Securitate Software

LABORATOR 3 – VULNERABILITĂȚI SPECIFICE LIMBAJULUI C

Integer overflow & underflow

Overflow: are loc atunci când valoarea maximă ce poate fi reprezentată pe un întreg este depășită (“in sus”):

```
unsigned int a;
a = 0xFFFFFFFF;
a = a + 1;
```

Underflow: are loc atunci când valoarea maximă ce poate fi reprezentată pe un întreg este depășită (“in jos”):

```
unsigned int a;
a = 0;
a = a - 1;
```

Riscuri induse de integer over/underflow

Se pot obține valori incorecte ale variabilelor

Aplicațiile pot avea un comportament nedeterminat sau le poate fi afectată integritatea

O depășire numerică poate produce efecte în lanț, lăsând posibilitatea unui atacator să influențeze execuția aplicației (atunci când rezultatele obținute în urma depășirii sunt utilizate)

Vulnerabilitățile se datorează operațiilor aritmetice efectuate cu date provenite de la utilizator

Exemplu overflow:

```
int input = read_number();
if (input > 0)
{
        output = malloc(input * sizeof(int));
        for (i = 0; i < input; i++)
        {
                output[i] = SourceBuffer[i];
        }
}
```

Mai multe exemple: https://www.owasp.org/index.php/Integer_overflow

TODO: exemplu underflow

Căutați un exemplu de underflow

Probleme legate de conversii de tip intreg

Pot avea loc prin următoarele metode:

- (Type) casts
- Asignări
- Apeluri de funcții (prin valoarea de return sau prin parametri transmiși prin adresă)

Trebuie acordată o foarte mare atenție:

- Dimensiunilor tipurilor implicate în conversie
- Tipurilor celor două date (signed / unsigned)

Exemple de conversii problematice

Un tip de date “mic” este convertit (extins) la un tip mai mare (ex: char → int)

```
unsigned char a = 128;
int b = (int)a;          // b = ???
```

Un tip de date “mare” este convertit (trunchiat) la un tip mai mic (ex: int → char)

```
int a = 511;
char b = (char)a; // b = ???
```

Conversii signed/unsigned problematice

```
int copy (char *dst, char *src, unsigned int len)
{
        while (len--)
        *dst++ = *src++;
}
int f = -1;
copy (d, s, f);
```

Aritmetică de pointeri

Operații asupra tipurilor de date de tip pointer (ex: int*)

Exemple:

- Adunare: o valoare întreagă este adunată la un pointer

Probleme posibile:

- Interpretare greșită a aritmeticii

Probleme legate de aritmetica de pointeri

```
int buf[1024];
int *b = buf;
for (i = 0; i < 1024; i++)
{
        *b++ = i;
}
```

Operatorul sizeof()

Probleme posibile:

- Utilizarea sizeof() pe un pointer în loc de datele referențiate
- Comparații are rezultatului sizeof() (unsigned) cu numere cu semn

Exemplu:

```
char buffer[100];
char *buf = buffer;
//sizeof(buffer) = ???
//sizeof(buf) = ???
//sizeof(buffer[0]) = ???
//sizeof(buf[0]) = ???
```

Probleme induse de operatori aritmetici

Anumite operații pe numere negative pot genera numere unsigned mari (shiftări la dreapta, împărțiri, modulo)

Operația modulo (restul împărțirii: %)se utilizează deseori pentru indexare în șiruri de dimensiuni fixe. Un rezultat negative poate indexa la o adresa aflată înainte de începutul șirului

Sunt greu de găsit în codul sursă

Ordinea evaluării operatorilor

De obicei este dependentă de compilator

Pot avea efecte neașteptate, ambigue

Este recomandată utilizarea parantezelor pentru a forța ordinea dorită

Exemple de expresii problematice

Exemplu 1

```
if (len & 0x80000000 != 0)
        if (len < 1024)
                memcpy(dst, src, len);
```

Exemplu 2

```
    if (len = getlen() > 30)
```

snprintf(dst, len - 30, "%s", src);

Exemplu 3

```
int a = b + c >> 3;
```

Macro-uri definite sau utilizate greșit

Exemplu:

```
#define SQUARE(x)                x*x
Y = SQUARE(z + t);
```

Bibliografie recomandată

The Art of Software Security Assessments”, chapter 6, “C Language Issues”, pp. 203 – 296