Rich Sharing for the Web

Marc Stiegler

HP Laboratories
HPL-2009-169

Keyword(s):
secure cooperation, usable security

Abstract:

We use email for a tremendous number of different purposes. Interestingly, we often use email for purposes for which other applications have been explicitly designed - email, even though not purpose-built, often "just works better". Why is this? We have identified 6 key features of sharing that are needed to support secure cooperation, features that enable the users themselves to build networks of access rights that implement the Principle of Least Authority(POLA), thereby maximizing the opportunities for cooperation among participants with limited mutual trust. Systems that do not implement these 6 features will feel rigid and inadequately functional once enough users are involved, forcing the users to seek alternate means to "work around" the limitations in those applications. Email is almost the only application on the Web that implements all six features, making it the natural fall-back application. We use email for everything because email is the only thing that works for everything. Here we describe the six features, highlight some of the consequences when the six features are not integrated, and look at the results when an application is built specifically to support the whole set in a "rich sharing" system.

External Posting Date: July 21, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: July 21, 2009 [Fulltext]

Rich Sharing for the Web

Marc Stiegler

HP Labs

marc.d.stiegler@hp.com

Abstract

We use email for a tremendous number of different purposes. Interestingly, we often use email for purposes for which other applications have been explicitly designed – email, even though not purpose-built, often "just works better". Why is this?

We have identified 6 key features of sharing that are needed to support secure cooperation, features that enable the users themselves to build networks of access rights that implement the Principle of Least Authority(POLA), thereby maximizing the opportunities for cooperation among participants with limited mutual trust. Systems that do not implement these 6 features will feel rigid and inadequately functional once enough users are involved, forcing the users to seek alternate means to "work around" the limitations in those applications. Email is almost the only application on the Web that implements all six features, making it the natural fall-back application. We use email for everything because email is the only thing that works for everything.

Here we describe the six features, highlight some of the consequences when the six features are not integrated, and look at the results when an application is built specifically to support the whole set in a "rich sharing" system.

Problem

Too many of our distributed system applications feel rigid and inadequately able to support our need to solve problems in cooperation with our teammates. Consider two real-life examples that represent failures of our systems to enable even the most basic sharing relationships.

- The employee's pay stub: The employee's pay stub is made available as an online web page that lives behind the corporate firewall and can be accessed only with the employee's single-signon password. However, suppose that the employee's spouse is the one who schedules vacations and maintains the family's financial accounts. The spouse needs access to the pay stubs (and would like to further grant access to the pay stubs to the family's tax accountant, during the tax preparation period). But to grant access, the employee would have to give the spouse both the means to hook up to the corporate VPN (to get inside the firewall) and the single-signon password. Each of these actions is necessarily a gross violation of corporate security policy, for they are gross violations of the principle of least authority (POLA): the spouse would acquire authority far in excess of what is needed, authority that could easily be abused either maliciously or by accident. In practice, today the employee resorts to the following stratagem: bring up the pay stub in the web browser, snapshot the window on the stub into a file, and email the file to the spouse (who then emails it forward to the tax accountant). This is tedious, time consuming, insecure (email is generally not encrypted, leaving this sensitive information open to MITM attacks), and error-prone. But it is the best current technology offers us.
- Simple File Sharing: Alice wishes to collaboratively edit a file with Bob. In theory, the access control list (acl) systems on our computers and networks would enable such sharing by allowing Alice to simply add an access control entry to the list that grants Bob the right to edit the file. In practice, this technique is often too awkward or is entirely nonfunctional. Suppose Bob works for another company. The sharing would not work across the firewall even if Alice could construct the appropriate access control entry – which she cannot, because Bob has no identity inside the administrative domain for which the acl system has been constructed. IT organizations may try to work around this in the most crucial circumstances by using federated identity management, but in practice, Alice will simply email the file to Bob with the note, "Bob, when you are done editing this, send it back to me."

