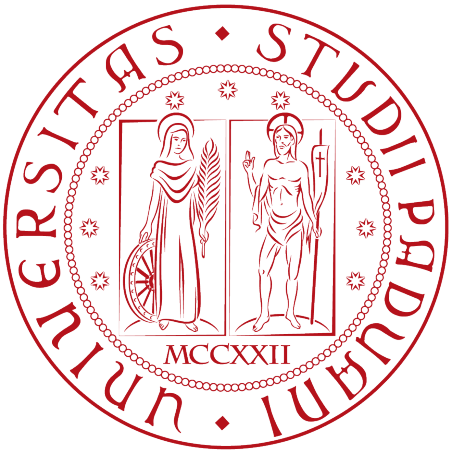
Università di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Project Projector: Never Mind the Bullets

Tesi di Laurea Triennale

Supervisori
Prof. Mauro Conti
Prof. Claudio Enrico Palazzi
Co-supervisore
Dott. Daniele Lain

Laureando
Eugen Saraci

Anno Accademico 2016-2017

Sommario

Questo elaborato presenta il lavoro svolto dal laureando Eugen Saraci presso il Dipartimento di Matematica "Tullio Levi-Civita" sotto la supervisione del Professor Mauro Conti e del Dott. Daniele Lain.

Il lavoro è stato preceduto da una fase di studio delle tecnologie e della letteratura riguardo i *side-channel attacks* che sono argomento centrale di questo elaborato. Partendo da una analisi teorica del problema si giunge all'analisi tecnica degli applicativi prodotti, per poi finire con un resoconto ed i risultati ottenuti.

"È certo che un uomo può fare ciò che vuole, ma non può volere ciò che vuole."

— Arthur Schopenhauer

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Mauro Conti, relatore della mia tesi, e all' Assegnista di Ricerca Daniele Lain per l'aiuto ed il sostegno fornitomi durante tutto l'ultimo anno universitario e la stesura di questo lavoro.

Desidero ringraziare la mia famiglia, per il supporto datomi in questi anni che mi ha permesso di arrivare dove sono.

Un grande ringraziamento agli amici universitari e non, che sono stati fonte di stimolo, sostegno e gioia in ogni momento.

Un ultimo ringraziamento è dedicato ai volontari degli esperimenti svolti durante lo stage.

Padova, Luglio 2017 Eugen Saraci

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo elaborato verranno presentati il *Projector Attack* e *Project Projector*. Il primo è un attacco innovativo in ambito *side-channel attacks*, il secondo è l'insieme di strumenti (*toolset*) correlato all'attacco, sviluppato durante il periodo di stage presso il Dipartimento di Matematica "Tullio Levi-Civita".

1.1 Projector Attack

Il *Projector Attack* è un *timing attack* che sfrutta la distanza di tempo tra la pressione di due tasti della tastiera, e sulla base di ciò tenta di risalire alle coppie di caratteri che hanno generato tali dati.

Lo scenario di applicazione principale dell'attacco prevede che la vittima abbia collegato il proprio laptop ad un proiettore; prima che la vittima digiti una password su una pagina di login, l'attaccante registra (con uno smartphone) tutto ciò che viene trasmesso dal proiettore; quando la vittima ha finito di digitare la password, l'attaccante interrompe la registrazione. Un'attenta analisi a posteriori del video permetterà all'attaccante di conoscere in primo luogo la lunghezza della password usata dalla vittima, in secondo luogo l'attaccante potrà analizzare *frame by frame* il video per indentificare in quale instante di tempo compaiono i singoli pallini (•)[1] usati generalmente per mascherare la password. Con tali dati l'attaccante potrà calcolare il tempo trascorso tra i caratteri digitati dalla vittima. Una coppia di caratteri verrà chiamata *digrafo*, il numero di digrafi in una parola corrsiponde al numero di caratteri della parola meno uno, ad esempio la parola "Projector" ha 9 caratteri ed 8 digrafi (Pr, ro, oj, je, ec, ct, to, or). La parte successiva dell'attacco prevede l'utilizzo di algoritmi di machine learning, in particolare l'attaccante utilizzerà un classificatore (Random Forest ad esempio), al quale verranno forniti come dati di training dei digrafi ed i relativi tempi (ottenuti ad esempio da *dataset* pubblici [2, 5, 8, 11]). Finita la fase di training, l'attaccante potrà fornire al classificatore i tempi ricavati dal video registrato nella prima fase; il classificatore restituirà per ogni tempo in input una lista di digrafi per i quali ha maggiore *confidence*. L'idea è che la password della vittima sia una combinazione dei digrafi forniti in output dal classificatore. I risultati dell'attacco e la percentuale di successo saranno discussi a fondo nel capitolo 5.

[1] Il carattere usato per mascherare la password non è rilevante ai fini dell'attacco.

1.2 Project Projector

Project Projector è il nome dell'insieme di strumenti che rende possibile testare ed attuare il *Projector Attack*, tale insieme consiste in tre applicativi differenti e ben distinti:

- ***P-Logger***: è uno strumento per la raccolta dati utilizzato in fase di sperimentazione. Consiste in una pagina HTML, degli script in JavaScript, e un Back-End in Python. È una replica della pagina di login di Google (`https://accounts.google.com/`); gli script in JavaScript fungono da *keylogger* mentre il Back-End in Python (Flask) viene utilizzato principalmente per il salvataggio dei dati. La pagina HTML è la pagina che verrà sottoposta ai volontari partecipanti all'esperimento;

- ***P-Extractor***: è uno strumento di automazione scritto in Python per la raccolta di dati da video, utilizza le librerie di *Computer Vision* OpenCV, lo scopo è quello di minimizzare il tempo speso ad estrarre manualmente i dati da video;

- ***P-Crack***: è l'applicativo principale, è scritto in Python ed ha tre modalità di funzionamento tra cui quella in cui viene attuato il *Projector Attack*.

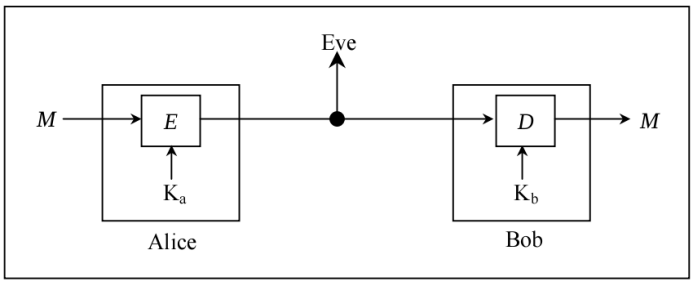
1.3 Opere correlate

1.3.1 Attacchi side-channel

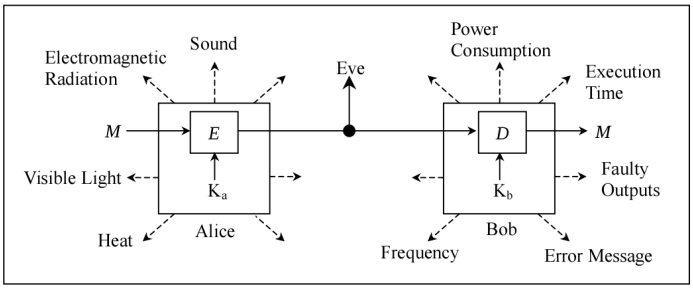
Il *Projector Attack* è un *timing attack* e come tale rientra in una categoria di attacchi detta *side-channel attacks*. I *side-channel attacks* sono per definizione attacchi che sfruttano l'implementazione fisica di un sistema crittografico, con ciò si intende che vengono sfruttate caratteristiche quali ad esempio: il tempo, il suono, il consumo di energia ed eventuali emanazioni elettromagnetiche. I *timing attack* sono attacchi che sfruttano il lasso di tempo impiegato da un unità per svolgere un operazione.

Si immagini un algoritmo crittografico e si assuma che tale algoritmo sia totalmente sicuro dal punto di vista logico; Se il tempo di esecuzione di una data operazione è dipendente da una chiave segreta (o dalla sua lunghezza), effettuando attente misure durante l'esecuzione dell'algoritmo sarà facile per un attaccante ottenere informazioni sulla chiave segreta utilizzata; banalmente l'attaccante potrebbe essere in grado di intuire la lunghezza della chiave basandosi sul tempo di esecuzione dell'operazione. L'esempio, seppur semplificato e banale, indica come un algoritmo sicuro dal punto di vista logico-matematico non possa essere considerato necessariamente sicuro da altri punti di vista, come quelli sfruttati dai *side-channel attacks*.

Parte fondamentale di un *side-channel attack* è spesso la mancanza di attenzione a dettagli non coinvolti direttamente nel sistema crittografico in sviluppo. Nel *Projector Attack*, l'utente non è conscio del rischio che corre nel mostrare il proprio schermo nel momento in cui viene digitata una password, i pallini (•) che mascherano i caratteri della password sono efficaci e conformi nel fare ciò per cui sono stati pensati, ma sono i "canali laterali" a rappresentare la vulnerabilità. In Figura 1.3.1 si notano le differenze fra un sistema crittografico visto in un'ottica tradizionale e uno visto in ottica *side-channel attacks*. Le immagini sono tratte da [13].



(a) Sistema crittografico in ottica tradizionale.



(b) Sistema crittografico in ottica *side-channel attacks*.

Figura 1.1: Un sistema crittografico visto da due punti di vista differenti: il normale funzionamento, e i possibili side-channel attacks.

1.3.2 Letteratura

Il primo *side-channel attack* si ritiene essere stato attuato nel 1965 da parte del MI5[2] per intercettare le conversazioni dell'ambasciata Egiziana a Londra. L' MI5 stava provando a "crackare" il sistema crittografico usato dall'ambasciata, ma ciò richiedeva risorse e potenza di calcolo non disponibili al tempo. Peter Wright, al tempo scienziato per il GCHQ[3], propose di mettere un microfono vicino alla macchina che controllava i rotori del sistema crittografico egiziano. Il suono dei rotori diede all'MI5 una quantità di dati sufficiente per poter portare a termine il "cracking" del sistema, ciò permise di continuare a spiare per anni le comunicazioni dell'ambasciata Egiziana [12].

Oltre a questo episodio, i primi studi rilevanti sui *side-channel attacks* sono ad opera di Paul C. Kocher che nel 1996 introdusse i *timing attacks*. In [6] Kocher attacca con successo l'implementazione di RSA studiando il tempo impiegato dall'algoritmo per svolgere alcuni calcoli. Nel 2001 Song et al. dimostrano in [9] come l'analisi del traffico di sessioni SSH interattive riveli informazioni su dove la password si trovi all'interno del traffico criptato e su come la distanza di tempo di arrivo dei pacchetti possa aiutare a risalire ai tasti digitati. Una contromisura a questo tipo di attacchi è rendere il tempo di esecuzione indipendente dalla dimensione dell'input, oppure utilizzare *noise injection*, ovvero iniettare rumore all'interno dei dati per renderne difficoltosa la "traduzione".

