Metoda analize otpornosti i kvalitete lozinki zaštićenih pomoću kriptografskih jednosmjernih funkcija

Kišasondi, Tonimir

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Tonimir Kišasondi

Metoda analize otpornosti i kvalitete lozinki zaštićenih pomoću kriptografskih jednosmjernih funkcija

DOKTORSKI RAD

Varaždin, 2014.

PODACI O DOKTORSKOM RADU

I. AUTOR

Ime i prezime	Tonimir Kišasondi
Datum i mjesto rođenja	24.10.1984, Varaždin, Hrvatska
Naziv fakulteta i datum diplomiranja na VII/I stupnju	Fakultet organizacije i informatike, Varaždin, 22.11.2007.
Naziv fakulteta i datum diplomiranja na VII/II stupnju	
Sadašnje zaposlenje	Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin

II. DOKTORSKI RAD

Naslov	Metoda analize otpornosti i kvalitete lozinki zaštićenih pomoću kriptografskih jednosmjernih funkcija
Broj stranica, slika, tabela, priloga, bibliografskih podataka	116 stranica, 14 tablica, 2 slike i 75 bibliografskih podataka
Znanstveno područje i polje iz kojeg je postignut doktorat znanosti	Društvene znanosti, informacijske znanosti
Mentori ili voditelji rada	prof.dr.sc. Mirko Čubrilo prof.dr.sc. Željko Hutinski
Fakultet na kojem je obranjen doktorski rad	Fakultet organizacije i informatike
Oznaka i redni broj rada	109

III. OCJENA I OBRANA

Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena tema	18.9.2012.
Datum predaje rada	2.12.2013.
Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena pozitivna ocjena rada	14.1.2014.
Sastav povjerenstva koje je rad ocijenilo	izv.prof.dr.sc. Marin Golub doc.dr.sc. Ivan Magdalenić doc.dr.sc. Tomaž Klobučar prof.dr.sc. Mirko Čubrilo prof.dr.sc. Željko Hutinski
Datum obrane doktorskog rada	30.1.2014.
Sastav povjerenstva pred kojim je rad obranjen	izv.prof.dr.sc. Marin Golub doc.dr.sc. Ivan Magdalenić doc.dr.sc. Tomaž Klobučar prof.dr.sc. Mirko Čubrilo prof.dr.sc. Željko Hutinski
Datum promocije	

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Tonimir Kišasondi

Metoda analize otpornosti i kvalitete lozinki zaštićenih pomoću kriptografskih jednosmjernih funkcija

DOKTORSKI RAD

Mentor(i):

prof.dr.sc. Mirko Čubrilo

prof.dr.sc. Željko Hutinski

Varaždin, 2014.

University of Zagreb

Faculty of Organization and Informatics

Tonimir Kišasondi

Method for analyzing password strength for passwords protected with cryptographic hash functions

DOCTORAL THESIS

Varaždin, 2014.

Sažetak

Najpopularnija metoda autentikacije korisnika u informacijskim sustavima jest primjena lozinki. Svaka lozinka koju napadač ili maliciozni kod može jednostavno pogoditi slaba je i sustav čini ranjivim. Administratori sustava i analitičari za sigurnost koriste metode pronalaženja slabih lozinki kako bi mogli proaktivno štititi informacijski sustav. Prilikom analize sustava u kojem se želi otkriti slabe lozinke, uvijek je poželjnija metoda koja uz manji utrošak vremena i računalnih resursa pronalazi veći broj slabih lozinki. Ovaj rad opisuje istraživački postupak kojim su razvijene tri metode za pronalaženje slabih lozinki. Prva metoda omogućava pronalaženje slabih, industrijski postavljenih lozinki, na uređajima koji su dostupni na mrežama ili Internetu, i stvara listu industrijskih i čestih lozinki iz raznih *online* repozitorija. Druga metoda koristi samostalno odabrane ključne riječi pomoću kojih generira listu riječi koristeći tražilicu Google. Ključne riječi mogu biti imena osoba, nadimci, imena tvrtki i slično, što omogućava da liste riječi sadrže pojmove povezane s ključnim riječima. Treća metoda omogućava pronalaženje slabih lozinki koje su zaštićene jednosmjernom kriptografskom funkcijom i temelji se na modeliranju korisničkog ponašanja prigodom odabira lozinke. Istraživanje korisničkog ponašanja provedeno je kroz razvoj i oblikovanje nove metode koja predstavlja novi algoritam strojnog učenja, koji stvara pravila koja opisuju kako korisnici kreiraju svoje lozinke. Metoda strojnog učenja primijenjena je na listi lozinki koje su bile dostupne za potrebe istraživanja. Pravila koja su pronađena uz pomoć treće metode mogu biti primijenjena u skoro svim popularnijim alatima za testiranje snage lozinki. Primjena takvih pravila pokazala se bržom od danas referentnih metoda za testiranje snage lozinki, pri čemu je metoda opisana u sklopu ovog rada otkrila veći broj slabih lozinki u kraćem vremenskom intervalu u odnosu na referentnu metodu.

Ključne riječi: sigurnost, lozinka, autentikacija, jednosmjerne funkcije, *hash*-funkcije, kriptografija, kriptoanaliza.

Prošireni sažetak

Istraživanje u sklopu ovog rada vezano je uz pronalaženje lozinki koje se mogu jednostavno pogoditi od strane napadača ili malicioznog koda. Zbog mogućnosti neovlaštenog pristupa sustavu, takve lozinke čine ga ranjivim. U slučaju lozinki, razlikujemo *online* analizu kojom se nastoji otkriti slabe lozinke u nekom sustavu koji je dostupan putem mreže, ili javno na internetu. Veliki broj uređaja koji su aktivni u produkcijskim uvjetima nema promijenjene početne lozinke i ostavlja se dostupnim putem Interneta, gdje automatizirani skeneri pokušavaju kombinacije industrijskih lozinki protiv takvih sustava. Ukoliko lozinke na javno dostupnim sustavima nisu promijenjene, one predstavljaju veliki rizik za naše sustave, jer su upravo ti sustavi centralne točke za prijenos podataka. U slučaju *online* analize, želimo provjeriti što manji broj industrijskih lozinki, kako ne bismo opterećivali mrežu i sustave koji su i inače pod konstantnim napadom od strane automatiziranih alata ili malicioznog koda iz javnog interneta.

Drugi oblik provjere snage lozinki jest *offline* analiza, gdje se provjerava lista lozinki koja se koristi u nekom sustavu za autentikaciju. U takvoj listi svaka lozinka je zaštićena kriptografskom jednosmjernom funkcijom. Kod *offline* analize ograničeni smo raspoloživim računalnim resursima, gdje želimo pronaći što veći broj lozinki u odnosu na uložene računalne resurse i vrijeme trajanja analize.

Prema najboljoj praksi, opisanoj u normama ISO/IEC 27002:2013 [32], NIST 800-115 [63] te NIST 800-63-1 [8], lozinke koje se koriste u sustavima moraju biti odgovarajuće složenosti i zaštićene na način da ne čuvamo izvorni tekst lozinke i s njim ne radimo. Pored toga, one moraju biti zaštićene nekim mehanizmom, poput jednosmjernih kriptografskih funkcija. Iz tog razloga potrebne su metode koje mogu provjeravati kvalitetu već zaštićenih lozinki.

Inspiracija za provođenje ovog istraživanja jest pretpostavka da korisnici ne biraju lozinke nasumično, nego da imaju svoj sustav ili obrazac ponašanja prema kojemu to čine. Ponašanje korisnika i malicioznog koda modeliramo pomoću modela koji prikazuju pojedine klase slabih lozinka, za koje su također razvijeni i alati koji omogućuju njihovu primjenu. Modele smo podijelili u tri glavne kategorije:

1. Model M1, koji prikazuje klasu napada koji koriste automatizirani skeneri ranjivosti ili razni oblici malicioznog koda protiv sustava koji su dostupni na javnom internetu.
2. Model M2, koji prikazuje klasu lozinki koje su vezane uz osobne podatke ili ključne riječi koje je korisnik mogao odabrati. Taj se model koristi za *offline* analizu lozinki.
3. Model M3, koji prikazuje način ponašanja korisnika, tj. način na koji korisnici stvaraju svoje lozinke; koje riječi, brojeve i znakove koriste, te prilagodbe koje koriste prilikom stvaranja svojih lozinki. I taj se model koristi za *offline* analizu lozinki.

Svaki pojedinačni model je razvijen i za njega je provedeno istraživanje tijekom kojega su polučeni sljedeći rezultati:

1. Za model M1 razvijena je metoda generiranja i ažuriranja liste lozinki iz više nestrukturiranih repozitorija koje održavaju konzultanti za sigurnost ili hakeri. To omogućava da konzultanti za sigurnost ili administratori sustava mogu jednostavno prikupiti liste lozinki iz navedenih izvora, te generirati jedinstvenu listu, temeljenu na učestalosti pojave pojedine lozinke u svim repozitorijima. Pomoću generirane liste nije potrebno ručno provjeravati moguću pojavu lozinke za uređaj ili sustav koji je predmet analize, već se omogućava automatizacija provjere i masovno skeniranje za slabim lozinkama.
2. Za model M2 razvijena je metoda koja omogućava korištenje internetske tražilice Google za generiranje liste riječi prema ključnim riječima koje unese korisnik.
    a. Primjenom metode kreiran je bazni korpus riječi sa popularnim riječima, koji se može koristiti za analize u modelu M3 ili za korištenje u drugim alatima.
    b. Razvijena je implementacija i generiranje rječnika iz korpusa *online* enciklopedije *Wikipedija*. Navedeni pristup omogućava kreiranje velikih korpusa lozinki, uz povećanu potrebu za računalnim resursima. Zbog tog ograničenja, razvijena je metoda koja omogućava generiranje liste riječi pomoću tražilice Google.
3. Za model M3 razvijena je nova metoda strojnog učenja, koju je autor nazvao *metodom sita*, koja omogućava klasifikaciju lozinke i stvaranje pravila koja opisuju način na koji je korisnik odabrao svoju lozinku.
    a. Metoda sita predstavlja novi algoritam i pristup klasifikaciji sadržaja, primijenjen na problem klasifikacije lozinki.
    b. Primjenom metode sita nad listom lozinki za učenje, razvijen je skup pravila koji opisuje ponašanje korisnika i način na koji oni odabiru lozinke.

c. Uz klasifikaciju lozinki provedeno je prikupljanje riječi te nizova brojeva i znakova koji su korisnici koristili prilikom stvaranja svojih lozinki, pomoću kojih je dopunjena lista lozinki stvorena pomoću metode iz modela M2.

4. Model M3 i pripadajuća pravila koja su otkrivena pomoću klasifikacije, mogu se koristiti kroz razvijeni alat nazvan "unhash" koji omogućava primjenu otkrivenih pravila u skoro svim popularnim alatima za pronalaženje slabih lozinki koji mogu koristiti standardni ulaz *stdin*. Za usporedbu brzine i performansi modela M3 predložena su četiri načina primjene pravila, koja su uspoređena s danas referentnom implementacijom, dostupnom u sklopu alata *John the Ripper* [14].

Provjerom modela M1 utvrđeno je da prikupljene liste mogu otkriti industrijske lozinke na skoro svim uređajima kojima je autor imao pristup i da su bili u njegovom vlasništvu. Implementacijom modela M2 i M3, i mjerenjem performansi u odnosu na danas referentnu implementaciju za pronalaženje slabih lozinki, dostupnu u sklopu alata *John the Ripper* [14], utvrđeno je da nova predložena implementacija pronalazi veći broj lozinki u kraćem vremenskom intervalu.

Ključne riječi: sigurnost, lozinka, autentikacija, jednosmjerne funkcije, hash funkcije, kriptografija, kriptoanaliza.

Extended abstract:

The essence of this research is related to the detection of weak passwords that can be guessed by an attacker or malware. Such passwords are a security vulnerability because they can provide a possibility of unauthorized access to a system. For password analysis, we can distinguish between online analysis where we try to identify weak passwords that are active on an authentication system that is accessible from the local network or from the public Internet. A large number of devices are vulnerable, since they are set with factory default passwords and are accessible via the Internet, where a large number of malware and automated scanners try to guess the password on those systems with malicious intent. If the passwords are not changed from their default values they are a major risk to the information system, because network facing systems are critical systems for data transmission. In the case of online analysis, we want to check a small number of passwords against a network service, so we don't affect the performance of systems that are already under constant attack by automated tools or malicious code from the Internet.

Another type of analysis is the offline analysis where we analyze a list of passwords that are used by an authentication system where each password is protected with a one-way hash function. In this case, our analysis is limited by our processing resources, where we want to spend as little resources as possible and detect as many weak passwords in the shortest timeframe possible.

Best practices as described in international norms ISO/IEC 27002:2013 [32], NIST 800-115 [63] and NIST 800-63-1 [8] state that passwords need to be complex and protected in the way that we don't handle or store the plaintext value of a password. The protection is usually with the use of a one-way cryptographic hash function. This is the reason why we need methods that can test the strength of already protected passwords.

The main premise of this research is that users don't pick their passwords randomly, but that they have a system or a behavioral pattern in which they pick their passwords. The behavior of users and malware is represented by models which represent the concrete classes of weak passwords, where the models are implemented as tools that enable us to use those models, where we have three main categories of the said models:

1. M1 model which represents the class of attacks that are used by automated scanners or various other malicious software against systems that are available on the public internet.
2. M2 model that represents the class of passwords that are tied to personal data or keywords that a user could have picked, where we use the M2 model for offline analysis.
3. M3 models which represent the behavior of the user and how the user creates a password and which elements and changes does he use when he creates his password, where we use the M3 model of offline analysis.

Each model was researched and developed in the following way:

1. For the M1 model, we developed a method for password list creation and updating from various unstructured repositories that are maintained by information security consultants or malicious hackers, which enables us to collect those lists and create a list that is based on the relative usage of each username / password pair. This enables us to automate the testing of our systems with the passwords that those groups published. This concept simplifies the evaluation of weak passwords on systems that are available on our networks.
2. For the M2 model, we developed a method that enables us to use the Google search engine to create wordlists with the usage of few keywords that we picked.
    a. Using the M2 method, a base word corpus with popular words was developed that can be used by the M3 model or in other tools.
    b. The concept of wordlist generation with the help of Wikipedia database dumps was developed. Such an approach creates a large word corpus, with the requirement for a large amount of compute resources. This drawback resulted in the development of the method that uses the Google search engine for wordlist generation.
3. For the M3 model, we developed a new method of machine learning which the author calls the sieve method, which enables us to classify passwords and develop a model that describes how a user created a password.
    a. The sieve method represents a new approach to classification problems and it's concept is shown in the application of password classification.
    b. Using the sieve algorithm on a training list of passwords, we developed a set of rules that describe the users behavior when they pick their passwords.

c. Alongside the classification, we also collected the elements like words, number and symbol patterns from which users created their passwords. Those elements were used to augment the wordlist created with the help of method used in the M2 model.

4. The M3 model and all accompanying rules that were discovered in the classification process can be used to enable the usage of M3 models, a tool named “unhash” was developed that enabled the usage of such rules in almost all popular password security testing tools that can use the standard input stdin. To test the performance and speed of the M3 model four concepts of rule usage were developed, which were compared to the baseline implementation which is available in the tool called John the Ripper [14].

With the implementation of M1 model, it has been shown that default passwords can be detected on all devices that were available and were owned by the author. Implementing the M2 and M3 models and comparing their performance with today’s baseline implementation that is available in the tool called John the Ripper [14], it has been shown that suggested implementation detects more weak passwords in a shorter timeframe.

Keywords: security, password, authentication, one-way functions, hash functions, cryptography, cryptanalysis.

Sadržaj

Popis tablica:

Popis slika:

1. Uvod

Lozinke kao autentikacijski mehanizam koriste se u većini informacijskih sustava zbog njihove jednostavnosti, skalabilnosti i minimalne cijene implementacije. Uz sve ove prednosti, njihova upotreba ne zahtijeva dodatnu sklopovsku opremu, kao što ih zahtijevaju pametne kartice ili biometrijski postupci. Zbog njihove velike izloženosti mogućoj zloporabi, bitna je zaštita lozinki od neovlaštenog korištenja u slučaju krađe s dva aspekta: složenosti lozinke i zaštite od krađe.

Složenost lozinke vezana je uz njezinu otpornost na mogućnost pogađanja od strane napadača. Znano je da su lozinke poput: 123456, 0000, 1234 i admin loše, zbog njihove jednostavnosti i učestalosti kao početne (inicijalne tj. industrijske) lozinke nepodešenih sustava. Problem nastaje želimo li udovoljiti zahtjevima složenosti lozinke sukladno najboljoj praksi [20], [64] i međunarodno priznatim normama, poput normi ISO/IEC 27002:2013 [32], NIST 800-115 [63] i NIST 800-63-1 [8]. Navedene norme zahtijevaju da sve lozinke koje koristimo u našem informacijskom sustavu odgovaraju određenim standardima složenosti i otpornosti na pogađanje, te zahtijevaju provjeravanje snage svih autentikacijskih elemenata koji uključuju i lozinke. U praksi, provjera se provodi pomoću alata kao što su *Hashcat*, *oclHashcat* [3] ili *John the Ripper* [14], koji danas čine dva najpopularnija alata za pronalaženje slabih lozinki. Jezgru svakog od ovih alata čini metoda za analizu lozinki, čiji je cilj pronaći što više slabih lozinki uz najmanju količinu računalnih resursa i vremena. Veliki problem u pogledu zahtjeva složenosti lozinki jest da su mjere kvalitete prema kojima bi korisnici trebali odabrati svoje lozinke loše definirane.

Prema preporuci, koja govori da lozinka mora imati duljinu deset znakova i biti zapisana velikim i malim slovima te brojkama, stvaraju se lozinke koje su slabije od lozinke "danas sutra korzo stari grad", koja je lakša za pamćenje i višestruko sigurnija, gleda li se kako složenost napada iscrpnim pretraživanjem raste slijedom povećanja broja kombinacija, nastalog povećanjem duljine lozinke, izlazi da prva preporuka daje jače lozinke od druge tek nakon što se duljina prve lozinke poveća na 23 znaka. Trenutno popularne mjere za prisilno primjenjivanje zahtjeva složenosti prilikom promjene lozinki, kao što je *pam-cracklib* [23], i

druge koje se za to koriste, bivaju relativno jednostavne i negativno utječu na kvalitetu lozinki.

Pod pojmom *zaštita od krađe lozinke*, smatramo zaštitu od ponovnog korištenja lozinke koja je otkrivena pomoću prisluškivanja, ili ukradena iz spremišta korisničkih imena i lozinki, kao što su baze podataka, u kojima su u određenoj tablici sadržani parovi korisničkih imena i lozinki koje su na neki način zaštićene. Uz baze podataka, česte su datoteke sa korisničkim imenima i lozinkama, kao što je datoteka za autentikaciju "/etc/shadow" [40], u raznim varijantama GNU/Linux operacijskih sustava. Najpopularnija metoda zaštite lozinke jest da se prilikom spremanja lozinke ne sprema izvorni tekst lozinke, već da se pomoću jednosmjerne kriptografske funkcije izračuna sažetak lozinke. Najpopularnije jednosmjerne (eng. *hash*) funkcije za tu namjenu su SHA1 (Secure Hash Algorithm-1) [54], SHA256 (Secure Hash Algorithm 256) [22] i MD5 (Message Digest 5), [53], koju bi trebalo izbjegavati, ali je vrlo popularna u praksi. Prilikom spremanja lozinke sprema se njezin sažetak, koji je generiran pomoću jednosmjerne kriptografske funkcije, tj. *hash* funkcije. Na taj način, prilikom svake autentikacije, izračunava se sažetak lozinke koji se uspoređuje sa spremljenim sažetkom koji ima sustav. Kriptografske jednosmjerne funkcije su dizajnirane tako da je od izvornog teksta jednostavno dobiti sažetak, ali je nemoguće doznati izvorni tekst, što bi lozinke trebalo štititi od otkrivanja. Nažalost, ukoliko je lozinka prekratka ili premalo složena, postoji vrlo velik broj metoda, poput napada iscrpnim pretraživanjem [35], napada korištenjem rječnika [5], napada postupkom prekomputacije (eng. TMTO – Time memory trade off) [30], [55], pomoću kojih je izvorni tekst lozinke moguće saznati temeljem njezinog sažetka, jasno, uz određeni utrošak vremena i računalne snage. Pod pojmom *zaštićena lozinka* u daljnjem tekstu smatramo izračunati kriptografski sažetak lozinke. U praksi postoji više varijacija na metodu zaštite lozinke, kao što je dodavanje "sjemena" (koje se i često naziva *sol* ili *dodatak*) uz čisti tekst lozinke [20], da bi spriječili TMTO-napade, ili se pokušavaju koristiti posebne jednosmjerne funkcija koje su spore za izračun, kao što su *scrypt* ili *bcrypt* [58].

Norme ISO/IEC 27002 [32], NIST 800-115 [63] i NIST 800-63-1 [8] zahtijevaju zaštitu lozinki od krađe pomoću primjene kriptografskih metoda, kao i redovito provjeravanje kvalitete lozinki svih korisnika u svim elementima informacijskog sustava koji zahtijevaju autentikaciju u sklopu penetracijskog, tj. sigurnosnog testiranja sustava, ili u trenutku mijenjanja korisničkih podataka koji uključuju i lozinku. Ukoliko lozinke čuvamo u izvornom tekstu, kako bismo omogućili lakšu provjeru njihove kvalitete, kršimo zahtjeve normi koje

traže da lozinke moraju biti zaštićene. U praksi, koristimo alate kao što je *John the Ripper* [14], *HashCat* [3], i druge, koji pomoću raznih metoda od zaštićene lozinke pokušavaju doznati izvorni tekst. Trenutno se najviše koriste metode koje se temelje na iscrpnom pretraživanju, koje se izvode tako da se uzima određeni skup znakova te ispituju sve moguće kombinacije znakova do određene duljine, koje se zaštićuju jednosmjernom kriptografskom funkcijom. Izračunati sažetak, i spremljeni sažetak koji analiziramo, uspoređuju se, i ukoliko su identični, otkrivena je ulazna kombinacija znakova koja odgovara tom sažetku. Nakon određenog broja iscrpno pretraženih kombinacija, moguće je pronaći kombinaciju čiji sažetak je identičan spremljenom sažetku koji analiziramo. Ukoliko je lozinka pronađena uz utrošak resursa koji odgovara potencijalnom napadaču, lozinku smatramo nedovoljno jakom.

2. Dosadašnja istraživanja

Analizom literature utvrđeno je da se metode analize zaštićenih lozinki mogu svesti na četiri glavne kategorije:

1. Kreiranje predložaka za ispitivanje
2. Ubrzavanje platforme za analizu
3. Napadi na algoritme ili iskorištavanje slabosti algoritama
4. Napadi na okolinu u kojoj sustav radi

2.1. Kreiranje predložaka za ispitivanje

Analiza kvalitete lozinki može se provoditi na dva glavna načina. Prvi način uključuje pribavljanje liste lozinki koje su zaštićene jednosmjernom *hash* funkcijom. Te liste najčešće sadrže korisničko ime i lozinku, kao i druge bitne podatke koje koristi operacijski sustav. Primjer takve liste je datoteka /etc/shadow na GNU/Linux sustavima, ili neka tablica s korisnicima i zaštićenim lozinkama iz baze podataka. Sukladno najboljoj praksi, takvim listama pristup mogu imati samo korisnici s administratorskim ovlastima na operacijskom sustavu ili aplikacija, tj. servis koji radi s umanjenim pravima i ima pristup dijelu sustava koji mu je potreban za obavljanje rada. Za potrebe analize takvih lozinki popis korisničkih imena i samih lozinki preuzima se iz sustava i analiza se provodi "offline" načinom, tj. na odvojenom računalu ili drugoj platformi, koja slijedom svojih tehničkih karakteristika ubrzava analizu koja se provodi. Drugi način je tzv. “online” način, u sklopu kojega se pokušava otkriti lozinka koju maliciozni kod ili automatizirani skener može jednostavno pogoditi na nekom mrežnom servisu. U slučaju *online* analize generirani predlošci se primjenjuju protiv aktivnog sustava ili servisa i cilj je isprobati što manji broj potencijalnih kandidata kako bi se aktivni mrežni servis što manje opteretio, odnosno isprobavati kandidate slabijom brzinom, tj. učestalošću, kako bi se smanjilo opterećenje na mrežnom sustavu. Svaki pokušaj pogađanja lozinke na mrežnom sustavu predstavlja jedan pokušaj prijave od strane jednog korisnika. Zbog ograničenja resursa mreže i servisa, to može predstavljati problem, ovisno o kapacitetima sustava, zaključavanju računa nakon određenog broja neuspješnih pokušaja prijave, njegovom dimenzioniranju i njegovim slobodnim resursima, pri čemu pozornost posebno treba posvetiti tome da se sustav ne uspori ili onesposobi.

Najpopularniji način za generiranje predložaka za testiranje jest iscrpno pretraživanje (eng. brute force attack) [35], za potrebe kojeg se bira skup simbola koji predstavljaju abecedu, sukladno kojoj generiramo predloške te duljinu lozinke do koje ispitujemo i kreiramo kombinacije (tj. riječi konačne duljine nad izabranom abecedom). Kao primjer za abecedu B = [a,b] i duljinu lozinke l = 2, generirali bismo predloške: aa,ab,ba,bb. Svaki od predložaka zaštitili bismo jednosmjernom funkcijom i usporedili sa zaštićenim lozinkama u listi koju analiziramo. Ukoliko pronađemo podudaranje, doznali smo koji čisti tekst stvara taj sažetak. Tijekom stvarnih analiza koristimo abecede koje sadrže velika i mala slova te brojke, što bazu znakova za analizu povećava na 64 znaka. Izračun broja kombinacija znakova za lozinke dulje od 8 znakova upućuje na smanjenu korisnost ove metode zbog eksponencijalnog porasta njihovog broja [39]. Taj problem se pokušava zaobići tako da se umjesto pojedinog znaka koriste trigrami. Jedno od mogućih rješenja problema povećanog broja kombinacija znakova je primjena Markovljevih lanaca, koji su se pokazali vrlo dobrim rješenjem jer prilikom stvaranja predložaka za analizu izbjegavaju manje učestale kombinacije znakova [15].

Nakon iscrpnog pretraživanja, druga najpopularnija metoda jest korištenje rječnika kao baze predložaka za ispitivanje (eng. Dictionary attack) [5], [12]. U ovom postupku koristimo datoteku koju zovemo *rječnik* i koji sadrži riječi ili moguće lozinke. Prilikom analize, uzimamo jedan redak iz datoteke rječnika, izračunavamo sažetak retka bez simbola za novi redak i uspoređujemo ga sa sažecima u listi koju analiziramo. Ukoliko pronađemo podudaranje doznali smo čisti tekst lozinke. Ova metoda korisna je ukoliko ispitujemo samo manju količinu mogućih lozinki ili želimo ispitati samo česte lozinke. Primjena rječnika najviše se koristi za "online" analizu, gdje želimo ispitati neki ograničeni skup mogućih slabih lozinki. Ne postoji neki standardizirani rječnik slabih lozinki koji bismo mogli primijeniti za analizu industrijskih početnih lozinki nekonfiguriranih sustava. Dodatno poboljšanje ove metode postiže se jednostavnijim pravilima supstitucije koje primjenjuju alati *John the Ripper* [14] i *Hashcat* [3].

Najnoviji znanstveni doprinos otkrivanja lozinki prikazan je u radu M. Weira iz 2010. godine [69], koji je primijenio koncept kontekstno slobodnih gramatika za generiranje predložaka za analizu. U Weirovom istraživanju, generirane su strukture koje opisuju vjerojatnosti za generiranje pojedinih predložaka za iscrpno pretraživanje. Na taj način započeto je kreiranje predložaka za analizu koji odgovaraju stvarnim uzorcima, koji mogu biti pronađeni u

stvarnim listama lozinki koje su javno objavljene na webu i predstavljaju metode učenja nad ponašanjem korisnika. Sličan koncept prikazao je William Glodek u svojem magistarskom radu [26], gdje je koristio riječi iz pjesama kao rječnik za analizu u kombinaciji sa kontekstno slobodnim gramatikama, gdje su istraživanja rađena u timu sa sveučilišta Florida State u kojem je i M. Weir. Bitne spoznaje o strukturi lozinki proveo je J. Bonneau [6] u svojem radu "The science of guessing: analyzing an anonymized corpus of 70 million passwords", gdje je analiziran anonimiziran korpus 70 milijuna nezaštićenih lozinki korisnika Yahoo! servisa. Prikazana je procjena vjerojatnosti pogađanja lozinke, pomoću već poznatih skupova podataka. Istraživanjem je utvrđeno da lozinke u prosjeku pružaju ekvivalentnu razinu zaštite kao 20 bitni nasumični niz znakova. Prikazani radovi potvrđuju da sve više istraživanja se počinje temeljiti na pitanju kako korisnici kreiraju svoje lozinke.

Uz prije navedene metode analize značajne su i metode prekomputacije [30] koje je opisao M. Hellman, poznate i kao *TMTO-postupci* (eng. Time Memory Tradeoff). TMTO-postupci se svode na kreiranje predložaka te spremanje zaštićenih lozinki i njihovih čistih tekstova u komprimirane lance pomoću postupka koji koristi kompresiju sažetka kako bi generirao sljedeći čisti tekst. Navedeni koncept opisao je P. Oechslin [55], i takve tablice nazvao “rainbow” (duga). Tablicama tipa “rainbow” potrebna je manja količina diskovnog prostora u odnosu na spremanje kombinacije čistog teksta i zaštićene lozinke. Drugi primjer TMTO postupka je spremanje mogućih kandidata čistog teksta u područja koja odgovaraju sažetku tj. kompresiji kriptografskog sažetka. Na taj način djeluju tablice tipa MCRC (eng. Modified Cyclic Redundancy Check) koje su razvijene tijekom istraživanja u sklopu ovog rada [38]. Metode prekomputacije prestaju biti korisne ukoliko se uz lozinku primjenjuje koncept kriptografskih dodataka (eng. salt). Danas najpopularnije tablice su tablice za kriptoanalizu algoritma A5/1 [24], koji štiti GSM mreže.

2.2. Ubrzavanje platforme za analizu

Pod pojmom ubrzavanje platforme za analizu smatramo izvršavanje metode za analizu na računalu ili specijaliziranom uređaju koji tu metodu ubrzava. Klasična implementacija iscrpnog pretraživanja svodi se na generator znakova i petlju koja vrši zaštitu i usporedbu zaštićene lozinke sa lozinkama koje su u listi lozinki koje analiziramo. Današnji procesori koriste više jezgri te je prostor pretraživanja potrebno podijeliti na više fizičkih ili logičkih

jezgri nekog računala. Problem podjele prostora pretraživanja trivijalan je problem paralelizacije i detaljno je opisan u radu "Accelerating penetration testing with distributed computing" [39] koji je autor objavio 2008. g. Alati kao *Hashcat* ili *John the Ripper* koriste slične metode raspodjele prostora pretraživanja ili primjenu mehanizma MPI (eng. Message Parsing Interface), koji omogućava paralelizaciju na računalnim klasterima, kao u slučaju modifikacije alata *John the Ripper*, koju su autori nazvali *djohn* [44].

