# Not One Click for Security

Alan H. Karp, Marc Stiegler, Tyler Close

HP Laboratories
HPL-2009-53

**Keyword(s):**
Secure cooperation, usable security

**Abstract:**

Conventional wisdom holds that security must negatively affect usability. We have developed SCoopFS (Simple Cooperative File Sharing) as a demonstration that need not be so. SCoopFS addresses the problem of sharing files, both with others and with ourselves across machines. Although SCoopFS provides server authentication, client authorization, and end-to-end encryption, the user never sees any of that. The user interface and underlying infrastructure are designed so that normal user acts of designation provide all the information needed to make the desired security decisions. While SCoopFS is a useful tool, it may be more important as a demonstration of the usability that comes from designing the infrastructure and user interaction together.

External Posting Date: March 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: March 6, 2009 [Fulltext]

# Not One Click for Security

Alan H. Karp
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
alan.karp@hp.com

Marc Stiegler
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
marc.d.stiegler@hp.com

Tyler Close
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304
tyler.close@hp.com

## ABSTRACT

Conventional wisdom holds that security must negatively affect usability. We have developed SCoopFS (Simple Cooperative File Sharing) as a demonstration that need not be so. SCoopFS addresses the problem of sharing files, both with others and with ourselves across machines. Although SCoopFS provides server authentication, client authorization, and end-to-end encryption, the user never sees any of that. The user interface and underlying infrastructure are designed so that normal user acts of designation provide all the information needed to make the desired security decisions. While SCoopFS is a useful tool, it may be more important as a demonstration of the usability that comes from designing the infrastructure and user interaction together.

## Categories and Subject Descriptors

H.1.2 [**User/Machine Systems**] Human factors; H.5.2 [**User Interfaces**]: User-centered design; H.5.3 [**Group and Organization Interfaces**]: Computer-supported cooperative work; K.4.3 [**Organizational Impacts**]: Computer-supported cooperative work

## General Terms

Security, Human Factors

## Keywords

Secure cooperation, usable security

## 1. INTRODUCTION

Most people agree with the statement, "There is an inevitable tension between usability and security." We don't, so we set out to build a useful tool to prove our point. Since people in our line of work often share work on documents, such as this paper, we decided to build a file sharing tool.

An informal survey revealed that most of our colleagues do not use file sharing products. Those in the enterprise who do primarily choose SharePoint. About half of those we asked use CVS for document sharing when there are no firewall problems. Everyone used email attachments to share versions of the document when firewalls prevented co-authors from accessing the document. Based on this result, we adopted an email metaphor for SCoopFS[1], a system for Simple Cooperative File Sharing.



Symposium On Usable Privacy and Security (SOUPS) 2009, July 15-17, 2009, Mountain View, CA, USA.

Even the people who never use anything but email for sharing work on documents voiced the same complaint. "You've got to remember to send the latest version and apply the updates when they come in." In addition, most of them reported resorting to convoluted conventions to avoid losing work due to edit conflicts. A few even mentioned the lack of security, since almost nobody bothers to encrypt email. Several people wanted to share files between Windows and Linux machines. We designed SCoopFS to address these requirements. This paper discusses the interaction design of SCoopFS. Details of its implementation will be reported separately.

SCoopFS demonstrates two points. First, it shows that managing rights at a fine granularity is easier than dealing with them in large chunks. Second, SCoopFS shows that, at the very least, security need not impede usability and can be largely invisible to the user. The degree to which this view is counter to people's intuitions forced us to change the name of our project. The first "S" in SCoopFS originally stood for "Secure", but several prospective users told us that they didn't need security enough to put up with the pain it caused. Hence, we changed the first word to "Simple."

SCoopFS allows users to manage read and write authorities on single files using only the actions they commonly take when sharing files. At the same time, SCoopFS provides server authentication, so there is no vulnerability to DNS redirection attacks, client authorization, in a way that reduces administrative overhead, and is designed for end-to-end encryption, but the user never sees the security infrastructure. In fact, the goal of the user interface is to include nothing specific to security.

The key to SCoopFS achieving these goals is moving from access decisions based on subject authentication to access decisions based on explicit, delegatable authorizations[7]. Section 2 contains an outline of the access control model we chose, which makes it possible for the infrastructure to infer what rights to grant based solely on actions the user takes to complete the task at hand. Making names meaningful yet free from confusion is important. We describe our apporach in Section 3. Section 4 sketches the user interaction. Several use patterns that emerged in our first, trial deployment are discussed in Section 5. The small user study described in Section 6 provided useful feedback for future work. In Section 7 we introduce the criteria we used to evaluate the degree to which we met our goal of usable security. In Section 8 we compare the usable security aspects of SCoopFS with those of Microsoft Live Mesh[12], the most evolved example of the many deployed file sharing systems. We finish in Section 9 with some conclusions and directions for future work.

[1] The "F" is silent.

## 2. ZBAC WITH WEBKEYS

Although the underlying mechanisms are not the thrust of this paper, this Section describes just enough to allow us to explain the user interaction.

Virtually all systems in use today decide whether or not to honor a request based on the authentication of the requester, an approach we refer to as autheNtication-Based Access Control (NBAC), whether it is identity (IBAC), role (RBAC), or attributes (ABAC) being authenticated. Our approach, which we call authoriZation-Based Access Control (ZBAC) [2], bundles the authorization for an access with the request itself. While subject authentication is one way to decide what rights to grant when using ZBAC, no examination of user authentication credentials is required at the time of access. One advantage of such authorization-carrying access requests is that the user interface can represent these authorizations in the application context so that the user need not be aware of the underlying security mechanisms. Manipulations in the user interface can then be mapped directly to access control decisions. This approach extends the fundamental property of capabilities, combining designation with authorization [5], to the user interaction, inferring authoriztion from designation.

An incomplete but useful analogy is the car key (ZBAC) as opposed to the driver's license (NBAC). Suppose our cars only allowed the holder of a driver's license listed in the car's approved list to drive the car. Valet parking would not work because you wouldn't know which attendant might need to drive your car. By using the car key instead, our automotive systems are able to support a more dynamic, evolving process of authority grant and revocation. The same is possible with our computer systems.

SCoopFS uses ZBAC in the form of webkeys[3], which are designed specifically to operate across the Web. A webkey, such as

https://y-yyx3b54gke3qg2dd.hp.com/#4ca26jq2quiugd

is a standard URL constructed to be used as a capability. Several of its parts are of interest.

**Protocol**: The webkey is an HTTPS URL, which guarantees end-to-end encryption between client and server with TLS/SSL and protects both the transmitted data and the URL itself while in transit.

**Subdomain prefix:** The domain name contains the fingerprint of the public key of the server hosting the object the webkey references. Servers and browsers suitably enabled to understand webkeys can use this fingerprint to authenticate the server, eliminating concerns about DNS redirection and man-in-the-middle attacks without the complexity of certificate authorities. Other browsers can still use these URLs, though with weaker security guarantees.

**Fragment**: The page name contains an unguessably large random string, which is put in the URL fragment so that it won't appear in a REFER header. Knowledge of the URL is proof that a request to the object it designates is authorized.