Un'altra categoria di *side-channel attacks* sono gli *acoustic attack*, ovvero gli attacchi che sfruttano le informazioni audio. L'aneddoto di Peter Wright può essere considerato il primo *acoustic attack* della storia. Shamir et al. nel 2004 in [10] e nel 2013 in [4]

[2]Ente per la sicurezza e il controspionaggio del Regno Unito.
[3]Quartier generale del governo per le comunicazioni.

sono riusciti a risalire ad una chiave RSA di 4096 bit solamente ascoltando i suoni prodotti dalle componenti elettriche interne di un computer. Un altro *acoustic attack* molto interessante è il *Dont Skype & Type!* [3] di Compagno et al. dove è dimostrato che intercettando i suoni della tastiera in una chiamata Skype si possa risalire ai tasti digitati dall'interlocutore.

Anche le onde elettromagnetiche possono essere sfruttate nei *side-channel attacks*. Uno dei documenti più importanti in questo ambito è *TEMPEST*[1]. Scritto dalla NSA[4], *TEMPEST* documenta metodi e tecniche per sfruttare le *compromising emanations*, ovvero tutte quelle emissioni involontarie di onde elettromagnetiche (e non solo) da parte di dispositivi elettronici.

1.4 Tecnologie utilizzate

Python. *Project Projector* è stato sviluppato principalmente utilizzando Python 2.7; tale linguaggio è largamente utilizzato per lo sviluppo di applicazioni in cui è richiesto calcolo scientifico e rapidità di sviluppo, Python infatti offre strutture dati di alto livello facilmente utilizzabili e comprensibili. Il codice ha una sintassi semplice e risulta facile da imparare e da capire grazie alla sua espressività.

Figura 1.2: Logo del linguaggio Python.

NumPy. È un'estensione open source di Python nata del 2005; aggiunge supporto a vettori (e matrici) multidimensionali e di grandi dimensioni oltre ad una moltitudine di funzioni intuitive per gestirli. È spesso usato in applicazioni che richiedono l'uso di calcolo scientifico in quanto fornisce dei miglioramenti importanti rispetto a ciò che Python fornisce di default.

Figura 1.3: Logo di NumPy.

Matplotlib. È una libreria open source per la generazione di grafici 2D; viene utilizzata durante tutto lo sviluppo di *Project Projector* per generare i grafici dei risultati ottenuti durante gli esperimenti.

Figura 1.4: Logo di Matplotlib.

SciKit-learn. È una libreria open source di apprendimento automatico (o *machine learning*) predisposta per lavorare con NumPy. Fornisce algoritmi di vario tipo, in particolare in *Project Projector* viene usato un algoritmo di classificazione chiamato Random Forest.

[4]National Security Agency, agenzia per la sicurezza nazionale statunitense.

Figura 1.5: Logo di SciKit-learn.

OpenCV. È una libreria open source di visione computazionale (o *computer vision*). Viene utilizzata in *Project Projector* durante lo sviluppo di *P-Extractor*.

Figura 1.6: Logo di OpenCV.

1.5 Organizzazione del testo

Il secondo capitolo descrive la parte iniziale dell'esperimento, quella in cui è stato sviluppato *P-Logger* e gli esperimenti effettuati grazie ad esso.

Il terzo capitolo approfondisce lo sviluppo *P-Extractor*, l'applicativo che automatizza l'estrazione dei dati da video.

Il quarto capitolo descrive *P-Crack*, il modello utilizzato per l'attacco e i risultati di alcuni test preliminari.

Il quinto capitolo approfondisce i risultati dell'esperimento.

Il sesto capitolo descrive gli obiettivi raggiunti, le conoscenze acquisite ed una valutazione personale dello stage.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Esperimenti preliminari

Nel seguente capitolo si analizza lo sviluppo di P-Logger, un tool che simula graficamente la pagina di login di Google (`https://accounts.google.com/`) e si occupa di raccogliere dati relativi ai tasti premuti sulla pagina. P-Logger è stato utilizzato durante gli esperimenti svolti presso il Dipartimento di Matematica "Tullio Levi-Civita". Ai volontari è stata sottoposta una pagina di login in cui è richiesto di digitare più password, i dati relativi ai tasti premuti dai volontari sono stati raccolti e salvati per i successivi test. All'analisi vera e propria del tool precede una descrizione dettagliata dell'esperimento.

2.1 Design dell'esperimento

L'esperimento consiste nel far digitare delle password a dei volontari in una pagina web sotto il nostro controllo. E' richiesto che il *laptop* del volontario sia collegato ad un proiettore, e che le immagini proiettate dal proiettore vengano filmate con uno smartphone. L'obiettivo dell'esperimento è la raccolta di dati per verificare la fattibilità del *Projector Attack*, ed in caso di esito positivo di valutarne l'efficacia.

2.1.1 Hardware utilizzato

L'*hardware* necessario per l'esperimento è composto principalmente da:

- **Laptop**: è il dispositivo utilizzato dal volontario per digitare le password fornitegli, è necessario che questo dispositivo sia personale, è importante a fini dell'esperimento che il volontario sia il più familiare possibile con la tastiera utilizzata;
- **Proiettore**: il laptop del volontario deve essere collegato ad un proiettore; in occasione dell'esperimento sono state reperite varie tipologie di adattatori in quanto non era conosciuto a priori il modello del laptop dei volontari;
- **Smartphone**[1]: nell'esperimento è stato usato un *Samsung Galaxy S5*, tuttavia l'unico requisito per l'esperimento è la capacità dello smartphone di registrare video con la funzione *slow-motion* in modo da ottenere più frame possibili per l'analisi successiva dei video;

[1] Lo smartphone rappresenta un oggetto comune e di facile utilizzo, l'attacco può essere attuato con un qualsiasi dispositivo in grado di registrare video.

- **Treppiede**: la registrazione delle immagini proiettate viene fatta appoggiando lo smartphone su un treppiede. L'utilizzo del treppiede non è strettamente necessario ai fini dell'esperimento o dell'attacco, ma facilita la successiva analisi dei video.

L'esperimento è stato svolto nell'aula 2BC60 in Torre Archimede, le luci dell'aula erano graduate al minimo e le tende erano posizionate a mezza altezza in modo da avere un'illuminazione tale da permettere una visione chiara delle immagini proiettate. Lo smartphone è stato posizionato centralmente rispetto all'immagine proiettata dal proiettore e ad un' altezza di circa 170 cm da terra. Ai volontari è stato permesso di sedersi in qualsiasi posizione purché rientrasse nei limiti del cavo del proiettore. Gli esperimenti sono stati svolti durante un singolo giorno dalle ore 10:00 fino alle ore 17:20 con una pausa dalle ore 12:30 fino alle 14:00.

2.1.2 I volontari

Sono stati selezionati 30 volontari fra studenti, ricercatori e dottorandi tutti interni al Dipartimento di Matematica. I volontari avevano diversi livelli di esperienza nella digitazione, ma fortunatamente nessuno di essi digitava utilizzando una sola mano o solo l'indice di una mano[2], questo ci ha permesso di non scartare nessuno dei video ottenuti durante gli esperimenti. Bisogna notare che nessuno dei volontari era a conoscenza a priori delle password da digitare, ciò non è ideale ai fini dell'esperimento in quanto il modo di digitare una parola a caso è molto diverso dal modo di digitare la propria password. Tuttavia non è possibile chiedere ai volontari di utilizzare la propria password durante l'esperimento, sarebbero stati costretti a rivelarcela per permetterci di proseguire con la ricerca. Anche chiedere ai volontari di esercitarsi sulle 4 password prima dell'esperimento non è facile. Tale alternativa presenta i seguenti problemi:

- non è facile definire un numero di tentativi tale per cui una persona si possa definire "pronta" per l'esperimento;
- verificare che i volontari si siano effettivamente esercitati è difficile se i tentativi sono svolti da remoto (da casa);
- se i tentativi fossero svolti in più giorni sotto la sorveglianza di un operatore, il numero di volontari disponibili sarebbe calato drasticamente;
- non è realistico pensare che in tempi ragionevoli un volontario sia in grado di imparare a digitare in maniera naturale 4 password diverse. La *muscle memory* che entra in azione durante la digitazione di una password personale è un processo che richiede molto tempo per stabilirsi, non bastano delle settimane.

2.1.3 Le password

Per rendere l'esperimento più verosimile alla realtà è stato deciso di non generare password casuali, ma di selezionare 4 password da un *leak*. I *leak* sono database di password rubate da siti internet, le password contenute nei *leak* sono password autentiche di persone reali, rappresentano quindi un buon campione su cui testare il

[2]Digitare con solo una mano o solo un dito produrrebbe dei tempi per i quali non abbiamo una quantità sufficiente di dati con cui confrontarli. Dal punto di vista teorico, tuttavia, una vittima che digiti in questa maniera sarebbe più facile da attaccare in quanto il tempo fra la pressione due tasti risulterebbe maggiore e soprattutto diverso da altre coppie di tasti.

Projector Attack. Il *leak* utilizzato è noto al mondo informatico, si tratta di *rockyou.txt*, una lista di circa 32 milioni di password alla quale però abbiamo dovuto applicare i seguenti filtri:

- lunghezza della password di esattamente 8 caratteri;
- la password contiene solo caratteri in [a-z][0-9].

Il motivo per cui non è stato possibile utilizzare l'intero insieme di password è dovuto ai seguenti due motivi:

- nel *Projector Attack* la lunghezza della password della vittima è immediatamente nota, motivo per cui, nel testare l'attacco, possiamo limitarci al sottoinsieme di password lunghe un certo numero di caratteri, 8 nel nostro caso;
- mancanza di dati di training sufficienti per caratteri esterni a [a-z][0-9];

Le password finali selezionate casualmente ed utilizzate durante l'esperimento sono le seguenti: **jillie02**, **william1**, **123brian**, **lamondre**.

2.1.4 Svolgimento di un esperimento

A turno ogni volontario è stato fatto entrare nella stanza secondo l'orario deciso a priori; per ogni volontario sono stati riservati 15 minuti di tempo sebbene l'esperimento ne richiedesse molti meno. Al volontari è concessa la libertà di sedersi dove volessero purché tale posizione non ostruisse il proiettore o lo smartphone. Il volontario utilizzando il browser Google Chrome (oppure Chromium) visita la pagina web di *P-Logger*, e segue le istruzioni riportate. Un operatore avvia la registrazione e dà il via al volontario. Dopo che il volontario digita tutte le password richieste (almeno 3 tentativi corretti per ciascuna) la registrazione viene interrotta e si passa al prossimo volontario.

2.1.5 Gestione degli errori di digitazione

Il giorno dell'esperimento è stato chiesto ad ogni volontario di digitare le 4 password selezionate 3 volte ciascuna, per un totale di 12 tentativi corretti per volontario. Essendo di vitale importanza per l'esperimento la corretta digitazione della password, risulta essenziale accettare solamente tentativi corretti, questo significa che in ogni istante c'è uno e un solo carattere corretto da digitare, la pressione di un tasto non previsto in quel determinato istante causa l'aggiornamento della pagina con l'apparsa di un piccolo messaggio di errore che informa il volontario di dover digitare nuovamente la password. Una soluzione alternativa ed apparentemente più semplice è quella di controllare che vengano digitati solamente 8 tasti per tentativo, tuttavia questa soluzione potrebbe portare maggior frustrazione al volontario che solo alla pressione del 9° tasto scoprirà di aver commesso qualche errore durante la digitazione. Si noti che la pressione di tasti come *backspace* e/o i tasti "freccetta" è da considerarsi come errore.

