Volume 3, Issue 3, March 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Implementation of Xshare Sharing of Mobile Phone by Using System Architecture of Win32, System File API

K.G.S.Venkatesan
Associate Professor, Dept. of CSE
Bharath University,Chennai, India

Kishore
Department of C.S.E.
Bharath University,Chennai ,India

Mukthar Hussain
Department of C.S.E
Bharath University,Chennai, India

Abstract - Sharing phones may be a fairly common observe among teens, and roughly one-quarter (23%) of these World Health Organization don't own their own mobile phone share one with somebody else. a lot of of this phone sharing is with others within the family, as well as siblings and fogeys. In fact, teens that have a parent with a mobile phone area unit over double as seemingly as those whose parent doesn't (27% vs. 12%) to report that they share a phone with somebody else. during which the sharing information differs from one person associate degree other to a different} within the % system parole is employed to stop sharing however it's larger disadvantage if we would like to share an specific file and conceal different file this can't be wiped out existing system. to beat this drawback we have a tendency to purpose AN Xshare mechanism that has customised sharing choices and profiles during which we will share and conceal every and each files, folders and applications. The Xshare mechanism offers file level access mechanism with profile construct so every profile will have completely different file sharing mechanism.

Keywords: xShare, Symbian, eXecution-In-Place (XIP).

I. INTRODUCTION

Existing mobile phones give inadequate support for sharing, there's no access management on non-public knowledge and pay-per-use applications. The owner shares their phone, the recipient can have identical access rights because the owner. Mobile phones use a word to stop unauthorized access, nevertheless it's for the whole system, and thus, the access management is nothing or everything. The iPhone incorporates a restriction feature which will disable some inherent applications. It will not apply to third-party applications nor will it give access management for knowledge. Windows Mobile phones will boot into a less-known closet mode, in that solely bound applications will be run. However, it needs a resuscitate and doesn't give access management to knowledge.

To address such limitations, we have a tendency to gift the results from a whole analysis and development cycle of xShare, a package answer for friendly, efficient, and secure phone sharing. xShare permits the owner to apace specify what they require to share and place the phone into a restricted mode wherever solely specifically shared applications. First, we have a tendency to gift a radical understanding of phone sharing obtained from 2 user studies, as well as interviews with existing smart phone users from four countries and long user studies with teenagers from the USA. Our user studies show that the bulk of existing and potential users share their mobile phones [4].

Our user studies give insight into why, where, with whom, and for what applications mobile users share their phones. Based on these findings, we have a tendency to propose the look of xShare supported file-level access management. xShare provides 2 modes of operation: traditional Mode and Shared Mode. once switch-ing from traditional Mode to Shared Mode, the owner specifies that files and applications to share, or a sharing policy. xShare creates a virtual atmosphere for Shared Mode from the sharing policy.

II. Existing System

Program executed when there is a pairing between the intended recipients mobile and received files. Achieved by using Blue tooth enabled Mobile Phones paired with a Blue tooth enabled computer. The program is designed in such a way that the transmitted file can be viewed by the recipient only when it is paired with the Recipient's mobile phone.

III. Proposed System

Data gets flushed automatically after viewed to ensure that it does not reside in the system. Even if some hacker is suppose to capture the files, they will not be in a position to view the files without the pairing of the recipient's mobile phone thus to ensure enhanced security.

IV. MODULES

A. Authentication:

Login module has two processes.

Administration:

The admin can access all the process in the phone and their login.

User :
The user has only limited access to the phone. User can login if and only If admin accept his registration.

B. File Sharing :
To protect the shared files from unwanted modification, share 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 redirect all the changes to private folder.

C. Application Sharing :
Many application 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 opened by the user in that application. As most mobile OSes, including windows mobile, Symbian, and iphone OS, store application files, and data files in different folders, xShare simply allow access to all the files in the same folder as the corresponding executable file [8]. Some application may access nonstorage 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.

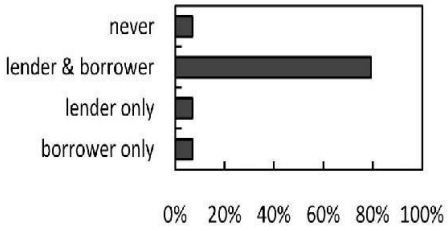


Fig. 1. Phone sharing statistics.