[2] Our earlier paper [7] uses the acronym ABAC, but that term is used in the US Department of Defense for Attribute-Based Access Control. Hence, we now use the term ZBAC.

A webkey is the moral equivalent of a password, but one the user treats as a bookmark and that controls access to a specific object rather than one the user must remember in order to access a large set of resources. While there is some concern that a webkey will be revealed unintentionally, its unguessability relative to most passwords, the ease of limiting the rights represented by a single webkey, and the safety from keystroke loggers and clickjacking makes the tradeoff worthwhile.

Webkeys have the properties needed to support rich sharing. Since they use HTTPS, references cross firewalls without changing firewall settings. Since they provide request authorization rather than subject authentication, authority passes across domain boundaries without administrative overhead. Webkeys can be used to jump directly to the resource, bypassing username-password logon pages and the host of problems associated with them. Using the *caretaker pattern* [14] allows delegating fine-grain, temporary authority to a resource, in a fashion akin to the process of using a lockbox to hold a car key as a temporary delegation, and then revoking the right by removing the car key from the lockbox.

## 3. NAMING

While webkeys allow us to enforce some useful security properties, they are clearly not suitable for people to use to designate things. People want to use names that are meaningful to them. The naming problem is best understood by looking at Zooko's Triangle [17], which lists three important properties for names.

**Global**: The name should have the same meaning no matter where the object being named resides or where the request is coming from.

**Securely Unique**: A name should point to exactly one thing, and it should not be possible to make that name point to anything else.

**Memorable**: The name should be meaningful to the people who use it.

No name can have all three properties, so SCoopFS uses three different kinds of name. Webkeys are global and securely unique, but not memorable, so SCoopFS users get to assign the *petname* they will see in the user interface for each item they deal with. A petname is a one-to-one mapping between a webkey and a string unique to that user. Petnames are securely unique, because they are local to the user, and memorable, because the user chooses them. However, petnames, because they are purely local, are not global, so they do not provide a means for out-of-band designation, such as over the telephone. For example, each co-author of this paper uses a different petname for his copy of the file containing this document, but we commonly refer to it as the "SOUPS paper" when discussing it. Such *nicknames* are useful, much in the way that "Bill" is a useful designation even though there is clearly more than one person in the world who answers to that name.

## 4. USER INTERACTION

In this section, we'll show all the actions in the SCoopFS user interface. At the end of the Section, we'll list some places where we think security becomes visible. Let the authors know if you find any others.

After installing the Waterken [2] server and SCoopFS software, users run a script that configures the system. This configuration constructs a public/private key pair, and registers a domain name at yurl.net, a domain we run. Software at yurl.net constructs the

domain name to contain the fingerprint of the specified public key. When that is done, the configuration constructs a webkey that points to a mailbox. None of these steps is visible to the user.

The user is instructed to bookmark the mailbox webkey, since there is no way to recover access if the URL is lost. This webkey allows access to the mailbox from any machine that can reach the corresponding server because the SCoopFS user interface runs as a Flash application on a single web page. Arguably, bookmarking a page is a common activity few users would consider security related. At any rate, we are primarily concerned with the user interaction once the setup is complete.

Figure 1 shows the SCoopFS user interface running in Firefox. The section above the gray bar is for navigating among the various views. The section below the gray bar is specific to the view. The buttons in the top section list the various views available to the user. We'll be showing each of them in this section. The only difference in this part of the user interface among the views is to gray out the button corresponding to the current view or to set its label to "Refresh."

The Mail view in Figure 1 should be familiar to users of Outlook and some other mail clients, but there are some differences. The CC and BCC buttons are disabled because they result in the recipients being introduced via SCoopFS, a feature we haven't implemented yet. Next to the To button is a combo box with a pull down menu. The default sharing mode is Read-Write, which means the person sharing the file will accept updates from the recipient. Selecting "Don't Accept Updates" is equivalent to granting only read permission. The other three modes are related to changing sharings and were originally the only way to carry out the indicated tasks. We later added buttons in other views to do the same things, so we'll defer discussing them.

Clicking the Attachment button takes the user to the File selection view shown in Figure 2. Although Flash provides a system specific file dialog box, the interface does not return the full file path. After experimenting with various system tools, we decided to provide our own. The user selects the file to be shared, and clicks OK, which results in the file's pathname being inserted in the attachment field in the Mail view.

Like many instant messaging applications, SCoopFS is a closed system, which means users must establish a relationship before they can communicate. That shows up in the Mail view as a requirement to select the addressee from a list by clicking the To button.

Figure 3 shows the Pals view. Clicking the To button in the Mail view takes the user to a view identical to this one, except the buttons below the gray bar are disabled. The first column in the table is the petname selected by the user for the Pal. It can be changed at any time, and all views will immediately show the new petname. The next column is the name the Pal has proposed, as in "You can call me Al." Users often pick the proposed name as the

