

## Debugging MBR - IDA + Bochs Emulator (CTF example)

This post will explain how to setup Bochs Emulator to debug MBR in IDA environment on Windows OS. It could be quite useful if we are struggling with some kind of MBR Locker/Wiper or another kind of MBR modifier where we have to RE the bootstrap code.

A master boot record (MBR) is a special type of boot sector at the very beginning of partitioned computer mass storage devices like fixed disks or removable drives. The MBR consists of 512 bytes located in the first sector of the drive.

We are especially interested in analyzing the Bootstrap code area which starts at file offset 0x0000.

Structure of a classical generic MBR			
Address	Description		Size (bytes)
0x0000 (0)	Bootstrap code area		446
0x01BE (446)	Partition entry №1	Partition table (for primary partitions)	16
0x01CE (462)	Partition entry №2		16
0x01DE (478)	Partition entry №3		16
0x01EE (494)	Partition entry №4		16
0x01FE (510)	0x55	Boot signature <sup>[2]</sup>	2
0x01FF (511)	0xAA		
Total size: 446 + 4×16 + 2			512

First of all, we could be in situation where we have the infected drive or already carved out MBR “bin file”. If we have the infected drive, we could carve out the first 512 bytes of it with some Hex editor or Unix “dd” command-line utility etc...

As an example in this post, we will be using MBR from one of challenges CrowdStrike Intelligence Adversary Quest [\[CrowdStrike Intelligence Adversary Quest\]](#) CTF.

This challenge was called "The Proclamation". We simple got MBR bin file and have to obtain the right flag.

MBR to download from this challenge is available here [\[MBR sample\]](#).

## Installing Bochs Emulator

Download and install Bochs x86 PC emulator available here [\[Bochs x86 PC emulator\]](#). IDA is recommending version v2.6.x. The latest version now (30. 1. 2021) is 2.6.11 so this should be ok.

## Creating a Bochs Disk Image

Now we have to create Bochs disk image where we will inject our MBR sample. Open a new Administrator Command Prompt window.

In the Administrator Command Prompt window, navigate to install directory of Bochs emulator and run these commands to create Bochs Disk image:

```
cd C:\Program Files\Bochs-2.6.11  
bximage.exe
```

The "bximage" tool launches, as shown below.

Enter these items:

- **1** to create a new image
- **Enter** for "hd"
- **Enter** for "flat"
- **Enter** for 512 byte hard disk sectors
- **Enter** for size "10"
- **Enter** to accept the default name or rename to " my\_new\_image\_.img"
- **Enter** to continue and exit bximage

```
Administrator: Command Prompt

C:\Program Files\Bochs-2.6.11>bximage.exe
=====
                        bximage
  Disk Image Creation / Conversion / Resize and Commit Tool for Bochs
    $Id: bximage.cc 13481 2018-03-30 21:04:04Z vruppert $
=====

1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info

0. Quit

Please choose one [0] 1

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] hd

What kind of image should I create?
Please type flat, sparse, growing, vpc or vmware4. [flat] flat

Choose the size of hard disk sectors.
Please type 512, 1024 or 4096. [512] 512

Enter the hard disk size in megabytes, between 10 and 8257535
[10] 10

What should be the name of the image?
[c.img] my_new_image_.img

Creating hard disk image 'my_new_image_.img' with CHS=20/16/63 (sector size = 512)

The following line should appear in your bochsrc:
  ata0-master: type=disk, path="my_new_image_.img", mode=flat
(The line is stored in your windows clipboard, use CTRL-V to paste)

Press any key to continue

C:\Program Files\Bochs-2.6.11>_
```

## Creating the Configuration File

In the Administrator Command Prompt window, execute these commands:

**notepad bochsrc.bxrc**

A Notepad box pops up, asking if we want to create a new file.

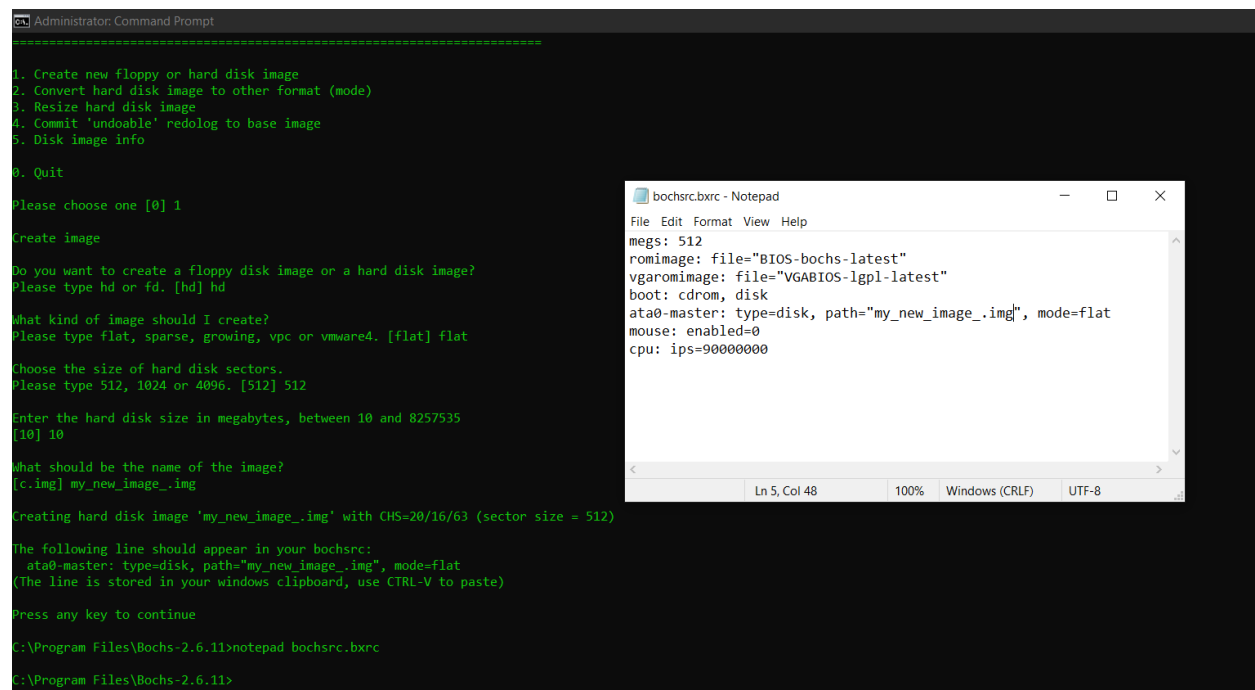
Click **Yes**.

