How to debug instructions mixed with data using GDB-GEF

medium.com/@therealdreg/how-to-debug-instructions-mixed-with-data-using-gdb-gef-5acffce3bb18

Dreg June 29, 2022

Dreg

Jun 28

.

3 min read

Celebrating GDB-GEF 10 year anniversary, **2022.06 — Upset Constant** version has been released. So, I will introduce how to debug instructions mixed with data using GDB-GEF.

Scenario

You want debug a shellcode/malware/packer code like this:

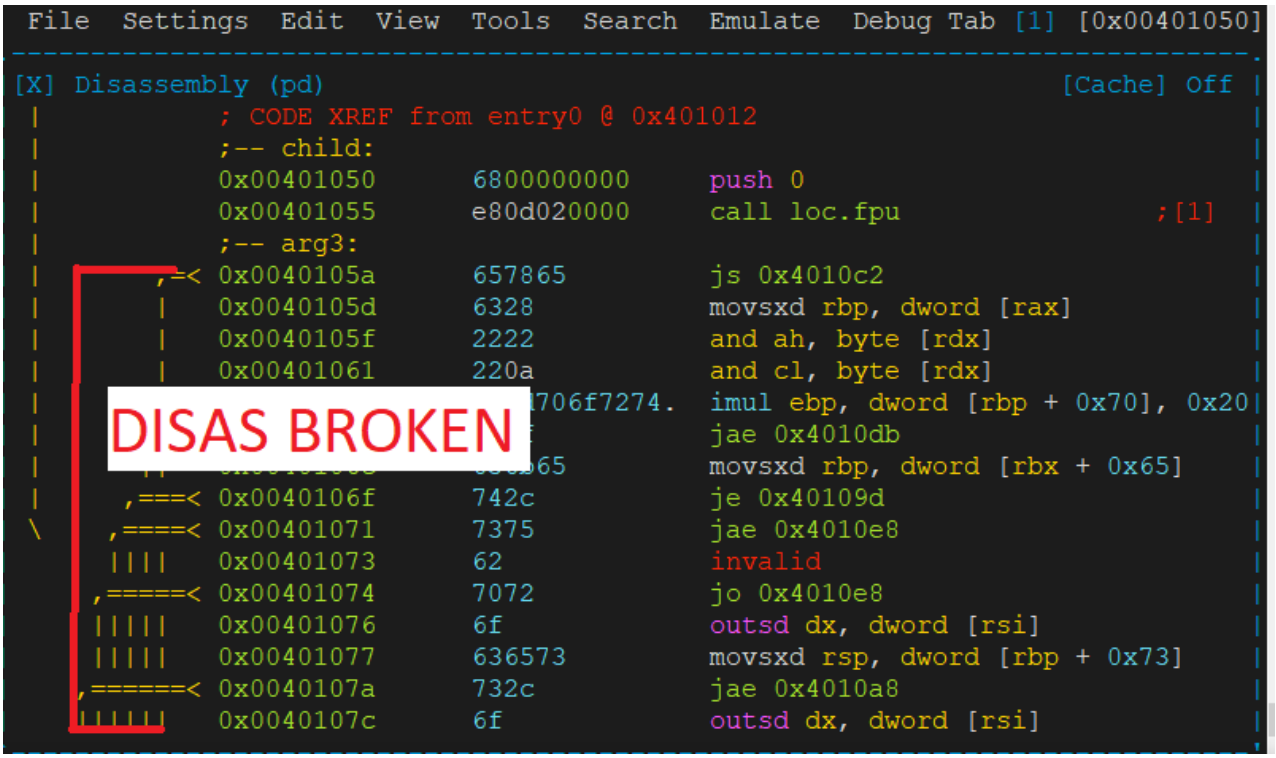
```
; nasm -felf64 -o poc.o poc.asm  && ld -m elf_x86_64 poc.o -o poc

section .text
        global _start
_start:
        times 2 db 90h
        call first
        msg_one db `/bin/sh`,0
        first:
        xor ebx, ecx
        add ebx, 1
        call second
        msg_two db `dregishot`,0
        second:
        mov rax, 60 ; exit
        mov rbx, 0
        syscall
```

