

Formal Verification of Load Message Utility in Linux kernel code using BLAST Tool

D. Rajesh 07305033

rajeshd@cse.iitb.ac.in

Ashish Kumar 07305073

ashishsippy@cse.iitb.ac.in

B. Sharath Chander 07305078

sharath@cse.iitb.ac.in

December 3, 2007

Abstract

The aim of this project is to formally verify the piece of linux kernel *v2.6* code. Verification refers to proving correctness or incorrectness of programs with reference to its intended desired properties. A program is said to be correct if and only if it performs desired functions.

1 Description Of The Code

In `msgutil.c` we choose the code fragment `load_msg()` which loads the message from user area to kernel area. The function takes two arguments. One is the address of the message and other one is length of the message. Code works like this, it makes the user message into piece of messages which are fit into predefined memory segment and makes as a list. For this purpose it uses to structures `msg_msg` and `msg_msgseg`.

2 Properties chosen for verification

1. Whether the complete message has been copied from the user area or not.
2. At the end of coping, whether the last node is pointing to `NULL`.
3. Function returns an error either there is no memory or it cannot copy.

4. In each iteration the newly created segment must be pointed by the previous segment.

3 Problems Faced

1. While verifying some properties, we faced the problem where the system is in safe state but tool says unsafe.
2. When we want to verify some properties, we faced a problem where tool gives `NoNewPredicatesExceptions`.
3. With out including `assert.h` header file, when we are trying to track this property `if(msg!=NULL&&mseg!=NULL)`, tool says the system is in unsafe state. But, when we tried the same by including the `assert.h` header file, tool says system is in safe state.

For the first problem, the solution is use of the correct option.

4 Modifications

We introduced new pointer variable called *mseg* and *tseg* to keep track the currently created segment in the while loop which is used outside the scope of loop. Another two integer variables to hold the return value returned by the function call `copy_from_user()`. We also put some assertion to trace the properties mentioned above.

5 Conclusion

For the verification a tool called BLAST has been taken, as it handles the requirements of the code. Set of predicates were taken and verified, and it has been proved to be SAFE by BLAST. During verification we learned that to handle pointers we have to use the *talias* option in BLAST. We were able to verify the required properties using BLAST tool successfully, and had proved the chosen code is SAFE.