Part(I)

(b)

Reason 1:
The likely-belief theory itself is not a declarative method to definitely indicate bugs. It’s based on statistics: it counts the occurrences of one pair and each element in that pair, sets a minimum threshold, and warns those instances who are above this threshold as bugs. This is not reasonable. No matter what threshold we choose, it NEVER can definitely says, this is a bug, because hundreds of, thousands of occurrences doesn’t mean that they must be paired.

Reason 2:
Reason 1 talks about why false positives occur from the point of quantity. But now we need to know something about “quality”, which corresponds to the semantic aspects of codes. Our bug detection method only cares about syntactical part: how many of the appear together, which one doesn’t occur in the pair... But it never sits down to examine what the two pair means from the semantic meaning. But our code is not just syntactic pairs or the physical combination. Whether they need to occur together ultimately depends on what the functions do. But our bug detection mechanism never dig deeper to that. So, this is the fundamental reason why there are false positives.

False: Number: 18 A bug: ap_ssi_get_tag_and_value in handle_fsize, pair: (ap_ssi_get_tag_and_value, strcmp)
False: Number: 18 A bug: ap_ssi_get_tag_and_value in handle_flastmod, pair: (ap_ssi_get_tag_and_value, strcmp)
False: Number: 18 B bug: apr_array_make in create_core_dir_config, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_method_list_add, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in apr_xml_insert_uri, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_directory_walk, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_add_per_dir_conf, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_make in apr_xml_parser_create, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_make in create_core_server_config, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_make in prep_walk_cache, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_file_walk, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_copy_method_list, pair:

(apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_add_file_conf, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_make in ap_make_method_list, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_make in ap_init_virtual_host, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in set_server_alias, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_location_walk, pair: (apr_array_make, apr_array_push)
False: Number: 18 B bug: apr_array_push in ap_add_per_url_conf, pair: (apr_array_make, apr_array_push)

Explanation:
1. /modules/filters/mod_include.c
bug: ap_ssi_get_tag_and_value in handle_fsize, pair: (ap_ssi_get_tag_and_value, strcmp)

“ap_ssi_get_tag_and_value” needs two field: tag and value, which are already provided in handle_fsize function. “strcmp” is a string comparison function. These two topics don’t appear to be paired always. Secondly, if we consider inter-procedural analysis, this false positive disappears. So, another reason is that it doesn’t consider inter-procedural analysis.

2. /srclib/apr-util/xml/apr_xml.c
bug: apr_array_make in apr_xml_parser_create, pair: (apr_array_make, apr_array_push)

This false positive doesn’t consider inter-procedural analysis. “apr_array_push” is called in “apr_xml_insert_uri” function, which in this case follows “apr_array_push”.

(c)

1. **Introduction:**
   How to run the code?

   The fourth command parameter to pipair to determine which function of no inter-procedural process and inter-procedural process should be implemented is a string : “-intp “.

   If pipair has no the fourth command parameter, the code will implement no inter-procedural process. Only the fourth parameter is “-intp”, can the code implement inter-procedural process.

Put Makefile, pipair, TestBug.java and bitcode file into a same directory, then implement command: make, at last implement the command:
./pipair <bitcode file> <T_SUPPORT> <T_CONFIDENCE> <-intp>,
e.g.
./pipair hello.bc 10 80 -intp, to specify support of 10 and confidence of 80% , implement inter-procedural algorithm,
or
./pipair <bitcode file> <T_SUPPORT> <T_CONFIDENCE> ,
e.g.
./pipair hello.bc 10 80, to specify support of 10 and confidence of 80%, implement no inter-procedural algorithm,
or
./pipair <bitcode file> <-intp>,
eg.
./pipair hello.bc -intp, to use the default support of 3 and default confidence of 65%, implement inter-procedural algorithm..
or
./pipair <bitcode file> ,
eg.
./pipair hello.bc, to use the default support of 3 and default confidence of 65%, implement no inter-procedural algorithm..

2. description of algorithm.

There are two kinds of algorithm to implement the inter-procedural process. I will analysis these two algorithms and get the result which one is better.

The first algorithm:
The main point : From the output of (a) to find the which one is false positive bug caused by inter-procedural problem.

