# INTEGER OVERFLOW

**BELGE HAKKINDA**



Bu belgedeki eksik, yanlış ya da geliştirilmesi gerektiğini düşündüğünüz yerleri e-posta yoluyla bildirebilirsiniz.

Belgenin ilk oluşturulma tarihi: 10 Haziran 2007

/**********************************************************
* Cihan KÖMEÇOĞLU *
* cihan [at] akademi.enderunix.org *
* *
* EnderUNIX Yazılım Geliştirme Takımı *
* *
* http://www.enderunix.org *
* *
* Sürüm : 1.0 *
* *
* Tarih : 13.07.2007 *
* *
* Etiketler : integer , overflow *
* *
* Seviye : Başlangıç - Orta Seviye *
* *
* Makalenin en yeni versiyonu : http://www.enderunix.org/docs/intof.pdf adresinden elde edilebilir. *
* *
**********************************************************/

*Sn. Murat Balaban'a teşekkürler...*

## 1.Giriş

Bu dökümanda programlama hatalarının integer overflow ile ilgili kısmından bahsedeceğim.Dikkatli bir şekilde programlama yapmadığımız zaman değişkenlerimiz anlamsız değerler alabilir.Bunlarda ciddi sorunlara sebep olabilir.Bu dökümandaki örnekler C programlama dili ile yazılmıştır.

### 1.1 Integer Nedir ?

Integer sisteminize bağımlı olarak 32 bit ya da 64 bit sayıları saklayabilecek bir türdür.
Integer'lardan bahsedildiğinde genelde daha anlaşılır olan onluk sayı sistemi kullanılmaktadır.Fakat bilgisayarlar onluk sayı sistemi yerine ikilik sayı sistemini kullanmaktadır.Onluk sayı sisteminde on rakam kullanılmasına karşılık ikilik sayı sisteminde sadece 2 rakam kullanılmaktadır.Bunlar ise 1 ve 0'dır.

Bilgisayar sisemlerinde negatif sayıların gösterimi için bir bitinin bu gösterim için kullanılması gerekir. En yüksek değerli biti sayını negatif ya da pozitif olup olmadığını göstermektedir.
Bu bit 1 ise sayının negatif olduğunu, 0 ise pozitif olduğunu gösterir.

### 1.2 Negatif sayıların gösterimi

Az öncede bahsettiğimiz gibi en yüksek değerli biti sayının negatif olup olmadığını göstermektedir.
Bilgisayar sistemlerinde negatif sayıları elde etmek için sayının 1'e tümleyeni alınıp 1 eklenmektedir.

Örneğin: -34 sayısının gösterimi için
34 sayısının ikilik sistemde gösterimi - > 0010 0010

34 sayısının 1'e tümleyeni - > 1101 1101
1
\+ _____
-34 sayısının ikilik sistemde gösterimi-> 1101 1110

Görüldüğü gibi sayının en yüksek değerli biti olan 8. biti 1'dir ve bu negatif bir sayı olduğunu göstermektedir. Peki biz ikilik düzende olan bu negatif sayının gerçekten -34 olduğunu nasıl bileceğiz.

İlk olarak negatif bir sayı olup olmadığını anlamak için daha öncedende bahsettiğimiz gibi en yüksek değerli biti olan sayının en soldaki bitine bakarız. Bu bir ise negatiftir. Bu negatif sayının değerini bulmak için yine sayının 1'e tümleyenine 1 ekleyerek buluruz.

-34 sayısının ikilik sistemde gösterimi - > 1101 1110

-34 sayısının 1'e tümleyeni - > 0010 0001
1
\+ _____
0010 0010

Görüldüğü gibi işlemin sonuncan 34 sayısını elde ettik ve bu bize 1101 1110 ikilik sayının -34 olduğunu gösterir.

Ya da başka bir hesaplama yöntemini de aşağıdaki gibi yapabiliriz.

1101 1110 ikilik sayısı için en yüksek öncelikli bitinin onluk sistemdeki değeri alınır.

Bu ise sayının 8. bitidir. 2^7 = 128

İkinci adım ise sayının diğer bitlerinin onluk sistemdeki değerine toplanır.

2^6 + 2^4 + 2^3+ 2^2 + 2^1 = 94

94 – 128 = -34
Görüldüğü gibi 1101 1110 ikilik sayının değeri -34 olarak bulunmuştur.

### 1.3 Integer Overflow tehlikeli midir?

Integer overflow oluştuktan sonra tespit edilemez. Sonucun yanlış olduğunu uygulamaya anlatmanın bir yolu yoktur. Bunun için oluşmasına izin vermemek gerekir ve kodlarımızı buna göre yazmamız gerekir.Integer overflow'ın doğrudan bir tehlikesi yoktur fakat başka hataların oluşmasına sebep olabilir. Örneğin buffer overflow gibi.

## 2. Integer Overflows

Bir integer sayının tutabileceği maksimum sayıdan daha büyük sayıları saklamasını istediğimizde integer overflow oluşmaktadır. Bu durum sonucunda sayının gerçek değeri değişmektedir.
Bu değişimin nasıl olacağı ise tanımlıdır. Modulo aritmetiği sonucunda çıkan değer sonuç olmaktadır.
Modulo aritmetiği ise bir sayının bir sayıya bölünüp kalan değerinin alınmasıdır.

Örneğin:

11 mod 5 = 1
10 mod 5 = 0

Integer bir sayıya tutabileceğinden fazla bir değer atadığımızda bu "MAXINT+1"değerine göre modulo işlemine tutulup sonucu alınmaktadır.

Aşağıdaki örnekte 32 bit olan iki say toplanmaktadır.32 bit bir sayının tutabileceği maksimum değerinden fazla bir değer başka bir 32 bit değişkenine aktarılmaktadır. Bu durumda modulo işlemine tutulup değişkene bu değer yazılmaktadır.

A = 0xffffffff
B = 0x1
C = A + B

C = (0xffffffff + 0x1) mod 0x100000000
C = 0
Yukarıda yapılan bu işleme ise "wrap around" denilmektedir. Wrap around sonucunda iki sayının toplamı 0 olmuştur.

```
int main(){
        int a = 257;
        char b;

        b = a;
        printf(“b = %d”,b);

}
```

Sonuç:
b = 1

Yukarıdaki kod ile 32 bit bir değişkenin değeri 8 bit bir değişkene atama yapılıyor. Daha öncedende bahsettiğimiz gibi bu durumda modulo aritmetiği sonucunda çıkan değer b’ye aktarılmaktadır.

Char bir değişkenin tutabileceği maksimum değer 255’dir.

257 mod (255 + 1) = 1

**Integer overflow ve Buffer overflow ile ilgili örnek**

```
#include <stdio.h>
#include <string.h>
int main(int argc,char **argv){
       int i;
       unsigned short int a;
       char *ptr="Integer overflow and buffer overflow";

       i = atoi(argv[1]);
       a = i;
       if(a > strlen(ptr)){     [1]
              printf("length is too long");
              return -1;
       }

       printf("character: %c",ptr[i]); [2]
}
```

#./hello 6
Character: r
#./hello 65535
length is too long
#./hello 65536
Segmentation fault(core dumped)

Son örnekte daha önceden bahsettiğimiz sorundan dolay segmentation fault oluştu.65536 değeri 65535 değerinden daha büyük olduğu için word around işlemi sonucunda aşağıdaki değer elde edilir.

65536 mod (65535 + 1 ) = 0

Bundan dolayı [1] ile belirtilen yerde 0 > strlen(ptr) şartın sonucunda if-else yapısının içine girmek yerine [2] ile belirtilen satıra gelmektedir.

