

PAC 2. Vulnerabilitats en el programari

Sílvia Sanvicente García

10 novembre 2021

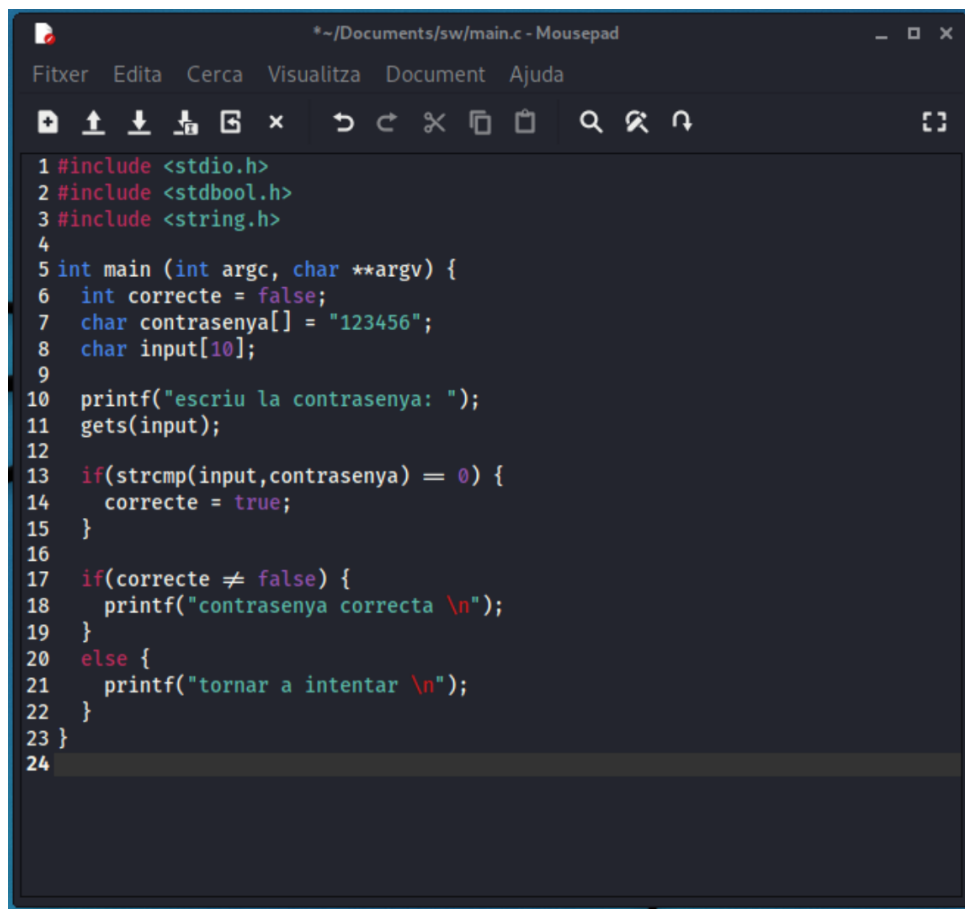
Índex

Funcions vulnerables	2
Realitza un programa senzill en el qual es faci servir una funció vulnerable	2
Mostra i raona la vulnerabilitat en temps d'execució	2
Vulnerabilitats Format string vs Stack overflow	4
Vulnerabilitats de tipus Format string	4
Exploitar un Format string Bug	5
Similituds i/o diferències entre Format string i Stack overflow	5
Programa senzill amb una vulnerabilitat de tipus Format string	6
Conclusions	7
Referències	7

Funcions vulnerables

Realitza un programa senzill en el qual es faci servir una funció vulnerable

Per fer aquesta PAC s’ha creat un petit programa que valida si una contrasenya és correcta, el qual es pot veure a la figura 1. La principal vulnerabilitat d’aquest programa és que utilitza la funció *gets*, la qual no comprova la mida del *buffer*. A banda d’aquesta vulnerabilitat, també s’està usant la funció *strcmp* que tampoc no valida la mida del *buffer*. A l’hora de verificar si la contrasenya inserida és correcta, es mira si la variable “correcte” ha deixat de ser *false*. Això també suposa una vulnerabilitat i s’hauria de mirar directament si ara és *true* per evitar que qualsevol valor diferent de *false*, encara que no sigui *true*, doni la contrasenya com a correcta.

