PROBIJANJE LOZINKI PUTEM HASHCAT PROGRAMA

Paleka, Zlatko

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:492950

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**



Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College





VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

PROBIJANJE LOZINKI PUTEM *hashcat* PROGRAMA

Zlatko Paleka

Zagreb, listopad 2019.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji nijedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao nijedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.“

U Zagrebu, datum.

Zlatko Paleka

Predgovor

Htio bih prvenstveno zahvaliti svom mentoru, prof. Robertu Petruniću, što me svojim znanjem i načinom predavanja inspirirao da se posvetim ovoj temi i odaberem upravo ovaj završni rad.

Također, htio bih zahvaliti svim profesorima i asistentima na Visokom učilištu Algebra što su uvijek strpljivo odgovarali na sva moja pitanja.

Prilikom uvezivanja rada, umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Većina ljudi danas koristi lozinke u nekom obliku. Također, činjenica je da velik broj ljudi ne vodi dovoljno računa o tome kakve lozinke koriste[1], tj. koliko jednostavnu ili kompleksnu lozinku imaju, niti razmišljaju o posljedicama kada bi im ta lozinka bila otkrivena.

Koliko duga treba biti lozinka kako bi bila sigurna za korištenje? Ako imamo zakriptiranu 7-zip datoteku, koje vrste napada postoje za probijanje te lozinke? Koliko je vremena potrebno za tako nešto te može li se i na koji način uopće probiti 7-zip lozinka? Odgovori na ova pitanja navode se u ovom završnom radu, bazirajući se prvenstveno na praktičnim mjerenjima vremena potrebnog za probijanje lozinke, koristeći program *hashcat*.

Hashcat podržava više različitih vrsta napada na sažetak neke lozinke, a glavna tri korištena u radu su napadi sirovom snagom, tj. maskom (engl. *Brute-force/mask*), napadi rječnicima (engl. *Dictionary*) te napadi korištenjem pravila (engl. *Rule-based*). Sva tri napada detaljno su prikazana te su izmjerena vremena potrebna kako bi se pokazalo koliko je koji napad bio uspješan ili neuspješan.

Ključne riječi: lozinka, probijanje, sažetak, napad.

[1] Izvor: https://www.helpnetsecurity.com/2019/05/13/people-password-habits/ Datum: 15. 12. 2019.

Abstract

Most people today are using passwords in some form. There is also a fact that a great majority of people are not taking enough care about what kind of passwords they use, that is whether they have a simple or a complex password, nor do they think about the consequences of what would happen, should their password be compromised.

What would be a sufficient password length in order to be safe for use? If we have an encrypted 7-zip file, what are the ways to attack this password, in order to crack it? How much time is needed for something like that, can it be done, and how can exactly do you approach cracking a 7-zip password? Answers to these questions were tried to be answered in this thesis, mainly being based on the practical measuring of time required to crack a password, using an application called *hashcat*.

Hashcat supports many different attack modes on the hash of some password, but the three main ones used in this thesis are as follows: Brute-force/mask attacks, Dictionary attacks and Rule-based attacks. All three have been shown in great detail, and the time needed to prove whether an attack was successful or not was measured and explained.

Key words: password, crack, hash, attack.

Sadržaj

1. Uvod 1

2. *Hash* 2

2.1. Općenito o *hashu* i lozinkama 2

2.1.1. Korisna znanja 3

2.1.2. Kako provjeriti *hash* neke datoteke 4

2.1.3. Metodologija probijanja lozinki 6

2.1.4. Upoznavanje softvera 7

2.2. Analiza probijanja različitih verzija *hasheva* 11

2.2.1. MD5 11

2.2.2. SHA-2 12

2.2.3. CRC32 13

2.3. *Hash* 7-zip datoteke 13

2.3.1. Kako generirati dobru lozinku 14

2.3.2. Kako generirati 7-zip *hash* 15

3. *Hashcat* 19

3.1. Što je *hashcat* i čemu služi? 19

3.1.1. Instalacija *hashcat* programa 20

3.1.2. *Hashcat* naredbe 21

3.1.3. Duljina ključa, prostor ključa 23

3.2. Vrste napada pomoću *hashcata* 26

3.2.1. Napad sirovom snagom (engl. *Brute-force Attack*) 26

3.2.2. Napad korištenjem maske (engl. *Mask Attack*) 27

3.2.3. Napad rječnicima (engl. *Dictionary Attack*) 35

3.2.4. Napad korištenjem pravila (engl. *Rule-based Attack*) 42
4. Probijanje lozinke 49
4.1. Određivanje vrste napada 49
4.1.1. Duljina lozinke i njezin utjecaj 50
4.1.2. Generiranje slučajno odabranih lozinki 50
4.2. Usporedna analiza napada na različitim računalima 51
4.2.1. Specifikacije računala za probijanje lozinke 51
4.2.2. Određivanje snage *hasha* 54
4.2.3. Usporedbe s drugim sustavima 57
4.3. Mjerenje vremena 60
4.3.1. Lozinke do pet znakova – napad sirovom snagom 61
4.3.2. Lozinke od pet do deset znakova – napad rječnikom 69
4.3.3. Lozinke od deset do petnaest znakova – napad korištenjem pravila 76
Zaključak 92
Popis kratica 95
Popis slika 96
Popis tablica 100
Literatura 101

1. Uvod

U današnje vrijeme, u svijetu informacijske tehnologije, česta je tema o sigurnosti lozinki. Gotovo da i ne postoji osoba koja koristi internet, a da nema barem jednu svoju lozinku. No, koliko su lozinke sigurne i mogu li se probiti?

Postoje specijalizirani programi koji služe upravo toj svrsi – probijanju lozinki. U ovom radu bit će obrađen jedan takav program koji se zove *hashcat*. Objasnit će se što je točno potrebno i kako se općenito pristupa pokušaju probijanja lozinke koristeći *hashcat* program.

Za probijanje lozinke potrebno je određeno vrijeme. Cilj ovog rada je predočiti, u mjerljivim jedinicama, koliko je vremena potrebno da bi se lozinka određene duljine uspješno ili neuspješno probila. Što se toga tiče, *hashcat* pruža velik broj mogućnosti u obliku različitih vrsta napada, a tri koja će se koristiti su: napadi sirovom snagom, tj. maskom, napadi rječnicima ili listama riječi te napadi korištenjem pravila. Svaki od napada ima određene prednosti i mane koji će se na konkretnim primjerima detaljno objasniti.

Prikazat će se što je sažetak (engl. *hash*) te kako je povezan s lozinkama. Detaljno će se obraditi program *hashcat* te će se na kraju fokusirati na praktično mjerenje vremena potrebnog za probijanje lozinki različitih duljina i sažetaka koristeći različite vrste napada.

2. *Hash*

Kada se govori o probijanju lozinki, često se može čuti riječ „sažetak“ (engl. *hash*). Konkretno, radi se o algoritmima za izračunavanje sažetka, tj. o kriptografskoj *hash* funkciji. U ovom poglavlju detaljnije će se opisati što je točno *hash,* koja je njegova uloga u probijanju lozinki te kako je s njima povezan. Koristit će se engleska riječ *hash* umjesto riječi sažetak.

2.1. Općenito o *hashu* i lozinkama

Kriptografska *hash* funkcija je matematički algoritam koji proizvoljnu zadanu ulaznu informaciju ili vrijednost mapira u niz fiksne veličine.[2] To je uvijek funkcija u jednom smjeru (engl. *one-way function*), što znači da se za istu ulaznu vrijednost uvijek dobije ista izlazna vrijednost, ali nemoguće je (teoretski) od izlazne vrijednosti dobiti natrag ulaznu. Ta izlazna informacija, fiksni niz koji se dobije, naziva se *hash*.

Sigurna kriptografska *hash* funkcija nastoji onemogućiti da se za dva različita ulazna podatka dobije isti *hash*. Isto tako, ne bi smjelo biti moguće uzeti samo *hash* i od njega „otkriti“ koja je bila ulazna vrijednost, tj. informacija. Postoji puno različitih kriptografskih *hash* funkcija koje se koriste u tu svrhu, a neke od njih bit će obrađene u ovom radu.

Sljedeća tablica na jednostavan način ilustrira kako rade algoritmi sažimanja. Koristit će se CRC32 *hash* funkcija. Ona za bilo koju ulaznu informaciju kreira *hash* fiksne veličine znakova.

Ulazna informacija	*hash* funkcija	Izlazna informacija (*hash*)
ana	CRC32	7ca41aa8
Ana	CRC32	4fd22902
Ana i Ivo	CRC32	2fdf5df0
Šalica	CRC32	fc94f1ff
Ana i Ivo piju kavu iz šalice	CRC32	3afaed79
Ana i Ivo piju kvau iz šalice	CRC32	f0f1db57

Tablica 2.1. CRC32 *hash* funkcija

Može se primijetiti da čak i mala promjena ulazne vrijednosti (u ovom slučaju namjerno krivo napisana riječ „kvau“ umjesto „kavu“) potpuno mijenja vrijednost izlazne informacije.

[2] Izvor: https://en.wikipedia.org/wiki/Cryptographic_hash_function Datum: 9. 10. 2019.

Kada bi netko imao pristup samo *hashevima*, ne bi mogao otkriti ulaznu vrijednost za svaki *hash*. Morao bi pokušavati sve moguće kombinacije ulaznih vrijednost da se vidi je li dobivena identična izlazna vrijednost. No, postoji li ipak neki način kojim se može doći do ulaznih informacija?

U ovom se području otkriva poveznica *hasheva* i lozinki. Većina ljudi danas ima svoju e-poštu, npr. *yahoo*. Kada se upiše korisničko ime i lozinka za ulazak u *yahoo* e-poštu, ta se ista lozinka sprema na *yahoo* poslužiteljima u *hash* obliku. Ako bi se dogodio upad u *yahoo* te ako bi potencijalni napadač došao do baze u kojoj se nalaze lozinke u *hash* zapisu, ti se *hashevi* mogu napadati raznim tehnikama kako bi se došlo do izvorne lozinke u običnom tekstualnom zapisu. Napadi neće uvijek biti uspješni jer to ovisi o vrsti *hash* zapisa, o jačini lozinke itd.

Slično vrijedi i za 7-zip[3] datoteke koje su kriptirane nekom lozinkom. Postoji način kako se može izvući *hash* datoteke te se tada „napada“ taj *hash* da bi se dobila lozinka. Neke od tih tehnika bit će obrađene u ovom radu.

2.1.1. Korisna znanja

U domeni kriptografije postoje tri izraza koji ljude, nove u tom području, lako mogu zbuniti. Ne smiju se pobrkati algoritmi sažimanja (engl. *hashing*), enkodiranja (engl. *encoding*) i šifriranja (engl. *encryption*).

Algoritmi sažimanja (engl. *hashing*)

Algoritam sažimanja je funkcija u jednom smjeru koja od ulazne vrijednosti vraća fiksni niz znakova (slova, brojevi i slično). To je samo prikaz koji je često heksadecimalni niz, ali to ovisi o *hash* funkciji koja se koristi.

Enkodiranje (engl. *encoding*)

[3] Izvor: https://www.7-zip.org/.

Enkodiranje je proces transformiranja podataka kako bi mogli biti preneseni preko nekog komunikacijskog kanala. Enkodiranje se ne koristi za siguran prijenos osjetljivih informacija s obzirom na to da se tu koristi algoritam koji je javno dostupan svima te dobro dokumentiran.[4]

Enkripcija (engl. *encrypting*)

Enkripcija je slična *hashiranju*, ali to je funkcija u dva smjera. Od izlaznih podataka mogu se dobiti i ulazni podaci uz pomoć ključa (ili ključeva). Postupak kriptiranja podataka naziva se kriptiranje, a u suprotnom smjeru dekriptiranje.

2.1.2. Kako provjeriti *hash* neke datoteke

Kao što se podacima može izračunati *hash*, tako se i svakoj datoteci na bilo kojem sustavu može dodijeliti njezin *hash*. Bitna odlika je da ako se promijeni samo jedan bit te datoteke, njezin se *hash* također mijenja. Često se može vidjeti da razni isporučitelji datoteka po internetu (npr. Linux .iso datoteke s cijelim operativnim sustavima) prikažu i koji je točno *hash* te datoteke. To nam služi da nakon što skinemo tu datoteku s interneta, možemo provjeriti njezin *hash*. Ako odgovara onome što je pružatelj te datoteke napisao, možemo biti sigurni da imamo identičnu kopiju te datoteke.

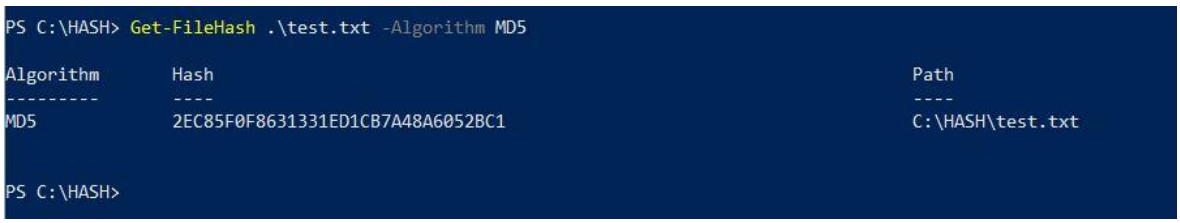
Na sljedećem primjeru pokazat će se kako se može provjeriti *hash* proizvoljno kreirane datoteke. Napravljen je folder C:\HASH, a u njemu je datoteka test.txt u kojoj je neki tekst. Naredba na Windows operativnom sustavu kojom možemo provjeriti *hash* te datoteke je *powershell* cmdlet Get-FileHash, prikazana na Slika 2.1. Prikaz *powershell* naredbe za ispis *hasha* datoteke.

[4] Izvor: https://stackoverflow.com/questions/4657416/difference-between-encoding-and-encryption Datum: 14. 10. 2019.



Slika 2.1. Prikaz *powershell* naredbe za ispis *hasha* datoteke

Primjećuje se da je standardna vrijednost algoritma koju *powershell* koristi algoritam SHA256. Odabir drugačijeg algoritma može se jednostavno promijeniti dodavanjem parametra „-Algorithm“ u naredbi.



Slika 2.2 Korištenje MD5 algoritma u *powershell* naredbi

Postoje i drugi načini kako se može izračunati *hash* neke datoteke. U jednom od sljedećih poglavlja bit će spomenuti programi koji služe upravo tome.

2.1.3. Metodologija probijanja lozinki

Metodologija[5] kojom se pokušava probiti lozinka sastoji se od nekoliko koraka određenog redoslijeda. To su sljedeći koraci:

1. Generiranje *hasha* – u ovom se koraku identificira *hashing* funkcija i izvlači *hash*.
2. Formatiranje *hasha* – *hash* se mora ispravno formatirati za korištenje u programu s kojim će se napadati.
3. Određivanje snage *hasha* (engl. *benchmark*) – ovo je bitan korak kojim se unaprijed određuje koliko je (otprilike) *hash* jak ili slab za probijanje. Drugim riječima, koliko otprilike *hasheva* u sekundi program može obraditi. Slijedi se pravilo da je više bolje. Odlične reference za to određivanje mogu se pronaći *online*.[6] Primjećuje se da je 7-zip *hash* poprilično jak i da nije jednostavan za probijanje. Ovisno o jačini *hasha*, prilagođava se i način napada.
4. Određivanje snage računala kojim će se napadati – ovo je direktno povezano s prethodnim korakom. Što je *hash* teži za probiti, i računalo mora biti snažnije.
5. Formuliranje plana napada – vrsta napada formulira se ovisno o tome znamo li primjerice dio lozinke ili neki od karaktera.
6. Napad na *hash* – u ovom se koraku provodi sam napad. Ovdje se koriste razne tehnike i domišljatost samog napadača da skrati vrijeme napada.
7. Prepravljanje (engl. *tweaks*) – dopunjavanje, ispravljanje i prepravljanje napadačkih tehnika ako se nije uspjelo iz prvog, drugog ili daljnjeg pokušaja.

