9 Defusing the Qualcomm Dragon

a short story of research by Josh "m0nk" Thomas

Earlier this year, Nathan Keltner and I started down the curious path of Qualcomm SoC security. The boot chain in particular piqued my interest, and the lack of documentation doubled it. The following is a portion of the results.[7]

Qualcomm internally utilizes a 16kB bank of one time programmable fuses, which they call QFPROM, on the Snapdragon S4 Pro SoC (MSM8960) as well as the other related processors. These fuses, though publicly undocumented, are purported to hold the bulk of inter-chip configuration settings as well as the cryptographic keys to the device. Analysis of leaked documentation has shown that the fuses contain the primary hardware keys used to verify the Secure Boot 3.0 process as well as the cryptographic information used to secure Trust Zone and other security related functionality embedded in the chip. Furthermore, the fuse bank controls hardwired security paths for Secure Boot functionality, including where on disk to acquire the bootable images. The 16kB block of fuses also contains space for end user cryptographic key storage and vendor specific configurations.

These one time programmable fuses are not intended to be directly accessed by the end user of the device and in some cases, such as the basic cryptographic keys, the Android kernel itself is not allowed to view the contents of the QFPROM block. These fuses and keys are documented to be hardware locked and accessible only by very controlled paths. Preliminary research has shown that a previously unknown 4kB subset of the 16kB block is mapped into the kernel IMEM at physical location `0x0070_0000`. The fuses are also documented to be shadowed at `0x0070_4000` in memory. Furthermore, there exists somewhat unused source code from the Code Aurora project in the Android kernel that documents how to read and write to the 4kB block of exposed fuses.

Aside from the Aurora code, many vendors have also created and publicly shared code to play with the fuses. LG is the best of them, with a handy little kernel module that maps and explores LG specific bitflags. In general, there is plenty of code available for a clever neighbor to learn the process.

The following are simple excerpts from my tool that should help you explore these fuses with a little more granularity. Please note, *and NOTE WELL*, that writing eFuse or QFPROM values can and probably will brick your device. Be careful!

One last interesting tidbit though, one that will hopefully entice the reader to do something nifty. SoC and other hardware debugging is typically turned off with a blown fuse, but there exists a secondary fuse that turns this functionality back on for RMA and similar requests. Also, these fuses hold the blueprint for where and how Secure Boot 3.0 works as well as where the device should look for binary blobs to load during setup phases.

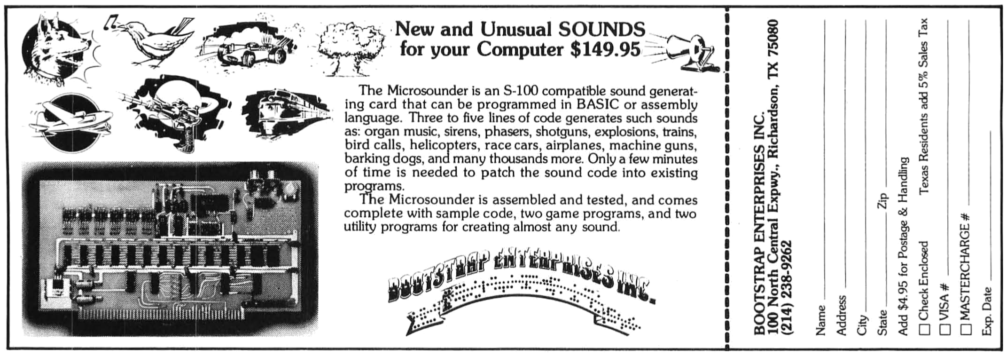
```
//--------------------------------------------------------------
// Before we can crawl, we must have appendages
//--------------------------------------------------------------
static int map_the_things (void) {
  uint32_t i;
  uint8_t stored_data_temp;
  //--------------------------------------------------------------
  // Stage 1: Hitting the eFuse memory directly (this is not supposed to work)
  //--------------------------------------------------------------
  pr_info("m0nk -> and we run until we read: %i lovely bytes\n", QFPROM_FUSE_BLOB_SIZE);

  for (i = 0; i < QFPROM_FUSE_BLOB_SIZE; i++) {
    stored_data_temp = readb_relaxed((QFPROM_BASE_MAP_ADDRESS + i));

    if (!stored_data_temp) {
      pr_info("m0nk -> location: , byte_number: %i, has no valid value\n", i);
      base_fuse_map[i] = 0;
    }else{
      pr_info("\tm0nk -> location: , byte_number: %i, has value: %x\n",
              i, stored_data_temp);
      base_fuse_values[i] = stored_data_temp;
      base_fuse_map[i] = 1;
    }
```

