SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1220

Generiranje i klasifikacija zaporki korištenjem Markovljevih modela

Matej Jularić

Zagreb, lipanj 2019.

Sadržaj

1. Uvod

Zaporke su od samih početaka računalnih sustava jedan od glavnih načina ograničavanja pristupa podacima. Koriste se za zaštitu samih fizičkih računala, a i za autentikaciju korisnika na svim današnjim internetskim servisima. Za korištenje operacijskog sustava ili internetskih servisa svaki korisnik mora posjedovati zaporku koju u većini slučajeva kreira samostalno ili mu je dodjeljuje administrator sustava. Odabir zaporke se na prvi pogled može smatrati trivijalnim procesom i korisnici često biraju zaporke koje krše barem jedno od sljedećih pravila:

- koristiti različite zaporke za pristup različitim sustavima
- prilikom kreiranja zaporke ne koristiti osobne informacije
- koristiti najdužu moguću zaporku koju sustav dopušta
- redovito mijenjati zaporke

Navedena pravila su preporuka američke organizacije za sigurnost (engl. *Cybersecurity and Infrastructure Security Agency*) [5]. Korisnici najčešće ne koriste različite zaporke za različite sustave koje koriste. U slučaju da napadač probije zaporku ima pristup svim sustavima koje korisnik koristi. Korištenje osobnih informacija u procesu kreiranja zaporke nije nimalo preporučljivo. Napadač može detaljnim pretraživanjem današnjih popularnih društvenih mreža i raznih drugih internetskih usluga skupiti veliku količinu osobnih informacija o svojoj meti i na temelju tih informacija izgraditi specijaliziranu listu zaporki koja će koristiti prilikom napada. Stvaranje najduže moguće zaporke koju sustav dopušta je također uvijek preporučljivo, ali korisnicima može predstavljati problem pamtiti dugačke nizove znakova.
Većina sustava zaporke klasificira po jačini: slabo, srednje i jako. Na temelju klasifikacije koju odrade će dozvoliti korisniku korištenje zaporke. Neka pravila koja znaju koristiti popularni sustavi za klasifikaciju zaporki su sljedeća:

- koristiti barem 8 znakova
- koristiti barem jedno veliko slovo
- koristiti barem jedan broj
- koristiti barem jedan poseban znak

Navedena pravila pretpostavljaju da će napadač koristiti napad silom (engl. *brute force*). Pretpostavka je u većini slučajeva kriva jer u današnje vrijeme postoje puno složeniji načini za probijanje zaporki.
Cilj ovoga rada je predložiti alternativni način za generiranje i klasifikaciju zaporki. U poglavlju *Matematički modeli* obradit će se matematički aparatus koji je korišten kroz rad. Poglavlje *Zaštita zaporki* objašnjava mehanizme zaštite lozinke od napadača, a u poglavlju *Napadi na zaporke* su opisani dobro poznati napadi na lozinke. Implementacija generatora i klasifikatora lozinki je opisana u poglavlju *Generator i klasifikator zaporki*, a rezultati usporedbe implementiranog klasifikatora s postojećim su prikazani u poglavlju *Usporedba klasifikatora*.

2. Matematički modeli

U ovom poglavlju će se opisati vjerojatnosni modeli koji su korišteni prilikom izrade rada. Poglavlje sadrži opise Markovljevih lanaca, skrivenih Markovljevih lanaca i n-gram modela.

2.1. Markovljevi lanci

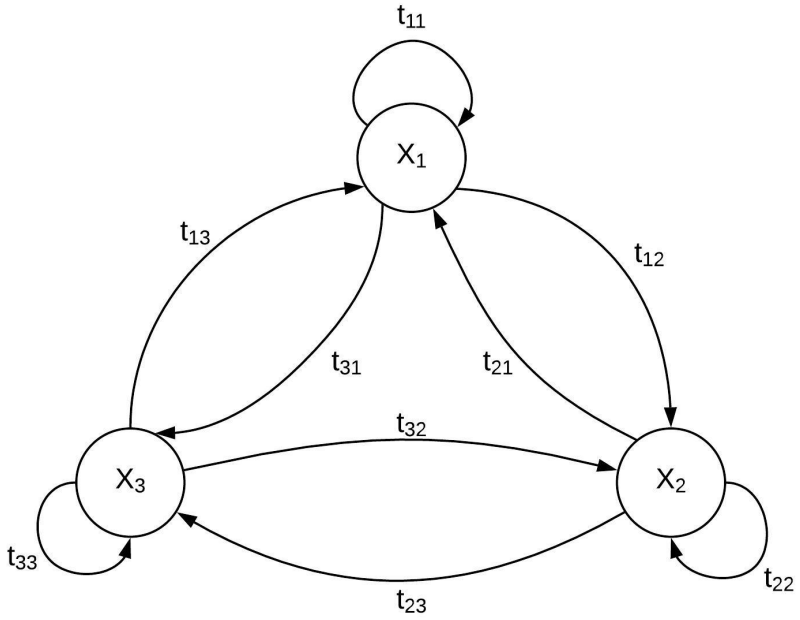
Markovljevi modeli predstavljaju stohastički vjerojatnosni model koji se sastoji od predefiniranih stanja $X = \{X_1, X_2...X_n\}$ i prijelaza između njih. Na slici 2.1 se nalazi primjer Markovljevog lanca sa stanjima X_1, X_2 i X_3. Ovisno o tome događaju li se prijelazi među stanjima u diskretnim ili kontinuiranim vremenskim trenutcima, pričamo o Markovljevom lancu ili Markovljevom procesu. Svi Markovljevi modeli zadovoljavaju Markovljevu pretpostavku.

Markovljeva pretpostavka govori za promatrani slijed stanja $Z_n, Z_{n-1}...Z_1$ da vjerojatnost dolaska u stanje Z_n ovisi samo o stanju u kojem se model trenutno nalazi što je u ovome slučaju Z_{n-1}. U vjerojatnosnom prostoru se navedena pretpostavka može prikazati kao $p(Z_n|Z_{n-1})$. Kako bi Markovljev lanac bio u potpunosti matematički iskazan potrebno je definirati matricu prijelaza $\mathbf{T}$.
newline

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{ij} & \cdots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & \cdots & \cdots & t_{nn} \end{bmatrix} \tag{2.1}$$

Općeniti element t_{ij} matrice $\mathbf{T}$ označuje vjerojatnost prijelaza iz stanja i u stanje j. Matrica prijelaza mora zadovoljavati sljedeću jednakost:

$$\sum_{j=1}^{n} t_{ij} = 1 \tag{2.2}$$



Slika 2.1: Markovljev lanac s 3 stanja

Navedena jednakost govori da suma vjerojatnosti svih prijelaza koji izlaze iz nekog stanja i moraju tvoriti potpuni vjerojatnosni prostor. Ako nije zadovoljen uvjet 2.2 može se dogoditi da programska implementacija Markovljevog lanca ne može izvršiti prijelaz što dovodi do greške u izvršavanju.

2.2. Skriveni Markovljevi modeli

Skriveni Markovljevi modeli (engl. hidden Markov model) se sastoje od dva Markovljeva lanca: "promatranog" i "skrivenog". Stanja koja se nalaze u "skrivenom" lancu uvjetuju u kojem se trenutnom stanju može nalaziti "promatrani" lanac.
U matematičkom smislu skriveni Markovljev model se sastoji od skupa skrivenih stanja $H = \{H_1, H_2...H_n\}$, skupa promatranih stanja $O = \{O_1, O_2...O_n\}$, matrice prijelaza $\mathbf{T}$ i matrice emisija $\mathbf{E}$.

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{ij} & \cdots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & \cdots & \cdots & t_{nn} \end{bmatrix} \tag{2.3}$$

$$\mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & \cdots & e_{1m} \\ e_{21} & e_{ij} & \cdots & e_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1} & \cdots & \cdots & e_{nm} \end{bmatrix} \tag{2.4}$$

U matrici prijelaza **T** su definirane vjerojatnosti prijelaza između stanja "skrivenog" lanca. Kvadratna je matrica dimenzije n gdje n odgovara broju stanja u lancu. U matrici emisija **E** su definirane vjerojatnosti da se određeni događaj ostvari ako se sustav nalazi u nekom stanju "skrivenog" lanca. Pravokutna matrica je dimenzija n x m gdje je n broj stanja "skrivenog" lanca, a m je broj mogućih događaja. Opći element e_{ij} je vjerojatnost događaja j kada je sustav u stanju i "skrivenog" lanca. U nastavku se nalazi pseudokod koji opisuje programsku implementaciju skrivenih Markovljevih modela. Više informacija o Markovljevim lancima i skrivenim Markovljevim modelima se može pronaći u [9].

`Pseudokod za realizaciju skrivenih Markovljevih modela`

Ulazne vrijednosti:

```
t[][] -> matrica prijelaza
e[][] -> matrica emisija
l -> broj stanja koji se želi generirati simulacijom
n -> broj stanja "skrivenog" lanca
m -> broj stanja "promatranog" lanca
trenutno_stanje -> prirodni broj u intervalu [0, n-1]
```

Algoritam:

```
inicijaliziraj brojač i na 0

dok je i < l:
    na temelju vjerojatnosti zapisanih u t[trenutno_stanje][]
    odaberi novo stanje "skrivenog" lanca

    ažuriraj trenutno_stanje

    na temelju vjerojatnosti zapisanih u e[trenutno_stanje][]
    odaberi događaj "promatranog" lanca

    uvećaj brojač i
```

2.3. N-gram model

N-gram model je statistički model korišten u obradi prirodnog jezika u svrhu predviđanja sljedećeg elementa u slijedu. Najčešće se odnosi na predviđanje sljedećeg slova u nizu slova ili sljedeće riječi u nizu riječi. N označava dužinu promatranog slijeda. 1-gram (unigram) označava slijed od jednog elementa, 2-gram (bigram) označava slijed od dva elementa i 3-gram (trigram) označava slijed od tri elementa.
U tablici 2.1 je prikazan rastav rečenice "Odabrali ste slabu lozinku", a u tablici 2.2 je prikazan rastav riječi "lozinka".

