scope3, pair: (B, D), support: 4, confidence: 80.00%
bug: D in scope2, pair: (B, D), support: 4, confidence: 80.00%
bug: A in scope3, pair: (A, D), support: 3, confidence: 75.00%
bug: A in scope2, pair: (A, B), support: 3, confidence: 75.00%
```

New Bugs:

```
bug: B in scope3, pair: (B, D), support: 5, confidence: 83.33%
bug: D in scope2, pair: (B, D), support: 5, confidence: 83.33%
bug: D in scope5, pair: (C, D), support: 4, confidence: 66.67%
bug: D in scope6, pair: (C, D), support: 4, confidence: 66.67%
bug: B in scope6, pair: (A, B), support: 5, confidence: 83.33%
bug: A in scope2, pair: (A, B), support: 5, confidence: 83.33%
bug: A in scope5, pair: (A, C), support: 4, confidence: 66.67%
bug: A in scope3, pair: (A, C), support: 4, confidence: 66.67%
bug: C in scope2, pair: (B, C), support: 3, confidence: 75.00%
```

Part II

Section A

Warning Message: 10065 CN Missing break in switch
Classification: Bug

For the following case where the input is of length 3, the code falls through and could exhibit undefined behaviour when it is evaluated against the case of length 4. If the input is 3 characters long and it enters the satisfies the if conditions in case of length 4, then the function will try to check the input's character at index=3, which it outside the bounds of the original input. This scenario causes a buffer overflow. A fix for this bug is to include a `break` statement at the end of the case of length 3.

Warning Message: 10066 CN Bad implementation of cloneable idiom
Classification: Intentional

The StringBuilder class implements the Cloneable interface without overriding the `clone()` method. Since each class in Java inherits from the Object class which has a default `clone()` method, then calling `clone()` in StringBuilder will call the default Object class implementation of it. Although the code does not cause any type of failure, implementing Cloneable without overriding `clone()` is redundant and could be removed from the class definition.

Warning Message: 10067 Dm: Dubious method used
Classification: False Positive

The `PrintStackTrace` method of NestableDelegate relies on the default encoding when it instantiates a new PrintWriter object on line 292. Since the object passed into the PrintWriter constructor (`PrintStream out`) uses the default encoding, the method does not contain any bugs. Line 292 would have been problematic if it was using a constructor that took in a File object, which may have encodings different from default.

Warning Message: 10068 Dm: Dubious method used
Classification: False Positive

The warning points to line 110 for using the random method in `nextDouble()` within the `nextInt(int n)` function instead of using Java's version of `nextInt(n)` which would do the same operation. No fault exists and `nextInt(int)` will correctly output a random number within the range from 0 inclusive to n exclusive.

Warning Message: 10069 Eq: Problems with implementation of equals()
Classification: False Positive

The function description says that objects with the same class names are equal, therefore checking class names of the two objects being compared is acceptable.

Warning Message: 10070 Eq: Problems with implementation of equals()
Classification: False Positive

See explanation for 10069.

Warning Message: 10071 ES: Checking String equality using == or !=
Classification: Bug

Line 614 compares strings using ==, which is bad practice since two strings can have identical values but are instantiated in different memory locations. Since == checks equality of memory locations, this could evaluate two strings to be false even though their literal value is identical. This can be fixed by substituting == with the default `String.equals()` method to check value equality.

Warning Message: 10072 ES: Checking String equality using == or !=
Classification: Bug

See explanation for 10071, except use Line 4865 as reference.

Warning Message: 10073 ES: Checking String equality using == or !=
Classification: Bug

See explanation for 10071, except use Lines 385, 389, 393, 397, 401,405, and 409 as references.

Warning Message: 10074 ES: Checking String equality using == or !=
Classification: Bug

See explanation for 10071, except use Line 485 as reference. In this scenario, the explanation applied to the second clause in the if condition (`previous.getValue() == value`).

