Codi segur

Josep Vañó Chic

PID_00217346

Universitat Oberta
de Catalunya

Els textos i imatges publicats en aquesta obra estan subjectes –llevat que s'indiqui el contrari– a una llicència de Reconeixement-NoComercial-SenseObraDerivada (BY-NC-ND) v.3.0 Espanya de Creative Commons. Podeu copiar-los, distribuir-los i transmetre'ls públicament sempre que en citeu l'autor i la font (FUOC. Fundació per a la Universitat Oberta de Catalunya), no en feu un ús comercial i no en feu obra derivada. La llicència completa es pot consultar a http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.ca

Índex

Introducció.. 5

Objectius.. 6

1. *Integer Overflow*.. 7
- 1.1. Representació dels nombres 8
- 1.2. Desbordament de tipus de dada 9
- 1.3. Atacs *integer overflow*.. 12

2. Desbordament de pila (*stack overvlow*).. 15
- 2.1. Els registres 15
- 2.2. Gestió de la pila 16
- 2.3. Crida i retorn de funcions 16
- 2.4. Exemple de desbordament de pila 17

3. Desbordament de *heap*.. 22

4. Funcions vulnerables.. 26

Bibliografia.. 33

Introducció

Un programari que funciona correctament és aquell que fa exactament tot allò pel que va ser creat i dissenyat. No obstant això, el programa pot ser correcte des del punt de vista funcional però a la vegada pot ser insegur.

Els errors en el programari poden ser utilitzats per a atacar el sistema i posar-ne en perill el bon funcionament, així com la confidencialitat i l'ús de les dades que hi ha emmagatzemades, a més a més, els errors poden ser utilitzats com a porta d'entrada per a executar codi maliciós.

El programari realitzat amb codi de programació no segur és fàcilment vulnerable, els *hackers* aprofiten aquestes vulnerabilitats per a provocar l'error i entrar en el sistema de l'ordinador atacat.

En aquest mòdul es podrà constatar el com, el perquè i les implicacions que comporta el codi no segur, a la vegada, es mostra la utilització d'eines per tal de inspeccionar el funcionament del programari durant la seva execució i així observar en quin moment és vulnerable i quins són els motius d'aquesta vulnerabilitat.

Així doncs és important ser conscients dels perills que poden comportar aquests errors per tal de posar els mitjans adients perquè no es produeixin.

Objectius

En finalitzar la lectura d'aquest material, els estudiants hauran aconseguit les competències següents:

1. Conèixer el risc de la programació de codi no segur.
2. Conèixer els principals tipus de codi no segur.
3. Conèixer com i per què es produeixen els errors de ***Integer Overflow.***
4. Conèixer com i per què es produeixen els errors de ***Buffer Overflow.***
5. Saber identificar la diferència entre els diversos tipus de ***Buffer Overflow.***
6. Conèixer les funcions no segures.
7. Saber les implicacions que comporta la programació no segura.
8. Utilitzar les eines de depuració i desassemblatge per a inspeccionar el funcionament de programari.

1. *Integer Overflow*

L'***Integer Overflow*** succeeix quan una operació aritmètica intenta crear un valor numèric que és massa gran per ser representat en l'espai d'emmagatzematge que té assignat.

En programació, una variable és un espai de memòria reservat per a emmagatzemar un valor que correspon a un tipus de dades suportat pel llenguatge de programació.

Els llenguatges de programació disposen de diversos tipus de variables, i la mida de l'espai de memòria reservat per a la variable anirà en funció del tipus de variable que es defineixi.

Per exemple, en ANSI C, les variables i les seves mides són:

Tipus	Rang de valors	Mida
char	De -128 a 127	8 bits
unsigned char	De 0 a 255	8 bits
short	De -32.768 a 32767	16 bits
unsigned short	De 0 a 65.535	16 bits
int	De -2.147.483.648 a 2.147.483.647	32 bits

A continuació es mostra un programa en C on es reflecteixen les mides d'aquests tipus de dades. Cal tenir en compte que el resultat pot variar d'un ordinador a un altre depenent de la versió del compilador que s'utilitzi i l'arquitectura de l'equip. Aquest exemple ha estat realitzat en un entorn virtualitzat en ***Oracle VM VirtualBox***, utilitzant el compilador **Cygwin GCC** a través de l'IDE de ***Code Blocks*** sobre un sistema operatiu Windows 7 Professional de 32 bits.

Aquest programa mostra el rang de valors mínims i màxims de diversos tipus de variables que es defineixen; al mateix temps, el programa mostra la seva representació en hexadecimal i la seva mida en bits.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
     char c =  127;              // 2^7 -1
     unsigned char uc = 255;     // 2^8 -1
```

```
    short s =  32767;           // 2^15 -1
    unsigned short us = 65535;  // 2^16 -1
    int i = 2147483647;         // 2^31 -1
    printf("Maximum values\n");
    printf("char           = %d (0x%x) [%d bits]\n", c, c, sizeof(c) * 8);
    printf("unsigned char  = %d (0x%x) [%d bits]\n", uc, uc, sizeof(uc) * 8);
    printf("short          = %d (0x%x) [%d bits]\n", s, s, sizeof(s) * 8);
    printf("unsigned short = %d (0x%x) [%d bits]\n", us, us, sizeof(us) * 8);
    printf("int            = %d (0x%x) [%d bits]\n", i, i, sizeof(i) * 8);

    c =  -128;        // -2^7
    s =  -32768;      // -2^15
    i = -2147483648;  // -2^31
    printf("\nMinimum values\n");
    printf("char           = %d (0x%x) [%d bits]\n", c, c, sizeof(c) * 8);
    printf("short          = %d (0x%x) [%d bits]\n", s, s, sizeof(s) * 8);
    printf("int            = %d (0x%x) [%d bits]\n", i, i, sizeof(i) * 8);

    return 0;
}
```

sizes.c

El resultat de l'execució del programa és el següent:

```
Maximum values
char           = 127 (0x7f) [8 bits]
unsigned char  = 255 (0xff) [8 bits]
short          = 32767 (0x7fff) [16 bits]
unsigned short = 65535 (0xffff) [16 bits]
int            = 2147483647 (0x7fffffff) [32 bits]

Minimum values
char           = -128 (0xffffff80) [8 bits]
unsigned char  = 0 (0x0) [8 bits]
short          = -32768 (0xffff8000) [16 bits]
unsigned short = 0 (0x0) [16 bits]
int            = -2147483648 (0x80000000) [32 bits]
```

1.1. Representació dels nombres

Com es pot observar en l'exemple anterior, en la sortida del programa la diferència entre ***signed*** i ***unsigned*** és que el bit més significatiu –conegut com a MSB (***most significant bit***)– en les variables ***signed*** és 0 en els valors positius (7 en hexadecimal és 0111 en binari), en canvi té el valor 1 en els valors negatius (f en hexadecimal és 1111 en binari).

Per posar un exemple, el tipus `signed char`, tot i ser de 8 bits, com que el ***MSB*** identifica el signe, només es poden utilitzar 7 bits per a representar el valor. Tenint en compte la representació en binari de ***complement a 2***, el rang d'enters representables en ***n*** bits és [-2^{n-1} , $2^{n-1} - 1$], per tant, en el cas del tipus `char` el rang de valors és de -2^{7} a $2^{7} - 1$, és a dir de -128 a 127. Dit d'altra forma, hi ha $2^{7} = 128$ possibles valors per a cada signe, tenint en compte que en C2 (***complement a 2***) el zero es considera un valor positiu, el rang de valors positius va del 0 fins al 127 i el dels negatius del -1 al -128.

Complement a 2

El format de complement a 2[1] (Ca2 o C2), és un sistema de representació de nombres amb signe en base 2.

Els nombres positius en Ca2 es codifiquen de la mateixa manera que en signe i magnitud. El bit MSB és 0, per a indicar signe positiu, i la resta conté la magnitud.

La codificació dels nombres negatius s'obté a partir de l'operació en binari de $2^{n} - |X|$ en base 2, on |X| és el valor absolut de ***X***.

(1)**Two's Complement:** http://en.wikipedia.org/wiki/Two%27s_complement.

Tenint en compte aquests conceptes, en positiu, el valor 127 té una representació en binari de 0111 1111.

Per a representar un valor en negatiu, per exemple el valor -26, es realitza l'operació següent:

$2^{8} - |-26| = 100000000_{(2} - 11010_{(2} = 11100110$

1.2. Desbordament de tipus de dada

Tot i que els tipus `char, int i short` tenen uns rangs de valors determinats, si s'assignen uns valors superiors als valors admesos, l'execució del programa no donarà cap error, sinó que truncarà els valors.

En l'exemple següent, en el programa s'assignen valors fora dels rangs de vàlids a les variables.

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

int main()
{
     char c =  129;
     unsigned char uc = 257;
     short s =  32769;
     unsigned short us = 65537;
     int i = 2147483649;
     printf("Maximum values\n");
     printf("char           = %d (0x%x) [%d bits]\n", c, c, sizeof(c) * 8);
```

```
    printf("unsigned char  = %d (0x%x) [%d bits]\n", uc, uc, sizeof(uc) * 8);
    printf("short          = %d (0x%x) [%d bits]\n", s, s, sizeof(s) * 8);
    printf("unsigned short = %d (0x%x) [%d bits]\n", us, us, sizeof(us) * 8);
    printf("int            = %d (0x%x) [%d bits]\n", i, i, sizeof(i) * 8);

    c =  -130;
    uc = -2;
    s =  -32770;
    us = -2;
    i = -2147483650;
    printf("\nMinimum values\n");
    printf("char           = %d (0x%x) [%d bits]\n", c, c, sizeof(c) * 8);
    printf("unsigned char  = %d (0x%x) [%d bits]\n", uc, uc, sizeof(uc) * 8);
    printf("short          = %d (0x%x) [%d bits]\n", s, s, sizeof(s) * 8);
    printf("unsigned short = %d (0x%x) [%d bits]\n", us, us, sizeof(us) * 8);
    printf("int            = %d (0x%x) [%d bits]\n", i, i, sizeof(i) * 8);

    return 0;
}
```

