Implementation

DRFTL. To implement our DRFTL, we have modified OpenNFM [3], an open-sourced flash controller framework written in C. To keep track of the physical page locations of new delta data, the DRFTL reserves a small region in the flash memory and, once new data are written, it will keep the corresponding PBA in this region (organized as a stack) for the next backup process. The set of PBAs being kept is called "PBAs-to-extract". To prevent the delta data (invalidated by the malware) from being deleted before being backed up remotely, the DRFTL uses a modified garbage collection strategy so that the new delta data created since the last backup process will not be deleted. During the backup process, the DRBack app will continuously read a reserved LBA until all the new delta data have been extracted and, once the DRFTL receives a read request from this reserved LBA, it will pick a new physical page from PBAs-to-extract (i.e, the corresponding page PBA is popped from the stack), and return the delta data stored in it; the DRFTL also computes and returns a cryptographic tag (instantiated using HMAC-SHA1) over these delta data, the corresponding logical page address, as well as the version number and sequence number. During the recovery process, the DRFTL will work with the DRecover app to place the data back to their original LBAs.
DRBack. The DRBack app consists of two major software components (both were implemented in C): one software component (CA) runs in the normal world, and the other software component (TA) runs in the secure world of ARM TrustZone. The CA issues a (secret) special write sequence to the reserved LBAs (there is a fixed mapping between the sector addresses of the disk and the LBAs in the FTL, e.g., for 512-byte sector size and 2KB page size, each LBA address can be converted to the sector address by multiplying 4); the DRFTL monitors the FTL_Write interface and, once detecting this special write sequence, it will change its state to the backup state, and work with the DRBack to extract delta data from the raw flash memory. Once the delta data are completely extracted (the DRFTL will inform the CA by returning a special string), the CA will send them to the TA for verification and, if the TA successfully verifies them, it will send them to the version control server. Deployment of TA relies on the support of ARM TrustZone technology. OP-TEE (Open Portable Trusted Execution Environment) [2] is an open-source project which provides complete and thorough platform for developers to build their own applications on ARM TrustZone. There are many electronic boards which are officially supported by OP-TEE. We ported OP-TEE to Raspberry Pi 3B with Raspbian as the root

file system. It should be noted that Raspberry Pi 3B did not support secure boot and other features [1]. However, as a prototype, Raspberry Pi is sufficient to demonstrate the feasibility of building the TA of DRBack. To enable the TA to communicate with the remote server, we relied on the TEE socket APIs.

DRecover. We have implemented DRecover in C. The DRecover app first extracts the most recent delta data from the flash memory (similar to the backup process), and then works with the remote version control server and the DRFTL to restore data at the corruption point. A significant step in DRecover is to commit an arbitrary historical version, retrieved from the remote server, back to the raw flash memory. This step is implemented as follows: 1) The DRecover app retrieves from the remote server a data version, and verifies it based on the associated tags. 2) The DRecover app issues a (secret) special write sequence to the reserved LBAs and, the DRFTL monitors the FTL_Write interface and detects this special write sequence. The DRFTL changes its state to the recovery state, and works with the DRecover to commit this entire data version back to the flash memory. Each data version is a collection of data chunks, each of which is corresponding to the data stored in a flash page and is associated with an LBA. Based on the LBA, the DRecover app can figure out the corresponding sector addresses (one page is corresponding to a few contiguous 512-byte sectors), and write this chunk to the corresponding sectors; the DRFTL will obtain this chunk from the FTL_Write, write it to a physical page, and update the mapping between the LBAs and the PBAs. 3) The DRecover app issues another special write sequence to the reserved LBAs, to terminate the recovery state.

We have also implemented a server program in C, which runs remotely and stores all the data sent by the mobile device in a version control manner. The server also allows the mobile device to retrieve an arbitrary data version.

References

[1] OP-TEE documentation - Raspberry Pi 3. `https://optee.readthedocs.io/en/latest/building/devices/rpi3.html`.

[2] Open Portable Trusted Execution Environment. `https://www.op-tee.org/`.

[3] Google Code. Opennfm. Retrieved May 17, 2019, from `https://code.google.com/p/opennfm/`, 2011.