Sicurezza

Marco Alberti

Programmazione e Laboratorio, A.A. 2016-2017

Dipartimento di Matematica e Informatica - Università di Ferrara
Ultima modifica: 14 dicembre 2016

Sicurezza informatica

Sicurezza

E’ la capacità di un sistema di difendersi da attacchi esterni alla sua integrità, come

- Installazione di virus, trojan, spyware
- Utilizzo non autorizzato dei servizi
- Accesso ad informazioni riservate

La sicurezza informatica è un processo che deve essere condotto su più ambiti:

- Preparazione degli utenti
- Amministrazione di sistema e di rete
- Programmazione

Termini

- risorse: entità di valore che deve essere protetta (tipicamente il sistema stesso, o i dati su cui opera)
- vulnerabilità: debolezza nella sicurezza di un sistema che mette in pericolo qualcuna delle sue risorse
- attacco: sfruttamento di una o più vulnerabilità al fine di compromettere l'integrità, la confidenzialità e la disponibiltà delle risorse.

Minacce

- integrità: un attacco può corrompere (modificare indebitamente) i dati gestiti dal sistema o il sistema stesso.
- confidenzialità: molti sistemi gestiscono informazioni che devono essere accessibili solo a persone autorizzate; un attacco può consentirne l’accesso ad agenti non autorizzati
- disponibilità: un attacco può impedire o rendere difficoltoso l’accesso ai servizi del sistema per gli utenti autorizzati

Programmazione sicura in C

Vulnerabilità in programmi C

Alcune scelte di progetto del linguaggio C e della libreria standard richiedono al programmatore una particolare attenzione per evitare vulnerabilità.

Questo è particolarmente importante perché il linguaggio C è usato in software di sistema, implementazioni di protocolli di rete, server.

Molte delle vulnerabilità in programmi C sono legate alla possibilità di accedere a parti di memoria non volute dal programmatore.

Array bound checking

Il seguente frammento di codice

```
int i, a[10];
for (i = 0; i<20; i++)
   a[i]=0;
```

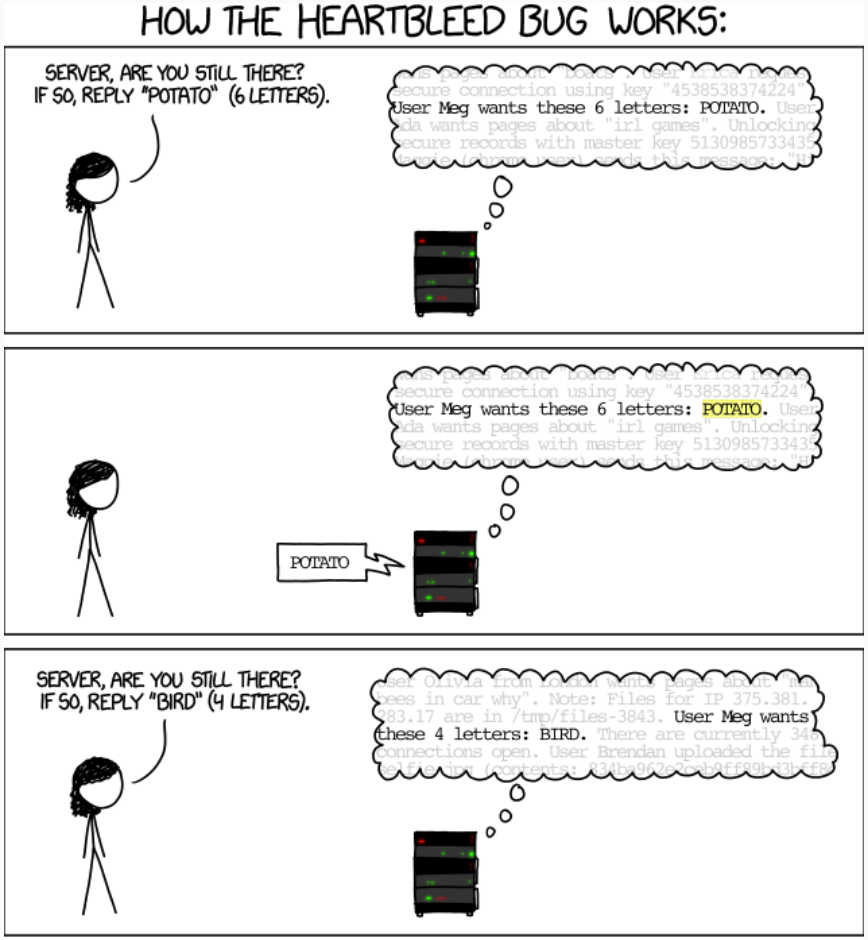
evidentemente scritto per errore, pone a 0, oltre a tutti gli elementi dell'array, anche i `sizeof(int) * 10` byte successivi, corrompendo il contenuto della memoria e potenzialmente causando un errore di segmentazione.

