



SAPIENZA
UNIVERSITÀ DI ROMA

Titolo:

Analisi delle vulnerabilità di sicurezza
dell'applicazione open-source rsync 3.0.5



Professore

Francesco Parisi-Presicce

Studenti

Antonio Arnesano – Paolo Parlapiano

A.A. 2008/2009



Introduzione

- Panoramica su rsync
- Tool utilizzati per l'analisi
- Confronto dei tool
- TOC-TOU
- Bugs
- Conclusioni



Che cos'è rsync?

Rsync è un'applicazione open source per piattaforme UNIX-LIKE che permette di:

- trasferire file da un server ad un client e viceversa
- effettuare una sincronizzazione tra server e client



Caratteristiche

- Copia solo le porzioni di file che differiscono -> velocità
- Comprime le porzioni di file che devono essere trasferite
- Permette di utilizzare il protocollo di sicurezza SSH per il trasferimento dei dati



Altre caratteristiche

Rsync è un tool di backup molto versatile che presenta inoltre le seguenti caratteristiche:

- non richiede privilegi di root
- utilizza la tecnica pipeline per il trasferimento dei file



Come funziona?

Su una macchina deve essere configurato un server che esegue rsync in modalità demone

Occorre installare e configurare rsync su tutte le altre macchine che devono sincronizzarsi

Attraverso l'algoritmo di rsync di trasmissione dati si ottiene un sistema di trasferimento efficiente...



Esecuzione

Local:

- `rsync [OPTION...] SRC... [DEST]`

Access via remote shell:

- `rsync [OPTION...] [USER@]HOST:SRC... [DEST]`

Access via rsync daemon:

- `rsync [OPTION...] SRC...rsync://[USER@]HOST[:PORT]/DEST`



Tools utilizzati per l'analisi

I tools utilizzati per l'analisi sono:

- flawfinder
- its4
- PScan



Caratteristiche generali

Eseguono una scansione statica di codice sorgente C e C++,
in modo da poter rilevare delle potenziali vulnerabilità di
sicurezza.

Eseguono un semplice parsing del codice sorgente,
evidenziando tutte le istruzioni che sono potenzialmente
oggetto di vulnerabilità.



its4

L'esecuzione di its4 sui sorgenti di rsync ha portato in breve ai seguenti risultati:

- 77 possibili vulnerabilità di livello Urgent
- 10 possibili vulnerabilità di livello Very Risky
- 37 possibili vulnerabilità di livello Risky
- 24 possibili vulnerabilità di livello Some Risk



flawfinder

L'esecuzione di flawfinder ha portato in breve ai seguenti risultati:

- 6 possibili vulnerabilità di livello 5/5
- 60 possibili vulnerabilità di livello 4/5
- 16 possibili vulnerabilità di livello 3/5
- 230 possibili vulnerabilità di livello 2/5
- 180 possibili vulnerabilità di livello 1/5



PScan

L'esecuzione di PScan ha portato al seguente risultato:

- 15 possibili vulnerabilità senza classificazione in base alla gravità



Prime considerazioni

Una prima considerazione sui risultati ottenuti porterebbe a concludere che:

- flawfinder tende ad aumentare il “rumore”, rilevando più falsi positivi rispetto agli altri tools
- PScan tende ad aumentare il “silenzio”: visti i pochi risultati probabilmente sono troppi i falsi negativi
- its4 sembra essere il tool che offre il miglior compromesso silenzio/rumore



Più da vicino: fprintf()

Secondo l'analisi di its4, le chiamate alla funzione fprintf(...) sono quelle con il livello di rischio più alto. Il tool rileva più di 30 chiamate a tale funzione.

In totale contrapposizione, flawfinder ne segnala una soltanto, e classifica il tipo di vulnerabilità con livello 4/5.

PScan invece, fa notare che le chiamate a fprintf() sono costruite in modo corretto...



Più da vicino: fprintf()

Seguono i messaggi segnalati in merito a fprintf() dai tre tools:

- its4:

“Non-constant format strings can often be attacked. Use a constant format string.”

- flawfinder:

“If format strings can be influenced by an attacker, they can be exploited. Use a constant for the format specification.”

