Socarrat for Plan 9: building 9P WORM Devices

Gorka Guardiola Múzquiz, Enrique Soriano-Salvador
{gorka.guardiola,enrique.soriano}@urjc.es
GSyC, Universidad Rey Juan Carlos

May 20, 2025

Abstract

WORM (Write Once Read Many) devices permit data to be written only once. Subsequently, the system can read the data an unlimited number of times. These devices are used for logging purposes and are essential for a wide range of applications. In addition, the data should be tamper-evident: if committed data has been manipulated, the auditor must to be able to detect it. We propose to build local WORM tamper-evident devices with small and inexpensive single-board computers (SBCs). Currently, we are developing Socarrat, a system implemented in Go that follows a Reverse File System approach to provide ext4 and exFAT volumes with WORM properties. This approach is based on inferring the operations that are performed over a file system by analyzing the blocks written to the disk. 9p is an excellent candidate for exporting those WORM devices to other machines.

1. Introduction

A recurring theme in data regulations is the necessity for regulatory-compliant storage to ensure WORM properties that enable guaranteed retention of the data, secure deletion, and compliant migration [1]. Furthermore, the data should be tamper-evident, that is, any modification of the committed writes must be detectable.

The traditional approaches to provide WORM systems use continuous feed printers [2] optical devices [3], content addressed storage [4] or specially designed hardware not available for the general public [5]. There are also numerous distributed approaches to address this problem (e.g. [6]).

In addition to these considerations, the possibility of a cyberattack on the machine storing the data must be taken into account. If the attacker takes control of the system, she should not be able to delete or modify the WORM data of the log. At this point, all solutions based on file system capabilities may fail: If the attacker elevates privileges, she can change the file system configuration or simply modify or delete the files, the data blocks (directly from the block device), the address of the data blocks if it is content-addressed or format the file system.

We have been working on a local device to solve these problems. This device is based on three components:

- Sealfs [7, 8], a file system that provides forward integrity and tamper-evident storage.
- USB OTG, a USB port which can act as a device.
- Socarrat, a Go userspace program that provides the WORM garantieses to log files. Socarrat is based on a novel if somewhat contorted approach: The **Reverse File System**. This approach consists of analyzing the blocks written to the storage device to infer the file system operations executed at the upper layers. Socarrat only takes into account write operations at the end of the corresponding files, ignoring any other operation. This is suitable to keep genuine copies of *log files*.

We are emulating a WORM by using software, so it is important to state upfront what guarantees our system provides. Provided the integrity of the USB link (it can only be used as a mass storage device), the guarantees Socarrat provides, are formalized by what we have named the The Continuous Printer Model (CPM):

1. Once committed, data cannot be deleted/rewritten.
2. Liveness (now and then we commit something while the system works).
3. We do not guarantee that spurious or bad data is not committed in the future, or that the system cannot be stopped from *printing* by breaking it, but guarantee 1 is always preserved: What is *printed* cannot be *unprinted*.

Currently, we are using Raspberry Pi 4 computers to build the prototypes.

2. Usage scenario

Once the three components are present, the following scenario would work:

There are three distinct actors in the scenario, the auditor, Alice and possibly a malicious third party, Malice. Malice is an external intruder trying to manipulate the log in any way possible. Malice has APT capabilities, but is hampered by the need to conduct the attack over the network.

The auditor initializes a USB black box[1] with an specific tool. Then, once the device is properly configured, transfers it to the user, Alice.

Alice connects the USB device to her server. The operating system of the server detects an USB mass storage device formatted as an ext4 or exFAT file system. Then, this drive is mounted in the system. In the mount point, there is a file named `log`[2]. The applications running in Alice's machine are able to use this volume as a normal one, by performing the traditional system calls to work with the files (`open`, `write`, `read`, `close`, etc.). The only difference is that, when the applications access to `log` file, only read operations and append-only write operations

[1] Socarrat and SealFS running on a SOC like the Raspberry Pi with an SSD drive connected, in a box.
[2] There can be many files, we choose log as an example

are effective; the rest of write operations over the `log` file (and its metadata) are ignored within the USB mass storage device.

After normal operation, the auditor can then extract the `log` file from the device, along with an additional file that authenticates all the entries added to the log during this period. Using a diagnostic tool, the auditor can verify that the log data corresponds to the entries made during the logging period, ensuring that no records have been deleted or tampered with.

