

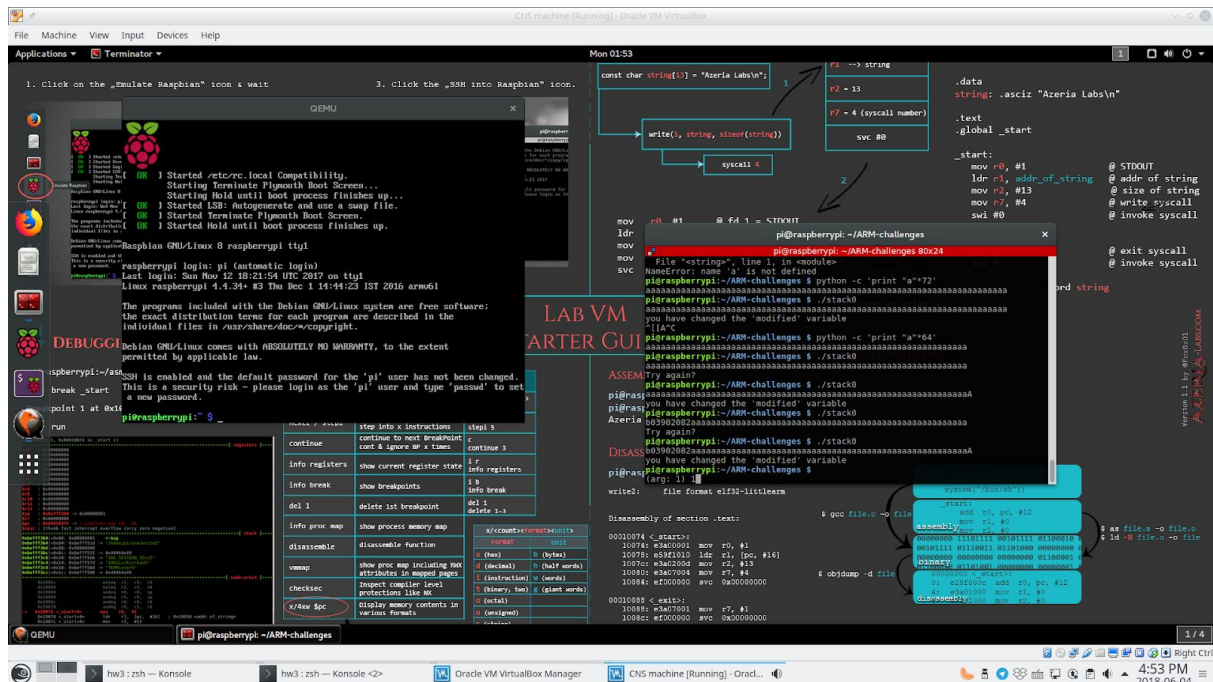
CNS HW3 Report

b03902082 江懿友

Problem 1

(a) stack0

The buffer length is 64, so at least *65* characters should be input to trigger a buffer overflow.



(b) stack1

The value is 0x61626364, which is "dcba" in little-endian.

I use objdump to disassemble the program and found the value inside function <main>. The value is a pc-relative data located at 0x1053c.

The image shows a Raspberry Pi desktop environment. At the top, there is a taskbar with icons for applications, a search bar, and system status. The main window is a terminal titled "pi@raspberrypi: ~/ARM-challenges" with a red header bar. The terminal displays assembly code for a program named "113x55". The code includes instructions like "push", "pop", "add", "sub", "ldr", "str", "bl", "bne", "b", "nop", "hlt", and "nop". Comments in the code indicate the flow of execution, such as "code flow successfully changed" and "Segmentation fault". The terminal also shows the output of the "python" command, which prints a string of characters. The desktop background is a dark theme with a Raspberry Pi logo. Other windows visible include a file manager showing a directory structure and a terminal window with a shell prompt. The system clock in the bottom right corner shows the time as 9:16 PM.

(a) aMAZEing

Flag: BALSNI{4M4z31nG_bUg_F0uNd_bY_KLEE!!}

(b) Flag Verifier

Flag: BALSNI{5ymb0l1c Ex3cut10n Hurr4y}

I changed this:

```
int cal(int a, int b, int c)
{
    int i, j, k;
    int total = 0;

    for(i = 0; i < a; i++)
    {
        for(j = 0; j < b; j++)
        {
            for(k = 0; k < c; k++)
            {
                if(i > k && j < i)
                    return cal(a-1, b-1, c-1);
                return total + cal(a-1, b-1, c-1);
            }
        }
    }

    return 1;
}
```

to this: `#define cal(a, b, c) (1)`

Problem 3

(a) Mirror Force

Use a DNS query amplifier. This DNS query works pretty well:

```
dig @8.8.8.8 RRSIG isc.org +notcp
```

I use wireshark to analyze the packets:

No.	Time	Source	Destination	Protocol	Length	Info
6	2.046075759	192.168.1.103	8.8.8.8	DNS	90	Standard query 0xa308 RRSIG isc.org OPT
8	2.191665126	8.8.8.8	192.168.1.103	IPv4	1482	Fragmented IP protocol (proto=UDP 17, off=0, ID=ea38) [Reassembled in #9]
9	2.191867135	8.8.8.8	192.168.1.103	DNS	762	Standard query response 0xa308 RRSIG isc.org RRSIG RRSIG RRSIG RRSIG RRSIG

I send an UDP packet of size 90 and google's public DNS server replies with an UDP packet of size 2244, so the amplification factor is $2244/90 = 24.93$.

Because this DNS query uses UDP to transfer query/response, there is no TCP handshake and I can spoof the query packet's source IP to victim's IP, and thus the amplified response packet would be sent to the victim.

(b) The Revenge of Hash Table

(c) 499 Chaos

Flag: BALS{Be_cAreFu1_of_F10A7ING_PoInt_And_NaN}

The screenshot shows a web browser window. A modal dialog box is open, displaying a transaction confirmation message from IP 140.112.31.96:10131. The message states: "Transaction complete: ['BALS{Be_cAreFu1_of_F10A7ING_PoInt_And_NaN}', 'Thanks you for buying standard plan. Get premium for the flag!']". Below the message is an "OK" button. In the background, the Balsn Telecom website is visible. It features a table with three plans: Economical Plan (2G, NT \$99), Standard Plan (5G, NT \$299), and Premium Plan (Unlimited, NT \$499). Each plan has a "Buy" column with a text input field and the word "month". The Standard Plan input field contains "10000000000" and the Premium Plan input field contains "0.0000000001". Below the table, there is a note: "*If you are VIP, please enter your telephone number:" followed by an input field containing "9453" and a "Buy Now" button. At the bottom, a copyright notice reads: "Copyright © Balsn Telecom™ 2018. All rights reserved."

Plan	Data	Price	Flag	Free For VIP*	Buy
Economical Plan	2G	NT \$99	-	✓	<input type="text" value="0"/> month
Standard Plan	5G	NT \$299	-	✓	<input type="text" value="10000000000"/> month
Premium Plan	Unlimited	NT \$499	✓	-	<input type="text" value="0.0000000001"/> month

*If you are VIP, please enter your telephone number:

Copyright © Balsn Telecom™ 2018. All rights reserved.

The vulnerability is caused by a floating point rounding error. I guess that the python server uses IEEE-754 formatted double-precision floating point. IEEE-754 formatted double-precision floating point stores up to roughly 15 digits of precision (in base 10), so buying $1e10$ months of "standard plan" and $1e-10$ months of "premium plan" costs $1e10 * 299 + 1e-10 * 499$ which rounds to $2.99e12$. The cost of "premium plan" is loss and "standard plan" costs no money for VIP. Thus the total cost equals zero.

Problem 4

Challenge 0

Pass in my student ID and it is solved.

Challenge 1

The solution is 777. The answer can be found in the smart contract source.

Challenge 2

Send 1 eth and get score.

Challenge 3

Brute force search the range of solution (0~255) and the answer can be found.

keccak256(146) =

0x313b2ea16b36f2e78c1275bfcca4e31f1e51c3a5d60beeefe6f4ec441e6f1dfc

Challenge 4

The solution of challenge 4 equals `uint16(keccak256(blockhash(block.number-1), block.timestamp))`.

The block number of the contract is 3245076.

The blockhash of block 3245075 is

0xb388f89fb847890e2d96c9a37c9d25beadef55fb84c2b1b2cf41dc7703bd08f5.

The timestamp of block 3245076 is 1526464861.

Thus the answer is

`uint16(keccak256(0xb388f89fb847890e2d96c9a37c9d25beadef55fb84c2b1b2cf41dc7703bd08f5, 1526464861)) = 47194`.

Challenge 5

Bonus

A hard fork is that a cryptocurrency undergoes some changes in its code and the blockchain splits into two. One fork is the new chain that runs the new code and the other fork is the old chain that runs the unchanged, "old", code.

In 2016, the Ethereum community decided to implement a hard fork in an attempt to undo the harm caused by DAO, this resulted in two separate chains, Ethereum, the new chain and Ethereum Classic, the old chain.

The DAO vulnerability is kind of double spend attack. The DAO contract's `withdrawBalance()` function calls the fallback function before setting the user's balance to zero. So in the case that the fallback function (which can be provided by the user/client/attacker) calls `withdrawBalance()` again, the attacker can withdraw double the amount of his balance.