Bezbjednost i zaštita informacionih sistema

Sigurnosni aspekti programiranja

Prof. dr Nikola Žarić

e-mail: zaric@ucg.ac.me

Uvod

- ✓ Pojam **sigurne aplikacije** vezuje se za njenu otpornost prema različitim vrstama napada.
- ✓ Statistika upućuju da se 70% sigurnosnih propusta vezuje za aplikativni sloj.
- ✓ **Pisanje sigurnog koda je od vitalnog značaja za održavanje sistema sigurnim.**
- ✓ Pravila sigurnog proramiranja podrazumijevaju određena odstupanja od programske jednostavnosti i dosta često od njegove efikasnosti. Argument za to prilično je ubedljiv:

  U slučaju pada sistema, do tada efikasan, kod više nikome ne koristi.
- ✓ Siguran kod, između ostalog, korisnika štiti i od njega samog. Neko može biti zloupotrebljen bez svog znanja i pristanka.

Uvod

- Prilikom kreiranja sigurne arhitekture za aplikaciju, mogući napadi mogu se klasifikovati u sljedeće kategorije:
  - **Subverzija aplikacije** - podrazumijeva akciju sa posljedicom izvšavanja nenamjeravane funkcionalnosti.
  - **Subverzija sistema i spoljnih aplikacija** - javlja se kada iskorišćavanje utiče na druge pokrenute aplikacije ili sistemske resurse. Ovo uključuje izvršavanje drugih aplikacija, kao što je shell u Unix-u, ili iskorišćavanje konekcije druge aplikacije. Efekti napada nisu ograničeni na pokrenutu aplikaciju i ta aplikacija se često koristi kao cjevovod za druge sisteme, aplikacije i operativni sistem.
  - **Prekid funkcionalnosti** - znači svaki oblik odbijanja servisa, u šta spada i grubo rušenje aplikacije.

Uvod

- Smjernice koje pri dizajnu zaštićenog koda ne bi trebalo zanemariti:
  - Svaki korisnik sistema treba da radi sa **što je moguće manjim privilegijama**. Na ovaj način se smanjuje mogućnost štete, greške ili zloupotrebe prilikom napada.
  - Kako je često neophodno ispitivati red po red koda, ekonomičnosti radi, sistem zaštite treba da bude što je moguće jednostavnijji.
  - Kod bi trebalo da bude izložen mogućoj kritici i na taj način temeljno testiran
  - Poželjno je da bude otvoren.

Prelivanje bafera

- **Bafer** je memorijski prostor predviđen za skladištenje podataka.
- Pošto je ograničenog kapaciteta, ukoliko se prepuni, to izaziva gaženje narednih memorijskih lokacija
- Preciznim stavljanjem novih podataka na određene adrese moguće je dovesti do izvršavanja napadačevog koda.
- Na taj način on može preuzeti računar žrtve.
- Prekoračenje bafera je napad koji se sastoji iz niza varijacija.

Prelivanje bafera

Razmotrimo sljedeću funkciju:

```
void func(char *str) {
        char buf[126];
        strcpy(buf,str);
}
```

Ukoliko je veličina stringa str veća od 126 karaktera, doći će do gaženja podatka na steku koji slijede poslije niza buf.
Povratna adresa funkcije koja se također nalazi na steku može biti pregažena i to specijalnom adresom koja označava početak nekog drugog malicioznog koda.

Prelivanje bafera

- Osnovni **problem** navedenog koda sastoji se u funkciji **strcpy(buf,str)** koja kopira niz str u niz buf sve dok u nizu str ne dođemo do "\0“koji predstavlja oznaku za kraj stringa. Veličina niza buf je totalno ignorisana.

- Funkcija strncpy(char *odr,const char *izvor,int n) dozvoljava kopiranje samo n znakova iz izvora u odredište - ostatak će biti odsiječen.

