"Software and cathedrals are much the same – first we build them, then we pray."
Samuel T. Redwine, Jr.

With an abundance of languages, tools, and frameworks, it is becoming easier than ever to produce software, at a fraction of the time and cost that were required just a decade ago. At the same time, software systems are becoming increasingly more complex; we are long past the point at which an individual developer is able to fully comprehend the behavior of an average piece of software. It is an unsettling trend: As society's dependence on software increases, our overall understanding of these systems seems to be diminishing in parallel.

My research is driven by observations about how complex software sometimes fails to behave as intended, and how such failures impact system safety and security. Based on these observations, I work on tools and techniques for automating detection of critical flaws in software, and designing systems to be robust against a failure. I am especially interested in (1) **system-level analysis**, which involves analyzing the overall behavior of a system as a sum of its components and their interactions, and (2) **proactive detection and mitigation**, which involves identifying and addressing potential flaws before a system is fully implemented.

To achieve my research goals, I leverage existing and develop new techniques in automated verification, software modeling, program synthesis, and static analysis. To address the increasingly interdisciplinary nature of systems being constructed today, I collaborate not only with computer scientists, but also with researchers from other areas, including cyber-physical systems, control theory and multi-agent systems in AI.

My approach is characterized by the demonstration of research results through concrete, realistic case studies. I have applied my techniques to understand and improve the safety and security of complex real-world systems, including a radiation therapy system [15], a water treatment system [1], electronic voting machines [2], mobile devices [7], and web protocol implementations [4].

Current Research

A failure in a complex system often involves a subtle, unanticipated interaction between different aspects of a system. My research so far can be categorized by different types of interactions that give rise to critical safety or security vulnerabilities.

Interaction between Abstraction Layers Abstraction, although a fundamental technique in system design, can be a double-edged sword in domains where security is a major concern. While a developer tends to work with one abstraction layer at a time, an attacker is not bound to such restrictions, and may exploit details across multiple layers. For instance, a popular authorization protocol, OAuth, formally verified to be secure at the specification level [8, 18], has been shown to be susceptible to numerous browser-specific attacks when deployed as a web application [17]. A program in a high-level language (e.g., Java) may inadvertently expose private data when translated into a low-level representation (bytecode) [6]. An abstract data type, depending on the choice of the underlying structure and algorithms, may become vulnerable to side-channel attacks once implemented [13]. In all these cases, security risks arise from unanticipated interaction between an abstract entity and its concrete representation.

To help developers detect these types of *cross-layer* vulnerabilities, I developed *Poirot*, a framework designed to support a security analysis over multiple abstractions of a system [4] (**Distinguished Paper Award**). With Poirot, a developer begins with an abstract model of a system, and incrementally elaborates it by specifying a *representation mapping*, which describes how an abstract entity is to be represented in terms of concrete primitives (e.g., “encode an abstract user token as a browser cookie”). At each iteration, Poirot’s engine performs an automated analysis to check the system against a desired security property, and discover attacks that exploit details across multiple levels of abstraction. Poirot can also be used as a design exploration tool: Given only a partially specified mapping, it automatically enumerates and evaluates each design candidate with respect to the given property (e.g., generates different attacks depending on whether the token is implemented as a cookie or a URL parameter).

I have applied Poirot to analyze a number of deployed web-based systems (including IFTTT, OAuth, and HandMe.In, an app for tracking personal items) and detected several previously unknown vulnerabilities, all of which could be used to compromise the confidentiality and integrity of sensitive user data. Many of the attacks generated by Poirot exploited system details at multiple levels of abstraction. For example, an attack that Poirot discovered in an online store combined a weakness in the payment protocol with a lower-level browser vulnerability to trick a victim into paying for the attacker’s item; this attack would not have been found if the analysis was confined to a single abstraction layer.

By leveraging our prior work on software synthesis [14] (**Distinguished Paper Award**), I am currently extending Poirot with a capability to *synthesize* a secure representation mapping, which would guarantee that a given property continues to hold in a target, low-level abstraction. I plan to apply this technique to synthesize secure implementation rules for various web protocols, including OAuth, where some of the implementations have suffered attacks that exploit details omitted from the specification.

Interaction between a System and its Environment Many software failures occur due to invalid assumptions that a system makes about its environment [12]. This is becoming a critical issue especially for traditional safety-critical systems, which tend to rely on outdated assumptions that fail to capture security risks that arise when they are retroactively deployed on a modern computer network.

I have been studying the safety and security of a fully operational water treatment system in collaboration with researchers in cyber-physical systems (CPSs). Following traditional safety engineering practices, the system is designed to monitor and recover from occasional faults in mechanical parts such as sensors, water pumps, and motorized valves. Its design, however, does not take into account actions of a malicious agent that may attempt to compromise the system through its wireless network.

