

David Molnar (dmolnar@eecs.berkeley.edu)
Research Statement

1 Research Overview

I work in computer security, cryptography, and electronic privacy. The most important problem in my field is that we work with an *adversary*, an intelligent and arbitrarily acting party who attempts to break everything we build. This is a problem because it makes it difficult to know when we are done, setting security apart from other areas of computer science where performance can be measured directly, such as by verifying the proof of a theorem or by measuring the speed of a system. Long and tearful experience shows that simply testing a few attacks on the system is not sufficient to establish security. To make progress, we need to address this central problem of malicious behavior. I will work in two areas of the field that have hope for this problem: software security and applied cryptography. My work in each area uses a combination of theoretical and practical approaches. Each approach reinforces the other, and we need both to make progress in this field.

2 Software Security

The area of software security concerns itself with bugs in software that allow an adversary to take over computer systems. The classic example of such a bug is a “buffer overflow” in a C program [20]. The last fifteen years have seen an immense amount of effort spent by adversaries at finding software bugs, then writing exploit code that uses the bug to make the computer do the adversary’s bidding. Depending on the details of the bug, the adversary can then package this exploit as part of a self-propagating “worm”, such as the Blaster worm, or use an exploit create web pages that silently install code on unsuspecting browsers. Such exploits may then add the unsuspecting user’s machine to a “botnet,” or collection of compromised computers; the Storm botnet, for example is estimated to include at least 160,000 machines, and possibly up to a million. Because these bugs are so important, major software companies such as Microsoft and Apple distribute patches for security bugs on a regular basis. Each such patch costs hundreds of thousands of dollars, or more, to develop and deploy. While these classic security bugs stem from failures of memory safety, recently web applications have introduced new classes of bugs, such as “cross-site scripting” and “SQL injection.” These are bugs that can lead to serious problems even if the application is written in a memory safe language such as Java or C#.

The key to addressing malicious behavior in software security is that the developers of the software we wish to secure are not (usually) malicious. Therefore, we can make changes to our development process to improve software security.

Each security critical bug in our software has a “life cycle” that begins with the introduction of the bug in new code. The next phase of the cycle is the detection of the bug, followed by the creation of a fix to the software. Finally, developers validate the fix and apply it to the software. Each phase offers opportunities for improvement.

My current work focuses on the bug detection phase of the cycle. While proper language design, such as sufficiently expressive type systems, can help a great deal with security bugs in the creation part of the cycle, new language ideas take time to be adopted. I look at bug detection because it offers a chance to serve the security needs of legacy code and can be adopted with minimal changes to a programmer’s work flow. My work on *automated whitebox fuzz testing*, with Patrice Godefroid and Michael Levin, combined a technique called *dynamic test generation* with the idea of *fuzz testing* [9]. In basic fuzz testing, we start with a “seed” input for an application, randomly change the seed, and then feed the new file to the application [16]. In dynamic test generation, we use symbolic execution to generate a *symbolic trace* of a path taken by an example input, then use a constraint solver to synthesize new inputs that explore new paths “close” to the original input path [2, 7]. Whitebox fuzz testing consists of applying dynamic test generation repeatedly starting from different initial seed inputs. We can then check the program on newly generated inputs to see if it finds a bug. A key property of the approach is that it always provides a test case for each claimed bug; this greatly simplifies communicating the bug to a developer and does not have “false positives” in the same sense as static analysis.

We built an automated whitebox fuzz testing system called SAGE (“Scalable, Automated, Guided Execution”), which found critical security bugs in shipping Microsoft applications missed by static analysis, human code review, and previous fuzz testing techniques. We then extended SAGE to incorporate *active property checking*, in which we direct the search towards inputs that violate specific security properties [8]. The SAGE tool is now used daily by several teams within Microsoft as part of their software testing process. Before our work on SAGE, it was not known whether the state of the art in constraint solving and symbolic execution could scale to commodity Windows software; our work shows it can. Our work also demonstrated that dynamic test generation can be applied to *binary* programs, even when source code is not available.

My recent work with David Wagner and Xue Cong Li continues this approach to focus specifically on *integer vulnerabilities*. An integer vulnerability arises due to a mismatch between machine arithmetic and mathematical arithmetic. We built a system called *SmartFuzz* that applies these techniques to commodity Linux programs, building on the open source Valgrind tool, and I have released the SmartFuzz code as open source for the benefit of other researchers. Our work shows that these techniques are effective at identifying bugs and security vulnerabilities; for example, we have used our tools to find and report over one hundred bugs in several open source tools.

