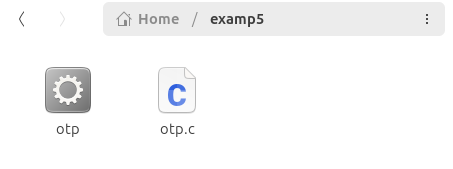
1. Компилируем программу с помощью gcc.



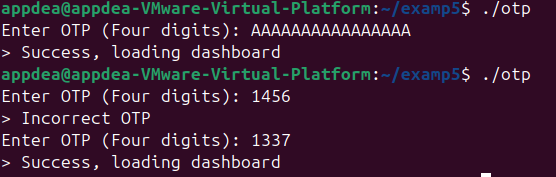
1. GetOTP возвращает статической OTP код «1337» (обычно он генерируется динамически. Успех = загрузке панели управления (LoadDashboard). На успех даётся 3 попытки, дальше программа закрывается
2. Функция GETS не проверяет длину ввода, что делает возможность эксплуатации BufferOverflow.
3. Если введено больше чем 4 символа – это переполнит буффер и может спровоцировать странное поведение (имеется ввиду запуск панели управления)

**КАК ВОСПРОИЗВЕСТИ:**

1. Компилируем программу при помощи gcc, но с дополнительными флагами. (новые версии компиляторов закрывают базовые эксплоиты, так что для наглядности делаем так)

*gcc -fno-stack-protector -z execstack -o otp otp.c*

1. Запускаем скомпилированную программу и вводим больше, чем 4 символа. Также вводим обычные варианты для проверки корректности работы программы.



**Как защитить:**

1. Меняем небезопасную функцию gets (о её небезопасности предупреждает даже сам компилятор) на fgets

*fgets(tryOTP, sizeof(tryOTP), stdin);*

1. Можно хешировать OTP и сравнивать хеши, а не просто числовые значения. Пример на python (но тут в целом сложно внедрить в код, ибо подробного алгоритма OTP не указано):

*import hashlib*

*# Функция для хеширования OTP*

*def hash\_otp(otp):*

*return hashlib.sha256(otp.encode()).hexdigest()*

*# Пример OTP*

*otp = "1337"*

*hashed\_otp = hash\_otp(otp) # Хешируем OTP*

*# Сохраняем хеш в базу данных (в реальной системе это делается на сервере)*

*print(f"Hashed OTP: {hashed\_otp}")*

*# Пользователь вводит OTP*

*user\_input = input("Enter OTP: ")*

*# Хешируем введенный OTP и сравниваем с сохраненным хешем*

*if hash\_otp(user\_input) == hashed\_otp:*

*print("> Success, OTP is correct!")*

*else:*

*print("> Incorrect OTP")*

1. Также можно использовать современную внутреннюю защиту на компиляторах (защита стека, ASLR)