Secure Deletion Myths, Issues, and Solutions[*]

Nikolai Joukov, Harry Papaxenopoulos, and Erez Zadok

Stony Brook University
Computer Science Department
Stony Brook, NY 11794-4400
{kolya,harry,ezk}@cs.sunysb.edu

ABSTRACT

This paper has three goals. (1) We try to debunk several held misconceptions about secure deletion: that encryption is an ideal solution for everybody, that existing data-overwriting tools work well, and that securely deleted files must be overwritten many times. (2) We discuss new and important issues that are often neglected: secure deletion consistency in case of power failures, handling versioning and journalling file systems, and meta-data overwriting. (3) We present two solutions for on-demand secure deletion. First, we have created a highly portable and flexible system that performs only the minimal amount of work in kernel mode. Second, we present two in-kernel solutions in the form of Ext3 file system patches that can perform comprehensive data and meta-data overwriting. We evaluated our proposed solutions and discuss the trade-offs involved.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Security and Privacy Protection*; D.4.3 [**Software**]: Operating Systems—*File Systems Management*

General Terms

Security, Design, Management

Keywords

Security, File systems, Unintended data recovery, Secure deletion

1. INTRODUCTION AND PRIOR WORK

For many years, computer systems have continued to mislead end users into thinking that when they delete a file (or purge the TrashBin), that those files are permanently deleted. In recent years, theft and loss of laptops and portable storage media have resulted in confidential and even top-secret data files reaching the wrong hands. In most cases users and even security experts do not know that their disk drives contain confidential information [5]. The problem is exacerbated by the fact that even those who know that deleted files can be recovered may forget that some storage device had been used to store confidential data in the past. Therefore, unintended recovery of data is mostly a psychological problem [21]. Generally, recovery of non-overwritten data is trivial [7].

There are two ways to perform secure deletion: (1) overwrite the data [9] or destroy the physical media; and (2) encrypt the data or whole media and use one of the first methods to securely delete the key [3]. Without the key the data cannot be decrypted. As with most security solutions [10], secure deletion does not come for free. Depending on the particular situation, either the first or the second method might be more preferable.

Secure deletion via encryption usually adds significant overheads and requires non-trivial efforts for the proper key management. However, new governmental regulations (e.g., Sarbanes-Oxley [22]) assume data encryption for publicly-traded companies, health and government organizations. Therefore, for these companies and organizations, secure deletion via encryption is much cheaper because the encryption-related overheads are paid in any case. As a result, recent secure deletion research is mostly aimed at secure deletion via encryption [18, 19, 28]. Unfortunately, some organizations, most individuals, and small businesses do not use data encryption and, therefore, the encryption-based secure deletion is too costly for them. At the same time, existing data-overwriting tools have many reliability, performance, and other problems. This has resulted in the unfortunate situation that today, deleted files' data remains in the clear, with no secure, efficient, and easy-to-use solutions available,

In this paper we have concentrated on the data-overwriting secure deletion and have researched two data-overwriting solutions. (1) We used FoSgen [13] to add interception functionality of data deletion events to any existing file system's sources. We perform data overwriting from user mode. This automates the secure-deletion functionality inside the OS, but provides only basic secure-deletion for files' data. This solution requires minimal OS modifications and is mostly implemented in user space. (2) We have developed two secure-deletion extensions for Linux's Ext3 journalling file system. Our design uses the journal to record one or more data-overwriting actions, using the NISPOM [23] and NIST [8] standards; we support meta-data overwriting; we designed features to trade-off efficiency for security and convenience; and we resume overwriting in the case of a system crash.

The rest of this paper is organized as follows. We describe secure deletion myths in Section 2 and related issues in Section 3. We present our solutions in Sections 4 and 5. We evaluate our solutions in Section 6. We conclude in Section 7.

[*]This work was partially made possible by NSF CAREER EIA-0133589 and CCR-0310493 awards and HP/Intel gifts numbers 87128 and 88415.1.

