DOI:10.1145/2980989

Article development led by acmqueue
queue.acm.org

Expert-curated guides to the best of CS research.

Research for Practice: Web Security and Mobile Web Computing

OUR THIRD INSTALLMENT of Research for Practice brings readings spanning programming languages, compilers, privacy, and the mobile Web.

First, **Jean Yang** provides an overview of how to use information flow techniques to build programs that are secure by construction. As Yang writes, information flow is a conceptually simple "clean idea": the flow of sensitive information across program variables and control statements can be tracked to determine whether information may in fact leak. Making information flow practical is a major challenge, however. Instead of relying on programmers to track information flow, how can compilers and language runtimes be made to do the heavy lifting? How can application writers easily express their privacy policies and understand the implications of a given policy for the set of values that an application user may see? Yang's set of papers directly addresses these questions via a clever mix of techniques from compilers, systems, and language design. This focus on theory made practical is an excellent topic for RfP.

Second, **Vijay Janapa Reddi** and **Yuhao Zhu** provide an overview of the challenges for the future of the mobile Web. Mobile represents a major frontier in personal computing, with extreme growth in adoption and data volume. Accordingly, Reddi and Zhu outline three major ongoing challenges in mobile Web computing: responsiveness of resource loading, energy efficiency of computing devices, and making efficient use of data. In their citations, Reddi and Zhu draw on a set of techniques spanning browsers, programming languages, and data proxying to illustrate the opportunity for "cross-layer optimization" in addressing these challenges. Specifically, by redesigning core components of the Web stack, such as caches and resource-fetching logic, systems operators can improve users' mobile Web experience. This opportunity for co-design is not simply theoretical: Reddi and Zhu's third citation describes a mobile-optimized compression proxy that is already running in production at Google.

As always, our goal in RfP is to allow readers to become experts in the latest, practically oriented topics in computer science research in a weekend afternoon's worth of reading time. I am grateful to this installment's experts for generously contributing such a strong set of contributions, and, as always, we welcome your feedback!
—*Peter Bailis*

Peter Bailis is assistant professor of computer science at Stanford University. His research in the Future Data Systems group (http://futuredata.stanford.edu/) focuses on the design and implementation of next-generation data-intensive systems.

» about RfP

Research for Practice combines the resources of the ACM Digital Library, the largest collection of computer science research in the world, with the expertise of the ACM membership. In every RfP column two or more experts share a short, curated selection of papers on a concentrated, practically oriented topic.

Practical Information Flow for Web Security

By Jean Yang

Information leaks have become so common that many have given up hope when it comes to information security.[3] Data breaches are inevitable anyway, some say.[1] I don't even go on the Internet anymore, other (computers) say.[6]

This despair has led yet others to the Last Resort: Reasoning about what our programs actually do. For years, bugs didn't matter as long as your robot could sing. If your program can go twice the speed it did yesterday, who cares what outputs it gives you? But we are starting to learn the hard way that no amount of razzle-dazzle can make up for Facebook leaking your phone number to the people you didn't invite to the party.[4]

This realization is leading us to a new age, one in which reasoning techniques that previously seemed unnecessarily baroque are coming into fashion. Growing pressure from regulators is finally making it increasingly popular to use precise program analysis to ensure software security.[5] Growing demand for producing Web applications quickly makes it relevant to develop new paradigms—well-specified ones, at that—for creating secure-by-construction software.

The construction of secure software means solving the important problem of *information flow*. Most of us have heard of trapdoor ways to access information we should not see. For example, one researcher showed that it is possible to discover the phone numbers of thousands of Facebook users simply by searching for random phone numbers.[2] Many such leaks occur not because a system shows sensitive values directly, but because it shows the results of computations—such as search—on sensitive values. Preventing these leaks requires implementing policies not only on sensitive values themselves, but also whenever computations may be affected by sensitive values.

Enforcing policies correctly with respect to information flow means reasoning about sensitive values and policies as they flow through increasingly complex programs, making sure to reveal only information consistent with the privileges associated with each user. There is a body of work dedicated to compile-time and runtime techniques for tracking values through programs for ensuring correct information flow.