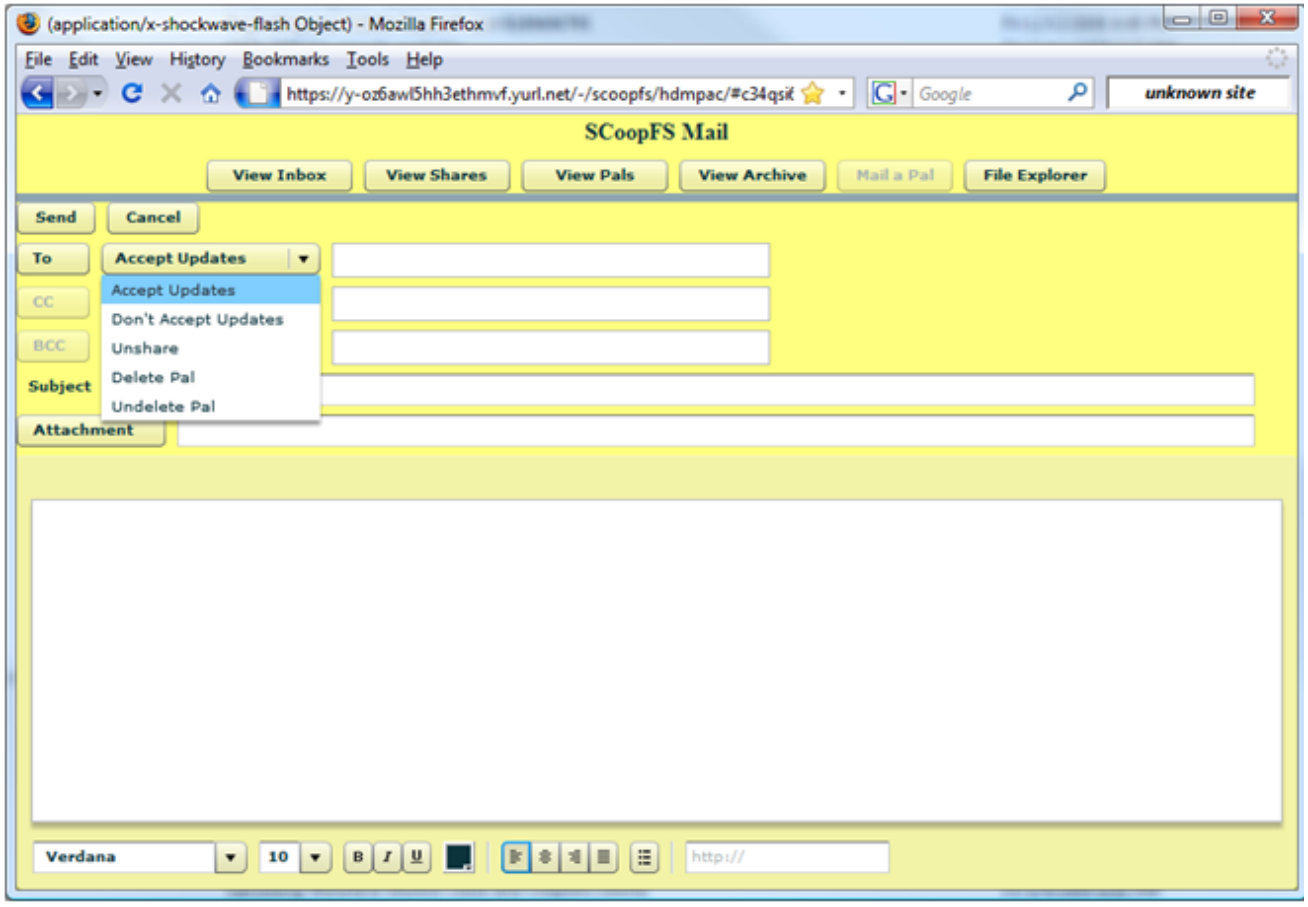


**Figure 1: View for creating a SCoopFS mail message.**

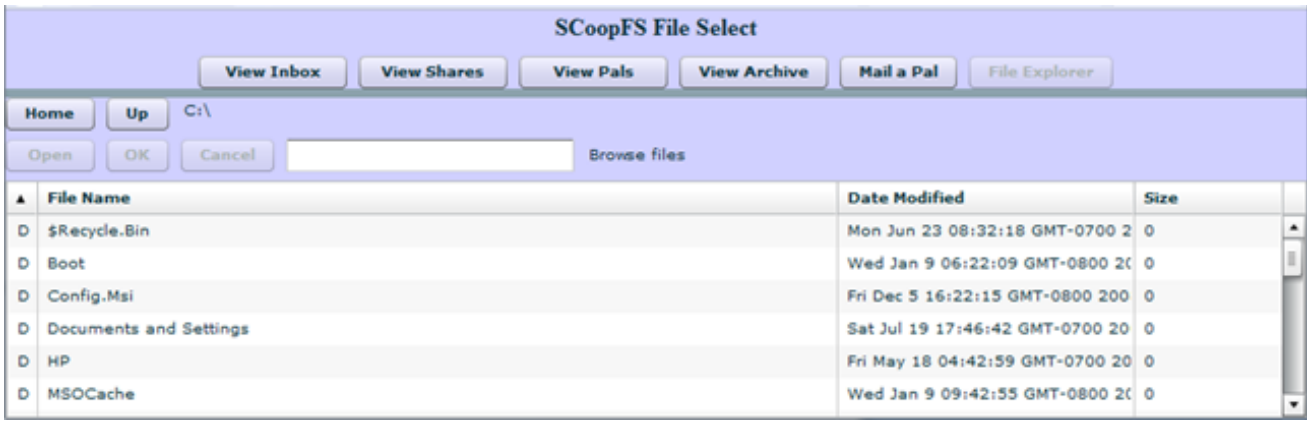


**Figure 2: File Selection view.**

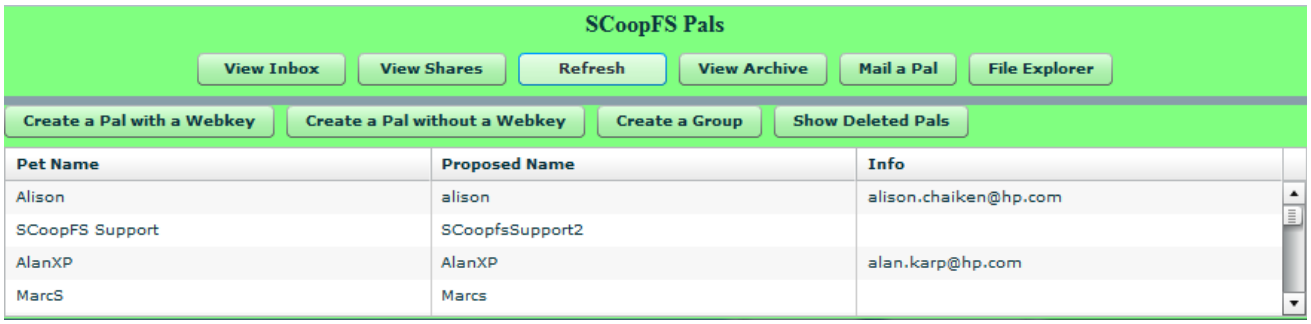


**Figure 3: Pals view.**

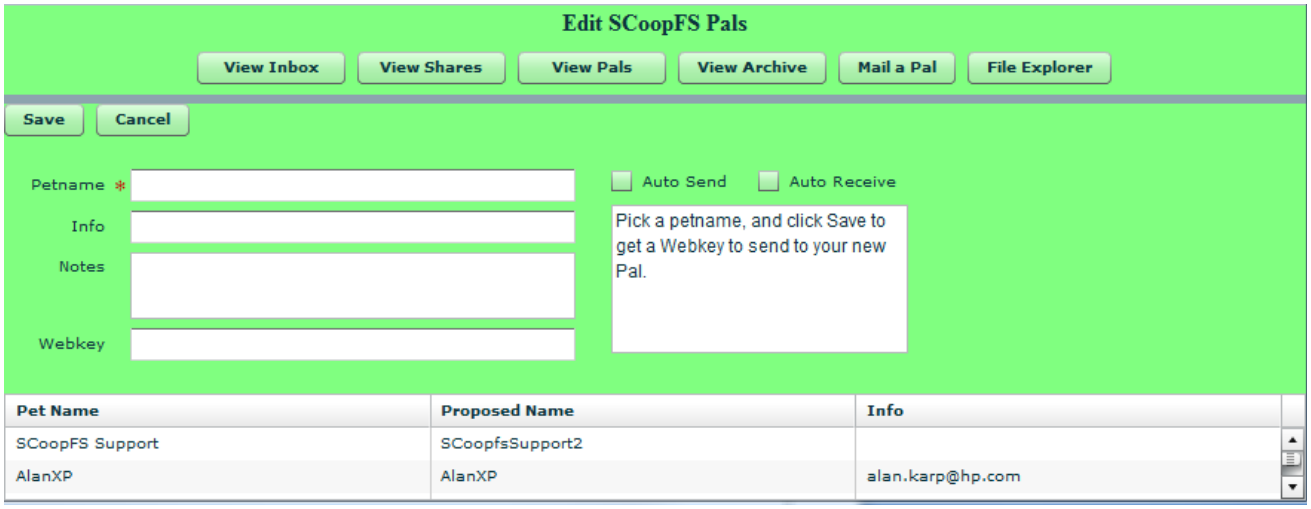


**Figure 4: View for creating a Pal.**

petname, but they need not. Indeed, they can't if they already have assigned that petname to another Pal. The proposed name is useful as a nickname, though. The Info field contains whatever information the user would like to see for this Pal. Here we show email addresses, but phone numbers are commonly listed. Every user has a Pal initially denoted "Me". Receipt of a SCoopFS message triggers a notification to the email address listed for this Pal, which means users will be informed of updates even if the SCoopFS user interface isn't open.