Model	Rastav
1-gram	Odabrali, ste, slabu, lozinku
2-gram	Odabrali ste, ste slabu, slabu lozinku
3-gram	Odabrali ste slabu, ste slabu lozinku

Tablica 2.1: Rastav rečenice "Odabrali ste slabu lozinku" na 1-gram, 2-gram i 3-gram

Model	Rastav
1-gram	l, o, z, i, n, k, a
2-gram	lo, oz, zi, in, nk, ka
3-gram	loz, ozi, zin, ink, nka

Tablica 2.2: Rastav riječi "Lozinka" na 1-gram, 2-gram i 3-gram

Pravilnost rastava na 1-gram, 2-gram i 3-gram se može provjeriti sljedećom metodom. Niz koji se rastavlja sastoji se od N elemenata. Broj 1-grama mora biti jednak N, broj 2-grama mora biti $N-1$ i broj 3-grama mora biti $N-2$. Navedena tvrdnja se poklapa s prikazanim primjerima u tablicama 2.1 i 2.2. Stvaranje N-gram modela se svodi na izračunavanje vjerojatnosti da će se neki element e pojaviti nakon određenog slijeda elemenata s, što se može označiti kao $p(e|s)$. U ovome radu se iz praktičnih razloga koristi predviđanje na temelju samo jednog prijašnjeg elementa u slijedu pa se za izračun $p(e|s)$ može koristiti Bayesova formula 2.5.

$$p(e|s) = \frac{p(se)}{p(s)} \tag{2.5}$$

Bitno je zamijetiti da u kontekstu N-gram modela vjerojatnost $p(se)$ iz formule 2.5 ne zadovoljava jednakost $p(es) = p(se)$. Jednakost iz klasične vjerojatnosti u ovome slučaju ne vrijedi jer N-gram model leksikografski razlikuje slijed *es* i *se*. Iz riječi "slabo" i "lozinka" su izračunate vjerojatnosti 1-grama i 2-grama i rezultat je prikazan u tablici 2.3.

Slijed	Vjerojatnost
s	$\frac{1}{12}$
l	$\frac{1}{6}$
b	$\frac{1}{12}$
o	$\frac{1}{6}$
z	$\frac{1}{12}$
i	$\frac{1}{12}$
n	$\frac{1}{12}$
k	$\frac{1}{12}$
a	$\frac{1}{6}$
lo	$\frac{1}{10}$
oz	$\frac{1}{10}$
zi	$\frac{1}{10}$
in	$\frac{1}{10}$
nk	$\frac{1}{10}$
ka	$\frac{1}{10}$
sl	$\frac{1}{10}$
la	$\frac{1}{10}$
ab	$\frac{1}{10}$
bo	$\frac{1}{10}$

Tablica 2.3: Izračunate vjerojatnosti 1-grama i 2-grama za riječ "lozinka"

Kada bi iz toga htjeli vidjeti kolika je vjerojatnost pojavljivanja slova **o** nakon slova **b** izračunali bi to koristeći formulu:

$$p(o|b) = \frac{p(bo)}{p(b)} \tag{2.6}$$

U ovome slučaju $p(o|b)$ iznosi $\frac{3}{5}$. Može se primijetiti da za neke slučajeve u tablici će vjerojatnost ispasti veća od 1 što je vjerojatnosno nemogući rezultat. Takva

greška se može javiti samo u ilustracijskom primjeru jer nije korišten dovoljno velik skup podataka pa statistika ne uspije konvergirati prema pravim vrijednostima. U nastavku je dan pseudokod koji opisuje logiku kojom se računa broj pojavljivanja 1-grama ili 2-grama.

```
Pseudokod za rastavljanje slijeda na 1-gram ili 2-gram
```

Ulazne vrijednosti:

```
    s -> ulazni slijed elemenata
    n -> označava stupanj n-grama (u ovom slučaju 1 ili 2)
    gram[] -> polje 1-grama ili 2-grama nakon podijele
    n -> ukupan broj 1-grama ili 2-grama
```

Algoritam:

```
Podijeli slijed na 1-gram ili 2-gram i spremi u gram[]

n = duljina(gram[])
inicijaliziraj brojač i na 0

dok i < n:
    za svako pojavljivanje 1-grama ili 2-grama uvećaj broj pojavljivanja
    uvećaj brojač i
```

Moguće je zaključiti da je brojanje pojavljivanja na velikom skupu podataka memorijski veoma zahtjevan proces kada se računaju 2-grami jer računalo sve rezultate čuva u RAM-u. U slučaju da računalu ponestane RAM-a program će završiti s greškom. Metode koje su korištene kako bi se izbjegao navedeni slučaj će biti opisane u daljnjim poglavljima. Više informacija o N-gram modelu se može pronaći u [9].

3. Zaštita zaporki

U ovom poglavlju će se opisati način na koji se lozinke spremaju u raznim distribucijama Linux operacijskih sustava. Razmatranja su dosta specifična i određeni pojmovi ne vrijede za druge operacijske sustave, ali određene temeljne ideje su jednake.

3.1. Hash funkcije

Zaporke se nikada ne smiju spremati na operacijskom sustavu u svojem izvornom obliku jer u tom slučaju napadač ima direktan uvid u zaporku ako dobije pristup sustavu. Prilikom stvaranja zaporke operacijski sustav primjenjuje hash funkciju nad zaporkom i sprema ju na neku lokaciju u datotečnom sustavu. Detaljniji uvid o načinu spremanja lozinki na Linux operacijskim sustavima se nalazi u potpoglavlju *Spremanje zaporki na Linux operacijskim sustavima.* Hash funkcija kao ulaz prihvaća slijed znakova neodređene duljine i kao izlaz daje slijed znakova fiksne duljine, koji se naziva hash. Kako bi hash funkcija bila valjana mora zadovoljavati sljedeća svojstva:

1. Za istu ulaznu vrijednost mora uvijek proizvesti istu izlaznu vrijednost.

2. Rezultat hash funkcije ne smije odavati nikakve informacije o ulaznoj vrijednosti. Navedeno svojstvo govori da za dani izlaz iz hash funkcije x je teško odrediti vrijednost v koja zadovoljava $x = hash(v)$.

3. Za dvije različite ulazne vrijednosti ne smije proizvesti jednaki hash. Za vrijednosti v_1 i v_2, gdje je
$$v_1 \neq v_2 \tag{3.1}$$
mora vrijediti da je
$$hash(v_1) \neq hash(v_2) \tag{3.2}$$

Operacijski sustav provjerava valjanost korisničke zaporke tako da na zaporku koju korisnik unese primjeni određenu hash funkciju i usporedi s hash vrijednošću koju ima spremljenu za određenog korisnika, ako izračunata hash vrijednost odgovara spremljenoj tada je korisnik unio ispravnu lozinku i smije se prijaviti u sustav.

Nezadovoljavanje pravila 1. bi u potpunosti onemogućilo korištenje hash funkcija kao mehanizam za spremanje zaporki jer bi se pri svakom hashiranju zaporke dobila druga vrijednost i ne bi bilo moguće provjeriti valjanost zaporke. U slučaju da pravilo 2. nije zadovoljeno napadač bi na temelju hash vrijednosti zaporke mogao zaključiti o kojoj zaporci se radi. Kada pravilo 3. ne bi vrijedilo postoji velika mogućnost da napadač unese krivu zaporku, ali se njezina hash vrijednost poklapa s hash vrijednošću ispravne lozinke i operacijski sustav dozvoli napadaču prijavu iako je unio krivu zaporku. Pravila 2. i 3. je moguće pokazati na primjeru gdje će se ista hash funkcija primijeniti na sljedeće veoma slične zaporke: "lozinka", "lozinka1" i "Lozinka". Koristit će se hash funkcija SHA-256 i rezultati su prikazani u tablici 3.1.

Zaporka	SHA-256 vrijednost
lozinka	6e5e60920c676ace408384d3c2edef02c9387820bcceb326c3a66ac5bbe7e2b1
Lozinka	174e78f57bfa79616defeadfdc43b516e5e214bb7b6cbf1e9a3492e054eb4353
lozinka1	c0a5c994f3995bcb3fd659ac2630b9652ac6d71e2a39c78e6dc0e33da2c86c80

Tablica 3.1: Hash vrijednosti zaporki "lozinka", "Lozinka" i "lozinka1"

Zaporka "lozinka" se razlikuje od zaporke "Lozinka" samo po prvome znaku, ali je očito prema podacima u tablici 3.1 da su izlazi iz hash funkcije različiti. Razlika između zaporki "lozinka" i "lozinka1" je u tome što "lozinka1" ima jedan dodatan znak, ali su i dalje izlazi iz hash funkcija različiti. Sličnost između hasha zaporki "lozinka" i "Lozinka" je 39%, a između "lozinka" i "lozinka1" je 38%. Hashevi nemaju veliki postotak sličnosti s obzirom na to da se radi o veoma sličnim lozinkama i na temelju toga se može zaključiti da je napadaču veoma teško na temelju samoga hasha zaključiti o kojoj zaporci je riječ.

