xShare: Supporting Impromptu Sharing of Mobile Phones

Yunxin Liu[1,3], Ahmad Rahmati[2], Yuanhe Huang[1,4], Hyukjae Jang[1,5], Lin Zhong[2], Yongguang Zhang[1], Shensheng Zhang[3]

[1]Microsoft Research Asia, Beijing, China [2]Rice University, Houston, TX, USA
[3]Shanghai Jiao Tong University, Shanghai, China [4]Tsinghua University, Beijing, China
[5]KAIST, Daejeon, Republic of Korea

ABSTRACT

Loaded with personal data, e.g. photos, contacts, and call history, mobile phones are truly personal devices. Yet it is often necessary or desirable to share our phones with others. This is especially true as mobile phones are integrating features conventionally provided by other dedicated devices, from MP3 players to games consoles. Unfortunately, when we lend our phones to others, we give away complete access because existing phones assume a single user and provide little protection for private data and applications. In this work, we present xShare, a protection solution to address this problem. xShare allows phone owners to rapidly specify what they want to share and place the phone into a restricted mode where only the data and applications intended for sharing can be accessed.

We first present findings from two *motivational user studies* based on which we provide the design requirements of xShare. We then present the *design* of xShare based on file-level access control. We describe the *implementation* of xShare on Windows Mobile and report a comprehensive *usability evaluation* of the implementation, including measurements and user studies. The evaluation demonstrates that our xShare implementation has negligible overhead for interactive phone usage, is extremely favored by mobile users, and provides robust protection against attacks by experienced Windows Mobile users and developers.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management
D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Management, Performance

Keywords

Sharing, Mobile phone, Privacy, Virtualization

1. INTRODUCTION

While mobile phones have been very personal devices for their users, two recent trends have made phone sharing increasingly attractive, as we have found from two user studies. First, as mobile phones become feature-rich and equipped with large amounts of storage for user-generated content, it has become socially attractive to share features or user generated content (e.g. music and pictures) with others. Second, as the price of feature-rich, Internet-capable mobile phones drops, they have become an attractive solution for providing under-served communities with access to information and communication technologies. In such communities, phone sharing is fundamentally necessary because it will still take a long time for such phones to become affordable to the majority of their community members.

However, existing mobile phones provide inadequate support for such sharing; there is no access control on private data and pay-per-use applications. Consequently, when the owner shares their phone, the borrower will have the same access rights as the owner. Some mobile phones use a password to prevent unauthorized access; yet it is for the entire system and therefore the access control is *nothing or everything*. The iPhone has a *restriction* feature that can disable some built-in applications. Yet it does not apply to third-party applications nor does it provide access control for data. Windows Mobile phones can boot into a less-known *Kiosk* mode, in which only certain applications can be run. However, it requires a reboot and does not provide access control to data.

To address such limitations, we present xShare, a software solution for friendly, efficient, and secure phone sharing. xShare is based on existing system support and allows the owner to rapidly specify what they want to share and place the phone into a restricted mode where only specifically shared applications and data are accessible. In this work, we present the results from a complete research and development cycle of xShare, including motivational user studies, design, implementation, and evaluation. Therefore, our contributions are fourfold.

Motivational User Studies (Section 3). We present a thorough understanding of phone sharing obtained from two user studies, including interviews with existing smartphone users from four countries and long-term user studies with

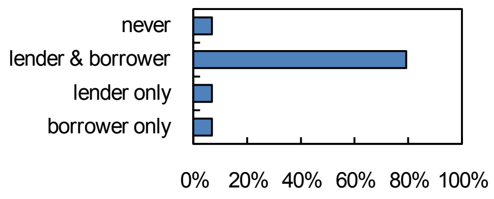


Figure 1: Phone sharing statistics

teenagers from the USA. Our user studies show that the majority of existing and potential users share their mobile phones. Our user studies provide insight into why, where, with whom, and for what applications mobile users share their phones. Our long-term study further shows that phone sharing serves as an important social networking tool for the participants. Our findings highlight the inadequate privacy protection for sharing in current phones.

Design (Section 4). Based on these findings, we propose the design of xShare based on file-level access control. xShare provides two modes of operation, Normal Mode and Shared Mode. When switching from Normal Mode to Shared Mode, the owner specifies which files and applications to share, or a *sharing policy*. xShare creates a virtual environment for Shared Mode from the sharing policy. The virtual environment contains only specifically shared files and applications and conceals the rest of the system. When switching from Shared Mode to Normal Mode, user authentication is required. The borrower uses the phone in Shared Mode.

Implementation (Section 5). We have implemented xShare on Windows Mobile. Our implementation works atop of the existing systems without requiring changes to the OS source code or the phone ROM image. As Windows Mobile lacks built-in file-level access control, we implement one based on system API interception at the kernel level. We implement namespace virtualization for resource access and create a virtual environment to contain shared data and applications in Shared Mode. Further we have addressed several practical challenges, such as the tight integration of system services and in-memory data. Through careful examination of the Windows CE kernel, we are able to provide a complete yet user-friendly implementation.

Evaluation (Section 6). We provide a comprehensive usability evaluation of the xShare implementation through performance measurements and two user studies. Our measurements show that xShare barely affects the overall system performance in Shared Mode. Our first user study shows phone owners' subjective opinions were extremely positive, almost unanimously responding that xShare is useful, satisfies their needs, and is easy to learn and use. With minimal training, our participants were able to specify common sharing policies in approximately 20 seconds. Our second user study was with Windows Mobile developers as malicious phone borrowers. It shows that xShare is resilient to attempts of unauthorized access. It further shows that xShare provides a satisfactory performance and user experience with shared applications and data.

To the best of our knowledge, our work is the first publicly reported study on supporting phone sharing. While we intend xShare to promote phone sharing for both social and economic purposes, it is important to note that our evaluation is limited to the usability of xShare, and not its social and behavioral impact. In addition, xShare is not intended to make mobile phones more secure than they already are. Instead, it is intended to limit temporary users to explicitly shared services and data. Therefore, even with xShare, temporary users may be able to exploit existing security flaws in the phone to overpower xShare.

The rest of the paper is organized as follows. We discuss related work in Section 2 and present the motivational user studies in section 3. We present the design, implementation, and evaluation of xShare in Sections 4-6, respectively. We address the limitations of xShare in Section 7 and conclude in Section 8.

2. RELATED WORK

Technology Sharing. The underpinning motivation of xShare is that mobile phone users are interested in sharing their devices. In recent years, there have been several pieces of work that have investigated information technology sharing [1, 2, 4, 10, 11, 14]. In particular, the authors of [11] pointed out "face-to-face media sharing" is desirable but not well supported by the existing technologies. The authors of [2] studied the culture factors behind phone sharing in rural India. Without providing a solution, their work highlighted the necessity of phone sharing in under-served communities. Both works are motivational to xShare.

Virtual Machine (VM). The system implementation of xShare is related to existing work in OS and application-level virtualization [6-9, 12, 13, 15]. Yet xShare has a very different design goal. Virtualization solutions aim at isolating multiple VMs from each other and preventing them from altering each other's data. Yet, they may not necessarily prevent VMs from reading each others' data, or system data, e.g. [15]. In contrast, xShare aims at preventing a single VM from accessing non-shared data and applications. Therefore, xShare can be significantly lighter in employing a different approach. Some companies, such as VMware [17] have recently announced upcoming VM solutions for mobile platforms. However, as mobile devices are expected to remain processor and energy constrained compared to their PC counterparts, we expect that the additional overhead of VM solutions in terms of processing power and battery life would remain significant.

Support for Multiple Users. Conventional multi-user support, as present in desktop OSes, are designed for a computer that will be actively used by multiple but usually known users. It creates a complete system profile for each user and provides different prestige levels to each of them. In contrast, we aim at supporting a mobile device that is owned and actively used by a single user, and occasionally lent to other users. Therefore, there is no need to create a system profile for each possible user. While a *Guest* account may be used for all temporary users, it does not allow the owner to grant different temporary users with different accesses in an impromptu manner. For example, one may be willing to lend their phone to a colleague to make a quick call but only be willing to share some photos captured at a recent party with close friends.

