A Brief Introduction to Usable Security

Researchers have studied usable computer security for more than 20 years, and developers have created numerous security interfaces. Here, the authors examine research in this space, starting with a historical look at papers that address two consistent problems: user authentication and email encryption. Drawing from successes and failures within these areas, they study several security systems to determine how important design is to usable security. Their discussion offers guidelines for future system design.

Bryan D. Payne and W. Keith Edwards
Georgia Institute of Technology

Newcomers to usable security research might find themselves overwhelmed by how much information currently exists. Similarly, it can be difficult for people to quickly distill "lessons learned" from this research that they can apply to their own areas. Here, rather than give a comprehensive study of all the security work from the past 20 years, we've identified a few select representative areas.

Two areas receiving significant attention from the usable security community are user authentication and email encryption, so we look at two case studies that provide insight into these areas. Design issues also play a significant role in usable security, and we examine in particular the emergence of design guidelines, exploring what they say about usable security research by looking at several successful and unsuccessful design efforts. Finally, some debate exists over the role of interface changes versus more structural solutions in providing usable security; we look at how reframing security problems can sometimes lead to better usability.

Case Studies

These case studies provide an overview of past research in passwords and authentication, and email encryption. Researchers have yet to solve either of these problems, so this work represents only the first part of a continuing story. The specific research that we

1089-7801/08/$25.00 © 2008 IEEE

discuss can serve as a starting point for anyone interested in learning more about this field.

Passwords and Authentication

The problem of how users authenticate to systems, particularly using passwords, is one of the oldest and most heavily studied topics in usable security. Passwords present an innate tension between usability (which is aided by having short, easily memorable passwords and reusing them across multiple systems) and security (which dictates longer, more diverse passwords that are difficult to "crack," as well as distinct passwords for each system). This tension isn't new, but it's becoming worse as people have more accounts and computers grow ever faster (thus, cracking passwords becomes easier). Guidelines for password selection focus largely on security rather than usability. In 1979, for example, Robert Morris and Ken Thompson[1] discussed a technique for having users choose less predictable passwords:

> The password entry program was modified so as to urge the user to use more obscure passwords. If the user enters an alphabetic password (all upper-case or all lower-case) shorter than six characters, or a password from a larger character set shorter than five characters, then the program asks him to enter a longer password.

Twenty-six years later, current suggestions for choosing passwords are as follows:[2]

> A secure password should be 8 characters or longer, random, with upper-case characters, lower-case characters, digits, and special characters.

Although these suggestions might, in fact, improve security, they reduce usability — and encourage counterproductive behaviors such as simply writing passwords down in plain sight.

Much usability research reframes the password problem more generally as a user-authentication one. Specifically, users should have no reason to present a password before accessing some resource: the real goal is to validate that they access only the resources they have permissions to use. System administrators can achieve this through numerous methods, including passwords, passphrases, personal identification numbers (PINs), graphical authentication, biometrics, and secure tokens. Research over the past 25 years has investigated many techniques for user authentication in an effort to strike a better balance between usability and security.

Passphrases. The first attempt to design a user-authentication system around usability came from Sigmund Porter's work on passphrases in 1982.[3] Porter argued that passphrases — which use sequences of words to authenticate users — are more usable because they're more memorable, especially compared to system-generated passwords. Additionally, because passphrases are longer than passwords, they offer a larger key space and thus more security.

Pass-algorithms. In 1984, James Haskett acknowledged the password memorability problem and developed a technique known as *pass-algorithms*.[4] A simple example suggests a pass-algorithm in which the user must type in the next letter in the alphabet for each letter in a prompt. Thus, the password for the prompt "BEL" is "CFM." One interesting property of this system is that the password would effectively change for each login while the pass-algorithm remained the same.

User-friendly password advice. At nearly the same time as Haskett introduced his pass-algorithm technique, Ben Barton and Marthalee Barton addressed the need for a user-friendly password system[5] by providing different ways to aid in password selection. The techniques they discussed included ways to convert a sentence or expression to a reasonably strong password (for example, "One for the money" becomes "14MUNNY" or "I Love Paris in the Springtime" becomes "ILPITST") that people still use today.