3.2. Spremanje zaporki na Linux operacijskim sustavima

Na Linux operacijskim sustavima se sve informacije vezane uz sigurnost korisnika nalaze u datotekama **/etc/passwd** i **/etc/shadow**.
U datoteci **/etc/passwd** se nalaze informacije o korisnicima sustava i za pristup toj datoteci nisu potrebna administratorska prava. Zapis iz datoteke **/etc/passwd** je sljedećeg formata gdje se znak ":" koristi kao graničnik podataka:

(1):(2):(3):(4):(5):(6):(7)

Opis formata u **/etc/passwd** je sljedeći:

1. Označava korisničko ime
2. Ako je zapis jednak znaku "x" to znači da je korisnička zaporka spremljena u datoteci **/etc/shadow**, a ako nema zapisa to znači da za navedenog korisnika ne postoji zaporka
3. Jedinstveni identifikacijski broj korisnika
4. Jedinstveni identifikacijski broj grupe korisnika kojoj korisnik pripada
5. Dodatne informacije o korisniku (ime i prezime, broj ureda, službeni broj telefona, kućni broj telefona)
6. Označava direktorij u kojem će se korisnik nalaziti nakon prijave na sustav
7. Ljuska operacijskog sustava koju će korisnik koristiti nakon prijave na sustav

Primjer zapisa u **/etc/passwd** za izmišljenog korisnika "linux" je oblika:

linux:x:1001:1001:Linux,c-01,0000000,0000000:/home/linux:/bin/bash

Potencijalni napadač u slučaju da dobije pristup operacijskom sustavu ali ne posjeduje administratorska prava može čitati **/etc/passwd**, ali ne dobiva nikakav uvid u korisničku zaporku. Na Linux operacijskim sustavima moguće je također definirati korisnike koji ne posjeduju zaporku. Korisnici za koje ne postoji zaporka mogu predstavljati potencijalni sigurnosni propust jer napadač uvidom u **/etc/passwd** može saznati kao koji korisnik se može prijaviti bez zaporke. Uvažena praksa je da se takvi korisnici koriste za pozadinske operacije na Linux sustavu i za njih administrator operacijskog sustava onemogućuje prijavu.

Datoteka u koju se spremaju lozinke je **/etc/shadow**. Pristupanje datoteci je po zadanim postavkama Linux operacijskih sustava onemogućeno bez administratorske lozinke. Zapisi u datoteci su sljedećeg oblika gdje se znak ":" koristi kao graničnik podataka:

(1):(2):(3):(4):(5):(6):(7):(8)

Podaci koji se o pojedinom korisniku nalaze **/etc/shadow** su sljedeći:

1. Označava korisničko ime
2. Predstavlja lozinku zapisanu u hash obliku
3. Broj dana kojih je prošlo od zadnje promjene lozinke
4. Broj dana unutar kojih je korisniku dozvoljena promjena lozinke
5. Maksimalan broj dana za koje je lozinka valjana
6. Predstavlja broj dana nakon kojih će korisnik biti obaviješten da mora promijeniti lozinku
7. Predstavlja koliko će dana korisnički račun još biti aktivan nakon isteka lozinke
8. Označava nakon koliko dana će korisnički račun automatski biti deaktiviran

Parametri od 3. do 8. omogućuju administratoru operacijskog sustava jednostavno upravljanje valjanošću korisničke lozinke i mogućnost prisile korisnika za redovitim promjenama lozinke. Primjer zapisa u **/etc/shadow** za proizvoljnog korisnika "linux" je oblika:

linux:1Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:::

Zaporka u **/etc/shadow** je zapisana u sljedećem obliku gdje znak "$" predstavlja graničnik podataka:

(1)(2)$(3)

Opis parametara koji čine zaporku je sljedeći:

1. Vrijednost koja označava koja se hash funkcija koristila prilikom spremanja lozinke. Moguće vrijednosti su 1, 2a, 2y, 5, i 6. U tablici 3.2 se nalazi prikaz koja vrijednost predstavlja koju hash funkciju

2. Označava salt

3. Označava hashiranu zaporku

Može se primijetiti da dobivanje uvida u **/etc/shadow** datoteku ne daje direktan uvid u zaporku već u njezinu hashiranu vrijednost i korišteni hash algoritam. Napadaču se tako znatno otežava probijanje lozinke jer hash vrijednost nije moguće koristiti kao zaporku.

Vrijednost	Hash algoritam
1	MD5
2a	Blowfish
2y	Blowfish
5	SHA-256
6	SHA-512

Tablica 3.2: Hash algoritmi za spremanje zaporki na Linux operacijskim sustavima

Detaljnije informacije o temi i implementacijske detalje moguće je pronaći u [1] i [2].

4. Napadi na zaporke

U ovom poglavlju će se opisati popularni napadi na zaporke kao što su napad silom (engl. brute force) i napad rječnikom (engl. dictionary attack). Napraviti će detaljna analiza napada silom na temelju javno dostupnih podataka i pokazati će se sigurnosne metode koje se mogu koristiti kako bi se smanjila efikasnost navedenih napada.

4.1. Napad silom

Napad silom je jedan od najpoznatijih napada na zaporke. Metoda se sastoji od iscrpnog isprobavanja svake moguće kombinacije za određeni skup znakova zadane duljine. Metoda se sastoji od hashiranja svih kombinacija i uspoređivanja s hashom tražene zaporke dok se ne naiđe na poklapanje njihovih hash vrijednosti. Kada bi se prilikom napada koristila sva slova, brojevi i posebni znakovi zagarantirana je 100 % uspješnost.
Napraviti će se analiza koliko je vremena potrebno za probijanje zaporke koristeći napad grubom silom ako se koristi računalo iz 2007. godine koje može probati 7×10^6 zaporki po sekundi. Navedeni podaci su dobiveni iz [12]. Zatim će se promotriti koliko je za probijanje istih zaporki potrebno koristeći grafičku karticu koja može pogađati 10.3×10^9 zaporki po sekundi. Podaci su preuzeti iz [10]. Analiza se radi za najgori slučaj kada se smatra da napadač mora probati sve kombinacije kako bi pogodio lozinku. Smatra se također da napadač pokušava probiti veliki broj lozinki, a ne samo jednu. U slučaju da pokušava probiti zaporku samo jednog korisnika onda mu je isplativo čekati čak i nekoliko dana.
U tablici 4.1 je prikazana su vremena potrebna za probijanje zaporke u slučaju kada se za lozinku koriste samo brojevi. Ukupan broj znamenki je 10, a ukupan broj kombinacija je tada 10^l gdje je l duljina zaporke.
Može se primijetiti da računalo koje može probati 7.1×10^6 kombinacija po sekundi nije u mogućnosti probiti zaporke veće od 13 znakova u razumnom vremenu, a

Duljina zaporke	$7.1 \times 10^6 zaporki/s$	$10.3 \times 10^9 zaporki/s$
5	14 ms	9 μs
10	23 min	1 s
12	39 h	2 min
14	163 dana	3 h
15	5 godina	27 h
17	446 godina	112 dana
20	446616 godina	307 godina

Tablica 4.1: Vrijeme potrebno za probijanje zaporki sastavljenih od brojeva

za računalo koje može probati 10.3×10^9 kombinacija po sekundi velika vremenska ograničenja se javljaju kod zaporki duljine 17 znakova. U tablici [referenca] prikazana su vremena potrebna za probijanje zaporke kada su prilikom kreiranja zaporke korištena mala slova engleske abecede. U engleskoj abecedi postoji 26 slova što znači da je ukupan broj kombinacija 26^l gdje je l duljina zaporke.
U tablici 4.2 prikazana su vremena potrebna za probijanje zaporke kada su prilikom kreiranja zaporke korištena mala slova engleske abecede. Engleska abeceda sadrži 26 slova što znači da je ukupan broj kombinacija 26^l gdje je l duljina zaporke.

Duljina zaporke	$7.1 \times 10^6 zaporki/s$	$10.3 \times 10^9 zaporki/s$
5	2 s	1 ms
8	8 h	20 s
9	9 dana	9 min
10	230 dana	4 h
11	16 godina	4 dana
12	426 godina	107 dana
13	11081 godina	8 godina

Tablica 4.2: Vrijeme potrebno za probijanje zaporki sastavljenih od malih slova engleske abecede

Tablica pokazuje da računalo koje može isprobati 7.1×10^6 kombinacija po sekundi nije u sposobnosti probiti zaporke veće od 8 znakova u razumnom vremenu, a računalo koje može probati 10.3×10^9 kombinacija po sekundi ne može u kraćem vremensku razdoblju probiti zaporke duže od 9 znakova. Vidljiva su drastična

poboljšanja prilikom korištenja zaporki koja koriste mala slova engleske abecede u odnosu na korištenje samo brojeva. U tablici 4.3 prikazan je slučaj kada se koriste mala i velika slova engleske abecede, brojevi i specijalni ascii znakovi. Broj simbola u navedenom skupu je 94. Ukupan broj mogućih kombinacija je tada 94^l gdje je l duljina zaporke.