2.2 P-Logger

P-Logger è il primo prodotto di *Project Projector*, esso ha lo scopo di raccogliere dati sui tasti digitati dai volontari durante gli esperimenti e di fornire una pagina di login familiare in cui gli utenti possano digitare una password. Le componenti principali di questo prodotto sono tre:

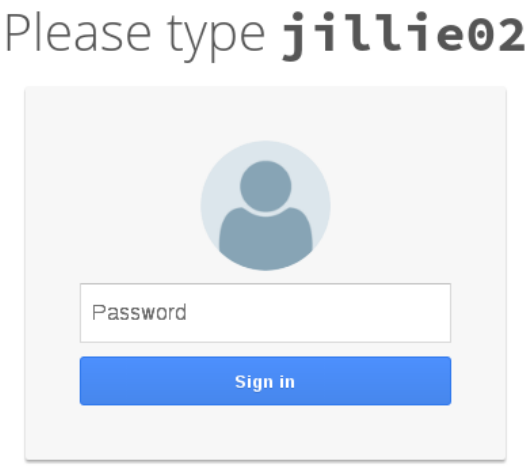


Figura 2.1: Pagina HTML di *P-Logger*

- **Pagina di Login**: è una pagina HTML (Figura 2.1) molto simile alla pagina di login di Google (`https://accounts.google.com/`);
- **Key Logger**: è il modulo Javascript che si occupa di raccogliere i tempi e controllare che i tasti digitati siano quelli corretti. I dati raccolti da questo modulo non sono i dati usati per effettuare l'attacco, essi servono solo per confronto con i dati ottenuti utilizzando *P-Extractor*;
- **Back End**: ha lo scopo di tracciare i volontari utilizzando degli *UUID*[3] e salvare tutti i dati degli esperimenti.

2.2.1 Analisi dei Requisiti

I requisiti di *P-Logger* possono essere visti come l'insieme dei requisiti delle tre componenti che lo compongono. In questo capitolo tuttavia essi verranno trascritti come requisiti di un unico prodotto. La notazione dei requisiti è la seguente: **R[X][Y][ID]** dove R sta per Requisito; X sta per {**F**unzionale, di **V**incolo, **Q**ualitativo}; Y sta per {**Obb**ligatorio, **Des**iderabile, **Opz**ionale}; ID è il numero identificativo del requisito.

I requisiti ad alto livello di *P-Logger* sono i seguenti:

- RVObb1: *P-Logger* deve essere sviluppato in Python (back-end);
- RFObb2: *P-Logger* deve fornire un' interfaccia utente in cui digitare password;
- RFObb3: *P-Logger* deve intercettare tutti i tasti premuti sulla tastiera da parte dell'utente;
- RFObb4: *P-Logger* deve salvare su un database i tasti premuti sulla tastiera;
- RFObb5: *P-Logger* deve fornire un metodo di tracciamento per i volontari ed i tentativi;

[3]Identificativo univoco universale.

- RFObb6: *P-Logger* deve fornire un metodo sicuro per scaricare i dati salvati;
- RFObb7: *P-Logger* deve mostrare un messaggio d'errore in caso di password sbagliata;
- RFObb8: *P-Logger* deve mostrare un messaggio di successo in caso di password corretta;
- RFObb9: *P-Logger* deve mostrare su schermo la password da digitare;
- RFObb10: *P-Logger* deve avvisare l'utente quando la sessione è finita;
- RFObb11: *P-Logger* deve fornire un'interfaccia in cui gli utenti possano essere registrati come volontari;

2.2.2 Progettazione Logica

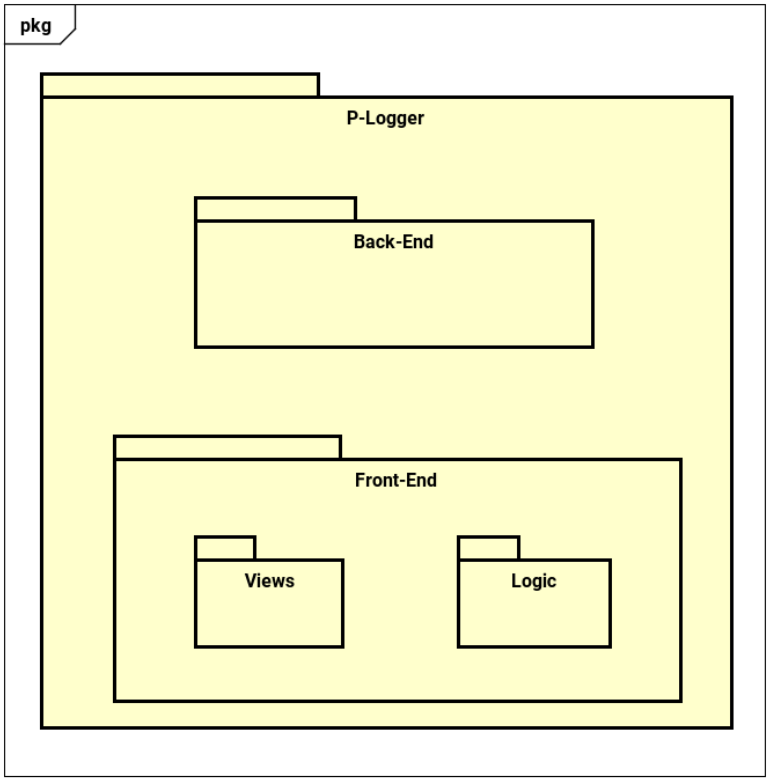


Figura 2.2: Main Package di *P-Logger*

Back-End

E' la parte scritta in Python, ha il compito di salvare i dati ricevuti dal Front-End e di fornire le pagine HTML quando richieste.

Front-End

Contiene le pagine HTML e il codice Javascript del modulo che fa da *keylogger*.

Front-End::Views

E' la componente logica che contiene tutti i template e le pagine HTML.

Front-End::Logic

Contiene la logica del Front-End di *P-Logger*, contiene le chiamate al server e la gestione degli errori lato *client*.

2.2.3 Progettazione di Dettaglio Back-End

Il Back-End offre le seguenti *routes*[4]

Route /create_db

È una route di amministrazione, serve per creare il database e le relative tabelle. È attivabile e disattivabile dal gestore del server tramite impostazione di variabili.

Route /reset_attempts

È una route di amministrazione, elimina tutti gli elementi nella tabella "attempts" del database. Anche in questo caso la *route* è attivabile e disattivabile dal gestore del server.

Route /get_uid - GET

Restituisce la pagina in cui un utente deve inserire un *nickname*.

Route /get_uid - POST

Assegna un UUID al *nickname*.

Route /experiment/<uid>

Dove <uid> è l'UUID assegnato da `get_uid`, restituisce la pagina in cui vengono digitate le password.

Route /over

La pagina mostrata alla fine di un esperimento.

Route /save/<attempt>

Salva il contenuto di <attempt> nel database.

[4]Il routing è un meccanismo che riscrive gli URL per renderli più user-friendly e per nascondere l'implementazione dell'architettura sottostante. Le routes rappresentano quindi delle risorse a cui l'utente può accedere visitando il realtivo URL.

Route /download

Richiede autenticazione, permette di visualizzare tutti risultati salvati nel database.

2.2.4 Progettazione di Dettaglio Front-End

La parte logica del Front-End è gestita dalla classe KeyLogger mostrata in Figura 2.3.

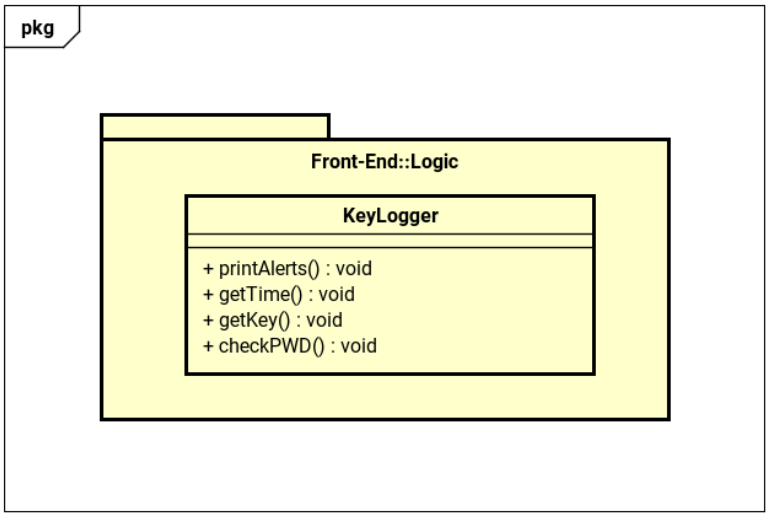


Figura 2.3: Classe KeyLogger, sono mostrate solo le principali funzionalità

Le funzioni più rilevanti della classe KeyLogger sono le seguenti:

- **printAlerts()**: si occupa di stampare i messaggi di errore e di successo ogniqualvolta venga effettuato il controllo sulla password;
- **getKey()**: ricava il *KeyCode* del tasto premuto dall'utente e controlla che sia il tasto atteso, in caso negativo chiama checkPWD() che restituirà un errore, altrimenti salva il tasto;
- **getTime()**: ricava il *timing* di un tasto premuto e lo salva;
- **checkPWD()**: controlla che la password sia stata digitata correttamente, se ciò non è vero verrà stampato un messaggio d'errore da printAlerts(), altrimenti viene inviata una richiesta POST a `/save` inviando i dati salvati in precedenza da getTime() e getKey().

2.3 Conclusione

P-Logger è il primo prodotto ad essere stato sviluppato. Sebbene alcune delle componenti siano risultate facili da sviluppare (pagina HTML, e Back-End in Python), lo sviluppo del *keylogger* si è dimostrato essere non banale per certi versi. In particolare, a prodotto finito si è presentato un errore logico che aveva luogo solo in determinate circostanze e che causava il *non* salvataggio di alcuni dati. Una appropriata metodologia di testing ha permesso di individuare l'errore e correggerlo prima che la circostanza si verificasse durante l'esecuzione dell'esperimento.

Tutti i tempi raccolti da *P-Logger* rappresentano la base di confronto con i tempi che verranno raccolti da *P-Extractor*: minore è la differenza fra i dati e maggiori sono le probabilità di successo del *Projector Attack*.

Capitolo 3

Estrazione dei dati

Il seguente capitolo tratta dello sviluppo di P-Extractor, un tool che riceve in input un video e ritorna in output un file di testo contente i timestamp *dei tasti digitati dai volontari. P-Extractor è stato sviluppato utilizzando la libreria OpenCV (`http://opencv.org/`) e utilizza algoritmi di* Computer Vision *per individuare in quale istante compaiono i pallini (•) che mascherano tipicamente le password. Il tool riduce notevolmente il tempo dedicato all'analisi dei video, questo ha permesso di utilizzare un numero di volontari e di tentativi adeguati all'esperimento descritto nel capitolo 2.*

3.1 Modalità di estrazione

Il numero di video registrati nella fase di sperimentazione ammonta ad un totale di 360 (30 volontari × 12 tentativi); sebbene una ventina di video siano stati analizzati a mano si è presto notato che l'analisi fotogramma per fotogramma di 360 video è un lavoro molto dispendioso in termini di tempo. Si è rivelato quindi necessario velocizzare e/o automatizzare il più possibile questo compito. *P-Extractor* è stato sviluppato per tale scopo: esso è un tool in grado di ricevere in input un video e di restituire in output un file di testo contenente il numero del fotogramma e il *timestamp* di comparsa di ogni pallino (•). Il vantaggio di avere dei video quasi identici tra loro (vedi paragrafo 2.1.1) permette di configurare *P-Extractor* una volta sola per poi eseguirlo su ogni video a disposizione. È necessario tenere in considerazione che uno strumento del genere può rivelarsi controproducente nel caso in cui i video in analisi siano diversi tra loro per posizione dello smartphone, illuminazione e/o altre condizioni fisiche. Infatti tali differenze portano inevitabilmente a dover riconfigurare *P-Extractor* per adattarlo ad ogni video, ciò comporta un consumo di tempo non indifferente.

