Today’s computer systems are large and complex. There have been tremendous research efforts to fully defend these systems, but it is still an elusive goal. Introducing radical design changes for security is not practical, or adding a small security enhancing component easily breaks the critical functionality of a system. Even if these issues are resolved, attackers are ever evolving, and they always find a vulnerability from tiny little corner cases.

My research focuses on building secure systems. In particular, my approach can be characterized by a practical prospect with a strong understanding on both attacks and computer systems. With the deep knowledge on attacks, my research comprehensively identified the root cause of underlying vulnerabilities, and built security solutions completely eliminating causes [1, 2]. Further, my research analyzed emerging security issues that are newly introduced in modern system’s design and implementation [6, 9, 12], which enlightened to formulate the principle of secure system designs. I have also proposed attack countermeasures [3, 7, 8], which protects the system from being compromised. In addition, with the thorough understanding of computer systems, many of these techniques are carefully designed to scale to large and complex systems including Chrome, Firefox, the Linux kernel, and the Android OS, and already demonstrated its effectiveness.

My research results are recognized for their highly practical impacts, as noted by the Internet Defense Prize (awarded by Facebook and USENIX) and the best applied research paper (awarded by CSAW). Moreover, my research software has been deployed as part of the security infrastructure in Google and Mozilla, and widely covered in popular medias including ZDNet, Science 2.0, Phys.org, etc. In addition, my research results discovered and accordingly fixed more than 100 highly critical security vulnerabilities in various software, such as the Linux kernel, Chrome, Firefox, Safari, and etc., and received several vulnerability discovery gifts and awards in recognition of helping to secure their products. Further, undertaking these security problems brought pleasant interdisciplinary collaboration experiences beyond the security research domain: compilers [1, 2]; systems [3, 9]; programming languages [8]; networks [11]; and data mining [4].

### Vulnerability Elimination

Many system components and network applications are written in unsafe programming languages that are prone to memory corruption vulnerabilities. To combat with countless catastrophes from these vulnerabilities, there have been much defense research efforts as well. However, these were limited largely because their techniques focus on certain negative side effects from vulnerabilities: there have been many unfortunate cases that security holes in these mitigation solutions are later uncovered, which significantly thwarts the security of underlying systems. Learning from these unfortunate lessons, my research focused on building a protection system which completely eliminates a root cause of vulnerabilities. Specifically, I have targeted two popular and emerging vulnerabilities, use-after-free and bad-casting, each of which is addressed by DangNull [1] and CAVER [2], respectively. Since DangNull and CAVER directly fix the origin of such issues, it does not leave any security holes that attackers may attempt to abuse in the future. The key challenge here is that today’s critical security vulnerabilities, such as use-after-free and bad-casting, are very complex: modern commodity systems are developed under object-oriented or event-driven designs in which these vulnerabilities are deeply involved.

DangNull and CAVER are recognized for their highly practical impacts from both academic and industry communities: Facebook and USENIX awarded the Internet Defense Prize, and CSAW awarded the best applied research paper; Google and Mozilla deployed DangNull and CAVER, respectively, in their development infrastructures;

DangNull [1] is a system that detects use-after-free (i.e., a temporal memory safety violations) during runtime. The key observation of this system is that the root cause of these violations is in that pointers are not nullified after the target object is freed. Based on this observation, DangNull automatically traces the object’s relationships via pointers and automatically nullifies all pointers when the target object is freed. With this unique design decision, DangNull offers several benefits. First, DangNull addresses the root cause of temporal memory safety violations. It does not rely on the side effects of violations, which can vary and may be masked by attacks. Second, DangNull checks object relationship information using runtime object range analysis on pointers, and thus is able to keep track of pointer semantics more robustly even in complex and large scale software. Lastly, DangNull does not require numerous explicit sanity checks on memory accesses because it can detect a violation with implicit exception handling, and thus its detection capabilities only incur moderate performance overhead. We implemented DangNull and applied it to the Chromium browser, which safely prevented all seven real-world use-after-free exploits that we evaluated.