Windows CE Kiosk Mode. Windows CE supports a Kiosk mode that can boot directly into an application without access to a shell, control panel, or any other method of launching other applications [3]. Similarly, some third-parties provide kiosk-like modes with multiple applications, e.g. SPB Kiosk Engine [16]. However, such kiosk modes are fatally limited for phone sharing. First, a customized Windows CE image is needed and a lengthy reboot is necessary, making impromptu sharing impossible. Moreover, they do not provide protection over data files.

3. UNDERSTANDING PHONE SHARING

In this section, we present findings from two user studies regarding mobile phone sharing. The first consists of interviews in four countries; the second is a four-month field trial with fourteen teenagers in the USA. The findings motivate the need for privacy protection during phone sharing and provide the design requirements for xShare.

3.1 International Interview

To learn about the current global status of phone sharing, we designed and conducted an interview of about 35 questions, of which several are open-ended. The interviews were structured and lasted between 30 and 45 minutes. Participants were required to have phones that at a minimum have a built-in camera and support video and music playback. They were recruited through the authors' social networks, e.g. by advertising the study through friends and coworkers. We recruited 60 participants in total, from China, South Korea, Iran and the USA, with a minimum of 10 from each country. All but one were aged between 18 and 40 years old. While we recruited the participants to provide a broad as possible sense of how phone users share their phones, we do not claim that our participants were a balanced or accurate representation of any demography.

3.1.1 Nature of Sharing

Our international interviews provide evidence that phone sharing is very popular and involves a wide range of applications, reasons, social settings, and relationships. 93% of our interviewees have participated in phone sharing, either as a lender or borrower, as shown in Figure 1. 22% of our participants only shared their phones when a borrower needed to make a phone call but did not have their phone at hand, or it was out of battery. We call them the *non-sharing type*. On the other hand, 64% of our participants lent and borrowed phones for other reasons as well, e.g. access to

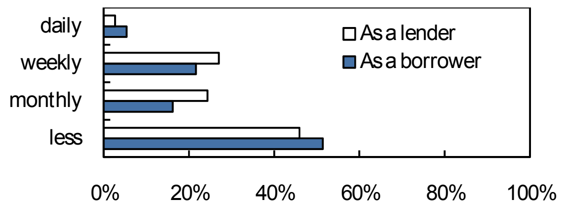


Figure 2: Frequency of phone sharing among our sharing-type participants, as a borrower (top) and lender (bottom)

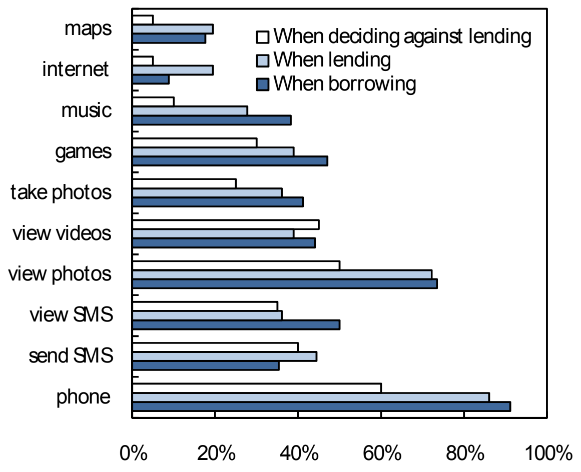


Figure 3: Applications involved in phone sharing

certain applications or data. We call them the *sharing type*. The non-sharing and sharing types had a significantly different frequency of sharing. All non-sharing type participants reported that they participate in sharing less than once a month. In contrast, more than half of the sharing type participants reported sharing their phones at least monthly, as shown in Figure 2. In the rest of this subsection (3.1), we will focus on and report *only* the sharing type, and "participants" refers to sharing type participants. The significant number of sharing type users highlights the potential for a privacy-maintaining phone sharing solution.

What applications: The most common applications shared are shown in Figure 3. As expected, placing phone calls was very popular. However, a wide range of other applications were also popular, and viewing photos was almost as popular as phone calls. We must note that many participants didn't have internet access on their phones, and data plans are very costly in some of the studied markets, therefore explaining the low usage. The range of applications that are shared indicates that a solution for phone sharing must be scalable and support a wide range of applications.

With Whom: Our participants shared phones with people from a wide range of social relationships, as shown in Figure 4. Because the same data may be considered as confidential or sharable depending on the relationship between

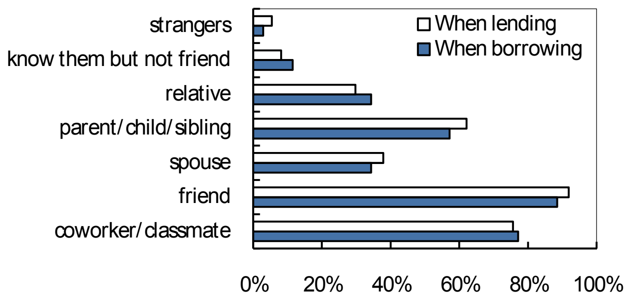


Figure 4: Relationships of those involved in phone sharing

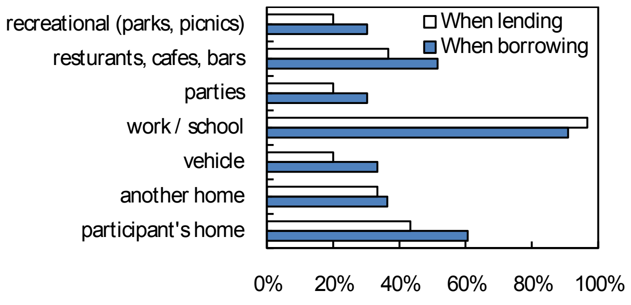


Figure 5: Locations of phone sharing

the parties, a solution for phone sharing must provide customizable access to phone functionalities and data.

Where: The most popular location for phone sharing was at school or in the workplace. However, sharing occurred at a variety of locations, both private and public, as shown in Figure 5. Therefore, a solution for phone sharing must be configurable in an impromptu manner and in any location. This also reinforces the need for a customizable solution, since the same data may be considered as confidential or sharable depending on the location.

Why: The most common reasons our participants shared their phones and used other people's phones are shown in Figure 6. We can see that it is very common to use a shared phone when the borrower's own phone is not at hand or is out of battery. However, accessing the owner's personal data is also a very common purpose of phone sharing. Therefore, a solution for phone sharing must provide privacy for the owner-created data. Showing or trying out a feature or a different phone, and accessing features unavailable on a user's own phone were also popular. Therefore, a solution for phone sharing must be able to support new applications installed by the owner.

Who is the initiator: When a borrower uses the lenders mobile phone, either the borrower or the lender may have initiated the use. That is, the borrower may ask the lender to use their phone; or the lender may ask the borrower to use their phone. When the borrower is the initiator, it is crucial that a privacy-maintaining solution can be configured quickly to avoid embarrassment. While all our participants reported being asked to share their phones, 64% of our participants reported initiating the sharing of their phones as well. This shows that many participants used sharing for social purposes by offering their phones to others.

3.1.2 *Privacy Concerns*

Our interviews highlight that privacy remains a major concern for phone sharing: the most common reason mentioned by our participants for not sharing their phone was concerns regarding the third party's access to their private data. This was followed by concerns regarding the borrower assuming their identity, e.g. caller ID, introducing too much monetary cost, and using too much battery, as shown in Figure 7.