[7]Thanks Mudge!

```
  }

  stored_data_temp = 0;

  //---------------------------------------------------------------
  // Stage 2: Hitting the eFuse shadow memory (this is supposed to work)
  //---------------------------------------------------------------
  // for (i = 0; i < QFPROM_FUSE_BLOB_SIZE; i++) {
  //     stored_data_temp = readb_relaxed((QFPROM_SHADOW_MAP_ADDRESS + i));
  //     if (!stored_data_temp) {
  //             pr_info("m0nk -> location: , byte number: %i, has no valid value\n", i);
  //             shadow_fuse_map[i] = 0;
  //     }else{
  //             pr_info("\tm0nk -> location: , byte number: %i, has value: %x\n", i, stored_data_temp);
  //             shadow_fuse_values[i] = stored_data_temp;
  //             shadow_fuse_map[i] = 1;
  //     }
  // }

  return 0;
}


//---------------------------------------------------------------
// Now we can crawl, and we do so blindly
//---------------------------------------------------------------
static int dump_the_things (void) {
  // This should get populated with code to dump the arrays to a file for offline use.
  uint32_t i;

  pr_info("\n\nm0nk->␣Known␣QF-PROM␣Direct␣Contents!\n");

  for (i = 0; i < QFPROM_FUSE_BLOB_SIZE; i++) {
    if (base_fuse_map[i] == 1)
      pr_info("m0nk␣->␣offset:␣0x%x␣(%i),␣has␣value:␣0x%x␣(%i)\n",
              i, i, base_fuse_values[i], base_fuse_values[i]);
  }

  // pr_info("\n\nm0nk-> Known QF-PROM Shadow Contents!\n");

  // for (i = 0; i < QFPROM_FUSE_BLOB_SIZE; i++) {
  //     if (shadow_fuse_map[i] == 1)
  //             pr_info("m0nk -> offset: 0%xx, has value: 0x%x (%i)\n",
  //                     i, shadow_fuse_values[i], shadow_fuse_values[i]);
  // }

  return 0;
}
```

Writing a fuse is slightly more complex, but basically amounts to pushing a voltage to the eFuse for a specified duration in order for the fuse to blow. This feature is included in my complete fuse introspection tool, which will be available through Github soon.[8]

Have fun, break with caution and enjoy.



New and Unusual SOUNDS for your Computer $149.95

The Microsounder is an S-100 compatible sound generating card that can be programmed in BASIC or assembly language. Three to five lines of code generates such sounds as: organ music, sirens, phasers, shotguns, explosions, trains, bird calls, helicopters, race cars, airplanes, machine guns, barking dogs, and many thousands more. Only a few minutes of time is needed to patch the sound code into existing programs.

The Microsounder is assembled and tested, and comes complete with sample code, two game programs, and two utility programs for creating almost any sound.

BOOTSTRAP ENTERPRISES INC.

BOOTSTRAP ENTERPRISES INC.
100 North Central Expwy., Richardson, TX 75080
(214) 238-9262

Name ______
Address ______
City ______
State ______ Zip ______
Add $4.95 for Postage & Handling
☐ Check Enclosed Texas Residents add 5% Sales Tax
☐ VISA # ______
☐ MASTERCHARGE # ______
Exp. Date ______

8https://github.com/monk-dot/DefusingTheDragon