CS0424IT — ESERCITAZIONE S2L5 - PROGETTO BUG HUNTING

Simone La Porta

GitHub

TRACCIA

Per agire come un Hacker bisogna capire come pensare fuori dagli schemi. L'esercizio di oggi ha lo scopo di allenare l'osservazione critica. Dato il codice in allegato, si richiede allo studente di:

- *Capire cosa fa il programma senza eseguirlo.*
- *Individuare dal codice sorgente le casistiche non standard che il programma non gestisce (esempio, comportamenti potenziali che non sono stati contemplati).*
- *Individuare eventuali errori di sintassi / logici.*
- *Proporre una soluzione per ognuno di essi.*

SVOLGIMENTO

Il programma fornito è un semplice assistente digitale che permette di moltiplicare e dividere due numeri, oppure inserire una stringa, come mostrato in Figura 1. Tuttavia, il codice contiene diversi errori e mancanze nella gestione dei casi non standard. Questo documento analizza il codice, identifica gli errori e propone delle soluzioni, classificando ciascun errore e discutendo le implicazioni in ambito di cybersecurity. In allegato il codice corretto e migliorato.

```
void menu ()
{
    printf ("Benvenuto, sono un assitente digitale, posso aiutarti a sbrigare
        alcuni compiti\n");
    printf ("Come posso aiutarti?\n");
    printf ("A >> Moltiplicare due numeri\nB >> Dividere due numeri\nC >>
        Inserire una stringa\n");

}
```

Figura 1: Menu iniziale.

Identificazione degli errori

Errori tipo di dato

- **Errore (riga 14)**

  La variabile `scelta` è dichiarata come `char` ma `scanf` usa `"%d"`.

  - **Soluzione**: Utilizzare `"%c"` per leggere un `char`.

```
 9  int main ()
10
11  {
12      char scelta = {'\0'};
13      menu ();
14      scanf ("%d", &scelta);
15
16      switch (scelta)
17      {
18          case 'A':
19          moltiplica();
20          break;
21          case 'B':
22                  dividi();
23                  break;
24          case 'C':
25                  ins_string();
26                  break;
27      }
28
29  return 0;
30
31  }
```

(a) Originale.

```
 8  int main()
 9  {
10      char scelta = '\0';
11      menu();
12      scanf(" %c", &scelta);
13
14      switch (scelta)
15      {
16          case 'A':
17          case 'a':
18              moltiplica();
19              break;
20          case 'B':
21          case 'b':
22              dividi();
23              break;
24          case 'C':
25          case 'c':
26              ins_string();
27              break;
28          default:
29              printf("Scelta non valida.\n");
30              break;
31      }
32
33      return 0;
34  }
```

(b) Corretto.

- **Errore (riga 47)**

  In `moltiplica()`, la funzione `scanf` utilizza `"%f"` per leggere un `short int`.

  - **Soluzione**: Utilizzare `"%hd"` per `short int` oppure dichiarare le variabili come `int` e leggerle utilizzando il comando `"%d"`.

```
43 void moltiplica ()
44 {
45     short int  a,b = 0;
46     printf ("Inserisci i due numeri da moltiplicare:");
47     scanf ("%f", &a);
48     scanf ("%d", &b);
49
50     short int prodotto = a * b;
51
52     printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
53 }
```

(a) Originale.

```
45 void moltiplica()
46 {
47     int a, b;
48     printf("Inserisci i due numeri da moltiplicare: ");
49     if (scanf("%d %d", &a, &b) != 2) {
50         printf("Errore: Input non valido.\n");
51         return;
52     }
53
54     int prodotto = a * b;
55     printf("Il prodotto tra %d e %d è: %d\n", a, b, prodotto);
56 }
```

(b) Corretto.

- **Errore (riga 64)**

  In `dividi()`, la divisione viene eseguita con l'operatore modulo (`%`) anziché con l'operatore divisione (`/`).

  - **Soluzione**: Utilizzare l'operatore corretto per la divisione (`/`).

  In `dividi()`, la divisione viene dichiarata come `int` anche se può restituire valori decimali.

  - **Soluzione**: Necessario dichiararla come `float`.

```
56 void dividi ()
57 {
58         int  a,b = 0;
59         printf ("Inserisci il numeratore:");
60         scanf ("%d", &a);
61     printf ("Inserisci il denumeratore:");
62         scanf ("%d", &b);
63
64         int divisione = a % b;
65
66         printf ("La divisione tra %d e %d e': %d", a,b,divisione);
67 }
```

(a) Originale.

```
58 void dividi()
59 {
60     int a, b;
61     printf("Inserisci il numeratore: ");
62     if (scanf("%d", &a) != 1) {
63         printf("Errore: Input non valido.\n");
64         return;
65     }
66     printf("Inserisci il denominatore: ");
67     if (scanf("%d", &b) != 1) {
68         printf("Errore: Input non valido.\n");
69         return;
70     }
71     if (b == 0) {
72         printf("Errore: Divisione per zero.\n");
73         return;
74     }
75
76     float divisione = (float)a / b;
77     printf("La divisione tra %d e %d è: %.2f\n", a, b, divisione);
78 }
```

(b) Corretto.

Errori di buffer overflow

- **Errore (riga 77)**

In `ins_string()`, l'uso di `scanf` senza specificare la lunghezza massima del buffer può causare un buffer overflow.

