SE 465
Project - 1

Member:
Shoubo Wang (20476417) sbwang@uwaterloo.ca
Yuxin Tang (20463362) y66tang@uwaterloo.ca
Aishwarya Pathak (20433910) a2pathak@uwaterloo.ca

Q1b)

Pair: (*apr_array_push, apr_hook_debug_show***)**
Support: 28, Confidence: 87.50%
File: core.c
Line: 552-559

```
AP_CORE_DECLARE(void) ap_add_per_dir_conf(server_rec *s, void *dir_config){
        ...
        void **new_space = (void **)apr_array_push(sconf->sec_dir);
        ...
}
```

File: apr_hooks.c
Line: 326 - 356

```
APU_DECLARE(void) apr_optional_hook_add(const char *szName,void (*pfn)(void),
                                        const char * const *aszPre,
                                        const char * const *aszSucc,int nOrder){
        ...
        pHook=apr_array_push(pArray);
        ...
        if(apr_hook_debug_enabled)
                apr_hook_debug_show(szName,aszPre,aszSucc);
}
```

Explanation:

This pair of function calls is reported as a bug but it is actually a false positive, and these two functions are '*apr_array_push*' and '*apr_hook_debug_show*'. The functionality of *apr_array_push* is to push an object type array to push an item to the array and return its new pointer. The functionality of *apr_hook_debug_show* is to print the contents of a object for debugging purposes. Because *apr_hook_debug_show* is used mainly for debugging, it is often paired with *apr_array_push* to display its content. However, *apr_hook_debug_show* is not needed after *apr_array_push* because the content does not need to be displayed. Thus, this pair of function call is not a bug.

As in the example in the source code, the first code calls *apr_array_push* inorder to push in some object. However, the second code displays the content of the *pHook* after *pArray* is pushed in during debug.

Pair: (*apr_array_push, apr_array_make***)**
Support: 40, Confidence: 86.96%

File: config.c
Line: 1869 - 1906

```
AP_CORE_DECLARE(const char *) ap_init_virtual_host(apr_pool_t *p,
                                    const char *hostname,
                                    server_rec *main_server,
                                    server_rec **ps){
        ...
        s->names = apr_array_make(p, 4, sizeof(char **));
        s->wild_names = apr_array_make(p, 4, sizeof(char **));
        ...
}
```

File: config.c
Line: 1546 - 1644

```
static const char *process_resource_config_nofnmatch(server_rec *s,
                                    const char *fname,
                                    ap_directive_t **conftree,
                                    apr_pool_t *p,
                                    apr_pool_t *ptemp,
                                    unsigned depth){
        candidates = apr_array_make(p, 1, sizeof(fnames));
        ...
        if (strcmp(dirent.name, ".")
                fnew = (fnames *) apr_array_push(candidates);
        ...
}
```

Explanation:

Like the last pair of function calls, the paired function calls *apr_array_push* and *apr_array_make* is not a bug as well. *apr_array_push* and *apr_array_make* is often called because it is very likely that after a array is created, some object will be pushed into the array. However, this is not always the case because it is perfectly legal to not push in objects after then creation of the array. For example, the array can be created with values already in it. As in the example in code 1, an array is made and values populated through the constructor, and ‘s’ is returned at the end.

Therefore, although in many instanced that some object is pushed into the array after it is made, is it still normal to create the array with pre-populated values and not push any object s into it. Therefore, this bug is a false positive.

SE465 – 1c Inter Procedure Documentation

Usage:
java pipair <input file> <support> <confidence> <Inter procedure>

Description:
<Inter procedure> - Boolean value. 'true' to enable inter procedure analysis and 'false' or leaving it blank to disable inter procedure analysis.

Once inter procedure analysis is enabled, pipair will further analyze bugs that were detected, by expanding the inner functions in the bugged function to find missing function.

Eg:

Inter-procedure analysis disabled	Inter-procedure analysis enabled
Scope1{ *A();* *Scope2();* *}* *Scope2{* *B();* *}*	*Scope1{* *A();* *Scope2();* *B();* *}* *Scope2{* *B();* *}*
A()B() Cannot be found in Scope1	1. A()B() Cannot be found in Scope1 2. Expand Scope2 3. A()B() is found
Bug is reported	A()B() is no longer a bug in Scope1

Algorithm:
Pseudo Code:

1. gather bugs that were reported at the end of pipair function, recode the bugged function and missing_inner_function
2. for each bug in bugs:
3. for each inner function in the bugged function
4. perform a DFS, the root is the inner function
5. if the DFS found the missing_inner_function, mark the bugged function as not a bug
6. print all bugs that is stilled marked as bug

Algorithm explanation:

Likely-invariants testing test the likelihood of paired function calls. In the previous version of pipair, the internal functions in each function call are paired. In the case of a missing paired function call, no further actions are taken. This could result in a false positive because for example:

```
Scope1{                          A(){
    A();                         }
    C();
    Scope2();
}
Scope2{                          C(){
    B();                         }
}
```

Assume A()B() is expected to be paired. Although in Scope1, A()B() pair does not exist directly, since Scope1 calls Scope2, and Scope2 contains B(), A() is technically paired with B() because in the execution of Scope1, B() always follow A(). Thus, the example above will result in a false positive.