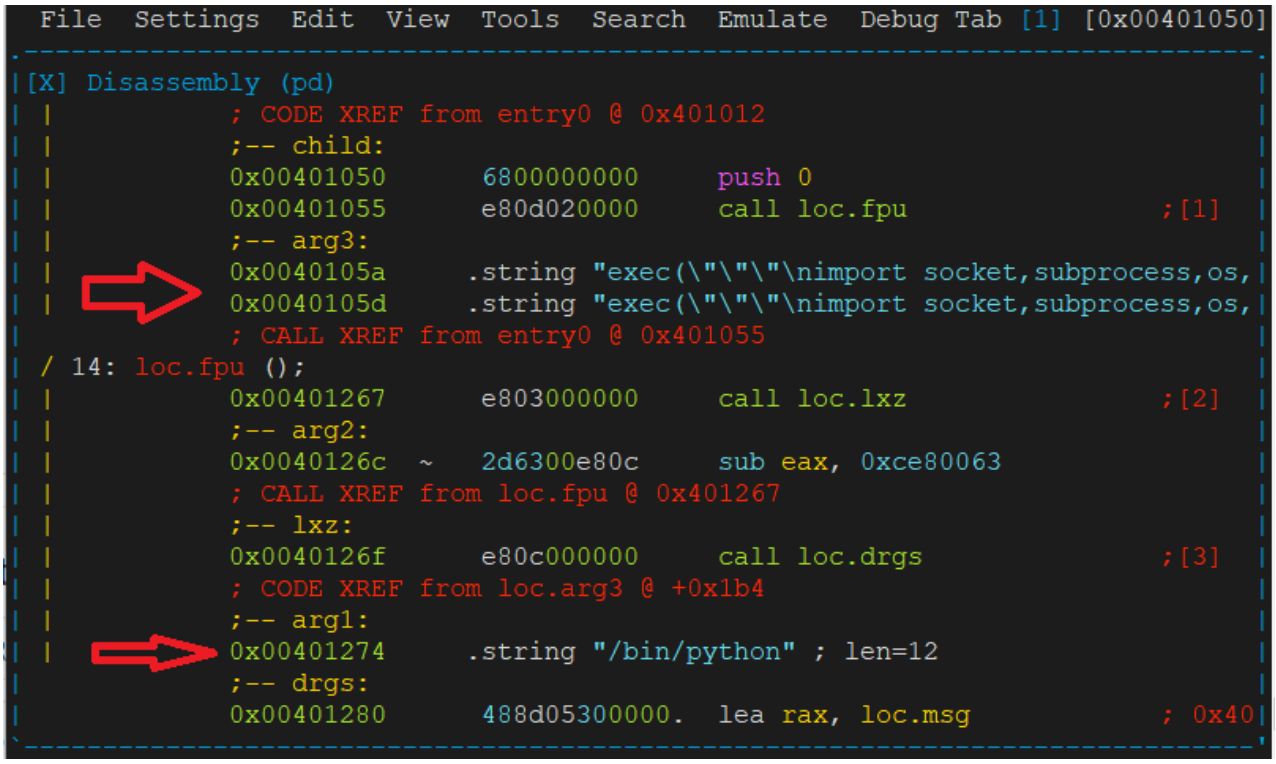
With IDA PRO, GHIDRA or RADARE its very easy, just selecting the strings as data the disas will be fixed.

Some years ago… I made a simple radare crappy-script-poc to auto-mark strings in .text section: call_trick_r2pipe

before call_trick_r2pipe execution:



After call_trick_r2pipe execution:



Problem

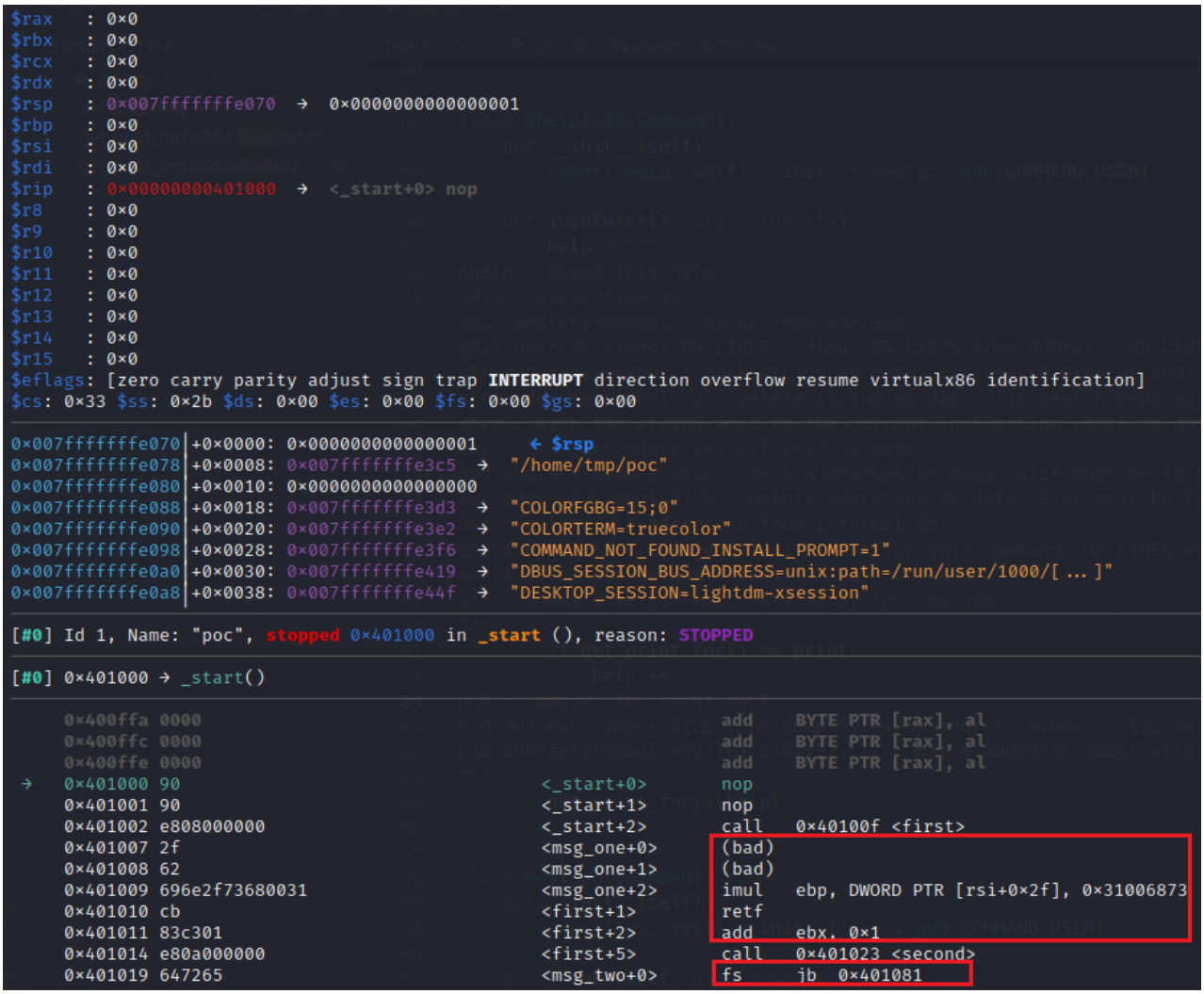
GDB-GEF TEAM (hugsy & contributors) are awesome. But due to GDB-GEF keep-minimal-as-possible philosophy (IMO) this feature will not be included. Maybe in a near future will be supported via gef-extras (I will try it, I promise).

