AFRL-OSR-VA-TR-2015-0119

HARDWARE, LANGUAGES, AND ARCHITECTURES FOR DEFENSE AGAINST HOSTILE OPE

David Wagner
REGENTS OF THE UNIVERSITY OF CALIFORNIA THE

05/21/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTC
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 13, 20	Final	August 2009 - July 2014

4. TITLE AND SUBTITLE
Final Performance Report: Hardware, Languages, and Architectures for Defense Against Hostile Operating Systems

5a. CONTRACT NUMBER
FA9550-09-1-0539

5b. GRANT NUMBER
AFOSR FA9550-09-1-0539

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
David Wagner

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of California, Berkeley
Computer Science Division, Berkeley, CA 94720-1776

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR

10. SPONSOR/MONITOR'S ACRONYM(S)
MURI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Defending Against Hostile Operating Systems research project focused on building systems that will remain secure even when the operating system is compromised or hostile. It was a collaborative effort among researchers from Harvard, Stony Brook, U.C. Berkeley, University of Illinois at Urbana-Champaign, and the University of Virginia. The project studied this problem from a number of angles, including operating system design, hypervisor enforcement, static analysis, and hardware design, and on a number of platforms, including desktop systems, servers, and mobile systems. The team developed numerous mechanisms and techniques for addressing this problem.

15. SUBJECT TERMS
computer security, operating systems

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			David Wagner
U	U	U	UU	24	19b. TELEPHONE NUMBER *(Include area code)* 510-642-2758

Reset

Final Performance Report: Hardware, Languages, and Architectures for Defense Against Hostile Operating Systems

David Wagner
University of California, Berkeley
Computer Science Division, Berkeley, CA 94720-1776

AFOSR MURI award number FA9550-09-1-0539

May 14, 2015

Abstract

The Defending Against Hostile Operating Systems research project focused on building systems that will remain secure even when the operating system is compromised or hostile. It was a collaborative effort among researchers from Harvard, Stony Brook, U.C. Berkeley, University of Illinois at Urbana-Champaign, and the University of Virginia. The project studied this problem from a number of angles, including operating system design, hypervisor enforcement, static analysis, and hardware design, and on a number of platforms, including desktop systems, servers, and mobile systems. The team developed numerous mechanisms and techniques for addressing this problem.

1 Objectives

The objective of this research project was to develop technology and scientific foundations to limit the harm that a hostile operating system can cause. The project explored whether it is feasible to impose useful limits on the damage caused by a hostile operating system, and if so, to identify the space of plausible techniques, assess their applicability, devise useful solutions and technology, and evaluate their effectiveness. More generally, the program also sought to jumpstart exploratory scientific research in a new area of computer security, demonstrate its promise, grow the portfolio of work in this area, and start to develop a longer-range research vision and agenda for the community.

2 Scientific Approach

Before this project, there was very little prior work on building secure systems when the operating system cannot be trusted; indeed, it is fair to say that conventional wisdom would have it that this is typically infeasible. The team is engaged in work showing that, in fact, this goal is achievable in some cases.

To grow our understanding of this emerging area, the team sought to develop foundations for the field and study the problem from a number of different—but complementary—

vantage points. First, we developed secure hardware designs, to provide a root of trust and a way to limit the damage done by a hostile OS. Second, building upon powerful machinery from programming languages and formal methods, we developed methods to establish trust in code and control what it can do by using transforming and rewriting—and, in particular, to control the harm that a compromised or malicious OS can do. Third, to provide a principled foundation for transformation and isolation, we built a platform called the Secure Virtual Architecture, which provides a powerful basis for running applications on an untrusted substrate without sacrificing security or performance. Fourth, to gain confidence in these approaches, we developed methods for formally verifying the correctness of these mechanisms. Finally, we studied systems-level approaches to build whole systems that are trustworthy, even when the operating systems on some or all of the machines may be malicious or untrusted. These mechanisms are complementary and, when used in concert, provide greater protection than any one alone could offer.

The team drew its technical toolset from language-based security, virtualization and paravirtualization, binary rewriting, formal methods, operating system design, hardware architecture design, and secure computation using multi-party cryptographic protocols. While the original focus of this project was solely on defending against a hostile operating system, in the final phase of the project we cautiously expanded the scope of the project to include (to a limited extent) work on reducing or eliminating trust in other parts of the programming platform: for instance, encouraged by our progress in defending against a hostile operating system, we have also begun to explore methods of defending against malicious hardware as well.

3 Personnel Supported

Faculty (8). The team was led by the following faculty: Vikram Adve (UIUC), Krste Asanovi (UC Berkeley), David Evans (U. Virginia), Sam King (UIUC), Greg Morrisett (Harvard), R. Sekar (Stony Brook), Dawn Song (UC Berkeley), David Wagner (UC Berkeley). Wagner was the principal investigator.

Postdoctoral scholars (14). UCB: Domagoj Babic, Matko Botincan, Vijay D'Silva, Lorenzo Martignoni, Stephen McCamant, Reza Moazzezi, Matthias Payer, William Robertson, Mohit Tiwari, Chao Zhang. Harvard: Adam Chlipala, Randy Pollack. Illinois: Anthony Cozzie, Shuo Tang.

Graduate students (62). UCB: Devdatta Akhawe, Jethro Beekman, Matko Botincan, Sarah Bird, Nicholas Carlini, Erika Chin, Henry Cook, Thurston Dang, Mattia Fazzini, Adrienne Felt, Warren He, Benjamin Hindman, Sakshi Jain, Mobin Javed, Noah Johnson, Yunsup Lee, Yaping Li, Eric Love, Martin Maas, Michael McCoyd, Mitar Milutinovic, Rebecca Pottenger, Justin Samuel, Richard Shin, Emil Stefanov, Cynthia Sturton, Zhangxi Tan, Matthias Vallentin, Andrew Waterman, Richard Xia. Harvard: Paul Govereau, Gregory Malecha. UVa: Ivan Alagentchev, Longze Chen, Simon Sibomana, Weilin Xu, Samee Zahur Al Islam, Yuchen Zhou. Illinois: Arushi Aggarwal, Sean Bartell, Anthony Cozzie, Joshua Cranmer, John Criswell, Nathan Dautenhahn, Matthew Finnicum, Matt Hicks, Theodoros Kasampalis, Andrew Lenharth, Haohui Mai, Anh Nguyen, Corbin Souffrant,

Shuo Tang, Hui Xue. Stony Brook: Anupama Chandwani, Niranjan Hasabnis, Markov Iaroslav, Riccardo Pelizzi, Rui Qiao, Alireza Saberi, Wai-Kit Sze, Laszlo Szekeres, Tung Tran, Mingwei Zhang.

4 Accomplishments

Generally speaking, the project has shown how a range of protections can be applied efficiently to legacy applications, while demonstrating more powerful protections for applications where we have access to source code or an opportunity to redesign the application architecture.

4.1 OS-level defenses

We developed a new system architecture that enables trustworthy execution of server applications, even if the operating system is malicious or compromised. We built a prototype implementation of our system, called PrivBSD, that provides two-way isolation between the application and OS. The PrivBSD architecture involves a trusted core (approximately 18K lines of code) that is responsible for providing memory isolation; building on this core, we then ensure that the rest of the operating system cannot tamper with application execution by using end-to-end cryptography, encrypted filesystems, and other mechanisms (approximately 12K lines of trusted application code). Using PrivBSD, we have demonstrated how to protect a simple web server against malicious OS drivers, malicious OS extensions, and other malicious OS code, by using end-to-end cryptography, library interposition, and other techniques.

To provide secure web browsing, we designed and built the IBOS operating system and browser. IBOS is designed to provide high assurance web browsing with a tiny trusted computing base. We built a mathematical model of important elements of IBOS and formally verified that it correctly enforces important security properties, such as the same-origin policy, isolation between origins, and a trustworthy user interface. Thus, IBOS can provide strong security for client devices that are dedicated to running only web applications. Based on the principles learned from this experience, we later built a new OS and runtime system for mobile devices to enable apps and mobile browsers to run with mathematically provable security policy enforcement (see discussion of ExpressOS below).