A screenshot of a code editor window titled '*~/Documents/sw/main.c - Mousepad'. The editor has a menu bar with 'Fitxer', 'Edita', 'Cerca', 'Visualitza', 'Document', and 'Ajuda'. Below the menu is a toolbar with icons for file operations and editing. The code is written in C and is as follows:

```
1 #include <stdio.h>
2 #include <stdbool.h>
3 #include <string.h>
4
5 int main (int argc, char **argv) {
6     int correcte = false;
7     char contrasenya[] = "123456";
8     char input[10];
9
10    printf("escriu la contrasenya: ");
11    gets(input);
12
13    if(strcmp(input,contrasenya) == 0) {
14        correcte = true;
15    }
16
17    if(correcte != false) {
18        printf("contrasenya correcta \n");
19    }
20    else {
21        printf("tornar a intentar \n");
22    }
23 }
24
```

Figura 1: Codi amb funcions vulnerables

Mostra i raona la vulnerabilitat en temps d’execució

Per analitzar la vulnerabilitat en temps d’execució, s’ha compilat el codi amb la següent instrucció:

```
i686-w64-mingw32-gcc -o main32.exe main.c
```

Amb l’eina OllyDbg podem obtenir el codi en llenguatge ensamblador, veure els registres i analitzar les vulnerabilitats en temps d’execució. La figura 2 mostra el codi en llenguatge ensamblador del programa creat prèviament.

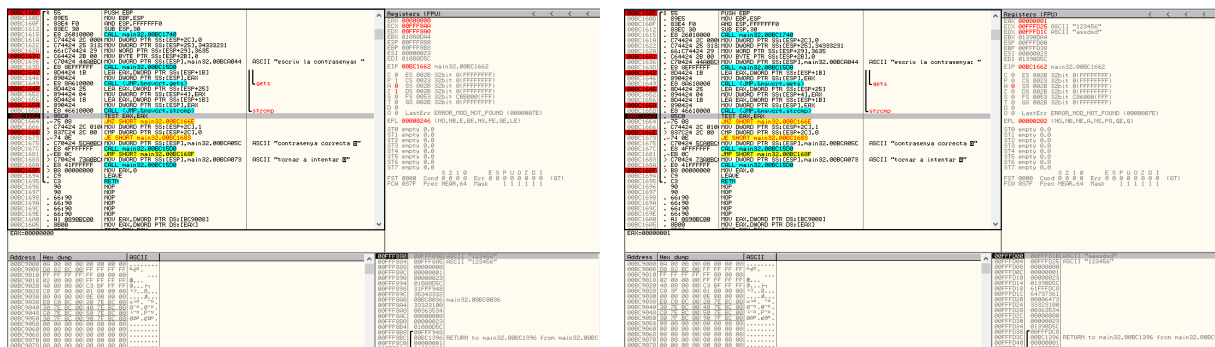
```

00BB160C  $ 55          PUSH EBP
00BB160D  . 89E5        MOV EBP,ESP
00BB160F  . 83E4 F0     AND ESP,FFFFFFF0
00BB1612  . 83EC 30     SUB ESP,30
00BB1615  . E8 26010000 CALL main32.00BB1740
00BB161A  . C74424 2C 0001 MOV DWORD PTR SS:[ESP+2C],0
00BB1622  . C74424 25 3131 MOV DWORD PTR SS:[ESP+25],34333231
00BB162A  . 66:C74424 29:01 MOV WORD PTR SS:[ESP+29],3635
00BB1631  . C64424 2B 00 MOV BYTE PTR SS:[ESP+2B],0
00BB1636  . C70424 4400BB1636 MOV DWORD PTR SS:[ESP],main32.00BBA044
00BB163D  . E8 8EFFFFFF CALL main32.00BB1500
00BB1642  . 8D4424 1B LEA EAX,DWORD PTR SS:[ESP+1B]
00BB1646  . 890424      MOV DWORD PTR SS:[ESP],EAX
00BB1649  . E8 8A610000 CALL <JMP.&msvcrt.gets>
00BB164E  . 8D4424 25 LEA EAX,DWORD PTR SS:[ESP+25]
00BB1652  . 894424 04 MOV DWORD PTR SS:[ESP+4],EAX
00BB1656  . 8D4424 1B LEA EAX,DWORD PTR SS:[ESP+1B]
00BB165A  . 890424      MOV DWORD PTR SS:[ESP],EAX
00BB165D  . E8 46610000 CALL <JMP.&msvcrt.stromp>
00BB1662  . 85C0      TEST EAX,EAX
00BB1664  . 75 08     JNZ SHORT main32.00BB166E
00BB1666  . C74424 2C 0101 MOV DWORD PTR SS:[ESP+2C],1
00BB166E  . 837C24 2C 00 CMP DWORD PTR SS:[ESP+2C],0
00BB1673  . 74 0E     JE SHORT main32.00BB1683
00BB1675  . C70424 5CA0BB1675 MOV DWORD PTR SS:[ESP],main32.00BBA05C
00BB167C  . E8 4FFFFFFF CALL main32.00BB1500
00BB1681  . EB 0C     JMP SHORT main32.00BB168F
00BB1683  . C70424 7300BB1683 MOV DWORD PTR SS:[ESP],main32.00BBA073
00BB168A  . E8 41FFFFFF CALL main32.00BB1500
00BB168F  . B8 00000000 MOV EAX,0
00BB1694  . C9       LEAVE
00BB1695  . C3       RETN

