Buffer overflows, securitate

Modificat: 15-Oct-18

Suport de curs

- Jon Erikson: Art of Exploitation, 2nd Edition
  - 310: Generalized Exploit Techniques
  - 320: Buffer Overflows
  - 330: Experimenting with BASH

Cuprins

- Gestiunea bufferelor în C
- Abuzarea bufferelor
- Inginerie inversă
- Gestiunea bufferelor în assembly
- Gestiunea stringurilor
- Exploatarea memoriei, buffer overflows
- Măsuri defensive
- Bune practici

GESTIUNEA BUFFERELOR

Ce este un buffer?

- O zonă de memorie
- Stocare temporară
- Definit prin
  - Adresă de start
  - Dimensiune (număr de octeți)

Buffere vs. vectori

- Bufferele conțin informații temporare
  - Read/write frecvent cu informații diverse
- Vectorii au informații "mai persistente"
- În general bufferele sunt referite ca vectori (indecși)
- Vectorii sunt o înșiruire de elemente de un anumit tip
- Bufferele sunt o zonă de memorie; tipul elementelor nu e relevant
- Programatic, definim bufferele ca vectori

Alocare de buffere

- Zona de date/bss (variabile globale)
- Pe heap (alocare dinamică)
- Pe stivă (variabile locale)
- Se alocă o dimensiune
- Adresa de start este stabilită de compilator, linker sau de sistemul de operare

Alocare de buffere: C vs. assembly

C

- int global_v[100] = { 0, };
- int global_v_uninit[200];
- int local_v[300];
- heap_v = malloc(1500);

Assembly

- global_v: times 100 dd 0
- global_v_uninit: resd 200
- sub esp, 1200
- push 1500
- call malloc

Alocare buffer de 128 de octeți pe stivă

C

- char v[128];
- unsigned char v[128];
- short v[64];
- unsigned short v[64];
- int v[32];
- unsigned int v[32];
- long long v[16];
- unsigned long long v[16];

Assembly

- sub esp, 128

Utilizare/referire buffere

- Citire/scriere la o adresă din intervalul [start, start+len]
- În C, se referă de obicei ca vector, sau aritmetică pe pointeri
- În assembly, pentru tipul de date referit, se folosesc offseți în octeți și construcțiile byte, word, dword, qword

Vectori (Arrays)

- Înșiruiri de elemente de același tip
- Modul de alocare de buffere
- Cuprind
  - Adresă (de start)
  - Tipul unui element (de unde dimensiunea elementului)
  - Număr de elemente (de unde spațiul ocupat)
    - size = num_items * sizeof(type)
- Referite prin indecși

Indecși

- Modul de adresare a unei poziții în vector
- 0 <= index <= num_items
  - dacă indexul nu este valid: array index out of bounds
- Putem avea și indecși negativi
  - comportament nedorit
- Câtă vreme zona de memorie referită este validă, folosirea unui index nevalid nu va genera excepție de memorie (segmentation fault)

Referire începând cu octetul 100 în buffer

- p = &buffer[100]; p = buffer+100
- v = buffer[100]; v = *(buffer + 100)
- buffer[100] = v; *(buffer+100) = v

- lea ebx, [buffer+100]
- mov edx, dword [buffer+100]
- mov dl, byte [buffer+100]
- mov dword [buffer+100], edx
- mov byte [buffer+100], dl

ABUZAREA BUFFERELOR

Abuzarea bufferelor/array-urilor

- Referirea dincolo de limitele bufferelor
  - out of bounds (OOB) errors
- Cauzate de erori de programare
- Două tipuri de abuz
  - citire: memory disclosure, information leak
  - scriere: overwrite, alteration, hijack execution

Out of bounds (OOB) errors

- Nu se face “bounds checking”
- Simplist: buf[-20] sau buf[200] (buf având 100 de elemente)
- Ce se pot dezvălui sau suprascrie?
  - Alte variabile/date
  - Valori salvate (registre)
  - Pointeri de cod (pointeri de funcție, adrese de cod)

Cum are loc out of bounds?