What distinguishes my work from other work in this area is its focus on large scale, real world applications of the tantalizing theoretical possibilities.

Previous work on dynamic test generation showed excellent results on smaller programs using source-based instrumentation, but scaling remained an open question. Concurrent independent work by Cadar, Dunbar, and Engler continues this trend by producing impressive code coverage results on over 450 common Unix utilities, but each individual program is fairly small [3]. My work, in contrast, has focused on scaling up these techniques to large programs; the traces processed by SmartFuzz and SAGE are typically hundreds of millions of instructions in length. This focus has aided tech transfer, helping SAGE move into production use and SmartFuzz to report bugs against commodity open source media players. By combining theory and practice, my work pushes our notion of what is possible with these techniques.

2.1 Research Direction: Software Security

The key organizing principle for my research direction in software security is that human time is expensive, while computer time in the cloud is cheap. Using the cloud makes it possible to employ sophisticated techniques on a scale previously not possible with a single desktop alone. My vision is a developer desktop that seamlessly works with the cloud to provide high quality testing with as little lag time as possible. Today’s trends in on-demand computing put this vision within reach.

I see three main ways I can improve bug discovery as part of this vision. First, recent work in empirical software engineering shows that bugs cluster in “hot spots,” code containing an unusually high proportion of bugs. Furthermore, we can find such hot spots by looking at code metrics or at previous discovery spots of security critical bugs [22, 19]. Common sense states that if particular code is a bug hot spot, then we should spend more resources on testing that code, but the best way to do this with whitebox fuzz testing is not clear. For example, we could give preference to generating tests that cover code paths in the hot spot. We could also use heavier weight static analysis techniques on the hot spot, or invest in generating “summaries” of the hot spot code [10].

Second, whitebox fuzz testing’s effectiveness depends critically on the choice of “seed” input. I am therefore interested in building a system that uses lightweight instrumentation to identify promising candidates for such seed inputs from machines “in the wild.” Ideally, such instrumentation would be lightweight enough to be on by default. This idea is in the spirit of cooperative bug isolation, but instead of focusing only on inputs that cause program crashes, I would look for inputs that exercise code previously untouched, or exercise new paths through hot spot code [15]. Because a user’s inputs can contain sensitive information, I will investigate whether it is sufficient to simply send back the path condition exercised by the input, as previously applied for crash-inducing inputs [5].

Third, I am interested in what properties can and can not be checked efficiently by the dynamic test generation approach. To be efficient, a tool’s memory usage must scale sublinearly in the size of the program trace. The algorithms community has developed a rich set of techniques for such “stream-

ing algorithms,” which I hope may yield efficient algorithms for dynamic test generation [18]. Also, I am interested in applying lower bounds for streaming algorithms to characterize which extensions of dynamic test generation can and cannot be efficiently implemented.” Such an analysis has been carried out for network intrusion detection properties, which suggests these techniques could also be applicable here [14].

Beyond bug finding, I will improve the rest of the bug cycle. For example, I am interested in reducing the amount of human time required to generate and test a patch for a security bug. While there is a rich line of work in automatically generating “vulnerability signatures” given an example input triggering a bug, this work is primarily aimed at worm defense [6, 1]. Therefore there is little attention paid to making the generated patch easy to maintain or understand. Even if we cannot automatically generate a patch, however, we should be able to check that a programmer’s candidate patch catches inputs that previously exhibited the underlying bug. Once we generate a huge number of bugs, this patching and understanding process becomes the next bottleneck where program analysis techniques can help.

3 Applied Cryptography

The last thirty years has seen the blossoming of *modern cryptography*, which concerns itself with more than simple secret codes, instead focusing on secure computation generally. One of the intellectual triumphs of the area has been the development of *reduction proofs* for establishing the security of a cryptographic construction. In this style of proof, we show that if the adversary can violate some security guarantee of the construction, then the adversary can solve some problem believed to be computationally difficult [11]. Therefore, if we believe the adversary cannot solve the difficult problem, it follows that the adversary cannot violate the security guarantee, no matter what it does. The reduction proof approach is one of our main tools for addressing malicious behavior. The other main tool is a library of primitives, such as one-way permutations, that have survived long vetting by the cryptographic community and can be used to build protocols which achieve our desired goals.