Osim višeprocesorskog računanja i računalnih klastera, analizu je moguće ubrzati i izvršavanjem odgovarajućih metoda na posebnoj hardverskoj platformi visokih performansi. Na primjer, može se koristiti FPGA (Field Programmable Gate Array) koji je programabilni integrirani sklop koji može vrlo brzo obavljati specijalizirane, unaprijed programirane radnje. Primjenu FPGA koristili su brojni autori, a jedan od najpopularnijih takvih sustava je COPACOBANA (eng. Cost optimized parallel code breaker [28]), koji omogućava analizu raznih kriptografskih algoritama. Današnji trendovi temelje se na primjeni grafičkih kartica za ubrzavanje algoritama pomoću ekstenzija CUDA ili OpenCL [43]. Najmoderniji alat koji izvršava svoje algoritme na GPU-u je varijanta alata *Hashcat*, nazvana *oclHashCat* [3], koji implementira iscrpno pretraživanje i napad rječnikom u jeziku OpenCL. Drugi primjer implementacije poznatih algoritama u OpenCL-u je “Pyrit” [45], koji je implementacija iscrpnog pretraživanja usmjerenog protiv protokola WPA za zaštitu bežičnih mreža. Algoritam za izračunavanje *hash* funkcije, koji se izvodi na platformi OpenCL, otprilike je 10 puta brži nego na standardnom procesoru. Istraživači primjenjuju i suvremene konzole, kao što je PlayStation 3, koja sadrži IBM-ov procesor Cell. Pomoću klastera igraćih konzola PS3 istraživački team Arjena Lenstre iz EPFL-a u Švicarskoj generirao je koliziju u MD5 *hash* funkcijama, kao i svoj, *ispravan*, tzv. "rogue" certifikat, koji je prolazio kao valjani certifikat u infrastrukturi javnog ključa [66].

Bitno je naglasiti da ubrzavanje platforme za analizu predstavlja samo način na koji možemo ubrzati izvođenje algoritma, tj. metode kojom želimo provesti analizu. Neke metode, kao što su primjerice "rainbow" tablice, nije isplativo ubrzavati, jer iako većina mehanizama ubrzavanja omogućuje brzo računanje, prijenos podataka između grafičke kartice i ostatka računala ipak ostaje spor i ne može prenijeti velike količine podataka.

2.3. Napadi na algoritme ili iskorištavanje slabosti algoritama

Uz mogućnost slabih lozinki, postoji drugi skup napada, koji cilja probleme koje mogu imati kriptografske jednosmjerne funkcije. Svaka jednosmjerna funkcija može primiti proizvoljno veliki ulaz i uvijek producira sažetak jednake duljine. Teorijski se smatra nemogućim da dva ulaza u *hash* funkciju kreiraju isti sažetak. Ukoliko dva ulaza produciraju isti sažetak, takvu pojavu nazivamo *kolizija*. Jedna od najnovijih metoda za izradu kolizije, koja se odnosi na funkciju MD5, prikazana je u radu [74], gdje se pokazuje da je potrebno samo nekoliko sekundi da se generira kolizija između dvije proizvoljne poruke. Primjeri korištenja takvih napada su tzv. "preimage"-napadi. Prvom kategorijom “preimage”-napada za zadani sažetak pokušava se pronaći poruka koja kao zaštićena taj sažetak i kreira. Danas postoje samo računalno neisplativi napadi na poznatije *hash* funkcije za prvu kategoriju otpornosti, tj. otpornosti na izračunavanje originala. Drugu kategoriju te vrste napada čine napadi za koje se traži otpornost na izračunavanje iste poruke, pri čemu se od poruke *m*, i sažetka poruke *h*, želi generirati poruka *n*, koja ima identičan sažetak [64].

Postoje i napadi koji eksploatiraju izravne algebarske slabosti nekih algoritma. Na primjer, algoritam MD4 toliko je algebarski slab da je generiranje kolizija za njega trivijalno [62]. Današnje jednosmjerne funkcije puno su kompleksnije i puno sigurnije, tako da je manje vjerojatno da će netko pronaći neke nove algebarske slabosti, ukoliko se ne otkrije neka nova metoda za generiranje kolizija.

2.4. Napadi na okolinu u kojoj sustav radi

Zadnja klasa je napad na okolinu u kojoj radi sustav za autentikaciju. Kao primjer možemo navesti napade sporednim kanalima (eng. side channel attacks), u sklopu kojih lozinku možemo prikupiti na neki sporedan način, primjerice iz memorije računala koje koristi sustav za autentikaciju, ili prisluškivanjem nezaštićene komunikacije, gdje se lozinke prenose u čistom tekstu. U zaštiti lozinki ljudski čimbenik također je bitan. Napadi, kao što su socijalni inženjering, tj. prijevara korisnika nagovaranjem da odaju svoje lozinke, korištenjem posebnih alata ili malicioznog koda za krađu lozinki prilikom utipkavanja u sustav, ili primjenom fizičke prisile nad osobom da otkrije svoju lozinku, ne mogu se zanemariti prilikom odabira kontrola i zaštitnih mjera u izgradnji cjelovitog sustava sigurnosti.

3. Ciljevi istraživanja

3.1. Motivacija za provođenje istraživanja

Autentikacijske metode koriste različite zaštite da bi se spriječilo neovlašteno korištenje lozinke u slučaju krađe. Da bi neutralizirali brzinu računala i sve dostupnije GPU računalstvo, razvijene su kriptografske jednosmjerne funkcije koje su složene i spore za izračun, poput funkcije "bcrypt" [58], zbog kojih napadači ne mogu isprobati veliki broj mogućih kombinacija u jednoj sekundi. Prekomputacija i korištenje napada tipa TMTO [55], [30], onemogućuje se pomoću korištenja kriptografskog *sjemena* (drugi naziv je *sol*, [20]). Primjena tih zaštita ne utječe na legalne korisnike, ali otežava mogućnost ispitivanja sustava u potrazi za slabim lozinkama koje ugrožavaju sigurnost informacijskog sustava. Zbog toga osobe koje se brinu o sigurnosti sustava pronalaze slabe lozinke pomoću alata kao što su *John*, *Hashcat*, i drugi. Cilj testiranja sigurnosti jest pronaći što veći broj slabih lozinki uz minimalni utrošak računalnih resursa i vremena.

Iscrpno pretraživanje je jedina metoda kojom se lozinke mogu analizirati neovisno o metodama zaštite koje se primjenjuje ili algoritma kojim se lozinke štite. Iz toga izlazi da unapređenje metode iscrpnog pretraživanja omogućuje najveću fleksibilnost analize lozinki, koja olakšava posao osobama koje testiraju sigurnost sustava.

Motivacija za provođenje istraživanja jest kreirati metode za analizu lozinki koje:

1. Omogućuju provjeru i pronalaženje slabih lozinki koje su aktivne u sklopu servisa, aplikacija i uređaja koji su dostupni na mreži, gdje želimo pronaći moguće industrijske i početne lozinke na automatizirani način. Jedan od većih problema predstavljaju uređaji koji imaju ugrađena "stražnja vrata" (eng. Backdoor) koja otvaraju administratorski pristup u sustav. Takve funkcije nisu dokumentirane, ali uključuju korištenje posebnog korisničkog imena i lozinke koje su proizvođači ugradili da bi pristup u sustav iz nekih razloga omogućili sami sebi. Takve funkcije bivaju otkrivene od strane zlonamjernih osoba, i u većini slučajeva nema načina da se takva stražnja vrata uklone, jer proizvođači ne dokumentiraju i ne priznaju implementaciju stražnjih vrata. Lozinke, pribavljene takvim načinom, zlonamjerni pojedinci otkrivaju i

objavljuju u svojim zbirnim listama lozinki na Web-u. Administratorima i osobama koje testiraju sigurnost sustava stoga je korisno pokušati testirati i moguće industrijske lozinke na uređajima kao što su usmjernici, preklopnici ili IP-telefoni, upravo s listama lozinki koje koriste napadači na takve lozinke, s ciljem povećanja sigurnosti istih. Pronalazak slabih lozinki na mrežnim servisima bitan je stoga što na javnom Internetu veliki broj automatiziranih skenera i malicioznog koda pokušava pogoditi lozinke na tim sustavima i zlorabiti sustav.

2. Omogućuju prikupljanje informacija s javnog Interneta, vezane uz određene ključne riječi, da se omogući stvaranje rječnika sa riječima koje su vezane uz unešeni skup pojmova.
3. Omogućuju provjeru i pronalaženje slabih lozinki kad postoji pristup datoteci ili tablici u bazi podataka koja sadrži korisnička imena i zaštićene lozinke. Pronalazak slabih lozinki u datotekama ili tablici bitan je zbog pronalaženja lozinki koje je moguće pogoditi ili koje bi napadači mogli zlorabiti protiv drugih sustava, ukoliko dobiju pristup toj datoteci i otkriju lozinke koje su zaštićene.

Cilj istraživanja je modelirati ponašanje automatiziranog koda i korisnika na način da možemo kreirati modele ponašanja korisnika i pravila kojima oni odabiru svoje lozinke. Prilikom analize mrežnih servisa želimo otkriti što veći broj industrijskih lozinki na mrežnim uređajima, a prilikom analize datoteka sa zaštićenim lozinkama želimo da analiza otkrije što veći broj lozinki za određene računalne resurse i vrijeme. Predložene metode trebale bi raditi kao potpora postojećim alatima, kao što su *John the Ripper* [14], *Hashcat* [3], *Metasploit Framework* [34], i drugi.

3.2. Istraživačka pitanja i hipoteze istraživanja

Metode analize lozinki zaštićenih jednosmjernim kriptografskim funkcijama, koje koriste metode iscrpnog pretraživanja, provjeravaju strukturu lozinke prema određenom skupu dopuštenih simbola, skupa riječi u zadanom rječniku ili modifikaciji tih skupova. Do sada, samo metoda M. Weira [69] koristi analizu pomoću struktura iz uzoraka pravih lozinki, koja bi predstavljala način na koji korisnici zapravo kreiraju lozinke. Strukture lozinki i obrazac prema kojem je neka lozinka kreirana zvat ćemo modelom, iz razloga što pojam strukture u kontekstu jednosmjernih funkcija i analize lozinki u [20] i [69] već postoji. Najpopularniji i najrašireniji alat za analizu kvalitete lozinki prema listi [46], koju održava zajednica vezana uz sigurnost informacijskih sustava, je *John the Ripper,* koji koristi tri metode: prva metoda je metoda "single mode", i ona koristi informacije o korisniku koje su vezane uz njegov korisnički račun, kao što su ime, prezime i korisničko ime, nakon koje slijedi analiza pomoću modificiranog rječnika, te na kraju iscrpno pretraživanje. Zbog odabira ovog alata kao referentnog u raznim metodologijama za penetracijsko testiranje, metodu koju koristi *John the Ripper* [14] smatrati ćemo referentnom implementacijom za usporedbu performansi s novopredloženom metodom. Tijekom ranijih istraživanja, prikazanih u radovima [39] i [38] te rada u istraživačkom ovom području istraživanja, autor ovog rada došao je do sljedećih istraživačkih pitanja:

1. *Bi li metoda koja se temelji na poznavanju načina prema kojem korisnici stvaraju lozinke, izvedenom putem analize velikog broja realnih lozinki, omogućavala kvalitetniju analizu od referentne metode?*
2. *Kakve bi performanse imala nova metoda analize u omjeru broja otkrivenih lozinki i vremena trajanja analize, u odnosu na referentnu metodu?*

Sukladno istraživačkim pitanjima, očekuje se potvrđivanje iz njih proisteklih hipoteza. Eksplicitno iskazane, hipoteze glase kako slijedi:

H1: *Metoda kriptoanalize zaštićenih lozinki, koja se temelji na poznavanju načina na koji korisnici te lozinke stvaraju, iskazana pripadajućim razvijenim modelima, omogućit će zamjetno bolje performanse od referentne metode.*

H2: *Posebice, metoda iz hipoteze H1 pokazat će se zamjetno učinkovitijom u pogledu omjera broja otkrivenih lozinki i vremena trajanja analize, u odnosu na referentnu metodu.*

3.3. Društvena opravdanost i relevantnost istraživanja

Prilikom testiranja zaštićenih lozinki, metoda koja otkriva više slabih lozinki uz isto vrijeme trajanja analize i uz jednake računalne resurse značajno bi ubrzala i olakšala rad osoba koje provode testiranje sigurnosti lozinki na elementima informacijskog sustava i omogućila bolje funkcioniranje sustava koji u svojem radu koriste metode analize lozinki. Kao očekivani znanstveni doprinos istraživanja možemo izdvojiti sljedeće elemente:

- Otkrivanje liste loših lozinki čija primjena predstavlja veliki rizik za mrežne servise u informacijskom sustavu. Pritom se misli na razvijenu listu za model tipa M1.
- Razvoj metode za utvrđivanje pravila (obrazaca) sukladno kojima je korisnik kreirao lozinku.
- Razvoj struktura podataka ili način zapisa pravila kako je korisnik kreirao lozinku.
- Otkrivanje skupa pravila iz lista za analizu koji opisuje na koji način su korisnici iz lista za analizu generirali lozinke.
- Razvoj modela za analizu (M1 do M3).
- Razvoj metode za analizu zaštićenih lozinki koja koristi modele M1 do M3.
- Utvrđivanje performansi predložene metode koja implementira model tipa M3 u odnosu na trenutačne standardne metode za analizu zaštićenih lozinki. Očekuje se da će istraživanje potvrditi bolje performanse predložene metode.

3.4. Primjena modela M1-M3

Modeli razvijeni tijekom istraživanja imaju praktičnu primjenu od strane administratora sustava, pri čemu modeli M1 do M3 omogućuje primjenu na sljedeći način:

Primjena modela tipa M1 odgovara klasi napada koju koriste automatizirani skeneri (mehanizmi otkrivanja ranjivosti) ili oblici malicioznog koda. Model tipa M1 može se pokretati jednom ili više puta dnevno, kako bi se uklonile najopasnije i najneposrednije slabe lozinke, koje mrežnu infrastrukturu u tom smislu čine ranjivom. Na taj način, automatizira se

proces detekcije najslabijih lozinki, koje za informacijski sustav predstavljaju iznimno velik rizik. Primjer takve ranjivosti su nekonfigurirani uređaji, poput preklopnika, usmjernika ili bežičnih pristupnih točaka, s industrijski standardnim početnim lozinkama. Drugi interesni element ovog modela su korisnici koji izaberu iznimno loše lozinke, kao što su 123456, qwerty, i ostale koje su uključene u model tipa M1. Model M1 namijenjen je za *online* analizu.

Primjena modela tipa M2 također predstavlja pretraživanje usmjereno nalaženju kritičnih slabih lozinki, ali vezanih uz osobne podatke ili podatke koje bi napadač mogao otkriti da su vezane uz korisnika. Na taj način, u metodu analize uvodimo drugu prednost koja uključuje pretraživanje za slabim lozinkama korisnika. Primjer takve loše lozinke jest datum rođenja korisnika ili naziv tvrtke u kojoj je zaposlen. Model tipa M2 može se koristiti za potrebe otkrivanja slabih, ali smislenih lozinki, koje automatski skeneri ne bi otkrili, ali koju bi sposoban napadač mogao otkriti jednostavnije negoli neku kompleksniju lozinku. Model M2 kreira listu riječi za analizu pomoću određenog skupa ključnih riječi ili putem upisivanja nekih riječi koje osoba koja testira sigurnost lozinke odredi. Model M2 je namijenjen za *offline* analizu.

Modeli tipa M3 prikazuju način ponašanja korisnika, tj. način na koji korisnici stvaraju svoje lozinke, koje riječi, brojeve i znakove koriste te kako mijenjaju i prilagođavaju lozinku da bi kreirali jaku lozinku. Model M3 koristimo za *offline* analizu.

4. Nacrt istraživanja i izvori podataka

Razvoj modela tipa M3 zahtjeva korištenje baze lozinki iz kojih ćemo saznati na koji način korisnici kreiraju svoje lozinke. Navedena baza lozinki predstavljati će skup za učenje na kojem ćemo provesti učenje metode i otkriti pravila sukladno kojima korisnici kreiraju svoje lozinke. Verifikacija modela M3 biti će provedena ne eksperimentalnom skupu koji ne uključuje liste za učenje.

4.1. Skup lozinki za učenje

Da bismo osigurali relevantnost podataka bez zadiranja u korisničku privatnost, u sklopu predistraživanja i istraživanja koristiti će se skup realnih lozinki. Lozinke su pribavljene iz tzv. "exposure"-dokumenata, koji su objavljeni javno od strane osoba koje su neovlašteno pristupile sustavu i htjele nanijeti dodatnu štetu napadnutom subjektu, na način da objave korisnička imena i lozinke, što oštećenim subjektima stvara velike probleme. Zbog toga subjekti gube na kredibilitetu, a gube i korisnike jer su svi oni primorani mijenjati svoje lozinke. Prikupljeni podaci su relevantni jer prikazuju stvarne lozinke koje su korisnici koristili. Svaka lista lozinki zapravo je bila rezultat velikog medijski eksponiranog i poznatog sigurnosnog proboja i svi korisnički računi nakon proboja su promijenjeni, tako da nema mogućih posljedica zbog mogućnosti da su navedene lozinke aktivne vjerodostojnice. U sljedećoj tablici prikazana je struktura odabranih podataka. Da bismo omogućili lakšu analizu, sve su liste spojene i grupirane po zajedničkim lozinkama sa zbrojenom učestalošću pojavljivanja. Lista za učenje sadržavala je ukupno 33.291.435 lozinki, tj. 14.662.210 jedinstvenih lozinki. Bitno je napomenuti da su iz svih analiza izbačene nelatiničke abecede kao grčka, ćirilične, arapske, kineske i hebrejske abecede, kao i sve lozinke dulje od 25 znakova.

Podaci za analizu odabrani su iz sljedećih značajnijih incidenata:

Tablica 1: Skup lozinki za učenje

Naziv liste	Opis	Broj lozinki
phpbb	Lista lozinki objavljena nakon proboja popularnog foruma PHPbb	255.421
yahoo	Lista lozinki koja je bila objavljena nakon proboja portala yahoo.	453.485
rockyou	Lista lozinki od korisnika aplikacija na društvenim mrežama od strane grupacije rockyou.	32.603.388

Strukturu duljine lozinki u skupu za učenje možemo vidjeti u sljedećoj tablici:

Tablica 2: Struktura duljine lozinki u skupu za učenje

Duljina	Broj jedinstvenih lozinki
1	52
2	409
3	3.036
4	20.568
5	263.711
6	1.985.991
7	2.550.619
8	3.063.505
9	2.241.715
10	2.059.376
11	886.260
12	575.921
13	367.077
14	249.989
15	161.927
16	118.584
17	36.912
18	23.354
19	15.429
20	13.064
21	7.409
22	6.025
23	4.736
24	3.756
25	2.785

Navedenu spojenu i pročišćenu listu u daljnjem tekstu zovemo *lista za analizu* ili *lista lozinki za učenje*.

4.2. Skup lozinki za verifikaciju rezultata

Da bismo mogli provjeriti performanse za analizu predložene metode, moramo moći mjeriti i usporediti performanse sa trenutno standardnim metodama analize. Kao skup za eksperimentalnu verifikaciju, odredili smo popularan skup lista iz sljedećih značajnijih incidenata:

Tablica 3: Skup lozinki za verifikaciju rezultata

Naziv liste	Opis	Broj lozinki
myspace	Lista lozinki ukradena pomoću *phishing*-napada na popularnoj društvenoj mreži *myspace*, koju su napadači javno objavili.	37144
elitehackers	Lista lozinki korisnika popularnog hakerskog foruma, objavljena kao dio *exposure*-dokumenta zfo05.	895
carders.cc	Lista lozinki korisnika popularnog foruma za kartične prijevare	1902
hak5	Lista lozinki korisnika popularnog hakerskog foruma, objavljena kao dio *exposure*-dokumenta zfo05	2351
hotmail	Lista lozinki koje su ukradene od korisnika sa popularnog webmail servisa *hotmail*	8926
Linkedin	Lista lozinki objavljena nakon proboja najpopularnije društvene mreže za poslovno umrežavanje *linkedin*	2968799

Lista *Linkedin* izvorno sadrži 6.143.150 lozinki, te je bila zaštićena algoritmom SHA1, gdje je zlonamjerni haker, koji je objavio *linkedin*-listu, na nekim lozinkama prvih 5 bajtova *hash*-vrijednosti promijenio u nule. Kao primjer, možemo vidjeti prvih par redaka iz liste *linkedin*:

```
00000fac2ec84586f9f5221a05c0e9acc3d2e670
...
b3344eaec4585720ca23b338e58449e4c3d2f628
674db9e37ace89b77401fa2bfe456144c3d2f708
```

Takvo ponašanje nije klasično niti za jedan popularni alat. Da bismo provjerili stoje li iza tih oznaka prave lozinke, lista je bila kriptoanalizirana pomoću alata *john*, na način da se pokušavalo ignorirati prvih 5 znakova jednosmjerne funkcije i na taj način dodatno pokušati doznati i te lozinke. Analiza te liste bila je pokrenuta, i nakon 111 dana i 13 sati od početka njezina rada, kad je bila prekinuta zbog kvara na sustavu za analizu, otkrila je 2.968.799 jedinstvenih lozinki, tj. 47,24434% ukupnog korpusa lozinki iz liste *linkedin*, koje ćemo koristiti u skupu liste za verifikaciju rezultata.

Stopljene liste, prikazane u tablici 3, u daljnjem tekstu zovemo *lista lozinki za verifikaciju* ili *eksperimentalna lista*.

4.3. Pribavljanje skupova za analizu i dostupnost lista lozinki

Većina skupova lozinki dostupna je u repozitorijima koje održavaju konzultanti za sigurnost. Primjer takvog repozitorija je popularni repozitorij *skull security* [7], koji održava Ron Bowes, i repozitorij *Password Research* [48], koji održava Bruce Marshall, koji su svjetski poznati stručnjaci za sigurnost.

Sve navedene liste također je moguće pronaći na rečenim repozitorijima, pomoću web tražilica, na *pastebin*-u [68], ili putem raznih bittorrent-tražilica, kao što je "thePirateBay", jer su zlonamjerni napadači namjerno htjeli da te liste lozinki dospiju u javnost i budu dostupne svima. U pravilu su liste javno dostupne i više godina nakon što je incident bio publiciran.

4.4. Nacrt istraživanja

Tijekom predistraživanja utvrđeno je da bi jezgra nove metode za analizu zaštićenih lozinki trebao biti skup zapisa koji opisuju način na koji korisnici kreiraju lozinke. Da bi se razvila nova metoda potrebno je provesti istraživanje tijekom kojeg ćemo razviti metodu koja može prikazati način na koji korisnik kreira svoju lozinku pomoću skupa pravila. Takvi modeli i pravila, koji nisu ograničeni fiksnim rječnikom ili dugotrajnim pretraživanjem svih mogućih kombinacija unutar definiranog raspona, trebali bi omogućiti bržu analizu lozinki.

Preliminarne analize mogućih modela koja je autor provodio dokazale su potrebu da se metoda realizira pomoću modela s tri razine (sloja):

1. Modeli koji opisuju slabe ili industrijske uobičajene lozinke koje nalazimo kao početne lozinke nekonfiguriranih uređaja. Primjeri takvih lozinki su: *admin*, *root*, *0000*, *123456*, *password*. Taj model nazivamo i modelom tipa M1.
2. Modeli koji opisuju lozinke korisnikove okoline ili osobnih podataka, koje su vezane za ključne riječi koje je korisnik mogao uzeti iz okoline. Primjeri takvih lozinki su: *Pero21061980*, *pero80* ili *foi80* gdje je recimo, ime korisnika Pero i radi na FOI-u. Taj model nazivamo i modelom tipa M2.
3. Modeli koji opisuju česte odabire, uzorke ili kombinacije odabira lozinki od strane korisnika. Takvi modeli opisu ponašanje korisnika i način na koji korisnici kreiraju svoje lozinke. Taj model nazivamo i modelom tipa M3.

Bitan dio metode za analizu jest generator lozinki koji bi iz modela razine M2 i/ili M3 generirao prijedloge lozinki za ispitivanje njihove snage, koje bi mogao koristiti alat za ispitivanje koji ih zaštićuje jednosmjernom kriptografskom funkcijom i provjerava nalazi li se ta lozinka u datoteci zaštićenih lozinki koje analiziramo. Ovaj dizajn sustava izabran je zbog mogućnosti da se u budućnosti primijeni na modernim računalnim sustavima, temeljenim na višejezgrenom, *grid* i GPU računalstvu ili računalnim klasterima. Na taj način moguće je instancirati više generatora i mehanizama za provjeru snage lozinki, koji bi mogli povećati učinkovitost sustava i omogućiti korištenje heterogenih infrastruktura i raznih računalnih sustava. Jedan od razloga za ovo jest i nemogućnost podjele liste lozinki koju provjeravamo između više mehanizama za provjeru, već raspon prostora pretraživanja moramo podijeliti na način definiran u radu [39].

Modele treba zapisati na način koji smanjuje količinu memorije potrebnu za njihovo trajno čuvanje (perzistenciju) i omogućuje brzu rekonstrukciju podataka koju generator lozinki može koristiti. U tijeku predistraživanja, pregledom literature i poznatih struktura podataka, utvrđeno je da je potrebno razviti oblik zapisa temeljen na prefiksnim stablima ili DAWG-grafovima, opisanima u [25] i [65]. Točna struktura i oblik zapisa biti će razvijena u tijeku istraživanja.

4.4.1. Modeli slabih ili uobičajenih industrijskih lozinki (M1)

Cilj modela tipa M1 jest razviti listu lozinki koja predstavlja najveći sigurnosni rizik tijekom uporabe na sustavima. Na primjer, poznati crv *Conficker*, uz mnoštvo propusta koje je koristio za inficiranje računala, također je koristio i listu čestih lozinki za pogađanje lozinki na mrežnim servisima te je 2008. godine uspio zaraziti velik broj računala, djelomično i pogađanjem lozinke na mrežnim servisima. U tijeku pisanja ovog teksta, još uvijek se pretpostavlja da se broj zaraženih računala crvom *Conficker* prema [71] broji u milijunima. Ovo je još jedan dokaz da administratori sustava moraju izbjegavati upotrebu iznimno slabih lozinki na bilo kojoj sastavnici informacijskog sustava.

4.4.2. Modeli korisničke okoline ili osobnih podataka (M2)

Poznato je da korisnici prilikom odabira lozinki traže elemente za lozinku koji su im poznati od ranije ili su lagani za pamćenje. Primjeri takvih elemenata mogu biti datumi rođenja bliskih osoba, imena bliskih osoba, nazivi predmeta iz okruženja, nazivi projekata na kojima korisnik radi, ime tvrtke u kojoj korisnik radi ili općenito trajni pisani tragovi koje korisnik može stalno vidjeti u svojoj okolini. Model tipa M2 ne može biti unaprijed izrađen kao modeli tipova M1 i M3, već mora biti kreiran od svih dostupnih osobnih podataka ili podataka vezanih uz osobu čija se lozinka želi doznati, a koje osoba koja provodi analizu može prikupiti. Da bismo generirali model tipa M2, treba razviti skup pravila koji će omogućiti mutaciju i kombinaciju ulaznog skupa podataka prema informacijama o načinu kako korisnici kreiraju lozinke.

4.4.3. Razvoj pravila mutacije i kombinacije iz liste za analizu (MK- Pravila)

Svako MK-pravilo predstavlja opis načina na koji je korisnik kreirao svoju lozinku. Pravila možemo podijeliti na dvije kategorije: pravila mutacije koja uzimaju ulazni podatak i mijenjaju ga na neki smisleni način. Primjer mutacije je zamjena slova i sa 1, e sa 3, o sa 0, skraćivanje godine 1984 na 84, pomaci prilikom pisanja znakova na tipkovnici i ostala pravila koja možemo izlučiti iz liste za analizu. Pravila mutacije čine dodatak na ograničeni ulazni

skup, na način da stvaraju više mogućnosti koje je korisnik možda mogao odabrati. Drugi oblik pravila su pravila kombinacije. Pravila kombinacije kombiniraju ulazni skup podataka prema značajkama koje su otkrivene u listama lozinki, te kao ulaz primaju ulazne podatke koje smo mutirali pomoću pravila mutacije.

Da bismo otkrili moguća pravila mutacije, ideja je razviti metodu analize koja može analizirati i uspoređivati elemente svake lozinke u datoteci za analizu s elementima koje imaju standardni rječnici (GNU Aspell [4] / HunSpell [52]) za provjeru pravopisa ili prevođenje i koji su javno dostupni za skoro sve svjetske jezike. Otkrivanje sličnosti između riječi može se realizirati primjenom metoda kao što je Levenshteinova [42] ili Damerau-Levenshteinova distanca [11] između nizova znakova. Primjena jednostavnijih distanci, kao što je Hammingova, nije pogodna stoga što u obzir želimo uzeti dodavanje, uklanjanje i supstituciju znakova. Eksperimentalno treba utvrditi koliko bi primjena distance bila korisna za identifikaciju pravila, gdje bi identifikacija bila potpuno automatizirana. Ukoliko treba ojačati snagu identifikacije i ukoliko Levenshteinova distanca neće biti dovoljna za navedeni postupak, možemo eksperimentalno provjeriti potrebu za metodom korjenovanja riječi (eng. stemming) za identifikaciju sufiksa i prefiksa na korijene riječi koje su korisnici koristili. Složene lozinke iz kojih ne možemo automatski kreirati pravila, možemo grupirati po sličnom uzorku i pokušati ih analizirati na bazi pojedinačnog skupa uzoraka te zatim razviti pravila koja bi odgovarala korisničkoj mutaciji ili kombinaciji koja je kreirala takve lozinke u navedenoj grupi. Radna ideja je pravila zapisati u obliku izraza koji su slični regularnim izrazima koje možemo primijeniti nad nekim ulaznim skupom znakova.

Da bismo neutralizirali utjecaj ekstremnih vrijednosti na mogućnost identifikacije pravila, metoda za analizu radit će inkrementalno. Jezgra metode je predložena Levenshteinova distanca, gdje možemo tvrditi da je distanca vrijednosti 1 zapravo jednostavna supstitucija, uklanjanje ili dodavanje jednog znaka. Za distance veće od 1 možemo grupirati moguća pravila i analizom utvrditi smislenost pravila pregledom lozinke, kandidata iz rječnika i pravila koja bi kreirala takvu mutaciju. Grupiranje omogućava automatizirano prepoznavanje pravila na osnovu većih uzoraka i izostanak potrebe za ručnom analizom.