Classified user data: We asked participants to select what types of information on their phones prevented them from

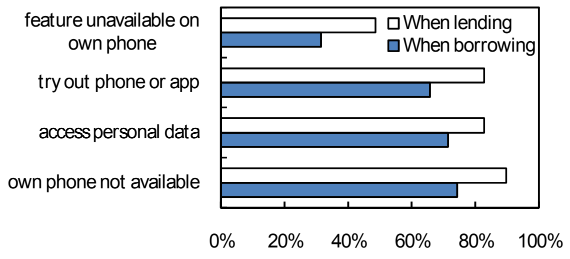


Figure 6: Reasons for sharing phones

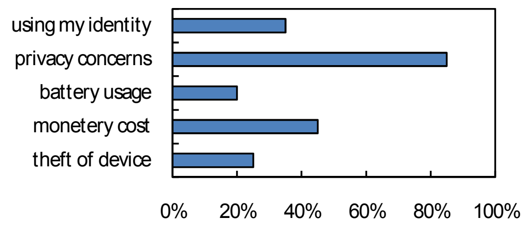


Figure 7: Reasons for deciding against sharing

sharing their phones, and which one was the most important. Photos and videos, text messages (SMSes), and contacts, were mentioned by most participants, along with a wide range of other items, as shown in Figure 8. The diversity in what types of personal information users consider private and most private confirms that we would need to support fine-grain access control for a wide range of data types in order to provide meaningful privacy to all users.

Existing Protection Inadequate: The built-in access control for most of our participants' phones was very simple: a password or PIN code for accessing the entire phone. A few of the phones had the ability to password protect certain functions, such as access to the user storage or to specific applications. However, all but one of participant told us they rarely or never use it, and some told us that in order for borrowers to access the shared data, they would have to disable the password protection anyway, so it is not useful.

How owners deal with concerns: Lacking useful privacy protection on phones, we found that our participants dealt with their concerns regarding phone sharing in three distinct ways. First, they simply share their phone less often. 56% of them recalled instances where they refrained from sharing due to privacy concerns, and all but three of them told us they would be more willing to share their phone if it supported better privacy protection. Second, the owners casually supervise the phone when sharing: 86% of our participants told us that they usually or always keep their phone in sight when sharing, and 60% told us they usually or always keep an eye on what the borrower is doing on their phone. This enables the enforcement of mutually agreed privacy boundaries, but places an extra physical and mental burden on the lender and may make the borrower feel uncomfortable or mistrusted. Third, the owners prepared their phones before handing it to others. 54% of our participants reported preparing their phone by moving or deleting private data, enabling or disabling certain features, and/or bringing the application being shared to the front. Of those, 55% reported that such preparation takes more than 30 seconds and 75% thought it takes too much time. This highlights the need for minimum latency when entering Shared Mode.

3.2 Long-Term Field Study

While our interviews examined sharing behavior and concerns of existing phone owners, they provide little insight into the underlying reasons they share phones and how they have reached this level of sharing. We have recently concluded a four-month field trial of a Windows Mobile phone in Pecan Park, a low-income urban community in Houston, TX. In the field trial, we distributed HTC Wizard phones to 14 high-school students from the community, recruited through a local non-profit organization. We conducted focus groups with the participants regularly throughout the study, on average every three weeks. The focus groups were recorded using a voice recorder, and later transcribed, coded, and analyzed manually.

We made the following observations regarding their sharing behavior. First, initially, all participants actively shared the mobile phones as a social networking tool, at various locations, including home and school, and with people of different relationships with them, from family members to newly introduced peers. Second, sharing decreased as the study went on. This was in part due to the diminishing excitement of having a new phone; and more importantly, the increasing privacy concern as the phones became personalized with privacy-sensitive data such as contacts, photos, and SMSes. Their sharing circle eventually became very small, limited to very close friends or siblings. Third, sharing is mostly impromptu, taking place at time and location of convenience and social significance, and more importantly, without prior planning. This is because sharing is driven by a social purpose and mobile phones can be conveniently used at most locations and times. Fourth, sharing is usually application-driven and data-driven. That is, as observed in our international interviews, the phone owners lend their phone to others for accessing specific data with a specific application, e.g. playing a song with Media Player and viewing photos under a certain directory. Fifth, the sharing policy is dynamically determined, mutually agreed upon implicitly, and often enforced casually. The lender often keeps a watchful eye to ensure that the borrower does not go beyond the mutually agreed boundaries, similar to what we found out from the first user study.

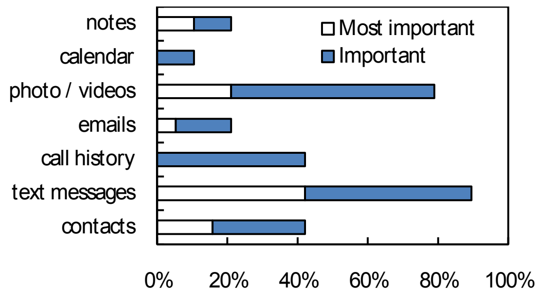


Figure 8: Private information on owners' phones that prevents them from sharing their phones

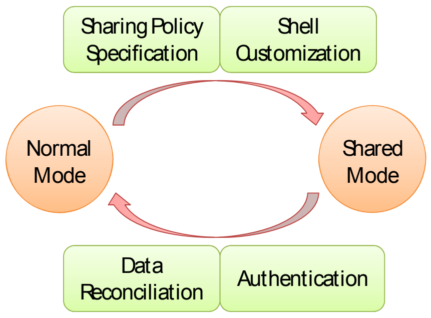
3.3 Threat Model and Design Requirements

Our two user studies highlight the need for a solution to the privacy challenge of sharing phones. Further, they show that sharing serves social purposes. In order to retain the value of sharing, a privacy solution must support impromptu configuration for a temporary guest user with minimum latency. This was further confirmed by our participants' opinion on preparing the phone to maintain privacy while sharing (e.g. by deleting/moving private data), 75% who had prepared the phone in this way told us it takes too long.

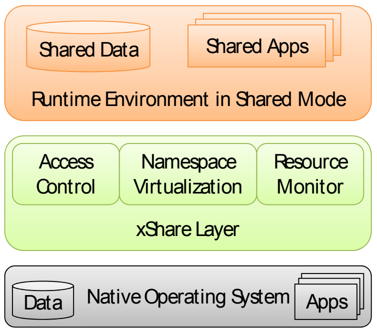
From our user studies, it is apparent that the user often knows the borrower and lends the phone under a socially defined agreement as to the limits of the borrowers' access. Therefore, the threat in phone sharing is from a curious or careless borrower who may snoop into or accidentally get exposed to personal data. Further, the borrower may inadvertently consume excessive exhaustible or billable resources, such as battery and cellular minutes. Therefore, we aim at limiting the data and services accessible to the borrower by providing in-situ configurable access control and resource accounting. We base our threat model on an attacker centric approach, considering the careless or curious borrower. It is important to note that our goal is not to make the system more resilient against existing security flaws that may be exploited by intentionally malicious borrowers, as well as worms and viruses.

Based on the findings from our user studies and the threat model described above, we can distill the following design requirements for supporting phone sharing through xShare.

- **Impromptu policy creation:** xShare should allow the owner to specify a policy for what to share with minimum latency. This is more important than the latency for exiting Shared Mode, as it can impact the social value of sharing.



(a) Mode transition



(b) Architecture of Shared Mode

Figure 9: Overview of xShare: xShare has two modes, Normal and Shared. From Normal to Shared, the phone owner specifies which files and applications to share. In Shared, xShare creates a virtual environment containing only the shared files and applications through access control and namespace virtualization. From Shared to Normal, user authentication is required

- **Access control:** xShare should provide access control for three different categories: individual applications, data files and folders, and system resources such as storage, battery, minutes, and cellular data usage.
- **Resource accounting:** to control exhaustible system resources and pay-by-use services, xShare should provide accounting for them and track their usage.
- **Borrower data reconciliation:** in many instances, the borrower may create or modify data files and system settings. For example, 65% of our participants shared their phone for a borrower to try out, and 36% shared their phone for the borrower to take photos. Therefore, xShare must track borrower changes and allow the owner to accept or reject them.

In addition, xShare must be resource efficient because mobile phones are highly resource-constrained. Further, the borrower's experience should be comparable to the owner's.

4. DESIGNING XSHARE

Based on the findings from the user studies, we present our platform-independent design of xShare and provide rationales for each design decision.

