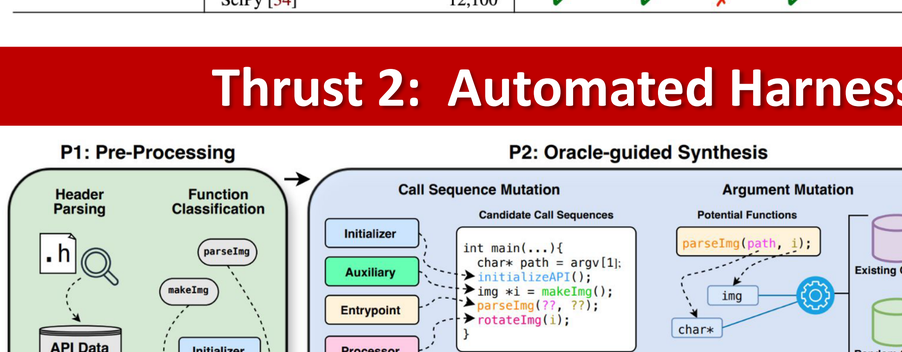
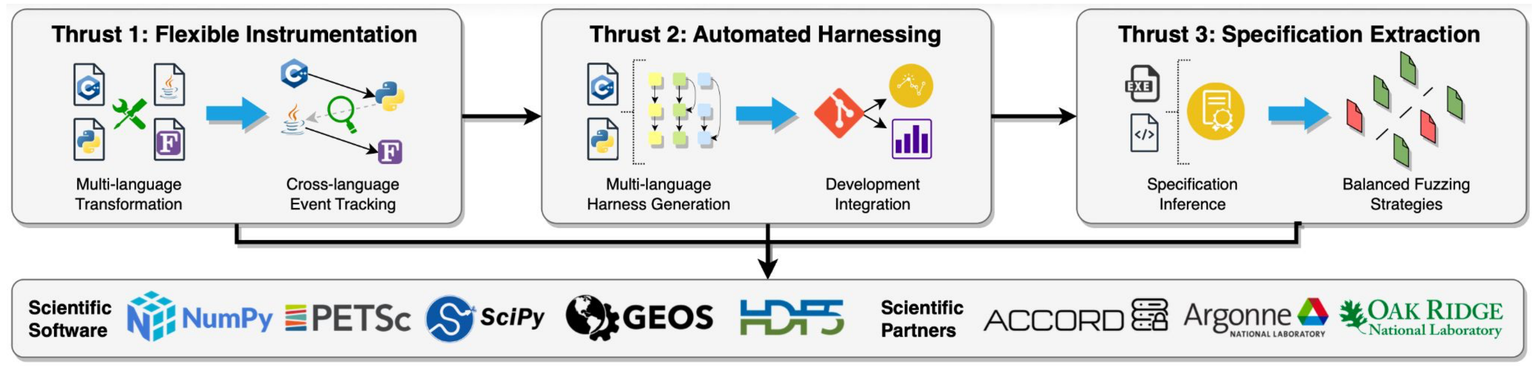
(#2419798) CICI: TCR: Practical, Systematic Fuzz Testing for Securing Scientific Software

Prof. Stefan Nagy (PI; University of Utah; snagy@cs.utah.edu), **Prof. Jack Davidson** (Co-PI; University of Virginia; jwd@virginia.edu)

Introduction: Project Motivation & Fundamental Research Contributions

Scientific software underpins critical infrastructure—from national labs to large-scale simulation platforms—yet remains largely under-vetted by modern software testing. While **coverage-guided fuzzing *(shown below)*** revolutionized vulnerability discovery in conventional software, its effectiveness breaks down on scientific software due to three fundamental gaps: **(1) lack of cross-language instrumentation** for heterogeneous codebases, **(2) the absence of scalable harness synthesis techniques** for isolating component-level testing, as well as **(3) missing input specifications** that prevent effective generation of test cases.





This project advances security vetting of scientific software by transitioning key advancements in fuzzing, developing new composable capabilities—**(1) cross-language instrumentation**, **(2) automated harness synthesis**, and **(3) input specification recovery**—extending the reach of fuzzing across today's increasingly diverse, critical, and under-vetted ecosystems of scientific software libraries and applications.

