My research interests broadly span the areas of operating systems and dynamic/static program analysis with a focus on building secure and robust program execution environments.

Cyber attacks against personal and enterprise computers are rapidly evolving. Due to the rise of mobile computing, cloud computing, cyber-physical systems, and the “Internet of Things,” the diversity of victim computers targeted by sophisticated attacks has increased significantly. Such movement has created a new series of challenges that require highly practical solutions that most existing systems do not yet provide. In particular, many advanced security mechanisms cannot be used in production environments where even moderate performance impacts are undesirable. Also, many existing attack countermeasures designed to defend desktop/server computers are not applicable to protect small mobile and embedded devices because these devices have limited hardware capabilities and resource constraints. Furthermore, modern systems are complex and built with diverse hardware and software components, and thus they often do not allow generalized solutions but require customization. Although there have been tremendous research efforts to address general problems in software and systems security, it is still an elusive goal to practically defend such systems against constantly evolving attacks while satisfying various security and performance requirements.

The primary goal of my research is building secure and robust computer systems that practically address these problems through the synergetic integration of ideas and techniques in both operating systems and program analysis. In particular, my research seeks to obtain a clear understanding of program behaviors and then use the obtained knowledge to build secure and robust run-time systems. A significant part of my past research has focused on developing novel techniques to analyze programs on various platforms [1, 2, 3, 4, 5, 6] and building customized systems to securely execute these programs while satisfying diverse requirements [5, 7, 8]. Specifically, my research comprehensively analyzed performance and security issues in personal/enterprise computers during production [1, 2, 3], identified stealthy attacks on web browsers [4], and developed a new security tool to analyze and modify binary programs on mobile and embedded devices [6]. Further, my research proposed new system architectures to protect cloud computing infrastructures from malicious code thefts [7] and kernel malware attacks [8]. With the deep understanding of program execution on diverse platforms, my research also enlightened to protect cyber-physical systems from malicious attacks while satisfying hard resource and time constraints [5]. For all of these efforts, building practical systems considering both security and performance impacts has been at the center of my research.

Program Analysis in Production Environments

Among many different software problems, a major portion of my research focused on analyzing and resolving performance and security issues. One thing that these issues have in common is that it is difficult to prevent them completely while the program is being developed and tested. As a result, many performance issues escape development/testing environments and cause costs and frustration to users, and attackers exploit zero-day vulnerabilities to adversely affect the security of run-time systems. Learning from these lessons, my research focused on analyzing performance and security issues in production environments where root causes of problems can be identified on the spot. Specifically, my research has developed powerful platforms to analyze performance and security issues in production environments based on system event tracing [1, 2] and instrument binary programs for highly efficient performance monitoring [3].

IntroPerf [1] is a lightweight system that analyzes performance problems of binary programs using operating system (OS) tracers. Diagnosing performance problems in production environments is important because performance problems are run-time software issues and there may be unexpected external causes of the problems, such as device driver faults and denial-of-service attacks. However, it is challenging to diagnose performance problems in production environments since detailed program semantics is often not available and high analysis overhead is hardly affordable in such environments. In addition, it is important to look into all program components throughout multiple software layers with a system-wide scope since the root cause of a performance problem can be anywhere in the software stack. IntroPerf leverages operating system tracers that are widely available in commodity operating systems to solve this research problem. Such a tracer collects system stack traces at every OS event, which cover all software layers from applications to the kernel. IntroPerf automatically examines the contents and timing of the low-level system stack traces and generates fine-grained performance analysis results. The analysis results are further processed to provide a list of potential root causes with ranks, such that the information can be used to resolve the performance problems by performance analytics without too much effort. IntroPerf introduced a new paradigm of performance analysis in production environments showing that synergetic integration of operating system and program analysis techniques can produce novel and practical solutions to solve realistic problems.
**IntroSec** [2] is a low-cost security analysis system that identifies the causal paths of malicious attacks. The goal of this security analysis is to find the “entry point” of an attack and understand how it leads to the detected security anomaly. Similar to IntroPerf, IntroSec utilizes an operating system tracer to efficiently log OS events and collect system stack traces. The collected stack traces are partitioned into fine-grained execution units based on the detection of event handling loops in the monitored program to address the dependency explosion problem. The partitioning allows IntroSec to eliminate garbage traces that are unrelated to attack causality, resulting in significant reduction of log size without any accuracy loss. IntroSec showed that integration of ideas and techniques in operating systems and program analysis is not only useful in performance analysis, but also enables security systems to have highly practical solutions.