We have designed a client-server architecture for construction of secure distributed systems, with resilience to hostile, malicious, or compromised operating systems. Our architecture combines trustworthy execution of applications on the server, using PrivBSD, with a client-side user interface that is implemented using HTML-based technology and executed on client machines using IBOS. This architecture reduces the size of the trusted computing base by several orders of magnitude and enables trustworthy deployment and use of distributed systems, without relying upon complex legacy operating systems to be trustworthy.

We designed and implemented Cloud Terminal, a new system for securing cloud-backed user-facing applications. Today's web applications and desktop applications offer no protection against client-side malware. If the user's client is infected with malware, then the

malware can steal credentials, observe all confidential information accessible to the application, modify its state and information, forge transactions, and defeat all application security goals. For instance, if the client's operating system is hostile or malicious, all security is lost. Cloud Terminal is designed to address this threat model and allow secure execution of end-user applications, even on machines whose operating system or other software is compromised or malicious.

Cloud Terminal works has two pieces: (a) a secure thin client that runs on the client machine and renders the user interface, as directed by the server, and (b) a server in the cloud that contains all application logic as well as a rendering engine. The thin client runs in a secure environment on the user's machine, isolated from everything else on that machine using a hypervisor or other secure execution environment. For assurance, it is constructed to be extremely small (23 KLOC) and simple (all that it does is display screen images provided by the server and forward user inputs to the server). Effectively, the thin client is a dumb terminal. All of the application logic is implemented in the server, and the server also renders the graphical user interface, producing a bitmap image that is sent to the thin client to be displayed on the screen. This architecture allows all of the application logic and state to live on the server, where it can be protected from attack and run on professionally administered machines. Our work suggests that Cloud Terminal is a powerful, deployable, pragmatic solution to client-side malware. With the upcoming deployment of Intel's SGX technology, the Cloud Terminal architecture may become even more appealing as a way to deal with compromised or hostile desktop operating systems.

We built ExpressOS, a new operating system that is designed to provide high assurance and reduce the need to trust the operating system. The core of ExpressOS is formally verified to provide seven security invariants, even if the rest of the OS or applications are compromised or hostile. The seven security invariants ensure memory isolation, secure storage on the filesystem, secure access control to files, encryption of all memory before it is swapped out to persistent storage, integrity for all code and data loaded from the filesystem, UI isolation, and secure IPC. We used contract-based programming, the Dafny verifier, and automatic theorem provers to verify all these properties. This provides a basis for trust in the system, even when the OS is under attack, and helps defend against vulnerabilities in the OS kernel and in privileged system services. ExpressOS requires only a modest annotation burden (annotations were about 3% of code of the kernel), modest performance overhead (about 15%, compared to Android), and supports legacy applications (a web browser and many Android apps).

4.2 Secure Virtual Architecture

The Secure Virtual Architecture (SVA) is a fundamental building block for our project. SVA provides a solid foundation for defending against a malicious OS. It establishes an instruction set for a virtual processor that is designed to facilitate binary analysis, binary translation, virtualization, and enforcement of security properties, through a form of paravirtualization. We have shown how to use the Secure Virtual Architecture to protect against a malicious OS. Effectively, this shows how clever use of compiler instrumentation and system design can be a powerful way to enforce security properties on code, even when that code might be untrusted or hostile.

Using SVA, we have developed techniques to ensure that the lowest-level interactions between an OS kernel and the underlying hardware cannot cause memory safety violations within the kernel. Such interactions include saving and restoring processor state, handling interrupts, and delivering and returning from signals. With these techniques, we have demonstrated that SVA can enforce comprehensive safety guarantees for almost the entire Linux 2.4.22 kernel, including both core kernel components and device drivers, while leaving all policies (process scheduling, page table management, I/O handling, interrupt handling, etc.) to the kernel.

SVA uses the popular LLVM instruction set, for which there is a significant amount of supporting open-source infrastructure. We have demonstrated how to validate optimizations to LLVM code, which enables provably correct optimization of SVA code. In particular, this research makes it possible to automatically prove the correctness of the output of the LLVM optimizer, helping to eliminate the optimizer from the trusted computing base. This enables SVA to benefit from the full suite of compiler optimizations found in existing LLVM compilers, without needing to rely upon the correctness of the optimizer and with the ability to survive a malicious compiler optimizer.

Building on this, we've built a system for running applications on an untrusted operating system, called Virtual Ghost. Past work (e.g., Overshadow, S3E, InkTag) used hardware page-table protection to protect the application, preventing the OS from reading or writing the application's memory. Virtual Ghost uses compiler instrumentation instead of page-table protection, enabling it to achieve better performance than those prior systems, while still providing strong security. In particular, Virtual Ghost is built using the SVA architecture. Virtual Ghost enables an application to choose what parts of the data and computation are secured, and to control the choice of encryption and hashing to manage the tradeoff between security and overhead. This demonstrates the power of compiler instrumentation and the SVA approach.

4.3 Formal methods

Formal verification and formal methods are a powerful way to ensure that security mechanisms are correctly implemented and achieve our desired security goals. To support the goals of this project, we have performed extensive work on applying formal methods to verify binary instrumentation and the SVA.

To provide a foundation for verification, we developed accurate, mathematical models of the semantics of machine instructions, especially the x86 instruction set. This is a key piece of technology for enabling high-assurance binary translation and for formal verification of tools for binary analysis and binary translation. This is a daunting task: for instance, the x86 architecture supports over 1100 instructions described in 1500-page manual. In addition to the sheer effort involved, instruction semantics specification is error-prone due to ambiguities inherent in textual descriptions. However, we developed methods to overcome these challenges.

In one line of work, we have shown how to build accurate, semi-formal models of instructions by taking advantage of the machine descriptions (MD) used in contemporary compilers such as GCC and LLVM. These machine descriptions reflect an enormous amount of effort by compiler developers, and have been extensively tested. They consist of rules to translate

snippets of RTL into instructions for the target machine. (RTL is a machine-independent low-level representation into which a compiler first translates high-level code.) Our key insight is that it is possible to use MD rules in reverse. We have developed techniques that take these rules as input, and automatically generate code that maps binary instructions to RTL. This is a challenge since a compiler must verify a number of rule-specific conditions before determining its applicability, these conditions can be expressed in C code. So far, our implementation can handle about 50% of x86 instructions and are able to process non-trivial SPEC INT benchmarks. In addition to enabling the development of an architecture-neutral instrumentation framework, our approach can take advantage of the optimizations implemented within compiler back-ends, thereby significantly boosting the performance of instrumented code.

In another complementary line of work, we have built a formal model for the integer portion of the x86-32 instruction set architecture, in Coq. This is a key step to facilitate formal verification of programs, at the assembly language level, which in turn will be an important tool for building secure systems in the presence of an untrusted operating system or other platform code. We have carefully validated our model of the x86-32 decoder against a range of torture tests and proven that key properties hold on the decoder. This approach enables us to build formal, high-assurance, very precise models of the processor instruction set. We have also integrated support for most of the floating point, SSE, and MMX instructions: sufficient to run the SPEC benchmarks through the model when compiled with GCC, LLVM, and ICC, which suggests taht we have achieved reasonable coverage of x86-32 instructions.

This foundational work has enabled us to build a tool that generates a verifier for Google's SFI policy for Native Client (NaCl), found in the Chrome web browser. The resulting verifier, RockSalt, is smaller and faster than Google's verifier: Google's verifier is 600 lines of C code, whereas RockSalt is only 100 lines of C code. Moreover, we have proven RockSalt correct using our formal model of the x86 instruction set. The proofs themselves required significant manual effort (they comprise over 10,000 lines of heavily commented Coq code), but the benefit is that RockSalt only need be verified once; the runtime performance remains excellent.

We developed methods to reduce the proof burden for building certified compilers by roughly an order of magnitude. We also designed a system, called Bedrock, which provides a way to write low-level, OS-like code (e.g., schedulers) and reason about correctness of both the scheduler and the application in a modular fashion.

We have also developed a number of techniques for improving formal verification of systems code. These advances are summarized below.