The detail:

1. pick one bug from the output of (a).
   Like bug: A in scope2, pair: (A, B), support: 3, confidence: 75.00%
2. inter-procedural process：
   Getting one callee from scope2's callees respectively.
   If the callee is A then keep the callee as scope2's callee, if the callee is not A or B then remove this callee from scope2's callees and add the callees of removed callee being as a caller.
   After this, get the new callees for scope2.

   For several levels inter-procedural process：
   if implementing several levels inter-procedural process, only using the new callees of scope2 to implement the above process.
3. search new callees of scope2 to judge whether B is called or not.
   If yes, this bug is a false positive bug; if not, this bug is not a false positive bug

caused by inter-procedural problem.

4. output the bugs
5. repeat 1,2 3,4 until all bugs are judged

The second algorithm:

The difficult thing is to judge which method should be replaced and which one should be not.
When dealing all pairs together like (a), it could not find a way to deal with this problem. So we choose each pair separately, this can easily cope this problem.

The main point: not use the output of (a). For a pair of function, getting its new map considering the inter-procedural problem. Then calculate the support and confidence to judge whether there are bugs.

The detail:
We have had the original map named mapO whose key is the caller of the code and whose value is a set that stores the callees called by the caller.

1. getting a set which stores all pairs of functions need to be judged whether there are bugs for each pair.
   Notice: this set should not be got from original map's key through choosing two in it, because this will produce a large set and there are a lot of pairs which are not existed.
   We should get the set form the all the callees (map's values).
2. Getting one pair named P from the above set respectively.
3. inter-procedural process:
   getting a new map without inter-procedural function:
   processing the original map to build the new map named mapN for P.
   for every caller and its callees, if one callee of caller's callees is one function of P, then the callee will not be removed from the caller's callees; if not, the callee will be removed and add the callees of removed callee being as a caller.
   After this, get the new callees for the caller, then put the caller and its set of callees into mapN

   For several levels inter-procedural process：
   if implementing several levels inter-procedural process, only using mapN to replace the original map to implement the above process.
4. calculate the parameters (support and confidence):
   for the P, using mapN to compute the support and confidence
5. judge the P's bug based on the criterion
6. output the bugs
7. repeat 2 3,4 ,5,6until all pairs are judged

3. Experiments:

A, two algorithms experiments

Three experiments:
Test case: test3
Default parameter: support = 3, confidence = 65. inter procedural level = 1
1, running (a)   2, running the first algorithm   3, running the second algorithm

The result:

Experiment	The quantity of bugs
running (a)	205

Experiment	The quantity of bugs	The quantity of bugs as (a)	New bugs
first algorithm	165	165	0
Second algorithm	1494	2	1492

B, inter procedural level experiments

Test case: test3
Default parameter: support = 3, confidence = 65. inter procedural level = 1,2,3,4,5

Experiment for first algorithm:

level	The quantity of bugs	The quantity of bugs as (a)	New bugs comparing to (a)
1	174	174	0
2	166	166	0
3	163	163	0
4	163	163	0
5	163	163	0

Experiment for second algorithm:

level	The quantity of bugs	The quantity of bugs as (a)	New bugs comparing to (a)
1	1494	2	1492
2	5795	3	5792
3	7496	0	7496
4	7416	0	7416
5	7416	0	7416

4. Conclusion

Comparison between the two algorithms:

From the experiment A，can get the following conclusion:
The advantages for the first one:
 1, the speed is faster 2, can reduce some false positive bugs
The disadvantages for the first one:
 1, precision is very low.
 it could only reduce a small part of bugs, and there may be some false bugs in the left bugs, because if implementing inter-procedural process before calculating support and confidence, some left bugs may be not bugs.
 2, bugs found are less
The advantages for the second one:
 1, reduce most false positive bugs caused by inter-procedural greatly.
 2, find much more new bugs
The disadvantages for the second one:
 the speed is slow relatively.

From the experiment B，can get the following conclusion:

The first algorithm:
with the increase of inter-procedural level, the quantity of bugs decreases and keeps the same value finally.

The reason for this: with the level increase, the inter-procedural problem will not exist and once a bugs was detected as false positive bug by inter-procedural process, then the bug will not be detected as bug any more in the later level process.

The second algorithm:
with the increase of inter-procedural level, the quantity of bugs increase greatly at first , then decreases, keeps a fixes value finally; the quantity of bugs as (a) decreases at first , keeps value as 0 finally .

