



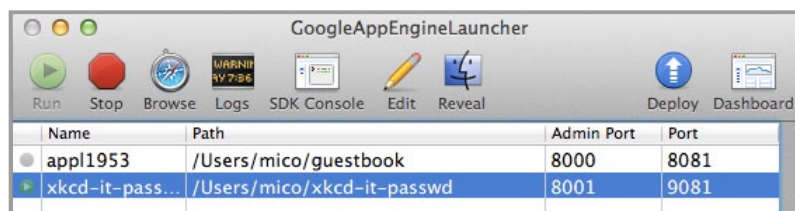
*Codici di accesso facili
da ricordare e molto sicuri.
Come crearli utilizzando
un programma Python,
qualche app e un sito web.
La ricetta passo per passo.*

ARCHITETTURA DI UNA SOLUZIONE

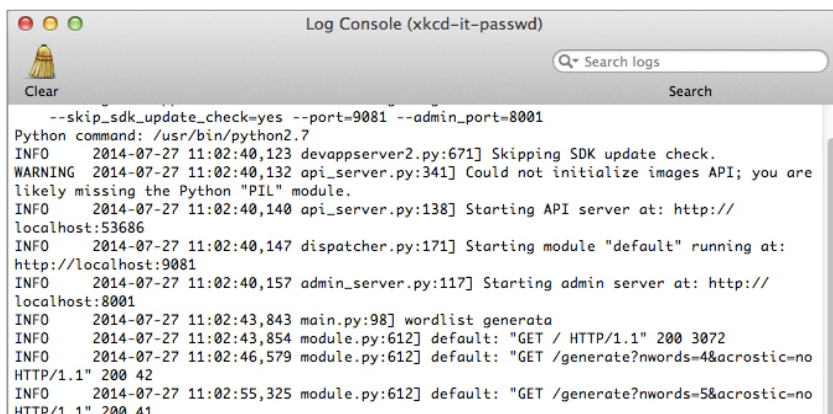
I requisiti sono soprattutto la libertà e la facilità d'uso, poiché non è richiesta una sicurezza di grado militare. Abbiamo

Questi tempi sono il frutto di una stima basata sul tempo di elaborazione per un tentativo di decrittazione e sulla frequenza con cui questi tentativi possono essere attuati. Dobbiamo, fortunatamente,

Noi utenti ci siamo dovuti allenare a creare password complicate, come *P4s5word!* Prendendo una parola, mescolando lettere maiuscole e inserendo dei simboli, per esempio un 4 al posto di una A, un 5 al posto di una S. Con questo sistema, e con l'aggiunta di un segno di punteggiatura, si porta la complessità intorno a 54 bit di entropia (secondo la stima di passwordstrengthcalculator.org). Peccato che la password risultante non sia così



154 PC Professionale > Settembre 2014

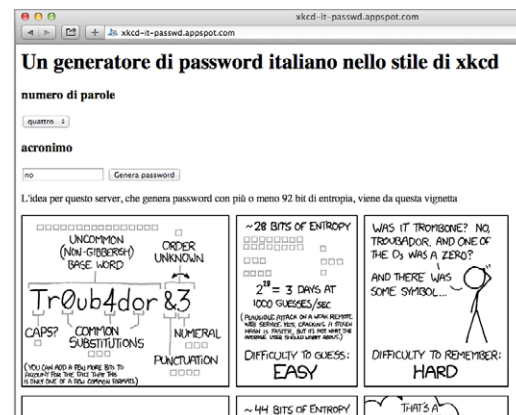


```

--skip_sdk_update_check=yes --port=9081 --admin_port=8001
Python command: /usr/bin/python2.7
INFO 2014-07-27 11:02:40,123 devappserver2.py:671] Skipping SDK update check.
WARNING 2014-07-27 11:02:40,132 api_server.py:341] Could not initialize images API; you are
likely missing the Python "PIL" module.
INFO 2014-07-27 11:02:40,140 api_server.py:138] Starting API server at: http://
localhost:53686
INFO 2014-07-27 11:02:40,147 dispatcher.py:171] Starting module "default" running at:
http://localhost:9081
INFO 2014-07-27 11:02:40,157 admin_server.py:117] Starting admin server at: http://
localhost:8001
INFO 2014-07-27 11:02:43,843 main.py:98] wordlist generata
INFO 2014-07-27 11:02:43,854 module.py:612] default: "GET / HTTP/1.1" 200 3072
INFO 2014-07-27 11:02:46,579 module.py:612] default: "GET /generate?nwords=4&acrostic=no
HTTP/1.1" 200 42
INFO 2014-07-27 11:02:55,325 module.py:612] default: "GET /generate?nwords=5&acrostic=no
HTTP/1.1" 200 41

```

Il log dell'applicazione è visibile localmente con lo strumento **log**, ma anche online con l'interfaccia di amministrazione. L'esame dei log è la prima risorsa utile per trovare gli errori.