Alice and the auditor are also adversaries. Alice may try to delete or manipulate the logs, and has physical access to the device. She is hampered by the possibility of getting noticed doing so by the auditor.

3. Threat model and mitigations

Against each actor we have a different threat model and mitigation strategy:

Malice has APT capabilities and may completely compromise Alice's computer over the network. Alice has physical access o the device, and may compromise both devices, but faces repercussions if found manipulating the device.

Against Malice, we have as first line of defense the USB link as attack surface. If it does not fall (i.e. the integrity of the USB link is preserved and it only provides the mass storage device within specifications), we provide CPM properties. Data already committed cannot be rewritten or deleted in any way.

Against Alice, or against Malice if the USB link is broken, we provide the Forward Integrity Model. Data which was already committed cannot be manipulated without detection.

As it is now, in Plan 9 only the CPM properties are provided, as SealFS is not ported to Plan 9 and only through a 9P connection (not a USB link).

4. Overview of the architecture of the system

An overview of the architecture of the system can be see in figure 1.

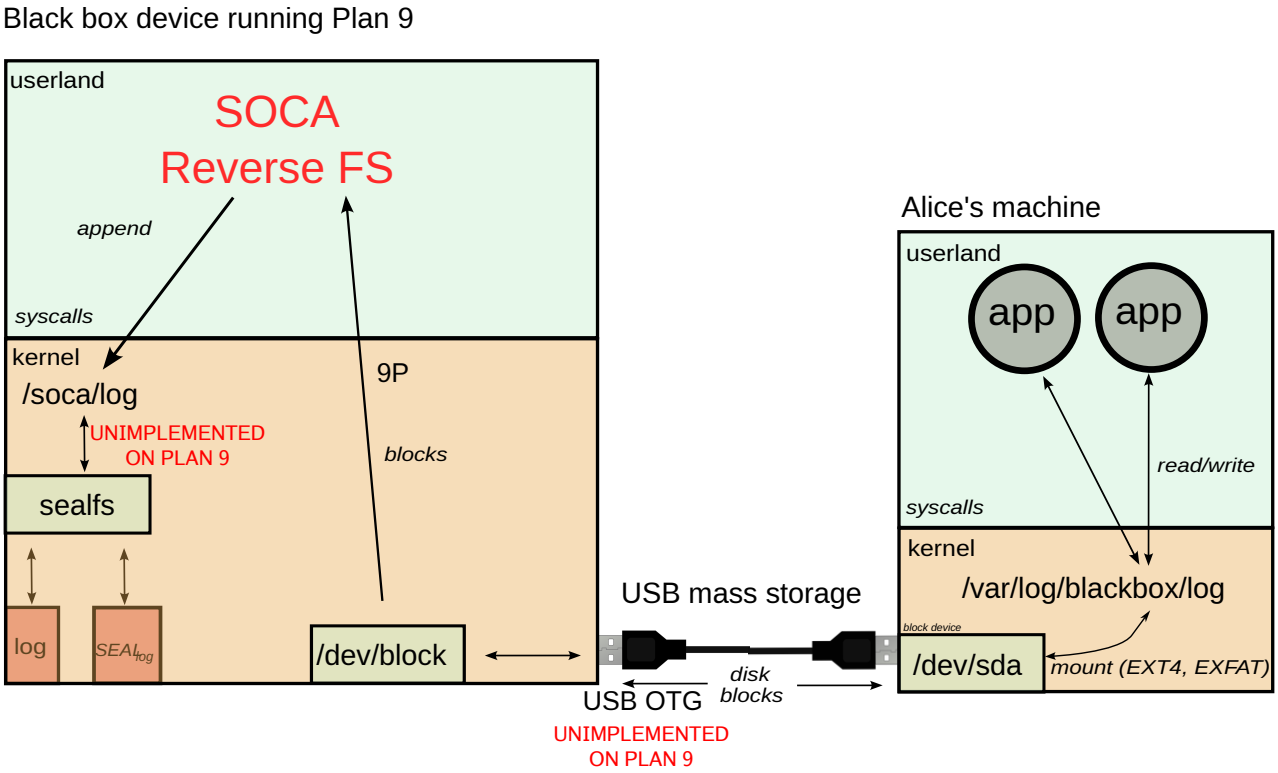

Figure 1: Simplified architecture of the system.