The reason for this: with the level increase, the inter-procedural problem will decrease and not exist finally. Why the quantity of bugs will decrease is that although a bug was detected as a bug by inter-procedural process at early level inter procedural process, the bug may not be detected as bug in the later level process as the pair’s support and confidence change and not be judged as bug.

In summary:

Although these two algorithms can reduce false positive bugs, but these output may also have false positive bugs, because different levels of inter procedural process will influence the pair’s support and confidence and may be the next level will make the past bugs be a false positive bugs.

Though experiments, though the second one is slow relatively, the second algorithm is much better for reducing false positive bugs greatly and increasing large of new bugs. So the second algorithm is much better.

(d)

Reduce false positive:

Consider this situation:

```
scope1{
   A,B,C
}
scope2{
   A,B,D
}
scope3{
   B,C,D,A
}
scope4{
     A,C
}    //A is bug, because confidence > 75%
scope5{
     B,C,D
}    //But, B is not a bug, because confidence = 60% <75%
scope6{
     B,D
}
```

In this case, our code from (a) will warn bug(Let's assume support >=3, confidence>=75% is the minimum requirement to become a bug):

```
bug: A in scope4, pair: (A,B)
because:
     support(A,B) = 3, support(A) = 4
     -> confidence = support(A,B)/support(A) = 3/4 = 75%

     But are B in scope5, scope 6 bugs? No!
     support(A,B) = 3, support(B) = 5
     -> confidence = support(A,B)/support(B) = 3/5 = 60% < 75%
```

This is odd. Our belief is that "A and B should be paired". But, in this particular case, A appears only once and becomes a bug, however, B appears twice and should be a worse bug but now it's not a bug, which does not make sense. (Note: not from the view of bug criteria, which is support >= 3 and confidence >= 75%, but from the general belief that A and B should be paired)

So, this means that A is “unfairly” reported as a bug, which is a false positive. Whether A or B is a bug or not depends on both of their behavior -- situations they are separate, but not only one’s, simply because our belief is that “A,B should be paired”.

Therefore, our solution is to make it harder to let A be reported as a bug: enforcing both confidence of A and B be larger than minimum threshold (in this case, 65%). If this condition is satisfied, then A will not be reported as a bug, which eventually reduces false positives.

Find more bugs:

Sometimes, we not only need “A and B should be paired”, but also “their frequency of occurrences also must be same”.

A typical and concrete example is “double free”. Double free() is quite simply where a chunk of memory that was previously allocated by one of the allocation routines that is later freed more than once. [1] For example,:

```
void *ptr = malloc(size); //malloc called once
if(ptr != NULL){
    free(ptr);
    free(ptr);      // free called twice
}
```

In this example, if we count the number of “malloc” and “free”, then we can find this bug. Because malloc appears once, but “free” twice, which might a mismatch that causes problem if it happens.

There are lots of pairs of functions that must be called together, as indicated by figure5 in [2]. For example, a file processing function should open a file and close the file after use. It’s not correct to open a file twice and close it only once, or open once and close twice; (without either one is not in this scope, as it is already predicted by solution in 1a); Lock must be required before unlock, and we can't unlock twice holding only one lock….

Though, this belief has a precondition: A and B should be paired, which means that it's based on our previous result. In our analysis before, we already know that both confidence of A and B should be larger that threshold to indicate that A and B are much more likely to appear together. Therefore, **we first confirm that a pair of functions need to occur together, then we are allowed to analyze if their number of occurrences are same or not.** If not, it is quite possible that something goes wrong. For example, forgetting to unlock at least once less or double free, etc..

A sample but output of test code is:

```
bug: apr_thread_mutex_unlock in apr_global_mutex_trylock, pair:
(apr_thread_mutex_lock, apr_thread_mutex_unlock), support: 43, confidence: 91.49%
```

This is our bug output from 1(a). We can see that support is greater than 43, and the confidence of “apr_thread_mutex_lock, apr_thread_mutex_unlock should be paired” is pretty high, up to 91.49%. Therefore, we can infer that apr_thread_mutex_lock and apr_thread_mutex_unlock should occur together.

```
bug: apr_thread_mutex_unlock in apr_file_datasync, pair: (apr_thread_mutex_lock,
apr_thread_mutex_unlock), apr_thread_mutex_unlockfrequency: 2,
apr_thread_mutex_lockfrequency: 1
```

This is the bug output from our improved solution. We can see apr_thread_mutex_unlock appears twice, but apr_thread_mutex_lock appears only once. This means that the programmer might unlock the mutex one more time somewhere in the code. This might be a potential bug that causes problem.

