USENIX ATC '12 Paper #108

#108 SymDrive: Testing Drivers Without Devices

Comment notification
If selected, you will receive email when updated comments are available for this paper.

Reject 281kB Tuesday 17 Jan 2012 2:39:57pm PST | 875df19a4a0a164f35515858bf35860915890e01

You are an **author** of this paper.

+ **Abstract**

Device driver development and testing is a complex and error-prone undertaking. For example, a developer may not have access to the device, and a single driver may support [more]

+ **Authors**

M. Renzelmann, A. Kadav, M. Swift [details]

+ **Topics and Options**

	Eva	Con	ShoPap
Review #108A	C	Y	2
Review #108B	C	Y	2

Edit paper | **Add response**

Reviews in plain text

Review #108A

Modified Sunday 19 Feb 2012 5:07:22am PST Plain text

Summary

The paper presents SymDrive, a framework being destined for testing Linux drivers without their devices. Based on symbolic execution and by taking a position on symbolic hardware, SymDrive provides symbolic devices for the driver under test on the one hand and emulates the other devices in the system an the other hand. The evaluation investigated 21 Linux drivers of different (11) classes and discovered 37 bugs (15 via a kernel warning or crash, 22 through checkers that verify and validate driver behaviour).

Strength

The paper addresses an important typic (that perfectly fits to ATC) and provides an applicable solution to system software developers. The approach has been evaluated using a non-trivial example (Linux).

Weakness

Overall structure and organization of the paper could be improved. People non-familiar with symbolic execution are handicaped in losing the plot here and there. The interaction with a virtual device does not become clear, also the consequences for automatically detecting true errors in drivers for these (real) devices.

Evaluation (?)

C. Weak paper, but I will not fight acceptance.

Confidence (?)

Y. I am knowledgeable in the area, though not an expert.

Detailed Comments

In general, it was a pleasure reading the paper. However, I had difficulties with understanding the big picture in terms of mapping real devices, with all its complex behaviour, to symbolic ones. You should have given some more details on this issue.

To give an example: I am wondering how a realistic behaviour of a symbolic device was recreated. I have something like a behaviour model in mind, but haven't found anything about that in the text. So I didn't really understand how a physical device was replaced by a symbolic counterpart while being able to maintain a specific device behaviour.

A further example, closely related to the former: I would expect that, by basing on such a model or the like, one will be able to also inject erratic behaviour. I am missing any discussion or (background) information on this aspect, including the testing coverage regarding erratic device behaviour.

My understanding of S2E (as used by SymDrive) is that if one reads from a virtual device "symbolic results" (values bounded by certain pre-conditions) will be returned. If one writes to a virtual device, however, these write operations are completely dismissed. Maybe, this explains that your analysis did not reach a 100% coverage. But most importantly, I am in doubt that based on such an approach true errors in device drivers can be revealed. Your analysis (as explained in the paper) ignores the interaction with the hardware, which is the essential aspect of driver programming. Of course, one will be able to check with SymDrive that a required order of API calls are followed. But for this it looks to me as if the effort one needs to take into account in using SymDrive is a bit high.

You reported a code coverage of 60% to 90% (cf. table 4), depending on the driver. Did you measure coverage before or after conditional compilation?

You also reported the detection of 37 bugs. Did you proposed driver patches to the Linux community? If you did, how many of these have been merged, accepted, or acknowledged?

Short paper (?)

2. Can't tell

Review #108B

Modified Friday 24 Feb 2012 12:36:35pm PST

Plain text

SUMMARY

Symdrive is a symbolic testing framework for device drivers based on s2e; it reduces testing effort when compared with previous solutions. It finds bugs.

STRENGTH

Driver bugs are pernicious and the need for hardware to test them is a serious problem. Real kernels and drivers are complex and dealing with them takes at lot patience and attention to detail. The paper is pretty well written and shows some good results.

WEAKNESS

No single contribution, but rather lots of little increments, mainly heuristical. Paper is glib about a few importent details -- more below.

EVALUATION (?)

C. Weak paper, but I will not fight acceptance.

CONFIDENCE (?)

Y. I am knowledgeable in the area, though not an expert.

DETAILED COMMENTS

My major problem with this paper is a little complicated to describe as it comes out of the interaction of several factors. First, if you have no real device model (that is, the HW can do anything at any time) then you will get false positives. This is a simple fact. Real drivers (I've written a bunch of fairly simple ones) count on the hardware following its side of the protocol. Example: when I set a flag in the HW telling the device to not give me any more interrupts, it better not give me any more interrupts. When I set a flag in the HW putting the device in transmit mode, I better not keep receiving bytes. When I read a timer or counter twice rapidly, my driver may malfunction is there is a huge change in the value and this may not be a bug.

Now: if you have not encountered any false positives (and this is the explicit claim in 4.5) then we had better understand why. One possibility is your hardware devices don't work the way the ones I've interacted with do. This seems improbable. Another possibility is that you have somehow implicitly encoded a way to suppress bad hardware behaviors inside the mountain of heuristical methods that comprise Symdrive. If this is the case, we'd sure like to understand better what is going on. The third possibility is that you have found false positives, but you don't know it. This is the possibility that most worries me, and it strikes me as quite likely given that (1) 6.2 is quite short and lacks detailed analysis and (2) 6.2 fails to mention the fact that you validated your bugs by reporting them to the driver maintainers. You have to do this, for several reasons. First, if you don't report the bugs you found, your research isn't actually making anyone's life better. Second, it is critical to verify that these bugs are real.

The paper implies that symbolic execution finds bugs not found by static analysis. That may well be true, but you do not want to underestimate the power of for example MSR's driver analysis tools. My guess is that their tools deal with stateful drivers and pointers perfectly well. Realistically, you would need to hold some sort of a driver bug bake-off with the static analysis people where you both analyze the same drivers, then analyze the resulting alarms with a fine-toothed comb. Until you actually do this, claims that you can find bugs that they cannot find have no substance.

Overall this paper is well-written and well-reasoned, but not everywhere.

On page 2 we read "Symdrive ... has no need for an operating system model." But this is wrong. The checkers described in section 5 are exactly an operating system model. Please don't play semantic games, just compare the size of your model with the size of other models.

I found the factoring of information between the "design" and "implementation" sections to be a bit arbitrary. There are plenty of things I would call design issues in the implementation part. You may be able to do a better job if you create a single section that just describes symdrive. If not, perhaps try to create a stronger rationale for structuring the two sections. As it is, I felt that you just went over it twice, with the second time giving more details than the first.

Wouldn't a simpler, better name for "favor success scheduling" be "DFS"?

"Symbolic execution is much slower than normal execution but still faster than buying the device." -- a bit too glib here

SUMMARY: I think this work has the potential to be very strong, but both the writeup and the underlying work require another iteration or two before it goes to press.

SHORT PAPER (?)

2. Can't tell

Response

The authors' response is intended to address reviewer concerns and correct misunderstandings. The response should be addressed to the program committee, who will consider it when making their decision. Don't try to augment the paper's content or form—the conference deadline has passed. Please keep the response short and to the point.

☑ This response should be sent to the reviewers.

Save