We have developed new algorithms for bounded model checking (BMC) of software. Software BMC is a powerful method for finding non-obvious bugs in software. In general, software BMC has the advantage that one can verify the code directly (rather than a manually built model); however, it does not scale to large programs. We developed new algorithms to improve the scalability of software BMC. In particular, we developed techniques for compositional BMC, so that the task of verifying a large software program can be broken up into multiple smaller verification problems. We implemented our methods in a tool called Blitz and showed that Blitz can find security vulnerabilities in programs as large as 100 KLOC, and we found new vulnerabilities in critical code used in the Internet

routing infrastructure.

We developed new methods for applying model checking to verify system code. Model checking is a powerful method for formal verification, but it is notoriously subject to the state-space explosion problem: it does not scale well to systems with a large state-space, such as systems with non-trivial data structures. Some researchers have explored bounded model checking (BMC) as a solution to these scaling problems, but BMC has the shortcoming that it can find bugs but typically cannot verify the absence of bugs. This is because BMC amounts to imposing a bound k and considering all executions of the system that involve at most k steps of execution; thus, BMC can find all bugs that are reachable within at most k steps, but cannot find any bugs that require more than k steps to trigger. Another approach to deal with the state-space explosion problem is to artificially restrict the size of data structures. For instance, if the system has a large table, we might replace it with a reduced-size table that contains only one or two rows. Practitioners report that this is often effective at finding bugs, but it typically cannot verify the absence of bugs, because of the possibility of bugs that only appear for large data structure sizes. There are narrow situations where one can prove a so-called "small-world theorem", where this approach can verify absence of bugs (because any bug that appears in a large data structure can also be found with a scaled-down data structure), but most systems do not satisfy the preconditions of those theorems.

In our work, we show how to combine these two methods, and in a way that lets us not only find bugs but also verify the absence of bugs. Our approach, S2W (small and short worlds), combines the small-world heuristic (artificially scaling down data structure sizes) with the short-world heuristic (bounded model checking), and then augments this with a method to automatically adjust the parameters of these methods so that the technique will be sound. In particular, we prove a soundness result: if S2W says there are no bugs, then there are none to be found. We applied S2W to verification of several security properties of operating systems and hypervisors, demonstrating its value.

One of the main shortcomings of traditional model checking is that it requires a formal mathematical model of the system. Once we verify that the model is secure, it becomes natural to wonder: does the model accurately capture all possible behaviors of the implementation? This is the model validation problem. We have developed new methods for model validation, using a combination of symbolic execution and SMT solvers. We have demonstrated our techniques on models derived from several software systems, including parts of the XMHF hypervisor, the Bochs x86 emulator, and other examples. One of the benefits of this approach is that it enables us to transfer theorems about the model to assurance about the implementation. Another benefit is that it provides automated methods to support verification under code maintenance: in practice, software evolves; our method provides an automated way to detect when changes in the software require updates to the model.

4.4 Binary instrumentation

We have developed methods for enforcing security policies on untrusted binary code, using binary translation and instrumentation. This acts as a complement to our work on SVA: SVA provides strong security when we have source code; binary instrumentation is useful

for systems where we don't have source code.

Binary instrumentation provides a way to enforce security policies on a hostile or compromised operating system, or on other untrusted applications. (These methods can also be used to enforce a confidentiality policy associated with private data, in a data-oriented secure computing environment; see our work on data-oriented security below.) We have focused especially on information flow and system integrity policies, but our techniques apply more broadly.

One of the most practical approaches is dynamic taint analysis, implemented by dynamic binary translation. However, dynamic taint analysis can introduce a significant performance overhead (as much as 10x or so). We have developed methods to significantly improve the performance of dynamic taint analysis, by performing static analysis of binary code. Many previous techniques on binary static analysis tend to make optimistic assumptions, e.g., that the code was generated by a certain compiler, or that certain calling conventions or stack discipline were used. These assumptions are problematic when dealing with hand-written assembly or low-level OS code. Therefore, we have designed conservative static analysis methods that avoid these assumptions, so that they can be applied to OS code. Also, a major challenge in performing binary static analysis is the difficulty of static disassembly. We have developed methods to facilitate static disassembly using type inference to infer code pointers. Our research has managed to improve the performance of dynamic taint analysis (compared to prior work) by 4–6x for a number of workloads.

A second shortcoming of traditional systems for dynamic taint analysis is that they often suffer from under-tainting errors caused by missing control dependencies, which can cause them to lose track of private data. To address this challenge we have proposed DTA++, a technique that augments traditional dynamic taint analysis with extra propagation rules targeted at those control dependencies that are likely to cause under-tainting. Our approach uses symbolic execution to select "culprit branches" implicated in under-tainting based on information-flow properties, then constructs rules that constructs rules that can be added to an existing dynamic taint analysis system with minimal overhead. In an evaluation that tracked private text through format translations in 8 binary-only word processors including Microsoft Word, our system resolved under-tainting while introducing orders of magnitude less over-tainting than previous techniques.

We developed new techniques for static binary instrumentation (SBI). Most methods for binary instrumentation use dynamic binary translation (DBI): they instrument code on-the-fly, as it is executed. However, static instrumentation offers the potential of better performance and simpler implementation. Unlike previous SBI techniques, our technique works on COTS binaries without any need for compiler support (such as the inclusion of symbol or relocation information). Prior to our work, only dynamic binary instrumentation (DBI) techniques were robust enough to deal with large and complex executables as well as shared libraries. Our SBI platform is the first to provide a comparable level of robustness and completeness. In doing so, it addresses some of the drawbacks of DBI, including high overheads for fine-grained instrumentation, very large start-up overheads even for coarse-grained instrumentation, and inability to share any code memory across processes. We have used our platform to develop several security instrumentations, including (a) control-flow integrity, (b) shadow stack (to detect return address corruption attacks), and (c) system call interception. The runtime and memory overheads reported by our system are typically much smaller than previous DBI techniques. Moreover, our overheads are more than order

of magnitude smaller than that of Pin, perhaps the most widely used DBI tool.

We built a system for static binary instrumentation (SBI), called PSI: a platform for secure static binary instrumentation. We used PSI to demonstration several applications: e.g., adding a shadow stack to defend against ROP attacks, system call monitors, and library policy enforcement. The performance overhead of PSI is significantly less than the next-best systems (e.g., DynamicRIO, Pin)—for some applications, as much as an order of magnitude faster.

We applied our methods for SBI to instrument legacy COTS binaries with enforcement of control-flow integrity (CFI). Previous work has required either source code, binaries with debug symbols, or binaries with relocation information. Unfortunately, on servers, binaries typically do not come with this information, and source code is not always available. We showed how to enforce CFI on legacy, stripped binaries. We implemented our techniques and showed that they are robust (we've tested them on over 300MB of binaries), practical (we've used them on real Linux software), and perform well (about 9% performance overhead on the SPEC CPU benchmark). This work received a best paper award at the conference where it was published and has already led to a significant amount of follow-on research.

Accurate disassembly of large binaries has been one of the challenges to SBI. Previous SBI techniques provided no protection against disassembly errors, and moreover, these errors would in turn affect instrumentation soundness. We have overcome these challenges with a combination of a new algorithm for discovering code pointers in binaries, together with an instrumentation technique that guarantees soundness in spite of potential errors. Thus, unlike previous techniques that represent best-effort policy enforcement, our technique ensures nonbypassable policy enforcement.

Another significant challenge faced in binary instrumentation is the problem of accurate modeling of instruction semantics, especially for large and complex instruction sets. The scale of this problem is multiplied by the diversity of hardware platforms in deployment today. We developed a novel approach for tackling this problem that leverages decades of work done by the compiler community in developing retargetable code generators. Specifically, we developed a symbolic execution engine that automatically computes the mapping between target-independent internal representation used by a compiler such as gcc, and the corresponding assembly instructions. From the map, we can compute the inverse of the mapping, thereby allowing disassembled instructions to be mapped into target-independent higher level representation. By working at this level, our instrumentation can work across diverse hardware architectures. Our mapping is guaranteed to be as complete and as accurate as the compiler analyzed using symbolic evaluation. (While symbolic and concolic execution techniques have been used to address related problems such as generating test inputs that exercise a particular program path, our problem poses a unique challenge: it requires the extraction of the complete behavior of a software module, and has consequently required the development of new symbolic execution techniques based on constraint-solving.) We have implemented the approach and demonstrated that the approach can support multiple instruction sets (x86, ARM, and AVR), can extract information from multiple compilers (gcc and LLVM), and is powerful enough to let us analyze binaries of significant size (e.g., openssl and binutils).

