CS 447

Group 65

Yunji Zhang, #20539393, y755zhan@uwaterloo.ca

Xingyu Xia, #20544730, x7xia@uwaterloo.ca

Yishi Wang, #20516631, y785wang@uwaterloo.ca

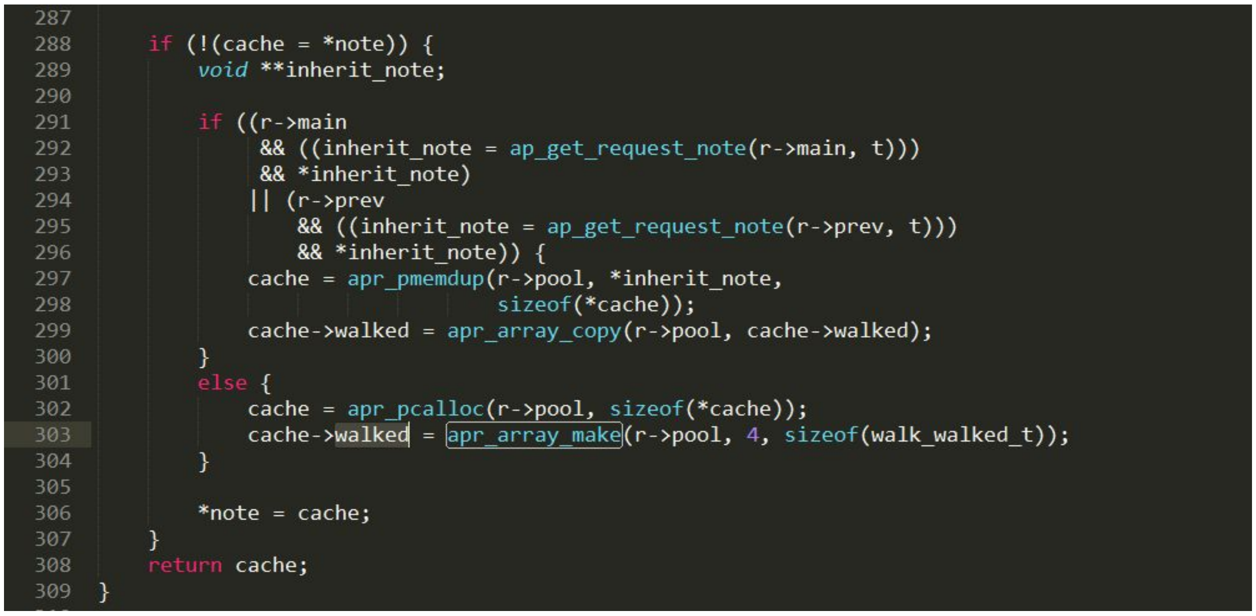
Part(I)

Part (b)

- Reasons that false positive occurs
    - Reason 1: Indirect function call
      The call of the other function of the pair may be contained in another function. After expanding that function, the function pair will be in the same function scope.

    - Reason 2: Unnecessary pair
      It might be a good style or handy operation to use a pair of functions; however, it is not necessary to always use them in pairs. If a programmer used the function pairs for a number of times and the programmer decides to use one of the functions for another purpose, which does not require the other function in the pair, then the false positive will occur.

- Identify two false positive pairs
    - First pair: (apr_array_make, apr_array_push)   Number of bug locations: 45
      apr_array_make is a function to create an array, while apr_array_push is a function to push element into an array. It is a good style to create and push so that empty array is always avoided. But create without push should not be considered as a bug, for programmers may want leave the pushing part to other functions.
      The code in screenshot is from walk_cache_t *prep_walk_cache(apr_size_t, request_rec *), request.c. The function is for checking the available cache that it can work with. When there is no parent or prior cached request, it will create cache for future use. In this case, the function wants to create an array for future use, therefore apr_array_push is not needed