Da bismo otkrili pravila kombinacije, možemo koristiti pravila mutacije, standardne rječnike i prije identificirane generalizacije elemenata zapisa da identificiramo načine kombinacije koji korisnici koriste. Prilikom identifikacije pravila pomoću skupa rječnika, možemo sortirati

korištene rječnike po učestalosti korištenja rječnika i riječi u rječniku po kumulativnom broju pojave pojedine riječi u nekoj lozinki. Navedena radnja koristi se prilikom izrade modela tipa M3. Jedan od većih istraživačkih problema odnosio se na način na koji treba opisati MK-pravila da se mogu koristiti tijekom istraživanja te odgovora na pitanje hoće li predloženi koncept istraživanja omogućiti kreiranje MK-pravila i njihovu upotrebu u sklopu modela M3.

4.4.4. Modeli ponašanja korisnika odabira (M3)

Model tipa M3 predstavlja korištenje skupa MK-pravila u kombinaciji sa standardnim rječnicima. Kao ilustraciju modela možemo prikazati sljedeće tri lozinke:

1. *cabbage84plant* : engleska riječ, česta godina, engleska riječ
2. *eigenartigervogel* : njemačka riječ, njemačka riječ
3. *19692007* : 8 numeričkih znakova
4. *S3curity*: Pretvoreno prvo slovo u veliko, promijenjen e u 3, engleska riječ

Modeli tipa M3 koriste MK-pravila za kombiniranje riječi iz svih rječnika koji se u sklopu analize koriste. Najbolji postupak kombiniranja elemenata iz rječnika i odabira pravila treba eksperimentalno utvrditi nad listom za eksperimentiranje. Frekvencija elemenata koje su korisnici koristili i učestalost primjene pravila kombinacije možemo identificirati iz liste za učenje. Informacije koje smo prikupili iz učestalosti korištenja pojedinih rječnika ili njihovih elemenata mogu biti korisne u analizi postojanja veza između elemenata pojedinih rječnika.

5. Istraživanje i razvoj modela tipa M1

5.1. Razvoj modela tipa M1

Cilj modela tipa M1 jest razviti skup korisničkih imena i lozinki čija primjena u testiranju otkriva početne industrijske lozinke na raznim uređajima i servisima. U vrijeme pisanja ovog rada bili su (i još uvijek su) dostupni repozitoriji čestih lozinki, u kojima je za odabranog proizvođača i model uređaja moguće dobiti popis potencijalnih industrijskih lozinki. Takav pristup manjkav je u slučaju kada želimo koristiti alate za penetracijsko testiranje koji mogu isprobavati više skupova korisničkih imena i lozinki za određene servise ili uređaje i značajno ubrzati testiranje većih mreža računala i servisa.

Tijekom istraživanja pronađeni su sljedeći repozitoriji početnih, industrijskih lozinki koje održavaju razni autori gdje je cilj razvoja modela tipa M1 bio omogućiti automatizaciju stvaranja jedinstvene liste lozinki za testiranje iz navedenih repozitorija:

- http://cirt.net/passwords
- http://www.phenoelit.org/dpl/dpl.html
- http://www.liquidmatrix.org/blog/default-passwords
- http://securityoverride.org/default-password-list/
- http://dexcms.com/default-passwords-list

U nekim od navedenih repozitorija čuvaju se i lozinke za poznata *stražnja vrata* koja su proizvođači opreme ugradili u svoju opremu, ali ih nisu javno objavili. Problem nastaje ako zlonamjerni napadači znaju za takva *stražnja vrata* i mogu ući u sustav, a osobe koje se brinu za sigurnost tog sustava za takav propust ne znaju. Opisani repozitoriji nemaju mogućnost preuzimanja baza podataka ili strukturiranog oblika sadržaja. Da bi omogućili preuzimanje svih sadržaja iz navedenih repozitorija, bilo je potrebno razviti metodu i alat koji omogućuju preuzimanje sadržaja iz tih repozitorija i izvode parove korisničkih imena i lozinki, te stvaraju globalni rječnik čestih industrijskih lozinki i *stražnjih vrata*. Alat u kojem je implementirana metoda tipa M1, nazvan je “default_password_updates.py” i dostupan je u digitalnom repozitoriju razvijenom u sklopu i za potrebe ovog rada.

Alat “default_password_updates.py” sadrži parsere za navedene repozitorije, preuzima sadržaj web-stranica tih repozitorija i pomoću popularne biblioteke za parsiranje HTML koda, *BeautifulSoup* [60], parsira tablice i kreira rječnik, gdje je svakom paru korisničkog imena i lozinke pridružena vrijednost učestalosti u toj listi. Navedeni postupak provodi se za svaki digitalni repozitorij, uz zbrajanje učestalosti pojedinog para korisničkog imena i lozinke, stvarajući rječnik sa skupom svih lozinki iz navedenih repozitorija, gdje poredak elemenata u listi ovisi o učestalosti njegovog ponavljanja u smislu odabira industrijske lozinke proizvođača uređaja.

5.2. Testiranje modela tipa M1

Testiranje učinkovitosti rječnika provedeno je na opremi koja je autoru bila dostupna u vrijeme pisanja doktorske disertacije, koja je uključivala uređaje u osobnom vlasništvu, odnosno *Laboratoriju za otvorene sustave i sigurnost*, u sklopu Fakulteta organizacije i informatike. Za sve testirane uređaje autor je imao dopuštenje vlasnika da ih koristi za testiranje. Test je proveden na način da su svi uređaji postavljeni na industrijske postavke i pokušano im je pristupiti uz pomoć alata “Metasploit Framework” [34] ili “Hydra” [29] ovisno o tipu autentikacije koji je trebalo provjeriti, s modulom za pogađanje lozinki na mrežnom servisu, koristeći rječnik koji generira alat “default_password_updates.py”. Rezultati testiranja prikazani su sljedećom tablicom:

Tablica 4: Rezultati testiranja rječnika M1

Uređaj	Početna kombinacija korisničkog imena i lozinka	Pozicija kombinacije u M1 rječniku
Uređaj: Cisco 2600 series Router	cisco:cisco	475
Uređaj: Buffalo WZR-HP-G300NH / DDWRT v24-sp2 (08/07/10) std	root:admin	21
Uređaj: Linksys WRT54GL v2.0 / DDWRT	root:admin	21
Uređaj: Mikrotik RouterBoard 750GL / RouterOS 5.2	admin : (bez lozinke)	3
Uređaj: Edimax PrintServer PSD49878	(bez korisnika):1234	11
Uređaj: Xerox WorkCentre 7345	11111:x-admin	578
OS za mrežno filtriranje: pfsense 2.0.3	admin:pfsense	663
OS za testiranje sigurnosti: Kali Linux 1.0.4	root:toor	820
CMS sustav Drupal	-neuspješno-	-neuspješno-
CMS sustav Wordpress	-neuspješno-	-neuspješno-

Svi uređaji koji su bili u vlasništvu autora i koji su imali početne industrijske lozinke bili su otkriveni uz pomoć generirane liste lozinki. Uređaji, aplikacije i mrežni servisi koji nisu imali početne industrijske lozinke, kao što su poznati CMS-portali Drupal [67], odnosno Wordpress [41], koji zahtijevaju unos početnog korisnika i njegove lozinke, nisu bili ranjivi.

6. Istraživanje i razvoj modela tipa M2

6.1. Razvoj modela tipa M2

Cilj modela tipa M2 jest kreirati rječnike koji sadrže podatke vezane uz osobu, entitete ili ključne riječi koje mogu koristiti druge metode za analizu. Dosadašnja praksa temeljila se na generiranju rječnika preuzimanjem sadržaja teksta s web-mjesta poduzeća čije lozinke analiziramo. Postoji nekolicina manje poznatih alata kao što je alat *CeWL* [73] koji pretražuju cijelo web-mjesto i kreiraju rječnik iz sadržaja teksta web-mjesta koji nisu pronašli širu primjenu zbog slabe učinkovitosti i općenito manjka istraživanja o učinkovitosti tog pristupa.

Prilikom izrade modela tipa M2 postavljeni su sljedeći zahtjevi:

1. Kreiranje rječnika moguće je uz pomoć nekolicine ključnih riječi, kao što je ime entiteta tj. poduzeća, imena i prezimena osoba koja su vezana uz entitet, ključnim riječima iz profesije ili raznim ključnim riječima koje odabere osoba koja provodi testiranje sigurnosti.
2. Moguće je proširiti početne rječnike pomoću proširivanja u širinu rekurzivnim upitima.
3. Kreirani rječnici ne smiju biti isključivo vezani za samo jedan izvor generiranja, kao što je web-mjesto entiteta ili samo *online*-enciklopedija, već mora obuhvatiti široku paletu mogućih izvora kako bi prikupljeni korpus riječi bio što raznolikiji.

6.2. Kreiranje rječnika pomoću tražilice Google: metoda *gwordlist*

Za potrebe ispunjenja gore navedenih uvjeta autor rada odlučio je kao izvor prijedloga za potencijalne lozinke koristiti web-tražilicu Google, držeći da ona posjeduju u tom smislu relevantne informacije. Tražilice za određeni ključni pojam vraćaju više rezultata koji se pozivaju i na web-mjesta, *online*-enciklopedije i druge izvore. U tom slučaju, raspršenost riječi koje se dobije pomoću web-tražilice predstavlja korisno svojstvo jer osigurava stvaranje dovoljno velike liste riječi. Tražilice također imaju i jezik za provođenje upita kojim je moguće ograničiti pretragu samo na jednu domenu, govorno područje ili državu te mogu puno

detaljnije specificirati pretragu vezanu uz ključne riječi. Kako bi se omogućilo kreiranje rječnika za analizu, kreiran je alat naziva ‘gwordlist’ [37], koji pomoću tražilice Google preuzima zadani broj web-mjesta na kojima se javlja ključna riječ koja je napisana u datoteci s ključnim pojmovima i parametrima. Navedeni postupak se ponavlja za svaku ključnu riječ. Preuzeta web-mjesta se pročišćavaju pomoću metoda prepoznavanja prirodnog jezika (NLTK) [57], što omogućava kreiranje rječnika, koji ima uklonjene višestruke nastupe istih riječi i koji je sortiran ovisno o učestalosti nastupa pojedine riječi u svim stranicama koje je alat preuzeo. Dodatna korist ove metode u tome je da se u ključnim riječima mogu koristiti razni specijalni upiti koje tražilica Google podržava, kao što su:

- *site:foi.hr* za ograničavanje pretrage na samo jednu domenu ili *site:.hr* za ograničavanje pretrage na samo jednu vršnu domenu koja će obuhvatiti samo riječi iz stranica koje su pod određenim domenskim prostorom.
- Operator *–riječ* za isključivanje pojave rezultata koji uključuju pojam nakon kojeg se nalazi minus.
- Operator *~riječ* za uključivanje svih sinonima koje Google smatra da su mogući za zadani pojam koji se nalazi nakon tilde.
- Operator .. za definiranje raspona brojeva kao što je “*najbolji albumi 2000..2013*” gdje se uključuje raspon svih brojeva koji su povezani točkama.
- Operator * za definiranje univerzalnoga pojma kao što je “*sigurnost je **” gdje će Google pokušati vratiti sve rezultate koji počinju sa “sigurnost je” i nakon toga imaju neku riječ u nastavku.

Uz navedene specijalne upite moguće je koristiti i bilo koji upit koji tražilica Google razumije, poput nekih primjera iz rada [72]. Metoda *gwordlist* korisna je za generiranje rječnika koji se koriste kao početni skup ključnih riječi za pogađanje lozinki, tj. za korištenje u sklopu modela tipa M3. U cilju kreiranja kvalitetne liste riječi, alat/metodu *gwordlist* moguće je koristiti na način da pokušamo razviti opću listu riječi.

6.3. Korištenje alata/metode *gwordlist* za kreiranje općenitih lista riječi

Osnovni dio svake analize snage lozinki koji se ne temelji na iscrpnom pretraživanju jest korištenje rječnika odgovarajuće veličine. Ukoliko je u rječniku premalo pojmova, manja je vjerojatnost da je korisnik koristio upravo riječ koja je sastavni dio tog rječnika. Ukoliko je rječnik prevelik, analiza će trajati predugo, pogotovo ako se koriste pravila mutacije ili kombinacije nad riječima, čime će se broj kombinacija drastično povećati. Trenutno je u uporabi više rječnika, od kojih većina njih sadrži premalen ili prevelik broj riječi. U cilju rješavanja tog problema i zbog potrebe za kvalitetnim rječnikom koji je nužan u sklopu daljnjih istraživanja, alat/metoda *gwordlist* primijenit će se za generiranje rječnika koji sadrži veliki broj specifičnih riječi koje bi korisnici mogli iskoristiti kao dio svoje lozinke i koji opet ne bi trebao biti prevelik, gdje konačni odabir o veličini rječnika određuje osoba koja provodi analizu ovisno o svojim potrebama. Pregledom popularnih kategorija na *wikipediji* i rezultata popularnih pretraživanja na tražilicama, pomoću alata *gwordlist* kreiran je skup općih kategorija pojmova, prikazan sljedećom tablicom:

Tablica 5: Skupina pojmova korištena za generiranje opće liste riječi

animals	metal music	literature	rap music
art	fabrics	medicine	recreational activities
basketball	famous artists	movies	religion
best songs	famous events	MTV	restaurants
brands	famous people	national parks	rock music
business	fantasy	nature	science
cars	fashion	numbers	science fiction
celebrities	finance	outdoors	services
cities	flowers	pets	sex
classical music	food	philosophy	soccer
clothing	football	places	sociology
CNN	football clubs	planets	space
colors	golf	plants	sports
common passwords	history	politics	stars
common phrases	hobbies	pop music	tourist spots
computers	hockey	popular artists	tv
countries	hotels	popular events	tv shows
country music	informatics	popular given names	unesco world
crafts	list of artists	popular music artists	heritage
culture	list of best songs	pornography	violence
economy	list of places	products	weak passwords

Za svaki od pojmova u prijašnjoj tablici preuzeto je prvih 50 poveznica koje su se pojavile kao rezultat pojedinog pojma na tražilici Google. Unutar 24 sata, koliko je analiza, pokrenuta s brze veze Fakulteta organizacije i informatike, trajala, razvijen je rječnik sa 334.168 riječi i rječnik sa 308.211 brojeva, koji su dostupni u digitalnom repozitoriju podataka ovog rada, pod oznakom "gwl-words" za rječnik s riječima te "gwl-numbers" za rječnik brojeva. Analiza odnosa učestalosti pojmova i veličine datoteke u pojedinim rječnicima prikazana je u sljedećim tablicama, gdje je pod oznakom KU (Kumulativna učestalost pojmova) opisana kumulativna učestalost pojmova u datoteci koja je u odnosu s parametrima LUD (Linija u datoteci) i PD (Postotak datoteke). Iz tablica vidimo da rječnik "gwl-numbers" ima jednakomjerni rast, dok "gwl-words" nema. Svaki rječnik velik je oko 3 MB. Uvidom u digitalni repozitorij, vidimo da je sadržaj rječnika "gwl-words" orijentiran na ključne riječi koje bi korisnik mogao koristiti u svojim lozinkama.

Alat *gwordlist* se može koristiti za generiranje liste ključnih riječi koje ćemo kroz kroz modele tipa M3 ili za kreiranje lista riječi kojima možemo nadograditi postojeće liste riječi ili razviti nove liste riječi.

Tablica 6: Struktura rječnika *gwl-words*

KU	LUD	PD
5,00%	2	0,0005985
10,00%	5	0,0014963%
15,00%	11	0,0032918%
20,00%	25	0,0074813%
25,00%	54	0,0161595%
30,00%	103	0,0308228%
35,00%	177	0,0529674%
40,00%	284	0,0849872%
45,00%	437	0,1307725%
50,00%	642	0,1921189%
55,00%	936	0,2800986%
60,00%	1.359	0,4066817%
65,00%	1.987	0,5946111%
70,00%	2.938	0,8791985%
75,00%	4.453	1,3325633%
80,00%	7.009	2,0974480%
85,00%	12.007	3,5931029%
90,00%	24.414	7,3059060%
91,00%	29.190	8,7351272%
92,00%	35.593	10,6512293%
93,00%	44.402	13,2873285%
94,00%	57.140	17,0991836%
95,00%	75.789	22,6799095%
96,00%	104.970	31,4123435%
97,00%	159.079	47,6044983%
98,00%	217.442	65,0696656%
99,00%	275.805	82,5348328%
100,00%	334.168	100,0000000%

Tablica 7: Struktura rječnika *gwl-numbers*

KU	LUD	PD
5,00%		0,0006489%
10,00%	4	0,0012978%
15,00%	6	0,0019467%
20,00%	10	0,0032445%
25,00%	15	0,0048668%
30,00%	22	0,0071380%
35,00%	31	0,0100580%
40,00%	42	0,0136270%
45,00%	59	0,0191427%
50,00%	94	0,0304986%
55,00%	173	0,0561304%
60,00%	435	0,1411371%
65,00%	2.744	0,8902992%
70,00%	41.814	13,5666800%
75,00%	86.214	27,9723955%
80,00%	130.613	42,3777866%
85,00%	175.013	56,7835022%
90,00%	219.412	71,1888933%
91,00%	228.292	74,0700364%
92,00%	237.172	76,9511795%
93,00%	246.052	79,8323227%
94,00%	254.932	82,7134658%
95,00%	263.812	85,5946089%
96,00%	272.692	88,4757520%
97,00%	281.572	91,3568951%
98,00%	290.452	94,2380382%
99,00%	299.332	97,1191813%
100,00%	308.211	100,0000000%

7. Istraživanje i razvoj modela tipa M3

7.1. Početni koncept istraživanja

Korištenje modela tipa M3 zahtijeva uzorke obrazaca načina na koje korisnici kreiraju lozinke. Početna ideja istraživanja bila je razviti poseban skup pravila, pod nazivom MK-pravila, koja se dijele na:

1. Pravila mutacije, koja uzimaju ulazni podatak i mijenjaju ga na neki smislen način, kao što je promjena slova i u 1, e u 3 i slično
2. Pravila kombinacije riječi, koja uzimaju više različitih zapisa tipa L0 ili L1, nastalih mutacijama sukladno odgovarajućim pravilima mutiranja.

U svrhu otkrivanja mogućih pravila mutacije, koncept istraživanja temeljio se na metodi koja bi uspoređivala postojeće lozinke sa sadržajem standardnih rječnika za provjeru pravopisa, kao što su *GNU Aspell* ili *MySpell*, koji su dostupni za skoro sve svjetske jezike. Otkrivanje mutacije bi se provodilo pomoću Levenshteinove ili Damerau-Levenshteinove distance između niza znakova, koje mogu otkriti dodavanje, supstituciju i uklanjanje znakova na korjenovanim (“stemiranim”) riječima. Pravila mutacije bila bi zapisana u formi pogodnoj za primjenu nad ulaznim skupovima znakova. Pravila kombinacije trebalo bi otkriti pomoću pravila mutacije, upotrebom rječnika i zapisa tipa L0, odnosno L1. Pod zapisom tipa L0 smatramo grupiranje iste klase znakova, kao što su slova, brojevi ili specijalni znakovi, uz jednu oznaku za pojedinu klasu, gdje za mala slova koristimo oznaku “a”, za velika slova oznaku “A”, oznaku “0” za brojeve i oznaku “#” za specijalne znakove, pri čemu jednom oznakom označavamo jedan ili više znakova. Kao primjer, lozinka !!!pero123 bila bi prikazana pomoću zapisa tipa L0 kao #a0. Zapis tipa L1 zapis istovjetan je zapisu tipa L0, osim što na način opisan za zapise tipa L0 označava *svaki* znak lozinke. Primjerice, lozinka !!!pero123 pomoću oznake tipa L1 bila bi zapisana kao ###aaaa000. Zapisi tipova L0 i L1 korisni su za označavanje klasa lozinki i za grupiranje lozinki po sličnosti.

Odabir riječi iz rječnika temelji se na generalizaciji zapisa tipa L0, gdje se za zapis "a" može spajati više mogućih kandidata iz rječnika do nekog praga spajanja. Dodatna korist od ovakve analize u tome je da se prilikom analize ulazni rječnici sortiraju po učestalosti korištenja riječi

sukladno ukupnom broju pojave svake pojedine riječi kao elementa neke lozinke, što ubrzava buduće analize.

7.2. Odabir rječnika

Inicijalna ideja istraživanja bila je koristiti rječnike iz kolekcije *GNU Aspell* [4] koji su javno dostupni i slobodni za korištenje pod licencom *GNU LGPL*. *Aspell* je dostupan za skoro sve popularne aktivne svjetske jezike, uključujući i hrvatski. *Aspell* nije jedan rječnik za jedan jezik, već se svodi na korištenje datoteke za jezik uz koju se koristi i datoteka afiksa koja definira moguće prefikse ili sufikse za određeni korijen riječi, definiran u rječniku.
Primjer sadržaja datoteke hr_HR.dic:

```
aktualizirana/A
aktualiziranoj
aktualiziranom
...
Amerikanci
amerikanizirati/GCED
amerikanskim
```

Primjer sadržaja datoteke hr_HR.aff:

```
SFX A 0 a [^aeiou]
SFX A 0 u [^aeiou]
SFX A 0 e [^aeiou]
SFX A 0 om [^aeiou]
```

U ovom slučaju prikazani su svi mogući sufiksi (SFX) korijena riječi, te su dodatno opisani uvjeti vezani za korištenje afiksa, poput mogućnosti rezanja korijena za dodavanje afiksa, mogućnosti korištenja afiksa u sprezi s drugim afiksima, uvjeti pod kojima se dotični afiks smije koristiti, te mnoštvo drugih opcija. Kao takav, *Aspell* koristi relativno malu količinu diskovnog prostora. Za hrvatski jezik, glavni rječnik je velik samo 2,2 MB sa 1,2KB afiksa. Složenost algoritma za provjeravanje pravopisa mala je i dizajn *Aspell*-a omogućuje brzo provjeravanje teksta s gramatičkim pogreškama. Tako dizajniran rječnik, koji osigurava veliku učinkovitost provjere valjanosti teksta, nije pogodan za korištenje u svojstvu liste riječi za generiranje lozinki, jer bi to zahtijevalo kreiranje sustava koji bi rječnik proširio svim kombinacijama valjanih afiksa.

Sljedeća razmatrana alternativa bila je korištenje Googleovih rječnika za provjeru pravopisa koji se koriste unutar Googleovih aplikacija, kao što je web-preglednik *Chrome*. Nažalost, ti

rječnici dostupni su pod nepoznatom licencom i nisu jasni dozvoljeni uvjeti korištenja bez konzultiranja Googleovih službi. Uvidom u izvorni kod web-preglednika *Chromium*, koji koristi istu jezgru za pregledavanje sadržaja i provjeru pravopisa, utvrđeno je da se rječnici sažimaju u posebnu vrstu digitalnog stabla i da postoji izvorni kod za provjeru riječi prema tom digitalnom stablu. Izvorni kod koji sažima riječi u stablo nije dostupan, i, da bi se omogućilo proširenje stablo u rječnik, trebalo bi ga saznati pomoću metoda reverznog inženjeringa.

Alternativa rječnicima za provjeru pravopisa su lingvistički korpusi koji su dostupni za većinu jezika i koji su razvijeni u svrhu lingvističkih istraživanja. Lingvistički korpusi su korisni, ali, nažalost, svaki od njih javlja se u više verzija. Za hrvatski jezik trenutno postoji korpus koji održava *Institut za hrvatski jezik i jezikoslovlje* [31], kao i korpus koji održava *Zavod za lingvistiku filozofskog Fakulteta u Zagrebu* [21]. Korpusi riječi predstavljaju dobru mogućnost za formiranje baznog rječnika, jer je slijedom njihova broja i volumena moguće sastaviti opsežan skup rječnika, strukturiran izdvajanjem riječi iz raznih korpusa. Trenutno najveći i najbogatiji skup korpusa za razne jezike održava *Google Research* u obliku *Google-ngram-korpusa* [50]. Za izgradnju rječnika potreban je *1-gram-korpus*, koji sadrži pojavu samo jedne riječi. Izbace li se svi numerički *1-gram*-i i koriste samo *1-gram*-i koji počinju slovima abecede, dolazi se do datoteka *1-gram*-a veličine od 4570 MB, uz dodatak da su sve datoteke sažete algoritmom *gzip*.

Svaka datoteka u *1-gram-korpus*-u sadrži sve elemente koji počinju jednim te istim slovom abecede, a sadržaj datoteke definiran je sljedećom strukturom:

```
security._ADJ	2008	19	19
```

Označena je riječ *security* kao pridjev pojavila se 2008. godine 19 puta u ukupno 19 knjiga iz engleskog govornog područja. Zanimljivo je da u *Google-ngram-korpus*-ima zapravo postoji pet korpusa engleskog jezika, pod nazivima “English”, “English One Million”, “American English”, “British English” i “English Fiction”, što zapravo znači da bi trebalo preuzeti sve korpuse i preraditi ih u primjerenu listu. *Googleovi* korpusi razvijeni su iz skeniranih i digitaliziranih knjiga iz projekta *Google Books*, kao što je prikazano u radu [50] i sadrže riječi koje se pojavljuju od 1800 do 2008 godine. Te knjige nemaju neke određene riječi kao što su imena glazbenih sastava ili popularnih zvijezda, imena likova iz video igara, knjiga ili filmova koji su popularni kao dio lozinke, što se može vidjeti iz skupa lozinki za učenje. Da

bi se razvio razumno veliki korpus koji sadrži skoro sve popularnije riječi iz više jezika, a uključuje i lingvistički nebitne elemente, poput imena slavnih osoba ili likova iz knjiga, bez zahtjeva za nekoliko TB diskovnog prostora, trebalo je primijeniti drugačiji koncept. Jedan od najpopularnijih otvorenih izvora podataka na Internetu je web-enciklopedija "Wikipedia" koja predstavlja veliki repozitorij enciklopedijskih članaka s bogatim fondom tema i riječi. Kao takva predstavlja vrlo velik lingvistički korpus, bogat raznovrsnim pojmovima.

Kako bi se moglo utvrditi na koji način su korisnici kreirali svoje lozinke, potreban je veći korpus riječi koji sadrži veliki broj pojmova, pri čemu tekst iz *Wikipedije* predstavlja dobru bazu pojmova sa širokim korpusom riječi i pojmova koji uključuju razna područja, što predstavlja dobru osnovicu za identifikaciju elemenata u lozinki koje je koristio korisnik.

7.3. Kreiranje liste riječi za analizu čistog teksta lozinke iz sadržaja *Wikipedije*

Prednost *Wikipedije* je u tome što je cijeli njezin sadržaj dostupan pod licencom *CC-BY-SA Creative Commons* [10]. Također, u formatu XML, sažetom pomoću algoritma *bzip*, javno su dostupne i sigurnosne kopije, tj. slike [70], cjelokupnog sadržaja Wikipedije, za sve jezike za koje ta virtualna enciklopedija postoji. Prednost ovog pristupa je da Wikipedija sadržava različite pojmove, specifične za određeni jezik, kao što su imena popularnih ličnosti, specifične izraze, izmišljene likove i mjesta, i druge pojmove koji ne bi bili uključeni u klasični lingvistički korpus.

Problem sa svim boljim lingvističkim korpusima jest u zahtjevnosti njihove analize. Engleski korpus Wikipedije, preuzet 14. 08. 2012. g., koji je zapravo jedna XML datoteka, u kojoj su sadržani samo tekstovi tematskih članaka, u raspakiranom (nesažetom) formatu ima veličinu od 40,035 GB. Veličina tog sadržaja nije nemoguća za analizu, ali je sama analiza vrlo nepraktična. Za potrebe stvaranja korpusa za identifikaciju riječi, preuzete su sljedeće sažete kopije Wikipedije:

1. Engleska Wikipedija – 9.7 GB (enwiki-latest-pages-articles-multistream.xml.bz2)
2. Njemačka Wikipedija – 2.9 GB (dewiki-latest-pages-articles.xml.bz2)
3. Hrvatska Wikipedija - 163 MB (hrwiki-latest-pages-articles.xml.bz2)
4. Francuska Wikipedija – 2.3 GB (frwiki-latest-pages-articles.xml.bz2)
5. Španjolska Wikipedija – 1.6 GB (eswiki-latest-pages-articles.xml.bz2)

6. Talijanska Wikipedija – 1.6 GB (itwiki-latest-pages-articles.xml.bz2)
7. Nizozemska Wikipedija – 862 MB (nlwiki-latest-pages-articles.xml.bz2)

Svaki komprimirani rječnik bio je analiziran na način da je pronađen blok teksta koji pripada članku, te je on pomoću metode NLTK i parsiranja teksta pretvoren u čisti tekst. Pojmovi za analizu spremaju se u tablici baze podataka koja bilježi sljedeće elemente:

1. Pojam tj. riječ koja se pojavila u čistom tekstu
2. Listu s oznakama rječnika gdje je pojam pronađen
3. Ukupni broj pojava u cijeloj bazi, tj. korpusu

Za svaki pojam praćena je učestalost njegovog pojavljivanja. Ukoliko oznaka rječnika koji trenutno analiziramo ne postoji u listi s oznakama, njegova se oznaka dodaje u listu pojmova, da bi se ubuduće znalo da se taj pojam nalazi u rječniku.

Za potrebe stvaranja baze podataka odabran je sustav za upravljanje bazom podataka PostgreSQL, posebice zbog njegovih funkcionalnosti u pogledu pretraživanja s uključenim posebnim znakovima te podrškom za zapise podataka pomoću mehanizma *hstore*, koji kreiranje zapisa omogućava u obliku ključ:vrijednost i ne inzistira na njihovoj strogoj strukturi, što je posebno pogodno za bilježenje mutacija i obrazaca sukladno kojima su korisnici kreirali svoje lozinke. Slijedom dobrih osobina u smislu pohranjivanja slabo strukturiranih sadržaja, u razmatranje su bili uzeti i sustavi *CouchDB* [1] te *MongoDB* [18]. Međutim, sustavi *CouchDB* i *MongoDB* nemaju napredne mogućnosti analize tekstova, poput izvođenja upita u pogledu sličnosti segmenata teksta, a tijekom eksperimentalne analize iskazale su određeni niz nedostataka, kao što je brzi rast baze *CouchDB* slijedom svojstva MVCC, odnosno potrebe za posebnom brigom o pohranjivanju zapisa u slučaju baze *MongoDB*, koja ne jamči da će svi zapisi biti pohranjeni ukoliko nije aktivan siguran način rada. Slijedom korištenja tehnike obrade podataka *MapReduce* [13], koja je pogodna za obradu podataka na više distribuiranih sustava za upravljanje bazom podataka, obje te baze usmjerene su prema pohrani podataka ili distribuiranim sustavima. Baza P*ostgreSQL* također je iskazala i neke druge prednosti u postupku pretraživanja podataka. Cijela tablica s bazom riječi iz 7 rječnika veličine je 11 GB i *PostgreSQL* omogućava pretraživanje s operaterima sličnosti, kao što je operator % koji zamjenjuje nula ili više znakova u riječi, te operatorom _ koji zamjenjuje pojedinačni znak, što je korisno želimo li identificirati mutacije i što ćemo pobliže opisati u sljedećem poglavlju. Dodatna prednost *PostgreSQL*-a bila je u korištenju

sustava *pg_trgm* koji omogućuje indeksiranje skupova riječi pomoću GIN-indeksa i time značajno ubrzava upite. Kao primjer možemo prikazati sljedeći upit:

```
unhash=# explain analyse select * from lang_dicts where word like
'%prent%';

QUERY PLAN
------------------------------------------------------------------------------
 Seq Scan on lang_dicts  (cost=0.00..417048.01 rows=1850 width=48) (actual
time=0.442..5794.316 rows=313 loops=1)
   Filter: (word ~~ '%prent%'::text)
 Total runtime: 5794.602 ms
(3 rows)
```

Za izvršenje gornjeg upita potrebno je 5,794 sekundi. Isti upit, niže prikazan, nakon kreiranja GIN-indeksa s trigramima,

```
unhash=# create extension pg_trgm;
unhash=# CREATE INDEX lang_dicts_trgm on lang_dicts using gin(word
gin_trgm_ops);
```

traje samo dio sekunde:

```
unhash=# explain analyse select * from lang_dicts where word like
'%prent%';
QUERY PLAN
------------------------------------------------------------------------------
 Bitmap Heap Scan on lang_dicts  (cost=70.72..7082.59 rows=1899 width=48)
(actual time=226.893..229.196 rows=313 loops=1)
   Recheck Cond: (word ~~ '%prent%'::text)
   ->  Bitmap Index Scan on lang_dicts_trgm  (cost=0.00..70.24 rows=1899
width=0) (actual time=226.803..226.803 rows=701 loops=1)
         Index Cond: (word ~~ '%prent%'::text)
 Total runtime: 229.261 ms
(5 rows)
```

Ukoliko analiza zahtijeva obradu velikog broja elemenata, kao što je slučaj s analizom liste lozinki za učenje, kada je početni skup riječi za učenje velik, razlike u vremenu postaju jako bitne.