gdis

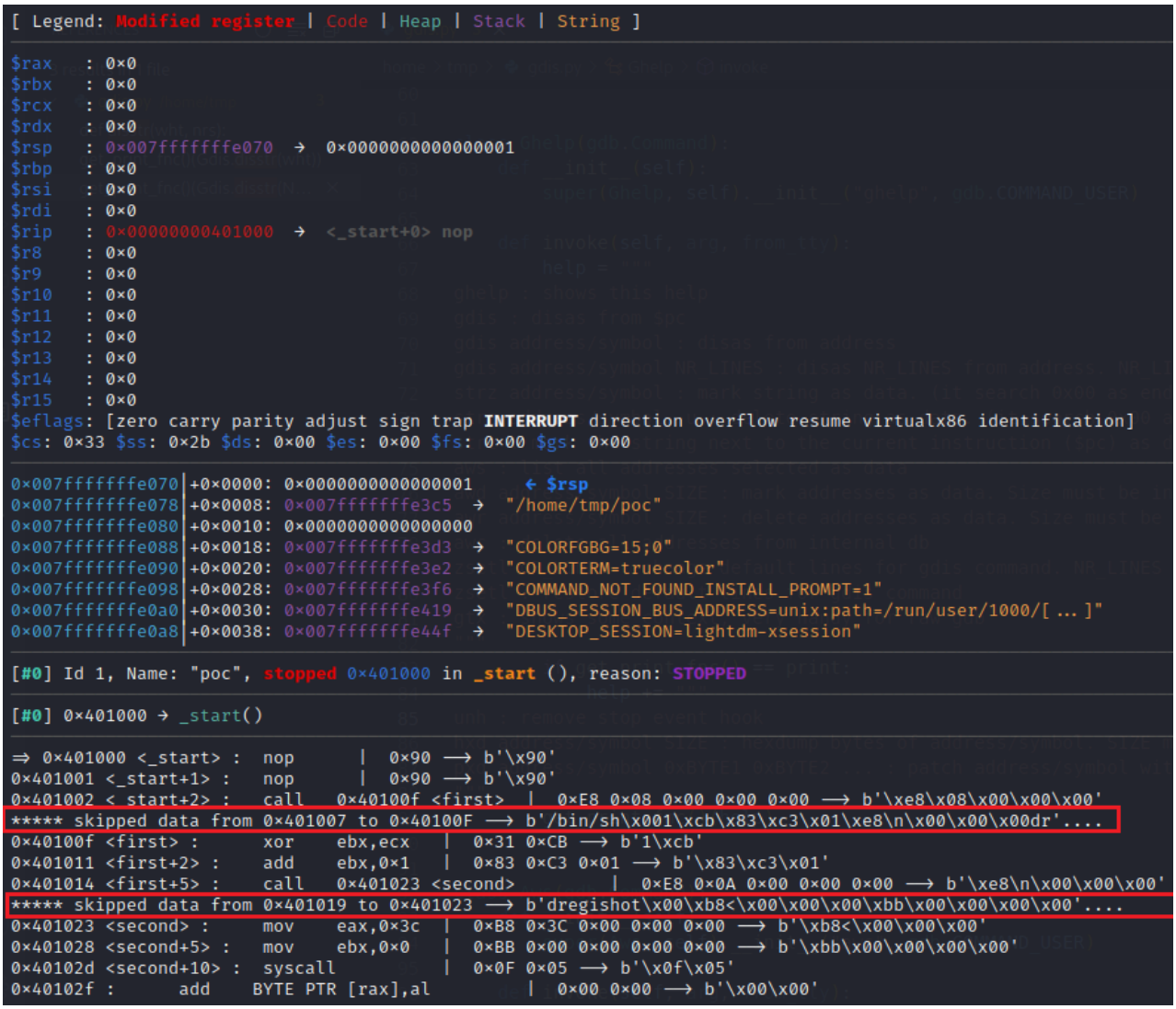
GDB-GEF API is very powerful and easy to use.

So, I have created gdis, a GDB plugin to solve this problem. It works with GDB-GEF and raw GDB (IMO raw GDB is ugly as hell).