[5] Izvor: https://www.amazon.com/Hash-Crack-Password-Cracking-Manual/dp/1975924584/. Datum 14.10.2019

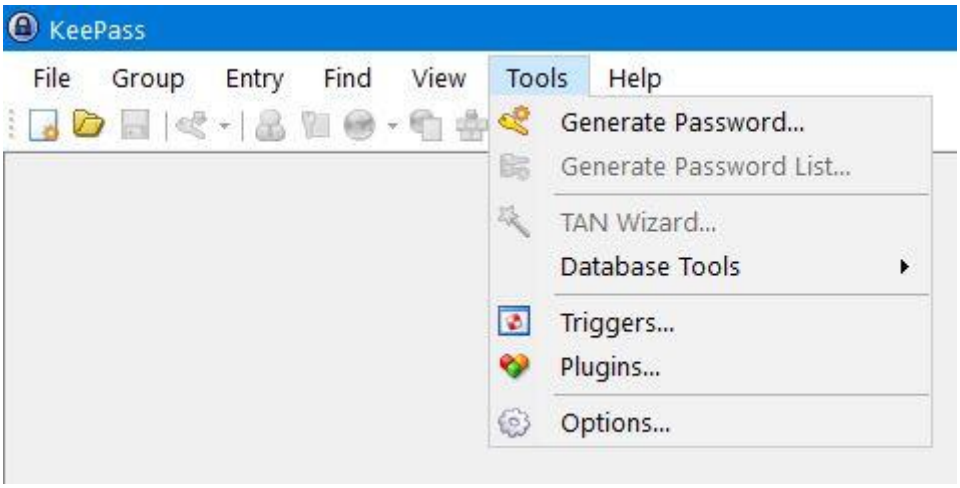
[6] Izvor: https://www.onlinehashcrack.com/tools-benchmark-hashcat-gtx-1080-ti-1070-ti.php Datum: 14. 10. 2019.

2.1.4. Upoznavanje softvera

U ovom dijelu ukratko će se opisati neki od softvera koji služe za probijanje lozinka. Navedeni softver koristit će se i u nastavku rada. Postoji mnogo programa koji mogu poslužiti u ovu svrhu, ali oni koji će u ovom radu biti korišteni su programi za generiranje lozinke, za generiranje *hasha* i za probijanje lozinke.

Generiranje lozinke

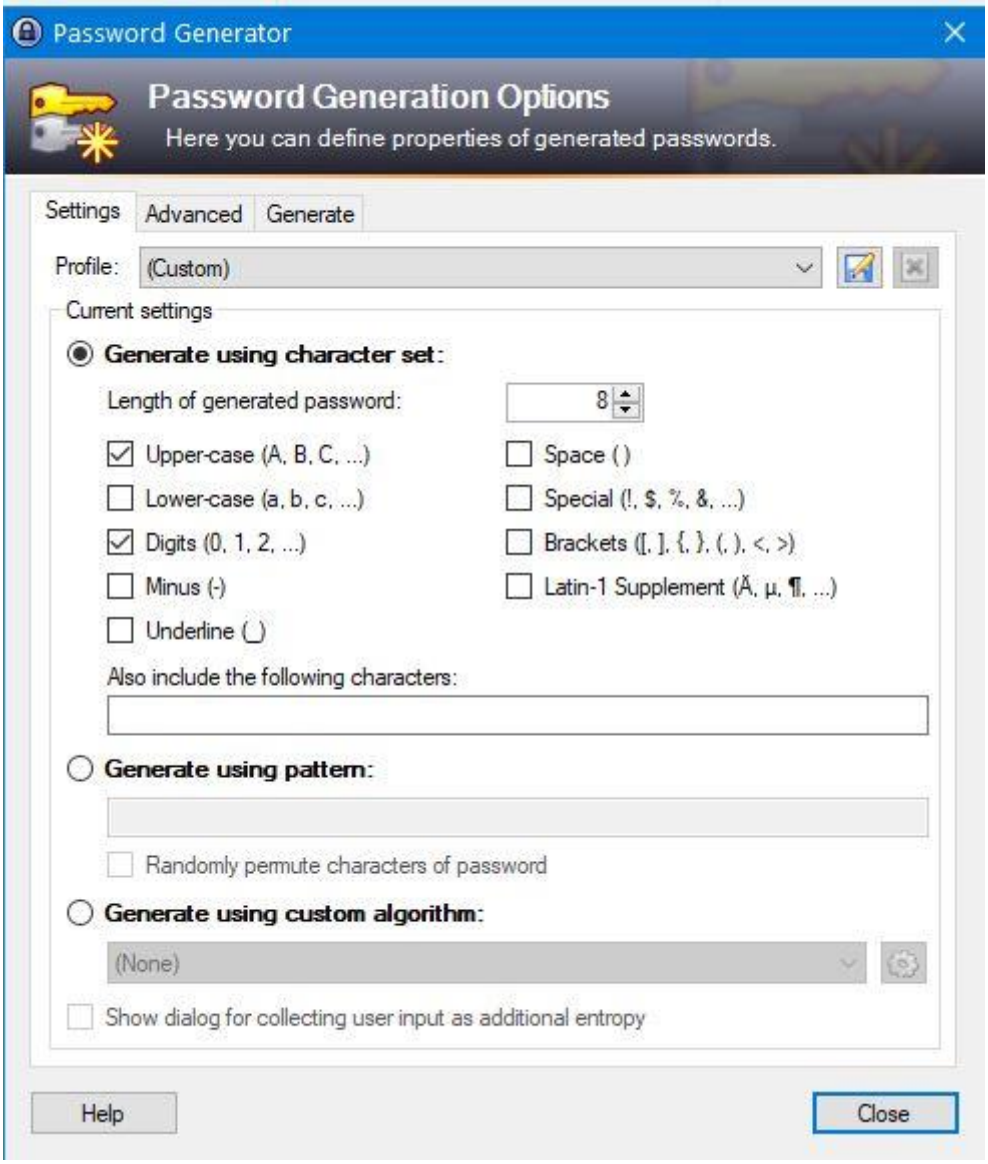
Kada se kaže generiranje lozinke, podrazumijeva se da se kreira neki nasumično odabrani niz znakova. To može uključivati slova (mala i velika), brojeve, posebne znakove itd. Broj tih znakova može se odrediti tako da se primjerice kaže da je generirana lozinka od deset znakova, uključujući samo velika i mala slova. Program koji će se za to koristiti je KeePass[7], a trenutna verzija je 2.43. KeePass je besplatan program otvorenog koda (engl. *open source*) koji služi za sigurno pohranjivanje lozinki. Također, unutar samog programa postoji potprogram koji služi za generiranje lozinki. Taj će program biti korišten u daljnjem radu. Nakon instalacije KeePass programa može se otvoriti opcija Generiranje lozinke (engl. *Generate Password*).



Slika 2.3. Opcija za generiranje lozinke unutar KeePass programa

[7] Izvor: https://keepass.info/.

Na sljedećoj slici prikazane su mogućnosti generiranja lozinke. Nakon što se postave opcije koje se žele koristiti, na kartici Generiraj (engl. *Generate*), automatski će biti generirano trideset različitih lozinki.

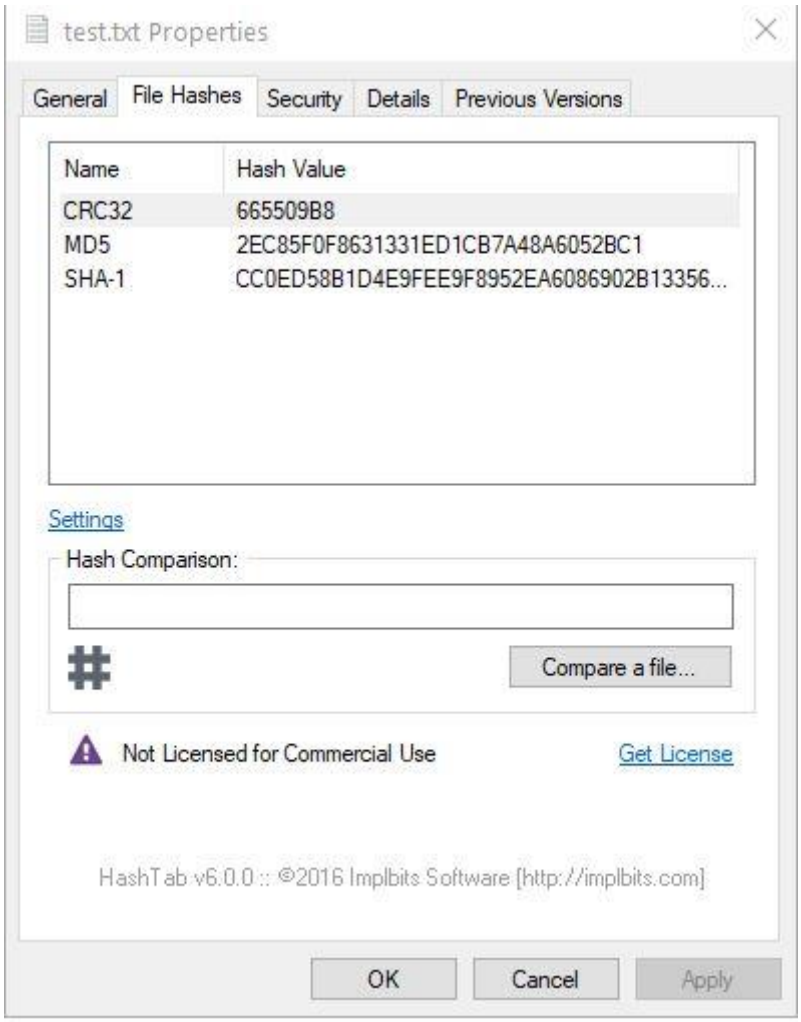


Slika 2.4. Opcije za generiranje lozinke

Postoji još ovakvih programa koji služe za generiranje lozinke. Dovoljno je u *Google* upisati „*random password generator*“ i dobiva se velik broj *online* aplikacija koje služe za generiranje lozinke.

Generiranje *hasha*

Za generiranje *hasha* datoteke može se koristiti velik broj aplikacija. Vidjelo se da i Windows operativni sustav ima ugrađeno putem *powershella* generiranje nekih od *hasheva*. Program koji će se koristiti u ovom radu zove se HashTab[8], a trenutna verzija je 6.0. Nakon instalacije HashTab se integrira u izbornik opcija prilikom klika desne tipke miša na neku datoteku. Može se odabrati opcija Mogućnosti (engl. *Properties*). Na kartici *hashevi* datoteka (engl. *File Hashes*) prikazani su odabrani *hashevi* te datoteke.

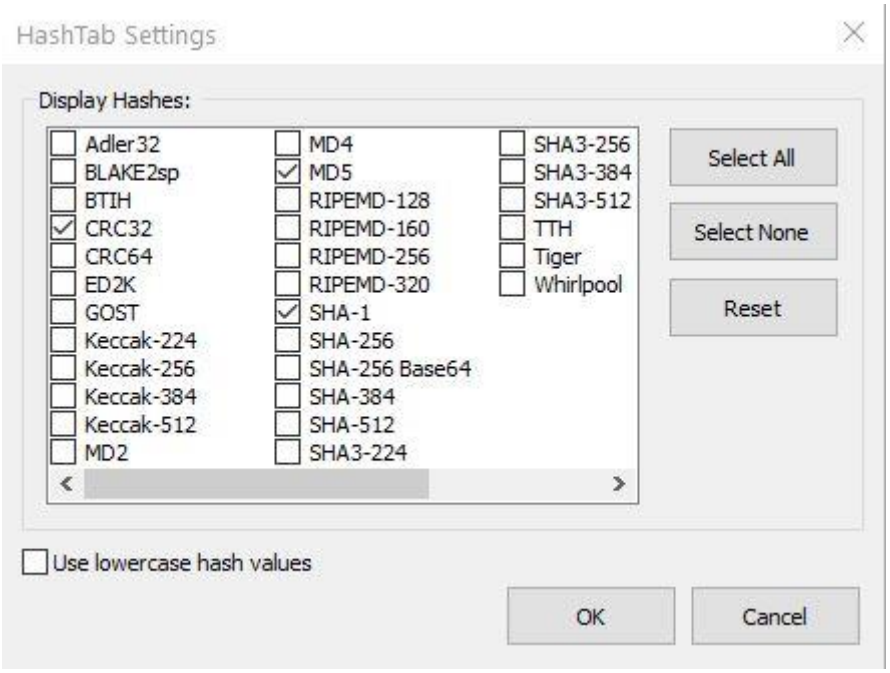


Slika 2.5. HashTab program

HashTab program će, ako to bude potrebno, biti korišten u ovom radu prilikom generiranja *hasheva* za određene datoteke.

[8] Izvor: http://implbits.com/products/hashtab/.

Klikom na Opcije (engl. *Settings*) mogu se odabrati drugi *hash* algoritmi.



Slika 2.6. HashTab opcije odabira algoritama

Probijanje lozinke

Za probijanje lozinke koristit će se program *hashcat*[9], koji će detaljno biti obrađen u trećem poglavlju. Jedan od programa koji je jednako dobar kao i *hashcat* zove se *John the Ripper*[10]. Oba programa su besplatna, otvorenog koda (engl. *open source*), dolaze standardno na Kali distribuciji Linuxa te se isto tako mogu skinuti s interneta i za druge operativne sustave. Verzija *hashcata* koja će biti korištena u ovom radu je 5.1.0 za Windows operativni sustav.

[9] Izvor: https://hashcat.net/hashcat/.
[10] Izvor: Patrick Engebretson; The Basics of Hacking and Penetration Testing; Elsevier Inc (2013).

2.2. Analiza probijanja različitih verzija *hasheva*

U ovom poglavlju objasnit će se tri popularna, tj. poznata *hash* algoritma te će se napraviti teorijska analiza kako bi se pristupilo probijanju tih *hasheva*.

2.2.1. MD5

MD5[11] *hash* funkcija je jako popularna i globalno korištena funkcija koja za svoj izlazni parametar vraća 128-bitni *hash*. Nažalost, za MD5 su pronađene velike sigurnosne rupe te kao takav nije prikladan za sigurna kriptografska rješenja. Jedan od uvjeta sigurne kriptografske funkcije jest velika matematička zahtjevnost dobivanja istog *hasha* za dvije različite ulazne informacije. Povezano s time, otkriveno je da je MD5 kompletno nesiguran[12] te se takve kolizije mogu bez problema pronaći na današnjim računalima. U današnje vrijeme MD5 uglavnom se koristi za nekriptografske radnje, npr. za usporedbu *hasha* neke datoteke. Unatoč svim ranjivostima, MD5 i dalje se koristi.

Kako bi se bolje predočio pristup probijanju MD5 lozinki, uzet će se za primjer već korištena tablica, ali ovoga puta s točnim *hash* vrijednostima MD5 algoritma. Za generiranje *hasha* ovih lozinki koristi se *online* MD5 generator[13], koji će isto tako biti korišten i u ovom radu.

Lozinka	*hash* funkcija	*hash*
ana	MD5	276b6c4692e78d4799c12ada515bc3e4
Ana	MD5	9aba45a7f1999a9c5fc96ef2a45810fb
Ana i Ivo	MD5	1795025bba142cdcf0adaa0462ae8d90
Šalica	MD5	1e7b01a5633e6088f69a4471015d8e31
Ana i Ivo piju kavu iz šalice	MD5	e5e3c9b69e45d54fb84faade5ad55265
Ana i Ivo piju kvau iz šalice	MD5	fbbba1e65854ed9ee8d11dacd6b548f0

Tablica 2.2. Primjer MD5 *hash* izlaznih vrijednosti

Fiktivna pretpostavka je da su lozinke u gornjoj tablici bile spremljene u nekoj bazi podataka. Naravno, ne direktno lozinke u običnom tekstualnom formatu, već njihovi *hashevi*.

