Eurosys2012 | **Paper #56**

mjr@cs.wisc.edu Profile | Help | Sign out

Main Edit

Your submissions (All) Search

#56 SymDrive: Testing Drivers without Devices

☑ **COMMENT NOTIFICATION**
If selected, you will receive email when updated comments are available for this paper.

Submitted 263kB Thursday 20 Oct 2011 9:06:27pm CEST | 8eac2e8ef295bde169a9d5d94637a2755a898979

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

+ ABSTRACT

Device driver development and testing is a complex and error-prone undertaking. For example, dozens of driver patches are committed to the Linux kernel include the comment [more]

+ AUTHORS

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

+ TOPICS AND OPTIONS

	OveMer	RevExp
Review #56A	4	2
Review #56B	3	2
Review #56C	2	2
Review #56D	3	1
Review #56E	4	2

Edit paper | **Add response**

Reviews in plain text

Review #56A

Modified Saturday 5 Nov 2011 12:20:36am CET   Plain text

PAPER SUMMARY

SymDrive allows extensive testing of kernel device drivers without
actually needing the device hardware. It does this by running the
kernel in a VM, and using symbolic execution to test as many code
paths as possible. SymDrive has techniques to prune the search space to improve efficiency (in terms of coverage of specific functions).

PAPER STRENGTHS

The paper identifies an important problem, and I think the use of symbolic execution to avoid the need to test drivers on real devices is a clever and promising one. The approach seems to work on real systems, and it doesn't seem to take very long to test a driver (once someone has written the required checkers and annotations).

PAPER WEAKNESSES

There's a lot of detail here, but the ideas behind SymDrive are sometimes hard to follow.

There are some important limitations (e.g., no support for multicore CPUs).

Overall merit (?)
4. Accept

Reviewer expertise (?)
2. Some familiarity

Comments for author

While the paper is generally well-written, in several places I really wished for a few concrete examples, since the text was rather abstract. For example, in section 2.1.2 -- especially since some of this is apparently non-trivial. In general, I think I would prefer less detail about the "how" and more insight into the ideas behind SymDrive.

It also wasn't clear to me if SymDrive has any way to check that the values sent from the driver to the device are correct. For example, if a driver patch causes it to set the wrong bit in a port I/O instruction, would this be detected? Only via the execution-differencing feature?

Review #56B

Modified Tuesday 15 Nov 2011 6:00:00pm CET

Plain text

Paper summary

The paper describes SymDrive -- a system that uses symbolic execution
to find bugs in device drivers without the actual device.

The system uses source code instrumentation and various heuristics to
increase the scalability of symbolic execution for the purpose of testing device drivers. In addition, it includes a framework to test
the driver which relies on checkers written by the programmer and
a mechanism to identify differences in the I/O path of different versions of a device driver. The paper reports on the authors' experience in applying the tool to various device drivers
which resulted in finding 23 bugs.

Paper strengths

A solid solution to an important problem.

Good results, showing the practicality of the approach.

Paper weaknesses

Unclear what the main / most novel insight behind this work is.

The tester must write these checker functions, and the more

elaborate
they are (i.e., close to a full spec) the more effective the
technique is.

OVERALL MERIT (?)

3. Weak accept

REVIEWER EXPERTISE (?)

2. Some familiarity

COMMENTS FOR AUTHOR

The interesting contribution of this paper is that it presents a
series of techniques that effectively circumvent the limitations
of
existing symbolic execution-based testing tools, when applied to
the
scenario of testing device drivers.

The proposed system addresses an important problem - device
driver
reliability - and does a good job at it. The approach described in
the
paper seems to be effective at finding device driver bugs and
reasonably practical for the programmer.

In particular, the solution had to overcome a few interesting
challenges, namely allowing for testing device drivers in a
reasonably
fast manner while avoiding the need to have the device itself, to
reimplement the device driver, or to extensively annotate the
device
driver source code.

There are also two main negative points worth highlighting.

The first is that while the system seemed to effectively address
a
series of engineering challenges behind getting the proposed
solution
to work in a practical way, it is unclear what the main insight or
the
general or principled ideas and solutions that can be extracted
from
the work. Highlighting such points would make for a stronger
submission.

Second, the proposed testing system tests the drivers for safety
violations (e.g., hangs, crashes, API violations) but because it
does
not have access to the specification of the device it cannot
ensure
that the driver actually performs the expected good actions - for
example, regardless of the return value given back to the
kernel, the
network driver should actually send packets through the
network when

instructed to do so. This essentially boils down to a tradeoff between
how complex and difficult to write the checker functions are, and how
complete the specification being tested is.

Detailed points:

It's not completely clear why avoiding the need to code the
consistency plugin manually is so important. Can you elaborate on what
this entails?

Given the various heuristics used by this approach there seem to be
some types of bugs and bugs in certain locations that may be harder or
impossible to detect by this approach. Perhaps the paper could have a
discussion section that attempts to give readers a better idea of
which are the bugs that can be found by this approach and which are
the bugs unlikely to be found.

In 2.1.1, I got slightly confused when the paper says that because S2E
is limited to a uniprocessor SymDrive is not able to detect data
races. For general applications, given that there can various possible
thread interleavings, data races can occur even in uniprocessor
machines. In other words, by injecting an interrupt, for instance, a
data race can be easily triggered. This point should be clarified.

