Deck Ext-Cmos Unlock

What's needed to know about UEFI for this

- UI Element are organized in **form**, and each form is identified by a GUID

Let's See what we know

We know that
* Main, Advanced, Security, Power, Boot and Exit are the "shown" form, and PBS and CBS are the hidden form

let's get the GUID of these one

Tool Needed:
* UEFI Tool
* Bios Image of the SD
* ifrextractor-rs

Let's open the SD Bios Image in UEFI Tool and Search for AMD CBS

the only match will be on CBSSetupDxe and make sense

Right click on the PE32 Image and extract the Body
do the same with PBS

Now let’s find the other one…
Let’s use for example the Advanced one

Search for advanced:
there will be more Result, but the only one that make sense, is the one in “SetupUtility”, as the other have little correlation to the things we are trying to do..

Extract also SetupUtility as Body..

* Drag every Extracted File on the ifrextractor.exe file (If you are using linux execute ./ifrextractor file_name)

There will created a bunch of txt file that contain the description of the Menu:

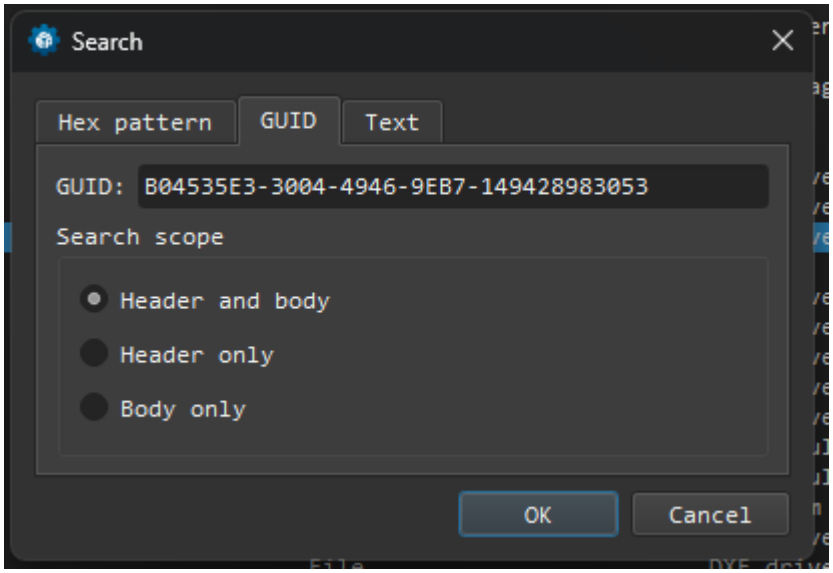
We are only interested in the GUID
Take for Example CBS, we open it’s txt and we get

```
Program version: 1.5.1, Extraction mode: UEFI
FormSet Guid: B04535E3-3004-4946-9EB7-149428983053, Title: "AMD CBS", Help: "AMD CBS Setup Page"
```

So we can Annotate, and do for all
AMD CBS : B04535E3-3004-4946-9EB7-149428983053
AMD PBS : B863B959-0EC6-4033-99C1-8FD89F040222
Power : A6712873-925F-46C6-90B4-A40F86A0917B
Advanced : C6D4769E-7F48-4D2A-98E9-87ADCCF35CCC
and so on

we now have the GUID, and we have to find where is USED, and who if that guid is displayed

let’s do a GUID search for CBS



Only 3 Result Nice:

- H2OFormBrowser
- SetupUtilityApp
- CBSSetupDXE, we can ignore this, of couse has a reference to it, we got the GUID from that module

Let's do for PBS
Only 3 Result Nice:

- H2OFormBrowserDxe
- SetupUtilityApp
- PBSSetupDXE, we can ignore this, of couse has a reference to it, we got the GUID from that module

