I returned to graduate school to study software engineering with a singular goal: I wanted to work on evidence-based approaches to making software. I specifically came to North Carolina State University because of the concentration of empirical software engineering faculty, chiefly my advisor, Dr. Laurie Williams. Because of my background in security, studying empirical methods for securing code was a natural fit. To date, my research has taken on two directions in the area of engineering secure software: both the **maintenance of existing systems** and in **providing a strong foundation for secure development through security education research**. My work on data-driven approaches to secure software development inspires me to investigate ways to improve the **usability of security**.

RASA - Risk Based Attack Surface Approximation

My dissertation is on security metrics, or how to drive security-based engineering decisions based on empirical data. My work focuses on crash dump stack traces and the code that appears on said traces as a proxy for potentially vulnerable code. We call this approach Risk Based Attack Surface Approximation, or RASA. RASA's development is based on several observations:

- **Code on crashes is by definition accessible to end users.** If code appears on a crash dump stack trace that was generated by an end user, we then have empirical evidence that the code was executed in some form by the end user. This observation means that metrics developed from crash dump stack traces generated by end users can give a profile of the end user that generated them: how frequently code crashes could be a proxy for both legitimate and illegitimate use, the order in which code crashes indicates known flawed data flows through the system, et cetera.
- **Code that crashes is indicative of code with data handling issues.** Crashing code is also indicative of code that has some sort of data or process handling issue, based on the fact that the code is crashing at all. The existence of this type of error indicates that the code may also have security errors, as security coding errors are often the result of some failure in data flow or process handling. The crash *itself* can be considered a denial of service attack if the crash can be reliably executed.
- **Crash dump stack traces are often collected but underutilized.** Finally, crash dump stack traces are often collected and displayed in dashboards by development teams, but are rarely used beyond this purpose. By leveraging this existing data that organizations already have, RASA can provide immediate insights to teams without requiring a brand new data source to be created. Metric development that makes creative use of existing data rather than the collection of new data can have a faster turnaround time for productive impact.

Based on these observations, I have run four studies on this topic, two appearing at conferences [1, 2], and two more under review. I also participated in the ACM Student Research Competition with RASA, finishing first at FSE in 2015 and third in the ACM Grand Finals the subsequent year. These studies have shown that the RASA approach can effectively target code that has potential security vulnerabilities, that fractions of the total set of crashes collected from a system can target vulnerabilities effectively, and that there is a correlation between the frequency and complexity of crashing code and vulnerable code. In addition to these four studies, I am currently working on

a comparison study between RASA and other approaches for predicting the location of security vulnerabilities in software systems.

Translating Metrics Into Developer Action

During the development of RASA, we solicited feedback from both the research community and the practitioner community on how RASA could have real-world impact for developers. The feedback we received provided us with some potential directions for research to help preventative security engineering:

- **Statistics need meaningful presentation.** Developers and managers may find a list of potentially vulnerable code artifacts too overwhelming to be actionable. Meaningful metrics can help teams catch vulnerabilities *before* they become a public issue. A *ranking* system for potentially vulnerable code is easier for teams to prioritize effort for, rather than a binary on/off system.
- **Developers are used to visualizations.** Early in the development process for RASA, I presented our preliminary findings to a security team at Microsoft. In that presentation, I created a graph representation of the attack surface of Windows to help introduce the concept of a crash dump-based attack surface of Windows. This graph was seized on by the team I was presenting to, and they asked if such a visualization could be developed for presenting the results. Visualizations can create overviews that are more easily digestible for developers, particularly developers who are visual learners.

Based on these observations, I have run one study on further prioritization of code for security inspection beyond a simple on/off approach, and I will run a study exploring the effect of RASA visualizations on the speed and accuracy of discovering security vulnerabilities with novices. Studies exploring the actual impact of cutting-edge security research on practitioners is needed to properly disseminate the knowledge being generated at top-tier security conferences.

The Science of Usable Security

Based on my exploration of software security, I would like to explore metrics designed to measure how people *use* software in a security context. One of the interesting pieces of RASA is that it provides insight into how software is used by end users, in the form of the frequency with which specific code crashes. I would like to take this a step further to understand how customers are using software in these security contexts. A lot of blame is placed on end users using software improperly. For example, an email user clicking what seems to experienced users to be an "obvious" phishing attempt by a malicious entity. Taking secure actions should be as easy as possible all users, not just knowledgeable ones. Developing scientific first principles for guiding users towards secure actions can help prevent security violations from taking place.

Security Education

My research has also covered the area of software security education. In 2015 and 2017, I helped my advisor, Dr. Laurie Williams, run a free, open, online course (i.e. a MOOC) on software security

based on the graduate level course by the same name at North Carolina State University. After the conclusion of both courses, we published studies covering the demographics of the students that participated, the course content, and lessons learned for future iterations of software security courses in the future [3, 4]. Creating better coursework for the education of the next generation of security professionals is important to meet future security threats. Disseminating cutting edge security research to protect customers and organizations is as important as the research itself. Stronger coursework both in a formal classroom setting and in materials available online for security professionals can make us all safer.

I believe a gap exists in security education between a basic introduction to the topic and advanced activities. Some universities have more theoretical security courses that focus on the mathematical aspects of security, such as the inner workings of network protocols and cryptography algorithms. On the other end, courses such as Offensive Security's penetration testing course focuses on complicated penetration testing activities. Further research into effective bridges between these two types of courses can help de-mystify cybersecurity. For example, guided examples on how to perform a basic exploit, and then fix the issue that caused the exploit, can help make security topics less intimidating for students who do not have an immediate interest or affinity for the topic.

Conclusion

My time in graduate school has firmly convinced me that the biggest issue in cybersecurity today is the dissemination of knowledge. We have a wealth of information available in cybersecurity, with incredible minds developing both interesting exploits and excellent techniques for protecting systems and users. Most breakdowns in practice come from a lack of dissemination of this knowledge. The Equifax breach in 2017 was the result of a failure to update software with a known vulnerability rather than an innovative exploit by a brilliant hacker. Codifying proper security hygiene and making sure defensive security is easy to execute and understand by non-experts is the best way forward to protecting the world from cyberattacks.

References

[1] Christopher Theisen et al. "Approximating Attack Surfaces with Stack Traces". In: *IEEE/ACM 37th IEEE International Conference on Software Engineering* (2015), pp. 199–208. DOI: 10.1109/ICSE.2015.148.

[2] Christopher Theisen et al. "Risk-based attack surface approximation: how much data is enough?" In: *Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017 IEEE 39th International Conference on*. IEEE. 2017, pp. 273–282.

[3] Christopher Theisen et al. "Software security education at scale". In: *Software Engineering Companion (ICSE-C), IEEE/ International Conference on*. IEEE. 2016, pp. 346–355.

[4] Christopher Theisen et al. "Teaching Secure Software Development Through an Online Course". In: *Secure Software Engineering in DevOps and Agile Development, International Workshop on*. ESORICS. 2017.