sizes2.c

El resultat de l'execució del programa és el següent:

```
Maximum values
char           = -127 (0xffffff81) [8 bits]
unsigned char  = 1 (0x1) [8 bits]
short          = -32767 (0xffff8001) [16 bits]
unsigned short = 1 (0x1) [16 bits]
int            = -2147483647 (0x80000001) [32 bits]

Minimum values
char           = 126 (0x7e) [8 bits]
unsigned char  = 254 (0xfe) [8 bits]
short          = 32766 (0x7ffe) [16 bits]
unsigned short = 65534 (0xfffe) [16 bits]
int            = 2147483646 (0x7ffffffe) [32 bits]
```

Cal observar que els valors que es mostren en l'execució del programa no són els valors que s'han assignat a les variables, el motiu d'aquest fet és que els valors que s'han assignat estan fora del rang admès segons la definició de cada tipus de variable.

Els propers exemples es basen en els resultats del tipus de variable `char`, però són extrapolables a la resta de tipus de variables d'enters.

Per què el valor 129 assignat a una variable de tipus `char` no ha donat cap error de fora de rang o ***overflow*** però en cavi ha mostrat un valor de -127?

El valor 129 té una equivalència en binari: 10000001. Però seguint la representació en C2, el fet que el bit més significatiu (MSB), és a dir, el bit de l'esquerra sigui un 1, indica que es tracta d'un nombre negatiu. Ara cal tenir en compte que la representació d'un nombre negatiu en binari i en C2 varia respecte de si es tracta d'un nombre positiu o de si es tracta d'un nombre negatiu.

Anteriorment s'ha mostrat com obtenir a partir d'un nombre negatiu en decimal la seva representació en binari en el format de ***complement a 2*** (C2), a continuació es mostra el pas a la inversa, és a dir, a partir d'un nombre negatiu en binari en format C2, el seu equivalent en el sistema decimal.

Seguint el TFN (**Teorema fonamental de numeració**), per a obtenir el valor en decimal del valor en binari de 10000001 tenint en compte el format C2, cal aplicar el TFN, com en el cas positiu, però considerant que el bit més significatiu és negatiu:

$$-1 \cdot 2^{7} + 0 \cdot 2^{6} + 0 \cdot 2^{5} + 0 \cdot 2^{4} + 0 \cdot 2^{3} + 0 \cdot 2^{2} + 0 \cdot 2^{1} + 1 \cdot 2^{0} = -127$$

Enllaç recomanat

Sobre el teorema fonamental de numeració:

http://es.wikipedia.org/wiki/Sistema_de_numeración

http://electronicamarti.files.wordpress.com/2010/01/sistemas-de-numeracion.pdf

Quin hagués estat el valor presentat si s'hagués assignat el valor 1524 a la variable de tipus `char`?

La representació de $1524_{(10}$ en binari és 10111110100.

Com que es tracta d'una variable de tipus `char` i per tant de 8 bits, el computador trunca aquest valor i només interpreta els 8 primers bits menys significatius, és a dir, 11110100. Es trunca el valor a l'alçada del màxim nombre de bits que pugui contenir el tipus de dada.

A partir d'aquest valor, cal realitzar la mateixa operació per a obtenir el nombre en decimal a partir d'una representació binària en C2.

$$-1 \cdot 2^{7} + 1 \cdot 2^{6} + 1 \cdot 2^{5} + 1 \cdot 2^{4} + 0 \cdot 2^{3} + 1 \cdot 2^{2} + 0 \cdot 2^{1} + 0 \cdot 2^{0} = -12$$

Així doncs, en aquest cas, si s'assigna el valor 1524 a una variable de tipus `char`, en realitat el valor assignat és: -12.

En el proper cas, s'observa quin serà el resultat d'assignar el valor 1031 a una variable de tipus `char`.

- La representació en binari del valor $1031_{(10}$ és 10000000111.

- S'agafen els 8 bits menys significatius, és a dir, els 8 primers començant per la dreta: 00000111.

- El bit més significatiu (el primer per l'esquerra) és 0, per tant, es tracta d'un nombre positiu.

- Com que es tracta d'un nombre binari en positiu, la seva conversió en decimal és : $0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7$.

En aquest cas, si s'assigna el valor 1031 a una variable de tipus `char`, en realitat el valor assignat és: 7.

1.3. Atacs *integer overflow*

La norma ISO C99 diu que un ***integer overflow*** causa "***undefined behaviour***", el que significa que els compiladors compatibles amb l'estàndard poden fer el que vulguin, des d'ignorar completament el desbordament a avortar el programa. El que fan la majoria dels compiladors és ignorar l'***integer overflow***.

Els ***integer overflow*** no poden ser detectats fins que hagin ocorregut, això pot ser perillós si el càlcul té a veure amb la mida d'un ***buffer*** o l'índex d'un ***array***. La majoria dels ***integer overflow*** no són explotables perquè la memòria no està sent directament sobreescrita, però a vegades poden conduir a altres classes de ***bugs***, freqüentment de ***buffer overflow***.

Els atacs d'***integer overflow*** no permetran sobreescriure zones de memòria, variables o codi, però sí canviar la lògica de l'aplicació i fins i tot desbordar estructures de memòria creades per mitjà de variables insegures.

Així doncs, es pot observar que si s'assignen valors fora de rang a variables de tipus enter (`char, short, int`), el resultat pot ser inesperat donat que el valor resultant no serà el previst segons la lògica definida en el programa, com succeeix amb l'exemple següent:

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

int main(int argc, char *argv[]){
   int i, j;
   char c;
   int result;
   if (argc == 4) {
      i = atoi(argv[1]);
      j = atoi(argv[2]);
      c = atoi(argv[3]);
      if (i<=0 || j<=0 || c<=0) {
         printf("Invalid values\n");
```

```
        }
        else {
           result = i + j + c;
           printf("i: %d | j: %d | c: %d | result: %d\n", i, j, c, result);
           if (result == 0) {
              printf("Protected area\n");
              }
        }
     }
     else {
           printf("Three parametres are needed\n");
           printf("Sample1 param1 param2 param3\n");
     }
     return(0);
     }
```

protectedArea.c

En aquest exemple es pot observar que no es permet introduir valors negatius en els paràmetres i després es realitza la suma dels tres paràmetres, això fa pensar que la suma dels tres paràmetres no pot donar com a resultat zero i per tant el programa no mostrarà per pantalla la frase "Protected area". Però i si s'introdueixen els valors següents:?

Variable	Valor en decimal	Valor en binari
i	2147483647	01111111 11111111 11111111 11111111
j	2147483647	01111111 11111111 11111111 11111111
c	2	00000010

En aquest cas la suma de i + j = 4294967294. Però aquest valor produeix un ***integer overflow*** donat que supera el valor màxim admès en una variable de tipus `int`.

La representació de 4294967294 en binari és : 11111111 11111111 11111111 11111110

Però en ser tractat en format de ***complement a 2***, es tracta del nombre negatiu –2 i per tant, el resultat de i + j + c = 0.

Així doncs el resultat de l'execució del programa és el següent:

```
C:\pcs>ProtectedArea 2147483647 2147483647 2

i: 2147483647 | j: 2147483647 | c: 2 | result: 0
```

```
Protected area
```

Uns altres valors possibles per a aconseguir el mateix són els següents:

Es tracta que la suma de dos nombres positius desbordi la capacitat de la variable de resultat (***integer overflow***), obtenint un nombre negatiu.

Valor de la variable c:	+127	01111111
Valor de la suma parcial i+j	-127	11111111 11111111 11111111 10000001

Si es considera 11111111 11111111 11111111 10000001 com un nombre positiu, en comptes de -127 s'obté 4294967169.

Ara es tracta de fer que c = 127 i que la suma de i + j = 4294967169, per exemple i = 2147483647 j = 2147483522.

