Conventional Attacks on Scientific Applications

Michael Polinski Peter Dinda

Prescience Lab | Northwestern

How vulnerable are scientific applications to conventional attacks mounted through their inputs? Prior work largely leaves this question unanswered.

Our answer is *very*. Untrusted data can have catastrophic effects.

The Stakes are High

Attacks on Computational Science Matter

- Scientific results feed national and global decisions: climate modeling, weather forecasts, materials, drug discovery
- Scientific software is large (millions of LOC) and written in *unsafe* languages (C/C++/Fortran)
- Inputs are huge, diverse, and unauthenticated: shared configurations, data files, even live sensor streams

An Overlooked Attack Surface

- Existing work focuses on *numerical correctness*: errors are treated as random, not adversarial
- Fuzzing dominates vulnerability discovery in other software, yet has not reached scientific applications
- No quantitative evidence for how vulnerable these programs are

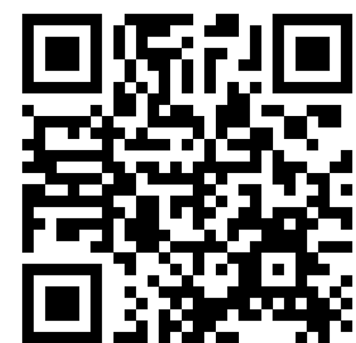
Research Questions

- How vulnerable are scientific applications to attack through their inputs, especially to **arbitrary code execution (ACE)**?
- What is the vulnerability landscape? Are there trends in the kinds of bugs present?

References

[1] Marc Heuse, Heiko Eißfeldt, Andrea Fioraldi, and Dominik Maier. AFL++. `https://github.com/AFLplusplus/AFLplusplus`, 2022.

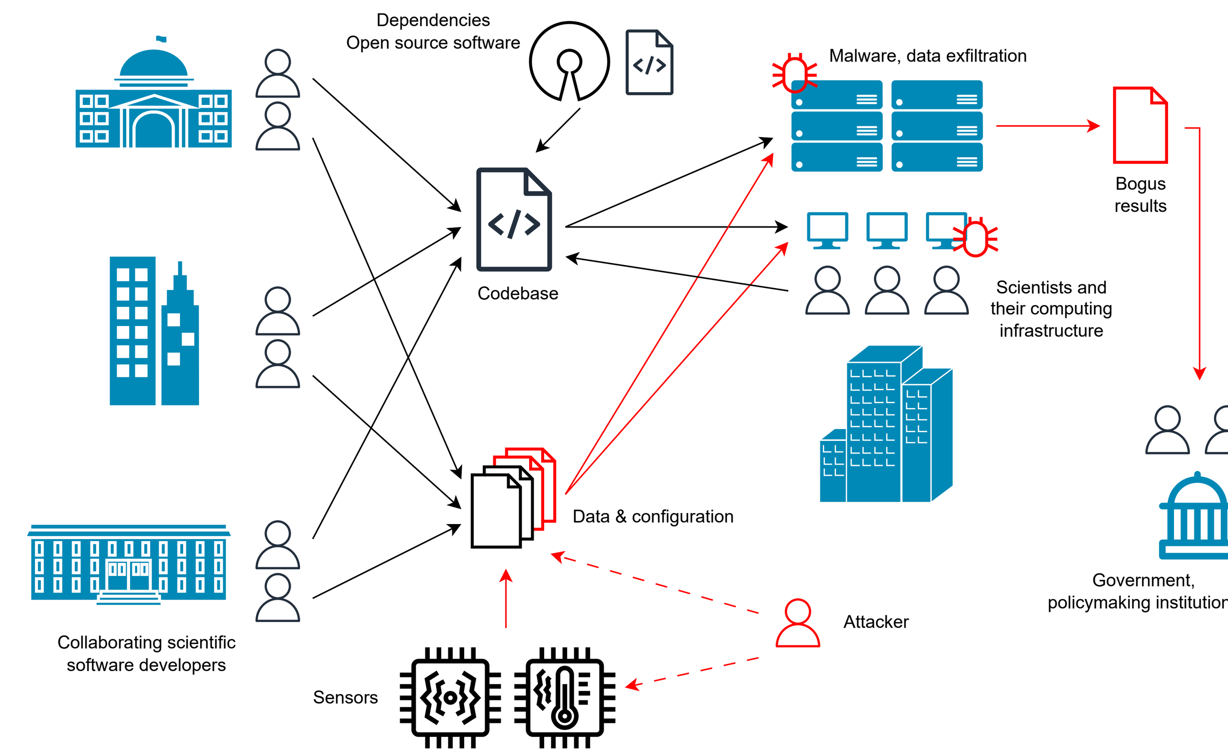
[2] Michael Polinski and Peter Dinda. A first look at conventional attacks on scientific applications. Under review, 2026.