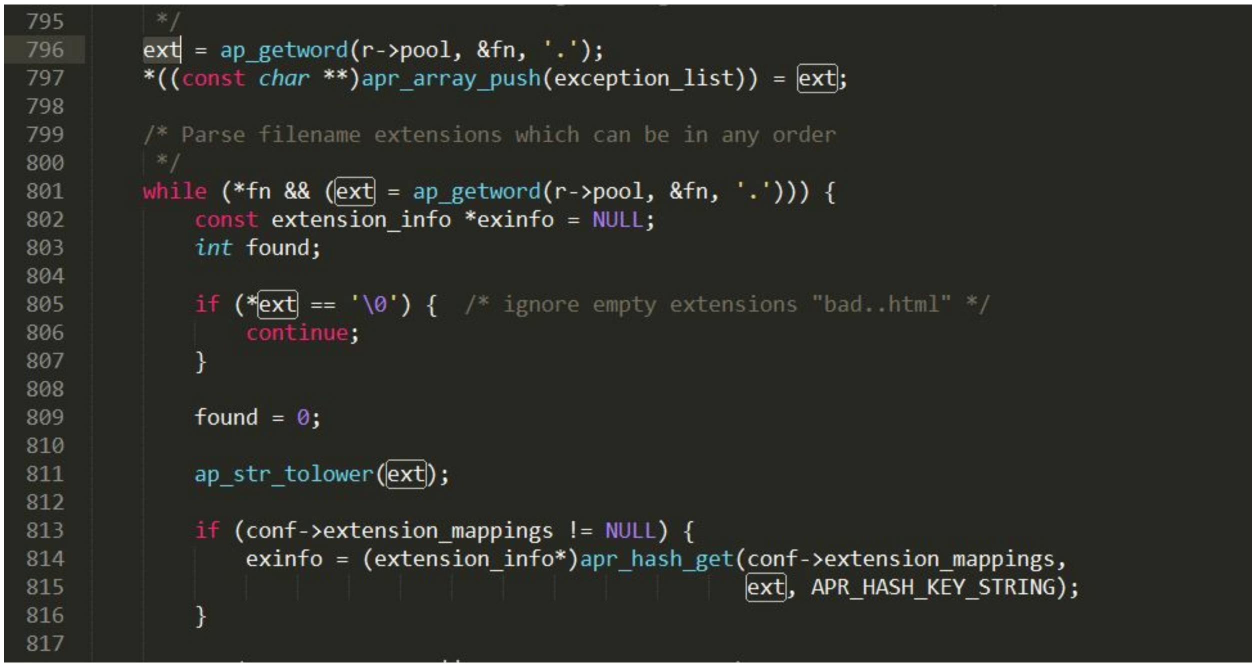


```
    if (!(cache = *note)) {
        void **inherit_note;

        if ((r->main
             && ((inherit_note = ap_get_request_note(r->main, t)))
             && *inherit_note)
            || (r->prev
                && ((inherit_note = ap_get_request_note(r->prev, t)))
                && *inherit_note)) {
            cache = apr_pmemdup(r->pool, *inherit_note,
                                sizeof(*cache));
            cache->walked = apr_array_copy(r->pool, cache->walked);
        }
        else {
            cache = apr_pcalloc(r->pool, sizeof(*cache));
            cache->walked = apr_array_make(r->pool, 4, sizeof(walk_walked_t));
        }

        *note = cache;
    }
    return cache;
}
```

    - Second pair: (ap_str_tolower, apr_pstrdup)   Number of bug locations: 4

ap_str_tolower is a function which converts the input string to all lower cases, while apr_pstrdup a function which duplicates a string into an allocated memory. It might be handy to use this pair of function when copy a string to another memory location, but obviously they are not required by each other.

The code in the screenshot is from int find_ct(request_rec *), mod_mine.c. The while loop is for keeping track of those filename components that are not associated with extensions indicating metadata. The char *ext is a list of names of all exceptions. The function call ap_str_tolower(ext) to get a lowercase string as a key to query a hashmap, therefore there is no need to duplicates it into memory. apr_pstrdup is not needed here.



```
     */
    ext = ap_getword(r->pool, &fn, '.');
    *((const char **)apr_array_push(exception_list)) = ext;

    /* Parse filename extensions which can be in any order
     */
    while (*fn && (ext = ap_getword(r->pool, &fn, '.'))) {
        const extension_info *exinfo = NULL;
        int found;

        if (*ext == '\0') {  /* ignore empty extensions "bad..html" */
            continue;
        }

        found = 0;

        ap_str_tolower(ext);

        if (conf->extension_mappings != NULL) {
            exinfo = (extension_info*)apr_hash_get(conf->extension_mappings,
                                                   ext, APR_HASH_KEY_STRING);
        }

```

Part (c)

1. Preparation

1.1 The input file is parsed into a hashmap scopes with type HashMap< String, ArrayList<String> >. The key is the scope name (like "scope1", "scope2"), and the value is an array (like "A", "B", "C"...) of function names called within the key scope.