[11] Izvor: https://en.wikipedia.org/wiki/MD5 Datum: 17. 10. 2019.
[12] Izvor: https://en.wikipedia.org/wiki/MD5CRK Datum: 17. 10. 2019.
[13] Izvor: https://www.md5hashgenerator.com/.

Zaposlenik u toj firmi koji radi na održavanju baze podataka odlučio je ukrasti i prodati *hasheve* na crnom tržištu. Osoba koja je došla u posjed tih *hasheva* želi iz njih dobiti lozinke. Po metodologiji probijanja lozinki, za napadača je prvi korak već napravljen – *hashevi* su generirani. Napadač stavlja *hasheve* u tekstualnu datoteku te s obzirom na to da zna da su to MD5 *hashevi*, odredio je da su jako slabi i da se mogu lako probiti. Napadač će koristiti svoje kućno računalo koje je izrazito snažno. Odlučio je za napad koristiti program *hashcat* te je prve četiri lozinke uspio probiti nakon samo nekoliko minuta. Nakon cijelog dana napadanja na preostale *hasheve*, uspio je pronaći sve lozinke.

Opisani fiktivni napad u stvarnosti je (nažalost) izrazito realan. Jedna od većih briga tvrtki u svijetu jest tko ima pristup zaštićenim podacima. Ulaže se veliki trud u profiliranje i edukaciju zaposlenika[14] (tko su, koliki pristup informacijama imaju, što smiju dijeliti itd.) koji rade na pozicijama s pristupom bazi povjerljivih podataka.

2.2.2. SHA-2

Hash funkciju može se zamisliti kao enkripciju u jednom smjeru. Kao takva prikladna je za sve vrste digitalnih potpisa. S obzirom na to da nisu sve *hash* funkcije jednako sigurne (kao primjerice MD5 i SHA-1), jedna od *hash* funkcija koja se u svijetu koristi za digitalne potpise je i SHA, tj. Sigurni *Hashing* Algoritam (engl. *Secure Hashing Algorithm*). SHA je prošao nekoliko iteracija. Danas je SHA-2 postao standard koji koriste i velike tvrtke[15] kao što su Google i Microsoft na svojim *web*-stranicama. Certifikati izdani od strane izdavača certifikata, kao što su Comodo ili Digicert, koriste SHA-2. SHA-2 je zapravo grupa *hash* algoritama s različitim duljinama, od kojih je najpopularniji SHA-256.

Primjer probijanja lozinke za SHA-2 je isti kao i MD5 primjer, dakle napada se *hash* neke lozinke. Ako se pogleda *benchmark* za SHA-2 algoritam (treći korak u metodologiji probijanja lozinki), može se primijetiti da osim što SHA-2 nema ranjivosti kao MD5, desetak puta ga je teže probiti od MD5 algoritma. Konkretno, prosječna brzina probijanja MD5

[14] Izvor: https://www.hiscox.co.uk/business-blog/the-importance-of-confidentiality-in-the-workplace/ Datum: 15. 12. 2019.
[15] Izvor: https://cheapsslsecurity.com/blog/what-is-sha2-and-what-are-sha-2-ssl-certificates/ Datum: 17. 10. 2019.

lozinki, koristeći GTX2080 GPU, je oko 25 000 MH/s (*megahasheva* po sekundi), dok za SHA-2 ta brojka iznosi otprilike 2800 MH/s.[16]

2.2.3. CRC32

CRC32 funkcija je kôd koji se koristi za detekciju pogrešaka[17], najčešće prilikom spremanja podataka u sustavima za pohranu. CRC prvi se put pojavio još davne 1961. godine, a CRC32 prvi put je izdan 1975. godine. S obzirom na to da CRC32 funkcija kao izlaznu vrijednost vraća uvijek 8-bitni niz znakova, može se koristiti kao da je *hash* funkcija. CRC32 sam po sebi nije napravljen za nikakvu sigurnost te bi lozinke „*hashirane*“ s CRC32 bile izuzetno lako probijene. To se prvenstveno odnosi na to da nema mehanizma otpornosti na koliziju. Iako se ne zna ulazna vrijednost funkcije, lako se može pronaći neka druga ulazna vrijednost koja daje istu izlaznu vrijednost. Jedan od temeljnih zahtjeva sigurne *hash* funkcije ovdje ne postoji jer CRC32 nije ni napravljen da bude *hash* funkcija, već služi za detekciju pogrešaka.

Kada bi se analiziralo probijanje lozinke „zaštićene“ CRC32 funkcijom, glavni cilj bio bi pronaći bilo koju ulaznu vrijednost ili informaciju koja daje isti izlaz kojim je ta lozinka zaštićena. I *hashcat* i *John the Ripper* programi mogu bez problema to napraviti.

2.3. *Hash* 7-zip datoteke

U Poglavlju 2.1.4. nije spomenut poseban program koji služi za generiranje *hasha* 7-zip datoteke. Taj se program zove 7z2hashcat[18]. I on je program otvorenog koda koji se može besplatno skinuti s interneta. Trenutna verzija je 1.3. Ovaj program služi za dobivanje *hasha* 7-zip datoteke, zakriptirane nekom lozinkom, na jednostavan način. U Poglavlju 2.3.2. bit će prikazan prvi praktični primjer – u ovom slučaju kako se dobiva *hash* 7-zip datoteke, koji se poslije može napadati *hashcat* programom.

---

[16] Izvor: Joshua Picolet; Hash Crack: Password Cracking Manual; CreateSpace Independent Publishing Platform; 2nd edition (2017).
[17] Izvor: https://en.wikipedia.org/wiki/Cyclic_redundancy_check Datum: 17. 10. 2019.
[18] Izvor: https://github.com/philsmd/7z2hashcat Datum: 19. 10. 2019.

2.3.1. Kako generirati dobru lozinku

Još jedan teorijski dio koji se mora spomenuti u ovom radu jest kako generirati dobru lozinku. Što uopće znači dobra lozinka? Lozinke se mogu podijeliti u dvije skupine: lozinka kao jedna riječ, tj. skup alfanumeričkih znakova, i lozinka kao skup znakova koji može sadržavati razmake, cijele rečenice itd. Engleske riječi za to su *password* i *passphrase*, koje u hrvatskom jeziku nemaju neku smislenu razliku. Obje se riječi prevode kao lozinka. U daljnjem tekstu upotrebljavat će se riječ „lozinka“ za *password* i riječ „fraza“ za *passphrase*, kako bi se smislenije podijelile te dvije skupine.

Primjer lozinke: „**P@ssw0rd1!**“ (bez navodnika).

Primjer fraze: „**Sunce J3 Veliko i Sjajno 123#!**“ (bez navodnika).

Kao što se može vidjeti, fraza se sastoji od više riječi. Također, fraze mogu biti smislene osobi koja ih koristi i imati neki značaj po kojem će ih ta osoba lako pamtiti. To mogu biti npr. stihovi najdraže pjesme ili nešto slično. Ako se ubace različiti znakovi u takvu frazu, dobije se poprilično dugačka i izuzetno teška fraza za probiti. Dakle, glavna razlika između lozinke i fraze jest da fraze imaju razmak u sebi i uvijek su duže od obične lozinke.[19]

Nepisano pravilo „dobrih lozinki“ jest da se uvijek koristi fraza umjesto lozinke s minimalno deset znakova. Čak i najnapredniji programi za probijanje lozinki ne mogu se, laički rečeno, snaći, niti znaju kako bi uopće krenuli probijati dugačku frazu, niti imaju efikasnu tehniku napada protiv takvih lozinki. Naravno, to vrijedi pod uvjetom da se koristi neka od snažnijih metoda kriptiranja. Neovisno o tome koliko je velika i kompliciranа fraza, ako je „zaštićena“ CRC32 metodom, i dalje se može probiti na različite načine.

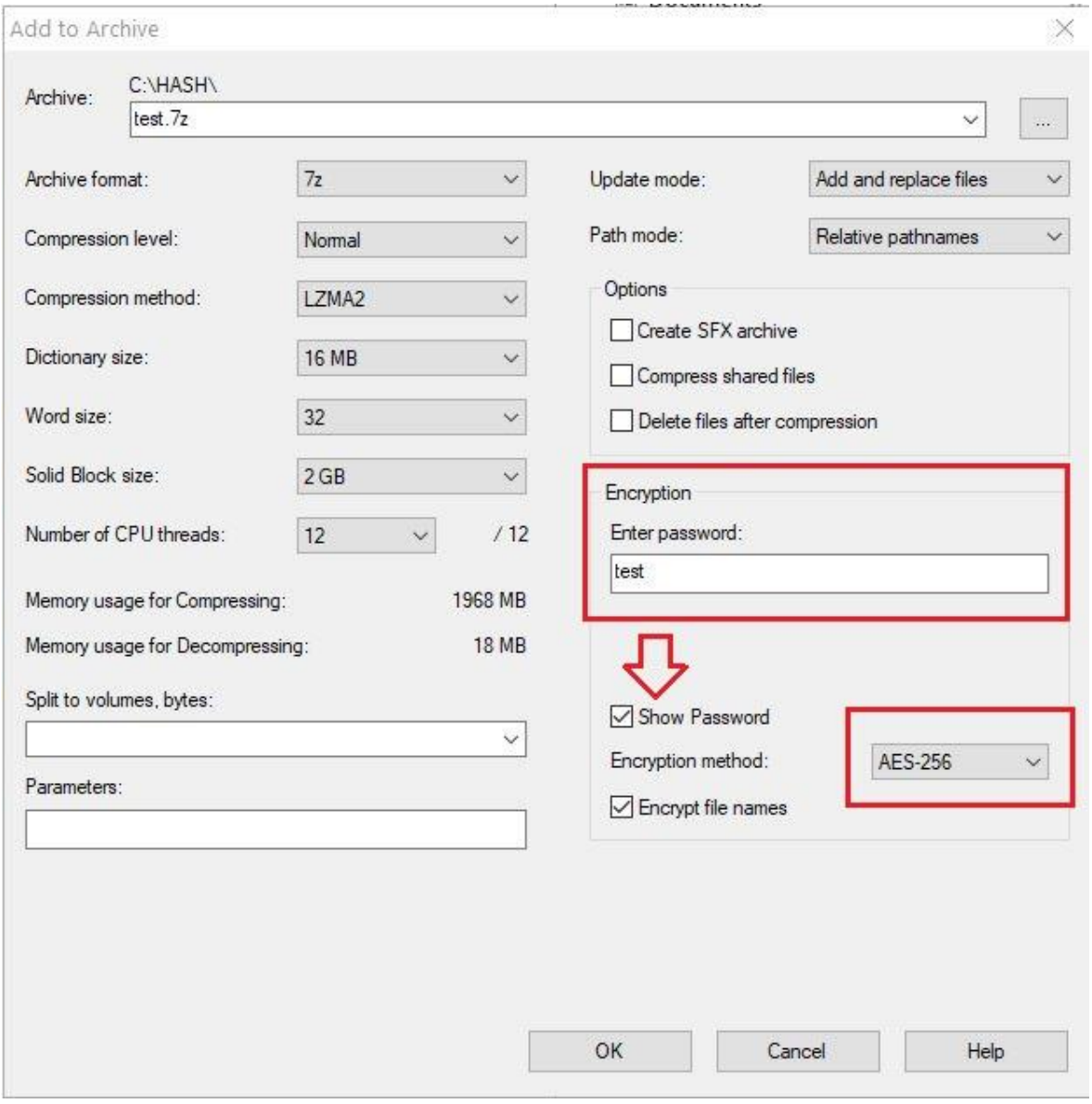
7-zip aplikacija ima u sebi ugrađenu AES-256 enkripciju koja je dovoljna za zaštitu tajnih američkih dokumenata[20] te ako se kombinira s jakom lozinkom (ili frazom), praktički je neprobojna.

[19] Izvor: https://www.passworddragon.com/password-vs-passphrase Datum: 15. 12. 2019.
[20] Izvor: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#Security Datum: 19. 10. 2019.

2.3.2. Kako generirati 7-zip *hash*

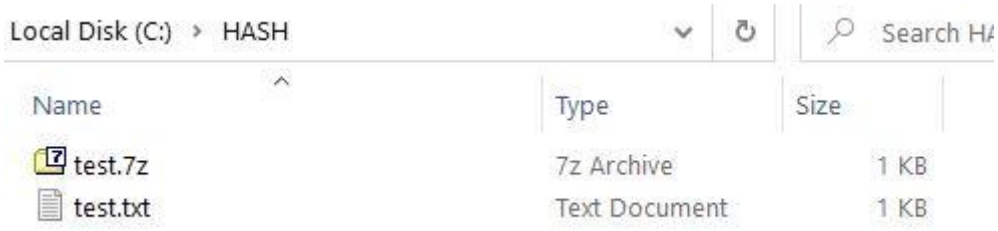
Pojednostavljen način pristupa probijanju 7-zip lozinke, uz pomoć metodologije probijanja lozinki, svodi se na dobivanje *hasha* zakriptirane 7-zip datoteke. Taj se *hash* napada raznim tehnikama koristeći *hashcat* program. U ovom poglavlju bit će prikazano kako se može dobiti *hash* zakriptirane 7-zip datoteke.

Iskoristit će se test.txt datoteka koja je korištena prije u tekstu. Datoteku ćemo komprimirati programom 7-zip koristeći AES-256 enkripciju. Lozinka koja će biti korištena je riječ „test“, pisana malim slovima.



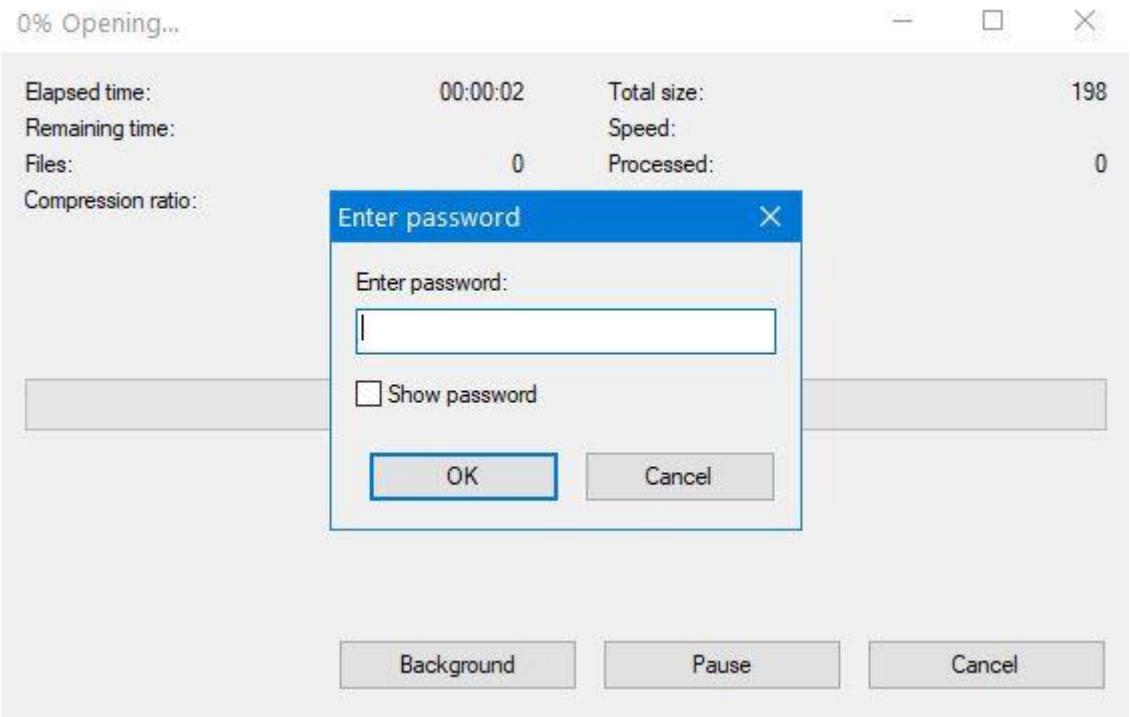
Slika 2.7. Komprimiranje i enkripcija test.txt datoteke

Time se dobije test.7z file kao na donjoj slici, koji u sebi sadrži test.txt datoteku.



