Pianificazione della Sicurezza Informatica

e Elementi di Informatica Forense

AA 2015/2016

FileSystemForensics

Obiettivo: Implementazione di una applicazione Android forense per l'acquisizione remota del file system di un device Android.

Studenti:

- Matteo Capparrucci
- Federico Corò

Docente:

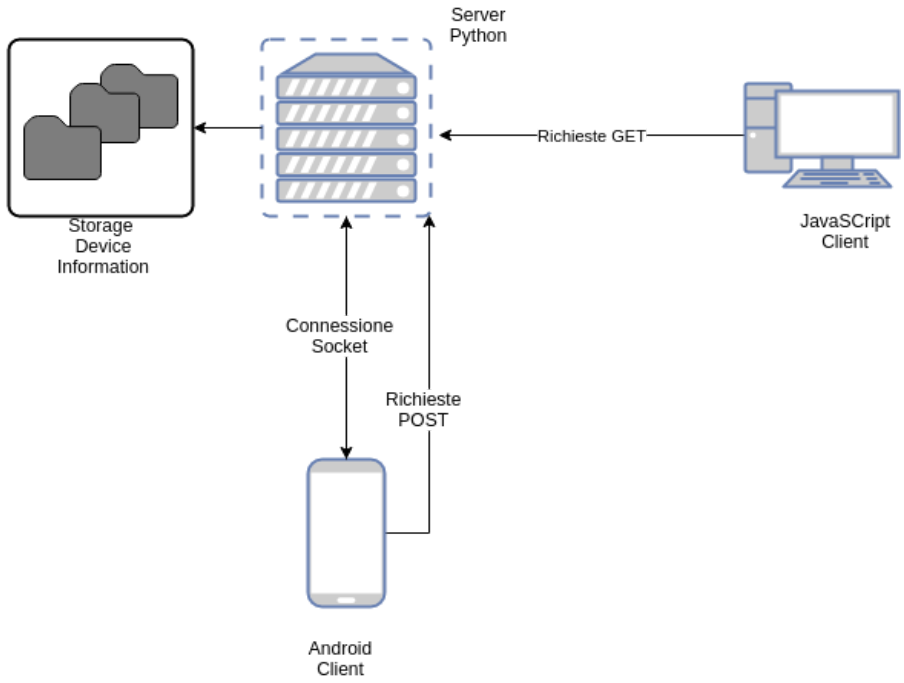
- Prof. Alfredo Miliani

Flusso di esecuzione

L'intera applicazione consiste in tre parti

- Server
- Client Android
- Client Browser

Si è deciso per questa suddivisione così da avere l'applicazione il più modulare possibile, infatti il Client Browser, scritto in linguaggio JavaScript, comunica con il server solo tramite richieste GET così da lasciare aperta la possibilità a futuri sviluppi di Client anche in linguaggi o modalità differenti.
Nel server vengono mantenuti uno storico dei device, viene infatti creata una cartella per ogni nuovo device che viene connesso e al suo interno verranno mantenute le mappe dei filesystem presi nel tempo e gli eventuali file recuperati dal device.
Lo schema seguente riassume un normale flusso di esecuzione dell'applicazione.



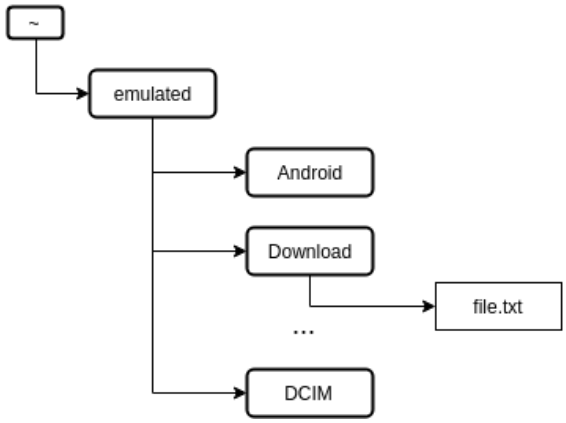
Nel Client JavaScript si è implementata l'emulazione di un terminale linux.
Al primo accesso viene chiesta l'autenticazione, si è poi in grado, tramite i comandi *list* e *choose* di scegliere quale device e in particolare quale file system scaricare.
Una volta ottenuto si può scorrere normalmente con i comandi *ls* e *cd*.
Nel caso in cui si voglia una nuova mappatura del filesystem è necessario solamente utilizzare il comando *takeAll*.
Verrà quindi inviata una richiesta al server che, tramite il socket connesso con il device scelto, invierà il comando richiesto. Nel device, come verrà spiegato anche nella sezione successiva riguardante il lato Android, verrà creato un Thread il cui compito è mappare tutte le cartelle e i file all'interno del device e scriverli in un Json che verrà poi inviato tramite chiamata POST al server rendendolo così accessibile al Client Browser.

