Reversing a firmware uploader

&

Others NFC stories

About me

- My name's Slurdge/Aurélien
- I'm an enthusiast about NFC, not an expert!
  My main work involves games 🕹️
- I know a bit of reverse engineering
- *Let's see how to combine them*
- I like to have open source solutions to handle my hardware.
  Especially embedded hardware where the proprietary tools are mainly Windows based with cumbersome licenses.

Plan

1. A short story
2. Reversing the Chameleon-Mini clone uploader
3. The strange case of the half working NFC tag

A short story

- I once put my hand on a LF tag of a friend. Excited, I tried to clone with my brand new proxmark.
  I was a bit too ambitious and erased the tag (it accepted any write command). Now, there was a **backup** tag in his car, which was in this garage.
- The garage could be opened only by a valid tag.
- And it was a Sunday night.
- Lesson learned: *be **very careful** with other's people tags! Or you will camp outside a garage waiting for someone to come in*

Chameleon-Mini

> The ChameleonMini is a versatile contactless smartcard emulator compliant to NFC. To support our project, buy it here: https://shop.kasper.it.

It was created by David Oswald and Timo Kasper.

The original ChameleonMini is now at revision G. ChameleonMini is open, you can find the whole hardware and firmware files at https://github.com/emsec/ChameleonMini.

Chameleon Mini: RevE Rebooted

> European Exclusive to Lab401, the Chameleon Mini: RevE Rebooted is a highly optimized fork of the original project.

- Project was done by ProxGrind (hardware) and dxls (firmware) for Lab401
- It was later open sourced
- You can find it pretty easily on *certain* Chinese websites.

How to upload the Firmware

If you browse the product page on AliExpress, this message is written in big green letters:

> Big News: we decide to make the reboot open source, so, after you place order, will give you the link.

