Anonymity Loves Company: Usability and the Network Effect

Roger Dingledine
The Free Haven Project
arma@freehaven.net

Nick Mathewson
The Free Haven Project
nickm@freehaven.net

January 2, 2005

Other chapters in this book have talked about how usability impacts security. One class of security software is anonymizing networks—overlay networks on the Internet that provide privacy by letting users transact (for example, fetch a web page or send an email) without revealing their communication partners.

In this chapter, we'll focus on the *network effects* of usability on privacy and security: usability is a factor as before, but the size of the user base also becomes a factor. As we will see, in anonymizing networks, even if you were smart enough and had enough time to use every system perfectly, you would nevertheless be right to choose your system based in part on its usability for *other* users.

1 Usability for others impacts your security

While security software is the product of developers, the security it provides is a collaboration between developers and users. It's not enough to make software that *can* be used securely; software that is hard to use often suffers in its security as a result.

For example, suppose there are two popular mail encryption programs: HeavyCrypto, which is more secure (when used correctly), and LightCrypto, which is easier to use. Suppose you can use either one, or both. Which should you choose?

You might decide to use HeavyCrypto, since it protects your secrets better. But if you do, it's likelier that when your friends send you confidential email, they'll make a mistake and encrypt it badly or not at all. With LightCrypto, you can at least be more certain that all your friends' correspondence with you will get some protection.

What if you used *both* programs? If your tech-savvy friends use Heavy-Crypto, and your less sophisticated friends use LightCrypto, then everybody will get as much protection as they can. But can all your friends really judge how able they are? If not, then by supporting a less usable option, you've made it likelier that your non-savvy friends will shoot themselves in the foot.

The crucial insight here is that for email encryption, security is a collaboration between multiple people: both the sender and the receiver of a secret email must work together to protect its confidentiality. Thus, in order to protect your own security, you need to make sure that the system you use is not only usable by yourself, but by the other participants as well.

This observation doesn't mean that it's always better to choose usability over security, of course: if a system doesn't address your threat model, no amount of usability can make it secure. But conversely, if the people who need to use a system can't or won't use it correctly, its ideal security properties are irrelevant.

Hard-to-use programs and protocols can hurt security in many ways:

- Programs with *insecure modes of operation* are bound to be used unknowingly in those modes.
- *Optional security*, once disabled, is often never re-enabled. For example, many users who ordinarily disable browser cookies for privacy reasons wind up re-enabling them so they can access sites that require cookies, and later leaving cookies enabled for all sites.
- *Badly labeled off switches* for security are even worse: not only are they more prone to accidental selection, but they're more vulnerable to social attackers who trick users into disabling their security. As an example, consider the page-long warning your browser provides when you go to a website with an expired or otherwise suspicious SSL certificate.
- *Inconvenient* security is often abandoned in the name of day-to-day efficiency: people often write down difficult passwords to keep from forgetting them, and share passwords in order to work together.
- Systems that provide a *false sense of security* prevent users from taking real measures to protect themselves: breakable encryption on ZIP archives, for example, can fool users into thinking that they don't need to encrypt email containing ZIP archives.
- Systems that provide *bad mental models* for their security can trick users into believing they are more safe than they really are: for example, many users interpret the "lock" icon in their web browsers to mean "You can safely enter personal information," when its meaning is closer to "Nobody can read your information on its way to the named website."[1]

2 Usability is even more important for privacy

Usability affects security in systems that aim to protect data confidentiality. But when the goal is *privacy*, it can become even more important. A large category of *anonymizing networks*, such as Tor, JAP, Mixminion, and Mixmaster, aim to

[1]Or more accurately, "Nobody can read your information on its way to someone who was able to convince one of the dozens to hundreds of CAs configured in your browser that they are the named website, or who was able to compromise the named website later on. Unless your computer has been compromised already."

hide not only what is being said, but also who is communicating with whom, which users are using which websites, and so on. These systems have a broad range of users, including ordinary citizens who want to avoid being profiled for targeted advertisements, corporations who don't want to reveal information to their competitors, and law enforcement and government intelligence agencies who need to do operations on the Internet without being noticed.