Slika 2.8. Prikaz novokreirane test.7z datoteke

Prilikom otvaranja datoteke dobije se sljedeća poruka.



Slika 2.9. Zakriptirana 7-zip datoteka ne može se otvoriti bez lozinke

Pod uvjetom da napadač ne zna lozinku, mogao bi pokušavati razne kombinacije. S obzirom na to da je odabrana lozinka u ovom slučaju riječ „test“, postoji mogućnost da bi se mogla i pogoditi. Ali, pretpostavimo da je napadač odustao od nasumičnog pogađanja i da je odlučio „napasti“ datoteku boljim i inteligentnijim rješenjem. Prvo što mu treba je *hash* navedene 7-zip datoteke. Prije korišten program, HashTab, nema u sebi opciju da izvuče *hash* ove datoteke. U tu svrhu koristi se program 7z2hashcat – u ovom primjeru to je 64-bitna verzija

tog programa. Iz podatkovnog retka (engl. *command prompt*) pokreće se sljedeća naredba, nakon što se sam program iskopira u istu mapu:

```
7z2hashcat64-1.3.exe test.7z
```

To izgleda kao što je prikazano na donjoj slici.



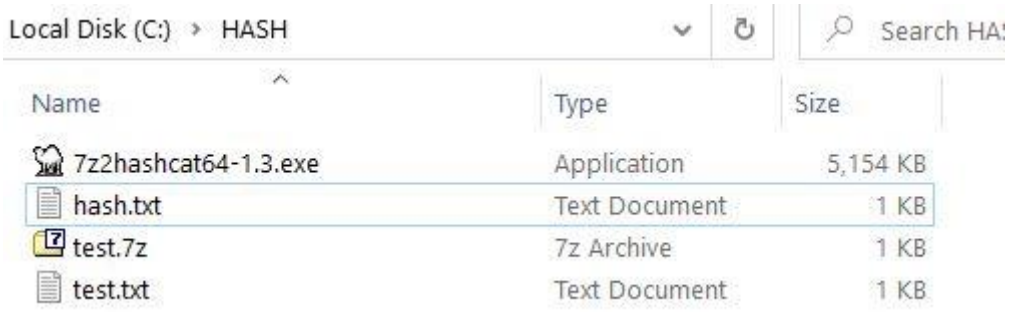
Slika 2.10. Prikaz naredbe za dobivanje *hasha* 7-zip datoteke

Prepravimo naredbu da se *hash* spremi u datoteku:

```
7z2hashcat64-1.3.exe test.7z > hash.txt
```

Time se dobije *hash* ove 7-zip datoteke koji je spremljen u hash.txt datoteku. Tu istu datoteku koristit ćemo u primjeru probijanja lozinke putem napada sirovom snagom, što će biti prikazano u jednom od narednih poglavlja.

Konačni izgled trenutne mape izgleda ovako.



Slika 2.11. Prikaz datoteka korištenih za dobivanje *hasha*

Hash se nalazi u datoteci hash.txt i njegov sadržaj je sljedeći:

$7z$0$19$0$$8$ac3814bb94d1aa9a0000000000000000$1945089461$112$106$b65b960fd17442c726cec618df0e5b77b9a1b44731363bed4db72469a98c4cfb1ce33e487624f39727bf8026039800521c1ef96999542ee377e52221eb1e8a77fad017f6efb7b4d05ef59f16bf9a056428cc5438de17e69b55ab96225062837e3b572f0bd72ac561707ed0dac0f1b528

Sam izlazni zapis ovog *hasha* je jedna linija, a sastoji se od jedanaest polja odvojenih znakom „$“. Odvojena polja i njihova značenja[21] objašnjena su kako slijedi:

$7z označava vrstu *hasha*; u ovom slučaju to je 7-zip *hash*

$0 indikator vrste podataka – ovaj broj može biti 0 – 255 te govori je li potrebno dekomprimirati podatak da bi se dobio CRC32 kontrolni broj. Broj nula označava da nije potrebno dekomprimirati.

$19 faktor učinkovitosti – govori koliko se iteracija mora izvršiti za ovaj *hash*, formula je 2^x, a u ovom slučaju to je 2^{19}

$0 duljina sljedećeg polja – soljenje (engl. *salt*)[22]. Soljenje je način kojim se još bolje može zaštititi lozinka tako da se algoritmu sažimanja doda nasumično odabrani podatak. U ovom slučaju soljenje nije korišteno i zato je duljina tog polja nula.

$ heksadecimalni zapis soljenja. S obzirom na to da nije korišteno, ovo je polje prazno.

$8 duljina inicijalizacijskog vektora – to je proizvoljni broj koji može biti korišten prilikom enkripcije podataka.[23]

$ac3814bb94d1aa9a0000000000000000 heksadecimalni zapis inicijalizacijskog vektora

$1945089461 CRC kontrolni broj zapisan u decimalnom formatu

$112 duljina zakriptiranog podatka

$106 duljina dekriptiranog podatka.

Na kraju sam zakriptirani podatak:

$b65b960fd17442c726cec618df0e5b77b9a1b44731363bed4db72469a98c4cfb1ce33e4876
24f39727bf8026039800521c1ef96999542ee377e52221eb1e8a77fad017f6efb7b4d05ef59f1
6bf9a056428cc5438de17e69b55ab96225062837e3b572f0bd72ac561707ed0dac0f1b528

[21] Izvor: https://github.com/philsmd/7z2hashcat#explanation-of-the-hash-format Datum: 15. 12. 2019.
[22] Izvor: https://en.wikipedia.org/wiki/Salt_(cryptography) Datum: 15. 12. 2019.
[23] Izvor: https://whatis.techtarget.com/definition/initialization-vector-IV Datum: 15. 12. 2019.

3. *Hashcat*

Iako je već ukratko spomenut *hashcat*, u ovom će dijelu biti detaljnije objašnjen i prikazat će se neke od njegovih (mnogobrojnih) mogućnosti.

3.1. Što je *hashcat* i čemu služi?

Dakle, što je točno *hashcat*? U jednoj rečenici – *hashcat* je program koji služi za probijanje lozinki. Ako se pogleda na službenoj stranici *hashcata*[24], jedan od opisa (na engleskom) kaže: „*World's fastest password cracker*.“ Prevedeno, to je najbrži program za probijanje lozinki na svijetu. Opis najbrži ovisi o nekoliko faktora, prvenstveno o duljini i o kompleksnosti lozinke (šifre) te o snazi računala kojim se pokušava probiti ta lozinka.

Hashcat je besplatan program otvorenog koda s MIT licencom[25], podržava multiplatforme operativnih sustava (Windows, Linux, MacOS) te cijelu paletu raznih opcija. Koristi se u naredbenom retku (engl. *command prompt*) na Windows operativnom sustavu, tj. u terminalu (Linux OS). Trenutna inačica je verzija 5.1.0 i dolazi u dvije varijante, 64-bitna i 32-bitna. U radu će se koristiti isključivo 64-bitna verzija programa, a izvršna datoteka koja će se pozivati zove se hashcat64.exe (Windows okruženje).

[24] Izvor: https://hashcat.net/hashcat/ Datum: 20. 10. 2019.
[25] Izvor: https://opensource.org/licenses/MIT Datum: 20. 10. 2019.

3.1.1. Instalacija *hashcat* programa

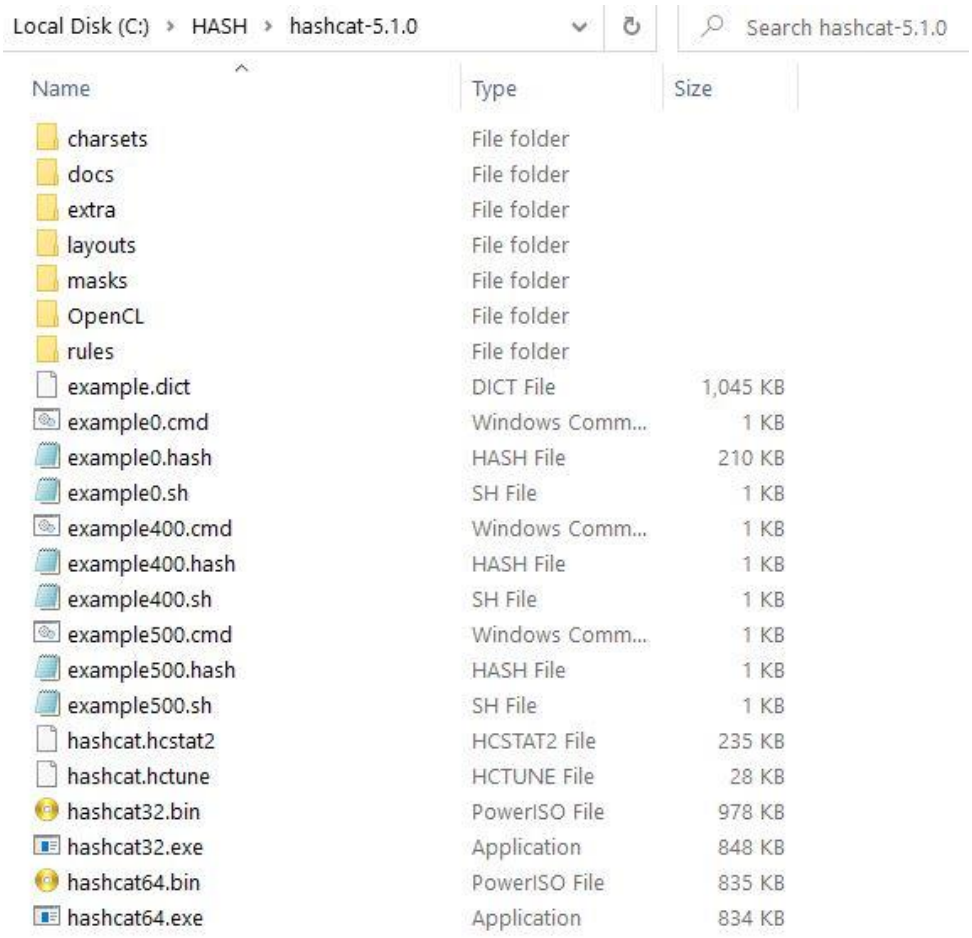
Prvo što je potrebno napraviti na početnoj stranici je *download* programa. Verzija koja će se koristiti prikazana je na donjoj slici. Dovoljno je kliknuti na *download*, preuzeti i raspakirati hashcat-5.1.0.7z datoteku u neku proizvoljnu mapu.

Download

Name	Version	Date	Download	Signature
hashcat binaries	v5.1.0	2018.12.02	Download	PGP
hashcat sources	v5.1.0	2018.12.02	Download	PGP

Slika 3.1. Preuzimanje *hashcat* programa sa službene stranice

Nije potrebno ništa instalirati, sve je spremno nakon raspakiravanja. Struktura mape nakon raspakiravanja *hashcat* programa izgleda ovako.



Slika 3.2. Struktura *hashcat* mape

Datoteka koja će se koristiti je hashcat64.exe. S obzirom na to da će prilikom korištenja naredbi biti potrebno referencirati razne putanje, dodat će se „hashcat-5.1.0“ mapa u sistemsku Path varijablu okruženja (engl. *Environment Variables*). Time će se moći pozivati hashcat64.exe izvršna datoteka s bilo kojeg mjesta na disku.

3.1.2. *Hashcat* naredbe

Hashcat je izuzetno moćan program i ima pregršt mogućnosti i opcija. U bilo kojem trenutku može se pozvati naredba pomoći (engl. *help*) koja će prikazati sve opcije programa. Ako se unese sljedeća naredba, bit će prikazane sve opcije koje se mogu koristiti:

```
hashcat64 --help | more
```

```
C:\HASH>hashcat64 --help | more
hashcat - advanced password recovery

Usage: hashcat [options]... hash|hashfile|hccapxfile [dictionary|mask|directory]...

- [ Options ] -

 Options Short / Long          | Type | Description                                          | Example
===============================+======+======================================================+=======================
 -m, --hash-type               | Num  | Hash-type, see references below                      | -m 1000
 -a, --attack-mode             | Num  | Attack-mode, see references below                    | -a 3
 -V, --version                 |      | Print version                                        |
 -h, --help                    |      | Print help                                           |
     --quiet                   |      | Suppress output                                      |
     --hex-charset             |      | Assume charset is given in hex                       |
     --hex-salt                |      | Assume salt is given in hex                          |
     --hex-wordlist            |      | Assume words in wordlist are given in hex            |
     --force                   |      | Ignore warnings                                      |
     --status                  |      | Enable automatic update of the status screen         |
```

Slika 3.3. Prikaz nekih od početnih opcija *hashcat* `--help` naredbe

Help ima nekoliko stranica. Zbog uštede prostora, ovdje je prikazano samo prvih desetak redaka. Isto tako, na službenoj stranici postoji odličan *wiki*[26] kojem se uvijek može pristupiti te pregledati iste naredbe i mogućnosti koje su prikazane naredbom `--help`.

[26] Izvor: https://hashcat.net/wiki/doku.php?id=hashcat Datum: 20. 10. 2019.

Neke od opcija koje su od bitnog značaja i koje će se koristiti u ovom radu su:

-m, --hash-type – koristi se za određivanje *hasha*. Za 7-zip koristi se *hash-type* 11600.

-a, --attack-mode – postoji više vrsta napada, bit će pomnije objašnjeno u sljedećem poglavlju.

-o, --outfile – naziv datoteke u koju će se spremati izlazna informacija, npr. pronađena lozinka.

-r, --rules-file – pravila koja se koriste prilikom napada korištenjem pravila.

No, prije svega, potrebno je opisati i objasniti sintaksu naredbi unutar *hashcat* programa. Prvo što se mora pokrenuti je sama naredba, dakle hashcat64.exe. Nakon toga idu opcije koje se ne moraju pisati po određenom redoslijedu. Postoje ulazna i izlazna datoteka (engl. *input file, output file*). Ulazna datoteka je datoteka koja sadrži *hash* ili više *hasheva*. Izlazna datoteka je ona u koju se može spremiti pronađena lozinka ili neki drugi izlazni podatak (ovisno o opcijama).

Prikazan je primjer *hashcat* naredbe:

```
hashcat64.exe -m 11600 -a 3 hash.txt -o password_found.txt ?l?l?l?l
```

„Prevedeno“, ova bi naredba značila sljedeće:

`hashcat64.exe` – pokreni *hashcat* 64-bitnu verziju

`-m 11600` – označava da *hash* koji se pokušava probiti pripada 7-zip *hashu*

`-a 3` – koristi napad sirovom snagom (engl. *brute force attack*)

`hash.txt` – u ovoj se datoteci nalazi *hash* (ili *hashevi*) koji se pokušavaju probiti

`-o password_found.txt` – u ovu datoteku spremi lozinku kao izlaznu informaciju, ako bude pronađena

`?l?l?l?l` – koristi masku (malo slovo L) koja označava sva mala slova (engl. *lower case*) te koristi točno četiri mala slova.

U Poglavlju 3.2. bit će detaljnije prikazani primjeri raznih vrsta napada.

3.1.3. Duljina ključa, prostor ključa

Da bi se pobliže objasnili napadi pomoću *hashcata* (i općenito napadi probijanja lozinki), potrebno je uvesti još dva pojma – duljina ključa (engl. *Key Length*)[27] i prostor ključa (engl. *Key Space*)[28]. Duljina ključa odnosi se na broj znakova (engl. *character set*) koje lozinka ima. Dakle, što je veći broj znakova u lozinci, to ju je teže probiti. To je u teoriji točno, ali u praksi samo djelomično točno jer ovisi i o tome koliko je velik prostor ključa. Prostor ključa je broj svih kombinacija (permutacija) koje se mogu sastaviti od duljine ključa zajedno sa svim znakovima. Bit će puno razumljivije na primjeru.