As a suggestion, in Table 4, the mechanism that detected the various
bugs could be listed (illegal operations, checker or execution
differencing).

Also, it would be nice if the evaluation could try to highlight
how effective the paper's proposed optimizations are. How does the
system perform with and without each of these?

Review #56C

Modified Friday 18 Nov 2011 10:45:03am CET Plain text

PAPER SUMMARY

uses a symbolic execution based tool to check driver code.

PAPER STRENGTHS

driver code is pretty broken. it'd be good to find errors in it.

Paper weaknesses

few bugs, few drivers. it's not clear what is really novel here.

Overall merit (?)	**Reviewer expertise** (?)
2. Weak reject	**2.** Some familiarity

Comments for author

the paper takes an existing tool and applies a SLAM like approach to writing driver checkers.

useful to do, but there's not a lot of novelty. i don't know what i learned, other than it appears really hard to push code through their system.

there has already been other papers on checking drivers w/ symbolic execution! you need to talk about how you differ.

it's good to see some coverage results. but: you only have 13 drivers, the fact that you didn't do them all makes me suspicious. and this weird notion of only counting coverage in touched functions is sketchy --- if you screw up and don't explore a path that calls foo() you don't in fact count foo's statements against your coverage. i can't tell how good this coverage actually is. there is no eval of the search strategies they have compared to what's in the tools they use.

i would like to see more bugs and more drivers.

the writing makes me wonder how much they understand of the techniques they are using. e.g., "no false negatives" --- you do realize the underlying system you use has many false negatives? such as it only checks code on paths it can execute. people have used symbolic execution to check drivers already (including efforts they do not cite). they have low driver counts and low bug counts.

section 2 talks about symdrive doing a lot of things that it, in fact, is not doing, but are entirely the acts of the code it is built on.

the comparision to static analysis needs work, since their statements undermine confidence in the authors' grasp of the field. there are some tools that don't handle procedure calls, but there are plenty that do. plus, if you look at the raw bug counts of static tools they dwarf the tiny number in this paper. from what i can tell you were checking properties that were already done by static (e.g., functions happen in certain orders or certain contexts, you check X before Y, etc). it's hard to tell, but it's far from clear that static wouldn't find the bugs this paper does. it would be able to handle way more than 13 drivers with much less effort than a couple of hours per driver. there are plenty of problems with static; focus on those.

Review #56D

Modified Thursday 15 Dec 2011 1:25:16am CET Plain text

PAPER SUMMARY

The paper proposes using a symbolic execution engine called SymDrive to exercise driver code without having access to the actual devices for said device driver. The authors extend the existing S2E tool to allow speedily testing the drivers. They provide an augmented execution engine, a tool for transforming code in order to speed up testing as well as a test framework that allows for easy invoking checkers to verify driver code. The authors uncover 23 bugs (or potential bugs) not previously known. They are able to write support for testing a new driver in under two hours.

PAPER STRENGTHS

- The paper is well written and easy to follow; it has a good structure.
- The tool has uncovered a number of bugs in existing drivers.

PAPER WEAKNESSES

Not really any that I can think of.

OVERALL MERIT (?)	REVIEWER EXPERTISE (?)
3. Weak accept	1. No familiarity

COMMENTS FOR AUTHOR

I do not have much to add to the paper contents. It was nice to read, well written and easy to follow. The results are promising.

- The paper is a bit lacking in figures and some parts of the text might be easier to understand if they would be accompanied by a good figure illustrating what is happening.
- It is unclear to me what the consequence is of reducing the paths that are explored. What are the odds you are missing out on a bug?

Other remarks

- The text at the start of Section 5.5 and the header in Table 5 do not use the same terminology. Please correct this. It is not hard to follow to which columns the text refers, yet having the same wording in place would be much nicer.

Review #56E

Modified Friday 23 Dec 2011 2:51:39am CET Plain text

PAPER SUMMARY

This paper proposes debugging device drivers through symbolic execution. Symbolic execution enables SymDrive to emulate all possible

values that a (physically nonexistent but under emulation) device can
possibly return. The paper evaluates this strategy with 13 Linux drivers
and identifies 23 bugs.

PAPER STRENGTHS

Neat application of S2E and symbolic execution to testing drivers.
Real system, with real results.

PAPER WEAKNESSES

Not sure how effective the system is.

OVERALL MERIT (?)

4. Accept

REVIEWER EXPERTISE (?)

2. Some familiarity

COMMENTS FOR AUTHOR

I really enjoyed this paper. Symbolic execution is a powerful technique, though it can entail such high overheads to become impractical if used naively. This paper describes a practical and principled approach to testing drivers without their associated devices.

The biggest weakness of any bug finding paper is that the universe of
bugs seems to be infinite. So I'm not sure how to contextualize "23
bugs found." Is this a high number? Is it a low number? What would
other techniques find?

One thing that would help provide better context would be to have a
panel of (properly motivated!) programmers examine a given driver, and
compare their results against the tool.

Another thing I worry about in bug-finding papers is that they tend to
describe only those efforts that yielded results, as opposed to the
total effort. Did you develop checkers that failed to uncover bugs?

Overall, however, I found the paper a joy to read -- it's of interest
to the EuroSys community as well as SIGOPS in general and it describes
a novel and practical application of symbolic execution.

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