Duljina zaporke	$7.1 \times 10^6 zaporki/s$	$10.3 \times 10^9 zaporki/s$
5	17 min	1 s
6	27 h	1 min
7	106 dana	1 h
8	27 godina	7 dana
9	2559 godina	2 godine

Tablica 4.3: Vrijeme potrebno za probijanje zaporki sastavljenih od malih i velika slova, brojeva i specijalnih znakova

Očito je da računalo koje probava 7.3×10^6 kombinacija u sekundi ne može probiti lozinke veće od 5 znakova u kratkom vremenskom intervalu, a za računalo koje obavlja 10.3×10^9 kombinacija u sekundi granice iskoristivosti se naziru za zaporke veće od 7 znakova. Preporuka stručnjaka iz Kaspersky laboratorija je da se koriste lozinke duže od 10 znakova koje sadrže mala i velika slova abecede, brojeve i specijalne ascii znakove [11]. U tablici 4.4 su prikazane duljine lozinki za koje pojedino računalo nije u mogućnosti provesti napad silom u razumnom vremenu.

Skup simbola	$7.1 \times 10^6 zaporki/s$	$10.3 \times 10^9 zaporki/s$
brojevi	11	14
mala slova	8	9
mala slova, velika slova, brojevi i specijalni znakovi	5	7

Tablica 4.4: Prikaz duljine lozinke nakon koje pojedino računalo ne provodi napad silom u razumnom vremenu

4.2. Napad rječnikom

Napadač konstruira popis popularnih lozinki koji se naziva rječnik i koristi ga prilikom pogađanja zaporki. Rječnik je neobrađena tekstualna datoteka koja se sastoji od jedne riječi ili fraze po retku. Najčešće se konstruiraju na temelju lozinki koje su dobivene iz prijašnjih napada na Twitter, Dropbox, RockYou itd. Svaki redak u rječniku predstavlja potencijalnog kandidata koji se može podudarati s korisničkom lozinkom koju napadač pokušava probiti. Razlika između napada rječnikom i napada grubom silom je da napad rječnikom koristi popis zaporki koje se potencijalno mogu podudarati s korisničkom zaporkom umjesto da napadač isprobava sve moguće kombinacije. Prilikom slaganja rječnika lozinki bitno je paziti da ne obuhvaća previše kombinacija jer se približuje tome da postane napad grubom silom i gubi svoju učinkovitost. Napad rječnikom je jedan od efektivnijih napada na lozinke ako se koriste optimizirani rječnici. Pseudokod koji opisuje način na koji je moguće napraviti napad rječnikom je sljedeći:

`Pseudokod za izvršavanje napada rječnikom`

Ulazne vrijednosti:

```
r -> datoteka koja sadrži rječnik lozinki
h -> hash vrijednost korisničke lozinke koju se pokušava probiti
l -> redak iz r
```

Algoritam:

```
dok ima zapisa u r:
    ako je hash(l) = h:
        pronađena je tražena lozinka
        prekini izvršavanje programa i lozinka je pronađena
    inače:
        l = pročitaj zapis iz r
```

Napad koji kombinira napad silom i napad rječnikom se naziva hibridni napad. Za napad se također koristi rječnik, ali se svakom zapisu unutar rječnika dodaju sve moguće kombinacije niza znakova određene duljine. Kada bi u rječniku postojao zapis "zaporka", a želi se na svaku potencijalnu zaporku dodati na početak sve kombinacije iz skupa $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ duljine 3 dobili bi sve kombinacije od "000zaporka" do "999zaporka". Povećanje vremena potrebnog

za izvršavanje navedenog napada se povećava ovisno o broju kombinacija koji se nadodaje na svaki zapis u rječniku.
Prilikom izvršavanja navedenih napada potrebno je svaki puta računati hash vrijednosti zaporki u rječniku. Napadači dosta često unaprijed generiraju velike tablice koje sadrže hashirane lozinke koje se nazivaju rainbow tablice. Koristeći takvu metodu napada nije potrebno prilikom svakog napada ponovno računati hash vrijednosti već se rainbow tablica pretražuje za hash vrijednost koja se pokušava probiti. Na stranicama od projekta pod nazivom RainbowCrack postoji velika količina unaprijed stvorenih rainbow tablica [14].

4.3. Metode obrane od napada na zaporke

Kako bi se onemogućilo korištenje rainbow tablica administratori računalnih sustava primjenjuju metode salt, pepper i iterativno hashiranje.
Iterativno hashiranje znači da se na korisničku zaporku ne primjenjuje hash funkcija samo jedanput već n puta. Na taj se način izbjegava mogućnost korištenja rainbow tablica jer bi napadači unaprijed morali znati koliko je puta primjenjena hash funkcija nad zaporkom kako bi mogli izračunati rainbow tablice. Navedena metoda usporava napadača prilikom korištenja napada grubom silom ili rječnikom jer je u mogućnosti probati puno manje lozinki u zadanom vremenu jer troši puno vremena provodeći hashiranje n puta. Iterativno hashiranje usporava autentikaciju korisnika na sustav jer umjesto da operaciju hashiranja provedemo jedanput provodimo ju n puta. Navedeno sigurnosno poboljšanje može imati negativan utjecaj na performanse sustava koji mora opsluživati velike količine korisnika.
Salt je slučajan slijed znakova koji se dodaje zaporkama prije hashiranja. Korisniku ne otežava pamćenje lozinke jer se salt sprema zajedno s hashem korisničke lozinke i njegovu vrijednost koristi samo proces koji obavlja autentikaciju korisnika na sustavu. Onemogućuje korištenje rainbow tablica jer napadač mora za svaku lozinku koju pokušava probiti generirati posebnu rainbow tablicu koristeći njezin salt. Prilikom korištenja salta bitno je ne koristiti istu vrijednost za sve korisnike. Napadač tada ne može koristiti postojeće rainbow tablice, ali zbog činjenice da sve zaporke koje pokušava probiti dijele isti salt ne mora za svaku zaporku generirati posebnu tablicu već mora generirati samo jednu rainbow tablicu. Preporučuje se ne koristiti prekratke vrijednosti za salt jer je napadač tada u mogućnosti izgenirariti sve kombinacije salta i za svaku od njih ponovno generirati rainbow tablicu. Pseudokod koji prikazuje napad rječnikom na lozinku

koja je hashirana koristeći salt je sljedeći:

Pseudokod za izvršavanje napada rječnikom u slučaju da je korišten salt

Ulazne vrijednosti:

```
r -> datoteka koja sadrži rječnik lozinki
h -> hash vrijednost korisničke lozinke koju se pokušava probiti
s -> salt vrijednost korisničke lozinke
l -> redak iz r
```

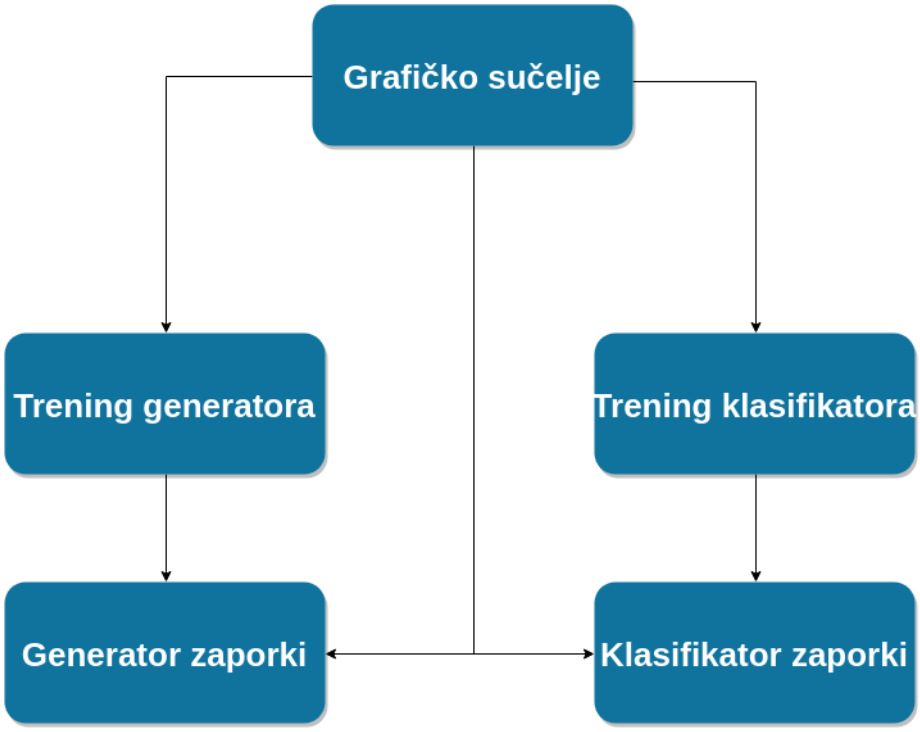
Algoritam:

```
dok ima zapisa u r:
    ako je hash(s + l) = h:
        pronađena je tražena lozinka
        prekini izvršavanje programa i lozinka je pronađena
    inače:
        l = pročitaj zapis iz r
```

Pepper je slijed znakova koji se kao i salt dodaje zaporkama prije hashiranja. Razlika između peppera i salta je u činjenici da se pepper ne sprema zajedno s hashiranom korisničkom lozinkom. Spremanje pepper vrijednosti odvojeno od hashirane lozinke onemogućuje napadaču da provede napad na lozinke u slučaju da posjeduje samo hashirane vrijednosti. Korištenje peppera pruža dodatan sloj sigurnosti dokle god je njegova vrijednost nepoznata. Napdaču je moguće otežati napad ako se pepper i salt spajaju s lozinkom prije hashiranja na netrivijalan način.

5. Generator i klasifikator zaporki

U poglavlju će se opisati izrađena programska implementacija generatora i klasifikatora zaporki. Arhitektura programske implementacije je prikazana na slici 5.1. Kako se rješenje ne bi moralo koristiti samo kroz ljusku operacijskog sustava implementirano je i grafičko sučelje. Grafičko sučelje upravlja modulima koji se koriste za treniranje generatora i klasifikatora kao i za upravljanje njihovim funkcionalnostima. Pruža sloj apstrakcije i jednostavno korištenje implementiranih funkcionalnosti. Implementacija je napravljena u programskom jeziku Python 3 [18] koristeći programsku knjižicu za obradu prirodnog jezika nltk (engl. Natural language toolkit) [13] i programsku knjižicu za izradu grafičkih sučelja tkinter [19].