Drugi bitni koncept bio je pokušaj kreiranja lista tipa M2 pomoću Wikipedije. Urednici ili autori članaka prilikom njihovog stvaranja ili ažuriranja, istima mogu pridodati određeni broj kategorija. Ideja koju je autor slijedio bila je mogućnost da osoba koja analizira sigurnost lozinki odabere kategorije pojmova i da izrađena metoda kreira listu riječi iz zadanih kategorija. Kreiran je jednostavan alat za provjeru kategorija i otkriveno je sljedeće:

1. 14.8.2012 Wikipedia je imala 2.487.112 jedinstvenih kategorija.

2. 14.8.2012 Wikipedia je imala 7.952.187 jedinstvenih skupova tj. kombinacija kategorija koje su se koristile u svim člancima.

Veliki problem je u tome da su kategorije po Wikipedijinom standardu definirane hijerarhijski u obliku stabla, ali, nažalost, autori članaka i urednici prilikom uređivanja članaka toga se dogovora nisu držali, što je rezultiralo mnoštvom petlji u skupinama kategorija, a nered je dodatno povećan gotovo svakodnevnim mijenjanjem samog sustava kategorija pojmova. Problem također nastaje i slijedom nepostojanja fiksne strukture kategorija, kao i činjenicom prisutnosti velikog broja članaka bez pridružene pojmovne kategorije, odnosno članaka kojima je pridružen nelogičan broj kategorija, što se vidi u sljedećoj tablici, u kojoj je broj članaka u Wikipediji prikazan po broju kategorija, s prikazanim postotnim odnosom broja članaka i ukupnog broja članaka.

Tablica 8: Broj kategorija po članku Wikipedije

Br. Kat	Članaka	%	Br. Kat	Čl.	%	Br. Kat	Čl.	%
0	2721534	34,22372%	30	500	0,00629%	60	7	0,00009%
1	1399735	17,60189%	31	403	0,00507%	61	2	0,00003%
2	1190350	14,96884%	32	344	0,00433%	62	2	0,00003%
3	836046	10,51341%	33	258	0,00324%	63	3	0,00004%
4	509392	6,40568%	34	244	0,00307%	64	3	0,00004%
5	351244	4,41695%	35	155	0,00195%	65	3	0,00004%
6	251118	3,15785%	36	152	0,00191%	66	2	0,00003%
7	181267	2,27946%	37	121	0,00152%	67	6	0,00008%
8	130587	1,64215%	38	129	0,00162%	68	2	0,00003%
9	95055	1,19533%	39	94	0,00118%	69	3	0,00004%
10	68803	0,86521%	40	76	0,00096%	71	5	0,00006%
11	50086	0,62984%	41	69	0,00087%	72	2	0,00003%
12	37615	0,47301%	42	51	0,00064%	73	1	0,00001%
13	28370	0,35676%	43	52	0,00065%	77	1	0,00001%
14	21664	0,27243%	44	35	0,00044%	83	2	0,00003%
15	16778	0,21099%	45	33	0,00041%	84	2	0,00003%
16	13108	0,16484%	46	45	0,00057%	86	1	0,00001%
17	10173	0,12793%	47	28	0,00035%	87	3	0,00004%
18	8063	0,10139%	48	20	0,00025%	88	2	0,00003%
19	6408	0,08058%	49	23	0,00029%	93	2	0,00003%
20	5081	0,06389%	50	29	0,00036%	108	1	0,00001%
21	3906	0,04912%	51	12	0,00015%	127	1	0,00001%
22	3138	0,03946%	52	17	0,00021%	134	1	0,00001%
23	2399	0,03017%	53	22	0,00028%	149	1	0,00001%
24	1969	0,02476%	54	14	0,00018%	150	1	0,00001%
25	1524	0,01916%	55	9	0,00011%	171	1	0,00001%
26	1264	0,01589%	56	6	0,00008%	175	1	0,00001%
27	1015	0,01276%	57	7	0,00009%	201	1	0,00001%
28	798	0,01003%	58	2	0,00003%	202	2	0,00003%
29	678	0,00853%	59	4	0,00005%	203	1	0,00001%

Dodatni nedostatak Wikipedije leži u potrebi opetovanog preuzimanja novih kopija baze s Wikipedijinih poslužitelja. Analiza XML-datoteka, sažetih programom *bzip*, izuzetno je spora, kategorije unutar Wikipedije vrlo su raspršene i nemaju strogu hijerarhiju, te stoga kreiranje takvih sadržaja nije bilo praktično i strukturu razapinjućeg stabla kojom bi se izbjegavale petlje u sustavu kategorija bilo bi teško razviti. Zbog navedenih razloga i nepraktičnosti za krajnjeg korisnika, testirana je mogućnost korištenja alata/metode *gwordlist*, koja se, zbog mogućnosti pretraživanja sadržaja pomoću tražilice, pokazala boljom za

generiranje specifičnih rječnika. Alat/metoda *gwordlist* pokazala je da za zadane ključne riječi stvara puno raznovrsniji skup riječi.

7.4. Implementacija i testiranje predloženog koncepta modela tipa M3 i identifikacija njegovih nedostataka

Zbog nemogućnosti identifikacije nekih skupina lozinki, polazni koncept istraživanja, opisan na početku ovog rada i u prijavi doktorske disertacije, pokazao se manjkavim. Također, sličnu metodu, imena PACK, koja ima puno više ograničenja, objavio je na konferenciji Passwords 2012 P. Kacherginsky [33]. Ni ta metoda nije dovoljno dobra za potrebe doznavanja pravila sukladno kojima korisnici kreiraju svoje lozinke nad velikim, općenitim listama lozinki. PACK metoda pokazuje identične probleme prilikom pokušaja učenja nad listama lozinki. Primjerice, korištenje distance za identifikaciju mutacije problematično je. Moguće je opisati nekoliko slučajeva čestih lozinki koje nije moguće identificirati pomoću predloženog koncepta mjera udaljenosti, što je zahtjevalo prilagodbu metode u cilju identificiranja što većeg broja tipova lozinki.

Prvi slučaj su lozinke koje su vezane uz slabe uzorke. Klasa tih lozinki nazvana je "weak_pattern" Primjeri takvih lozinki mogu biti:

- Lozinke koje predstavljaju jednostavne nizove jednog znaka:
    - 111111
    - 000000,
    - 999999999
    - aaaaa
    - .....
- Lozinke koje koji predstavljaju nizove ponavljajuće grupe znakova
    - blablabla
    - hehehe
    - yoyoyo

Drugi slučaj su lozinke koje predstavljaju uzorke koji slijede povezani niz znakova na tipkovnici. Klasa tih lozinki nazvana je "keyboard_pattern" Takve lozinke mogu izgledati

prilično sigurne i snažne, ali predstavljaju uzorak koji se dobiva slijednim postupkom na tipkovnici. Takve su lozinke učestale u listama lozinki. Primjeri takvih lozinki mogu biti:

- qwertyuiop
- q1w2e3r4t5y6u7i8
- mki*&^yhju&^%$#
- r$t%y^u&

Treći česti slučaj mogu biti lozinke koje imaju umetke, čiju klasu označavamo oznakom "weak_inserts", Takve lozinke imaju jednostavne dodatke ubačene u slabu lozinku. Primjeri takvih lozinki mogu biti:

- p.i.m.p
- x1x2x3x4
- 'i'i'i'i
- f.r.i.e.n.d.s
- j1j2j3j4j5

Četvrti česti slučaj su lozinke koje imaju određeni broj simetričnih elemenata, čiju klasu označavamo sa "symmetric_elements",

- lol123lol
- *friends*
- 123hello123
- 69sexy69
- 123password123

Tome se mogu dodati dvije popularne klase lozinki koje su identificirane već prije, a to su kombinacije i mutacije. Prve četiri kategorije također mogu biti kombinirane s ostalim kategorijama, uz korištenje pravila mutacije i kombinacije. Problem s navedenim kategorijama lozinki jest da loše utječu na njihovu moguću identifikaciju. Naime, teško je razlikovati mutaciju od kombinacije, a još teže je razlikovati njihov spoj, slijedom kojega dolazi do drastičnog rasta pogrešaka klasifikatora.

Dodatni problem klasifikacije je da je skoro nemoguće klasificirati sve lozinke, jer napredni korisnici mogu izmisliti kompleksne sheme za kreiranje pseudoslučajnih lozinki velike

duljine. Zbog toga, metode za identifikaciju postupaka stvaranja lozinki mogu jedino biti heurističke, slijedom čega je poželjno koristiti velik i reprezentativan skup lozinki za učenje, kojim se može postići veća generalizaciju obuhvata lozinki i smanjiti pogreške identifikacije.

Jedan od pokušaja implementacije i neutralizacije problema identifikacije mutacije i kombinacije, opisanih u prijašnjem poglavlju, temeljio se na korištenju Needleman–Wunsch-ova algoritma [51], koji se koristi u bioinformatici za poravnavanje proteinskih sljedova i DNA sljedova. Glavni koncept predložene metode bio je sljedeći:

1. Za svaku lozinku koju želimo analizirati, pronađemo sve riječi u rječniku za analizu čistog teksta, za koje vrijedi da je vrijednost poravnanja riječi prema Needleman-Wunsch-ovu algoritmu, u odnosu na lozinku koju analiziramo, bez rupa u poravnanju koje su veće od 3 znaka u kontinuitetu, gdje je broj 3 konstanta gp.
2. Svaka riječ u rječniku za identifikaciju ima pridruženu oznaku učestalosti korištenja u korpusu iz kojeg je lista riječi stvorena.

   Riječi koje smo otkrili u koraku 1 sortiramo prema najvećoj učestalosti korištenja u korpusu.
3. Za svaku riječ u sortiranom skupu riječi provjeravamo udaljenost prema Needleman-Wunsch-ovu algoritmu u odnosu na lozinku koju analiziramo:
   a. Ako poravnavanje nema rupa i odgovara dijelu ili cijeloj lozinki, taj dio lozinke možemo označiti poznatim.
   b. Pokušamo otkriti mutaciju pomoću razlike moguće riječi i dijela lozinke za riječi koje imaju najveću učestalost, najmanji broj rupa i najveću duljinu predložene riječi.
   c. Ostale elemente kao što su brojevi ili specijalni znakovi označimo poznatima jer ih ne možemo identificirati.
   d. U odnosu na poznate dijelove lozinke označimo dijelove koji su pisani velikim slovima.
   e. Ako su svi dijelovi lozinke poznati i nema dijelova koje treba identificirati, postupak je završen.

Needleman–Wunsch-ov algoritam ima kvadratnu složenost. Uz primjenu klasifikatora, metrike udaljenosti i upite u PostgreSQL bazu podataka te uz implementaciju proširenja *pg_similarity* [56], za postavljanje upita u bazu PostgreSQL nad velikim skupom lozinki za učenje, predložena implementacija bila je vrlo spora i dolazilo je do velikog broja preklapanja

riječi i pogrešnih klasifikacija. Upravo veliki broj pojmova koji smo htjeli imati u rječniku za identifikaciju zbog boljeg raspoznavanja riječi koje korisnici koriste u svojim lozinkama, otežalo je klasifikaciju, jer je bilo moguće pronaći više pojmova koji se preklapaju i riječi koje imaju preklapanja. Eksperimentirano je s metodama smanjivanja značaja kratkih riječi i preklapanja, gdje se željelo pronaći raspored koji bi poboljšao implementaciju. Niti ova implementacija nije mogla identificirati prije opisane četiri klase lozinki, te uz velika preklapanja i visoku razinu pogreške nije bilo moguće identificirati većinu lozinki. Iz tih razloga trebalo je razviti algoritam koji će moći prepoznati više klasa lozinki i omogućiti lakše dodavanje novih klasa lozinki i načina njihove identifikacije.

7.5. Prijedlog novog modela i implementacija metode sita

Zbog velike raspršenosti lozinki u skupu za učenje, bilo je razumno pokušati implementirati algoritam po principu "podjeli pa ovladaj" [65]. Drugi problem koji se susreće prilikom identifikacije lozinki jest da je broj načina na koji ih je moguće sastavljati vrlo velik, što znači da bi se i pripadajući skup klasifikatora morao s tim valjano nositi. Analizom literature i znanstvenih radova koji se tom problematikom bave, nije pronađena metoda koja bi zadovoljila toliku općenitost i bila praktična za primjenu nad skupom za učenje. Jedna od razmatranih mogućnosti bilo je korištenje genetskih algoritama ili neuralnih mreža [19], što je odbačeno zbog nedostatne brzine procesiranja i složenosti rješenja.

Ipak, jedan dobitak od procjene upotrebe neuralnih mreža za potrebe otkrivanja lozinki bila je u tome da je koncept povezanih neurona u mreži dao temelj za kreiranje metode koja povezuje klasifikatore i procjenjuje vjerojatnije mogućnosti građe lozinki. Predloženi koncept autor je nazvao *metodom sita*. Ona radi na konceptu određenog broja specijaliziranih klasifikatora, organiziranih u hijerarhiju, koji provode klasifikaciju i komuniciraju pomoću skupa poruka koji opisuje sam postupak klasifikacije. U trenutnoj postavci, metoda sita koristi sljedeće razine klasifikacije:

1. Globalnu klasifikaciju koja pokušava identificirati četiri slabe klase lozinki: slabe uzorke, nizove na tipkovnici, umetke i simetrične elemente
2. Raščlanjivanje po tipu L0, koje pokušava identificirati jednostavne kombinacije
3. Mutacijski klasifikator koji pokušava identificirati mutacije i kombinacije.

Bitan koncept je *poruka*, u kojem svaka metoda vraća poruku u polju (eng. array), gdje je svaki element polja *katalog* (eng. dictionary) koji sadrži rezultat klasifikatora, a zadnji element polja mogu koristiti drugi klasifikatori. To polje zovemo *dnevnički zapis* (eng. log) i ono na kraju analize čuva postupak dekompozicije lozinke, tj. "recept" sukladno kojem je korisnik tu lozinku sastavio. Krajnji rezultat dnevnika moguće je pretvoriti u MK-pravilo, koje će koristiti metoda za generiranje predložaka. Svaki element dnevničkog polja ima jedinstveni način označavanja sljedećeg elementa oznakom 'x', koju mogu preuzeti drugi klasifikatori i nastaviti daljnju dekompoziciju lozinke. Prema tome, klasifikatori koji su viši na hijerarhiji olakšavaju posao nižim klasifikatorima. Zbog praćenja broja pojedinih modela i prikupljanja podataka radi stvaranja modela o tome kako korisnici kreiraju svoje lozinke, takav dnevnik mora se spremiti u bazi podataka.

Metoda sita sastoji se od tri glavna klasifikatora, *globalni klasifikator* ima za zadaću identificirati vrlo slabe lozinke, te se nakon njega pokušavaju druge dvije procjene: *L0-raščlanjivanje* i *mutacijski klasifikator*, koji su heuristike i potrebno je ocijeniti koja bi procjena producirala najvjerojatnije rješenje. Metodu sita možemo prikazati pseudokodom:

```
Sito (l):
      log = []
      log += globalni_klasifikatori(l)
      ako log sadrži slabi uzorak ili niz na tipkovnici:
            vrati log, Kraj analize
      inače:
            l = neklasificirani dio lozinke

      l0_rasc = l0_raščlanjivanje(l)
      Za svaki element e u l0_rasc:
            l0_tok += terminalna_klasifikacija(e)
            l0_score = izračunaj_ocjenu_vjerojatnosti(l0_klas)

      mut_tokovi = mutacijski_klasifikator(l)
      mut_tok = procjeni_tokove(mut_tokovi)
      mut_score = izračunaj_ocjenu_vjerojatnosti(mut_tok)

      ako je l0_score >= mut_score:
            log += l0_tok
      inače:
            log += mut_tok
      vrati log
```

Da bismo prikazali rad metode sita, moramo prikazati i funkcije koje ta metoda koristi. Prvi klasifikator, koji otkriva slabe lozinke, jest globalni klasifikator koji se sastoji od 4 glavne funkcije:

1. Pronađi slabe uzorke
2. Pronađi nizove na tipkovnici
3. Pronađi umetke
4. Pronađi simetrične elemente

Da bi se razumio koncept rada globalnoga klasifikatora, navedene funkcije potrebno je opisati:

```
pronađi_slabe_uzorke(l,t):
      ls = duljina(l)
      lc = broj_jedinstvenih_znakova_u(l)
      ako je lc < ls:
            razdvoji l sljedno na blokove duljine lc
            ako su svi blokovi jednaki:
                  vrati {‘el’:l,’tip’:’slabi_uzorak’}
      vrati 0
```

Funkcija “pronađi slabe uzorke” provjerava ponavlja li se u lozinki l neki skup znakova. Jednostavan način provjere toga jest provjera broja jedinstvenih znakova koje lozinka koristi. Ukoliko je taj broj manji od veličine lozinke, lozinku možemo podijeliti na slijedne blokove duljine jednake broju jedinstvenih znakova u njoj. Ukoliko su ti blokovi identični, riječ je o slabom uzorku u lozinki.

```
pronađi_niz_na_tipkovnici(l):

      kbd = {'1234567890':0,
             '!@#$%^&*()':0,
             'abcdefghij':0,
             'qwertyuiop':0,
             'qwertzuiop':0,
             'asdfghjkl':1,
             'zxcvbnm,./':2,
             'yxcvbnm,./':2}

      ako je duljina(l) <= 1:
            vrati 0

      E = pretvori_sva_slova_u_mala(l)
      za svaki znak l u E:
            za svaki red r u kbd:
                  Ako je l pronađen u r:
                        ako je l prvi znak u nizu:
                              mz = mjesto znaka l u r (indeks)
                              br = kbd[r]
```

```
                              t0 = (mz,br)
                        inače:
                              mz = mjesto znaka l u r (indeks)
                              br = kbd[r]
                              t1 = (mz,br)
                              ako je vektorska udaljenost između t0,t1>0:
                                    vrati 0
                              inače
                                    t0 = t1
      vrati {‘el’:l,’tip’:’niz_na_tipkovnici’}
```

Funkcija “pronađi niz na tipkovnici” koristi strukturu podataka *katalog* (eng. dictionary), gdje kao ključ držimo niz znakova u istom redu tipkovnice, a kao vrijednost uz ključ oznaku retka na tipkovnici, kao što vidimo u katalogu *kbd*. U katalogu *kbd* dodani su reci, tako da odgovaraju dvama najčešćim tipovima tipkovnice koji imaju raspored *qwerty* ili *qwertz* te su pod istim indeksom uključeni i specijalni znakovi koji nastaju korištenjem tipke *shift* i brojeva na tipkovnici. Kako bi se otkrili sljedovi znakova na tipkovnici, svaki znak prikazuje se kao točka s vrijednošću položaja tog znaka u nizu i broja retka u kojem se taj znak nalazi. Ukoliko dvije točke imaju udaljenost od jedne tipke između svaka dva slijedna znaka u cijeloj lozinki, znači da je taj uzorak moguće ispisati slijednim nizom na tipkovnici, što je za stvaranje lozinki loša praksa. Ovaj modul trenutno prepoznaje samo dva najpopularnija rasporeda tipkovnica: *QWERTY* i *QWERTZ*. U upotrebi su još neki rasporedi, kao *AZERTY* (za francuska govorna područja), *FGĞIOD* (za turska govorna područja), *JCUKEN* (za ruska govorna područja) te *DVORAK* i *COLEMAK* za napredne korisnike koji žele raspored tipkovnice koji im omogućava što veću brzinu tipkanja. Svaki od navedenih rasporeda trebao bi posebni klasifikator, ali problem je da kod nekih riječi kao što su “trees” ili “wasser” uzorci iz tipkovnice kreiraju lažne pozitive.

```
pronađi_umetke(l):

      p1 += svaki znak iz l gdje je indeks znaka u l parni broj
      p2 += svaki znak iz l gdje je indeks znaka u l neparni broj

      ako se p1 sastoji samo od jednog tipa znaka:
            w = p1
            x = p2
      inače ako se p2 sastoji samo od jednog tipa znaka:
            w = p2
            x = p1
      inače
            vrati 0

      vrati {‘el’:w, ’x’:x, ’tip’:’umetci’}
```

Da bi se otkrili mogući umeci u lozinku, ona se mora podijeliti u dvije liste. Jedna lista uključuje samo znakove čiji je indeks, tj. pozicija u listi, paran broj, a druga lista uključuje znakove gdje je taj indeks neparan broj. Na taj način, ovisno o tome je li korisnik lozinku započeo umetkom ili tekstom, u listama p1 i p2 nalaziti će se samo umeci ili čisti tekst koji je bio podloga za umetanje. Nakon toga mora se provjeriti u kojem polju se nalazi samo jedan tip znaka. Polje u kojem je to slučaj upravo predstavlja umetke, a drugo polje sadrži čisti tekst za daljnju klasifikaciju.

Za pronalazak simetričnih elemenata, koristit ćemo koncept l0-raščlanjivanja, gdje određenu lozinku želimo slijedno razdvojiti u elemente koji odgovaraju tipu znakova. Raščlanjivanje tipa L0 korisno je za raščlanjivanje slova od brojki i specijalnih znakova, što znači da bi lozinka !!!pero22 bila slijedno razdvojena na !!!, pero, 22, što je korisno u cilju identifikacije dodataka na neku riječ. Pomoću l0-raščlanjivanja, postupak za otkrivanje simetričnih elemenata izgleda na sljedeći način:

```
pronađi_simetrične_elemente(l)
      l0 = l0_raščlanjivanje(l)

      l0r =  l0_raščlanjivanje( unatraške(l) )

      ako je duljina l0 == 1:
            ako je l0 == l0r:
                  vrati 0  # l je palindrom.

      za b1, b2 u l0, l0r:
            ako su blokovi b1 i b2 identični:
                  sim += b1
                  m = 0
            inače ako su b1 i unatraške(b2) identični:
                  sim += b1
                  m = 1
            inače:
                  x = b1
      ako je sim != '':
            vrati {'el':sim, 'x':x, 'zrcalno':m}
      inače
            vrati 0
```

Lozinku l raščlani se u dva polja, l0 koje ima normalan poredak lozinke i l0r gdje je lozinka l okrenuta unatraške. Nakon toga, može se uspoređivati blok po blok, l0 i l0r. Ako su blokovi identični, onda se može provjeravati sljedeći blok da bi se otkrio najveći broj simetričnih elemenata. Naravno, mora se provjeriti i jesu li, u situaciji kad je jedan blok postavljen

unatraške jednak drugom bloku, oni identični. Ako jesu, to je znak da je korisnik koristio zrcaljenje simetričnih elemenata, kao u primjeru lozinke 123pero321.

Navedene klasifikatore moguće je spojiti u globalni klasifikator:

```
globalni_klasifikator(l):
      log = ‘’
      x = pronađi_slabe_uzorke(l)
      ako je x != 0:
            vrati x
      x = pronađi_nizove_na_tipkovnici (l)
      ako je x != 0:
            vrati x
      x = pronađi_umetke(l)
      ako je x != 0:
            log += x
      x = pronađi_simetrične_elemente(l)
      ako je x != 0:
            log += x
      z = zadnji_element_u(log)
      log += globalni_klasifikator(z)

      vrati log
```

Globalni klasifikator prvo pokušava otkriti dvije najjednostavnije klase lozinki, slabe uzorke i nizove na tipkovnici. Ukoliko je jedna od te dvije klase pronađena, to je kraj analize. Ukoliko je moguće da u lozinki postoje dodani umeci ili simetrični elementi, njihovi klasifikatori će ih detektirati i dodati poruke o tome u dnevničko polje, i ostatak lozinke, tj. ostatak čistog teksta s uklonjenim elementima, predati na daljnju klasifikaciju. Nakon prve iteracije, globalni klasifikator radi još jednu iteraciju, s namjerom odvajanja slučajeva u kojima postoji ugniježđen skup slabih klasa, kao što je kombinacija simetričnih elemenata i niza na tipkovnici, i sl. Ukoliko je druga iteracija neuspješna, globalni klasifikator predaje dnevnički zapis i ostatak lozinke koja nije dekomponirao na daljnju obradu pomoću druge dvije metode.

Prva metoda je korištenje l0-raščlanjivanja u cilju identifikacije kombinacije. U slučaju da globalni klasifikator nije pronašao slabu kombinaciju, ukoliko je lozinka slaba riječ, broj ili niz specijalnih znakova, ili je ona nastala jednostavnom kombinacijom, l0-klasifikator će klasificirati te dijelove na sljedeći način:

Prva faza klasifikatora je l0-raščlanjivanje pomoću kojeg lozinku želimo pretvoriti u polje u kojem su elementi grupirani po tipu znakova. Nakon toga, svaki element nastoji se terminalno klasificirati. Terminalni klasifikator radi na sljedeći način:

```
terminalni klasifikator(e):
      ako je e skup malih znakova (a-z): # l0 == 'a'
            ako za e postoji riječ u bazi riječi za analizu:
                  vrati {'x':e, 'tip':'riječ' , dodatni_podaci_iz_baze}
            inače
                  vrati {'x':e, 'tip':'nepoznato'}

      ako je e skup velikih znakova (A-Z): # l0 == 'A'
            ako za e postoji riječ u bazi za analizu čistog teksta:
                  vrati {'x':e, 'tip':'riječ' , 'mutacija':'uppercase'
                  , dodatni_podaci_iz_baze}
            inače
                  vrati {'x':e, 'tip':'nepoznato'}
      ako je e skup brojeva: # l0 == '0'
            vrati {'x':e, 'tip':'sekvenca'}
      ako je e skup posebnih znakova: # l0 == '#'
            ako je e == ' '
                  vrati {'x':e, 'tip':'razmak'}
            inače
                  vrati {'x':e, 'tip':'specijalni_znakovi'}
      inače:
            vrati {'x':e, 'tip':'nepoznato'}
```

Iz prikazanog vidimo da terminalni klasifikator može prepoznati riječi koje su napisane samo malim, odnosno samo velikim slovima, koristeći za to bazu riječi za analizu koja je opisana u prijašnjem poglavlju te da može klasificirati sekvence i specijalne znakove. Također, iz baze podataka moramo povezati riječ s dodatnim podacima iz kao što su učestalost riječi u korpusu baze za analizu i oznake rječnika u kojima se nalazi ta riječ. Da bi se l0-klasifikaciju mogla usporediti s mutacijskim klasifikatorom, nužno je prvo izabrati mjeru sukladno kojoj će se to činiti. Izabrana mjera vezana je uz učestalost riječi u korpusu za učenje i klasifikacije rangira prema učestalosti identificiranih riječi koje su korisnici izvorno koristili za stvaranje lozinki. Postupak izračuna ocjene vjerojatnosti proveden je eksperimentalno nad skupom lozinki za učenje i kao ulaz uzima gotov dnevnik analize jedne lozinke:

```
izračunaj_ocjenu_vjerojatnosti(log):
      ocjena = 0
      za svaki element u log:
            ako element ima vrijednost 'tip':'nepoznato':
                  vrati 0
            ako element ima vrijednost 'mutacija':
                  ako mutacija nije jedinstvena:
                        vrati 0
            ako duljina riječi u elementu <= 2:
                  zanemari vrijednost učestalosti
            ako je duljina riječi u elementu <= 4:
                  ocjena += učestalost riječi u bazi / 100
            inače:
                  ocjena += učestalost riječi u bazi
      vrati ocjena
```

Ocjena vjerojatnosti zanemaruje sve vrijednosti učestalosti u slučaju riječi koje su kraće ili jednake 2 znaka. Zbog velike učestalosti riječi koje su velike do 4 znaka u gotovo svim jezicima, značaj takvih riječi smanjen je na stoti dio njihove učestalosti. Vrijednosti 2 i 4 odabrane su eksperimentalnim putem da omogućuju čim točniju klasifikaciju.

U ostalim slučajevima zbrajaju se učestalosti svih riječi u dnevničkom zapisu. Bitno je napomenuti da brojčane ili specijalne elemente nije moguće rangirati zbog nedostatka nužnog parametra ocjenjivanja. Navedene vrijednosti smanjivanja značaja utvrđene su eksperimentalno nad skupom lozinki za učenje. Ukoliko se značaj kraćih riječi ne bi smanjio, postojala bi puno veća šansa da će mutacijski klasifikator najvjerojatniju kombinaciju odabrati na pogrešan način.

Ukoliko se u dnevniku pojavi opisana mutacija, treba provjeriti je li ona nejedinstvena, što znači da za mutaciju koja jedan ulazni znak pretvara u neki drugi znak, postoji slučaj da sa ulazne strane ili izlazne strane postoji više istih znakova. To znači da je osoba koristila kontradiktornu mutaciju, koja bi značila da je lozinka kreirana nasumično, što onemogućava klasifikaciju, ili je, što je puno vjerojatnije, došlo do pogrešne klasifikacije, koja se opisuje kao netočna.