1.2. Two hashmaps are created. funcApparence with type HashMap< String, ArrayList<String> >, the key is a function name (like "A"), the value is an array of scope names (like "scope1", "scope2"...) that calls this function. pairFrequency with type HashMap< Pair, Integer >, the key is a pair of functions, and the value is the number of occurrence of the pair.

2. Algorithm Explanation

2.1. Abstraction:

For every bug b in function f the program found (we refer the function that is missing as "absent function", and the function that is reported as "present function"), we do not report it until we finish analyzing (expanding) f. To analyze f, the algorithm recursively expand each function in f; if the absent function is found within other functions, the bug will not be reported. However, this process may lead to a situation where the absent function being found in other functions is paired with another present function. In this case, we want to find a same number of absent functions. Therefore, we adopt two indices NUM and ELEVEL for number of absent functions we need and Expansion Level respectively. The expansion of present function and absent function is avoided. Here is the algorithm logic.

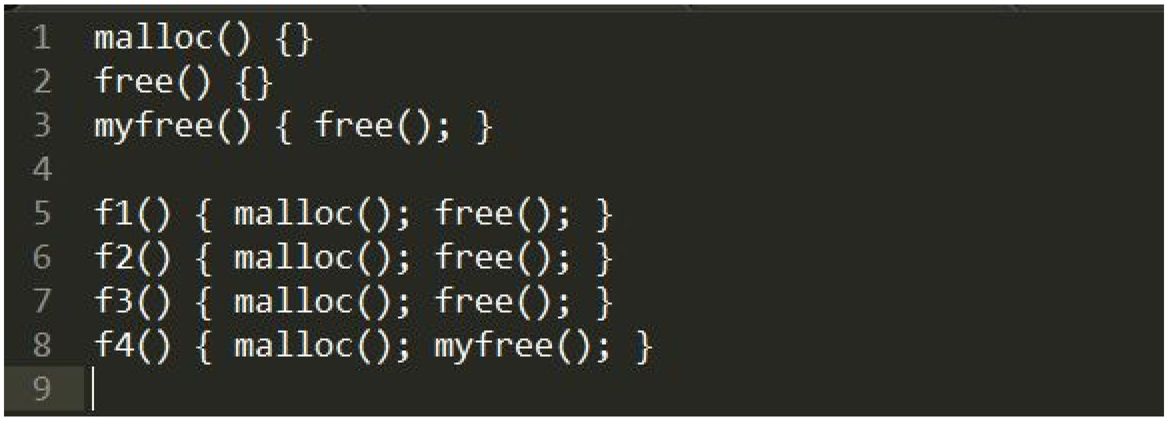
- If ELEVEL = 0, stop and report the bug
- For each function call m in f
    - - if m is the missing function we are looking for, decrement NUM
    - - if m is another present function, increment NUM
    - - else
        - - If m does not contain any other functions calls, delete m
        - - else, substitute m with all functions it contains. They will be analyzed in the next loop.
- If all m have been expanded:
    - - if NUM <= 0, stop and do not report the bug
    - - else decrement EL and start the loop again.

The algorithm will terminate at either EL reaches 0 or we have successfully found the missing function. The bug will be reported in the former situation, but will not be reported in the latter situation.

The code for the algorithm is a function called interProcedureExpand(String, String, String), locating at the file pi/partC/ConfidenceCalc.java.

3. Example and Experiment

3.1 A general example



```
malloc() {}
free() {}
myfree() { free(); }

f1() { malloc(); free(); }
f2() { malloc(); free(); }
f3() { malloc(); free(); }
f4() { malloc(); myfree(); }
```

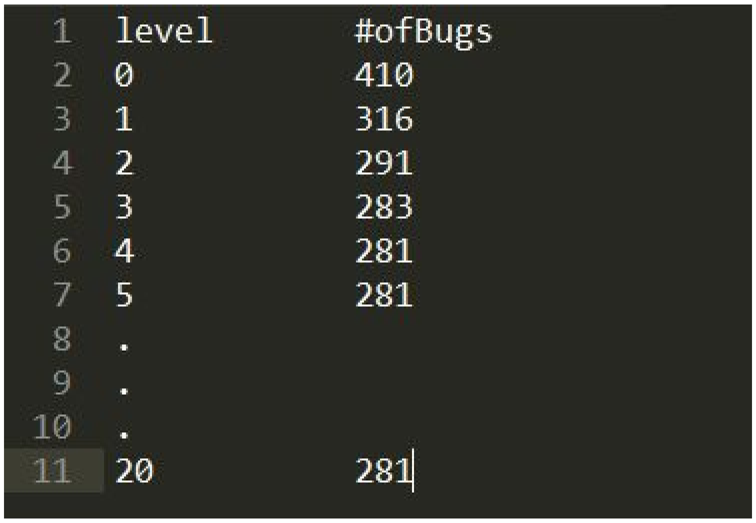
We tested the above example input with both the default algorithm and inter-procedural algorithm of depth 1. The T_SUPPORT and T_CONFIDENCE are set to be 3 and 65 respectively.