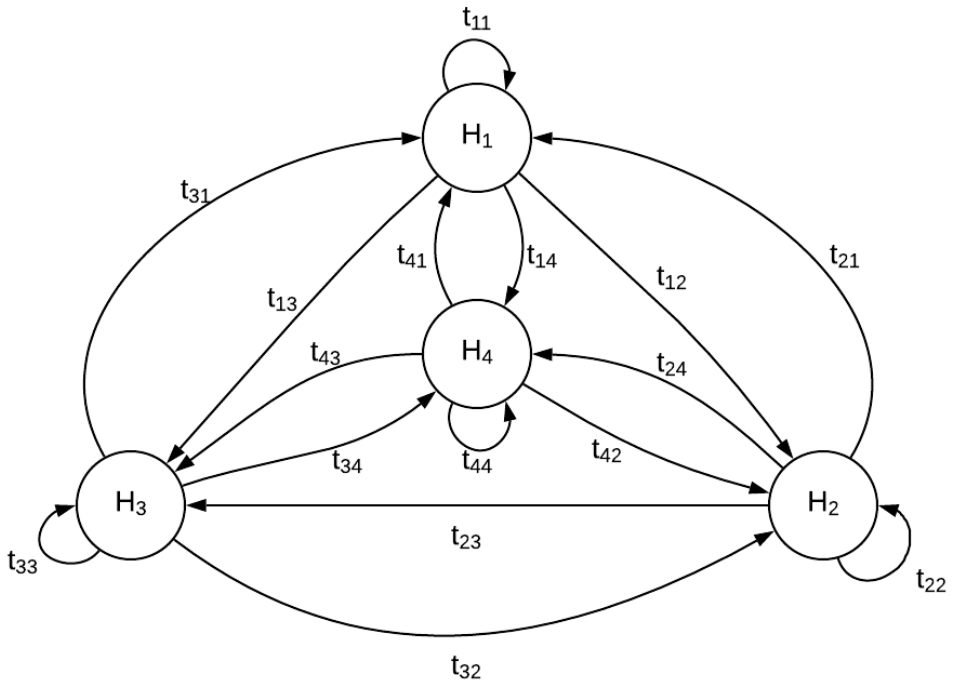
Slika 5.1: Arhitektura programske implementacije

5.1. Generator zaporki

Modul za generiranje zaporki koristi Markovljeve modele. Omogućuje stvaranje dvije vrste zaporki: čitljive i nečitljive. Nečitljive zaporke se sastoje od znakova koji ne tvore riječ koja pripada nekom jeziku i predstavljaju najveću razinu zaštite. Nažalost takve zaporke su korisnicima teško pamtljive i često zahtijevaju zapisivanje na operacijski sustav korisnika što predstavlja sigurnosni propust. Čitljive zaporke koriste riječi iz nekog jezika i smatraju se manje sigurnima za korištenje, ali ih korisnici mogu lako zapamtiti.

5.1.1. Generator nečitljivih zaporki

Implementacija koristi skrivene Markovljeve lance i omogućuje kreiranje sigurnih nečitljivih zaporki. Korišteni skriveni Markovljev lanac se sastoji od četiri skrivena stanja $H = \{H_1, H_2, H_3, H_4\}$, gdje stanje H_1 predstavlja mala slova engleske abecede, H_2 velika slova engleske abecede, H_3 znamenke i H_4 specijalne ascii znakove. Na slici 5.2 prikazana su skrivena stanja Markovljevog lanca korištenog u implementaciji i prijelazi među njima. Promatrana stanja ovise o skrivenom sta-



Slika 5.2: Skrivena stanja Markovljevog lanca korištenog u implementaciji

nju u kojem se slučajni proces nalazi u tom trenutku i također ih postoji četiri,

a označavaju se s $O = \{O_1, O_2, O_3, O_4\}$. Svako od promatranih stanja u skupu O je također skup koji ima sljedeći opis:

- Stanje O_1 se javlja za vrijeme skrivenog stanja H_1 i odgovara svim malim slovima engleske abecede $\{a, b, c, ..., z\}$
- Stanje O_2 se javlja za vrijeme skrivenog stanja H_2 i odgovara svim velikim slovima engleske abecede $\{A, B, C, ..., Z\}$
- Stanje O_3 se javlja za vrijeme skrivenog stanja H_3 i odgovara znamenkama $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Stanje O_4 se javlja za vrijeme skrivenog stanja H_4 i odgovara specijalnim ascii znakovima $\{!, ?, *, ...\}$

Matrica prijelaza $\mathbf{T}$ je prikazana matematičkim izrazom 5.1. Može se primijetiti da je vjerojatnost za svaki prijelaz jednaka odnosno iznosi $p(t_{ij}) = \frac{1}{4}$. U skrivenom stanju se aktivira promatrano stanje vezano za njega. Postavljeno je da su svi elementi u svakom promatranom stanju također jednako vjerojatni, što bi značilo da je u stanju O_1 vjerojatnost pojave bilo kojeg elementa $\frac{1}{26}$, u stanju O_2 je $\frac{1}{26}$, u stanju O_3 je $\frac{1}{10}$ i u stanju O_4 je $\frac{1}{32}$. Jednolika razdioba je namjerno odabrana za skriveni i promatrani lanac kako bi se postigao potpuno nasumičan odabir elemenata koji će se koristiti prilikom stvaranja zaporke i time povećala težina njezinog probijanja.

$$\mathbf{T} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \tag{5.1}$$

U tablici 5.1 su prikazane zaporke raznih duljina dobivene generatorom. Može se primijetiti da zaporke stvarno ne predstavljaju riječ niti jednog jezika i mogućnost njihovog pamćenja je manja s porastom njihove duljine.

Duljina zaporke	Zaporka
4	*HA7
6	0F3r?F
8	3[@ Bf;H
10	zLsVVv[$O]

Tablica 5.1: Prikaz zaporki raznih duljina dobivenih generatorom nečitljivh zaporki

U nastavku je prikazana implementacija generatora nečitljivih lozinki u programskom jeziku Python 3. Implementacija je napravljena samo s programskim knjižicama koje dolaze zajedno s Python programskim jezikom. Korištena je objektno orijentirana programska paradigma. Klasa sadrži konstruktor tj. **___init___** funkciju i **generate_password** funkciju koja stvara zaporku željene duljine.

```
import random
import string

class AlphanumericMarkov:

    lower_case_letters = string.ascii_lowercase
    upper_case_letters = string.ascii_uppercase
    punctuation = string.punctuation
    digits = string.digits
    states = [0, 1, 2, 3]

    #state 0 = lower_case
    #state 1 = upper_case
    #state 2 = digits
    #state 3 = special_characters

    transition_matrix = [[1/4, 1/4, 1/4, 1/4],
                         [1/4, 1/4, 1/4, 1/4],
                         [1/4, 1/4, 1/4, 1/4],
                         [1/4, 1/4, 1/4, 1/4]]

    def __init__(self, initial_state=None):
        secure_random = random.SystemRandom()

        if initial_state is None:
            self.state = secure_random.choice(self.states)
            return
        self.state = initial_state
```

```
def generate_password(self, length):
    password = ''

    for i in range(length):
        secure_random = random.SystemRandom()
        new_state = secure_random.choices(self.states,
        self.transition_matrix[self.state],
        k=1)[0]

        if new_state == 0:
            password = password +
            secure_random.choice(self.lower_case_letters)
        if new_state == 1:
            password = password +
            secure_random.choice(self.upper_case_letters)
        if new_state == 2:
            password = password +
            secure_random.choice(self.digits)
        if new_state == 3:
            password = password +
            secure_random.choice(self.punctuation)

        self.state = new_state
    return password
```

5.1.2. Generator čitljivih zaporki

Modul za generiranje čitljivih zaporki se sastoji od dva podmodula: podmodul za treniranje generatora i podmodul za generiranje zaporki. Podmodul za treniranje generatora kao ulaz mora primiti smisleni tekst kao što su npr. književna djela. U implementaciji je korišteno književno djelo Moby Dick koje dolazi uz programsku knjižicu nltk [13]. Na temelju ulaznog teksta se računa broj pojavljivanja pojedine riječi i broj pojavljivanja svih riječi koje slijede tu riječ, ali samo na razini rečenice. Navedena ideja će biti jasnija kada se promotri sljedeći primjer. U slučaju kada bi ulazni tekst bio sljedeći:

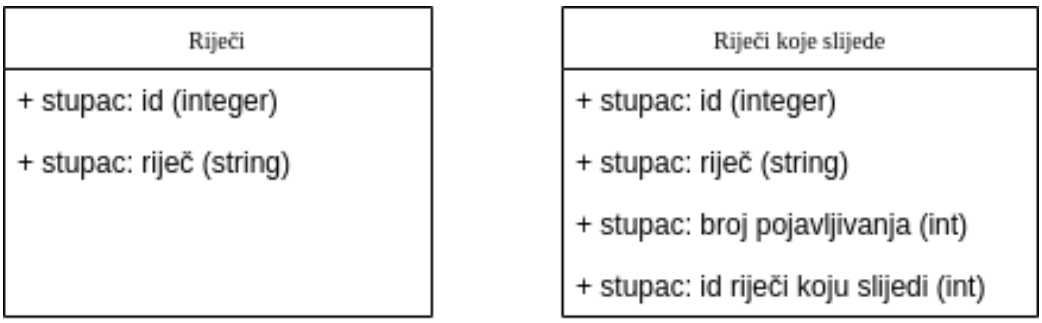
"Unesite neku zaporku. Unijeli ste neku besmislicu."