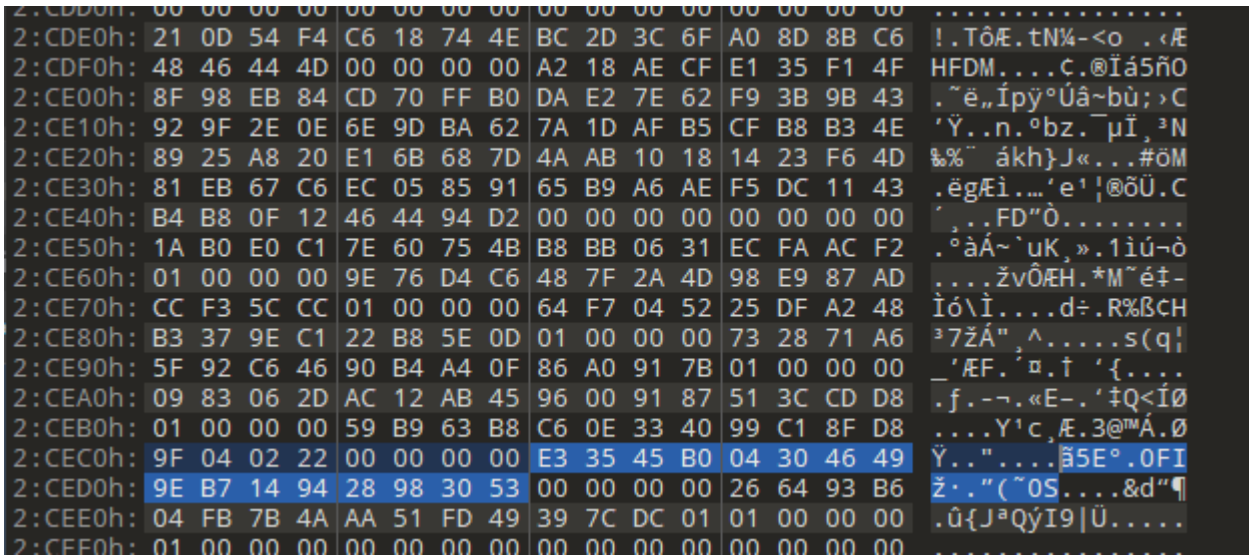
Ok, so our target is either H2OFormBrowser or SetupUtilityApp

here I cheat, and for experience I know that the FormBrowser is the target to start with, if we are wrong we will try to analyze the SetupUtilityApp.

Let's extract as body H2OFormBrowserDxe and as starting point let's open in a Hex editor to see if there's any recognizable pattern.

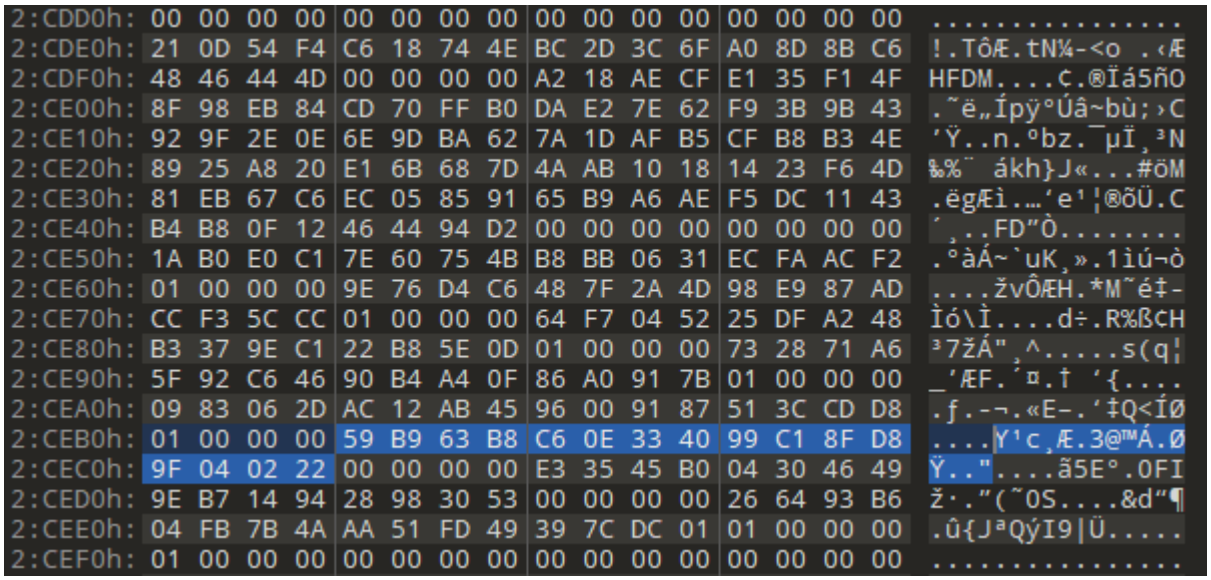
get the CBS GUID:
B04535E3-3004-4946-9EB7-149428983053
and convert to the little-Endian Form (use for example https://robobunny.com/cgi-bin/guid) if you don't want to do by hand

