Research Statement
Payas Awadhutkar

Today, nearly every facet of daily life depends on software. Software systems connected to the internet make up the critical infrastructure necessary for our livelihoods, ranging from managing wealth to even medical care. This critical infrastructure is increasingly facing malicious cyber intrusions. For instance, in 2020 the Federal government faced serious data breaches due to a flaw in the software supply chain. Cybersecurity is recognized as a problem of national importance and President Biden issued an executive order in 2021 with the intent of improving the Nation's Cybersecurity [1]. It is important to ensure that the existing infrastructure is resilient to cyberattacks and maintain it against potential vulnerabilities. My research is aimed at developing methods and tools to help solve this problem.

My research makes it possible to identify security flaws in software systems and fix them automatically. The overarching goal is to make software systems more resilient and maintainable. There are two key challenges in building secure software systems. First is the soft infinity of execution behaviors. Software systems encode a practically infinite number of possibilities and all of them must be verified. Even a few lines of code can encode execution behaviors more than the number of atoms in the universe. The second is about proving the presence or absence of the vulnerability. How does a human know that the software is really secure and the analysis did not make a mistake? My research is about developing novel solutions using new mathematical foundations to address these challenges.

Algorithmic Complexity and Side-Channel Vulnerabilities

Adversaries are constantly looking for new classes of vulnerabilities to exploit software systems. Defense Advanced Research Projects Agency (DARPA) identified two new classes of cybersecurity vulnerabilities, Algorithmic Complexity and Side-channel vulnerabilities. Algorithmic Complexity attacks are sophisticated Denial-of-Service (DoS) attacks. A well-known example is the Billion Laughs attack that plagues a majority of Microsoft applications [2]. Side-channel vulnerabilities enable information breaches in the systems. Spectre [3] which affected nearly every computer system on the planet is an example of a side-channel attack. The commonality between the two is that both are made possible due to a fundamental flaw in the protocol or algorithm rather than due to an implementation mistake. This creates endless variations of the two and often the analyst does not even know what the flaw looks like. Combined with the challenge of the soft infinity of execution behaviors, the situation is akin to finding a needle in a haystack where you don't know what the needle looks like.

DARPA created the Space/Time Analysis for Cybersecurity (STAC) program to develop new program analysis techniques and tools to identify these two vulnerabilities. I participated in the program as part of the Iowa State University(ISU)/EnSoft Corp. team. I identified that the core cause of both classes of vulnerabilities is in specific kinds of loops. I developed a taxonomy to describe the vulnerable loops, leading to the creation of the loop catalog [4]. Loop Catalog enabled my team to narrow down the scope of analysis and overcome the challenge of soft infinity of execution behaviors. I developed DISCOVER [5], a tool to identify potentially vulnerable code segments that a human analyst could analyze to identify if a vulnerability is present in the code. DISCOVER also presents human-comprehensible evidence of its analysis, aiding the human

analysts as they only need to verify or refute DISCOVER's analysis. DISCOVER enabled my team to become the top-performing team on the STAC program. This success led to its adoption by the Air Force Research Laboratory as part of its analysis tool suite.

Verification of the Linux kernel

Linux kernel is arguably the most important software in use today. It is part of practically all critical cyber-physical systems. The two most common sources of vulnerabilities in the Linux kernel are Lock/Unlock pairing and Memory Leak. I worked on developing tools to verify the Linux kernel to identify these two flaws. While working on this problem, I realized that simply stating the presence or absence of the flaw is not enough as the analysis can make a mistake. I discovered an example of a well-known analyzer that erroneously declared a part of the Linux kernel secure when it contained a flawed Lock/Unlock mechanism [6]. Thus, evidence of the analysis that could be verified by both a computer and a human was needed. To this end, I developed a novel mathematical abstraction called Evidence Graph [7] that provides evidence for cross-checking.

Using the Evidence Graph, I developed a mathematical foundation to describe memory leak mechanisms. This led to the development of a verifier that can automatically verify memory leak instances in 21 million lines of code [7,8]. This verifier is the most accurate tool to verify lock/unlock and memory leaks in the Linux kernel. The results were crosschecked using Evidence Graph, instilling greater confidence in the correctness of the verifier.