There are two ways to create a Pal, each requiring a webkey. Clicking "Create a Pal without a Webkey" to initiate a new connection takes the user to the view shown in Figure 4. The user selects a petname for the new Pal, fills in the Info and Notes field, and optionally selects Auto Send and Auto Receive. The first of these sends updates of shared files to this Pal as soon as a file shared with the Pal is saved. The second automatically replaces the user's copy with the new version when it is received if there is not an edit conflict.

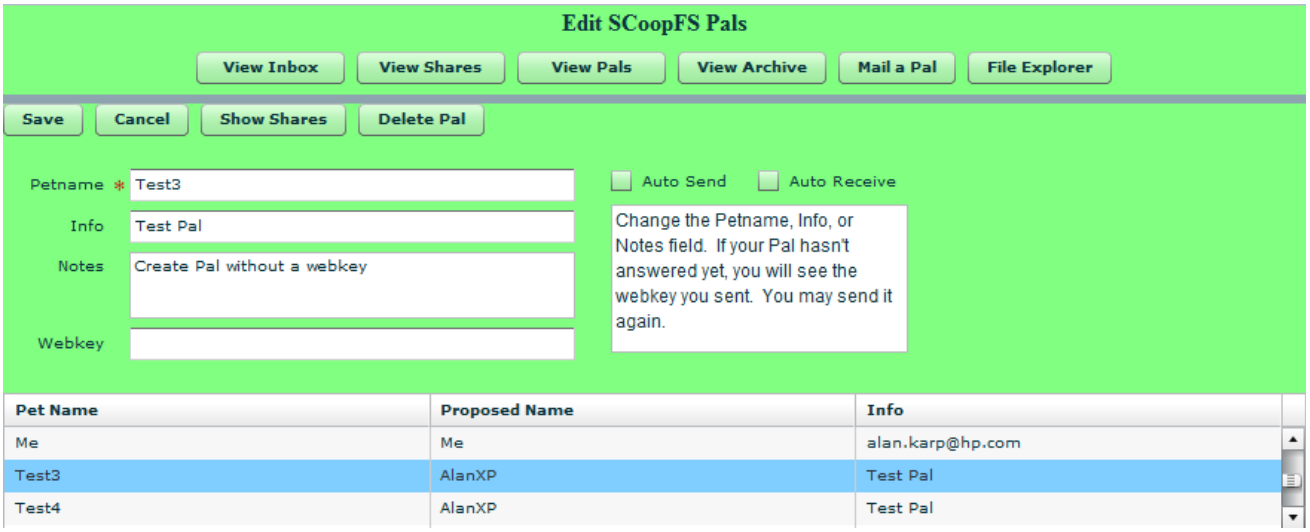


**Figure 5: View for editing a Pal's info.**

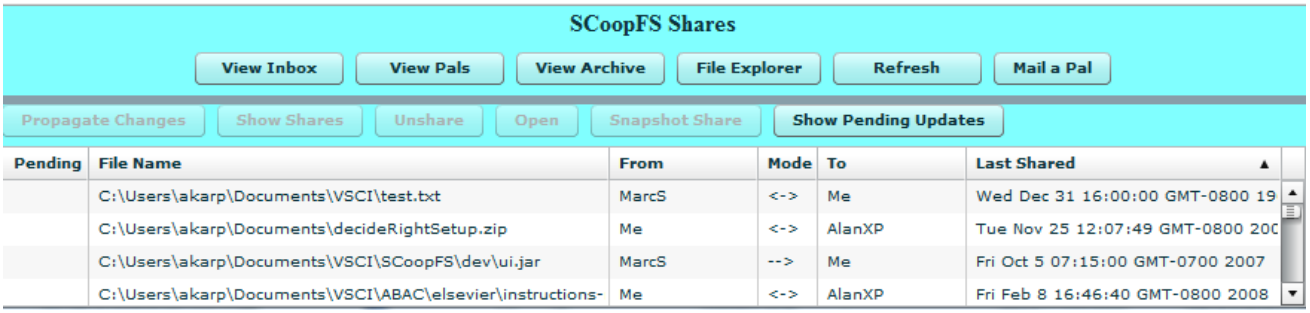


**Figure 6: Shares view.**

A webkey is displayed in the Webkey field after the user clicks Save. The user sends this webkey to the new Pal by whatever means the user desires, typically via a conventional email, much as one would send an email address. Users can prevent a third party who intercepts the webkey from impersonation attacks by confirming the webkey out of band, such as over the telephone, much as they would double check an email address. The recipient clicks "Create a Pal with a Webkey", fills out the form, inserts the webkey, and clicks Save. The difference between these two ways of creating a Pal avoids the confusion some early users had when they generated a webkey instead of using one they had been sent.

Selecting a Pal in the Pals view allows the user to edit the Pal's information, as shown in Figure 5. Changing the Pal's petname automatically updates it in all views. "Delete Pal" takes the user to the Mail view with the combo box's "Delete Pal" option selected. In fact, all sharing changes require that a SCoopFS mail be sent because we thought it would be polite. An interesting observation is how users react to the default mail message, which is "I am deleting you as a Pal because ..." We have seen them complete that sentence even when just experimenting with the system.

Clicking Show Shares takes the user to the Shares view shown in Figure 6 filtered to show only files shared with the selected user. The Pending column indicates if a shared file has been changed but not yet sent to this Pal, as might happen if Auto Send wasn't selected for this Pal, or if changes from the Pal have been received but not yet applied to the user's copy, as might happen if Auto Receive was not selected or if there is an edit conflict. The "Propagate Changes" button is enabled when the user selects a shared file with an unsent change. The From and To fields show the sender and recipient of the message initiating the sharing, and the Mode shows the direction that changes propagate. The string <-> denotes read/write sharing, while --> denotes read only. The last column shows the time the last update was applied.

Selecting a shared file enables the rest of the buttons. These allow the user to see all shares of the selected file, revoke the sharing of this file with the displayed Pal, open the file, or take a snapshot. The last of these is similar to what people do manually with email attachments, assigning different filenames to significantly different versions.

Figure 7 shows the Inbox view with a new share selected. There are several items of note in this figure. The string "Alan" in the top line is the nickname that the owner of this mailbox has chosen. It shows up in the Proposed Name column in the Pals view of Alan's Pals. Two SCoopFS users can discuss a third party with a nickname "Alan", but they can't be sure it's the same person. Nevertheless, a nickname is a useful reference, especially within a small group.