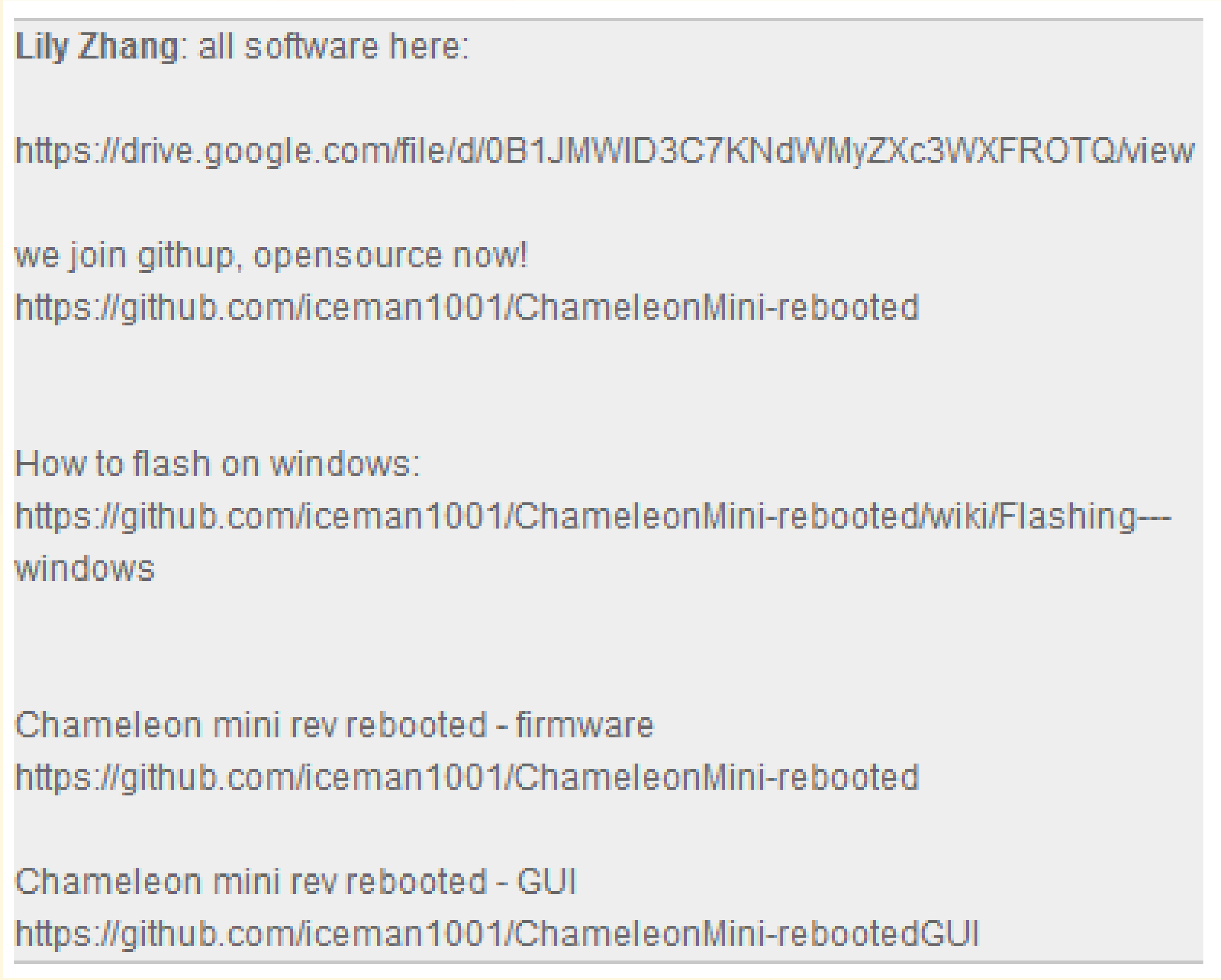
How to upload the Firmware

Good. Let's ask them then ➡️

Oh, it was ❄️iceman's repository all along... 🤦

Let's download this Google Drive package anyway.

❄️*iceman of proxmark fame*



Lily Zhang: all software here:

https://drive.google.com/file/d/0B1JMWID3C7KNdWMyZXc3WXFROTQ/view

we join githup, opensource now!
https://github.com/iceman1001/ChameleonMini-rebooted

How to flash on windows:
https://github.com/iceman1001/ChameleonMini-rebooted/wiki/Flashing---windows

Chameleon mini rev rebooted - firmware
https://github.com/iceman1001/ChameleonMini-rebooted

Chameleon mini rev rebooted - GUI
https://github.com/iceman1001/ChameleonMini-rebootedGUI

Goal

We want to:

- Build a firmware
- Upload a firmware
- Get rid of those pesky executable files
- Work from Windows, Linux, MacOS
- *Should be an interesting challenge!*

What we have so far

Two executables

`BOOT_LOADER_EXE.exe`
`Createbin.exe`

Yeah, can't get much more generic than that... Except maybe `BOOT_LOADER_EXE_DOT_EXE`

It's pretty obvious that `Createbin.exe` is responsible to create the file used by `BOOT_LOADER_EXE.exe`, since it's written in the github wiki.

What we have so far

There are also issues on https://github.com/iceman1001/ChameleonMini-rebooted/ that are talking about some AES encryption, file manipulation and so on...

So it may simply be a case of finding the AES key! Should be pretty easy to find in the executable... However that would mean there is an AES engine in the bootloader?

Strange (as the chip is not that powerful)... Let's investigate it later!

Running `BOOT_LOADER_EXE.exe`

If we use the `BOOT_LOADER_EXE.exe` file on Windows, after putting the Chameleon-Mini rebooted in DFU mode, we get this:

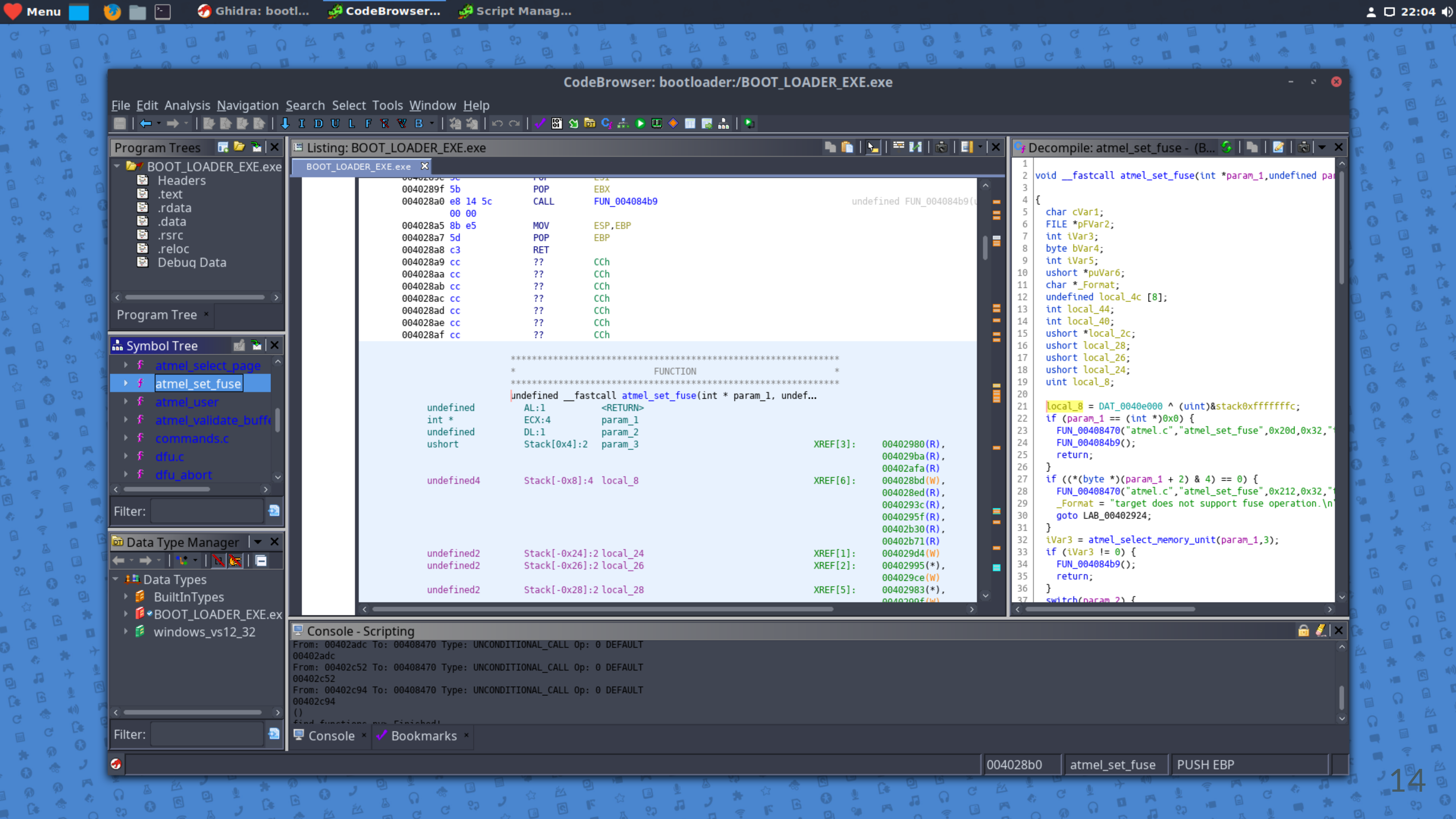
```
old_driver_bootloader
Erasing flash... Success
Checking memory from 0x0 to 0x6FFF... Empty.
0% 100% Programming 0x20 bytes...
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] Success
0% 100% Reading 0x400 bytes...
0% 100% Programming 0x5B00 bytes...
[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>] Success
0% 100% Reading 0x7000 bytes...
load_success!
```

Reversing: `Createbin.exe`

Our first step is to duplicate the `Createbin.exe` so we can create binary files that would be accepted by the uploader part. Let's fire... Ghidra 🐲!

It's simply a matter of opening the exe in Ghidra and it decompiles itself nicely. It's very easy to follow the flow and find the `main` function. Only slight editing has been done.