Cognitive passwords. Another method for addressing the problems inherent in picking passwords that are both easy to remember and hard to guess is cognitive passwords.[6] The idea behind this technique is to give the user a series of questions that are easier for them to answer than for others. An empirical study verified that this technique generally works; however, people close to the user — especially a spouse — could successfully answer many of the questions. Thus, this technique doesn't appear viable for high-security systems.

Passfaces and graphical passwords. In 2000, graphical passwords quickly became a hot re-

search area. Three different groups published variations on the theme of using images to support the authentication process. Sacha Brostoff and Angela Sasse evaluated the Passfaces technique, in which a user selects an image of a person's face known to them from a grid of nine faces (see Figure 1a).[7] The user repeats this four times with different faces to complete the authentication process. Ian Jermyn and his colleagues developed a graphical password technique in which the password is essentially a pencil-style drawing (see Figure 1b).[8] Finally, the Deja Vu technique[9] is very similar to Passfaces, except that it uses various images, rather than just faces. Each of these graphical techniques compared favorably to standard passwords from a user's perspective, but later work revealed various security problems with them.[10,11]

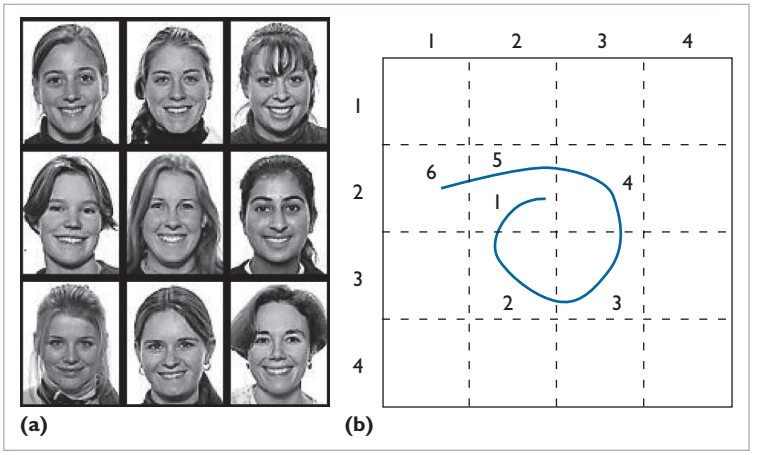

Figure 1. Graphical passwords. (a) The Passfaces authentication system[7] uses a face grid for user authentication, whereas (b) another technique uses a pencil drawing as input.[8]

PassPoints. Finally, in 2005, Susan Wiedenbeck and her colleagues presented a variation on using images as passwords called PassPoints.[2] Here, users select from different regions within a single image to create a password, as Figure 2 shows. This initial work focused on creating an implementation that was acceptable from the viewpoint of user-selection tolerance regions. Specifically, the study focused on determining how many pixels surrounding an initial user click must be included in the region used for a valid password. Obviously, a smaller tolerance region leads to better security but can negatively affect the application's usability.

Figure 2. The PassPoints system. Users click on regions to create a graphical password. Selected regions indicate where users have clicked.[2]

Email Encryption

The computer security community has long understood that email is not a secure medium. However, this understanding hasn't stopped users from treating it as if it were secure. The obvious solution seems to be email encryption, but although the technology is available, most people don't take this precaution. Here, we look at email encryption's history to see why usability is an important factor in this feature's acceptance.

Privacy-Enhanced Mail. In 1985, the Internet Architecture Board (IAB) began work on Privacy-Enhanced Mail (PEM).[12] Unfortunately, PEM never caught on, in large part because it lacked flexibility. Its most serious problem, however, was that it required all entities worldwide to trust a single certificate authority (CA) infrastructure,[13] leading to "organizational usability" problems. So, although many PEM implementations existed, it never achieved wide deployment.[14]

Pretty Good Privacy. In 1991, Phil Zimmermann released an email encryption scheme known as Pretty Good Privacy (PGP; www.philzimmermann.com). It quickly became popular, and by 1996, Bruce Schneier suggested that PGP was "the closest you're likely to get to military-grade encryption."[15] Today, PGP is both a product and an open standard, implemented in various projects including PGP (www.pgp.com), OpenPGP, and GnuPG (www.gnupg.org). It's so widely used in large part because it employs a decentralized trust model.[13] Instead of using a centralized CA, PGP introduced the idea of a "web of trust" in which each participant can validate the other participants' trust. Although this model mimics typical human interaction, it doesn't scale well. Thus, PGP seems limited to smaller user communities.