```
C:\pcs>ProtectedArea 2147483647 2147483522 127
i: 2147483647 | j: 2147483522 | c: 127 | result: 0
Protected area
```

Per a evitar que es produeixi un ***integer overflow***, la comprovació dels valors numèrics ha de ser exhaustiva perquè no es produeixin errors inesperats. Per exemple, en el codi anterior, la solució hauria implicat incloure una comprovació per a detectar si els valors introduïts estan entre un rang de valors determinat i, per descomptat, comprovar la mida del tipus de dades abans de començar a utilitzar-la.

2. Desbordament de pila (*stack overvlow*)

La memòria té una zona dedicada a les variables del programa que es divideix en dos: la pila o ***stack*** i el ***heap*** o zona de memòria dinàmica. La zona de pila té un creixement de dalt a baix pel que fa a posicions de memòria en forma de LIFO (***Last in, First out***). Quan un programa fa una crida a una funció es crea un nou ***stack frame,*** aquest ***stack frame*** s'utilitza per passar arguments als procediments i funcions i per a emmagatzemar les variables locals, i s'hi va reservant memòria a mesura que el programa va definint variables, emmagatzemant-les en format ***little endian***, és a dir, amb el bit menys significatiu a l'esquerra.

Enllaç recomanat

CWECommon Weakness Enumeration
CWE-121: *Stack-based buffer overflow:*http://cwe.mitre.org/data/definitions/121.html

A més amés, les adreces de retorn de les crides a les funcions també s'emmagatzemen a la pila, aquesta és la causa que en ocasions succeeixi el desbordament de la pila, ja que si la mida del valor d'una variable en una funció és superior a la mida de l'espai que se li ha reservat en la pila, pot sobreescriure l'adreça de retorn d'aquesta funció, el que podria permetre a un usuari maliciós executar qualsevol codi que ell vulgui. Aquest desbordament pot ser provocat i fer de forma intencionada que el valor que es sobreescriu en l'adreça de retorn sigui una adreça escollida per l'atacant, això faria que el retorn de la crida no es realitzaria a l'adreça prevista per l'execució del programa, sinó que el retorn de la crida aniria a una adreça escollida per l'atacant i es podria executar codi amb finalitats malicioses.

Els atacs per ***buffer overflow***, tant el d'***stack overflow*** com el ***heap overflow***, són un tipus d'atacs molt utilitzats pels *hackers*, ja que permet aprofitar aquesta vulnerabilitat per tal d'executar el seu propi codi i a la vegada realitzar atacs a la resta del sistema; així doncs, cal tenir molta cura a l'hora d'escriure el codi de programació per tal que no es puguin produir aquests desbordaments i així evitar ser vulnerables als atacs.

2.1. Els registres

Dins de l'arquitectura x86, en un mateix llenguatge de programació, per exemple en llenguatge ANSI C, es poden generar diferents codis en assemblador depenent del compilador, que a la vegada poden generar diverses maneres de gestionar els registres del sistema; tot i així, hi ha una sèrie de registres i instruccions bastant comuns en la majoria dels compiladors que s'utilitzen. Entre tots els registres, destacarem els tres següents:

- **EIP (*extended instruction pointer*)**. Conté l'adreça de la pròxima instrucció que cal executar. Quan la funció A crida a la funció B, la següent adreça que hem d'executar un cop es retorna de la funció B s'emmagatzema a la pila. Quan retorna la funció B, la CPU recull l'adreça de la pila i

l'emmagatzema en el registre EIP. L'adreça que hi ha en el registre EIP determina a quina adreça hi ha el codi en què ha de continuar l'execució del programa

- **ESP (*extended stack pointer*).** Conté l'adreça que apunta al valor superior de la pila, és a dir, al cap de la pila. Cal tenir en compte que la pila creix de manera invertida, és a dir, cada vegada que la pila creix, l'adreça de memòria decreix.

- **EBP (*extended base pointer*).** Conté l'adreça de memòria on comença la pila. Com que creix de forma invertida, és l'adreça més gran dins de la pila.

A part d'aquests tres registres especials del codi, es pot veure que es fa ús del registre EAX com a variable auxiliar per a moure valors entre variables.

Quan una funció està en execució, el registre EIP apunta a la instrucció en execució, el registre EBP apunta a l'adreça base de la pila i el registre ESP apunta al principi de la pila.

2.2. Gestió de la pila

L'***stack frame*** és la zona de memòria situada entre l'`EBP` i l'`ESP` i marca la zona de memòria de la funció assignada a la pila.

Quan es fa una crida a la funció `POP` amb un registre com a paràmetre, el valor situat en la posició de memòria apuntat per `ESP` serà assignat al registre i el valor d'`ESP` es desplaçarà per a treure aquest valor de la pila. En aquest cas, la pila es desplaça a la mida del registre extret, de manera que `ESP` quedarà assignat a `ESP` +4 [en arquitectures de 32 bits].

Quan es fa una crida a la funció `PUSH` amb un registre o valor com a paràmetre, aquest es posarà al cim de la pila, i el registre `ESP` es desplaçarà per a indicar que la pila està en la nova posició. En aquest cas `ESP` serà assignat a `ESP` –4.

2.3. Crida i retorn de funcions

Quan es produeix la crida a una funció, hi ha tres funcions importants que també afecten els registres i la pila:

- **`CALL`.** Quan es produeix una crida a una altra funció, és necessari fer una sèrie d'accions. `CALL` automatitza aquestes accions. En primer lloc fa un `PUSH` del valor següent d'`EIP`, és a dir, de l'adreça on es troba la instrucció següent per executar després que el control de programa torni de la crida a la funció. Aquest valor serà el valor de retorn de la funció cridada. En segon lloc actualitzarà el valor d'EIP a l'adreça de la funció cridada. És a

dir, `CALL Address` genera un `PUSH EIP` següent i una crida a la funció `MOV` (moure) `EIP`, `Address`.

- **`LEAVE`.** Quan s'abandona l'execució d'una rutina, es pot usar la crida a `LEAVE` per a preparar la sortida. Per a això, aquesta funció situa el cap de la pila en l'adreça de la base, és a dir, `MOV ESP`, `EBP`. Després fa un `POP` al registre `EBP`, és a dir, restaura el valor original d'EBP. Aquest valor d'`EBP` és desat al començament de la funció. Aquesta situació deixa el registre `ESP` apuntant a l'adreça de retorn `CALL`, és a dir, a la instrucció següent per executar després de la finalització de la funció.

- **`RTN`.** La crida a `RTN` genera la fi de l'execució d'una funció i el que es fa és actualitzar el valor d'`EIP` al valor d'`'ESP`, és a dir, és un `POP EIP`.

2.4. Exemple de desbordament de pila

En aquest exemple es presenta un programa en llenguatge C que és vulnerable a un desbordament de pila (***stack overflow***), en aquest cas, si s'introdueix com a paràmetre una cadena d'una determinada llargària, es podrà sobreescriure l'adreça de retorn del seu ***stack frame***.

Entorn de desenvolupament

- Sistema operatiu: Windows 7 Professional SP1 32 bits funcionant en màquina virtual.

- Virtualització: Oracle VM VirtualBox 4.3.12.

- Compilador C: CodeBlocks 13.12 with GNU GCC Compiler.

- Desassemblador i depurador: Ollydbg 2.01 Beta 2.

Aquest programa rep com a paràmetre una adreça d'email i imprimeix el domini al qual pertany l'adreça de l'email. El format és el següent: acount@domain.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char** argv) {
    char *domain;
    char email[40];

    if (argc != 2){
        printf("Usage: obtain email address\n");
        return(-1);
    }
```

```
    strcpy(email, argv[1]);      //La funció strcpy copia l'adreça de l'email
                                 //introduïda com a paràmetre (es a dir
                                 //argv[1]) en la variable local email.
    domain = strchr(email,'@');  //La funció strchr busca el caràcter '@'
                                 //en la cadena email, si el troba torna un
                                 // punter a aquesta posició, si no el troba
                                 //torna NULL. El valor tornat s'emmagatzema
                                 //en la variable local domain.
    if (domain!= NULL){          // Si ha trobat el caràcter '@'
                                 // incrementa en una posició el punter domain
        domain++;                //per que se salti el caràcter @ i només apunti al domini.
        printf("Domain: %s", domain);
    }
    else{
        printf("Incorrect email");
    }
    return (0);
}
```

`stackOverflow.c`

El resultat de l'execució del programa pot ser per exemple el següent:

```
C:\pcs>stackOverflow AAAABBBBCCCCDDDDEEEEFFFF@domainname.com
Domain: domainname.com
```