2. MYTHS

Even security experts hold several misconceptions which we criticize in this position paper.

2.1 Encryption Solves the Problem

Many researchers promote encryption as a solution to secure deletion. The advantage of encryption is that it protects live data as well as deleted data. Also, it simplifies secure deletion from archived copies: archives cannot be decrypted without a key that is discarded. Unfortunately, encryption also carries with it additional, significantly increased costs: (1) managing keys for many files or data blocks is cumbersome and costly; (2) encryption adds CPU overheads [25] for most file system operations; and (3) keys could be lost or broken (especially for long-term archival storage). Worse, a compromised key may allow recovery of both existing as well as deleted files. Using per-file keys for the purpose of secure deletion adds even more overhead and key management costs [18].

To comply with new security regulations, public companies and many organizations do encrypt their data. This means that they are paying the encryption overheads and key management costs anyway. Therefore, these companies and organizations can use secure deletion via encryption with a small extra cost. However, most individual users and small businesses do not encrypt their data and, therefore, do not use secure deletion by discarding the key.

In addition, key revocation is problematic even for existing (not deleted) files because it sometimes requires re-encrypting the data. For example, if one creates a file encrypted with the new key and then deletes the old file which was encrypted with the old (compromised) key, then the old key can still be used to decrypt the data: the old encrypted file can be easily recovered if not overwritten. Even if one writes the file encrypted with the new key directly over the old file, there is still a chance that the old data can be recovered using hardware tools. This suggests that for optimal security, encryption must be combined with data overwriting [19].

2.2 Data Overwriting Works Well

There is a plethora of existing user-level tools available today, many of which are commercial and incorrectly tout strong security [2, 6]. In practice, they are inconvenient and unreliable. Also, inconvenience usually results in lower security as users refrain from performing inconvenient operations. Most tools either require users to overwrite whole disks, whole partitions [11], or all the free space on a file system (by creating a new file that is as big as the amount of the free space on a file system). Such procedures are lengthy and do not guarantee data overwriting [2, 6]. We will further describe secure deletion problems via data overwriting in Section 3.

2.3 Overwrite Data Many Times

Years ago it was shown that there is a chance that even after the data is overwritten, it can potentially be recovered [15]. Many experts believe that unless one can overwrite the data numerous times, that it is not worth to overwrite it even once [9]. Nothing could be further from the truth. Even the government's own NIST and NISPOM standards for secure deletion of top-secret files call for overwriting no more than three-times [8, 23]; and, for most users, a single overwrite will suffice and greatly enhance security. In particular, one overwrite will make any software-based data recovery impossible. Thus, hackers who gain privileged access to the system will not be able to recover files deleted from its hard disks. To date, *no* commercial services are available to recover data that was overwritten even just once [24].

3. ISSUES WITH SECURE DELETION VIA OVERWRITING

While experts continue to debate how many times a file should be overwritten to prevent, say, the military from recovering one's files, several practical issues have been largely ignored.

3.1 Automation

Most secure-deletion systems work on whole drives, partitions, or file systems' free space [2, 6]. In these cases secure deletion takes hours if not days even for small hard drives. Reliable, automatic, efficient, per-file-delete data overwriting requires instrumentation in file systems [13]. File-system–specific solutions are complicated and are not portable even between versions of the same file system [1].

3.2 Reliability

Convenient per-file overwriting tools target file systems that overwrite previous data while writing new data (e.g., Ext2 and VFAT). Overwriting large files or overwriting files multiple times can add significant overheads. For performance reasons many existing tools use asynchronous data overwriting which can leave data not overwritten in case of a power failure. Most modern file systems use journalling for reliability reasons. Journalling can be beneficial in ensuring that overwritten data is committed to the journal and can be replayed even after a power failure or system crash.

