Integer Overflow & Carry Bit

Tots sabem que la suma de bits, sempre requereix d'un bit addicional, el "carry bit". Si sumem dos bits:

Base 2: 1 + 1 = 10

En els llenguatges de programació, tenim tipus diferents per emmagatzemar valors enters, int, long, ... I enters sense signe, uint, ulong,... La diferència entre uns i altres, és l'espai d'emmagatzemament que tenen assignat. Per exemple, si agafem el c# com a llenguatge, i el tipus UInt64, aquest té una reserva de 64 bits en memòria per guardar un valor enter sense signe. Fins aquí de moment, no tenim cap problema. Ara mirem el seguent codi:

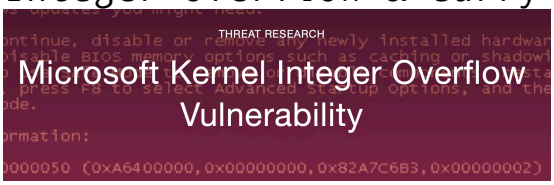
```
static void Suma64BitsErronea()
{
    UInt64 x = UInt64.MaxValue;
    UInt64 y = 1;

    Console.WriteLine($"x={x}");
    Console.WriteLine($"y={y}");

    var sum = x + y;
    Console.WriteLine($"x + y = {x} + {y} = {sum}");

    Console.WriteLine("We have a problem!!!");

    Console.WriteLine($"  x={ToBits(x)}");
    Console.WriteLine($"  y={ToBits(y)}");
    Console.WriteLine($"sum={ToBits(sum)}");

    Console.WriteLine("We need another bit!!");


}
```

I la sortida d'aquest:

```
x=18446744073709551615
y=1
x + y = 18446744073709551615 + 1 = 0
We have a problem!!!
  x=1111111111111111111111111111111111111111111111111111111111111111
  y=0000000000000000000000000000000000000000000000000000000000000001
sum=0000000000000000000000000000000000000000000000000000000000000000
We need another bit!!
```

Podem observar com el resultat no és correcte. Hem provocat un "Integer Overflow". Dit d'una altre manera, el resultat no cap en la variable de tipus UInt64 on volem emmagatzemar el resultat. Però, el programa no falla, no ens diu res, i el que és més rellevant, és que ens està donant un valor incorrecte. Podem dir, "que no sap sumar?". Amb altres llenguatges, com Java, també passa el mateix. Per veure que això pot arribar a ser un gran problema, dues notícies:

Integer Overflow & Carry Bit



https://www.fortinet.com/blog/threat-research/microsoft-kernel-integer-overflow-vulnerability
https://nikhilh20.medium.com/integer-overflow-vulnerability-20f9ea48aff9

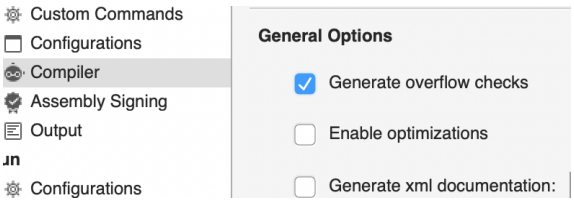
A qué se debe el 'Efecto 2022' de Microsoft Exchange

Desde 2013, Microsoft utiliza por defecto el sistema FIP-FS en Exchange. Fue en una de las renovaciones cuando los desarrolladores de Microsoft **decidieron almacenar el valor de la fecha en una variable *int32***. Una aparentemente inocente decisión que ha tenido consecuencias fatales para el funcionamiento de su sistema.

https://www.xataka.com/servicios/microsoft-exchange-sufre-su-particular-efecto-2022-bug-cambio-ano-filtro-antispam-impide-recibir-correos

Bé, que els "Integer Overflow" donen problemes, crec que queda clar. I com ho podem solucionar? Doncs, és la gran pregunta.

Si estudiem els compiladors, aquest porten un flag de compilació que fa que llanci uns excepció si el programa detecta un "Integer Overflow".




```
x=18446744073709551615
y=1
Unhandled exception. System.OverflowException: Arithmetic operation resulted in an overflow.
   at Proof.Program.Suma64BitsErronea() in /Users/jcastells/Projects/github/jc4st3lls/UInt256/Proof/Program.cs:line 47
   at Proof.Program.Main(String[] args) in /Users/jcastells/Projects/github/jc4st3lls/UInt256/Proof/Program.cs:line 9
```

Però si programem "una mica bé", hauríem de capturar aquestes excepcions i controlar-les. Això en cada operació aritmètica entre dos enters, sumes, restes, multiplicacions, divisions... en tot el codi, una bestiesa, de fet amb els anys que porto programant i mirant codi, no ho he vist enlloc (i és un gran problema, que després explicaré).