Treći glavni klasifikator je mutacijski klasifikator koji radi na sljedećem principu:

```
mutacijski_klasifikator(l,br_kandidata=4):

      e = []
      za svaki x od 1 do br_kandidata:
            e  += mutacijsko_raščlanjivanje(l, x)

      linije = []
      za svaki kandidat u e:
            linije += pred_terminalna_klasifikacija(kandidat)

      ocj = 0
      za svaki x u linije:
      izr = izračunaj_ocjenu_vjerojatnosti(x)
      ako je izr >= ocj:
            l = x
            ocj = izr

      vrati l
```

Glavni dio mutacijskog klasifikatora je predterminalna klasifikacija, koja može identificirati pretvaranje malih slova u velika, kao i način na koji je neka riječ mutirana. Taj klasifikator se

naziva predterminalnim jer terminalni klasifikator nastaje njegovom nadogradnjom. Kako bi se mutacija mogla identificirati, nužno je omogućiti raščlambu lozinke na način da se može utvrditi koje znakove je korisnik zamijenio dok je svoju lozinku birao. Taj postupak provodi mutacijsko raščlanjivanje, koje radi na sljedećem principu:

```
mutacijsko_raščlanjivanje(l,prag):
      l0 = l0_raščlanjivanje(l)
      raščlamba = []
      niz = ‘’
      za svaki element u l0:
            ako je duljina element > prag i ako element nije tekstualni:
                  ako ima elemenata u nizu:
                        raščlamba += niz
                  raščlamba += element
                  niz = ‘’
            ako je element tekstualan:
                  niz += element
            ako je duljina element <= prag i ako element nije tekstualni:
                  niz += element
      ako je duljina niza > 0
            raščlamba += niz

      vrati raščlamba
```

Mutacijsko raščlanjivanje identično je l0-raščlanjivanju, osim što spaja specijalne znakove i brojeve s tekstom, ukoliko je broj tih elemenata jednak ili manji od vrijednosti praga. Ukoliko je prag 0, mutacijsko raščlanjivanje jistovjetno je l0-raščlanjivanju. Tako lozinka S3curity!!1337 predstavlja primjer mutacijskog raščlanjivanja, s pragovima 0 do 4:

- mutacijsko_raščlanjivanje('S3curity!!1337',0) -> ['S', '3', 'curity', '!!', '1337']
- mutacijsko_raščlanjivanje ('S3curity!!1337',1) -> ['S3curity', '!!', '1337']
- mutacijsko_raščlanjivanje ('S3curity!!1337',2) -> ['S3curity!!', '1337']
- mutacijsko_raščlanjivanje ('S3curity!!1337',3) -> ['S3curity!!', '1337']
- mutacijsko_raščlanjivanje ('S3curity!!1337',4) ->['S3curity!!1337']

Iz gornjeg primjera vidi se da je mutacijsko raščlanjivanje s pragom 1 zapravo ono koje je točno, jer prepoznaje da je prvo slovo riječi veliko slovo, prepoznaje mutaciju e u 3 i dodatke koje je korisnik dodao. Također, teško je prepoznati mjesta u riječi na kojima će korisnik izvršiti mutacije i koliko znakova se smije spojiti, a da se ne spoje i znakovi koji ne bi trebali i ne onemogući točna dekompozicija. Iz tog razloga, mutacijski klasifikator stvara 4 moguća prijedloga raščlambe vezana uz prag do treće razine, jer prema listi za učenje, rijetko koji korisnik radi mutaciju koja uključuje četiri zamjene koje se nalaze zajedno jedna do druge. Svaki od prijedloga za mutacijsko raščlanjivanje mora biti klasificiran pomoću

predterminalnog klasifikatora, koji preuzima polje koje je rezultat mutacijskog raščlanjivanja i provodi sljedeći postupak:

```
pred_terminalni_klasifikator(s):
      log = []
      za svaki element u s:
            ako je element tekst samo brojevi ili specijalni znakovi:
                  log = terminalna_klasifikacija(element)
            ako element ima velikih slova u s:
                  m = pretvori_u_mala_slova(element)
                  up = na_kojim_mjestima_se_nalaze_velika_slova(element)
                  log += {'x':m , 'uppercase':up}
                  element = m
            mut = otkrij_mutaciju(element)
            ako je mut != 0:
                  log += mut
                  element = mut[x] # mut[x] je čisti tekst nakon mutacije
            log += terminalna_klasifikacija(element)
      vrati log
```

Iz prikazanog pseudokoda vidimo da:

1. Ukoliko element u raščlanjenom elementu ima samo brojke ili specijalne znakove, takav element terminalno klasificiramo.
2. Ukoliko element ima velika slova, dodajemo zapis u dnevnik koji govori koja slova treba povećati na tom elementu, nakon čega se sadržaj elementa, pretvoren u mala slova, prosljeđuje na daljnju analizu.
3. Pokušava se otkriti mutacija, što je zapravo postupak uklanjanja zamjena kako bi se ipak mogla izvršiti terminalna klasifikacija.
4. Mutaciju možemo otkriti na način da koristimo sustav za upravljanje bazom podataka (SUBP) s rječnikom za analizu, gdje SUBP omogućava upite sličnosti.
5. Upiti sličnosti će vratiti najvjerojatnije riječi, jer su riječi rangirane po učestalosti korištenja. Upit sličnosti provodi se na način da se brojke i specijalne znakove u elementu zamijene s oznakom koja označava jedan zamjenski znak i upit rangira po učestalosti korištenja riječi.

Nakon toga, originalni element moguće je usporediti s dobivenim rezultatom upita i vidjeti koji su znakovi zamijenjeni. Pseudokod funkcije otkrij_mutaciju je sljedeći:

```
otkrij_mutaciju(s):
      k = zamjeni_sve_znakove_koji_nisu_slova_sa(s,‘_’)
      kandidat = upit_sličnosti_u_SUBP(k)
      ako je kandidat pronađen:
            mut = {}
            za svaki element gdje se kandidat i s razlikuju
            mut[element u s]  = element u kandidat
            vrati {‘x’:kandidat, ‘zamjeni’:mut, ‘tip’:’mutacija’}
      inače
            vrati {‘x’:s, ‘tip’:’nepoznato’}
```

Kao primjer otkrivanja mutacije, možemo prikazati izlaz iz funkcije otkrij_mutaciju koja je implementirana unutar skripti za analizu:

```
[2]: otkrij_mutaciju ('s3curity')
     {'zamjeni': {'e': '3'}, 'tip': 'mutacija', 'x': 'security'}

[3]: otkrij_mutaciju ('70n1mir')
     {'zamjeni': {'i': '1', 'o': '0', 't': '7'}, 'tip': 'mutacija', 'x':
'tonimir'}

[4]: otkrij_mutaciju ('f4kult3t')
     {'zamjeni': {'a': '4', 'e': '3'}, 'tip': 'mutacija', 'x': 'fakultet'}
```

Nakon otkrivanja mutacije, rezultat elementa moguće je klasificirati terminalnim klasifikatorom, jer je rezultat otkrivanja mutacije čista riječ, kao što možemo vidjeti u prijašnjem prikazu izlaza funkcije otkrij_mutaciju.

Taj postupak provodi se za tri predložene razine mutacijske raščlambe, nakon čega se bira ona koja ima najveću ocjenu vjerojatnosti sukladno funkciji ocjene vjerojatnosti. Nakon toga, moguć je odabir između l0-raščlambe ili mutacijske raščlambe, koji se opet uspoređuju pomoću funkcije ocjene vjerojatnosti. Raščlamba koja ima najveću ocjenu vjerojatnosti jest ona koja se smatra najtočnijom.

Kao primjer, možemo prikazati nekolicinu klasifikacija koje su provedene pomoću metode sita, implementirane u programskom jeziku *python*, što bilo potrebno za realizaciju eksperimenata. Bitno je napomenuti da u ovom slučaju implementacija koristi englesko nazivlje zbog kasnijeg objavljivanja alata i mogućnosti da alat koriste osobe koje nisu iz hrvatskog govornog područja.

```
In [14]: sieve('foifoifoifoi')
Out[14]: [{'el': 'foifoifoifoi', 'len': 3, 'n': 4, 'type': 'weak_pattern'}]


In [7]: sieve('sigurnost17271!')
Out[7]:
[{'el': 'sigurnost',
  'langs': ['en', 'de', 'es', 'fr', 'hr', 'it', 'nl'],
  'len': 9,
  'occ': 1547,
  'type': 'word'},
 {'el': '17271', 'len': 5, 'type': 'sequence'},
 {'el': '!', 'len': 1, 'type': 'special'}]


In [11]: sieve('1984c0mpl1cated!(&%')
Out[11]:
[{'el': '1984', 'len': 4, 'type': 'sequence'},
 {'REPLACE': {'i': '1', 'o': '0'}, 'type': 'mutation', 'x': 'complicated'},
 {'el': 'complicated',
  'langs': ['en', 'de', 'es', 'fr', 'hr', 'it', 'nl'],
  'len': 11,
  'occ': 43561,
  'type': 'word'},
 {'el': '!(&%', 'len': 4, 'type': 'special'}]


In [13]: sieve('lo(ikju&^%$')
Out[13]: [{'el': 'lo(ikju&^%$', 'len': 11, 'type': 'keyboard_pattern'}]
```

U ovom primjeru vide se elementi koji moraju biti pohranjeni u bazu podataka na način da budu pogodni za daljnju obradu i analizu.

7.6. Prikupljanje podataka i kreiranje modela ponašanja

Prikupljene dnevničke datoteke spremaju se u sustav za upravljanje bazom podataka *PostgreSQL*, iz razloga jer je isti sustav korišten za čuvanje rječnika za analizu i provedbu upita sličnosti. Kao oblik pohrane, iz dnevničkoga zapisa pohranjeno je, uz praćenje učestalosti, pojavljivanja sljedećih elemenata:

1. Sve riječi, brojčane kombinacije i kombinacije posebnih znakova te razmaka.
2. Sve riječi koje nije bilo moguće identificirati pomoću upita u bazu podataka.
3. Svi slabi uzorci.
4. Svi nizovi koji odgovaraju sljedovima znakova iz tipkovnice.
5. Oblik simetričnih elemenata i raspored njihovog zrcaljenja.
6. Oblik umetaka.
7. Oblik i način mutacije:

a. Prijevod za ključ u vrijednost praćen pojedinačno po mutaciji.
b. Prijevod za ključ u vrijednost praćen grupno za cijelu mutaciju koja se primjenjuje nad jednom riječi.

8. Pretvaranje malih znakova u velike u ovisnosti o duljini riječi i lokaciji u riječi.
9. Cjelokupni dnevnički zapis sa svim informacijama.
10. Sadržaj dnevničkog zapisa sa uklonjenim vrijednostima čistog teksta (vrijednost x).
11. Sadržaj dnevničkog zapisa sa uklonjenim vrijednostima čistog teksta (vrijednost x) i uklonjenim duljinama.

Točke od 1 do 8 su jasne zbog prikupljanja podataka koje možemo analizirati u cilju boljeg uvida o korištenju pojedinih elemenata u lozinkama i za korištenje u daljnjim istraživanjima o navikama korisnika prilikom odabira lozinke.

Riječi, brojčane kombinacije i kombinacije posebnih znakova korist će se za izradu rječnika koji se spaja s rječnikom koji je generiran u prethodnom poglavlju rada pomoću metode *gwordlist*. Spajanje rječnika mora biti izvedeno po učestalosti pojavljivanja riječi, što bi trebalo generirati skup riječi koji je puno bolje raspršen nego da su riječi birane proizvoljno. Takva lista kreirana je iz realnih korisničkih odabira, što bi svaku analizu trebalo ubrzati jer sadrži elemente koje su koristili stvarni korisnici. Svaka od listi za riječi, brojeve ili znakove podijeljena je na više lista, od kojih svaka sadržava samo riječi, brojeve ili znakove, što će se poslije koristiti u dva glavna modela koji će biti opisati kasnije.

Prikupljanje mutacija po grupi zamjena (prema 7.b i 7.a) i po pojedinom obliku mutacije potrebno je iz razloga da bi se pogrešno klasificirane lozinke identificirale i takve klasifikacije uklonile. Na taj način, promatrajući grupe mutacija, moguće je isključiti mutacije koje imaju nisku učestalost pojavljivanja, odnosno pogrešne mutacije, i prema tome isključiti modele koji ih koriste, kako bi se iz sustava uklonio *šum*.

Prednost spremanja dnevnika u tri razine zapisa je ta što je dvije zadnje razine moguće koristiti kao jedan oblik generalizacije. Ako se iz svih dnevnika izbaci čisti tekst, tj. polje ‘x’, u njemu će ostati samo duljina i tip elementa. Ponašanje korisnika na taj način suženo je na duljinu riječi i tip elementa, tj. na to je li korisnik koristio riječ iz jezika, iz skupa brojeva, ili riječ sročenu od specijalnih znakova. Drugo suženje iz razmatranja izuzima duljine riječi, tako

da se pohranjuje samo tip elementa. Taj pristup omogućuje jednostavno kreiranje dvaju glavnih tipova modela koji opisuju ponašanje korisnika:

1. Suženi model – za koji je bitna samo informacija o tome je li korisnik koristio riječ, broj ili specijalni znak, te način na koji je lozinka sastavljena. Mutacije, odnosno pretvaranje znakova u velika ili mala slova, vezane su uz element riječi, broja ili specijalnog znaka. Nakon prevođenja sadržaja baze podataka, isključujući mutacije koje su imale nisku razinu pojavljivanja, razvijeno je 10.587 suženih pravila kojima se opisuje kako su korisnici kreirali lozinke. Suženi model koristi po jedan tip datoteke rječnika za riječi, brojeve ili znakove, gdje su elementi u tim datotekama organizirani po učestalosti korištenja.
2. Specijalizirani model – istovjetan je suženom modelu, ali razlikuje i duljine riječi, brojeva i specijalnih znakova. Nakon prevođenja sadržaja baze podataka, isključujući mutacije koje su imale nisku razinu pojavljivanja, razvijeno je 94.131 specijaliziranih pravila kojima se opisuju kako su korisnici kreirali lozinke. Specijalizirani model koristi više skupova datoteka za riječi, brojeve ili znakove, gdje svaka datoteka sadrži elemente određene duljine organizirane po učestalosti korištenja.

Oba modela zapravo predstavljaju skup prije predloženih MK-Pravila, koja zahtijevaju eksperimentalnu verifikaciju pojedinačnih implementacija, kako bi se utvrdilo koja od njih će dati najbolje rezultate.

Jedan od uzoraka koji je pronađen vezan je uz to da određeni korisnici ne primjenjuju mutacije nad svim znakovima u čistom tekstu. Kao primjer, možemo prikazati sljedeće upite u bazu podataka, gdje su posebno interesantne lozinke prikazane debelo otisnutim slovima:

```
unhash=# SELECT  * from unanalysed_plaintexts where plaintext like 'pas5%';
 plaintext  | occ
------------+-----
 pas5word   |  15
 pas5w0rd   |   3
 pas5w0rd08 |   1
 pas578413  |   1
 pas55word  |   1
 pas5150    |   1
 pas510ns   |   1
 pas509     |   1
(8 rows)
```

```
unhash=# SELECT  * from unanalysed_plaintexts where plaintext like 'pa5s%';
 plaintext | occ
-----------+-----
 pa5sword  |   1
 pa5sw0rd  |   1
 pa5ssion  |   1
 pa5sord   |   1
(4 rows)
```

Iz prikazanog rezultata vidi se da je u prvom upitu postojalo čak 15 korisnika koji su izveli zamjenu samo drugog slova *s* u vrijednost 5, dok je samo jedan korisnik to napravio s prvim slovom. Ova pojava je rijetka i može se vidjeti u bazi analiziranih lozinki, ali pridaje važnost promatranju mutacija kroz koncept permutiranja svih mogućih kombinacija za jednu mutaciju, ukoliko cijena značajno veće analize nije problematična.

Da bi se olakšalo korištenje specijaliziranih i suženih modela, razvijen je alat naziva "unhash" koji koristi pravila iz modela tipa M3, uz sljedeći način označavanja i opisa pravila:

1. Pravila su grupirana i sortirana po učestalosti pojave.
2. Specijalizirani i generalizirani model sadrže skup polja.
3. Jedno polje predstavlja jedno pravilo u sustavu.
4. Svako polje sadrži određeni broj kataloga ili niz znakova.
5. Svaki element polja tj. pravila predstavlja jednu operaciju:
    a. Niz znakova predstavlja konstantu koja se ne mijenja.
    b. Katalog predstavlja čitanje podataka iz datoteke, nad kojim je moguće pretvarati slova iz malih u velike i provoditi mutacije.
6. Ako postoji 2 ili više elementa u polju, rezultat kombinacije je Kartezijev produkt tih elemenata (kombinacija, tj. spoj svih vrijednosti tih elemenata). Kombinacija poštuje raspored elemenata u polju. Ovim načinom se ostvaruje kombinacija rječnika ili pojmova.
7. Svaki katalog može koristiti sljedeće oznake:
    a. Oznakom 'f' u imenu datoteke označava se datoteka u kojoj se nalaze riječi koje treba iterirati.
    b. Oznakom 'u' označava se indeksirano polje slova koja treba pretvoriti u velika, što se najčešće koristi u obliku 'u':[0] za samo prvo slovo, ili 'u':'ALL' za pretvaranje svih slova u velika.

c. Oznakom ‘m’ označava se mutaciju za koju katalog koji nakon nje slijedi prikazuje preslikavanje između odgovarajućih elemenata. Primjerice, ‘m’:{‘a’:’4’ , ‘i’:’1’} prikazuje zamjenu slova *a* sa 4 i slova *i* sa 1.
d. Postavljanjem zastavice *rperm* u 1 označava se da svaku mutaciju treba iterirati kroz sve moguće kombinacije zamjena.
e. Oznaka ‘g’ označava generiranje znakova u obliku iscrpnog pretraživanja, kojoj se dodaje niz znakova pomoću kojih će generator generirati sve moguće kombinacije riječi duljine definirane oznakom ‘l’
f. Oznaka ‘gl’ označava generiranje znakova u obliku iscrpnog pretraživanja, za koje generator generira znakove od 1 do broja koji je definiran oznakom ‘l’

Takva struktura rada s pravilima nastoji biti bliska domenski specifičnom jeziku (DSL), uz omogućavanje korištenja svih konstrukta jezika *Python* u kojem je alat “unhash” implementiran. Namjera je bila omogućiti korištenje svih konstrukata u programskom jeziku *Python,* i u isto vrijeme omogućiti jednostavno opisivanje pravila za analizu lozinke. Jedna od velikih prednosti toga je da su opisi modela zapravo skripte u jeziku Python, i da je, umjesto potpune putanje datoteke, moguće koristiti varijable, i na taj način jednostavno apstrahirati model i njegova pravila, tako da je moguće koristiti bilo koji skup datoteka i jednostavno eksperimentirati ili provoditi svakojake analize.

Kao primjer strukture pravila prikazano je njih 10 u suženom modelu. Ta su pravila sadržana u datoteci s pravilima:

```
[{ 'f':word }],
[{ 'f':sequence }],
[{ 'f':word },{ 'f':sequence }],
[{ 'f':word, 'u':'ALL' }],
[{ 'f':word, 'u':[0] }],
[{ 'f':word },{ 'f':string }],
[{ 'f':word, 'u':'ALL' },{ 'f':sequence }],
[{ 'f':sequence },{ 'f':word }],
[{ 'f':word, 'u':[0] },{ 'f':sequence }],
[{ 'f':word },{ 'f':sequence },{ 'f':word }],
```

U ovim pravilima, u kojima ključ ‘f’ stoji za čitanje datoteke, riječi *word*, *sequence* i *string* predstavljaju samorazumljive varijable. Putanje do odgovarajućih datoteka određuju se naknadno, ovisno o tome koji skup datoteka se za analizu želi koristiti. Iz prikazanog se vidi

da je omiljena kombinacija koju korisnici opetovano koriste dodavanje broja na neku riječ, a omiljena mutacija pretvaranje prvog slova u veliko slovo.

Drugi primjer sadrži prikazi prvih 20 pravila u specijaliziranom modelu. I ta su pravila sadržana u datoteci s pravilima:

```
[{ 'f':dictlen6 }],
[{ 'f':numlen6 }],
[{ 'f':dictlen7 }],
[{ 'f':dictlen8 }],
[{ 'f':dictlen5 }],
[{ 'f':dictlen6 },{ 'f':numlen2 }],
[{ 'f':dictlen5 },{ 'f':numlen2 }],
[{ 'f':numlen8 }],
[{ 'f':dictlen6 },{ 'f':numlen1 }],
[{ 'f':dictlen9 }],
[{ 'f':dictlen5 },{ 'f':numlen1 }],
[{ 'f':dictlen7 },{ 'f':numlen1 }],
[{ 'f':numlen7 }],
[{ 'f':dictlen7 },{ 'f':numlen2 }],
[{ 'f':dictlen4 },{ 'f':numlen4 }],
[{ 'f':dictlen3 },{ 'f':numlen4 }],
[{ 'f':dictlen5 },{ 'f':numlen3 }],
[{ 'f':dictlen4 },{ 'f':numlen2 }],
[{ 'f':dictlen5 },{ 'f':numlen4 }],
[{ 'f':dictlen8 },{ 'f':numlen1 }],
```

Iz gornjeg isječka vidi se da se opet koriste varijable koje se naknadno mogu uputiti prema određenim putanjama, na čijim krajevima leže rječnici koji sadrže riječi točno određene duljine. U ovom slučaju, *dictlen* su rječnici s riječima točno određene duljine, *numlen* rječnici s brojevima točno određene duljine, a *strlen* rječnici sa znakovima točno određene duljine.

Bitna klasa lozinki su slabi uzorci ili nizovi znakova tipkovnice. Da bi analiza mogla uključiti i takve elemente, oni su također prikupljeni i postavljeni u svoje datoteke, tj. rječnike, koji su organizirani po učestalosti korištenja, i njihova pravila izvršavaju se prije suženog ili specijaliziranog modela. Pretraživanje po modelima tipa M2 moguće je provesti na dva načina; tako da se koristi suženi model, gdje umjesto rječnika iza koji stoji iza oznake “word”, koristi datoteka s ključnim riječima, odnosno određivanjem manjeg broja elemenata iz suženog modela.

Sva pravila za oba modela raspoloživa su u sklopu digitalnog repozitorija podataka vezanog uz ovaj rad.

7.7. Kreiranje alata za korištenje modela tipa M3

Implementacija metode za korištenje specijaliziranih ili generaliziranih modela i njihovih pravila provedena je u programskom jeziku *Python* 2.7.2 [61] uz pomoć interpretera *pypy* [2], koja znatno ubrzava osnovnu implementaciju jezika. Implementacija metode *unhash* koristila je velik broj poznatih optimizacija za brzinu [9] *Python*-skripata, ali, nakon provedenih analiza i eksperimenata, postalo je očigledno da bi s tehničkog aspekta alat bilo bolje implementirati u nekom bržem jeziku, kao što je C [36] ili Go [27], jer zbog spore implementacije nekih konstrukta u pythonu alat ne koristi maksimalni raspoloživi potencijal strojne opreme. Za provjeru koncepta programski jezik *Python* odabran je zbog brzine stvaranja prototipa za verifikaciju i čisto testiranje koncepta. Razliku u brzini izvođenja implementacije zorno prikazuje već i samo promjena interpretera jezika *Python*. Ona je ilustrirana mjerenjima u kojima je korišteno samo prvih 10 pravila u specijaliziranom modelu:

Prvo mjerenje prikazuje implementaciju jezika *Python* koja ne koristi paralelizam:

```
root@hashkill2:~/uh3# time python specialized_model.py >/dev/null

real  0m44.548s
user  0m43.634s
sys   0m0.590s
```

Ukoliko tu implementaciju zamijenimo implementacijom *pypy* jezika *Python*, dobivamo sljedeći rezultat:

```
root@hashkill2:~/uh3# time pypy specialized_model_para.py >/dev/null

real  0m37.654s
user  0m34.813s
sys   0m1.165s
```

Za drugo mjerenje korištena je paralelizacija od 4 procesa, na način da svaki od 4 procesa stvara moguće kandidate za testiranje. Cijeli proces, pokrenut pomoću implementacije *Python*, daje sljedeće rezultate:

```
root@hashkill2:~/uh3# time python specialized_model_para.py >/dev/null

real  0m38.532s
user  0m58.076s
sys   0m2.265s
```

Ukoliko se implementacija *Python* zamijeni implementacijom *pypy*, rezultat je sljedeći:

```
root@hashkill2:~/uh3# time pypy specialized_model_para.py >/dev/null

real  0m25.269s
user  0m41.659s
sys   0m2.235s
```

Tijekom trajanja testa, nadzirano je opterećenje računala, i kao usko grlo pokazale su se ulazno-izlazne operacije, koje nisu bile problem zbog brzine samog diska, jer su sva računala imala *SSD* diskove, već se primijetilo opterećenje zbog korištenja skriptnog jezika kojemu nije svojstvena visoka učinkovitost. Slijedom navedenih kratkih testova, kao referentna implementacija *Python*-interpretera odabrana je implementacija *pypy*, s ciljem iskorištenja inherentnog paralelizma sustava i slobodnih procesorskih jezgri.

Implementacija specijaliziranih i suženih modela realizirana je kroz alat "unhash", koji se sastoji od sljedećih elemenata:

1. Implementacije generatora znakova koji može preuzeti pravila iz modela i generirati kandidate za testiranje te ih proslijediti na standardni izlaz računala (*stdout*) tj. ispisivati moguće znakove u terminal.
2. Sadržaj modela, tj. lista pravila, koja se daje generatoru.

Kao izlaz alata korišten je standardni izlaz *stdout* koji u svim operacijskim sustavima predstavlja ispis na tekstualni terminal. Svi popularniji alati za analizu lozinki, kao što su *John the Ripper* ili *Hashcat*, imaju mogućnost preuzimanja ulaza iz standardnog izlaza i korištenja tog ulaza umjesto svojeg generatora kandidata lozinki za testiranje, pri čemu se ulazi koje preuzmu s izlaza *stdout* zaštite jednosmjernom funkcijom i provjeri se postoji li zaprimljena lozinka u skupu lozinki za analizu. Programeri alata *John* i *Hashcat* razvili su veliki skup optimizacija za algoritme zaštite te su njihovi alati u tom pogledu zamjetno brži od standardnih implementacija algoritama za zaštitu u klasičnim programskim jezicima. Zbog toga nije isplativo kreirati još jedan alat u moru danas već postojećih alata, već, umjesto toga, kreirati alat koji može upotpuniti današnje standardne alate koje konzultanti i administratori sustava znaju koristiti.

Odvajanje generatora, pravila i modela provedeno je kako bi se omogućilo da se pravila i modeli konstantno unapređuju ili prilagođavaju, ovisno o potrebama osobe koja lozinke testira. Jedna od prednosti uključivanja moguće verzije modela jest mogućnost prilagodbe skupova pravila potrebama normi koje zahtijevaju određenu razinu složenosti, što korisnike dodatno "motivira" da kreiraju što jednostavniju lozinku koja odgovara postavljenim uvjetima. Na taj način, moguće je prilagoditi modele za testiranje samo onih lozinki koje zadovoljavaju zahtjevima složenosti, isključujući lozinke koje u sustav nije moguće niti prijaviti. Takav način rada skraćuje potrebno vrijeme analize, jer ne troši procesorsko vrijeme na kombinacije za koje se zna da su nemoguće.

Drugi razlog takvog odvajanja jest paralelizacija procesiranja. Alat *john* paralelizaciju trenutno podržava samo korištenjem posebnih zakrpa za sustav MPI [44]. Analiza lozinki predstavlja problem koji se jednostavno paralelizira, na način da se prostor pretraživanja raspodijeli na više računala.

8. Primjena modela, analiza performansi i rezultati istraživanja

Eksperimentalna validacija i mjerenje performansi metode provedena je na istovjetnim virtualnim poslužiteljima koji se mogu kod pružatelja usluge za uslužno računalstvo (eng. Cloud Computing) DigitalOcean [16] iznajmiti po satu. Unajmljeni poslužitelji imali su sljedeća obilježja:

- 4 CPU-jezgre sa 64 bitnim instrukcijskim skupom
- 8 GB RAM-a
- 80 GB SSD-diska
- Operacijski sustav: *Ubuntu Gnu/Linux* 13.04 x86_64
- Jezgra *Gnu/Linux* operacijskog sustava 3.8.0-19-generic *#30-Ubuntu SMP*
- Lokacija poslužitelja / regija: New York 2
- Cijena po satu: 0,119 USD po poslužitelju

Za provjeru performansi korišten je alat *John the ripper* [14], koji stručnjaci za sigurnost popularno zovu *jtr* ili *john*. Za rad analize korišten je alat *tmux* [47] koji omogućava multipleksiranje terminala i pokretanje poslova koji se odvijaju na nekom terminalu. Na početku i kraju svake analize, pokrenuta je procedura testiranja performansi alata *John*, tj. testiranja koliko puta John može lozinku zaštititi i provjeriti tu lozinku protiv nekog kandidata, što zapravo prikazuje i performanse tog računala. Provjera je pokrenuta na početku i kraju testiranja kako bi se provjerilo je li došlo do promjene brzine virtualnog stroja. Promjena brzine nije zabilježena, a zbog zakupljenih i zajamčenih performansi virtualnog stroja nije ni trebala biti. Bazne performanse virtualnog stroja prikazane su u prilogu ”Brzina provjere lozinki virtualnog stroja za analizu”. Zadnja dostupna verzija alata *john* u vrijeme pisanja ovog rada iznosila je 1.7.9, uz razinu dodataka jumbo-7. Jumbo-dodaci predstavljaju dodatne zakrpe i poboljšanja koja su alatu *john* u cilju poboljšavanja performansi i dodavanja podrške za razne algoritme zaštite. Dodatni razlog za korištenje alata *john* je taj što je on jedan od najstarijih i najpopularnijih alata za provjeru snage lozinki, otvorenog je koda, što znači da je raspoloživ na skoro svim platformama, poput različitih verzija operacijskih sustava *Windows*, *Linux*, *BSD*, *OSX*, kao i mnogih drugih, a pored toga na njemu je izuzetno lagano izvršiti željene preinake i ukomponirati ih u alat za svoje potrebe.

Za potrebe testiranja *john* je kompajliran sa kompajlerom *gcc*, sa linux-x86-64-native opcijom za 64 bitne operacijske sustave. Test, raspoloživi algoritmi i brzina analize prikazani su u prilogu "Brzina provjere lozinki virtualnog stroja za analizu"

8.1. Skupovi podataka, način provjere i ograničenje metode *unhash*

Za provjeru performansi predložene metode *unhash* koristila su se dva skupa lozinki. Podrijetlo lista lozinki objašnjeno je u prijašnjem poglavlju, "Nacrt istraživanja i izvori podataka". Stvorena su dva skupa lozinki koja su zaštićena kriptografskom jednosmjernom funkcijom *SHA256* [22] koja je trenutno jedna od funkcija toga tipa koja se preporučuje za zaštitu lozinki. Funkcija *SHA256* također je vrlo popularna funkcija za zaštitu lozinki u raznim alatima i sustavima i često se koristi sa ili bez kriptografskog dodatka. Odabir neke druge funkcije utjecao bi samo na brzinu analize i broj otkrivenih lozinki. Ako bi se, primjerice, koristile funkcije *bcrypt* [58], ili Blowfish [59], iz sastava operacijskog sustava OpenBSD, koja su zamjetno sporije, otkrivenih lozinki u jednakom vremenu bilo bi značajno manje, jer su te funkcije sporije oko 500 do 10.000 puta, ovisno o tipu implementacije i broju iteracija. Odabir kriptografske funkcije za uspoređivanje performansi nad općenitijim modelima nije toliko bitan, jer se alati uspoređuju s istim kriptiranim skupom lozinki i promatraju se općenite performanse nad širim općenitim skupom lozinki, a ne specijaliziranim podskup jednog slučaja.