4.1 Design Overview

xShare runs atop of the OS. It has two modes: Normal and Shared. In Normal Mode, it appears to be a regular application that can be launched and closed, and does not affect the operation of other applications. In Shared Mode, it runs in the background, customizes the system shell, and enforces the sharing policy based on the owner's specification. Figure 9 provides an overview of xShare, in particular, transitions between the two modes and the architecture of Shared Mode. When switching from Normal Mode to Shared Mode, xShare provides a UI for the owner to specify the *sharing policy*: what files and applications to share. xShare then creates a virtual environment and runs in the background enforcing the policy. When switching from Shared Mode to Normal Mode, user authentication is required. After that, xShare provides another UI for the owner to either accept or reject changes made in Shared Mode. We call this *borrower data reconciliation*. Therefore, the key components of the proposed design include UIs for sharing policy specification, virtual environment creation and access control, resource accounting, and borrower data reconciliation.

File-based Access Control: A key design decision we made is to base xShare's access control at the file level. There are multiple rationales behind this design decision. First, all major mobile operating systems, including Symbian, Linux, Windows Mobile, iPhone OS, Blackberry, and Palm, support files as an abstraction for both data and applications, implement file systems, and provide APIs for file operations. In addition, Unix-style mobile OSes, including the iPhone OS, already provide some access control for the file system. Second, most applications on major mobile OSes leverage files as a level of data abstraction, and our user studies show that accessing personal data is one of the popular reasons for phone sharing (Figure 6). Therefore, a file-based access control allows an application-independent solution. Third, file-based access control can be implemented without rebuilding the ROM image. While finer access controls are possible and may be desirable, e.g. access control to individual entries in the SMS history, they will require application-specific implementation.

Many popular smartphone OSes, including Symbian and Windows Mobile, do not support file access control. Therefore, xShare has to provide file access control, which may be platform-dependent. In Section 5.1, we will present the case for Windows Mobile. Linux and iPhone OS provide Unix-style file access control with which read, write, and execution rights can be granted to a user. Even with such file-level access control, there are multiple practical challenges to realizing access control for xShare, as will be addressed in Section 4.3.

4.2 User Interfaces

xShare provides multiple UIs for the owner. It also customizes the shell environment for the borrower.

Policy Specification: xShare provides a UI for owners to rapidly specify a sharing policy. A sharing policy consists of three components: shared files selection, shared service/application selection, and resource allowance specification (Figure 11). xShare obtains the list of sharable files and applications from the OS. This is possible because mobile OSes typically store user-created data in a specific folder and install applications or their links in another specific folder. We utilize a hierarchical list-based selection that lists all possible choices. For file sharing, we employ a hierarchical list similar to the built-in file browser to enable the owner to explore and select individual folders and files. Each item on the list has an indicator that indicates the sharing status of the file, folder, or application. Clicking on the indicator changes the sharing status. Below are several design considerations for quick policy specification.

- Because the OS associates a default application to most file types, xShare asks the owner to select the shared files before applications and automatically selects appropriate applications for the selected files. Obviously, this can be overridden by the owner when they select the shared applications.
- All items start as not shared, except those applications that are automatically selected based on the shared data file types.
- The owner specifies whether a file or application is shared or not. Only shared applications can be run, and xShare gives them read and write access to shared files, as well as permission to create new files. xShare employs profiles to enable the owner to save frequently used sharing policies.
- xShare provides a quick launch method, called Quick Share. With Quick Share, xShare enters Shared Mode only allowing access to the current front application and the files currently opened by it. This is motivated by our user studies; 76% of our Sharing Type interviewees reported that they rarely or never shared multiple applications in one sharing session. To use Quick Share, the owner first launches xShare, brings the application to share to the front and opens files to be shared, and then confirms Quick Share from xShare.

Shell Customization in Shared Mode: The shell UI of a phone often shows some data items such as calendar events and contains icons or links to launch applications. In Shared Mode, these items are hidden from the borrower by concealing the non-shared data items and removing the icons and links of the non-shared applications. The borrower can still access shared data applications from the shell. Therefore, a consistent user experience is maintained in Shared Mode.

Borrower Data Reconciliation: When exiting from Shared Mode, xShare prompts the owner to manage changes made to files and settings in Shared Mode. It displays the changes and their location in a UI similar to the policy specification UI. The owner can reject or accept the changes. The default choice for modified items is *reject*; the default for new files is *accept*. This is based on our user study findings that show most file sharing is intended for read only access, e.g. music and pictures. In contrast, many shared applications are intended for the borrower to create files, e.g. the camera.

4.3 Access Control

The key component of xShare is impromptu configurable access control to applications and data. Once the owner specifies the sharing policy, xShare must rapidly identify the files it should grant access in order to block access to non-shared applications and data. We next present our design decisions for several important practical challenges.

In-Memory Services and Applications: Mobile OSes can keep some key services and recently used applications in memory to expedite their launch. Such services and applications create a challenge to access control when shared, due to the private data they may have loaded. For example, on Windows Mobile, all SMSes are stored in the file "cemail.vol", which is kept open once the OS is booted. As a result, if the owner allows the borrower to use the SMS application, they may allow the access to the SMS history. Most sharable applications and services can be terminated or restarted without any impact on the rest of the system. Therefore, xShare simply terminates their corresponding processes before entering Shared Mode. Further system and application cooperation may eliminate the need of terminating and restarting specific applications. Yet it requires modifications to existing applications.

Certain applications and system services are so tightly integrated with the system that they cannot be terminated properly. Addressing this issue is highly dependent on the APIs provided by these services and applications. We will describe our solutions when we present our Windows Mobile-based implementation in Section 5.

Identifying Files for Application Sharing: Many applications require more than their executable files to run, e.g. configuration files and DLLs. xShare must locate an adequate set of files for the application to run properly. It is generally difficult to distinguish between files necessary for an application to run and private data files that are opened by the user in that application. However, most mobile OSes, including Windows Mobile, Symbian, and iPhone OS, store application files and data files in different folders. Therefore, xShare simply allows access to all the files in the same folder as the corresponding executable. Some applications may access non-storage peripheral devices, such as the camera and the microphone. These non-storage peripherals do not contain any private information, therefore xShare grants access to them in Shared Mode.

Implementing Access Control to Files: Once the files to be shared are identified, xShare creates a file that contains the sorted list of all shared files, which we call the *container file*. In Shared Mode, xShare loads it into memory and employs a binary search to determine whether a given file is in the list and if access to it should be granted.

4.4 Virtual Environment

Given the sharing policy, xShare creates a virtual environment for Shared Mode. The virtual environment confines access to the shared data and applications through namespace virtualization based on file-level access control. In the virtual environment, xShare tracks all changes made by permitted applications and allows the owner to manage them when exiting Shared Mode.

Namespace Virtualization: Namespace virtualization is the natural choice for sandboxing the shared applications and data to implement access control and a virtual environment for Shared Mode. Namespace virtualization controls resource access by renaming those resources. By concealing unshared files, redirecting write access, and maintaining consistency, namespace virtualization provides a view of the system resources that is consistent with the sharing policy.

Change Separation: Namespace virtualization allows xShare to separate changes made to files and settings in Shared Mode and ensure those changes cannot affect the system in Normal Mode. Recall that xShare only asks the owner to decide which files or directories to share. Once a file is shared, xShare allows both read and write access to it in Shared Mode. To protect the shared files from unwanted modification, xShare creates a private folder to hold all the modified and created files in Shared Mode. Instead of changing the original files, xShare employs *copy-on-write* to re-direct all the changes to the private folder. We will provide a detailed explanation in Section 5.2.

Hiding Non-shared Files: Through namespace virtualization, xShare hides non-shared resources from shared applications in Shared Mode to provide a consistent system view in Shared Mode. For example, if we share two of the five files in a folder, when an application lists the folder, it will only see the two shared files.