S/MIME. Whereas PGP is available as a plug-in to many email applications, vendors already include Secure Multipurpose Internet Mail Extensions (S/MIME) in nearly every major email application. S/MIME is a set of protocols for securely sending messages encoded using the MIME format. Security in S/MIME comes from the Public Key Cryptography Standard (PKCS) #7, an RSA Data Security standard. S/MIME takes the middle ground between PEM and PGP in establishing its trust model: although a CA is required, you can use any CA.

Even with email encryption's availability, and the fact that most people believe securing email communications to be important, many users don't employ PGP or S/MIME.[16] Two important empirical studies have shown that usability plays a major role in this situation.

Email encryption usability studies. In 1999, Alma Whitten and Doug Tygar released a paper entitled "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0."[17] It received considerable attention because it verified the usability problems inherent in security interfaces. The authors' results revealed numerous difficulties users face with PGP:

- Several test participants emailed secrets without encryption; such errors are irreversible and thus very serious.
- Participants chose passphrases that were similar to standard passwords (8 to 10 characters, no spaces). This mistake significantly decreases the key space of the user passphrase, making it easier for an adversary to attack the system.
- Although test participants were educated and experienced email users, only a third could correctly sign and encrypt a message within 90 minutes.

In short, this study showed that most people couldn't effectively use PGP due to usability problems, rather than simple technical failings. Whitten and Tygar concluded that generic usability standards aren't necessarily applicable to security applications and suggested a need for new guidelines for creating usable security applications.

Six years later, Simpson Garfinkel and Robert Miller repeated this study,[18] but focused on S/MIME with *key continuity management* (KCM) instead of PGP. Although many email clients natively support S/MIME, certificate acquisition is a burdensome process and is likely why more people don't use it. KCM addresses this problem by automatically creating a public-private key pair whenever a user creates a new email identity. The results showed that KCM was effective at stopping impersonation attacks, but not as useful against new identity (that is, phishing) attacks. In general, KCM was an improvement over current email encryption techniques, but it wasn't a perfect solution, highlighting the continuing problems in this application domain.

Usable Security Design

Passwords, authentication, and email encryption are considered "canonical" focus areas for usable security. Another significant question is what makes a new design better — or worse — from a usable security perspective. First let's look at design guidelines that are intended to improve security-oriented design.

Design Guidelines

Although several researchers have looked at usable security design, Kai-Ping Yee's work in 2002 is among the most cited.[19] His guidelines focus on addressing *valid* and *nontrivial* concerns specific to usable security design; this list is an updated version from Yee's Web site:[20]

- *Path of least resistance*. Match the most comfortable way to do tasks with the least granting of authority.
- *Active authorization*. Grant authority to others in accordance with user actions indicating consent.
- *Revocability*. Offer the user ways to reduce others' authority to access the user's resources.
- *Visibility*. Maintain accurate awareness of others' authority as relevant to user decisions.
- *Self-awareness*. Maintain accurate awareness of the user's own authority to access resources.
- *Trusted path*. Protect the user's channels to agents that manipulate authority on the user's behalf.
- *Expressiveness*. Enable the user to express safe security policies in terms that fit the user's task.
- *Relevant boundaries*. Draw distinctions among

objects and actions along boundaries relevant to the task.
- *Identifiability*. Present objects and actions using distinguishable, truthful appearances.
- *Foresight*. Indicate clearly the consequences of decisions that the user is expected to make.

As Garfinkel states, "There are of course no set of rules, principles or formalisms that, when followed, are guaranteed to produce usable computer systems. If such rules existed, we would almost certainly all be using them, and the usability problem would be solved."[16] Yet, even though these guidelines don't address and solve every issue, they can help remind designers of good practices. To that end, they can be very useful.

Let's now examine successful and flawed designs — for each case, we can refer back to Yee's guidelines to see if a correlation exists between adherence to these guidelines and successful design.