PTS2019 note: I did again the reverse engineering with Ghidra to show how to do it with open source tools.



```
0040289f 5b           POP      EBX
004028a0 e8 14 5c     CALL     FUN_004084b9                        undefined FUN_004084b9(
         00 00
004028a5 8b e5        MOV      ESP,EBP
004028a7 5d           POP      EBP
004028a8 c3           RET
004028a9 cc           ??       CCh
004028aa cc           ??       CCh
004028ab cc           ??       CCh
004028ac cc           ??       CCh
004028ad cc           ??       CCh
004028ae cc           ??       CCh
004028af cc           ??       CCh

                 **************************************************************
                 *                          FUNCTION                          *
                 **************************************************************
                 undefined __fastcall atmel_set_fuse(int * param_1, undef…
     undefined         AL:1           <RETURN>
     int *             ECX:4          param_1
     undefined         DL:1           param_2
     ushort            Stack[0x4]:2   param_3                        XREF[3]:     00402980(R),
                                                                                  004029ba(R),
                                                                                  00402afa(R)
     undefined4        Stack[-0x8]:4  local_8                        XREF[6]:     004028bd(W),
                                                                                  004028ed(R),
                                                                                  0040293c(R),
                                                                                  0040295f(R),
                                                                                  00402b30(R),
                                                                                  00402b71(R)
     undefined2        Stack[-0x24]:2 local_24                       XREF[1]:     004029d4(W)
     undefined2        Stack[-0x26]:2 local_26                       XREF[2]:     00402995(*),
                                                                                  004029ce(W)
     undefined2        Stack[-0x28]:2 local_28                       XREF[5]:     00402983(*),
```

```c
void __fastcall atmel_set_fuse(int *param_1,undefined par

{
  char cVar1;
  FILE *pFVar2;
  int iVar3;
  byte bVar4;
  int iVar5;
  ushort *puVar6;
  char *_Format;
  undefined local_4c [8];
  int local_44;
  int local_40;
  ushort *local_2c;
  ushort local_28;
  ushort local_26;
  ushort local_24;
  uint local_8;

  local_8 = DAT_0040e000 ^ (uint)&stack0xfffffffc;
  if (param_1 == (int *)0x0) {
    FUN_00408470("atmel.c","atmel_set_fuse",0x20d,0x32,"
    FUN_004084b9();
    return;
  }
  if ((*(byte *)(param_1 + 2) & 4) == 0) {
    FUN_00408470("atmel.c","atmel_set_fuse",0x212,0x32,"
    _Format = "target does not support fuse operation.\n
    goto LAB_00402924;
  }
  iVar3 = atmel_select_memory_unit(param_1,3);
  if (iVar3 != 0) {
    FUN_004084b9();
    return;
  }
  switch(param_2) {
```

```
From: 00402adc To: 00408470 Type: UNCONDITIONAL_CALL Op: 0 DEFAULT
00402adc
From: 00402c52 To: 00408470 Type: UNCONDITIONAL_CALL Op: 0 DEFAULT
00402c52
From: 00402c94 To: 00408470 Type: UNCONDITIONAL_CALL Op: 0 DEFAULT
00402c94
()
```

```c
void __cdecl main(int argc,char **argv)
{
  _File = fopen(argv[1],"rb");
  if (_File == (FILE *)0x0) {
    printf("Not find file");
  }
  else {
    fseek(_File,0,2);
    uVar2 = ftell(_File);
    _DstBuf = malloc(-(uint)(0xffffffef < (uint)uVar2) | (uint)uVar2 + 0x10); //rounding
    _Str = malloc(((uint)uVar2 + 0x10) * 5);
    if (_DstBuf == (void *)0x0) {
      fclose(_File);
      printf("Not get space");
    }
    else { /*Doing some interesting stuff!*/ }
      //Writing file routines...
      printf("Write done!");
    }
  }
}
```

```
fread(_DstBuf,(uint)uVar2,1,_File);
local_40 = uVar2;
if ((uVar2 & 0xf) != 0) { //padding
  while (local_40 < uVar2 + (0x10 - ((uint)uVar2 & 0x8000000f))) {
    *(undefined *)((int)_DstBuf + (uint)local_40) = 0;
    local_40 = local_40 + 1;
  }
}
uVar2 = uVar2 + (0x10 - (uVar2 & 0xf));
iVar3 = thunk_FUN_00414870((uint *)&DAT_00420138,(byte *)"designed by dxls",0x80);
counter._0_2_ = 0;
while ((uint)(ushort)counter < (uint)((int)(uint)uVar2 >> 4)) {
  thunk_FUN_00415800(counter * 0x10 + _DstBuf),
                     counter * 0x10 + '-',0x10);
  aes_operation((uint *)&DAT_00420138,iVar3,
                (byte *)((uint)(ushort)counter * 0x10 + (int)_DstBuf),
                (undefined *)((uint)(ushort)counter * 0x10 + (int)_DstBuf));
  counter._0_2_ = (ushort)counter + 1;
}
```

Reversing: `Createbin.exe`

Did we... did we just find the AES key?

```
>>> len("designed by dxls")
16
```

It was that easy! Just need to find the algorithm used. At that point, happy, I wrote a python script that would try all modes of AES and compare the output.

Reversing: `Createbin.exe`

Nothing matches...

Back to the drawing board. We didn't investigate what this function does:

```
thunk_FUN_00415800(counter * 0x10 + _DstBuf),
                           counter * 0x10 + '-',0x10);
```

Reversing: `Createbin.exe`

```
void __cdecl FUN_00415800(char *param_1,char xor_byte,int size)
{
  counter = 0;
  while (counter < size) {
    tmp = (xor_byte + counter) ^ param_1[counter];
    param_1[counter] = tmp;
    counter = counter + 1;
  }
  return;
}
```

So... they single byte `xor` with a rolling counter... Let's integrate it!

Reversing: `Createbin.exe`

Nothing matches **again** 🤬💢...

I went from happy to sad in a few hours.