3.1.1 Estrazione automatica

L'approccio inizialmente adottato per lo sviluppo di *P-Extractor* è stato quello di utilizzare algoritmi che facessero *shape* e *blob detection* in modo che il corretto funzionamento del programma dipendesse il meno possibile dai movimenti dello smartphone, tuttavia tale approccio si è rivelato fallimentare principalmente per i seguenti motivi:

- qualità del video troppo bassa e presenza di disturbo con conseguente rilevazione di falsi positivi;

- difficoltà nel parametrizzare le funzioni di *shape* e *blob detection*.

Alla luce dei risultati ottenuti si è scelto di ricorre a metodi più grezzi. L'idea è nata dall'osservazione che una volta individuata la casella di testo in cui viene inserita la password, sappiamo con precisione la posizione in cui prima o poi dovranno comparire i pallini. Non ci resta a quel punto che aspettare un cambiamento di colore relativamente importante in una certa zona, appena ciò accade vengono salvati il numero del frame e il *timestamp* e si passa ad osservare il punto d'interesse successivo. Più in dettaglio, per ogni frame del video vengono eseguite le seguenti operazioni:

1. applicazione di filtri per ridurre il rumore;
2. conversione in scala di grigi;
3. applicazione filtro soglia[1];
4. osservazione di un area di pixel molto ristretta in cui sappiamo che comparirà il pallino (ovvero ci sarà un cambiamento di colore importante, da bianco a nero).

Sebbene *P-Extractor* funzioni molto bene nel nostro scenario, il suo utilizzo in un attacco reale potrebbe non essere l'idea migliore. *P-Extractor* come già detto funziona perché ci sono una serie di pre-condizioni che vengono soddisfatte dai video registrati in fase di sperimentazione, ad esempio l'utilizzo del treppiede ci permette di assumere che lo smartphone non si sposti mai durante la registrazione del video e quindi possiamo essere certi che i pallini compaiano in posizioni prefissate, se così non fosse, *P-Extractor* non funzionerebbe a meno di modifiche importanti al codice. Non bisogna, però, confondere questo strumento come mezzo necessario o utile per un attacco realistico, lo scopo del programma è di ottenere dati il prima possibile per poi verificare la fattibilità dell'attacco stesso. Nel paragrafo seguente viene descritto come un attaccante potrebbe agire per attuare la fase di estrazione dei dati.

3.1.2 Estrazione manuale

L'estrazione manuale dei *timestamp* da video è un lavoro che richiede molto tempo, ma in uno scenario realistico è molto probabile che il numero di campioni disponibili per vittima equivalga ad 1. Un numero così basso rende più che fattibile l'analisi *frame by frame* del video, inoltre confrontando i tempi ottenuti dall'analisi manuale effettuata su una ventina di video e quelli ottenuti da *P-Extractor* si può verificare che l'analisi manuale risulta più precisa (seppur di qualche millisecondo) rispetto a quella automatica, per cui un attaccante sarà probabilmente propenso a perdere qualche minuto in più nel momento in cui ciò significhi guadagnare qualche millisecondo di precisione.

L'analisi manuale prevede che l'utente attraverso l'utilizzo di qualche software esterno scorra ogni fotogramma del video e si segni l'istante di tempo in cui compaiono i pallini. Solo quando si è in possesso di questi dati si può passare alla fase successiva dell'attacco.

[1] Ogni pixel ha un valore da 0 a 255 che rappresenta la gradazione di grigio, il filtro soglia rende bianchi i pixel sopra una certa soglia e neri quelli sotto trasformando l'immagine in pixel bianchi e pixel neri.

3.1.3 Computer Vision e OpenCV in P-Extractor

La visione computazionale o *Computer Vision* è un campo interdisciplinare che si occupa principalmente di come permettere alle macchine di carpire informazioni di alto livello da immagini o video. Lo scopo principale è quello di permettere alle macchine di comprendere il contesto di un'immagine; con "comprendere il contesto" si intende riuscire a distinguere oggetti ed ambiente con precisione ed usare tali informazioni per prendere decisioni.

OpenCV è una libreria open source di *Computer Vision* che offre strutture dati e metodi pronti all'uso per elaborare immagini. Come già descritto nei paragrafi precedenti, tale libreria si rivela molto utile per *P-Extractor*, attraverso essa siamo riusciti ad automatizzare un lavoro monotono e decisamente costoso in termini di tempo.

OpenCV in *P-Extractor* viene utilizzata per rilevare automaticamente i pallini (•) e salvare i *timestamp* relativi su un file di testo. *P-Extractor* è frutto di iterazioni ed integrazioni in quanto alcuni approcci iniziali come *shape detection* e *blob detection* si sono rivelati fallimentari. Di seguito vengono analizzati gli approcci adottati per lo sviluppo di *P-Extractor*.

Shape Detection

È un algoritmo che permette di rilevare ed identificare forme geometriche, si basa su un altro algoritmo di *Computer Vision* chiamato *Contour Approximation*. Quest'ultimo si occupa di approssimare il numero di spigoli di un dato oggetto in un immagine, l'algoritmo di *shape detection* si occupa di identificare la forma di un oggetto e di decidere se esso sia un quadrato, un rettangolo o una qualsiasi altra forma geometrica. Lo scopo iniziale era quello identificare i pallini (•) come forme circolari, ma è stato verificato che i pallini (nei video registrati) non hanno una forma circolare perfetta, bensì hanno una forma a "diamante", ovvero larghi nella metà alta e stretti nella metà bassa, ciò ha portato ad ottenere risultati non corretti. Questa tecnica è stata scartata.

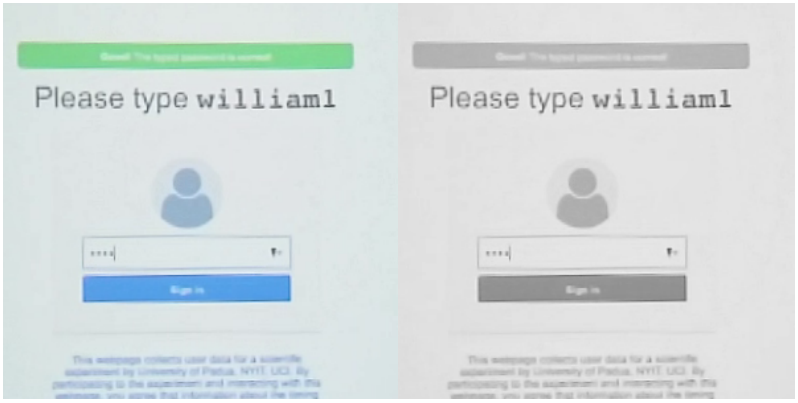
Blob Detection

È un algoritmo che ricerca i *blob*. I *blob* sono insiemi di pixel adiacenti che differiscono dal resto dell'immagine per varie caratteristiche (come ad esempio luminosità e dimensione). Anche in questo caso, si voleva utilizzare tale algoritmo per identificare i pallini della password in quanto essi avevano caratteristiche quasi identiche fra loro. Nel fare ciò si sono incontrate alcune difficoltà: (1) a causa del disturbo dell'immagine si incorreva spesso in falsi positivi, ovvero venivano rilevati dei *blob* in altre parti dell'immagine; (2) qualche *blob* veniva rilevato troppo tardi[2] rispetto alla comparsa sullo schermo effettiva; (3) la configurazione dei parametri della funzione non è banale, il tentativo di parametrizzare la funzione in modo che rilevasse *solo* i pallini della password si è rivelato insoddisfacente. L'algoritmo è stato scartato.

Object Tracking

L' *Object Tracking* è un ulteriore motivo per cui i due algoritmi appena descritti non hanno avuto successo. Riconoscere una forma geometrica o un *blob* al fotogramma f_{i} costituisce solo la prima parte della per rilevare correttamente un pallino. La

[2] Un ritardo anche di solo due frame può portare ad errori di una trentina di millisecondi, troppi per una buona riuscita dell'esperimento.



(a) Fotogramma senza nessun filtro **(b)** Conversione in scala di grigi

Figura 3.1: Filtro BGR to GRAY

seconda parte prevede che ai fotogrammi f_{i+1} e successivi, *P-Extractor* sia in grado di determinare quali siano i pallini che ha già rilevato nei fotogrammi precedenti. Ciò purtroppo non accade, spesso il disturbo presente nell'immagine causa la scomparsa (scomparsa agli "occhi" dell'algoritmo, non dall'immagine) di alcuni *blob*, per poi farli ricomparire qualche fotogramma più avanti e generare un falso positivo.

Dopo il fallimento degli approcci precedenti si è deciso di analizzare nuovamente il problema e di adottare una nuova strategia. Il primo passo è stato quello di rimuovere dall'immagine le informazioni non necessarie, il secondo passo è stato quello di utilizzare tecniche più grezze, ma più performanti rispetto a quelle precedenti. Vengono descritte di seguito le tecniche che hanno avuto successo.

BGR to GRAY

Per la rilevazione dei pallini risulta inutile l'informazione riguardante il colore, motivo per cui ad ogni fotogramma viene applicato una conversione dello spazio dei colori. L'immagine viene convertita in scala di grigi e ad ogni pixel viene ora assegnato un valore che va da 0 (colore nero) a 255 (colore bianco). Mostriamo in Figura 3.1 un esempio di fotogramma in BGR ed in scala di grigi.

Gaussian Blur

Nei video registrati durante gli esperimenti è facile notare del *flickering* o tremolio, esso è la principale causa dei falsi positivi degli approcci iniziali, risulta decisivo perciò rimuovere tale disturbo. Per fare ciò si può applicare un filtro come il *Gaussian Blur* che ha come effetto principale quello di "sfocare" e rendere meno nitida l'immagine. Una conseguenza di ciò è la rimozione del *flickering*.

Threshold Filter

Threshold Filter (o filtro soglia) è un filtro che permette di dividere i pixel di un immagine in due gruppi distinti, il criterio di divisione è appunto il valore di soglia. In *P-Extractor* dopo che l'immagine è stata processata dai due filtri citati sopra, viene

(a) Fotogramma con BGR to GRAY e Gaussian Blur

(b) Fotogramma dopo l'applicazione del filtro soglia

Figura 3.2: Filtro Threshold

applicato il filtro soglia. Per ogni pixel dell'immagine (che ricordiamo è ora in scala di grigi), viene valutato il valore della gradazione di grigio ([0;255]), se un pixel ha un valore minore della soglia, tale pixel verrà trasformato in un pixel nero, in caso contrario il pixel viene trasformato in un pixel bianco. Questo filtro è utile perché rimuove molti oggetti non utili all'esperimento. In Figura 3.2 si può notare l'effetto del filtro soglia su di un fotogramma di esempio.

