

Apr 1st, 11:00 AM - 2:30 PM

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

Let us know how access to this document benefits you

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

Follow this and additional works at: <https://scholarworks.uni.edu/rcapitol>



Part of the [Software Engineering Commons](#)

Recommended Citation

Berns, Andrew; Curbow, James; Hilliard, Joshua; Jorkeh, Sheriff; and Sanders, Miho, "Lightweight Formal Methods for Improving Software Security" (2019). *Research in the Capitol*. 11.
<https://scholarworks.uni.edu/rcapitol/2019/all/11>

This Open Access Poster Presentation is brought to you for free and open access by the 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.

Author

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

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.

Best Buy shoppers payment information may have been exposed in data breach

The Marriott hack exposed the passport numbers of more than 5 million people

Facebook hack update: Nearly 30 million users' data stolen. How to find out if you're one of them

UIDAI's Aadhaar Software Hacked, ID Database Compromised, Experts Confirm

Bezos case exposes billionaires' vulnerability to hackers

Security flaw could expose credit card data

While formal methods have gained popularity in safety-critical systems, they are much less common in other categories of software. Formal methods can help create better software, however, which would also be more secure. "Half of cyber vulnerabilities are software defects and the cost of avoiding and mitigating software errors approaches \$100B annually." [1] Our project hopes to help bring formal methods to more types of software.

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.

The screenshot shows the CVE Details page for CVE-2017-1000393. The vulnerability is titled "Jenkins 2.73.1 and earlier, 2.83 and earlier users with permission to create or configure 'master'". The description states: "This allowed them to run arbitrary shell commands on the master node whenever Scripts permission typically only granted to administrators." The CVSS score is 9.0 (Critical). The affected product is Jenkins. A file named "jenkins-versions.txt" (142 Bytes) is shown, containing the following content:

```
1 REPO: https://github.com/jenkinsci/jenkins
2
3 NAME: cve-2018-001
4 DESCRIPTION: Some generic description can go here.
5 PRE: 7472cb2
6 POST: 638ddee
7
```

- Step 2:** Identify the incorrect code and the corresponding fix.

The screenshot shows a code editor with the file "admin_only.c". The code is as follows:

```
... @ -6,5 +6,7 @@ typedef struct {
6     } Session;
7
8     void print_if_admin(Session s) {
9         - printf("Only admin should be printing this.\n");
10        + if (s.admin) {
11        +     printf("Only admin should be printing this.\n");
12    }
13 }
```

- Step 3:** Create annotations which might identify the error.

The screenshot shows the "annotations.json" file (134 Bytes) with the following content:

```
1 [
2   {
3     "filename": "admin_only.c",
4     "label": "Only admin",
5     "annotation": "//@ assert s.admin;"
6   }
7 ]
```

- Step 4:** Test which annotations, if any, actually identify the error.

The screenshot shows the terminal output of a formal verification tool. The output is as follows:

```
Checking annotations in pre ...
... done.
Checking annotations in post ...
... done.
pre: ['[wp] Proved goals: 0 / 2']
post: ['[wp] Proved goals: 1 / 2']
```

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 [2].

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

References

- [1] Schaffer, K., & Voas, J. (2016). What Happened to Formal Methods for Security? *Computer*, 49(8), 70-79. doi:10.1109/mc.2016.228
- [2] Chong, S., Guttman, J., Datta, A., Myers, A., Pierce, B., Schaumont, P., . . . Zeldovich, N. (2016, August 1). Report on the NSF Workshop on Formal Methods for Security. Retrieved from <http://dl.acm.org/citation.cfm?id=3040225>