- PScan:

“Last argument is constant string: OK”



Più da vicino: fprintf()

its4 e flawfinder eseguono un'analisi semplice dei codici sorgente: non appena rilevano l'uso di una funzione o di un'istruzione potenzialmente dannosa, la segnalano all'utente

Sembra invece che PScan esegua un controllo più approfondito del codice, ma è doveroso osservare che il suo campo d'azione è molto più limitato



Più da vicino: fprintf()

Dopo un'attenta analisi di tutte le chiamate alla funzione fprintf(), abbiamo infatti constatato che:

- se la stringa da stampare è contenuta in una variabile, nella chiamata sono sempre incluse le specifiche di conversione
- in caso contrario, viene da sé che la chiamata alla funzione è già sicura.



Generalizzazione

I tools restituiscono la stessa tipologia di warnings anche per quanto riguarda le funzioni `printf()`, `snprintf()`, `vsprintf()`, `syslog()` ecc..

Dall'analisi del codice di `rsync` è emerso che tutte le chiamate a tali funzioni sono eseguite in sicurezza.



PScan, unica eccezione

PScan è più “intelligente”

Supponiamo di fargli processare la seguente istruzione presa da rsync:

```
fprintf( stdout , " -l, --link-times display the time on a symlink\n");
```

Il tool risponderà in questo modo:

“FUNC fprintf Last argument is constant string: OK”

Modificando l’istruzione in questo modo:

```
fprintf(stdout,buffer);
```

Ci apparirà il seguente messaggio:

SECURITY: fprintf call should have "%s" as argument 1



readlink()

readlink() inserisce in un buffer il path a cui punta un link simbolico.

La stringa restituita non è NULL TERMINATED.

La funzione ritorna il numero di caratteri letti.

Un uso inappropriato può dar luogo a buffer overflow.

L'uso di readlink() è segnalato da its4 e flawfinder.



readlink()

rsync gestisce correttamente le chiamate a readlink(),
segue un esempio di uso di tale funzione:

```
case TYPE_SYMLINK:
```

```
    if ((len = readlink(cmpbuf, lnk, MAXPATHLEN-1)) <=
0)
```

```
        continue;
```

```
    lnk[len] = '\0';
```

```
    if (strcmp(lnk, F_SYMLINK(file)) != 0)
```

```
        continue;
```

```
    break;
```



getenv()

Un'altra funzione il cui utilizzo è segnalato sia da its4 che da flawfinder è `getenv()`, che permette di accedere al valore di una variabile d'ambiente.

Problema: le variabili d'ambiente sono in generale inaffidabili:

- il valore può avere una qualsiasi lunghezza
- il valore è una semplice stringa, quindi non ha sintassi



getenv()

Esempio di uso di getenv() all'interno di rsync:

```
if (am_server) {
    strcpy(addr_buf, "0.0.0.0", sizeof addr_buf);

    if ((ssh_info = getenv("SSH_CONNECTION")) != NULL
        || (ssh_info = getenv("SSH_CLIENT")) != NULL
        || (ssh_info = getenv("SSH2_CLIENT")) != NULL) {

        strcpy(addr_buf, ssh_info, sizeof addr_buf);

        /* Truncate the value to just the IP address. */
        if ((p = strchr(addr_buf, ' ')) != NULL)
            *p = '\0';
    }
}
```



Rsync VS buffer overflow

Occorre evitare fenomeni di buffer overflow

rsync, ove occorre copiare/concatenare una stringa in un buffer, come ad esempio l'output di `getenv()`, usa le funzioni non standard (OpenBSD 1999):

```
size_t strlcpy(char *dst, const char *src, size_t size);
```

```
size_t strlcat(char *dst, const char *src, size_t size);
```



`strncpy()` e `strncat()`

Evitano gli errori di programmazione possibili con l'uso di `strncpy()` e `strncat()` (vogliono la lunghezza della stringa da copiare < della lunghezza del buffer di destinazione per garantire NULL termination)

Sono più efficienti di quest'ultime (no zero-fill di dest)

Richiedono nel parametro `size` la lunghezza della destinazione, spesso calcolabile a compile-time mediante l'operatore `sizeof`.

Ritornano la lunghezza della stringa creata, e garantiscono SEMPRE la terminazione con NULL (eventuale troncamento).

