User-Level Secure Deletion on Log-structured File Systems

[Extended Abstract] *

Joel Reardon
ETH Zurich, Switzerland
reardonj@inf.ethz.ch

Claudio Marforio
ETH Zurich, Switzerland
maclaudi@inf.ethz.ch

Srdjan Capkun
ETH Zurich, Switzerland
capkuns@inf.ethz.ch

David Basin
ETH Zurich, Switzerland
basin@inf.ethz.ch

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; D.4.2 [**Operating Systems**]: Storage Management

General Terms

Secure deletion, privacy, flash memory

1. INTRODUCTION

Deleting a file from a storage medium serves two purposes: it reclaims storage resources and ensures that any sensitive information contained in the file becomes inaccessible. When done for the latter purpose, it is critical that the file is *securely* deleted, meaning that its content does not persist on the storage medium after deletion. Secure deletion is the act of deleting data from a storage medium such that the data is afterwards irrecoverable from the storage medium. The time between deleting data and it becoming irrecoverable is called the *deletion latency*.

Secure deletion enables users to protect the confidentiality of their data if their storage media are compromised, stolen, or confiscated under a subpoena. In this work, we present three user-level solutions for secure deletion on log-structured file systems assuming a coercive attacker capable of compromising both the storage medium and any secret keys required to access it. We consider the active case where users want to continually delete data, not simply sanitize their entire device before disposing it. We propose user-level solutions because they allow users themselves to achieve secure deletion without relying on manufacturers to provide this functionality.

The secure deletion problem is easily solved when the storage medium permits in-place updates: the user overwrites the data with a non-sensitive version [1]. Flash memory, however, does not support in-place updates [2]. Instead, erasures occur at a large granularity called the *erase block*. An erase block is orders of magnitude larger than a *page*, which is the granularity for read and write operations. Erase blocks can typically handle between 10^4 to 10^5 erasures before becoming unusable [3]. As such, achieving secure deletion by simply erasing a page of flash whenever some data is deleted is too expensive: it requires rewriting all the colocated valid data on the erase block and wears out the storage medium.

ASIACCS '12 Seoul, South Korea

Size	Deletion latency (hours) median	95th %ile
200 MB	41.5 ± 2.6	46.2 ± 0.5
1 GB	163.1 ± 7.1	169.7 ± 7.8
2 GB	349.4 ± 11.2	370.3 ± 5.9

Table 1: Deletion latency for different storage media sizes.

2. DATA DELETION IN YAFFS

YAFFS is a file system designed to operate on flash memory. YAFFS is log structured [4], so it consists of an ordered list of changes from an initial empty state. This ensures no in-place updates are performed on the flash memory, but rather new entries are appended to the log's end.

Delete data remains on a flash storage medium until the erase block on which it is stored is erased. Log-structured file systems like YAFFS perform erasures when the erase block only contain deleted data, or the space consumed by deleted data is needed to be recovered to store new data. As erase blocks will contain a mixture of valid and deleted data, YAFFS uses a garbage collection mechanism to copy valid data to a new location before performing an erasure.

We investigate data persistence on log-structured file systems by analyzing the internal memory of a Nexus One running Android/YAFFS and by simulating larger drives using the flash simulation kernel module `nandsim` based on the observed file system writing behaviours from the mobile phone. Table 1 show the median and 95th percentile data deletion latency from our experiment. We note that the 100th percentile measurement is undefined, as some deleted data remained accessible after the experiment.

The most important observation is that the time data remains on the device grows with the size of the device if the writing behaviour does not change. By having smaller amounts of free space available, the expected deletion time

decreases, which motivates our solutions that follow.

3. USER-SPACE SECURE DELETION

User-level solutions have very limited access to the flash storage medium. Such solutions can only create, modify, and delete the user's own local files. The principle behind our three solutions—purging, ballooning, and our hybrid solution—is that they reduce the file system's available free space to encourage more frequent garbage collection, thereby decreasing the deletion latency for deleted data. Ballooning and purging work in different ways and the hybrid solution combines them and relieves the disadvantages of each.