La semplice interfaccia utente della nostra applicazione web. Una piccola form e il link alla famosa tavola di Xkcd.

quindi pensato a un archivio di password condiviso nel cloud fra tutti i sistemi che ci può capitare di impiegare, un set che comprende almeno Pc con Windows, Mac con OS X, iPhone e iPad. Una possibile soluzione è un archivio di password condiviso su Dropbox, protetto da una master password piuttosto lunga. Per decrittare l'archivio occorre un'applicazione disponibile su tutti i sistemi, possibilmente open source, meglio ancora se compilata in proprio per avere la garanzia che negli eseguibili non sia nascosto nulla. Naturalmente, ben sappiamo che mettere l'archivio su Dropbox consente tentativi di decrittazione alla massima velocità possibile a chiunque riesca ad avere accesso al file. L'app che abbiamo scelto è KeePassX (<http://www.keepassx.org>), una versione cross platform di KeePass, disponibile per Windows, Linux e OS X. Per i sistemi portatili abbiamo scelto Mini-KeePass, che può essere eseguito su iOS e Android. Risolta la questione di dove archiviare le password, rimane da scegliere come creare password sicure. Per una questione di accessibilità abbiamo optato per una soluzione basata su un sito web, naturalmente gratuito. Deve trattarsi di un sito, che consente di installare applicazioni. Per quanto riguarda il meccanismo di generazione della password, con una ricerca su Github abbiamo trovato due moduli utilizzabili: uno scritto in Perl e uno in Python.

GOOGLE APPENGINE

Abbiamo scelto di usare Python con Google AppEngine (Gae), un application server, di cui abbiamo dato

notizia nell'agosto del 2008, creato in Python da niente meno che dall'autore del linguaggio, Guido van Rossum in persona. Originariamente era solo per programmatori Python, successivamente ha acquistato anche il supporto per Java, Php e, infine, Go (il simpatico linguaggio creato per Google da Ken Thompson).

AppEngine è un ambiente di sviluppo web leggero, agile e agevole da usare. Lo sviluppo locale si basa su un launcher, disponibile per Windows, Linux e OS X, che fornisce l'ambiente operativo e le interfacce di amministrazione. Creare un nuovo progetto è semplice come creare una directory, mentre un clic basta per avviare l'esecuzione locale. L'ambiente mette a disposizione servizi applicativi e includono un database, che non abbiamo usato per questo progetto. La console di gestione del servizio permette di controllare in modo fine le risorse utilizzate.

Un tasto del launcher consente il deploy su appspot.com, il sito Google che ospita le applicazioni create con Gae. I limiti del servizio gratuito sono liberali: 1 Gbyte di database e qualche centinaio di migliaia di accessi al giorno. Con i limiti concessi, occorre porsi il problema di pagare per ospitare il sito solo se il successo è tale da rendere il servizio profittevole.

L'unico limite di Gae è che ci si lega a Google per l'hosting, ma il supporto per il server applicativo Django e la disponibilità dei sorgenti di Gae mitigano il problema per chi volesse trovare un'altra casa all'applicazione.