3.3 Meta-Data Overwriting

Most secure-deletion systems concern themselves with the file's data, neglecting the file's meta-data: the file's name, owner, group, access and modification times, size, and other information available in the file's inode. All these additional pieces of information, if left alone, can provide attackers important information about the original file's contents. Alas, secure-deletion of meta-data is trickier, because meta-data tends to be spread out throughout a file system; moreover, individual disk blocks contain meta-data for multiple files, and hence must be overwritten partially and carefully.

3.4 Versioning Systems

With the advent of large and inexpensive disks, versioning file systems and versioning storage systems have become more popular [16]. This means that when a piece of data is overwritten, it is *not* overwritten in place due to copy-on-write semantics. Modern versioning file systems, therefore, must support secure-deletion functionality that is capable of deleting a file's data, meta-data, and all historical versions thereof. Only encryption-based solutions are currently available [19].

3.5 Persistent Caches

Probably the most difficult problem to solve is that, due to an ever-widening performance gaps between CPUs and disk I/O, vendors have increased their use of caching using non-volatile RAM in storage systems. These caches not just defer writing to the actual physical media, but they also *prevent* multiple overwrites to the same storage location, so as to coalesce costly repeated writes to the same file into one bulk write operation. This performance feature interferes with the desire to overwrite data multiple times. Therefore, one needs to disable write-caching for hard disks that may require multiple data overwrites.

In general, the solution to this issue requires changes to lower level, well established disk APIs, as well as by storage system vendors. Currently, most SCSI disks and some ATA disks support hardware-based overwriting. Unfortunately, the API allows only whole drive overwriting [11].

4. SOLUTIONS

In this section we describe our solutions that allow consistent information overwriting: (1) a portable file system extension for versatile automatic data overwriting via renaming; and (2) our two Ext3 patches.

4.1 Secure Deletion via Renaming

User mode tools are attractive because they are portable and easy to develop. `shred` is a standard utility available on many popular systems. It performs synchronous file overwriting and must be invoked manually [20]. To overwrite deleted files automatically and immediately when they are deleted, file system support is needed. FoSgen is our tool to add file system extensions to any file system on a number of OSs automatically [13]. It parses extensions written using the FiST language [27] and applies them directly to the file systems' code as shown in Figure 1. If the source code for a file system is not available, FoSgen can add extensions to stackable file systems. A stackable file system can be inserted between the Virtual File System (VFS) and a lower file system to intercept or modify file system requests [26]. Our portable and consistent data-overwriting system consists of two components: (1) a file system extension which can be applied to any file system and (2) a user-mode `shred` tool.

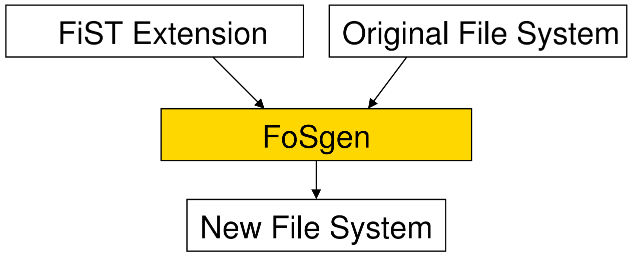

Figure 1: FoSgen operation.

Our file system extension intercepts file system events that require overwriting and move the corresponding files into a special per-file-system directory called `ForSecureDeletion`. The extension can either do it for all the files or only for these marked with a special file attribute. There are two file system operations that may require overwriting of the old data: `unlink` (called to delete a file) and `truncate` (called to change the file size). Our extension first moves files to the `ForSecureDeletion` directory. Then, for `set_attr` operations that truncate the file, our extension creates a new file with the name of the original file and may need to copy a portion of the original file to the new one (in case of a partial file truncation). Partial file truncations may require copying substantial amounts of data but, fortunately, such operations are rare.