Successful Designs

Before looking at specific examples, we first consider what makes a design successful. Here, we're most interested in user acceptance and not business considerations or market pressures. With this in mind, we can consider the following examples to be successful designs because they were viewed favorably in user testing and showed significant improvement in users' ability to achieve appropriate security levels. The first example we review is an improvement to the file permissions interface in Microsoft Windows XP. The second is a system that enables rapid setup and enrollment in a secure wireless network.

Salmon file permissions interface. Correctly setting file permissions in an enterprise environment is a difficult task. First, users must understand the different groups and the implications of assigning permissions to each group. Second, they must understand numerous file permission types, such as full control, modify, read-and-execute, read, write, and special permissions. (In Windows XP, these are the standard permission options — clicking on the Advanced tab adds even more complexity.) Finally, complex interactions can create unexpected end results when setting permissions.

Robert Reeder and Roy Maxion developed Salmon to replace the traditional Windows XP file permissions interface (see Figure 3).[21] Results from their user study show that Salmon delivers a four-fold increase in accuracy and a 94 percent reduction in errors committed. The primary design concept Salmon uses is *anchor-based subgoaling* (ABS), "a principle for ensuring that a user interface provides all the information a user will need to complete the tasks for which the interface is intended, and provides such information in a clear and accurate display that the user will notice." When it comes to setting file permissions, the effective permissions are difficult to determine in Windows XP because the native interface requires the user to drill down through the Advanced button to find them. Salmon addresses this problem by making relevant information more readily available, letting users make informed decisions about file permission settings.

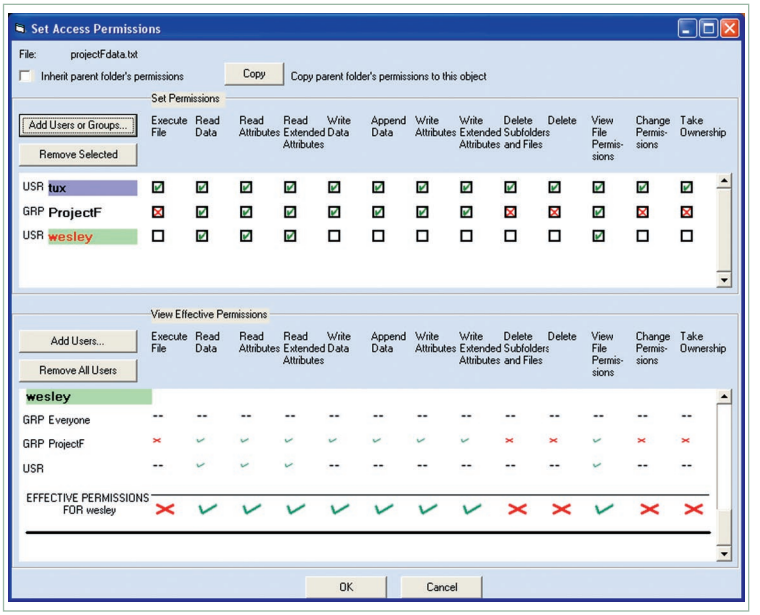

Figure 3. The Salmon interface. Salmon provides users with all relevant information in a single window, letting them make informed security decisions.[21]

Salmon successfully addresses all but one of Yee's design guidelines: by providing all the relevant access-control information in a single user interface, it has visibility, self-awareness, revocability, and foresight. Given that access controls are associated with a distinct object, Salmon has relevant boundaries and identifiability. The access-control granularity — an underlying feature of the operating system that Salmon didn't hinder — provides for expressiveness and active authorization. Finally, the reduction in human error shows that Salmon

achieves a path of least resistance. Salmon doesn't offer a trusted path because users don't get assurance that the system itself actually carries out their actions.

Salmon is clearly a successful example of how simply changing an interface's design can result in better user security. Next, we consider a system that required more than interface changes to reach its goals.