Kreditna kartica ima svoj PIN, sastavljen od četiri znamenke. Svaka od tih znamenki može biti broj 0 – 9, npr. PIN je broj 4567. Duljina ključa ovog PIN-a je četiri jer se taj PIN sastoji od četiri znakova (u ovom slučaju brojeva).

0000 – prvi broj PIN-a

0001 – drugi broj PIN-a

0002 – treći broj PIN-a

....

9999 – zadnji broj PIN-a.

Prostor ključa dobije se iz svih kombinacija. Svaki broj PIN-a može biti jedna od deset znamenki (brojevi 0 – 9). Dakle, ukupan broj kombinacija je 10^4 = 10 000 različitih kombinacija za ovaj PIN. Kaže se da je prostor ključa za PIN od četiri znamenke velik 10 000.

Drugi primjer je lozinka duljine sedam znakova, sastavljena od malih i velikih slova engleske abecede i brojeva. Duljina tog ključa je sedam. A prostor ključa? Svaki od tih sedam znakova može biti jedno od 26 slova engleske abecede. Također, može biti veliko ili malo slovo i neki broj 0 – 9. Ukupan broj kombinacija za samo jedan znak je 26 + 26 + 10 = 62. Za svih sedam znakova ukupan broj kombinacija je 62^7 = 3,521,614,606,208 različitih kombinacija. To bi trebao biti prostor ključa, no *hashcat* ima svoj algoritam kako izračunava

[27] Izvor: https://en.wikipedia.org/wiki/Key_size Datum: 15. 11. 2019.
[28] Izvor: https://en.wikipedia.org/wiki/Key_space_(cryptography) Datum: 15. 11. 2019.

svoj prostor ključa. Tom algoritmu treba vjerovati, umjesto da se „ručno“ računa. Naredba kojom se može vidjeti prostor ključa je `--keyspace`.

Ovako bi izgledala naredba za izračun prostora ključa u gornjem primjeru, gdje se koriste mala i velika slova i brojevi:

```
hashcat64 -a 3 --keyspace -1 ?l?u?d ?1?1?1?1?1?1?1
```

`-1 ?l?u?d` – ovime se označava set znakova broj 1 (engl. *character set*) koji se dalje koristi u maski

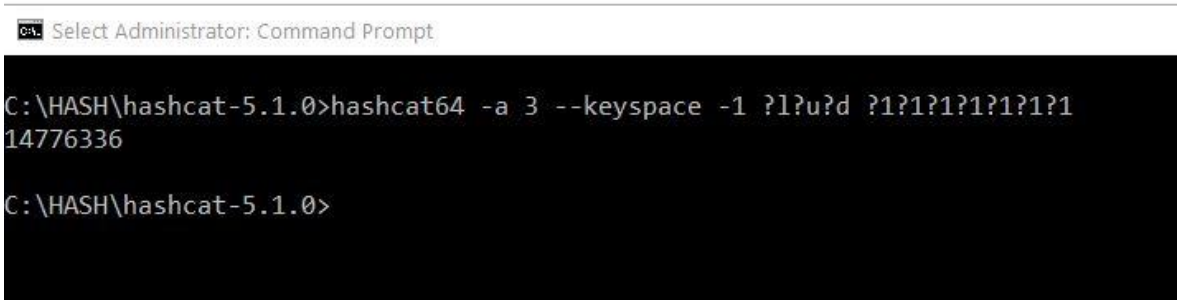
`?l` = sva mala slova (engl. *lowercase*)

`?u` = sva velika slova (engl. *uppercase*)

`?d` = brojevi 0 – 9 (engl. *digits*)

`?1?1?1?1?1?1?1` – za svih sedam znakova koristi se jedan od znakova iz seta znakova broj 1.

Na donjoj slici vidljiv je rezultat gornje naredbe.



Slika 3.4. Izračunavanje prostora ključa

Dobije se prostor ključa 14,776,336 (62^4), što nije isto kada bi se „ručno“ računalo za svih sedam znakova. To je zato što je *hashcat* naredba `--keyspace` specifično napravljena za optimiziranu distribuciju posla prilikom probijanja lozinke.[29]

Ako se koristi više uređaja za probijanje lozinki, da bi se odredilo koji će kandidati za lozinke biti generirani na grafičkom procesoru, *hashcat* prati neke od kandidata na glavnom

[29] Izvor: https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#what_is_a_keyspace Datum: 16. 11. 2019.

računalu. Da bi to ostvario, koriste se dvije petlje: jedna koja se vrti na glavnom računalu, tzv. osnovna petlja (engl. *base loop*) i mod petlja, tj. petlja koja se vrti na samom uređaju (engl. *modifier loop*). Da bi radio između više komputacijskih čvorova, *hashcat* mora podijeliti i distribuirati dijelove osnovne petlje. Tu se koristi `--keyspace` opcija. Ona govori kolika je veličina osnovne petlje koja se izvršava na glavnom računalu tako da se može ispravno podijeliti posao.

`--keyspace` opcija izračunava se prema sljedećim formulama:

-a 0 – broj riječi u listi riječi

-a 0 + pravila – broj riječi u listi riječi

-a 1 – broj riječi u većoj listi riječi (ako se koriste dvije liste riječi)

-a 3 – pomnoži se broj mogućih znakova u svakoj poziciji maske za osnovnu petlju, isključujući mod petlju

-a 6 – broj riječi u listi riječi

-a 7 – broj riječi u listi riječi.

Razumijevanje prostora ključa kod probijanja lozinki pomaže boljem određivanju vrsta napada. Vrsta napada koristiti se u skladu s veličinom prostora ključa. Veći prostor zahtijeva bolju, inteligentniju i kompleksniju vrstu napada ako se želi uspješno probiti neka lozinka.

3.2. Vrste napada pomoću *hashcata*

Kako se sve može pristupiti napadu, tj. pokušaju probijanja neke lozinke? Ovisno o odgovoru, određuje se i vrsta napada. *Hashcat* razlikuje nekoliko vrsta napada, od kojih se najčešće koriste napadi rječnicima, napadi sirovom snagom/maskom i napadi korištenjem pravila. Svaki od ovih napada ima određene prednosti i mane, ali važno je znati da vrsta napada koji se koristi uvelike ovisi o tome koliko su poznate bilo kakve informacije o lozinki koju želimo probiti. Primjerice, ako je poznato da lozinka sadrži isključivo velika slova i brojeve i da nije veća od pet znakova, može se zaključiti da bi prikladna vrsta napada na takvu lozinku bila napad korištenjem maske, tj. napad sirovom snagom. S druge strane, ako nije poznato apsolutno ništa o nekoj lozinki, napad sirovom snagom je potencijalno izrazito zahtjevan i dug proces (prostor ključa može biti izuzetno velik). Ali, napad rječnikom bio bi (potencijalno) učinkovitiji u ovom slučaju, s obzirom na to da *hashcat* može jako brzo proći kroz sve pojmove u nekom rječniku. Sve navedeno, s primjerima, bit će objašnjeno u narednim poglavljima.

3.2.1. Napad sirovom snagom (engl. *Brute-force Attack*)

Napad sirovom snagom je najjednostavniji, ali ujedno i najsporiji napad. Napad sirovom snagom može se jednostavno definirati kao pokušaj svih mogućih kombinacija u nekom zadanom prostoru ključa. Mora se odrediti set znakova koji će se koristiti (npr. velika i mala slova i brojevi) te kolika je duljina lozinke. Potom se pokušavaju sve moguće kombinacije da bi se pronašao *hash* koji odgovara tim znakovima. Napad sirovom snagom se „samo pokrene“ (prije toga se odredi prostor ključa i duljina ključa). Ako je *hash* koji se probija neki od sporih *hasheva* (dakle, potrebna je velika komputacija i procesorska snaga), ovakva vrsta napada je jako neučinkovita. Jedan od takvih sporih *hasheva* je upravo i *hash* 7-zip datoteke.

Danas je napad sirovom snagom zastarjela vrsta napada i u potpunosti je zamijenjen napadom korištenjem maske. Konkretni primjer napada sirovom snagom bit će prikazan u sljedećem poglavlju, 3.2.2., gdje će se koristiti maska prilikom napada sirovom snagom.

3.2.2. Napad korištenjem maske (engl. *Mask Attack*)

Riječ „maska“ u ovom se slučaju odnosi na to da „maskiramo“ neki od znakova tako da unaprijed odredimo ili koji je to točno znak ili da odredimo da je to veliko slovo, broj i slično. Za napad korištenjem maske tipično je korištenje opcije `-a 3` koja određuje da će biti korišten napad sirovom snagom.

Pretpostavimo da je lozinka koju pokušavamo probiti riječ: „Lozinka123“ (bez navodnika). Ona se sastoji od velikih i malih slova i od brojeva. Mogle bi se prilikom pokretanja *hashcata* koristiti opcije ?l za sva mala slova, ?u za sva velika slova i ?d za sve brojeve. Lozinka se sastoji od deset znakova, dakle duljina ključa je deset. Ali, mi, kao napadač u ovom slučaju, to ne znamo. Jedino što znamo unaprijed (premisa) je da je prvi znak u toj lozinki veliko slovo.

Kada bi se računao prostor ključa bez te premise, on bi iznosio 62^{10} = 839,299,365,868,340,224 različitih kombinacija. Ako uključimo u računanje prostora ključa pretpostavku da prva znamenka može biti isključivo veliko slovo, tada dobijemo: $26 * 62^9$ = 351,964,250,202,852,352. Vidljivo je da je drugi broj osjetno manji od prvog.

Pretpostavimo dalje da *hashcat* može računati milijun različitih ključeva u sekundi. Koliko vremena bi bilo potrebno da se izvrte sve kombinacije u prvom slučaju, a koliko u drugom? Izračun je jednostavan: ukupni broj kombinacija podijeljen s brojem ključeva u sekundi = broj sekundi potreban da se probije lozinka (preračunat u veću jedinicu vremena).

U prvom bi slučaju bilo potrebno 26,596 godina, dok bi u drugom slučaju bilo potrebno 11,153 godine.[30] Ovime se može dobiti osjećaj koliko je teško probijati lozinke koristeći napade sirovom snagom. Dok ljudi ne izume brže metode računanja (npr. kvantna računala, AI i slično), ovakve će lozinke za sve normalne slučajeve biti nemoguće probiti.

Ali, u drugom smo slučaju korištenjem maske smanjili potrebno vrijeme za probijanje lozinke. Pokušajmo iskoristiti masku još jednom i pretpostavimo (npr. saznali smo na neki način) da su zadnja tri znaka u lozinki brojevi. Izračun prostora ključa za ovaj, treći slučaj, bio bi: $26 * 62 * 62 * 62 * 62 * 62 * 62 * 10 * 10 * 10$ = 1,476,806,125,184,000 različitih kombinacija. Kada se to preračuna u potrebno vrijeme za probijanje te lozinke, i dalje pod

[30] Izvor: https://www.calculateme.com/time/seconds/to-years/.

istim uvjetom da *hashcat* probija milijun ključeva (lozinki) u sekundi, dobije se da je potrebno otprilike 46 godina. To je manje od prva dva slučaja. Da bismo još bolje približili ova vremena, uzmimo da *hashcat* ne računa milijun, već sto milijuna različitih ključeva u sekundi. To bi značilo da sve gornje brojeve podijelimo sa sto. Dobiju se sljedeći rezultati:

- prvi slučaj: 265 godina potrebno za probijanje lozinke
- drugi slučaj: 111 godina potrebno za probijanje lozinke
- treći slučaj: 168 dana potrebno za probijanje lozinke.

Dakle, razlika je drastična! Korištenjem maske smanjeno je vrijeme potrebno za probijanje lozinke iz kompletno nerealne vrijednosti (100 i više godina) na „samo“ 168 dana! Što je, ako bi napadač bio stvarno strpljiv, zapravo moguće.

Ovo je bio pojednostavljeni način kako se može „inteligentno“ optimizirati probijanje lozinke. Što se više toga može smanjiti, pretpostaviti i slično, to će manje vremena biti potrebno za probijanje lozinke.

No, ovo nisu točne brojke. Postoje razni drugi algoritmi kojima se prostor ključa može optimizirati. Kada bi se to napisalo tako da bude razumljivo *hashcatu*, prvo bi se trebao odrediti skup znakova. To se određuje opcijama -1, -2, -3 i -4 za četiri različita moguća skupa znakova. U našem je slučaju dovoljan jedan skup znakova i koristit će se opcija: -1 ?u?d?l, što *hashcat* interpretira kao skup znakova (nazvan) broj 1, sadrži sva velika slova, sve brojeve i sva mala slova. U slijedeća tri slučaja vidjet će se vrijednosti koje daje *hashcat* za svaki pojedini prostor ključa.

Prvi slučaj – jednako su se maskirali svi znakovi da sadrže sva velika i mala slova i brojeve za svih deset znakova. Naredba je sljedeća:

```
hashcat64 -a 3 --keyspace -1 ?u?d?l ?1?1?1?1?1?1?1?1?1?1
```

Broj koji vraća *hashcat* iznosi 3,521,614,606,208. Ako to pretvorimo u vrijeme, koristeći milijun *hasheva* po sekundi, dobije se četrdeset dana.

Drugi slučaj – maskirao se prvi znak da sadrži isključivo mala slova. Sada je naredba za izračun prostora ključa sljedeća:

```
hashcat64 -a 3 --keyspace -1 ?u?d?l ?l?1?1?1?1?1?1?1?1
```

(u maski je nakon prvog upitnika malo slovo L, sve ostalo su brojevi 1). *Hashcat* i u ovom slučaju vraća identičnu brojku – 3,521,614,606,208. No, to ne bi trebalo biti točno jer bi se prostor ključa definitivno trebao smanjiti. Razlog ovome je upravo način na koji *hashcat* radi. On ima svoje unutarnje mehanizme kako izračunava prostore ključa. Oni ovise i o vrsti napada i o duljini ključa te o algoritmima kako sam *hashcat* radi. Dakle, procjenjuje da će neovisno o tome koristi li se prvo slovo kao veliko ili malo (ili broj), i dalje biti potrebno otprilike četrdeset dana da se ta lozinka probije.

Treći slučaj – dodatno su se maskirale zadnje tri znamenke da sadrže isključivo brojeve 0 – 9. Prostor ključa sada se dobije sljedećom naredbom:

```
hashcat64 -a 3 --keyspace -1 ?u?d?l ?l?1?1?1?1?1?d?d?d
```

Nakon maskiranja i zadnje tri znamenke, duljina ključa i dalje je ostala ista (10), ali se prostor ključa sada ipak promijenio. Broj koji vraća *hashcat* je 14,776,336,000 – što pretvoreno u vrijeme sada iznosi malo više od četiri sata.

Iz ovoga se zaključuje da što se preciznije odredi maska prilikom napada sirovom snagom (korištenjem maske), to će biti lakše probiti lozinku. Probijanje lozinke korištenjem maske zahtijeva od napadača da bude inventivan, da ima barem neke poznate ulazne podatke ako je moguće i da pokušava „prodrijeti“ u um onoga koji je tu lozinku zakriptirao.

