MARYLAND
CYBERSECURITY CENTER

Build it, Break it, Fix it

A new security contest

Prof. Michael Hicks

co-conceived with Andrew Ruef and co-developed with Jan Plane, Atif Memon, and David Levin

University of Maryland, College Park USA

Funding provided by National Science Foundation
WHERE DISCOVERIES BEGIN

Contests are cool

DEFCON CTF

Collegiate Cyber defense challenge (CCDC)

Pwn to Own

- Rewards those who can reverse engineer vulnerabilities in real or custom systems

- But what about the opposite? I.e., reward those who can build more secure systems
  – Fallacy: if you know what/how to find the vulnerabilities you can build systems without them

Build it, Break it, Fix it

Must satisfy basic correctness and performance requirements

Round 1:
Build-it team
Contestants build software to specification

72 hours

Round 2:
Break-it team
Contestants report bugs in submissions

Bug reports are (failing) executable test cases, including exploits

72 hours

Doing so may wipe out many bug reports in one go: all count as the same bug

Round 3:
Build-it team
Fixes bugs found by break-it teams

72 hours

Then: Judges tally final results

Scoring

- Build-it team
  - Gains points for good performance
  - Loses points for (unique) bugs found by breakers
- Break-it team
  - Gains points for unique bugs found (scaled by how many other teams found the same bug)

- Winners for both categories at end of round 3

Goals

- Encourage defense, not just offense
  - Tie together security with reliability: Bugs are bad, whether they are exploitable or not
  - Elevate real concerns: performance and maintainability
- Provide direct feedback
  - A lack of security is penalized: “feel” the mistake!
- Empirically assess what actually works
  - Correlate features of submission with score
    - Programming language, framework, library, …
    - Developer experience, S/W process, …
    - Using static analysis, fuzz testing, etc. …

Requirements: Making it work

- Scalability – hundreds of submissions
  - Requires (mostly) automated testing, scoring
- Handle adversarial participants
  - DOS the scoring system
  - Report the same bug multiple times in slightly different ways
  - Collusion
- Get data from which we can draw interesting conclusions

Platform

- Submissions run in a VM that we provide
  - We unpack their submission in a defined directory and then run tests etc. within the VM
- Several benefits
  - VM is isolated from other software, limiting its negative effects on ours and others’ software
  - Run-time environment is clearly defined (in advance), yet affords plenty of flexibility

Data

- Teams must use our git repository
  - So we can see their process and intermediate checkins
- Teams must answer (brief) popup surveys during each phase
  - What are you working on? What problems are you dealing with? Who is doing what?
- And, of course, tests and final submissions available

Challenge I

- How to automatically judge whether a bug claim (submitted as a test) is valid?
  - Use Bayesian network to judge the likelihood test is valid based on outcome for all submissions
  - Seed network with results of true tests
  - Builder teams can, during the fix-it phase, argue that any bugs that slip through are not bugs
    - Human judges arbitrate

Challenge II

- How to automatically judge whether two submitted tests are morally the same?
  - **Incentive for builders**: find bugs that are the same in fix-it phase
  - **Incentive for breakers**: only allowed 10 test cases per submission (want to avoid duplicates)
  - (Best effort) **automation**:
    - Idea: test case minimization (e.g., delta debugging)
    - Idea: “footprint” across all submissions

Challenge III

- How to determine scores?
  - More points for an exploit vs. a correctness bug
  - Want to encourage coverage – don’t want to crown winner only because no one looked at code
    - Limit 10 bugs per submission
  - Want to encourage finding deep/challenging bugs
    - Bugs are worth more (to break-it teams) if fewer teams find them

Challenge IV

- How to avoid collusion or behavior not in the spirit of the competition?
  - Disallow direct obfuscation (judges will check)
    - Indirect uses (spaghetti code that looks human-written) might hurt performance, or might actually be relevant
  - Disallow cooperation among build-it teams
    - Goal would be to obtain more than one prize position
    - Run similarity detection tools on submissions

What are the right tasks?

- Must be interesting
- Must be able to complete in 72 hours
- Must have a reasonable attack surface

- Examples: parsers/interpreters/game engines
  - Pilot: SDXF parser (arcane file format)

- Ideas?

Let’s go write some secure code!

MARYLAND
CYBERSECURITY CENTER