Using related techniques, we have also shown a powerful method for enhancing assurance in back-end code generators found in compilers. Using our system, ArCheck, we were able to

test a significant fraction of the instructions in the GCC code generator; this work found 7 apparent bugs, where the code generator can generate assembly code whose semantics does not match the intended semantics of the intermediate representation (IR) code. ArCheck was also able to detect past bugs in the GCC code generator. ArCheck does not provide a guarantee of correctness, but we believe it has value for increasing assurance in the compiler. This is useful because the correctness of the compiler's code generator is part of the trusted computing base for many security systems.

We have also studied how to increase the assurance of dynamic binary instrumentation (DBI) systems. DBI systems can be used to provide isolation and virtualization of binary code and to add inlined reference monitors or instrumentation for security checking such as information-flow tracking. However, a potential weakness of DBI systems is that they do not always correctly emulate the semantics that instructions have on real hardware; for instance, previous research that applied random testing discovered thousands to tens of thousands of deviating test cases for x86 emulators. To improve the trustworthiness of such emulators, especially for security applications, we developed a checking framework for emulators that provides high-coverage test-generation and bounded verification of emulation systems used in practice. Our approach uses binary-level symbolic execution to extract bit-precise models of an emulator's behavior. By operating at the level of the host instruction set, this approach applies to direct interpreters, IR-based interpreters, and just-in-time compilers, without requiring source-code access.

4.5 Data-oriented security

We've developed client-side and server-side mechanisms for enforcing data-oriented policy. These mechanisms enable fine-grained policy enforcement driven by centralized data policies.

We have built a framework for building secure web applications using data policies. The framework is a source-to-source translator for Ruby on Rails apps with annotations added to the data model to specify access control and use context policies, which are enforced throughout the application. We've also developed client-side mechanisms for enforcing fine-grained access control policies on web content, able to express restrictions at the level of DOM elements and scripts.

We have applied these concepts to securing cloud computing. With today's cloud computing infrastructures, users lose control of their data in the cloud, and have to fully trust the cloud provider to safeguard the privacy of their data. Unfortunately, there exist numerous threats to the privacy of data in the cloud, including the risk of inadvertent disclosures, malware, and insider attacks.

To address these threats, we designed and implemented a "platform for private data", or "privacy platform" for short. The privacy platform allows a cloud provider to demonstrate privacy evidence to users, proving that their privacy policies have been satisfied. The privacy platform also allows rich data analytics over sensitive user data, while maintaining privacy. The platform for private data leverages a new primitive called self-protecting data capsules. A data capsule is a container comprising an encrypted data object and its privacy policies. Applications can access and operate on sensitive data inside capsules in a Secure Execution Environment isolated from the untrusted OS and applications.

We leverage Trusted Computing and code attestation techniques to allow a user to gain confidence in the privacy evidence produced by the privacy platform, without having to trust the OS and applications on the remote platform. Bootstrapping trust from hardware, we can establish a small trusted computing base on the cloud provider's platform. The total trusted computing base is the union of a hardware TCB, including the Trusted Platform Module (TPM) and processor virtualization features, and a software TCB, including a small security hypervisor and the monitoring and auditing mechanisms provided by the privacy platform.

We also applied these concepts to securing clients and computation that spans a user's client and a cloud service. We believe it is natural to associate data with real-world contexts and want to control access at the level of human contacts. Weddings, anniversaries, reunions, projects at work, and even one-off meetings are all examples of contexts that data might be associated with, even though such data is produced by multiple applications and spans multiple devices. Accordingly, we developed a system, called Bubbles, that provides privacy controls for users based upon these contexts and makes it easy to ensure that data is not shared or leaked outside of the context it was intended for. This approach holds the promise of cross-application, cross-device data protection for users.

4.6 Cryptography

We've developed a framework for multi-party secure computation. This approach allows computing on encrypted data, so that useful computations can be performed without ever decrypting the data into cleartext. The promise of this approach is that it allows rich computation to be performed on untrusted servers: we can encrypt the data, send the encrypted data elsewhere (e.g., to the cloud), and then others can perform rich computations without ever seeing the data or any cryptographic keys that would let them decrypt the data. For instance, even if the operating system and all applications on the cloud server are hostile or compromised, the confidentiality of the data remains inviolate.

In particular, we show how to implement garbled circuit protocols with orders of magnitude performance improvements over the best previous work, and demonstrated its effectiveness for several applications. We get 10–1000x speed improvements over the best previous results. This is exciting, because it enables computation to be performed on wholly untrusted machines: the untrusted machines receive only encrypted data, and cannot learn the data or tamper with the computation. Thus, this approach not only protects against a malicious or compromised OS on the untrusted machine, but in fact provides security even if all the software and hardware on the untrusted machine is malicious or compromised—thus providing security beyond what we initially envisioned for the program.

We've also shown that our methods—which are generic and can be applied to any multi-party secure computation problem—yield similar or better performance as prior specialized methods that are optimized for a particular problem. In particular, there has been considerable work on custom protocols for private set intersection. We show that garbled circuits yield solutions that are competitive with those custom protocols, and in some cases even faster. This suggests that the price of generality is low, and we might as well build and use general toolkits for multi-party secure computation, rather than trying to develop a custom cryptographic protocol for each new problem or application we face.

Inspired by this insight, we have developed general-purpose frameworks and methods for secure multi-party computation. Early work focused on circuits: the user specifies their problem as a boolean circuit, and then the framework automatically builds a secure protocol using garbled circuits. This works well for some problems, but for other problems, it can be limiting. Some problems are more naturally expressed using programs rather than circuits. While one can automatically compile programs to boolean circuits, this can cause blow-up in the size of the circuit (to the detriment of performance) in some cases, such as when programs use random-access arrays or other data structures. We have developed methods for compiling such programs to circuits more efficiently. In particular, we have efficient compilation strategies for several common data structures, including stacks, queues, and associative maps. This allows us to automatically generate more efficient cryptographic protocols for secure multi-party computation problems that are most naturally specified using programs that contain such data structures. These new techniques give an order-of-magnitude improvement for some problems and advance our vision of general frameworks for efficiently generating cryptographic protocols for secure multi-party computation.

We have also demonstrated how to provide security against active adversaries (going beyond the so-called honest-but-curious model) using dual encryption: what we call Quid-Pro-Quotocols. The resulting protocols achieve strong security properties: secure against active adversaries, with a guarantee that each execution of the protocol will leak at most one bit of information (and in practice often much less) about the confidential inputs. At the same time, the performance is good—not too much worse than honest-but-curious protocols, and significantly faster than competing protocols for security against active adversaries with zero leakage.

4.7 Reliability and availability

We developed a technique to make the OS more resilient to failures. When a request to an OS (e.g., a system call, a network packet, a disk operation) triggers a fatal kernel error today, all applications on the system are killed. This makes applications completely vulnerable to kernel errors, regardless of what application was responsible for triggering that error. We developed a strategy to enable a commodity OS kernel (in our case, Linux 2.4 and 2.6) to recover and continue execution on such an error, typically returning an error code for the offending request while allowing all other applications to continue unaffected. This technique can handle errors in *any* part of the kernel, not just in unloadable device drivers, like previous work. It is compiler-based and mostly automatic, requiring only a small number of annotations from the kernel programmer to identify certain kinds of requests. Moreover, for all but one extreme case, the techniques adds low overhead during normal operation.

We built Capo3, a system for implementing record and replay capabilities on Linux systems. Record and replay systems include support for capturing any sources of non-determinism while applications run and replaying them at a later time deterministically and efficiently. By replaying software, Capo3 can recreate arbitrary past states on computer systems, making time travel a first class abstraction in the Capo3 system.

