Лабораторна робота 4

Аналіз механізмів захисту додатку та їх блокування

Приходько Юрій ФБ-12, варіант 13

Мета роботи

Навчитися використовувати засоби статичного і динамічного аналізу програм. Отримати навички модифікації бінарного коду додатка.

Вміст завдання

Досліджувати надану програму, що володіє захистом від свого несанкціонованого використання і зламати захист різними способами. У процесі виконання лабораторної роботи необхідно:

1. Виділити в досліджуваній програмі ділянку коду, що виконує функцію прийняття рішення про коректність введеного пароля. Визначити файл або файли, в яких зберігається зашифрований пароль.
2. Здійснити блокування встановленої захисту, реалізувавши відключення захисного механізму, шляхом модифікації функції прийняття рішення про коректність введеного пароля.
3. Виділити в програмі ділянку коду, відповідальний за формування коректного пароля, відповідного введеному імені користувача. Досліджувати даний код і формально записати алгоритм формування коректного пароля. Використовуючи код програми, відповідальний за формування правильного пароля, створити генератор паролів.
4. Здійснити злом встановленої захисту, використовуючи деякий користувальницький ідентифікатор (ім'я користувача) і відповідний йому коректний пароль, сформований по знайденому в п.3 алгоритму.
5. Виділити в досліджуваній програмі ділянку коду, що виконує функцію прийняття рішення про перевищення встановленої межі запусків. Визначити ключі реєстру, в яких зберігається лічильник запусків.
6. Здійснити блокування встановленої захисту - або підміною функції прийняття рішення про перевищення встановленої межі, або шляхом зміни ключів реєстру.

Хід роботи

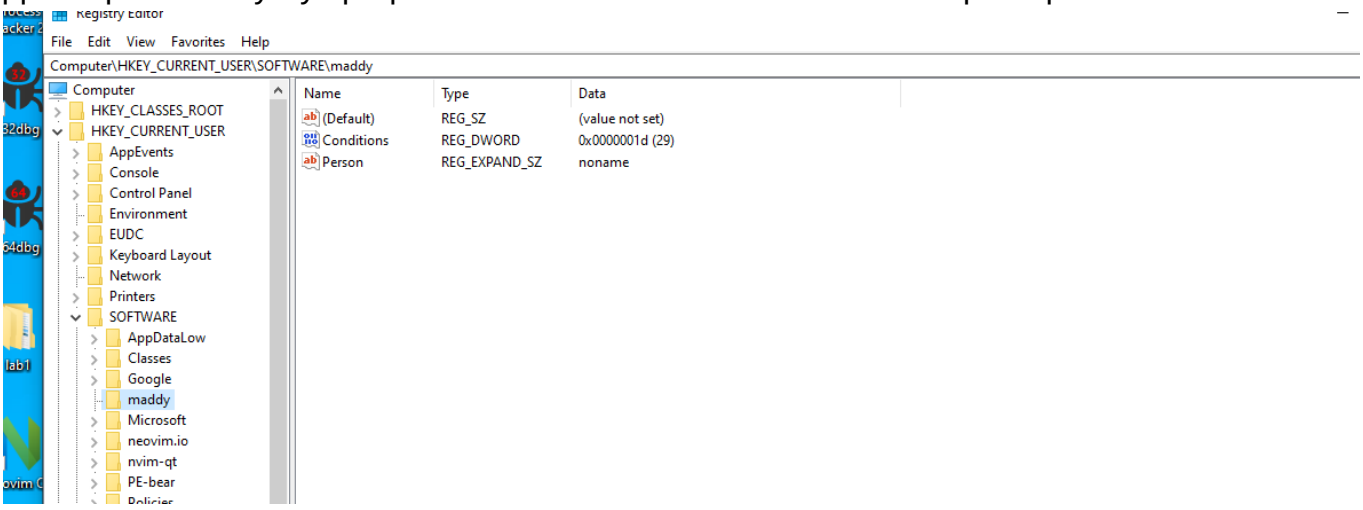
1. Використовуючи засоби декомпіляції, знайдемо ділянку коду в якій відбувається перевірка паролю.

