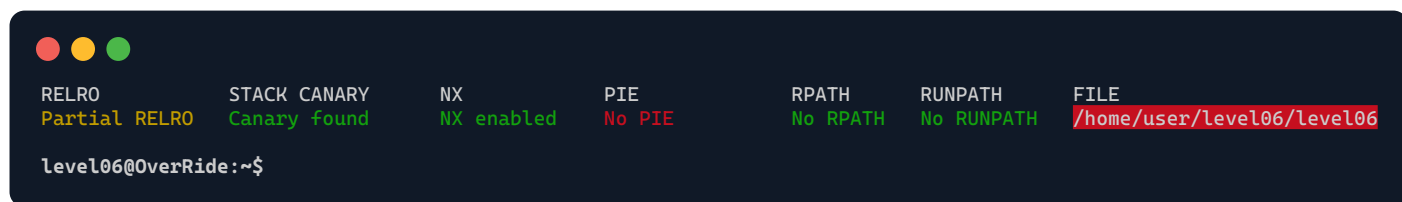


./level06



Decompiled file with *Ghidra*:

```
int auth(char *username, unsigned int serial)
{
    username[strcspn(username, "\n")] = '\0';
    size_t len = strlen(username, 32);

    if (len < 6)
        return 1;

    if (ptrace(PTRACE_TRACEME, 0, 1, 0) == -1)
    {
        puts("\x1b[32m.-----.");
        puts("\x1b[31m| !! TAMPERING DETECTED !! |");
        puts("\x1b[32m\'-----\'");
        return 1;
    }

    unsigned int checksum = (username[3] ^ 0x1337) + 0x5eeded;
    for (int i = 0; i < len; i++)
    {
        if (username[i] < ' ')
            return 1;
        checksum += (username[i] ^ checksum) % 0x539;
    }

    if (serial != checksum)
        return 1;
    return 0;
}

int main(void)
{
    unsigned int serial;
    char username[32];

    puts("*****");
    puts("*           level06           *");
    puts("*****");
    printf("-> Enter Login: ");
    fgets(username, 32, stdin);

    puts("*****");
    puts("***** NEW ACCOUNT DETECTED *****");
    puts("*****");
    printf("-> Enter Serial: ");
    scanf("%u", &serial);

    int ret = auth(username, serial);
    if (ret == 0)
    {
        puts("Authenticated!");
        system("/bin/sh");
    }

    return ret != 0;
}
```

This program is designed as a simple authentication system that uses a **username** and a **serial number** to validate a user and then attempts to authenticate them based on certain criteria:

Firstly, the program removes any newline character from the end of the **username** and checks that it is at least six characters long. If the **username** is too short, the **authentication** fails.

Next, the program uses the **ptrace** system call with the **PTRACE_TRACEME** flag. This is a common way to detect if a program is being debugged; if it is, the program prints a tampering detection message and fails the authentication.

For the actual authentication, the program calculates a **checksum** from the **username**. The program initializes a checksum by **XOR**-ing the third character of the **username** with **0x1337** and adding **0x5eeded** to it. It then iterates over each character in the **username**, confirming it's printable, and for each character, it **XORs** it with the checksum, takes the result modulo **0x539**, and adds it to the checksum.

The authentication is successful if the final checksum matches the **serial number** provided by the user, at which point the program acknowledges the successful login and grants **shell** access.

To crack this program, we simply need to replicate the checksum calculation using a chosen **username** to generate a matching **serial number** for authentication:

```
#include <stdio.h>
#include <string.h>

int main(int ac, char **av)
{
    size_t len = strlen(av[1]);
    unsigned int checksum = (av[1][3] ^ 0x1337) + 0x5eeded;
    for (int i = 0; i < len; i++)
    {
        if (av[1][i] < ' ')
            return 1;
        checksum += (av[1][i] ^ checksum) % 0x539;
    }

    printf("%u\n", checksum);
}
```

```
level06@Override:~$ ({
    echo '#include <stdio.h>
    #include <string.h>

    int main(int ac, char **av)
    {
        size_t len = strlen(av[1]);
        unsigned int checksum = (av[1][3] ^ 0x1337) + 0x5eeded;
        for (int i = 0; i < len; i++)
        {
            if (av[1][i] < 32)
                return 1;
            checksum += (av[1][i] ^ checksum) % 0x539;
        }

        printf("%u\n", checksum);
    }' > /tmp/fndsum.c;
    gcc -std=c99 -o /tmp/fndsum /tmp/fndsum.c;
    export username="$USER";
    export serial=$(/tmp/fndsum $username);
    echo $username;
    echo $serial;
    sleep 0.1;
    echo "cd ../level07 && cat .pass && exit";
    rm -rf /tmp/fndsum.c /tmp/fndsum;
} | ~/level06)
```

```
*****
*           level06           *
*****
-> Enter Login: *****
***** NEW ACCOUNT DETECTED *****
*****
-> Enter Serial: Authenticated!
```

GbcPDRgsFK77LNnnuh7QyFYA2942Gp8yKj9KrWD8

```
level06@Override:~$ su level07
Password: GbcPDRgsFK77LNnnuh7QyFYA2942Gp8yKj9KrWD8
```

```
level07@Override:~$
```