The table below the gray bar in the Inbox view shows some familiar headings, such as "From" and "Subject". There are several


**Figure 7: Inbox view for a new share.**

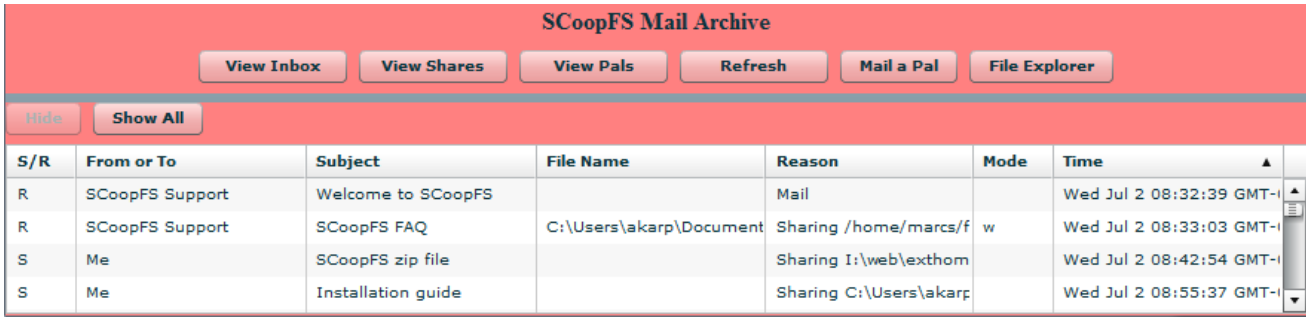


**Figure 8: Archive view.**

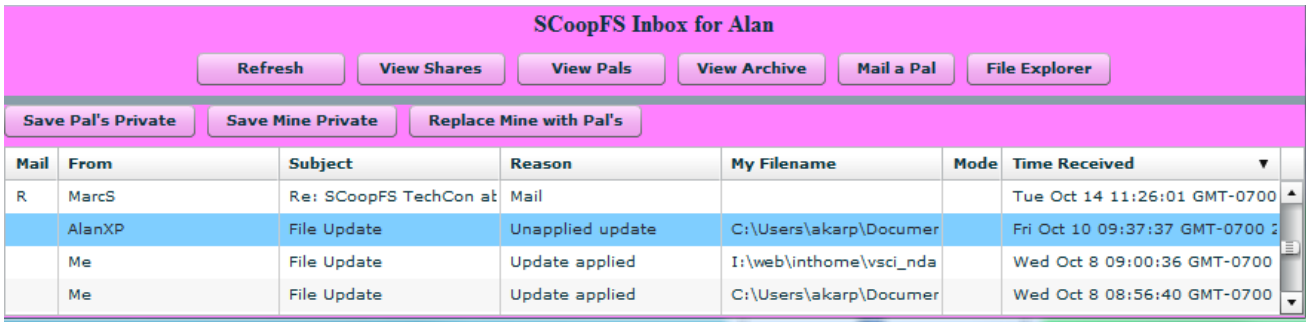


**Figure 9: Inbox view for applying an update.**

kinds of messages in SCoopFS. Users can send what appear to be ordinary email messages. Until these are read, the "Mail" column denotes them as "U", with a tooltip "Unread." After, they are marked "R" with a tooltip "Read." Other messages are related to file sharing and are discussed below.

The "From" column lists the user's petname for the sender. It is impossible to forge the sender's identity because each petname is bound to a specific webkey. The sender of a SCoopFS mail selects what appears in the "Subject" column, but the value is "File Update" for system generated messages. The "Reason" column can contain the values "Mail", for messages without an attachment, "Sharing", for messages with an attachment, "Update Applied", for file updates that were accepted either manually or automatically, "Unapplied Update", for file updates that have not been applied because Auto Receive was not selected, and "Edit Conflict", where SCoopFS has detected overlapping updates. Users may remove any message from the Inbox, except one from a new share until they've saved the attachment, but the message is still available in the Archive view as shown in Figure 8. Messages can be hidden in the Archive view but not removed. SCoopFS never forgets.

Selecting a "Sharing" message enables the "Save Attachment" button, which when clicked takes the user to the File Select view. Selecting a "File Update" message enables the "Remove from List" and "Open" buttons. Selecting an "Unapplied Update" message, as shown in Figure 9, lets the user accept the update or keep a private copy of either version. The "Replace Mine with Pal's" button is disabled for an "Edit Conflict" message, which makes it

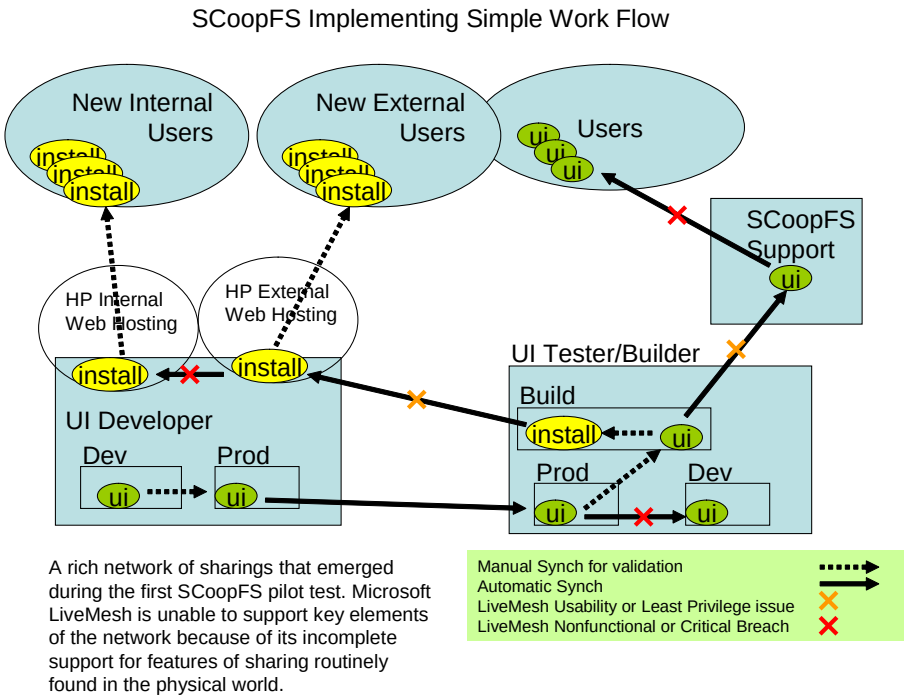


**Figure 10: Using SCoopFS sharing to implement distribution of software updates.**

harder to inadvertently lose work by replacing a file containing one set of changes with another version having a different set.

SCoopFS has no "Help" button. However, most buttons and input fields have explanatory tooltips, and the views for creating Pals have some instructions on them. We encourage our users to play around, as they would with a new video game. Any action can be undone.[3]

So, did you see any security? Any server authentication? Any user authorization? Any encryption? Arguably, the answer is yes in two places. The first is the option to choose whether or not to accept updates. While this choice can be considered security related, it is also common to distinguish between editors and reviewers when sharing files. It is possible to argue that this choice is just shorthand for a decision that the sender would have to include manually in the body of the message. None of the users in our study raised this point as a security issue, but we did not point out the option to them. The second is in handling the webkey needed to set up a Pal. To the user it is just a URL with no more relation to security than a rental car confirmation with a URL containing a confirmation number or a connection invitation from LinkedIn. The users in our study emphatically disagree with this interpretation, stating that the webkey is a confusing security mechanism. A future version will have to encapsulate the webkey so the users don't see it.