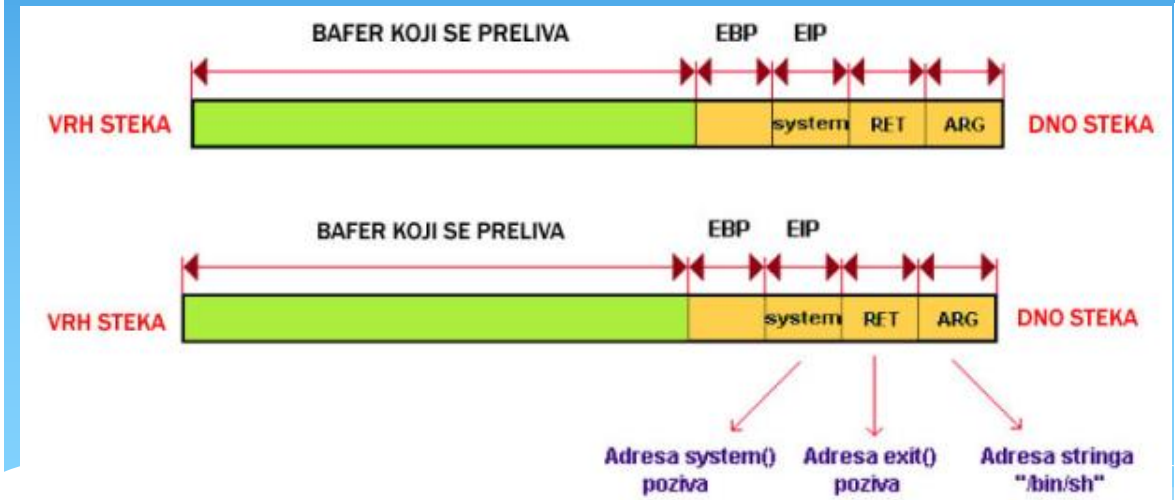
- Prekoračenje bafera se može desiti i na Hip dijelu memorije. Recimo, ako se pokazivač p na funkciju fun() nalazi na memorijskoj lokaciji pp, tada neka druga funkcija fun1() može pozvati funkciju fun() (*pp)(...)

Prelivanje bafera

- Za iskorišćavanje ovog prelivanja bafera postoji nekoliko ključnih postavki bez kojih prikazana tehnika ne bi bila moguća:
  • memorija koja je rezervisana za stek mora biti izvršna (engl. executable), jer se, u suprotnom, shellcode na steku ne bi mogao izvršavati.
  • memorijska adresa na kojoj se nalazi shellcode mora biti poznata, kako bi neovlašćeni korisnik znao sa kojom vrednošću treba prepisati povratnu adresu na steku. Ta memorijska adresa se može i pogoditi brute-force tehnikom, ali to znatno komplikuje proces iskorišćavanja ranjivosti.
  • operativni sistem na kojem se iskorišćava prelivanje bafera mora omogućavati prepisivanje povratne adrese na steku

Prelivanje bafera

- Ret-into-libc tehnika




```
Libc.c
#include <stdio.h>
callme (char *a)
{
  char buf[24];
  printf ("RET-INTO-LIBC #1\n");
  system("/bin/ls");
  strcpy (buf,a);
}
main (int argc, char **argv)
{
  callme(argv[1]);
  exit(0);
}
```

- Prekoračenja cjelobrojnih vrijednosti

- Prekoračenje cjelobrojnih vrijednosti puno je teže uočiti nego npr. *format string* propuste ili *race condition* propuste, pa samim time predstavljaju i veću prijetnju sigurnosti
- Iskorišćavanje prekoračenja cjelobrojnih vrijednosti na kraju se, uglavnom, svodi na prelivanje bafera, pa neovlašććeni korisnici kombinuju propuste kod prekoračenja cjelobrojnih vrijednosti sa nekim klasičnim metodama iskorišćavanja prelivanja bafera.
- Prekoračenja cjelobrojnih vrijednosti i propusti temeljeni na manipulaciji integerima otkriveni su u vrlo popularnim programima, kao što su: Apache Web server, Sendmail MTA program, Jezgro operativnih sistema Linux, OpenBSD i FreeBSD, Internet Explorer, Sunove RPC XDR biblioteke itd.