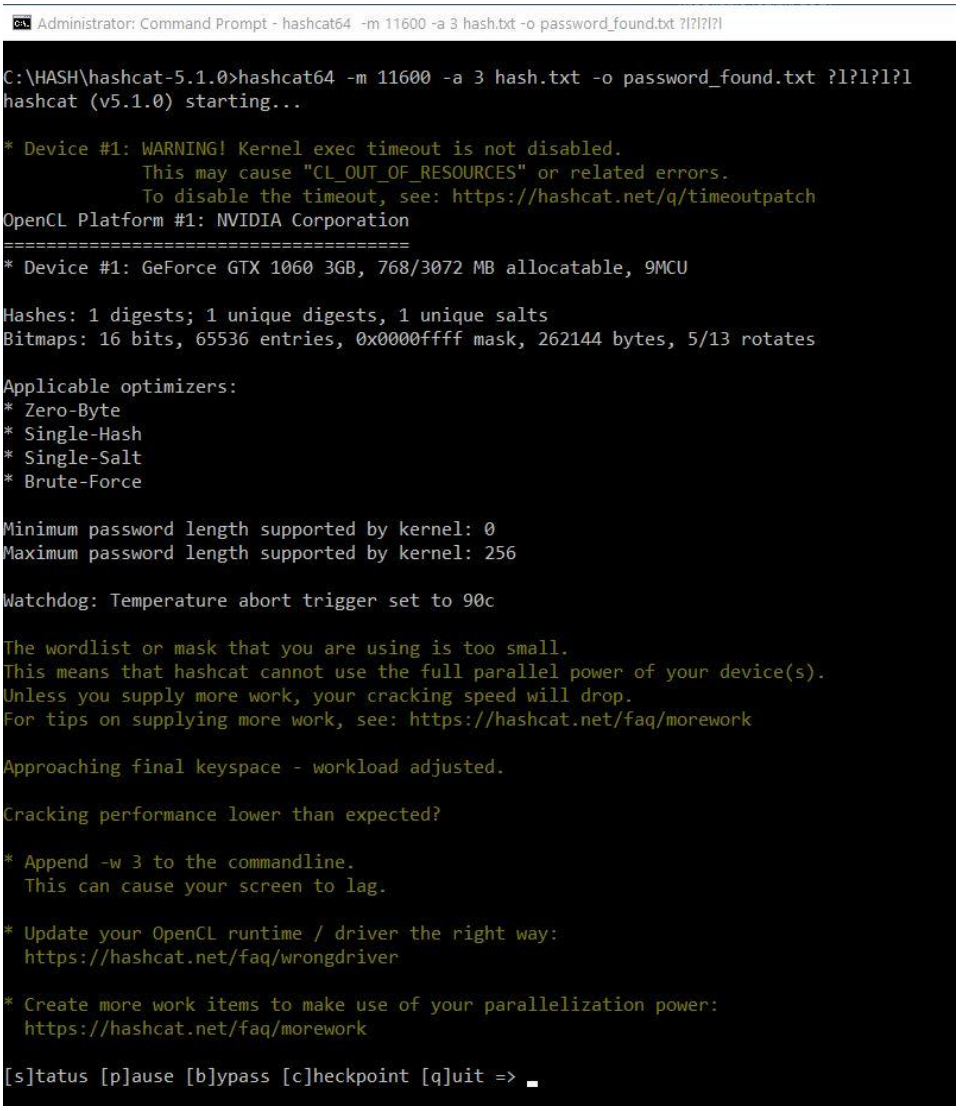
Također, u zadnjem dijelu ove vrste napada u ovom poglavlju prikazat će se primjer kako izgleda kada se napada lozinka od samo četiri znaka.

Primjer – 7-zip *hash*, lozinka je riječ „test“, napad korištenjem maske

Ovo je primjer za koji je već rečeno da će biti prikazan. Datoteka koju pokušavamo probiti je test.7z, zaštićena lozinkom „test“. *Hash* se nalazi u hash.txt datoteci, a sada će biti prikazano prvi put kako se probija *hashcat* programom. Ovo je samo uvod u neke od izvrsnih mogućnosti koje će biti prikazane u nastavku rada. Iskoristit će se ista ona naredba koja je već pokazana.

```
hashcat64.exe -m 11600 -a 3 hash.txt -o password_found.txt ?l?l?l?l
```

S obzirom na to da za većinu napada ne vrijedi to što je *hashcat* stavljen u sistemsku varijablu okruženja, iskopirat će se hash.txt i test.7z datoteke u mapu gdje se nalazi *hashcat* aplikacija. Nakon pokretanja gornje naredbe, ovako izgleda trenutna radnja.



Slika 3.5. Početak probijanja testne lozinke korištenjem napada maske

Pritiskom na tipku [s] dobije se trenutni status radnje.

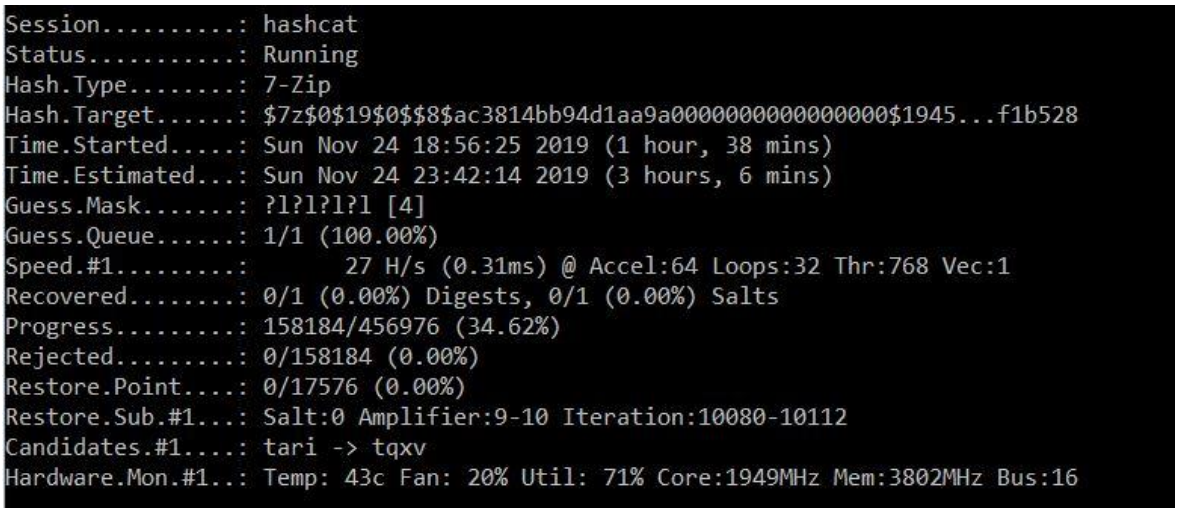
```
Session..........: hashcat
Status...........: Paused
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$ac3814bb94d1aa9a0000000000000000$1945...f1b528
Time.Started.....: Sun Nov 24 18:56:25 2019 (1 hour, 26 mins)
Time.Estimated...: Sun Nov 24 20:25:24 2019 (2 mins, 35 secs)
Guess.Mask.......: ?l?l?l?l [4]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     2601 H/s (0.31ms) @ Accel:64 Loops:32 Thr:768 Vec:1
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 52728/456976 (11.54%)
Rejected.........: 0/52728 (0.00%)
Restore.Point....: 0/17576 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:2-3 Iteration:524256-524288
Candidates.#1....: cari -> cqxv
Hardware.Mon.#1..: Temp: 34c Fan: 20% Util:  0% Core: 139MHz Mem: 405MHz Bus:16
```

Slika 3.6. Trenutni status *hashcat* radnje

Na Slici 3.6. vidljivo je nekoliko bitnih stvari. Speed.#1 prikazuje brzinu probijanja lozinke. U ovom slučaju to je 2600 *hasheva* po sekundi. Trenutni „kandidat“, koji *hashcat* računa da bi bio ispravna lozinka, prikazan je na dijelu Candidates.#1. Progress prikazuje trenutno koliko je (u postocima) izračunato *hasheva*.

Što se točno računa? S obzirom na to da je maska korištena ?l?l?l?l, to znači da se gledaju sve kombinacije isključivo četiriju malih slova. Dakle, od aaaa do zzzz, pa sve kombinacije između. Za prvu kombinaciju (aaaa) *hashcat* izračuna 7-zip *hash* te ga usporedi s onime što mu je zadano u hash.txt datoteci. Potom uzme sljedeću vrijednost (aaab) te računa *hash*. Postupak se ponavlja sve dok ne izračuna sve *hasheve*. Ako u toj datoteci ima više *hasheva*, uzima svaki od njih i uspoređuje s trenutno izračunatim *hashom*. Kada (tj. ako) pronađe istu vrijednost (pogodak), to znači da je pronađena lozinka. Ta će lozinka biti zapisana u datoteku password_found.txt. Brzina od 2600 *hasheva* po sekundi spada u izuzetno spore *hasheve*, premda bi u ovom primjeru *hashcat* trebao relativno brzo pronaći lozinku jer se koriste samo četiri mala slova.

Hashcat je trenutno pauziran. Klikom na tipku [r] nastavlja se (engl. *resume*) probijanje lozinke. Na donjoj je slici vidljivo da je trenutni status *Running*, što znači da *hashcat* trenutno radi „svoj posao“.



Slika 3.7. Nastavak probijanja lozinke

Kada *hashcat* pronađe ispravnu lozinku, tj. kada se ostvari pogodak, zaustavlja svoju radnju i izlazi iz programa. Status koji se ispiše je sada *Cracked*, što znači da je lozinka probijena, tj. pronađena.



Slika 3.8. Lozinka pronađena

U ovom primjeru lozinka je spremljena u datoteku password_found.txt. Kada se pogleda u nju, vidljivo je da je pronađena ispravna lozinka (riječ „test“).



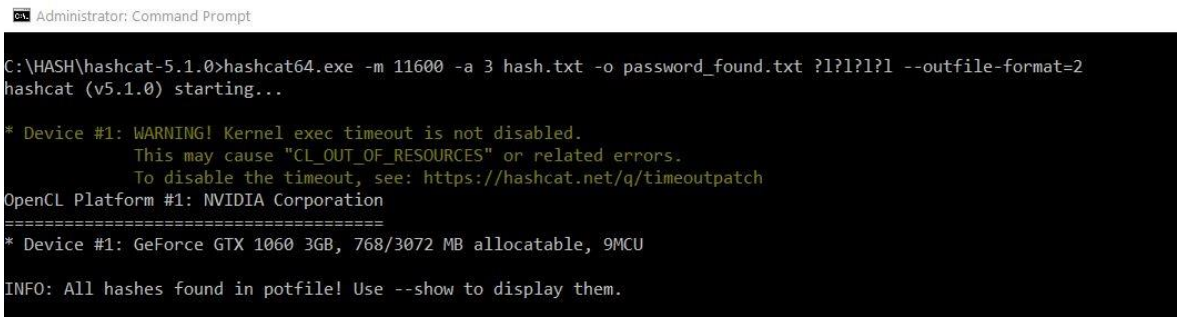
Slika 3.9. Riječ „test“ odgovara *hashu*

Datoteka password_found.txt sadrži cijeli *hash* te se na samom kraju, odvojeno dvotočkom, nalazi riječ „test“. U svim budućim primjerima koristit će se format izlazne datoteke koji

sadrži samo lozinku, u običnom tekstualnom formatu. To se može postići korištenjem sljedeće naredbe:

```
hashcat64.exe -m 11600 -a 3 hash.txt -o password_found.txt ?l?l?l?l --outfile-format=2
```

Pokretanjem gornje naredbe dobije se poruka da je lozinka već pronađena i spremljena u poddatoteci (engl. *potfile*).



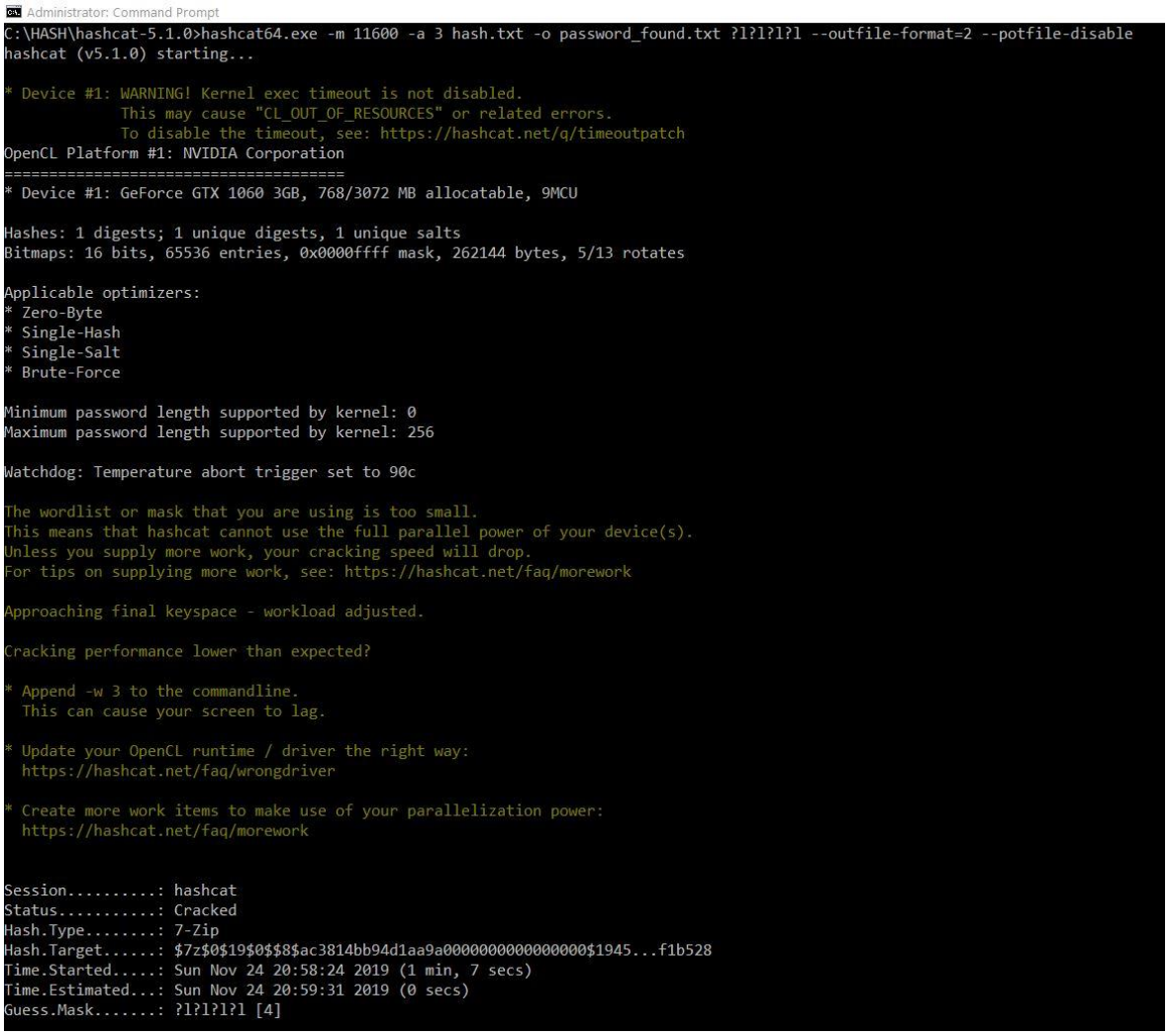
Slika 3.10. Lozinka spremljena u poddatoteci

Poddatoteka služi za spremanje već probijenih lozinki[31]. Ako se već nalazi u *potfile* datoteci, neće ponovo biti probijana. Ako se na kraj naredbe doda opcija --show, datoteka password_found.txt bit će kreirana te će u nju biti spremljena već pronađena lozinka (riječ „test“) bez cijelog ispisanog *hasha* prije toga.

U slučaju da ipak želimo ponovo probijati lozinku, može se iskoristiti opcija `--potfile-disable`, koja služi da se ne koristi poddatoteka.