За допомогою декомпілятора ghidra відкриємо додаток отриманий в завданні та переглянемо структуру коду. Важливо звернути увагу на потік виконання для того аби знайти умови що необхідні для запуску застосунку, та логіну в ньому.

При запуску додатку і введенні неправильного логіну та паролю ми отримуємо повідомлення `Sorry, Mikel, Buzzenesss:)`, якщо знайти його в декомпільованому коді то ми легко можемо побачити де відбувається перевірка паролю. Такий висновок може бути зроблений за двума речами, це перевірка резхультату виконання функції і виклик повідомлення помилки при цьому, а також параметрам функції що складають юзернейм, пароль та довжину юзернейму. (На наступному зображенні `FUN_00401200` це функція перевірки паролю `lpData` це `username`, `lpString` це `password` введені в логін формі застосунку, останній параметр довжина введеного імені).

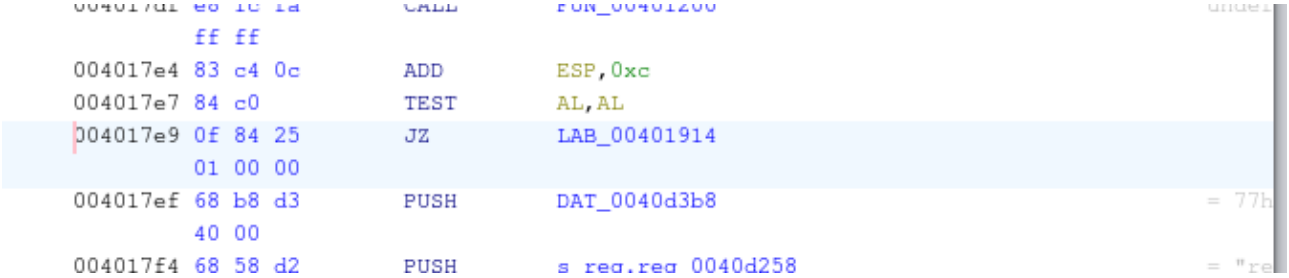
```
29    pBVar2 = pBVar2 + 1;
30  } while (BVar1 != '\0');
31  uVar3 = FUN_00401200((int)lpData,(char *)lpString,(int)pBVar2 - (int)(lpData + 1));
32  if ((char)uVar3 == '\0') {
33    MessageBoxA((HWND)0x0,"Sorry, Mikel, Buzzinesss:)","Let the SuXX bi wiz U",0);
34    UVar6 = 0;
35    pvVar5 = GetCurrentProcess();
36    TerminateProcess(pvVar5,UVar6);
37    return 0;
38  }
39  _File = _fopen("reg.reg","w");
40  if (_File == (FILE *)0x0) {
41    MessageBoxA((HWND)0x0,"Sorry, but I can do this :((( Ask you admin why :)","SUXX",0);
42    UVar6 = 0xf;
```

Для першого запуску програми встановлюються такі значення в реєстрі.



2. Для обходу перевірки відповідності паролю та імені, ми можемо змінити логіку контролю потоку програми замінивши стрибок `JZ` на `JNZ` аби отримувати сесію користувача при введенні неправильного паролю.
   Це місце можна знайти у дизасембльованому коді за адресою знаходження опкоду

`0f 84 26 01 00 00` - дезгідно документації intel x86 `0f 84` відповідає за тип стрибку (Jump near if 0 (ZF=1))



Ми можемо пропатчити бінарник за допомогою скрипта написаного мовою пайтон, знайти необхідну адресу опкоду, та замінити його на протилежний отримуючи `0f 85` що відповідає за інструкцію `JNZ`.

В результаті виконання завдання був отриманий наступний скрипт.