**PerfGuard** [3] is a system that automatically analyzes and instruments binary programs for highly efficient performance monitoring in production environments. IntroPerf loses a degree of efficiency and accuracy since it is based on the sampling of OS event traces, agnostic about the high-level semantics of analyzed programs. Motivated by this problem, PerfGuard proposes a new method that dynamically analyzes program execution to generate a performance profile, which is a “summary” of the program’s performance behaviors. Similar to IntroSec, PerfGuard partitions the program execution into a set of execution units based on event handling loop identification and temporal loop relationship. Identical or similar execution units are grouped together using a hierarchical clustering algorithm, and the time threshold of each unit group is determined. The performance profile is then used as a “hint” to create a set of performance monitoring code which can be inserted into the target program’s binary code. During the production run, the inserted code automatically monitors the performance of the program with high efficiency. When an unexpected performance delay is observed, the inserted code invokes a comprehensive performance diagnosis to help analytics resolve the issue. While the program is running without any performance issue, PerfGuard only induces very small overhead. PerfGuard demonstrated that program analysis and instrumentation techniques can retrofit run-time systems with more powerful capabilities to monitor/inspect program behaviors without sacrificing practicality.

**Other Research in Program Analysis**

In addition to program analysis in production environments, my research has also contributed to web security and privacy by discovering hidden malware behaviors in JavaScript programs.

**J-Force** [4] is a forced execution system that explores all possible execution paths of JavaScript programs to reveal hidden behaviors of stealthy JavaScript malware. J-Force performs analysis on concrete JavaScript program execution, thereby obfuscated programs can be analyzed. On top of that, it forces the program execution to go through different paths iteratively until all possible paths are explored. In case that the program generates an exception due to the forced execution, J-Force tolerates the exception to make sure that the program does not crash during the analysis. Such design allows J-Force to uncover stealthy attack behaviors in hidden execution paths that are enabled only when certain conditions are met. In our experiments with real-world JavaScript exploits and Chrome extensions, J-Force successfully disclosed the hidden code in most of the exploits and detected that a substantial number of the extensions inject advertisements into the web browser.

**Mobile/Embedded/Cyber-Physical Systems Security**

Protecting the security of non-traditional systems, such as mobile, embedded and various cyber-physical systems, is becoming more and more important as their popularity grows rapidly. These systems, however, have limited hardware capabilities and require defense mechanisms to meet certain conditions that existing state-of-the-art security solutions cannot easily satisfy. With my experiences in program analysis and building systems considering both performance and security, I have targeted to develop highly practical but radical solutions to protect such systems while meeting diverse constraints.

**JIGSAW** [5] is a security architecture that minimizes the attack surfaces of cyber-physical systems (CPS) built based on micro-controllers. Memory spaces of these systems play an important role since all software modules and hardware devices are physically mapped to a shared memory space through memory-mapped I/O. Due to the design, attackers can easily subvert the entire system by compromising a small software module in the systems’ firmware. However, conventional memory protection schemes, such as process and kernel memory isolation, are typically not available in these systems as micro-processors do not support virtual memory and these schemes have negative impacts on real-time constraints of the systems. JIGSAW proposes a lightweight view switching mechanism...
that enforces each process to have a minimized memory view using hardware-based memory isolation. JIGSAW automatically identifies a minimized memory view per process through static firmware analysis. The identified views are tailored by a clustering algorithm to be compatible with currently limited memory protection hardware. With such design, JIGSAW effectively protects the target system from various attacks in different attack vectors while preserving the responsiveness of the system. Our experiments with a commodity unmanned aerial vehicle showed that JIGSAW significantly reduces the attack surface of the system against realistic attacks without violating the real-time constraints. While working on this project, we identified several new memory corruption vulnerabilities in the firmware of the vehicle. All of these vulnerabilities have been confirmed and patched by the firmware developers with the help of our reports.