4.5 Resource Accounting

As indicated by our user studies, phone owners are concerned with the over-usage of exhaustible system resources, such as battery and storage, and network chargeable features, such as phone minutes, SMS and data counts. Existing mobile operating systems provide APIs to assess battery capacity and file sizes and xShare leverages them for battery and storage usage accounting. xShare enables the owner to set restrictions on the amount of resources used in Shared Mode, e.g. 5 MB storage or stopping sharing when the battery drops to as 20%.

5. IMPLEMENTATION

Windows Mobile provides rich support for development, especially for system-level programming, but presents two unique challenges for xShare implementation. First, it provides no file-level access control, unlike its desktop counterpart, the iPhone OS, and Linux. Second, Windows Mobile tightly integrates system services and critical applications; as a result, changes to one are likely to impact others. We next present our solutions in addressing these challenges. While we focus on the Windows Mobile implementation, some components, such as namespace virtualization and the UI, are common for both Windows Mobile and other mobile OSes, such as the iPhone OS.

5.1 API Interception for File Access Control

Similar to most mobile OSes, Windows Mobile provides no native support for strict file-level access control. Any application can use the *CreateFile*() to open any file. We implement file-level access control for Windows Mobile by intercepting system APIs at the kernel-level. API interception is a natural choice for file-level access control since all applications use system APIs to access files. While access control can be implemented at either the user-level or the kernel-level, user-level implementation requires changing the phone's ROM image. This is because Windows Mobile extensively employs eXecution-In-Place (XIP), i.e. executing native programs and DLLs directly from ROM without copying them into RAM, and therefore we cannot change the import address tables of programs or the export address tables of DLLs [5] without rebuilding the ROM image.

Implicit System APIs: Windows Mobile uses a client/server model for APIs and implements all system APIs in separated server processes. Its system APIs are either implicit or handle-based. Implicit APIs are globally registered and dispatched through the system API table, e.g. *CreateFile*(). Handle-based APIs are attached to a kernel object such as a file or an event and called through a handle to the kernel object, e.g. *WriteFile*(). As shown in Figure 10, when an application calls a system API, it causes a trap and jumps into the kernel. The kernel then searches the system API set table for the address of the method implementing the API and calls the method.

xShare intercepts system APIs by locating the system API set table and manipulating its entries. We implement our interception routines in the xShare interception DLL, load it into the address space of an API set server process, and direct the corresponding entry in the system API set table to an interception routine. Consequently, the corresponding system API calls will be directed to xShare interception routines for access control. The interception DLL also holds the pointer to the original table in the server process so that the interception routines can call the original system API methods if the access control checking is passed.

Handle-Based System APIs: Replacing entries in the system API set table can only intercept implicit APIs. As all handles are created in implicit APIs, we track their creation in corresponding implicit interception routines. Before returning those handles to applications, we attach our own handle-based interception routines to them. As a result, all calls to the handled-based APIs are directed to the xShare interception DLL. Figure 10 illustrates how we intercept both implicit and handle-based API sets, using file system APIs as an example.

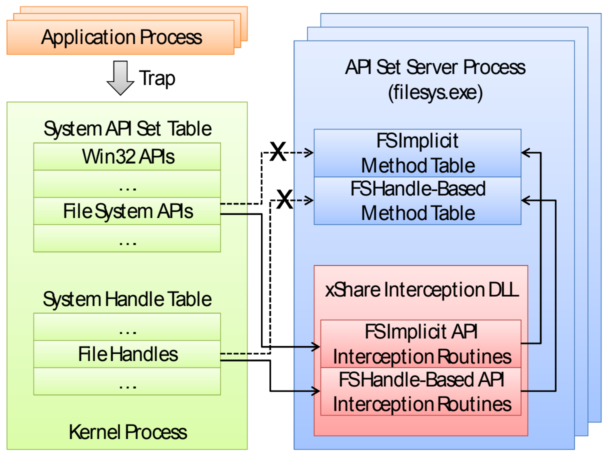


Figure 10: System API interception on Windows Mobile: xShare injects its interception DLL into the API set server processes and directs the corresponding entries in the System API Set table to its interception routines

Load Interception DLL: To intercept the APIs, we need to load the xShare interception DLL into the corresponding server process. We leverage the capability of Windows Mobile to perform a callback function in a given process. By setting the callback function to *LoadLibrary*() and its parameter as the name of a DLL, we force a process to execute *LoadLibrary*() to load the DLL into its address space. Similarly, we can unload it by setting the callback function to *FreeLibrary*(). Using this approach, we are able to dynamically load and unload the xShare interception DLL. When switching between Normal and Shared Modes, xShare automatically changes the system API set table and loads or unloads its interception DLL for the target processes.

Access Control Implementation: As described in Section 4.3, xShare generates the container file based on the user-specified sharing policy. It employs an interception routine for *CreateFile*() to enforce the policy. The routine takes the path parameter passed to *CreateFile*() and looks it up in the container file, using a binary search to determine whether access to the file should be granted. If so, it calls the original *CreateFile*() to open the file and returns the file handle. Otherwise, it denies the access by returning an error. Because *CreateFile*() is called to open the executable when an application is launched, the interception routine ensures access control for both files and applications.

5.2 Namespace Virtualization

We implement xShare namespace virtualization on a Windows Mobile by system API interception.

5.2.1 File System Virtualization

The main purposes of file system virtualization are to track and separate changes made in Shared Mode, hide non-shared files and provide a consistent appearance to the borrower. Because a Windows Mobile application may use a variety of APIs in arbitrary order to modify existing files/directories and create new ones, it is nontrivial to correctly track all these changes, maintain correct states, and ensure a consistent appearance regarding the file system in Shared Mode. In total, we intercept 18 file-system APIs, and employ a virtual link technique to track changes and maintain consistency, described below, according to the design described in Section 4.3.

Change Separation through Path Mapping: We employ a *virtual-intermediate-physical* path mapping to separate changes made in Shared Mode. We call the path used by an application the *virtual path*, and translate it into the *intermediate path* by prefixing it with "\xShare\Root". We call the path to store the file in Normal Mode the *physical path*. For existing files that are shared from Normal Mode, their physical path is initially the same as their virtual path. For files created in Shared Mode, their physical path depends on how they are created. We employ a *virtual link* to maintain the mapping relationship between an intermediate path and its physical path. For each intermediate path, we create its virtual link as a file in the intermediate path by suffixing it with ".vlink". The virtual link file contains the physical path of the corresponding intermediate path. If a file shared from Normal Mode is opened for write in Shared Mode, xShare copies it to its intermediate path and generates its virtual link file. If a file is copied or moved to another path, xShare generates a virtual link file under the intermediate path of the destination. If a file is deleted, xShare treats it as moving the file to a *virtual recycle bin*. For pre-existing shared files that have not been changed in Shared Mode, their virtual path is the same as their physical path and there is no need for a virtual link. By updating the corresponding virtual links, we record all the file change operations under "\xShare\Root\" without directly modifying any file shared from Normal Mode. This enables xShare to allow the owner to later manage changes made in Shared Mode.

Hiding Non-shared Files: To hide non-shared files, the interception routine for *CreateFile*() returns *ERROR_FILE_NOT_FOUND* when an application tries to open a non-shared file. We also intercept *FindFirstFile*() and *FindNextFile*() so that when an application searches a directory, non-shared files are invisible. To ensure a consistent view of the virtual file system in Shared Mode, xShare sets and obtains the attributes of the requested file using its intermediate path for each call to *SetFileAttributes*() and *GetFileAttributes*(). For each call to *FindFirstFile*(), it returns a fake *Find* Handle and creates two internal *Find* Handles: one for file access through the physical path, the other for file access through the intermediate path. On each call to *FindNextFile*(), we internally iterate and filter the files in the two paths, and return the next file that is visible in Shared Mode.