```

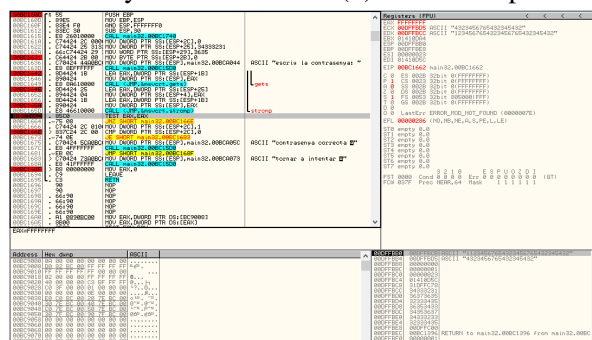
Figura 2: Codi assemblador del codi amb funcions vulnerables

Per estudiar el codi en temps d'execució ho hem executat tres vegades. Una primera vegada inserint la contrasenya correcta, una segona amb una contrasenya incorrecta i una tercera amb una contrasenya massa llarga desbordant el *buffer*, modificant així les adreces de memòria veïnes.



(a) Cas on l'input és la contrasenya correcta

(b) Cas on l'input no és la contrasenya correcta



(c) Cas on forcem l'input per desbordar el buffer

Figura 3: Diferents execucions del codi per analitzar els diferents casos

A la figura 3 podem observar l'estat de la pila i els registres a les diferents execucions. Al registre *ECX*, es desa la contrasenya correcta amb què es compararà l'input, la qual és "123456". Al registre *EDX* es guarda l'input i al registre *EAX* el resultat de comprar la contrasenya amb l'input. En el cas que hem inserit un input massa gran, desbordant el *buffer*, el registre *ECX* que contenia la contrasenya real s'ha sobreescrit amb part de l'input. Això passa perquè hem utilitzat la funció *gets*, la qual no valida la mida del *buffer*.