Quindi i troncamenti sono facilmente rilevabili, senza usare `strlen()` per determinare la lunghezza della stringa creata.



fopen()

I tools notificano inoltre che nel codice si fa uso della funzione `fopen()`, anch'essa a rischio di attacchi di tipo TOCTOU.

In `rsync` tale funzione è utilizzata per:

- accedere in sola lettura al file di configurazione del demone (editabile solo da un super utente)
- accedere in sola lettura ai file da analizzare per eseguire i trasferimenti (ne parleremo tra poco)



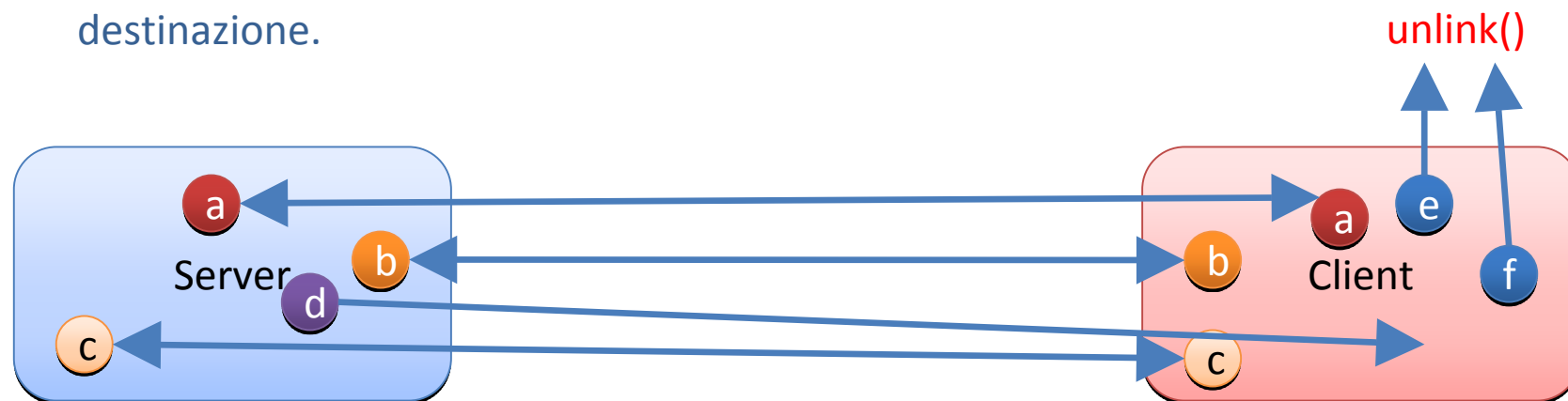
unlink()

La funzione unlink() permette di rimuovere un hard-link ad un inode esistente.

Se dopo la chiamata, il numero di hard-links è pari a zero, l'inode viene rimosso.

Supponiamo di voler sincronizzare due directory tra client e server...

I file non presenti nella cartella con precedenza possono essere rimossi dalla cartella destinazione.





unlink()

L'uso di unlink() è ovviamente segnalato dai tools.

La funzione accetta come unico parametro una stringa rappresentante il path+nome del file da eliminare.

La unlink() invocata su un link simbolico, cancella il link simbolico, non il file da esso puntato.

L'attacco consiste nel modificare tale path in un suo punto intermedio, inserendo un opportuno link simbolico al posto di un'omonima directory, e lasciando il resto del path inalterato.



Politica di rsync – Il server

rsync adotta alcune semplici regole per evitare attacchi di questo genere:

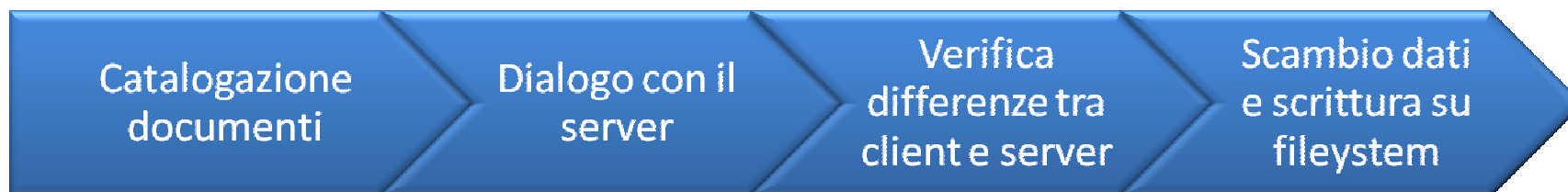
- la directory di sincronizzazione sul server costituisce un'area protetta dalla quale è vietato uscire
- tutti i path ai file in essa contenuti sono interpretati come relativi ad essa
- non è possibile percorrere link simbolici al suo interno che puntino ad un oggetto ad un livello superiore