Purging.

Purging fills the storage medium to capacity, thus ensuring that no deleted data remains on the storage medium. Purging executes intermittently and halts after completion.

Completely filling the storage medium is possible provided the user is not subjected to disk-quota limitations. It typically requires garbage collecting (i.e., erasing) most erase blocks on the storage medium. This is because deleted chunks can occur in any erase block that sees active use, resulting in small data gaps throughout the file system. Therefore, purging provides guaranteed deletion at the worst-case cost of erasing every erase block on the storage medium.

Ballooning.

Ballooning continually occupies some fraction of the storage medium's empty space with junk files to ensure the free space remains within a target range. This reduces the total number of erase blocks available for allocation, thereby reducing the expected data deletion latency; erase blocks with deleted data will be garbage collected earlier than before. This is particularly useful for very large storage media, since the deletion latency often increases with the size of the storage medium.

Ballooning executes continually so that when the free capacity of the storage medium exits the desired range, the junk files are appropriately created or destroyed. Ballooning provides probabilistic improvements on deletion latency with an increase in erase block erasures that is parameterizable as the target free space.

Hybrid Solution.

Our hybrid solution combines both purging and ballooning to obtain the benefits of both approaches. At all times, ballooning is used to limit the amount of free space on the device. Periodically, purging is performed to obtain a guarantee of secure deletion. Moreover, as the space occupied by the ballooning files will not be erased, there are fewer erase blocks that need filling so purging is much quicker.

Results.

We implemented our solutions and evaluated their efficiency by simulating the writing behaviour of an Android mobile phone. We measured the block allocation rates, purge cost, and the 95th percentile deletion latency for different partition sizes and ballooning amounts. Table 2 presents a subset of the results available in the full version of this paper. The table is divided into results for two different partition sizes, and the first row are the observations without ballooning. The fill ratio is computed as the average percent of each erase block that contain live, valid data—that is, the percentage of data that needs to be copied during garbage collection.

Size	Free blocks	Fill ratio	Block allocs per hour	Purge cost (blocks)	Deletion latency (hr)
200 MB	603.8	20%	32.7 ± 2.3	1556.8	46.2 ± 0.5
	91.8	63%	53.4 ± 4.7	705.2	14.6 ± 1.3
	21.0	80%	95.0 ± 24.2	429.8	6.6 ± 0.2
	15.1	84%	166.5 ± 42.5	357.8	5.4 ± 1.5
2000 MB	9503.7	4%	25.3 ± 0.8	15663.8	370.3 ± 5.9
	387.8	43%	36.6 ± 1.5	1630.5	43.1 ± 8.6
	56.4	76%	87.5 ± 5.8	845.8	13.0 ± 0.4
	37.2	80%	205.4 ± 24.3	484.8	9.4 ± 1.9

Table 2: Block allocations and deletion times for the YAFFS file system for various partition sizes and ballooning amounts.

Ballooning operates by increasing the fill ratio of erase blocks. The deletion latency decreases quickly with even mild amounts of ballooning, a trend which is stronger for large storage media. We also see that the purge cost can be significantly reduced by using mild amounts of ballooning. Ballooning comes with a cost, however, which we see as an increase in the erase block allocations per hour. Since flash memory allows only a finite number of erasures, users can tune the amount of ballooning to a desired tradeoff between device lifetime and secure deletion.

Our solutions are applicable to any log-structured file system provided it has the following property: when it reports that the device is full, it must be the case that all deleted data is no longer available from the storage medium. When the source code of the file system is available, then the existence of this property is easily inspected. Otherwise, analysis of the raw flash memory's contents is necessary to determine if deleted data still remains.

4. REFERENCES

[1] S. Bauer and N. B. Priyantha. Secure Data Deletion for Linux File Systems. *Usenix Security Symposium*, pages 152–164, 2001.

[2] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.

[3] Micron Technology, Inc. Technical Note: Design and Use Considerations for NAND Flash Memory. 2006.

[4] M. Rosenblum and J. K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10:1–15, 1992.