CAVER [2] is a system that detects bad-casting during runtime. Type casting, which converts one type of an object to another, plays an essential role in enabling polymorphism in C++ because it allows a program to utilize certain general or specific implementations in the class hierarchies. However, if not correctly used, it may return unsafe and incorrectly casted values, leading to so-called bad-casting vulnerabilities. Bad-casting vulnerabilities are security critical not only because it is specified as undefined behavior that compilers cannot guarantee the correctness of a program execution, but also because it allows an attacker to corrupt memory beyond the true boundary of objects. The key observation of CAVER is that the root cause of these violations is in that objects are casted to other objects semantically mismatching. Therefore, CAVER
automatically instruments the program to verify type castings at runtime during the compilation phase with a new metadata scheme that efficiently keeps track of rich type information. In order to be compatible with the C++ standards and thus support all legacy programs, CAVer uses a disjoint metadata scheme (i.e., the reference to an object’s metadata is stored outside the object). Armed with this unique design decision, CAVer was able to protect all type castings of both polymorphic and non-polymorphic classes that none of previous work was able to handle. We implemented CAVer and applied to the Chrome browser and the Firefox browser. While a vulnerability finding is not a primary goal of CAVer, it identified eleven previously unknown bad-casting vulnerabilities in Firefox and GNU libc++ during evaluation, all of which have been accordingly fixed by corresponding vendors.

Vulnerability Discovery and Analysis

Discovering and analyzing new vulnerabilities is an important security research asset. Newly discovered vulnerabilities allow vendors to immediately close the critical attack vector, and a comprehensive vulnerability analysis enlightens how to securely design similar systems in the future.

HyLeak [6] analyzes critical security vulnerabilities newly introduced in commodity systems. HyLeak analyzed the information leakage attacks due to the deterministic data structure implementations. As the code relating to this vulnerability is performance critical, we have explored several possible ways to mitigate this vulnerability which also meets performance requirements. To demonstrate the feasibility of this vulnerability, we have discovered and attacked one class of such a vulnerability in WebKit (the foundation of the Safari browser), which has been fixed by the WebKit development community.

Juxta [9] is a tool to discover semantic vulnerabilities. The key idea behind Juxta is to compare and contrast multiple existing implementations that should obey implicit high-level semantics. For example, the implementation of open() at the file system layer expects to handle an out-of-space error from the disk in all file systems. Using symbolic executions, Juxta discovered 118 previously unknown semantic vulnerabilities in the Linux kernel’s file systems, many of which has been fixed based on our finding.

Jailios [12] discovered and analyzed yet another way to jailbreak an Apple’s iPhone (i.e., privilege escalation attacks). This new jailbreaking mechanism involves unpatched vulnerabilities, which do not critical harm if solely abused and thus left unpatched. We demonstrated to chain these unpatched vulnerabilities to eventually achieve a jailbreak, which alerts the security risks of having such unpatched vulnerabilities.

Attack Countermeasures

In order to fully compromise a system, adversaries have to successfully launch a sequence of multiple attacks. My research aims at rendering a certain critical attack futile, thereby protecting the system from being compromised.

Morula [3] is a countermeasure against new security threats in Android. We discovered that Zygote, an Android design for speeding up application launches, is at odds with security because all applications processes are created with largely identical memory layouts. This violates critical security guarantees of each process, and we further demonstrated that attackers are capable of executing return-oriented programming on Android using real applications, such as the Chrome browser and the VLC media player. As a countermeasure of this attack, we built Morula which fortifies Address Space Layout Randomization (ASLR) of Android by redesigning Android’s process creation mechanisms. The key idea of Morula is to adaptively maintain a pool of Zygote processes with distinct memory layouts (i.e., Morula processes): when an app is about to start, a unique Morula instance is available as a one-time and pre-initialized process template in which the app can be loaded.

AslrGuard [7] aims at rendering information leak attacks useless in deriving code addresses. A general prerequisite for a code reuse attack is that the attacker needs to locate code gadgets that perform the desired operations and then direct the control flow of a vulnerable application to those gadgets. The main idea behind AslrGuard is to render leak of data pointer useless in deriving code address by separating code and data, provide a secure storage for code pointers, and encode the code pointers when they are treated as data. Therefore, AslrGuard makes either code pointer leaks or render their leaks harmless. That is, AslrGuard makes it impossible to overwrite code pointers with values that point to or will hijack the control flow to a desired address when the code pointers are dereferenced.