Enter this text into Notepad, as shown below.

```
megs: 512
romimage: file="BIOS-bochs-latest"
vgaromimage: file="VGABIOS-lgpl-latest"
boot: cdrom, disk
ata0-master: type=disk, path="my_new_image_.img", mode=flat
mouse: enabled=0
cpu: ips=900000000
```

Be careful on the “path” parameter – must match the name we chase during image creation.

In Notepad, click **File, Save**. Close Notepad.



The screenshot shows a Windows command prompt window titled "Administrator: Command Prompt" and a Notepad window titled "bochsrc.bxrc - Notepad".

The command prompt shows the following commands and output:

```
1. Create new floppy or hard disk image
2. Convert hard disk image to other format (mode)
3. Resize hard disk image
4. Commit 'undoable' redolog to base image
5. Disk image info
0. Quit

Please choose one [0] 1

Create image

Do you want to create a floppy disk image or a hard disk image?
Please type hd or fd. [hd] hd

What kind of image should I create?
Please type flat, sparse, growing, vpc or vmware4. [flat] flat

Choose the size of hard disk sectors.
Please type 512, 1024 or 4096. [512] 512

Enter the hard disk size in megabytes, between 10 and 8257535
[10] 10

What should be the name of the image?
[c:\img] my_new_image_.img

Creating hard disk image 'my_new_image_.img' with CHS=20/16/63 (sector size = 512)

The following line should appear in your bochsrc:
  ata0-master: type=disk, path="my_new_image_.img", mode=flat
(The line is stored in your windows clipboard, use CTRL-V to paste)

Press any key to continue

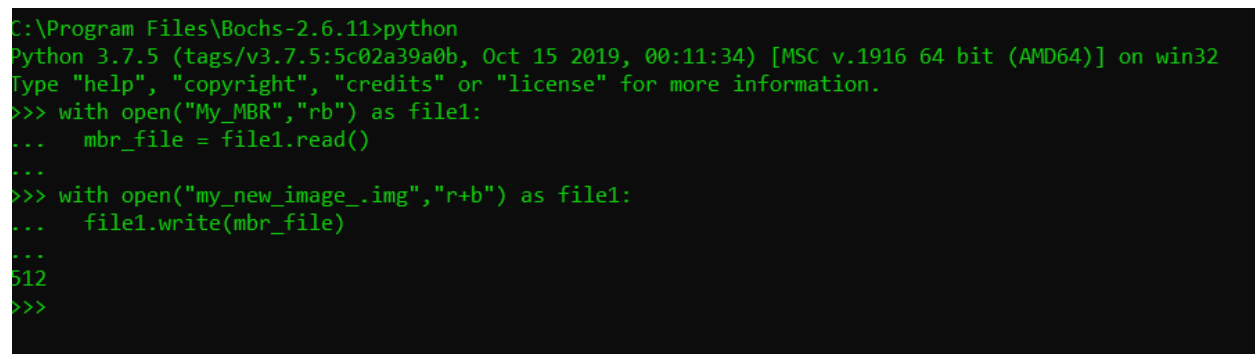
C:\Program Files\Bochs-2.6.11>notepad bochsrc.bxrc
C:\Program Files\Bochs-2.6.11>
```

The Notepad window shows the following content:

```
File Edit Format View Help
megs: 512
romimage: file="BIOS-bochs-latest"
vgaromimage: file="VGABIOS-lgpl-latest"
boot: cdrom, disk
ata0-master: type=disk, path="my_new_image_.img", mode=flat
mouse: enabled=0
cpu: ips=90000000
```

## Injecting MBR in newly created Bochs Disk Image

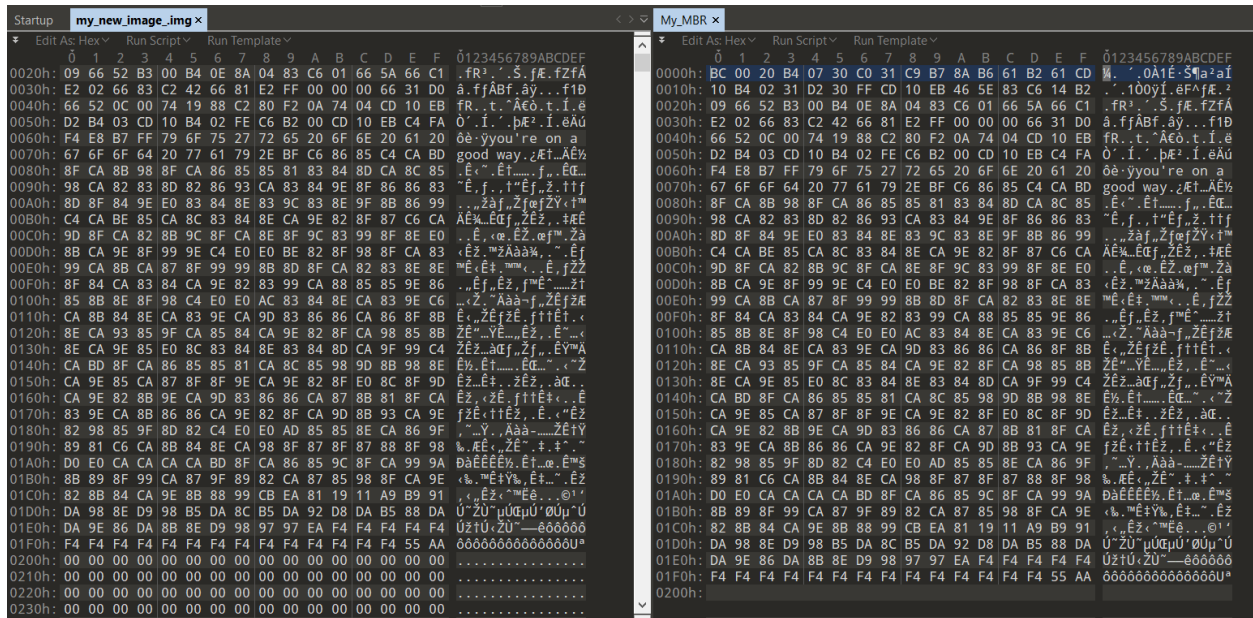
When we are injecting our MBR bin file we are simply replacing first 512 bytes of created Bochs image “my\_new\_image\_.img” with 512 bytes of our MBR. We can do this in any HEX editor or simply with python as you can see in the picture below.