The `shred` tool, invoked either manually or by a `cron` daemon, overwrites the files moved to the `ForSecureDeletion` directory. The frequency of `shred` invocations can be used to balance security and convenience. On one hand, frequent `shred` invocations reduce the time window when data is still left on the disk not overwritten but adds overheads for overwriting. On the other hand, nightly overwrites of data can significantly improve performance even compared with file systems that do not support secure deletion. Indeed, ordinary file `unlink` operations are significantly slower than file `rename` operations. There even exist patches to several Linux file systems that perform unlink operations asynchronously [4]. Note that users can have many different secure-deletion policies. For example, they can invoke `shred` manually on the files stored in the special directory right after the file is moved there. Our design has several benefits:

- The overwrite operations are performed automatically. Users do not need to invoke `shred` manually on every file they want to delete.
- Our file system FiST extension and the `shred` tool are portable and can be used on most Linux and FreeBSD file systems without any modifications.
- The system contains only the minimal amount of kernel code necessary to detect deletion events reliably. Therefore, the system is easier to maintain.
- All overwrite operations are performed consistently. Even if a file is not completely overwritten before a power failure or a system crash, it will be overwritten after the system is rebooted.
- The system is flexible and offers a number of ways to balance security with convenience and performance. Users can use a `cron` daemon to overwrite the data as many times as necessary when the system is idle. This can even improve overall performance due to faster delete operations. If, however, the data has to be overwritten immediately (e.g., before giving away a used data drive) then one can call `shred` manually. Also, one may raise the overwriting frequency and reduce the time data is kept on disk after deletion.
- A beneficial side effect of our FiST extension is the presence of a trash-bin functionality. Files that are not overwritten right away can be recovered if deleted by mistake.

One may argue that this design opens a window of vulnerability before the data is overwritten. However, this window exists for any data-overwriting system that overwrites the data asynchronously (for efficiency). In our system, the size of the window can be configured to balance security with convenience and performance. In fact, we believe that a hybrid secure deletion and trash-bin functionality approach has another significant advantage: it is not difficult to add reliable and convenient secure deletion functionality for file systems that already support trash-bin functionality.

4.2 Ext3 Secure Deletion Enhancements

We have created two secure-deletion patches for the Ext3 file system. The first patch is small and simple but allows only single data overwrite. It is designed for the majority of users. Although somewhat slower, the second patch supports comprehensive overwriting policies including multiple overwrites and random character overwrites. It is designed to protect much more sensitive information, as it is tailored to secure delete with the best possible protection.

4.2.1 Single Overwrite Ext3 Patch

Ext3's `ext3_free_blocks` method is called when some previously used blocks need to be deallocated: upon file deletion and truncation. These are the cases when we need to overwrite the old data. Important for us, Ext3 provides atomicity for file delete and truncate operations. First, it encapsulates all related sub-operations into a single journal transaction. Second, it adds information about a file being deleted to the *orphan list*. This is necessary to handle deletion of large files atomically, whose deletion transaction may not fit in the journal. The orphan list is, in fact, a special journal for file-deletion transactions.

We have created a patch for the Linux Ext3 file system that encapsulates data-block overwrite and data-block release operations into a single transaction. The fact that both operations are encapsulated within one transaction guarantees overwriting consistency. Even if the file-overwriting process is interrupted, it will be completed after the system is restarted and the journal is replayed. We have submitted a simple version of the patch to the core Linux file systems developers [12]. Our patch consistently overwrites data once in all three of Ext3's journalling modes: writeback, ordered (default), and data. Therefore, it is impossible to recover overwritten data at least using the software-recovery methods. Note that in the data-journalling mode of Ext3 (which is rarely used) one may want also to wipe out the journal by writing more data than the journal's size to any file. This is simple and fast because the journal is usually small and in most cases gets reused as a cyclic buffer relatively quickly.

Ext3 (and Ext2) already support special attributes to mark files that require secure deletion, using

```
chattr +s filename
```

However, Ext3 does not support secure deletion functionality itself. Our patch can either overwrite all files or only these marked with the secure deletion attribute.

4.2.2 Comprehensive Ext3 Patch

The second Ext3 patch that we created (1) supports multiple overwrites, and (2) it securely deletes a file's meta-data (user and group IDs and access, modification, and creation times) and its directory entry (name) in addition to its data. Similar to the first patch, it works in all three of Ext3's journalling modes and can either overwrite all files or only those marked with the secure deletion file attribute.