Algoritmo di rilevazione

Ora che l'immagine permette di vedere distintamente i pallini, è possibile procedere con la parte restante dell'algoritmo. Analizzando dei video a campione si è intuito che la posizione dei pallini è generalmente statica (cambia con la risoluzione dello schermo del laptop che si sta utilizzando). Questo ci ha permesso di adottare un approccio semplice, ma funzionante: conoscendo a priori dove compariranno i pallini, possiamo posizionarci su un'area molto piccola del fotogramma ed aspettare che ci sia un cambio di colore importante. Ad ogni istante l'algoritmo controlla che la media del colore nell'area sotto osservazione sia sempre maggiore di una certa soglia, quando ciò non è più vero significa che è comparso un pallino. Appena accade ciò, l'area sotto osservazione si sposta di una quantità di pixel prefissata (i pallini sono equidistanti) in attesa del pallino successivo. Quando l'algoritmo ha rilevato 8 pallini, il programma salva i tempi di apparizione dei pallini su un file di testo e termina.

3.2 P-Extractor

P-Extractor è il secondo prodotto di *Project Projector*, ha lo scopo di automatizzare l'estrazione dei *timestamp* da video utilizzando librerie di Computer Vision.

3.2.1 Analisi dei Requisiti

La notazione dei requisiti è la seguente: **R[X][Y][ID]** dove R sta per Requisito; X sta per {**F**unzionale, di **V**incolo, **Q**ualitativo}; Y sta per {**Obb**ligatorio, **Des**iderabile, **Opz**ionale}; ID è il numero identificativo del requisito.

I requisiti di alto livello di *P-Extractor* sono i seguenti:

- RVObb12: *P-Extractor* deve essere sviluppato in Python utilizzando le librerie OpenCV;

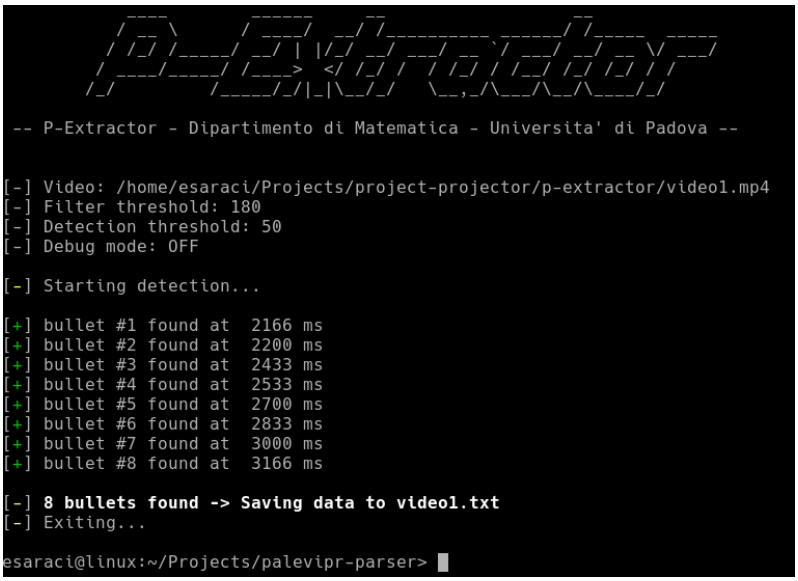


Figura 3.3: Screenshot di un'esecuzione di *P-Extractor*

- RFObb13: *P-Extractor* deve estrarre automaticamente i tempi dato un video in input;
- RFObb14: *P-Extractor* deve salvare i dati in un file JSON;
- RFObb15: *P-Extractor* deve essere in grado di processare più video alla volta;
- RQObb16: *P-Extractor* deve estrarre dei tempi che abbiano un errore minore di 10ms (rispetto all'estrazione manuale);
- RFDes17: *P-Extractor* deve fornire una modalità "debug" in cui venga mostrato graficamente il processo di estrazione.

3.2.2 Progettazione Logica

Le componenti che compongono l'architettura di alto livello di *P-Extractor* (vedasi Figura 3.4) sono le seguenti:

- VideoLoader: è la componente che si occupa di caricare il/i video;
- Extractor: si occupa di applicare tutti i filtri (di OpenCV) necessari all'immagine e di estrae i tempi correttamente;
- Output: salva i tempi in un file JSON.

3.2.3 Progettazione di Dettaglio

P-Extractor::VideoLoader

La classe VideoLoader (Figura 3.5) contiene i seguenti campi dati:

- videos: str[]: è un array di stringhe contenente i path dei video in una determinata cartella.

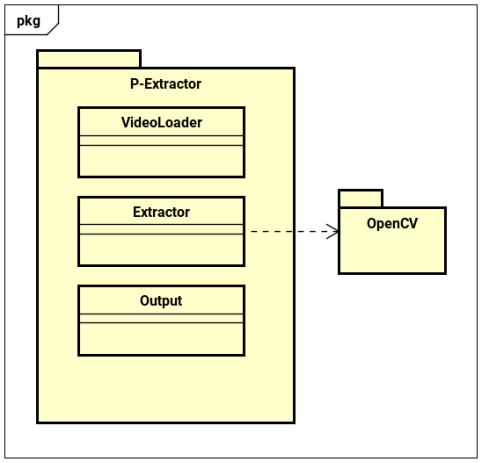


Figura 3.4: Architettura di alto livello di *P-Extractor*

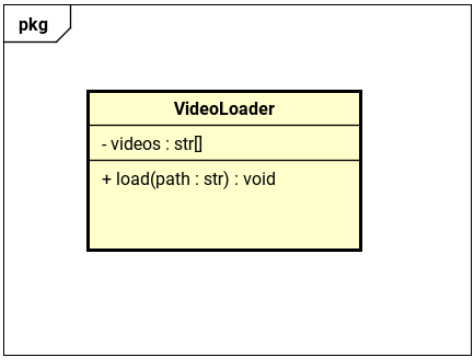


Figura 3.5: Dettaglio della classe VideoLoader

La classe VideoLoader (Figura 3.5) contiene i seguenti metodi:

- load(path: str): void: ha il compito di inserire come stringhe nel campo dati *videos* i video trovati nella cartella *path*, se *path* è un file allora sarà quest'ultimo ad essere inserito come stringa nel campo dati privato *videos*.
- getVideo(): string[]: restituisce il campo dati *videos*

P-Extractor::Extractor

La classe Extractor (Figura 3.6) contiene i seguenti campi dati:

- timestamps: float[]: array contenente i timestamp estratti da video;
- frames: int[]: array contenente il numero del frame in cui sono comparsi i pallini;
- videos: str[]: array contenente i path dei video in analisi;
- debug: boolean: se true vengono mostrate le finestre con le applicazioni dei vari filtri.

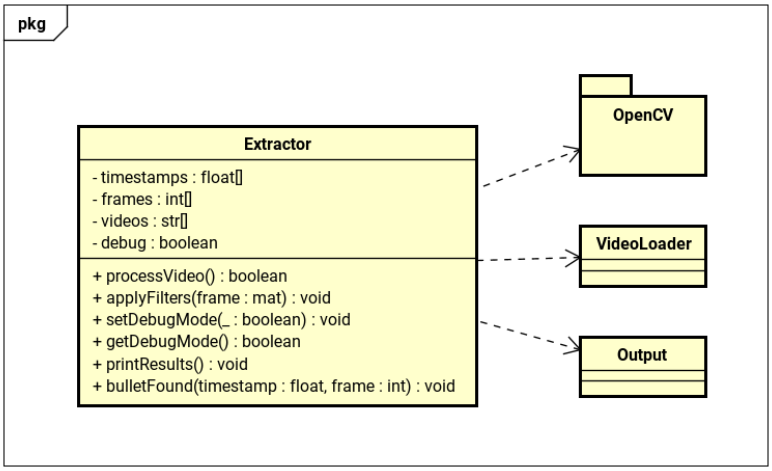


Figura 3.6: Dettaglio della classe Extractor

La classe Extractor (Figura 3.6) contiene i seguenti metodi:

- processVideo(): boolean: effettua tutte le chiamate necessarie per processare un video, ritorna true se tutto è andato a buon fine, false altrimenti;
- applyFilters(frame: mat): void: applica tutti i filtri necessari, è chiamato per ogni frame;
- setDebugMode(_: bool): void: *setter* per il campo dati *debug*;
- getDebugMode(): boolean: *getter* per il campo dati *debug*;
- printResults(): void chiama la funzione della classe Output per stampare i dati contenuti nei campi dati *timestamps* e *frame*;
- bulletFound(): void: metodo da chiamare ogniqualvolta si trovi un pallino da video.

P-Extractor::Output

La classe Output (Figura 3.7) contiene i seguenti metodi:

- save(name: str): void: salva un file di testo con nome.

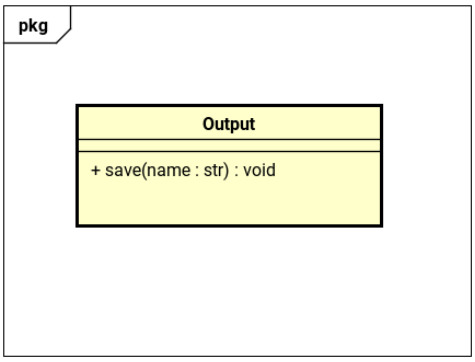


Figura 3.7: Dettaglio della classe Output

3.3 Conclusione

In conclusione, *P-Extractor* è stato un prodotto che ha richiesto molte iterazioni, i fotogrammi dei video non erano conosciuti a priori e risultava complicato prevedere quale sarebbe stata la strategia migliore da adottare. *P-Extractor* ha avuto un peso importante all'interno di *Project Projector* e del *Projector Attack*, in quanto ci ha permesso di ottenere una grande quantità di dati (360 video analizzati in totale) in un tempo molto ristretto. Tutti i dati ottenuti da *P-Extractor* sono pronti per essere elaborati da *P-Crack*.

Capitolo 4

Analisi dei dati

In questo capitolo viene spiegato quale sia il modello dell'attacco e come un attaccante possa agire per arrivare ad ottenere la password di una vittima. Prima di passare alla descrizione tecnica di P-Crack vengono introdotti i concetti di Machine Learning *e* Random Forest*, oltre ad altri aspetti teorici che sono alla base dell'attacco.*

4.1 Machine Learning

Il *Machine Learning* o apprendimento automatico è un'area fondamentale dell'intelligenza artificiale che si occupa del design e costruzione di sistemi o algoritmi in grado di sintetizzare conoscenza partendo da osservazioni esterne. In intelligenza artificiale ci sono tre principali paradigmi di apprendimento:

- **Apprendimento supervisionato**: all'algoritmo vengono forniti input da elaborare ed output desiderato, l'obiettivo è che l'algoritmo sia in grado di apprendere una regola generale che mappi l'input all'output corretto;
- **Apprendimento non supervisionato**: l'algoritmo deve trovare una struttura negli input forniti i quali però non hanno un'esplicita suddivisione;
- **Apprendimento con rinforzo**: il sistema è posto in un ambiente dinamico in cui deve adattarsi, l'unico modo per capire se si sta adattando correttamente sono le "ricompense" chiamate rinforzo che consistono in una valutazione delle prestazioni.

Di seguito un'introduzione all'apprendimento supervisionato, il paradigma usato da *P-Crack*.