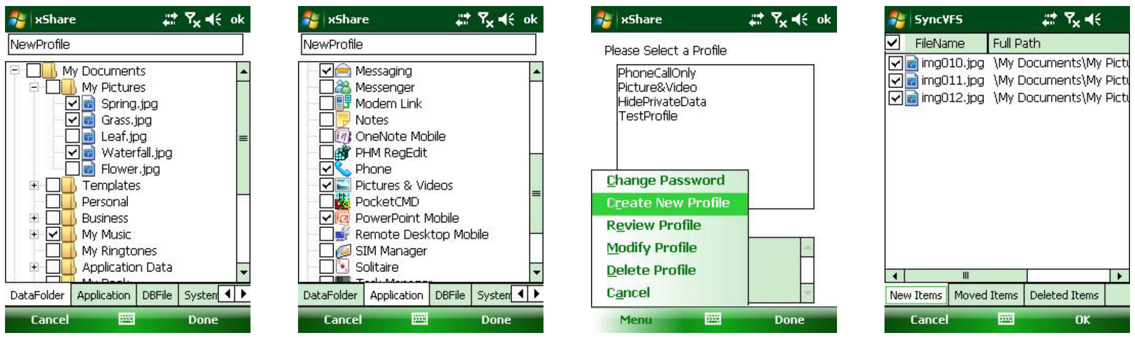


Figure 11: User interface for owner. From left to right: select files or folders to share, select applications to share, manage profiles, and review the files changed in Shared Mode

5.2.2 *Registry Virtualization*

We implement namespace virtualization for the registry as well, to protect it from unwanted changes. The registry plays a key role on Windows Mobile; it stores data and settings of the system, and information about applications and drivers. As a temporary user may change registry settings in Shared Mode, xShare virtualizes registry access to track the changes and separate them from Normal Mode. The technique used for registry virtualization is the same as the one for file system virtualization, but the intercepted APIs are different. We intercept 10 registry APIs in total.

5.3 Managing Unstoppable Services: CEMAPI

In Section 4.4, we described the challenge that a mobile OS may load private data into memory through services that are started at boot time. Protecting such data highly depends on the interaction between the service and the system. As an example, we next present our solution to an important service of this kind on Windows Mobile, CEMAPI, which provides COM style APIs for applications to access SMSes and emails. CEMAPI keeps SMS history in memory. Unfortunately, the shell process, shell32.exe, loads CEMAPI at boot time and CEMAPI cannot be terminated as it is tightly integrated with the OS. Therefore, it will retain the SMS history until reboot, further complicating SMS history protection when the SMS application is shared.

Our solution is to leverage the CEMAPI APIs as necessary upon entering Shared Mode to read all existing SMSes, save them into a non-shared file, and delete them from CEMAPI internal memory. As a result, the borrower will see none of the owner's SMSes. When switching back to Normal Mode, we restore all backed up SMSes to CEMAPI internal memory.

5.4 User Interfaces

We have implemented the UI design described in Section 4. We next provide details specific to Windows Mobile.

Owner UI: When xShare is launched for the first time, it prompts the owner to set a password and then stores the MD5 hash code of the password in a registry key. In Shared Mode, the registry key is visible only to xShare itself through access control. To switch back to Normal Mode, the owner must input the password correctly.

We have implemented a hierarchical list-based UI design for policy specification and borrower data reconciliation, as illustrated in Figure 11. In addition, through the xShare UI, the owner can configure other xShare properties, such as whether installing new applications or connecting to a PC is permitted. The owner can also limit the resources used in Shared Mode, specifically the usable storage capacity, battery percentage, and network traffic volume. These resources can be tracked using APIs already available on Windows Mobile. A button is devoted to Quick Launch, which places the phone into Shared Mode, with the application in the front and all of its open files being shared.

Borrower Shell Customization: The standard shell interface of Windows Mobile consists of the following main components: the start menu, title bar, today screen, tray bar, and menu bar. The user can launch programs from any of the above mentioned components, with the exception of the title bar. In addition, most Windows Mobile phones have several physical buttons to launch pre-defined programs, such as placing phone calls or starting the camera. To enable a clean and simplified shell listing only shared applications and data, xShare customizes the shell for the borrower so that only shared applications and data are shown.

6. EVALUATION OF XSHARE

We evaluate our xShare implementation using the HTC Wizard phone. We use Windows Mobile 6.1 Professional with CE OS 5.2. Our evaluation consists of two major parts, measurements and user studies.

6.1 Measurement

We measure the memory footprint of xShare, its latency for switching from Normal Mode to Shared Mode, and the execution overhead in Shared Mode in terms of performance and energy. Our measurements demonstrate that our implementation achieves the design objectives in efficiency and mode switching latency.

6.1.1 Memory Footprint

xShare has two components: the interception DLL and the UI program. The interception DLL has 4,841 lines of C++ code and takes about 90 KB memory. The UI program has 5,023 lines of C# code, excluding Visual Studio auto-generated code, and takes up to 1.3 MB memory. As the UI program is only used for mode switching, xShare requires only 90 KB memory in Shared Mode.

6.1.2 Mode Switching Latency

As our user studies in Section 3 indicate, a key design requirement for xShare is that the owner must be able to switch to Shared Mode with minimum latency. We break down the latency according to the main tasks involved in switching, as below.

- Terminate the running application processes (T1);
- Create the container file based on sharing policy (T2);
- Backup and delete SMSes (T3) if SMS is shared;
- Inject the interception DLL (T4);
- Customize the shell (T5);
- Other minor tasks (T6).

Measurement Procedure: To measure the latency of switching to Shared Mode, we first create a representative system context on the HTC Wizard by loading 50 SMSes and launching Contacts, Messaging, Solitaire, ActiveSync, Phone and Notes applications in Normal Mode. We then specify the sharing policy to allow access to Pictures & Videos, Solitaire, Phone and Messaging applications along with 10 photos. We repeat the experiment 10 times.

Results: Figure 12 shows the average time costs to perform these tasks sequentially. The sum of all the time costs is less than 5.8 seconds. We can see that it takes more than a second to terminate the existing processes. The reason is that xShare has to wait for the processes to release their resources and exit. Fortunately, this latency can be easily concealed from the owner: as it will take the owner a few seconds to interact with xShare to specify a policy, process termination can take place simultaneously in the background without introducing any noticeable latency.

Another potentially lengthy task is to backup and delete the existing SMSes (~1.8s). The main reason is that deleting SMSes using CEMAPI APIs is quite slow. As aforementioned, xShare backs up SMSes only if the owner shares the SMS application. Moreover, similar to process termination (T1), this step can also take place in parallel as (T4-T6) after the owner specifies the SMS application is to be shared.

Interception DLL injection creates the virtual environment for Shared Mode. The time cost is less than one second (0.83s). It shows that xShare is able to create the virtual environment very quickly, even on our phone with just a 195MHz CPU.

We also measure the latency of switching back to Normal Mode from Shared Mode. It takes about 3 seconds to unload the xShare interception DLL, terminate the applications launched in the Shared Mode, restart the auto-start programs, restore the shell and system settings, and other necessary tasks. However, it can take place simultaneously with borrower data reconciliation. The time cost of borrower data conciliation depends on how much time the owner spends to review the data.

6.1.3 Execution Overhead

Because xShare is based on file-level access control, it introduces an overhead to most file accesses when enforcing the sharing policy. Our measurements show xShare access control has little effect on the overall system operation in terms of performance and energy consumption. Moreover, as xShare does not intercept *ReadFile*() and *WriteFile*(), there is no overhead for reading and writing opened files. Unlike VMs, xShare will not introduce any overhead to other system operations. We next present measurements for a set of carefully designed benchmarks that involve file access in very different ways. Each measurement is repeated 10 times and we present the averaged results.

Benchmarks: We evaluate the performance impact of xShare on applications by preparing four benchmarks, as described below, and comparing their performance in Normal Mode and Shared Mode. First, since *CreateFile*() is called for every file access, we compare the time costs of calling *CreateFile*() on 10, 20, 50 and 100 empty files. We measure three cases: open existing files for read, open existing files for write, and create new files. Second, since xShare conceals non-shared files when listing the files in a folder, we use a test folder containing 100 files to measure the time costs when different numbers of files are shared. Third, to stress xShare with realistic file-intensive applications, we write a *GZip* program that compresses and decompresses 100 files from three folders. The file sizes vary from 2KB to 1MB and the total data size is 7 MB. Fourth, to evaluate the overhead in application launching, we write a C# GUI program of 9KB which simply opens a WinForm window and uses another program to launch the windows UI program and measure the launch latency.