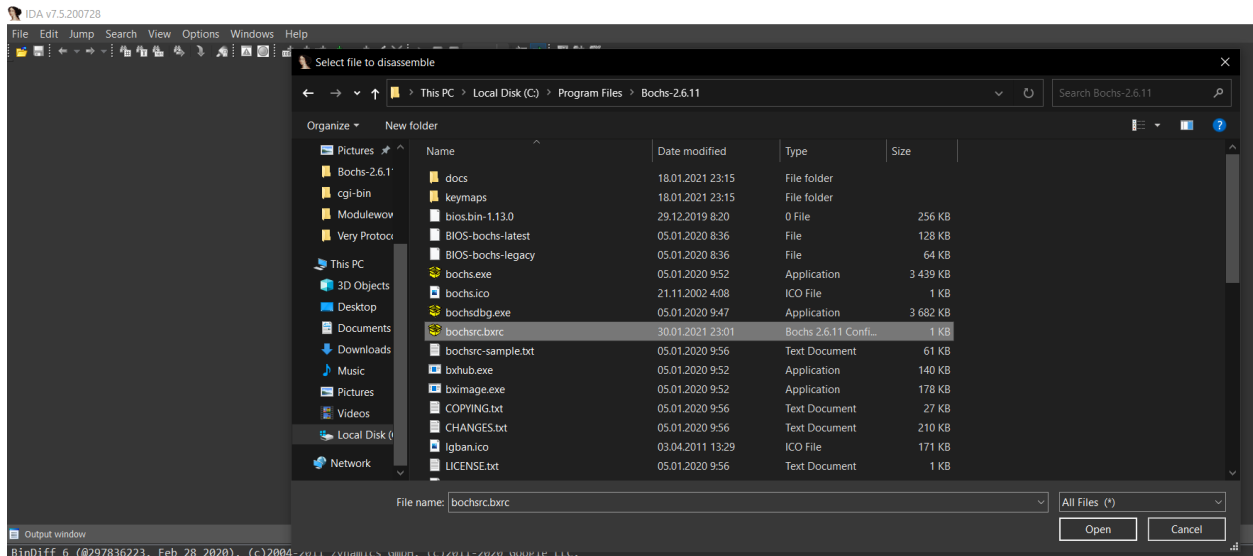
The screenshot shows a Windows command prompt window with the following commands and output:

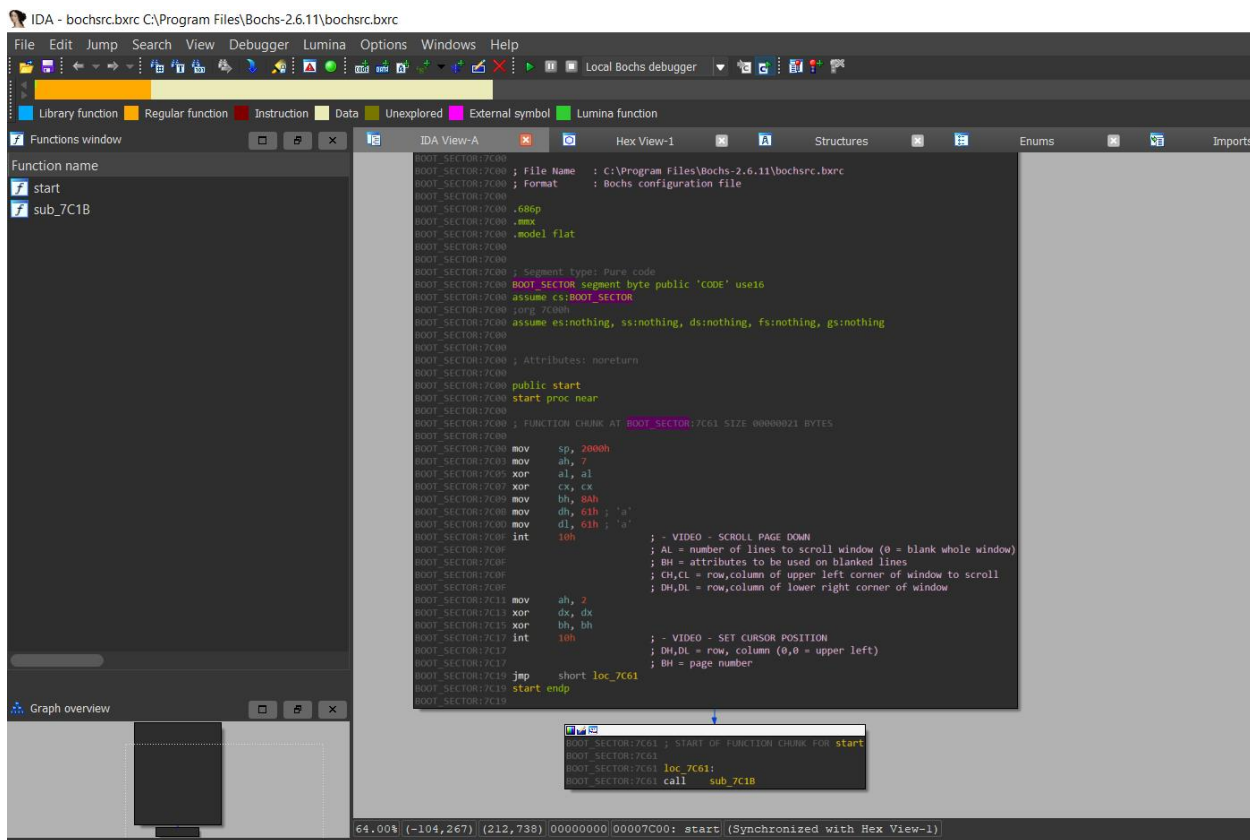
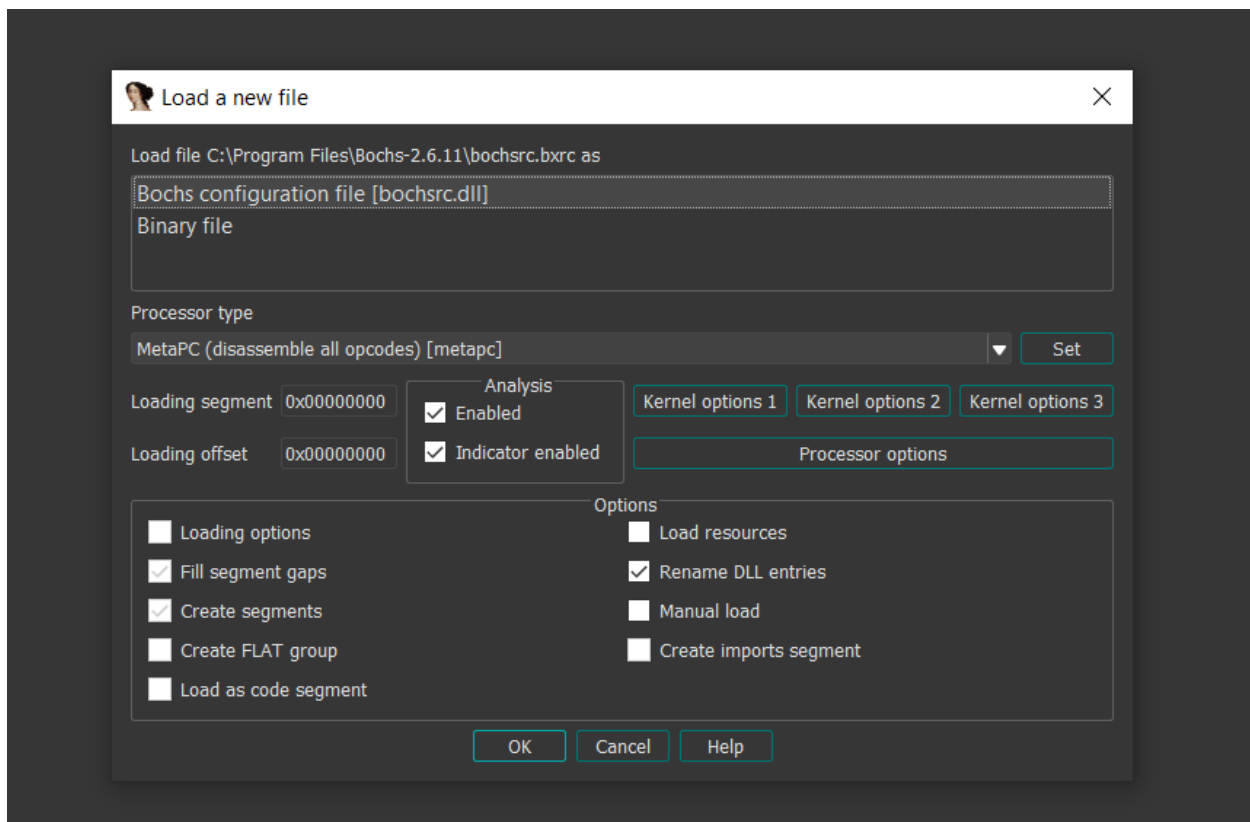
```
C:\Program Files\Bochs-2.6.11>python
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> with open("My_MBR", "rb") as file1:
...     mbr_file = file1.read()
...
>>> with open("my_new_image_.img", "r+b") as file1:
...     file1.write(mbr_file)
...
512
>>>
```