Si observem el registre *EAX*, podem veure el retorn de la funció *strcmp*. En el cas de la contrasenya correcta, es desa el valor 0. Quan inserim una contrasenya incorrecta, però d'una mida vàlida se'n guarda el valor 1, però en el cas del desbordament del *buffer*, podem observar que es guarda el valor "FFFFFFFF" en hexadecimal. En el programa en C guardem el resultat de la comparació a la variable "correcte" i simplement verifiquem si aquesta ha deixat de ser *false* (que correspon a 1). Això fa que tot i inserir un input que no coincideix amb la contrasenya, accedim a la part del codi de "contrasenya correcta".

A la figura 4 podem observar que s'ha validat la contrasenya tot i que l'input no era correcte.

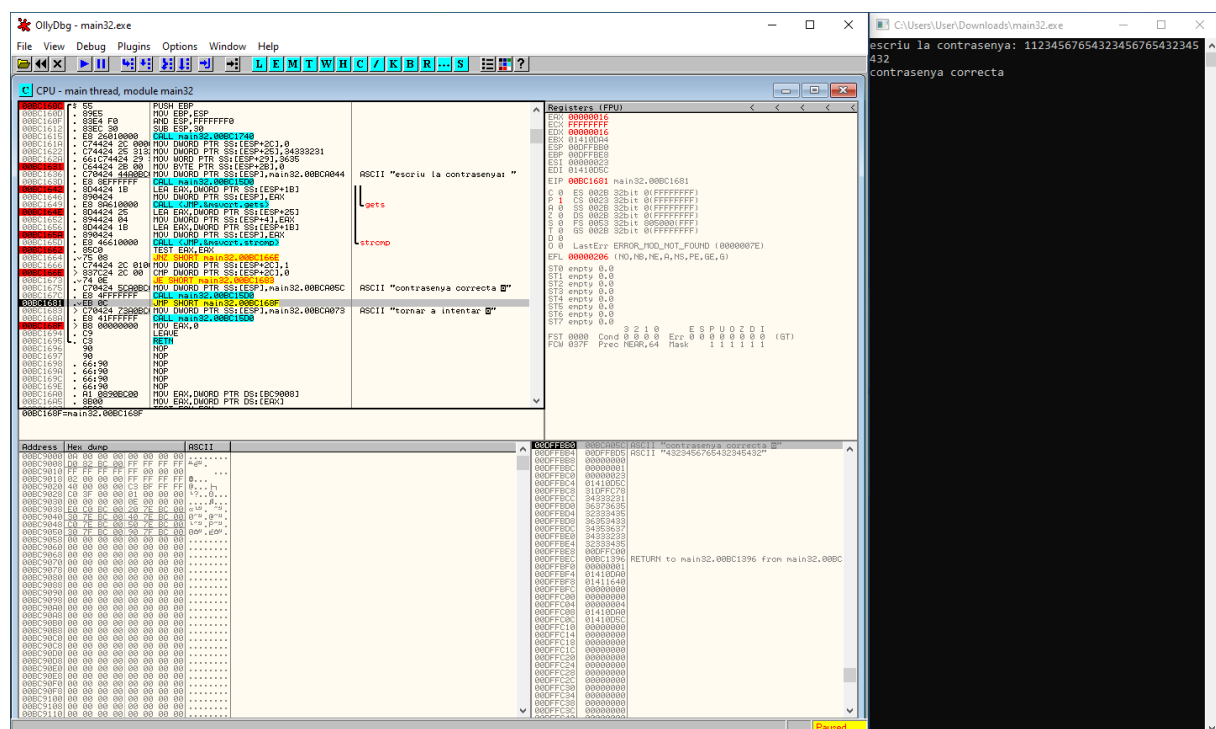


Figura 4: Accedim a la part del codi de contrasenya correcta amb un input invàlid