Kenali [8] is a principled defense mechanism against memory corruption based privilege escalation attacks. Kenali leverages data-flow integrity (DFI) to enforce security invariants of the kernel access control system. Further, in order for our
protection mechanism to be practical, we developed two new techniques: 1) we devised a systematic and formalized program analysis technique to discover security sensitive data regions that scales to Linux-like, large commodity software, and further proved that our approach is a sound solver for this problem; and 2) we presented a new data-flow integrity enforcement technique that has lower performance overhead, and provides identical security guarantees as previous approach. We have implemented a prototype of Kenali for the ARM64 Linux kernel on an Android device. We also evaluated the security of Kenali with various types of control-data and non-control-data attacks that reflect the real kernel privilege escalation attacks used for rooting Android devices, and showed Kenali can effectively mitigate all of these attacks.

Other Research

LocPriv [4] aims at protecting location privacy of users based on historical data of users. We designed new distance measure to compute semantic differences of locations, and leveraged mined information to further compute the secure area for location based services. TrackMeOrNot [10] is a personalized anti-tracking mechanism in Web browsing. We redesigned the browsing context management in Chrome so that users can selectively disclose their browsing activities to vendors based on their specified privacy demands. Expector [11] is a system that automatically inspects and identifies browser extensions that injects ads, and then classifies these ads as malicious or benign based on their behaviors. Binob+ [5] is an obfuscation tool that transforms a program binary to prevent a reverse engineering while preserving the outer execution behavior.

Future Research

As the computing environment undergoes a rapid change, many new security issues arise and thus underlying system’s security requirement is ever changing. Based on my past research experience, I plan to continue to build secure systems.

Beyond Memory Corruption Attacks. I envision computing systems would encounter many more attacks relating semantic or logical vulnerabilities. These cannot be easily described as the typical safety property used for detecting memory corruption vulnerabilities, but it requires to handle more challenging problems: capturing hidden semantics introduced by its design or implementation. I have already begun to investigate some related problems, which either analyzed or identified concrete instances of such vulnerabilities: HTLeak [6] analyzed information leak vulnerabilities due to the deterministic data structure design; and Juxtap [9] automatically inferred high-level semantics by comparing multiple implementations using symbolic executions. Based on these experiences, I am seeking to step forward to build a system proactively protecting from such attacks. I believe the key challenge to address this problem is to capture intuitive primitive semantics. This will be similar to my previous research on vulnerability free systems. Dang Null [1] and Caver [2], in a sense that it identified the origin of vulnerabilities, but this new direction would require more deep understanding on program patterns and its semantics.

In the long term, I would like to explore the following potential research directions. One is to propagate key semantics to the whole system to systematically construct the protection system. For example, using program analysis techniques, a function’s return value can be the anchor to discover hidden semantics of a system. The other potential direction would be to apply machine learning techniques. Since programs are built under the common design pattern or programming practice, statistically discovering such patterns may enlighten the way to construct unique and effective protection systems.

Trusted Computing Platforms. Trusted computing is the foundational technology that ensures confidentiality and integrity in modern computing. In response to the high demands on such platforms, Intel recently released Secure Guard Extensions (SGX) that are embedded in their commodity CPUs. I believe this would spark a major paradigm shift in the security research because now we finally have the trusted anchor that is implemented solely with hardware. While securing hardware components would be the important research subject as well, I would like to first explore to secure software stacks on this platform. In particular, computing based on SGX would pose a distinctive adversary model in that none of underlying software artifacts (e.g., operating systems, or any runtime libraries) can be trusted but itself. Continuing my research direction in building runtime protection systems both in operating systems [3, 8] and application layers [1, 2, 7], I believe I am ready to tackle these problems under this new challenging problem setting. I have already begun to bring back traditional security mechanisms for SGX, such as address space layout randomization and data execution prevention, under this restricted setting.

In the long term, I would like to further explore the possibility of enabling practical security applications on trusted computing platforms. One is to enable a restricted version of private information retrieval (PIR). With the precise understanding on adversarial model and proper secure enforcement on the system, this new application would enable a practical version of PIR in that even application owners cannot arbitrary control the operational semantics of programs to learn...
runtime computations. In addition to PIR applications, I want to investigate other potential security use cases as well based on this trusted computing platform: deploying to biomedical devices; designing privacy-aware machine learning or recommendation systems; or identifying theoretical possibilities of security or privacy leakage attacks.

References


