Sicurezza

Marco Alberti

Dipartimento
di Matematica
e Informatica

Università
degli Studi
di Ferrara

Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 28 dicembre 2020

Sommario

Sommario

Sicurezza

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

- Installazione di virus, trojan, spyware MALWARE RANSOMWARE
- 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 INGEGNERIA SOCIALE
- Amministrazione di sistema e di rete
- Programmazione

Termini

- risorsa: 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

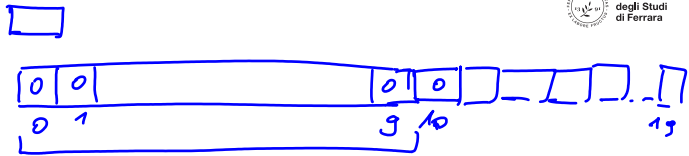
- integrità: un attacco può corrompere (modificare indebitamente) i dati gestiti dal sistema o il sistema stesso.
- confidenzialità: molti sistemi gestiscono informazioni che devono essere accessibili solo a persone autorizzate; un attacco può consentirne l'accesso ad agenti non autorizzati
- disponibilità: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

Sommario

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.
  Nelle prossime diapositive vediamo alcuni esempi di vulnerabilità.

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).

Password

Università degli Studi di Ferrara

300_sicurezza/password.c

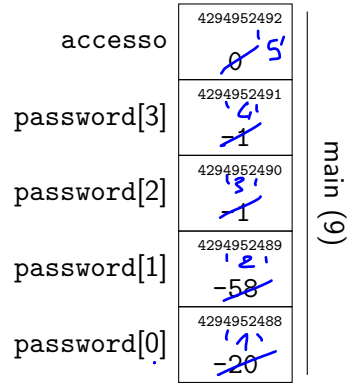
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5    int accesso = 0;
6    char password[4];
7
8    printf("Password?\n");
9    scanf("%s", password);
10
11   if (strcmp(password, "abc") == 0)
12     accesso = 1;
13   if (accesso)
14     printf("Accesso consentito\n");
15   else
16     printf("Accesso negato\n");
17 }
```

Comportamento

Il programma si comporta correttamente se l'utente digita password di non più di 3 caratteri.
Che cosa succede se l'utente digita una password di 4 caratteri? E di 5 caratteri? Perché?

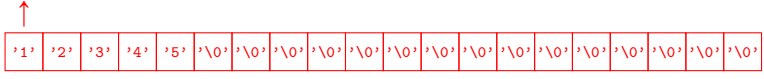
Password

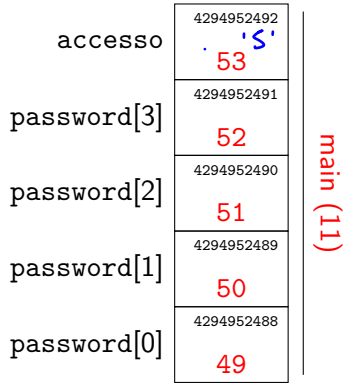
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5    int accesso = 0;
6    char password[4];
7
8    printf("Password?\n");
9    scanf("%s", password);
10
11   if (strcmp(password, "abc") == 0)
12     accesso = 1;
13   if (accesso)
14     printf("Accesso consentito\n");
15   else
16     printf("Accesso negato\n");
17 }
```



Password

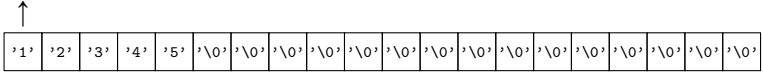
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5    int accesso = 0;
6    char password[4];
7
8    printf("Password?\n");
9    scanf("%s", password);
10
11   if (strcmp(password, "abc") == 0)
12     accesso = 1;
13   if (accesso)
14     printf("Accesso consentito\n");
15   else
16     printf("Accesso negato\n");
17 }
```

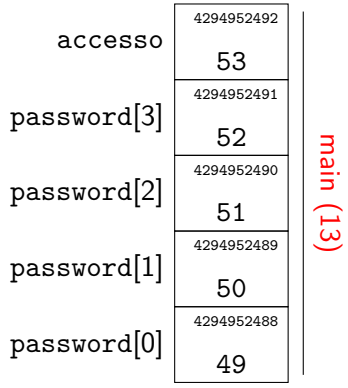




Password

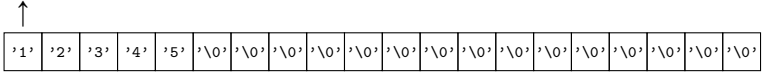
```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5    int accesso = 0;
6    char password[4];
7
8    printf("Password?\n");
9    scanf("%s", password);
10
11   if (strcmp(password, "abc") == 0)
12     accesso = 1;
13   if (accesso)
14     printf("Accesso consentito\n");
15   else
16     printf("Accesso negato\n");
17 }
```

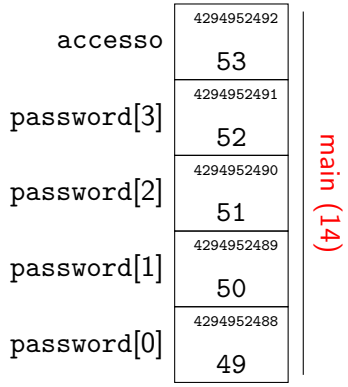




Password

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5    int accesso = 0;
6    char password[4];
7
8    printf("Password?\n");
9    scanf("%s", password);
10
11   if (strcmp(password, "abc") == 0)
12     accesso = 1;
13   if (accesso)
14     printf("Accesso consentito\n");
15   else
16     printf("Accesso negato\n");
17 }
```