We can check that 512 bytes of MBR were really injected in Bochs Disk Image in Hex editor:

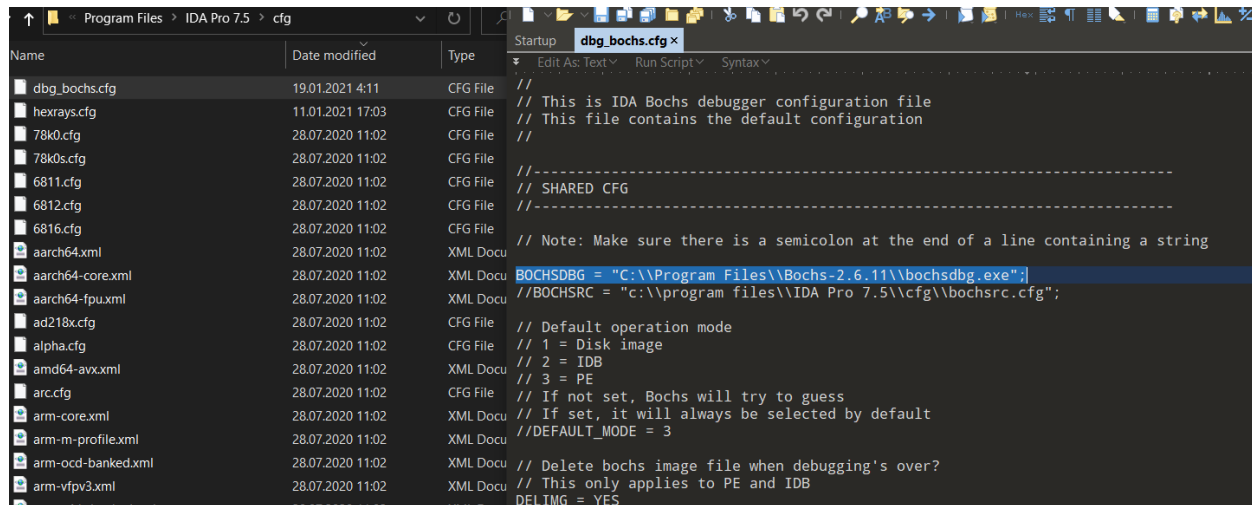


Opening previously created configuration file “bochsrc.bxrc” in IDA.





If IDA got a problem with locating the Bochs debugger “bochsdbg.exe”, check the dbg\_bochs.cfg in location (“IDA install directory\cfg\dbg\_bochs.cfg”) and modify the (BOCHSDBG = "C:\\Program Files\\Bochs-2.6.11\\bochsdbg.exe");) to point to the location where we installed the Bochs emulator.



We do not have to bother with the line “//BOCHSRC = "c:\\program files\\IDA Pro 7.5\\cfg\\bochsrc.cfg";” because we already gave IDA the Bochs configuration file as “bochsrc.bxrc”.

## Let’s start debugging

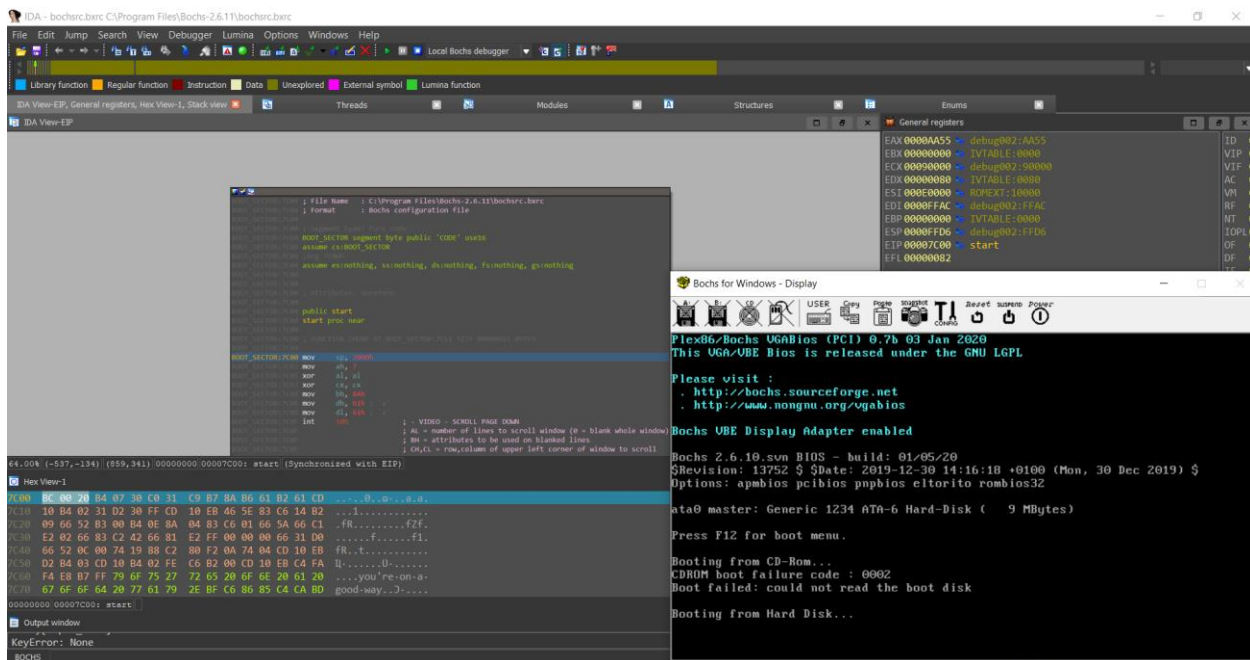
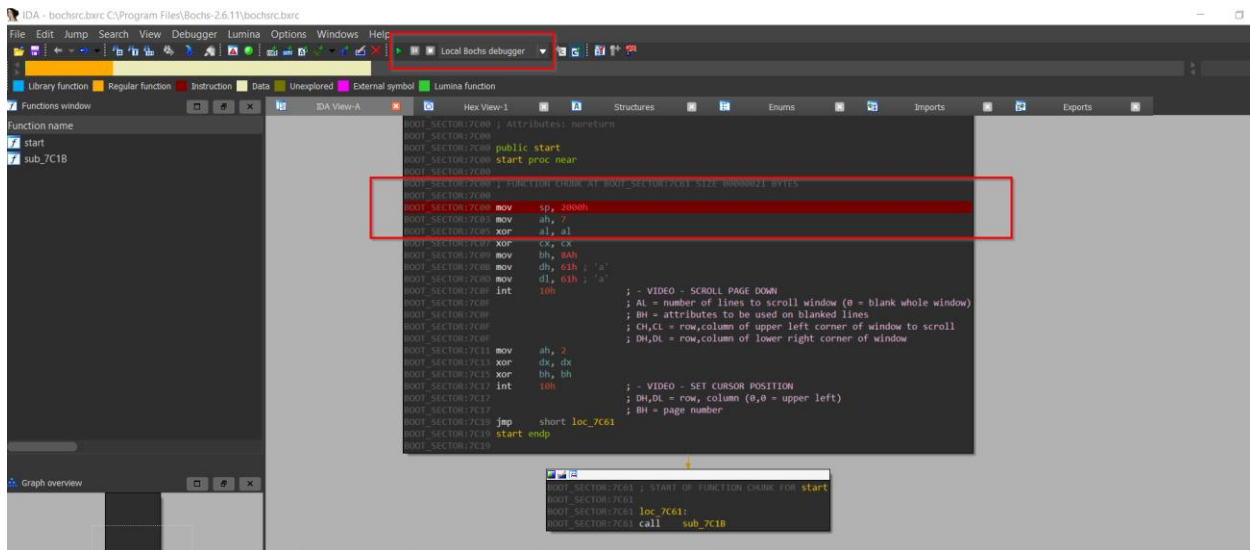
Put a breakpoint on the start address „BOOT\_SECTOR:7C00“. Don’t be surprised about the 0x7C00 address – little explanation:

A bootloader runs under certain conditions. The following pertains to bootloaders initiated by the PC BIOS:

- The first sector of a drive contains its boot loader.
- One sector is usually 512 bytes — the last two bytes must be 0xAA55 (i.e. 0x55 followed by 0xAA), or else the BIOS will treat the drive as unbootable.
- If everything is in order, said first sector will be placed at RAM address 0000:7C00, and the BIOS's role is over as it transfers control to 0000:7C00. (I.e. it Jumps to that address)

IDA sets almost everything for us automatically after opening “bochsrc.bxrc” file but in case we would not like to use the Bochs configuration file and want to analyze the bootstrap code of MBR alone just remember to load it as x86 16 bit LE code.

Start debugging:



Now we are in state of debugging bootstrap code of MBR so next section will be focusing on solving the challenge “The Proclamation” of CrowdStrike Intelligence Adversary Quest [\[CrowdStrike Intelligence Adversary Quest\]](#) CTF.