While information flow is a clean idea, getting it to work on real programs and systems requires solving many hard problems. The three papers presented here focus on solving the problem of secure information flow for Web applications. The first one describes an approach for taking trust out of Web applications and shifting it instead to the framework and compiler. The second describes a fully dynamic enforcement technique implemented in a Web framework that requires programmers to specify each policy only once. The third describes a Web framework that customizes program behavior based on the policies and viewing context.

Shifting Trust to the Framework and Compiler through Language-Based Enforcement

Chong, S., Vikram, K. and Myers, A.C.
SIF: Enforcing confidentiality and integrity in Web applications. *Proceedings of the 16th Usenix Security Symposium, 2007.*
https://www.usenix.org/conference/16th-usenix-security-symposium/sif-enforcing-confidentiality-and-integrity-Web

In securing Web applications, a major source of the burden on programmers involves reasoning about how information may be leaked through computations across different parts of an application and across requests. Without additional checks and balances, the programmer must be fully trusted to do this correctly.

This first selection presents a framework that shifts trust from the application to the framework and compiler. The Servlet Information Flow (SIF) framework follows a line of work in language-based information flow focused on checking programs against specifications of security policies. Built using the Java servlet framework, SIF prevents many common sources of information flow—for example, those across multiple requests. SIF applications are written in Jif, a language that extends Java with programmer-provided labels specifying policies for information flow. SIF uses a combination of compile-time and runtime enforcement to ensure security policies are enforced from the time a request is submitted to when it is returned, with modest enforcement overhead. The major contribution of the SIF work is in showing how to provide assurance (much of it at compile time) about information flow guarantees in complex, dynamic Web applications.

Mitigating Annotation Burden through Principled Containment

Giffin, D.B. et al.
Hails: Protecting data privacy in untrusted Web applications. *Proceedings of the 10th Usenix Symposium on Operating Systems Design and Implementation*, 2012.
https://www.usenix.org/node/170829

While compile-time checking approaches are great for providing assurance about program security, they often require nontrivial programmer effort. The programmer must not only correctly construct programs with respect to information flow, but also annotate the program with the desired policies.

An alternative approach is confinement: running untrusted code in a restricted way to prevent the code from exhibiting undesired behavior. For information flow, confinement takes the form of tagging sensitive values, tracking them through computations, and checking tags at application endpoints. Such dynamic approaches are often more popular because they require little input from the programmer.

This paper presents Hails, a Web framework for principled containment. Hails extends the standard MVC (model-view-controller) paradigm to include policies, implementing the MPVC (model-*policy*-view-controller) paradigm where the programmer may specify label-based policies separately from the rest of the program. Built in Haskell, Hails uses the LIO (labeled IO) library to enforce security policies at the thread/context level and MAC (mandatory access control) to mediate access to resources such as the database. It has good performance for an information flow control framework, handling approximately 47.8K requests per second.

Hails has been used to build several Web applications, and the startup Intrinsic is using a commercial version of Hails.

The Hails work shows it is possible to enforce information flow in Web applications with negligible overhead, without requiring programmers to change how they have been programming.

Shifting Implementation Burden to the Framework

Yang, J., et al.
Precise, dynamic information flow for database-backed applications. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2016, 631–647.
http://dl.acm.org/citation.cfm?id=2908098

With the previous two approaches, the programmer remains burdened with constructing programs correctly with respect to information flow. Without a change in the underlying execution model, the most any framework can do is raise exceptions or silently fail when policies are violated.

This paper looks at what the Web programming model might look like if information flow policies could be factored out of programs the way memory-managed languages factor out allocation and deallocation. The paper presents Jacqueline, an MPVC framework that allows programmers to specify how to compute an alternative default for each data value; and high-level policies about when to show each value that may contain database queries and/or depend on sensitive values.

A plausible default for a sensitive location value is the corresponding city. A valid policy is allowing a viewer to see the location only if the viewer is within some radius of the location. This paper presents an implementation strategy for Jacqueline that works with existing SQL databases. While the paper focuses more on demonstrating feasibility than on the nuts and bolts of Web security, it de-risks the approach for practitioners who may want to adopt it.

