Announcements (0 minutes)
Learning Objectives for Today (1 minute)
The Langsec Concept: Parables and Seeds (15 minutes)
The Gostak (20 minutes)
Readings (1 minute)
Part of what I’m trying to do here is articulate a clear “tutorial trail” for learning about Langsec — this is a work-in-progress. Please pardon our appearance.
→ Why do things break?
→ Introduction to Langsec and Weird Machines
→ Langsec principle: a thing is not what it is named; rather a thing is what can be done to it.
Sure, C is a “dangerous” language, and some programmers are worse than others.

What reasons do you see?
Why

Perhaps because they are attempting to do something fundamentally impossible.

The problem is that we have not spent sufficient effort in defining the boundary between “useful computation with limited computational power” (as a way of minimizing the vuln. surface) and “arbitrarily powerful computation subject to exploit.”
The Tale of the Grad Student and the Professor

Professor has deep thought.

Professor express deep thought to graduate student at great length.

Student hears some parts of great speech.

Student thinks about idea.

Professor asks question to gauge understanding.

Student repeats some version of words back to professor.

Professor does not recognize the idea.
There once was a somewhat halfway enlightened Developer.

“Let us achieve security,” said the Manager to the Developer.

“Use a firewall to filter connections,” suggested the Intern.

“Write a module that accepts input,” intoned the Architect. “Place it in front of the application.”

“OK, but can I just use HTTP-Auth-Basic?” asked the Developer.

“It is good.” beamed the Boss.

The Tester smirked.
A little while later, the Developer knocked on the Architect’s door.

“How do I implement this input filter component?” he begged.

“Squeeze the input. Recognize bad things. Remove the filth.” barked the Architect, sitting bolt upright.

“Oh, like a regular expression that throws an IllegalArgumentException. Thanks!” (The Developer was a somewhat halfway enlightened Developer and knew just enough about regular expressions to be dangerous.)

“Composition kills,” chuckled the Tester to himself.
The actual semantics of a computational construct are found in the operators (and combinations thereof) that operate on that construct rather than the *a priori* name or label for that construct.

It is the disconnect between these “operational” semantics and the “label” semantics that often serve as the source of security flaws.

Example: an instance of a type is not the type unless you only operate on it in accordance with the type rules. Is that an `int` or a jump table?

Why? Even though this is a later piece of work in the langsec area, it is a compact and succinct review of the exploit engineering workflow: we need to eliminate your preconceived notions of that craft before showing you the fundamental ideas of langsec, particularly langsec phrased as a “theory of insecurity.”