Kao duljina trajanja analize odabran je segment od 24 sata. Ukoliko se promatra rad konzultanta ili osobe koja u što kraćem vremenu želi otkriti što veći broj slabih lozinki, rok od 24 sata čini u najboljem slučaju najmanje 2 radna dana, odnosno 3 radna dana, ako se u to računa i vrijeme za blokiranje ili iskorištavanje slabih lozinki, komunikaciju s klijentima ili ako je konzultant lozinke počeo analizirati krajem radnog dana. Naravno, postoje i slučajevi kada je konzultantu ili osobi koja lozinke analizira u interesu iskušati dulje vrijeme analize, što ne predstavlja dodatni problem, jer su iz skupa lozinki za učenje generirana sva moguća pravila za taj skup, ukupno 94.131 specijaliziranih pravila koja opisuju načine na koje korisnici generiraju svoje lozinke, što, uz rječnik prosječne veličine, već rezultira golemim vremenom analize, koje unaprijed nije moguće utvrditi točno, već samo približno, poznavajući listu riječi i skup pravila. Također, ukoliko konzultant ili osoba za potrebe

analize lozinki koristi svoje osobno računalo, manje je vjerojatno da će analizu provoditi dulje od 6-8 sati, budući da isto računalo koristi za svakodnevne potrebe. Vremena dulja od 8 sati zahtijevaju posebne poslužitelje, namijenjene samo analizi, jer ona koristi svu raspoloživu procesorsku snagu računala. U situaciji neraspolaganja poslužiteljem, posvećenom samo analizi lozinki, danas je popularno koristiti se resursima tzv. *uslužnog* (eng. cloud) računalstva. Zbog naplaćivanja poslužitelja po satu, bitno je ocijeniti koliki je budžet dostupan za jednu analizu. Zbog navedenih razloga, autor je odlučio da je period od 24 sata prosjek truda koji se većinu vremena koristi za analizu slabih lozinki.

Skup lista za verifikaciju podijeljen je na dva glavna skupa. Prvi skup čini grupa lozinki iz liste *Linkedin* koja predstavlja općeniti profil korisnika, dok drugu listu čini zbirna lista lozinki iz lista *carders.cc*, *elitehackers*, *hak5*, *hotmail* i *myspace*, koja bi u slučaju lista *carders.cc*, *elitehackers* i *hak5* trebala sadržavati ponajviše lozinke visoke složenosti, a u slučaju lista *hotmail* i *myspace* i one prosječne složenosti. Ove liste su odabrane zbog popularnosti i dostupnosti. Nakon liste *rockyou* najpopularnija lista s poznatim podrijetlom je lista *Linkedin*, a isto vrijedi i za ostale odabrane. Na raznim *web*-stranicama, kao što je *pastebin* [68], ili na raznim portalima hakerskih skupina, koji se mogu pronaći na mreži *tor* [17], koja osigurava vrlo veliku razinu internetske anonimnosti, moguće je pronaći veliki broj lista lozinki koji su rezultat napada. Za većinu tih lista nije jasno njihovo podrijetlo, kao ni struktura, te se zbog toga analiza zadržala nad dobro poznatim listama, koje su i inače referenca za uspoređivanje metoda za analizu lozinki za potrebe znanstvenih istraživanja u tom području.

8.2. Postavke eksperimenta i primjena modela pomoću metode i alata *unhash*

U cilju usporedbe performansi različitih metoda, provedeni su sljedeći eksperimenti. Prvi skup eksperimenata uključivao je listu lozinki *linkedin*, nad kojom je unutar 24 sata mjeren broj otkrivenih lozinki i njihova duljina. Za potrebe referentnog mjerenja postojećih metoda korišten je alat *john* verzije 1.7.9 s razinom dodataka *jumbo-7*. Rezultati tog mjerenja u daljnjem tekstu označeni su oznakom *JTR*. Provjeru performansi metode *unhash* trebalo je provesti provjerom dvije glavne moguće implementacije:

1. Primjenu suženih modela. Navedeni test u daljnjem tekstu označen je oznakom *GEN*.
2. Primjenu specijaliziranih modela. Navedeni test u daljnjem tekstu označen je s *SP*.

Nakon eksperimenta, koji je pokazao da su specijalizirani modeli znatno brži od suženih, ako se gleda opća primjena (sami rezultati analize bit će prikazani kasnije u ovom poglavlju), valjalo je provjeriti i nadogradnju boljeg modela s konceptima modela M2 i iscrpnog pretraživanja, za raspon znakova kratkih riječi do 6 slova i brojeva do 8 znakova, koji po pitanju vremena analize nije zahtjevan, što se vidi iz sljedećeg testa:

```
root@hashkill2:~/uh3# time pypy -c 'from unhash import *; import string;
p({"gl":string.digits,"l":8})' > /dev/null

real    1m25.989s
user    1m25.768s
sys     0m0.216s
root@hashkill2:~/uh3# time pypy -c 'from unhash import *; import string;
p({"gl":string.lowercase,"l":6})' > /dev/null

real    4m7.540s
user    4m7.012s
sys     0m0.520s
```

Razvidno je da je za mala i velika slova te brojeve potrebno 11tak minuta, što je za analizu jedne jako interesantne klase slabih lozinki zanemarivo vrijeme.

Uz prva dva mjerenja, navedena razmatranja dodaju potrebu za još dva mjerenja:

1. Primjenu specijaliziranih modela s listom sukladno modelu tipa *M2*. Navedeni test u daljnjem tekstu je oznakom *SPM2*. Lista, građena sukladno modelu tipa *M2*, koristila je samo najopćenitija pravila suženog modela, a to su prefiksiranje i sufiksiranje s brojevima i znakovima.
2. Primjenu specijaliziranih modela s listom u skladu s modelom tipa *M2* i iscrpnim pretraživanjem za:
    - Mala slova od 1 do 6 znakova (abcdefghijklmnopqrstuvwxyz)
    - Velika slova od 1 do 6 znakova (ABCDEFGHIJKLMNOPQRSTUVWXYZ)
    - Brojevi od 1 do 8 znamenki (0123456789)
    - Navedeni test u daljnjem tekstu označen je s *SPM2BF*.

Za listu, građenu sukladno modelima tipa *M2*, koja se koristi u analizi, odabrani su sljedeći pojmovi na koje će biti prefiksiran ili sufiksirani najčešći brojevi i simboli prema generiranim rječnicima (*unhash_dict_v03*):

- linkedin
- business

- networking
- recruit
- recruiting
- job
- arbeit
- contacts

Pojmovi za model tipa M2 odabrani su proizvoljno, što predstavlja način koji bi koristili konzultanti za sigurnost. Prednost modela tipa M2 je da želimo iskoristiti svoja saznanja o mogućim ključnim riječima koje su dio načina kako je korisnik kreirao svoju lozinku.

Nakon analize prvog skupa lozinki, kao najbržom pokazala se metoda *SPM2BF*, koju je protiv referentne implementacije na skupu lozinki koja sadrži liste *carders.cc*, *elitehackers*, *hak5*, *hotmail* i *myspace* dodatno provjerena u drugom eksperimentu. U ovom mjerenju, referentna implementacija imala je oznaku *exJTR*, a metoda *SPM2BF* oznaku *exSPM2BF*.
Za drugi skup lozinki, odabrane su sljedeće lozinke u skladu s modelima tipa *M2*:

- carderscc
- carder
- carding
- skim
- skimmer
- elitehackers
- profit
- hax
- hacking
- hak5
- hacker
- own
- elite
- 1337
- hotmail
- message
- mail
- myspace
- friend

Prilikom dodavanja modela tipa *M2*, odnosno iscrpnog pretraživanja, u testu nisu optimirana pravila da bi se uklonila možebitna preklapanja u analizi, kao što je ispitivanje popularnih riječi od 5 znakova, već su ta pravila zadržana. Dodatni razlog tome je da je cilj ovog istraživanja razviti metodu i alate koje mogu koristiti konzultanti za sigurnost, osobe koje provjeravaju sigurnost sustava te administratori sustava, koji pravila vjerojatno neće posebno

prilagođavati. Određeni broj korisnika, kao što su konzultanti ili profesionalci, pravila i modele vjerojatno će željeti modificirati za svoje potrebe, ali prosječan administrator sustava će koristiti sustav kako mu je u početku podešen. Drugi razlog bio je taj da su modeli koji su u preklapanju dosta mali, tako da je gubitak u performansama na razini petnaestak minuta rada sustava.

Alat *john the ripper* prilikom analize stvara dvije datoteke. Prva datoteka je "john.pot" u kojoj se spremaju lozinke koje su pronađene, gdje datoteka ekstenzije .pot predstavlja jednu vrstu postupka *TMTO* [30], koji se može koristiti za kasnije analize ili tome da se korisniku omogući lakši prikaz otkrivenih slabih lozinki u odnosu na neku datoteku s lozinkama.

Druga datoteka je "john.log", koja je dnevnička datoteka u kojoj se sprema vrijeme s oznakom dana, sata, minute i sekunde u kojem je lozinka otkrivena, te korisničko ime u trenutku kada se to dogodilo. Kombiniranjem te dvije datoteke može se dobiti niz podataka s vremenom otkrivanja lozike i samom otkrivenom lozinkom.

Za potrebe analize tih datoteka korištena je za analizu podataka vrlo korisna biblioteka *pandas* jezika *python* (*python data analysis library*) [49].

Skripte za analizu podataka i rezultati analize, svrstane u zasebne datoteka ekstenzija *.log* i *.pot*, u smislu vremena izvođenja ograničene su na jedan, a arhiv, s cijelim sadržajem analize, mapom vezanom za alat *john* i svim binarnim datotekama te izvornim kodom, dostupne su u digitalnom repozitoriju ovog rada.

8.3. Rezultati provjere prvog skupa podataka:

U sljedećoj tablici vide se rezultati mjerenja za prvi eksperiment koji je izveden na listi *linkedin*. Zbog lakše identifikacije mjerenja koje je pronašlo najveći broj lozinki, taj je broj za svaki odsječak jednog sata prikazan podebljanim pismom:

Tablica 9: Prikaz vremena analize i broja otkrivenih lozinki za listu *linkedin* po metodama

Vrijeme	JTR	GEN	SP	SPM2	SPM2BF
2013-10-01 00	700274	528594	833662	859071	**951491**
2013-10-01 01	**113039**	50110	6350	7209	61316
2013-10-01 02	64378	42702	107480	149588	**157718**
2013-10-01 03	54398	32768	**103839**	61663	3061
2013-10-01 04	35420	27264	3104	2932	**114045**
2013-10-01 05	**22169**	22211	1087	1212	7817
2013-10-01 06	25696	21287	76072	**81241**	1853
2013-10-01 07	24255	18108	**37271**	35456	821
2013-10-01 08	**25117**	17083	6941	4246	711
2013-10-01 09	**20019**	15388	2071	1906	**133208**
2013-10-01 10	**10368**	13069	1206	1015	2834
2013-10-01 11	**17942**	11281	775	783	760
2013-10-01 12	**22903**	11083	513	584	448
2013-10-01 13	7334	**10734**	417	462	282
2013-10-01 14	**9908**	9634	395	471	216
2013-10-01 15	6516	9230	420	**108249**	203
2013-10-01 16	8633	8244	137078	**24471**	190
2013-10-01 17	9013	7974	11794	1490	**21691**
2013-10-01 18	**14455**	7873	1111	642	1048
2013-10-01 19	**13080**	6779	541	364	434
2013-10-01 20	6063	**7953**	334	306	7627
2013-10-01 21	5568	**7671**	252	234	3443
2013-10-01 22	4887	6865	174	218	**37341**
2013-10-01 23	4068	**6223**	155	123	443
Ukupno	1225503	900128	1333042	1343936	1509001
Postotak	100,0000%	73,4497%	108,7751%	109,6640%	**123,1332%**

U prikazanoj tablici vidljivo je da je suženi model po broju pogođenih lozinki u konačnici bio slabiji od referentne implementacije, dok je specijalizirani model pokazao nešto bolje rezultate. Dodavanje ključnih riječi sukladno modelima tipa *M2* povećalo je broj pogođenih lozinki za 10894 lozinke, što je priličan broj, slijedom čega izlazi da je primjena modela tipa *M2* rentabilna, jer za osam ključnih riječi koje smo odabrali u sklopu modela tipa M2

možemo značajno poboljšati broj pogođenih lozinki. Najbolji rezultat ostvario je specijalizirani model u sprezi s modelom tipa *M2* i iscrpnim pretraživanjem za znakove na čije se otkrivanje troši malo vremena. Trend i brzina otkrivanja vide se iz odnosa utrošenog vremena i kumulativnog broja otkrivenih lozinki:

Tablica 10: Ukupni broj otkrivenih lozinki prema vremenu za listu *linkedin*

Vrijeme	JTRk	GENk	SPk	SPM2k	SPM2BFk	Razlika JTR / SPM2BF %
2013-10-01 00	700274	528594	833662	859071	951491	135,8741%
2013-10-01 01	813313	578704	840012	866280	1012807	124,5286%
2013-10-01 02	877691	621406	947492	1015868	1170525	133,3641%
2013-10-01 03	932089	654174	1051331	1077531	1173586	125,9092%
2013-10-01 04	967509	681438	1054435	1080463	1287631	133,0872%
2013-10-01 05	989678	703649	1055522	1081675	1295448	130,8959%
2013-10-01 06	1015374	724936	1131594	1162916	1297301	127,7658%
2013-10-01 07	1039629	743044	1168865	1198372	1298122	124,8640%
2013-10-01 08	1064746	760127	1175806	1202618	1298833	121,9852%
2013-10-01 09	1084765	775515	1177877	1204524	1432041	132,0139%
2013-10-01 10	1095133	788584	1179083	1205539	1434875	131,0229%
2013-10-01 11	1113075	799865	1179858	1206322	1435635	128,9792%
2013-10-01 12	1135978	810948	1180371	1206906	1436083	126,4182%
2013-10-01 13	1143312	821682	1180788	1207368	1436365	125,6319%
2013-10-01 14	1153220	831316	1181183	1207839	1436581	124,5713%
2013-10-01 15	1159736	840546	1181603	1316088	1436784	123,8889%
2013-10-01 16	1168369	848790	1318681	1340559	1436974	122,9897%
2013-10-01 17	1177382	856764	1330475	1342049	1458665	123,8905%
2013-10-01 18	1191837	864637	1331586	1342691	1459713	122,4759%
2013-10-01 19	1204917	871416	1332127	1343055	1460147	121,1824%
2013-10-01 20	1210980	879369	1332461	1343361	1467774	121,2055%
2013-10-01 21	1216548	887040	1332713	1343595	1471217	120,9337%
2013-10-01 22	1221435	893905	1332887	1343813	1508558	123,5070%
2013-10-01 23	1225503	900128	1333042	1343936	1509001	123,1332%

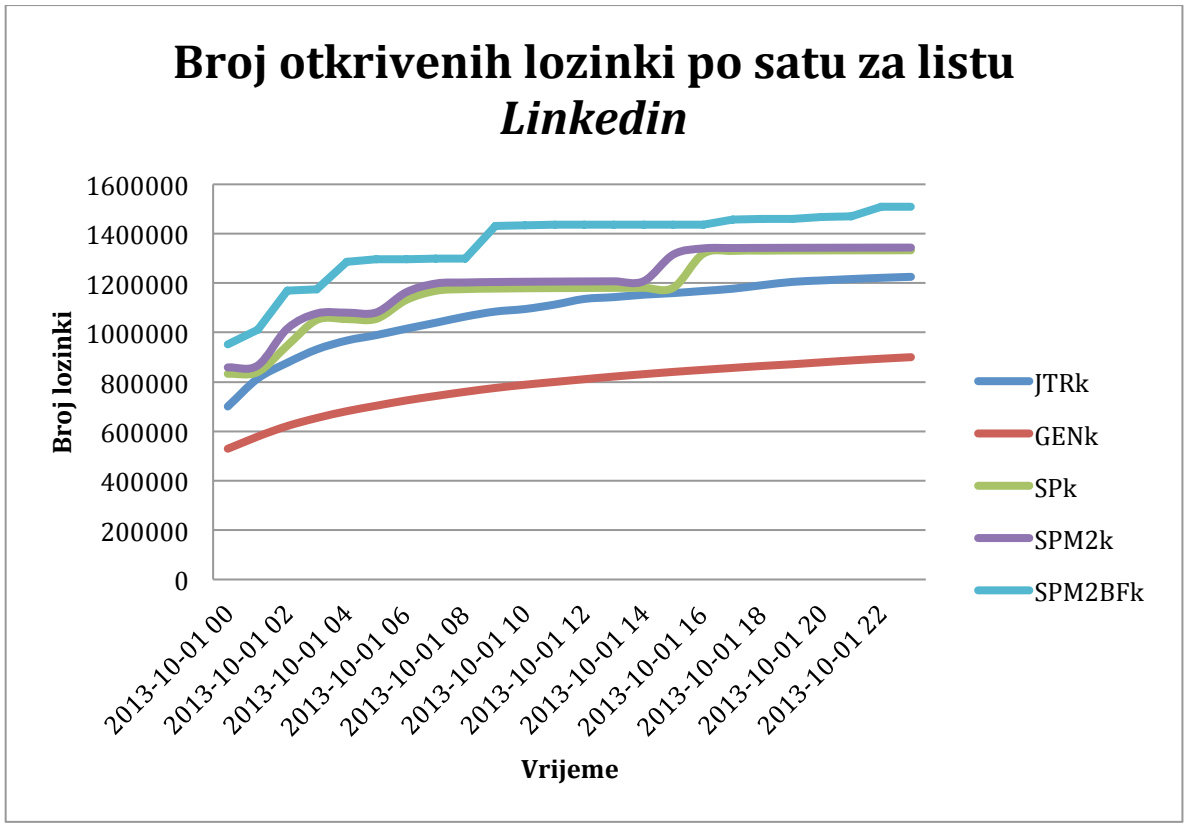
Iz prijašnje tablice vidi se da su značajni skokovi u broju otkrivenih lozinki uočeni na početku analize, tj. unutar prvih 12 sati, tijekom kojih iscrpnim pretraživanjem metoda *SPM2BF* uspijeva otkriti veliku količinu slabih lozinki, kao i velik broj lozinki korištenjem početnih pravila kombinacije. Dodatni uvid u performanse dobiva se promatranjem strukture lozinki koje je pojedina metoda otkrila. U sljedećoj tablici prikazan je broj lozinki koje je određena metoda otkrila u odnosu na duljinu lozinke:

Tablica 11: Duljina otkrivenih lozinki prema metodi za listu *linkedin*

Duljina loz.	JTR	GEN	SP	SPM2	SPM2BF
0	1	1	1	1	1
1	**12**	10	11	10	**12**
2	**12**	6	7	6	**12**
3	**23**	13	13	14	17
4	**67**	21	19	23	25
5	**133**	43	40	36	40
6	**519144**	255671	327497	330404	429517
7	**325288**	167801	268950	269294	284254
8	375991	260111	436764	438347	**454328**
9	3185	108694	189125	190335	**191104**
10	1247	68446	110615	110912	**113619**
11	165	23683	0	2130	**33644**
12	234	**10705**	0	2108	2111
13	1	**3217**	0	220	220
14	0	**1100**	0	43	43
15	0	**407**	0	15	15
16	0	**186**	0	37	38
17	0	**9**	0	1	1
18	0	**2**	0	0	0
20	0	**1**	0	0	0
23	0	**1**	0	0	0

Iz prijašnje tablice vidi se da je referentna implementacija u alatu *john* otkrila oko 26% više lozinki koje su duge 3 znaka, te više od dvostruko znakova duljine 4-5 znakova. Metoda *SPM2BF* otkrila je zamjetno veći broj lozinki duljine od 8 do 11 znakova, nakon čega metoda suženih modela biva značajno bolja od bilo koje druge implementacije. Sporija implementacija suženih modela zapravo je otkrila puno složenije lozinke, duljine 12 do 23 znaka, što govori o mogućoj korisnosti generaliziranih modela za analizu duljih, složenijih lozinki, na štetu jednostavnijih.

Rezultate iz tablice 10 mogu se prikazati sljedećim grafikonom:



Slika 1: Broj otkrivenih lozinki po satu za listu *linkedin*

8.4. Rezultati provjere drugog skupa podataka

Metoda *SPM2BF*, s istovjetnim postavkama, trebala je biti provjerena i na drugoj eksperimentalnoj listi u odnosu na referentnu implementaciju. U sljedećoj tablici prikazani su rezultati analize druge eksperimentalne liste:

Tablica 12: Prikaz vremena analize i broja otkrivenih lozinki za eksperimentalnu listu po metodama

Vrijeme	exJTR	exSPM2BF
2013-10-01 00	16689	**18174**
2013-10-01 01	1549	**2840**
2013-10-01 02	**747**	52
2013-10-01 03	478	**3123**
2013-10-01 04	**339**	90
2013-10-01 05	**423**	13
2013-10-01 06	**346**	22
2013-10-01 07	336	**2540**
2013-10-01 08	**148**	140
2013-10-01 09	**523**	22
2013-10-01 10	**130**	26
2013-10-01 11	**145**	11
2013-10-01 12	**85**	8
2013-10-01 13	**126**	6
2013-10-01 14	263	**276**
2013-10-01 15	**72**	17
2013-10-01 16	**89**	10
2013-10-01 17	**53**	72
2013-10-01 18	111	**145**
2013-10-01 19	132	**353**
2013-10-01 20	**62**	12
2013-10-01 21	**64**	13
2013-10-01 22	**36**	8
2013-10-01 23	**85**	7
Ukupno	23031	27980
Postotak	100,0000%	**121,4884%**

Iz prijašnje tablice također se vidi da je metoda *SPM2BF* brža za sličan iznos vremena kao i u slučaju liste *linkedin*. Trend i brzina otkrivanja vide se iz odnosa utrošenog vremena i kumulativnog broja otkrivenih lozinki:

Tablica 13: Ukupan broj otkrivenih lozinki prema vremenu za eksperimentalnu listu

Vrijeme	exJTRk	exSPM2BFk	Razlika
2013-10-01 00	16689	18174	108,8981%
2013-10-01 01	18238	21014	115,2210%
2013-10-01 02	18985	21066	110,9613%
2013-10-01 03	19463	24189	124,2820%
2013-10-01 04	19802	24279	122,6088%
2013-10-01 05	20225	24292	120,1088%
2013-10-01 06	20571	24314	118,1955%
2013-10-01 07	20907	26854	128,4450%
2013-10-01 08	21055	26994	128,2071%
2013-10-01 09	21578	27016	125,2016%
2013-10-01 10	21708	27042	124,5716%
2013-10-01 11	21853	27053	123,7954%
2013-10-01 12	21938	27061	123,3522%
2013-10-01 13	22064	27067	122,6749%
2013-10-01 14	22327	27343	122,4661%
2013-10-01 15	22399	27360	122,1483%
2013-10-01 16	22488	27370	121,7094%
2013-10-01 17	22541	27442	121,7426%
2013-10-01 18	22652	27587	121,7862%
2013-10-01 19	22784	27940	122,6299%
2013-10-01 20	22846	27952	122,3496%
2013-10-01 21	22910	27965	122,0646%
2013-10-01 22	22946	27973	121,9080%
2013-10-01 23	23031	27980	121,4884%

Ovdje se vidi da značajnih oscilacija nije bilo, osim što je metoda *SPM2BF* na početku analize bila nešto sporija. To je moguće zato što početna analiza iscrpnim pretraživanjem nije bila previše učinkovita, budući da korisnici iz druge eksperimentalne liste nisu koristili takve slabe lozinke. U nastavku analize metoda *SPM2BF* održava napredak od oko 20%.

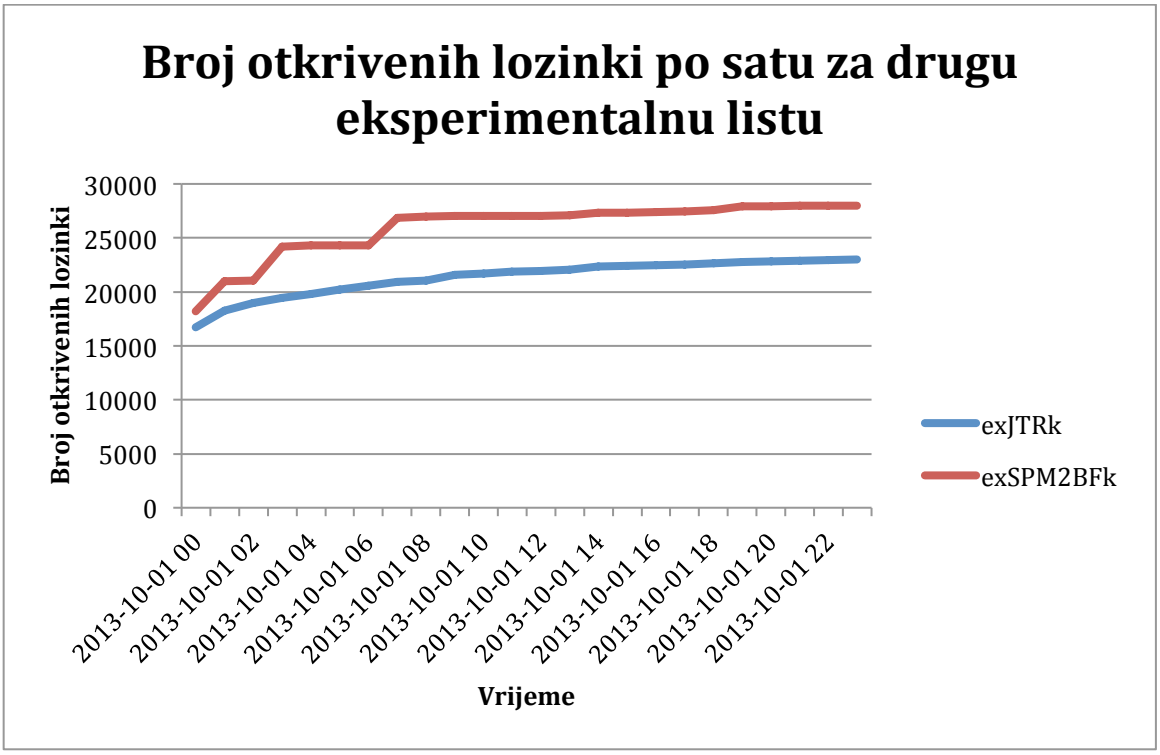
Prikaz duljine otkrivenih lozinki prema metodi za eksperimentalnu listu prikazan je sljedećom tablicom:

Tablica 14: Duljina otkrivenih lozinki prema metodi za eksperimentalnu listu

Duljina loz.	exJTR	exSPM2BF
0	**1**	**1**
1	**31**	**31**
2	**50**	49
3	**96**	89
4	**372**	353
5	**658**	586
6	**7842**	6419
7	**7335**	6694
8	6029	**6702**
9	460	**3970**
10	149	**2624**
11	5	**444**
12	3	**12**
13	0	**3**
14	0	**2**
18	0	**1**

Iz tablice se vidi da, do duljine od 8 znakova, referentna implementacija u alatu *john* po broju otkrivenih lozinki za neveliki broj vodi ispred implementacije *SPM2BF*, dok je implementacija *SPM2BF* puno bolja od referentne implementacije za lozinke od 8 znakova nadalje.

Rezultati iz tablice 13 mogu se prikazati sljedećim grafikonom:



Slika 2: Broj otkrivenih lozinki po satu za drugu eksperimentalnu listu

Referentna metoda otkrila je više kratkih lozinki od metode *SPM2BF* jer za otkrivanje kraćih lozinki koristi iscrpno pretraživanje, što je bolje kada je riječ o otkrivanju pseudoslučajnih lozinki. Metoda *SPM2BF* koristi iscrpno pretraživanje samo u slučaju nekih slabih uzoraka. Trivijalno je napisati pravilo poput *[{'gl':string.printable, 'l':10}]*, koje će alat *unhash* iskoristiti da iscrpno pretraži sve kombinacije mogućih znakova do duljine 10, što će uključiti i generirane pseudoslučajne lozinke. Praksa velikog iscrpnog pretraživanja izbjegnuta je jer su metode iscrpnog pretraživanja iznimno brze ako se izvode na *GPU*-platformama [75], a cilj ovog istraživanja je identifikacija uzoraka koji korisnici koriste za stvaranje svojih lozinki. Takve lozinke često su puno dulje, ali iz tog razloga postoji neki sustav prema kojem ih je korisnik stvorio. Drugi razlog je korištenje ove metode za analizu sporih *hash*-funkcija, gdje ubrzanje od 20% pruža veliku korist zbog malog broja kombinacija koje se mogu isprobati u jednoj sekundi.

9. Interpretacija rezultata istraživanja

U prijašnjim poglavlju rada, cilj i motivacija istraživanja prikazani su kroz sljedeća istraživačka pitanja:

1. Bi li metoda koja se temelji na poznavanju načina prema kojem korisnici stvaraju lozinke, izvedenom pomoću analize velikog broja realnih lozinki, omogućavala kvalitetniju analizu od referentne metode?

2. Kakve bi performanse imala nova metoda analize u omjeru broja otkrivenih lozinki i vremena trajanja analize, u odnosu na referentnu metodu?

Za provjeru brzine nove metode, kao referentna, odabrana je metoda implementacija analize zaštićenih lozinki u alatu *john*, verzije 1.7.9, s razinom dodataka *jumbo-7*, koja je u vrijeme provedbe eksperimenta (rujan-listopad 2013) bila najnovija verzija. Alat *john* popularan je kod konzultanata za sigurnost i administratora sustava i nalazi se u listi najpopularnijih alata koju održava zajednica stručnjaka za sigurnost [46]. On je ujedno proglašen i najpopularnijim alatom za *offline* analizu lozinki, koji radi na operacijskim sustavima *GNU/Linux* i podržava veliki broj algoritama za zaštitu.

Na četiri predložene i evaluirane metode i mjerenja provedena nad listom *linkedin*, zamijećene su sljedeće pojave:

1. Od četiri predložene metode, najbolje performanse imala je metoda *SPM2BF*.
2. U tablici 9 vidi se da je prilikom usporedbe referentne metode i metode *SPM2BF*, potonja otkrila 283.498 lozinki više od referentne metode, što je 23,1332% više otkrivenih lozinki u usporedbi sa referentnom metodom, unutar perioda od 24 sata.
3. U tablici 10 vidi se da je metoda *SPM2BF* u istom odsječku vremena otkrila veći broj lozinki negoli referentna implementacija.
4. U tablici 11 vidi se da je metoda *SPM2BF* otkrila značajno veći broj lozinki duljine od 8 do 11 znakova negoli referentna metoda, koja je otkrila veći broj znakova u rasponu od 3 do 7 znakova.