Referenced papers:
[1] Justin N. Ferguson. Understanding the heap by breaking it, in *IOActive,* page 23 - 24
[2] Benjamin Livshits, Thomas Zimmermann. Locating Matching Method Calls by Mining Revision History Data. Page 4

Part (II)

(a) Resolving Bugs in Apache Commons

CID:10065
Category:False Positives

The ‘case’ syntax is not terminated by break statements. However, the ‘return’ syntax can be used to replace ‘break’ and jump out of ‘switch’ structures. Hence, there is no fault here.

CID:10066

Category:Bug line 70

A class ‘StrBulider’ implements an interface called Cloneable. However, the class does not provide any implements of abstract methods ruled in the interface.

CID:10067

Category:Bug line 292

A object ‘pw’ is created by a class PrintWriter with default encoding depending on the host charset. It is obvious to cause the behavior to vary between platforms. A viable solution is using an API or specify a charset name or Charset object.

E.g., Line 292 can change its statement to PrintWriter pw =new PrintWriter(new OutputStream Writer(out,charsetName),false)

CID:10068

Amethod Math.random() is used to generate a random value with double type. Then the value is transformed to integer through multiplying by n. However, it is less efficient than math.nextint(), because math.random() is an internal method in the Random.nextDouble(),calling a method Random.next() twice. Math.nextint() calls the method Random.next() once.

An expected solution is shown below.

Line 110 return (int)(Math.random()*n) $\Rightarrow$ return return Math.nextint()

CID:10069 10070

Category:Bug

This method checks to see if two objects are the same class by checking to see if the names of their classes are equal. A case can have different classes with the same name if they are loaded by different class loaders. Just check to see if the class objects are the same.

Resolution: Delete the comment from line 598 to 600.

CID:10071 10072 10073 10074

Category:Bug

This code compares java.lang.String objects for reference equality using the == or != operators. The same string value may be represented by two different String objects. Hence using == or != will create wrong results even if two String objects have the same String comment. Consider using the equals(Object) method instead.

CID:10075

Catagory:Bug

Using >> represents the division. However, considering the operand low and high with big value, overflow will occur and cause the wrong result, because “<<” operand focuses on the unsigned integer. Using “<<<” can resolve the trouble, because it is used to signed integer.

CID:10076-10081

Category:Intentionals Line 65 226 346 538 567

Both methods will return Boolean True, Boolean False or null. Since the return type is declared Boolean directly, returning a null value will lead to a null pointer exception. I believe the warning points to code that the developer intentionally created, and it is inevitable. Thus a expected resolution is proposed.

E.g.

```
Public static Boolean methodName(Boolean object)
{
    Try{
        …
```

```
            …
            …
            If(bool==null)
                  {
                  return null;
                  }
}catch(NullPointerException )
            {
                  return null;
            }
```

CID:10082
Category: Intentionals

Coverity declares that there is a Runtime Exception triggered in the code. However, Line 96 calls a class method named getMethod, which may activate a NoSuchMethodException when there is no method with the name of initCause. Perhaps the developers consider both of two exceptions are subclasses of Excpetion, and using Exception statement can contain both of the situation.

CID:10083
Category:Bug

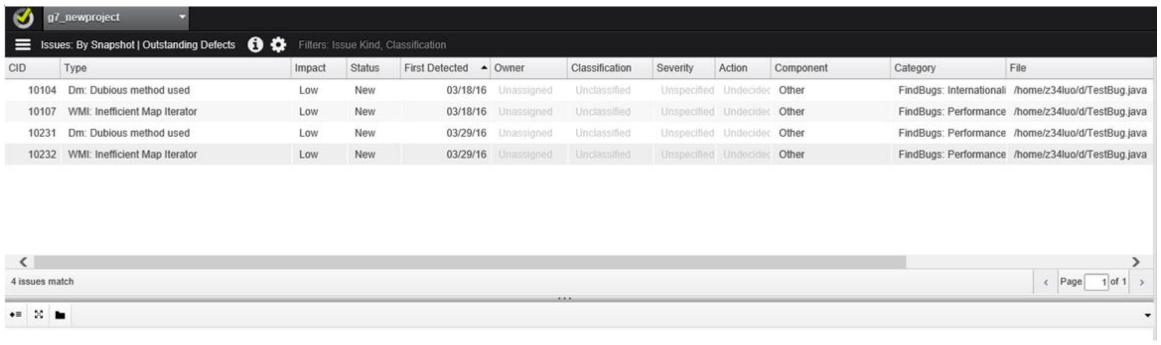
An array mRule is explicitly declared only with an interface statement called Rule, but not stated which type or class the elements in the array are belonging to. Hence Coverity recognized that this array cannot be serialized.
A solution is shown below.

```
Line 137 private Rule[] mRules;  →
Private Rule[] mRules;
for (i=0; i< mMaxLengthEstimate;i++)
{
      mRules[i]=new ClassLoader();
}
```