Path Algebra

Soft infinity of the execution behaviors is the fundamental challenge of software verification. To secure software systems, each of the execution behaviors must be verified. Thus, to develop scalable techniques, a transformative solution to address this challenge is required. I contributed to the development of a novel mathematical technique based on Non-Commutative Ring Algebra to reduce the exponential number of execution behaviors in a representation requiring a linear amount of space. Algebraic representation can still be automatically analyzed, improving the scalability tremendously.

At EnSoft, I am currently working on developing commercial analyzers based on the Path Algebra. Path Algebra has been granted a patent [9] and I have created a tool to analyze legacy software using it. Path Algebra is a fundamental advance that has far-reaching implications. Apart from software verification, it also has potential applications in the analysis of metabolic pathways, critical in medical care.

Software Maintenance

Identifying the presence of a flaw in software is only the first step in securing software systems. The flaw also needs to be fixed. Fixing the software flaws is a non-trivial process. For instance, fixing the recent Log4j vulnerability [10] required countless resources. Defense Advanced Research Projects Agency (DARPA) has created multiple programs to address this need [11,12,13]. I participated in the DARPA Intent-Defined Adaptive Software (IDAS) program as an evaluator. I developed challenge problems to drive the research forward to fix the software flaws automatically. This experience helped me understand the difficulties in maintaining legacy software systems. Recently, I have turned my attention to addressing them.

At EnSoft, I am currently working on two commercial products. The first product is called SimHance which is aimed at the Automotive and Avionics industry. Automotive and Avionics software is perhaps the most common safety-critical software and failure to secure it can lead to the loss of wealth and human lives. Automotive and Avionics software is commonly developed as Simulink models. SimHance automatically enhances the models by making them easy to comprehend for engineers and fixing violations if any. A particular challenge is to display the model to engineers in an easy-to-grasp form. I developed a novel Computational Geometry algorithm to address this need.

The second product is for Binary Rewriting. Mission-critical legacy software is usually in the form of binary code and the source code is not available. Thus, a tool that can analyze the binaries and fix them by rewriting them on the fly is needed. Binary code is unique because it is a lossy representation and analyzers need to reconstruct the missing information about the compiler used to generate it. I developed a technique to extrapolate this information accurately and created a tool that can rewrite binaries with high fidelity. A patent is being filed for the technique.

References

[1] Executive Order on Improving Nation's cybersecurity, https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/
[2] Denial of Service in Microsoft Office 2007-2013, CVE-2014-2730, National Vulnerability Database
[3] Unauthorized disclosure of information via side channel, CVE-2017-5753, National Vulnerability Database
[4] Awadhutkar, Payas, Ganesh Ram Santhanam, Benjamin Holland, and Suresh Kothari. "Intelligence amplifying loop characterizations for detecting algorithmic complexity vulnerabilities." In 2017 24th Asia-Pacific Software Engineering Conference (APSEC), pp. 249-258. IEEE, 2017.
[5] Awadhutkar, Payas, Ganesh Ram Santhanam, Benjamin Holland, and Suresh Kothari. "DISCOVER: detecting algorithmic complexity vulnerabilities." In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1129-1133. 2019.
[6] Kothari, Suresh, Payas Awadhutkar, and Ahmed Tamrawi. "Insights for practicing engineers from a formal verification study of the linux kernel." In 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 264-270. IEEE, 2016.
[7] Awadhutkar, Payas Rajendra. "Human-centric verification for software safety and security." PhD diss., Iowa State University, 2020.
[8] Kothari, Suresh, Payas Awadhutkar, Ahmed Tamrawi, and Jon Mathews. "Modeling lessons from verifying large software systems for safety and security." In 2017 Winter Simulation Conference (WSC), pp. 1431-1442. IEEE, 2017.
[9] Kothari, Suraj C. 2023. “Method for Analyzing and Verifying Software for Safety and Security.” U.S. Patent 11,669,613, Issued June 6, 2023.
[10] Vulnerability in Apache Log4j Library, CVE-2021-44228, National Vulnerability Database
[11] Assured Micropatching, DARPA I2O, https://www.darpa.mil/program/assured-micropatching
[12] Intent-Defined Adaptive Software, DARPA I2O, https://www.darpa.mil/program/intent-defined-adaptive-software

[13] Verified Security and Performance Enhancement of Large Legacy Software, DARPA I2O, https://www.darpa.mil/program/verified-security-and-performance-enhancement-of-large-legacy-software