## 2.1 Aritmetik overflow

Bir integer sayının tutabileceği maksimum değerinden daha büyük bir değer saklamaya çalıştığımız zaman word around yapıldığını biliyoruz.Buna benzer bir örneği aşağıdaki işaretli integer bir sayı için inceleyelim.

```
#include <stdio.h>

int main(){
       short int a=32767;
       a = a+1;
       printf("%d", a);

}
```

#./signedint
-32768

Yukarıdaki örnekte bir short integer sayının tutabileceği maksimum pozitif değer atanmıştır.
Biz bu sayıya 1 eklediğimiz zaman negatif bir değer almaktadır. Yukarıdaki işlemi ikilik sayı sistemi ile bakalım

32767 -> 0111 1111 1111 1111
1
\+ _____
1000 0000 0000 0000

Daha önceden bahsettiğimiz gibi bilgisayarda negatif sayıların ifade edilmesinde sayının en yüksek değerli bitinden (burada 16. bit) yararlanarak ifade edilmekteydi. Bu bit 1 ise negatif sayı anlamına gelmektedir. İkilik sitemde de görüldüğü gibi işlemin sonucunda sadece pozitif sayının sınırları aşılmaktadır ve negatif bir sayıyı ifade etmektedir. Burada 16 bitin dışına çıkılmamaktadır fakat değişkenimiz unsigned olmadığı için sorun oluşturmaktadır.

Yukarıda belirttiğimiz sorun bir bufer overflow sorununa da sebep olabilir. Daha önceden yaptığımız örneğin aynısı burada da geçerlidir fakat “unsigned short int” değişken tipini signed short int yapmamız gerekir.

```
#include <stdio.h>
#include <string.h>
int main(int argc,char **argv){
       int i;
       short int a;
       char *ptr="Integer overflow and buffer overflow";

       i = atoi(argv[1]);
       a = i;
       if(a > (short int)strlen(ptr)){     [1]
              printf("length is too long");
              return -1;
       }

       printf("character: %c",ptr[i]); [2]
}
```

#./hello 32768
Segmentation fault (core dumped)

Burada a değişkenin değeri -32768 olacaktır.
Bir diğer örnek ise minimum negatif sayının sınırlarının aşılması ile ilgilidir. Aşağıdaki kodu inceleyelim.

```
#include<stdio.h>
int main(){
       char c = 4;
       c = c - 255;
       printf("%d",c);
}

#./neg
5
```

Yukarıdaki örneği ikilik sistemde incelersek:
4 – 255 ile 4 + (-255) aynı anlama geldiğinden

4 -> 0000 0010
255 -> 1111 1111

-255'i elde etmek için 1'e tümleyenine 1 ekleriz.
255 sayısının 1'e tümleyeni -> 0000 0000

```
                    1
+_________
  0000 0001 ?
```

Görüldüğü gibi sonuç 1 çıkmıştır ve -255 elde etmek için 8 bit yeterli değildir.
Fakat işlem sonucunda
4 + 1 = 5 program sonlandığında sonuç 5 olacaktır.

Bunun sebebi ise char tipinin tutabileceği minimum -128'dir. Biz bu değerin çok altına düştüğümüz için sonuç yanlış çıkmıştır.

**Exploit**

```
char buf[100];
  void copying(char *str, int len){


        if(len > sizeof(buf)){    [1]
            return -1;
        }

        memcpy(buf, str, len);  [2]
    }
```

Bu kod buffer overflow sebep olabilir ve bunun sonucunda segmantation fault verebilir.
Kodun [1] kısmında len değerine biz -1 yolladığımızda (hexadecimal olarak 0xffffffff) herhangi bir sorun vermeden kodun [2] kısmına geçer. Burada ise buffer overflow sebep olur. Memcpy fonksiyonun len parametresi unsigned integer'dır. Fakat biz len parametresine -1 yolladığımız için bunu negatif bir sayı olarak yorumlamaz. Buradaki negatif sayı, unsigned integer'da maksimum değer olarak yorumlanır. Yani -1 değilde 4.294.967.295 olarak yorumlanır ve str değişkeninden 4gb'lık veriyi buf dizisine kopyalamayı çalışır.Bu da segmentation fault hatasına sebep olur.

**Kaynaklar:**

http://en.wikipedia.org/wiki/Integer_overflow
http://www.enderunix.org/docs/bof.txt