Building on Capo3, we built Cyrus, which provides hardware support for faster record and replay. By adding extra support for recording into the processor, Cyrus is able to achieve

very good performance: recording involves no noticeable performance overhead during program execution, and replaying can be done at 50% slowdown compared to the initial execution. Cyrus is designed to be implementable on modern processors without requiring invasive changes to their architecture. Record-and-replay has a number of applications: for developers, it allows time-travel debugging; for reliability, it facilitates rapid fail-over; for security, it can support on-the-fly real-time intrusion analysis and forensics.

4.8 Hardware support for security

We have designed a hardware scheme to provide timing isolation between applications on multi-core processors. This work leverages technology trends towards parallelism and multi-core processors, which makes it feasible for hardware support to provide isolation (including neutering of covert channels) between multiple applications running on different cores. The scheme relies upon static time-slice scheduling (time division multiplexing) to control access to certain critical shared resources on-chip.

We developed a new secure processor, PHANTOM, that obfuscates its memory access trace. To an adversary who can observe the processor's output pins, all memory access traces will be computationally indistinguishable (a cryptographic property known as Oblivious RAM or ORAM). Such a system can help protect against powerful adversaries, such as malicious insiders in a cloud data center, and could be combined with emerging technologies such as Intel SGX to achieve stronger security guarantees than are possible without it.

Our work on PHANTOM represents the culmination of a line of research we conducted into ORAM, from both a theoretical and practical perspective. We first developed improvements to existing ORAM algorithms and constructed empirical models for their performance. Then, we designed PHANTOM, an oblivious processor whose novel memory controller aggressively exploits DRAM bank parallelism and scales well to a large number of memory channels.

Finally, we built a complete hardware implementation of PHANTOM on a commercially available FPGA, and through detailed experiments show that PHANTOM is efficient in both area and performance. A single ORAM access latency is 32x slower than a single memory access (for a 1GB ORAM), and SQLite queries on a population database see 1.2-6x slowdown.

PHANTOM is the first demonstration of a practical, oblivious processor and can provide strong confidentiality guarantees when offloading computation to the cloud. Since our work in this area, the cryptographic research community has seen substantial further progress and attention to ORAM, so we expect to see further improvements in the future.

We have also studied methods for protecting against malicious backdoors in hardware, in the hope of further extending the degree of protection we can provide applications against the threat of a hostile platform. This work included algorithms for trying to find malicious hardware in hardware designs and a system for removing this malicious hardware automatically. Subsequent work showed that the original algorithm was broken, and shed further light on the challenges of detecting malicious hardware.

We subsequently designed ways to protect against security bugs in processors. Modern processors are incredibly complex, containing billions of transistors, and contain bugs—

sometimes security-critical bugs. These bugs are a major concern: if the processor is not trustworthy, it is very hard to build a system that we can be confident will be secure. This is not a purely theoretical concern: for instance, the Intel Core 2 Duo processor family lists 129 known bugs; AMD processors have had at least 28 security-critical bugs in the time period 2006–2013.

We have developed new techniques to guard against security-critical bugs in processors. Our approach involves introducing extra runtime checks into the hardware design to ensure that specific security invariants always hold, with a recovery mechanism to handle violations of these invariants. We have implemented the approach in a system called SPECS. Our evaluation found that SPECS could detect and guard against 86% of historical processor bugs, increases the size of the processor by less than 5%, and has no software run-time overhead. This suggests that the approach is viable for security-critical applications. One of the core research challenges was how to handle detected violations of the security invariants, in a way that allows to continue execution but without violating security. We developed a recovery-and-repair scheme that allows the processor to continue making forward progress after an invariant violation is detected, while preserving the security properties.

4.9 Securing web applications

We also studied how to provide strong protection for web applications. The web is widely used to deliver applications, and HTML5 and web technologies effectively provide a new "operating system" and runtime platform for many applications. We've studied how to protect these kinds of applications. Our techniques significantly reduce the TCB of these modern web applications with very little overhead. In particular, we show how to provide integrity for critical code as well as how to provide confidentiality and data confinement for critical data. We also developed a new primitive to enable a reference-monitor-like abstraction for HTML5 applications and prototyped it in browsers. Our work was well received by industry and we are currently working with Google engineers to implement a newer version of the idea in Google Chrome and Google applications.

We have shown how to build web applications that support client-side encryption of confidential data. We implemented our techniques in a system called ShadowCrypt and showed that it has low overhead and can successfully add client-side encryption to a wide variety of rich web applications (e.g., Gmail, Reddit, OpenEMR, Trello, and many others). When data is encrypted on the client side, then the server need not be trusted: a hostile or malicious operating system on the server cannot compromise data confidentiality. Thus, ShadowCrypt provides powerful guarantees while supporting many familiar web applications.

We have explored using N-version programming to protect web browsers against attack. This was inspired by the observation that, while it is not unusual for an individual web browser to suffer from a vulnerability for a period of time (until it is fixed), these vulnerabilities often affect only a single browser. It is rare for all major browsers—e.g., Firefox, Chrome, and Opera—to all be vulnerable at the same time. Accordingly, we designed Cocktail, which uses N-version programming to run the Firefox, Chrome, and Opera browsers simultaneously in parallel. Every time the user visits a page, all three browsers are used to render the page, and a security kernel checks that they produce the same screen image. Also, the security kernel checks that all network operations, side effects on user-visible

browser state, and filesystem operations are identical. Consequently, an attacker who can exploit a vulnerability in one browser cannot do any harm; any attempt to tamper with system state or mis-render the page will be immediately detected and blocked. To defeat the system, an attacker would need to be able to exploit and compromise all three browser instances at the same time. Our implementation demonstrates that the scheme is practical, offers good compatibility with existing web pages, and offers reasonable performance (page load latency with Cocktail is comparable to Opera, about 40% higher than Firefox, and about 2x higher than Chrome). This suggests that, surprisingly, N-version programming could be an effective way to defend against browser compromise.

4.10 Code hardening

Our research on defending against a malicious or hostile operating system also led to several spin-off results, where the techniques we developed for the project were useful for other important problems in computer security. Several of the techniques we developed for binary instrumentation and policy enforcement turned out to enable powerful new methods for hardening application and kernel code against attack. Several of our most significant results along these lines are outlined below.

We have also applied these techniques to system hardening and demonstrated new methods for control-flow integrity. Our approach, CCFIR, enforces control-flow integrity on modern x86 binaries, protecting against ROP attacks at performance much better than any prior scheme. CCFIR has a low overhead: between 3.6% and 8.6%, an overhead that is practical for modern interactive applications and systems. CCFIR could be applied to operating systems to protect against OS compromise—but it is also useful more broadly, for hardening all kinds of applications on services. For instance, applying CCFIR to browsers such as IE6 and Firefox 3.6, we found that our technique would have prevented all exploits on these.

We have designed code-pointer integrity (CPI), a new method for software hardening that protects the integrity of all pointers to code. Code-pointer integrity is an alternative to control-flow integrity and offers protection against ROP attacks and other attacks. The advantage of CPI is that its performance overhead is surprisingly low: 1.2% – 8.4%, depending upon the level of security desired.

CPI is especially exciting because it offers useful hardening protections at a performance cost that is plausibly low enough to be deployable. Moreover, CPI's performance overhead is lower than the best state-of-the-art schemes that provide comparable protection: it is faster than the leading control-flow integrity (CFI) schemes and faster than bounds checking. Moreover, it can be applied to legacy code with little developer effort. For these reasons, CPI has attracted considerable attention from the research community. Researchers are currently actively studying the level of the security provided by CPI and what degree of protection it can and cannot provide.

We built KCoFI, which provides control-flow integrity for commodity operating system kernel. The system hardens a commodity kernel against code injection, control-flow hijacking, and ROP attacks. Inspired by our work on SVA, KCoFI combines control-flow integrity to prevent control-flow hijacking, software fault isolation to protect security-critical state, and memory protection for drivers and DMA. In this way, KCoFI addresses some of the

challenges specific to applying CFI to operating system kernels. In comparison to SVA, KCoFI provides weaker guarantees but better performance. We implemented KCoFi on the FreeBSD kernel and observe performance overheads that vary from 0% (for a workload running a web server), 0–27% (for a workload running an OpenSSH server), to 96% (for a workload simulating a mail server). We built a formal model of KCoFI and used the model to provide a partial proof that the approach prevents violations of CFI, though we have not attempted to formally verify the implementation itself.