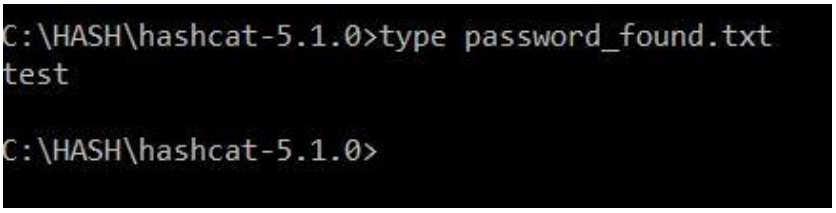
[31] Izvor: https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#what_is_a_potfile Datum: 24. 11. 2019.

Konačna cijela naredba i prikaz ekrana probijanja ove jednostavne lozinke prikazuje donja slika, gdje je vidljivo da je ukupno vrijeme koje je bilo potrebno iznosilo jednu minutu i sedam sekundi.



Slika 3.11. Lozinka probijena za jednu minutu i sedam sekundi

Također, vidljivo je da je lozinka spremljena u tekstualnom formatu u datoteku password_found.txt.



Slika 3.12. Pronađena lozinka u tekstualnom formatu

3.2.3. Napad rječnicima (engl. *Dictionary Attack*)

Kako se izvodi i što je uopće napad rječnikom?

Pretpostavimo da je datoteka zaštićena lozinkom, a ta je lozinka neka engleska riječ. Riječi mogu sadržavati manji ili veći broj slova. Kao što je do sada pokazano, upravo duljina te riječi predstavlja duljinu ključa. Što više slova ima riječ, to je veća duljina ključa i to je teže probiti riječ koristeći napade sirovom snagom. Ako riječ ima primjerice dvadeset ili više slova, gotovo ju je nemoguće probiti nekim standardnim *brute-force* napadom. No, napadač je saznao da je korištena baš engleska riječ. Taj podatak, spoznaja o ključnoj informaciji može biti tema za sebe i nije dio teme ovog završnog rada. Ali, samo za osvrt – saznavanje se vrši putem krađa podataka, industrijskih špijunaža, hakerskim probijanjem u sustave, curenjem podataka unutar firme – svi načini na koje se može doći do lozinki (tj. *hasheva* tih lozinki). Važno je napomenuti da je većina tih načina ilegalna.

Dakle, u sljedećem fiktivnom primjeru napadač je saznao da se koristi engleska riječ za lozinku, a ta je riječ „test“. Iskoristit će se isti primjer na test.7z datoteci da se pokaže kako se može probiti ta riječ, ali ovog puta ne pomoću napada sirovom snagom, već koristeći rječnik (engl. *dictionary*), tj. listu riječi (engl. *wordlist*). Lista riječi je obično datoteka koja sama po sebi nije rječnik, u smislu da sadrži sve poznate riječi nekog jezika, već je to velika lista riječi koje su otkrivene da se najčešće koriste kao lozinke.

Na početku će se prikazati sintaksa naredbe. Za ovu vrstu napada koristi se opcija `-a 0`, koja označava „pravilan“ napad (engl. *Straight*), tj. ovaj se napad naziva i napad rječnikom (engl. *Dictionary Attack / Wordlist attack*).[32] Napad se temelji na određenom rječniku ili listi riječi s engleskim riječima (u ovom slučaju). Potom se *hashcatu* daju naredbe da pokuša svaku riječ, tj. red iz tog rječnika, usporediti s lozinkom. Postavlja se logično pitanje. Gdje nabaviti rječnik / listu riječi? Odgovor na to pitanje u današnje je vrijeme vrlo jednostavan. Dovoljno je pretražiti u *Google* tražilici engleske pojmove kao što su *password wordlist, hashcat dictionary download* itd. Također, bitno je i pripaziti da se rječnici ne skidaju s nesigurnih *web*-stranica.

[32] Izvor: https://hashcat.net/wiki/doku.php?id=dictionary_attack Datum: 25. 11. 2019.

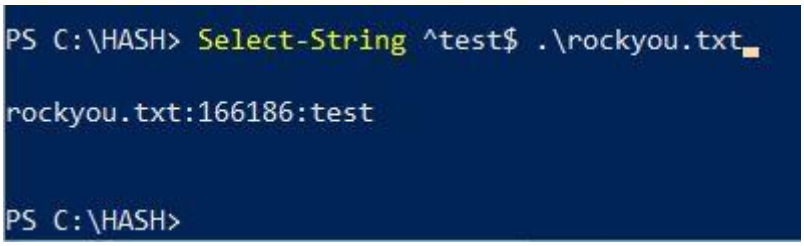
Jedna od poznatijih lista riječi koja se može besplatno skinuti s interneta je datoteka koja se zove rockyou.txt. Ona se nalazi na Kali distribuciji Linuxa[33] i sadrži veliku listu različitih riječi koje su korištene nekom prilikom kao lozinke. Velika je vjerojatnost da sadrži i riječ „test“ u sebi koja će se pokušati probiti. Rockyou.txt datoteka je preuzeta s interneta i postavlja se pitanje sadrži li uistinu riječ „test“. Ali, koliko uopće riječi sadrži ova datoteka? Odgovor je prikazan na donjoj slici uz pomoć dviju jednostavnih naredba na Windows operativnom sustavu.



Slika 3.13. Broj riječi (redova) u rockyou.txt datoteci

Broj riječi koje sadrži ova datoteka je preko četrnaest milijuna! Dakle, to je izuzetno velik broj riječi i stoga ne čudi da ta sama .txt datoteka ima 133 MB. Ali, *hashcat* može u relativno prihvatljivom roku proći cijelu datoteku i pregledati svih 14+ milijuna riječi. Time se ističe i jedna od prednosti napada rječnicima – brzina pretrage! No, to je i mana – ako se koriste lozinke koje se ne mogu naći u rječnicima ili listama riječi poznatih lozinki, ovakav način je kompletno uzaludan, neovisno o brzini.

Donja slika potvrđuje da riječ „test“ postoji u rockyou.txt datoteci.



Slika 3.14. Riječ „test“ unutar rockyou.txt datoteke

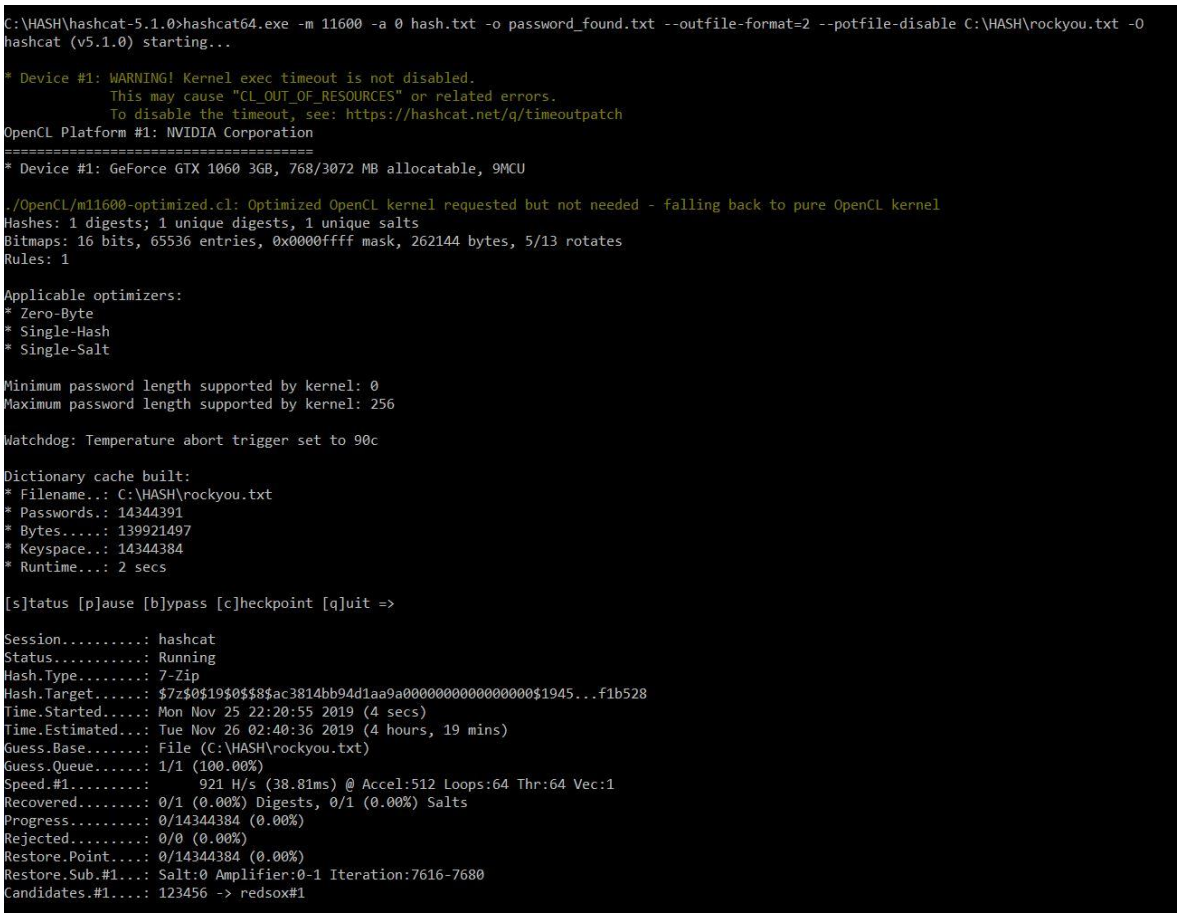
[33] Izvor: https://www.kaggle.com/wjburns/common-password-list-rockyoutxt Datum: 25. 11. 2019.

U ovom primjeru konačna cijela naredba izgleda ovako:

```
hashcat64.exe -m 11600 -a 0 hash.txt -o password_found.txt --
outfile-format=2 --potfile-disable C:\HASH\rockyou.txt -O
```

Prevedeno:

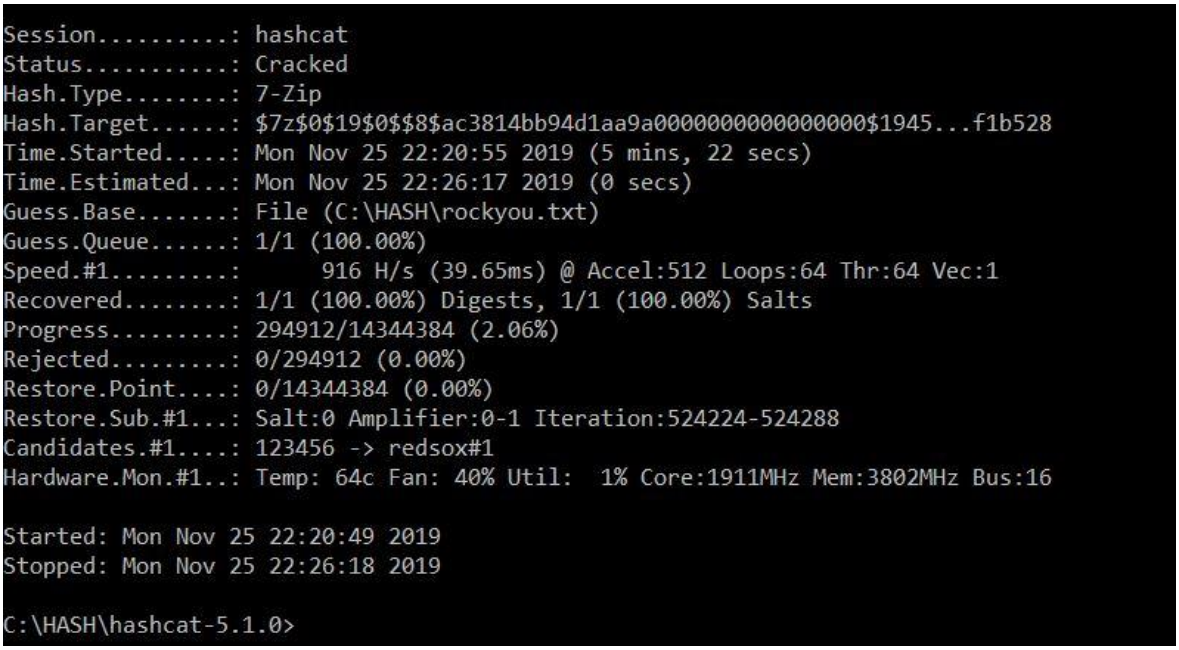
...pokreni 64-bitnu verziju *hashcata*, napadaj 7-zip *hash* koji se nalazi u datoteci hash.txt koristeći „*Straight*“ način napada, tj. napad rječnikom. Ako se pronađe lozinka, ispiši ju u izlaznu datoteku password_found.txt koja je formatirana kao čisti tekst. Nemoj koristiti *potfile* datoteku. Rječnik se nalazi u datoteci C:\HASH\rockyou.txt. Zadnja opcija (-O) govori da se koristi optimizirani kernel.



Slika 3.15. Primjer napada rječnikom

Iz gornje je slike vidljivo da je procijenjeno vrijeme probijanja lozinke, uz ovaj način napada, malo više od četiri sata. To i nije toliko loše vrijeme, s obzirom na velik broj riječi u rockyou.txt datoteci. Samo mjerenje ovakvog napada na pravom primjeru i konačni rezultati bit će prikazani u zadnjem, četvrtom poglavlju ovog rada.

Sljedeća slika pokazuje koliko je vremena trebalo da se probije ova lozinka (riječ „test“) s napadom pomoću rockyou.txt datoteke.



Slika 3.16. Probijena lozinka „test“ pomoću rockyou.txt datoteke

Bilo je potrebno 5 minuta i 29 sekundi, što je impresivno s obzirom na datoteku koja sadrži četrnaest milijuna riječi.

Ako želimo kraće vrijeme napada, iskoristit će se upravo rječnik engleskog jezika[34], koji sadrži manje riječi (194,433) od rockyou.txt datoteke. Na sljedećem primjeru pokazat će se vrijeme koje je potrebno za probijanje lozinke koristeći datoteku s manje riječi. Naravno, lozinka će biti probijena ako se nalazi u rječniku. Rječnik bi trebao sadržavati riječ „test“.

[34] Izvor: http://www.gwicks.net/textlists/english3.zip Datum: 25. 11. 2019.

Za svaki slučaj, provjerit će se sadrži li rječnik koji se nalazi u datoteci english3.txt riječ „test“.

```
PS C:\HASH> Select-String ^test$ .\english3.txt

english3.txt:169404:test
```

Slika 3.17. Rječnik english3.txt sadrži riječ „test“

S obzirom na to da je vidljivo da ovaj rječnik sadrži riječ „test“, pokreće se sljedeća naredba:

```
hashcat64.exe -m 11600 -a 0 hash.txt C:\HASH\english3.txt -o
password_found.txt --outfile-format=2 --potfile-disable
```

Namjerno je sada izmijenjen redoslijed pisanja *hashcat* naredbe. Rječnik koji je sada korišten nalazi se na drugom mjestu u naredbi. Također, izbačena je opcija -O, s obzirom na to da nije potreban optimizirani kernel u ovom slučaju. Ovako sada izgleda izlaz te naredbe.

```
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => 

Session..........: hashcat
Status...........: Running
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$ac3814bb94d1aa9a0000000000000000$1945...f1b528
Time.Started.....: Mon Nov 25 23:38:13 2019 (12 secs)
Time.Estimated...: Mon Nov 25 23:42:23 2019 (3 mins, 58 secs)
Guess.Base.......: File (C:\HASH\english3.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      817 H/s (21.70ms) @ Accel:256 Loops:64 Thr:64 Vec:1
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 0/194433 (0.00%)
Rejected.........: 0/0 (0.00%)
Restore.Point....: 0/194433 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:35520-35584
Candidates.#1....: a -> scratchcards
Hardware.Mon.#1..: Temp: 41c Fan: 20% Util: 99% Core:1961MHz Mem:3802MHz Bus:16
```

Slika 3.18. Probijanje lozinke putem english3.txt rječnika

Vidimo da je sada značajno smanjeno procijenjeno vrijeme za probijanje lozinke. S predviđenih četiri i više sata, vrijeme je spušteno na manje od četiri minute.