We'll have to keep digging deeper. Let's look at `thunk_FUN_00414870`

```c
void FUN_00414870(uint *param_1,byte *param_2,int param_3)
{
  local_18 = thunk_FUN_00414da0(param_1,param_2,param_3);
  local_24 = 0;
  local_30 = local_18 << 2;
  while (local_24 < local_30) {
    uVar1 = param_1[local_24];
    param_1[local_24] = param_1[local_30];
    param_1[local_30] = uVar1;
    uVar1 = param_1[local_24 + 1];
    param_1[local_24 + 1] = param_1[local_30 + 1];
    param_1[local_30 + 1] = uVar1;
    uVar1 = param_1[local_24 + 2];
    param_1[local_24 + 2] = param_1[local_30 + 2];
    param_1[local_30 + 2] = uVar1;
    local_3c = param_1[local_24 + 3];
    param_1[local_24 + 3] = param_1[local_30 + 3];
    param_1[local_30 + 3] = local_3c;
    local_24 = local_24 + 4;
    local_30 = local_30 + -4;
  }
```

```
int AES_set_encrypt_key(const unsigned char *userKey, const int bits,
                        AES_KEY *key)
{
    u32 *rk;
    int i = 0;
    u32 temp;
    if (!userKey || !key)
        return -1;
    if (bits != 128 && bits != 192 && bits != 256)
        return -2;
    rk = key->rd_key;
    if (bits==128)
        key->rounds = 10;
    else if (bits==192)
        key->rounds = 12;
    else
        key->rounds = 14;
```

```c
int AES_set_decrypt_key(const unsigned char *userKey, const int bits,
                        AES_KEY *key)
{

    u32 *rk;
    int i, j, status;
    u32 temp;

    /* first, start with an encryption schedule */
    status = AES_set_encrypt_key(userKey, bits, key);
    if (status < 0)
        return status;

    rk = key->rd_key;

    /* invert the order of the round keys: */
    for (i = 0, j = 4*(key->rounds); i < j; i += 4, j -= 4) {
        temp = rk[i    ]; rk[i    ] = rk[j    ]; rk[j    ] = temp;
        temp = rk[i + 1]; rk[i + 1] = rk[j + 1]; rk[j + 1] = temp;
        temp = rk[i + 2]; rk[i + 2] = rk[j + 2]; rk[j + 2] = temp;
        temp = rk[i + 3]; rk[i + 3] = rk[j + 3]; rk[j + 3] = temp;
    }
```

The solution

It was setting a **decryption** key all along!
The whole program can be rewritten as the following python script:

```
def createbin(file_inp, file_out):
    data_inp = file_inp.read()
    for i in range(0,len(data_inp),16):
        aes = AES.new(b'designed by dxls', AES.MODE_CBC, '\0'*16)
        block = data_inp[i:i+16]
        scrambled = [(block[j] ^ ((0x2d + i + j)&0xff)) for j in range(16)]
        out = aes.decrypt(bytes(scrambled))
        file_out.write(out)
```

Moving to `BOOT_LOADER_EXE.exe`

- Use the same way to decompile
- Exe is basically a recompile of `dfu-programmer`
- The strings are the same, so you can take https://github.com/dfu-programmer/dfu-programmer/blob/master/src/commands.c and rename all functions

`execute_flash` function

```
iVar2 = intel_hex_to_buffer(local_2c,uVar4,uVar1);
if (iVar2 == 0) {
  iVar2 = FUN_004069d0(*(char **)(param_2 + 0x58),(int)local_2c);
  pcVar5 = fprintf_exref;
  if (-1 < iVar2) {
    pcVar6 = __iob_func_exref;
    if (0 < iVar2) {
      DEBUG("commands.c","execute_flash",0x108,0x28,
            "WARNING: File contains 0x%X bytes outside target memory.\n");
      if (local_30 == 0) {
        DEBUG("commands.c","execute_flash",0x10b,0x28,
              "There may be data in the user page (offset %#X).\n");
        DEBUG("commands.c","execute_flash",0x10c,0x28,"Inspect the hex file or try flash-user.\n")
        ;
      }
```

There is a quite suspicious `FUN_004069d0` function...

The complete solution

`FUN_004069d0` is AES *encryption*. But it does not removes the rolling `xor`!