METTERE INSIEME I PEZZI

Per completare il lavoro, ci mancava una lista di parole italiane. Ne abbiamo trovata una – anzi due – all'indirizzo coding.napolux.com/post/42114980272/ruzzle-lista-parole-italiane. Ci siamo accontentati della lista più corta, dopo avere verificato che non ci fossero problemi di licenza. Presumibilmente, con un po' più di dedizione avremmo potuto trovare qualche lista più ufficiale, magari ricorrendo a qualche dizionario accreditato, ma quella che abbiamo scelto va bene per lo scopo a cui serve. Con tutte le parti sul tavolo, mancava solo un po' di collante per i collegamenti fra l'application server e il

modulo di calcolo e un minimo di interfaccia utente. Tutti i pezzi si sono incastrati insieme in un paio d'ore e non per merito della nostra maestria con Python. L'architettura di Gae è quanto mai lineare: si crea una classe per ogni pagina web, con metodi get e post per gestire i diversi verbi http con cui può essere evocata una risorsa. Nel codice di inizializzazione dell'applicazione, l'associazione fra classi e url del server è fatta con un array associativo. Nel nostro caso, il seguente

```

app = webapp2.WSGIApplication([
    ('/', MainHandler),
    ('/generate', Generate)
],
debug=True)

```

Con questo codice definiamo una homepage, gestita dalla classe MainHandler

AppEngine

L'ambiente di sviluppo è disponibile (anche gratuitamente) per le principali piattaforme operative

e una pagina di generazione, con la uri /generate, affidata alla classe Generate. Prima di parlare di classi, però ci interessava creare la lista di parole in un array globale all'avvio dell'applicazione. Una breve ricerca su Stackoverflow ci ha fatto trovare il consiglio di Van Rossum: scrivere il codice di inizializzazione subito dopo le direttive import e prima della definizione delle classi. Ecco il risultato

```
import webapp2
import logging

from xkcd_password import
generate_wordlist,
generate_xkcdpassword

mywords = generate_
wordlist(wordfile='static/
italian.txt', min_length=5,
max_length=8,)

logging.getLogger().
setLevel(logging.DEBUG)
logging.info("wordlist generata")
```

Come si vede, importiamo dal modulo di supporto xkcd_password, che deve essere nella radice dell'applicazione, le funzioni generate_wordlist e generate_xkcdpassword. La prima legge un file e genera la lista di parole, la seconda produce le password. Per generare la wordlist, chiamata mywords, leggiamo il file italian.txt, che metteremo nella directory static, a fianco al file originale – default.txt – impiegato dal modulo originale. Scegliamo parole comprese fra cinque e otto caratteri. Le ultime due righe di codice configurano un logger, una strumentazione da inserire in un progetto dall'inizio. Ci servirà, perché le possibilità di debug si riducono all'esame dei log e agli stack trace di errore.

L'INTERFACCIA UTENTE

Il codice della pagina radice è semplicissimo:

```
class MainHandler(webapp2.
RequestHandler):
    def get(self):
        self.response.
headers['Content-Type'] = 'text/
html; charset=UTF-8'
        self.response.
write(homepage)
```



Ecco una delle password generate dalla nostra applicazione



Il codice può essere sviluppato con l'editor preferito. Noi abbiamo usato Emacs, il più ubiquo tra i gestori di testo, disponibile su ogni sistema che meriti un editor.

Ricordiamo, per chi conosce poco Python, che un'istruzione che controlla un blocco di codice è marcata con due punti, mentre il blocco di codice è marcato dall'indentazione. Il codice, quindi, è esattamente come sembra senza possibilità di errore.

La variabile homepage contiene il testo della nostra pagina ed è impostata in inizializzazione

```
homepage = """
<html>
<head></head>
<body>
<h1>Un generatore di password
italiano nello stile di xkcd</h1>
<form action="/generate"
method="get">
<p>
<h3>numero di parole</h3>
<select name="nwords">
  <option value="3">tre</option>
  <option value="4">
```

```
selected>quattro</option>
  <option value="5">cinque</
option>
  <option value="6">sei</option>
</select>
</p>
<p>
<h3>acronimo</h3>
<input type="text"
name="acrostic" value="no">
<input type="submit"
value="Genera password">
</form>
...
</html>
"""
```

La possibilità di definire stringhe multilinea con tre apici è una caratteristica utile di Python.

GENERARE LA PASSWORD

La pagina di generazione, quella che è

il bersaglio della form in homepage, è abbastanza semplice. La classe Generate ha un unico metodo, get, quindi non supporta il post. Questo è solo per poter vedere i parametri e poterli alterare dal testo della Url. Nella versione definitiva, si potrebbe aggiungere un metodo post, che rimanda a get.

```
class Generate(webapp2.
RequestHandler):
    def get(self):
```

All'inizio del codice, estraiamo i parametri dalla Url, come di solito nelle applicazioni web

```
nwords = self.request.
get('nwords')
acrostic = self.request.
get('acrostic')
passwd = 'error'
```

Possiamo generare una serie di parole secondo una sigla, per esempio "proclamo canute piramidi rifinir ovatta frughino" che ha come acronimo "pcprof".