Izlaz koji bi proizveo podmodul za treniranje generatora je sljedeći:

Riječ	Riječi koje slijede	Broj pojavljivanja riječi koje slijede
Unesite	neku	1
Unijeli	ste	1
ste	neku	1
neku	zaporku, besmislicu	1, 1

Tablica 5.2: Prikaz izlaza iz podmodula za treniranje generatora

Navedeni podaci se spremaju u SQLite bazu podataka [6] kako se treniranje na velikim tekstovima ne bi moralo raditi više puta jer je vremenski veoma zahtjevno. Ovakva implementacija podmodula za treniranje omogućuje dodatno treniranje postojeće statistike u bazi podataka s novim tekstovima. Baza podataka sadrži tablice prikazane na slici 5.3.



Slika 5.3: Tablice korištene za treniranje generatora

Prilikom spremanja svake riječi u tablicu *Riječi* ona dobiva jedinstveni identifikator odnosno *id*. Sve riječi koje slijede tu riječ će se spremiti u tablicu *Riječi*

koje slijede gdje će također dobiti jedinstveni identifikator *id*. Spremit će se njihova vrijednost, broj pojavljivanja i *id* riječi koju slijede. U nastavku se nalazi programski kod napisan u programskom jeziku Pythonu 3 koji opisuje kako se izvodi proces treninga. Korištena je biblioteka za obradu prirodnog jezika nltk, a funkcija **store_to_db** se brine o spremanju podataka u tablice sa slike 5.3.

```
import nltk

def create_statistic(text_file):
    txt_file = open(text_file)
    text = txt_file.read()

    sentences = nltk.tokenize.sent_tokenize(text)

    main_iters = int(len(sentences))

    for i in range(main_iters):
        bigrams = nltk.ngrams(sentences[i].split(), 2)

        for gram in bigrams:
            word = gram[0]
            consequent_word = gram[1]
            store_to_db(word, consequent_word)
```

Generator čitljivih zaporki koristi bazu podataka koju podmodul za treniranje ispuni podacima. Zaporke se stvaraju tako da se prva riječ odabere potpuno nasumično iz tablice *Riječi* i na temelju odabrane riječi se iz tablice *Riječi koje slijede* nasumično odabire sljedeća riječ i postupak se ponavlja dok se ne dosegne željena duljina zaporke. Ulazna vrijednost generatora zaporki je željena duljina zaporke koja odgovara broju riječi od kojih će se zaporka sastojati. U nastavku je prikazana implementacija generatora čitljivih zaporki u programskom jeziku Python 3. Funkcija **get_word_from_db(word)** pronalazi u tablici *Riječi* traženu riječ ako postoji dok funkcija **get_word_by_id(id)** pronalazi riječ u tablici *Riječi* na temelju *id* polja. Metoda **get_consequent_word(id)** na temelju vrijednosti *id* određene riječi vraća nasumično odabranu riječ koja slijedi tu riječ.

```
import random

def generate_password(length):

    password = ""
    choice = "␣"
    generated = 0

    while generated < length:

        try:
            word = get_word_from_db(choice)
            id = word.id
            choice = get_consequent_word(id)
            password = password + "␣" + choice
            generated = generated + 1

        except DoesNotExist:
            random_id = self.secure_random.randint(1, 27260)
            word = get_word_by_id(random_id)
            choice = get_consequent_word(random_id)
            if generated == 0:
                password = password + word.word
                password = password + "␣" + choice
                generated = generated + 2
            else:
                password = password + "␣" + word.word
                generated = generated + 1
                if generated >= length:
                    break

                password = password + "␣" + choice
                generated = generated + 1

    return password
```

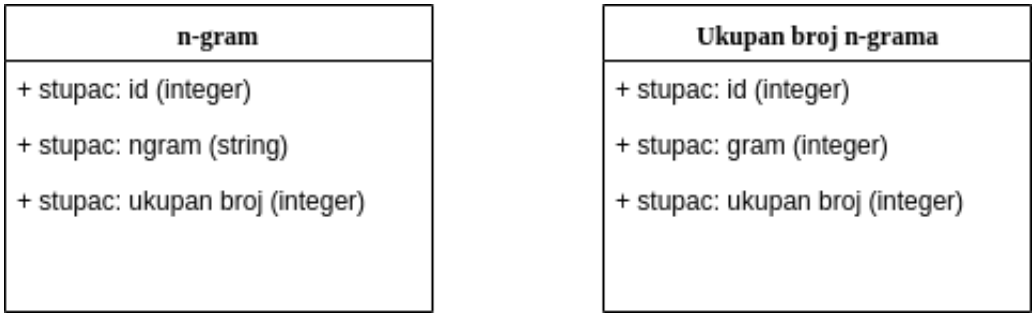
U tablici 5.3 su prikazane zaporke dobivene generatorom čitljivih zaporki i može se primijetiti da zaporke sadrže riječi engleskog jezika koje slijedno u određenim slučajevima mogu imati smisao. Zaporke stvorene generatorom čitljivih zaporki su puno nesigurnije nego zaporke stvorene generatorom nečitljivih zaporki, ali korištenjem dovoljno dugačke čitljive zaporke se također može postići zadovoljavajuća razina sigurnosti.

Duljina zaporke	Zaporka
2	gripping at
3	habits they say
4	outcries and persuasions, all
5	courteously, therein certainly an extinct
6	Bunyan, the mizentop for answer to
7	mermaid, to this ball; I have given
8	moods I don't know but old Bowditch in
9	farthingale being picked up again, perhaps the German's aside

Tablica 5.3: Prikaz zaporki dobivenih generatorom čitljivih lozinki

5.2. Klasifikator zaporki

Modul koji obavlja klasifikaciju zaporki koristi n-gram model tj. unigram i bigram. Sastoji se od podmodula za treniranje i klasifikaciju. Ulazna vrijednost u podmodul za treniranje je tekstualna datoteka koja u svakome retku sadrži po jednu zaporku. U ovome radu je za treniranje korištena jedna od najpoznatijih datoteka sa zaporkama koja je nastala kao posljedica napada na web servis RockYou [16]. Treniranje se provodi tako da se svaka zaporka podijeli na unigrame i pobroji se koliko kojih unigrama postoji unutar cijele tekstualne datoteke. Identična stvar se provede i za bigrame. Rezultat treniranja se zapisuje u SQLite bazu podataka [6]. Baza podataka se koristi kako se ne bi moralo prilikom svakog pokretanja klasifikatora provoditi treniranje, koje je vremenski veoma zahtjevno. Baza podataka sadrži tablice prikazane na slici 5.4. Tablica *n-gram* se sastoji od



Slika 5.4: Tablice korištene za treniranje klasifikatora

polja *id*, *ngram* i *ukupan broj*. U polje *ngram* se zapisuje string vrijednost unigrama odnosno bigrama, a u polje *ukupan broj* se zapisuje koliko se puta unutar obrađenog skupa podataka pojavio pojedini unigram odnosno bigram. Tablica *Ukupan broj n-grama* se sastoji od polja *id*, *gram*, *ukupan broj*. Podmodul za treniranje klasifikatora u polje *gram* upisuje vrijednost 1 ili 2, ovisno o tome radi li se o unigramu ili bigramu. U polje *ukupan broj* upisuje ukupan broj unigrama ili bigrama koji se nalaze u analiziranoj tekstualnoj datoteci. U slučaju da se za sve unigrame ili bigrame u tablici *n-gram* zbroje vrijednosti iz polja *ukupan broj* dobila bi se vrijednost koja odgovara polju *ukupan broj* u tablici *Ukupan broj n-grama*. Polje *id* je jedinstveni identifikator retka unutar tablice i baza podataka ga automatski stvara za svaki novi unos u tablici. U nastavku je prikazana programska implementacija koja na temelju tekstualne datoteke populira tablice *n-gram* i *Ukupan broj n-grama*. Funkcija **store_to_db** nije prikazana ovdje jer ona obavlja spremanje u bazu podataka koji joj se proslijede. Navedenu funkciju

je moguće implementirati na više načina i pri tome koristiti različite baze podataka. Prilikom učitavanja velikih tekstualnih datoteka bi se moglo na računalima s manje RAM-a dogoditi da program prestane raditi jer bi bilo potrebno puno podataka čuvati u RAM-u. Kako bi program radio na svim računalima tekstualna datoteka se ne obrađuje cijela odjedanput već se obradi manji dio koji se zatim spremi u bazu podataka te se nakon otpuštanja dotada zauzetih resursa nastavlja dalje s obradom tekstualne datoteke.

```
import os
import nltk

def create_statistic(password_file):
        pass_file = open(password_file)
        lines = pass_file.readlines()
        maximum = int(len(lines) / 30)

        main_iters = int(len(lines) / maximum)
        k = 0

        for i in range(main_iters):
            parsed_n_grams = []

            for j in range(k, (i + 1) * maximum):
                list_line = list(lines[j].strip())

                ngrams = nltk.ngrams(list_line, n)

                for gram in ngrams:
                    appender = ""
                    for g in gram:
                        appender = appender + g
                    parsed_n_grams.append(appender)

            store_to_db(nltk.FreqDist(parsed_n_grams), n)
            k = (i + 1) * maximum
```

```
parsed_n_grams = []

for j in range(main_iters * maximum, len(lines)):
    list_line = list(lines[j].strip())

    ngrams = nltk.ngrams(list_line, n)

    for gram in ngrams:
        appender = ""
        for g in gram:
            appender = appender + g
        parsed_n_grams.append(appender)

store_to_db(nltk.FreqDist(parsed_n_grams), n)
```