Performance Overhead: From Figure 13 and 14, we can see that creating or opening a file for reading or writing, and file listing operations have a significant execution overhead in Shared Mode. The reason is that xShare has to access the corresponding .vlink file for each file access, thus increasing the file access cost, especially when opening a file to write, as xShare uses copy-on-write. For file listing, xShare must traverse both the physical folder and the intermediate folder. Maintaining the path mapping information in memory instead of storage may reduce the overhead but we do not adopt this approach due to two reasons. First, memory is scarce on most phones. Second, while the relative overhead of file operations is large, the absolute time cost is small. For example, it takes only about 11ms to open a file with read access, 62ms to open a file with write access, and 31ms to create a new file in Shared Mode. As most programs only spend a small fraction of its execution time opening, creat-

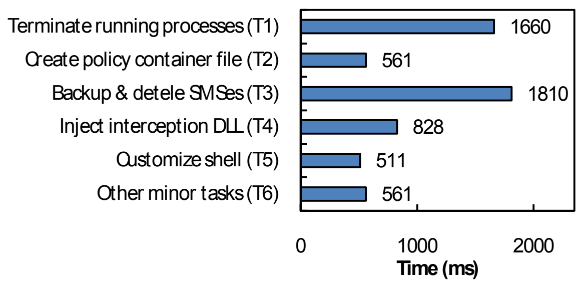


Figure 12: Switching latency between Normal Mode and Shared Mode

ing or listing files, the overall overhead of a program can be small. For example, the extremely file-intensive *GZip* benchmark takes only 5.9% extra time to compress and decompress the 100 files in Shared Mode. Launching our simple GUI application requires more than 50 API calls on *CreateFile()* and *GetFileAttributes()* for the OS and .Net Compact Framework to load the execution file and other system binaries and resources. Yet, it only takes an extra 270ms (10%) to launch in Shared Mode.

Energy Overhead: We have used a Data Translation DT9802 data acquisition module to measure and calculate the energy overhead of xShare in Shared Mode. We measure and compare the total energy consumption in three benchmark tests in Shared Mode and Normal Mode: compressing and decompressing 100 files using *GZip*; playing a one minute MP3 file (192kbps) in Media Player; and playing a one minute WMV video clip (1mbps, 30fps, 640x480) in Media Player. Our measurements indicate the file-intensive *GZip* benchmark costs only 4.7% extra energy in Shared Mode. We observed no measurable difference in energy consumption for the audio/video playback benchmarks. We attribute this to the fact that their main file operations are reading files which has no execution overhead in xShare.

6.2 User Studies

6.2.1 Evaluation by Owners

We evaluated the design of xShare by 12 mobile users to extremely positive feedback. The participants were aged between 18 and 39 years old and were recruited through the authors' social networks. They all used a PC daily and 5 of them were existing Windows Mobile users. We designed and conducted a user study in which participants were given instructions on how to operate the phone if necessary, and then given an introduction to various xShare functionalities. Each participant played around with xShare afterwards for least five minutes, before being asked to specify policies *A* and *B* from Table 1 three times in succession, starting from the phone's home screen. To measure the participants' performance, we timed them on each run using a stopwatch. Figure 15 shows the average time required by the participants separately, for prior Windows Mobile users and non-users. Our results indicate that xShare has a very quick

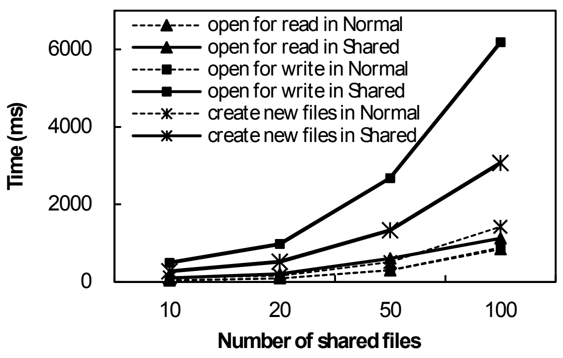


Figure 13: Time costs to open files for read, open files for write, and create new files in Normal and Shared modes

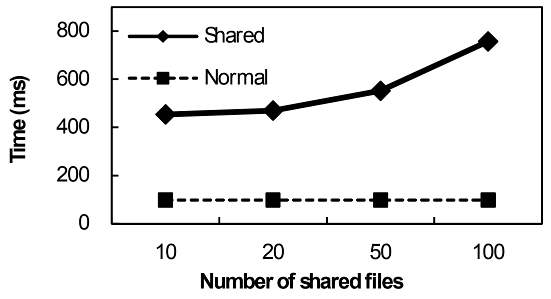


Figure 14: Time costs to list a folder of 100 files when 10, 20, 50, and 100 files are shared

learning curve; our participants required only 20 seconds on average to specify the last policy, and non-Windows Mobile users required only 17% more time compared to prior Windows Mobile users.

After the timed experiments, we interviewed the participants with a structured survey about their subjective opinions on xShare. All participants agreed that xShare is easy to learn, and all but one agreed that it is useful. Only two participants responded that using xShare "takes too much time" and only one told us it is not "easy to use". In contrast, nine of the participants responded their phones password protection "takes too much time". We attribute this to two factors. First, some phones may require entering the password at every access. Second, the built-in password protection takes too long compared to the little value it provides.

We received extremely positive feedback from our participants about xShare Quick Launch mode and borrower data conciliation. All but two responded that Quick Launch is useful. All the participants agreed xShare is quick and easy to use. Similarly, all but one responded that xShare borrower data conciliation is useful. Only one participant responded that it takes too much time and another responded that it is not easy to use.

Overall, all participants responded that xShare satisfies their need for maintaining privacy when sharing their phone; in contrast, only one had the same opinion about the built-in password function of their phone. Furthermore, all but two

Table 1: Policies in Evaluation

Policy	Apps.	Data
A	Phone call/SMS	none
B	Camera	5 photos

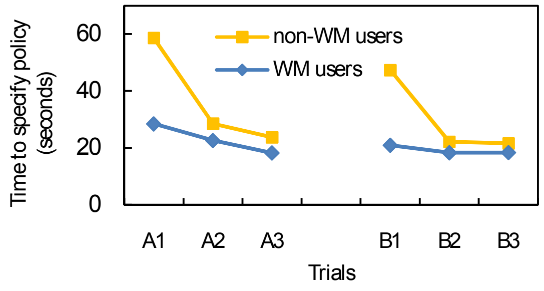


Figure 15: Average time to start xShare and specify a policy, for WM (Windows Mobile) and non-WM users. Each policy was timed in three consecutive trials

responded that they would feel more comfortable sharing their own phones if they had xShare.

6.2.2 *Evaluation by Borrowers*

We evaluated the subjective performance, usability, and security of xShare for borrowers by 8 expert Windows Mobile users to extremely positive feedback. The participants were aged between 18 and 39 years old and were recruited through the authors' social networks. They all had significant Windows Mobile experience, either as a programmer or as a power user who had at least changed their OS (ROM image) and phone software. In the user study, the participants were given a brief introduction on xShare and presented with two identical Windows Mobile phones, one without xShare and the other in xShare Shared Mode with the policies specified in Table 1.

To assess the security of xShare, the participants were offered a $30 prize for breaking xShare protection in either of the two policies. Breaking xShare protection was defined as accessing any of the functionality or files not intended for sharing, i.e. protected by xShare. Participants were allowed to do anything with the phone, including connecting it to a PC, with the exception of physical access to the phone internals and reflashing/reformatting the phone. Each participant had 30 minutes to break each profile, and we extended the time for any participant who requested it. Still, none of the participants were able to break xShare protection.