Network-in-a-Box. From a security expert viewpoint, wireless network security has suffered from many problems in recent years. The initial form of security available on 802.11b networks, known as Wired Equivalent Privacy (WEP), had many vulnerabilities.[22,23] As subsequent standards improved wireless networks' security posture, it became increasingly difficult for average users to set up such networks properly. Today, home users attempting to set up a secure wireless network need to understand numerous security standards and protocols as well as the interactions between them. Seeing this situation, Dirk Balfanz and his colleagues designed a system called Network-in-a-Box (NiaB)[24] that reframes the problem away from setting keys and acquiring certificates and toward the more fundamental task of getting on a wireless network. Thus, NiaB uses a gesture-based user interface in which users point a wireless device at an access point to initiate a secure connection between them; this initial exchange passes SHA-1 digests of 802.1x certificates and the wireless network name over an infrared connection. The remainder of the connection setup occurs over the 802.11a/b/g wireless link. User studies that the NiaB paper discusses confirm that this idea is viable. Users were able to set up the network roughly 10 times faster (51 seconds versus 9 minutes 39 seconds for a commercial AP), and they had higher confidence and satisfaction with the process.

The difference between a poor interface and a good interface can influence users' ability to perform tasks securely.

Like Salmon, NiaB closely adheres to Yee's design guidelines. Because it grants access to the network on a per-user basis, it has revocability, visibility, self-awareness, relevant boundaries, and foresight. The gesture-based setup mechanism provides a path of least resistance, active authorization, identifiability, and foresight. A wireless access point simply provides network access, and, because NiaB can do this on a per-user basis, it has expressiveness. However, as with Salmon, NiaB doesn't offer a trusted path because the initial key exchange is performed over an unsecured infrared connection.

Both NiaB and Salmon successfully improved the user experience and achieved a more secure system configuration. However, Salmon provided a new user interface, whereas NiaB provided a new system architecture with many modifications to existing protocols.

Flawed Designs

The difference between a poor interface and a good interface can influence users' ability to perform tasks securely. Here, we consider two designs that didn't promote sound security decisions, as shown via user studies (which doesn't mean these designs are complete failures, only that their usability properties might lead to bad security decision making).

Kazaa. In 2003, Nathan Good and Aaron Krekelberg performed a user study on the Kazaa interface and found some potentially serious usability problems.[25] Most peer-to-peer (P2P) file-sharing applications, including Kazaa, provide a mechanism for downloading files and sharing your own. In Kazaa, the download folder is shared by default, and users can also select additional folders to be shared. The first problem with this setup is that users might need to refer to multiple locations to see all externally visible files because the download folder isn't displayed on the shared folders interface. Next, the software automatically shares any selected folder's subfolders. Thus, if the user chooses to store his or her downloaded files in c:\, then the entire c: drive is accessible to anyone on the Internet. Finally, the Kazaa architecture lets users search for all shared files, making it easy for someone to look for sensitive data such as email, financial information, calendars, and address books. Looking at this setup, we can see how users could be mistaken about exactly what information Kazaa is sharing from their computer. Good and Krekelberg's study showed that "only

2 of 12 users were able to determine correctly the files and folders that were being shared." Given the ramifications of improper configuration, this is a situation in which the interface design resulted in security and privacy violations against users themselves.

Kazaa's interface satisfies fewer of Yee's guidelines than the successful examples discussed previously. Kazaa does provide revocability, assuming the user knows where to click. Also, because it uses the familiar directory metaphor, it provides relevant boundaries and identifiability. Self-awareness is provided implicitly because a user always has access to his or her own files. However, given that multiple locations control file access, Kazaa fails to provide active authorization, visibility, and foresight. And because subdirectories are automatically shared, it isn't expressive. The default settings share files, so it also fails on the path of least resistance guideline. Finally, there is no trusted path.

Similar to the Windows XP file permissions interface, it seems likely that Kazaa's problems could be fixed through interface changes alone. However, the next example shows a system with design flaws that go deeper.