Anonymity networks work by hiding users among users. An eavesdropper might be able to tell that Alice, Bob, and Carol are all using the network, but should not be able to tell which of them is talking to Dave. This property is summarized in the notion of an *anonymity set*—the total set of people who, so far as the attacker can tell, might be the one engaging in some activity of interest. The larger the set, the more anonymous the participants.[2] When more users join the network, existing users become more secure, even if the new users never talk to the existing ones! [1, 2] Thus, "anonymity loves company."[3]

In a data confidentiality system like PGP, Alice and Bob can decide by themselves that they want to get security. As long as they both use the software properly, no third party can intercept the traffic and break their encryption. However, Alice and Bob can't get anonymity by themselves: they need to participate in an infrastructure that coordinates users to provide cover for each other.

No organization can build this infrastructure for its own sole use. If a single corporation or government agency were to build a private network to protect its operations, any connections entering or leaving that network would be obviously linkable to the controlling organization. The members and operations of that agency would be easier, not harder, to distinguish.

Thus, to provide anonymity to any of its users, the network must accept traffic from external users, so the various user groups can blend together.

In practice, existing commercial anonymity solutions (like Anonymizer.com) are based on a set of single-hop proxies. In these systems, each user connects to a single proxy, which then relays the user's traffic. Single proxies provide comparatively weak security, since a compromised proxy can trivially observe all of its users' actions, and an eavesdropper only needs to watch a single proxy to perform timing correlation attacks against all its users' traffic. Worse, all users need to trust the proxy company to have good security itself as well as to not reveal user activities.

The solution is distributed trust: an infrastructure made up of many independently controlled proxies that work together to make sure no transaction's privacy relies on any single proxy. With distributed-trust anonymity networks like the ones discussed in this chapter, users build tunnels or *circuits* through

[2] Assuming that all participants are equally plausible, of course. If the attacker suspects Alice, Bob, and Carol equally, Alice is more anonymous than if the attacker is 98% suspicious of Alice and 1% suspicious of Bob and Carol, even though the anonymity sets are the same size. Because of this imprecision, recent research is moving beyond simple anonymity sets to more sophisticated measures based on the attacker's confidence.

[3] This catch-phrase was first made popular in our context by the authors of the Crowds [8] anonymity network.

a series of servers. They encrypt their traffic in multiple layers of encryption, and each server removes a single layer of encryption. No single server knows the entire path from the user to the user's chosen destination. Therefore an attacker can't break the user's anonymity by compromising or eavesdropping on any one server.

Despite their increased security, distributed-trust anonymity networks have their disadvantages. Because traffic needs to be relayed through multiple servers, performance is often (but not always) worse. Also, the software to implement a distributed-trust anonymity network is significantly more difficult to design and implement.

Beyond these issues of the architecture and ownership of the network, however, there is another catch. For users to keep the same anonymity set, they need to act like each other. If Alice's client acts completely unlike Bob's client, or if Alice's messages leave the system acting completely unlike Bob's, the attacker can use this information. In the worst case, Alice's messages stand out entering and leaving the network, and the attacker can treat Alice and those like her as if they were on a separate network of their own. But even if Alice's messages are only recognizable as they leave the network, an attacker can use this information to break exiting messages into "messages from User1," "messages from User2," and so on, and can now get away with linking messages to their senders as groups, rather than trying to guess from individual messages. Some of this *partitioning* is inevitable: if Alice speaks Arabic and Bob speaks Bulgarian, we can't force them both to learn English in order to mask each other.

What does this imply for usability? More so than with encryption systems, users of anonymizing networks may need to choose their systems based on how usable others will find them, in order to get the protection of a larger anonymity set.