Podmodul za klasifikaciju kao ulaz prima zaporku u tekstualnom obliku, a kao izlaz daje vrijednost koja govori kolika je vjerojatnost probijanja zaporke. Na primjeru zaporke "lozinka" će se pokazati na koji način algoritam radi. Vjerojatnost probijanja zaporke "lozinka" se može dobiti izrazom 5.2.

$$p(lozinka) = p(l) \times p(o|l) \times p(z|o) \times p(i|z) \times p(n|i) \times p(k|n) \times p(a|k) \tag{5.2}$$

Može se primijetiti da nisu direktno poznate potrebne uvjetne vjerojatnosti, ali se njihove vrijednosti mogu dobiti koristeći izraz 5.3.

$$\begin{aligned} p(o|l) &= \frac{p(lo)}{p(l)} \\ p(z|o) &= \frac{p(oz)}{p(o)} \\ p(i|z) &= \frac{p(zi)}{p(z)} \\ p(n|i) &= \frac{p(in)}{p(i)} \\ p(k|n) &= \frac{p(nk)}{p(n)} \\ p(a|k) &= \frac{p(ka)}{p(k)} \end{aligned} \tag{5.3}$$

Vjerojatnosti $p(l)$, $p(o)$, $p(z)$, $p(i)$, $p(n)$, $p(k)$ i $p(a)$ se dobivaju tako da se u tablici *n-gram* potraži zapis za svaki od znakova i pribavi se polje *ukupan broj*. Zatim je potrebno podijeliti ukupan broj pojedinog znaka s vrijednošću *ukupan*

broj (vrijednost se dohvaća za unigram) iz tablice *Ukupan broj n-grama*. Navedeni postupak je identičan za vjerojatnosti $p(lo)$, $p(oz)$, $p(zi)$, $p(in)$, $p(nk)$ i $p(ka)$, ali je potrebno dohvatiti vrijednost *ukupan broj* za bigrame iz tablice *Ukupan broj n-grama*. Nakon izračunavanja vjerojatnosti $p(lozinka)$ potrebno je korisniku predočiti jačinu zaporke koju pokušava testirati. Vjerojatnost probijanja nije metrika koja će krajnjem korisniku imati veliko značenje pa je potrebno konstruirati skalu za ocjenu jačine lozinke. U većini slučajeva će vjerojatnost probijanja zaporke biti veoma malena brojka u intervalu $[0, 1]$. Iz tog razloga se uvodi logaritamska skala. Izraz 5.2 se tada pretvara u sljedeći izraz:

$$\begin{aligned}\log_2[p(lozinka)] = \log_2[p(l)] + \log_2[p(o|l)] + \log_2[p(z|o)] + \log_2[p(i|z)] \\ + \log_2[p(n|i)] + \log_2[p(k|n)] + \log_2[p(a|k)]\end{aligned} \tag{5.4}$$

Logaritmiranje izraza 5.2 omogućuje da se računa vjerojatnost probijanja i za iznimno dugačke zaporke. Računanje umnoška puno veoma malenih brojeva može uzrokovati probijanje najmanje pozitivne vrijednosti koja se može zapisati u IEEE 754 formatu. Klasifikator ocjenjuje jačinu zaporke po sljedećoj skali:

1. Slaba zaporka - Izlaz iz klasifikatora je broj u intervalu $[0, 60]$
2. Srednje jaka zaporka - Izlaz iz klasifikatora je broj u intervalu $< 60, 90]$
3. Jaka zaporka - Izlaz iz klasifikatora je broj u intervalu $< 90, +\infty >$

Interval za slabu zaporku je dobiven tako da je klasifikator ocijenio tekstualnu datoteku koja sadrži 500 najgorih lozinki. Datoteku je sažela zajednica foruma za sigurnost u internetu pod nazivom Skull Security [15]. Klasifikator je svaku od lozinki u navedenoj tekstualnoj datoteci ocijenio s vrijednošću manjom od 50. Odabran je za desni rub intervala broj 60 kako bi postojala određena zalihost u odnosu na izmjerene vrijednosti. Interval za srednje jaku zaporku je odabran da bude polovica duljine intervala za slabu zaporku, a interval za jaku zaporku se nastavlja od vrijednosti 90 pa nadalje. U nastavku je prikazana programska implementacija koja računa vjerojatnost probijanja zaporke u logaritamskoj skali. Prilikom računanja vjerojatnosti probijanja zaporke potrebno je koristiti podatke iz tablica *n-gram* i *Ukupan broj n-grama* što znatno usporava izvršavanje proračuna u slučaju kada se izvršava na većem broju zaporki. Navedeno se izbjeglo tako da se baza podataka prilikom prvog pokretanja učita u RAM (što je moguće jer baza podataka nije velike veličine) i programska implementacija učitava sve potrebno direktno iz RAM-a umjesto s datotečnog sustava.

```
import nltk
import math

def calculate_probability(password_text):
        ngrams = { hash tablica koja sadrzi sve unigrame
                    i bigrame iz tablice n-gram}

        unigram_count = ukupan broj unigrama
        iz tablice Ukupan broj n-grama

        bigram_count = ukupan broj bigrama
        iz tablice Ukupan broj n-grama

        password_text_list = list(password_text)
        unigrams = nltk.ngrams(password_text_list, 1)
        bigrams = nltk.ngrams(password_text_list, 2)

        unigrams_list = []

        for gram in unigrams:
            unigrams_list.append(gram[0])

        bigrams_list = []

        for gram in bigrams:
            bigrams_list.append("" + gram[0] + gram[1])

        probability_result = 0

        first_character_count = ngrams[unigrams_list[0]]

        probability_result = probability_result +
            math.log2(first_character_count / unigram_count)


        for i in range(len(bigrams_list)):
```

```
                unigram_count = self.ngrams[unigrams_list[i]]
                bigram_count = 1
                try:
                    bigram_count = self.ngrams[bigrams_list[i]]
                except Exception:
                    pass
                print(unigrams_list[i])
                probability_unigram = unigram_count / unigram_count
                probability_bigram = bigram_count / bigram_count

                probability_result = probability_result +
                    math.log2(probability_bigram /
                    probability_unigram)

        return abs(probability_result)
```

Klasifikator također zahtijeva unos korisničkog imena za koje se testira određena zaporka. U slučaju da zaporka sadrži korisničko ime u sebi automatski se odbacuje i klasificira kao slaba zaporka. Integriran je s online bazom podataka *Have I Been Pwned?* koja sadrži trenutno najveću zbirku probijenih zaporki u svijetu [8]. Zaporka se šalje na *Have I Been Pwned?* API koji odgovara koliko puta je zaporka sadržana u njegovoj bazi. Kako *Have I Been Pwned?* ne bi saznao koja se zaporka pokušava testirati potrebno je zaporku hashirati SHA-1 algoritmom i API-ju poslati samo prvih 5 znakova hasha. API će u odgovoru poslati sve lozinke koje započinju s tih 5 znakova i njihov broj pojavljivanja u bazi podataka. Nakon primanja odgovora od API-ja potrebno je provjeriti odgovara li ijedan od hasheva iz odgovara zaporki koja se testira. U slučaju da se hash podudara s ijednim hashem iz API-jevog odgovora korisniku je preporučeno da odmah mijenja zaporku, a ako ne postoji podudaranje zaporka ne postoji u *Have I Been Pwned?* bazi podataka i korisnik može biti siguran da *Have I Been Pwned?* ne zna njegovu zaporku već samo prvih 5 znakova SHA-1 hasha. U nastavku je prikazana implementacija integracije s *Have I Been Pwned?* API-jem. Ideja za klasifikator je došla iz [3]

```
import requests
import hashlib


class BeenPwned:

    def __init__(self, password):
        self.password = password

    def is_pwned(self):
        sha1_password = hashlib.sha1(self.password.encode())
        hash = sha1_password.hexdigest()
        hash = hash.upper()
        hash_list = list(hash)

        first_5_chars = ""

        for i in range(5):
            first_5_chars = first_5_chars + hash_list[i]

        data = requests.get(
            url="https://api.pwnedpasswords.com/range/" +
            first_5_chars)

        lines = data.text.split("\n")

        for line in lines:
            h, num_of_occurence = line.split(":")
            if (first_5_chars + h) == hash:
                return int(num_of_occurence)
        return 0
```

6. Usporedba klasifikatora

Napravit će se usporedba implementiranog klasifikatora s komercijalno dostupnim klasifikatorima od Microsofta, Twittera, Yahooa. Komercijalni klasifikatori su dostupni kroz PARS alat (engl. *Password Analysis and Research System*) razvijen od Georgia instituta za tehnologiju [17]. Usporedba se provodila nad tekstualnom datotekom koja sadrži 375853 zaporki, a potiče iz proboja web sjedišta tvrtke Ashley Madison [7]. Pokazat će se za koliko zaporki se implementirani klasifikator slaže s komercijalnim klasifikatorima i za koliko slučajeva se ne slaže. Tablica 6.1 prikazuje rezultate usporedbe implementiranog i Microsoftovog klasifikatora. Implementirani klasifikator je precijenio jačinu zaporke u slučaje-