[3] We haven't implemented moving an item from the Archive back to the Inbox.

## 5. EMERGENT USES

Our first trial with users outside our team taught us that the most common use for SCoopFS wasn't what we thought it would be. It turns out that most people were more interested in sharing files between their own machines, a desktop at work and a laptop, for example. That observation led us to provide a tool to initiate sharing from the Windows file explorer. Also, since these users rarely sent or received SCoopFS mail messages, we added the ability to request a notification be sent to their usual mail clients when an update arrives.

We found some interesting use patterns. We have a machine running a SCoopFS account as "SCoopFS Support" that shares a read/write copy of the file containing the SCoopFS FAQ with all users. We encourage them to share this file with Pals on other machines that they configure. The result is that all users see changes made by any user, making that file effectively a wiki. Any user can ask a question, which can be answered by any other user. Had SCoopFS Support not accepted updates to the shared file, it would serve as a blog.

Our distribution is a ZIP file maintained by one of us, Marc, and made available on the web site of another of us, Alan. Marc shared this file read/only with Alan, who put his shared copy on the remote drive hosting his internal web site and shared that version with his external web site. This configuration allows Marc to control exactly one file on those two web sites, something that would be difficult to do with conventional mechanisms since Alan doesn't have permission to create an account for Marc on that machine.

We discovered that we could implement the workflow involved in distributing software updates by properly configuring the sharing

as shown in Figure 10. Alan maintains a development copy of the user interface and a test version. The latter is shared read/only with Marc. When Alan copies the development version to the test version, Marc gets the update. When Marc completes his testing, he copies the file to his production version. When he creates a new ZIP file for the distribution, the change propagates to Alan's web sites. Meanwhile, the file containing the updated user interface propagates to SCoopFS Support. The changes then propagate to all SCoopFS users because SCoopFS Support shared the file with them. If those users also shared the file with Pals on machines they configured, every SCoopFS user gets the new version without the need to verify certificates [13].

# 6. USER STUDY

We sent a call for volunteers to HP Labs and selected 10 of our technical colleagues, excluding anyone who had prior knowledge of SCoopFS or who had a computer security background. The invitation included the short document in the Appendix, which is intended to give prospective users a basic understanding of what SCoopFS is intended to do.

Since the installation procedure was not part of the study, we installed the software for the users and set up SCoopFS Support as the first Pal. The configuration resulted in the users receiving a welcoming SCoopFS mail sharing a file containing the instructions shown in the Appendix and a message sharing the USE [8] survey. We augmented that form with the following questions:

1. Which tasks were too hard?
2. Did the security get in your way? If so, how?
3. Did you look for the Help button?
4. Do you want us to remove SCoopFS from your machine, or would you like to try it for a while?

We set the initial view to show the Inbox, told the user to read the welcoming message and follow the instructions in it. Those instructions were

1. Familiarize yourself with the user interface.
2. Save the attachment to a file.
3. Wait for the file to be updated
4. When the update arrives, open the file and follow the instructions in it.

Those instructions were:

5. Use the webkey in the file to create a new Pal.
6. Share your shared file with the new Pal.
7. Unshare the file.
8. Save the attachment in the second message.
9. Fill out the survey contained in that file and save it.
10. Collect your scoop (of ice cream).

Items 2-9 are the most common SCoopFS activities. Not wanting to impose too much on our volunteers, we stopped there.

We didn't set any time limit, but one of us did watch the test and noted when an activity took longer than we expected. Although we had planned to help when the user got stuck on a task, no intervention was necessary.

The questions in the USE survey [8] fall into two classes, usefulness and usability. Each question is graded on a scale of 1 (bad) to 7 (good). As noted by one of the testers, some of the questions make sense only after more use than the 15 to 20 minutes users spent on this study. We excluded questions users marked N/A from the average scores shown in Table 1. Not surprisingly, the testers who found SCoopFS useful also found it useable and would recommend it to their friends.

Table 1. Summary of user study results.

| Tester | Usefulness | Usability | Recommend |
|---|---|---|---|
| 1 | 3.7 | 3.9 | 3 |
| 2 | 4.9 | 5.4 | 5 |
| 3 | 2.7 | 2.9 | 1 |
| 4 | Additional participants to be added after the paper deadline. | | |
| 5 | | | |

Most users commented that the usefulness of the tool depends on how many colleagues choose to use it, the network effect. Every user felt that having to see the webkey to set up a new Pal was the weakest part of the user interaction. However, that was the only place users felt that the security showed through. Although several of them looked for a Help button, none felt its absence was critical because, as one tester noted, the tooltips answered his questions. Each tester found at least one feature puzzling, indicating the need for a user's guide.

In retrospect, we learned some of the dangers of user testing an early prototype [6]. One test failed because the updated version of Flash on the user's machine has more stringent security settings than the version we used for development. Some users were bothered by weaknesses in the user interface caused by the lack of features we didn't know we wanted when building the infrastructure. One test failed due to a race condition that we haven't been able to fix.

# 7. SCORECARD

Entire libraries worth of books have been written on security evaluations. Usability evaluations are not far behind. Far less has been written on usable security, but there is still a substantial literature. For this paper we score SCoopFS on the 10 design guidelines for secure interaction design [19].

1. Make the most convenient way to do something the way that grants the least authority. Score: 0.5

   We made a conscious decision to violate this guideline by making the default sharing mode Read/Write rather than Read/Only to better support the most common use. Even the user interface developer kept forgetting to change the mode from Read/Only when sharing files with himself. We still get half credit because by default users share access to single files rather than the large chunks of rights that often come from changes to the ACL.

2. Use user actions indicating consent when granting authorities. Score: 1.0

   Setting up a new Pal explicitly grants that Pal the right to send messages. Sharing a file gives the Pal the authority to read changes and/or send changes to the sharer. Both are explicit actions taken in the user interface.

3. Make it possible for the user to reduce the authority others have to the user's resources. Score: 1.0

Users share only the files they want to share and can decide if they want to accept updates from the Pal. Turning off Auto Receive allows the user to decide which updates to accept. Turning off Auto Send lets the user decide which versions of the file the Pal will see. Users may also unshare a file or delete a Pal, revoking the associated rights.

4. Make the user aware of other's authorities to the user's resources when making decisions. Score: 1.0

   The Shares view allows the user to see all shares, shares with a specific Pal, or all shares of a specific file. The Pals view allows the user to identify who may send messages and whether sends and/or receives are handled automatically.

5. Make the user aware of the user's own authorities. Score: 1.0

   The Pals view lets the user see which Pals will accept messages, and the Shares view shows the user which shares will accept updates. We can't do anything about the user's authorities to files not shared through SCoopFS, but we didn't think that was cause to reduce our score.