3 Case study: usability means users, users mean security

We'll consider an example. Practical anonymizing networks fall into two broad classes. *High-latency* networks like Mixminion or Mixmaster can resist strong attackers who can watch the whole network and control a large part of the network infrastructure. To prevent this "global attacker" from linking senders to recipients by correlating when messages enter and leave the system, high-latency networks introduce large delays into message delivery times, and are thus only suitable for applications like email and bulk data delivery—most users aren't willing to wait half an hour for their web pages to load. *Low-latency* networks like Tor, on the other hand, are fast enough for web browsing, secure shell, and other interactive applications, but have a weaker threat model: an attacker who watches or controls both ends of a communication can trivially correlate message timing and link the communicating parties.

Clearly, users who need to resist strong attackers must choose high-latency networks or nothing at all, and users who need to anonymize interactive applications must choose low-latency networks or nothing at all. But what should flexible users choose? Against an unknown threat model, with a non-interactive application (such as email), is it more secure to choose security or usability?

Security, we might decide. If the attacker turns out to be strong, then we'll prefer the high-latency network, and if the attacker is weak, then the extra protection doesn't hurt.

But since many users might find the high-latency network inconvenient, suppose that it gets few actual users—so few, in fact, that its maximum anonymity set is too small for our needs. In this case, we need to pick the low-latency system, since the high-latency system, though it always protects us, never protects us enough; whereas the low-latency system can give us enough protection against at least *some* attackers.

This decision is especially messy because even the developers who implement these anonymizing networks can't recommend which approach is safer, since they can't predict how many users each network will get and they can't predict the capabilities of the attackers we might see in the wild. Worse, the anonymity research field is still young, and doesn't have many convincing techniques for measuring and comparing the protection we get from various situations. So even if the developers or users could somehow divine what level of anonymity they require and what their expected attacker can do, the researchers still don't know what parameter values to recommend.

4 Case study: against options

Too often, designers faced with a security decision bow out, and instead leave the choice as an option: protocol designers leave implementors to decide, and implementors leave the choice for their users. This approach can be bad for security systems, and is nearly always bad for privacy systems.

With security:

- Extra options often delegate security decisions to those least able to understand what they imply. If the protocol designer can't decide whether the AES encryption algorithm is better than the Twofish encryption algorithm, how is the end user supposed to pick?
- Options make code harder to audit by increasing the volume of code, by increasing the number of possible configurations *exponentially*, and by guaranteeing that non-default configurations will receive little testing in the field. If AES is always the default, even with several independent implementations of your protocol, how long will it take to notice if the Twofish implementation is wrong?

Most users stay with default configurations as long as they work, and only reconfigure their software as necessary to make it usable. For example, suppose the developers of a web browser can't decide whether to support a given extension with unknown security implications, so they leave it as a user-adjustable option, thinking that users can enable or disable the extension based on their security needs. In reality, however, if the extension is enabled by default, nearly all users will leave it on whether it's secure or not; and if the extension is disabled by default, users will tend to enable it based on their perceived demand for the extension rather than their security needs. Thus, only the most savvy and security-conscious users—the ones who know more about web security than the developers themselves—will actually wind up understanding the security implications of their decision.

The real issue here is that designers often end up with a situation where they need to choose between 'insecure' and 'inconvenient' as the default configuration—meaning they've already made a mistake in designing their application. (This issue is discussed more in chapters X and Y.)

Of course, when end users *do* know more about their individual security requirements than application designers, then adding options is beneficial, especially when users describe their own situation (home or enterprise; shared versus single-user host) rather than trying to specify what the program should do about their situation.

In privacy applications, superfluous options are even worse. When there are many different possible configurations, eavesdroppers and insiders can often tell users apart by which settings they choose. For example, the Type I or "Cypherpunk" anonymous email network uses the OpenPGP encrypted message format, which supports many symmetric and asymmetric ciphers. Because different users prefer different ciphers, and because different versions of encryption programs implementing OpenPGP (such as PGP and GnuPG) use different cipher suites, users with uncommon preferences and versions stand out from the rest, and get little privacy at all. Similarly, Type I allows users to pad their messages to a fixed size so that an eavesdropper can't correlate the sizes of messages passing through the network—but it forces the user to decide what size of padding to use! Unless a user can guess which padding size will happen to be most popular, the option provides attackers with another way to tell users apart.

