

CS615 COURSE PROJECT REPORT

Course Instructor: Prof. Supratik Chakraborty

NAME: ANIL KUMAR PUVATHUMKADAVIL

ROLL NO: 06305908

M.TECH2 CSE

PROJECT TITLE: Linux Kernel Module Verification

Kernel Module: linux-source-2.6.18/arch/i386/mm/fault.c

Tool Used: BLAST(Berkeley Lazy Abstraction Software Verification Tool)

Description of the code fragment:

In the kernel module fault.c, I choose to verify the function `__is_fetch()` which returns the prefetch bit(which is either set or reset). This function has a for loop which does the necessary check and decides whether to set or reset the prefetch bit, which is later returned by the `__is_prefetch` function. The variable `instr` gets the value returned by the function `get_segment_eip()`, which works as follows:

If CS is no longer a valid code segment, or if it is a kernel address when CS is not a kernel segment, then the returned value will be greater than EIP limit. If `instr > limit` then the for loop is exited with prefetch bit reset, else another check is performed which calls the function `__get_user()` which returns True or False depending on the values of opcode and `instr`, if the function returns False then the for loop is exited. These two checks are implemented through if statements, if the IF statements are satisfied then further evaluations are performed in the FOR loop. The main area of interest is:

1. The for loop uses the variable `scan_more` as a condition check, so if it is equal to 0 then further iterations are not possible.
2. The for loop uses a switch function which takes the `instr_hi` as the input parameter and uses this information to decide whether setting the prefetch bit is possible.
3. Each case condition of the Switch function will modify the `scan_more` variable.

The properties I decided to verify:

1. If the value of `scan_more` is 1 at a certain point(I defined it in the code where I have commented the point of reference) of the program during the present iteration then it has been 1 for all the previous iterations as well.
2. If the value of `scan_more` is 1 at a certain point of the program during the current iteration then, in all the previous as well as in this present iteration `instr_hi` was never equal to 0x00.
3. If the value of `scan_more` is 0 and value of prefetch is 1 at this point of the program during the present iteration then for all the previous iterations the value of `scan_more` was 1 and prefetch bit was never set in the previous iterations

Modifications to the original source code:

I have done certain modifications to the original source code, but during these modifications I have not modified the original variables:

1. I have used some external variables to keep track of the previous states of the for loop iterations.
2. Abstracted out two functions `get_segment_eip()` and `__get_user()`, this was necessary because

these functions make use of values pointed to by pointers and does manipulations on them and then returns the value. By making use of `__BLAST_NONDET` variable I have abstracted out or in other words made the domain bigger without losing any information.

3. I have used if statements and external variables to store the state of previous iterations of for loop.

4. Finally I used assert statements to check the error location reachability.

Difficulties faced by me in order to verify the function `__is_prefetch` using BLAST tool:

1. Due to requirement of storing the previous states of the for loop I had to modify the source code in order to verify the function.

2. Also I had to find a smarter method to use external variables to store the state and also use assert statements to verify the safety properties, I have observed that even though the for loop iterates for only 15 times, due to the usage of excessive assert statements the BLAST tool terminates with timeout error after running for a long period(roughly about 30 mins).

3. Since the two functions `get_segment_eip` and `__get_user` make use of values pointed to by pointers and then since the manipulate them using other variables which are themselves pointed to by pointers, I had to abstract out these functions since BLAST tool wont know the values of the variables which are pointed by a pointer. The variable `__BLAST_NONDET` came in handy to do this, thus I was able to make the domain bigger without losing information.

What I have learnt from this exercise:

1. I learnt to analyze the code based on the strengths of BLAST tool which is predicate abstraction, rather than concentrating on its several weaknesses(does not support analysis of multithreaded process, arrays, pointers).

2. Learnt to analyze a safety property which depends on the loop invariant, so that if the BLAST tool does not calculate the loop invariant then the verification fails.

3. Learnt to analyze the linux kernel source files with respect to what the various function calls does, the values returned by them, in general the control flow of a program through various functions calls.

Test run of the code:

I have managed to verify the stated properties using BLAST, the modified code fragment is heavy commented in order to explain the changes and their meaning.