4.1.1 Apprendimento supervisionato

L'apprendimento supervisionato punta a costruire un sistema in grado risolvere autonomamente un compito a seguito di un periodo di esperienza (fase di *training*) in cui gli vengono mostrati degli esempi ideali sempre corretti ovvero coppie di input e output aspettato. L'idea è che dopo l'apprendimento il sistema abbia abbastanza esperienza da essere in grado di ricevere un input mai visto in passato e mapparlo correttamente nell'output corretto (fase di *testing*). L'insieme di input e output forniti in fase di

training viene chiamato *training set*, l'insieme fornito al sistema in fase di *testing* si chiama *test set*. Si noti che durante la fase di *testing* noi siamo già a conoscenza dell'output corretto, ma vogliamo verificare che il sistema abbia appreso correttamente il modo per adempiere al compito. Se il sistema produce una buona percentuale di output corretti possiamo affermare che l'apprendimento sia andato a buon fine.

È utile notare che training e test set devono essere insiemi separati; se il *testing* venisse fatto con i dati del *training set* non saremmo in grado di dire che l'algoritmo abbia effettivamente appreso, in quanto gli eventuali output corretti prodotti dall'algoritmo potrebbero essere frutto di memorizzazione (di input e output atteso) e non di generalizzazione (o apprendimento). Per questo motivo, in fase di *testing*, è necessario fornire degli output che l'algoritmo non ha mai visto prima, così da verificare che ci sia stato un *vero* apprendimento.

4.1.2 Random Forest

Random Forest è il nome dell'algoritmo che utilizza *P-Crack*. Random Forest rientra in una categoria di algoritmi chiamati classificatori, il cui scopo è quello di mappare i dati in input nelle cosiddette *classi* obiettivo. Le classi sono gruppi dentro i quali chiediamo a Random Forest di inserire un certo elemento in input, i criteri secondo cui Random Forest compie una scelta sono generalmente le caratteristiche dell'input. Nel nome Random Forest, "Forest" deriva dal fatto che l'algoritmo fa uso di molti alberi decisionali. Un albero decisionale può essere visto come un grafo in cui i nodi rappresentano delle condizioni *if* e gli archi rappresentano le risposte a tale condizione. Le foglie degli alberi rappresentano invece le classi obiettivo. Essendoci molti alberi decisionali, ogni albero potrebbe ottenere una risposta diversa dagli altri, la risposta finale di Random Forest sarà quella presentatasi più volte. Il nome "Random" invece sta ad indicare il criterio con cui si decidono le condizioni nei nodi. Supponiamo di dover classificare dei fiori, a Random Forest diamo in input la lunghezza del gambo, il numero di petali, il numero di foglie e la classe obiettivo del fiore. Quando Random Forest costruirà il modello di alberi interno, le condizioni sui nodi saranno scelte randomicamente tra le 3 *features* offerte dagli oggetti in input.

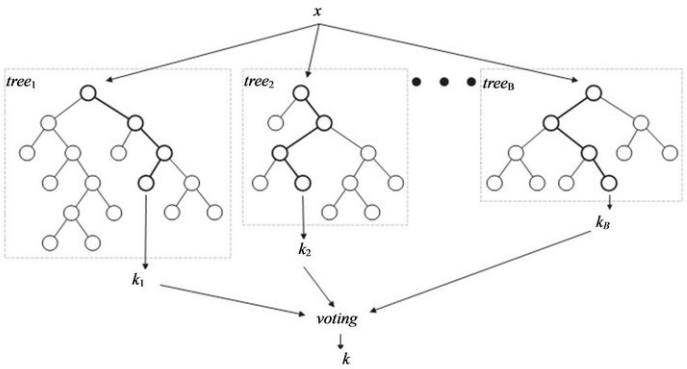


Figura 4.1: Esempio di Random Forest
Dato un input x, le risposte k_n degli alberi possono non essere tutte uguali, la risposta finale k dell'algoritmo sarà la risposta comparsa più volte. Immagine tratta da [7]

4.2 Design dell'attacco

Il *Training Set* è l'insieme di coppie di input ed output aspettati dati in pasto a Random Forest, il quale ha il compito di imparare una regola generale che gli permetta in futuro di generare output corretti dati input mai visti in passato. Nel *Projector Attack* il *training set* è composto da coppie di digrafi e tempi (o *timing*). I digrafi sono coppie di caratteri ('av', 'b8', etc), mentre i tempi sono i relativi tempi impiegati a digitare tali coppie di caratteri, quindi il training set nel nostro modello può essere visto come una lista di digrafi d_i, e una lista dei relativi tempi t_i. In fase di testing i digrafi d_i rappresenteranno le classi obiettivo, i tempi t_i saranno invece ciò che verrà fornito in input a Random Forest. Il classificatore (Random Forest) dovrà quindi, dato un timing t_i in input, restituire un digrafo d_i tale che d_i sia il digrafo più probabile. Un attaccante che abbia a disposizione i tempi di digitazione di una password (ottenuti come spiegato nel paragrafo 3.1.2) ed un modello allenato[1], potrà dare in input a quel modello i tempi t_i che ha rilevato; il classificatore restituirà per ogni t_i, un digrafo d_i per cui ha maggior *confidence*.

Come è forse facile intuire, se fosse così facile ottenere una password, il *Projector Attack* sarebbe già stato utilizzato in passato e non verrebbe considerato innovativo. Per l'attaccante, la probabilità che il classificatore restituisca il digrafo d_i corretto, è vicina allo 0. La Figura 4.2 mostra il perché di questa probabilità: le distribuzioni dei *timing* dei digrafi risultano essere molto simili per digrafi differenti, ciò rende difficile distinguere un digrafo da un altro basandosi solo sul *timing*. Una soluzione a questo problema è quella è quella di far restituire al classificatore le probabilità per ogni digrafo di essere il digrafo corretto, in liste ordinate. La password poi andrebbe composta dall'attaccante provando tutte le possibili combinazioni coi digrafi ottenuti.

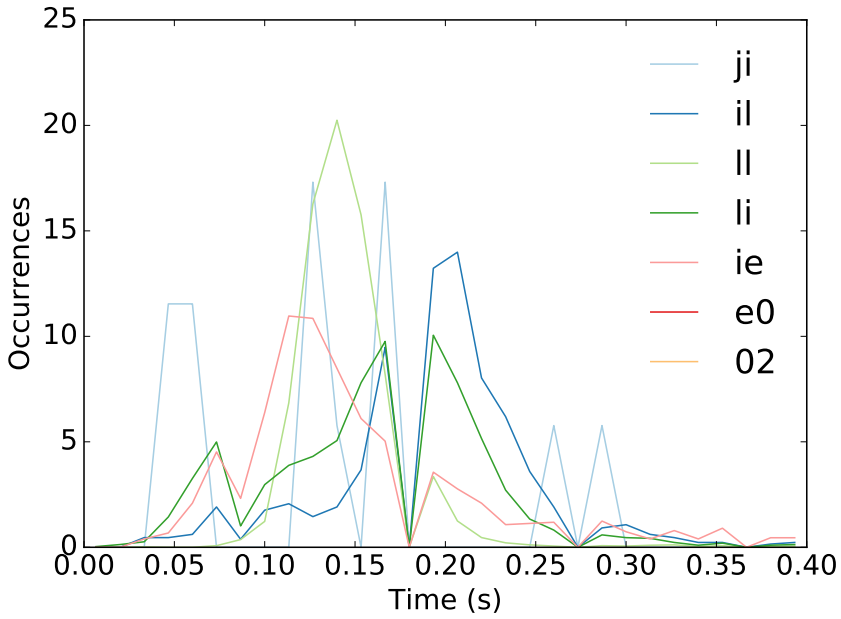
4.2.1 Training Set

I dati ottenuti dai *dataset* pubblici [2, 5, 8, 11] non sono dati nel formato utile al training di Random Forest, essi sono per la maggior parte testo digitato da volontari e timestamp per ogni tasto digitato. Una parte del tempo è stata spesa a convertire tali dati in digrafi e *timing* relativi. Durante questa conversione sono state fatte delle scelte che hanno portato a scartare alcuni possibili digrafi. Le scelte sono le seguenti:

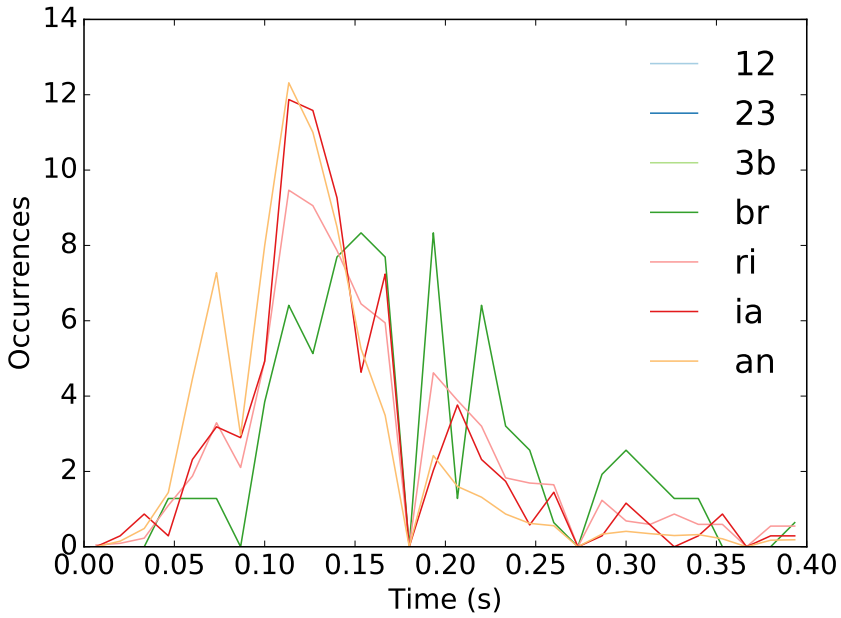
- le lettere maiuscole sono state scartate come semplificazione; considerare il tasto "Shift" sarà fatto come estensione futura, ad esempio considerando che tra due lettere intercorre un tempo innaturalmente lungo;
- sono stati scartati gli spazi e tutti i caratteri non appartenenti a [a-z][0-9] in quanto i dati disponibili relativi ad essi nei dataset non erano sufficienti.

In totale, come conseguenza dei filtri di cui sopra, il numero di digrafi univoci possibili nel nostro sistema è: $(|[a-z]| + |[0-9]|)^2 = 1296$. Tuttavia il numero di digrafi univoci forniti in *training* al classificatore risulta minore, questo perché alcuni dei digrafi contenuti in quel numero sono digrafi che non sono mai stati digitati nei *dataset*. Il nostro classificatore non è in grado di predire una classe che non gli è mai stata fornita durante la fase di *training*. I digrafi e i tempi ottenuti dai *dataset* pubblici sono stati uniti in unico insieme, tale insieme verrà chiamato *population data* e rappresenta ciò che il nostro classificatore ha ricevuto in input durante la fase di *training*.

[1] Allenato = che ha subito la fase di training.



(a) jillie02



(b) 123brian

Figura 4.2: Distribuzione dei tempi dei digrafi delle password **jillie02** e **123brian**.

Come si può notare dai grafici, la distribuzione dei *timing* risulta essere molto simile per alcuni digrafi, questo rende difficile per *Random Forest* distinguere un digrafo da un altro basandosi solo sul tempo, che purtroppo continua ad essere l'unica *feature* disponibile nello scenario in analisi.