CID:10084
Category:False positives

The Entry method receive the arguments called “key” and then assign it to the attribute in the local class. It seems right because the value of “Key” is definitely used. Hence there is no bug here.

(b) Analyzing Your Own Code


g7_newproject

Issues: By Snapshot | Outstanding Defects

Filters: Issue Kind, Classification

CID	Type	Impact	Status	First Detected	Owner	Classification	Severity	Action	Component	Category	File
10104	Dm: Dubious method used	Low	New	03/18/16	Unassigned	Unclassified	Unspecified	Undecidec	Other	FindBugs: Internationali	/home/z34luo/d/TestBug.java
10107	WMI: Inefficient Map Iterator	Low	New	03/18/16	Unassigned	Unclassified	Unspecified	Undecidec	Other	FindBugs: Performance	/home/z34luo/d/TestBug.java
10231	Dm: Dubious method used	Low	New	03/29/16	Unassigned	Unclassified	Unspecified	Undecidec	Other	FindBugs: Performance	/home/z34luo/d/TestBug.java
10232	WMI: Inefficient Map Iterator	Low	New	03/29/16	Unassigned	Unclassified	Unspecified	Undecidec	Other	FindBugs: Performance	/home/z34luo/d/TestBug.java

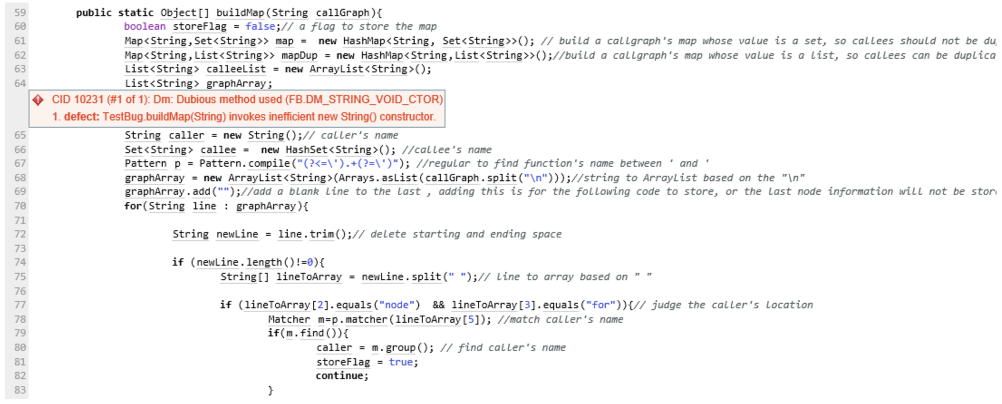
4 issues match

Page 1 of 1

Figure 12

As is shown in Figure 12, our code is analyzed by Coverity and has 4 bugs with low impact. Hence our code seems reliable.