Ext3's journalling facility has the notion of compacting transactions. For performance reasons, Ext3's journal merges write requests to the same block if these requests are within the same transaction. In other words, under a single transaction, if one changes some buffer-heads multiple times, only the last change will be committed to disk. Therefore, our patch has to create a *separate* transaction for every overwrite. This means that we cannot overwrite from within the same Ext3 function as we did for the simple patch because it is already called within a file delete transaction. Fortunately, Ext3 supports a special journalling-like mechanism for deletion operations by using the orphan list. Therefore, even if it is performed within several separate transactions, the deletion operation is still performed atomically. The orphan list is permanently stored on the storage media. Any members of that list are deleted at the beginning of the Ext3 mount process. Therefore, our secure deletion operations are restarted in case of a power failure. Our Ext3 patch is indirectly called through the `unlink` and `setattr` operations that may delete or truncate a file, respectively. Inside of these operations we add a file to the orphan list and create separate transactions for every round of overwrites. After that we allow Ext3 to add its ordinary transaction to handle the rest of the `unlink` or `setattr` operations and remove the file from the orphan list at the end.

In addition to data overwriting, our comprehensive patch can also overwrite file names and other meta-data. Despite the fact that most users perceive file names as yet another meta-data, they are part of the namespace, stored differently, and require a separate overwriting implementation. Also, our comprehensive patch supports overwriting with arbitrary patterns that may include random characters. One can separately choose the overwriting policies for each of the data, meta-data and directory entry (name) of a file. Patterns can consist of printable characters, hexadecimal numbers, and random numbers. For example, a user can choose to overwrite the file with the characters '\FRs'. This pattern indicates that the file is to be first overwritten with the hex number 0xF, followed by a random character, and the literal character 's'. There is a limit on the length of the mount options and the syntax of our overwriting policies parameters is designed to minimize the length of the corresponding string. For example, the following mount command specifies that file data (denoted as 'D') should be overwritten with '0's three times, and file names (denoted as 'N') should be overwritten with random characters once:

```
mount -t ext3 \
    -o secdel="D:3:\\0;N:1:R" /dev/sda1 /mnt
```

We confirmed that out patch indeed overwrites files for the specified number of times and does not merge overwrites together by tracing write requests at the device driver level.

5. IMPLEMENTATION

Table 1 summarizes the complexity of our proposed solutions in terms of the number of lines of kernel C code.

Solution	No. of lines of C code
Move on delete	92
Ext3 basic	68
Ext3 comprehensive	609

Table 1: Number of kernel lines of C code for our proposed solutions (The move on delete solution is written in C-based FiST language).

6. EVALUATION

We evaluated our system on a P4 1.7GHz machine with 1GB of memory. Its system and test disks were 30GB 7200 RPM Western Digital Caviar IDE drives and were formatted with Ext3. We remounted the test file systems before every benchmark run to purge file system caches. We ran each test at least ten times and used the Student-t distribution to compute the 95% confidence intervals for the mean elapsed, system, user, and wait times. Wait time is the elapsed time less CPU time used and consists mostly of I/O, but process scheduling can also affect it. In each case, the half-widths of the confidence intervals were less than 5% of the mean. The test machine was running a Fedora Core 4 Linux distribution with a vanilla 2.6.15 kernel.

We evaluated and compared the following three configurations: vanilla Ext3 (EXT3); Ext3 with a FiST extension to move files to a special folder on delete (MOVE); and Ext3 3 versions of instrumented with secure deletion patches. PATCH-BASIC is Ext3 instrumented with our basic patch for one data overwrite. PATCH-COMPR is Ext3 instrumented with our comprehensive patch that overwrites file names, meta-data, and data twice. PATCH-COMPR-R is the same configuration but with an overwriting pattern composed of randomly generated characters.