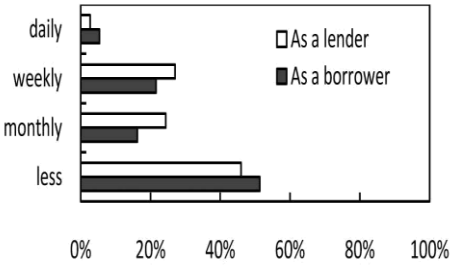


Fig 2: Frequency of phone sharing manong our

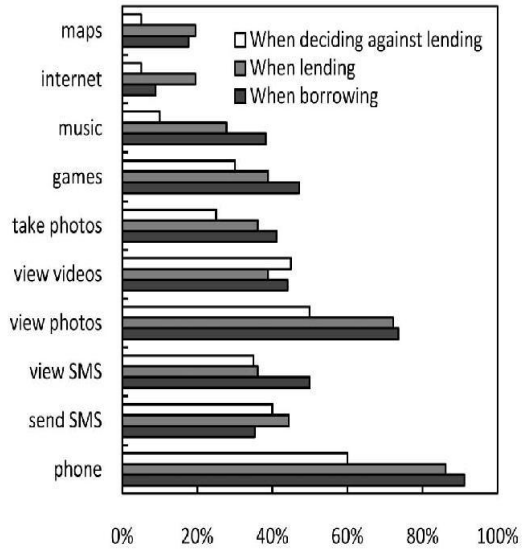


Fig. 3. Applications involved in phone sharing

D. Profile Management :
There are Different profiles which consist of different sharing mechanism with different access category. We have to switch profiles with minimum latency without affecting the access criteria for each profile.

V. 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.

A. 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. 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 [6]. 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. File- based access control. A key design decision we made is to base xShare's access control at the file level. The rationale behind this design decision is for generality. 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. 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 Fig. 6. Therefore, a file-based access control allows an application-independent solution. 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 smart phone 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 IV (A), 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.

B. 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 applications selection, and resource allowance specification. 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 [14]. 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.

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 nonshared data items and removing the icons and links of the nonshared 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 that 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.

C. 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 nonshared applications and data [16]. In runtime, xShare checks the list of the files to determine whether a given file is in the list and if access to it should be granted. We next present our design decisions for two 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 modifying existing applications.

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 opened by the user in that application. As most mobile OSes, including Windows Mobile, Symbian, and iPhone OS, store application files, and data files in different folders, xShare simply allows access to all the files in the same folder as the corresponding executable file. Some applications may access nonstorage peripheral devices, such as the camera and the microphone. These nonstorage peripherals do not contain any private information; therefore, xShare grants access to them in Shared Mode.

B. File Sharing *To* protect the shared files from unwanted modification, share 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 redirect all the changes to private folder.

C.Application Sharing

Many application 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 opened by the user in that application. As most mobile OSes, including windows mobile, Symbian, and iphone OS, store application files, and data files in different folders, xShare simply allow access to all the files in the same folder as the corresponding executable file [8]. Some application may access nonstorage peripheral devices, such as the camera and the microphone. These nonstorage peripherals do not contain any private information; therefore, xShare grants access to them in shared mode. Given the sharing policy, xShare creates a virtual environ-ment for Shared Mode. The virtual environment confines access to the shared data and applications through name-space 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 :

xShare separates changes made to files and settings in Shared Mode and ensures that 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 [13]. 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 redirect all the changes to the private folder.

Hiding Nonshared Files.Through namespace virtualization, xShare hides nonshared resources from shared applications 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.Resource Accounting As indicated by our user studies, phone owners are concerned with the overusage of exhaustible system