In summary

- `Createbin` xor the buffer and *decrypts* it with AES.
- `BOOTLOADER` *encrypts* (therefore decrypts it) with AES and uploads it.
- Bootloader undoes the rolling `xor`

The bootloader

We have 3 versions of bootloader compatible with the device:

- `atxmega32a4u_104`: 'Original' ATMEL bootloader
- `RevE-atxmega32a4u_104_modified.bin`: ATMEL bootloader but with correct PINs for buttons
- `ChameleonMiniRDV2.0_ATxmega32A4U`: Factory driver

Inside the bootloader

Ghidra is very good, can even decompile AVR code

```
void FUN_code_000318(undefined4 uParm1,undefined2 uParm2)
{
  sVar6 = 0;
  while( true ) {
    bVar5 = (byte)sVar6;
    cVar7 = (char)((ushort)sVar6 >> 8);
    if ((char)(cVar4 + (bVar5 < bVar3)) <= cVar7) break;
    pbVar8 = (byte *)CONCAT11((char)((ushort)uVar1 >> 8) + cVar7 + CARRY1((byte)uVar1,bVar5),
                              (byte)uVar1 + bVar5);
    *pbVar8 = *pbVar8 ^ cVar2 + bVar5;
    sVar6 = sVar6 + 1;
  }
}
```

What we can do now

We are able to:

- Build a firmware ✓
- Upload a firmware ✓
- Get rid of those pesky executable files ✓
- Work from Windows, Linux, MacOS ✓

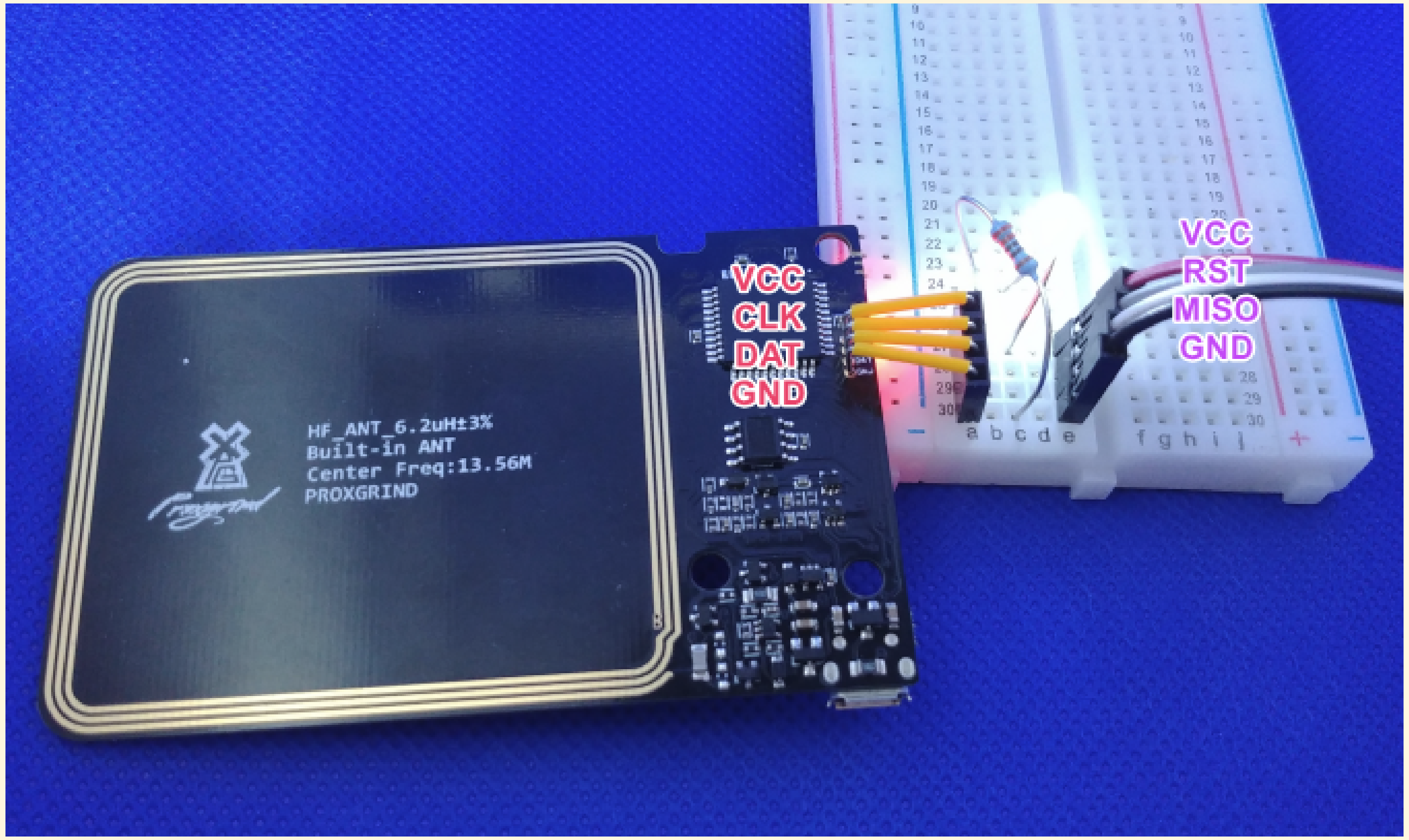
How to handle errors

Mistakes happened. While reversing, I tried to upload "prepared" `.hex` files to the target. However, I would upload a "wrong" file now and then, thus soft bricking my unit😟.