File Json

Si è deciso di schematizzare l'albero del file system con un file json che conterrà

- Mappa dell'albero, cartelle e file comprensivi di informazioni supplementari come data di ultima modifica e peso in byte
- Informazioni riguardanti il device come: seriale, build, etc..
- Data di acquisizione del file system
- Hash del file system

L'albero del file system android, per quanto riguarda le cartelle accessibili all'utente, è gestito tramite una o due cartelle, una si chiama *emulated* e sarebbe la memoria interna al cellulare, l'altra sarà *extSdCard* se al cellulare è connessa una eventuale sdcard esterna.



Di seguito un esempio di Json:

```
{"sha1": "054fd15f30a738ccbf371b3fbb7a2b103d6057d1",
"BRAND": "samsung", "BUILDN": "JSS15J", "DEVICE": "m0", "MODEL": "GT-I9300",
"DATA": "1465665334", "SERIAL": "4df126df11735f37", "MANUFACTURER": "samsung"
"FileSystem":
[{"isDirectory": true, "lastModDate": "", "nome": "emulated"
   "sub":
       [{ "isDirectory": true, "nome": "Download", "lastModDate": "10/giu/2016 08:51:38",
            "sub": [{
              "Byte": 1713388, "lastModDate": "10/giu/2016 12:58:22",
              "sub": "", "isDirectory": false, "nome": "1234.jpg"
       }, ...
        ]
  }, ...
]}
```

Android - FileSystemForensics

La parte *mobile* del progetto è rappresentata da una piccola applicazione Android chiamata “FileSystemForensics”. Lo scopo di questa applicazione è quello di mantenere un collegamento, attraverso la rete internet, tra il device mobile e il server Python descritto in precedenza. All’invio di un comando da remoto, originato dal server, l’applicazione sarà in grado di fare l’upload tramite chiamata POST del file json contenente la mappatura del file system appena generato o nel caso in cui la richiesta sia di tipo *get-path* di eseguire l’upload del file dal device al server.

All’interno dell’applicazione sono presenti, oltre ai necessari file di configurazione Android, tre classi java il cui funzionamento verrà descritto in seguito:

- MainActivity.java
- MyIntentService
- Utility

MainActivity

La main activity dell’applicazione ha solo il compito di lanciare il servizio, senza binding, MyIntenteService e poi gestire la chiusura dell’app stessa. L’applicazione inoltre non sarà visibile nel menù del device.

MyIntenteService

MyIntenteService è la parte fondamentale dell’applicazione. Al suo interno infatti viene definito un loop che, in caso di avvenuta connesione tramite socket con il server, rimarra in uno stato di attesa (bloccante) con lo stesso.
A questo punto eventuali comunicazione avvenute tramite socket saranno opportunamente gestite per distinguere le tre tipologie di messaggio implementate:

- ack
- takeAll
- get : path

I messaggi di ack verranno utilizzati per il controllo della bontà della connessione tra le due parti. Nel caso invece di takeAll verrà immediatamente invocato il metodo **doTakeAll** il quale non farà altro che avviare un nuovo thread di esecuzoìone il quale dovrà gestire l’esecuzione della funzione takeAll vera e propria e poi inviare tramite chiamata rest POST il risultato di takeAll.

Il metodo **takeAll** avrà il compito di ottenere il path a tutte gli elementi di memoria del device: interna attraverso la directory *emulated* e *esterne* nel caso di eventuali schede sd, tramite il metodo **getExternalMounts**. Individuati i path corretti questi saranno quindi utilizzati come

argomento per il metodo *lsRecursive* il cui risultato, ovvero il file system opportunamente codificato, sarà poi incapsulato in un oggetto di tipo JSON assieme agli altri dati di interesse e restituito.

lsRecursive è invece il metodo centrale dell'intera applicazione, questo infatti preso in input un file di qualsiasi tipo per prima cosa controlla se questo è una directory e in caso affermativo crea un nuovo file JSON contenente i dati di interesse di ogni file (nome, dimensione, data di ultima modifica), quindi controlla se ci sono file che sono a loro volta delle directory e in caso affermativo invoca nuove chiamate di lsRecursive per ogni directory trovata salvando il risultato di queste ulteriori computazioni nel json restituito.

Nel caso in cui il messaggio ottenuto dal server sia di tipo *get-file* la procedura invoca il metodo **makeZip** il quale a sua volta per prima cosa generà un nuovo thread di esecuzione al quale affida la ricerca del file indicato nella richiesta get e, in caso di riscontro positivo, invoca il metodo **zip** tramite il quale il file viene incapsulato in un archivio di tipo .zip e convertito in array di byte. L'array di byte così generato verrà poi a sua volta inviato tramite chiamata rest POST assieme agli header necessari per la generazione dei log da parte del server.

Utility

L'ultima classe è Utility la quale al suo interno contiene due soli metodi statici, **zip** e **unzip** i quali tramite la libreria java.util.zip.* consentono di generare archivi rar e gestirli efficacemente.