Durant el procés de l'execució, es pot observar, per exemple amb l'eina Ollydbg, que es realitza la crida `CALL stackOvervlow.00401340`, en aquest moment l'adreça en execució és 004010F8. En aquest punt es fa una crida (CALL) i per tant, el control del programa passarà a l'adreça 00401340 per a executar un bloc de instruccions fins que es realitzi un retorn amb la instrucció RETN. Llavors el control del programa tornarà cap a la instrucció següent al de la crida, és a dir, tornarà el control del programa cap a l'adreça 004010FD. Dit d'una altra manera, es realitza una crida a una funció, i quan finalitza l'execució d'aquesta funció, el programa continua l'execució a partir de la següent instrucció que ha realitzat la crida.

```
004010F8   .  E8 43020000    CALL stackOverflow.00401340
004010FD   .  89C3           MOV EBX,EAX
004010FF   .  E8 DC0A0000    CALL <JMP.&msvcrt._cexit>                    MSVCRT._cexit
00401104   .  891C24         MOV DWORD PTR SS:[ESP],EBX                   ExitCode
00401107   .  E8 3C0B0000    CALL <JMP.&KERNEL32.ExitProcess>             KERNEL32.ExitProcess
```

En l'adreça 0022FF2C de la pila s'emmagatzema l'adreça de retorn que haurà d'utilitzar per a tornar el control del programa a la següent instrucció posterior a la crida CALL. Com es pot observar, aquesta adreça de retorn és 004010FD.

```
0022FF2C  004010FD  RETURN from stackOverflow.00401340 to stackOverflow.004010FD
```

Un cop es realitza la crida, el control del programa passa a la funció que es troba en l'espai comprès des de l'adreça 00401340 fins a la 004013CB, que és quan es realitzarà el retorn a la següent instrucció de la crida.

```
00401340  $  55              PUSH EBP
00401341  .  89E5            MOV EBP,ESP
00401343  .  83E4 F0         AND ESP,FFFFFFF0                              DQWORD (16.-byte) stack alignment
00401346  .  83EC 40         SUB ESP,40
00401349  .  E8 32060000     CALL stackOverflow.00401980
0040134E  .  837D 08 02      CMP DWORD PTR SS:[ARG.1],2
00401352  .v 74 13           JE SHORT stackOverflow.00401367
00401354  .  C70424 243040   MOV DWORD PTR SS:[LOCAL.16],OFFSET stac      string => "Usage: obtain email address"
0040135B  .  E8 90080000     CALL <JMP.&msvcrt.puts>                       MSVCRT.puts
00401360  .  B8 FFFFFFFF     MOV EAX,-1
00401365  .v EB 63           JMP SHORT stackOverflow.004013CA
00401367  >  8B45 0C         MOV EAX,DWORD PTR SS:[ARG.2]
0040136A  .  83C0 04         ADD EAX,4
0040136D  .  8B00            MOV EAX,DWORD PTR DS:[EAX]
0040136F  .  894424 04       MOV DWORD PTR SS:[LOCAL.15],EAX               src
00401373  .  8D4424 14       LEA EAX,[LOCAL.11]
00401377  .  890424          MOV DWORD PTR SS:[LOCAL.16],EAX               dest => OFFSET LOCAL.11
0040137A  .  E8 79080000     CALL <JMP.&msvcrt.strcpy>                     MSVCRT._mbscpy
0040137F  .  C74424 04 40    MOV DWORD PTR SS:[LOCAL.15],40                c => '@'
00401387  .  8D4424 14       LEA EAX,[LOCAL.11]
0040138B  .  890424          MOV DWORD PTR SS:[LOCAL.16],EAX               string => OFFSET LOCAL.11
0040138E  .  E8 6D080000     CALL <JMP.&msvcrt.strchr>                     MSVCRT.strchr
00401393  .  894424 3C       MOV DWORD PTR SS:[LOCAL.1],EAX
00401397  .  837C24 3C 00    CMP DWORD PTR SS:[LOCAL.1],0
0040139C  .v 74 1B           JE SHORT stackOverflow.004013B9
0040139E  .  834424 3C 01    ADD DWORD PTR SS:[LOCAL.1],1
004013A3  .  8B4424 3C       MOV EAX,DWORD PTR SS:[LOCAL.1]
004013A7  .  894424 04       MOV DWORD PTR SS:[LOCAL.15],EAX               <%s> => [LOCAL.1]
004013AB  .  C70424 403040   MOV DWORD PTR SS:[LOCAL.16],OFFSET stac      format => "Domain: %s"
004013B2  .  E8 51080000     CALL <JMP.&msvcrt.printf>                     MSVCRT.printf
004013B7  .v EB 0C           JMP SHORT stackOverflow.004013C5
004013B9  >  C70424 4B3040   MOV DWORD PTR SS:[LOCAL.16],OFFSET stac      format => "Incorrect email"
004013C0  .  E8 43080000     CALL <JMP.&msvcrt.printf>                     MSVCRT.printf
004013C5  >  B8 00000000     MOV EAX,0
004013CA  >  C9              LEAVE
004013CB  .  C3              RETN
```

En la pila es reserva l'espai per a emmagatzemar el valor de la variable ***email***, aquest espai es troba a partir de l'adreça de la pila 0022FEF4 i té reservat un espai per a emmagatzemar fins a 40 caràcters. En aquest exemple s'ha introduït com a paràmetre el valor AAAABBBBCCCCDDDDEEEEFFFF@domainname.com, en el qual es poden observar les posicions on està emmagatzemat en la pila durant l'execució del programa.

```
0022FEE0  0022FEF4  ¶■"   ASCII "AAAABBBBCCCCDDDDEEEEFFFF@domainname.com"
0022FEE4  00000040  @
0022FEE8  0022FF08  ◘ "   ASCII "FFFF@domainname.com"
0022FEEC  0022FFC4  — "
0022FEF0  75CB8CD5  'ï╥u
0022FEF4  41414141  AAAA
0022FEF8  42424242  BBBB
0022FEFC  43434343  CCCC
0022FF00  44444444  DDDD
0022FF04  45454545  EEEE
0022FF08  46464646  FFFF
0022FF0C  6D6F6440  @dom
0022FF10  6E6E6961  ainn
0022FF14  2E656D61  ame.
0022FF18  006D6F63  com
0022FF1C  0022FF0D  ♪ "   ASCII "domainname.com"
0022FF20  0022FF28  ( "
0022FF24  75CA9E34  4×╧u  RETURN from msvcrt.75CA9E3E to msvcrt.75CA9E34
0022FF28  0022FF94  ö "
0022FF2C  004010FD  ²▶@   RETURN from stackOverflow.00401340 to stackOverflow.004010FD
```

En aquest exemple el programa funciona sense cap incidència anormal i finalitza mostrant el següent resultat:

```
C:\pcs>stackOverflow AAAABBBBCCCCDDDDEEEEFFFF@domainname.com
Domain: domainname.com
```

L'espai reservat per a la variable ***email*** és de 40 posicions. Què passaria si s'introdueix com a paràmetre un email amb una longitud de més de 40 caràcters, com per exemple AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKK@domainname.com ?

En el moment de fer la crida a la funció, es guarda el valor de l'adreça de retorn en l'adreça 0022FF2C i tal i com s'observa, l'adreça de retorn és 004010FD.

```
0022FF2C  004010FD  RETURN from stackOverflow.00401340 to stackOverflow.004010FD
```

Posteriorment, s'emmagatzema el valor del paràmetre introduït en la pila, però la funció `strcpy` no controla si la mida de la informació que ha d'emmagatzemar en la pila és superior a l'espai que hi ha reservat per a la variable, en aquest cas, la variable ***email***. Aquest fet fa que en aquest cas es sobreescrigui la posició 0022FF2C, que és on hi havia l'adreça de retorn de la crida `CALL stackOvervlow.00401340` que realitza el programa.