Politica di rsync – Il client

Sul client invece rsync opera nel seguente modo:

- È possibile uscire mediante link simbolici dalla directory di sincronizzazione (ma solo se si abilita l'opzione --unsafe-links)
- In caso contrario, tutti i link simbolici puntanti all'esterno sono ignorati





Politica di rsync – Prove su campo

Per testare la sicurezza di rsync rispetto ad attacchi di tipo TOCTOU abbiamo forzato il verificarsi di alcune race conditions

Basta inserire una chiamata bloccante nel posto giusto e cambiare le carte in tavola...

Quindi proseguire con l'esecuzione del programma e valutare i risultati ottenuti.





Politica di rsync – Prove su campo

Le nostre prove si basano sulle seguenti ipotesi:

- I file contenuti nelle directory di sincronizzazione sono accessibili in scrittura da un determinato gruppo di utenti
- Un utente del gruppo è il responsabile dell'esecuzione di rsync, e proprietario dei file da trasferire
- Quest'ultimo può anche essere un utente privilegiato, tutti gli altri utenti del gruppo sono limitati



Politica di rsync – Prove su campo

Nonostante rsync lavori implicitamente con path relativi, non è impossibile creare qualche problema, dimostrando che questa non è una buona configurazione di utilizzo.

In particolare, siamo riusciti ad uscire dalla directory di sincronizzazione senza abilitare alcuna opzione sui link simbolici.



Politica di rsync – Prove su campo

Sia “Source” la directory di sincronizzazione, sulla quale possiamo scrivere.

Inseriamo in essa una sotto-directory, sia essa “Source/A/”, contenente un file vuoto, con lo stesso nome di un noto file nel sistema. Sia esso “target” per semplicità.

Eventualmente possiamo nascondere il tutto.

Ora rimane da sfruttare una opportuna race condition tra la lettura di rsync e il successivo aggiornamento...



Politica di rsync – Prove su campo

Il risultato sarà il seguente:

- 1) “A/target” non esiste sul server, quindi deve essere rimosso dal client
- 2) Il client esegue unlink(“A/target”) (assume sempre “Source” come directory corrente)
- 3) Il file è rimosso e si prosegue con la sincronizzazione



Politica di rsync – Prove su campo

Ma `unlink()` segue eventuali link simbolici nel path ricevuto come parametro...

Se un istante prima si riesce a rinominare la cartella “A” con un nome diverso, e ad inserire un link simbolico “A” puntante ad una qualsiasi cartella sulla macchina contenente un file omonimo, rsync esce dalla directory di sincronizzazione ed elimina il file sbagliato.



Politica di rsync – Prove su campo

Ciò significa che:

- chi esegue rsync non può assumere che il programma non esca mai dalla directory di sincronizzazione
- l'unico modo per evitare ciò, è impedire l'accesso in scrittura sulla directory agli altri utenti
- eseguire rsync con privilegi di amministratore è una pessima idea



Politica di rsync – Prove su campo

Per quanto riguarda le scritture di nuovi file a destinazione, il comportamento di rsync è il seguente:

- 1) creare un file temporaneo mediante `mkstemp()` nella directory destinazione, inizialmente editabile solo dal legittimo proprietario
- 2) usando il File Descriptor ritornato da `mkstemp()`, scrivere in esso i dati da copiare dalla sorgente
- 3) rinominare il file mediante `rename()` con il nome del file omonimo sulla sorgente, e settare i permessi del file originale



Politica di rsync – Prove su campo

1) Passo sicuro:

*“We initially set the perms **without the setuid/setgid bits or group access** to ensure that there is no race condition. They will be correctly updated after the right owner and group info is set.”*

```
int fd = mkstemp(template);

if (fd == -1)
    return -1;

if (fchmod(fd, perms) != 0) {

    close(fd);
    unlink(template);

    return -1;
}
return fd;
```



Politica di rsync – Prove su campo

2) Passo sicuro:

il File Descriptor utilizzato è ritornato da `mkstemp()`, progettata appositamente per evitare race conditions tra la creazione del file temporaneo e la sua apertura, quindi da essa deriva la sicurezza di questo passo.