- **Soluzione**: Utilizzare `scanf("%9s", stringa);` per limitare l'input a 9 caratteri.

```
73 void ins_string ()
74 {
75     char stringa[10];
76         printf ("Inserisci la stringa:");
77         scanf ("%s", &stringa);
78 }
```

(a) Originale.

```
80 void ins_string()
81 {
82     char stringa[10];
83     printf("Inserisci la stringa (max 9 caratteri): ");
84     scanf("%9s", stringa);
85     printf("Hai inserito: %s\n", stringa);
86 }
```

(b) Corretto.

Errori di logica

- **Errore (riga 16)**

  La funzione `menu()` non gestisce un input non valido, portando a comportamenti indefiniti.

  - **Soluzione**: Aggiungere una gestione degli errori nel `switch` per input non validi.

```
 9 int main ()
10
11 {
12     char scelta = {'\0'};
13     menu ();
14     scanf ("%d", &scelta);
15
16     switch (scelta)
17     {
18         case 'A':
19         moltiplica();
20         break;
21         case 'B':
22                 dividi();
23                 break;
24         case 'C':
25                 ins_string();
26                 break;
27     }
28
29 return 0;
30
31 }
```

(a) Originale.

```
 8 int main()
 9 {
10     char scelta = '\0';
11     menu();
12     scanf(" %c", &scelta);
13
14     switch (scelta)
15     {
16         case 'A':
17         case 'a':
18             moltiplica();
19             break;
20         case 'B':
21         case 'b':
22             dividi();
23             break;
24         case 'C':
25         case 'c':
26             ins_string();
27             break;
28         default:
29             printf("Scelta non valida.\n");
30             break;
31     }
32
33     return 0;
34 }
```

(b) Corretto.

- **Errore (riga 43)**

  In `moltiplica()`, l'operazione di moltiplicazione è eseguita su variabili `short int`, che potrebbe portare a un overflow aritmetico.

  - **Soluzione**: Usare `int` o `long int` per evitare l'overflow.

```c
void moltiplica ()
{
    short int  a,b = 0;
    printf ("Inserisci i due numeri da moltiplicare:");
    scanf ("%f", &a);
    scanf ("%d", &b);

    short int prodotto = a * b;

    printf ("Il prodotto tra %d e %d e': %d", a,b,prodotto);
}
```

(a) Originale.

```c
void moltiplica()
{
    int a, b;
    printf("Inserisci i due numeri da moltiplicare: ");
    if (scanf("%d %d", &a, &b) != 2) {
        printf("Errore: Input non valido.\n");
        return;
    }

    int prodotto = a * b;
    printf("Il prodotto tra %d e %d è: %d\n", a, b, prodotto);
}
```

(b) Corretto.

- **Errore (riga 56)**

  In `dividi()`, non viene gestito il caso in cui il denominatore sia zero.

  - **Soluzione**: Aggiungere un controllo per verificare che il denominatore non sia zero prima di eseguire la divisione.

```c
void dividi ()
{
        int  a,b = 0;
        printf ("Inserisci il numeratore:");
        scanf ("%d", &a);
    printf ("Inserisci il denumeratore:");
        scanf ("%d", &b);

        int divisione = a % b;

        printf ("La divisione tra %d e %d e': %d", a,b,divisione);
}
```

(a) Originale.

```c
void dividi()
{
    int a, b;
    printf("Inserisci il numeratore: ");
    if (scanf("%d", &a) != 1) {
        printf("Errore: Input non valido.\n");
        return;
    }
    printf("Inserisci il denominatore: ");
    if (scanf("%d", &b) != 1) {
        printf("Errore: Input non valido.\n");
        return;
    }
    if (b == 0) {
        printf("Errore: Divisione per zero.\n");
        return;
    }

    float divisione = (float)a / b;
    printf("La divisione tra %d e %d è: %.2f\n", a, b, divisione);
}
```

(b) Corretto.

Implicazioni in ambito cybersecurity

Il codice fornito presenta diverse vulnerabilità che possono essere sfruttate da un attaccante per compromettere la sicurezza del sistema. Di seguito sono descritte le principali debolezze e le loro implicazioni in termini di sicurezza informatica.

Debolezze del codice

Uso di `scanf` per la lettura dell'input L'uso della funzione `scanf` senza una corretta verifica della validità dell'input può portare a comportamenti imprevedibili o al crash del programma.

- **Implicazione**: Possibilità di **buffer overflow** o **injection attacks**.

Dichiarazione e inizializzazione delle variabili Nella funzione `moltiplica`, i tipi di variabili usate (`short int` e `float`) e la mancanza di inizializzazione coerente possono portare a risultati imprevisti.

- **Implicazione**: Possibilità di **corruzione di dati** e **comportamenti indefiniti**.

Gestione dell'input dell'utente L'uso di `scanf` con `%s` senza specificare la dimensione massima del buffer può causare un **buffer overflow**.

- **Implicazione**: Possibilità di **esecuzione di codice arbitrario** tramite overflow del buffer.

Controllo dei tipi di input Il controllo del tipo di input per la scelta del menu è errato, utilizzando `%d` per leggere un carattere.

- **Implicazione**: **Fallimento nell'interpretare correttamente l'input**, portando a comportamenti imprevisti.

Operazioni matematiche Nella funzione `dividi`, l'operazione di modulo (%) è utilizzata invece di una divisione, e non c'è controllo per divisione per zero.

- **Implicazione**: Possibilità di **crash** per divisione per zero e **comportamenti errati** che possono portare a denial of service.

Le vulnerabilità descritte dimostrano come una gestione impropria dell'input e una cattiva progettazione del codice possano compromettere la sicurezza del sistema. È essenziale adottare buone pratiche di programmazione e verificare accuratamente l'input dell'utente per prevenire exploit e attacchi malevoli.