Warning Message: 10075 IM: Questionable integer math
Classification: False Positive

Line 649 in the function binarySearch correctly computes the midpoint by shifting the result right by one instead of dividing by two. It is impossible for an overflow to occur when shifting because low and high are guaranteed to be non-negative numbers. Both low and high will have leading 0s in their signed representation. When you shift right a 0 will always be shifted in, so an overflow cannot occur.

Warning Message: 10076 NP: Null pointer dereference
Classification: False Positive

Line 64 in the negate function (`BooleanUtils.java`) explicitly returns the value `null` when the Boolean parameter passed in is `null`, which is valid behaviour according to its function description.

Warning Message: 10077 NP: Null pointer dereference
Classification: False Positive

Line 314 in the toBooleanObject function explicitly returns the value `null` when the value equals the `nullValue` passed in the parameters, which is expected behaviour according to its function description.

Warning Message: 10078 NP: Null pointer dereference
Classification: False Positive

See explanation for 10076, except use Line 226 as reference.

Warning Message: 10079 NP: Null pointer dereference
Classification: False Positive

See explanation for 10077, except use Line 346 as reference.

Warning Message: 10080 NP: Null pointer dereference
Classification: False Positive

The warning points at Line 538 where the function `toBooleanObject`() returns `null` when all the parameter passed does not match any of the cases in the function, which is expected behaviour from the description of the function.

Warning Message: 10081 NP: Null pointer dereference
Classification: False Positive

The warning points to lines 566 and 573 where `null` is returned. However, the code behaves according to how it is described in the comments above. If the first string is `null`, then `null` should be returned if the other parameters are true, false, and `null`. `null` should also be returned if the first string is the actual `nullString`, which is what the code does. `null` being returned in some cases is expected for the function.

Warning Message: 10082 REC: Runtime Exception Capture
Classification: Intentional

Inside the catch block in of lines 97-99, the programmer does not handle the case where a RuntimeException is caught. The catch block takes any Exception object caught and handles them as all the same. The warning could be suppressed by adding a separate catch block for RuntimeException and adding line 98 into it.

Warning Message: 10083 Se: Incorrect definition of Serializable class
Classification: Bug

Since the FastDateFormat class implements Serializable from extending the Format class, all the fields identified in the class must either be marked as transient or be serializable. On line 137, the field `mRules` is not marked as transient and its type is not serializable. Parsing an instance of FastDateFormat that makes use of Serializable behaviour will fail because all fields do not meet the Serializable requirements. A solution to this is to either to make the Rule class implement Serializable or mark `mRules` as a transient field.

Warning Message: 10084 UrF: Unread field
Classification: False Positive
The field key is not used in the IntHashMap class and could be removed if classified as Intentional, but we classify it as False Positive because the field could be used in the future when extending the class with new functionality.

Section B

Upon running Coverity on our source code for Part I, 6 warnings are reported. Of the 6 results, we analyze two specific warnings that detect both a legitimate defect and a false positive in our codebase.

Warning Message: 10321 Resource leak
Classification: Bug

For the resource leak warning, we instantiate two reader stream objects: FileReader `fstream` and BufferedReader `br` when we read a file in the function `readFile`. After using the two streams, both fstream and br are never closed. Omitting the close() method will result in a resource leak warning since the system resources associated with the two stream will never be released back and will not be available for future use. A fix to this bug would be to call close() for both fstream and br once the file read is finished. These close() statements can be inserted immediately after the while loop the ends on line 53.

Warning Message: 10324 Eq: Problems with implementation of equals()
Classification: False Positive

We overrode the equality function for our PiPairs class. A PiPair consists of two Nodes with the relevant confidence levels and support between them. The equality function checks if some PiPair object is equal to the current PiPair and does comparisons based on the values of the Nodes. Coverity detects a bug in the equals implementation: `PiPairs defines equals(PiPairs) method and uses Object.equals(Object)`. This bug means that if a PiPair object is compared with a non PiPair object using equals, it will use the default `equals(Object)` method. In our code, we only ever use PiPair's equals when comparing two objects that are of type PiPair. Thus, this is not actually a bug.