We have developed techniques for making mandatory access control and information flow control more usable on desktops. Our work is an extension of Biba integrity policies, specifically of low-watermark and LOMAC, extended to address the self-revocation problem. The benefit is that this allows our system to provide strong security against malware. We have built a system, PIP (portable integrity protection), and shown that it provides improved usability, good compatibility with existing desktop applications, and low performance overhead. We also built a related system, SRFD (self-revocation free downgrading), that applies similar policies but enforced from a LSM module within the Linux kernel.

We built LBC, a bounds-checking compiler for C programs that is one of the state-of-the-art systems for this task. LBC provides compatibility with legacy C code and has been tested on over 7 millions lines of C code. Its performance overhead is about as low as the best competing schemes: about a 23% overhead on SPECINT 2000 and even less on server applications like the Apache webserver.

4.11 Other applications

We examined how to provide security for robots. We envision a future where robots play a significant role in people's lives. This vision introduces many challenges from a security perspective. One of the unique challenges of robots as that they are physical things that can interact with the physical world, and thus if a robot runs amok, it can cause property damage, privacy violations, or risks to human safety. We began an exploration of these issues so that we have a scientific and engineering base for such a future world. We analyzed what kind of operating systems are appropriate, how to build an app store for robots and support third-party applications securely, and what sort of policies and mechanisms are well-suited to security for robot apps.

5 Technology transition

Our research on IBOS forms the basis for a company that Sam King and his students Shuo Tang and Haohui Mai founded, named Adrenaline Mobility. This helped transition the technology and techniques we've developed for secure web browsing to industry. Adrenaline Mobility was acquired by Twitter.

Our work on the Capo3 deterministic replay system has already seen some use in industry. Currently, Intel is using the Capo3 software with experimental hardware that they are developing to improve replay support for multi-core computer systems. The Capo3 software is open source and can be found at `https://github.com/kingst/capo3`. (Capo3 was jointly supported by this MURI and by other funding sources.)

We have demonstrated Secure Virtual Architecture (SVA) on the Linux kernel. SVA has been released under an open-source license and is available from `http://sva.cs.illinois.edu` and `https://github.com/jtcriswell/SVA`.

Our implementation of PIP is available under an open-source license at `http://www.seclab.cs.sunysb.edu/seclab/pip`. The related system SRFD (self-revocation free downgrading) is available under an open-source license at `http://www.seclab.cs.sunysb.edu/seclab/srfd/`.

Our implementation of PSI (a platform for secure static binary instrumentation) is available under an open-source license at `http://www.seclab.cs.sunysb.edu/mingwei/psi/`.

Our prototype implementation of code-pointer integrity (CPI) is available at `http://levee.epfl.ch`.

Our implementation of LBC (lightweight bounds-checking for C code) is available under an open-source license at `http://www.seclab.cs.sunysb.edu/seclab/lbc/`.

Professor King has been invited to and has given lectures at the NSA, Sandia, DARPA, Intel, Microsoft, Samsung, and Qualcomm about his work on malicious hardware.

We demo'd Bubbles (the system for context-based privacy) at Microsoft Research, Google, and Walmart Labs.

6 Publications

All publications are available via our website, `http://www.dhosa.org/`. We have 85 publications in peer-reviewed conferences/workshops and 0 in journals.

2015

Niranjan Hasabnis, R. Sekar. Automatic Generation of Assembly to IR Translators Using Compilers. In Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT), February, 2015.

Niranjan Hasabnis, Qiao Rui, R. Sekar. Checking Correctness of Code Generator Architecture Specifications. In ACM/IEEE International Symposium on Code Generation and Optimization (CGO), February, 2015.

Matthew Hicks, Cynthia Sturton, Samuel T. King, Jonathan M. Smith. SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs. In International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2015.

2014

John Criswell, Nathan Dautenhahn, Vikram S. Adve. Virtual ghost: protecting applications from hostile operating systems. In ASPLOS 2014.

Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, Dawn Song. ShadowCrypt: Encrypted Web Applications for Everyone. In 21st ACM Conference on Computer and Communications Security (CCS), November 2014.

Volodymyr Kuznetsov, Laszlo Szekeres, Mathias Payer, George Candea, R. Sekar, Dawn Song. Code-Pointer Integrity. In USENIX Operating System Design and Implementation (OSDI), October, 2014.

Wai-Kit Sze, R. Sekar. Comprehensive Integrity Protection for Desktop Linux. In ACM Symposium on Access Control Models and Technologies (SACMAT), June, 2014.

Wai-Kit Sze, Bhuvan Mital, R. Sekar. Towards More Usable Information Flow Policies for Contemporary Operating Systems. In ACM Symposium on Access Control Models and Technologies (SACMAT), June, 2014. Honorable mention for Best paper.

Zhiwei Li, Warren He, Devdatta Akhawe, Dawn Song. The Emperors New Password Manager: Security Analysis of Web-Based Password Managers. In Proceedings of the 23rd USENIX Security Symposium, August 2014.

Nicholas Carlini, David Wagner. ROP is Still Dangerous: Breaking Modern Defenses. In Proceedings of the 23rd USENIX Security Symposium, August 2014.

Mingwei Zhang, Qiao Rui, Niranjan Hasabnis, R. Sekar. A Platform for Secure Static Binary Instrumentation. In Virtual Execution Environments (VEE), March, 2014.

John Criswell, Nathan Dautenhahn, Vikram S. Adve. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In IEEE Symposium on Security and Privacy 2014.

Martin Christoph Maas. PHANTOM: Practical Oblivious Computation in a Secure Processor. M.S. Thesis, University of California, Berkeley.

2013

Wai-Kit Sze, R. Sekar. A Portable User-Level Approach for System wide Integrity Protection. In Annual Computer Security Applications Conference (ACSAC) December, 2013.

Daniel Huang, Greg Morrisett. Formalizing the SAFECode Type System. In Proceedings of the Certified Proofs and Programs Conference (CPP 13), December 2013.

Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovi, John Kubiatowicz, Dawn Song. A High-Performance Oblivious RAM Controller on the Convey HC-2ex Heterogeneous Computing Platform. In 46th Annual IEEE/ACM International Symposium on Microarchitecture, December 2013.

Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovi, John Kubiatowicz, Dawn Song. PHANTOM: Practical Oblivious Computation in a Secure Processor. In ACM Conference on Computer and Communications Security, November 2013.

Chi Yuan Cho, Vijay DSilva, Dawn Song. Blitz: Compositional Bounded Model Checking for Real-World Programs. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), November 2013.

Cynthia Sturton, Rohit Sinha, Thurston Dang, Sakshi Jain, Michael McCoyd, Wei Yang Tan, Petros Maniatis, Sanjit Seshia, David Wagner. Symbolic Software Model Validation. In 11th ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2013), October 2013.

Dan Caselden, Alex Bazhanyuk, Mathias Payer, Stephen McCamant, and Dawn Song. Hi-CFG: Construction by binary analysis and application to attack polymorphism. In Proceedings of ESORICS, September 2013.

Kevin Zhijie Chen, Noah Johnson, Vijay DSilva, Shuaifu Dai, Kyle MacNamara, Tom Magrino, Edward XueJun Wu, Martin Rinard, Dawn Song. Contextual Policy Enforcement in Android Applications with Permission Event Graphs. In Network and Distributed System Security Symposium 2013.

Devdatta Akhawe, Frank Li, Warren He, Prateek Saxena, Dawn Song. Data-confined HTML5 Applications. In ESORICS, September 2013.

Mingwei Zhang, R. Sekar. Control Flow Integrity for COTS Binaries. In 22nd USENIX Security Symposium, August 2013. Best paper award.

Matthew Finifter, Devdatta Akhawe, David Wagner. An Empirical Study of Vulnerability Rewards Programs. In 22nd USENIX Security Symposium, August 2013.

Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Laszlo Szekeres, Stephen McCamant, Dawn Song, and Wei Zou. Practical Control Flow Integrity and Randomization for Binary Executables. In IEEE Symposium on Security and Privacy (IEEE S&P) May, 2013.

Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Stephen McCamant, and Laszlo Szekeres. Protecting Function Pointers in Binary. In ACM Symposium on Information, Computer and Communications Security (ASIACCS), May, 2013.

Yan Huang, Jonathan Katz, and David Evans. Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose. In 33rd International Cryptology Conference (CRYPTO), August 2013.

Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In 22nd USENIX Security Symposium, August 2013.

Tianhao Tong and David Evans, GuarDroid: A Trusted Path for Password Entry. In Mobile Security Technologies (MoST), San Francisco, CA, 23 May 2013.

Laszlo Szekeres, Mathias Payer, Tao Wei, R. Sekar. SoK: Eternal War in Memory. In IEEE Symposium on Security & Privacy, May 2013.

Samee Zahur and David Evans, Circuit Structures for Improving Efficiency of Security and Privacy Tools. In 34th IEEE Symposium on Security and Privacy, San Francisco, CA, 19-22 May 2013.

N. Honarmand, N. Dautenhahn, J. Torrellas, S.T. King, G. Pokam and C. Pereira, Cyrus: Unintrusive Application-Level Record-Replay for Replay Parallelism, in Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS), March 2013.

Haohui Mai, Edgar Pek, Hui Xue, Samuel T. King, and P. Madhusudan, Verifying Security Invariants in ExpressOS, in Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS), March 2013.

2012

Yan Huang. Practical Secure Two-Party Computation. Ph.D. thesis, University of Virginia Computer Science.

Neil Zhenqiang Gong, Wenchang Xu, Ling Huang, Prateek Mittal, Emil Stefanov, Vyas Sekar, Dawn Song. Evolution of Social-Attribute Networks: Measurements, Modeling, and Implications using Google+. In ACM/USENIX Internet Measurement Conference (IMC), November 2012.

Rohit Sinha, Cynthia Sturton, Petros Maniatis, Sanjit A. Seshia, and David Wagner. Verification with Small and Short Worlds. In Formal Methods in Computer-Aided Design (FMCAD) 2012. October 2012.

Ralf Sasse, Samuel T. King, Jose Meseguer, and Shuo Tang. IBOS: A Correct-By-Construction Modular Browser, In Proceedings of the International Symposium on Formal Aspects of Component Software (FACS), Mountain View, CA, September 2012.

Devdatta Akhawe, Prateek Saxena, Dawn Song. Privilege Separation in HTML5 Application. In Usenix Security, August 2012.

Mohit Tiwari, Prashanth Mohan, Andrew Osheroff, Hilfi Alkaff, Elaine Shi, Eric Love, Dawn Song, Krste Asanovic. Context-centric Security, In Proceedings of the 7th USENIX Workshop on Hot Topics in Security (HotSec 2012), Bellevue, WA. August 2012.

Neil Zhenqiang Gong, Ameet Talwalkar, Lester Mackey, Ling Huang, Eui Chul Richard Shin, Emil Stefanov, Elaine (Runting) Shi, Dawn Song. Jointly Predicting Links and Inferring Attributes using a Social-Attribute Network (SAN). In Proceedings of The 6th Social Network Mining and Analysis Workshop (SNA-KDD), August 2012.

Lorenzo Martignoni, Pongsin Poosankam, Matei Zaharia, Jun Han, Stephen McCamant, Dawn Song, Vern Paxson, Adrian Perrig, Scott Shenker, and Ion Stoica. Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems, Proceedings of the USENIX Annual Technical Conference (ATC 2012), Boston, MA. June 2012.

Greg Morrisett, Gang Tan, Joseph Tassarotti, Jean-Baptiste Tristan, Edward Gan. RockSalt: better, faster, stronger SFI for the x86, Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI12). ACM, June 2012.

Haohui Mai, Shuo Tang, Samuel T. King, Calin Cascaval, and Pablo Montesinos. A Case for Parallelizing Web Pages, Proceedings of 4th USENIX Workshop on Hot Topics in Parallelism (HotPar 2012), Berkeley, CA, June 2012.

Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution. Proceedings of 33rd IEEE Symposium on Security and Privacy (Oakland 2012), San Francisco, CA, May 2012.

Riccardo Pelizzi and R. Sekar. Protection, Usability and Improvements in Reflected XSS Filters. ACM Symposium on Information, Computer and Communications Security (ASI-ACCS), May 2012.

Niranjan Hasabnis, Ashish Misra and R. Sekar. Light-weight Bounds Checking. ACM/IEEE International Symposium on Code Generation and Optimization (CGO), April 2012.

Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? Proceedings of 19th Network and Distributed Security Symposium (NDSS 2012), San Diego, CA, February 2012.

Hui Xue, Nathan Dautenhahn, and Samuel T. King, Using Replicated Execution for a More Secure and Reliable Web Browser, Proceedings of the 2012 Network and Distributed System Security Symposium (NDSS), San Diego, CA, February 2012.

Emil Stefanov, Elaine Shi, Dawn Song. Towards Practical Oblivious RAM. In Proceedings of the 2012 Network and Distributed System Security Symposium (NDSS), San Diego, CA. February 2012.

2011

Yan Huang, Chih-hao Shen, David Evans, Jonathan Katz, and abhi shelat. Efficient Secure Computation with Garbled Circuits. Invited paper for Seventh International Conference on Information Systems Security (ICISS 2011), Jadavpur University, Kolkata, India, 15-19 December 2011.

Riccardo Pelizzi and R. Sekar. A Server- and Browser-Transparent CSRF Defense for Web 2.0 Applications, Annual Computer Security Applications Conference (ACSAC), December, 2011.

Lorenzo Cavallaro and R. Sekar, Taint-Enhanced Anomaly Detection, International Conference on Information Systems Security (ICISS) December, 2011.

Mike Samuel, Prateek Saxena, and Dawn Song. Context-Sensitive Auto-Sanitization in Web Templating Languages Using Type Qualifiers, Proceedings of 18th ACM Conference on Computer and Communications Security (CCS 2011), Chicago, IL, October 2011.

Shuo Tang, Nathan Dautenhahn, and Samuel T. King, Fortifying Web-Based Applications Automatically, Proceedings of 18th ACM Conference on Computer and Communications Security (CCS 2011), Chicago, IL, October 2011.

Peter Chapman and David Evans. Automated Black-Box Detection of Side-Channel Vulnerabilities in Web Applications, Proceedings of 18th ACM Conference on Computer and Communications Security (CCS 2011), Chicago, IL, October 2011.

Yikan Chen and David Evans. Auditing Information Leakage for Distance Metrics. Proceedings of Third IEEE Conference on Privacy, Security, Risk and Trust, Boston, MA, 9-11 October 2011.

Yuchen Zhou and David Evans. Protecting Private Web Content from Embedded Scripts, European Symposium on Research in Computer Security (ESORICS 2011), September 2011.

Murph Finnicum, Samuel T. King. Building Secure Robot Applications. Proceedings of the 2011 Usenix Workshop on Hot Topics in Security (HotSec). August 2011

Yan Huang, Peter Chapman, and David Evans. Privacy-Preserving Applications on Smartphones. Proceedings of the 2011 Usenix Workshop on Hot Topics in Security (HotSec). August 2011

Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel T. King. Debugging the Data Plane with Anteater. Proceedings of the ACM SIGCOMM, August 2011.

Yan Huang, David Evans, Jonathan Katz, Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits, 20th USENIX Security Symposium, August 2011.

Cynthia Sturton, Matthew Hicks, David Wagner, Samuel T. King. Defeating UCI: Building Stealthy and Malicious Hardware, Proceedings of the 2011 IEEE Symposium on Security and Privacy, May 2011.

Petros Maniatis, Devdatta Akhawe, Kevin Fall, Elaine Shi, Stephen McCamant, and Dawn Song. Do You Know Where Your Data Are? Secure Data Capsules for Deployable Data Protection, 13th Workshop on Hot Topics in Operating Systems (HotOS), May 2011.