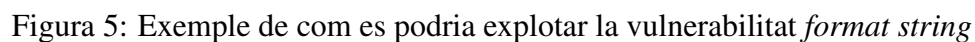
Vulnerabilitats Format string vs Stack overflow

Vulnerabilitats de tipus Format string

Hi ha diverses funcions que donen format a tipus de dades primitives i les escriuen per una sortida, com ara la funció *printf* que imprimeix un missatge per pantalla. Aquestes funcions poden tenir diversos arguments. El primer argument s'anomena *format string* i és on incloem el text a imprimir i els *format parameters* a reemplaçar pels arguments addicionals. Quan es

```
1 printf ("Any %d", 2021)
```

Per explicar com es podria explotar aquesta vulnerabilitat farem servir l'exemple de la fundació de ciberseguretat OWASP [5].



Un *Stack overflow* succeeix quan en escriure dades en un *buffer*, se supera la capacitat d'aquest, sobreescrivint així les adreces de memòria veïnes. Normalment, un *Stack overflow* succeeix quan utilitzem funcions que no validen la mida del *buffer* o quan fixem una mida de *buffer* insuficient. La similitud entre *Format string* i *Stack overflow* és que tots dos sobreescriuen fragments de memòria que no s'haurien de modificar provocant que el programa no funcioni correctament o que un atacant pugui manipular les dades de la pila. Tot i això, no es consideren el mateix tipus d'atac, ja que el *Format string* s'aprofita de funcions que prenen un nombre

variable d'arguments i el *Stack overflow* s'aprofita de funcions que no comproven la mida del *buffer*.

Programa senzill amb una vulnerabilitat de tipus Format string

Per estudiar en temps d'execució la vulnerabilitat de tipus *format string* hem escrit un codi molt senzill que no utilitza els *format parameters* adequadament en la funció *printf*.

```
1 #include <stdio.h>
2 int main(int argc, char *argv[]){
3     printf("%x %x %x\n");
4     printf("you are here \n");
5 }
```

El string “you are here” ens permet localitzar de manera fàcil la funció dins l'eina OllyDbg. A la figura 6 podem veure el codi en llenguatge ensamblador.

00A2160C	55	PUSH EBP	
00A2160D	89E5	MOV EBP,ESP	
00A2160F	83E4 F0	AND ESP,FFFFFFF0	
00A21612	83EC 10	SUB ESP,10	
00A21615	E8 C6000000	CALL ex2_1.00A216E0	
00A2161A	C70424 44A0A2	MOV DWORD PTR SS:[ESP],ex2_1.00A2A044	ASCII "%x %x %x"
00A21621	E8 A9FFFFFF	CALL ex2_1.00A215D0	
00A21626	C70424 4EA0A2	MOV DWORD PTR SS:[ESP],ex2_1.00A2A04E	ASCII "you are here "
00A2162D	E8 9EFFFFFF	CALL ex2_1.00A215D0	
00A21632	B8 00000000	MOV EAX,0	
00A21637	C9	LEAVE	
00A21638	C3	RETN	

Figura 6: Codi ensamblador del codi amb una vulnerabilitat de tipus *format string*

La funció *printf* no coneix quants arguments rep, sinó que determina el nombre d'arguments per la quantitat de *format parameters* presents al *format string*. Quan arriba a un *format parameter*, el reemplaça pel contingut de la pila.