The project will transition its outcomes through **deployment on real-world scientific cyberinfrastructure** (e.g., ACCORD) as well as collaboration with scientific software developers, while releasing open-source fuzzing tools and integrating them into education and outreach efforts. To date, this project has already uncovered **227 new software defects**, including in science-critical software like CPython and HDF5.

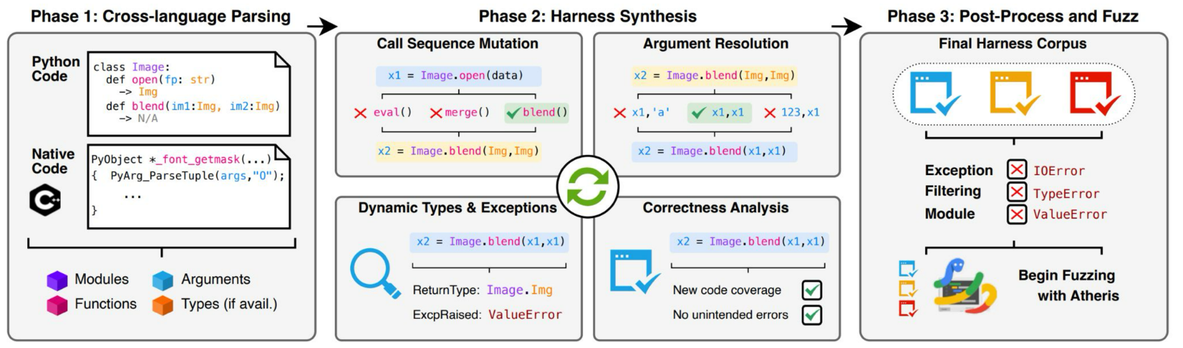
Challenges: Why Scientific Software is Hard to Fuzz

Domain	Name	Git ★	Challenge 1: Programming Languages C / C++	Fortran	Java	Python	Other	Challenge 2: Harnessing	Challenge 3: Specifications
Algebraic	BLIS [1]	1,700	✔	✔	✗	✗		✗	✗
	Eigen [8]	860	✔	✗	✗	✗		✔ [21]	✗
	GinkGo [12]	351	✔	✗	✗	✗		✗	✗
	LAPACK [14]	1,300	✔	✔	✗	✗		✗	✗
	OpenBLAS [20]	5,700	✔	✔	✗	✗	x86	✗	✗
	PyAMG [30]	517	✔	✗	✗	✔		✗	✗
Data Storage	HDF5 [13]	447	✔	✔	✔	✗		👎 [24]	✔ [6]
	NetCDF [18]	472	✔	✔	✔	✗		👎 [25]	✔ [17]
	Zarr [37]	1,300	✗	✗	✗	✔		✗	✗
Geometry	CGal [3]	4,400	✔	✗	✗	✗		✗	✗
	CVXPY [5]	4,900	✔	✗	✗	✔		✗	✗
	Firedrake [10]	445	✗	✗	✗	✔		✗	✗
	Geos [11]	1,000	✔	✗	✗	✗		👎 [23]	✗
	libMesh [15]	595	✔	✗	✗	✗		✗	✗
	PMP [29]	1,100	✔	✗	✗	✗		✗	✗
Signal Processing	Pyo [31]	1,200	✔	✗	✗	✔		✗	✗
	FFTW3 [9]	2,500	✔	✗	✗	✗	OCaml	👎 [22]	✗
	PyWavelets [33]	1,800	✔	✗	✗	✔		✗	✗
Visualization	Mayavi [16]	1,200	✗	✗	✗	✔		✗	✗
	PyVista [32]	2,100	✗	✗	✗	✔		✗	✗
	Vedo [35]	1,800	✗	✗	✗	✔		✗	✗
	yt [36]	422	✗	✗	✗	✔		✗	✗
Multi-purpose Toolkit	DifferentialEquations.jl [7]	2,700	✗	✗	✗	✗	Julia	✗	✗
	NumPy [19]	25,400	✔	✔	✗	✔		✔ [26]	✗
	PETSc [28]	139	✔	✔	✗	✔		✗	✗
	SciPy [34]	12,100	✔	✔	✗	✔		👎 [27]	✗

Thrust 1: Flexible Instrumentation

Overview: We are developing a flexible, cross-language instrumentation framework that captures unified execution feedback across heterogeneous scientific software stacks (e.g., Python–C/C++–Fortran). Our **SnakeCharmer** (FSE'26) results highlight how critical program behavior spans language boundaries, such as Python interfaces and native C components jointly defining API semantics, data flow, and exception behavior.