Solution: The Six Features of Sharing

Ironically, problems like this have been solved smoothly in the physical world since before the advent of computers. We have identified six key features of sharing that are routinely implemented in the physical world. When integrated properly these features enable ordinary people to comfortably and intuitively implement good approximations of POLA, thus enabling people to work together even when they have limited trust in one another. Let us consider the six features in the context of a simple three-sentence physical world example that uses all six features at the same time:

Alice, in a race to her next meeting, turns thunder-struck to Bob and says, "Bob, I just remembered I need to get my daughter Carol's car to Dave's repair shop. I've got to go to this meeting. Can you take Carol's car over there?"

Here are the six features, and how each plays into solving this simple problem.

- Dynamic – Alice must be able to grant Carol's car-driving authority quickly and effortlessly without requiring action from Carol's, Alice's, or Bob's IT departments.
- Attenuated – Alice has Carol's car key on Alice's key ring. Alice must be able to give Bill the key to Carol's car without surrendering the entire ring of keys that include Alice's own car and Alice's house keys. I.e., Alice must be able to grant the authority without giving Bill her single signon password.
- Chained – The authority Alice is giving Bob does not originate with Alice, it originates with Carol. Alice must be able to re-delegate to Bob the authority delegated to her by Carol. Further, Bob will have to be able to re-delegate the authority again, to Dave at the repair shop. Building attenuating chains of delegation is particularly hard for traditional access control list oriented sharing. Transform this car-delegation problem into a file sharing scenario. Carol has given read-only (attenuated) authority to Alice. Now Alice needs to enable Bob to fulfill her file-reading responsibilities while Alice is in the meeting. Simplify by assuming that Carol and Alice and Bob are all behind the same firewall and are all part of the same administrative domain. Even with these simplifications, the acl system fails: Alice cannot delegate her read authority to Bob. Why? Because, in order to grant a read authority Bob, Alice would have to have access control list editing authority – which would confer to Alice *de facto* full control over the file.
- Cross domain – The car-driving authority originates in Carol's domain, passes through the domain where Alice works with Bob, and winds up at Dave's repair shop. The authority must seamlessly pass across all these domains without requiring construction of a federated identity management system for Alice Bob, Carol, and Dave. This is again difficult for traditional access-control list security, which stops at the administrative boundary where the identities cease to be recognized.
- Recomposable – Dave must be able to operate the repair shop's garage door and drive the car at the same time. If Carol's car were behind one firewall, and the garage door were behind another firewall, this would not be possible – only one VPN can used to cross one firewall at a time, otherwise a significant POLA violation may occur.
- Accountable – Accountability in the physical world has numerous subtleties. If Bob wrecks Carol's car, Carol will want to hold Alice accountable, not Bob – Carol probably doesn't even know Bob, and she has no ongoing relationship with Bob via which she can pursue remediation. On the other hand, once Bob passes the car into Dave's hands, if Dave wrecks the car, Carol will want to hold Dave responsible – despite the transitive grant of authority across Bob whom Carol does not know, Carol once again has an ongoing relationship and a means of remediation with Dave.

Given these six features, the users in the field with the most knowledge of who needs authority to do what are in position to build networks of resource access that embody the principle of least authority. Chaining is the feature that ensures that the individual principle has enough authority. Attenuation is the feature that ensures the individual principle does not have excess authority. Dynamicity, domain-crossing, and recomposability are requirements that need recognition mainly because the web has haphazardly constructed so many impediments to sharing that these features must be explicitly designed into systems in order to overcome the web's complexities.

Accountability is required because of the risks that still remain even in a POLA environment: sometimes some people need large amounts of authority to have adequate authority, and in those cases accountability is needed to remediate in case of abuse.

Henceforth we refer to sharing with all six features as *rich* sharing.

Sharing In Action

We have two applications that suggest the power of rich sharing. One is a ubiquitous existing web application, the other is an application we built principally to experiment with the quality of user experience rich sharing confers on its users.