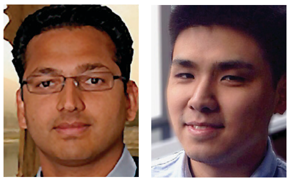
Final Thoughts

The past few years have seen a gradual movement toward the adoption of practical information flow: first with containment, then with microcontainers and microsegmentation. These techniques control which devices and services can interact with policies for software-defined infrastructures such as iptables and software-defined networking. Illumio, vArmour, and GuardiCore are three among the many startups in the microsegmentation space. This evolution toward finer-grained approaches shows that people are becoming more open to the system re-architecting and runtime overheads that come with information flow control approaches. As security becomes even more important and information flow techniques become more practical, the shift toward more adoption will continue.

Acknowledgments. Thanks to A. Aufrichtig, S. Chong, V. Iozzo, L. Meyerovich, and D. Stefan.

References
1. Balluck, K. Corporate data breaches 'inevitable,' expert says. *The Hill* (Nov. 30 2014); http://thehill.com/policy/cybersecurity/225550-cybersecurity-expert-data-breaches-inevitable.
2. Cunningham, M. Facebook security flaw could leak your personal info to criminals. Komando.com (Aug. 10, 2015); http://bit.ly/2fRXp8L
3. Information is beautiful. World's biggest data breaches, 2016; http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/.
4. Gellman, B. and Poitras, L. U.S., British intelligence mining data from nine U.S. Internet companies in broad, secret program. *Washington Post* (June 7, 2013); http://wapo.st/1LcAw6p
5. Open Web Application Security Project (OWASP). Static code analysis, 2016; https://www.owasp.org/index.php/Static_Code_Analysis.
6. Zetter, K. Hacker lexicon: What is an air gap? *Wired* (Dec. 8, 2014); http://www.wired.com/2014/12/hacker-lexicon-air-gap/.

Jean Yang is an assistant professor in the computer science department at Carnegie Mellon University. In 2015 she cofounded the Cybersecurity Factory accelerator to bridge the gap between research and practice in cybersecurity.



The Red Future of Mobile Web Computing

By Vijay Janapa Reddi and Yuhao Zhu

The Web is on the cusp of a new evolution, driven by today's most pervasive personal computing platform—mobile devices. At present, there are more than three billion Web-connected mobile devices. By 2020, there will be 50 billion such devices. In many markets around the world mobile Web traffic volume exceeds desktop Web traffic, and it continues to grow in double digits.

Three significant challenges stand in the way of the future mobile Web. The papers selected here focus on carefully addressing these challenges. The first major challenge is the *responsiveness* of Web applications. It is estimated that a one-second delay in Web page load time costs Amazon $1.6 billion in annual sales lost, since mobile users abandon a Web service altogether if the Web page takes too long to load. Google loses eight million searches from a four-tenths-of-a-second slowdown in search-results generation. A key bottleneck of mobile Web responsiveness is resource loading. The number of objects in today's Web pages is already on the order of hundreds, and it continues to grow steadily. Future mobile Web computing systems must improve resource-loading performance, which is the focus of the first paper.

The second major challenge is *energy efficiency*. Mobile devices are severely constrained by the battery. While computing capability driven by Moore's Law advances approximately every two years, battery capacity doubles every 10 years—creating a widening gap between computational horsepower and the energy needed to power the device. Therefore, future mobile Web computing must be energy efficient. The second paper in our selection proposes Web programming language support for energy efficiency.

The third major challenge is *data usage*. A significant amount of future mobile Web usage will come from emerging markets in developing countries where the cost of mobile data is prohibitively large. To accelerate the Web's growth in emerging markets, future mobile Web computing infrastructure must serve data consciously. The final paper discusses how to design a practical and efficient HTTP data compression proxy service that operates at Google's scale.

Developers and system architects must optimize for RED (responsiveness, energy efficiency, and data usage), ideally together, to usher in a new generation of mobile Web computing.

Intelligent Resource Loading For Responsiveness

Netravali et al.
Polaris: Faster page loads using fine-grained dependency tracking. *Proceedings of the 13th Usenix Symposium on Networked Systems Design and Implementation*, 2016.
https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/netravali

A key bottleneck for mobile Web responsiveness is resource loading. The bottleneck stems from the increasing number of objects (for example, images and Cascading Style Sheets files) on a Web page. According to the HTTP Archive, over the past three years alone, Web pages have doubled in size. Therefore, improving resource-loading performance is crucial for improving the overall mobile Web experience.

