Side-channels in Runtime Systems:
An Idea Overview

Tegan Brennan Miroslav Gavrilov

April 14, 2017

1 Side-channel Overview

A practically useful system always presents a complicated landscape, in an information-theoretical view: to be deemed "useful", computations have certain characteristic behaviours they should follow (e.g. be of a certain complexity class, execute at a certain speed, etc.), which more or less exerts certain properties of the physical implementation of the system in question. By not focusing on the result of the computation, but rather on the side-effects manifest in this physical implementation, one can gain knowledge of some implementational details, and thus the structure of the computation itself. This is referred to as using a side-channel to learn things. Similarly, we say that looking at the computation directly is a main-channel.

Side-channel exploration and exploitation have mostly been a part of the realm of cryptography, but with the coming of Web 2.0 and mobile technologies, as well as the exponential increase in processing power compared to power output or computation time required, the possibility of finding a side-channel grew significantly. One of the foci of side-channel analyses is preventing side-channel attacks, in which an attacker is using one or potentially more side-channels to discover some private data or gain insight into a hidden process. That being as important as it is, side-stepping security, one (friendly individual) can learn a lot about a system by observing side-channels. For example, timing side-channels present a view into different execution times of a program, most frequently dependent only on the input passed into it. If the program isn't observably polynomial with the size of the inputs (which would be a trivial side-channel to catch), but we do see a pattern occuring with differing inputs, we can conclude that there exists a branch in the program structure which uses the input as a choice point: a certain class of inputs activates one path in that branch, leading to some execution time t_1, while the other leads to an observably different execution time t_2. This, for example, reflects a very common coding practice that is used widely in imperative programming, namely: early returns. In many environemnts where early returns are predominant, programmers value optimizing for speed, but such optimizations could never cover the space of all programs for which they are implemented, for all inputs[1], making it possible for analyses to get knowledge about the system under test without breaking the seal on the black-box.

This phenomenon builds a kind of philosophical construct similar to the Heisenberg uncertainty principle wherein the limitations of our physical reality are stopping us from being both optimized (take less time in most cases) and, for example, completely computationally private (as observed in [1]). The case in which the least side-channel activity is present is the case in which the lengths of all the possible routes of execution of the main-channel are of similar size, and thus observation becomes noisy in the presence of other physical factors. This, unsurprisingly, is the case in which there is no optimization made and all the lengths are similar to the longest one.

1.1 Analysing runtime systems

Runtime systems are complex architectures built for speed, and as such are a logical source of many observable side-channels. Given that they operate over different but generally equally-powerful languages, our main field of exploration would cover the following questions:

- Given two programs written in two languages, recognized by some dependable method of semantic clone detection[2] as semantic clones, how many of the side-channels arising from their execution are recognizable as cloned channels?
- Given two runtimes running the same language, and a program in that language, how many of the side-channels in that program's execution are cloned channels across runtimes?
- Given one runtime with several languages executing in it, and a program's semantic clones in those languages, how many of the observable side-channels are also cloned channels across that runtime?
- Given a runtime's different levels of optimization and JIT/GC settings, for an observable side-channel, how does tweaking those setting change its strength?

The answers to these questions could give an idea of how certain implementation details decide on how suitable a runtime or language is for tasks that need a level of discretion in lieu of a potential side-channel, as well as how they could decide some further optimizations.

[1] If an optimization could ever cover the space of all programs for which it is implemented, for all inputs, that means that it would collapse the whole complexity class in which it is by one and lead to the side-channel itself not being recognizable.

References

[1] Jason Oberg, Sarah Meiklejohn, Timothy Sherwood, and Ryan Kastner. *Leveraging Gate-Level Properties to Identify Hardware Timing Channels* http://ieeexplore.ieee.org/document/6879637/?reload=true

[2] Mark Gabel, Lingxiao Jiang, and Zhendong Su. *Scalable detection of semantic clones* http://dl.acm.org/citation.cfm?id=1368132