Secure Audio DRM: System Design and Implementation

Team Cacti
Rochester Institute of Technology
2020 MITRE eCTF

1 Overview

This document presents the design and implementation of Team Cacti's submission to MITRE eCTF 2020. The goal of this project is to design a secure audio DRM system. For more information of the requirements, please refer to MITRE's 'Challenge Description and Rules' document.

2 System Modules

Team Cacti's submission has the following modules:

1. A Linux application (`mipod`) that takes users' input, reads and writes to the SD card for protected music files. The Linux application runs on the Cortex-A CPU (`cortex`). The application communicates with the firmware running on the MicroBlaze CPU mainly through the main memory (`dram`). This application is not in the trusted computing base, since the underlying Linux OS may be malicious.
2. The firmware (`fw`) runs on the MicroBlaze CPU (`mb`), which is a soft-core on the FPGA. Besides the `dram`, `fw` can also access the block RAM on the FPGA (`bram`), which the `cortex` cannot access.
3. An AES module implememented in the FPGA.
4. A suite of Python tools (`ptools`) that is supposed to be run by MITRE in provisioning.

In addition, we removed the MDM block that is the debug module of the MicroBlaze. We also disabled the Ethernet 0 port and other unnecessary peripherals on the Zynq processor.

3 System Workflow

At a very high level, our system works in the following steps:

1. During provisioning, the `ptools` hashes users' PINs using `PDKDF_SHA512`, generates a global encryption key `k1` for AES encryption, generates a global key `k2` for HMAC to protect music content integrity, generates a key `ku` for each user. `k1` is hardcoded into the FPGA AES module. `k2` and `kus` only reside on `bram` at runtime.
2. The `ptools` will divide each music into multiple segments. Each segment is encrypted using `k1`, SHA1-HMACed using `k2`. A new music file is created and the header is SHA512-HMACed using each owner's `ku`.
3. At runtime when a user tries to login, her input PIN will be copied from `dram` to `bram`. `fw` compares the hash value.
4. When a user tries to play a song she owns, encrypted song segments are copied from `dram` to `bram`. Decryption and integrity check are performed on `bram` when before they are played.

5. When a legit user shares a song to another user, a new header is constructed and HMAC generated using the ower's `ku`. The header is then copied from `bram` to `dram`. `mipod` then writes the new header to the protected music file.
6. If the user is invalid or in a locked region, she can only play or digital out the first 30s of the music.
7. When playing the song, the operations provided are pause, resume, restart, stop, forward 5s, and rewind 5s.

4 Data Structures

4.1 drm_header

```
typedef struct __attribute__((__packed__)) {
    uint8_t song_id[SONGID_LEN]; //size should be macroized. a per-song unique
       ↪ ID.
    uint8_t ownerID; //the owner's name.
    uint8_t pad[3];
    uint8_t regions[MAX_SHARED_REGIONS]; //this is a bit on the large size, but
       ↪ disk is cheap so who cares
    //song metadata
    uint32_t len_250ms; //the length, in bytes, that playing 250 milliseconds of
       ↪  audio will take. (the polling interval while playing).
    uint32_t nr_segments; //the number of segments in the song
    uint32_t first_segment_size; //the size of the first song segment (which may
       ↪  not be the full SEGMENT_BUF_SIZE), INcluding trailer.
    wav_header wavdata;
    //validation and sharing
    uint8_t mp_sig[HMAC_SIG_SIZE]; //a signature (using the mipod private key)
       ↪ for all preceeding data
    uint8_t shared_users[MAX_SHARED_USERS]; //users that the owner has shared
       ↪ the song with.
    uint8_t owner_sig[HMAC_SIG_SIZE]; //a signature (using the owner's private
       ↪ key) for all preceeding data. resets whenever new user is shared with.
} drm_header;
```

4.2 mipod_buffer

```
typedef volatile struct __attribute__((__packed__)) {
    uint32_t operation; //IN, the operation id from enum mipod_ops
    uint32_t status; //OUT, the completion status of the command. DO NOT read
       ↪ this field.
    char shared_user[UNAME_SIZE];
    union {
        mipod_login_data login_data;
        mipod_query_data query_data;
        mipod_digital_data digital_data;
        char buf[MAX_SONG_SZ];
    };
}mipod_buffer;
```