6. Protect the means by which the user indicates to software how to manipulate authorities. Score: 0.8

   The SCoopFS trusted path, based on secure labeling with petnames, is not truly trustworthy because of the vulnerabilities of the browsers through which it operates. However, webkeys prevent Cross Site Request Forgeries and clickjacking. Since they are never typed in, webkeys are also safe from keystroke loggers.

7. Express security policy in terms relevant to the user's task. Score: 1.0

   All security policy is inferred from actions users take to get their jobs done.

8. Make apparent the distinctions among objects and actions in a way that is relevant to the user's task. Score: 0.5

   The Shares view shows which files are shared read/only and the direction changes propagate. Editing a Pal shows whether updates are sent and/or received automatically, but this information is not shown when sending a message to the Pal. Sharing and propagation information is not shown when the user edits a shared file. These shortcomings can be fixed in a later version.

9. Clearly and truthfully distinguish different objects and actions. Score: 1.0

   SCoopFS uses color cues to help the user avoid being confused by similar views, such as the Inbox and Archive. Petnames chosen by the user are less likely to lead to confusion than other methods of choosing names.

10. Make the implications of the user's actions clear. Score: 0.8

    Creating a Pal adds an entry to the Pal's list, but we do not highlight the new Pal. Sharing a file with a Pal creates a new entry in the Shares view, but we do not highlight new shares. These shortcomings can be fixed in a later version.

These are common sense guidelines, yet it is hard to think of a single system in common use that implements even one of them. The previous highest score went to CapDesk [18], which gets a score of 7. Although we are happy with our score of 8.6 out of 10, there are improvements we can make in a future version.

# 8. RELATED WORK

CapDesk [18] introduced the concept of inferring which rights to grant from user acts of designation, an idea we used in developing Polaris, a virus safe computing environment for Windows XP [16]. Polaris encountered ambiguities that resulted from the fact that the Windows user interface was developed independently of our infrastructure. One way to get around such problems is to provide a means to express a policy that is independent of the user interface [1]. SCoopFS takes a different approach, avoiding that problem by considering user interactions throughout the development process. This approach assumes full control over the user experience, so it is not applicable to legacy environments.

When we started this project some two years before writing this paper, searching Google for "file sharing tools" returned mostly links to tools for sharing files that are unlikely to be modified, usually downloads of music and videos. The same search done when this paper was being written, still returns a lot of hits to such tools, but an increased number of them are for sharing of files that are likely to be modified. Approximately 50 of the 100 top ranked links are for unique file sharing tools. Browsing Google to Top > Computers > Software > Internet > Clients > File_Sharing turns up 30, about 10 of which are to tools such as Napster, and the rest for sharing files likely to be modified. Space and time do not permit reviewing them all.

File sharing tools fall into three main categories. There are tools where the shared space is controlled by an administrator, such as Groove [10] and SharePoint [11]. These tools are based on user accounts and require that files be moved to their workspaces in order to be shared, although the most recent version of SharePoint provides some automation. There are tools that store your files on the provider's servers. Some of them provide for off-line access, but they all require substantial trust in the provider. Neither of these classes of tools is directly comparable to SCoopFS, so we won't discuss them further. Their most important weakness is the inability to do chained sharing. ContentCircles [4] is a hybrid, sharing files in a workspace that is copied on all participants' machines. It does allow chained sharing, but it centralizes the means by which participants connect.

Closer to SCoopFS in spirit are peer-to-peer tools. Microsoft Windows Vista natively supports sharing of files between users on the same LAN [9]. However, unlike SCoopFS, the shared files are not copied, so there is no off-line use, the transport is not encrypted, so the contents can be snooped on the LAN, and the user interface for managing shares is accessed from the Control Panel, which is inconvenient. Partly to fix these problems, Microsoft developed Live Mesh [12]. Microsoft would claim that Live Mesh meets the requirements of SCoopFS users. In the rest of this Section we'll show the ways that it does not.

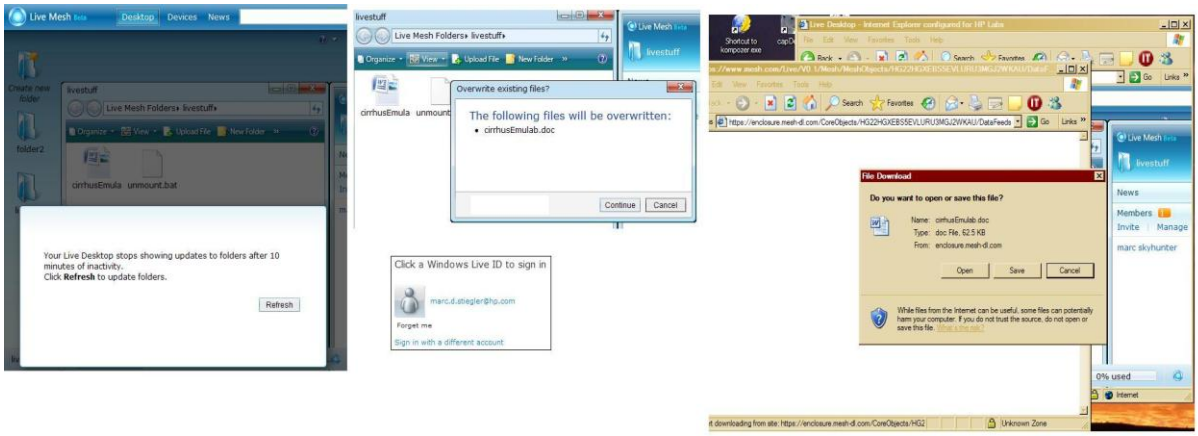


**Figure 11: Screen shot of Live Mesh in use.**

SCoopFS starts with a user interface concept (rich *seamless* sharing) and constructs a software infrastructure specifically designed to support the user experience. With the Live Mesh system, users must "log on" to their Mesh with a user name and password – a clear violation of seamlessness. Synchronized files must be organized the same away on all machines – even if the organization on one machine for one purpose does not make sense on another with another purpose. Explicit directions must be given to the system when the user decides to work offline – another violation of seamlessness.

Even obvious forms of attenuation, such as single-file access, are not supported. Chained attenuation, for example, the ability of the holder of a read-only authority to further share a separately revocable read-only authority to another person, is not supported.

Figure 11 shows a screen shot with typical Live Mesh dialogs that present obstacles to the user. Many of the required user actions are independent of the task at hand, in contrast to SCoopFS. The tools and techniques used for SCoopFS, like petnames and webkeys, supply infrastructure support that enables the building of superior user interfaces.

Figure 10, which we discussed in Section 5 is the clearest way to show where Live Mesh falls short of SCoopFS. The software update workflow requires attenuated, chained delegation of the authority to use the file. The orange Xs in Figure 10 show places where a violation of Least Privilege or a violation of experience seamlessness is the only way to get the desired sharing from Live Mesh. The red Xs show where either Live Mesh explicitly forbids the sharing, or the only possible implementation results in a critical vulnerability.