GDB-GEF before gdis:



After gdis:



Nice! you can debug and disas the correct code :-)

How to use gdis in GDB-GEF

git clone

Edit **~/.gdbinit**

Add **gdis.py**:

source /home/dreg/gef/gef.py
source /home/dreg/gdis/gdis.py

Open **gdb**

Type: **gef save**

Type: **quit**

Edit **~/.gef.rc**

(Optional step) Remove original “code” panel from layout:

> layout = legend regs **code** stack args source memory threads trace extra

layout = legend regs stack args source memory threads trace extra

POC example for Linux x86_64

open **gdb gdis/poc**

Type: **starti**

```
[#0] 0×401000 → _start()

⇒ 0×401000 <_start> :  nop      |  0×90 ⟶ b'\x90'
0×401001 <_start+1> :  nop      |  0×90 ⟶ b'\x90'
0×401002 <_start+2> :  call   0×40100f <first>  |  0×E8 0×08 0×00 0×00 0×00 ⟶ b'\xe8\x08\x00\x00\x00'
0×401007 <msg_one> :   (bad)    |  0×2F ⟶ b'/'
0×401008 <msg_one+1> :  (bad)    |  0×62 ⟶ b'b'
0×401009 <msg_one+2> :  imul   ebp,DWORD PTR [rsi+0×2f],0×31006873       |  0×69 0×6E 0×2F 0×73 0×68 0×00 0×31 ⟶ b'in/sh\x001'
0×401010 <first+1> :   retf     |  0×CB ⟶ b'\xcb'
0×401011 <first+2> :   add    ebx,0×1   |  0×83 0×C3 0×01 ⟶ b'\x83\xc3\x01'
0×401014 <first+5> :   call   0×401023 <second>         |  0×E8 0×0A 0×00 0×00 0×00 ⟶ b'\xe8\n\x00\x00\x00'
0×401019 <msg_two> :   fs jb  0×401081  |  0×64 0×72 0×65 ⟶ b'dre'
0×40101c <msg_two+3> :  imul   esi,DWORD PTR [ebx+0×68],0×b800746f       |  0×67 0×69 0×73 0×68 0×6F 0×74 0×00 0×B8 ⟶ b'gishot\x00\xb8'
0×401024 <second+1> :   cmp    al,0×0    |  0×3C 0×00 ⟶ b'<\x00'
```

As you can see the disas is broken, for gdis help just type: **ghelp**

Fixing disas

POC contains two strings between instrucions: msg_one and msg_two. Mark them as data with **strz address/symbol** command:

strz msg_one

strz msg_two

Type: **context**

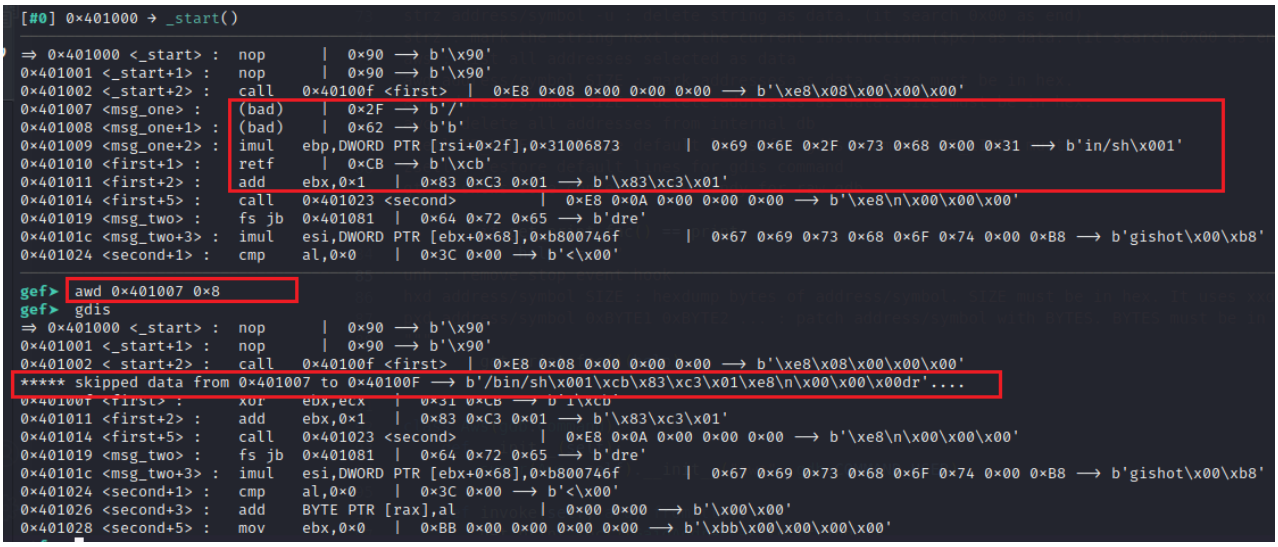
```
[#0] 0×401000 → _start()

⇒ 0×401000 <_start> :  nop      |  0×90 ⟶ b'\x90'
0×401001 <_start+1> :  nop      |  0×90 ⟶ b'\x90'
0×401002 <_start+2> :  call   0×40100f <first>  |  0×E8 0×08 0×00 0×00 0×00 ⟶ b'\xe8\x08\x00\x00\x00'
***** skipped data from 0×401007 to 0×40100F ⟶ b'/bin/sh\x001\xcb\x83\xc3\x01\xe8\n\x00\x00\x00dr'....
0×40100f <first> :     xor    ebx,ecx   |  0×31 0×CB ⟶ b'1\xcb'
0×401011 <first+2> :   add    ebx,0×1   |  0×83 0×C3 0×01 ⟶ b'\x83\xc3\x01'
0×401014 <first+5> :   call   0×401023 <second>         |  0×E8 0×0A 0×00 0×00 0×00 ⟶ b'\xe8\n\x00\x00\x00'
***** skipped data from 0×401019 to 0×401023 ⟶ b'dregishot\x00\xb8<\x00\x00\x00\xbb\x00\x00\x00\x00'....
0×401023 <second> :    mov    eax,0×3c  |  0×B8 0×3C 0×00 0×00 0×00 ⟶ b'\xb8<\x00\x00\x00'
0×401028 <second+5> :  mov    ebx,0×0   |  0×BB 0×00 0×00 0×00 0×00 ⟶ b'\xbb\x00\x00\x00\x00'
0×40102d <second+10> :  syscall         |  0×0F 0×05 ⟶ b'\x0f\x05'
0×40102f :      add    BYTE PTR [rax],al         |  0×00 0×00 ⟶ b'\x00\x00'
```

hooray! the disas is fixed!

How to mark N bytes as data?

easy, just use **awd address/symbol 0xSIZE** command:



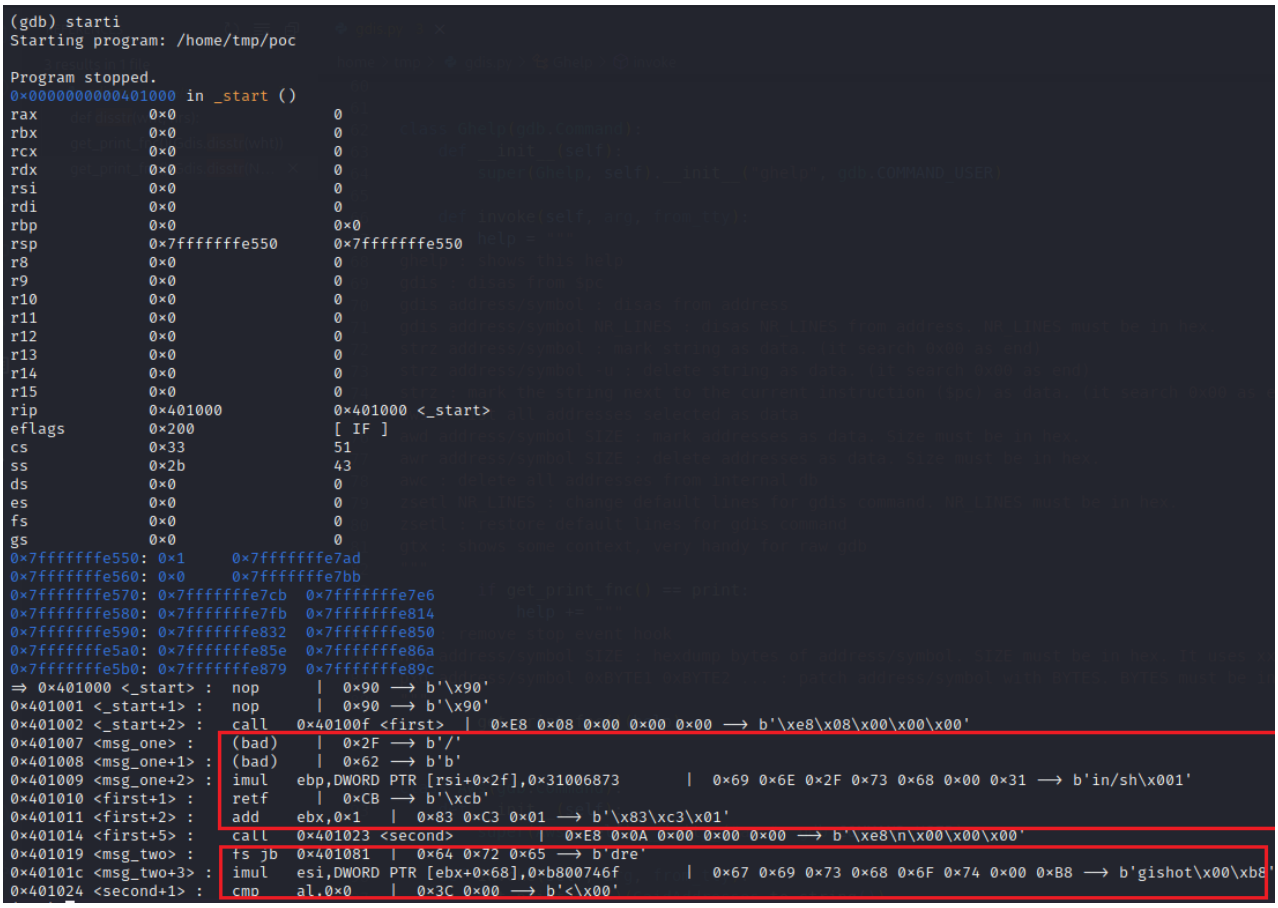
done!

How to use gdis in raw GDB (x86_64 Linux POC)

