UNIVERSITY OF WATERLOO

SE 465 - Project

Gwangseung 'Eric' Kim	20429117	g28kim	SE465-001
Eric McAlister	20412233	ejmcalis	CS 447
Kyle Platt	20423636	kbplatt	CS 447

March 31st, 2016

Part (1)
(b)

(apr_array_make, apr_array_push)

```
bug: apr_array_push in ap_directory_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
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_url_conf, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_copy_method_list, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in apr_xml_insert_uri, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_method_list_add, 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_location_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in ap_file_walk, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
bug: apr_array_push in set_server_alias, pair: (apr_array_make, apr_array_push), support: 40, confidence: 80.00%
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 create_core_dir_config, 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_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_make in ap_make_method_list, pair: (apr_array_make, apr_array_push), support: 40, confidence: 86.96%
```

The two functions are, as mentioned in their names, array functions that create and push to an array. The following is the source code of the highlighted ‘apr_xml_insert_uri’ function in directory ‘srclib/apr-util/xml/apr_xml.c’.

```c
/* return the URI's (existing) index, or insert it and return a new index */
APU_DECLARE(int) apr_xml_insert_uri(apr_array_header_t *uri_array,
                                    const char *uri)
{
    int i;
    const char **pelt;

    /* never insert an empty URI; this index is always APR_XML_NS_NONE */
    if (*uri == '\0')
        return APR_XML_NS_NONE;

    for (i = uri_array->nelts; i--;) {
        if (strcmp(uri, APR_XML_GET_URI_ITEM(uri_array, i)) == 0)
            return i;
    }

    pelt = apr_array_push(uri_array);
    *pelt = uri;                /* assume uri is const or in a pool */
    return uri_array->nelts - 1;
}
```

The method either returns the URI’s index or insert to an already existing array passed as a parameter, so there is no need to call apr_array_make to create an array. The following is the source code of the make and push functions for reference, in srclib/apr/tables/apr_tables.c

```c
APR_DECLARE(apr_array_header_t *) apr_array_make(apr_pool_t *p,
                                        int nelts, int elt_size)
{
    apr_array_header_t *res;

    res = (apr_array_header_t *) apr_palloc(p, sizeof(apr_array_header_t));
    make_array_core(res, p, nelts, elt_size, 1);
    return res;
}

...

APR_DECLARE(void *) apr_array_push(apr_array_header_t *arr)
{
    if (arr->nelts == arr->nalloc) {
        int new_size = (arr->nalloc <= 0) ? 1 : arr->nalloc * 2;
        char *new_data;

        new_data = apr_palloc(arr->pool, arr->elt_size * new_size);

        memcpy(new_data, arr->elts, arr->nalloc * arr->elt_size);
        memset(new_data + arr->nalloc * arr->elt_size, 0,
               arr->elt_size * (new_size - arr->nalloc));
        arr->elts = new_data;
        arr->nalloc = new_size;
    }

    ++arr->nelts;
    return arr->elts + (arr->elt_size * (arr->nelts - 1));
}
```

(ms_release_conn, strlen)

bug: ms_release_conn in mc_version_ping, pair: (ms_release_conn, strlen), support: 6, confidence: 66.67%
bug: ms_release_conn in apr_memcache_stats, pair: (ms_release_conn, strlen), support: 6, confidence: 66.67%
bug: ms_release_conn in apr_memcache_version, pair: (ms_release_conn, strlen), support: 6, confidence: 66.67%

The following is the source code of the function apr_memcache_version in srclib/apr-util/memcache/apr_memcache.c:

```
APU_DECLARE(apr_status_t)
apr_memcache_version(apr_memcache_server_t *ms,
                  apr_pool_t *p,
                  char **baton)
{
    apr_status_t rv;
    apr_memcache_conn_t *conn;
    apr_size_t written;
    struct iovec vec[2];

    rv = ms_find_conn(ms, &conn);

    if (rv != APR_SUCCESS) {
        return rv;
    }
    /* version\r\n */
    vec[0].iov_base = MC_VERSION;
    vec[0].iov_len  = MC_VERSION_LEN;

    vec[1].iov_base = MC_EOL;
    vec[1].iov_len  = MC_EOL_LEN;

    rv = apr_socket_sendv(conn->sock, vec, 2, &written);

    if (rv != APR_SUCCESS) {
        ms_bad_conn(ms, conn);
        return rv;
    }

    rv = get_server_line(conn);
    if (rv != APR_SUCCESS) {
        ms_bad_conn(ms, conn);
        return rv;
    }

    if (strncmp(MS_VERSION, conn->buffer, MS_VERSION_LEN) == 0) {
        *baton = apr_pstrmemdup(p, conn->buffer+MS_VERSION_LEN+1,
                                conn->blen - MS_VERSION_LEN - 2);
        rv = APR_SUCCESS;
    }
    else {
        rv = APR_EGENERAL;
    }

    ms_release_conn(ms, conn);

    return rv;
}
```

The purpose of strlen function seems to be served by MS_VERSION_LEN, which likely is the length of MS_VERSION, and therefore the missing strlen function is not a bug.