3) Passo che merita qualche osservazione:

`rename()` può essere soggetta ad attacchi di tipo TOCTOU...



Politica di rsync – Prove su campo

Abbiamo creato le due seguenti situazioni anomale:

- a) creazione di un nuovo file con il nome aspettato prima dell'esecuzione di `mkstemp()`: il file viene sovrascritto con la nuova copia.





Politica di rsync – Prove su campo

b) creazione di un nuovo file con il nome aspettato dopo l'esecuzione di `mkstemp()`: la `rename()` non fallisce ma NON sovrascrive il file da noi creato.

Abbiamo perso tutti i dati del file originale.





Politica di rsync – Prove su campo

La differenza sta nel fatto che il file creato nel passo b) ha una data di creazione successiva a quella del file temporaneo creato con `mkstemp()`

Funziona anche con un semplice eseguibile che esegue `rename()` di un qualsiasi file.

Una nota: se nel sistema non c'è `mkstemp()`, `rsync` passa ad usare `mktemp()`, molto meno sicura della prima.



Politica di rsync – Opzione --update

`“rsync --verbose --update --append localhost::Repository/* ./Source/”`

Permette l’aggiornamento di ogni file destinazione in loco, senza cioè appoggiarsi su un file temporaneo. Di default è disabilitato.

```
if (inplace) {
    fd2 = do_open(fname, O_WRONLY|O_CREAT, 0600);
    if (fd2 == -1) {
        rsyserr(FERROR_XFER, errno, "open %s failed",
            full_fname(fname));
    }
}
```



Politica di rsync – Opzione --update

Dal manuale di rsync:

“WARNING: you should not use this option to update file that are being accessed by others, so be careful when choosing to use this for a copy.”



Politica di rsync – Opzione --update

Supponiamo la seguente situazione tipo:

- Un capo-ufficio ha il diritto di eseguire rsync per collegarsi alla sede centrale
- Gli altri impiegati dell'ufficio possono solo accedere in locale ai documenti scaricati dal capo in una cartella condivisa
- Il capo-ufficio ha privilegi di amministratore



Politica di rsync – Opzione --update

Supponiamo che sia stata rilasciata una nuova versione per un documento, e che gli impiegati ne siano a conoscenza.

Un malintenzionato nel gruppo potrebbe effettuare un attacco di tipo TOCTOU:

- sostituisce la vecchia versione con un link simbolico puntante ad un file di sistema
- attende l'aggiornamento del file per opera del capo



Politica di rsync – Opzione --update

L'attacco ha successo:

- 1) rsync apre mediante `open()` il file da aggiornare
- 2) `open()` segue il path contenuto nel link simbolico
- 3) i dati sono scritti nel file da esso puntato

Motivo: anche qui rsync non controlla che il file su cui si scrive sia lo stesso di partenza.



Politica di rsync – Opzione --update

Problema di fondo: è stato utilizzato rsync per aggiornare il contenuto di un file accessibile in scrittura da altri utenti.

Soluzione: usare --update su file accessibili in scrittura solo dal proprietario.

Anche se l'utente è avvisato del problema, rsync effettivamente non dispone degli opportuni controlli per evitare l'inconveniente.



Politica di rsync – Opzione --update

Usando un utente limitato, siamo quindi riusciti a modificare il contenuto di un file il cui accesso in scrittura era limitato al “capo-ufficio”.

In esso abbiamo trovato la porzione di bytes in cui i due file da sincronizzare differivano.



Problematiche correlate

In fase di avvio, l'istanza di rsync eseguita sul client deve raggiungere la directory di lavoro.

A tal scopo viene utilizzata la funzione `chdir()`, segnalata da `its4` ma non da `flawfinder` e `PScan`.