To assess the performance and usability of xShare, we interviewed the participants with a structured survey regarding their subjective opinion on the shared applications and phone. We focused on two items: 1) user friendliness, and 2) performance of xShare. We asked the participants to perform several identical tasks with the non-xShare phone and the phone in Shared Mode. They were encouraged to perform each action on both phones as many times as necessary for an accurate assessment. The items we compared were the general phone UI, making a phone call, sending and receiving SMSes, and taking and viewing photos.

All participants responded that the UI of the xShare phone in sharing mode is consistent with the non-xShare phone. Furthermore, all but one participant told us that the two phones have the same user friendliness and speed for all the inquired functions. The one exception told us that the phone in Sharing Mode is actually faster to use in some cases since he has to go through less unwanted items. This confirms the fact that xShare does not induce a noticeable latency for most operations.

7. DISCUSSION

xShare is intended to provide usable, lightweight protection against unauthorized access by borrowers, instead of making the system more resilient against attacks to existing security flaws. While none of our participants managed to break xShare protection, this does not mean that xShare is unbreakable. Even with xShare, existing security flaws in the phone may be used to compromise the phone and xShare. Even so, xShare improves overall system security in Shared Mode by providing file access control. For example, without xShare, the borrower can use the Internet Browser to download pre-prepared malicious software. With xShare, the malicious software cannot be simply downloaded and run, as xShare prevents launching an application that has not been explicitly shared. Malicious borrowers can still overpower xShare by exploiting inherent system security flaws, including ActiveX in Internet Explorer, and OS bypassing. Further, if someone hacked the system and posted the method online, other people would be able to leverage it. Addressing such unauthorized access based on system and application security shortcomings is out of the scope of this work.

xShare is limited in that it is built atop of the OS, assuming the ROM image may not be modified. As seen in our Windows Mobile based implementation, many challenges are indeed posed by the OS itself, e.g. regarding CEMAPI. Our design and implementation, however, provide insight into how the OS can be modified to better support sharing atop of the OS. Moreover, our work serves as a blueprint for possible OS-based designs.

Implementing xShare heavily depends on the underlying OS, and different approaches may be used on systems other than Windows Mobile. For example, on the iPhone OS, one may leverage the native multi-user support in the kernel to simplify the access control required by xShare. The shell and services on the iPhone OS are not closely integrated with the OS kernel and can be terminated and restarted. Therefore, in-memory data is less challenging for an iPhone OS implementation of xShare. Even so, implementing Shared Mode on the iPhone still requires system-level changes (e.g., system API interception) and non-trivial effort to create a virtual runtime environment and to virtualize resource access and separate changes made in Shared Mode.

8. CONCLUSIONS

In this paper we present a complete research and development cycle of xShare for friendly, efficient, and secure sharing of mobile phones. We find that phone sharing is very popular and involves a wide range of applications, reasons, social settings, and relationships. Yet privacy remains a major concern that prevents users from sharing their phone, and existing systems provide inadequate privacy protection for sharing. Our user studies highlight the need for an efficient and secure solution for impromptu sharing of mobile phones.

Based on these findings, we present xShare, a software solution to support friendly, efficient, and secure phone sharing on existing systems. xShare allows phone owners to easily create impromptu sharing policies to decide which files and applications to share and control how much resource a borrower can use. Using the sharing policies, xShare creates a virtual environment, called Shared Mode, where only the explicitly shared files and applications are visible. Further, xShare enables the owner to manage data files and settings created or modified by the borrower.

We present the design and implementation of xShare in detail. The core components of xShare are file-level access control and namespace virtualization. Through access control, xShare prohibits unauthorized access to non-shared files and applications and ensures the sharing policies specified by the owner. Using namespace virtualization, xShare conceals non-shared resources and tracks and separates changes made by the borrower in Shared Mode.

We evaluate our Windows Mobile-based implementation through a set of carefully designed benchmarks and user studies. The measured results and positive user feedback demonstrate that our implementation has negligible overhead for various applications to run in Shared Mode, is favored by mobile users, and provides robust protection on sensitive data and applications.

9. ACKNOWLEDGMENTS

The authors would like to thank their colleagues for fruitful discussions and valuable suggestions. The Rice team was supported in part by NSF awards IIS/HCC 0803556, CNS/NeTS 0721894, and CRI/IAD 0751153. We also thank our reviewers and our Shepherd, Professor Eyal de Lara, whose comments and shepherding helped improve this paper. Last but not least, we thank the participants of our user studies for their participation and valuable comments.

10. REFERENCES

[1] A. Brush and K. Inkpen, "Yours, Mine and Ours? Sharing and Use of Technology in Domestic Environments," in *Proc. of Ubiquitous Computing 2007*, pp. 109-126, 2007.

[2] A. L. Chavan and D. Gorney, "The Dilemma of the Shared Mobile Phone---Culture Strain and Product Design in Emerging Economies," *ACM Interactions,* vol. 15, pp. 34-39, 2008.

[3] M. Hall, "Create a Windows CE Image That Boots to Kiosk Mode", *http://msdn.microsoft.com/en-us/libraryaa446914.aspx*.

[4] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas, "Enabling Context-aware and Privacy-Conscious User Data Sharing," in *Proc. of IEEE Int. Conf. Mobile Data Management*, 2004.

[5] G. C. Hunt and D. Brubacher, "Detours: Binary Interception of Win32 Functions," in *Proc. of Conf. USENIX Windows NT Symposium,* Seattle, WA, USA: USENIX Association, 1999.

[6] S. Jain, F. Shafique, V. Djeric, and A. Goel, "Application-level Isolation and Recovery with Solitude," in *Proc. of the 3rd ACM SIGOPS/EuroSys European Conf. on Computer Systems,* Glasgow, Scotland UK: ACM, 2008.

[7] P. H. Kamp and R. N. M. Watson, "Jails: Confining the Omnipotent Root," in *Proc. of the 2nd International SANE Conf.*, 2000.

[8] Z. Liang, V. N. Venkatakrishnan, and R. Sekar, "Isolated Program Execution: an Application Transparent Approach for Executing Untrusted Programs," in *Proc. of 19th Annual Computer Security Applications Conf.,* 2003.

[9] B. des Ligneris, "Virtualization of Linux Based Computers: the Linux-VServer Project," in *Proc. of 19th International Symposium on High Performance Computing Systems and Applications*, pp. 340-346, 2005.

[10] J. S. Olson, J. Grudin, and EricHorvitz, "A Study of Preferences for Sharing and Privacy," in *Proc. of Extended Abstracts on Human Factors in Computing Systems,* Portland, OR, USA: ACM, 2005.

[11] T. Pering, D. H. Nguyen, J. Light, and R. Want, "Face-to-Face Media Sharing Using Wireless Mobile Devices," in *Proc. of IEEE Int. Symp. Multimedia*: IEEE Computer Society, 2005.

[12] D. Price and A. Tucker, "Solaris Zones: Operating System Support for Consolidating Commercial Workloads," in *Proc. of the 18th USENIX Conf. on System Administration* Atlanta, GA: USENIX Association, 2004.

[13] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based Operating System Virtualization: a Scalable, High-Performance Alternative to Hypervisors," *ACM SIGOPS Operating Systems Review* vol. 41, pp. 275-287, 2007.

[14] A. Voida, R. E. Grinter, N. Ducheneaut, W. K. Edwards, and M. W. Newman, "Listening in: Practices Surrounding iTunes Music Sharing," in *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems,* Portland, OR, USA: ACM, 2005.

[15] Y. Yu, F. Guo, S. Nanda, L.-c. Lam, and T.-c. Chiueh, "A Feather-Weight Virtual Machine for Windows Applications," in *Proc. of the 2nd International Conf. on Virtual Execution Environments,* Ottawa, Ontario, Canada: ACM, 2006.

[16] Spb Software House, "Spb Kiosk Engine," *http://www.spbsoftwarehouse.com/products/kioskengine*.

[17] VMware Corporation, "VMware Mobile Virtualization Platform", *http://www.vmware.com/technology/mobile/*.