Course Projects

CS 6335: Language-based Security

Binary Verification using Picinæ

- Project Goal: Prove safety and correctness of a binary subroutine
- Teams: 3-5 people (self-formed)
- Info due to me by Monday: One email per team listing…
  - your team members (cc all team members)
  - the subroutine you want to target (see new folder on eLearning)
    - try not to pick the same targets – I'll recommend different targets if overlaps occur
- Short in-class presentations of work-in-progress (12/4 and 12/6)
- Final deliverables due 12/15:
  - Your proof.v file(s)
  - Written report (5-10 page pdf)

First Steps

- Download & Install Picinæ
  - Make sure the sample proofs all work on your machine
- Download your chosen subroutine.v target and make sure it works
  - Compile it, make sure the “welltyped” proof succeeds
- Reverse-engineer the code
  - Draw out the control-flow graph
  - Figure out how it works before you start proving anything!
- Define a correctness specification in Coq, consisting of:
  - function that identifies exit-points (usually all the returns)
  - property of the CPU state that defines “correctness” of your target subroutine
- Identify invariant points and define your invariants
  - Recommendation: Put an invariant at:
    - the start address,
    - the start of each loop,
    - any addresses where multiple flows converge (“meet-points”)
    - all exit points (always!)

Main Workload

- Do some “easy” preservation proofs in the style of the strcmp sample.
- Try to prove full functional correctness (like the final proof in the sample).
- Warnings:
  - You will almost certainly discover that some of your invariants are incorrect (untrue or inadequate or both), but this only becomes apparent after you’re deeply into the proof. So if you don’t get deeply into the proof until December, you won’t succeed in this project!
  - Managing your time and team is very important. All members need to be productive each week throughout November in order to be successful.
  - Learning curve is steep – this project moves you from “toy” classroom exercises to a real-world verification task of medium-to-high complexity.

Class Presentations

- About 15 min. per team
- Use slides (classroom machine has a web browser)
- Select one spokesperson to present or take turns
- Content:
  - Show us your target subroutine
  - Show your proposed correctness specification
  - Explain the code's control-flow structure and how it works
  - Summarize how far along you are and where you're struggling
  - Take questions/recommendations from audience (we can help you!)
- Each team receives one presentation grade as part of your project grade, no matter who presents.

Final Reports

- Submit as pdf (5-10 pages)
- Content:
  - Summarize your target subroutine informally.
  - Explain how your specification captures “correctness” for your target.
  - Show the control-flow graph and briefly describe the functionality of each part of the code.
    - Note any particularly hard or weird implementation anomalies!
  - Summarize what you accomplished and what you didn’t.
    - If you invented any cool new things (e.g., new Coq tactics, new general-purpose theorems you proved), highlight those.
  - Document your team organization (who did what parts)
    - This only factors into your grade if you had a team dysfunction (e.g., some members hardly contributed anything until the final days of the project, forcing everyone else to do most of the work).
    - If “everyone did everything” then at least give me a kind of timeline of major milestones accomplished.

Troubleshooting

- Nobody has ever machine-verified any of these codes (as far as I know).
- It's entirely possible that some amazing/weird/impossible code feature will throw a wrench into your project. (Welcome to the real world!)
- If so, it's up to you to discover such things *early,* and we can work through them together.
  - I will help you succeed if you give your team (and me) enough time!