we get E33545B0043046499EB7149428983053, lets search for it in the hex editor

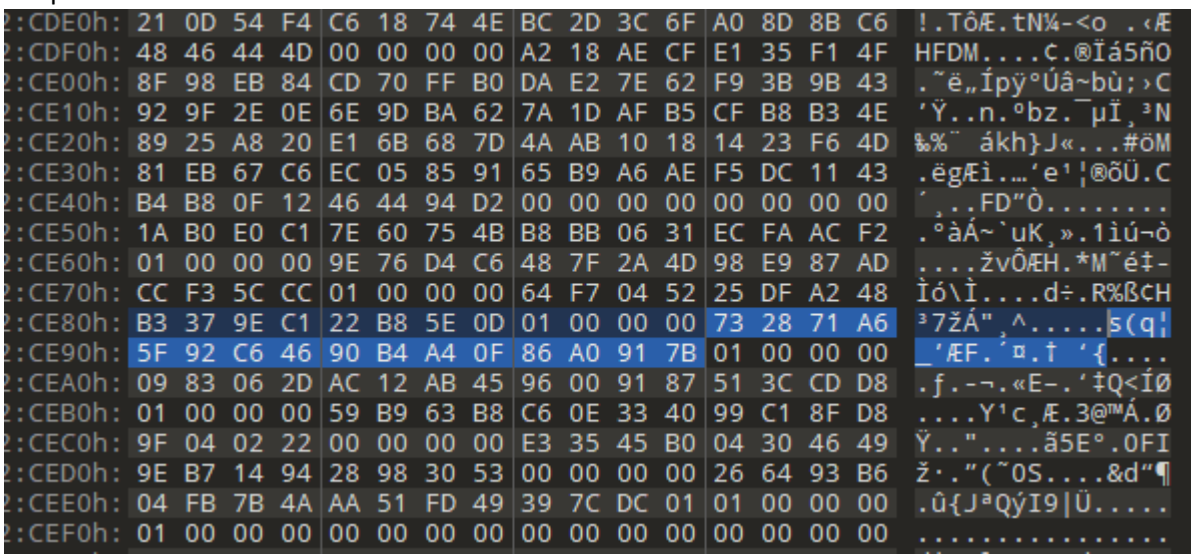


Perfect one match, in what is seem random data…

Let’s do for PBS



For power



you can do for all the other ..

interesting all the GUID that can be shown are all in the same place..

If we look carefull the pattern seem to be

4 Byte + Guid, again 4 Byte + GUID etc…
from these 4 byte the last 3 are always zero….

Let’s look at the power tab
is
01 00 00 00 POWER_GUID

CBS is
00 00 00 00 CBS_GUID

Oh, then, we can safely guess that the first byte is “Enabled”/is shown bit.

Now that we understood how the hide it, let’s see if exist a switch that flip that byte to 1..

Let’s get a Copy of Ghidra (For this not even any Plugin is required) :

Import into Ghidra and do a standard auto analysis…

We want to see if that bit is set from somewhere to 1

so let’s search again for E33545B0043046499EB7149428983053, press S and
input E3 35 45 B0 04 30 46 49 9E B7 14 94 28 98 30 53

Oh, nice one match…

```
                    DAT_0002cec4                                    XREF[1]:     FUN_00028dc8:00028e72(W)
0002cec4 00             ??          00h
0002cec5 00             ??          00h
0002cec6 00             ??          00h
0002cec7 00             ??          00h
0002cec8 e3             ??          E3h
0002cec9 35             ??          35h    5
0002ceca 45             ??          45h    E
0002cecb b0             ??          B0h
0002cecc 04             ??          04h
0002cecd 30             ??          30h    0
0002cece 46             ??          46h    F
0002cecf 49             ??          49h    I
0002ced0 9e             ??          9Eh
0002ced1 b7             ??          B7h
0002ced2 14             ??          14h
0002ced3 94             ??          94h
0002ced4 28             ??          28h    (
0002ced5 98             ??          98h
0002ced6 30             ??          30h    0
0002ced7 53             ??          53h    S
```

Even more nice, is that Ghidra identified a Function that write to it..

right click to DAT_0002cec4, and do Reference -> Show Referce to Address

double click on the Result and will jump to the address, ignore the Listing and focus on the decompiled code..:

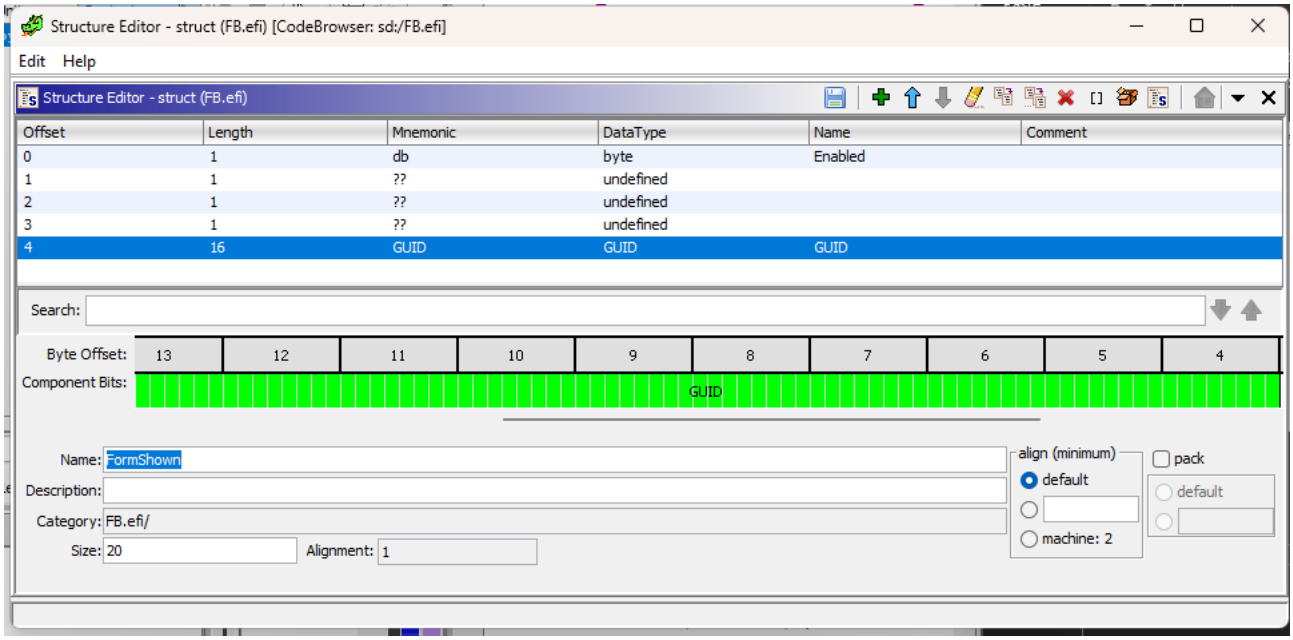
```
33      if (puVar4 != (undefined8 *)0x0) {
34        uVar5 = FUN_00029470();
35        uVar9 = 0;
36        uVar15 = 0;
37        do {
38          uVar10 = uVar9;
39          if (uVar9 < uVar12) {
40            do {
41              if ((char)uVar5 == 'w') {
42                DAT_0002cec4 = 1;
43                DAT_0002ced8 = 1;
44              }
45              lVar3 = puVar4[uVar10];
```

That sound promising
let’s make a bit more readable with the info we discovered earlier..
In the data Type Manager right click on the File Name-> New Structure

we know that we have 20 byte (4 byte +16 of GUID), the first is the enable bit, 3 padding and then the GUID

and let’s call it FormShown..

will lock something like this



save and go back to the decompiled view, right click on 0002cec4, do Retype and assing the just created FormShown

already start to look better

```
if (puVar4 != (undefined8 *)0x0) {
  uVar5 = FUN_00029470();
  uVar9 = 0;
  uVar15 = 0;
  do {
    uVar10 = uVar9;
    if (uVar9 < uVar12) {
      do {
        if ((char)uVar5 == 'w') {
          FormShown_0002cec4.Enabled = 1;
          DAT_0002ced8 = 1;
        }
```

Rename it to CBS, do the same for PBS and you end with

```
uVar5 = FUN_00029470();
uVar9 = 0;
uVar15 = 0;
do {
  uVar10 = uVar9;
  if (uVar9 < uVar12) {
    do {
      if ((char)uVar5 == 'w') {
        CBS.Enabled = 1;
        PBS.Enabled = 1;
      }
```

we are almost there, we need to find when uVar5 is set to ‘w’

if you don’t like ascii you can right click and select hexadecimal

```
uVar5 = FUN_00029470();
uVar9 = 0;
uVar15 = 0;
do {
  uVar10 = uVar9;
  if (uVar9 < uVar12) {
    do {
      if ((char)uVar5 == 0x77) {
        CBS.Enabled = 1;
        PBS.Enabled = 1;
      }
```

the only thing left is understand what FUN_00029470() does

```
ulonglong FUN_00029470(void)

{
  byte bVar1;
  ushort uVar2;
  ulonglong in_RAX;
  ulonglong uVar3;

  uVar2 = 0;
  if (_DAT_0002dfa8 == 0) {
    uVar2 = FUN_00000344();
    in_RAX = FUN_00000340();
  }
  out(0x72,0xf7);
  bVar1 = in(0x73);
  uVar3 = in_RAX & 0xffffffffffffff00;
  if ((_DAT_0002dfa8 == 0) && (uVar3 = 0x200, (uVar2 & 0x200) != 0)) {
    uVar3 = FUN_00000342();
  }
  return uVar3 & 0xffffffffffffff00 | (ulonglong)bVar1;
}

```

Now I'm bored so let's skip to the juicy part

```
out(0x72,0xf7);

bVar1 = in(0x73);
```

and
return uVar3 & 0xffffffffffffff00 | (ulonglong)bVar1;

here you need a bit of knowledge of in and out instruction but a quick google search will point out at I/O port and you can see that is
using the port 0x72/0x73 and all that this function do is read offset F7 of that io device…

SD Unlocked