Socarrat (`soca`) is a program that watches a block device and operations on it (at the block level) and infers a subset (limited by the information available and possibly aggregated) of the operations updating the filesystem at a file/directory level of abstraction, as seen on figure 2. In our system, we use the inferred operations to generate a log in a separate filesystem controlled by SealFS [8], which will provide forward integrity.

Note that the USB link or, more precisely the block device interface, which we extend to the other machine via the USB mass storage device, acts as a choke point: the set of operations on it is limited. This enables the *reverse filesystem* to act as a kind of data diode by only honoring whatever operations it wants (writes at the end of the file) and ignoring the rest.

Note that the idea of a reverse filesystem is general and not limited to our application. We are using it to provide a WORM, but this approach can be used for OS fingerprinting (i.e. detecting the OS of the other machine by analyzing patterns in the block usage), debugging file systems, fuzzing, etc.

One of the benefits of the reverse filesystem approach is its portability. The machine where the device running the reverse filesystem is plugged in does not have to know anything about it. As long as the filesystem being reversed is supported, no special software needs to be run on it.

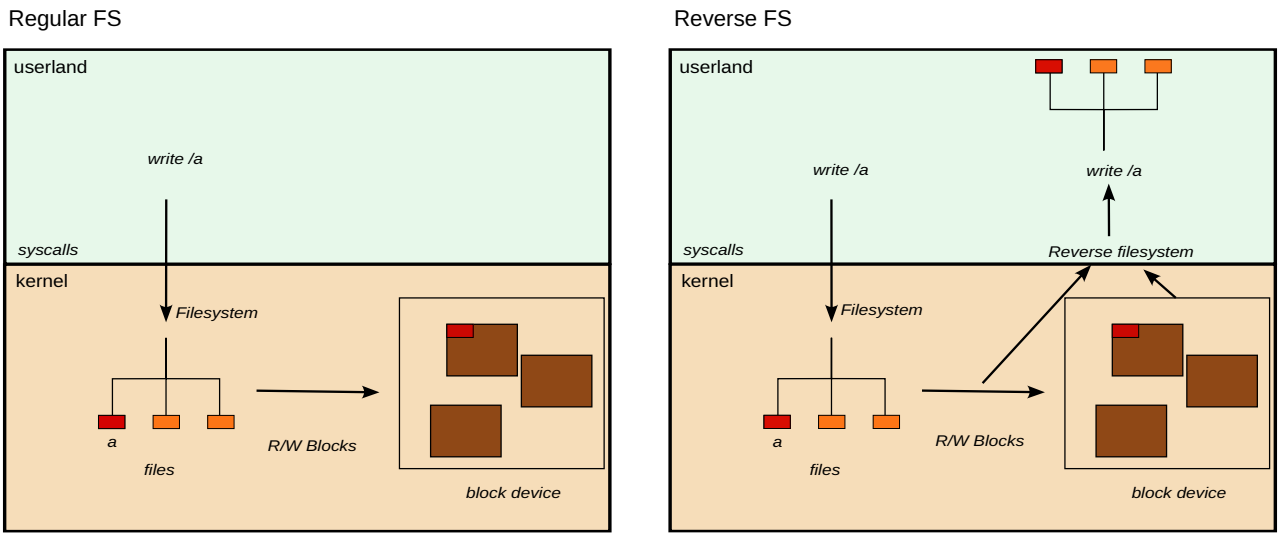

Figure 2: Reverse filesystem: recover (some) write operations from block operations.

5. Plan 9 port

While we built all the components originally for Linux, we have ported Socarrat to Plan 9. This has been a very simple endeavour due to the fact that it is written in Go with minimal external dependencies.

The original version for Linux exported the device using the NBD protocol [9]. This protocol is designed to serve a block device from userspace. The same can be done in a simpler and more general way with 9P. While NBD has several extensions and variants for special cases, 9P has several advantages over it.

First, 9P is a general purpose protocol not specially designed to export block devices. While this may seem like a weakness, it is not. As a consequence of it being general purpose, it has libraries to support it which have been tested thoroughly because they are needed for other uses.

Also, whenever a special operation has to be added, new operations on the `ctl` file, can be added. Therefore, extensibility is easier. It is also easier to model the behavior after it is implemented (for example, who can do which operations can be controlled by the permissions of the files).