RevARM [6] is a binary instrumentation tool for mobile, embedded, and cyber-physical systems based on ARM processors. Existing techniques for ARM binary instrumentation are limited to have significantly high space and performance overhead or only applicable to certain types of ARM binary programs. RevARM is capable of analyzing and instrumenting ARM binary programs overcoming such limitations of existing techniques. Unlike many existing approaches that add complex control flows to the instrumented program for the execution of inserted code, RevARM simply stretches the program binary to create a space for the new code and preserves the original control flow of the program. This allows RevARM to be applicable to any ARM binary format, such as iOS and stripped firmware binaries, which existing tools cannot accurately instrument. Further, we solved a number of new ARM-specific challenges encountered while implementing RevARM and that previous work did not solve. We demonstrated the effectiveness of RevARM in various security applications including illegal API usage prevention in iOS, vulnerability patching in an embedded firmware, and run-time state monitoring of an unmanned aerial vehicle.

Cloud Computing Security

The demands of cloud-based services continue to grow as small devices (e.g., mobile and IoT devices) need to offload expensive computation to more capable virtual machines. My research has addressed security issues in cloud computing environments based on the deep understanding of operating systems and the virtualization technology.

CAFE [7] protects the confidentiality of application binary code in cloud computing infrastructures. Major cloud computing infrastructures, such as Amazon Web Services and Microsoft Azure, provide marketplaces where developers can upload and retail cloud applications, similar to mobile app marketplaces (e.g., the App Store and Google Play). While this form of distribution simplifies the deployment of cloud applications, it allows malicious cloud users to access the application binary code for the purpose of piracy and reverse engineering. Since cloud users typically own an entire virtual machine with privileged permissions in a cloud infrastructure, technically they have no restriction on accessing and replicating application binaries installed in the VM. CAFE addresses this problem by providing a new execution environment where confidentiality of sensitive binary code can be protected throughout the entire lifespan of the application. The key idea of CAFE is to securely distribute sensitive binary code through encryption and execute the code in a container in isolation from the user virtual machine. In the design of CAFE, the hypervisor dynamically receives protected binary code from the cloud through a cryptographically secure channel, and it securely loads the binary into an isolated memory using hardware-based memory isolation. The integrity of the hypervisor is remotely attested using a hardware module, and thus the binary code can be protected even when the hypervisor is compromised. The effectiveness of CAFE is tested with popular cloud applications, and our benchmark results showed that CAFE only has small performance overhead while providing strong code confidentiality. Along with other related works in trusted computing, CAFE stimulated the development of recent hardware-based enclaving technology, such as Intel Software Guard Extensions (SGX).

KMAG [8] is a virtualization-based system that efficiently detects kernel malware attacks through kernel data access profiling. Most kernel malware attacks, such as kernel rootkits, tamper with core kernel data objects and there exist unique kernel data access patterns. Such data access patterns can be used to detect at kernel malware attacks with the same or similar patterns at run-time. Previous work, DataGene, profiled and detected kernel malware using this approach through dynamic analysis of OS kernel execution at the binary level. However, binary-level execution monitoring is known to have large run-time overhead, and hence cannot be used in production environments. KMAG is a hypervisor system that efficiently monitors kernel data accesses for data-centric kernel malware detection. KMAG leverages the hardware-assisted memory virtualization technology to transparently monitor kernel memory accesses with low overhead. In particular, it marks the pages containing kernel data as non-accessible in the nested page tables of the virtual machine and examines accesses to the pages when the hardware detects the page faults. Memory locations of dynamic kernel data objects are traced at run-time through the instrumentation of kernel
memory allocation and deallocation functions. Our experimental results with kernel rootkits showed that KMAG can effectively profile and detect kernel malware attacks with high efficiency in virtualized execution environments. KMAG has been delivered and demonstrated to our industrial partner Telcordia as part of the ARMY-sponsored DEFIANT project.

