COMP9242 Final: Paper 2
SIDE: Isolated and Efficient Execution of Unmodified Device Drivers

November 12, 2013

1 Summary

The paper describes SIDE (Streamlined Isolated Driver Execution), a method of safely executing existing device drivers in an environment that is isolated from the rest of the kernel and is fault-tolerant. It achieves this through the conventional virtual memory protection mechanisms and a custom wrapper around driver-kernel interactions. The driver is able to run without modifying its original code, making legacy and closed-source drivers compatible with SIDE.

SIDE is designed to sandbox existing x86 kernel-mode device drivers and protect the kernel from misbehaving driver code and hardware faults. It achieves this by hoisting the driver code and data into a user-level address space and by executing the driver functions in user mode. Access to kernel data structures and privileged instructions is mediated through an additional SIDE library which is loaded into the same address space. The driver's import table is used to point the driver to appropriate library replacements.

When the driver accesses functions or data which are in the kernel but not passed through the SIDE library (due to inlined functions or otherwise), the driver cops a protection fault and a special handler resolves this request. Similar faults occur when the driver tries to execute privileged instructions (which may not have been monkey-patched to use SIDE stubs). Thus all kernel data and code is protected. Each invocation is analysed to check for invalid arguments and to keep track of resources acquired by the kernel (e.g. locks and buffers).

The page directory is shared between the kernel and the driver, and so the kernel may access the driver's data without needing to switch to another address space. The only exception is the driver code — it is unmapped from the kernel page directory for SIDE_base implementation, as to intercept the calls to driver functions.

The authors have done their evaluation on a gigabit network interface with an unmodified Intel driver. They have added further optimisations to produce SIDE_optimised, which has performance comparable to in-kernel device drivers. On the downside, it requires modifications in the kernel to proxy all driver calls through SIDE.

2 Strengths and successes

The authors have successfully sandboxed an existing network driver with a minimal latency or throughput penalty. Their evaluation results compare SIDE with the same driver running in-kernel (i.e. as a conventional kernel module) and results indicate a 1% slow-down. This is a fairly negligible penalty with the added benefit of fault detection and complete driver recovery.

Some thought has been put into making optimisations on top of the SIDE_base implementation, reducing overheads that prevail in their design. The authors went as far as virtualising some kernel counters and shared variables which are later synchronised with the kernel data without an extra user-kernel crossing.

Some analysis has been performed on the added overheads, as presented in Table V. This gives directions for further improvements that can be made in the future.

3 Weaknesses and criticisms

First of all, it seems that SIDE is unable to have more than one sandboxed driver securely running at any time. Due to the user-kernel page directory sharing, their approach seems to not protect two separate driver instances from accessing each other's memory mappings (they run in the same address space, both in ring 3). While this does not violate the kernel's stability directly, it is possible for a malicious or compromised driver to corrupt and crash other SIDE-boxed drivers. This can be avoided with a separate page directory per driver, but it works against the authors' idea of a shared address space.

The CPU overhead for small packets is too high, with the given lower bound being 11%. This is by SIDE's design, but I do not think it is an acceptable value; 45% overhead on small packets is too much.

The above point is reinforced when the unmodified driver attempts many calls into the kernel functions or pokes into multiple data structures. The authors lead me to believe that this will either require thorough modifications on the kernel's side, or to trigger a fault with every such access. This is incredibly slow.

No benchmarking has been done for TCP connections; further, no macrobenchmarks have been performed. Their testing is limited to just 1 microbenchmark that they have written specifically for this. It would be useful to provide results on common benchmarks — theirs could be designed in a way to avoid all relative penalties while providing substandard performance.

To complement the above, they do not provide any absolute times or throughputs for their benchmarks. It is unclear whether their benchmarks

are even capable of saturating a 1 Gbps link, that is, using the driver to its maximal capacity.

The paper is plagued with awkward wording and grammatical errors, making it hard to understand what the authors meant in some places. In particular, they point out a problem with running TCP/UDP connections in last paragraph of section IV-C. It is unclear what problem they are trying to address and what its effect is.

Most importantly, SIDE does not seem capable of always detecting malicious actions on the driver's behalf. It proxies calls to the kernel and checks their arguments, but this is not a secure way of ensuring kernel consistency or security. From their description, I believe it is possible for a driver to kill processes or grant root permissions to a process. This is something that should be addressed for proper driver isolation.

Similarly to this, it seems that SIDE does not always succeed in keeping track of allocated resources, as pointed out in section IV-D. The authors mention inline functions causing problems, and it is unclear whether or not I could acquire a lock without going through a SIDE proxy.

In addition to the last problem, SIDE copies kernel structures into the driver address space for it to modify, and then copies the changes back into the kernel. It may be possible for both the kernel and the driver to modify these structures before they are copied back, resulting in races and/or data corruption. It is unclear how the authors are managing this.

After reading the whole paper, it seems that SIDE is simply an optimisation of a prior work, Nooks, as cited by the paper in section II-B. They have not convinced me that their approach is at all different. As a disclaimer, I am not familiar with Nooks and it may be that the authors' description just happens to ignore significant differences between them.

4 Conclusion

SIDE seems to re-use prior ideas that have been published before, e.g. Nooks, to build an isolation mechanism that seems to work under the authors' special test conditions. It is unclear if the isolation is sufficiently strong, if it allows multiple such isolated drivers to co-exist with little overhead, and if the performance overhead is as small as they make it out to be.

I am not convinced that this paper is a strong one, and would like to see more performance evaluation and isolation tests performed, at the very least, and the above uncertainties addressed.