Lightweight Formal Methods for Improving Software Security

Andrew Berns  
*University of Northern Iowa*

James Curbow  
*University of Northern Iowa*

*See next page for additional authors*

Copyright ©2019 Andrew Berns, James Curbow, Joshua Hilliard, Sheriff Jorkeh, and Miho Sanders

Follow this and additional works at: [https://scholarworks.uni.edu/rcapitol](https://scholarworks.uni.edu/rcapitol)

Part of the [Software Engineering Commons](https://scholarworks.uni.edu/rcapitol)

*Let us know how access to this document benefits you*

**Recommended Citation**

https://scholarworks.uni.edu/rcapitol/2019/all/11

This Open Access Poster Presentation is brought to you for free and open access by the University Honors Program at UNI ScholarWorks. It has been accepted for inclusion in Research in the Capitol by an authorized administrator of UNI ScholarWorks. For more information, please contact scholarworks@uni.edu.
Lightweight Formal Methods for Improving Software Security
University of Northern Iowa
Dr. Andrew Berns, James Curbow, Joshua Hilliard, Sheriff Jorkeh, Miho Sanders

Introduction
Software plays an increasing role in managing the responsibilities of many areas in society. As we’ve continued to cede more responsibility to software, the potential damage done from insecure software has grown. It is not hard to find examples of security breaches that have resulted in major financial losses and personal hardships for consumers.

Research Goals
- Measure the effectiveness of modern static analysis tools at identifying, with programmer-defined annotations, security vulnerabilities.
- Identify a set of best practices for developers to follow when using formal specifications while programming or retrofitting existing programs.

Methods
- **Step 1**: Select a vulnerability in an open-source product.
  - ![CVE Details](https://github.com/jenkinsci/jenkins)
  - ![jenkins-versions.txt](https://github.com/jenkinsci/jenkins)
- **Step 2**: Identify the incorrect code and the corresponding fix.
- **Step 3**: Create annotations which might identify the error.
- **Step 4**: Test which annotations, if any, actually identify the error.

Lessons Learned
Retrofitting is still hard.
While the tools for static analysis with formal methods have improved in the past few years, it is still not easy to take a project which has not used formal methods and retrofit the code to work with current tools. This is often given as one of the main reasons why formal methods have not gained widespread acceptance for cybersecurity.

More annotations would be helpful.
Today’s static analysis tools can only check a subset of possible operations. Some of the annotations that have yet to be implemented, such as whether or not a particular variable is assigned a value inside a method, have only limited support. If these annotations were checkable, it might improve the success rate for detecting vulnerabilities.

Future Work
We are continuing to add to our dataset of software vulnerability corrections and annotations for these corrections.

As the dataset grows, we will be better able to identify the best practices for using formal methods to improve computer security.

References