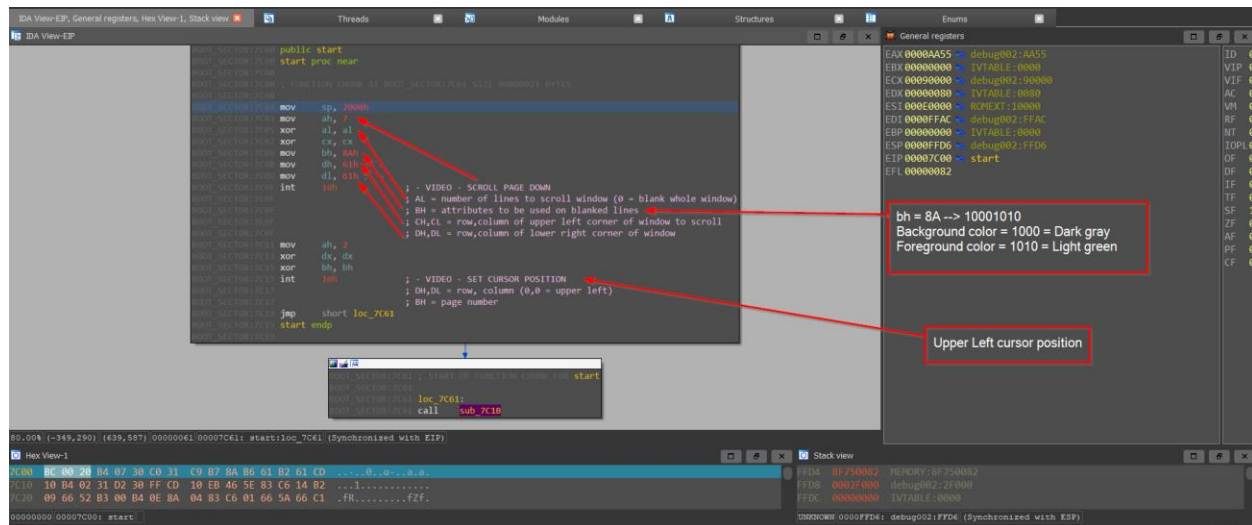
MBR to download from this challenge is available here [\[MBR sample\]](#).



## Analyzing and solving challenge “The Proclamation”

At first, when we reach breakpoint on the start address 0x7C00, we can see two interrupts and calling function sub\_7C1B.

These two interrupts are only setting cursor position and color of the background and foreground text which will be printed as it is described in the picture below.



IDA already helped us a lot with arguments description. But for sure, below is more information to INT 10h / AH = 07h

**INT 10h / AH = 06h** - scroll up window.  
**INT 10h / AH = 07h** - scroll down window.

*input:*

**AL** = number of lines by which to scroll (00h = clear entire window).

**BH** = attribute used to write blank lines at bottom of window.

**CH, CL** = row, column of window's upper left corner.

**DH, DL** = row, column of window's lower right corner.

For BH value as attribute is table explaining this value below.

**bit color table:**

**bit color table:**  
character attribute is 8 bit value, low 4 bits set fore color, high 4 bits set background color.

note: the emulator and windows command line prompt do not support background blinking, however to make colors look the same in dos and in full screen mode it is required to turn off the background blinking.

HEX	BIN	COLOR
0	0000	black
1	0001	blue
2	0010	green
3	0011	cyan
4	0100	red
5	0101	magenta
6	0110	brown
7	0111	light gray
8	1000	dark gray
9	1001	light blue
A	1010	light green
B	1011	light cyan
C	1100	light red
D	1101	light magenta
E	1110	yellow
F	1111	white

note:

```

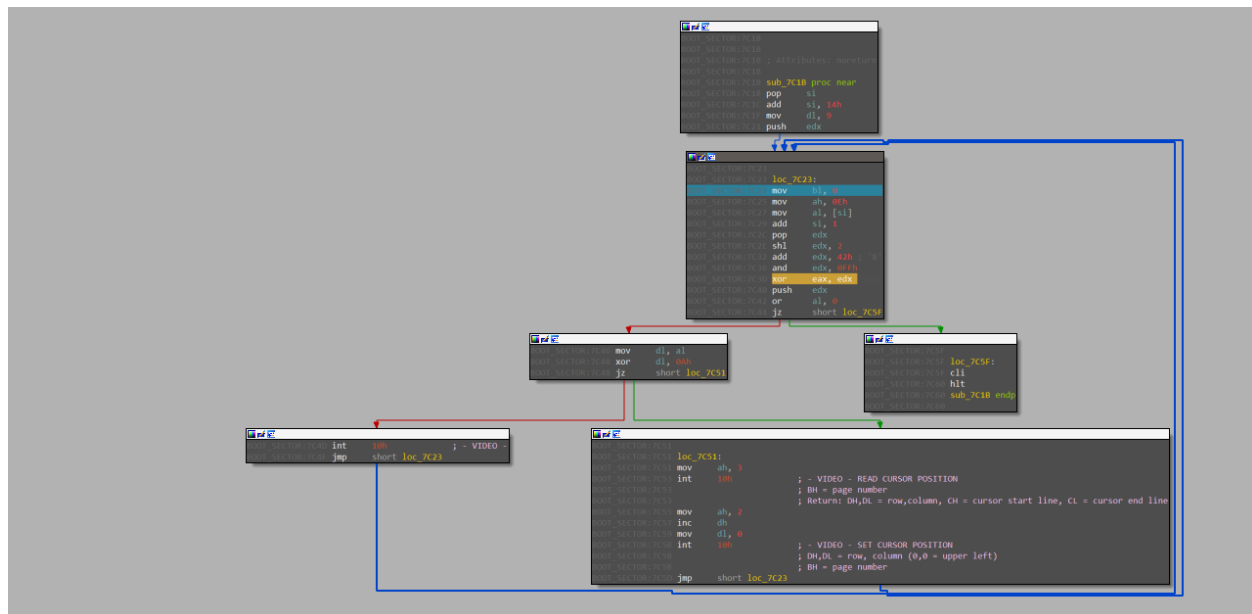
; use this code for compatibility with dos/cmd prompt full screen mode:

```

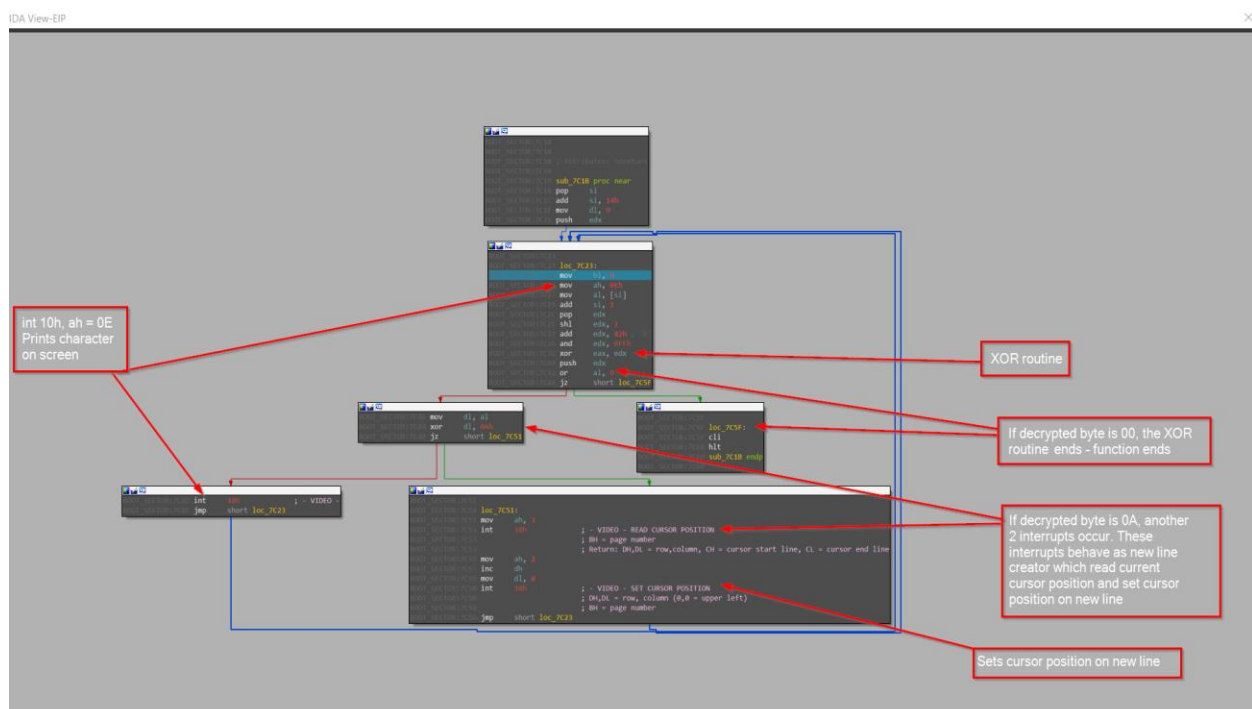
```
mov ax, 1003h
mov bx, 0 ; disable blinking.
int 10h
```

So the background color is set to Dark grey and foreground color is set to light green. Cursor position is upper left. Let's move on to analyzing function sub\_7C1B.

In sub\_7C1B we can immediately see some XOR routine.



This function is basically reading byte after byte from some offset and decrypt it via XOR routine. If decrypted byte is 00, the XOR routine ends – function ends. If decrypted byte is 0A, another 2 interrupts occur. These interrupts behave as new line creator which read current cursor position and set cursor position on new line. If decrypted byte is not 00 or 0A, another interrupt occurs – int 10h, ah = 0E → Prints character on screen (this interrupt is described below). All this routine we can see annotated:



Description “int 10h, ah = 0E”:

INT 10h / AH = 0Eh - teletype output.

input:

AL = character to write.

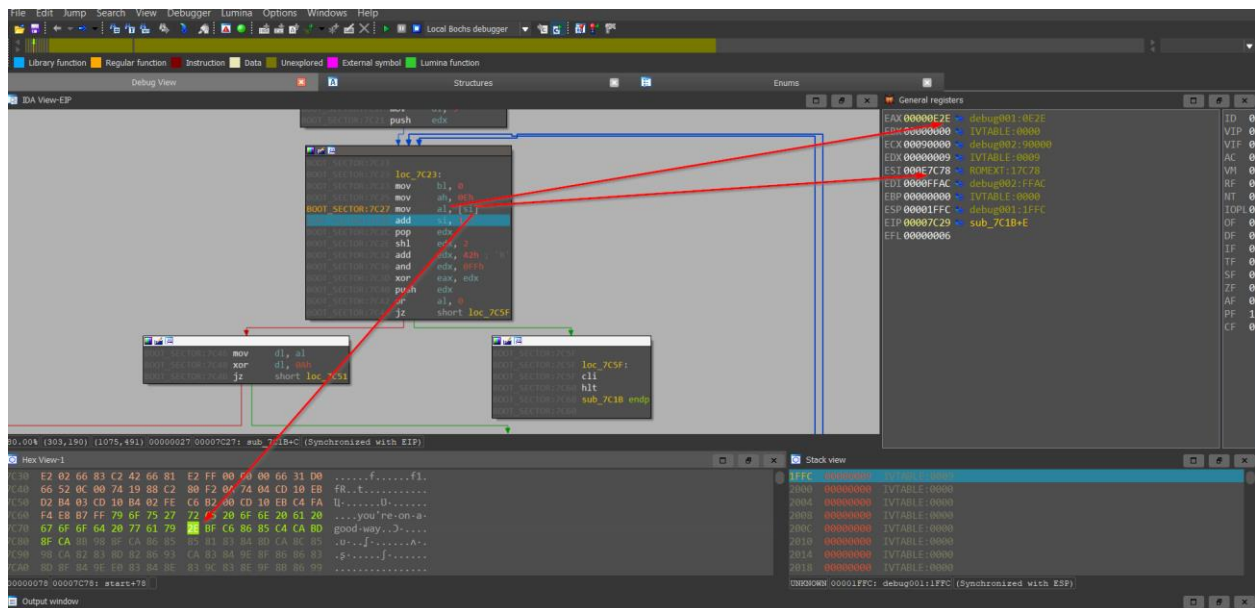
this functions displays a character on the screen, advancing the cursor and scrolling the screen as necessary. the printing is always done to current active page.

example:

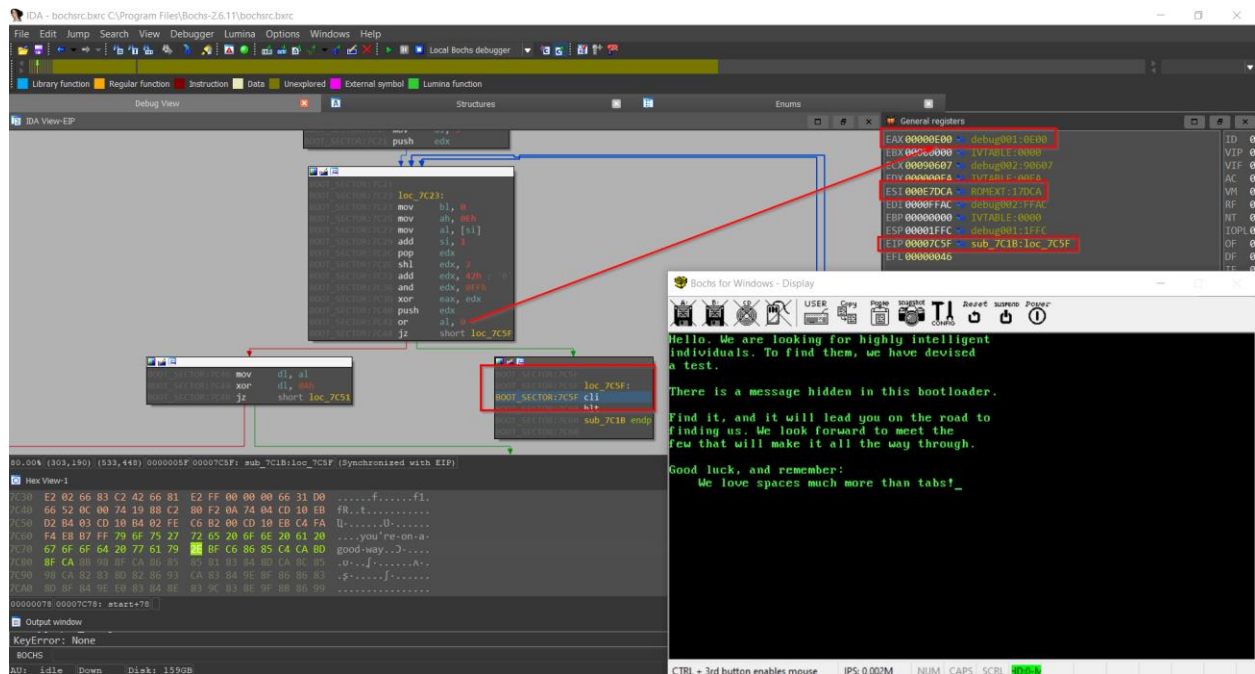
```
mov al, 'a'  
mov ah, 0eh  
int 10h
```

; note: on specific systems this  
; function may not be supported in graphics mode.

We can clearly see that first byte which will be decrypted on address 0x7C78, the value of this byte is 2E:



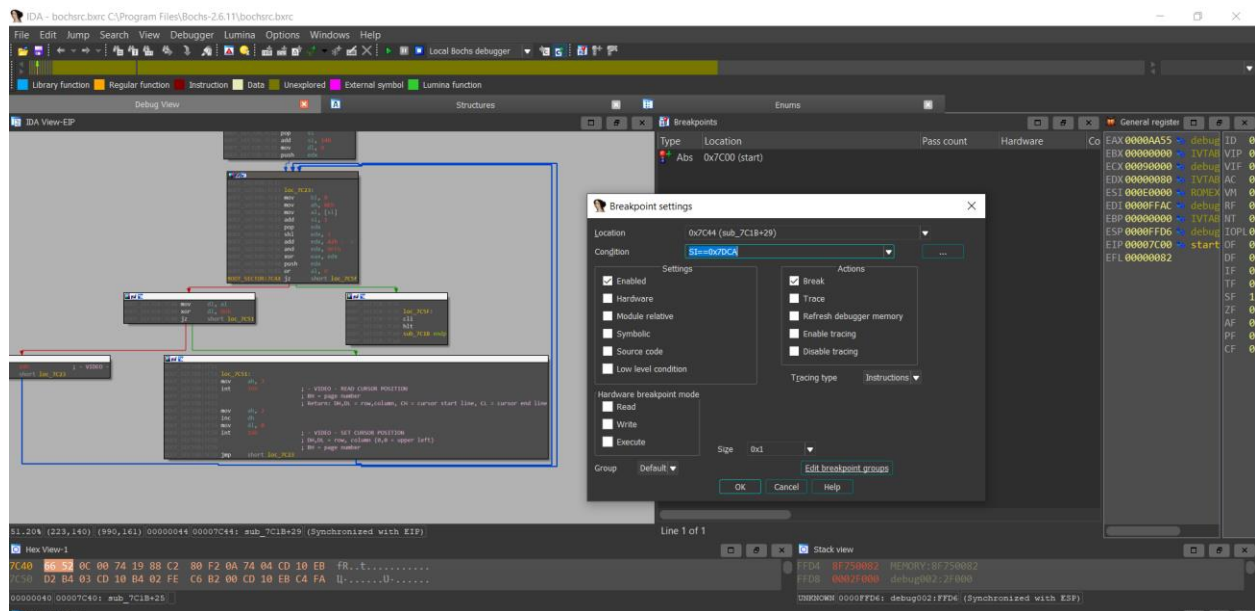
Let's put a breakpoint on the jump address when decrypted byte is 00 (loc\_7C5F). After we continue debugging, we can see that decrypted text is printed on the screen.



But as we can see, the flag is somehow missing. Let's note the SI value = 0x7DCA which is pointing on the next byte which could be decrypted if decrypted byte 00 did not occur.

The main point is that there is more data to be decrypted after decrypted byte 00 which ends our decryption function. Restart the debugging and put a conditional breakpoint on address 0x7C44 (SI==0x7DCA), where conditional jump occurs and is done if decrypted byte is 00.

Setting the conditional breakpoint on address 0x7C44 when value of SI register is 0x7DCA:



Conditional breakpoint hit = decrypted byte is 00

Let's set the ZF = 0 to continue decrypting next bytes and not (jump\_7C5F)

There are clearly more bytes which could be decrypted after 0xDC9.

[illegible]

**Author:**

[\[Twitter\]](#)

[\[Github\]](#)