For the default algorithm, the function will get the Support({malloc, free}) = 3, and Support({malloc}) = 4. Since the confidence is ¾ = 75% > 65%, it will lead to a bug report in function **f4()**. However, this is a false positive since the **free()** is contained by **myfree()**. In runtime **myfree()** will call **free()**, then **malloc()** and **free()** appear in the same function scope.

On the other hand, the inter-procedural algorithm expands **myfree()** and replaces it with all functions **myfree()** calls. As a result, in the bug checking phrase, the program will find **malloc()** and **free()**'s presence in the **f4()** together, so the bug report will not be triggered.

3.2 Experiment on Apache httpd server file

From the aforementioned example we know that the inter-procedural algorithm we implemented can effectively reduce false positives caused by indirect function calls. Then we made an experiment on test3, by setting T_SUPPORT as 3 and T_CONFIDENCE as 65, the variable is the level of expansion. We tested it from level 0 to level 20, and record the lines of bugs it reported. Here is the result:



```
level      #ofBugs
0          410
1          316
2          291
3          283
4          281
5          281
.
.
.
20         281
```

As the data shows, the lines of bug reported decreases as the level of expansion goes up. Then it stays at 281 when the level reaches 4, we guess that it is because that most functions have reached its bottom after being expanded 4 times. So the algorithm can remove false positive effectively, as what we expected.

Part(II)

Part(a)

CID: 10001

False Positive

The referenced type 1x0 is null checked at line 520, and if it's null, sb.append(" x0,") is called. In this situation, at line 532, since sb is not an empty string anymore, then it will throw an exception and line 537 is jumped over. Therefore, there is no worry about unboxing issue at line 537.

CID: 10004

Bug

rootDirectory can be null, so that rootDirectory.getOffest() might throw an uncatched exception. It' a bad practice and null check should be added to avoid the potential exception.

FIle: TiffImageWriterLossless.java
Line: 306

```
 writeImageFileHeader(headerBinaryStream, rootDirectory.getOffset());
```

CID: 10009

Bug

bitsPerPixel * width can cause overflow because they are evaluated using 32-bit arithmetic despite the fact bytesPerRow is 64 bit. Casting bitsPerPixel and width to long fixes this problem.

FIle: DataReaderStrips.java
Line: 203

```
 final long bytesPerRow = (bitsPerPixel * width + 7) / 8;
```

CID: 10010

Bug

read2Bytes might throw an IOExecption that will be caught by finally. This happens before the "notFound" flag is set. Later the resource of "is" will not be released due to the error state of "notFound". This bug can be fixed by adding a flag that indicates "is" is opened or not. This flag should be set right after "is" is opened and whether "is" should be caused should depend on this flag instead of "notFound"

FIle: PsdImageParser.java
Line: 309

```
 if (notFound && is != null) {
```

CID: 10011

Intentional

The developer probably forgot to close outputStream because he/she needs to return the value of outputStream. The value the developer desires to return should be stored in the heap, then close outputStream and finally return the value in the heap.

CID: 10016

Bug

Dividing a double by an int might lose the remainder of the quotient. Should make sure all values in the calculation are double to remain accuracy.

FIle: ColorConversions.java

Line: 734

double var_Y = (L + 16) / 116;

CID: 10017

Bug

File: BinaryConstant.java

Line: 31-33

Code: public BinaryConstant clone() throws CloneNotSupportedException {
    return (BinaryConstant) super.clone(); }

Since the java object class does not implements the Cloneable interface, if a clone() method is called without Cloneable interface, it will throw an exception. In other to solve the problem, implement a Cloneable interface that modifies the object class' behavior. For example, do something like

Public class BinaryConstant implements Cloneable {
        Public Object clone() { … } }

CID: 10018

Bug

File: PluginManager.java

Line: 190

Code: throw new PluginException("Unable to locate plugin rules for plugin" + " with id [" + id + "]"
+ ", and class [" + pluginClass.getName() + "]" + ":" + e.getMessage(), e.getCause());

The pluginClass is null checked on line 173 of file Declaration.java, but when pass it to class findLoader(), the function pluginClass.getName() is called which lead dereferencing to a null value. The programmer should avoid call a relative function from a null variable.