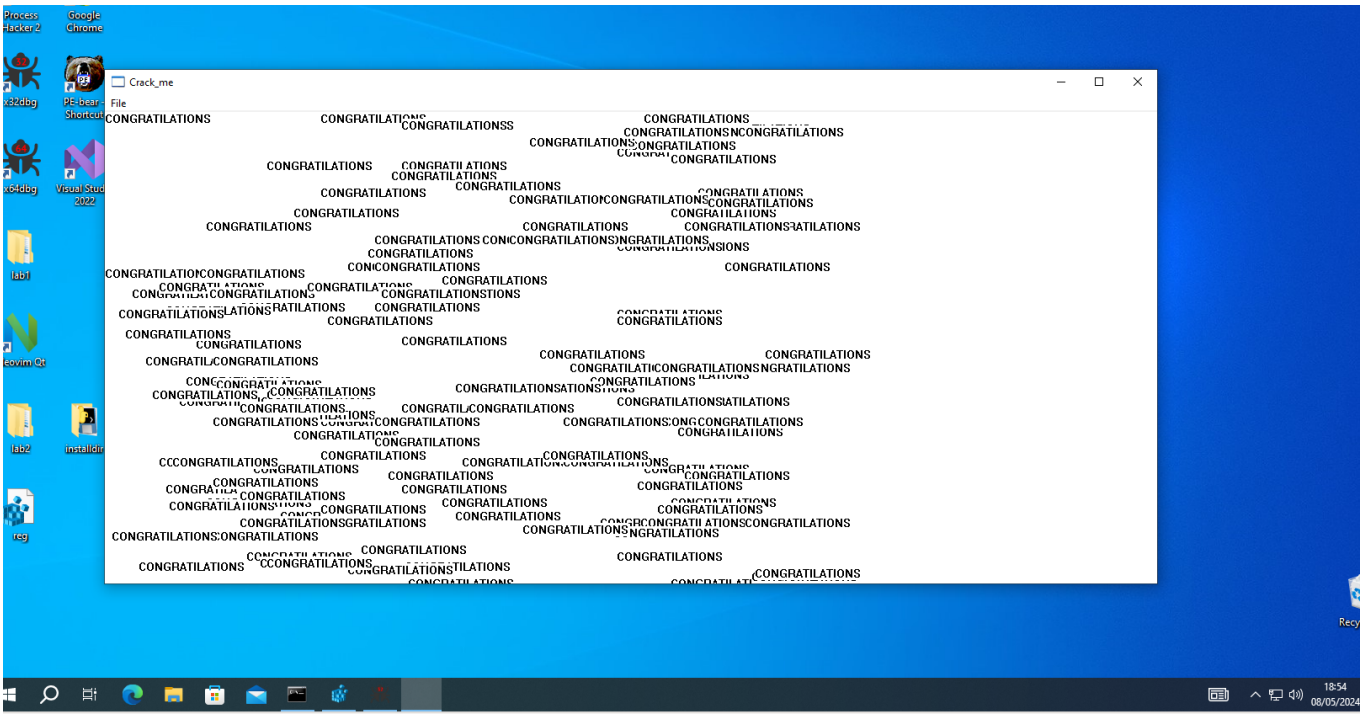
```
#!/usr/bin/env python3
from pwn import *

ORG_FILENAME = "./Crack_me_up!.exe"
OUT_FILENAME = "./patched.exe"
JZ_SEQ = b'\x0f\x84\x25\x01\x00\x00'

context.arch = 'i386'

def main() -> None:
    with open(ORG_FILENAME, 'rb') as f:
        file = f.read()

    print(disasm(JZ_SEQ))
    patched_seq = JZ_SEQ[:1] + b'\x85' + JZ_SEQ[2:]
    print(disasm(patched_seq))

    j_offset = file.find(JZ_SEQ)

    patched_bin = file[:j_offset] + patched_seq +
file[j_offset+len(patched_seq):]
    assert len(patched_bin) == len(file)

    with open(OUT_FILENAME, 'wb') as f:
        f.write(patched_bin)
```

```
if __name__ == "__main__":
    main()
```

Результат виконання.

```
~/Documents/progsec/lab4
./solve.py
   0:   0f 84 25 01 00 00       je     0x12b
   0:   0f 85 25 01 00 00       jne    0x12b

~/Documents/progsec/lab4
file Crack_me_up\!.exe
Crack_me_up!.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 5 sections

~/Documents/progsec/lab4
file patched.exe
patched.exe: PE32 executable (GUI) Intel 80386, for MS Windows, 5 sections
```

Запущений бінарний дозволяє залогінитись з неправильним паролем.



Як результат отримуємо спам `CONGRATULATIONS`

3. Розберемо ділянку коду що відповідає за перевірку паролю.

Саме тут ми можемо спостерігати алгоритм що обробляє введене ім'я та на його результаті формує пароль.

Алгоритм скаладається з двух функцій що послідовно викликаються під час перевірки паролю. В них виконується ряд математичних перетвореннь включаючи здвиги та ксор.

Функція 1.

```
void __cdecl name_transform(int username,uint *output,int strlen_username)

{
  int counter;

  *output = 0xfacc0fff;
  counter = 0;
  if (0 < strlen_username) {
    do {
      *output = ((uint)*(byte *)(username + counter) ^ *output) << 8 | *output >> 0x18;
      counter = counter + 1;
    } while (counter < strlen_username);
  }
  return;
}
```

Функція 2

```
1
2  void __cdecl FUN_004011b0(uint param_1,int name_transformed)
3
4  {
5    uint _Value;
6    int counter;
7    uint local_8;
8
9    local_8 = 0;
10   counter = 0;
11   do {
12     _Value = param_1 & 0xf;
13     param_1 = param_1 >> 4;
14     if (9 < _Value) {
15       _Value = 9;
16     }
17     __itoa(_Value,(char *)&local_8,10);
18     *(undefined *)(counter + name_transformed) = (undefined)local_8;
19     counter = counter + 1;
20   } while (counter < 8);
21   return;
22 }
23
```

Не важко відтворити спостережений алгоритм засобами мови python. При створенні функцій що відповідатимуть функціям застосунку мовою C важливо звернути увагу на переповнення змінних і тримати їх в межі 4 байт для `uint`, адже в python немає переповнення int змінної, що може суттєво вплинути на генерацію паролю.

В результаті розробки було отримано наступні функції:

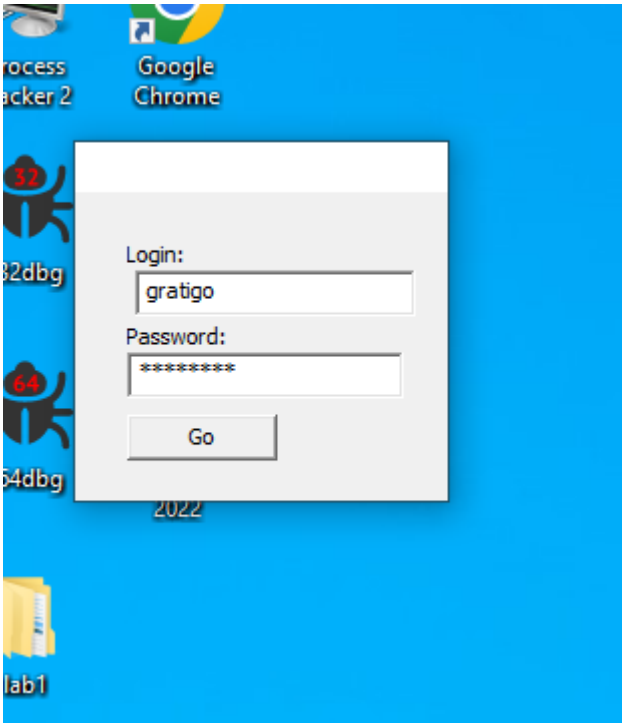
```
#!/usr/bin/env python3
from pwn import *
import logging

logging.basicConfig(level=logging.DEBUG)
logging.getLogger("pwnlib").setLevel(logging.WARNING)

...
<SNIP>
...

def transformName(username:str) -> int:
        username = username.encode()
        strlen_username = len(username)
        output = 0xfacc0fff
        for counter in range(strlen_username):
                output = ((username[counter] ^ output) << 8) | (output >>
0x18)
                output = output & 0xffffffff
        return output



def getPass(name_transformed:int) -> str:
        password = ''
        for _ in range(8):
                low_byte = name_transformed & 0xf
                if 9 < low_byte:
                        low_byte = 9
                password += str(low_byte)
                name_transformed = name_transformed >> 4
        return password

...
<SNIP>
...

def main() -> None:
        name = "gratigo"
        logging.info(f"Username: {name}")
        name_trasnformed = transformName(name)
        passwd = getPass(name_trasnformed)
        logging.info(f"Password: {passwd}")
```

```
if __name__ == "__main__":
        main()
```

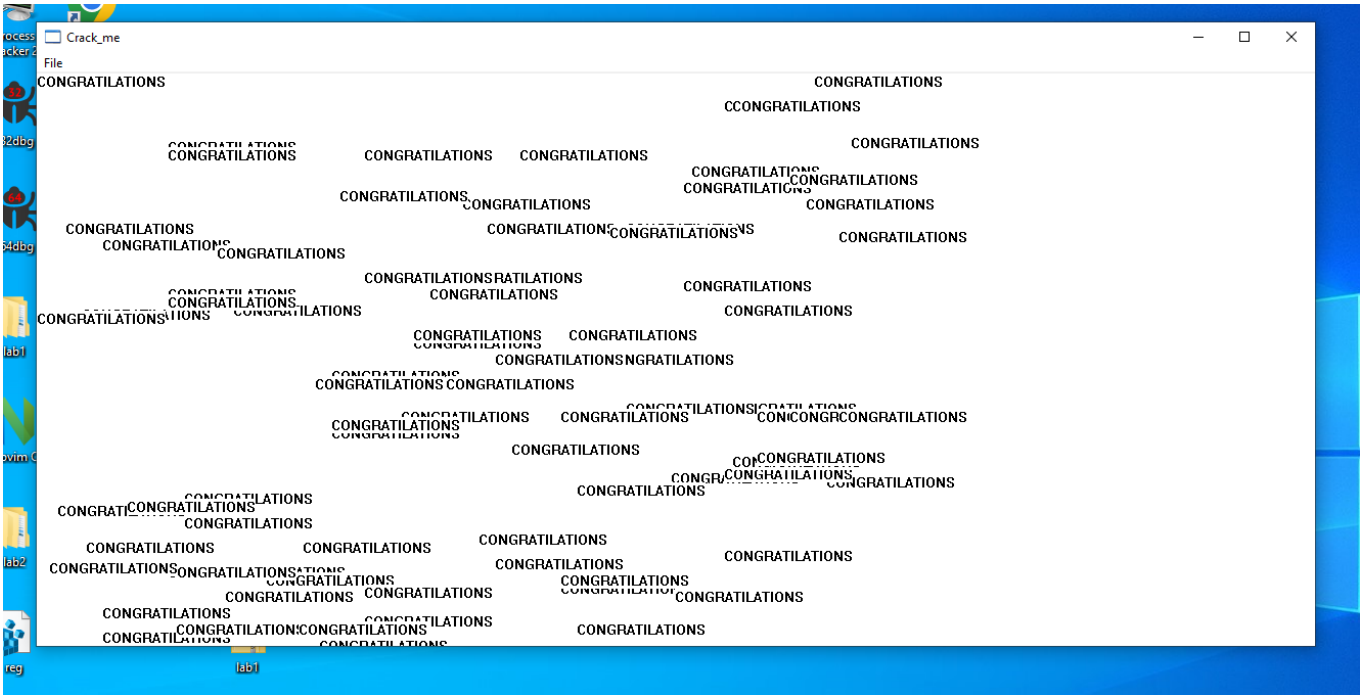
За допомогою них було згенеровано пароль для довільного користувацького імені.

```
$ ./solve.py
INFO:root:Username: gratigo
INFO:root:Password: 97299919
```

4. Використавши оригінальний файл застосунку ми можемо отримати логін за довільного користувача (в прикладі `gratigo`) надавши відповідний згенерований пароль.

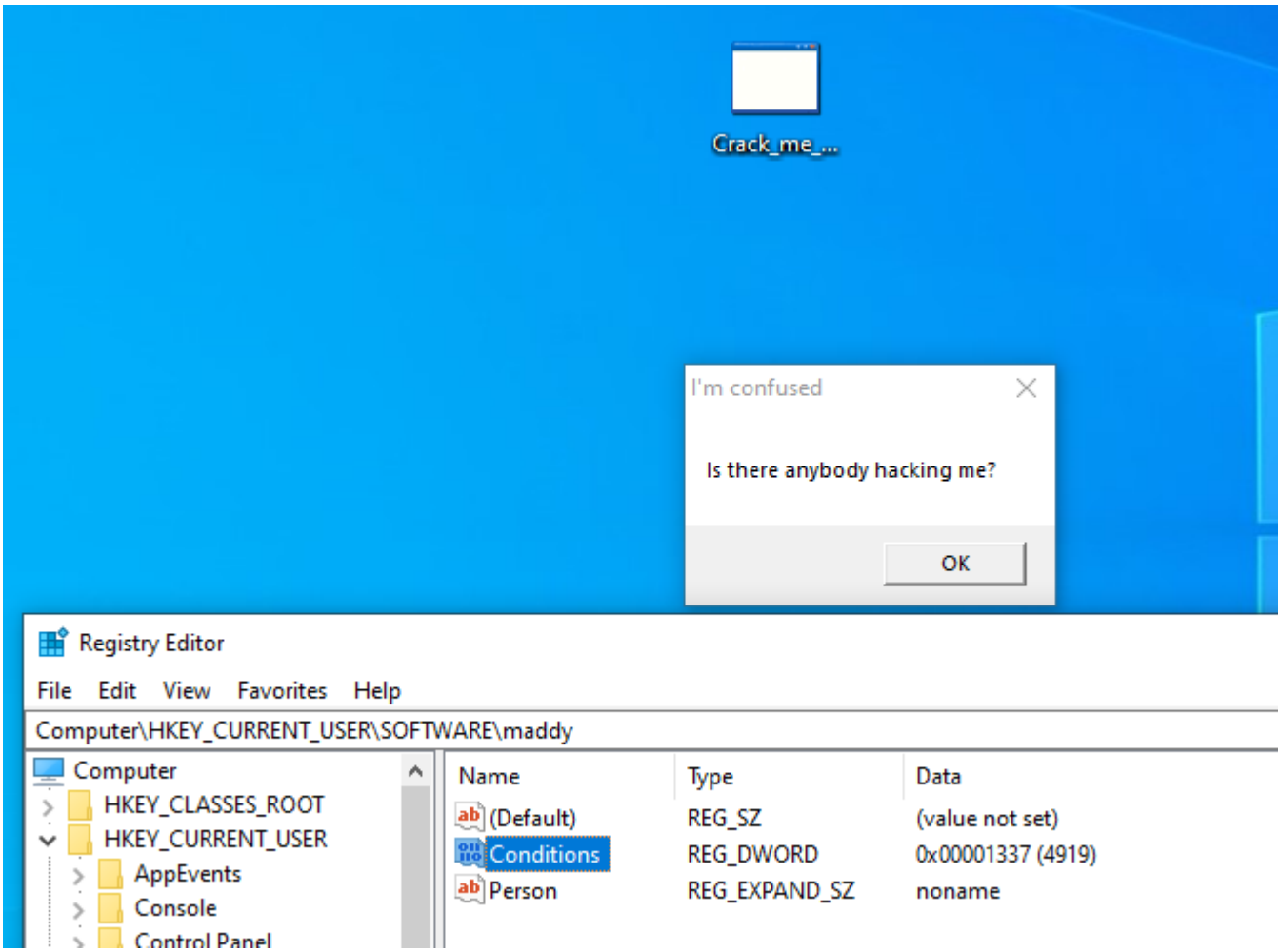


Результат виконання такий самий як і при обході логіну.



5. Переглянувши потік виконання програми легко помітити де використовуються змінні реєстру, і який сенс в них закладений. На наступному шматку декомпільованого коду видно що змінна `Conditions` відповідає за кількість спроб. Вона має бути меншою за `0x1e == 30` та більшою за 0. При кожній спробі запуску застосунку її значення зменшується на одиницю. При спробі виставити занадто велику кількість спроб через зміну реєстру отримуємо відповідне повідомлення.

```
_00401455:
  if (iVar4 == 0) {
    cond_res = RegQueryValueExA(local_810,"Conditions",(LPDWORD)0x0,&local_81c,(LPBYTE)&cond,
                                &local_818);
    if (cond_res != 0) {
      MessageBoxA((HWND)0x0,"Sorry, but in this condition, I\'ll not working","SUXX",0);
      UVar9 = 3;
      pvVar2 = GetCurrentProcess();
      TerminateProcess(pvVar2,UVar9);
    }
    if (0x1e < cond) {
      MessageBoxA((HWND)0x0,"Is there anybody hacking me?","I\'m confused",0);
      UVar9 = 3;
      pvVar2 = GetCurrentProcess();
      TerminateProcess(pvVar2,UVar9);
    }
    if (cond == 0) {
      MessageBoxA((HWND)0x0,"Sorry, Michael, Buzziness :), BYE :)","Let the SUXX be with you :)"
                  ,1);
      UVar9 = 5;
      pvVar2 = GetCurrentProcess();
      TerminateProcess(pvVar2,UVar9);
    }
    cond = cond - 1;
    cond_res = RegSetValueExA(local_810,"Conditions",0,4,(BYTE *)&cond,4);
    if (cond_res != 0) {
      MessageBoxA((HWND)0x0,"Sorry, but in this condition, I\'ll not working","SUXX",0);
      UVar9 = 3;
      pvVar2 = GetCurrentProcess();
      TerminateProcess(pvVar2,UVar9);
    }
```



Для обходу блокування роботи ми можемо пропатчити бінарник змінивши логіку потоку виконання на стрибки `JMP` заміть логічний стрибків на місцях перевірки змінної `Conditions` аби повністю прибрати логіку перевірки змінної.

Також у застосунку передбачено встановлення імені логіну після успішного входу, замість `noname` у змінну реєстру `Person`. При тому відповідні дані для перевірки записуються у локальний файл `reg.reg`. Ми можемо додатково обійти цю перевірку, аби без обмеженнь логінитись як будь-який користувач після перезапуску застосунку. Для цього ми можемо забити `nop` байтами стрибок який викликає помилку при невідповідності імені реєстру та файлу `reg.reg`.

6. У результаті виконання завдання скрипт розроблений раніше був доповнений необхідним функціоналом, що дозволяє провести логін як будь який користувач за знерованим паролем, без обмеженнь на перевірки за файлом чи реєстром. Вихідний код:

```
#!/usr/bin/env python3
from pwn import *
import logging
```

```
logging.basicConfig(level=logging.DEBUG)

logging.getLogger("pwnlib").setLevel(logging.WARNING)

ORG_FILENAME = "./Crack_me_up!.exe"
OUT_FILENAME = "./patched.exe"
JZ_SEQ = b'\x0f\x84\x25\x01\x00\x00'

LAUNCH_COND_SEQ_JBE = b'\x76\x17\x6a\x00\x68\xc8\xd2\x40\x00'
LAUNCH_COND_SEQ_JNZ = b'\x75\x17\x6a\x01\x68\x8c\xd2\x40\x00'

CHECK_NAME_SEQ_JNZ = b'\x0f\x85\xdb\x00\x00\x00'


context.arch = 'i386'

def transformName(username:str) -> int:
    username = username.encode()
    strlen_username = len(username)
    output = 0xfacc0fff
    for counter in range(strlen_username):
        output = ((username[counter] ^ output) << 8) | (output >> 0x18)
        output = output & 0xffffffff
    return output

def getPass(name_transformed:int) -> str:
    password = ''
    for _ in range(8):
        low_byte = name_transformed & 0xf
        if 9 < low_byte:
            low_byte = 9
        password += str(low_byte)
        name_transformed = name_transformed >> 4
    return password




def getPatch(name_bypass:bool = False, login_bypass:bool = False) -> None:
    with open(ORG_FILENAME, 'rb') as f:
        file = f.read()

    patched_bin = file

    if login_bypass:
        logging.info(f"Applying login bypass patch")
```

```
        logging.debug(f"Login bypass disasm:")
        logging.debug(disasm(JZ_SEQ))
        patched_seq = JZ_SEQ[:1] + b'\x85' + JZ_SEQ[2:]
        logging.debug(disasm(patched_seq))

        j_offset = file.find(JZ_SEQ)

        patched_bin = file[:j_offset] + patched_seq +
file[j_offset+len(patched_seq):]

    cond_jbe = file.find(LAUNCH_COND_SEQ_JBE)
    cond_jnz = file.find(LAUNCH_COND_SEQ_JNZ)

    logging.debug(f"Launch conditions disasm:")
    logging.debug(disasm(LAUNCH_COND_SEQ_JBE[:2]))
    logging.debug(disasm(LAUNCH_COND_SEQ_JNZ[:2]))

    patched_seq_jbe = b'\xeb' + LAUNCH_COND_SEQ_JBE[1:2]
    patched_seq_jnz = b'\xeb' + LAUNCH_COND_SEQ_JNZ[1:2]
    logging.debug(f"Patch disasm:")
    logging.debug(disasm(patched_seq_jbe))
    logging.debug(disasm(patched_seq_jnz))

    jbe_offset = file.find(LAUNCH_COND_SEQ_JBE)
    jnz_offset = file.find(LAUNCH_COND_SEQ_JNZ)

    patched_bin = patched_bin[:jbe_offset] + patched_seq_jbe +
patched_bin[jbe_offset+len(patched_seq_jbe):]
    patched_bin = patched_bin[:jnz_offset] + patched_seq_jnz +
patched_bin[jnz_offset+len(patched_seq_jnz):]

    if name_bypass:
        logging.info(f"Applying name bypass patch")
        logging.debug(f"Check 'noname' disasm:")
        logging.debug(disasm(CHECK_NAME_SEQ_JNZ))

        check_name_offset = file.find(CHECK_NAME_SEQ_JNZ)
        patched_seq_check_name = b'\x90' * len(CHECK_NAME_SEQ_JNZ)
        logging.debug(f"Patch disasm:")
        logging.debug("\n" + disasm(patched_seq_check_name))

        patched_bin = patched_bin[:check_name_offset] +
patched_seq_check_name +
patched_bin[check_name_offset+len(patched_seq_check_name):]
```

```
    assert len(patched_bin) == len(file)
    with open(OUT_FILENAME, 'wb') as f:
        f.write(patched_bin)

def main() -> None:
    getPatch(True)

    name = "gratigo"
    logging.info(f"Username: {name}")
    name_trasnformed = transformName(name)
    passwd = getPass(name_trasnformed)
    logging.info(f"Password: {passwd}")


if __name__ == "__main__":
    main()
```

Результат виконання

```
$ ./solve.py
DEBUG:root:Launch conditions disasm:
DEBUG:root:   0:   76 17                    jbe    0x19
DEBUG:root:   0:   75 17                    jne    0x19
DEBUG:root:Patch disasm:
DEBUG:root:   0:   eb 17                    jmp    0x19
DEBUG:root:   0:   eb 17                    jmp    0x19
INFO:root:Applying name bypass patch
DEBUG:root:Check 'noname' disasm:
DEBUG:root:   0:   0f 85 db 00 00 00        jne    0xe1
DEBUG:root:Patch disasm:
DEBUG:root:
   0:   90                      nop
   1:   90                      nop
   2:   90                      nop
   3:   90                      nop
   4:   90                      nop
   5:   90                      nop
INFO:root:Username: gratigo
INFO:root:Password: 97299919
```