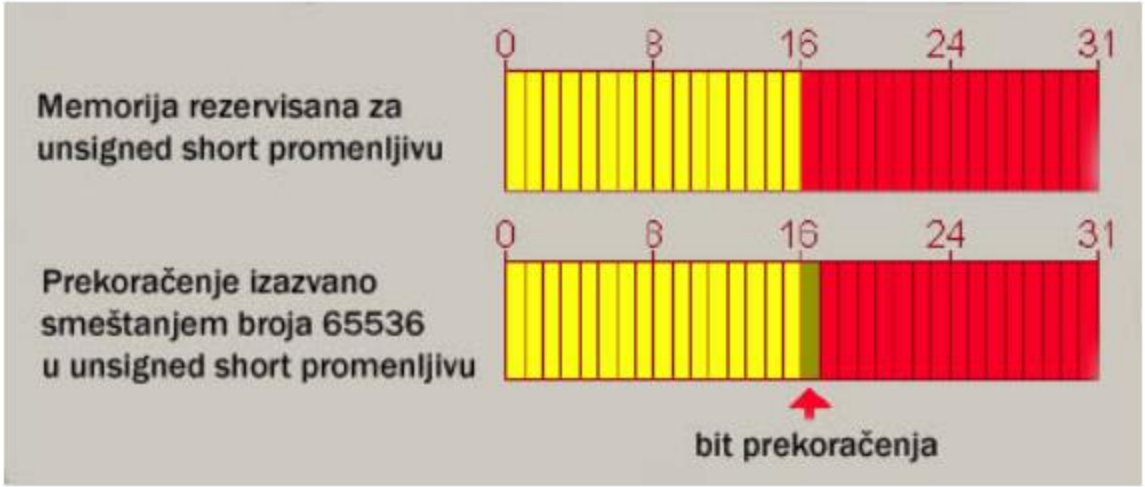
- Prekoračenja cjelobrojnih vrijednosti

- Za prekoračenja celobrojnih vrijednosti važno je napomenuti da oni ne vode direktno do prepisivanja određene memorijske lokacije kao kod klasičnog prelivanja bafera, nego su opasni ukoliko se celobrojne promenljive, kod kojih se dogodlilo prekoračenje, koriste za određivanje veličine bafera kod funkcija za kopiranje znakovnih nizova kao što su:
    - memcpy(),
    - strncpy(),
    - snprintf(),
    - memset().

- Prekoračenja cjelobrojnih vrijednosti

TIP PODATKA	Veličina	VREDNOST
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

- Prekoračenje cjelobrojnih vrijednosti



Vrednosti unsigned short promenljive	Sadržaj promenljive u binarnomn obliku	Sadržaj promenljive u decimalnom obliku
Minimalna	0000 0000 0000 0000	0
Maksimalna	1111 1111 1111 1111	65535
Prekoračenja	1 0000 0000 0000 0000	0 (65536)

- Prekoračenje cjelobrojnih vrijednosti

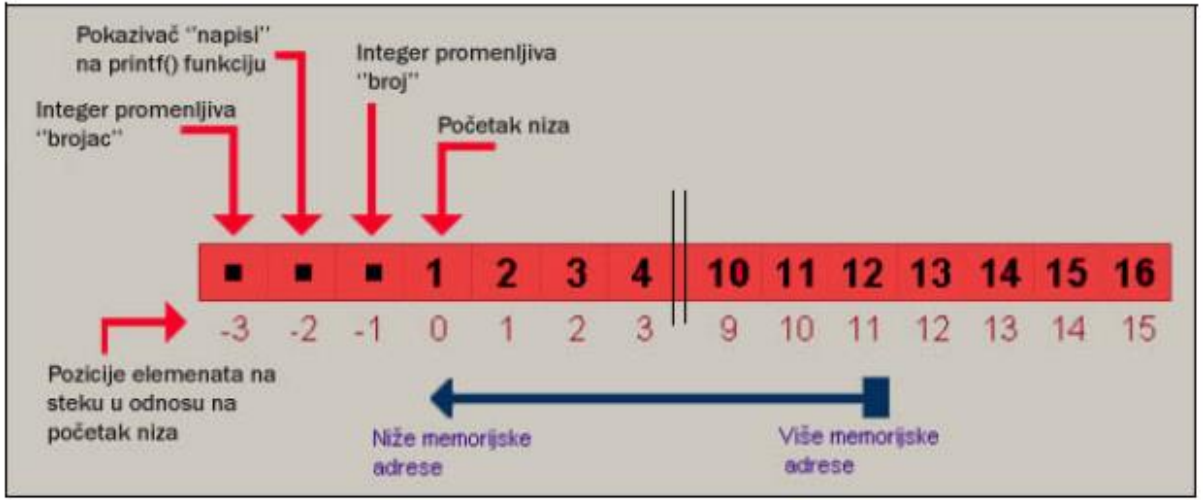
Short.c

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

```
$ gcc short.c -o short
$ ./short 1
Sadrzaj promenljive unsigned short x=1
Sadrzaj promenljive int y=1
$ ./short 65536
Sadrzaj promenljive unsigned short x=0
Sadrzaj promenljive int y=65536
$
```

- Prekoračenje cjelobrojnih vrijednosti

- Prekoračenja izazvana razlikom između tipova cjelobrojnih vrijednosti
- Prekoračenje cjelobrojnih vrijednosti zbog aritmetičkih operacija
- Manipulacije cjelobrojnim vrijednostima u nizovima



- Ostali propusti

- Neki osnovne propuste koji su lako uočljivi u radu Web aplikacija.
  - Kada se koristi HTTP protokol, ne bi trebalo koristiti GET zahtev koji uzrokuje davanje određenih podataka u URL zaglavlju.
    • Web adresa www.toolkit.org može lako biti zamijenjena sa www.t001kit.0rg jer broj 0 podsjeća na slovo o, a broj 1 na slovo l.
    • Treba izbjegavati stavljanje komentara jer oni mogu napadaču dati potrebne informacije o radu programa.