The only solution is to use a hardware writer with a setup such as an AVRISP mkII.

You can program application data, or, if you reset the whole chip, even reupload a new bootloader.

Hardware setup



Bonus 🎁

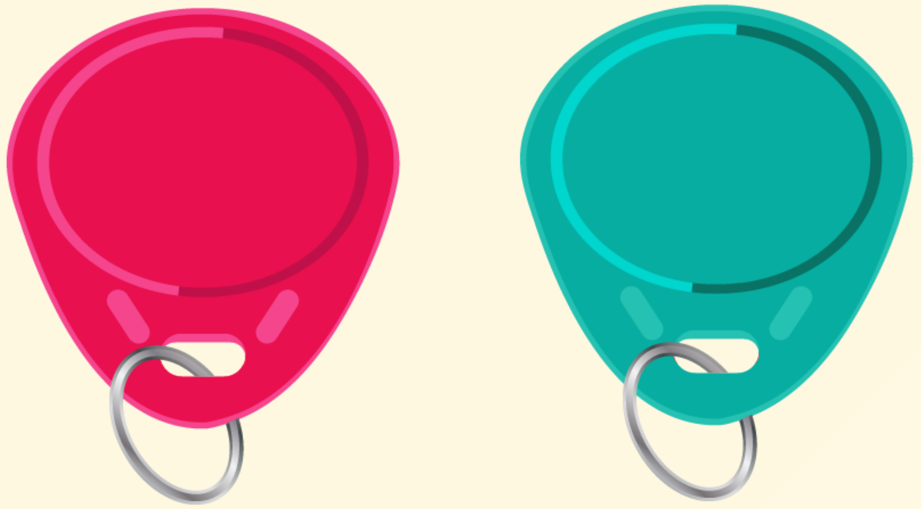
```python
loc = 0x408470 #debug print function
refs = getReferencesTo(toAddr(loc))
for r in refs:
        callee = r.getFromAddress()
        inst = getInstructionAt(callee)
        inst = getInstructionBefore(inst) #C file name push
        inst = getInstructionBefore(inst) #Function name push
        pushaddr = toAddr(inst.getDefaultOperandRepresentation(0))
        if pushaddr > 0x408000: #Simple filter
                func = getFunctionBefore(inst.getAddress())
                if (func.getName().startswith("FUN_")): #Don't rename twice
                        newname = getDataAt(pushaddr).getValue()
                        print(func, "=>", newname)
                        func.setName(newname, ghidra.program.model.symbol.SourceType.USER_DEFINED)
```

Conclusion

- We have a pretty good insight on what's needed in order to create a complete open source solution for "unlocking" the device.
- If you can program the AVR directly, it's quite easy to upload the correct bootloader.
- Next steps
  - Convince a default `dfu-programmer` to program without verifying (half yes)
  - Upload a program that reprograms the bootloader
  - See if the SPM helper is present in the original bootloader.