The algorithm for inter procedure call will perform a DFS in every inner function in Scope1. The DFS will search for the missing function, in this case, the missing function is B(). If any of the inner function returns true for the missing function, then the pair of function is made and the false positive is avoided.

In the example, A() will return false for B(), C() will return false for B(), but Scope2() will return true for B(). Thus, A()B() pair in Scope1 is found and bug is marked as not a bug.

Experiment with test 3:

With support = 3, confidence = 65, 205 bugs were found in the test 3 when inter_procedure disabled. Same setting with inter_procedure enabled found 156 bugs.

2a)

CID 10022 - Missing call to super class
file: ListOrderedMap.java:262

Triage - False Positive
This warning is a false positive as this base class has no need to call its super class. All other six subclass instances out of nine return the key set of the parent class as it is. The ListOrderedMap class returns a new keySetView() instance of the object instead, which is modifiable. So in both cases a KeySet() is returned.

CID 10023 - Missing call to super class
file: FixedSizeSortedMap.java:144

Triage - Intentional
This warning seems to be bad practice. The line the developer of code the developer uses:
—> Set set = map.keySet() is more or less the same as that which is used by the other six subclass instances out of nine —> Set set = super.keySet(). The warning would not appear if the developer used the second line of code instead.

CID 10024 - Missing call to super class
file: FixedSizeMap.java:144

Triage - Intentional
This warning seems to be bad practice. The line the developer of code the developer uses:
—> Set set = map.keySet() is more or less the same as that which is used by the other six subclass instances out of nine —> Set set = super.keySet(). The warning would not appear if the developer used the second line of code instead.

CID 10025 - Dereference after null check
file: ObjectGraphIterator.java:147

Triage - Bug
This is a bug because line 186 of the same file calls a method on this.currentIterator when it might be null, as indicated by the null check on line 140 of the same file. A fix is simply to add a null check for currentIterator before the method call.

CID 10026 - Unguarded read
file: FastArrayList.java:852

Triage - Bug
This is a bug because the function does not get the lock required for sublist, which could result in incorrect values or values modified by another thread being returned. An example fix would be to grab a lock before accessing sublist, following the examples provided by Coverity.

CID 10027 - Dereference null return value
file: DoubleOrderedMap.java:883

Triage - False Positive
Lines 881, 882 check to see whether the left and right children of deletedNode are null or not. Since they are not null and this is a DoubleOrderedMap class, there must be one that is greater and thus nextGreater cannot return null.

CID 10028 - Unguarded write
file: FastArrayList.java:1241

Triage - Bug
This is a bug because the function does not get the lock required for sublist, which could result in incorrect values or values modified by another thread being returned. An example fix would be to grab a lock before accessing sublist, following the examples provided by Coverity.

CID 10029 - Check of thread-shared field evades lock acquisition
file: FastHashMap.java:665

Triage - Bug
If two threads access this data at the same time, there is a chance that the field lastReturned could acquire inconsistent data due to them trying to update it at the same time. A possible fix for this would be update all other threads when one edits the value of lastReturned.

CID 10030 - Thread deadlock
file: FastHashMap.java:548

Triage - Intentional
There is only a single task/line of code that can be executed before another thread could interfere, so there can't be a deadlock. Intentional because it still might be a better idea to grab all current locks to be certain.

CID 10031 - Dereference null return value
file: TreeList.java:656

Triage - Intentional
While it seems there is a real possibility that getLeftSubTree() could be called upon a null value, we have to notice that rotateLeft() is a private function. Looking through the TreeList class, we can see that the function balance() is the only one that calls rotateLeft() and it does so only after checking that the height is value where 'faedelung' does not apply (i.e. a right subtree definitely exists). Still there is always a possibility for human error and a simple null check would be a good idea here (hence not a false positive).

CID 10032 - Check of thread-shared field evades lock acquisition
file: StaticBucketMap.java:513

Triage - Bug

If two threads access this data at the same time, there is a chance that the field bucket could acquire inconsistent data due to them trying to update it at the same time. A possible fix for this would be update all other threads when one edits the value of bucket.

CID 10033 - Arguments in wrong order
file: TreeBidiMap.java:1603

Triage - False Positive
This function call is made for the case that is INVERSEMAPENTRY, in which case the argument switch makes sense.

CID 10034 - Check of thread-shared field evades lock acquisition
file: StaticBucketMap.java:522

Triage - Bug
If two threads access this data at the same time, there is a chance that the field bucket could acquire inconsistent data due to them trying to update it at the same time. A possible fix for this would be update all other threads when one edits the value of bucket.