Eudora PGP Encryption Plug-in. In our earlier discussion of email encryption usability studies, we discussed a study of PGP 5.0 that specifically looked at the PGP plug-in for Eudora on the Macintosh platform.[17] Although a cognitive walkthrough of the interface suggested several areas for improvement, many user errors that occurred during the study seemed to be more fundamental. In other words, users didn't understand the various metaphors in public-key cryptography: they encrypted with the wrong keys, had difficulty publishing their public keys correctly, or didn't know if they should trust keys from the key server. However, we might question why users should need to understand public-key cryptography at all. Shouldn't sending secure email be as simple as pressing the Send button? Similar to the Kazaa interface, the Eudora PGP plug-in only satisfies a few of Yee's guidelines, including active authorization, visibility, and expressiveness, because individual emails can be encrypted for each recipient. It also satisfies the self-awareness guideline because users can always read their own emails and any messages received from others. However, the plug-in fails to properly address the other guidelines — encrypting emails is much harder than sending them unencrypted, which violates the path of least resistance. Once sent, users can't revoke emails, and they don't understand cryptography metaphors enough to use the interface correctly, breaking the relevant boundaries, identifiability, and foresight guidelines. Finally, like the other designs reviewed here, there is no trusted path.

Our previous discussion of an email encryption system with S/MIME and KCM[18] showed better results than the PGP plug-in, perhaps in part because the system's users didn't need to understand much more than how to click on an "Encrypt" or "Sign" button before sending the email. The design problems seen with this plug-in seem to run deeper than the interface. Much like the wireless security problems NiaB addresses, the issue of email security might be better approached when designers solve problems through reframing. We must step back and look at email itself as the task in question, rather than simply providing a better interface for key management.

The examples we've presented illustrate how system design can greatly influence the user's ability to make appropriate security decisions. In particular, the two successful design cases made it easy for users to achieve their desired security goals, whereas the two flawed ones made it difficult for even very motivated users to operate securely. Table 1 shows how each design satisfies Yee's guidelines. The determination of which guidelines each design satisfied is, to a certain extent, subjective — this is unavoidable because Yee created the guidelines as general suggestions for usable security design and didn't necessarily intended them to be used as a grading or scoring criteria. However, using them in this way can provide a systematic way for us to evaluate two very different designs.

As we can see from the table, even though neither Salmon nor NiaB achieved a trusted path, they satisfied all the remaining criteria. Certainly this explains, in part, why these designs received high user satisfaction. Kazaa and the PGP plug-in satisfied far fewer of the guidelines and were less successful.

More generally, however, the examples we

Table 1. System designs and how they satisfy Kai-Ping Yee's guidelines for usable security.

	Salmon	Network-in-a-Box	Kazaa	Pretty Good Privacy Plug-in
Path of least resistance	✗	✗		
Active authorization	✗	✗		✗
Revocability	✗	✗	✗	
Visibility	✗	✗		✗
Self-awareness	✗	✗	✗	✗
Trusted path				
Expressiveness	✗	✗		✗
Relevant boundaries	✗	✗	✗	
Identifiability	✗	✗	✗	
Foresight	✗	✗		

chose for this article show that we can approach security problems in multiple ways. Salmon is an example of a successful redesign focused purely at the user-interface level. NiaB achieves its results not through interface tweaks but by reframing the problem away from a security-oriented task and toward a more general task (getting on the wireless network).

We believe that many of the more difficult challenges in usable security require such a shift in thinking. In domains in which the problem is fundamentally scoped around security, we might have no other option than to work to improve the interface to that security task. In others, however, simply putting a nicer coat of paint on a fundamentally unworkable set of abstractions is unlikely to lead to success. In such cases, broadening the definition of the task to change the assumptions involved could open up new design opportunities.