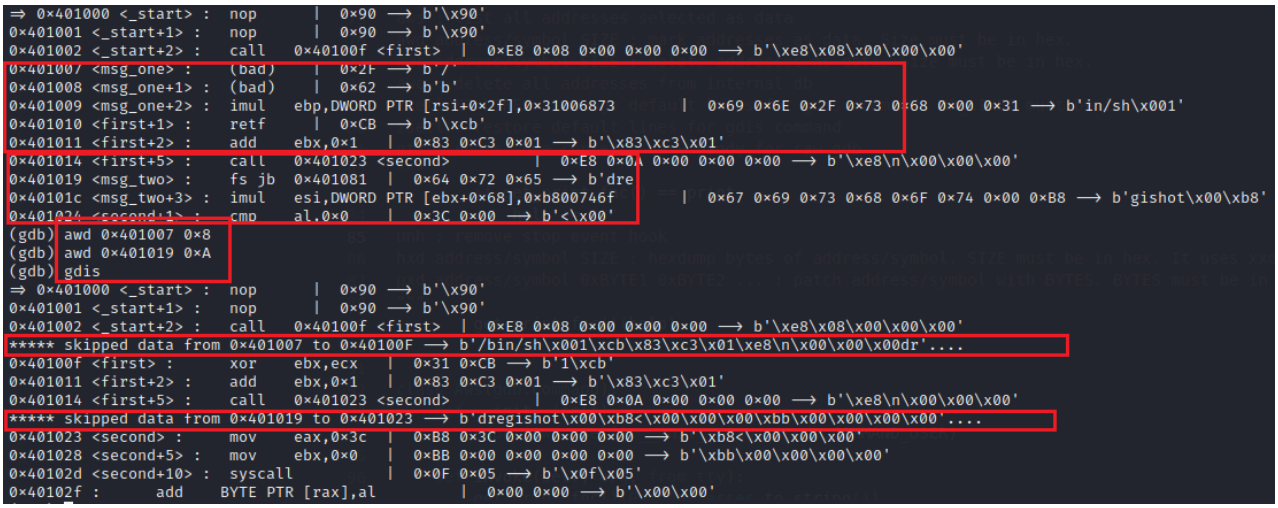
open **gdb gdis/poc**

type: **source /home/gdis/gdis.py**

type: **starti**



Done, just use **awd** and/or **strz** commands to mark data

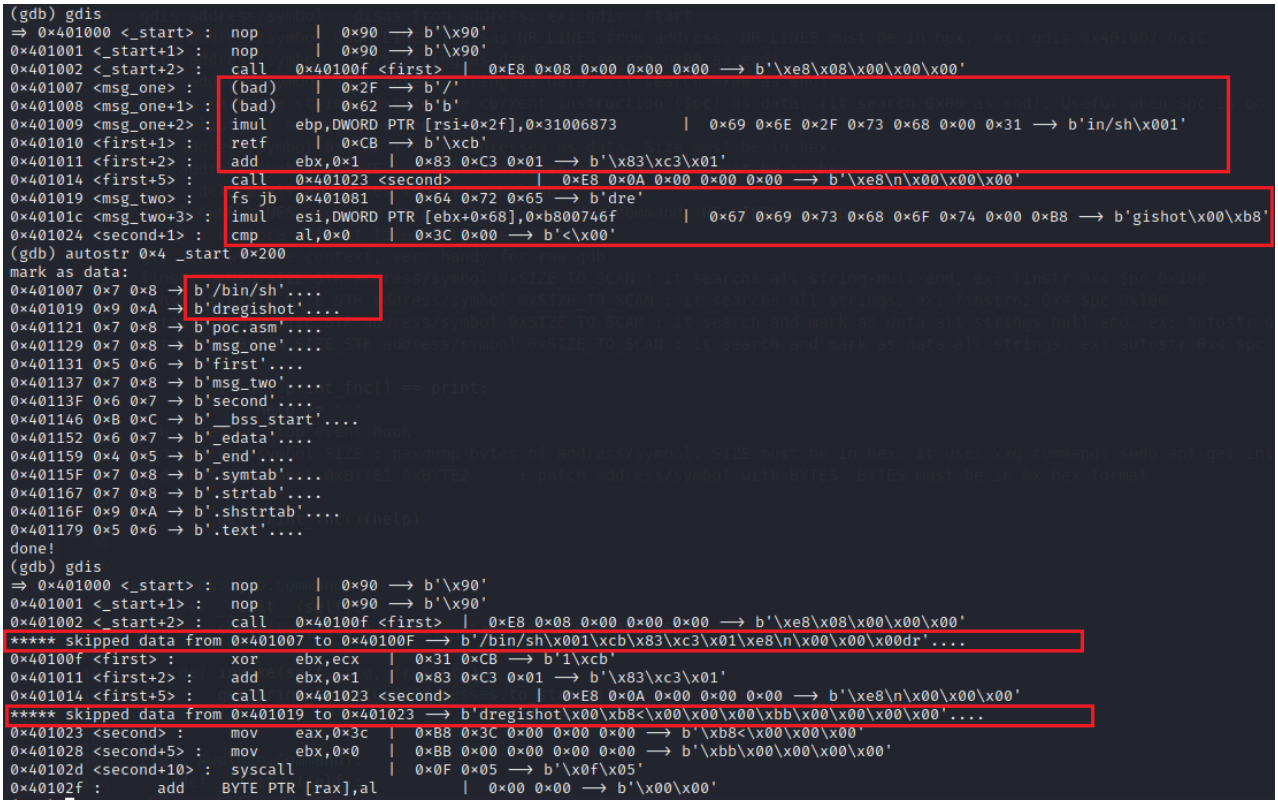


Disas fixed!

Auto mark null-end-strings as data (x86_64 Linux POC)

Just use **autostr 0xMIN_SIZE_STR address/symbol 0xSIZE_TO_SCAN** command:

autostr 0x4 _start 0x200



gdis can do more things, just explore **ghelp** command.

warning: code is pure crap, PRs are welcome! x)

btw, sorry foy my bad english and don't be polite in the comments!

RT @ therealdreg