4.2.2 Modalità di utilizzo di P-Crack

P-Crack offre tre modalità di utilizzo:

- **Benchmark**;
- **Testing**;
- **Attack**.

Benchmark Mode

È la modalità sviluppata per prima, ci permette di valutare quanto bene funziona *P-Crack* e il *Projector Attack* in generale. Il *training set* consiste nei *population data*, mentre il *test set* consiste in una serie di password generate casualmente utilizzando i digrafi contenuti in una parte, esclusa dal training, di *population data*. L'algoritmo di generazione casuale della password funziona nel seguente modo:

1. sceglie un digrafo d_1 a caso come primo digrafo della password e lo salva;
2. sceglie un digrafo d_2 a caso tale che $d\_2\hat{}1 = d\_1\hat{}2$, dove $d\_x\hat{}y$ indica l'y-esimo carattere del digrafo x-esimo, e lo salva;
3. ripete il procedimento fino all'estrazione di d_7;
4. ricomincia dall'inizio e genera una nuova password.

Supponiamo ad esempio che d_1 estratto a caso sia il digrafo ab, il digrafo d_2 deve essere scelto a caso, ma nel sottoinsieme di digrafi che iniziano con l'ultima lettera del digrafo precedente. In questo caso il secondo il digrafo viene scelto a caso fra i digrafi che iniziano con la lettera b. Tale ragionamento viene utilizzato fino a comporre password di 8 caratteri (= 7 digrafi).

Una volta che sono state generate n password, per ogni password psw_i vengono forniti in input al classificatore i tempi dei 7 digrafi che la compongono, *P-Crack* controlla se nelle liste di digrafi predetti da Random Forest è presente una combinazione di digrafi per cui la password psw_i possa essere trovata, in caso positivo calcola il numero di tentativi che sarebbero stati necessari a trovarla.

Testing Mode

È una modalità molto simile a quella precedente, la differenza sostanziale è che il *test set* stavolta è composto da password e tempi noti e non generati a caso partendo dal *training set*. È il procedimento utilizzato per testare i dati raccolti tramite l'estrazione automatica dei tempi descritta nel paragrafo 3.1.1. Per ogni tentativo di ogni volontario vengono estratti i tempi fra un tasto e l'altro e vengono forniti al classificatore insieme all'output atteso (la password stessa). Anche in questo caso *P-Crack* si occupa di controllare che Random Forest abbia generato delle liste di predizioni che permettano all'attaccante di trovare la password componendo i digrafi restituiti.

Attack Mode

È una modalità senza *test set*, il motivo è che non si sta cercando di testare l'apprendimento del classificatore, ma si sta cercando di effettuare un attacco vero e proprio. Questo significa che non siamo a conoscenza dei digrafi che compongono la password,

ma solo dei tempi fra un tasto e l'altro, ricavati magari come descritto in 3.1.2. In questo caso P-Crack si limita a restituire una lista di digrafi per ogni *timing* inserito in input. Non si sa a priori se la password è presente in una delle combinazioni di digrafi restituiti. L'attaccante dovrà provare tutte le password che ha ottenuto e sperare che una di quelle sia quella giusta.

4.3 P-Crack

Figura 4.3: Esecuzione di P-Crack in modalità benchmark

4.3.1 Analisi dei Requisiti

La notazione dei requisiti è la seguente: **R[X][Y][ID]** dove R sta per Requisito; X sta per {**F**unzionale, di **V**incolo, **Q**ualitativo}; Y sta per {**Obb**ligatorio, **Des**iderabile, **Opz**ionale}; ID è il numero identificativo del requisito.

I requisiti di alto livello di *P-Crack* sono i seguenti:

- RVObb18: *P-Extractor* deve essere sviluppato in Python utilizzando le librerie SciKit-learn;
- RVObb19: *P-Extractor* deve poter essere eseguito in parallelo;
- RFObb20: *P-Extractor* deve poter salvare file di testo o grafici;
- RFObb21: *P-Extractor* deve gestire dataset in training e testing;
- RQObb22: *P-Extractor* nella modalità *attack* deve restituire una lista di predizioni;
- RFObb23: *P-Extractor* deve poter *crackare* una password nel caso in cui le liste predette lo permettano.
- RQDes24: *P-Extractor* deve fornire più algoritmi di *cracking*.

4.3.2 Progettazione Logica

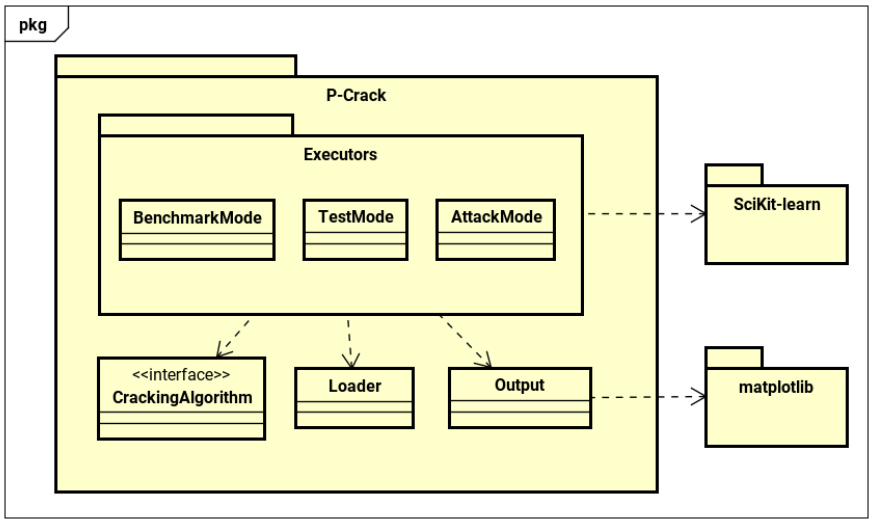


Figura 4.4: Architettura di alto livello di *P-Crack*

Le componenti che compongono l'architettura di alto livello di *P-Crack* (vedasi Figura 4.4) sono le seguenti:

- Executors: sono le classi che rappresentano le varie modalità di esecuzione descritte nel capitolo 4.2.2, non sono tre classi separate. Le principali differenze fra le modalità di esecuzione stanno nelle differenti tipologie di training e test set.
- CrackingAlgorithm: è l'interfaccia che sta ad indicare che per aggiungere un nuovo algoritmo di cracking a *P-Crack* è sufficiente rispettare il contratto di questa interfaccia. Il compito dell'algoritmo è quello comporre in maniera differente i digrafi ottenuti da Random Forest;
- Loader: è il modulo che gestisce i dataset, il training set ed il test set;
- Output: è il modulo a cui è delegata la responsabilità di creare e salvare grafici.

4.3.3 Progettazione di Dettaglio

P-Crack::Loader

La classe Loader (Figura 4.5) contiene i seguenti campi dati:

- training_set_x: float[]: è un array di float;
- training_set_y: str[]: è un array di str;
- test_set_x: float[]: è un array di float;
- test_set_y: str[]: è un array di str.

La classe Loader (Figura 4.5) contiene i seguenti metodi:

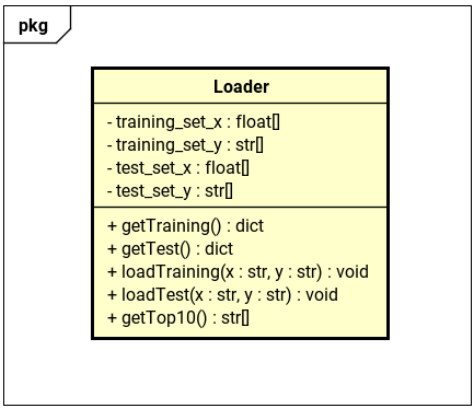


Figura 4.5: Dettaglio della classe Loader

- loadTraining(x: str, y: str): void: carica i file di testo indicati in x (digrafi) e y (*timing*);
- loadTest(x: str, y: str): void: carica i file di testo indicati in x (digrafi) e y (*timing*, y può essere vuota);
- getTop10(): str[]: restituisce la lista dei 10 digrafi più ripetuti;
- getTraining(): dict ritorna la coppia training_set_x, training_set_y;
- getTest(): dict ritorna la coppia test_set_x, test_set_y.

P-Crack::CrackingAlgorithm

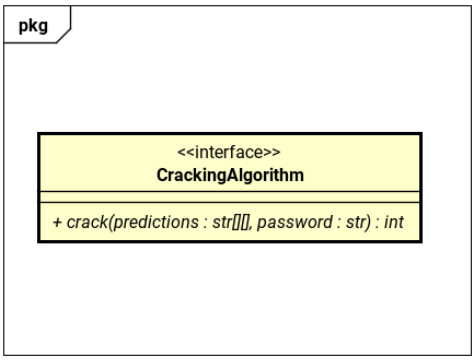


Figura 4.6: Dettaglio dell'interfaccia CrackingAlgorithm

Per utilizzare un nuovo algoritmo in *P-Crack* è necessario implementare il metodo chiamato "crack". Le specifiche del metodo crack sono le seguenti:

crack(predictions: str[][], password: str) : int,

dove *predictions* è un array di 7 array contenenti *top_ n* digrafi predetti in ciascuno, e password è la password che si sta cercando di *crackare*. Il parametro di ritorno è il numero di tentativi impiegati a *crackare* la *password*.

P-Crack::Executor

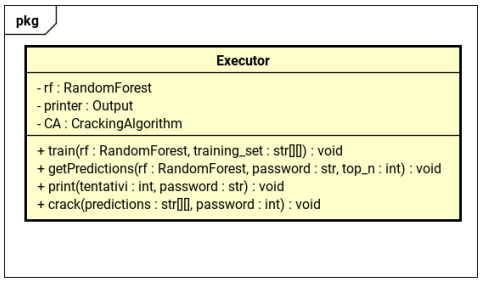


Figura 4.7: Dettaglio della classe Executor

La classe Loader (Figura 4.7) contiene i seguenti campi dati:

- rf: RandomForest: è un riferimento ad un oggetto di tipo RandomForest dalla libreria SciKit-learn;
- CA: CrackingAlgorithm: è un'istanza di un CrackingAlgorithm;
- printer: Output: è un oggetto Output, ha lo scopo di stampare testo o grafici a seconda dell'implementazione.

La classe Executor (Figura 4.7) contiene i seguenti metodi:

- train(rf: RandomForest, training_set: dict): delega il train a *rf*;
- getPredictions(timings: float[], RandomForest: object, top_n: int): str[][]: ottiene la lista di digrafi predetti per ogni *timing* in *timings*. Delegato ad *rf*;
- crack(predictions: str[][], password: string): int: delega a CA.crack();
- print(tentativi: int, password) void: funzione di stampa delegata a printer.print().

4.4 Conclusione

P-Crack è certamente l'applicativo più complesso dei tre. Grazie ad esso siamo riusciti ad ottenere molti dati utili alla ricerca. I risultati ottenuti dagli esperimenti effettuati con *P-Crack* sono riportati nel capitolo seguente.

Capitolo 5

Risultati

In questo capitolo vengono raccolti i risultati dell'esperimento. Vengono prima mostrati una serie di risultati preliminari, alcuni dei quali sono necessari per comprendere al meglio la configurazione successiva di P-Crack. Successivamente si mostrano quali siano le capacità effettive del Projector Attack.