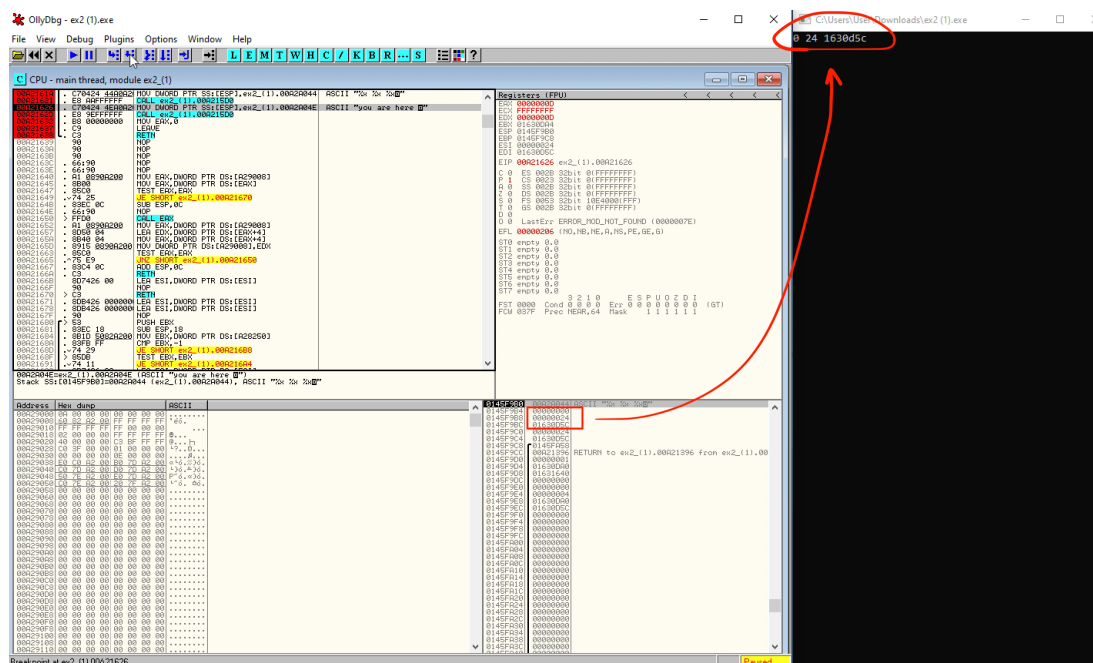


Figura 7: Anàlisi de la pila durant l'execució del codi

En aquest cas, no detecta `%x` com una cadena de caràcters sinó com un *unsigned int* (*hexadecimal*). Tal com podem veure a la figura 7, quan es crida la funció *printf*, aquesta mapeja els `%x` a les darreres tres instruccions que tenim a la pila.

Conclusions

En llenguatges com C, hi ha diverses funcions que són vulnerables i el seu ús no està recomanat. En aquesta PAC hem pogut estudiar algunes d'aquestes funcions com ara *gets* o *strcpy*. Aquest tipus de funcions no comproven la mida del *buffer* i poden ocasionar un *Stack overflow* que impedeixi al programa funcionar correctament o fins i tot donar la possibilitat de llegir o escriure a la pila de manera malintencionada. També s'han estudiat funcions vulnerables de tipus *Format string* que tenen arguments variables i poden ser manipulades si no es construeixen correctament. Gràcies a l'eina OllyDbg hem pogut analitzar aquestes vulnerabilitats a temps real, veient com s'actualitzen els registres i la pila.

Referències

- [1] Josep Vañó Chic, Eines (setembre 2014). Disponible a:
https://materials.campus.uoc.edu/daisy/Materials/PID_00208391/pdf/PID_00208380.pdf
- [2] Josep Vañó Chic, Codi segur (setembre 2014). Disponible a:
https://materials.campus.uoc.edu/daisy/Materials/PID_00217403/pdf/PID_00217346.pdf
- [3] CERN Computer Security Information. (2021). Disponible a:
<https://security.web.cern.ch/recommendations/en/codetools/c.shtml>
- [4] OllyDbg v1.10 — OllyDbg. (2021). Disponible a:
<https://www.ollydbg.de/>
- [5] Format String Software Attack — OWASP Foundation. (2021). Disponible a:
https://owasp.org/www-community/attacks/Format_string_attack
- [6] Buffer Overflow Software Attack — OWASP Foundation. (2021). Disponible a:
https://owasp.org/www-community/attacks/Buffer_overflow_attack