6.1 CPU-Bound Workload

First, we evaluated our secure deletion systems under a compile workload—a CPU-intensive workload that is similar to the workloads generated during normal user activities (i.e., more CPU activity than file system I/O activity). We compiled the Am-utils version 6.1.1 [17]. Am-utils contains over 60,000 lines of C code in 430 files. The build process begins by running several hundred

Solution	Simplicity	Portability	Atomicity	Automation	Meta-data
User-mode	+++	+++	+	+	no
Move on delete	++	++	++	++	no
Ext3 basic	++	+	+++	+++	no
Ext3 comprehensive	+	+	+++	+++	yes

Table 2: Comparison of traditional user-mode solutions, our move-on-delete solution, and our two Ext3 patches. More '+' symbols means better. The comprehensive Ext3 patch is the only solution that can overwrite meta-data.

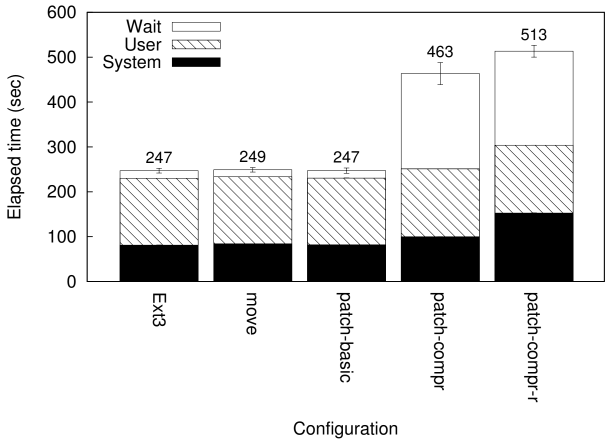

Figure 2: Am-utils benchmark times.

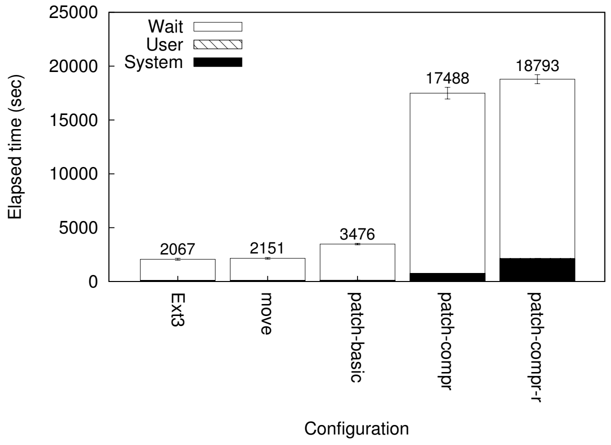

Figure 3: Postmark benchmark times.

small configuration tests to detect system features. It then builds a shared library, ten binaries, four scripts, and documentation: a total of 152 new files and 19 new directories. Although the Am-utils compile is CPU intensive, it contains a fair mix of file system operations. The Am-utils build process uses 25% writes, 22% lseek operations, 20.5% reads, 10% open operations, 10% close operations, and the remaining operations (12.5%) are a mix of readdir, lookup, etc. Most important for us, the build process performs 4,696 file deletions.

Figure 2 shows the measured build times. Both MOVE and PATCH-BASIC add less than 1% elapsed time overheads over the EXT3 configuration. PATCH-COMPR adds 87% elapsed time overheads mostly due to synchronous journal and data writes. Generation of random characters adds extra 37% overhead to the system time.

6.2 I/O-Bound Workload

We evaluated our system using an I/O-intensive workload generator. Postmark [14] simulates the operation of electronic mail servers. It performs a series of file appends, reads, creations, and deletions. We configured Postmark to create 20,000 files, between 512–1M bytes, and perform 200,000 transactions. We selected the create/delete and read/append operation ratios with equal probability. In this test, Postmark's final phase is to delete 20,000 such files (totaling 10–20GB of data).