# 9. CONCLUSIONS

We started the SCoopFS project with three goals in mind. The first was to produce a useful tool. We find that SCoopFS simplifies our work, but we're biased. The early versions of SCoopFS were missing some features users thought important, such as an integrated file explorer and a means to share a file directly from the Windows desktop. We did our user study after adding these features. Based on the lukewarm reaction, we will not be quitting our day jobs to form a startup based on SCoopFS.

The number one complaint about SCoopFS is, "Oh, no. Not another mail client." We are sympathetic, but we felt that having full control over the user experience was important for the first prototype. We plan to examine building a SCoopFS plug-in for Microsoft Outlook and one of the mail clients for Linux. We don't plan to build one for any of the web-based mail clients because the security model is so different from that of SCoopFS.

The most common requests for additional features are for sharing folders and integration with a version control system. We added the ability to snapshot a file in the File Select view as a primitive form of version control that is consistent with the way people deal with email attachments. We plan to add support for folders, but there are interesting questions of the semantics of operations that we'll need to answer. For example, what should be done if a file is deleted in a read only sharing of a folder?

The second goal was to demonstrate that security need not interfere with the user experience. The user study supports the claim that we succeeded. Even more telling is the quote from one of our early users. "This tool would be a lot better if it had some security. Is there any way I can turn some on?" While that question shows we achieved our goal, it also indicates that achieving our goal is not enough. While "security reality" is necessary, the "feeling of security" is important, too [15]. We need to find a way to make people feel secure without making security interfere with their work

The success in meeting our third goal is hard to quantify. We believe that we've shown that it is easy for users to manage rights to individual files. We also believe that existing systems for managing rights in big chunks, such as NFS and shared drives in Windows, require a lot of work on the part of the user or an administrator to avoid serious violations of Least Privilege. Perhaps we'll know after we gain some experience with sharing folders with SCoopFS. Quantifying the difference will be a challenge.

# Acknowledgements

We'd like to thank Jhilmil Jain for help with the user study and for her patience as an early user of SCoopFS. We'd also like to thank the seven people who gave up part of their day to participate in our user study.

# 10. References

[1] Cannon, Brett, and Eric Wohlstadter. "Enforcing Security for Desktop Clients using Authority Aspects." *AOSD '09: Proceedings of the 8th international conference on Aspect-oriented software development.* Charlottesville, VA : ACM,

2009.

[2] Close, Tyler. *Waterken Server.* 2008. http://waterken.sourceforge.net/ (accessed February 9, 2009).

[3] —. "web-key: Mashing with Permission." *IEEE W2SP 2008: Web 2.0 Security and Privacy.* Oakland: IEEE, 2008.

[4] ContentCircles. *Collaborative Content Management for Distributed Teams.* 2009. http://www.contentcircles.com/ (accessed February 9, 2009).

[5] Dennis, Jack B., and Earl C. Van Horn. "Programming Semantics for Miltiprogrammed Computations." *Comm. ACM* (ACM) 9, no. 3 (March 1966): 143-155.

[6] Greenberg, Saul, and Bill Buxton. "Usability evaluation considered harmful (some of the time)." *Proc.26th SIGCHI conf. on Human factors in computing systems.* Florence, Italy: ACM, 2008.

[7] Karp, Alan H. "Authorization Based Access Control for the Services Oriented Architecture." *Proc. 4th Int. Conf. on Creating, Connecting and Collaborating through Computing (C5 2006).* Berkeley, CA: IEEE Press, 2006.

[8] Lund, Arnold. *USE Questionnaire Resource Page.* November 11, 1998. http://usesurvey.com/ExampleQuestionnaire.html (accessed February 9, 2009).

[9] Microsoft Corp. *File and Printer Sharing in Windows Vista.* May 14, 2007. http://technet.microsoft.com/en-us/library/bb727037.aspx (accessed February 9, 2009).

[10] —. *Groove Home Page.* 2009. http://office.microsoft.com/en-us/groove/FX100487641033.aspx (accessed February 9, 2009).

[11] —. *Microsoft Office SharePoint Server 2007.* 2009. http://www.microsoft.com/Sharepoint/default.mspx (accessed February 9, 2009).

[12] —. *What's inside Live Mesh?* 2008. https://www.mesh.com/Welcome/features/features.aspx (accessed December 17, 2008).

[13] —. *Windows Update.* 2009. http://update.microsoft.com (accessed February 9, 2009).

[14] Redell, D. D. *Naming and Protection in Extendible Operating Systems.* Ph. D. Thesis, Project MAC TR-140, MIT, 1974.

[15] Schneier, Bruce. "Reconceptualizing Security." *LISA '08: 22nd Large Installation System Administration Conf.* San Diego: Usenix, 2008.

[16] Stiegler, Marc D, Alan H Karp, Ka-Ping Yee, and Tyler Close. "Polaris: Virus Safe Computing for Windows XP." *Comm. ACM*, September 2006: 83-88.

[17] Stiegler, Marc D. *An Introduction to Petname Systems.* http://www.skyhunter.com/marcs/petnames/IntroPetNames.html (accessed December 16, 2008).

[18] —. *E and CapDesk: POLA for the Distributed Desktop.* 2002. http://wiki.erights.org/wiki/CapDesk (accessed December 16, 2008).

[19] Yee, Ka-Ping. "Guidelines and Strategies for Secure Interaction Design." In *Security and Usability: Designing Secure Systems That People Can Use*, by Lorrie Faith Cranor and Simson Garfinkel, 247-273. Sebastopol, CA: O'Reilly Media, Inc., 2005.

# Appendix: Documents Given to Participants

## Call for Volunteers for User Study

SCoopFS (The "F" is silent.) is a system that allows you to share files between any two machines anywhere on the Web. While designed to allow people to collaboratively edit documents, many other uses for the system have been identified. The most popular use of SCoopFS is to share files between your own devices, such as between your laptop and your desktop computers, so that you can pick up where you left off no matter which machine you happen to be using.

Although there are a wide variety of file sharing tools, in practice, most people share files by sending a copy as an attachment to an email with a note say, "Please edit this and send it back when you are done." SCoopFS builds on this already successful model of sharing by using an email-like interface to send file attachments to the people or machines in your SCoopFS address book. Once a file has been shared, SCoopFS automates the propagation of updates and warns you if there is an edit conflict.

If you agree to test SCoopFS, we will install our software on your machine and ask you to carry out 10 tasks. One of these tasks is to fill out a survey form and answer a few questions. The entire process should take about 30 minutes. There will be a small reward for participants.

## User Instructions for File Sharing with SCoopFS Experimental Study

In your FireFox bookmarks please find a bookmark called "SCoopFS Inbox." Click the link and familiarize yourself with the SCoopFS user interface. When you are comfortable, read the Greeting message already in your inbox and follow the instructions found there.

## User Instructions in the Greeting Message

1. Familiarize yourself with the user interface.
2. Save the attachment to a file.
3. Wait for the file to be updated
4. When the update arrives, open the file and follow the instructions in it.

## User Instructions in the File Shared in the Greeting Message

1. Use the webkey in the file to create a new Pal.
2. Share your shared file with the new Pal.
3. Unshare the file.
4. Save the attachment in the second message.
5. Fill out the survey contained in that file and save it.
6. Collect your scoop (of ice cream).