La form in home page ha un campo di testo per l'acronimo che è inizializzato a "no". Normalizziamo il valore di acrostic rimuovendo gli spazi e eventuali maiuscole

```
acrostic = acrostic.
strip().lower()
```

Ora, se la stringa è ancora "no", o è vuota, generiamo una password con un numero di parole, che per default è 4

```
if acrostic in ('no', ''):
    passwd = generate_
    xkcdpassword(mywords, n_words=4)
else:
    try:
        passwd =
        generate_xkcdpassword(mywords,
        acrostic=acrostic)
    except:
        passwd = "errore
        nella stringa specificata come
        acronimo"
```

La try serve a segnalare caratteri illegali, come punteggiatura o numeri, nell'acronimo.

Infine, quello che serve per mandare al browser il testo calcolato, con l'encoding giusto

```
self.response.
headers['Content-Type'] = 'text/
```

```
html; charset=UTF-8'
self.response.
write('<h1>' + passwd + '</h1>')
```

Lo stesso encoding è quello che abbiamo specificato per il sorgente Python, mettendo all'inizio un commento appropriato

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Indicare un encoding ricco per il sorgente, consente di usare caratteri nazionali nel testo della home page, che è una stringa nel codice. Una soluzione migliore sarebbe quella di usare template e esternalizzare la struttura dell'interfaccia html, mantenendo nel codice solo le variabili. Prima o poi lo faremo.

Questo è più o meno tutto quello che serve per una funzionalità minima. Pubblicando il nostro sito, abbiamo quello che i nostri lettori possono sperimentare da sé all'indirizzo xkcd-it-passwd.appspot.com. Il codice dell'applicazione è su Github (github.com/michelecos/XKCD-password-generator). Si tratta di un fork del progetto originario per la creazione di password, con le modifiche per l'integrazione con AppEngine e l'uso di una lingua diversa. Il codice originario è all'indirizzo github.com/redacted/

XKCD-password-generator.

CONCLUSIONI E SPUNTI

Abbiamo provato a razionalizzare la gestione di un portafoglio di password, puntando su un servizio web per generarle, spazio cloud per archivarle e un'applicazione multiplatforma per gestire l'archivio crittografato di password, tutto quando usando software il più possibile libero e controllabile.

Il processo per la creazione di una password su misura, è organizzato in due passi: generare una password con l'applicazione web e registrarla nell'archivio condiviso. Si potrebbe fare di meglio con la collaborazione fra browser e servizi vari. Bisognerebbe poter invocare la generazione della password e la sua archiviazione, insieme all'indirizzo del sito per cui è stata generata, direttamente dal browser.

Si tratta del genere di integrazione che è alla portata di Apple o Microsoft, più che di uno sviluppatore singolo. Forse il modo più semplice di arrivarci è creare una estensione di Firefox, ma non è facile pensare una soluzione che funzioni su tutti i sistemi operativi che copriamo con questa, da Android a Windows. Ricordiamo, comunque, che il codice è in Github.

~28 BITS OF ENTROPY
 UNCOMMON (NON-GIBBERISH) BASE WORD
 ORDER UNKNOWN
 CAPS? COMMON SUBSTITUTIONS NUMERICAL PUNCTUATION
 (YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS)
 2²⁸ = 3 DAYS AT 1000 GUESSES/SEC
 (PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT)
 DIFFICULTY TO GUESS: EASY
 DIFFICULTY TO REMEMBER: HARD

~44 BITS OF ENTROPY
 correct horse battery staple
 FOUR RANDOM COMMON WORDS
 2⁴⁴ = 550 YEARS AT 1000 GUESSES/SEC
 DIFFICULTY TO GUESS: HARD
 DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?
 AND THERE WAS SOME SYMBOL...

THAT'S A BATTERY STAPLE.
 CORRECT!

SIAMO ABITUATI A UTILIZZARE PASSWORD DIFFICILI DA RICORDARE MA FACILMENTE DECFRABILI CON LA POTENZA DEI CALCOLI DEI MODERNI SISTEMI DI ELABORAZIONE