Resource loading is largely determined by the critical path of the resources that Web browsers load to render a page. This critical path, in the form of a resource-dependency graph, is not revealed to Web browsers statically. Therefore, today's browsers make conservative decisions during resource loading. To avoid resource-dependency violations, a Web browser typically constrains its resource-loading concurrency, which results in reduced performance.

Polaris is a system for speeding up the loading of Web page resources, an important step in coping with the surge in mobile Web resources. Polaris constructs a precise resource-dependency graph offline, and it uses the graph at runtime to determine an optimal resource-loading schedule. The resulting schedule maximizes concurrency and, therefore, drastically improves mobile Web performance. Polaris also stands out because of its transparent design. It runs on top of unmodified Web browsers without the intervention of either Web application or browser developers. Such a design minimizes the deployment inconvenience and increases its chances of adoption, two factors that are essential for deploying the Web effectively.

Web Language Support for Energy Efficiency

Zhu, Y., Reddi, J.
GreenWeb: Language extensions for energy-efficient mobile Web computing. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2016, 145–160.
http://dl.acm.org/citation.cfm?id=2908082

Energy efficiency is the single most critical constraint on mobile devices that lack an external power supply. Web runtimes (typically the browser engine) must start to budget Web application energy usage wisely, informed by user QoS constraints. End-user QoS information, however, is largely unaccounted for in today's Web programming languages.

The philosophy behind GreenWeb is that application developers provide minimal yet vital QoS information to guide the browser's runtime energy optimizations. Empowering a new generation of energy-conscious Web application developers necessitates new programming abstractions at the language level. GreenWeb proposes two new language constructs, *QoS type* and *QoS target*, to capture the critical aspects of user QoS experience. With the developer-assisted QoS information, a GreenWeb browser determines how to deliver the specified user QoS expectation while minimizing the device's energy consumption.

GreenWeb does not enforce any particular runtime implementation. As an example, the authors demonstrate one implementation using ACMP (asymmetric chip-multiprocessor) hardware. ACMP is an energy-efficient heterogeneous architecture that mobile hardware vendors such as ARM, Samsung, and Qualcomm have widely adopted—you probably have one in your pocket. Leveraging the language annotations as hints, the GreenWeb browser dynamically schedules execution on the ACMP hardware to achieve energy savings and prolong battery life.

Data Consciousness in Emerging Markets

Agababov, V. et al.
Flywheel: Google's data compression proxy for the mobile Web. *Proceedings of the 12th Usenix Symposium on Networked Systems Design and Implementation*, 2015;
http://research.google.com/pubs/pub43447.html

The mobile Web is crucial in emerging markets. The first order of impedance for the mobile Web in emerging markets is the high cost of data, more so than performance or energy efficiency. It is not uncommon for spending on mobile data to be more than half of an individual's income in developing countries. Therefore, reducing the amount of data transmitted is essential.

Flywheel from Google is a compression proxy system to make the mobile Web conscious of data usage. Compression proxies to reduce data usage (and to improve latency) are not a new idea. Flywheel, however, demonstrates that while the core of the proxy server is compression, there are many design concerns to consider that demand a significant amount of engineering effort, especially to make such a system practical at Google scale. Examples of the design concerns include fault tolerance and availability upon request anomalies, safe browsing, robustness against middlebox optimizations, and so on. Moreover, drawing from large-scale measurement results, the authors present interesting performance results that might not have been observable from small-scale experiments. For example, the impact of data compression on latency reduction is highly dependent on the user population, metric of interest, and Web page characteristics.

"At present there are more than three billion Web-connected mobile devices. By 2020, there will be 50 billion such devices."

Conclusion

We advocate addressing the RED challenge holistically. This will entail optimizations that span the different system layers synergistically. The three papers in our selection are a first step toward such cross-layer optimization efforts. With additional synergy we will likely uncover more room for optimization than if each of the layers worked in isolation. It is time that we as a community make the Web great again in the emerging era.

Vijay Janapa Reddi is an assistant professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin.

Yuhao Zhu is a Ph.D. candidate at the University of Texas at Austin.