The first example is derived directly from the 2 earlier examples of how current software systems fail to enable rich sharing, the employee's pay stub and the simple file sharing scenarios. In each case, the fallback strategy of users was to use email. This works because email integrates all six features of sharing. Email is

- Dynamic: You can send email to anyone any time.
- Attenuated: a read-only authority on an attached document is referred to as an "information copy": there is no suggestion that the sender will accept editing changes from the recipient.
- Chained: Chaining in email is called "forwarding". Note that, unlike the problem faced by the acl user for sharing a read-only authority, the email user has no problem: informational copies can be forwarded as easily as read/write ("send me your changes") copies.
- Cross-Domain: Not even the most rigid and restrictive of IT organizations can shut off email across its firewalls. No one would be able to get their work done.
- Recomposable: Data from emails from within one firewall can be seamlessly integrated with data from emails from behind another firewall.
- Accountable: If Alice asks Bob to edit a file and email it back, and Bob asks Carol to edit the file, and Bob then emails it back, Alice will hold Bob responsible if the edits are erroneous. If Carol (whom Alice may not know) emails her result directly to Alice, either Alice will ask Carol who she is before accepting the changes, or if Carol includes the history of messages in the message, Alice will directly see, once again, that she should hold Bob responsible. Email does not have accounting quite powerful enough to do all the things we expect in the physical world (with the car key example of accountability at Dave's shop), but it is better than what we get from a Windows or Unix access control list system, and gives us a taste of what correct accounting would be like.

Email is the ubiquitous application that no one can live without because it enables rich sharing.

We have also built an application that explicitly implements rich sharing for file. This application, the Secure Cooperative File Sharing (SCoopFS) system[Karp09], has been used in a pilot program to see what if any difference it makes if an application supports rich sharing. Indeed, we did get one emergent network of usage that surprised us after it had unfolded, a network that was a direct consequence of proper rich sharing support. This usage was in the SCoopFS development and deployment system itself.

Figure 1 shows a network of sharings among participants in the SCoopFS development and testing process. This network was not planned: it grew sporadically in pieces, built by the participants who were themselves unaware of the size of the network. A brief look at how this network evolved:

Alan was the user interface developer, Marc was the ui tester, and Carol ran SCoopFS Support. In the beginning, when a new version of the ui passed the Alan's preliminary testing, he would email the new version (embodied in a single flash swf file) to Marc (who was also a developer and the builder of new installations). Once SCoopFS started operating successfully, it was natural for Alan to simply share his "production" version (the version that passed preliminary testing) with Marc.