See this poster and related publications at `https://buoyancy-project.org/#publications`

Computational Science is Vulnerable

- Inputs are collaboratively assembled, shared, and rarely sanity-checked: this is the reality of **open science**
- Attacker can eventually control some input (data, configuration, sensor stream); *not* an insider, no prior compromise needed
- Goals: *bogus but plausible results*, data exfiltration, persistent malware



Fuzzing with AFL++

- **AFL++**, a coverage-guided fuzzer, drives every program
- Per-application **fuzz harness** with a tiny in-memory virtual filesystem
- Two builds per target: **ASAN** + stack-hardened, combined with SAND
- **Triage** clusters by backtrace (with AFLTriage) and label exploitability heuristically (with Exploitable gdb plugin)

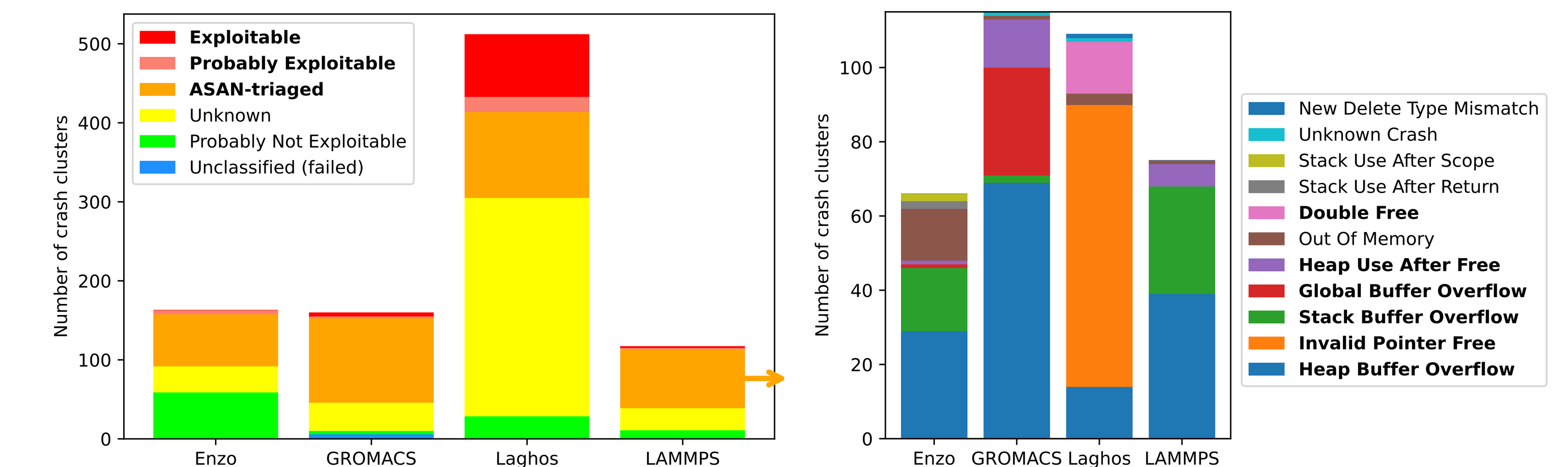
Almost 1000 Bugs in 1 Week!