- buf[i] în program
  - nu se validează (bounds check) valoarea indexului i
- Un atacator detectează vulnerabiliatea și folosește un index necorespunzător

Out of bounds în zona de date

- Array/buffer definit ca variabilă globală
- Disclose sau overwrite de alte variable
- De aflat "distanța" între array și variabile
  - se poate folosi nm pentru aflarea adresei simbolurilor
  - se determină indexul

Out of bounds pe stivă

- Array/buffer definit ca variabilă locală
- Disclose sau overwrite la alte variabile, la valori salvate pe stivă, la adresa de retur
- Pot fi afectate informații din alt stack frame
- Compilatorul poate plasa discontinuu array-ul de alte variabile
  - se dezasamblează codul pentru a se urmări plasarea datelor pe stivă

Suprascrierea adresei de retur

- Adresa de retur este un “code pointer”
- Suprascriere = control flow hijack
  - salt în altă parte din program
  - alterarea fluxului normal de execuție al programului
- Se stochează pe stivă
- Un out of bounds poate suprascrie adresa de retur

INGINERIE INVERSĂ

Inginerie inversă

- Înțelegerea funcționării unui program, protocol, algoritm, sistem fără acces la specificații
- În cazul unui program: acces la executabil, nu la codul sursă
- În cazul unui fișier non-executabil: acces la fișier de format cvasi-necunoscut, fără specificații
- În cazul unui protocol: acces la pachete (interceptare), nu la specificații (RFC etc.)

Analiză statică

- Analiză fără rulare
- Poate fi și pe cod sursă (CoverityScan, lint, pylint)
- Uzual pe fișiere binare
- Pe executabile: nume de funcții, variabile, dimensiune buffere, dezasamblare
- Obiective: înțelegere, descoperire vulnerabilități, descoperire aspecte ne-conforme
- Avantaj: acoperire (coverage)
- Dezavantaj: greu de realizat acoperirea completă

Analiză dinamică

- Analiză la rulare
- Investigarea fluxului de control
- Observarea pas cu pas a procesului
- Investigarea memoriei și registrelor
- Obiective: înțelegerea programului, efectul input-ului
- Avantaj: flexibilitate
- Dezavantaj: nu are acoperire (coverage)

Dezasamblare

- Obținerea codului în limbaj de asamblare dintr-un executabil
- Se poate face și la runtime, într-un debugger
  - utilă pentru a observa pas cu pas
- Înțelegerea executabilului
  - se citește mult cod în limbaj de asamblare
- Obținerea fluxului de control
- Identificarea potențialelor riscuri de securitate
- Identificarea șirurilor sau funcțiilor apelate

nm, objdump

- nm: listarea simbolurilor (nume, zonă, adresă)
- objdump: investigare de executabil
  - util pentru dezasamblare
  - objdump -d -M intel <path-to-executable>

radare2, IDA

- Dezasambloare interactive
- IDA este “standard”-ul în materie
  - interfață grafică
  - multe componente și module
  - scump
- radare2: variantă open source
  - folosit preponderent în linia de comandă

Folosirea unui debugger

- Permite analiza dinamică detaliată a unui proces
- Memorie, registre
- Execuție pas cu pas
- Dezasamblare
- Interpretarea datelor
- Modificarea datelor în timp real pentru observarea efectului

GESTIUNEA ȘIRURILOR

Ce este un șir?

- Un array de char-uri
- Are la bază un buffer
- Lungimea unui șir e dată de prezența null-byte (NUL terminator)
  - Mai mică decât dimensiunea buffer-ului
- De regulă conține caractere afișabile (ASCII)

Șiruri valide/nevalide

- Un șir trebuie să conțină terminatorul de șir (null-byte, NUL, 0, 0x0, ‘\x00’, ‘\0’)
- Dacă nu conține folosirea sa poate genera erori
- Șirul trebuie să se încadeze în bufferul care-l susține

Lucrul cu șiruri

- Se citesc informații: variabile de mediu, argumente în linia de comandă, fgets, read, fread
- Șirurile transferă între ele informații: strcpy, strcat, strdup, strsep
- Lungimea unui șir e dată de poziționarea ‘\0’: strlen()
- Se pot referi elemente individuale ale unui șir (în format vector: s[i] sau *s)

Probleme cu șiruri

- Dacă nu este NUL-terminat, poate genera overflow
- Pot fi trunchiate (pierdute) informațiile ca să încapă într-un șir
- Buffer overflow-urile sunt cele mai prezente
- Off-by-one errors

BUFFER OVERFLOWS

Buffer overflow

- Operație dincolo de limita buffer-ului
- Copiere de informații dincolo de limita buffer-ului (overwrite)
- Citire de informații dincolo de limita buffer-ului (information leak)

Exemple de buffer overflow

- char buf[32];
- fgets(buf, 64, stdin);
- read(STDIN_FILENO, buf, 64);
- write(STDOUT_FILENO, buf, 72);
- memcpy(buf, old_info, 128);
- strcpy(buf, my_very_long_string);

Eroare după buffer overflow?