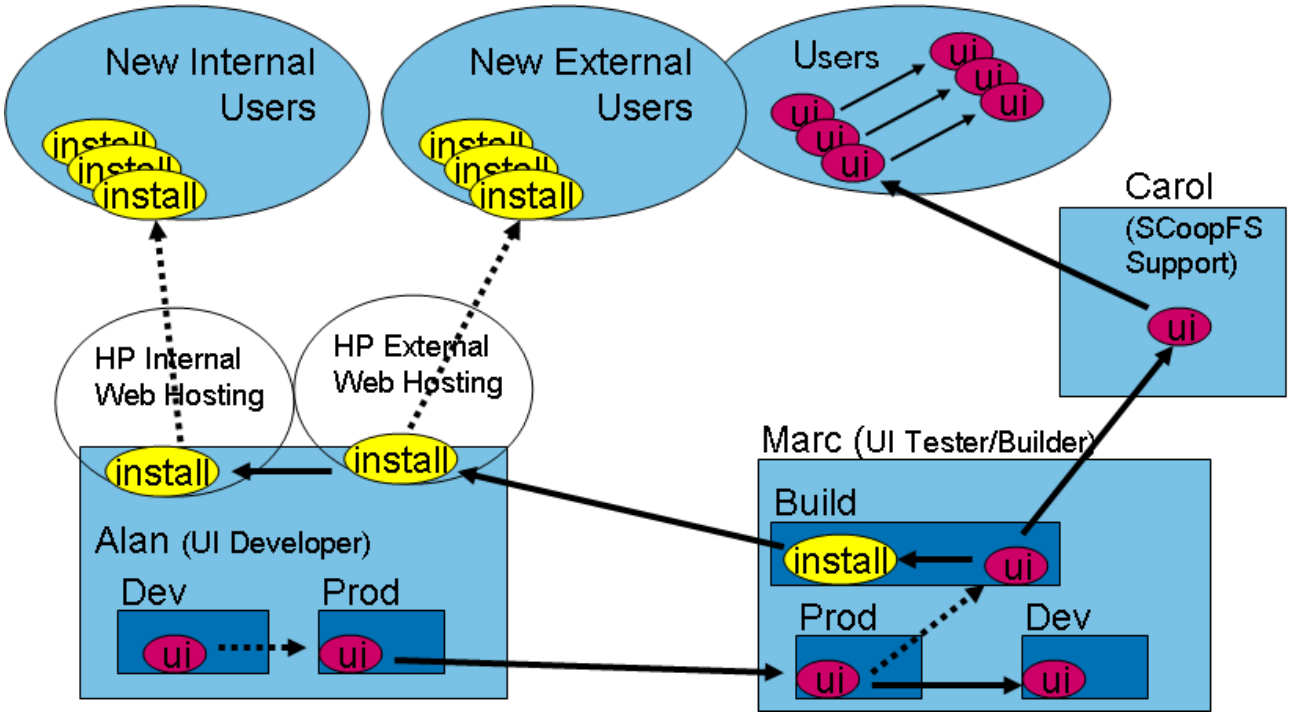

Figure 1. Network of rich sharings in the SCoopFS development/deployment process. Dotted lines are places where human participants must take an explicit action to copy the file, thereby demonstrating that the file has passed their verification/decision requirement.

Marc set up the share sent from Alan to go into his own production environment for more testing. Meanwhile, unbeknownst to Alan, Marc further shared the new ui from his production environment into his own development environment, so that his development system was automatically updated as well.

Separately, when Marc was satisfied with his more extensive testing, he would place the ui into the build environment and build a new installation, which he would email back to Alan (who also maintained the web sites from which new users would download the installation). Eventually they realized that this was too should be handled automatically with file sharing: Marc shared the installation file in the build system with the developer, who placed the share into the external web hosting server from which new external users acquired the software. Unbeknownst to Marc, Alan then further shared it into the internal web server where internal users would download.

Meanwhile, once the new ui was good enough to go into the installation, it was certainly good enough for the Carol, the SCoopFS support person. So the ui file in the build environment was shared to Carol, who then realized that she should share it with all the current users. The current users, in turn, shared it to all the different machines upon which they ran ScoopFS (a common usage of SCoopFS was to synchronize documents across all of a single person's machines, so they could work anywhere anytime). The developers of SCoopFS neither knew nor cared how many firewalls and administrative domains were crossed by the time the sharing of the ui was completed.

A couple of points are worthy of note. First, a series of uncoordinated, spontaneous rich sharings yielded a primitive but effective work flow system. Second, a part of that work flow system was a patch update system – the sequence of sharings from the build environment through the SCoopFS support system to the users and all their machines fulfilled a purpose that normally requires a purpose-built system with extensive provisioning to deliver the updates. The SCoopFS patch update system needed no central server, it used the compute power and the bandwidth of the end user machines, acting as peers, to achieve sharing in a manner akin to Bittorrent.