As part of this study, I developed a framework that can be used by an engineer to assess the safety of a system operating in a potentially malicious environment [1]. The framework takes as inputs a model of the system, a safety requirement (e.g., “water tank should never overflow”), and a model of the environment. The latter model is parameterized in terms of attacker capabilities; for instance, in the case of the water treatment system, the parameters describe the number of sensors and actuators that the attacker may compromise in order to manipulate the system into an unsafe state. By leveraging a verification engine, the framework systematically discovers potential attacks (with a set of minimal parameters) that lead to a violation of the requirement. My analysis of the water treatment system revealed a number of critical vulnerabilities that had not been previously known to engineers, and has resulted in an implementation of additional safety measures in the system.

Building on this study, I am currently working with experts in control theory to construct a general, expressive model of possible attacks on a CPS. Having defined a formal framework, our goal is to provide an automated analysis that can answer the following types of questions: What operations does the attacker need to perform to undermine system safety? What information is needed by a monitor to detect such attacks? We plan to evaluate the effectiveness of our framework by demonstrating potential attacks and detection mechanisms in the context of the water treatment system.

Interaction between Components of a System A modern system consists of multiple, heterogenous components that are composed together to satisfy the overall requirements of the system. Ideally, to ensure that the system functions as desired, one would thoroughly validate the behavior of individual components as well as their potential interactions. In practice, however, a development team has a limited amount of resources allocated to verification and testing, and rarely has the luxury of ensuring correctness across the entire system. So how does one argue that the system as a whole is reliable, despite the fact some of its parts may be unreliable?

One approach is to identify the parts of a system that are responsible for its most critical properties, and design the system to ensure that the failure of a non-critical component does not interfere with a critical one. This approach, while long advocated in safety engineering and computer security, has been little explored in the context of software development. To make this approach systematic, I developed a formal methodology for evaluating the reliability of a system based on the notion of *trusted bases* [2]. A trusted base is the subset of system components that together ensure that a desired property holds, even if other components fail to behave as intended. The key idea behind this methodology is that smaller the size of the trusted base, more feasible it is to verify the correctness of those components, and thus ensure critical properties of the system as a whole. To support the methodology, I developed (1) an analysis technique that, given a system model and a property, automatically identifies its trusted base [2, 5], and (2) design transformation patterns that can be used to reduce the size of an existing trusted base [3].

I applied this methodology to show how the most widely used voting system has a trusted base that spans the entire system, implying that a failure in any one part may compromise the integrity of the whole election [2]. I then evaluated an alternative voting system called Scantegrity [9], which is designed to preserve the election integrity, even with potential vulnerabilities in software components. In another case study [15], my colleagues and I analyzed a safety property of a radiation therapy machine at Massachusetts General Hospital (MGH), and showed that its trusted base included unreliable components that may undermine the safety of patients. Based on our analysis, we suggested ways to eliminate those components from the trusted base and improve the safety of the system.

Future Directions

With the rapidly increasing complexity of software systems, and greater consequences of their failures on our society, the widespread approach to ensuring quality based on ex post facto methods such as testing and patches will no longer be sufficient. Moving forward, I am interested in developing tools and methodologies for constructing software-intensive systems that are **safe and secure by design**.

Robustness in Software Design Programs that we build will continue to contain bugs, and our systems will rely on increasingly complex off-the-shelf components whose behaviors we do not fully comprehend. Unfortunately, software is notorious for its brittleness; due to its discrete nature, a seemingly innocuous flaw in one part of a system, or a slight deviation from an expected input, may undermine a critical property of the entire system. Is there an approach to construct a software system so that it continues to provide its most critical functions even in the presence of unexpected errors?

Building on my work on system-level analysis and trusted bases, I am interested in developing the notion of *robustness* as a criterion for evaluating software design. The key idea is to define robustness in terms of the degree of *assumptions* that a component makes about other components or the environment that it interacts with. As an example, consider the OAuth authentication protocol. Two different versions of the protocol (OAuth 1.0 and 2.0) have been developed and deployed widely across the web, but interestingly, version 1.0 is considered to be more secure and robust against potential attacks (by numerous security experts, including the original creator of OAuth [10]). The reason is that OAuth 2.0 relies on additional assumptions about the behavior of a browser-based component, which are particularly challenging to establish due to the abundance of browser-specific attacks. Currently, this type of system evaluation is performed in an informal manner by domain experts; given a well-defined notion of robustness, I believe that it can be made rigorous and automated.

The idea of using assumptions as a design criterion dates back to a seminal paper by David Parnas on information hiding, where he recognized the role of an interface as documenting assumptions that a module makes about its surroundings, and its impact on modularity [16]. I am eager to explore how a similar type of criterion may be used to evaluate and compare alternative designs of a system for properties such as safety and security. I plan to develop techniques that can be used to explicitly identify assumptions for a given system, generate feedback on the impact of an assumption violation on a critical property, and suggest ways to improve robustness by eliminating or reducing unnecessary assumptions in the form of program transformation rules.