Cosa accade se prima che sia invocata `chdir()` si sostituisce la directory con un link simbolico omonimo?



Problematiche correlate

Il risultato è ovviamente lo stesso che si avrebbe se tale operazione fosse fatta prima di avviare rsync (chdir() segue i link simbolici).

Non è possibile cioè per rsync (come per qualsiasi altro programma) effettuare controlli per evitare questa situazione.

Di conseguenza la directory di lavoro deve essere opportunamente protetta.



Osservazioni

Dalle prove eseguite possiamo constatare che:

- rsync non esegue controlli di coerenza tra la fase di analisi dei dati e la successiva fase di scrittura
- più i tempi di attesa per i trasferimenti sono lunghi, più è alta la probabilità che qualcosa possa cambiare su filesystem nel frattempo



Bugs di altra natura

Sono stati trovati inoltre i seguenti bugs:

1. referenziamento di un link simbolico che punta ad una directory che si trova ad un livello superiore o uguale alla directory in cui si trova il link
2. un file il cui nome inizia con il “-” viene interpretato come opzione



Funzione unsafe_symlink (1/2)

```
int unsafe_symlink(const char *dest, const char *src)
{
    const char *name, *slash;
    int depth = 0;

    /* all absolute and null symlinks are unsafe */
    if (!dest || !*dest || *dest == '/')
        return 1;

    /* find out what our safety margin is */
    for (name = src; (slash = strchr(name, '/')) != 0; name = slash+1) {
        /* ".." segment starts the count over. "." segment is ignored. */
        if (*name == '.' && (name[1] == '/' || (name[1] == '.' && name[2] == '/'))) {
            if (name[1] == '.')
                depth = 0;
        } else
            depth++;
        while (slash[1] == '/') slash++; /* just in case src isn't clean */
    }
    if (*name == '.' && name[1] == '.' && name[2] == '\0')
        depth = 0;
}
```



Funzione unsafe_symlink (2/2)

```
for (name = dest; (slash = strchr(name, '/')) != 0; name = slash+1) {  
  
    if (*name == '.' && (name[1] == '/' || (name[1] == '.' && name[2] == '/'))) {  
        if (name[1] == '.') {  
            /* if at any point we go outside the current directory  
            then stop - it is unsafe */  
            if (--depth < 0)  
                return 1;  
        }  
    } else  
        depth++;  
    while (slash[1] == '/') slash++;  
}  
  
if (*name == '.' && name[1] == '.' && name[2] == '\0')  
    depth--;  
  
return depth < 0;  
}
```



Bug 1 (1/3)

Basta creare un link simbolico nel seguente modo:

In `-s “../”` link

Problemi riscontrati:

- il programma va in loop nella generazione di directory
- viene fermato solo dai limiti del sistema operativo che non permette di creare più di 40 livelli



Bug 1 (3/3)

- sono stati effettuati correttamente tutti i controlli relativi alla copia di link simbolici
- nonostante ciò, con l'opzione `-r` riusciamo ugualmente a percorrerli
- spreco di banda e spazio, computazioni inutili che potrebbero essere evitate...



Bug 2 (1/2)

Basta creare un file che si chiama ad esempio nel seguente modo:

“-e”

Problemi:

- Il programma interpreta il file come un'opzione...



Bug 2 (2/2)

- L'opzione `-e` indica il processo di fare fork ed `execvp` di una shell indicata da un utente
- Sfruttando questa caratteristica si potrebbero lanciare eseguibili a propria discrezione
- Come `-e` potrebbe essere aggiunta qualsiasi altra opzione prevista da `rsync`
- Manca nella sintassi di comando una separazione tra i flag d'esecuzione e i file su cui lavorare
- Nota: l'attacco funziona solo se si è all'interno della directory di sincronizzazione



Conclusioni

Possiamo concludere quindi che rsync-3.0.5:

- + esegue un'accurata validazione dell'input ed un'accurata gestione delle strutture dati interne
- + fa uso di funzioni non standard per aumentare il livello di protezione rispetto a fenomeni di buffer overflow
- non esegue gli opportuni controlli di coerenza su file system
- garantisce l'integrità dei dati solo se accompagnato da un'opportuna configurazione dell'ambiente di lavoro