- **952 distinct bugs** surface across the four targets (~3 million lines of code) in a week of fuzzing
- ~**40%** cause a segfault, ~**30%** are caught by ASAN
- Discovery rate *does not plateau*: more bugs are waiting

Program	Total Clusters [Distinct Bugs]	SEGV [Memory Crashes]	ABRT [Deliberate Aborts] ASAN Caught heap	ASAN Caught stack	Application Caught	Other
Enzo	163 (100%)	91 (56%)	30 (18%)	21 (13%)	21 (13%)	0 (00%)
GROMACS	160 (100%)	45 (28%)	82 (52%)	2 (01%)	31 (19%)	0 (00%)
Laghos	512 (100%)	196 (38%)	90 (18%)	0 (00%)	219 (43%)	7 (01%)
LAMMPS	117 (100%)	42 (36%)	45 (38%)	29 (25%)	1 (01%)	0 (00%)
Totals:	**952 (100%)**	**374 (39%)**	**247 (25%)**	**52 (06%)**	**272 (29%)**	**7 (01%)**

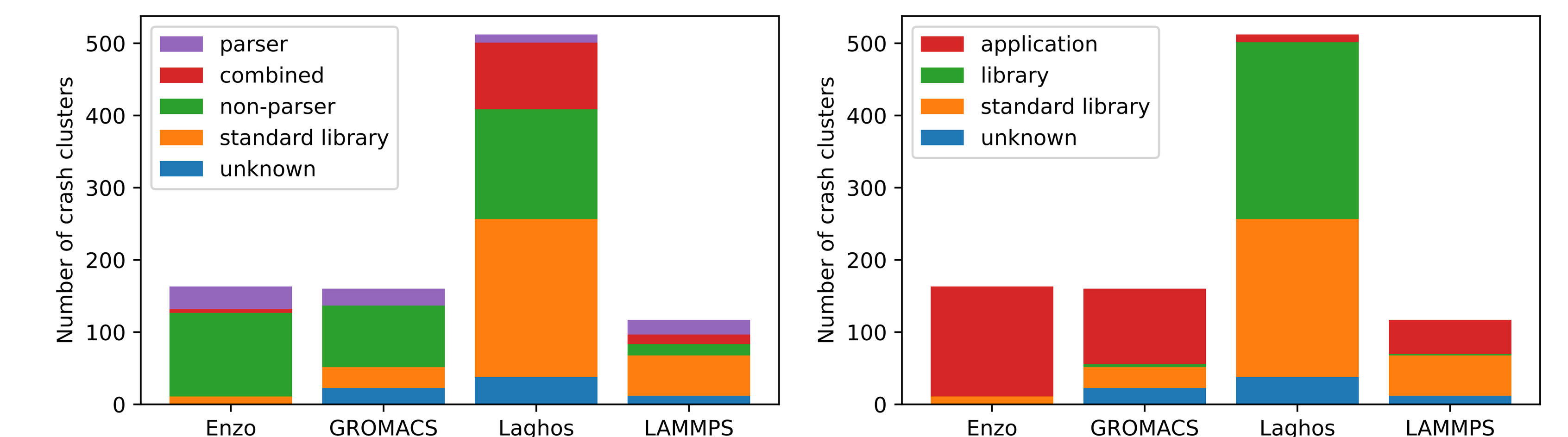
Many Bugs are Exploitable!

- Per-target average of **7.3%** of discovered bugs are likely exploitable for ACE, ranging from **3%** (Enzo) to **19%** (Laghos)
- **25%** are heap memory safety issues; GROMACS at a striking **52%**
- Heap buffer overflows dominate ASAN-detected bugs across every target; all seven most frequent types are well-studied avenues of attack



Bugs cluster, but not (only) in parsers

- $\leq$ 5% of source files in each codebase host all detected bugs
- For most targets, parsers are *not* the dominant crash site. Some of these bugs are probably *not* surface-level.



Going Forward

- All input — files, sensor streams, command-line arguments — belongs outside the security boundary
- Memory corruption from external input is *always* a bug
- Mitigation has tractable starting points (concentrated bug sites, known bug classes) but cannot stop at parsers. Carefully applied sandboxing (e.g. Linux *Landlock*) would help, but cannot correct bogus results.
- **Is it also feasible to steer computational results without conventional exploits?** We have ongoing work on this question.