Erik Poole

```
 174|  4.94k|            //Should be impossible to reach...
 175|  4.94k|            default: {
 176|     0|                assert(false);
 177|     0|            }
```

The above code is in a portion of my main file and is the final possibility in a switch. I used this assertion to make sure that my randomized switch inputs never were out of bounds from my expectations. Since everything operated as intended, this code was never reached.

```
  57|      0|void StackDestroy(stk_stack *theStack, void DestFunc(void *a)) {
  58|      0|  stk_stack_node *x = theStack->top;
  59|      0|  stk_stack_node *y;
  60|      0|
  61|      0|  if (theStack) {
  62|      0|    while (x) {
  63|      0|      y = x->next;
  64|      0|      DestFunc(x->info);
  65|      0|      free(x);
  66|      0|      x = y;
  67|      0|    }
  68|      0|    free(theStack);
  69|      0|  }
  70|      0|}
```

The StackDestroy function was not actually even listed in the stack.h file and it made no sense to call it in our fuzzer. The only time in which a stack is produced by our red-black tree is when you call the RBEnumerate function on our working tree and cleaning up the pointers in our tree is already performed by RBTreeDestroy. Calling stackDestroy would cause errors use after free errors.

```
   3|   243k|stk_stack *StackJoin(stk_stack *stack1, stk_stack *stack2) {
   4|   243k|  if (!stack1->tail) {
   5|      0|    free(stack1);
   6|      0|    return (stack2);
   7|   243k|  } else {
```

This code could be called if a provided stack was empty but since our stacks are always provided by RBEnumerate on non-empty trees they are never empty when returned.

```
306|  56.2k|  } else {
307|      0|    y = x->parent;
308|      0|    while (x == y->right) { /* sentinel used instead of checking for nil */
309|      0|      x = y;
310|      0|      y = y->parent;
311|      0|    }
312|      0|    if (y == root)
313|      0|      return (nil);
314|      0|    return (y);
315|      0|  }
```

This code is from the TreeSuccessor function which is called during RBDelete. It does not seem to be necessary for RBDelete to function correctly.

```
276|   248k|#ifdef DEBUG_ASSERT
277|   248k|  Assert(!tree->nil->red, "nil not red in RBTreeInsert");
278|      0|  Assert(!tree->root->red, "root not red in RBTreeInsert");
279|      0|#endif
280|      0|}
```

This code is only utilized if the compiler flag DEBUG_ASSERT is provided. Since we didn't provide that flag it is never reached by the fuzzer;

```
53|       |/*  NullFunction does nothing it is included so that it can be passed */
54|       |/*  as a function to RBTreeCreate when no other suitable function has */
55|       |/*  been defined */
56|       |
57|     0|void NullFunction(void *junk) { ; }
```

This NullFunction was not used since we had a suitable function for RBTreeCreate provided in our test_red_black_tree.c file.

```
46|  4.05M|  } else {
47|      0|    printf("memory overflow: malloc failed in SafeMalloc.");
48|      0|    printf("  Exiting Program.\n");
49|      0|    exit(-1);
50|      0|    return (0);
51|      0|  }
```

No errors were ever seen in our Malloc Function (fortunately) and so this code was never reached by the fuzzer.

```
20|   618k|void Assert(int assertion, char *error) {
21|   618k|  if (!assertion) {
22|      0|    printf("Assertion Failed: %s\n", error);
23|      0|    exit(-1);
24|      0|  }
25|   618k|}
```

No assertions were failed in the original code's implementation and so this code was never reached.