- În limbaje interpretate se poate detecta (Java)
- În limbaje compilate, ține de sistemul de operare
  - Dacă se accesează adrese valide în spațiul de adresă al procesului, operația este permisă
  - Se primește excepție de acces (segmentation fault) doar în caz de accesare a unei zone nevalide

Locuri pentru buffer overflow

- Heap
- Data/Bss
- Stivă

Stack buffer overflow

- Overflow la un buffer aflat pe stivă
- Buffer-ul e variabilă locală
- Se trece de dimensiunea buffer-ului și ajunge la alte informații din stack frame

Daune provocate de buffer overflow

- Alterarea fluxului de control al programului (control flow) sau al fluxului de date (data flow)
- Aplicația crash-uiește sau face altceva
  - Information leak
  - Obținerea controlului aplicației

Suprascriere

- Alterează fluxul de control al programului
- Variabile locale
  - Alterează execuția funcției curente
- Adresă de retur
  - Poate apela altă funcție sau altă secțiune dintr-o funcție
- Parametrii funcției

Vector de atac buffer overflow

- Se determină vulnerabilitatea
- Se determină offset-ul dintre buffer și zona care va fi suprascrisă
- Se determină cu ce suprascriem
- Se creează payload-ul de exploatare
- Se trimite payload-ul de exploatare aplicației
- Profit!

Determinarea vulnerabilității

- Analiză pe codul sursă (dacă există)
- Analiză statică pe executabil: dezasamblare
  - Analiză dinamică pentru validarea existenței vulnerabilității
- E nevoie de mult efort
  - Multe vulnerabilități sunt detectate automat de analizoare statice în faza de dezvoltare (CoverityScan)

Offset-ul buffer – zonă de suprascriere

- Analiză statică: dezasamblare + adresă buffer
- Exemplu:
  - buffer la ebp-48
  - variabilă de suprascris la ebp-12
  - offset-ul este (ebp-12)-(ebp-38) = 36
- Alt exemplu
  - buffer la ebp-72
  - adresă de retur la ebp+4
  - offset-ul este (ebp+4)-(ebp-72) = 76

Ce suprascriem?

- Variabile locale (pe stivă)
  - Schimbăm fluxul normal de execuție: un if, o buclă if funcționează altfel
  - "Ajustăm" un index de vector
  - Schimbăm un parametru pentru altă funcție
- Code pointeri
  - pointeri de funcție (ca variabile pe stivă)
  - adresa de return
  - Determină apelul unei alte funcții

Cu ce suprascriem?

- Date generate de noi
- Adrese ale altor funcții sau variabile
- Determinarea adreselor altor funcții sau variabile
  - Analiză statică: nm, objdump, IDA
  - Analiză dinamică: GDB
  - Information leak din alte atacuri
  - Dificil de realizat dacă sistemul are suport de ASLR (Address Space Layout Randomization)

Payload

- Șirul/Datele de transferat pentru a genera exploit-ul
- De obicei conține atât caractere ASCII cât și octeți
- Transmis programului vulnerabil
  - La intrarea standard
  - Ca un parametru
  - Pe un socket

Suprascriere variabilă locală

- De determinat offsetul și valoarea cu care suprascriem
- Suprascriere cu valoarea 0x12345678
- payload = offset * “A” + “\x78\x56\x34\x12”
- offset * “A” este padding

Suprascriere adresă de retur

- De determinat offsetul și adresa funcției cu care suprascriem
- Suprascriem cu adresa unei funcții locale f cu adresa 0x08043892
  - Aflată cu analiză statică sau analiză dinamică
- payload = offset * “A” + “\x92\x38\x04\x08”

Suprascriere adresă de retur cu funcție cu parametri

- După apelul ret, se sare la începutul funcției f
- Funcția f are așteptarea că pe stivă este o adresă de retur; punem 4 octeți random
- payload = offset * “A” + “\x92\x38\x04\x08” + 4 * “B” + “\x78\x56\x34\x12” + “\xab\xcd\xef\x01”
  - Ultimele două valori sunt cei doi parametri ai funcției f (cu adresa 0x08043892)

Apeluri de funcții din biblioteca standard

- Putem apela funcții din libc (biblioteca standard C)
  - Dacă este dezactivat ASLR (Address Space Layout Randomization)
- Se pot afla dintr-un debugger și apoi folosi în program
- Exemplul canonic este apelul system("/bin/sh")
  - Se caută adresa funcției system() în libc
  - Se caută șirul "/bin/sh" în libc

Construcții Python utile

- generăm payload-uri
- Payload = offset * “A” + “\x9d\x84\x04\x08”
- p32: struct.pack(“<I”, system_address)
  - ‘<‘: little endian
  - ‘I’: integer
- payload = offet * A + p32(system_address)