Applied cryptography focuses on the tricky interaction between theory and practice in cryptography. Much of my work in the area has focused on security and privacy issues in Radio Frequency Identification, or RFID. RFID refers to a technology where a small chip with a radio, called a “tag,” is attached to an item or group of items. The tag carries information about the item and can be queried remotely by radio. This raises privacy questions if the tag is attached to a sensitive item or carries sensitive data. This also raises security questions if the tag is believed to be unique but in fact can be “cloned” to produce an identical tag.

With David Wagner, I worked on the use of RFID technology in libraries [17]. We surveyed existing systems for RFID and discovered privacy and security problems with each of them. As part of our work, we introduced the *symmetric-*

key private authentication problem: how can a reader and tag authenticate each other without an eavesdropping adversary learning the tag's identity? The key issue here is that there may be thousands or even millions of tags, each with its own secret that is shared with the reader. When the protocol starts the reader does not know which secret to use. The naive approach is for the reader to try each secret in turn, but this does not scale well. We gave the first protocol for this problem that scales sub-linearly in the number of tags.

With Ari Juels and David Wagner, I then worked on RFID in *e-passports* [13]. An e-passport is a passport book that contains a chip with information about the bearer, as standardized by the International Civil Aviation Organization. This information includes a picture of the holder, the name, and the passport number at a minimum, with many other optional elements. While the e-passport standard specifies many security features, most of them are optional. The original United States deployment choices for e-passports chose to forego these features, leading to a passport that could be scanned remotely to reveal all its information about the bearer. Our work was a critical analysis of these choices and recommended using at least a feature called "Basic Access Control." We submitted our report as part of a formal public comment to the U.S. Department of State, which joined over 1000 other public comments critical of e-passport security. Afterwards, the State Department changed its mind and adopted Basic Access Control for U.S. e-passports, in line with our recommendations.

Most recently I worked with Alex Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, Benne de Weger, and Dag Arne Osvik to show that previous work on collisions in the cryptographic hash function MD5 could be applied in practice [21]. We used chosen prefix collisions to obtain a rogue Certification Authority certificate trusted by all common web browsers. This certificate allows us to impersonate any web site we choose, such as Amazon, Bank of America, or eBay, without raising any browser warnings, even if that web site uses HTTPS. Our work showed how the poor practices of an existing CA combined with chosen-prefix MD5 collisions to make possible a real, practical attack. Verisign, who owns the certificate authority we used, has since ceased to issue certificates using MD5 from that authority and has committed to stopping the use of MD5 entirely by the end of January 2009.

Both the MD5 work and the e-passport paper are examples of work that point out problems in deployed cryptographic systems. Besides the intrinsic interest of an individual system, each problem contributes *case studies* to the field. This is important because applied cryptography is a discipline where we learn much more from our failures than from our successes. Examining failures is the main method we have for discovering general engineering principles that guide our design of new systems.

Work in applied cryptography also naturally leads to an interest in theoretical questions. For example, I have worked with David Wagner and Nick Hopper on the question of how to properly define security for *digital watermarking* [12]. In a digital watermarking scheme, we wish to "mark" an item, such as a movie or image, with information that is difficult to remove. Despite a long history of practical work in the area, few attempts existed to formally define security, and

the existing attempts were incomplete. We gave a new definition of security for a “weak” watermark notion that captures the watermarks designed in practice, and we showed how to construct “strong” watermarks that rule out a variety of attacks from weak ones.

3.1 Research Direction: Applied Cryptography

Today’s trends towards more powerful embedded, wearable, and mobile computing will make sophisticated cryptographic protocols available to a wide audience. At the same time, these trends will fundamentally change the way we interact with such protocols, as contrasted with today’s standalone computers. In some cases, most notably in developing countries, the cell phone will be the first and main computer anyone will ever use. Another key use case is identity documents: already many such documents, such as e-passports, contain nontrivial computing power. These documents will become platforms for providing a range of goods and services, including everything from parking tickets to travel across international borders. Applied cryptography in these settings is exciting because it offers the potential to find both new theory and practice. On the theoretical side, we will need new security models and to specify new notions of security appropriate for these changing situations. Then, we need to discover how theory meets practice by building and breaking real systems. Work in these directions will therefore directly affect the experience of millions of people. I now sketch two specific directions here that I find interesting.