Jonathan Burket, Patrick Mutchler, Michael Weaver, Muzzammil Zaveri, David Evans. GuardRails: A Data-Centric Web Application Security Framework, 2nd USENIX Conference on Web Application Development (WebApps 11), June 2011.

Jean-Baptiste Tristan, Paul Govereau, Greg Morrisett. Evaluating Translation Validation for LLVM, Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI), June 2011.

Yan Huang, Lior Malka, David Evans, and Jonathan Katz. Efficient Privacy-Preserving Biometric Identification, Proceedings of the 2011 Network and Distributed System Security Symposium, February 2011.

2010

Shuo Tang, Haohui Mai, and Samuel T. King, Trust and Protection in the Illinois Browser Operating System, Proceedings of the 2010 Symposium on Operating Systems Design and Implementation (OSDI), October 2010.

J. Siefers, G. Tan, and G. Morrisett. Robusta: Taming the native beast of the JVM, 17th ACM Conference on Computer and Communications Security (CCS), November 2010.

Sruthi Bandhakavi, Samuel T. King, P. Madhusudan, Marianne Winslett. VEX: Vetting Browser Extensions For Security Vulnerabilities, Proceedings of 2010 USENIX Security Symposium, August 2010.

Heng Yin, Pongsin Poosankam, Steve Hanna, Dawn Song. HookScout: Proactive Binary-Centric Hook Detection, Seventh Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), July 2010.

Devdatta Akhawe, Adam Barth, Peifung Lam, John C. Mitchell, Dawn Song. Towards a

Formal Foundation of Web Security, 2010 IEEE Computer Security Foundations Symposium, July 2010.

Steve Hanna, Richard Shin, Devdatta Akhawe, Arman Boehm, Dawn Song. The Emperors New API: On the (In)Secure Usage of New Client Side Primitives, Web 2.0 Security and Privacy, May 2010.

Matthew Hicks, Murph Finnicum, Samuel T. King, Milo M. K. Martin, Jonathan M. Smith. Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically, 2010 IEEE Symposium on Security and Privacy (Oakland), May 2010.

C. Y. Cho, J. Caballero, C. Grier, V. Paxson, D. Song. Insights from the Inside: A View of Botnet Management from Infiltration, Proceedings of the 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 2010), April 2010.

Shuo Tang, Chris Grier, Onur Aciicmez, Samuel T. King. Alhambra: A system for creating, enforcing and testing browser security policies, Proceedings of 19th International Word Wide Web Conference (WWW2010), Raleigh, NC, April 2010.

Yves Younan, Pieter Philippaerts, Lorenzo Cavallaro, R. Sekar, Frank Piessens and Wouter Joosen, PAriCheck: An Efficient Pointer Arithmetic Checker for C Programs, ACM Symposium on Information, Computer and Communications Security (ASIACCS), March 2010.

Adam Barth, A. Porter Felt, Prateek Saxena, Aaron Boodman. Protecting Browsers from Extension Vulnerabilities, NDSS 2010, February 2010.

Matthew Finifter, Joel Weinberger, Adam Barth. Preventing Capability Leaks in Secure JavaScript Subsets, NDSS 2010, February 2010.

Adam Barth, Benjamin I. P. Rubinstein, Mukund Sundararajan, John C. Mitchell, Dawn Song, Peter Bartlett. A Learning-Based Approach to Reactive Security, Proceedings of Financial Cryptography and Data Security 10. Fourteenth International Conference. January 2010.

2009

Anh M. Nguyen, Nabil Schear, HeeDong Jung, Apeksha Godiyal, Sam T. King, Hai Nguyen. MAVMM: A Lightweight and Purpose-Built VMM for Malware Analysis, ACSAC 2009, December 2009.

Lixin Li, Jim Just, and R. Sekar, Online Signature Generation for Windows Systems, Annual Computer Security Applications Conference (ACSAC), December 2009.

Juan Caballero, Pongsin Poosankam, Christian Kreibich, Dawn Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering, Proceedings of the 16th ACM Conference on Computer and Communication Security, November 2009.

Juan Caballero, Zhenkai Liang, Pongsin Poosankam, Dawn Song. Towards Generating High Coverage Vulnerability-Based Signatures with Protocol-Level Constraint-Guided Exploration, Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID), September 2009.

Prateek Saxena, Pongsin Poosankam, Stephen McCamant, Dawn Song. Loop-Extended Symbolic Execution on Binary Programs, Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), July 2009.

Alok Tongaonkar, R. Sekar and Sreenaath Vasudevan, Fast Packet Classification using Condition Factorization, Applied Cryptography and Network Security (ACNS), June 2009.

James Newsome, Stephen McCamant, Dawn Song. Measuring Channel Capacity to Distinguish Undue Influence, Proceedings of the Fourth ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS), June 2009.

Adam Barth, Juan Caballero, Dawn Song. Secure Content Sniffing for Web Browsers or How to Stop Papers from Reviewing Themselves, Proceedings of the IEEE Symposium on Security and Privacy, May 2009.

7 Awards

Dawn Song is a 2010 MacArthur Fellow.

The paper "VEX: Vetting Browser Extensions For Security Vulnerabilities" (partly supported by this grant) received the Best Paper Award at Usenix Security 2010.

Krste Asanovi was named ACM Distinguished Scientist in 2011.

The paper "Faster Secure Two-Party Computation Using Garbled Circuits" (partly supported by this grant) was a Finalist for the 2011 NYU-Poly AT&T Best Applied Security Paper Award.

The paper "Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution" (partly supported by this grant) was a Finalist for the 2012 NYU-Poly AT&T Best Applied Security Paper Award.

The paper "Control Flow Integrity for COTS Binaries" (partly supported by this grant) received the Best Paper Award at Usenix Security 2013.

The paper "PHANTOM: Practical Oblivious Computation in a Secure Processor" (partly supported by this grant) was listed as a Top 10 Finalist for the Best Applied Security Paper Contest organized by CSAW 2013.

The paper "Towards More Usable Information Flow Policies for Contemporary Operating Systems" (partly supported by this grant) received honorable mention for best paper at SACMAT 2014.

Vikram Adve was named an ACM Fellow in 2014.

AFOSR Deliverables Submission Survey

Response ID:4579 Data

1.

1. Report Type

F na Report

Primary Contact E-mail
Contact email if there is a problem with the report.

daw@cs.berke ey.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

510-642-2758

Organization / Institution name

Un vers ty of Ca forn a, Berke ey

Grant/Contract Title
The full title of the funded effort.

Hardware, Languages, and Arch tectures for Defense Aga nst Host e Operat ng Systems

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-09-1-0539

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Dav d Wagner

Program Manager
The AFOSR Program Manager currently assigned to the award

Tr stan Nguyen

Reporting Period Start Date

08/01/2009

Reporting Period End Date

07/31/2014

Abstract

The Defend ng Aga nst Host e Operat ng Systems research project focused on bu d ng systems that w rema n secure even when the operat ng system s comprom sed or host e. It was a co aborat ve effort among researchers from Harvard, Stony Brook, U.C. Berke ey, Un vers ty of I no s at Urbana-Champa gn, and the Un vers ty of V rg n a. The project stud ed th s prob em from a number of ang es, nc ud ng operat ng system des gn, hyperv sor enforcement, stat c ana ys s, and hardware des gn, and on a number of p atforms, nc ud ng desktop systems, servers, and mob e systems. The team deve oped numerous mechan sms and techn ques for address ng th s prob em.

Distribution Statement
This is block 12 on the SF298 form.

D str but on A - Approved for Pub c Re ease

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

coverpage.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

f na report.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

We have 85 pub cat ons n peer-rev ewed conferences/workshops and 0 n journa s. They are sted n the f na report. A pub cat ons are a so ava ab e v a our webs te, http://www.dhosa.org/.

Changes in research objectives (if any):

None

Change in AFOSR Program Manager, if any:

Bob Herk otz ret red. Tr stan Nguyen s the new program manager.

Extensions granted or milestones slipped, if any:

None

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Start ng FY	FY+1	FY+2
Sa ary			
Equ pment/Fac t es			
Supp es			
Tota			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

May 15, 2015 02:29:51 Success: Ema Sent to: daw@cs.berke ey.edu