(c)

The Algorithm

The algorithm that was added for part c) is relatively straightforward. Our original implementation without interprocedural analysis was very fast so we had lots of space and timing to work with. After parsing the initial call graph from the provided file the function interProcedural() is run. Depending on your desired depth you can run the function several times to continually “flatten” linked functions.

The function operates by first taking a scope and looking at all the functions called in that scope. It takes all of these functions children (i.e. the functions they call, if there are any) and adds them as direct children to the original scope. It then removes the parent function from the scope which effectively “flattens” the function into all of its children. As it is rebuilding the new call graph it counts individual occurrences and pairs of functions.

Results

The first point of discussion is a small report on the original results from test2 vs the new results. Running the default (and correct) function results in the following 4 prospective bugs:

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

Running it again with inter-procedural analysis activated and with the same parameters (3 and 65) we get a prospective bug report of:

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

We see that there are 3 likely false positives when running the default algorithm on test 2. By using interprocedural analysis and just going 1 function layer deep we eliminate 75% of the bug reports and are left with just 1 of the 4. This is concrete evidence of the more advanced algorithm being more effective for test 2 specifically and likely many others.

Additional Info
It is interesting to note that with this implementation it is actually possible to remove some false positives and generate several more in return. Running on test3 (httpd), the default algorithm provides 205 likely bugs. Running with interprocedural analysis activated with a depth of 1 results in a whopping 719 likely bugs. This is a serious increase and strongly indicates that single depth interprocedural analysis does not always help. That being said, running it again with a depth of 2 results in 534, and 3 results in just 64 likely bugs. This indicates to me a strong correlation between complexity of the provided program and needed depth for interprocedural analysis to be of benefit.

Part (2)

(a)

1. **CID:** 10065
   Error: Missing Break In Switch
   Classification: Bug
   Reasoning: Because there is no break statement in case 3, the program will fall through and execute case 4. There are cases where this could throw an exception as the switch occurs based on string length and there are conditional checks in case 4 that look for charAt(4). This is not possible in a string of length 3 that falls through and thus erroneous. The developer should be sure to include a break statement at the end of

each case of the switch.

2. **CID:** 10066
   Error: CN: Bad implementation of cloneable idiom
   Classification: Intentional
   Reasoning: According to Java's documentation, implementing Cloneable indicates for a particular class that it is legal for the Object.clone() method to make field-for-field copies of that class. The implementing of Cloneable here appears to signal, for lack of a better term, the developer's intent to access this functionality. However, Object.clone() is a private method so it is convention to override this with a public method. The developers could implement their own clone method, but this could prove tedious, and probably unnecessary given what their intentions appear to be.

3. **CID:** 10067
   Error: Dm: Dubious method used
   Classification: False Positive
   Reasoning: This error is more specifically a reliance on default encoding of PrintWriter. This was likely flagged because of possible different encodings that may exist in different environments, however there appear to be no negative consequences that could flow from this reliance on a given environments default settings.

4. **CID:** 10068
   Error: Dm: Dubious method used
   Classification: Intentional
   Reasoning: This error more specifically complains about the use of nextDouble method in Random being used to generate a random integer when nextInt would be more efficient. This looks like an intentional choice given the cast to int made in the return value and comments preceding the function explaining it's intent and implementation.

5. **CID:** 10069
   Error: Eq: Problems with implementation of equals()
   Classification: False Positive
   Reasoning: This warning complains that the equals method, as implemented, only compares class names as opposed to class objects. However, from the comments preceding the function, this is the intended functionality and meets the developer's definition of equality.

6. **CID:** 10070
   Error: Eq: Problems with implementation of equals()
   Classification: False Positive
   Reasoning: This case, and the reasoning that follows from the identified warning, is identical to that which was previously identified in bullet 5 (CID 10069).

7. **CID:** 10071
   Error: ES: Checking String equality using == or !=
   Classification: Intentional

Reasoning: It is terrible practice to compare strings using == or !=, even if it doesn't always necessarily result in an error. The developer should instead use the equals() method (and the '!' logical operator where necessary) to ensure good coding practices are in use.