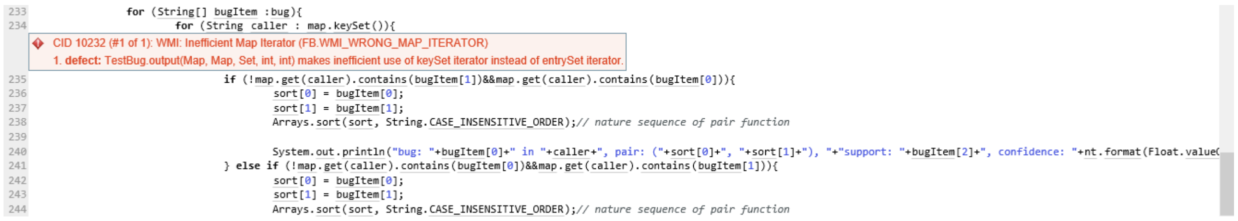
CID: 10231


```
    public static Object[] buildMap(String callGraph){
            boolean storeFlag = false;// a flag to store the map
            Map<String,Set<String>> map =  new HashMap<String, Set<String>>(); // build a callgraph's map whose value is a set, so callees should not be du
            Map<String,List<String>> mapDup = new HashMap<String,List<String>>();//build a callgraph's map whose value is a list, so callees can be duplica
            List<String> calleeList = new ArrayList<String>();
            List<String> graphArray;
```
CID 10231 (#1 of 1): Dm: Dubious method used (FB.DM_STRING_VOID_CTOR)

1. **defect:** TestBug.buildMap(String) invokes inefficient new String() constructor.
```
            String caller = new String();// caller's name
            Set<String> callee =  new HashSet<String>(); //callee's name
            Pattern p = Pattern.compile("(?<=\').+(?=\')"); //regular to find function's name between ' and '
            graphArray = new ArrayList<String>(Arrays.asList(callGraph.split("\n")));//string to ArrayList based on the "\n"
            graphArray.add("");//add a blank line to the last , adding this is for the following code to store, or the last node information will not be stor
            for(String line : graphArray){

                    String newLine = line.trim();// delete starting and ending space

                    if (newLine.length()!=0){
                            String[] lineToArray = newLine.split(" ");// line to array based on " "

                            if (lineToArray[2].equals("node")  && lineToArray[3].equals("for")){// judge the caller's location
                                    Matcher m=p.matcher(lineToArray[5]); //match caller's name
                                    if(m.find()){
                                            caller = m.group(); // find caller's name
                                            storeFlag = true;
                                            continue;
                                    }
```
Figure 13

Because using constructors without any arguments for String object is an inefficient method, wasting the space in memory. Moreover, it is easy to be confused with the empty String. In Java using an empty String serve the same propose.

CID:10232


```
            for (String[] bugItem :bug){
                    for (String caller : map.keySet()){
```
CID 10232 (#1 of 1): WMI: Inefficient Map Iterator (FB.WMI_WRONG_MAP_ITERATOR)

1. **defect:** TestBug.output(Map, Map, Set, int, int) makes inefficient use of keySet iterator instead of entrySet iterator.
```
                            if (!map.get(caller).contains(bugItem[1])&&map.get(caller).contains(bugItem[0])){
                                    sort[0] = bugItem[0];
                                    sort[1] = bugItem[1];
                                    Arrays.sort(sort, String.CASE_INSENSITIVE_ORDER);// nature sequence of pair function

                                    System.out.println("bug: "+bugItem[0]+" in "+caller+", pair: ("+sort[0]+", "+sort[1]+"), "+"support: "+bugItem[2]+", confidence: "+nt.format(Float.value(
                            } else if (!map.get(caller).contains(bugItem[0])&&map.get(caller).contains(bugItem[1])){
                                    sort[0] = bugItem[0];
                                    sort[1] = bugItem[1];
                                    Arrays.sort(sort, String.CASE_INSENSITIVE_ORDER);// nature sequence of pair function
```
Figure 14

As is shown in Figure 14, Coverity recommends that we should use Entry to traverse the map rather than we use the above one. In fact, it is correct. One method traverses every key and then seek the corresponding value. It seems duplicate and redundant because we can directly obtain the value by traversing Entry.

Let us looking up the HashMap.get() and find the reason.

```
public V get(Object key) {
        if (key == null)
                return getForNullKey();
        int hash = hash(key.hashCode());
        for (Entry<K,V> e = table[indexFor(hash, table.length)];
                  e != null;
                  e = e.next) {
                Object k;         hash
                if (e.hash == hash && ((k = e.key) == key || key.equals(k)))
                        return e.value;
        }
        return null;
}
```

As is shown above, the code implement a hash method to calculate the hash value, then obtaing the entry from the hashtable. Regardless of either calculating the hash value or execute the loop and equals method, it is lavish for limited CPU resource.

Solution:

```
for (Map.Entry<String, List<String>> entry :map.entrySet()) {
if (!entry.getValue().contains(bugItem[1])&& entry.getValue().contains(bugItem[0]))
        {
                                                ………../Basic block
        }
```