Questo avviene perché il C, a differenza di altri linguaggi, non controlla che l'accesso a un elemento di un array sia entro il limite di indici consentiti (da 0 alla dimensione dell'array meno 1).

Heartbleed

- Vulnerabilità dell’implementazione del protocollo TLS (Transport Layer Security) nella libreria crittografica OpenSSL
- Introdotta nel 2012, pubblicata e risolta nel 2014
- Effetto: rubate informazioni riservate (come password degli utenti) su molti dei servizi Internet più usati (fra cui BitBucket)
- Dovuta a un errore di programmazione (mancato array bound checking) dell’estensione *heartbeat* del protocollo.

https://xkcd.com/1354/



Credenziali

```
#include <stdio.h>

typedef struct {
    char nome_utente[10];
    char password[10];
} credenziali;

int main()
{
    credenziali c;
    printf("Registrazione\n");
    printf("Digita␣nome␣utente\n");
    gets(c.nome_utente);
    printf("Digita␣password\n");
    gets(c.password);
    printf("...\n\n\nConsultazione␣da␣parte␣di␣altro␣
        utente\n");
    printf("%s\n",c.nome_utente);
}
```

Comportamento

Che cosa succede se l’utente digita un nome utente di 10 caratteri?

Vulnerabilità

1. Se il nome utente salvato in `nome_utente` è di 10 o più caratteri, "sconfina" nel campo `password`; in particolare, tutti i caratteri di `nome_utente` saranno diversi da `0`.
2. Successivamente, la password digitata viene salvata nel campo `password`.
3. La stampa del campo `nome_utente` non trova il terminatore `0` all'interno del campo, e continua nel campo `password`, stampandolo di seguito.

Password

```c
#include <stdio.h>

int main() {
  int accesso_consentito = 0;
  char password[12];

  printf("Password?\n");
  scanf("%s", password);

  if (strcmp(password, "unife") == 0)
    accesso_consentito = 1;
  if (accesso_consentito)
    printf("Accesso consentito\n");
  else
    printf("Accesso negato\n");
}
```

Comportamento

Il programma si comporta correttamente se l’utente digita password di non più di 11 caratteri.

Che cosa succede se l’utente digita una password di 12 caratteri? E di 13 caratteri? Perché?

Vulnerabilità

La variabile `accesso_consentito` inizia 12 byte dopo il primo byte dell'array `password`.

- Se l'utente digita una password di meno di 12 caratteri, questa rimane contenuta nell'array `password`.
- Se l'utente digita una password di 12 caratteri, il terminatore sovrascrive il primo byte di `accesso_consentito`, che quindi rimane `0`.
- Se l'utente digita una password di 13 o più caratteri, la parte finale della stringa sovrascrive `password` con valori sicuramente diversi da `0`, rendendolo quindi vero e (nel programma) consentendo l'accesso.

Come evitare?

Occorre assicurarsi che il buffer abbia spazio sufficiente per memorizzare la stringa.

Le funzioni `gets()` e `scanf("%s",...)` non consentono di limitare il numero di caratteri letti.

A questo scopo si può usare una combinazione delle funzioni

- `fgets(__buffer__,__nMax__,stdin)`, che legge da input (`stdin`) un massimo di `__nMax__` caratteri
- `sscanf(__buffer__, "%s", __s__)`, che legge la stringa `__s__` non da standard input, ma dall'array `__buffer__`

In molti casi è opportuno scartare eventuali caratteri non letti da `fgets`.

Lettura sicura di una stringa

```c
#include <string.h>
#include <stdio.h>

int min(int a, int b) {
  return a <= b ? a : b;
}

void leggiStringa(char* s, int lunghezza) {
  int c;
  char riga[81];
  fgets(riga, min(81, lunghezza + 1), stdin);
  if (riga[strlen(riga) - 1] != '\n')
    do {
      c = getchar();
    } while (c != EOF && c != '\n');
  sscanf(riga, "%s", s);
}
```