Figure 3 shows the benchmark results for our three test configurations. The MOVE configuration that moves data for overwriting into a special directory adds less than 4% overhead. The PATCH-BASIC configuration which overwrites the data once adds 61% elapsed time overheads, mostly in I/O time. PATCH-COMPR runs 8.5 times slower relative to Ext3. This is because of the synchronous journal and data writes. Postmark represents the worst-case workload for secure deletion tools because it has numerous deletion operations (every fourth operation is a truncation) combined with other I/O-intensive operations.

7. CONCLUSIONS

Erasure of data from the storage, upon file delete, is consistent with users' perception of what a delete operation should do. We have discussed several common myths about secure deletion of data. We have shown that (1) existing secure deletion tools are inconvenient for most file systems (including journalling file systems) and have security problems; (2) encryption alone does not solve the secure deletion problem and should be combined with data overwriting; and (3) overwriting data many times is unnecessary: even a single data overwrite can significantly improve security, and three overwrites are sufficient even by the highest government standards.

We have designed a portable data-overwriting system which uses an existing user mode tool called `shred`, as well as a file-system–independent extension. Our data-overwriting system overwrites data automatically, is portable and simple, consistently overwrites even in case of power loss or system failures, and allows users to balance security, convenience, and efficiency. In case of delayed overwriting, our system can also provide a trash-bin like functionality to further enhance user convenience. We envision that existing file systems that already support trash-bin functionality for delayed deletion can be easily extended to support reliable and convenient secure deletion.

Also, we have designed two patches for the Ext3 journalling file system. The first one automatically overwrites data one time in a consistent way and thus provides protection against possible software-based deleted-data recovery tools. The second Ext3 patch supports overwriting of meta-data and comprehensive overwriting policies including those described in the NIST and NISPOM standards. We present our solutions and compare them to traditional user-mode solutions in Table 2.

We have demonstrated that under typical user workloads that are not I/O intensive and single overwrite policies, overheads are negligible. Also, we have shown that even under I/O-intensive and delete-intensive workloads, our data overwriting systems add acceptable overheads of 4–61% in the case of a common single overwrite policy.

Future Work

We plan to integrate our patch with the Ext3 journal which will allow us to decrease overheads. For example, it will allow us to overlap random number generation time with the wait time and decrease total elapsed time. Also, we plan to port FoSgen and FiST extension to a number of other OSs.

Acknowledgments

We would like to thank Theodore T'so for the project support and Charles P. Wright for his help with the paper's preparation.

More information about the project is available at `www.fsl.cs.sunysb.edu/project-sdfs.html`.

8. REFERENCES

[1] S. Bauer and N. B. Priyantha. Secure Data Deletion for Linux File Systems. In *Proceedings of the 10th Usenix Security Symposium*, pages 153–164, Washington, DC, August 2001. USENIX Association.

[2] H. Berghel and D. Hoelzer. Disk wiping by any other name. *Communications of the ACM*, 49(8):17–21, August 2006.

[3] D. Boneh and R. Lipton. A Revocable Backup System. In *Proceedings of the Sixth USENIX UNIX Security Symposium*, pages 91–96, San Jose, CA, July 1996. USENIX Association.

[4] M. Cao, T. Y. Tso, B. Pulavarty, S. Bhattacharya, A. Dilger, and A. Tomas. State of the art: Where we are with the ext3 filesystem. In *Proceedings of the Linux Symposium*, Ottawa, ON, Canada, July 2005.

[5] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum. Understanding Data Lifetime via Whole System Simulation. In *Proceedings of the 13th USENIX Security Conference*, pages 321–336, San Diego, CA, August 2004. USENIX Association.

[6] T. J. Fitzgerald. Deleted but Not Gone. *New York Times*, November 2005.

[7] Free Downloads Center. Linux Disk DoD. `www.freedownloadscenter.com/Best/linux-disk-dod.html`.

[8] T. Grance, M. Stevens, and M. Myers. *Guide to Selecting Information Security Products*, chapter 5.9: Media Sanitizing. National Institute of Standards and Technology (NIST), October 2003.