Integrating Safety and Security Building a reliable cyber-physical system (CPS) poses a new unique set of challenges in both computer security and safety engineering. Due to increased human dependence, a failure of a CPS often has safety-related consequences, but the cause of its failure may stem from security vulnerabilities. For instance, the well-known Stuxnet attack employed a series of security exploits in order to ultimately cause catastrophic physical damage to critical infrastructures.

Traditionally, safety and security have been treated as separate fields of study, with relatively few attempts to integrate the two. With the recent growth of cyber-physical systems, I believe that knowledge and methods developed in both fields will become an indispensable part of system development. For instance, a wealth of information about attacks and mitigations available in the security literature will be essential for identifying risks that must be taken into account while designing a CPS. At the same time, safety analysis methods such as fault trees and failure mode effects analysis will need to be employed in order to accurately evaluate and predict the impact of security attacks on the overall safety of the system. I plan to investigate and define potential connections between safety and security in a more precise manner, and develop a methodology for designing a CPS that systematically leverages the techniques from both disciplines.

Design Methodologies for Autonomous Systems Our society is becoming increasingly reliant on computer systems that operate with little or no human intervention, such as self-driving cars, delivery drones, robots, and smart home devices. These types of systems, characterized by a group of interacting agents without a central controller, have been studied in the context of AI as *muti-agent systems* (MASs). I am excited about developing methodologies for building safe and secure autonomous systems by leveraging potential synergies between software engineering and MAS. For instance, in collaboration with researchers in MAS, I have developed an efficient technique for automatically synthesizing *norms*, which are a set of rules enforced on agents in order to guarantee a global property (e.g., "no collision among self-driving vehicles") [11]. Building on this collaboration, I plan to further investigate questions that are becoming increasingly relevant in our society: How do we design an autonomous system that is capable of adapting to a dynamic environment with a rapidly evolving set of agents? Can we construct a system that provides safety guarantees even when some of its agents may be untrustworthy?

Publications

[1] **E. Kang**, S. Adepu, D. Jackson, and A. P. Mathur. Model-based security analysis of a water treatment system. In *ICSE Workshop on Smart Cyber-Physical Systems*, 2016.

[2] **E. Kang** and D. Jackson. Dependability arguments with trusted bases. In *International Conference on Requirements Engineering (RE)*, 2010, pages 262–271.

[3] **E. Kang** and D. Jackson. Patterns for building dependable systems with trusted bases. In *Conference on Pattern Languages of Programs*, 2010, pages 1–19.

[4] **E. Kang**, A. Milicevic, and D. Jackson. Multi-representational security analysis. In *International Symposium on the Foundations of Software Engineering (FSE)*. **ACM SIGSOFT Distinguished Paper Award**, 2016, pages 181–192.

[5] **E. Kang**. A framework for dependability analysis of software systems with trusted bases. Master's thesis. Massachusetts Institute of Technology, 2010.

[7] H. Bagheri, **E. Kang**, S. Malek, and D. Jackson. Detection of design flaws in the Android permission protocol through bounded verification. In *International Symposium on Formal Methods (FM)*, 2015, pages 73–89.

[11] J. Hao, **E. Kang**, J. Sun, and D. Jackson. Designing minimal effective normative systems with the help of lightweight formal methods. In *International Symposium on the Foundations of Software Engineering (FSE)*, 2016, pages 50–60.

[14] A. Milicevic, J. P. Near, **E. Kang**, and D. Jackson. Alloy*: A general-purpose higher-order relational constraint solver. In *International Conference on Software Engineering (ICSE)*. **ACM SIGSOFT Distinguished Paper Award**, 2015, pages 609–619.

[15] J. P. Near, A. Milicevic, **E. Kang**, and D. Jackson. A lightweight code analysis and its role in evaluation of a dependability case. In *International Conference on Software Engineering (ICSE)*, 2011, pages 31–40.

External References

[6] M. Abadi. Protection in programming-language translations. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 1998, pages 868–883.

[8] S. Chari, C. S. Jutla, and A. Roy. Universally Composable Security Analysis of OAuth v2.0. *IACR Cryptology ePrint Archive*, 2011:526, 2011.

[9] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *USENIX Electronic Voting Workshop (EVT)*, 2008.

[10] E. Hanmer. OAuth 2.0 and the Road to Hell. `https://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell`. 2012.

[12] D. Jackson, M. Thomas, and L. I. Millett. *Software for Dependable Systems: Sufficient Evidence?* National Research Council, 2007.

[13] B. W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[16] D. L. Parnas. Information distribution aspects of design methodology. In *IFIP Congress (1)*, 1971, pages 339–344.

[17] S. Sun and K. Beznosov. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In *ACM Conference on Computer and Communications Security (CCS)*, 2012, pages 378–390.

[18] X. Xu, L. Niu, and B. Meng. Automatic verification of security properties of OAuth 2.0 protocol with cryptoverif in computational model. *Information Technology Journal*, 12(12):2273, 2013.