Preventing timing attacks through the type

Erik and Mads

Timing attacks is something programmers traditionally give very little attention. It is a type of attack that initially seems inconsieveable. Even after one has concinces herself that timing attacks are real threads, the prevention mechanisms are hard. We propose a technique to prevent timing attacks that takes advantage of the type system. The approach forces the programmer to prove the absense of timing attacks.

Introduction

The Language

Core Types

As the most important core types we have

- **Unit:**
- **Sum:**
- **Product:**
- **List:**

Perifical Types

Many target architectures utilize special operations to perform fast computations for specific types. It is therefore very common to have special types built into a source language, to allow the programmer to utilize these coonstructs.

To make it easier for us to program applications we have built-in support for following types:

- **Int:** These fixed-width integers. We built in literal constructor and the usualy operations. We assume all operations take constant equal time.
- **Booleans:** Booleans along with typical operations. The assumption is, again, that the operations take constant, equal time.

Applications

It is hard to directly write in expressed language. Even though we rely on much inference, the type errors and other things are.

User Authentication

Security

In this section we will go over the security

Leackage

Leackage is expressed as a function over the input list-size.

Target Architecture

As what all we do is static, ew have no guarantee that it is actually carried out on the target architecture. For everything to work in reality, we need the interpreter or the compiler to take the type annotations into account.

With compilation we frst need to varify that the target platform satisfies our

Perspectives

Weighting Operation Time

We have assumed that all aritmetical operantions, `plus`, `minus`, `times`, and `divide`, take the same time, namely one time step. Furthermore we have also assumed that `or` and `and` does so.

It might be unfair to assume that operations on different types take the same time to execute. But depending on the target architecture the proportions reduce to good approximations.

Single Cycle Instructions:

Complex Architectures:

Conclusion