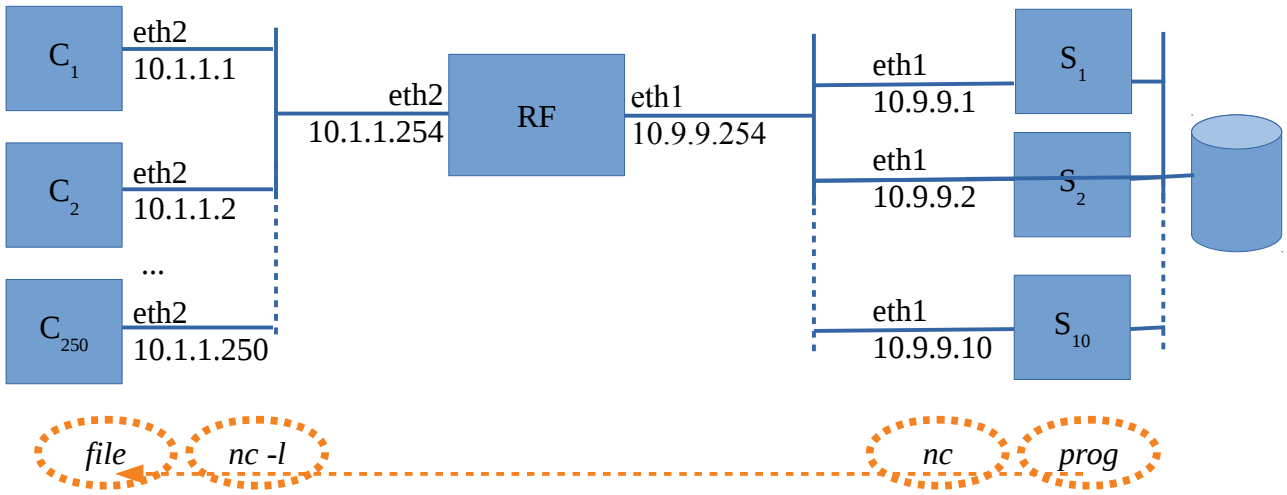
Laboratorio di Amministrazione di Sistemi T

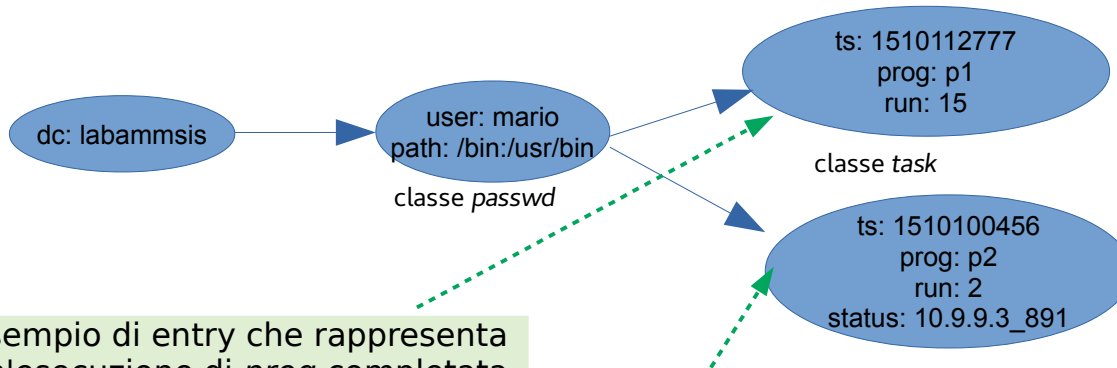
Prova pratica del 20 dicembre 2018

Descrizione generale del problema



Il sistema di cloud computing rappresentato in figura prevede un insieme di 250 Client, 10 Server che condividono un filesystem, e un Router-Firewall (RF) tra le rispettive reti. Il RF media le richieste inviate dai client per far eseguire task ai server.