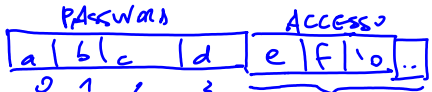


Vulnerabilità

La variabile `accesso_consentito` inizia 4 byte dopo il primo byte dell'array `password`.

- Se l'utente digita una password di meno di 4 caratteri, questa rimane contenuta nell'array `password`.
- Se l'utente digita una password di 4 caratteri, il terminatore sovrascrive il primo byte di `accesso_consentito`, che quindi rimane `0`.
- Se l'utente digita una password di 5 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.

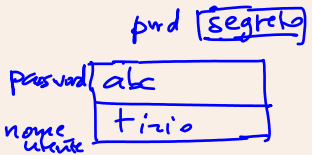
Questo è un esempio di buffer overflow, cioè scrittura di una sequenza di byte in un'area di memoria troppo piccola per contenerla, con conseguente sovrascrittura delle celle di memoria adiacenti.



Credenziali

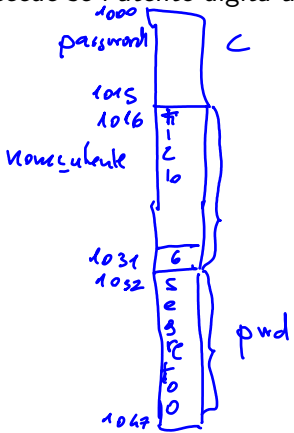
300_sicurezza/credenziali.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  typedef struct {
5    char password[16];
6    char nome_utente[16];
7  } credenziali;
8
9  int main() {
10   char pwd[16];
11   credenziali c;
12   printf("Nome utente?\n"); scanf("%s", c.nome_utente);
13   printf("Password?\n"); scanf("%s", c.password);
14   strcpy(pwd, "segreto");
15   if (strcmp(c.password, pwd) == 0)
16     printf("Accesso consentito per %s\n", c.nome_utente);
17   else
18     printf("Accesso negato per %s\n", c.nome_utente);
19 }
```



Comportamento

Che cosa succede se l'utente digita un nome utente di 16 caratteri?



Vulnerabilità

1. Se il nome utente salvato in `c.nome_utente` è di 16 o più caratteri, "sconfina" nell'array `pwd`; in particolare, tutti i caratteri di `c.nome_utente` saranno diversi da `0`.
2. Successivamente, nell'array `pwd` viene copiata la password.
3. La stampa di `c.nome_utente` non trova il terminatore `0` all'interno del campo, e continua nell'array `pwd`, stampandolo di seguito.

Come difendersi da utenti maligni?

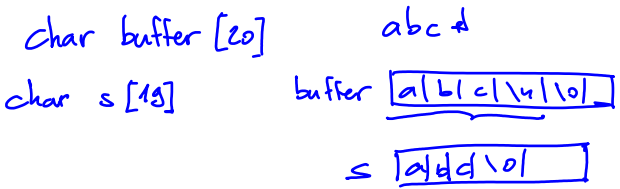
NO BUFFER OVERFLOW

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

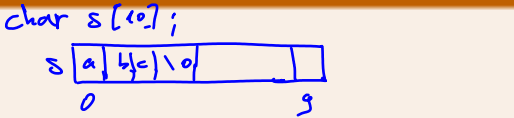
Università degli Studi di Ferrara

300_sicurezza/stringhe.c

```c
#include <stdio.h>
#include <string.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);
}
```



Password - versione sicura

300_sicurezza/password-sicuro.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include "stringhe.h"
4
5  int main() {
6    int accesso_consentito = 0;
7    char password[4];
8
9    printf("Password?\n");
10   leggiStringa(password,3);
11
12   if (strcmp(password, "abc") == 0)
13     accesso_consentito = 1;
14   if (accesso_consentito)
15     printf("Accesso consentito\n");
16   else
17     printf("Accesso negato\n");
18 }
```

Credenziali - versione sicura

Università degli Studi di Ferrara

300_sicurezza/credenziali-sicuro.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include "stringhe.h"
4
5  typedef struct {
6    char password[16];
7    char nome_utente[16];
8  } credenziali;
9
10 int main() {
11   char pwd[16];
12   credenziali c;
13   printf("Nome utente?\n"); leggiStringa(c.nome_utente, 15);
14   printf("Password?\n"); leggiStringa(c.password, 15);
15   strcpy(pwd, "segreto");
16   if (strcmp(c.password, pwd) == 0)
17     printf("Accesso consentito per %s\n", c.nome_utente);
18   else
19     printf("Accesso negato per %s\n", c.nome_utente);
20 }
```

Per saperne (molto) di più

https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard