Programmazione Sicura

Buffer Overflow
Laboratorio: Cap. 6

Indice

- Il caso `s03.c` della lezione lab_1
- Buffer Overflow over non control data
- Installazione JDK
- Installazione di Codesonar client
- Esempi di bug sui *buffer*, di varia natura
- Static Analisys con Codesonar

Il caso `s03.c` 1/3

- Nel esempio `s03.c` se in input si passa una stringa lunga 1024 caratteri qualsiasi seguita da `ls -l`
  - viene eseguito il listing della directory.
    - Potrebbe essere un problema di sicurezza?
- Creare un file da 1024 caratteri seguiti da `ls -l` senza ritorno a capo. Copiatelo e incollatelo alla richiesta di input di s03.

```
aaaaaa.............aaaaaals -l
```

- Cosa succede? Apparentemente il programma esegue anche il comando `ls -l`

Il caso `s03.c` 2/3

- Per capire cosa accade usiamo il comando linux `strace`
  - `strace` visualizza le system call eseguite dal programma (passato come argomento) con i relativi argomenti
    - di ogni system call vengono visualizzati gli argomenti e i valori restituiti
- `strace ./s03`

Il caso `s03.c` 3/3

- La shell (bash) crea un figlio che invoca `execve` (prima linea di output di strace)
  - l'output si ferma sulla `read(0, …., 1024)` che legge al più 1024 byte dallo `stdin` (fd 0)
  - esegue i calcoli e termina.
- Che fine fa `ls -l` nell'input? (N.B. i file descriptor vengono passati al figlio e non vengono alterati da exec).
  - viene letto dalla shell che al termine del processo riprende a leggere da stdin, lo stesso di `s03` e consuma i caratteri rimanenti, ovvero il comando `ls -l`
- Esercizio: cosa succede se gli passiamo una stringa di 1025 caratteri seguita da `ls -l` ?

Buffer overflow against non control data 1/4

```
/* pseudo codice dall'esempio di 10K*/
get_medical_info()
{
boolean authorized = false;
char name [10];
authorized = check();
read_from_network (name);

if (authorized)
   show_medical_info (name);
else
   printf (“sorry, not allowed”);
}
```

Buffer overflow against non control data 2/4

```
#include <stdio.h>
int check(){ return 0; }
get_medical_info(){
    int authorized = 0;
    char name[10];
    authorized = check();
    gets(name);
    if(authorized)
        printf("your are authorized!");
    else
        printf ("sorry, not allowed");
}
```

```
void main(){
    get_medical_info();
} //vedi dopo per compilare
```

Buffer overflow against non control data 3/4

- Compilatelo e forzate il buffer overflow per stampare le informazioni riservate.
  - `make 10k_1`
- Notate che sia il compilatore che il linker vi avvisano che non dovreste usare `gets`
- Trovare un input che causi il buffer overflow e consenta di accedere ai dati riservati
  - a me sono bastati 13 caratteri in un esempio
- Cosa succede se l'input è molto più lungo?

Buffer overflow against non control data 4/4

- Abbiamo usato un buffer overflow per accedere a una sezione riservata del programma.
- RISPONDERE AL CASO DI INPUT PIU' LUNGO

Esercizi

- Copiate i sorgenti di lab3 nella cartella projects. Non fate il merge con i sorgenti dei lab precedenti.
  - È incluso un makefile completo per tutti i sorgenti
- Compilate tutto per essere certi di non avere errori.
- ATTENZIONE a RAND32() restituisce valori con segno a 32 bit (~-2*10^9 - +2*10^9) quindi anche numeri negativi.

Note sugli esercizi

- Contengono le seguenti tipologie di vulnerabilità
  - Stack based buffer overflow
  - Heap based buffer overflow
  - Buffer overread (stack e heap)
  - Buffer underread (stack e heap)
  - Buffer underwrite (stack e heap)
  - Memory leak
  - Double free
  - Use after free

Installiamo JDK

- Necessario per alcune funzioni del visualizzatore di Codesonar

```
sudo add-apt-repository ppa:webupd8team/java
```

invio per accettare

```
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

USARE TAB PER SELEZIONARE I PULSANTI NELLA GUI

```
sudo update-java-alternatives -s java-8-oracle
sudo apt-get install oracle-java8-set-default
jcontrol // ATT.NE NON COME SUPER USER
```

tab sicurezza, "modifica lista siti" -> aggiungere **http://193.205.186.5:7340** e **http://192.168.7.15:7340**

salvare e riavviare browser se necessario.

Installiamo Codesonar

- Estrarre nella home l'archivio con:
  - `tar -xvzf codesonar.....tar.gz`
- Entrare nella directory codesonarXYZ/codesonar/bin
- Comando pwd per ottenere il path completo. Selezionarlo e copiarlo negli appunti.
- Aprire il file .bashrc nella home.
- Alla fine del file aggiungere:
  - `PATH=$PATH:incollare_qui_path_codesonar_bin`
- Chiudere e riavviare terminale.
- `echo $PATH #codesonar nella PATH?`

Analisi di un progetto con Codesonar

- Testate *Codesonar* lanciando il comando `codesonar` dal terminale.
- Codesonar “osserva” la compilazione di un *progetto* per estrarre le informazioni (sorgenti e relazioni tra essi) necessarie alla static analisys.
- Queste informazioni vengono inviate allo HUB (il server) che esegue l'analisi statica e crea i report
- dalla directory con i sorgenti da testare, comando per analizzare un progetto è:
  - `codesonar analyze NOME_PROJ 193.205.186.5:7340 make TARGET`
  - Alla fine del processo, se tutto va bene sarà restituito una *url* per accedere al report. Copiatelo e incollatelo nel browser.
  - all'interno dell'università: **192.168.7.15**, all'esterno: **`193.205.186.5`**
- NOME_PROJ deve essere del tipo targetCognome.
  - esempio: s09Rossi
  - NOTA: Codesonar supporta fino a 10 analisi di progetti concorrenti

Esempio

- `codesonar analyze s07Rizzuti 193.205.186.5:7340 make s07`
- `Created /home/luca/.../sources_prof/s07Rizzuti.conf from /home/luca/progs/codesonar-4.0p1/codesonar/template.conf`
- `codesonar: Logging to s07Rizzuti.prj_files/log.txt...`
- `gcc  -o s07 s07.c io.c`
- `codesonar: Building s07Rizzuti.prj...`
- `codesonar: Analysis initialized.`
- `codesonar: Live progress and results are visible at:`
- `codesonar: http://193.205.186.5:7340/analysis/7.html`
- Salvate le url in un file testo per confrontarli dopo le correzioni

Codesonar – schermata principale

Codesonar – dettagli warning

Show Events | Options

s07 *(/home/luca/Documenti/Lavoro/borsa_mbda/lab3/sources_prof/s07.c)*