[9] P. Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Proceedings of the Sixth USENIX UNIX Security Symposium*, pages 77–90, San Jose, CA, July 1996. USENIX Association.

[10] R. Hasan, S. Myagmar, A. Lee, and W. Yurcik. Toward a Threat Model for Storage Systems. In *Proceedings of the First ACM Workshop on Storage Security and Survivability (StorageSS 2005)*, pages 94–102, FairFax, VA, November 2005. ACM.

[11] G. Hughes. CMRR Protocols for Disk Drive Secure Erase. Technical report, Center for Magnetic Recording Research, University of California, San Diego, October 2004. `http://cmrr.ucsd.edu/Hughes/CmrrSecureEraseProtocols.pdf`.

[12] N. Joukov. Patch: Secure Deletion Functionality in Ext3. `http://lwn.net/Articles/171924/`, February 2006.

[13] N. Joukov and E. Zadok. Adding Secure Deletion to Your Favorite File System. In *Proceedings of the third international IEEE Security In Storage Workshop (SISW 2005)*, San Francisco, CA, December 2005. IEEE Computer Society.

[14] J. Katcher. PostMark: A New Filesystem Benchmark. Technical Report TR3022, Network Appliance, 1997. `www.netapp.com/tech_library/3022.html`.

[15] I. Mayergoyz, C. Seprico, C. Krafft, and C. Tse. Magnetic Imaging on a Spin-Stand. *Journal of Applied Physics*, 87(9):6824–6826, May 2000.

[16] K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pages 115–128, San Francisco, CA, March/April 2004. USENIX Association.

[17] J. S. Pendry, N. Williams, and E. Zadok. *Am-utils User Manual*, 6.1b3 edition, July 2003. `www.am-utils.org`.

[18] R. Perlman. Secure Deletion of Data. In *Proceedings of the third international IEEE Security In Storage Workshop (SISW 2005)*, San Francisco, CA, December 2005. IEEE Computer Society.

[19] Z. N. J. Peterson, R. Burns, A. Stubblefield J. Herring, and A. D. Rubin. Secure Deletion for a Versioning File System. In *Proceedings of the Fourth USENIX Conference on File and Storage Technologies (FAST '05)*, pages 143–154, San Francisco, CA, December 2005. USENIX Association.

[20] C. Plumb. *shred(1) - delete a file securely, first overwriting it to hide its contents*. Free Software Foundation, August 2004.

[21] J. Rosenbaum. In Defence of the DELETE Key. *The Green Bag*, 3(4), Summer 2000. `www.greenbag.org/rosenbaum_deletekey.pdf`.

[22] P. Sarbanes and M. G. Oxley. *Sarbanes-Oxley Act of 2002*. U.S. Government Printing Office, July 2002.

[23] Defense Security Service. *National Industrial Security Program Operating Manual (NISPOM)*, chapter 8: Automated Information System Security. U.S. Government Printing Office, January 1995.

[24] C. H. Sobey. Recovering Unrecoverable Data: The Need for Drive-Independent Data Recovery. Technical report, Action Front Data Recovery Labs, Inc., April 2004.

[25] C. P. Wright, J. Dave, and E. Zadok. Cryptographic File Systems Performance: What You Don't Know Can Hurt You. In *Proceedings of the Second IEEE International Security In Storage Workshop (SISW 2003)*, pages 47–61, Washington, DC, October 2003. IEEE Computer Society.

[26] E. Zadok, R. Iyer, N. Joukov, G. Sivathanu, and C. P. Wright. On Incremental File System Development. *ACM Transactions on Storage (TOS)*, 2(2):161–196, May 2006.

[27] E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. In *Proc. of the Annual USENIX Technical Conference*, pages 55–70, San Diego, CA, June 2000. USENIX Association.

[28] Q. Zhu and W. W. Hsu. Fossilized index: The linchpin of trustworthy non-alterable electronic records. In *Proceedings of the ACM SIGMOD Conference*, pages 395–406, June 2005.