References

1. R. Morris and K. Thompson, "Password Security: A Case History," *Comm. ACM*, vol. 22, no. 11, 1979, pp. 594–597.
2. S. Wiedenback et al., "Authentication using Graphical Passwords: Effects of Tolerance and Image Choice," *Proc. Symp. Usable Privacy and Security*, ACM Press, 2005, pp. 1–12.
3. S.N.A. Porter, "A Password Extension for Improved Human Factors," *Computers & Security*, vol. 1, no. 1, 1982, pp. 54–56.
4. J.A. Haskett, "Pass-Algorithms: A User Validation Scheme Based on Knowledge of Secret Algorithms," *Comm. ACM*, vol. 27, no. 8, 1984, pp. 777–781.
5. B.F. Barton and M.S. Barton, "User-Friendly Password Methods for Computer-Mediated Information Systems," *Computers & Security*, vol. 3, no. 3, 1984, pp. 186–195.
6. M. Zviran and W.J. Haga, "Cognitive Passwords: The Key to Easy Access Control," *Computers & Security*, vol. 9, no. 8, 1990, pp. 723–736.
7. S. Brostoff and A.M. Sasse, "Are Passfaces More Usable than Passwords? A Field Trial Investigation," *Proc. Human-Computer Interactions* (CHI 00), ACM Press, 2000, pp. 405–424.
8. I. Jermyn et al., "The Design and Analysis of Graphical Passwords," *Proc. 9th Usenix Security Symp.*, Usenix Assoc., 2000, pp. 1–14.
9. R. Dhamija and A. Perrig, "Deja Vu: A User Study Using Images for Authentication," *Proc. 9th Usenix Security Symp.*," Usenix Assoc., 2000, pp. 45–58.
10. D. Davis, F. Monrose, and M.K. Reiter, "On User Choice in Graphical Password Schemes," *Proc. 13th Usenix Security Symp.*, Usenix Assoc., 2004, pp. 151–164.
11. J. Thorpe and P.C.V. Oorschot, "Graphical Dictionaries and the Memorable Space of Graphical Passwords," *Proc. 13th Usenix Security Symp.*, Usenix Assoc., 2004, pp. 135–150.
12. S.T. Kent, "Internet Privacy Enhanced Mail," *Comm. ACM*, vol. 36, no. 8, 1993, pp. 48–60.
13. L. Lundblade, "A Review of E-Mail Security Standards," *Proc. 7th Ann. Conf. Internet Soc.* (INET 97), 1997; www.isoc.org/inet97/proceedings/A4/A4_1.HTM.
14. S. Garfinkel, "Signed, Sealed and Delivered," *CSO Online*, April 2004, www.csoonline.com/read/040104/shop.html.
15. B. Schneier, *Applied Cryptography*, 2nd ed., Wiley, 1996.
16. S.L. Garfinkel, *Design Principles and Patterns for Computer Systems that Are Simultaneously Secure and Usable*, PhD thesis, Mass. Inst. of Technology, 2005.
17. A. Whitten and J.D. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," *Proc. 8th Usenix Security Symp.*, Usenix Assoc., 1999, pp. 169–184.
18. S.L. Garfinkel and R.C. Miller, "Johnny 2: A User Test of Key Continuity Management with S/MIME and Out-

look Express," *Proc. Symp. Usable Privacy and Security*, ACM Press, 2005, pp. 13–24.
19. K.-P. Yee, "User Interaction Design for Secure Systems," *Proc. 4th Int'l Conf. Information and Communications Security*, Springer-Verlag, 2002, pp. 278–290.
20. K.-P. Yee, "Secure Interaction Design," 2007, http://zesty.ca/sid.
21. R.W. Reeder and R.A. Maxion, "User Interface Dependability through Goal-Error Prevention," *Proc. Int'l Conf. Dependable Systems & Networks*, IEEE CS Press, 2005, pp. 60–69.
22. N. Borizov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11," *Proc. Int'l Conf. Mobile Computing and Networking* (Mobicom 01), ACM Press, 2001, pp. 180–189.
23. S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Proc. 8th Ann. Workshop Selected Areas in Cryptography*, Springer, 2001, pp. 1–24.
24. D. Balfanz et al., "Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute," *Usenix Security Symp.*, Usenix Assoc., 2004, pp. 207–222.
25. N.S. Good and A. Krekelberg, "Usability and Privacy: A Study of Kazaa P2P File-Sharing," *Proc. Human-Computer Interactions* (CHI 03), vol. 5, ACM Press, 2003, pp. 137–144.

Bryan D. Payne is a research scientist in the School of Computer Science at the Georgia Institute of Technology, and is working toward his PhD in computer science. His current research focuses on techniques for monitoring and analyzing the memory of systems running inside virtual machines. Payne has an MS in computer science from the University of Maryland. Contact him at bdpayne@cc.gatech.edu.

W. Keith Edwards is an associate professor in the School of Interactive Computing at the Georgia Institute of Technology. His research focuses on bringing a human-centered perspective to lower-layer technology concerns such as security and networking infrastructure. He is especially interested in increasing the ability of home users to better manage their own online security. Edwards has a PhD in computer science from Georgia Institute of Technology. Contact him at keith@cc.gatech.edu.