Even when users' needs genuinely vary, adding options does not necessarily serve their privacy. In practice, the default option usually prevails for casual users, and therefore needs to prevail for security-conscious users *even when it would not otherwise be their best choice.* For example, when an anonymizing network allows user-selected message latency (like the Type I network does), most users tend to use whichever setting is the default, so long as it works. Of the fraction of users who change the default at all, most will not, in fact, understand the security implications; and those few who do will need to decide whether the increased traffic-analysis resistance that comes with more variable latency is worth the decreased anonymity that comes from splitting away from the bulk of the user base.

5 Case study: Mixminion and MIME

We've argued that providing too many observable options can hurt privacy, but we've also argued that focusing too hard on privacy over usability can hurt privacy itself. What happens when these principles conflict?

We encountered such a situation when designing how the Mixminion anonymous email network [5] should handle MIME-encoded data. MIME (Multipurpose Internet Mail Extensions) is the way a mail client tells the receiving mail client about attachments, which character set was used, and so on. As a standard, MIME is so permissive and flexible that different email programs are almost always distinguishable by which subsets of the format, and which types of encodings, they choose to generate. Trying to "normalize" MIME by converting all mail to a standard only works up to a point: it's trivial to convert all encodings to *quoted-printable*, for example, or to impose a standard order for *multipart/alternative* parts; but demanding a uniform list of formats for *multipart/alternative* messages, normalizing HTML, stripping identifying information from Microsoft Office documents, or imposing a single character encoding on each language would likely be an impossible task.

Other possible solutions to this problem could include limiting users to a single email client, or simply banning email formats other than plain 7-bit ASCII. But these procrustean approaches would limit usability, and turn users away from the Mixminion network. Since fewer users mean less anonymity, we must ask whether users would be better off in a larger network where their messages are likelier to be distinguishable based on email client, or in a smaller network where everyone's email formats look the same.

Some distinguishability is inevitable anyway, since users differ in their interests, languages, and writing styles: if Alice writes about astronomy in Amharic, her messages are unlikely to be mistaken for Bob's, who writes about botany in Basque. Also, any attempt to restrict formats is likely to backfire. If we limited Mixminion to 7-bit ASCII, users wouldn't stop sending each other images, PDF files, and messages in Chinese: they would instead follow the same evolutionary path that led to MIME in the first place, and encode their messages in a variety of distinguishable formats, with each client software implementation having its own *ad hoc* favorites. So imposing uniformity in this place would not only drive away users, but would probably fail in the long run, and lead to fragmentation at least as dangerous as we were trying to avoid.

We also had to consider threat models. To take advantage of format distinguishability, an attacker needs to observe messages leaving the network, and either exploit prior knowledge of suspected senders ("Alice is the only user who owns a 1995 copy of Eudora"), or feed message format information into traffic analysis approaches ("Since half of the messages to Alice are written in English, I'll assume they mostly come from different senders than the ones in Amharic."). Neither attack is certain or easy for all attackers; even if we can't defeat them in the worst possible case (where the attacker knows, for example, that only one copy of LeetMailPro was ever sold), we can provide vulnerable users with protection against weaker attackers.

In the end, we compromised: we perform as much normalization as we can, and warn the user about document types such as MS Word that are likely to reveal identifying information, but we do not forbid any particular format or client software. This way, users are informed about how to blend with the largest possible anonymity set, but users who prefer to use distinguishable formats rather than nothing at all still receive and contribute protection against certain attackers.

6 Case study: Tor Installation

Usability and marketing have also proved important in the development of Tor, a low-latency anonymizing network for TCP traffic. The technical challenges Tor has solved, and the ones it still needs to address, are described in its design paper [6], but at this point many of the most crucial challenges are in adoption and usability.

While Tor was in it earliest stages, its user base was a small number of fairly sophisticated privacy enthusiasts with experience running Unix services, who wanted to experiment with the network (or so they say; by design, we don't track our users). As the project gained more attention from venues including security conferences and articles on Slashdot.org and Wired News, we added more users with less technical expertise. These users can now provide a broader base of anonymity for high-needs users, but only when they receive good support themselves.

