Сначала я решил проверить и посмотреть, как работает программа. Я обычно ввожу длинную текстовую строку, отсматриваю вывод и потом ищу вхождения строк в дизассемблере или ссылки на строки в отладчике

Однако в данном случае я заметил кое-чего интересного уже на этапе анализа ввода-вывода

```
enter password:notapassword
invalidinvalid
```

```
enter password:password
invalid
```

```
enter password:111111111
invalid
```

```
enter password:verylongstring
invalidinvalid
```

```
enter password:pass
invalidinvalid
```

Судя по всему, происходит несколько проверок на длину строки, так что уже изначально готовимся к тому, что пароль должен быть как минимум определенной длины, и, видимо, двойное invalid выводится при непрохождении проверки качества пароля и при проверке его статуса

Открываем программу в дизассемблере и пытаемся найти вхождения строк и xref-функции, которые с ними взаимодействуют

```
                s_invalid_140004012                             XREF[8]:     check3:140001554 (*),
                                                                             check3:14000155b (*),
                                                                             check2:14000157e (*),
                                                                             check2:140001585 (*),
                                                                             check:1400015a6 (*),
                                                                             check:1400015ad (*),
                                                                             main:140001652 (*),
                                                                             main:140001659 (*)
12  69 6e 76        ds          "invalid"
    61 6c 69
    64 00
```

Видим, что строка некорректного статуса пароля вызывается в функции main и в нескольких функциях проверки. Сначала проанализируем, что происходит в main

```
int iVar1;
FILE *_File;
size_t length;
char pwd_char_array [32];
char input [36];
int i;
int local_10;
int int_length;

__main();
printf("enter password:");
_File = (FILE *)__acrt_iob_func(0);
fgets(input,0x19,_File);
length = strlen(input);
int_length = (int)length;
if (input[int_length + -1] == '\n') {
  int_length = int_length + -1;
}
local_10 = 0;
while( true ) {
  if (int_length <= local_10) {
    for (i = 0; i < int_length; i = i + 1) {
      pwd_char_array[i] = input[i];
    }
    check(int_length);
    check3(pwd_char_array);
    return 0;
  }
  iVar1 = isalpha((int)input[local_10]);
  if (iVar1 == 0) break;
  local_10 = local_10 + 1;
}
printf("invalid");
return 0;
```

Да, я уже немного поменял названия переменных, чтобы удобнее было ориентироваться, и моя догадка на счет нескольких практически параллельных независимых проверок уже подтверждена:

Check, как я выяснил из дизассемблированной функции, проверяет длину пароля, а check3 – её символы

Разберемся сначала с проверкой длины пароля

```
2  void check(int param_1)
3
4  {
5    if (param_1 < 11) {
6      check2(param_1);
7    }
8    else {
9      printf("invalid");
0    }
1    return;
2  }
```

Пароль не должен быть больше 11 символов

```
void check2(int param_1)

{
  if (param_1 < 5) {
    printf("invalid");
  }
  return;
}
```

И не может быть меньше 5. Идем смотреть функцию проверки самого пароля

```
if (((int)pwd[1] * (int)*pwd == 7326) && ((int)pwd[2] * (int)pwd[1] == 12321)) {
  if ((((int)pwd[3] / (int)pwd[4] == 1) && ((int)pwd[3] % (int)pwd[4] == 1)) &&
     ((pwd[2] + -100) * (pwd[1] + -100) == (int)pwd[4])) {
    printf("go grab a beer :)");
```

Значит, проверяются не сами символы, а их целочисленные эквиваленты по заранее установленным соотношениям, причем в два этапа

На первом шаге проверки проверяются результата умножения первого на второй, второго на третий символ.

На втором шаге проверки 4 и 5 символы должны давать результаты целочисленного деления и с остатком одинаковый – единицу. Кроме того, второй и третий символы пароля при умножении их уменьшенных на 100 значений должны совпасть с 5 символом

Поскольку задание англоязычное, я предположу, что логичнее всего было бы рассматривать печатные символы ASCII

Коды печатных, возможных в использовании в качестве пароля от 65 до 126 – числа мы точно не используем.

Значит, напишем скрипт, который проверит комбинации и если сможет найти решение, то выдаст его:

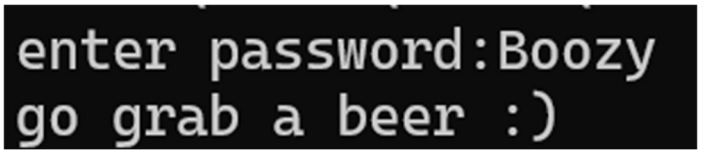
```
def script():
  for b in range(65, 127):
    if 7326 % b == 0 and 12321 % b == 0:
      a = 7326 // b
      c = 12321 // b
      if 32 <= a <= 126 and 32 <= c <= 126:
        e = (c - 100) * (b - 100)
        if 32 <= e <= 126:
          d = e + 1
          if 32 <= d <= 126:
            return ''.join(chr(x) for x in [a, b, c, d, e])
  return None

result = script()
print(f"{result}")
```

Решение найдено

```
Boozy
```

Проверяем



Из интересного:

Так как на втором этапе проверки нет else-ветки, то, дав пароль, проходящий только первый этап, программа вообще ничего не напишет