Com a solució, més o menys acceptable, podem utilitzar llibreries especialitzades en operacions matemàtiques que controlin bé les operacions, i que avisin si el resultat no és pot donar o és incorrecte. Això requereix però, que tot l'equip de programadors, en facin ús, i no es saltin les normes.

Integer Overflow & Carry Bit

Com exemple de solució, podem observar el següent codi.

```
public static (bool,bool) Sum(bool x, bool y, bool carry = false)
{
    var _value = x ^ y ^ carry;
    var _carry = (x & y) | ((x | y) & carry);

    return (_value, _carry);
}
```

Una simple funció que suma bits, i retorna, el resultat i el “carry bit”.

Ara, observem la següent funció:

```
private static (UInt64,bool) SumUint64(UInt64 x, UInt64 y)
{


    bool _carry = false;

    ulong z = 0;
    for (int i = 0; i < 64 ; i++) //64 bits
    {

        var xi = (((x >> i) & 1) == 1);
        var yi = (((y >> i) & 1) == 1);

        var result = Sum(xi, yi, _carry);

        _carry = result.Item2; //Carry

        ulong _value = (ulong)(result.Item1 ? 1 : 0);
        _value = (ulong)(_value << i);

        z = (ulong)(z | _value);

    }

    return (z, _carry);
}
```

Simplement, aprofitem la primera per sumar dos UInt64 i tornem el resultat i el “carry bit”. Si el provem:

```
private static void Suma64Bits()
{
    UInt64 x = UInt64.MaxValue;
    UInt64 y = 1;

    var sum = SumUint64(x, y);
    Console.WriteLine($"  x= {ToBits(x)}");
    Console.WriteLine($"  y= {ToBits(y)}");
    char carry = sum.Item2 == true ? '1' : '0';
    Console.WriteLine($"sum={carry}{ToBits(sum.Item1)}");

    y = UInt64.MaxValue;

    sum = SumUint64(x, y);
    Console.WriteLine($"  x= {ToBits(x)}");
    Console.WriteLine($"  y= {ToBits(y)}");
    carry = sum.Item2 == true ? '1' : '0';
    Console.WriteLine($"sum={carry}{ToBits(sum.Item1)}");

    Console.WriteLine("Fantastic, but how do I represent it? UInt65,..., UInt128,..., UInt256");

}
```

Integer Overflow & Carry Bit

```
  x= 11111111111111111111111111111111111111111111111111111111111111111
  y= 00000000000000000000000000000000000000000000000000000000000000001
sum=10000000000000000000000000000000000000000000000000000000000000000
  x= 11111111111111111111111111111111111111111111111111111111111111111
  y= 11111111111111111111111111111111111111111111111111111111111111111
sum=11111111111111111111111111111111111111111111111111111111111111110
Fantastic, but how do I represent it? UInt65,..., UInt128,..., UInt256
```

Veiem, que la suma ara és correcta, però amb aquest valor ja no podem fer operacions aritmètiques perquè el seu valor no està emmagatzemat en cap tipus d'enter. Necessitaríem un tipus enter més gran UInt128 o UInt256, amb operacions aritmètiques habilitades. De fet existeixen diferents implementacions d'aquests, ja que s'utilitzen molt en algoritmes de xifratge, seguretat i en solucions blockchain.

Integer Overflow: Un Bug?

Com he dit abans, si no habilitem el flag de compilació per llençar excepcions, el resultats dels algoritmes poden no ser correctes (cas Exchange). D'altre banda, si llacem les excepcions i no les controlem, podem deixar el programari en memòria en un estat inestable, i això pot provocar bugs de seguretat en el programari (cas Kernel). De fet, els que es dediquen a trobar bugs en programari, realment un dels vectors d'atac són els overflows de memòria, com per exemple de tipus enter. En alguns casos, no en tots, aquest bugs són aprofitats per col·locar codi arbitrari, i executar-lo, per tal d' aconseguir el control del procés, i través d'aquest, del maquinari. De fet, si el procés s'executa amb privilegis elevats, ... tenim un PROBLEMA!!!

jc4st3lls