For example, it has proven difficult to educate less sophisticated users about DNS issues. Anonymizing TCP streams (as Tor does) does no good if applications reveal where they are about to connect by first performing a non-anonymized hostname lookup. To stay anonymous, users need either to configure their applications to pass hostnames to Tor directly by using SOCKS4a or the hostname-based variant of SOCKS5; to manually resolve hostnames with Tor and pass the resulting IPs to their applications; or to direct their applications to application-specific proxies which handle each protocol's needs independently.

None of these is easy for an unsophisticated user, and when they misconfigure their systems, they not only compromise their own privacy, but also provide no cover for the users who *are* configured correctly: if Bob leaks a DNS request whenever he is about to connect to a website, an observer can tell that anybody connecting to Alice's website anonymously must not be Bob. Thus, experienced users have an interest in making sure inexperienced users can use the system correctly. Tor being hard to configure is a weakness for everybody.

We've tried a few solutions that didn't work as well as we hoped. Improving documentation only helped the users who read it. We changed Tor to warn users who provided an IP address rather than a hostname, but this warning usually resulted in several email exchanges to explain DNS to the casual user, who had typically no idea how to solve his problem.

At the time of this writing, the most important solutions for these users have been improve Tor's documentation for how to configure various applications to use Tor; to change the warning messages to refer users to a description of the solution ("You are insecure. See this webpage.") instead of a description of the problem ("Your application is sending IPs instead of hostnames, which may leak information. Consider using SOCKS4a instead."); and to bundle Tor with the support tools that it needs, rather than relying on users to find and configure them on their own.

7 Case study: JAP and its anonym-o-meter

The Java Anon Proxy (JAP) is a low-latency anonymizing network for web browsing developed and deployed by the Technical University of Dresden in Germany [3]. Unlike Tor, which uses a *free-route* topology where each user can choose where to enter the network and where to exit, JAP has fixed-route *cascades* that aggregate user traffic into a single entry point and a single exit point.

The JAP client includes a GUI (screenshot in Figure 1). Screenshot:

`http://anon.inf.tu-dresden.de/img/screen_en.jpg`

Notice the 'anonymity meter' giving the user an impression of the level of protection for his current traffic.

How do we decide the value that the anonym-o-meter should report? In JAP's case, it's based on the number of other users traveling through the cascade at the same time. But alas, since JAP aims for quick transmission of bytes from one end of the cascade to the other, it falls prey to the same end-to-end timing correlation attacks as we described above. That is, an attacker who can watch both ends of the cascade won't actually be distracted by the other users [4]. The JAP team has plans to implement full-scale padding from every user (sending and receiving packets all the time even when they have nothing to send), but—for usability reasons—they haven't gone forward with these plans.

As the system is now, anonymity sets don't provide a real measure of security for JAP, since any attacker who can watch both ends of the cascade wins, and the number of users on the network is no obstacle to this attack. However, we think the anonym-o-meter is a great way to present security information to the user, and we hope to see a variant of it deployed one day for a high-latency system like Mixminion, where the amount of current traffic in the system is more directly related to the protection it offers.

8 Bootstrapping, confidence, and reputability

Another area where human factors are critical in privacy is in bootstrapping new systems. Since new systems start out with few users, they initially provide only small anonymity sets. This starting state creates a dilemma: a new system with improved privacy properties will only attract users once they believe it is popular and therefore has high anonymity sets; but a system cannot be popular without attracting users. New systems need users for privacy, but need privacy for users.

Low-needs users can break the deadlock. The earliest stages of an anonymizing network's lifetime tend to involve users who need only to resist weak attackers who can't know which users are using the network and thus can't learn the contents of the small anonymity set. This solution reverses the early adopter trends of many security systems: rather than attracting first the most security-conscious users, privacy applications must begin by attracting low-needs users and hobbyists.