Outcomes: Our cross-language fuzzing efforts uncovered **20 new bugs** in pure and native Python libraries, including a 17 year old error within CPython itself. We are now working on expanding this to other languages.



Thrust 2: Automated Harnessing

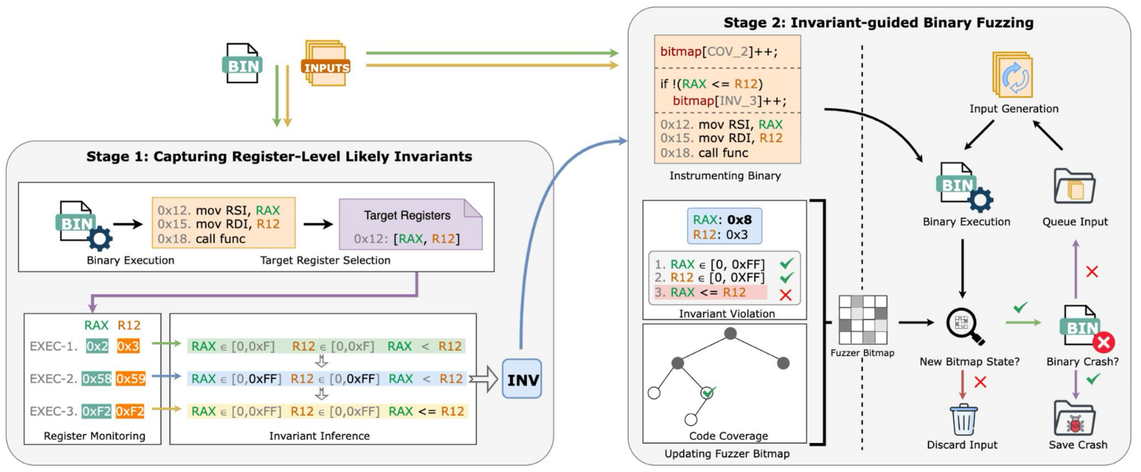
Overview: We are developing automated techniques to synthesize semantically-correct API harnesses that expose deep functionality of scientific software without manual effort. By generating and refining candidate call sequences, arguments, and data flows using mutation and correctness checking, our approach converges on constructing valid harnesses that enable effective fuzzing of complex, multi-language scientific software APIs.

Outcomes: Our approach **Oracle-guided Harnessing** (ICSE'25) discovers **41 new bugs** across real-world libraries, outperforming both developer-written harnesses and prior auto-harnessing, with **0% false positives**.

Thrust 3: Specification Extraction

Overview: We are developing techniques to automatically recover input structure and semantics directly from programs and executions, enabling fuzzing to create valid, high-quality inputs for complex scientific software.

Outcomes: Our semantics-aware binary fuzzing technique, **Register-level Likely Invariants** (FSE'26), surfaces **20 new bugs** missed by coverage-guided fuzzing, while exploring **27× more** unique program states.



Research Impacts: Education, Publications, Discovered Bugs, and Open-Sourced Software

Undergraduate/graduate courses developed:

- **CS 5493/6493:** Applied Software Testing (Utah)
- **CS 4501/6501:** Software Security Testing (UVA)

University of Utah students found & reported **60+ security defects** in real-world software.

cs.utah.edu/~snagy/courses/cs5493/bugs

Papers in top security / systems / SE venues:

1× ACSAC'25 | 3× FSE'26 | 1× ICSE'25 | 1× ATC'25 | 1× ASE'25

futures.cs.utah.edu/publications

Software defects discovered & reported:

227 bugs

Affected scientific software: GEOS, pillow, python, HDF5

futures.cs.utah.edu/bugs

Open-source fuzzing tools released:

BIN2WRONG, OGHarn, TeTRIS, PROGnosticator, GUIFuzz++, Binvariants

github.com/orgs/FuturesLab/repositories

The 2nd Cybersecurity Innovation for Cyberinfrastructure (CICI) Principle Investigator Meeting
May 11-12, 2026 | Los Angeles, California

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH
UNIVERSITY of VIRGINIA
ENGINEERING
Department of Computer Science