Prompt, Reliable, and Safe Security Updates for Cyberinfrastructure

NSF CICI: Transition to Cyberinfrastructure Resilience (TCR) | Award #2319880

PI: Jun Xu | **Co-PIs: Thomas Cheatham, Hang Liu**

University of Utah & Rutgers University | Deployment Partner: Utah Center for High-Performance Computing (CHPC)

MOTIVATION

Cyberinfrastructure (CI) drives modern scientific discovery. It carries high-value data and massive computing resources, becoming a prime target for cyber attacks.

*"As many as **85% of targeted attacks are preventable** via patching." — US-CERT*

Timely security updates are the most practical defense, yet CI facilities face unique challenges that they lack the resources or expertise to address.

OPEN CHALLENGES

✖ **Lacking Patch Management** *[Promptness]*

CI cannot afford commercial systems. Self-compiled software escapes package managers.

✖ **Missing Patch Testing** *[Reliability]*

Open-source patches lack quality control. They can be incomplete, defective, or malicious.

✖ **Unsafe Patch Deployment** *[Safety]*

Patches break dependents (e.g., GPU driver updates breaking CUDA/PyTorch).

OUR GOAL

Enable prompt, reliable, and safe security updates for CI

Three research thrusts:

Promptness → **Reliability** → **Safety**

THRUST 1: Patch Presence Management

Task 1.1: Patch Management System

• Open-source frameworks (Uyuni, Foreman) + community-driven data source for CI software

Task 1.2: Patch Presence Detection

• Semantic analysis to detect unadopted patches in self-compiled software (PDiff)

Task 1.3: Patch Urgency Assessment

• Automatic Exploit Generation to demonstrate attack consequences (FUZE, EXPGraph)

THRUST 2: Patch Reliability Testing

Task 2.1: Patch-driven Regression Testing

• Hybrid fuzzing tailored for patch sites with patch-driven scheduling and mutation (SAVIOR)

Task 2.2: Differential Patch Analysis

• Compare patched vs. unpatched versions to verify fix effectiveness and detect regressions (DarkSea)

THRUST 3: Patch Safety Assessment

Task 3.1: Dependency Graph Construction

• System-wide dependency graph via static + dynamic analysis for C/C++, Python, and binaries

Task 3.2: Patch Safety Testing

• Minimal testbed with only affected components; coverage-based fuzzing for safety assessment

TRANSITION & EVALUATION

3-Phase Transition to CHPC: Phase 1 - Lab (VMs simulating CHPC) → Phase 2 - Testbed (retired HPC nodes) → Phase 3 - Production (full CHPC deployment)

Evaluation: Maturity (defects/week, outages, coverage) | Security (# unadopted/critical/defective patches) | Cost (admin hours, computing resources)

Deployment: Systems deployed to Utah CHPC (5,600 users). Will disseminate to PNNL, ORNL, and Rutgers OARC. All tools to be open-sourced.

RESULTS & FINDINGS

Toolchain: AI-based audit — agent with binary decompilation + static analysis checks patch alignment (Vulnerable / Patched / No Use).

CVE + Patch Database (2020–2026): 216 projects, 3,898 unique CVEs, 1,910 patch commits.

CHPC Deployment (Apr 2025 – Mar 2026): 813 projects monitored (568 open-source, 113 closed-source, 132 custom); 6,528 binaries scanned.

What Our Scientists Are Running On: **399 unpatched CVEs** identified in CHPC binaries loaded by users. Popular software include libxml2, git, gdal, and postgresql. The earliest CVE traces back to 2020 (CVE-2020-24977, libxml2 2.9.9).

This material is based upon work supported by the National Science Foundation under Award #2319880 | Contact: junxzm@cs.utah.edu