But this analysis relies on users' accurate perceptions of present and future anonymity set size. As in market economics, expectations themselves can bring about trends: a privacy system which people believe to be secure and popular will gain users, thus becoming (all things equal) more secure and popular. Thus, security depends not only on usability, but also on *perceived usability by others*, and hence on the quality of the provider's marketing and public relations. Perversely, over-hyped systems (if they are not too broken) may be a better choice than modestly promoted ones, if the hype attracts more users.

Yet another factor in the safety of a given network is its reputability: the perception of its social value based on its current users. If I'm the only user of a system, it might be socially accepted, but I'm not getting any anonymity. Add a thousand Communists, and I'm anonymous, but everyone thinks I'm a Commie. Add a thousand random citizens (cancer survivors, privacy enthusiasts, and so on) and now I'm hard to profile.

The more cancer survivors on Tor, the better for the human rights activists. The more script kiddies, the worse for the normal users. Thus, reputability is an anonymity issue for two reasons. First, it impacts the sustainability of the network: a network that's always about to be shut down has difficulty attracting and keeping users, so its anonymity set suffers. Second, a disreputable network attracts the attention of powerful attackers who may not mind revealing the identities of all the users to uncover the few bad ones.

While people therefore have an incentive for the network to be used for "more reputable" activities than their own, there are still tradeoffs involved when it comes to anonymity. To follow the above example, a network used entirely by cancer survivors might welcome some Communists onto the network, though of course they'd prefer a wider variety of users.

The impact of public perception on security is especially important during the bootstrapping phase of the network, where the first few widely publicized uses of the network can dictate the types of users it attracts next.

9 Technical challenges to guessing the number of users in a network

In addition to the social problems we describe above that make it difficult for a typical user to guess which anonymizing network will be most popular, there are some technical challenges as well. These stem from the fact that anonymizing networks are good at hiding what's going on—even from their users. For example, one of the toughest attacks to solve is that an attacker might sign up many users to artificially inflate the apparent size of the network. Not only does this *Sybil attack* increase the odds that the attacker will be able to successfully compromise a given user transaction [7], but it might also trick users into thinking a given network is safer than it actually is.

And finally, as we saw when discussing JAP above, the feasibility of end-to-end attacks makes it hard to guess how much a given other user is contributing to your anonymity. Even if he's not actively trying to trick you, he can still fail to provide cover for you, either because his behavior is sufficiently different from yours (he's active during the day, and you're active at night), because his transactions are different (he talks about physics, you talk about AIDS), or because network design parameters (such as low delay for messages) mean the attacker is able to track transactions more easily.

10 Bringing it all together

Users' safety relies on them behaving like other users. But how can they predict other users' behavior? If they need to behave in a way that's different from the rest of the users, how do they compute the tradeoff and risks?

There are several lessons we might take away from researching anonymity and usability. On the one hand, we might remark that anonymity is already tricky from a technical standpoint, and if we're required to get usability right as well before anybody can be safe, it will be hard indeed to come up with a good design: if lack of anonymity means lack of users, then we're stuck in a depressing loop. On the other hand, the loop has an optimistic side too. Good anonymity can mean more users: if we can make good headway on usability, then as long as the technical designs are adequate, we'll end up with enough users to make everything work out.

In any case, declining to design a good solution means leaving most users to a less secure network or no anonymizing network at all. Cancer survivors and abuse victims would continue communications and research over the Internet, risking social or employment problems; and human rights workers in oppressive countries would continue publishing their stories.

The temptation to focus on designing a perfectly usable system before building it can be self-defeating, since obstacles to usability are often unforeseen. We believe that the anonymity community needs to focus on continuing experimental deployment.

References

[1] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography*. Springer-Verlag, LNCS 2742, Jan 2003.

[2] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Information Hiding (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, 2001.

[3] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, Jul 2000.

[4] George Danezis. The traffic analysis of continuous-time mixes. In David Martin and Andrei Serjantov, editors, *Privacy Enhancing Technologies (PET 2004)*, May 2004.

[5] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE CS, May 2003.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, Aug 2004.

[7] John Douceur. The Sybil Attack. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS)*, Mar 2002.

[8] Michael Reiter and Aviel Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.