🔍 The strange case of the half working NFC tags

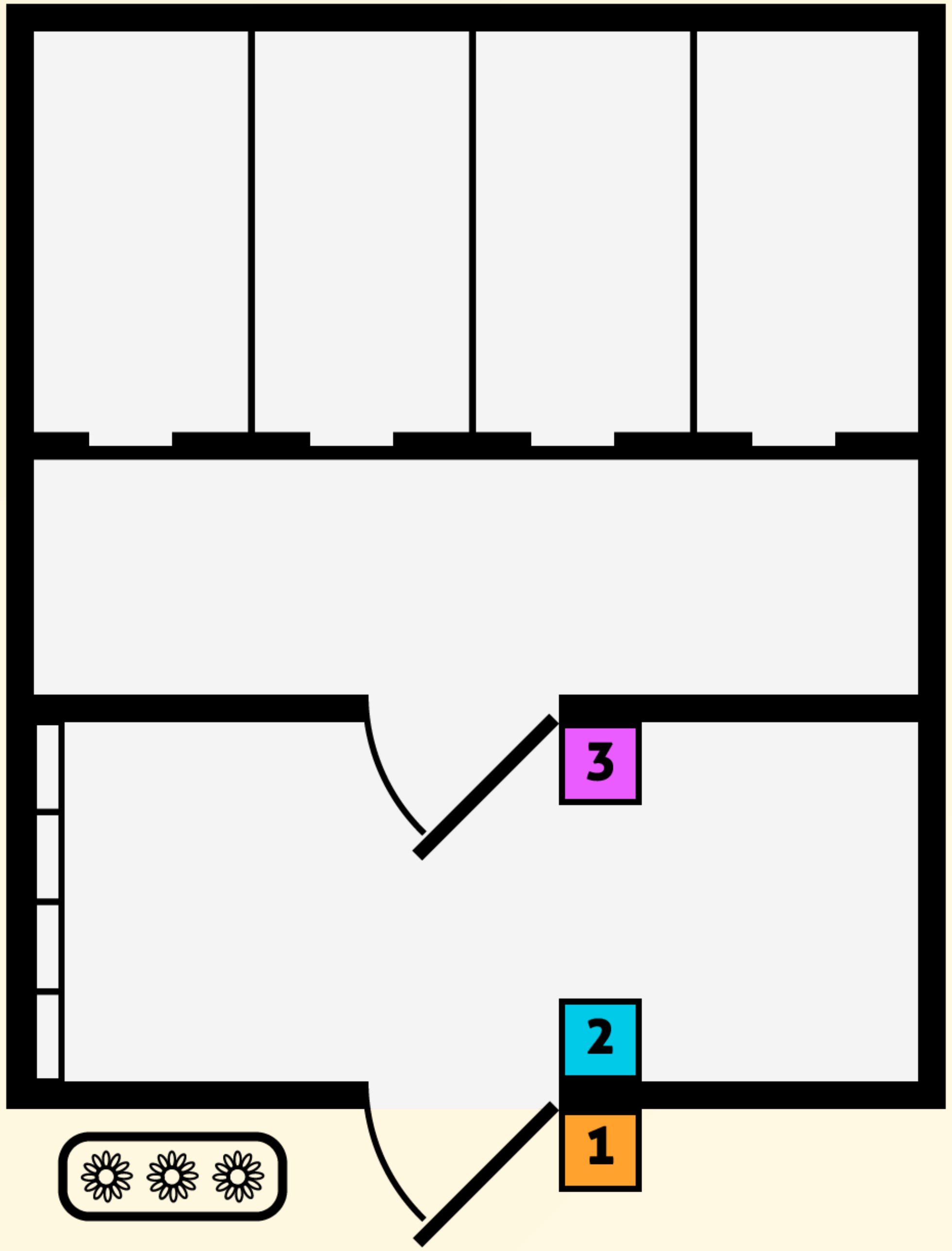
A little while ago, a friend had an apartment inside a larger structure. He wanted some additional tags so I was happy to make two clones of his two tags.
Let's call them red🏷 and green🏷.



Plan of the door locks

There are basically 3 readers:

- One to get to the complex from outside 1 (orange)
- One to get *out* from the complex 2 (cyan)
- One to get to get inside 3 (magenta)



A few month in

Everything works fine and my friend is happy! However, after some months, the clones worked... erratically.

Sometimes the red tag wouldn't work, or green, or both. And I was getting very strange reports.

Some times the door 2 would open, or the door 1, with different tags anytime. I would clone again the tags but the situation would become erratic very quickly.

So... what happened ?

Diffing to the rescue

I had the chance to be able to get to the location myself. Armed with proxmark and Mifare Classic Tool, I could finally crack the mystery.

When you approached a tag which was working on reader `1` or `2` (but not 3), it would *write* the tag and it would produce the following diff:
`AE05FF531F10` ⇒ `AE06FF531F10`
and in another location,
`..0A........` ⇒ `..09........`

But why the erratic behaviour

Turns out that the counter wasn't on 8 (or 16...) bits but only on 4 bits and would wrap around. That means, a clone with a wrong counter would work one time out of 16.

So, by pure luck tags would 'work', deactivate the others, etcetera...

Still doesn't explain why the tags would have worked perfectly before 🤔

Final take

By digging in my backups, I found an old dump and where the counter sector is, it was only `0`s...
The reader's software must have been upgraded!

But, as with all upgrades, it means you need to have an upgrade plan...
So you can craft a 'old' tag with only `0` and it would overwrite the sector and update the counter. But there is a feature of Mifare that specifies access rights😄

So... let's create a tag with only `0` in the sector and readonly on that sector. It actually half works! (Exit only)

Thank you/Questions

If you want to contact me:

- I'm here all 3 days! Come and chat!
- @slurdge on almost any platform/social network
- slurdge@slurdge.org

No emojis were harmed during the making of this presentation