CID: 10027

Bug

File: FromAnnotationsRuleModule.java

Line 172

Code: visitElements(annotatedElements);

The resource annotatedElements is not closed or saved in visitElements() function, then there would be no control of this resource when the visitElements() is done. In order to solve it, the programmer should handle the acquired resource properly.

CID: 10028

False Positive

Since the comments at the class above says that the implementation is about a mutable undirected graph, therefore the head and tail order does not matter.

CID: 10029

Bug

File: AbstractExporter.java
Line: 81
Code: to(new OutputStreamWriter(checkNotNull(outputStream, "Impossibe to export the graph in a null stream")));
The OutputStreamWriter is constructed with single argument, then the function would use default file encoding format based on the current working OS. If someone else uses different OS with different default setting, the result might be different. In order to solve this issue, the file encoding format should be choose like
OutputStreamWriter(..., Charset.forName("UTF-8"));

CID: 10030

Bug
File: DisjointSetNode.java
Line: 117-120
Code: public int compareTo(DisjointSetNode<E> o) {
    return rank.compareTo(o.getRank());
This problem could be solved by implementing an equals() method and let it has same behavior as compareTo(). Now the situation is that the return value for compareTo() is not the same as the as the inherits equals(). If the return values are not same, the error occurs. So, it a class implements comparable function, then the equal() function is also needed to override the inherited equals() function.

CID: 10032

Intentional
The explanation given by Coverity: “This class extends a class that defines an equals method and adds fields, but doesn't define an equals method itself.” However, the developer never used the equal method of this class; therefore, the developer might have skipped defining a new equal method on purpose. This is a bad practice because it is risky for other developers and future developers. It would be very easy to mis-use the outdated equal method and cause a problem.

File: SynchronizedMutableGraph.java
Line: start from 25

CID: 10033

Intentional (bad practice)
Implementing serializable with comparators could help ordering methods in serializable data structures. Since the class is not used in a way where serialization is required, it’s should be considered as intentional.

CID: 10035

This is warning is similar as for CID: 10033, please refer to CID: 10033

CID: 10039

False Positive
The developer means to call the overridden method so the superclass method does not necessarily have to be available.

CID: 10048

Bug
File: Array2DRowFieldMatrix.java

Line: 105
Code : super(extractField(d));
Since there is no guarantee that the input variable d is not null, in this situation, the null case check step should be performed before any other function calls. Therefore, the null check function from line 109-111 should be moved to the beginning of the function.

CID: 10059

Bug
File: BrentOptimizer.java
Line: 169, 174
Code: if ((fu <= fw) || (w == x)) {
          } else if ((fu <= fv) || (v == x) || (v == w)) {
In java, exact comparison (aka ==) cannot be used between two double values. Instead, there are few more ways that could help programmer to compare such values. For example, use build in function like double1.compareTo(double2) or compare(double1, double2). Another way is using < or > to compare.

CID: 10061

Bug
File: EigenDecompositionImpl.java
Line: 1438, 1444
Code: if (dMin == dN || dMin == dN1) {
          if (dMin == dN && dMin1 == dN1) {
In java, exact comparison (aka ==) cannot be used between two double values. Instead, there are few more ways that could help programmer to compare such values. For example, use build in function like double1.compareTo(double2) or compare(double1, double2). Another way is using < or > to compare.

CID: 10062

Bug
File: FirstMoment.java
Line: 157
Code: dest.nDev = dest.nDev;
Since the function is used for copying source to dest, then for each assign command, dest should be on left hand side and source should be on right hand side. However, for the above code example, the right hand side is dest, which is wrong. Should be dest.nDev = source.nDev.

Part(b)

*The analysis output folder is located in the group_065/pii/output.

CID: 10213

Intentional

The analysis claims it should have been “%n” instead of “\n” because System.out.format takes %n as a specifier. But in fact, the developer meant to print out a newline character. However, the developer should have used System.out.println instead since no specifier was ever used here.

File: projectMain.java
Line: 139

```
 System.out.format("======================\n"); // divider
```

CID: 10209

Intentional
File: projectMain.java
Line: 89
Since we know exactly the encoding of the file we want to read, so we use FileReader witch always use the platform default encoding and that cause no errors. However, the better way is that for either general or special input file encoding format, the InputStreamReader should be used, like InputStreamReader(filePath, Charset.forName("UTF-8"));