CID 10035 - Unguarded write
file: FastArrayList.java:1221

Triage - Bug
This is a bug because the function does not get the lock required for sublist, which could result in incorrect values or values modified by another thread being returned. An example fix would be to grab a lock before accessing sublist, following the examples provided by Coverity.

CID 10036 - Check of thread-shared field evades lock acquisition
file: FastTreeMap.java:768

Triage - Bug
If two threads access this data at the same time, there is a chance that the field lastReturned could acquire inconsistent data due to them trying to update it at the same time. A possible fix for this would be update all other threads when one edits the value of lastReturned.

CID 10037 - Dereference null return value
file: TreeList.java:683

Triage - Intentional
While it seems there is a real possibility that getRightSubTree() could be called upon a null value, we have to notice that rotateRight() is a private function. Looking through the TreeList class, we can see that the function balance() is the only one that calls rotateRight() and it does so only after checking that the height is a value where ‘faedelung’ does not apply (i.e. a left subtree definitely exists). Still there is always a possibility for human error and a simple null check would be a good idea here (hence not a false positive).

CID 10038 - Thread Deadlock
file: FastArrayList.java:1136

Triage - Intentional

There is only a single task/line of code that can be executed before another thread could interfere, so there can't be a deadlock. Intentional because it still might be a better idea to grab all current locks to be certain.

CID 10039 - Dereference null return value
file: TreeBidiMap.java:1019

Triage - False Positive
Line 1018 checks to see whether the left and right children of deletedNode are null or not. Since they are not null and this is a tree class, there must be one that is greater and thus nextGreater cannot return null.

CID 10040 - Thread deadlock
file: FastTreeMap.java:653

Triage - Intentional
There is only a single task/line of code that can be executed before another thread could interfere, so there can't be a deadlock. Intentional because it still might be a better idea to grab all current locks to be certain.

CID 10041 - Volatile not atomically updated
file: ReferenceMap.java:555

Triage - Bug
Since there is no lock held, it is possible that modCount could be overwritten by an intervening thread and the current thread would then change the value of modCount based on old values that are no longer valid, so this is a bug. Fix would be to grab a lock before modifying the values.

CID 10042 - Volatile not atomically updated
file: ReferenceMap.java:582

Triage - Bug
Since there is no lock held, it is possible that modCount could be overwritten by an intervening thread and the current thread would then change the value of modCount based on old values that are no longer valid, so this is a bug. Fix would be to grab a lock before modifying the values.

2B)

Analyzing Your Own Code

Error #1	
Meta Variable	**Value**
Checker	FB.DM_DEFAULT_ENCODING
File	/home/y66tang/jack/JWATIP465/src/pipair.java
Function	pipair.ParserAndReader(java.lang.String)
Ordered	true
Event	
Variable	**Value**
Main	True
Tag	Defect
Description	Found reliance on default encoding: new java.io.FileReader(String).
Line	65

Error#1 is detected by Coverity because when FileReader() is called with the default encoding (as shown on line 65: FileReader fileReader = new FileReader(fileLocation);) .

The constructor of FileReader is designed to use the platform default encoding if encoding is not specified, which is generally *a bad idea* since the default encoding depends on the system settings of the computer and is usually the most popular encoding among users in that locale. Therefore if a file were encoded in a different encoding, such way of reading file would not be able to successfully read in the correct content.

A possible fix for this bug is to use new InputStreamReader(new FileInputStream(filePath), encoding) and ideally get the encoding from metadata about the file.

Error #2	
Meta Variable	**Value**
Checker	FB.WMI_WRONG_MAP_ITERATOR
File	/home/y66tang/jack/JWATIP465/src/pipair.java
Function	pipair.PrintMissingPairsWithConfidence()
Ordered	true
Event	
Variable	**Value**
Main	True
Tag	Defect
Description	pipair.PrintMissingPairsWithConfidence() makes inefficient use of keySet iterator instead of entrySet iterator.
Line	210

Error #2 is identified by Coverity as a defect because it uses KeySet iterator instead of entrySet iterator. The use of KeySet does the following: retrieving all the keys (accessing the whole map), and then for some keys accessing the map again to get the desired value. A more efficient way as Coverity suggests is to iterate over the map to get map entries (Map.Entry) (couples of keys and values) while accessing the map only once.

Map.entrySet() delivers a set of Map.Entrys each one with the key and corresponding value.
Instead of the following code:

```
196         // Get the first function calls from the first method
197         HashSet<String> functionCalls = FUNCTION_MAP.get(firstKey);
198         for(String firstFunctionCall: functionCalls){
199
                        ... ...
210             int combinationCount = occurrenceCalls.get(secondCall);
                        ... ...
}
```

We could use:

```
196         // Get the first function calls from the first method
197         Set <Map.Entry<String, HashSet<String>>> functionCalls =
FUNCTION_MAP.entrySey();
198         for(Map.Entry<String, HashSet<String>>firstFunctionCall: functionCalls){
199
                        ... ...
210             int combinationCount = occurrenceCalls.get(secondCall);
                        ... ...
}
```

Request memory takes: 27 /10