All the general purpose debugging tools to spy on the protocol and the expertise dealing with the corner cases, the concurrency model, etc. can be applied from the general knowledge. When working with NBD, the protocol has to be learnt from scratch with all its special corner cases. In a fraction of time of what took to make the NBD version work, we had a 9P version working which was more robust and passed all the same tests.

As it is now, Socarrat posts on `srv` a directory with two files, the device, and a `ctl` file. The `ctl` file can be read and will print the name of the backing image file, and the internal and external files mapped by the *reverse filesystem*. It also accepts a `shutdown` command.

The current version is runnable, but it is missing some of the other components to make the whole system work in Plan 9. The first one is SealFS [7, 8], which provides tamper-evidence following the forward integrity model. The second component is support for USB OTG, present, for example, in the Raspberry Pi. The third is a way to generate an image for an ext4 or exFAT filesystem to serve, but this can be done on another system if needed.

6. Related work

The idea of a reverse filesystem is quite unconventional and there is very little in terms of comparable systems or devices. The closest two we found are related to one another (one was inspired by the other). The first one is vvfat [10]. Vvfat is a block driver served by qemu which serves a virtual VFAT filesystem in which the files represent the files of an underlying directory and follow them when read an written. Inspired on this, there is the nbdkit floppy plugin [11]. These two devices implement a kind of *synthetic reverse filesystem*. It has two main differences:

1. These systems are unconstrained by a liveness property. They only need to guarantee the underlying files are completely synchronized when they are unmounted. Our reverse filesystem needs to extract valid operations continuously in order to work.
2. The second is an implementation difference. A *reverse filesystem* observes the data structures present in a block device image. A *synthetic reverse filesystem* invents these data structures on the fly.

As we stated in the introduction, to implement WORM devices the traditional approaches are using continuous feed printers [2] optical devices [3], content addressed storage [4] or specially designed hardware not available for the general public [5]. There are also numerous distributed approaches to address this problem, leveraging Cloud Computing and blockchain technologies, see as an example [6] (the list is too extensive to be enumerated here).

We can classify all these approaches as:

1. Requiring special expensive, which provide strong guarantees but are cumbersome or unavailable hardware (printers, optical devices).
2. Software based which do not provide any guarantees in the cases of a cyberattack (content-addressed systems).
3. Distributed, which may stop working if the network is unavailable or there is a denial of service attack.

4. Closed systems like [12], for which the inner workings and it is difficult to know what real guarantees they provide.

Our approach requires inexpensive hardware, provides strong security guarantees and works locally without dependence on third parties or distributed systems. Our approach is also specialized for file systems containing log append-only files, not a general purpose file system.

References

[1] R. Sion, "Strong worm," in *2008 The 28th International Conference on Distributed Computing Systems*, pp. 69–76, 2008.

[2] M. Bellare and B. S. Yee, "Forward integrity for secure audit logs," tech. rep., University of California at San Diego, 1997.

[3] S. Quinlan, "A cached worm file system," *Software: Practice and Experience*, vol. 21, no. 12, pp. 1289–1299, 1991.

[4] S. Quinlan, J. McKie, and R. Cox, "Fossil, an archival file server," *World-Wide Web document*, 2003.

[5] J. Leppäniemi, T. Mattila, T. Kololuoma, M. Suhonen, and A. Alastalo, "Roll-to-roll printed resistive worm memory on a flexible substrate," *Nanotechnology*, vol. 23, no. 30, p. 305204, 2012.

[6] M. Li, C. Lal, M. Conti, and D. Hu, "Lechain: A blockchain-based lawful evidence management scheme for digital forensics," *Future Generation Computer Systems*, vol. 115, pp. 406–420, 2021.

[7] E. Soriano-Salvador and G. Guardiola-Múzquiz, "Sealfs: Storage-based tamper-evident logging," *Computers and Security*, vol. 108, p. 102325, 2021.

[8] G. Guardiola-Múzquiz and E. Soriano-Salvador, "Sealfsv2: combining storage-based and ratcheting for tamper-evident logging," *Int. J. Inf. Secur.*, vol. 22, p. 447–466, Dec. 2022.

[9] "Network block device github," 2024.

[10] "Qemu block driver for virtual vfat." [Online; accessed jan-2025].

[11] "Nbd virtual floppy disk from directory." [Online; accessed jan-2025].

[12] "Wormdisk zt storage," 2024.