Socarrat: Building Cost-Effective Secure WORM Devices Following the Reverse File System Approach

Abstract

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

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. A recurring theme in data regulations is the necessity for regulatory-compliant storage to ensure WORM assurances that enable guaranteed retention of the data, secure deletion, and compliant migration [8]. In addition, the data should be tamper-evident: if the data has been manipulated, the auditor must to be able to detect it.

Although it may appear straightforward, implementing secure WORM devices is highly complex. The traditional approach to this problem involves using paper printers or optical devices (i.e. CD-ROMs or DVDs) [5], which inherently provide WORM capabilities. Another approach is to use some kind of content addressed storage like the filesystem Fossil [7] does with Venti [6]. Some manufacturers offer WORM hard disks compatible with standard applications and file formats, including conventional file systems such as NTFS, ExFAT and EXT4 [1]. However, the underlying technology of these devices is closed and obtaining further details about the architecture and pricing appears to be challenging. Although there is extensive research focused on developing WORM memories using different physical approaches (e.g. organic components), this technology is not currently available. Naturally, there are numerous distributed approaches to address this problem, leveraging Cloud Computing and blockchain technologies, see as an example [4] (the list is too extensive to be enumerated here).

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 propose Socarrat, a radical and cost-effective solution to address this problem locally with a simple external usb device: a small single board running Linux and with USB OTG support.[1] For example, a Raspberry Pi with Socarrat can export a 1 TB USB mass storage WORM device, which can be mounted on any regular operating system as an EXT4 or EXFAT file systems (i.e. without running special software) for approximately $100.

Socarrat is based on a novel if somewhat contorted approach: The ***Reverse File System***. This approach consists of analyzing the blocks that are written to the storage device in order 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.

Suppose the following scenario: Alice connects a USB black box[2] 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`. The applications running in the server 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 are effective; the rest of write operations over the `log` file (and its metadata) are ignored within the USB mass storage

[1] USB OTG enables a USB port to be used as a device.

[2] Socarrat running on a SOC like the Raspberry Pi with an SSD drive connected, in a box.

device. Later, she can 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, she or a third party (the auditor) can verify that the log data corresponds to the entries made during the operational period, ensuring that no records have been deleted or tampered with.

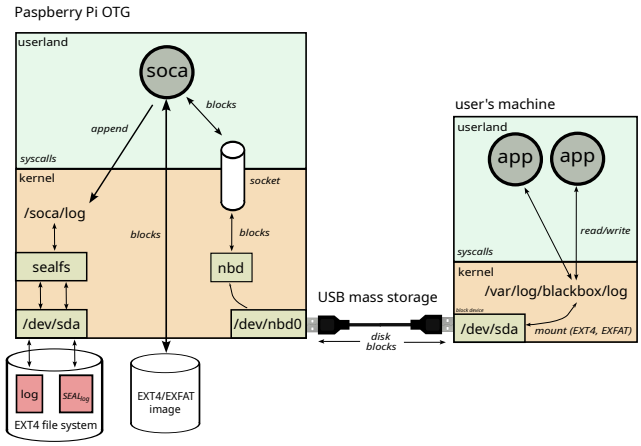

Figure 1. General architecture of the Socarrat system.

Figure 1 shows the general architecture of our system:

- In the user's machine (Linux), the USB mass storage device (`/dev/sda`) is mounted in `/var/log/blackbox/`. In this directory, there is the `log` file cited above[3]. Applications read and write `/var/log/blackbox/log` as a normal file. If we are using EXT4, this file is an append-only file (using the extended attributes supported by this file system). All operations over the file system are translated to operations over the block device. When data is written in the log file, the corresponding blocks of the data and the metadata (i.e. the i-node) must be written to the disk (`/dev/sda`).
- Our Socarrat device receives the read/write operations on the blocks through the USB and the NDB (network block device), that are redirected to the `/dev/nbd0` block device. The operations are forwarded to our program, `soca`, by using a local unix domain socket.
- `soca` is a userspace program written in Go that processes the NDB requests to read and write blocks, following the NDB protocol [2].