8. **CID:** 10072
   Error: ES: Checking String equality using == or !=
   Classification: Intentional
   Reasoning: The reasoning is identical to that of bullet 7 (CID 10071)

9. **CID:** 10073
   Error: ES: Checking String equality using == or !=
   Classification: Intentional
   Reasoning: The reasoning is identical to that of bullet 7 (CID 10071)

10. **CID:** 10074
    Error: ES: Checking String equality using == or !=
    Classification: Intentional
    Reasoning: The reasoning is identical to that of bullet 7 (CID 10071)

11. **CID:** 10075
    Error: IM: Questionable integer math
    Classification: Bug
    Reasoning: The given warning specifically warns of a possible integer overflow. This, unfortunately, could happen in line 649, within the while loop, if a large value is assigned to high and low has been previously modified to another suitably large value in lines 652-653. A fix could be to implement range checks using if-statements and pre-defined MAX and MIN values.

12. **CID:** 10076
    Error: NP: Null pointer dereference
    Classification: False Positive
    Reasoning: This warning complains of null being returned when the return type is Boolean, however this appears to be intentional based on the comments preceding the function, allowing for null to be the value passed to the function. Note that this is allowable when using Boolean as opposed to boolean. This allows the developer to handle the null case explicitly rather than leaving to chance the return value when null is passed.

13. **CID:** 10077
    Error: NP: Null pointer dereference
    Classification: False positive
    Reasoning: Similar to bullet 12 (CID 10076), the comments preceding the function explicitly allow for null to be returned even while the stated return type is Boolean, which is allowable (as opposed to when using boolean).

14. **CID:** 10078
    Error: NP: Null pointer dereference
    Classification: False positive
    Reasoning: This warning, and the reasoning that results, is identical to that which was previously identified in bullet 12 (CID 10076).

15. **CID:** 10079
    Error: NP: Null pointer dereference
    Classification: False positive
    Reasoning: This warning, and the reasoning that results, is identical to that which was previously identified in bullet 13 (CID 10077).

16. **CID:** 10080
    Error: NP: Null pointer dereference
    Classification: False positive
    Reasoning: As in the previous cases, Boolean is being used as opposed to boolean which allows for null values. The comments preceding the function specify null to be returned under specific conditions.

17. **CID:** 10081
    Error: NP: Null pointer dereference
    Classification: False positive
    Reasoning: This warning, and the reasoning that results, is identical to that which was previously identified in bullet 16 (CID 10081)

18. **CID:** 10082
    Error: REC: RuntimeException capture
    Classification: Intentional
    Reasoning: By Exception as opposed to specific exception types, the developer creates a scenario where RuntimeException may be caught, which is generally undesirable. Very little is done with the exception itself in this code, thus this is of little consequence, although better practice would be to try and catch specific types of exceptions (which would admittedly be much more tedious and time consuming during execution).

19. **CID:** 10083
    Error: Se: Incorrect definition of Serializable class
    Classification: Bug
    Reasoning: Serializing is the process of converting an object state into bytes. To declare an object transient would mean that a member variable is not serialized. Since mRules in line 137 is non-serializable and non-transient this could lead to erroneous states in execution. A simple fix would be to declare mRules to be transient.

20. **CID:** 10084
    Error: UrF: Unread field
    Classification: Intentional
    Reasoning: This warning complains that the key field is unread, however the preceding

comments note that key is to be used for storing each Entry in a table. While unused fields are bad practice, there does appear to be some motivation for leaving this as is.

(b)

During our code analysis with Coverity a total of 5 potential bugs were uncovered, as shown in the three .errors.xml files in the pii directory. This report will discuss three of these errors, two of which were identical.

The first of these errors (CID 10282) was of the type “Dm: Dubious method used (FB.DM_DEFAULT_ENCODING).” This error is given by Coverity when handling a string without declaring what text encoding is to be used. This causes the system to rely on the default encoding, which could theoretically lead to issues when reading from files that have different encodings. Using the classifications from 2(a), this error was labelled as intentional as we had developed the code in the same environment where we knew testing was to be done. This is fairly bad practice, but was deemed acceptable by the group as we had no concerns about how the text was being handled during our testing.

The second and third errors (CID 10281 and 10280) were both of the type “Result is not floating-point (UNINTENDED_INTEGER_DIVISION).” This happened in two lines of our code where integer division was performed and the quotient (also an integer) was assigned to a variable of type double, converting the result in the process. This was determined to be a bug because of the loss of precision the conversion caused to be observed. In both lines where the error was observed, the numerator and denominator were both cast as doubles to ensure precision. After testing the new code and a second pass with Coverity (results not included) the error had disappeared and our testing results were observed to be more accurate.