U cilju dodatne verifikacije najbolje metode, ona je uspoređena s referentnom implementacijom na drugom skupu podataka, pri čemu su na provedenim mjerenjima zamijećene sljedeće pojave:

1. U tablici 12 vidi se da je metoda *SPM2BF* otkrila 4.949 lozinki više od referentne implementacija, što je 21,4884% više od referentne implementacije.
2. U tablici 13 vidi se da je metoda *SPM2BF* u odnosu na referentnu implementaciju, u istom odsječku vremena, otkrila veći broj lozinki negoli referentna implementacija.
3. U tablici 14 vidi se da je metoda *SPM2BF* otkrila veći broj lozinki duljine od 8 do 18 znakova od referentne implementacije, koja je otkrila veći broj lozinki duljine od 2 do 7 znakova.

Iz provedenih eksperimenata i njihovih rezultata, prikazanih u tablicama 9, 10, 11, 12, 13 i 14 može se zaključiti da:

1. Specijalizirani model, u sprezi s modelima tipa *M2* i iscrpnim pretraživanjem za ograničeni skup znakova, imenovan modelom *SPM2BF*, ima bolje performanse od danas standardne metode koja se koristi za analizu zaštićenih lozinki, što potvrđuje hipotezu:

   H1: *Metoda kriptoanalize zaštićenih lozinki koja se temelji na poznavanju načina na koji korisnici te lozinke stvaraju, iskazana pripadajućim razvijenim modelima, omogućit će zamjetno bolje performanse.*

2. Metoda *SPM2BF* otkriva veći broj lozinki u kraćem vremenskom intervalu, što potvrđuje hipotezu:

   H2: *Posebice, metoda iz hipoteze H1 pokazat će se zamjetno učinkovitijom u pogledu omjera broja otkrivenih lozinki i vremena trajanja analize, u odnosu na referentnu metodu.*

10. Zaključak i buduća istraživanja

Istraživanje provedeno u sklopu ove disertacije bilo je usmjereno na problem identifikacije slabih lozinki koje informacijske sustave čine ranjivima. Prvi tip ranjivosti čine slabe lozinke koje su dostupne na mrežnim servisima ili mrežnim uređajima dostupnima s Interneta, gdje slabe lozinke predstavljaju industrijske početne lozinke koje nisu promijenjene, slabe lozinke koje je odabrao korisnik ili lozinke koje je ugradio proizvođač u sklopu tzv. *stražnjih vrata*, koja njemu omogućavaju pristup u sustav, najčešće radi omogućavanja korisničke potpore, a ponekad i iz malicioznih nakana.

Zlonamjerni hakeri i maliciozni kod koji imaju raspoložive i implementirane metode pogađanja lozinki na sustavima koji su javno dostupni, koriste automatizirane skenere ili sustave, nakon što se objavi postojanje *stražnjih vrata*, ciljano napadaju i traže takva *stražnja vrata*. Takvi sustavi bivaju zaraženi i dalje služe za napade na druge sustave, uz velike štete koje nastaju zbog krađe ili uništavanja podataka koji se na pogođenom sustavu nalaze. Zlonamjerni napadači u tom su slučaju u prednosti, jer uz puno bolje liste lozinki koje su pribavili i poboljšavali tokom svojih ilegalnih aktivnosti, te međusobnim razmjenama lozinki, imaju i ogroman broj računala za širenje svojih pretraživanja i za daljnje napade. U tom slučaju, napadači imaju nesrazmjernu prednost ispred administratora sustava koji su slabo obučeni za provjeru ranjivosti.

Cilj istraživanja, iskazan modelima tipa *M1*, bio je sustavskim administratorima i konzultantima za sigurnost omogućiti stvaranje puno šire liste lozinki, koja bi odgovarala listama koje se i inače koriste u realnim napadima protiv njihovih sustava. Implementacija modela tipa *M1* omogućava preuzimanje više lista lozinki iz raznih repozitorija koji su dostupni na web-u, i stvaranje jedinstvene liste lozinki. Takva lista može se koristiti u sklopu postojećih alata za provjeru ranjivosti, u cilju provjere mrežnih servisa ili uređaja i potrage za lošim lozinkama te mogućim *stražnjim vratima*. Tijekom ispitivanja utvrđeno je da takva lista lozinki uspješno otkriva industrijske lozinke na uređajima koji su bili dostupni autoru za potrebe eksperimentiranja.

Drugi oblik ranjivosti vezan je uz slabe lozinke koje se čuvaju na samim sustavima. U tom slučaju, na sustavu se nalazi određeni broj korisničkih imena i lozinki koje su zaštićene jednosmjernom kriptografskom funkcijom. Ukoliko se te lozinke koriste na mrežnom sustavu, metodom koja koristi modele tipa *M1* moguće je provjeriti njihovu snagu. Međunarodne norme, kao što je *ISO 27002* [32], i najbolja praksa, definiraju zahtjeve za provjerom snage i otpornosti lozinke. Drugi značajniji problem u tome je da zlonamjerni napadači, dođe li do sigurnosnog incidenta i provale u sustav, najčešće sami nastoje probiti lozinke na listama koje su u sklopu sustava dostupne, kako bi ta korisnička imena i lozinke mogli zloupotrijebiti protiv drugih sustava, računajući na nepažljivost korisnika u smislu korištenja iste lozinke na više mjesta. Takvi incidenti sve su češći i predstavljaju problem protiv kojega treba proaktivno djelovati detekcijom slabih lozinki i predočavanjem korisniku zahtjeva za promijenom lozinke, ili, u nekim slučajevima, blokiranjem korisničkog računa zbog zaštite sustava.

Jedan od načina rješavanja takvog problema jest korištenje sustava koji od korisnika zahtijeva da odabere snažnu lozinku koja odgovara zahtjevima složenosti koji su najčešće vezani uz minimalnu duljinu od 8 znakova, korištenje jednog velikog slova, jednog broja i jednog posebnog znaka ili nekih drugih, najčešće strožih zahtjeva. Korisnici najčešće koriste razne metode da zaobiđu takve zahtjeve i lozinku što lakše zapamte. Iz tog razloga razvijen je postupak koji omogućava generiranje lista lozinki koje sadrže korpus riječi za odabrane ključne riječi koje unese osoba koja testira sustav sigurnosti. Kao riješenje problema, razvijena je metoda koja koristi internetsku tražilicu Google da kreira proizvoljan skup riječi s *web*-mjesta koja je tražilica predložila slijedom izabranih ključnih riječi. Uz pomoć te metode kreiran je bazni rječnik od velikog broja popularnih kategorija riječi (pojmova), uz pozamašan broj preuzetih *web*-mjesta po kategoriji, u cilju stvaranja raspršene liste ključnih riječi koja predstavlja baznu listu riječi za daljnju analize snage lozinki. Također je korištena i slika popularne *web*-enciklopedije *Wikipedia*, koja je služila za stvaranje rječnika koji je korišten u sklopu metode strojnog učenja za potrebe identifikacije i klasifikaciju načina na koji korisnici stvaraju svoje lozinke, jer je u tom slučaju zahtjevan opsežan rječnik iz kojeg bi se moglo (trebalo) identificirati što više lozinki.

Da bi se omogućilo bolje testiranje snage lozinki, uvjet u sklopu istraživanja bio je da korisnici ne odabiru svoje lozinke nasumično, već da imaju neki sustav ili uzorak po kojem to rade. Za potrebe modeliranja ponašanja korisnika korištene su liste lozinki koje su objavili zlonamjerni napadači nakon poznatih incidenata, koje su zatim podijeljene na skup za učenje i skup za verifikaciju. Razvijena je nova metoda, koju je autor nazvao *metodom sita*, koja se temelji na hijerarhiji klasifikatora i razmjeni poruka između istih, što omogućava klasifikaciju kompleksnih sadržaja kao što su lozinke. U ovom istraživanju *metoda sit*a primijenjena je u cilju otkrivanja pravila i zakonitosti načina na koje korisnici stvaraju svoje lozinke. Uz prikupljanje informacija o stvaranju lozinke, prikupljani su i elementi koje su korisnici koristili prilikom stvaranja svojih lozinki, u cilju dopunjavanja početne liste, generirane uz pomoć tražilice *Google*, realnim lozinkama koje su korisnici koristili. Prikupljeni sadržaji omogućavali su kreiranje četiri glavna tipa modela opisa korisničkog ponašanja, koji su bili nad skupom lozinki za verifikaciju uspoređivani s referentnom metodom. Kao referentna metoda odabrana je implementacija u alatu *john*, koji je jedan od najpopularnijih alata za testiranje snage lozinki koji se koristi od strane administratora sustava i konzultanata za sigurnost. Provedenim eksperimentima utvrđeno je da metoda razvijena analizom, provedenom nad skupom lozinki za učenje, pokazuje značajnije bolje performanse po pitanju broja otkrivenih lozinki u kraćem vremenskom intervalu od referentne implementacije, te da otkriva složenije lozinke negoli referentna metoda, čime su potvrđene obje postavljene hipoteze istraživanja.

Primjena metoda opisanih u ovom radu omogućava bolje performanse prilikom identificiranja slabih lozinki u kraćem vremenskom intervalu. Odabir tijekom razvoja metoda bio je da se one ne koncentriraju na pseudoslučajne lozinke, koje su svakako sigurne ako su dovoljno dugačke. Rezultati koji su dobiveni iz liste lozinki za učenje pokazuju da korisnici zaista koriste sustav kreiranja lozinki, te korisnost *metode sita*. Nastavak istraživanja autor će usmjeriti prema optimizaciji pojedinih klasifikatora, istraživanju i razvoju novih klasifikatora, poboljšanju metode ocjene vjerojatnosti prilikom odabira klasifikacije i poboljšanju *metode sita* prema općenitijim problemima te unapređenju alata *unhash*.

11. Dostupnost alata i podataka

Implementacije metoda, alata i modela biti će dostupne nakon obrane doktorske disertacije u sklopu digitalnog repozitorija na *web*-stranici *github*, koja služi za pohranu izvornog koda i čuvanje repozitorija podataka. Alati i modeli biti će dostupni na adresi:

```
https://github.com/tkisason/unhash
```

Kao što je navedeno u zaključku, istraživanje i razvoj nad konceptima *metode sita* i razvoj budućih modela, koji bi trebali biti bolji trenutnih, nastavit će se u sklopu autorovih daljnjih istraživanja. Originalni izvorni kod i modeli uvijek će biti dostupni pod *git*-granom (eng. branch) "phd".

Sadržaj baze podataka nakon analize bio je prevelik da bi ga se moglo jednostavno priložiti ovom doktorskom radu, što je vidljivo iz upita u bazu podataka:

```
           relation             |    size
--------------------------------+------------
 public.lang_dicts              | 11 GB
 public.lang_dicts_trgm         | 6811 MB
 public.lang_dicts_pkey         | 5851 MB
 public.sieves                  | 3388 MB
 public.sieves_pkey             | 1156 MB
 public.plaintexts              | 983 MB
 public.ptstats                 | 925 MB
 public.unanalysed_plaintexts   | 631 MB
 public.plaintexts_pkey         | 402 MB
 public.unknown_stats           | 282 MB
 public.m2_models_pkey          | 238 MB
 public.unknown_stats_pkey      | 236 MB
 public.m2_models               | 184 MB
 public.element_stats           | 159 MB
 public.word_stats              | 131 MB
 public.m1_models_pkey          | 118 MB
 public.element_stats_pkey      | 107 MB
 public.m1_models               | 94 MB
 public.word_stats_pkey         | 45 MB
 public.m0_models_pkey          | 30 MB
 public.m0_models               | 26 MB
 public.mutation_stats_pkey     | 15 MB
 public.mutation_stats          | 15 MB
 public.pattern_stats           | 4000 kB
 public.pattern_stats_pkey      | 3264 kB
 public.experiments             | 1624 kB
 public.experiments_pkey        | 616 kB
 pg_toast.pg_toast_2618         | 264 kB
 public.translation_stats       | 112 kB
 pg_toast.pg_toast_2619         | 80 kB
 public.translation_stats_pkey  | 56 kB
```

```
 pg_toast.pg_toast_2619_index  | 16 kB
 pg_toast.pg_toast_2618_index  | 16 kB
 pg_toast.pg_toast_16458_index | 16 kB
 pg_toast.pg_toast_11564_index | 8192 bytes
 pg_toast.pg_toast_11559_index | 8192 bytes
 pg_toast.pg_toast_2609_index  | 8192 bytes
 pg_toast.pg_toast_16532_index | 8192 bytes
 pg_toast.pg_toast_11554_index | 8192 bytes
 pg_toast.pg_toast_2604_index  | 8192 bytes
(40 rows)
```

Cijela baza obima je oko 32 GB i s ostalim listama i analizama, kopijama slika *Wikipedije* i popratnim sadržajima, prelazi 100 GB, zbog čega je nije moguće na praktičan način držati javno dostupnom na internetu ili priložiti na DVD uz ovaj rad. Cijelu bazu i skupove podataka moguće je dobiti od autora kontaktom na e-mail:

- tonimir.kisasondi@foi.hr
- tonimir.kisasondi@gmail.com

12. Literatura

[1] J. C. Anderson, J. Lehnardt, N. Slater: *CouchDB: The Definitive Guide*; vol. 5. O'Reilly Media, 2010, p. 272.

[2] R. Armin, M. Fijałkowski: PyPy - a fast python implementation; 2013. Dostupno na: http://pypy.org/ [01.10.2013].

[3] at0m: HashCat, oclHashCat. Dostupno na: http://hashcat.net/ [01.10.2013].

[4] K. Atkinson: GNU Aspell. Dostupno na: http://aspell.net/ [01.10.2013].

[5] M. Bernaschi, M. Bisson, E. Gabrielli, S. Tacconi: An Architecture for Distributed Dictionary Attacks to Cryptosystems; *Journal of Computers*; vol. 4, no. 5; pp. 378–386; May.2009.

[6] J. Bonneau: The science of guessing: analyzing an anonymized corpus of 70 million passwords; in *2012 IEEE Symposium on Security and Privacy*.

[7] R. Bowes: Skull security password lists; 2013. Dostupno na: http://wiki.skullsecurity.org/Passwords [01.10.2013].

[8] W. E. Burr, D. F. Dodson, R. A. Perlner, W. T. Polk: NIST 800-63-1 Electronic Authentication Guidelines; *Nist Special Publication*; 2008.

[9] B. Cleary: https://wiki.python.org/moin/PythonSpeed/PerformanceTips; 2013. Dostupno na: https://wiki.python.org/moin/PythonSpeed/PerformanceTips [01.10.2013].

[10] Creative Commons: About The Licenses; *Creative Commons website*; 2013. Dostupno na: http://creativecommons.org/licenses/ [01.10.2013].

[11] F. J. Damerau: A technique for computer detection and correction of spelling errors; *Communications of the ACM*; vol. 7, no. 3; pp. 171–176; Mar.1964.

[12] Y. S. Dandass: Using FPGAs to Parallelize Dictionary Attacks for Password Cracking; *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*; pp. 485–485; Jan.2008.

[13] J. Dean, S. Ghemawat: MapReduce : Simplified Data Processing on Large Clusters; *Communications of the ACM*; vol. 51; pp. 1–13; 2008.

[14] S. Designer: John the ripper. Dostupno na: http://www.openwall.com/john/ [01.10.2013].

[15] S. Designer: John the ripper parallelization. Dostupno na: http://openwall.info/wiki/john/parallelization [01.10.2013].

[16] DigitalOcean: DigitalOcean. Dostupno na: www.digitalocean.com [01.10.2013].

[17] R. Dingledine, N. Mathewson, P. Syverson: Tor: The second-generation onion router; *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium*; vol. 13; p. 21; 2004.

[18] K. C. & M. Dirolf: *MongoDB: The Definitive Guide*; vol. 203. O'Reilly Media, 2010, p. 216.

[19] R. O. Duda, P. E. Hart, D. G. Stork: *Pattern Classification*; vol. 2. Wiley-Interscience, 2001, p. 654.

[20] N. Ferguson, B. Schneier, T. Kohno: *Cryptography Engineering - Design Principles and Practical Applications.* Wiley, 2010, pp. I–XXIX, 1–353.

[21] Z. za lingvistiku FFZG: Hrvatski nacionalni korpus; 2013. Dostupno na: http://www.hnk.ffzg.hr/ [01.10.2013].

[22] P. FIPS: FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3. 2008.

[23] K. Geisshirt: *Pluggable Authentication Modules. The Definitive Guide to PAM for Linux SysAdmins and C Developers.* Packt Publishing, 2007, p. 124.

[24] T. Gendrullis, M. Novotny, A. Rupp: A Real-World Attack Breaking A5/1 within Hours; *Cryptology ePrint Archive, Report 2008/147*; no. Horst Gortz Institute for IT-Security, Ruhr-University Bochum, Germany.; 2008.

[25] U. Germann, E. Joanis: Tightly Packed Tries : How to Fit Large Models into Memory , and Make them Load Fast , Too; *Computational Linguistics*; no. June; pp. 31–39; 2009.

[26] W. Glodek: Using A Specialized Grammar to Generate Probable Passwords; Florida State University; 2008.

[27] R. Griesemer, R. Pike, K. Thompson: Go programming language; 2013. Dostupno na: http://golang.org/ [01.10.2013].

[28] T. Güneysu, T. Kasper, M. Novotný, C. Paar, A. Rupp: Cryptanalysis with COPACOBANA; *IEEE Transactions on Computers*; vol. 57, no. 11; pp. 1498–1513; Nov.2008.

[29] V. Hauser: Hydra; 2013. Dostupno na: https://www.thc.org/thc-hydra/ [01.10.2013].

[30] M. Hellman: A cryptanalytic time-memory trade-off; *IEEE Transactions on Information Theory*; vol. 26, no. 4; pp. 401–406; Jul.1980.

[31] Institut za hrvatski jezik i jezikoslovlje: Hrvatski jezični korpus; 2011. Dostupno na: http://riznica.ihjj.hr/ [01.10.2013].

[32] ISO/IEC: ISO 27002:2013 - Information technology -- Security techniques -- Code of practice for information security management, vol. 2005. p. 80, 2013.

[33] P. Kacherginsky: PACK - Password analysis and cracking kit. Dostupno na: http://thesprawl.org/projects/pack/ [01.10.2013].

[34] M. Kandias, D. Gritzalis: Metasploit the Penetration Tester's Guide; *Computers & Security*; vol. 32; pp. 268–269; 2013.

[35] Y. Kedem. G, Ishihara: Brute force attack on unix password s with simd computer; *Proceedings of the 8th USENIX Security Symposium*; 1999.

[36] B. W. Kernighan, D. M. Ritchie: *The C programming language*. Prentice Hall, 1988, p. 228.

[37] T. Kišasondi: gcrack & gwordlist; 2011. Dostupno na: https://github.com/tkisason/gcrack [01.10.2013].

[38] T. Kišasondi, M. Bača, M. Schatten: Improving Computer Authentication Systems with Biometric Technologies; in *Information Systems Security, MIPRO 2006*; 2006.

[39] T. Kišasondi, Ž. Hutinski: Accelerating penetration testing with distributed computing; in *Proceedings of the 31st International Convention MIPRO*; 2008.

[40] G. Kroah-Hartman: *Linux in a Nutshell (5th edition)*. O'Reilly Media, 2006.

[41] S. Leary: *Beginning WordPress 3*. Apress, 2010, p. 350.

[42] V. I. Levenstein: *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*; vol. 10. 1966, p. 4.

[43] T. Lilja: GPGPUs , CUDA and OpenCL; 2010. Dostupno na: http://wiki.tkk.fi/display/GPGPUK2010/Home [01.10.2013].

[44] R. Lim: Parallelization of John the Ripper (JtR) using MPI; *Strategy*; vol. 1; pp. 1–9; 2004.

[45] L. Lueg: Pyrit. Dostupno na: http://code.google.com/p/pyrit/ [01.10.2013].

[46] G. Lyon: Security tools list - password crackers; 2013. Dostupno na: http://sectools.org/tag/crackers/ [01.10.2013].

[47] N. Marriott: tmux - terminal multiplexer; 2013. Dostupno na: http://tmux.sourceforge.net/ [01.10.2013].

[48] B. Marshall: Password research; 2013. Dostupno na: http://passwordresearch.com/ [01.10.2013].

[49] W. (AQR) McKinney: pandas : a python data analysis library; *New York*; 2009. Dostupno na: http://pandas.pydata.org [01.10.2013].

[50] J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, E. L. Aiden: Quantitative analysis of culture using millions of digitized books.; *Science (New York, N.Y.)*; vol. 331; pp. 176–182; 2011.

[51] S. B. Needleman, C. D. Wunsch: A general method applicable to the search for similarities in the amino acid sequence of two proteins.; *Journal of molecular biology*; vol. 48; pp. 443–453; 1970.

[52] L. Németh: Hunspell. Dostupno na: http://hunspell.sourceforge.net/ [01.10.2013].

[53] Network Working Group: *Request for Comments: 1321: The MD5 Message-Digest Algorithm*. .

[54] Network Working Group: *Request for Comments: 3174: US Secure Hash Algorithm 1 (SHA1)*. .

[55] P. Oechslin: *Making a Faster Cryptanalytic Time-Memory Trade-Off*; vol. 3. Springer, 2003, pp. 617–630.

[56] E. T. de Oliveira: pg_similarity; 2012. Dostupno na: http://pgsimilarity.projects.pgfoundry.org/ [01.10.2013].

[57] J. Perkins: *Python Text Processing with NLTK 2.0 Cookbook*. 2003, p. 544.

[58] N. Provos, D. Mazières, D. Mazi: A Future-Adaptable Password Scheme; *Most*; pp. 81–91; 1999.

[59] T. de Raadt, N. Hallqvist, A. Grabowski, A. D. Keromytis, N. Provos: Cryptography in OpenBSD: An Overview; in *Proceedings of the 1999 USENIX Annual Technical Conference, Freenix Track*; 1999; pp. 93–101.

[60] L. Richardson: Beautiful Soup Documentation. http://www.crummy.com/software/BeautifulSoup/documentation.html, 2007.

[61] G. van Rossum: Python programming language; *Python Software Foundation*; 2013. Dostupno na: http://www.python.org/.

[62] Y. Sasaki, L. Wang, K. Ohta, N. Kunihiro: New Message Difference for MD4; *14th International Workshop, FSE 2007, Luxembourg*; pp. 329–348; 2007.

[63] K. Scarfone, A. Orebaugh: 800-115: Technical Guide to Information Security Testing and Assessment Recommendations of the National Institute of Standards and Technology; *Nist Special Publication*; p. 80.

[64] B. Schneier: *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*; no. December. Wiley, 1996, p. 631.

[65] S. S. Skiena: *The Algorithm Design Manual*; vol. 1. Springer London, 2008, p. 1192.

[66] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, B. De Weger: MD5 considered harmful today; *25th Chaos Communication Congress*; pp. 1–25; 2008.

[67] T. Tomlinson: *Beginning Drupal 7*. Apress, 2010, p. 316.

[68] Unknown: Pastebin; 2013. Dostupno na: http://pastebin.com/ [01.10.2013].

[69] M. Weir, S. Aggarwal, B. De Medeiros, B. Glodek: Password Cracking Using Probabilistic Context-Free Grammars; *2009 30th IEEE Symposium on Security and Privacy*; pp. 391–405; May.2009.

[70] Wikipedia: Wikipedia database dumps; 2013. Dostupno na: https://en.wikipedia.org/wiki/Wikipedia:Database_download [01.10.2013].

[71] C. Wisniewski: The Conficker worm, three years and counting. Dostupno na: http://nakedsecurity.sophos.com/2011/11/24/the-conficker-worm-three-years-and-counting/ [01.10.2013].

[72] L. W. Wong: Information gathering using Google; in *Proceedings of 3rd Australian Computer, Network and Information Forensics Conference*; 2005; pp. 87–107.

[73] R. Wood: CeWL; 2013. Dostupno na: http://www.digininja.org/projects/cewl.php [01.10.2013].

[74] T. Xie, D. Feng: How To Find Weak Input Differences For MD5 Collision Attacks; *Advances in Cryptology - CRYPTO 2009*; pp. 1–20; 2007.

[75] A. Zonenberg: Distributed Hash Cracker : A Cross-Platform GPU-Accelerated Password Recovery System; *Rensselaer Polytechnic Institute 27*; 2009.