  This is where the *Reverse File System* approach is implemented. By analyzing the operations on the blocks, `soca` infers the operations that the user's machine is performing on the file system objects inside the block device served by `soca` itself. We currently support EXT4 and ExFAT file systems, so the system works on all major operating systems (Linux, Windows and macOS).

  In the EXT4 case, `soca` watches the read/write operations on the blocks of the `log` file i-node. When it grows, the new data added to the last blocks referenced by the i-node are appended to `/soca/log` file. In fact, the real `/soca/log` file is stored in the main file system of the Socarrat device, which is not served to the user's machine.

  In the ExFAT case, `soca` watches the FAT table (or the contiguous blocks if it bitmap allocated) of `log` file and does the analogous thing appending the last part as it grows to the `/soca/log` file.

  There is no operation to delete or overwrite the externally kept `/soca/log` file: it behaves as a WORM.
- In order to provide the block device, `soca` has been configured to use a EXT4 (or ExFAT) image with the initial file system withing the USB mass storage device plugged to the user's machine.
- As stated before, the real `/soca/log` file is stored in a private volume (`/dev/sda` in the Linux system), which is only accessible within the Socarrat device. This file is secured by SealFS [3, 9], a tamper-evident mechanism that provides forward integrity to log files following a HMAC ratchet/storage-based hybrid scheme.

  At the end, we have two main files in this private volume: (i) The real `/soca/log` file with all the data appended by the applications running in the user's machine and (ii) the authenticated log file (also known as $SEAL_{log}$, see [9]) that contains the metadata of all the append operations performed over the `log` file. The metadata is authenticated with HMACs ratchets that make it tamper-evident. After normal operation, the first file can be verified using the second file. This can be done by the user or by a third party (i.e. an auditor).

In this system, in case of a total compromise of the user's machine after a remote logical attack, the attack surface to delete or modify the data already stored in the log file is restricted to the USB link. Moreover, if the attacker ultimately succeeds in modifying the data previously stored in the log file, the attack will be detected.

The USB link can be hardened in various ways, like having a intermediary program that watches the USB transactions and alerts of anything out of what is expected, the endpoints for the mass storage and the corresponding block operations. Also, the device can be powered through the USB link, but this can be a weakness, so it can carry a small battery to close this attack surface which can go from power analysis side channels to powering on and off to try to find weak states of the system.

We are currently developing the first research prototype of Socarrat, which will be released as libre software under the GNU General Public License.

[3] Note that the name and number of log files can be configured, we suppose that there is only one file for simplicity.

References

[1] Wormdisk zt storage, 2024. URL `https://greentec-usa.com/products/`.

[2] Networn block device github, 2024. URL `https://github.com/NetworkBlockDevice`.

[3] G. Guardiola-Múzquiz and E. Soriano-Salvador. Sealfsv2: combining storage-based and ratcheting for tamper-evident logging. *Int. J. Inf. Secur.*, 22(2):447–466, Dec. 2022. ISSN 1615-5262. doi: 10.1007/s10207-022-00643-1.

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

[5] S. Quinlan. A cached worm file system. *Software: Practice and Experience*, 21(12):1289–1299, 1991. doi: https://doi.org/10.1002/spe.4380211203.

[6] S. Quinlan and S. Dorward. Venti: A new approach to archival data storage. In *Conference on file and storage technologies (FAST 02)*, 2002.

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

[8] R. Sion. Strong worm. In *2008 The 28th International Conference on Distributed Computing Systems*, pages 69–76, 2008. doi: 10.1109/ICDCS.2008.20.

[9] E. Soriano-Salvador and G. Guardiola-Múzquiz. Sealfs: Storage-based tamper-evident logging. *Computers and Security*, 108:102325, 2021. ISSN 0167-4048. doi: 10.1016/j.cose.2021.102325.