La base di utenti autorizzata all'uso degli script sui server è registrata sulla directory LDAP ospitata da RF. Le entry servono sia a rappresentare le identità degli utenti che a gestire le richieste da essi effettuate, come nell'esempio di struttura qui riportato:



Esempio di entry che rappresenta un'esecuzione di *prog* completata all'istante *ts* dopo una durata in secondi pari a *run*

Esempio di entry che rappresenta una richiesta di eseguire *prog*, inserita in LDAP all'istante *ts* e assegnata al server 10.9.9.3, dove è in esecuzione da run secondi con PID 891

File da consegnare

(tra parentesi quadre la collocazione e il punteggio, non fanno parte del nome)

cloud.schema.ldif [router - 5] – Definire i tipi di attributo *ts, run* (intero), *user, pass, prog, status* (stringhe) e le classi *passwd* e *task* che li utilizzino come in figura.
Nota generale:

- le entry di classe *passwd* conservano nome (*user*) di un utente e percorso ad esso consentito per l'esecuzione di programmi (*path*)
- le entry di classe *task* hanno sempre gli attributi *ts, run* e *prog*;
  - le richieste in corso si riconoscono dalla presenza dell'attributo *status*, che può assumere come valori la stringa **`denied`** o una stringa che rappresenta l'indirizzo IP del server su cui il programma è correntemente in esecuzione;
  - le esecuzioni di task già completate sono prive di tale attributo.

go.sh [client - 22] – Lo script accetta come parametri un nome relativo di programma e un nome di file, ed è usato per chiedere a RF di scegliere un server ed eseguire su di esso il programma.
L'output del programma deve essere ricevuto attraverso una connessione ad hoc, e scritto sul file passato come parametro; per fare questo, *go.sh* si avvale di **`nc`** lanciato in ascolto su di una porta TCP scelta a caso (ma verificando che sia libera!) approssimativamente nell'intervallo tra 20000 e 50000.
La richiesta viene fatta inviando a RF via rsyslog IP del client, timestamp, nome utente, programma e porta di ricezione.
Lo script si pone successivamente in attesa che appaia la entry che rappresenta la richiesta, e continua ad osservarne il contenuto ogni secondo

- se *status* diventa **`denied`**, lo script stampa questo esito e termina;
- se scompare l'attributo *status*, lo script stampa il tempo di esecuzione e termina.

In ogni caso, alla terminazione lo script deve provvedere alla terminazione anche di **`nc`**.
Indicare nei commenti come configurare rsyslog di client e RF in modo che i messaggi inviati dal client vengano scritti su `/var/log/go.log` di RF.

run.sh [router - 26] – Lo script configura il packet filter per bloccare tutto il traffico non essenziale ai vari script, poi legge continuamente il file `/var/log/go.log` e per ogni nuova riga:

- cerca con SNMP il server, tra i 10, che ha il più basso carico medio negli ultimi 5 minuti
- configura il proprio packet filter per consentire le connessioni TCP tra tale server e il client che ha inviato il messaggio, sulla porta indicata nel messaggio stesso
- ricava da LDAP l'attributo **`path`** dell'utente
- configura cron per eseguire ogni minuto *monitor.sh* passandogli come parametri il server, il path, e tutti i dati estratti dal messaggio

Inserire nei commenti come configurare *snmpd* sui server per consentire la verifica.

monitor.sh [router - 20] – Se non esiste la entry di classe task relativa ai dati passati come parametri, lancia sul server via SSH *exec.sh*, passandogli come parametri il path, il programma, la porta e l'IP del client, e attende un risultato, che può essere **`denied`** o il PID del processo creato; crea la entry LDAP di classe task in modo corrispondente inizializzando *run* a 0.
Se la entry esiste, controlla via SSH se il processo è ancora in esecuzione, e in tal caso aggiorna la entry incrementando *run* del tempo trascorso .
Se il processo è terminato, rimuove l'attributo *status* dalla entry, e le regole iptables e la configurazione di cron inserite da *run.sh*.

exec.sh [server - 9] – Lo script esplora le directory del path e non appena trova il programma lo lancia in background in modo che invii con **`nc`** l'output alla porta del client specificati come parametri, restituendo il PID su standard output.
Se non trova il programma in nessuna directory del path, restituisce la stringa **`denied`**.