13. Prilog 1: Brzina provjere lozinki virtualnog stroja za analizu

```
root@hashkill1:~# ~/john-1.7.9-jumbo-7/run/john --test
Benchmarking: Traditional DES [128/128 BS SSE2-16]... DONE
Many salts: 2672K c/s real, 2672K c/s virtual
Only one salt:    2542K c/s real, 2542K c/s virtual

Benchmarking: BSDI DES (x725) [128/128 BS SSE2-16]... DONE
Many salts: 86912 c/s real, 86912 c/s virtual
Only one salt:    84864 c/s real, 84864 c/s virtual

Benchmarking: FreeBSD MD5 [128/128 SSE2 intrinsics 12x]... DONE
Raw:  21216 c/s real, 21216 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/64 X2]... DONE
Raw:  480 c/s real, 480 c/s virtual

Benchmarking: Kerberos AFS DES [48/64 4K]... DONE
Short:      252928 c/s real, 252928 c/s virtual
Long: 846848 c/s real, 846848 c/s virtual

Benchmarking: LM DES [128/128 BS SSE2-16]... DONE
Raw:  36342K c/s real, 36342K c/s virtual

Benchmarking: dynamic_0: md5($p) (raw-md5) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  17332K c/s real, 17332K c/s virtual

Benchmarking: dynamic_1: md5($p.$s) (joomla) [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 11312K c/s real, 11312K c/s virtual
Only one salt:    8969K c/s real, 8969K c/s virtual

Benchmarking: dynamic_2: md5(md5($p)) (e107) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  8943K c/s real, 8943K c/s virtual

Benchmarking: dynamic_3: md5(md5(md5($p))) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  6025K c/s real, 6025K c/s virtual

Benchmarking: dynamic_4: md5($s.$p) (OSC) [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 12887K c/s real, 12887K c/s virtual
Only one salt:    9287K c/s real, 9287K c/s virtual

Benchmarking: dynamic_5: md5($s.$p.$s) [128/128 SSE2 intrinsics 10x4x3]...
DONE
Many salts: 10625K c/s real, 10625K c/s virtual
Only one salt:    8040K c/s real, 8040K c/s virtual

Benchmarking: dynamic_6: md5(md5($p).$s) [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 17188K c/s real, 17188K c/s virtual
Only one salt:    6219K c/s real, 6219K c/s virtual

Benchmarking: dynamic_8: md5(md5($s).$p) [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 12605K c/s real, 12605K c/s virtual
```

```
Only one salt:    9656K c/s real, 9656K c/s virtual

Benchmarking:   dynamic_9:   md5($s.md5($p))   [128/128   SSE2   intrinsics
10x4x3]... DONE
Many salts: 12164K c/s real, 12164K c/s virtual
Only one salt:    5400K c/s real, 5400K c/s virtual

Benchmarking:  dynamic_10:  md5($s.md5($s.$p))  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 6198K c/s real, 6198K c/s virtual
Only one salt:    5380K c/s real, 5380K c/s virtual

Benchmarking:  dynamic_11:  md5($s.md5($p.$s))  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 6243K c/s real, 6243K c/s virtual
Only one salt:    5398K c/s real, 5398K c/s virtual

Benchmarking:   dynamic_12:   md5(md5($s).md5($p))   (IPB)   [128/128   SSE2
intrinsics 10x4x3]... DONE
Many salts: 2315K c/s real, 2315K c/s virtual
Only one salt:    1793K c/s real, 1793K c/s virtual

Benchmarking:  dynamic_13:  md5(md5($p).md5($s))  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 2320K c/s real, 2320K c/s virtual
Only one salt:    1794K c/s real, 1794K c/s virtual

Benchmarking:  dynamic_14:  md5($s.md5($p).$s)  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 9922K c/s real, 9922K c/s virtual
Only one salt:    5387K c/s real, 5387K c/s virtual

Benchmarking:  dynamic_15:  md5($u.md5($p).$s)  [32/64  64x2  (MD5_Body)]...
DONE
Many salts: 5849K c/s real, 5849K c/s virtual
Only one salt:    2432K c/s real, 2432K c/s virtual

Benchmarking:    dynamic_16:    md5(md5(md5($p).$s).$s2)    [32/64    64x2
(MD5_Body)]... DONE
Many salts: 2951K c/s real, 2951K c/s virtual
Only one salt:    1781K c/s real, 1781K c/s virtual

Benchmarking: dynamic_17: phpass ($P$ or $H$) [128/128 SSE2 intrinsics
4x4x3]... DONE
Raw:  11952 c/s real, 11952 c/s virtual

Benchmarking: dynamic_18: md5($s.Y.$p.0xF7.$s)(Post.Office MD5) [32/64 64x2
(MD5_Body)]... DONE
Many salts: 2310K c/s real, 2310K c/s virtual
Only one salt:    2175K c/s real, 2175K c/s virtual

Benchmarking:  dynamic_19:  Cisco  PIX  (MD5)  [128/128  SSE2  intrinsics
10x4x3]... DONE
Raw:  11566K c/s real, 11566K c/s virtual

Benchmarking: dynamic_20: Cisco PIX (MD5 salted) [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 11969K c/s real, 11969K c/s virtual
Only one salt:    9348K c/s real, 9348K c/s virtual

Benchmarking: dynamic_21: HTTP Digest Access Auth [128/128 SSE2 intrinsics
10x4x3]... DONE
Many salts: 1290K c/s real, 1290K c/s virtual
```

```
Only one salt:    1218K c/s real, 1218K c/s virtual

Benchmarking: dynamic_22: md5(sha1($p)) [128/128 SSE2 intrinsics 10x4x1]...
DONE
Raw:  4567K c/s real, 4567K c/s virtual

Benchmarking: dynamic_23: sha1(md5($p)) [128/128 SSE2 intrinsics 10x4x1]...
DONE
Raw:  4055K c/s real, 4055K c/s virtual

Benchmarking: dynamic_24: sha1($p.$s) [128/128 SSE2 intrinsics 10x4x1]...
DONE
Many salts: 6170K c/s real, 6170K c/s virtual
Only one salt:    5497K c/s real, 5497K c/s virtual

Benchmarking: dynamic_25: sha1($s.$p) [128/128 SSE2 intrinsics 10x4x1]...
DONE
Many salts: 6145K c/s real, 6145K c/s virtual
Only one salt:    5302K c/s real, 5302K c/s virtual

Benchmarking: dynamic_26: sha1($p) raw-sha1 [128/128 SSE2 intrinsics
4x1]... DONE
Raw:  7581K c/s real, 7581K c/s virtual

Benchmarking: dynamic_27: FreeBSD MD5 [128/128 SSE2 intrinsics 4x3]... DONE
Raw:  19836 c/s real, 19836 c/s virtual

Benchmarking: dynamic_28: Apache MD5 [128/128 SSE2 intrinsics 4x3]... DONE
Raw:  19584 c/s real, 19584 c/s virtual

Benchmarking: dynamic_29: md5(unicode($p)) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  11399K c/s real, 11399K c/s virtual

Benchmarking: dynamic_30: md4($p) (raw-md4) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  22891K c/s real, 22891K c/s virtual

Benchmarking: dynamic_31: md4($s.$p) [128/128 SSE2 intrinsics 10x4x3]...
DONE
Many salts: 16668K c/s real, 16668K c/s virtual
Only one salt:    12007K c/s real, 12007K c/s virtual

Benchmarking: dynamic_32: md4($p.$s) [128/128 SSE2 intrinsics 10x4x3]...
DONE
Many salts: 15508K c/s real, 15508K c/s virtual
Only one salt:    11427K c/s real, 11427K c/s virtual

Benchmarking: dynamic_33: md4(unicode($p)) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  13168K c/s real, 13168K c/s virtual

Benchmarking: dynamic_34: md5(md4($p)) [128/128 SSE2 intrinsics 10x4x3]...
DONE
Raw:  10177K c/s real, 10177K c/s virtual

Benchmarking: dynamic_35: sha1(uc($u).:.$p) (ManGOS) [128/128 SSE2
intrinsics 10x4x1]... DONE
Many salts: 5504K c/s real, 5504K c/s virtual
Only one salt:    4675K c/s real, 4675K c/s virtual

Benchmarking: dynamic_36: sha1($u.:.$p) (ManGOS2) [128/128 SSE2 intrinsics
10x4x1]... DONE
```

```
Many salts: 5690K c/s real, 5690K c/s virtual
Only one salt:    5048K c/s real, 5048K c/s virtual

Benchmarking: dynamic_37: sha1(lc($u).$p) (SMF) [128/128 SSE2 intrinsics
10x4x1]... DONE
Many salts: 6199K c/s real, 6199K c/s virtual
Only one salt:    5442K c/s real, 5442K c/s virtual

Benchmarking: dynamic_38: sha1($s.sha1($s.($p))) (Wolt3BB) [32/64 128x1]...
DONE
Many salts: 597120 c/s real, 597120 c/s virtual
Only one salt:    582336 c/s real, 588160 c/s virtual

Benchmarking: dynamic_1001 md5(md5(md5(md5($p)))) [128/128 SSE2 intrinsics
10x4x3]... DONE
Raw:  4365K c/s real, 4365K c/s virtual

Benchmarking:   dynamic_1002   md5(md5(md5(md5(md5($p))))   [128/128   SSE2
intrinsics 10x4x3]... DONE
Raw:  3499K c/s real, 3499K c/s virtual

Benchmarking: dynamic_1003 md5(md5($p).md5($p)) [32/64 64x2 (MD5_Body)]...
DONE
Raw:  1564K c/s real, 1564K c/s virtual

Benchmarking: dynamic_1004 md5(md5(md5(md5(md5(md5($p))))))  [128/128  SSE2
intrinsics 10x4x3]... DONE
Raw:  2920K c/s real, 2920K c/s virtual

Benchmarking: dynamic_1005 md5(md5(md5(md5(md5(md5(md5($p)))))))  [128/128
SSE2 intrinsics 10x4x3]... DONE
Raw:  2498K c/s real, 2498K c/s virtual

Benchmarking:    dynamic_1006    md5(md5(md5(md5(md5(md5(md5(md5($p)))))))
[128/128 SSE2 intrinsics 10x4x3]... DONE
Raw:  2198K c/s real, 2198K c/s virtual

Benchmarking:  dynamic_1007  md5(md5($p).$s)  (vBulletin)  [128/128  SSE2
intrinsics 10x4x3]... DONE
Many salts: 19967K c/s real, 19967K c/s virtual
Only one salt:    7790K c/s real, 7790K c/s virtual

Benchmarking: dynamic_1008 md5($p.$s) (RADIUS User-Password) [128/128 SSE2
intrinsics 10x4x3]... DONE
Many salts: 14451K c/s real, 14451K c/s virtual
Only one salt:    11238K c/s real, 11238K c/s virtual

Benchmarking: dynamic_1009 md5($s.$p) (RADIUS Responses) [128/128 SSE2
intrinsics 10x4x3]... DONE
Many salts: 17111K c/s real, 17111K c/s virtual
Only one salt:    13025K c/s real, 13025K c/s virtual

Benchmarking: dynamic_1010 md5($p null_padded_to_len_100) RAdmin v2.x MD5
[32/64 64x2 (MD5_Body)]... DONE
Raw:  2854K c/s real, 2854K c/s virtual

Benchmarking: Eggdrop Blowfish [32/64]... DONE
Raw:  19516 c/s real, 19516 c/s virtual

Benchmarking: DIGEST-MD5 C/R [32/64]... DONE
Raw:  1277K c/s real, 1277K c/s virtual
```

```
Benchmarking: Lotus Notes/Domino 6 More Secure Internet Password [8/64]...
DONE
Many salts: 66351 c/s real, 66351 c/s virtual
Only one salt:    39495 c/s real, 39495 c/s virtual

Benchmarking: EPiServer SID salted SHA-1 [32/64]... DONE
Many salts: 3496K c/s real, 3496K c/s virtual
Only one salt:    3192K c/s real, 3192K c/s virtual

Benchmarking:  HTTP  Digest  access  authentication  MD5  [128/128  SSE2
intrinsics 12x]... DONE
Many salts: 4000K c/s real, 4000K c/s virtual
Only one salt:    3804K c/s real, 3804K c/s virtual

Benchmarking: Invision Power Board 2.x salted MD5 [128/128 SSE2 intrinsics
12x]... DONE
Many salts: 8158K c/s real, 8158K c/s virtual
Only one salt:    7696K c/s real, 7696K c/s virtual

Benchmarking: Kerberos v4 TGT DES [32/64]... DONE
Raw:  2417K c/s real, 2417K c/s virtual

Benchmarking: Kerberos v5 TGT 3DES [32/64]... DONE
Raw:  43323 c/s real, 43323 c/s virtual

Benchmarking: MSCHAPv2 C/R MD4 DES [32/64]... DONE
Many salts: 2499K c/s real, 2499K c/s virtual
Only one salt:    1800K c/s real, 1800K c/s virtual

Benchmarking: LM C/R DES [32/64]... DONE
Many salts: 2485K c/s real, 2485K c/s virtual
Only one salt:    827904 c/s real, 827904 c/s virtual

Benchmarking: LMv2 C/R MD4 HMAC-MD5 [32/64]... DONE
Many salts: 749440 c/s real, 749440 c/s virtual
Only one salt:    584448 c/s real, 584448 c/s virtual

Benchmarking: NTLMv1 C/R MD4 DES (ESS MD5) [32/64]... DONE
Many salts: 2412K c/s real, 2412K c/s virtual
Only one salt:    1733K c/s real, 1733K c/s virtual

Benchmarking: NTLMv2 C/R MD4 HMAC-MD5 [32/64]... DONE
Many salts: 638976 c/s real, 638976 c/s virtual
Only one salt:    527616 c/s real, 527616 c/s virtual

Benchmarking: HalfLM C/R DES [32/64]... DONE
Many salts: 2404K c/s real, 2404K c/s virtual
Only one salt:    1203K c/s real, 1203K c/s virtual

Benchmarking: Netscreen MD5 [32/64]... DONE
Raw:  4286K c/s real, 4286K c/s virtual

Benchmarking: NT MD4 [128/128 X2 SSE2-16]... DONE
Raw:  32791K c/s real, 32791K c/s virtual

Benchmarking:  PHPS  md5(md5($pass).$salt)  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 21510K c/s real, 21510K c/s virtual
Only one salt:    7521K c/s real, 7521K c/s virtual

Benchmarking: Post.Office MD5 [32/64]... DONE
Many salts: 2836K c/s real, 2836K c/s virtual
Only one salt:    2639K c/s real, 2639K c/s virtual
```

```
Benchmarking: Mac OS X 10.4 - 10.6 salted SHA-1 [128/128 SSE2 intrinsics
4x]... DONE
Many salts: 12875K c/s real, 12875K c/s virtual
Only one salt:    10973K c/s real, 10973K c/s virtual

Benchmarking: CRC-32 [32/64]... DONE
Many salts: 72892K c/s real, 72892K c/s virtual
Only one salt:    38223K c/s real, 38223K c/s virtual

Benchmarking: GOST R 34.11-94 [64/64]... DONE
Raw:  328881 c/s real, 328881 c/s virtual

Benchmarking: Mac OS X Keychain PBKDF2-HMAC-SHA-1 3DES [32/64]... DONE
Raw:  398 c/s real, 398 c/s virtual

Benchmarking: Lotus Notes/Domino 5 [8/64]... DONE
Raw:  246528 c/s real, 246528 c/s virtual

Benchmarking: Generic salted MD4 [32/64]... DONE
Many salts: 5384K c/s real, 5384K c/s virtual
Only one salt:    4949K c/s real, 4949K c/s virtual

Benchmarking:  MediaWiki  md5($s.'-'.md5($p))  [128/128  SSE2  intrinsics
10x4x3]... DONE
Many salts: 10118K c/s real, 10018K c/s virtual
Only one salt:    5627K c/s real, 5627K c/s virtual

Benchmarking: M$ Cache Hash MD4 [32/64]... DONE
Many salts: 11446K c/s real, 11446K c/s virtual
Only one salt:    4699K c/s real, 4699K c/s virtual

Benchmarking:  M$  Cache  Hash  2  (DCC2)  PBKDF2-HMAC-SHA-1  [128/128  SSE2
intrinsics 4x]... DONE
Raw:  534 c/s real, 534 c/s virtual

Benchmarking: MS Kerberos 5 AS-REQ Pre-Auth MD4 MD5 RC4 [32/64]... DONE
Many salts: 616704 c/s real, 616704 c/s virtual
Only one salt:    339520 c/s real, 339520 c/s virtual

Benchmarking: MS SQL SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 9985K c/s real, 9985K c/s virtual
Only one salt:    6793K c/s real, 6793K c/s virtual

Benchmarking: MS SQL 2005 SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 10020K c/s real, 10020K c/s virtual
Only one salt:    8950K c/s real, 8950K c/s virtual

Benchmarking: MySQL 4.1 double-SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Raw:  4977K c/s real, 4977K c/s virtual

Benchmarking: MySQL [32/64]... DONE
Raw:  21469K c/s real, 21686K c/s virtual

Benchmarking: Netscape LDAP SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Raw:  9452K c/s real, 9452K c/s virtual

Benchmarking: NT MD4 [128/128 SSE2 intrinsics 12x]... DONE
Raw:  24558K c/s real, 24558K c/s virtual

Benchmarking: ODF SHA-1 Blowfish [32/64]... DONE
Raw:  734 c/s real, 734 c/s virtual
```

```
Benchmarking: Office 2007/2010 SHA-1/AES [32/64]... DONE
Raw:  24.0 c/s real, 24.0 c/s virtual

Benchmarking: Oracle 11g SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 9846K c/s real, 9846K c/s virtual
Only one salt:    8416K c/s real, 8416K c/s virtual

Benchmarking: Oracle 10 DES [32/64]... DONE
Raw:  524063 c/s real, 524063 c/s virtual

Benchmarking:   osCommerce   md5($salt.$pass)   [128/128   SSE2   intrinsics
10x4x3]... DONE
Raw:  12680K c/s real, 12680K c/s virtual

Benchmarking: phpass MD5 ($P$9) [128/128 SSE2 intrinsics 4x4x3]... DONE
Raw:  12000 c/s real, 12000 c/s virtual

Benchmarking: PIX MD5 [128/128 SSE2 intrinsics 10x4x3]... DONE
Many salts: 11766K c/s real, 11766K c/s virtual
Only one salt:    11769K c/s real, 11769K c/s virtual

Benchmarking: PKZIP [32/64]... DONE
Many salts: 5776K c/s real, 5776K c/s virtual
Only one salt:    4159K c/s real, 4159K c/s virtual

Benchmarking: RACF DES [32/64]... DONE
Many salts: 1974K c/s real, 1974K c/s virtual
Only one salt:    1866K c/s real, 1866K c/s virtual

Benchmarking: Raw MD4 [128/128 SSE2 intrinsics 12x]... DONE
Raw:  24404K c/s real, 24404K c/s virtual

Benchmarking: Raw MD5 [128/128 SSE2 intrinsics 12x]... DONE
Raw:  18800K c/s real, 18800K c/s virtual

Benchmarking: Raw SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Raw:  9457K c/s real, 9457K c/s virtual

Benchmarking: Raw SHA-1 LinkedIn [128/128 SSE2 intrinsics 4x]... DONE
Raw:  9478K c/s real, 9478K c/s virtual

Benchmarking: md5(unicode($p)) [128/128 SSE2 intrinsics 12x]... DONE
Raw:  18487K c/s real, 18487K c/s virtual

Benchmarking: Salted SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 9696K c/s real, 9696K c/s virtual
Only one salt:    8554K c/s real, 8554K c/s virtual

Benchmarking: SAP CODVN B (BCODE) [128/128 SSE2 intrinsics 12x]... DONE
Many salts: 5849K c/s real, 5849K c/s virtual
Only one salt:    5308K c/s real, 5308K c/s virtual

Benchmarking: SAP CODVN F/G (PASSCODE) [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 2552K c/s real, 2552K c/s virtual
Only one salt:    2491K c/s real, 2491K c/s virtual

Benchmarking: Generic salted SHA-1 [32/64]... DONE
Many salts: 2734K c/s real, 2734K c/s virtual
Only one salt:    2614K c/s real, 2614K c/s virtual

Benchmarking: SIP MD5 [32/64]... DONE
Raw:  1198K c/s real, 1198K c/s virtual
```

```
Benchmarking: VNC DES [32/64]... DONE
Raw:  1536K c/s real, 1536K c/s virtual

Benchmarking: WoltLab BB3 salted SHA-1 [32/64]... DONE
Raw:  588096 c/s real, 588096 c/s virtual

Benchmarking: HMAC MD5 [128/128 SSE2 intrinsics 12x]... DONE
Many salts: 5946K c/s real, 5946K c/s virtual
Only one salt:    3686K c/s real, 3686K c/s virtual

Benchmarking: HMAC SHA-1 [128/128 SSE2 intrinsics 4x]... DONE
Many salts: 2747K c/s real, 2747K c/s virtual
Only one salt:    1820K c/s real, 1820K c/s virtual

Benchmarking: Raw SHA-0 [32/64]... DONE
Raw:  2486K c/s real, 2486K c/s virtual

Benchmarking: Raw SHA-224 [32/64]... DONE
Raw:  1447K c/s real, 1447K c/s virtual

Benchmarking: Raw SHA-256 [32/64]... DONE
Raw:  1450K c/s real, 1450K c/s virtual

Benchmarking: Raw SHA-384 [64/64]... DONE
Raw:  1165K c/s real, 1165K c/s virtual

Benchmarking: Raw SHA-512 [64/64]... DONE
Raw:  1148K c/s real, 1148K c/s virtual

Benchmarking: HMAC SHA-224 [32/64]... DONE
Many salts: 402419 c/s real, 406483 c/s virtual
Only one salt:    316889 c/s real, 316889 c/s virtual

Benchmarking: HMAC SHA-256 [32/64]... DONE
Many salts: 401833 c/s real, 401833 c/s virtual
Only one salt:    343129 c/s real, 343129 c/s virtual

Benchmarking: HMAC SHA-384 [64/64]... DONE
Many salts: 319377 c/s real, 319377 c/s virtual
Only one salt:    310665 c/s real, 310665 c/s virtual

Benchmarking: HMAC SHA-512 [64/64]... DONE
Many salts: 317695 c/s real, 317695 c/s virtual
Only one salt:    310734 c/s real, 310734 c/s virtual

Benchmarking: Mac OS X 10.7+ salted SHA-512 [64/64]... DONE
Many salts: 1209K c/s real, 1209K c/s virtual
Only one salt:    1181K c/s real, 1181K c/s virtual

Benchmarking: hMailServer salted SHA-256 [32/64]... DONE
Many salts: 1471K c/s real, 1471K c/s virtual
Only one salt:    1387K c/s real, 1387K c/s virtual

Benchmarking: Sybase ASE salted SHA-256 [32/64]... DONE
Many salts: 194016 c/s real, 194016 c/s virtual
Only one salt:    192576 c/s real, 192576 c/s virtual

Benchmarking: DragonFly BSD $3$ SHA-256 w/ bug, 64-bit [32/64]... DONE
Many salts: 1449K c/s real, 1449K c/s virtual
Only one salt:    1401K c/s real, 1401K c/s virtual

Benchmarking: DragonFly BSD $4$ SHA-512 w/ bugs, 64-bit [64/64]... DONE
Many salts: 1164K c/s real, 1164K c/s virtual
```

```
Only one salt:    1142K c/s real, 1142K c/s virtual

Benchmarking: DragonFly BSD $3$ SHA-256 w/ bug, 32-bit [32/64]... DONE
Many salts: 1456K c/s real, 1456K c/s virtual
Only one salt:    1414K c/s real, 1414K c/s virtual

Benchmarking: DragonFly BSD $4$ SHA-512 w/ bugs, 32-bit [64/64]... DONE
Many salts: 1163K c/s real, 1163K c/s virtual
Only one salt:    1140K c/s real, 1140K c/s virtual

Benchmarking: Drupal 7 $S$ SHA-512 (x16385) [64/64]... DONE
Raw:  75.2 c/s real, 76.0 c/s virtual

Benchmarking: sha256crypt (rounds=5000) [32/64]... DONE
Raw:  183 c/s real, 183 c/s virtual

Benchmarking: sha512crypt (rounds=5000) [64/64]... DONE
Raw:  231 c/s real, 231 c/s virtual

Benchmarking: EPiServer salted SHA-1/SHA-256 [32/64]... DONE
Many salts: 2676K c/s real, 2676K c/s virtual
Only one salt:    2542K c/s real, 2542K c/s virtual

Benchmarking: KeePass SHA-256 AES [32/64]... DONE
Raw:  50.9 c/s real, 51.4 c/s virtual

Benchmarking: Password Safe SHA-256 [32/64]... DONE
Raw:  754 c/s real, 754 c/s virtual

Benchmarking: Django PBKDF2-HMAC-SHA-256 (x10000) [32/64]... DONE
Raw:  28.4 c/s real, 28.4 c/s virtual

Benchmarking: Raw SHA-1 (pwlen <= 15) [128/128 SSE2 intrinsics 4x]... DONE
Raw:  10920K c/s real, 11030K c/s virtual

Benchmarking: generic crypt(3) DES [?/64]... DONE
Many salts: 189312 c/s real, 189312 c/s virtual
Only one salt:    189216 c/s real, 189216 c/s virtual

Benchmarking: Tripcode DES [128/128 BS SSE2-16]... DONE
Raw:  2383K c/s real, 2383K c/s virtual

Benchmarking: SSH RSA/DSA (one 2048-bit RSA and one 1024-bit DSA key)
[32/64]... DONE
Raw:  20910 c/s real, 20910 c/s virtual

Benchmarking: PDF MD5 RC4 [32/64]... DONE
Many salts: 24010 c/s real, 24010 c/s virtual
Only one salt:    24216 c/s real, 24216 c/s virtual

Benchmarking: WPA-PSK PBKDF2-HMAC-SHA-1 [32/64]... DONE
Raw:  198 c/s real, 196 c/s virtual

Benchmarking: RAR3 SHA-1 AES (4 characters) [32/64]... DONE
Raw:  37.6 c/s real, 37.6 c/s virtual

Benchmarking: WinZip PBKDF2-HMAC-SHA-1 [32/64]... DONE
Raw:  238 c/s real, 238 c/s virtual

Benchmarking: dummy [N/A]... DONE
Raw:  72944K c/s real, 72944K c/s virtual
```

14. Biografija:

Tonimir Kišasondi rođen je 24.10.1984 u Varaždinu, gdje je završio osnovnu i srednju školu. Diplomirao je na Fakultetu organizacije i informatike, 2007. godine, s temom diplomaskog rada "Kriptoanaliza hash funkcija pomoću distribuiranog računalstva". Na istom Fakultetu zapošljava se 2008. godine kao znanstveni novak, na Katedri za informatičke tehnologije i računarstvo, gdje sudjeluje u izvođenju nastave iz kolegija "Sigurnost informacijskih sustava". Znanstveni i stručni interes vezan mu je uz informacijsku sigurnost, primijenjenu kriptografiju i otvorene sustave.

Znanstveni radovi u časopisima:

1. Kišasondi, Tonimir; Bača, Miroslav; Lovrenčić, Alen.
Biometric cryptography and network authentication. // Journal of Information and Organizational Sciences. 31 (2007) , 1; 91-99 (članak, znanstveni).

2. Kišasondi, Tonimir; Lovrenčić, Alen.
Selecting neural network architecture for investment profitability predictions. // Journal of Information and Organizational Sciencs. 30 (2006) , 1; 93-103 (članak, znanstveni).

Znanstveni radovi u zbornicima skupova s međunarodnom recenzijom:

1. Gržinić, Toni; Kišasondi, Tonimir; Šaban, Josip.
Detecting anomalous Web server usage through mining access logs // 24th Central European Conference on Information and Intelligent Systems / Tihomir Hunjak, Sandra Lovrenčić, Igor Tomičić (ur.). Varaždin : University of Zagreb, Faculty of Organization and Informatics, 2013. 228-296 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

2. Alen Delić; Natalia Nađ; Tonimir Kišasondi.
Analysis of user needs to create specific learning environments // Central European Conference on Information and Intelligent Systems 23rd International Conference / Hunjak,

Tihomir ; Lovrenčić Sandra ; Tomičić Igor (ur.).Varaždin : Faculty of organization and informatics, 2012. 395-399 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

3. Švogor, Ivan; Kišasondi, Tonimir.
Two Factor Authentication using EEG Augmented Passwords // Proceedings of the ITI 2012 34th International Conference on Information Technology Interfaces / Luzar-Stiffer, Vestna ; Jarec, Iva ; Bekic, Zoran (ur.). Dubrovnik : IEEE Region 8, 2012. 373-378 (međunarodna recenzija,objavljeni rad,znanstveni).

4. Kišasondi, Tonimir; Klasić, Domagoj; Hutinski, Željko.
A multiple layered approach to malware identification and classification problem // Proceedings of the 21th Central European Conference on Information and Intelligent Systems, September 22-24.9.2010, Varaždin, Croatia / Boris Aurer, Miroslav Bača, Markus Schatten (ur.). Varaždin : FOI Varaždin, 2010. 429-433 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

5. Balaban, Igor; Kišasondi, Tonimir.
A Lightweight ePortfolio Artifact Integrity Method // Proceedings of the ICL 2009 Conference: The Challenges of Life Long Learning / Michael E. Auer (ur.). Villach : Kassel University Press, 2009. 681-686 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

6. Hutinski, Željko; Kišasondi, Tonimir; Bambir, Danijela.
A method for analysis of corporate information security awareness // Proceedings of The 1th International Conference on Information Society and Information Technologies (ISIT 2009) / Mertik, Matej ; Damij, Nadja (ur.).Novo mesto : Faculty of information studies, 2009. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

7. Kišasondi, Tonimir; Hutinski, Željko.
Cryptographic routing protocol for secure distribution and multiparty negotiated access control // Conference proceedings: 19th International Conference Central European Conference on Information and Intelligent Systems / Aurer, Boris ; Bača, Miroslav ; Rabuzin, Kornelije (ur.).

Varaždin : FOI Varaždin, 2009. 267-270 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

8. Kišasondi, Tonimir; Hutinski, Željko.
Accelerating penetration testing with distributed computing // Proceedings of the 31st International Convention / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Mauher, Mladen ; Dragšić, Veljko (ur.). Opatija : Croatian Society for Information and Communication Technology, 2008. 100-103 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

9. Kišasondi, Tonimir; Hutinski, Željko; Dušak, Vesna.
Reverse engineering unknown protocols // Conference proceedings 19th International Conference Central European Conference on Information and Intelligent Systems / Aurer, Boris ; Bača, Miroslav (ur.). Varaždin : FOI Varaždin, 2008. 427-431 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

10. Kišasondi, Tonimir; Bača, Miroslav; Lovrenčić, Alen.
New biometric characterstic: Colorimetric keyboard // Proceedings of the 18th International Conference on Information and Intelligent Systems / Aurer, Boris ; Bača, Mieoslav (ur.).
Varaždin : Fakultet organizacije i informatike, 2007. 295-297 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

11. Lovrenčić, Alen; Čubrilo, Mirko; Kišasondi, Tonimir.
Modelling Functional Dependences in Databases using Mathematical Logic // 11th International Conference on Intelligent Engineering Sysms, Proceedings / Ridas, Imre J. (ur.).
Budapest : IEEE, 2007. 307-312 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

12. Kišasondi, Tonimir; Bača, Miroslav; Lovrenčić, Alen.
Biometric Network Authentication // Information and Intelligent Systems : IIS 2006 : Conference Proceedings / Aurer, Boris ; Bača, Miroslav (ur.).
Varaždin : FOI 2006, 2006. 293-296 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

13. Kišasondi, Tonimir; Bača, Miroslav; Schatten, Markus.
Improving Computer Authentification Systems with Biometric Technologies // Proceedings of the 29th International Convention : Information Systems Security : MIPRO 2006 / Čišić, Dragan ; Hutinski, Željko ; Baranović, Mirta ; Sandri, Roberto (ur.). Rijeka : Croatian Society for Information and Communication Tachnology, 2006. 166-171 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

14. Kišasondi, Tonimir; Lovrenčić, Alen.
New Neural Network Architevture and Algorithm for Memory Sequence Learning and Modelling // Conference Proceedings of the 16th International Conference on Information and Intelligent Systems / Aurer, Boris ; Bača, Miroslav (ur.). Varaždin : FOI, 2005. 415-422 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

Stručni radovi u zbornicima sa recenzijom:

1. Protrka, Nikola; Kišasondi, Tonimir.
Pregled metoda kibernetičkih napada na kritičnu nacionalnu infrastrukturu, studije slučaja iz prakse // Zbornik radova s III. Međunarodne znanstveno-stručne konferencije "Nove sigurnosne ugroze i kritična nacionalna infrastruktura" / doc.dr.sc. Krunoslav Antoliš (ur.).
Zagreb : Ministarstvo unutarnjih poslova Republike Hrvatske, 2013. 290-300 (predavanje,domaća recenzija,objavljeni rad,stručni).

2. Kišasondi, Tonimir; Hutinski, Željko.
Česte implementacijske greške kriptografskih sustava u informacijskim sustavima // KOM 2009.
(predavanje,domaća recenzija,objavljeni rad,stručni).

3. Kišasondi, Tonimir; Hutinski, Željko.
Analiza kvalitete kriptografskih sustava sa nepoznatim kriptoalgoritmima // CarNET Users Conference - 16-19.11.2009.
(predavanje,domaća recenzija,objavljeni rad,stručni).

Stručna predavanja:

1. Tonimir Kisasondi.
Security tips and tricks, part 0×01 // .
(pozvano predavanje,domaća recenzija,ppt prezentacija,stručni).

2. Tonimir Kisasondi.
Leveraging the public internet // .
(pozvano predavanje,ppt prezentacija).

3. Tonimir Kišasondi.
OSINT for the masses // .
(predavanje,domaća recenzija,ppt prezentacija).

4. Tonimir Kišasondi.
Leveraging public services: Doing interesting stuff with random services // -.
(predavanje,domaća recenzija,ppt prezentacija,stručni).

5. Tonimir Kišasondi.
Hide and seek: Interesting uses of forensics and covert channels // Dani otvorenih računarskih sustava "Hrvatska konvencija korisnika Linuxa DORS/CLUC 2011".
(predavanje,domaća recenzija,ppt prezentacija,stručni).

6. Tonimir Kišasondi.
Brzi savjeti za implementaciju kriptografije // FSEC - FOI Security / Tonimir Kišasondi, Vlatko Košturjak (ur.).
Varaždin : Fakultet organizacije i informatike, 2011. (predavanje,domaća recenzija,ppt prezentacija,stručni).

7. Kišasondi, Tonimir.
Sigurne komunikacije i tokeni // Dani otvorenih računarskih sustava "Hrvatska konvencija korisnika Linuxa DORS/CLUC 2010" : zbornik / Zimmer, Kristijan (ur.).
Zagreb (predavanje,domaća recenzija,ppt prezentacija,stručni).

8. Kišasondi, Tonimir.
Osiguravanje povjerljivosti i privatnosti komunikacija u nesigurnim okolinama // .
Čakovec : Autonomni centar: ACTnow, 2010. (predavanje,domaća recenzija,ppt prezentacija,stručni).

9. Kišasondi, Tonimir.
Analiza kvalitete sustava za enkripciju // .
(predavanje,domaća recenzija,ppt prezentacija,stručni).

10. Tonimir Kišasondi.
Privatnost i otvoreni kod // Dani otvorenih računarskih sustava / Hrvatska konvencija korisnika Linuxa DORS/CLUC 2009 / Kristijan Zimmer (ur.).
Zagreb, 2009. (predavanje,domaća recenzija,ppt prezentacija,stručni).

11. Bača, Miroslav; Schatten, Markus; Kišasondi, Tonimir.
Biometric characteristic's metrics in security systems // 1. Znanstveno-stručna konferencija s međunarodnim sudjelovanjem : Menadžment i sigurnost : M&S 2006 : Zbornik.
Čakovec (predavanje,ppt prezentacija,stručni).

12. Schatten, Markus; Bača, Miroslav; Kišasondi, Tonimir.
Unique Identification System // 1. znanstveno-stručna konferencija s međunarodnim sudjelovanjem, Menadžment i sigurnost : M&S 2006 : zbornik.
Čakovec (predavanje,ppt prezentacija,stručni).

13. Kišasondi, Tonimir; Bača, Miroslav; Schatten, Markus.
Sustavi za detekciju i prevenciju upada u korporativnim računalnim mrežama // 1. znanstveno-stručna konferencija s međunarodnim sudjelovanjem : Menadžment i sigurnost : M&S 2006 : zbornik.
Čakovec, 2006. (predavanje,ppt prezentacija,stručni).

Radovi u stručnim časopisima:

1. Tonimir Kišasondi.
Botnet: Je li i vaše računalo postalo zombi. // VIDI. 167 (2010) ; 120-121 (popularni rad, ostalo).

2. Bača, Miroslav; Kišasondi, Tonimir; Vuk, Igor.
Uzorak glasa kao biometrijski podatak. // Zaštita. 3 (2007) ; (popularan rad, stručni).

3. Bača, Miroslav; Schatten, Markus; Kišasondi, Tonimir.
Prstom otključaj vrata. // Zaštita. 2 (2006) ; (popularan rad, ostalo).

4. Bača, Miroslav; Schatten, Markus; Kišasondi, Tonimir.
Identifikacija šarenice - od SF-a do primjene. // Zaštita. 2 (2006) ; (popularan rad, ostalo).

5. Bača, Miroslav; Schatten, Markus; Kišasondi, Tonimir.
Potpis, kralj autentičnosti. // Zaštita. 2 (2006) ; (popularan rad, ostalo).