```
 5  void s07()
 6  {
 7      int data;
 8      data = -1;
 9      char inputBuffer[CHAR_ARRAY_SIZE] = "";
10      if (fgets(inputBuffer, CHAR_ARRAY_SIZE, stdin) != NULL)
11      {
12          data = atoi(inputBuffer);
13      }
14      else
15      {
16          printLine("fgets() failed.");
17      }
18
19      int i;
20      int buffer[10] = { 0 };
21      if (data >= 0)
22      {
23          buffer[data] = 1; //data can be > 10
```

⚠ Event 2: atoi() returns a potentially dangerous value [?].
- This determines the position accessed during the buffer overrun later.

▲ ▼ hide

Posizionate il mouse su data e su inputBuffer
InputBuffer è tainted → data è tainted

Buffer Overrun
This code could write past the end of buffer.
- The code writes 4 bytes starting at offset 4 * data from the beginning of buffer, whose capacity is 40 bytes.
  - The number of bytes written could exceed the number of allocated bytes beyond that offset.
  - 4 * data evaluates to 4 * atoi(inputBuffer)s07.c:12, which is bounded below by 0. See related event 3.
- If 4 * data is higher than 36, an overrun will occur. The analysis cannot rule ou… warning.
- The overrun occurs in stack memory.

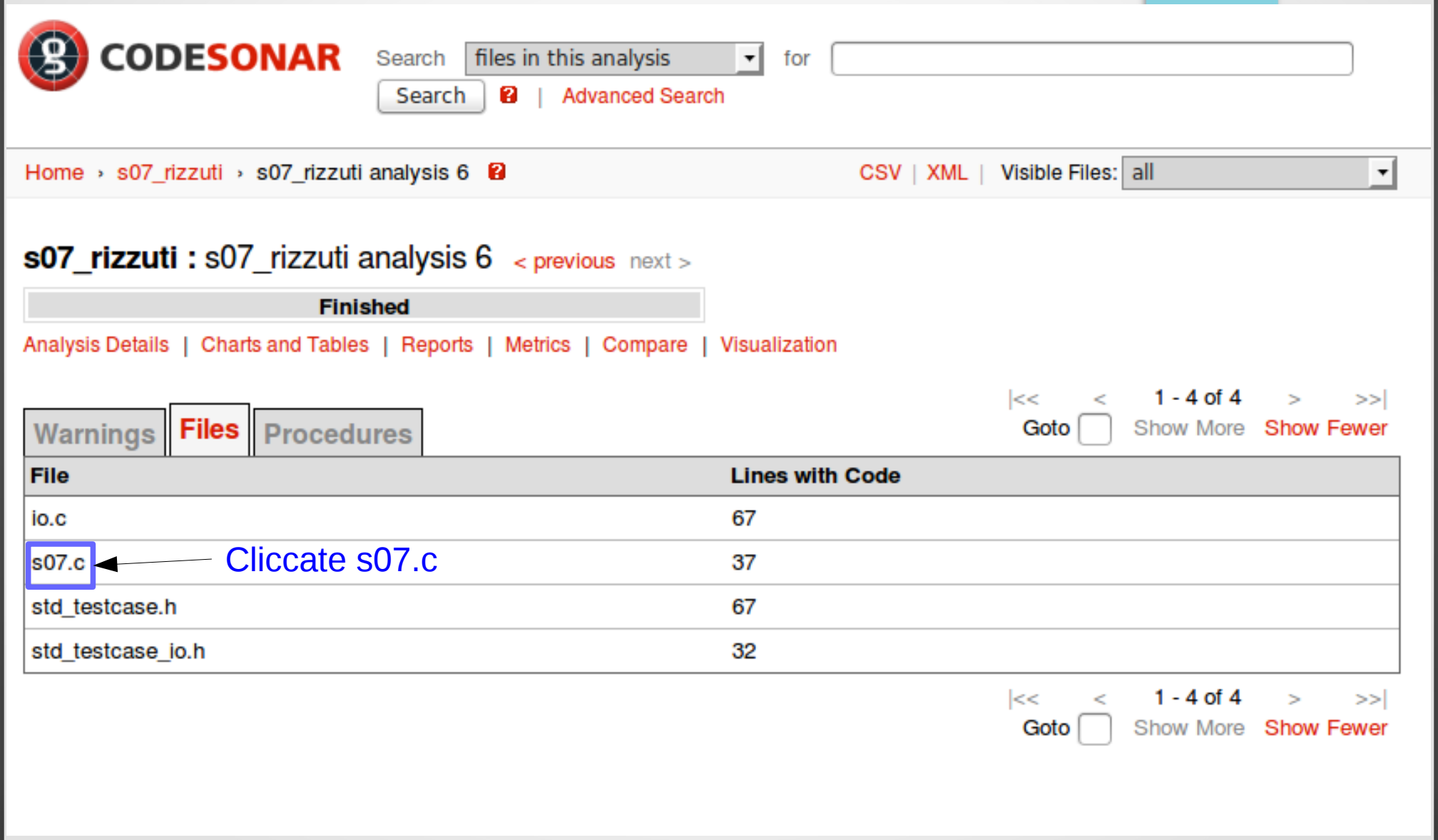
The issue can occur if the highlighted code executes.

See related event 3.
Show: All events | Only primary events

info | def | **data**

[–] Definitions (all)
(variable) s07.c: 7, **int** data

Codesonar – file view



Codesonar – dettagli file

|<< < 1 - 45 of 45 > >>|
Goto Show More Show Fewer

Options

/home/luca/Documenti/Lavoro/borsa_mbda/lab3/sources_prof/s07.c

```
 1 #include "std_testcase.h"
 2 #define CHAR_ARRAY_SIZE (3 * sizeof(data) + 2)
 3
 4 //stack buffer overflow
 5 void s07()
 6 {
 7     int data;
 8     data = -1;
 9     char inputBuffer[CHAR_ARRAY_SIZE] = "";
10     if (fgets(inputBuffer, CHAR_ARRAY_SIZE, stdin) != NULL)
11     {
12         data = atoi(inputBuffer);
13     }
14     else
15     {
16         printLine("fgets() failed.");
17     }
18
19     int i;
20     int buffer[10] = { 0 };
21     if (data >= 0)
22     {
23         buffer[data] = 1; //data can be > 10
24         for(i = 0; i < 10; i++)
25         {
26             printIntLine(buffer[i]);
27         }
28     }
29     else
30     {
31         printLine("ERROR: Array index is negative.");
32     }
33 }
34
```

Line 23 has 1 warning:
Buffer Overrun (warning 14.83)

data