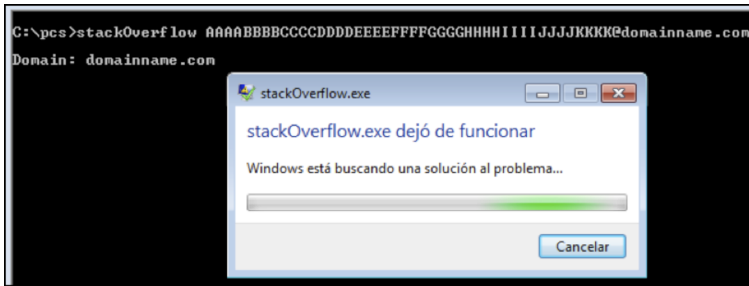
```
0022FEF4  41414141  AAAA
0022FEF8  42424242  BBBB
0022FEFC  43434343  CCCC
0022FF00  44444444  DDDD
0022FF04  45454545  EEEE
0022FF08  46464646  FFFF
0022FF0C  47474747  GGGG
0022FF10  48484848  HHHH
0022FF14  49494949  IIII
0022FF18  4A4A4A4A  JJJJ
0022FF1C  4B4B4B4B  KKKK
0022FF20  6D6F6440  @dom
0022FF24  6E6E6961  ainn
0022FF28  2E656D61  ame.
0022FF2C  006D6F63  com
```

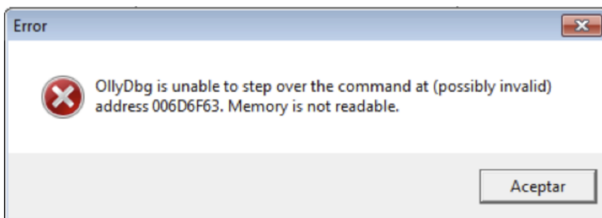
Quan el programa executa la instrucció RETN per a retornar a la instrucció següent de la crida, accedeix a l'adreça 0022FF2C per a obtenir l'adreça de retorn. Aquesta adreça de retorn hauria de ser 004010FD, però en el seu lloc troba el valor 006D6F63.

En aquest punt, l'EIP apunta a l'adreça 006D6F63, que és on hauria d'haver la instrucció que s'ha d'executar en retornar després de la crida a la funció. Però com que aquesta adreça no era la prevista en el flux previst del programa i a més aquesta adreça és ***not readable***, en aquest cas el sistema provocarà la interrupció del programa i apareixerà un missatge d'error.

```
Registers (FPU)
EAX 00000000
ECX 75CAC620 msvcrt.75CAC620
EDX 76F370F4 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0022FF30
EBP 2E656D61
ESI 00000000
EDI 00000000
EIP 006D6F63
```

A continuació es mostra el missatge motivat com a conseqüència del desbordament de pila.





En aquest exemple el sistema ha mostrat un missatge d'error i el programa ha deixat de funcionar. Tot i així, aquest desbordament de pila podia haver estat aprofitat per un ***hacker***. En cas que s'hagués tractat d'un atac intencionat, s'hagués pogut enviar un paràmetre a través d'un ***shellcode*** o un ***payload***, de forma que en l'adreça de la pila 0022FF2C s'hagués sobreescrit un valor intencionat que correspongués a una adreça de memòria, en la qual el ***hacker*** hagués injectat algun codi amb les variables i així poder-lo executar.

En aquest exemple s'ha mostrat que es pot sobreescriure l'adreça de memòria per mitjà d'un desbordament de les variables de la pila, així doncs, seria possible fer que el control de programa fos a qualsevol part de la memòria, és a dir, es podria executar qualsevol programa carregat en el sistema o fins i tot injectat en les variables.

3. Desbordament de *heap*

El ***heap*** és un segment de memòria que s'utilitza per a emmagatzemar dades assignades dinàmicament en temps d'execució.

Una altra zona de memòria o segment de dades molt similar és el ***BSS***, que és una zona destinada a emmagatzemar variables globals sense inicialitzar, que també s'assignen en temps d'execució i en aquest cas són emplenades amb zeros fins que se'ls assigna un nou valor. Cal tenir en compte que els errors relacionats amb el desbordament de ***heap*** són idèntics als ocorreguts en el ***BSS***.

Enllaç recomanat

CWECommon Weakness Enumeration

CWE-122: *heap-based buffer overflow:* http://cwe.mitre.org/data/definitions/122.html

El desbordament de ***heap*** és un tipus de ***buffer overflow***, on la memòria intermèdia (***buffer***) que pot ser sobreescrita s'assigna a la part del ***heap*** de la memòria, això significa que, en general, aquest *buffer* ha estat assignat utilitzant una instrucció de tipus `malloc()`.

La funció `malloc()` s'utilitza per a assignar un bloc de memòria en el ***heap***. El programa accedeix a aquest bloc de memòria via un punter que retorna la funció `malloc()`.

Els desbordaments de la memòria intermèdia sovint es poden utilitzar per a executar codi arbitrari, que és en general fora de l'àmbit de la política de seguretat implícita d'un programa.

A més de les dades importants de l'usuari, els atacs ***heap overflows*** es poden utilitzar per sobreescriure punters de funció, que poden estar actualment en la memòria, i apuntar-los cap al codi de l'atacant. Fins i tot en les aplicacions que no utilitzen explícitament els punters de funció, el ***runtime*** sol deixar-ne en la memòria. Per exemple, els mètodes d'objectes en C++ s'implementen generalment usant els punters a funcions, fins i tot en els programes en C, sovint hi ha una taula de desplaçament global que s'utilitza en temps d'execució subjacent.

El següent exemple és un programa senzill que utilitza memòria ***heap***. El programa conté un ***bug*** explotable de ***buffer overflow***.

En el primer cas, el procés del programa és normal i no es produeix cap ***heap overflow***, ja que la longitud del paràmetre d'entrada no provoca cap desbordament de memòria; en canvi, en el segon cas es pot observar que en introduir un paràmetre sobredimensionat, ha provocat un desbordament de ***heap***.

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

int main(int argc, char *argv[])
```

```
{
......char *input = malloc (20);
......char *output = malloc (20);

......strcpy (output, "normal output");
......strcpy (input, argv[1]);

......printf ("input at %p: %s\n", input, input);
......printf ("output at %p: %s\n", output, output);

......printf("\n\n%s\n", output);
}
```

heapOverflow.c

A través de l'eina Ollydbg es pot observar el procés de l'execució del programa.