Ukupan broj zaporki	375853
Broj poklapanja	249949
Broj nepoklapanja	125904
Implementirani precijenio zaporku kao jaku	189
Implementirani precijenio zaporku kao srednje jaku	1885
Komercijalni precijenio zaporku kao srednje jaku	113782
Komercijalni precijenio zaporku kao jaku	9553

Tablica 6.1: Prikaz rezultata usporedbe Microsoftovog i implementiranog klasifikatora

vima kada su veće duljine zaporki. Navedena situacija se događa jer se množi puno malenih vjerojatnosti koje povećavaju izlaznu vrijednost klasifikatora. Većina krivo klasificiranih zaporki se nalaze u drugim izvorima probijenih zaporki koje Microsoftov klasifikator očigledno koristi i zato ih automatski ocjenjuje kao slabe zaporke. Iznimno velik broj zaporki je Microsoftov klasifikator procijenio kao jake, a implementirani klasifikator kao slabe. Primjer nekih zaporki koje je Microsoftov klasifikator ocijenio kao jake su : "zoom123", "zonk69", "yeayea" i "3369dead". Rezultati usporedbe implementiranog i Twitterovog klasifikatora su prikazani u tablici 6.2. Detaljnim proučavanjem rezultata može se vidjeti da su

Ukupan broj zaporki	375853
Broj poklapanja	224327
Broj nepoklapanja	151526
Implementirani precijenio zaporku kao jaku	149
Implementirani precijenio zaporku kao srednje jaku	197
Komercijalni precijenio zaporku kao srednje jaku	150432
Komercijalni precijenio zaporku kao jaku	748

Tablica 6.2: Prikaz rezultata usporedbe Twitterovog i implementiranog klasifikatora

rezultati iznimno slični za Twitterov i Microsoftov klasifikator. Razlika se javlja prilikom usporedbe s Yahooovim klasifikatorom čiji su rezultati prikazani u Tablici 6.3. Yahooov klasifikator je jako puno zaporki ocijenio kao jake, a implementirani klasifikator ih je većinu ocijenio kao slabe i dio kao srednje jake. Velika većina zaporki su malene duljine i sastoje se ili samo od malih slova engleske abecede ili od malih slova engleske abecede i znamenki. Dodatno će se

Ukupan broj zaporki	375853
Broj poklapanja	182997
Broj nepoklapanja	192856
Implementirani precijenio zaporku kao jaku	23
Implementirani precijenio zaporku kao srednje jaku	445
Komercijalni precijenio zaporku kao srednje jaku	0
Komercijalni precijenio zaporku kao jaku	192388

Tablica 6.3: Prikaz rezultata usporedbe Yahoovog i implementiranog klasifikatora

napraviti usporedba implementiranog klasifikatora s popularnim online alatom za testiranje jačine zaporke The Password Meter [4]. Stvorit će se više zaporki kojima će baza biti riječ "diplomski" i promatrat će se kako se oba klasifikatora ponašaju u odnosu na promjene. Rezultati su prikazani u tablici 6.4. Konačna verzija zaporke "Diplomski1234!" je zaporka koju bi većina web sjedišta prihvatilo kao zaporku. Gmail servis uspješno prihvaća navedenu zaporku. Može se zaključiti da zaporke "Diplomski1234" i "Diplomski1234!" nisu veoma kompliciranе i napadač bi uz poznavanje korijena zaporke "diplomski" vrlo vjerojatno probao sve navedene kombinacije. Može se zaključiti da se implementirani klasifikator u većini slučajeva ponaša puno bolje nego komercijalni klasifikatori, ali jedan od

Zaporka	Implementirani	The Password Meter
diplomski	slaba	slaba
Diplomski	slaba	slaba
Diplomski1234	slaba	jaka
Diplomski1234!	srednje	jaka

Tablica 6.4: Prikazani rezultata usporedbe The Password Meter i implementiranog klasifikatora

nedostatak mu je što je znatno sporiji.

7. Zaključak

Nepoštivanje bilo koje od preporuka iz poglavlju *3. Zaštita zaporki* može dovesti do gubljenja korisničke hashirane zaporke, a pošto je pokazano da komercijalni klasifikatori jačine zaporki nemaju stroge kriterije prilikom provjere zaporki napadač koristeći neke od napada iz poglavlja *4. Napadi na zaporke* može veoma lako probiti korisničku zaporku. Komercijalni klasifikatori zaporki nemaju stroge kriterije jer popularna web sjedišta većinom žele što više olakšati prijavu na svoj sustav. Simplifikacija prijave najčešće uključuje olakšavanje stvaranja korisničke zaporke tako da se ne zahtijeva njezina prevelika kompleksnost. Implementirani klasifikator je pokazivao znatno bolje sposobnosti procjene jačine zaporke u većini slučajeva, ali veoma loše klasificira dugačke zaporke. Velika mana u odnosu na komercijalne klasifikatore je brzina izvršavanja što je veoma bitno za sustave koji imaju puno korisničkih registracija. Prilikom korištenja popularnih web servisa dio odgovornosti vezan uz sigurnost zaporke ovisi o korisniku sustava. Korisnik prilikom stvaranja zaporke može koristiti generator nečitljivih zaporki koji će pružiti najveću sigurnost, ali je veoma teško pamtiti nečitljive zaporke veće duljine. U slučaju da se koristi generator čitljivih zaporki moguće je puno lakše zapamtiti zaporku jer je građena od riječi koje pripadaju nekome jeziku, ali nije najsigurniji način generiranja zaporki.

Literatura

[1] Canonical. Passwd file, 2019. URL http://manpages.ubuntu.com/manpages/xenial/man5/passwd.5.html.

[2] Canonical. Shadow file, 2019. URL http://manpages.ubuntu.com/manpages/xenial/man5/shadow.5.html.

[3] Claude Castelluccia, Markus Dürmuth, i Daniele Perito. Adaptive password-strength meters from markov models. 02 2012.

[4] Password Strength Checker. Password meter, 2019. URL http://www.passwordmeter.com/.

[5] CISA. Choosing and protecting passwords, 2009. URL https://www.us-cert.gov/ncas/tips/ST04-002.

[6] SQLite Consortium. Sqlite documentation, 2019. URL https://www.sqlite.org/docs.html.

[7] Jason Haddix Daniel Miessler i g0tmi1k. Ashley madison password list, 2019. URL https://github.com/danielmiessler/SecLists/blob/master/Passwords/Leaked-Databases/Ashley-Madison.txt.

[8] Troy Hunt. Have i been pwned?, 2018. URL https://haveibeenpwned.com/API/v2.

[9] Daniel Jurafsky i James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2008.

[10] George Khalil. Password security thirty five years later, 2019. URL https://www.sans.org/reading-room/whitepapers/basics/password-security-thirty-five-years-35592.

[11] Kaspersky Lab. What's a brute force attack?, 2019. URL https://www.kaspersky.com/resource-center/definitions/brute-force-attack.

[12] Mandylion Labs. Brute force calculator, 2007. URL https://tmedweb.tulane.edu/content_open/bfcalc.php.

[13] Natural language toolkit. Nltk documentation, 2019. URL https://www.nltk.org/.

[14] RainbowCrack. List of rainbow tables, 2019. URL http://project-rainbowcrack.com/table.htm.

[15] Skull Security. 500 worst passwords, 2019.. URL https://wiki.skullsecurity.org/Passwords.

[16] Skull Security. Rockyou password list, 2019.. URL https://wiki.skullsecurity.org/Passwords.

[17] Raheem Beyah Changchang Liu Ting Wang Shouling Ji, Shukun Yang i Wei-Han Lee. Password analysis and research system, 2019. URL http://www.pars.gatech.edu/.

[18] Python software foundation. Python website, 2019. URL https://www.python.org/.

[19] Tkinter. Tkinter documentation, 2019. URL https://wiki.python.org/moin/TkInter.

Generiranje i klasifikacija zaporki korištenjem Markovljevih modela

Sažetak

Lozinke su u današnje doba glavni način za ograničavanje pristupa resursima na mreži. Korisnici računalnih usluga prilikom odabira lozinke rijetko razmišljaju o razini sigurnosti koju pruža. Komercijalni alati za klasifikaciju lozinki rade na principu empirijski zadanih pravila koja će korisnika dosta često navesti da stvori teško pamtljivu lozinku ili u većini slučajeva slabu lozinku. Klasifikacija koju takvi alati provode je temeljena na ideji da će napadač raditi brute force napad iako postoje puno sofisticiranije metode za probijanje lozinki koje koriste leksikografska svojstva jezika prilikom pogađanja lozinke i smanjuju vrijeme potrebno za probijanje lozinke. Cilj ovoga rada je izrada rješenja koje će omogućiti korisniku stvaranje sigurne i lako pamtljive lozinke. Predložit će se sustav za klasifikaciju koji utvrđuje vjerojatnost da napadač pogodi lozinku ako koristi statistički napad Markovljevim modelom.

Ključne riječi: zaporke, sigurnost, Markovljevi modeli, klasifikator zaporki, generator zaporki

Generation and classification of passwords using Markov models

Abstract

Passwords are nowadays the main way to restrict access to online content. When choosing a password, computer users rarely think about the level of security they provide. Commercial password classification tools will often force the user to create a password that is hard to remember or in most cases a weak password. Such tools do classification based on the idea that an attacker will perform a brute force attack, although there are much more sophisticated methods for cracking passwords. The goal of this paper is to create a solution that will allow the user to create a secure and easy to remember password. A classification system will be proposed to determine the likelihood of an attacker guessing a password if using a statistical attack based on Markov models.

Keywords: passwords, security, Markov models, password classification, password generator