First, I am interested in applied cryptography for systems that integrate with the physical world, particularly embedded and mobile applications. This direction offers opportunities to leverage properties of the physical world to obtain security guarantees. For example, consider the problem of determining whether a wireless access point seen in a coffee shop is in fact the “correct” access point run by the shop or simply an impostor run by the person at the next table. On its own, this problem is difficult, but we can observe that many coffee shops play music in the background. Can we combine this sonic channel with information sent by the access point to thwart the impostor? A potentially useful tool here is the use of *integrity codes*, which allow us to encode information that is resistant to change in settings where an adversary can erase but not arbitrarily change data [4]. The promise here is that the physics of sound may give us a setting where we can apply these integrity codes under an assumption about the adversary’s *physical* capabilities.

Second, another direction is access control and sharing for data overlaid on physical spaces. Today’s mobile devices already use GPS and cell localization to provide real-time maps and search. In the near future, I believe social networking applications and more specialized applications will offer us the possibility of annotating physical locations with messages to ourselves and others. How can we share this data effectively, preferably without needing to store it entirely in the cloud? I expect this area will raise challenges in specifying the proper security models, and in turn this will lead to nontrivial new cryptography.

4 Conclusion

In the next five years, I will blend theory and systems approaches to make progress in computer security. By building systems, I will bring ideas to people in a way that enables substantial feedback. By proving theorems, I will improve our confidence in these systems. By studying empirical data, I will detect whether the theory matches practice and gather new phenomena for theoretical explanation. Today's trends in on-demand computing enable large scale projects with modest investment, which makes this an exciting time to be in research.

References

- [1] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 2–16, Washington, DC, USA, 2006. IEEE Computer Society.
- [2] C. Cadar and D. Engler. EGT: Execution generated testing. In *SPIN*, 2005.
- [3] Cristian Cadar, Daniel Dunbar, and Dawson Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of OSDI 2008*, 2008.
- [4] Mario Cagalj, Jean-Pierre Hubaux, Srdjan Čapkun, Ramkumar Rengaswamy, Ilias Tsigkogiannis, and Mani Srivastava. Integrity (i) codes: Message integrity protection and authentication over insecure channels. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 280–294, Washington, DC, USA, 2006. IEEE Computer Society.
- [5] Miguel Castro, Manuel Costa, and Jean-Philippe Martin. Better bug reporting with better privacy. In *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pages 319–328, New York, NY, USA, 2008. ACM.
- [6] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: end-to-end containment of internet worms. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 133–147, New York, NY, USA, 2005. ACM.
- [7] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation)*, pages 213–223, Chicago, June 2005.

- [8] P. Godefroid, M. Y. Levin, and D. Molnar. Active Property Checking. In *EMSOFT*, 2008.
- [9] P. Godefroid, M.Y. Levin, and D. Molnar. Automated Whitebox Fuzz Testing. In *Proceedings of NDSS'2008 (Network and Distributed Systems Security)*, San Diego, February 2008. http://research.microsoft.com/users/pg/public_psfiles/ndss2008.pdf.
- [10] Patrice Godefroid. Compositional dynamic test generation. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–54, New York, NY, USA, 2007. ACM.
- [11] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.
- [12] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *Theory of Cryptography Conference*, 2007.
- [13] Ari Juels, David Molnar, and David Wagner. Security and privacy issues in e-passports. In *SecureComm 2005*, 2005.
- [14] Kirill Levchenko, Ramamohan Paturi, and George Varghese. On the difficulty of scalably detecting network attacks. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 12–20, New York, NY, USA, 2004. ACM.
- [15] Benjamin Robert Liblit. *Cooperative bug isolation*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2004. Chair-Alexander Aiken.
- [16] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of UNIX utilities. *Communications of the Association for Computing Machinery*, 33(12):32–44, 1990.
- [17] David Molnar and David Wagner. Privacy and security in library rfid : Issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 12–20, New York, NY, USA, 2004. ACM.
- [18] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2007.
- [19] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, October 2007.
- [20] Aleph One. Smashing the stack for fun and profit. *Phrack*, 49(14), 1996.

- [21] Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. MD5 considered harmful today: Creating a rogue CA certificate. In *25th Chaos Communications Congress (CCC)*, Berlin, December 2008. <http://www.win.tue.nl/hashclash/rogue-ca/>.
- [22] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 531–540, New York, NY, USA, 2008. ACM.