Το κείμενο αυτό περιέχει κάποιες εντολές του gdb οι οποίες είναι χρήσιμες για το project 1, και ένα παράδειγμα για το πως μπορούν να χρησιμοποιηθούν. Ένας πιό πλήρης κατάλογος εντολών είναι διαθέσιμος στην σελίδα http://*users.ece.utexas.edu/~adnan/**gdb**-refcard.pdf* .

Γενικές εντολές

disassemble : Η εντολή disassemble εκτυπώνει τον κώδικα assembly για την συνάρτηση που δίνουμε ως όρισμα στην εντολή.

info registers: Η εντολή αυτή εκτυπώνει τις τιμές των καταχωρήτων για την δεδομένη στιγμή της εκτέλεσης του πρόγραμματος. Αυτό μπορεί να είναι είτε όταν το πρόγραμμα έχει δεχτεί κάποιο signal, είτε σε κάποιο breakpoint (βλ παρακάτω)

info frame: Δείχνει αρκετές πληροφορίες για το stack frame της συνάρτησης που εκτελείται.

backtrace: Δείχνει όλα τα frames της στοίβας του προγράμματος.

Στον αναλυτικό κατάλογο μπορεί να βρείτε και άλλες εντολές που θα σας φανούν χρήσιμες ανάλογα με το πρόβλημα που αντιμετωπίζετε.

Εξέταση διευθύνσεων μνήμης - Εκτύπωση

x: Η εντολή x μας επιτρέπει να εξετάσουμε τα περιεχόμενα κάποιας διεύθυνσης μνήμης. Η εντολη μπορεί να εκτυπώσει σε διάφορα formats. Κάποια χρήσιμα είναι:
x/x <addr>: εκτυπώνει τα περιεχόμενα της μνήμης addr στο 16αδικο.
x/i <addr>: εκτυπώνει την εντολή assembly που υπάρχει στην διεύθυνση addr.
x/s <addr>: εκτυπώνει το string που βρίσκεται στην διεύθυνση addr.

Για παράδειγμα αν είμαστε μέσα σε κάποια συνάρτηση η εντολή x/x $ebp+4 θα μας δώσει την τιμή της διεύθυνσης επιστροφής, ενώ η εντολή x/i $eip θα μας δώσει την επόμενη εντολή που θα εκτελεστεί. Αυτό είναι αρκετά χρήσιμο όταν εξετάζουμε segmentation faults και θέλουμε να δούμε την εντολή που πήγε να εκτελεστεί και το πρόγραμμα πήρε το segfault.

Αντίστοιχη εντολή είναι και η print, η οποία εκτυπώνει σε αντίστοιχα formats την τιμή κάποιας μεταβλητής. Η διαφορά με την x είναι οτι η x εκτυπώνει τα περιεχόμενα στην θέση μνήμης που της δίνουμε ένω η print εκτυπώνει απλά την τιμή της μεταβλητής που θα δώσουμε. Ο κατάλογος εντολών περιέχει όλες τις επιλογές για τις εντολές print και x.

Breakpoints

Μέσω του gdb μπορούμε να παγώσουμε το πρόγραμμα σε ένα συγκεκριμένο σημείο (συνάρτηση , εντολή etc) και να εξετάσουμε την μνήμη στο σημείο του παγώματος.
Τα breakpoints μπαίνουν μέσω της εντολής break. Μπορούμε να εισάγουμε ένα breakpoint είτε σε μία συνάρτηση είτε σε μία εντολή. Παραδείγματα για breakpoints φαίνονται και παρακάτω.
Για να διαγράψουμε κάποιο breakpoint που έχουμε βάλει μπορούμε να χρησιμοποιήσουμε την εντολή delete. Η εντολή αυτή παίρνει ως όρισμα τον αριθμό του breakpoint που θα διαγράψει.

Παράδειγμα

Καθώς αρκετοί είχαν ένα πρόβλημα με το παράδειγμα του aleph one (example3.c), ας δούμε πως μπορούμε μέσω του debugger να διορθώσουμε το πρόγραμμα ώστε να λειτουργήσει και να προσπεράσουμε την εντολή x=1.
Ο κώδικας του προγράμματος είναι ο εξής

----example3.c----
```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
   int *ret;

   ret = buffer1 + 12;
   (*ret) += 8;
}

void main() {
  int x;

  x = 0;
  function(1,2,3);
  x = 1;
  printf("%d\n",x);
}
```

Αρχικά δοκιμάζουμε να τρέξουμε το πρόγραμμα χωρίς κάποια αλλαγή:

(gdb) run
Starting program: /home/george/atp

Program received signal SIGSEGV, Segmentation fault.
0x080483f6 in main ()
Current language: auto; currently asm
(gdb) x/i $eip
0x80483f6 <main+52>: movl $0x1,-0x8(%ebp)
(gdb) x/x $ebp
0xc79b8f08: Cannot access memory at address 0xc79b8f08

Παρατηρούμε οτι το πρόγραμμα δεν πέρασε την εντολή x=1, αλλά επιπλέον όταν πήγε να την εκτελέσει πήρε ένα segmentation fault. Εξετάζοντας την εντολή βλέπουμε οτι το πρόγραμμα προσπάθησε να προσπελάσει την διεύθυνση στην οποία δείχνει ο %ebp, όμως η διεύθυνση αυτή δεν είναι προσπελάσιμη και έτσι δημιουργήθηκε το segfault.
Αν θυμηθούμε την δομή της στοίβας, όταν κλήθηκε η συνάρτηση function ο ebp αποθηκέυτηκε στην στοίβα ακριβώς από κάτω από την διεύθυνση επιστροφής και η τιμή αυτή επαναφέρθηκε στο τέλος της εκτέλεσης της συνάρτησης function. Πράγματι μπορούμε να δούμε:
(gdb) disassemble function
Dump of assembler code for function function:
….
….
0x080483c0 <function+28>: leave
0x080483c1 <function+29>: ret

End of assembler dump.
To instruction leave είναι ισοδύναμο με τις εντολές
movl %ebp, %esp
popl %ebp

Δηλαδή η τιμή στην οποία δείχνει ο %esp, είσάγεται στον %ebp. Άρα βλέπουμε οτι η εντολή του προγράμματος
ret = buffer1 + 12;
που θα έβαζε την μεταβλητή ret να δείχνει στην διεύθυνση επιστροφής ουσιαστικά δεν παίρνει την σωστή τιμή και έτσι αντί να αλλάξει η διεύθυνση επιστροφής, αλλάζει ο αποθηκευμένος frame pointer. Άρα πρέπει να βρούμε το σωστό offset του buffer1 από την διεύθυνση επιστροφής. Αν έχουμε κάνει compile το πρόγραμμα με debugging symbols (-ggdb) αυτό είναι εύκολο:

(gdb) break function
Breakpoint 1 at 0x80483aa: file db.c, line 7.
(gdb) run
Starting program: /home/george/db

Breakpoint 1, function (a=1, b=2, c=3) at db.c:7
7 ret = buffer1 + 12;
(gdb) print $ebp+4 - buffer1
$2 = 13

Γνωρίζουμε οτι η διεύθυνση επιστροφής είναι 4 bytes πάνω απο την διεύθυνση που δείχνει ο frame pointer. Κάνοντας την αφαίρεση βρίσκουμε οτι το σωστό offset δεν είναι 12 αλλά 13. Ακόμα όμως και αν το πρόγραμμα δεν είχε debugging symbols μπορούσαμε εύκολα να βρούμε την σωστή απόσταση:
(gdb) disassemble function
Dump of assembler code for function function:
….
0x080483aa <function+6>: lea -0x9(%ebp),%eax
0x080483ad <function+9>: add $0xc,%eax
….

Βλέπουμε ο,τι οι 2 αυτές εντολές αντιστοιχούν στην εντολή του προγράμματος
ret = buffer1 + 12. Άρα το buffer1 βρίσκεται 9 bytes κάτω απο τον frame pointer, και καθώς ο fp βρίσκεται 4 bytes από την διεύθυνση επιστροφής το συνολίκο offset είναι 13.
Αντικαθιστούμε στον κώδικα το νέο offset και ξανατρέχουμε το πρόγραμμα:
(gdb) run
Starting program: /home/george/atp
-1078102916

Program exited with code 014.

Τώρα φαίνεται οτι αλλάξαμε την διεύθυνση επιστροφής, παρόλ' αυτά δεν εκτυπώθηκε η τιμή 0, όπως θα περιμέναμε.Ελέγχουμε να δούμε την διεύθυνση επιστροφής που βάζει η μεταβλητή ret στην συνάρτηση function. Θα το κάνουμε αυτό βάζοντας ένα breakpoint στο τέλος της συνάρτσησης function που έχει αλλαχτεί η διεύθυνση επιστροφής:

```
(gdb) disassemble function
Dump of assembler code for function function:
….
….
0x080483c0 <function+28>:  leave
0x080483c1 <function+29>:  ret
End of assembler dump.
(gdb) break *function+28
Breakpoint 1 at 0x80483c0
(gdb) run
Starting program: /home/george/atp

Breakpoint 1, 0x080483c0 in function ()
Current language:  auto; currently asm
(gdb) x/x $ebp+4
0xbf907c0c:    0x080483fe
(gdb) x/5i 0x080483fe
0x80483fe <main+60>:        inc    %ebp
0x80483ff <main+61>:        clc
0x8048400 <main+62>:        mov    %eax,0x4(%esp)
0x8048404 <main+66>:        movl   $0x80484e0,(%esp)
0x804840b <main+73>:        call   0x80482d8 <printf@plt>
```

Παρατηρούμε οτι η διεύθυνση επιστροφής έχει αλλάξει, αλλά αυτήν την φορά δείχνει σε κάποιες εντολές οι οποίες κανονικά δεν υπήρχαν στην συνάρτηση main. Αυτο συμβαίνει διοτί η return address δείχνει στη μέση κάποιας εντολής. Αν το opcode μίας εντολής αποτελείται από 4 bytes, τότε ξεκινώντας ο %eip από το πρώτο byte θα διάβαζε την εντολή αυτή, αλλα ξεκινώντας από το 2ο πχ θα διάβαζε κάποια τελείως διαφορετική εντολή.
Τώρα πρέπει να βρούμε την τιμή που θέλουμε να βάλουμε στην return address για να περάσει τελείως την εντολή x = 1.

```
(gdb) disassemble main
Dump of assembler code for function main:
….
0x080483f1 <main+47>:       call   0x80483a4 <function>
0x080483f6 <main+52>:       movl   $0x1,-0x8(%ebp) // x = 1
0x080483fd <main+59>:       mov    -0x8(%ebp),%eax
0x08048400 <main+62>:       mov    %eax,0x4(%esp)
….
```

Η εντολή x=1 αντιστοιχεί στην εντολή στην διεύθυνση 0x080483f6, που είναι και η διεύθυνση επιστροφής της συνάρτησης. Εμείς θέλουμε να εκτελεστεί η επόμενη εντολή που βρίσκεται στην διεύθυνση 0x080483fd. Άρα πρέπει να προσθέσουμε στην διεύθυνση επιστροφής 0x080483fd - 0x080483f6 = 7. Δοκιμάζουμε το πρόγραμμα πάλι με την νέα τιμή της διεύθυνσης επιστροφής και βλέπουμε οτι καταφέρνουμε να περάσουμε την εντολή ανάθεσης.