Future Research

As the computing environment undergoes a rapid change, many new security issues arise and the requirements of underlying systems are ever diversifying. Based on my past research experience, I am going to continue to build practical systems that address these challenges.

Time-Sensitive Attack Countermeasures. A long line of existing anti-malware research has focused on modeling benign system behaviors or characterizing known malware activities based on control and data flows. For example, control flow integrity (CFI) schemes dynamically monitor the control flow of a program to detect any unexpected program behavior caused by an attack. Similarly, data flow integrity (DFI) schemes check if there is an unexpected data flow in the program execution. However, the effectiveness of these approaches significantly relies on the granularity of control and data flow monitoring. Fine-grained CFI and DFI schemes with high precision have too much run-time overhead making them undeployable in production environments. In addition, attackers have already found ways to circumvent coarse-grained CFI and DFI schemes. With such limitation, systems protected by existing CFI and DFI techniques are prone to highly stealthy attacks, such as an advanced persistent threat (APT). I believe that time-sensitive attack countermeasures can be a great solution to overcome such limitations. Recent advances in computer processors, such as Intel Processor Trace (PT), have made it possible to trace entire or part of system execution with minimal performance overhead. These tracing mechanisms provide fine-grained information of run-time program execution, and more importantly, how much time is spent by small program units. Time information can be used to detect highly elusive attacks since it is difficult for attackers to tamper with hardware-based time measurement. Even if possible, such tampering attempts will be reflected in the time information since any deviation from time specification may indicate an attack payload. Starting with simple embedded devices, I would like to use this time-sensitive approach to secure more complex systems against advanced attacks, such as mobile and cyber-physical systems. In the long term, I would like to understand the correlation between the accuracy of time-sensitive attack detection and hardware cache behaviors in order to increase the effectiveness of the approach and employ it to protect more complex systems.

SITL System Analysis Platforms. My past research has shown that precise and deep understanding of program execution and run-time systems through program analysis is a key to building robust and secure computer systems. It is, however, still a challenging problem to obtain highly clear and informative views of dynamic program execution across all software layers in diverse platforms, although such views allow researchers to locate new problems to solve and develop novel ideas and techniques. In particular, I am interested in developing new analysis platforms for embedded and cyber-physical systems with hard resource constraints. Static analysis techniques can be used to analyze these systems, but they do not allow investigating non-deterministic attributes of programs, such as various inputs and outputs, hardware events, and intermediate program state. Further, building a generic dynamic analysis platform for such systems is difficult since they are equipped with diverse peripheral devices and each of these devices needs to be virtualized on an ad-hoc basis. My idea to solve this problem is to build a whole-system dynamic analysis platform that integrates a hardware virtualization system with a software in the loop (SITL) simulator. An SITL simulator allows running the system to analyze without any hardware. The inputs generated by the simulator will be transparently converted into low-level hardware inputs and passed to the hardware virtualization system through a fixed number of uniformly virtualized peripherals. Program components that react to the hardware inputs will be dynamically analyzed with a system-wide scope through a generic analysis interface provided by the analysis platform. Specifically, I am working on a new analysis platform for unmanned aerial vehicles with an abstraction to interconnect between an ARM hardware virtualization system (QEMU) and a flight control simulator (ArduPilot). Such a virtualized analysis platform allows advanced software testing techniques to automatically analyze program execution across software layers for security and reliability. For example, fuzz testing can be applied on the integrated simulator to identify and analyze new security vulnerabilities that attackers may exploit by spoofing hardware inputs, such as GPS signals. Another example is analyzing the impact on energy consumption due to critical security attacks that target devices with energy constraints. After the research on unmanned aerial vehicles, my plan is to apply the approach to other embedded and cyber-physical systems, such as smart grid and autonomous automobile systems.
References