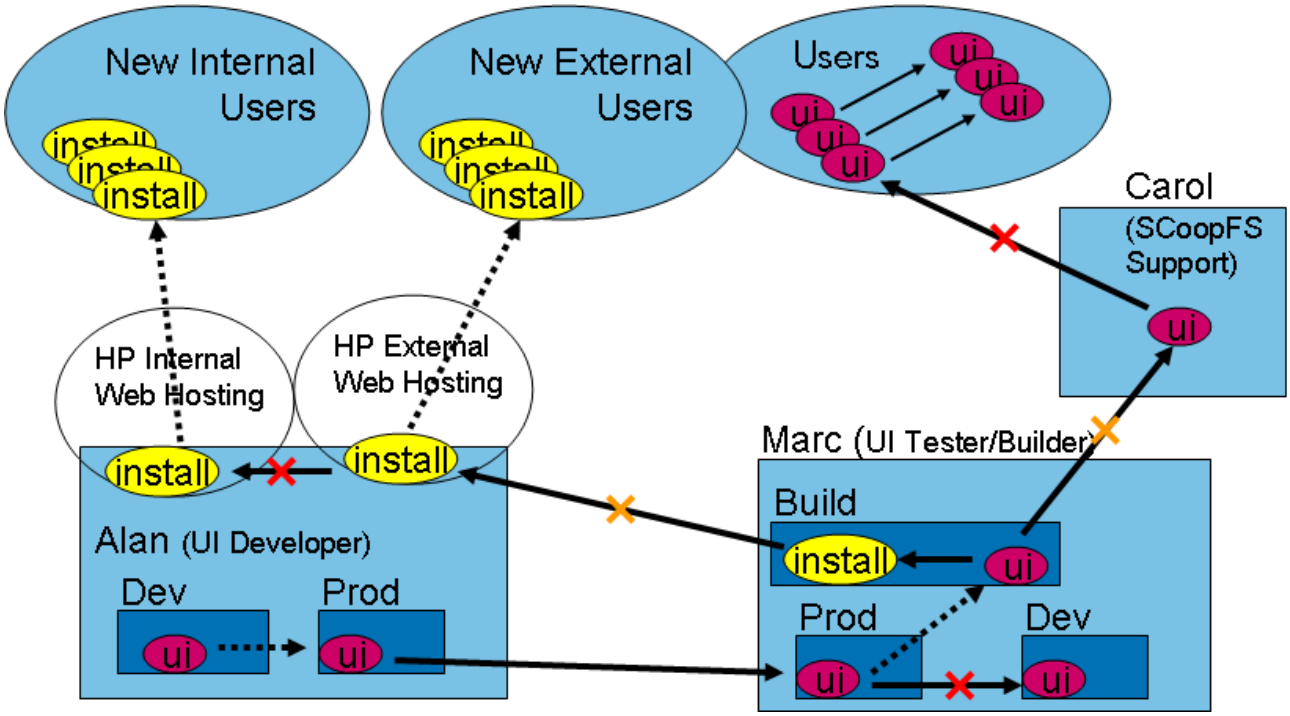

Figure 2. The SCoopFS development and deployment network with those elements not supported by Microsoft Live Mesh removed. The two orange x's represent either a violation of POLA or a usability failure that causes the user to have to work around inadequacies in the software. The three red x's represent failures of functionality, or a violation of POLA so severe it results in a full breach of all the machines in a subnetwork.

One might conclude that any sufficiently powerful file sharing system would achieve this result, but that is not true. Do to a fortunate happenstance, while the SCoopFS testing was proceeding, Microsoft released Live Mesh, a state-of-the-art file sharing/synchronization system with many features in common with SCoopFS. One of the ways in which Live Mesh differed, however, was that it was not explicitly designed to support rich sharing. The contrast is portrayed in Figure 2, wherein the arrows representing flows of authorization and sharing that are not properly supported have been marked with x's. A striking example of the limitations of a non-rich sharing system is that the patch update system cannot be implemented. The reason goes directly back to points made earlier in the paper: the patch update system requires that one be able to chain an attenuated authority. Specifically, one must be able to re-delegate a read-only file sharing, which (as explained earlier) is not possible with a traditional acl system like that found in Live Mesh.

Conclusion

We have described the six key features to support rich sharing. We have explained how rich sharing enables the implementation of the principle of least authority, which maximizes the extent to which we can work with people with whom we have only limited trust, enabling secure cooperation. We have shown how such rich sharing supports construction of more flexible and effective networks of authority that can evolve over time to meet changing requirements, in both the context of email and file sharing. We hypothesize that the benefits of extending the principles of rich sharing to every form of digital resource would produce benefits equally as striking as those shown in the two sample cases.

References

[Karp09] "Not One Click for Security", Alan Karp, Marc Stiegler, Tyler Close, http://www.hpl.hp.com/techreports/2009/HPL-2009-53.html?mtxs=rss-hpl-tr