Qualità di sicurezza dei programmi

Lezione 8 di Sicurezza dei sistemi informatici 1

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2007-8

Indice

sicurezza dei programmi

oggetto del resto del corso è la **sicurezza del software**, considerata:
- dapprima **in termini generali**, validi per qualsiasi tipo di software
- quindi per due **categorie speciali** di software, dalla cui sicurezza dipende in maniera critica la sicurezza di quasi tutti i sistemi informatici:
  - **sistemi operativi**
  - **basi di dati**

nella lezione introduttiva, la sicurezza dei sistemi informatici è stata articolata nelle tre caratteristiche di: **riservatezza, integrità, disponibilità**

articolazioni più fini sono in pratica necessarie:
- per stabilire cosa si intende quando si qualifica come "sicuro" un determinato programma
- per determinare i **requisiti di sicurezza** che un programma deve soddisfare per poter essere ritenuto sicuro

le qualità di sicurezza di un programma dipendono dalle sue **funzionalità**
- la norma ISO 9126 classifica le qualità di sicurezza del software come una sottocategoria della categoria delle qualità di funzionalità
- ciò perché la norma definisce la sicurezza di un prodotto software come la sua "capacità di prevenire accessi non autorizzati alle informazioni e funzioni gestite dal prodotto"

risoluzione degli errori

tradizionalmente, si ritiene "sicuro" un programma quando questo "non contiene errori"
- cos'è un **"errore"** ?

occorre innanzitutto distinguere fra cause ed effetti, spesso confusi nello stesso termine:
- un **malfunzionamento** è una deviazione del comportamento di un programma da quello prescritto (dai requisiti)
- i malfunzionamenti sono causati da **errori** (ad es., di interpretazione dei requisiti)
- gli errori si possono **propagare**, cioè essere causa di altri errori
- non tutti gli errori danno luogo a (manifestazione di) malfunzionamenti
  - gli errori più "subdoli", cioè più difficili da scoprire, sono quelli che danno luogo a malfunzionamenti solo in condizioni molto particolari (ingl. race conditions)

metodi tradizionali di risoluzione degli errori si basano sul collaudo di una squadra di esperti (ingl. tiger team), sfidata a tentare di penetrare nel sistema e violarne la sicurezza
- all'eventuale successo del tentativo segue l'identificazione dell'errore e la sua risoluzione: approccio penetrate and patch

purtroppo non c'è alcuna garanzia che la patch non sia fonte di altri errori!

difetti di sicurezza dei programmi

una definizione del concetto di **difetto di sicurezza di un programma:**

> comportamento non conforme alle attese di sviluppatori e utenti

valutazione critica:

> questa definizione coglie solo i **malfunzionamenti**, giacché solo questi sono osservabili nel **comportamento**
>
> inoltre dipende da non meglio precisate "attese di sviluppatori e utenti", il che introduce un discutibile elemento di soggettività nella definizione

meglio assumere una **specifica di requisiti di sicurezza** per il programma
e quindi identificare i difetti di sicurezza con le **violazioni di tali requisiti**

tassonomia dei difetti dei programmi

tassonomia di (Landwehr et al., 1993) dei **difetti dei programmi:**

intenzionali:

- nocivi
- non nocivi

involontari:

- errore di convalida (incompleta o incoerente)
- errore di dominio
- serializzazione e alias
- autenticazione e identificazione inadeguate
- violazione di condizione limite
- altri errori logici

vengono forniti nel seguito, e nella prossima lezione, alcuni esempi di difetti involontari

errori non intenzionali, un esempio

ecco un caso di errore di programmazione, sicuramente involontario, classificabile come errore logico secondo la tassonomia dei difetti vista sopra

l'errore è riscontrabile in un esempio di programmazione in C++ nel testo (Sedgewick, 2003), che a p. 41 propone due metodi simili, espressi da rispettive istruzioni C++, per il calcolo del più piccolo intero maggiore di lg **N** :

```
for (lgN = 0; N > 0; lgN++, N /= 2) ;
for (lgN = 0, t = 1; t < N; lgN++, t += t) ;
```

i due metodi **non** sono equivalenti

invero, posto che il valore della variabile intera **lgN** debba essere, all'uscita dal ciclo, il più piccolo intero maggiore di lg **N**, il secondo metodo è erroneo

esercizio: trovare e correggere l'errore nel secondo metodo

buffer overflow non intenzionale

un buffer è un'area di memoria di capacità finita, ad es. quella allocata a un array o a una stringa

si ha un overflow (trabocco) dal buffer quando si accede alla struttura dati, a cui esso è allocato, con un valore dell'indice esterno all'intervallo corrispondente a locazioni del buffer

la conseguenza del buffer overflow dipende dalla collocazione della locazione di memoria a cui si tenta l'accesso erroneo, che può trovarsi nell'area:

dati dell'**utente**, o **codice** dell'**utente**, o **dati** di **sistema**, o **codice** di **sistema**

è difficile predisporre una protezione generale dal buffer overflow, perché:

il valore dell'indice non è generalmente noto a tempo di compilazione
controlli a tempo di esecuzione penalizzerebbero eccessivamente le prestazioni dei programmi

ecco un esempio di buffer overflow sicuramente non intenzionale, che si manifesta solo al verificarsi di una condizione molto rara, tipico esempio di race condition :

il testo (Sedgewick, 2003) propone un programma per la simulazione del lancio di monete (Programma 3.7, p. 88), dove si considerano **M** esperimenti, ciascuno dei quali consta di **N** lanci

se **k** è il numero di volte che il lancio dà "testa" in un esperimento, si incrementa **f[k]** di 1
l'allocazione di memoria per l'array **f** prevede **N+1** valori dell'indice (da 0 a **N**)
tuttavia ... nel ciclo di simulazione dei lanci di un esperimento, la variabile di iterazione del ciclo va da 0 a **N** incluso, dunque vengono effettivamente simulati **N+1** lanci!
si ha dunque un buffer overflow quando tutti i lanci di un esperimento hanno esito "testa"