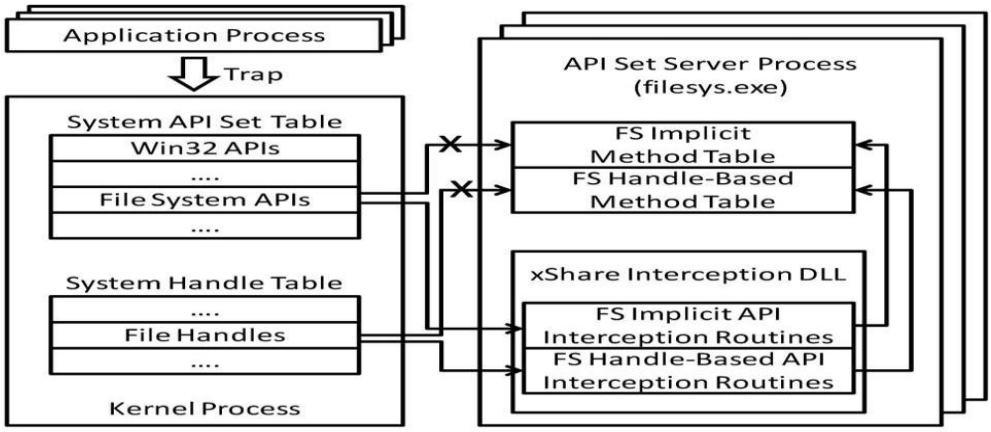


Fig 4: System API interception on Windows Mobile.

Vi. 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 [17], 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.

A. 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 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() [14]. 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 implicit 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 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.

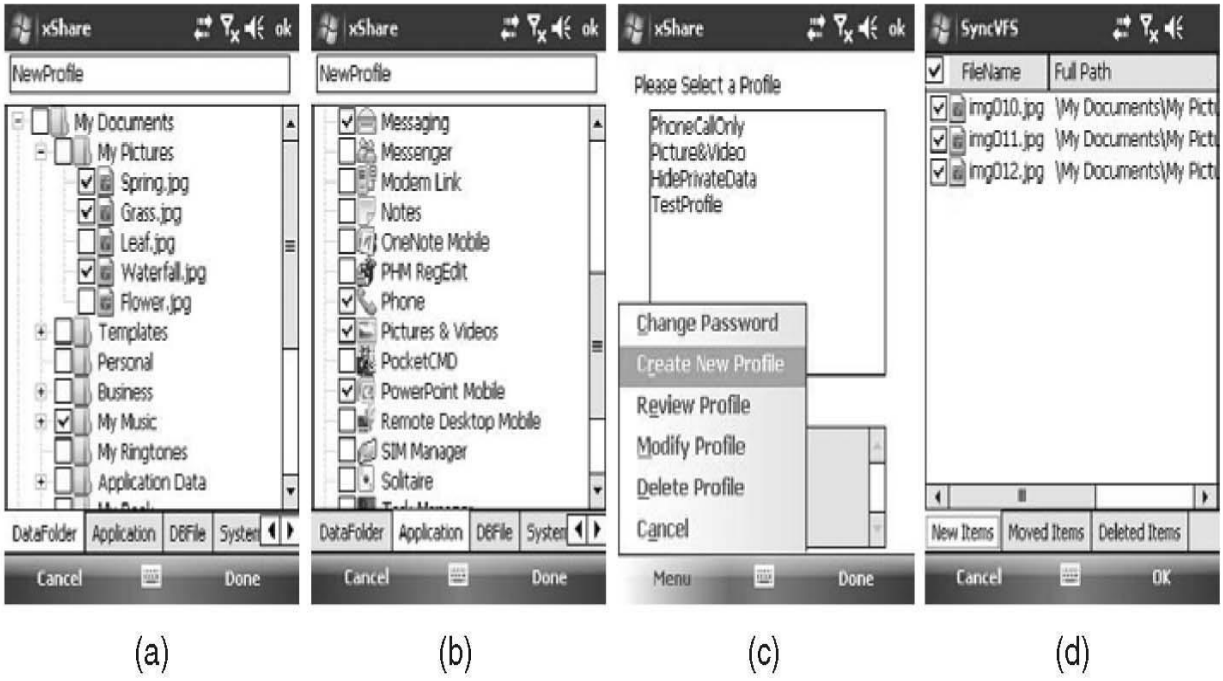


Fig 5: User interface for owner. (a) Select files or folders to share, (b) select applications to share, (c) manage profiles, and (d) review the files changed in Shared Mode.

*B.**Namespace Virtualization** :*

We implement xShare namespace virtualization on Windows Mobile by system API interception. We intercept 28 APIs in total. File System Virtualization According to the design described in Section III (C), in Shared Mode, we provide a virtual file system to track and separate changes made in Shared Mode, hide nonshared files, and provide a consistent appearance to the borrower. 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 inter-mediate 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 [11]. For preexisting 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 nonshared files. To hide nonshared files, the interception routine for CreateFile() returns ERROR_FILE_ NOT_FOUND when an application tries to open a nonshared file.

VII. EVALUATION OF XSHARE

We evaluate our xShare implementation by measurement, usability study, and field evaluation. For measurement and

usability study, we use the HTC Wizard phone running Windows Mobile 6.1 Professional with CE OS 5.2.

A.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.

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 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.

Mode Switching Latency

As our user studies in Section 2 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 [9]. We then specify the sharing policy to allow access to Pictures and Videos, Solitaire, Phone, and Messaging applications along with 10 photos. We repeat the experiment 10 times. Results.. 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 back-ground without introducing any noticeable latency.

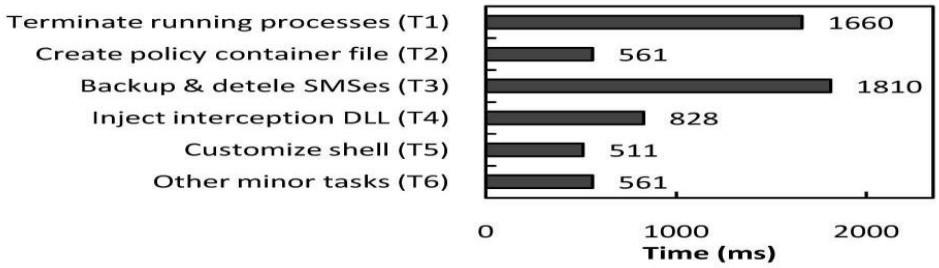
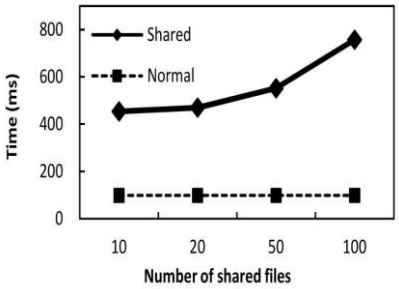


Fig 6: Mode switching latency to specify policy.

B. Usability Evaluation :

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, e.g., by advertising the study through friends and coworkers. They all used a PC daily and five 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 afterward for least 5 minutes, before being asked to specify policies A and B for 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 [16]. The average time required by the participants separately, for prior Windows Mobile users and nonusers. Our results indicate that xShare has a very quick learning curve; our participants required only 20 seconds, on average, to specify the last policy, and non-Windows Mobile users required only 17 percent more time compared to prior Windows Mobile users.



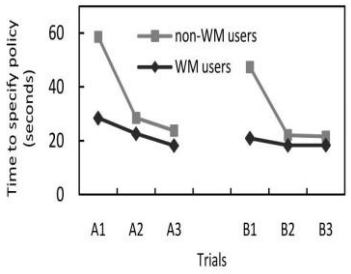


Fig 7: Time costs for listing a folder. **Fig 8: Average time to startxShare and specify a policy.**

VIII. Conclusion

In this paper, we have a tendency to gift an entire analysis and development cycle of xShare for friendly, efficient, and secure sharing of mobile phones. we discover that phone sharing is extremely fashionable and involves a good vary of applications, reasons, social settings, and relationships. however privacy remains a serious concern that forestalls users from sharing their phone, and existing systems offer inadequate privacy protection for sharing and nomical and secure resolution for impromptu sharing of mobile phones.

Based on these findings, we have a tendency to gift xShare, a software package resolution to support friendly, efficient, and secure phone sharing on existing systems. xShare permits phone house owners to simply produce impromptu sharing policies to make your mind up that files and applications to share and management what proportion resource a receiver will use. victimization the sharing policies, xShare creates a virtual setting, referred to as Shared Mode, wherever solely the expressly shared files and applications area unit visible [19]. Further, xShare permits the owner to manage knowledge files and settings created or changed by the receiver. We show that xShare will be complete for Windows Mobile with none read-only memory image changes or additional support from the software. We valuate our Windows-Mobile-based implementation through a collection of rigorously designed benchmarks and user studies. The measured results and positive user feedback demonstrate that our implementation has negligible overhead, provides strong protection on sensitive knowledge and applications, and is in a position to attain its style goals.

Acknowledgment

The author would like to thank the Vice Chancellor, Dean-Engineering, Director, Secretary, Correspondent, Principal, HOD , Dean CSE Dr.A.Kumaravel of Bharath University, Chennai for their motivation and constant encouragement. The author would like to specially thank Dr. V.Khanaa for his guidance and for critical review of this manuscript and for his valuable input and fruitful discussions in completing the work and the Faculty Members of Department of Computer Science & Engineering. Also, he takes privilege in extending gratitude to his parents and family members who rendered their support throughout this Research work.

References :

[1] A. Brush and K. Inkpen, "*Yours, Mine and Ours? Sharing and Use of Technology in Domestic Environments,"* Proc. Int'l Conf. Ubiquitous Computing, 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, 2009.

[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,"* Proc. IEEE Int'l Conf. Mobile Data Management, 2004.

[5] G.C. Hunt and D. Brubacher, *"Detours: Binary Interception of Win32 Functions,"* Proc. Conf. USENIX Windows NT Symp., 1999.

[6] S. Jain, F. Shafique, V. Djeric, and A. Goel, *"Application-Level Isolation and Recovery with Solitude,"* Proc. Third ACM SIGOPS/ EuroSys European Conf. Computer Systems, 2008.

[7] P.H. Kamp and R.N.M. Watson, *"Jails: Confining the Omnipotent Root,"* Proc. Second Int'l SANE Conf., 2000.

[8] A.K. Karlson, A.J.B. Brush, and S. Schechter, *"Can I Borrow Your Phone?: Understanding Concerns When Sharing Mobile Phones,"* Proc. SIGCHI, 2009.

[9] B. Lampson, *"Computer Security in the Real World,"* Proc. Ann. Computer Security Applications Conf., 2000.

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

[11] B. des Ligneris, "*Virtualization of Linux Based Computers: The Linux-VServer Project*," Proc. 19th Int'l Symp. High Performance Computing Systems and Applications, pp. 340-346, 2005.

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

[13] T. Pering, D.H. Nguyen, J. Light, and R. Want, *"Face-to-Face Media Sharing Using Wireless Mobile Devices,"* Proc. IEEE Int'l Symp. Multimedia, 2005.

[14] D. Price and A. Tucker, "Solaris Zones: *"Operating System Support for Consolidating Commercial Workloads,"* Proc. 18th USENIX Conf. System Administration, 2004.

[15] S. Soltesz, H. Po¨tzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, *"Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors,"* ACM SIGOPS Operating Systems Rev., vol. 41, pp. 275-287, 2007.

[16] A. Voida, R.E. Grinter, N. Ducheneaut, W.K. Edwards, and M.W. Newman, "*Listening In: Practices Surrounding iTunes Music Sharing,"* Proc. SIGCHI, 2005.

[17] Y. Yu, F. Guo, S. Nanda, L.-c. Lam, and T.-c. Chiueh, *"A Feather-Weight Virtual Machine for Windows*

Applications," Proc. Second Int'l Conf. Virtual Execution Environments, 2006.
[18] Likert Scale, http://en.wikipedia.org/wiki/Likert_scale, 2010.
[19] Parents Using Smartphones to Entertain Bored Kids, CNN Living with Technology, http://www.cnn.com/2010/TECH/04/26/ smartphones.kids/index.html?hpt= Mid, 2010.
[20] SPB Software House, "*SPB Kiosk Engine,"* http://www.spb softwarehouse.com/products/kioskengine.

ABOUT THE AUTHOR

K.G.S.Venkatesan received his B.Tech degree in Computer Science & Engineering from JNT University, Hyderabad and received his M.Tech degree in Computer Science & Engineering from Bharath University. He is currently pursuing his Ph.D in Computer Science & Engineering at Bharath University, Chennai. He has 10 years of Teaching experience and has guided many B.Tech and M.Tech projects. He is having Membership in Indian Society of Technical Education (MISTE).He attended HIGH IMPACT Teaching Skills Programme conducted by WIPRO MXLA (Mission 10X Learning Approach).

K.M. Mukthar Hussain is currently pursuing his B.Tech in Computer Science & Engineering from Bharath University,Chennai. He has secured Second Rank Holder in Schools and many Cultural activities participation certificates in the school

.