5.1 Risultati preliminari

Prima di procedere coi risultati del *Projector Attack* è bene analizzare alcuni risultati preliminari necessari per la totale comprensione di alcune scelte fatte nei passi successivi.

5.1.1 Training Set

Nella lettura della seguente tabella bisogna tenere a mente che nei dati di training sono stati mantenuti solamente i digrafi con caratteri in [a-z][0-9]. Ciò permette al massimo 1296 possibili digrafi.

Tabella 5.1: Alcuni dati sul training set

Numero di digrafi nel training set	2309155
Numero di digrafi univoci	883/1296 (68%)
Timing medio di un digrafo	0.156 s
Deviazione standard dei timing	0.144 s

5.1.2 Quantità di digrafi predetti dal classificatore

Come specificato nel capitolo 4.2, il classificatore restituisce in output una lista di digrafi per i quali ha maggior *confidence*. Il numero di digrafi da restituire non è scelto a caso, ma è frutto dei seguenti test. Tale numero verrà chiamato top_n nei prossimi paragrafi.

Per il primo esperimento abbiamo usato come training set i dati estratti da [2] e come test set i dati estratti da [8]. L'esperimento è stato testato con $top\_n = 50$, ovvero per ogni *timing* fornito in input il classificatore restituisce 50 digrafi ordinati per *confidence*. Lo scopo dell'esperimento era principalmente quello di analizzare la natura dei nostri dataset, un risultato troppo basso ci avrebbe fatto dubitare della

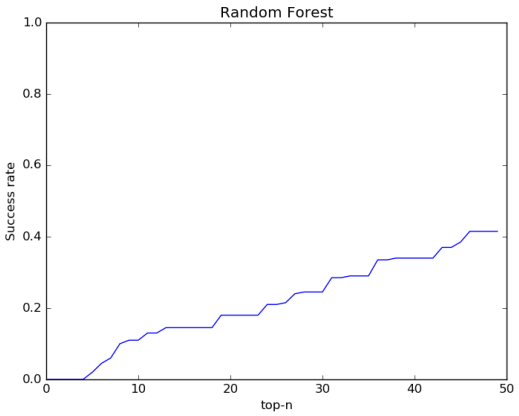


Figura 5.1: Risultati del primo esperimento.

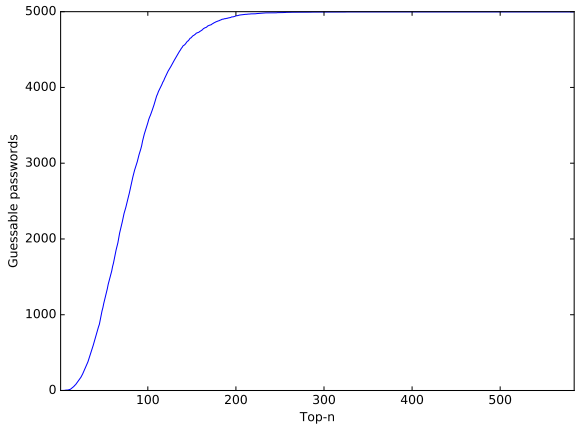


Figura 5.2: Risultati del secondo esperimento.

bontà dei dati contenuti in [2, 5, 8, 11]. Fortunatamente il risultato mostrato nella Figura 5.1 è relativamente positivo: con $top\_n = 50$ si ha una probabilità del 40% che il digrafo cercato sia contenuto nei 50 digrafi restituiti dal classificatore.

Il secondo esperimento è stato una prova della modalità *benchmark* di *P-Crack*. Il training set era composto dai *population data*, mentre il test set era composto da una piccola parte dei *population data* che non sono stati forniti in training. Sono state generate 5000 password da *crackare* e per ognuna di esse abbiamo testato vari valori di top_n. Il risultato in Figura 5.2 indica con $top\_n = 250$, il classificatore ha predetto correttamente, all'interno di 250 elementi, il digrafo che stavamo cercando. Di conseguenza, abbiamo prova che il *Projector Attack*sia fattibile: se tra 1296 digrafi possibili il digrafo corretto è costantemente nei primi 250, significa che il *timing* contiene informazioni sufficienti per ottenere miglioramenti rispetto, ad esempio, al selezionare digrafi casuali (il classico approccio a forza bruta).

5.2 Risultati dell'attacco

L'attacco vero è proprio è molto simile all'esperimento precedente, l'unica differenza è che *P-Crack* viene eseguito in modalità *test*, quindi il test set non è formato da password casuali, ma è composto dai dati estratti utilizzando *P-Extractor*. Con *top_ n* = 250 i risultati sono i seguenti:

Tabella 5.2: Risultati finali

Numero di password testate	5000
Top N	250
Password crackate	2379/5000 (48%)
Media iterazioni	188195154

Il risultato di per sé non dice molto, ma la vera efficacia dell'attacco si nota in confronto a bruteforce. Bruteforce impiegherebbe in media[1] $36^8/2$ tentativi che equivalgono a 1410554953728 tentativi. Il nostro attacco quando ha successo impiega in media 188195154 tentativi, arrivando così a ridurre il tempo impiegato da bruteforce di 5 ordini di grandezza. Il tutto va considerato tenendo conto che il *Projector Attack* presenta un costo di esecuzione quasi nullo, rendendo l'attacco decisamente *cost-efficient*.

Supponendo un rate di 20 milioni di tentativi al secondo, un bruteforce normale richiederebbe circa 20 ore di calcolo. Grazie alle informazioni sfruttate dal nostro attacco, il tempo scenderebbe in media a 9 secondi.

[1] Per crackare una password da 8 caratteri composta solo da caratteri in [a-z][0-9]

Capitolo 6

Conclusioni

Il capitolo riporta un bilancio finale dal punto di vista degli obiettivi raggiunti e dal punto di conoscenze e competenze acquisite.

6.1 Obiettivi raggiunti e analisi del prodotto

Gli obiettivi definiti nel piano di lavoro iniziale sono stati raggiunti con successo. Gli applicativi prodotti si sono dimostrati efficaci e verranno utilizzati ed estesi ulteriormente fino alla conclusione della ricerca. *Project Projector* ha il principale vantaggio di essere modulare, questo significa che le componenti che lo compongono non dipendono l'una dalle altre e possono essere riutilizzate in altri ambiti purché si rispettino i *contratti* richiesti. Un'altra buona caratteristica del prodotto è l'estensibilità.

In *P-Crack*, ad esempio, è facilitata l'aggiunta di algoritmi di cracking, con l'unico limite (necessario) di mantenere l'interfaccia richiesta dal programma.

P-Extractor ha molto da offrire in termini di adattamento a nuovi scenari. Durante lo sviluppo di *P-Extractor* sono stati provati molti approcci, alcuni dei quali sono falliti a causa della natura dei video registrati. In scenari alternativi, *P-Extractor* potrebbe essere nuovamente integrato con le funzionalità di *blob* e *shape detection* diventando così un tool più automatizzato e con meno necessità di configurazione.

Anche *P-Logger* per come è strutturato offre margine di adattamento. La pagina HTML ad esempio potrebbe essere personalizzata e ristrutturata per meglio simulare altri scenari, ciò non comporta nessuna modifica nelle altri componenti di *P-Logger* proprio perché sono stati sviluppati per adempiere a compiti separati.

6.2 Conoscenze acquisite e valutazione personale

Lo studio del *Projector Attack* e lo sviluppo di *Project Projector* mi hanno permesso di studiare aree dell'informatica che ho sempre visto come distanti. I corsi della Laurea Triennale mi hanno permesso di apprendere in fretta le nuove tecnologie da utilizzare all'interno dello stage. In particolare prima di iniziare lo stage avevo conoscenze basilari del linguaggio Python e nessuna conoscenza di *machine learning* e/o *computer vision*; il fatto che sia stato in grado di comprendere, applicare ed utilizzare ragionevolmente tali tecnologie è, secondo me, la prova di quanto il Corso di Laurea Triennale in Informatica porti con sé e di quanto aiuti gli studenti a prepararsi per ciò che non hanno ancora affrontato. Grazie allo stage ho avuto modo di applicare concetti studiati nei vari corsi

in uno scenario realistico e pratico, arrivando a produrre applicativi modulari, aperti all'estensione e capaci di eseguire parallelamente.

Infine, lo stage si colloca nell'ambito della sicurezza informatica, che è sempre stata la mia passione; essere introdotto in un *environment* in cui si svolge ricerca in tale ambito si è rivelato essere fonte di stimolo per il periodo dello stage e sicuramente anche per il futuro.

Bibliografia

[1] National Security Agency. *TEMPEST: a signal problem*. 1972. URL: http://web.archive.org/web/20080726115754/www.nsa.gov/public/pdf/tempest.pdf (cit. a p. 4).

[2] Yejin Choi. «Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays». In: *Empirical Methods on Natural Language Processing (EMNLP)* (2014) (cit. alle pp. 1, 27, 35, 36).

[3] Alberto Compagno et al. «Don'T Skype & Type!: Acoustic Eavesdropping in Voice-Over-IP». In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 703–715. ISBN: 978-1-4503-4944-4. DOI: 10.1145/3052973.3053005. URL: http://doi.acm.org/10.1145/3052973.3053005 (cit. a p. 4).

[4] Daniel Genkin, Adi Shamir e Eran Tromer. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*. Cryptology ePrint Archive, Report 2013/857. http://eprint.iacr.org/2013/857. 2013 (cit. a p. 3).

[5] Kevin S Killourhy e Roy A Maxion. «Comparing anomaly-detection algorithms for keystroke dynamics». In: *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE. 2009, pp. 125–134 (cit. alle pp. 1, 27, 36).

[6] Paul C. Kocher. «Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems». In: *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*. A cura di Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113. ISBN: 978-3-540-68697-2. DOI: 10.1007/3-540-68697-5_9. URL: http://dx.doi.org/10.1007/3-540-68697-5_9 (cit. a p. 3).

[7] H. Nguyen. «Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic». In: *Journal of Biomedical Science and Engineering* (2013), pp. 551–560 (cit. a p. 26).

[8] Joseph Roth, Xiaoming Liu e Dimitris Metaxas. «On Continuous User Authentication via Typing Behavior». In: 23.10 (ott. 2014), pp. 4611–4624 (cit. alle pp. 1, 27, 35, 36).

[9] Dawn Xiaodong Song, David Wagner e Xuqing Tian. «Timing Analysis of Keystrokes and Timing Attacks on SSH.» In: *USENIX Security Symposium*. Vol. 2001. 2001 (cit. a p. 3).

[10] Eran Tromer e Adi Shamir. *Acoustic cryptanalysis: on nosy people and noisy machines*. 2004. URL: http://cs.tau.ac.il/~tromer/acoustic/ec04rump (cit. a p. 3).

[11] Esra Vural et al. «Shared research dataset to support development of keystroke authentication». In: *Biometrics (IJCB), 2014 IEEE International Joint Conference on*. IEEE. 2014, pp. 1–8 (cit. alle pp. 1, 27, 36).

[12] Peter Wright. *SpyCatcher. The Candid Autobiography of a Senior Intelligence Officer*. 1988 (cit. a p. 3).

[13] YongBin Zhou e DengGuo Feng. *Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing*. zyb@is.iscas.ac.cn 13083 received 27 Oct 2005. 2005. URL: http://eprint.iacr.org/2005/388 (cit. a p. 2).