```
00401340  $ 55              PUSH EBP
00401341  . 89E5            MOV EBP,ESP
00401343  . 83E4 F0         AND ESP,FFFFFFF0                          DQWORD (16.-byte) stack alignment
00401346  . 83EC 20         SUB ESP,20
00401349  . E8 62060000     CALL heapOverflow.004019B0
0040134E  . C70424 14000    MOV DWORD PTR SS:[LOCAL.8],14             size => 20.
00401355  . E8 C6080000     CALL <JMP.&msvcrt.malloc>                 MSVCRT.malloc
0040135A  . 894424 1C       MOV DWORD PTR SS:[LOCAL.1],EAX
0040135E  . C70424 14000    MOV DWORD PTR SS:[LOCAL.8],14             size => 20.
00401365  . E8 B6080000     CALL <JMP.&msvcrt.malloc>                 MSVCRT.malloc
0040136A  . 894424 18       MOV DWORD PTR SS:[LOCAL.2],EAX
0040136E  . 8B4424 18       MOV EAX,DWORD PTR SS:[LOCAL.2]
00401372  . C700 6E6F726    MOV DWORD PTR DS:[EAX],6D726F6E
00401378  . C740 04 616C    MOV DWORD PTR DS:[EAX+4],6F206C61
0040137F  . C740 08 7574    MOV DWORD PTR DS:[EAX+8],75707475
00401386  . 66:C740 0C 7    MOV WORD PTR DS:[EAX+0C],74
0040138C  . 8B45 0C         MOV EAX,DWORD PTR SS:[ARG.2]
0040138F  . 83C0 04         ADD EAX,4
00401392  . 8B00            MOV EAX,DWORD PTR DS:[EAX]
00401394  . 894424 04       MOV DWORD PTR SS:[LOCAL.7],EAX            src
00401398  . 8B4424 1C       MOV EAX,DWORD PTR SS:[LOCAL.1]
0040139C  . 890424          MOV DWORD PTR SS:[LOCAL.8],EAX            dest => [LOCAL.1]
0040139F  . E8 84080000     CALL <JMP.&msvcrt.strcpy>                 MSVCRT._mbscpy
004013A4  . 8B4424 1C       MOV EAX,DWORD PTR SS:[LOCAL.1]
004013A8  . 894424 08       MOV DWORD PTR SS:[LOCAL.6],EAX            <%s> => [LOCAL.1]
004013AC  . 8B4424 1C       MOV EAX,DWORD PTR SS:[LOCAL.1]
004013B0  . 894424 04       MOV DWORD PTR SS:[LOCAL.7],EAX            <%p> => [LOCAL.1]
004013B4  . C70424 24304    MOV DWORD PTR SS:[LOCAL.8],OFFSET heapO   format => "input at %p: %s◘"
004013BB  . E8 70080000     CALL <JMP.&msvcrt.printf>                 MSVCRT.printf
004013C0  . 8B4424 18       MOV EAX,DWORD PTR SS:[LOCAL.2]
004013C4  . 894424 08       MOV DWORD PTR SS:[LOCAL.6],EAX            <%s> => [LOCAL.2]
004013C8  . 8B4424 18       MOV EAX,DWORD PTR SS:[LOCAL.2]
004013CC  . 894424 04       MOV DWORD PTR SS:[LOCAL.7],EAX            <%p> => [LOCAL.2]
004013D0  . C70424 35304    MOV DWORD PTR SS:[LOCAL.8],OFFSET heapO   format => "output at %p: %s◘"
004013D7  . E8 54080000     CALL <JMP.&msvcrt.printf>                 MSVCRT.printf
004013DC  . 8B4424 18       MOV EAX,DWORD PTR SS:[LOCAL.2]
004013E0  . 894424 04       MOV DWORD PTR SS:[LOCAL.7],EAX            <%s> => [LOCAL.2]
004013E4  . C70424 47304    MOV DWORD PTR SS:[LOCAL.8],OFFSET heapO   format => "◘◘%s◘"
004013EB  . E8 40080000     CALL <JMP.&msvcrt.printf>                 MSVCRT.printf
004013F0  . C9              LEAVE
004013F1  . C3              RETN
```

En fer la crida CALL la funció `malloc()`, s'ha creat un ***stack frame*** on es pot observar el punter cap a l'adreça de la memòria ***heap***.

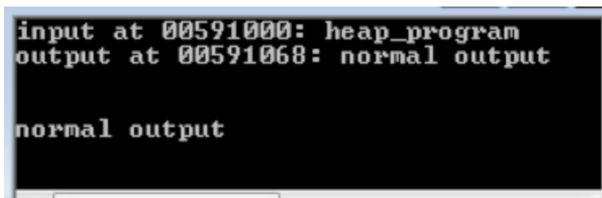
Address	Value	ASCI	Comments
0022FEE0	00590000	Y	Heap = 00590000
0022FEE4	00000000		Flags = 0
0022FEE8	00000014	¶	Size = 20.
0022FEEC	00000000		
0022FEF0	7FFDF000	²⌂	
0022FEF4	00000000		
0022FEF8	0022FF28	("	
0022FEFC	0040135A	Z‼@	RETURN from msvcrt.malloc to heapOverflow.0040135A
0022FF00	00000014	¶	size = 20.

En aquesta imatge es pot observar el mapa de memòria i la ubicació de l'espai reservat per al ***heap***.

Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial
00010000	00010000			Heap	Map	RW	RW
00020000	00010000			Heap	Map	RW	RW
0022D000	00001000				Priv	RW Gua:	RW Gua:
0022E000	00002000			Stack of main thread	Priv	RW	RW
00230000	00004000				Map	R	R
00240000	00001000				Priv	RW	RW
00290000	00004000			Default heap	Priv	RW	RW
00390000	00067000				Map	R	R
00400000	00001000	heapOverflow		PE header	Img	R	RWE Cop:
00401000	00001000	heapOverflow	.text	Code	Img	R E	RWE Cop:
00402000	00001000	heapOverflow	.data	Data	Img	RW Cop:	RWE Cop:
00403000	00001000	heapOverflow	.rdata		Img	R	RWE Cop:
00404000	00001000	heapOverflow	/4		Img	R	RWE Cop:
00405000	00001000	heapOverflow	.bss		Img	RW Cop:	RWE Cop:
00406000	00001000	heapOverflow	.idata	Imports	Img	RW Cop:	RWE Cop:
00407000	00001000	heapOverflow	.CRT		Img	RW Cop:	RWE Cop:
00408000	00001000	heapOverflow	.tls		Img	RW Cop:	RWE Cop:
00409000	00001000	heapOverflow	/14		Img	R	RWE Cop:
0040A000	00001000	heapOverflow	/29		Img	R	RWE Cop:
0040B000	00001000	heapOverflow	/41		Img	R	RWE Cop:
0040C000	00001000	heapOverflow	/55		Img	R	RWE Cop:
00590000	00003000			Heap	Priv	RW	RW

En aquest cas, el programa finalitza sense cap incidència.



En el proper cas, l'entrada del paràmetre sobrepassa l'espai de memòria reservat, a més, la funció `strcpy()` no té cap control referent a la seva mida.

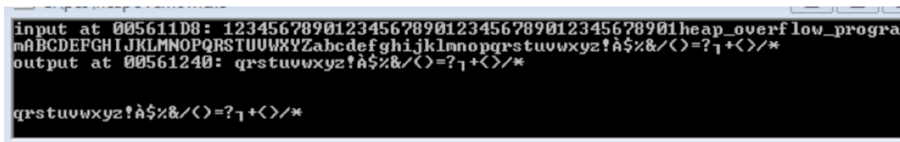
En aquest punt es pot observar el contingut de la pila durant l'execució de la funció `malloc()`.

Address	Value	ASCI	Comments
0022FEE0	00560000	V	Heap = 00560000
0022FEE4	00000000		Flags = 0
0022FEE8	00000014	¶	Size = 20.
0022FEEC	00000000		
0022FEF0	7FFD9000	É² ⌂	
0022FEF4	00000000		
0022FEF8	0022FF28	("	
0022FEFC	0040135A	Z‼@	RETURN from msvcrt.malloc to heapOverflow.0040135A
0022FF00	00000014	¶	size = 20.

Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial
00010000	00010000			Heap	Map	RW	RW
00020000	00010000			Heap	Map	RW	RW
0022D000	00001000				Priv	RW Gua:	RW Gua:
0022E000	00002000			Stack of main thread	Priv	RW	RW
00230000	00004000				Map	R	R
00240000	00001000				Priv	RW	RW
00280000	00004000			Default heap	Priv	RW	RW
00380000	00067000				Map	R	R
00400000	00001000	heapOverflow		PE header	Img	R	RWE Cop:
00401000	00001000	heapOverflow	.text	Code	Img	R E	RWE Cop:
00402000	00001000	heapOverflow	.data	Data	Img	RW Cop:	RWE Cop:
00403000	00001000	heapOverflow	.rdata		Img	R	RWE Cop:
00404000	00001000	heapOverflow	/4		Img	R	RWE Cop:
00405000	00001000	heapOverflow	.bss		Img	RW Cop:	RWE Cop:
00406000	00001000	heapOverflow	.idata	Imports	Img	RW Cop:	RWE Cop:
00407000	00001000	heapOverflow	.CRT		Img	RW Cop:	RWE Cop:
00408000	00001000	heapOverflow	.tls		Img	RW Cop:	RWE Cop:
00409000	00001000	heapOverflow	/14		Img	R	RWE Cop:
0040A000	00001000	heapOverflow	/29		Img	R	RWE Cop:
0040B000	00001000	heapOverflow	/41		Img	R	RWE Cop:
0040C000	00001000	heapOverflow	/55		Img	R	RWE Cop:
00560000	00003000			Heap	Priv	RW	RW

Tal i com es pot observar, en aquest cas s'ha produït un desbordament de ***heap***.



Posteriorment a la presentació de la informació per pantalla amb la funció `printf()`, també dóna el següent error abans de finalitzar el programa.

Access violation when reading [705F776F]

4. Funcions vulnerables

L'elecció d'un llenguatge de programació comporta conèixer el nivell de seguretat de les llibreries i funcions del llenguatge que s'utilitza. S'ha de tenir en compte que no totes les llibreries i funcions d'un llenguatge de programació són segures en totes les circumstàncies, una dada sobredimensionada, un valor fora del rang previst introduït com a paràmetre en una funció no segura, no donarà error en el moment de compilació del programa però sí pot provocar, per exemple, un desbordament de memòria. S'ha de tenir en compte que un ***hacker*** pot provocar la introducció de valors sobredimensionats, els quals poden provocar un desbordament de pila i aprofitar aquest fet per a executar el codi que hagi injectat per tal d'atacar el sistema.

En aquest punt no analitzarem cadascun dels llenguatges de programació ni totes les seves funcions no segures o vulnerables, però per posar un exemple, el llenguatge de programació C no proporciona una funció de protecció contra l'accés o sobreescriptura de dades en la memòria; no comprova que l'escriptura de dades en una memòria intermèdia es trobi dins dels límits d'aquesta memòria intermèdia, el llenguatge C tampoc realitza la comprovació automàtica dels límits de les matrius o indicadors com molts altres llenguatges, i a més, a la biblioteca estàndard de C hi ha gran diversitat de funcions vulnerables, com per exemple ***gets, getwd, strcpy, strcat, sprintf, scanf, sscanf, fscanf, vfscanf, vsprintf, vscanf, vsscanf, streadd, strecpy, realpath, syslog ,getopt, getopt_long, getpass***, ***realpath***, etc. La majoria d'aquestes funcions poden provocar un ***buffer overflow*** si els valors dels paràmetres que s'hi assignen no són correctes o estan sobredimensionats.

Enllaços recomanats

Unsafe (C# Reference): http://msdn.microsoft.com/es-es/library/chfa2zb8.aspx

Codi no segur i punters (Guia de programació de C#): http://msdn.microsoft.com/es-es/library/t2yzs44b.aspx

Pointer types (Guia de programació de C#: http://msdn.microsoft.com/es-es/library/y31yhkeb.aspx

En el llenguatge ***C#***, per exemple, per a mantenir la seguretat de tipus, no suporta punters aritmètics (+, ++, -, --, *, &, ==, !=, >, etc.) per defecte o de manera predeterminada. Tot i així, si s'utilitza la paraula clau ***unsafe***, es pot definir un context no segur en el qual es poden utilitzar aquests punters.

Així doncs, el coneixement de les particularitats de les llibreries i funcions que s'utilitzaran en el llenguatge de programació escollit, i alhora escollir les llibreries i funcions segures, és de vital importància per a realitzar una programació segura.

A continuació es mostra un exemple de la funció vulnerable ***strcat.***

```
char strcat(char *dest, const char *src)
```

La funció ***strcat*** no valida la mida de les cadenes que va a concatenar. Aquesta funció concatena una cadena ***src*** (cadena origen) a la cadena ***dest*** (cadena destí). Aquesta operació pot provocar un desbordament de memòria si la longitud de la concatenació de les dues cadenes és superior a l'espai reservat.

En realitat la funció ***strcat*** afegeix una còpia de la cadena apuntada per ***src*** al final de la cadena apuntada per ***dest*** i retorna un punter a ***dest*** en el qual resideix la cadena concatenada resultant.

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

void countStr(char str[])
{
      char frase[100] = "The string: \"";
      char total[6];
      snprintf(total, 6 , "%d", strlen(str));

      strcat(frase, str);
      strcat(frase, "\" has ");
      strcat(frase, total);
      printf("%s\n", frase);
      return;
}

int main(int argc, char *argv[])
{
      if(argc < 2){
            printf("Usage>text:");
            return -1;
      }
      countStr(argv[1]);
      printf("The program completes successfully \n");
}
```

strcatExample.c

El programa rep un tex com a paràmetre d'entrada, i sense validar la seva mida, el concatena amb la variable ***frase*** que té reservat un espai de 100 bytes, dels quals ja en té ocupats 13. Si la mida del paràmetre d'entrada es superior a 87 caràcters, en concatenar-lo amb la variable ***frase*** se sobreescriurà un espai de la pila no reservat per a aquesta variable. Segons la longitud del valor del paràmetre d'entrada, pot ser que aquesta sobreescriptura no tingui cap efecte o en canvi alteri l'execució del programa.

A més a més, a la variable ***frase*** se li concatena text addicional i el nombre de caràcters, per tant, si encara no s'havia superat l'espai reservat, podria succeir que en afegir aquest text, el superi.

El comportament del programa variarà en funció de la longitud del paràmetre d'entrada. Si la cadena té fins a 78 caràcters, no es supera l'espai reservat per a la variable ***frase***: 78 + 13 (The string: ") + 2 (número de caràcters) + 6 (" has) + caràcter final de cadena = 100.

Una execució correcte pot ser com la següent:

```
C:\pcs>strcatExample 123456789012345678901234567890
The string: "123456789012345678901234567890" has 30
The program completes successfully
```

Cal observar que en l'adreça 00401439 hi ha la instrucció corresponent a la crida a la funció `countStr(argv[1]);` és a dir, `CALL strExemple.00401340` de la funció ***main***, la qual crida a la funció ***countStr***, l'execució de la qual retorna de la funció ***countStr***. L'execució del programa continua a la següent instrucció de la crida CALL, és a dir, continua l'execució en l'adreça 0040143E.

```
004013FF  .   90            NOP
00401400  .   83EC 80       SUB ESP,-80
00401403  .   5B            POP EBX
00401404  .   5F            POP EDI
00401405  .   5D            POP EBP
00401406  L.  C3            RETN                                              Return to 0040143E from CALL strcatExample 00401340
00401407  r$  55            PUSH EBP
00401408  .   89E5          MOV EBP,ESP
0040140A  .   83E4 F0       AND ESP,FFFFFFF0                                  DQWORD (16.-byte) stack alignment
0040140D  .   83EC 10       SUB ESP,10
00401410  .   E8 EB050000   CALL strcatExample.00401A00
00401415  .   837D 08 01    CMP DWORD PTR SS:[ARG.1],1
00401419  .v  7F 13         JG SHORT strcatExample.0040142E
0040141B  .   C70424 27804  MOV DWORD PTR SS:[LOCAL.4],OFFSET strcatExampl   [format => "Usage>text:"
00401422  .   E8 C1530000   CALL <JMP.&msvcrt.printf>                         [MSVCRT.printf
00401427  .   B8 FFFFFFFF   MOV EAX,-1
0040142C  .v  EB 1C         JMP SHORT strcatExample.0040144A
0040142E  >   8B45 0C       MOV EAX,DWORD PTR SS:[ARG.2]
00401431  .   83C0 04       ADD EAX,4
00401434  .   8B00          MOV EAX,DWORD PTR DS:[EAX]
00401436  .   890424        MOV DWORD PTR SS:[LOCAL.4],EAX
00401439  .   E8 02FFFFFF   CALL strcatExample.00401340                       Call with possible stack overflow
0040143E  .   C70424 34804  MOV DWORD PTR SS:[LOCAL.4],OFFSET strcatExampl   [string => "The program completes successfully"
00401445  .   E8 96530000   CALL <JMP.&msvcrt.puts>                           [MSVCRT.puts
0040144A  >   C9            LEAVE
0040144B  L.  C3            RETN
```

En la següent imatge es pot observar la informació de la pila (***stack***) en el moment de fer la crida a la funció ***countStr*** des de la funció ***main***. En l'adreça 0022FF0C de la pila, es guarda la següent adreça de la crida a la funció, es a dir, l'adreça de retorn a la funció ***main***, que en aquest cas és 0040143E.

```
0022FF0C  0040143E  >¶@   RETURN from strcatExample.00401340 to strcatExample.0040143E
0022FF10  00570F67  g*W   ASCII "123456789012345678901234567890"
0022FF14  00301EF8  °▲0   ASCII ""C:\pcs\strcatExample.exe" 123456789012345678901234567890"
0022FF18  0000003A  :
0022FF1C  00000003  ♥
0022FF20  0022FF28  ( "
0022FF24  77649E34  4×dw  RETURN from msvcrt.77649E3E to msvcrt.77649E34
```

Posteriorment, es reserva un espai en la pila per a la definició corresponent a `char frase[100]`, aquest espai es pot observar que correspon a l'adreça 0022FE9C, fins a l'adreça 0022FF0F (0022FF0C + 3, 0022FF0C+ 4 = 0022FF10)).

```
0022FE7C  00401382  é‼@   RETURN from msvcrt.strlen to strcatExample.00401382
0022FE80  00820F67  g*é   ASCII "12345678901234567890123456789012345678901234567890123456789012
0022FE84  FFFFFFFF
0022FE88  00000024  $
0022FE8C  0022FE98  ÿ■"
0022FE90  00000004  ♦
0022FE94  00000000
0022FE98  6BFA8F9A  üÅ·k
0022FE9C  20656854  The
0022FEA0  69727473  stri
0022FEA4  203A676E  ng:
0022FEA8  00000022  "
0022FEAC  00000000
0022FEB0  00000000
0022FEB4  00000000
0022FEB8  00000000
0022FEBC  00000000
0022FEC0  00000000
0022FEC4  00000000
0022FEC8  00000000
0022FECC  00000000
0022FED0  00000000
0022FED4  00000000
0022FED8  00000000
0022FEDC  00000000
0022FEE0  00000000
0022FEE4  00000000
0022FEE8  00000000
0022FEEC  00000000
0022FEF0  00000000
0022FEF4  00000000
0022FEF8  00000000
0022FEFC  00000000
0022FF00  7FFD9000   É²⌂
0022FF04  00000000
0022FF08  0022FF28  ( "
0022FF0C  0040143E  >¶@   RETURN from strcatExample.00401340 to strcatExample.0040143E
0022FF10  00820F67  g*é   ASCII "12345678901234567890123456789012345678901234567890123456789012
0022FF14  00591F68  h▼Y   ASCII ""C:\pcs\strcatExample.exe" 123456789012345678901234567890123459
0022FF18  00000072  r
```

En aquest exemple es pot observar que el valor resultant de la concatenació no supera l'espai reservat en la memòria de la pila per a la variable de destí, i per tant, el programa finalitza sense incidències.

```
0022FE7C  004013F4  ¶‼@   RETURN from msvcrt._mbscat to strcatExample.004013F4
0022FE80  0022FE9C  £■"   ASCII "The string: "123456789012345678901234567890" has 30"
0022FE84  0022FE96  ü■"   ASCII "30"
0022FE88  00408024  $Ç@   ASCII "%d"
0022FE8C  0000001E  ▲
0022FE90  00000004  ♦
0022FE94  30330000    30
0022FE98  ED212600   &†Ÿ
0022FE9C  20656854  The
0022FEA0  69727473  stri
0022FEA4  203A676E  ng:
0022FEA8  33323122  "123
0022FEAC  37363534  4567
0022FEB0  31303938  8901
0022FEB4  35343332  2345
0022FEB8  39383736  6789
0022FEBC  33323130  0123
0022FEC0  37363534  4567
0022FEC4  22303938  890"
0022FEC8  73616820   has
0022FECC  00303320   30
0022FED0  00000000
0022FED4  00000000
0022FED8  00000000
0022FEDC  00000000
0022FEE0  00000000
0022FEE4  00000000
0022FEE8  00000000
0022FEEC  00000000
0022FEF0  00000000
0022FEF4  00000000
0022FEF8  00000000
0022FEFC  00000000
0022FF00  7FFD9000   É²⌂
0022FF04  00000000
0022FF08  0022FF28  ( "
0022FF0C  0040143E  >¶@   RETURN from strcatExample.00401340 to strcatExample.0040143E
```

En el següent exemple la longitud de la cadena és de 86 caràcters:

```
C:\pcs>strcatExample 123456789012345678901234567890123456789012345
67890123456789012345678901234567890123456
The string: "1234567890123456789012345678901234567890123456789012345
6789012345678901234567890123456" has 86
The program completes successfully
```

Es pot observar en la informació de la pila que el resultat de la concatenació de la cadena ha sobrepassat l'espai reservat per a emmagatzemar la cadena i ha sobreescrit en les adreces següents de la pila que corresponia a la variable

frase, en aquest cas la sobreescriptura no ha afectat l'execució del programa, donat que aquest espai no s'ha tornat a utilitzar posteriorment i no ha arribat a afectar l'adreça de retorn ni el procés d'execució, i el programa ha finalitzat sense incidències.

```
0022FE9C  20656854 The
0022FEA0  69727473 stri
0022FEA4  203A676E ng:
0022FEA8  33323122 "123
0022FEAC  37363534 4567
0022FEB0  31303938 8901
0022FEB4  35343332 2345
0022FEB8  39383736 6789
0022FEBC  33323130 0123
0022FEC0  37363534 4567
0022FEC4  31303938 8901
0022FEC8  35343332 2345
0022FECC  39383736 6789
0022FED0  33323130 0123
0022FED4  37363534 4567
0022FED8  31303938 8901
0022FEDC  35343332 2345
0022FEE0  39383736 6789
0022FEE4  33323130 0123
0022FEE8  37363534 4567
0022FEEC  31303938 8901
0022FEF0  35343332 2345
0022FEF4  39383736 6789
0022FEF8  33323130 0123
0022FEFC  22363534 456"
0022FF00  73616820  has
0022FF04  00363820  86
0022FF08  0022FF28 ( "
0022FF0C └0040143E >¶@  RETURN from strcatExample.00401340 to strcatExample.0040143E
```

En el següent exemple s'introdueixen 98 caràcters, en aquest cas se sobreescriurà l'adreça de retorn i per tant afectarà l'execució del programa.

```
C:\pcs>strcatExample 123456789012345678901234567890123456789012345678901234567890123456789
01234567890123456789012345678
The string: "12345678901234567890123456789012345678901234567890123456789012345678901234567
8901234567890123456789012345678" has 98
```

strcatExample.exe

strcatExample.exe dejó de funcionar

Windows está buscando una solución al problema...

Cancelar

Analitzant l'evolució de la pila, es pot observar que l'adreça de retorn 0040143E s'emmagatzema en la posició 0022FF0C.

```
0022FEF8  00000000
0022FEFC  00000000
0022FF00  7FFDA000  ã² ⌂
0022FF04  00000000
0022FF08  0022FF28 ( "
0022FF0C └0040143E >¶@  RETURN from strcatExample.00401340 to strcatExample.0040143E
0022FF10 ┌00870F67 g*ç  ASCII "12345678901234567890123456789012345678901234567890123
0022FF14 │005B1F80 Ç▼[  ASCII ""C:\pcs\strcatExample.exe" 123456789012345678901234567
```

En aquest cas el resultat de la concatenació sobrepassa la mida reservada per a la variable ***frase*** i sobreescriu la posició 0022FF0C de manera que canvia el valor de l'adreça de retorn 0040143E pel valor 73616820.

```
0022FE9C  20656854  The
0022FEA0  69727473  stri
0022FEA4  203A676E  ng:
0022FEA8  33323122  "123
0022FEAC  37363534  4567
0022FEB0  31303938  8901
0022FEB4  35343332  2345
0022FEB8  39383736  6789
0022FEBC  33323130  0123
0022FEC0  37363534  4567
0022FEC4  31303938  8901
0022FEC8  35343332  2345
0022FECC  39383736  6789
0022FED0  33323130  0123
0022FED4  37363534  4567
0022FED8  31303938  8901
0022FEDC  35343332  2345
0022FEE0  39383736  6789
0022FEE4  33323130  0123
0022FEE8  37363534  4567
0022FEEC  31303938  8901
0022FEF0  35343332  2345
0022FEF4  39383736  6789
0022FEF8  33323130  0123
0022FEFC  37363534  4567
0022FF00  31303938  8901
0022FF04  35343332  2345
0022FF08  22383736  678"
0022FF0C  73616820   has
0022FF10  00383920   98
0022FF14  005B1F80  Ç▼[   ASCII ""C:\pcs\strcatExample.exe" 123
```

Quan el programa executa la instrucció RETN per a retornar a la funció ***main***, accedeix a la posició 0022FF0C de la pila per a obtenir l'adreça de retorn. Aquesta adreça de retorn hauria de ser 0040143E, però en el seu lloc troba el valor 73616820.

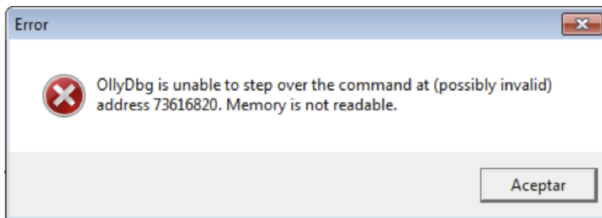
```
004013D2  .  C700 2220686  MOV DWORD PTR DS:[EAX],61682022
004013D8  .  66:C740 04 7: MOV WORD PTR DS:[EAX+4],2073
004013DE  .  C640 06 00    MOV BYTE PTR DS:[EAX+6],0
004013E2  .  8D45 8E       LEA EAX,[LOCAL.29+2]
004013E5  .  894424 04     MOV DWORD PTR SS:[LOCAL.33],EAX      ┌Arg2
004013E9  .  8D45 94       LEA EAX,[LOCAL.27]                   │
004013EC  .  890424        MOV DWORD PTR SS:[LOCAL.34],EAX      │Arg1 => OFFSET LOCAL.27
004013EF  .  E8 E4530000   CALL <JMP.&msvcrt.strcat>            └msvcrt._mbscat
004013F4  .  8D45 94       LEA EAX,[LOCAL.27]
004013F7  .  890424        MOV DWORD PTR SS:[LOCAL.34],EAX      ┌string => OFFSET LOCAL.27
004013FA  .  E8 E1530000   CALL <JMP.&msvcrt.puts>              └MSVCRT.puts
004013FF  .  90            NOP
00401400  .  83EC 80       SUB ESP,-80
00401403  .  5B            POP EBX
00401404  .  5F            POP EDI
00401405  .  5D            POP EBP
00401406  └. C3            RETN
```

En aquest punt, l'EIP apunta a l'adreça 73616820, que és on hi hauria d'haver la instrucció que s'ha d'executar en retornar a la funció ***main***, però aquesta adreça no és l'adreça correcte que havia d'haver en aquesta posició i en aquest cas dóna un error de memòria, Memory is not readable.

```
Registers (FPU)
EAX 00000000
ECX 75738E8A msvcrt.75738E8A
EDX 00020180
EBX 31303938
ESP 0022FF10 ASCII " 98"
EBP 22383736
ESI 00000000
EDI 35343332

EIP 73616820
```

A continuació es mostra el missatge motivat com a conseqüència del desbordament de pila que ha provocat la concatenació.



En aquest cas es produeix una situació similar a la mostrada en altres exemples en els quals el sistema mostra un missatge d'error i el programa deixa de funcionar. Aquest desbordament també podia haver estat aprofitat per un ***hacker***. En cas que s'hagués tractat d'un atac intencionat, s'hagués pogut enviar un paràmetre a través d'un ***shellcode*** o un ***payload***, de manera que en l'adreça de la pila 0022FF0C s'hagués sobreescrit un valor intencionat que correspongués a una adreça de memòria, en la qual el ***hacker*** hagués injectat algun codi amb les variables i així poder-lo executar, o executar qualsevol programa carregat en el sistema.

Bibliografia

Howard, M.; LeBlanc, D. (2002). *Writing Secure Code, Second Edition*. Redmond Washington: Microsoft Press.

Foster, J. C.; Osipov, V. (2005). *Buffer Overflow Atacks*. Syngress Publishing Inc.