No, vidimo da je vrijeme koje je bilo potrebno za probijanje lozinke minimalno kraće nego što je bilo u prvom slučaju u kojem se koristila rockyou.txt datoteka.

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$ac3814bb94d1aa9a0000000000000000$1945...f1b528
Time.Started.....: Mon Nov 25 23:38:13 2019 (5 mins, 11 secs)
Time.Estimated...: Mon Nov 25 23:43:24 2019 (0 secs)
Guess.Base.......: File (C:\HASH\english3.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      624 H/s (7.34ms) @ Accel:256 Loops:64 Thr:64 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 194433/194433 (100.00%)
Rejected.........: 0/194433 (0.00%)
Restore.Point....: 147456/194433 (75.84%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:524224-524288
Candidates.#1....: scratched -> zythum
Hardware.Mon.#1..: Temp: 58c Fan: 36% Util: 41% Core:1923MHz Mem:3802MHz Bus:16

Started: Mon Nov 25 23:38:08 2019
Stopped: Mon Nov 25 23:43:26 2019

C:\HASH\hashcat-5.1.0>
```

Slika 3.19. Probijena lozinka „test“ pomoću english3.txt datoteke

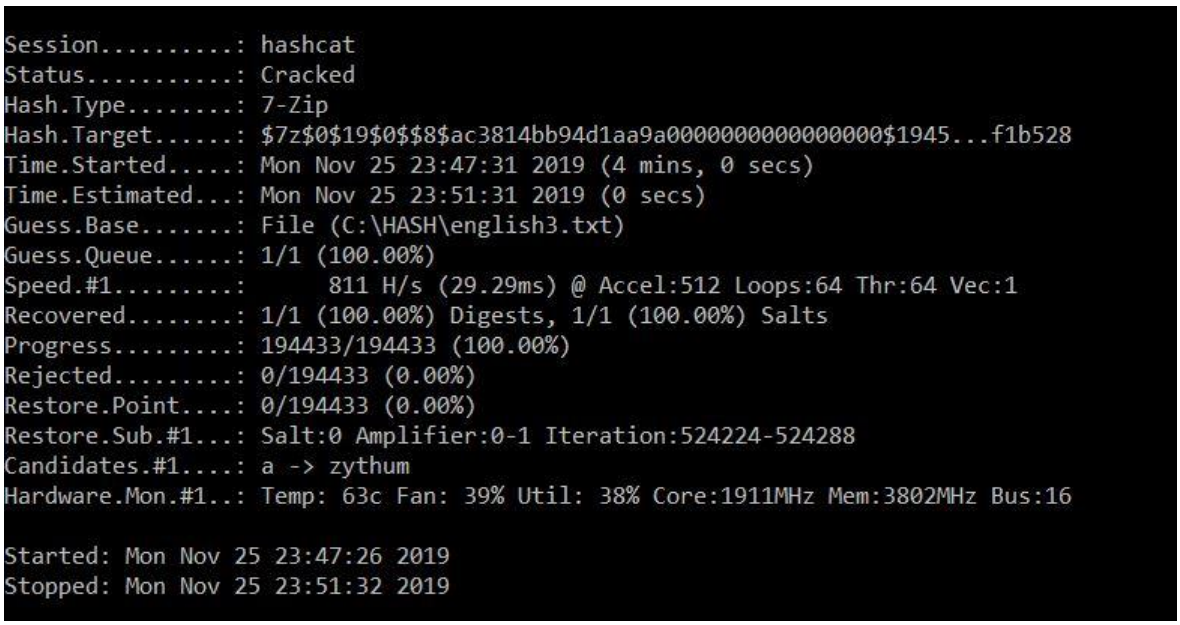
Zašto je to tako? Zar ne bi trebalo biti puno brže s obzirom na broj riječi? Odgovor leži u načinu na koji *hashcat* radi s „malim“ rječnicima te manjim dijelom u činjenici da je u prvom slučaju ipak korištena opcija za optimiziranje kernela. Naime, prilikom pokretanja i korištenja english3.txt rječnika *hashcat* je izbacio sljedeću poruku.

```
Dictionary cache hit:
* Filename..: C:\HASH\english3.txt
* Passwords.: 194433
* Bytes.....: 1908036
* Keyspace..: 194433

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework
```

Slika 3.20. *Hashcat* poruka prilikom korištenja malih rječnika

Kada se uključi opcija -O, za optimizirani kernel, vrijeme koje je bilo potrebno za probijanje lozinke palo je na 4 minute i 6 sekundi.



Slika 3.21. Probijena lozinka „test“ pomoću english3.txt datoteke uz korištenje optimiziranog kernela

Korištenje optimiziranog kernela je napredna opcija. Ubrzava se probijanje lozinke, ali limitirano je otprilike na duljinu od 27 znakova.[35]

Još je vrijedno spomenuti da postoji i kombinacijski napad rječnicima. Opisuje ga upotreba opcije -a 1, koja služi za korištenje točno dvaju rječnika. Svaka riječ iz jednog rječnika (ili liste riječi) doda se na kraj ili na početak drugog rječnika.

Napad rječnikom može se uvelike poboljšati u kombinaciji s pravilima, što je tema sljedećeg poglavlja.

[35] Izvor: https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#what_is_the_maximum_supported_password_length_for_optimized_kernels Datum: 26. 11. 2019.

3.2.4. Napad korištenjem pravila (engl. *Rule-based Attack*)

Napad korištenjem pravila je najkompleksniji, ali ujedno i najzanimljiviji napad koji se može koristiti. O čemu je riječ? Pravila, koja se generiraju, služe da bi se proširile, izmijenile ili na sličan način promijenile riječi koje postoje u nekoj listi riječi (ili rječniku). Zamislimo sljedeću situaciju: većina ljudi koristi neku riječ kao lozinku. Obično se na kraju te riječi dodaju određeni brojevi, npr. godina rođenja, neka druga značajna godina i slično. Također, često je prvo slovo te riječi veliko. Tipičan primjer takvih lozinki (izmišljenih) prikazan je u donjoj tablici.

Korijen riječi	Lozinka
ana	Ana123
ivo	Ivo321
kava	KAVA2019
salica	Salica0011

Tablica 3.1. Izmišljene lozinke od korijena riječi

Ljudi često moraju mijenjati svoje lozinke, primjerice, zbog politike firme u kojoj rade svakih trideset dana. I što se onda dogodi? Obično se zadnja brojka povećava za 1. Nakon nekog vremena promijeni se i sama lozinka; opet se dodaju neki brojevi i tako se ta „kultura lozinki“ održava. Nažalost, zbog ovakvih stvari, ako potencijalni napadač dođe do *hasheva,* može iskoristiti napad pravilima da lakše dođe do lozinki. Ovakav je princip tih pravila: za svaku riječ kreira se pravilo koje tu riječ permutira, nadodaje brojeve na kraj riječi, pretvara sva slova u velika, dodaje brojeve na kraj riječi itd. Mogućnosti su velike i ovise o mašti napadača. U slučaju iz Tablice 3.1. od ovih četiriju riječi može se pravilima kreirati veći broj potencijalnih lozinki.

Vrste napada koje se u opcijama mogu odabrati za napad pravilima su 0, 6 i 7. Opcije `-a 6` i `-a 7` su specifične i za hibridni napad, koji će isto biti spomenut.

Prije nego što se pokaže na konkretnom primjeru, potrebno je objasniti kako se kreiraju ta pravila te koja je točno sintaksa koju *hashcat* koristi.

Na početku želimo napraviti jednostavno pravilo koje sadrži ove uvjete:

- svaka riječ treba započinjati malim slovom i na kraju imati broj 123
- svaka riječ treba započinjati velikim slovom i na kraju imati broj 123
- svaka riječ treba započinjati malim slovom i na početku imati broj 123
- svaka riječ treba započinjati velikim slovom i na početku imati broj 123.

Riječi iz liste riječi su sljedeće: ana, ivo, kava, salica.

Rezultat koji želimo dobiti nakon primjene pravila su sljedeće riječi (potencijalne lozinke).

ana123	ivo123	kava123	salica123
Ana123	Ivo123	Kava123	Salica123
123ana	123ivo	123kava	123salica
123Ana	123Ivo	123Kava	123Salica

Tablica 3.2. Kreiranje prvog pravila

U ovom slučaju to ćemo pravilo sami kreirati i spremiti ga u datoteku moje_pravilo.txt. Također, kreirat ćemo i svoju listu riječi (koja će se sastojati od četiri riječi, svaka u svojoj liniji – ana, ivo, kava, salica) i spremiti ju u datoteku moja_lista.txt. Izlazna datoteka u koju će se spremiti sve potencijalne lozinke kreirane pravilom zvat će se moje_lozinke.txt.

Prije nego što se objasni kako se kreiraju pravila, prikazat će se izgled sintakse ove naredbe:

```
hashcat64.exe moja_lista.txt -r moje_pravilo.txt --stdout
```

Ova naredba nije sam napad, već samo služi da se prikaže (izlaz na ekranu) kako će izgledati lista riječi nakon primjenjivanja pravila. Prevedena naredba glasi ovako: pokreni 64-bitni *hashcat*, učitaj sve riječi iz datoteke moja_lista.txt, na svim tim riječima primijeni pravilo koje se nalazi u datoteci moje_pravilo.txt te ispiši novu dobivenu listu riječi na ekran (opcija `--stdout`).

Ako bi se htjelo spremiti dobivenu novu listu riječi u novu datoteku (koja bi time postala nova lista riječi – to će biti datoteka moje_lozinke.txt), umjesto da samo bude ispisano na ekranu, to se može učiniti dodajući sljedeću opciju u gornju naredbu:

```
-o moje_lozinke.txt --outfile-format=2
```

Sada će se objasniti kako se kreiraju pravila. *Hashcat* za to ima ugrađene funkcije[36] koje se koriste za određivanje pravila. Postoji jako velik broj tih funkcija, a u donjoj su tablici prikazane neke od njih.

Funkcija	**Opis**	**Primjer pravila**	**Ulazna riječ**	**Izlazna riječ**
l	Pretvori sva slova u mala	l	Ana	ana
u	Pretvori sva slova u velika	u	Ana	ANA
c	Prvo slovo veliko	c	ivo	Ivo
t	Izmijeni sva velika slova u mala i obratno	t	Ana	aNA
$X	Dodaj znak X na kraj riječi	$1	ana	ana1
^X	Dodaj znak X na početak riječi	^1	ana	1ana
iNX	Dodaj znak X nakon pozicije N	i2T	ana	anTa

Tablica 3.3. Primjer nekih od pravila *hashcat* programa

Pravila se kreiraju tako da se u neku proizvoljnu tekstualnu datoteku stave funkcije koje želimo koristiti – svaku u svoj red. Ovisno o tim funkcijama, bit će izvršeno i pravilo. Ako bi se htjelo izmijeniti svim riječima koje se učitaju iz liste riječi sva velika slova u mala te sva mala slova u velika, u tu tekstualnu datoteku samo će se staviti slovo „t“ u prvi red. Nakon toga može se pozvati *hashcat* opcijom `-r`, koja će učitati i to novo kreirano pravilo. Pravilo će se staviti u datoteku moje_pravilo.txt. U datoteku moja_lista.txt stavit će se riječi: Ana, Ivo, Kava, Salica, a izlazna datoteka zvat će se moje_lozinke.txt.

[36] Izvor: https://hashcat.net/wiki/doku.php?id=rule_based_attack Datum: 28. 11. 2019.

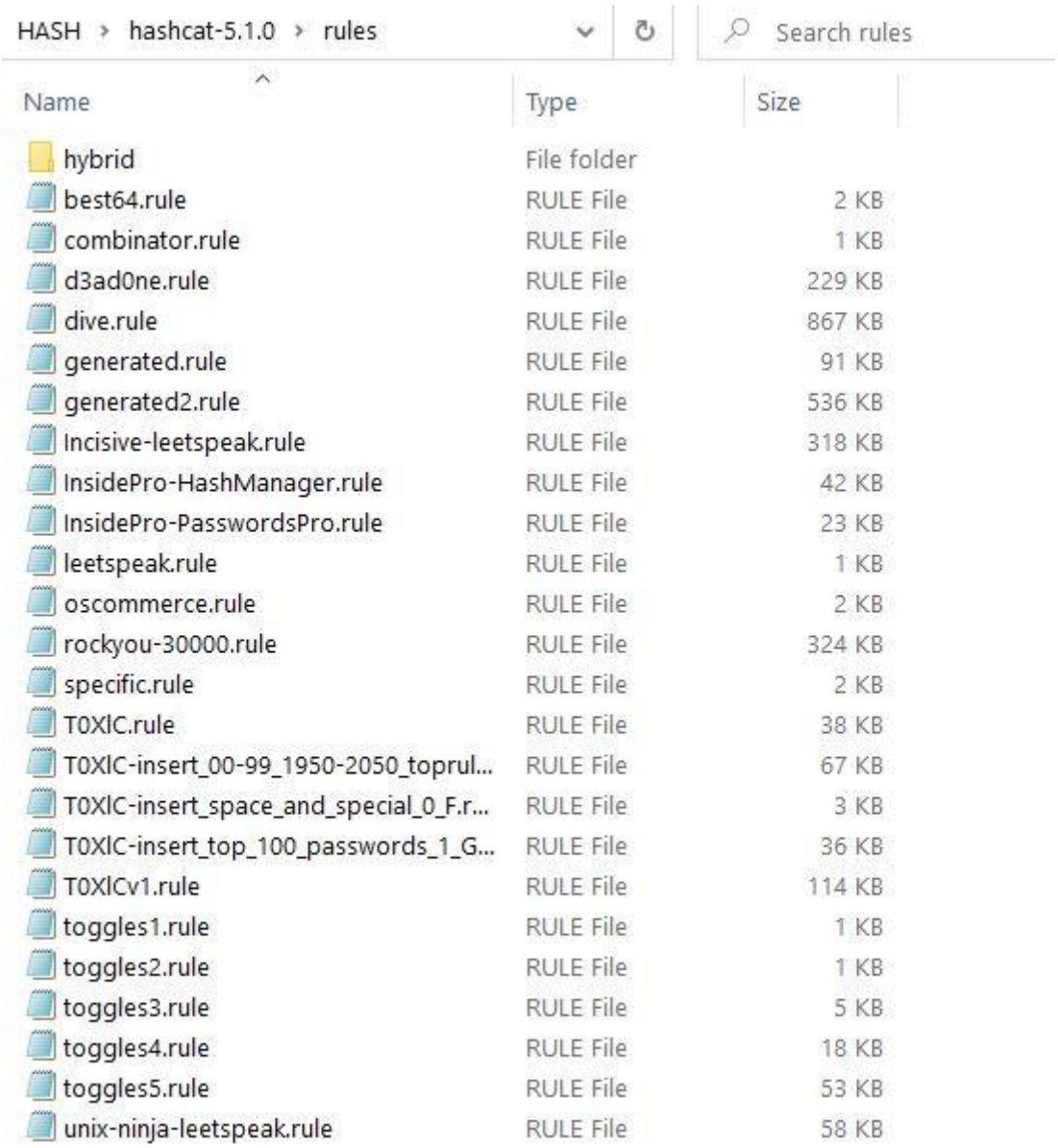
Na donjoj slici prikazan je točan gornji primjer s pravilom „t“.



Slika 3.22. Primjer novokreiranog pravila „t“

Kao što slika pokazuje, sva su velika i mala slova izmijenjena. Kada bi se baš probijao *hash* koristeći ovo pravilo, u naredbu bi se još uključile opcije `-a` za vrstu napada, `-m` za vrstu *hasha* te `-o` za spremanje potencijalno pronađene lozinke u neku izlaznu datoteku. Dakle, pokušavale bi se lozinke aNA, iVO, kAVA, sALICA te ako bi jedna od njih uspjela, taj bi se podatak zapisao u izlaznu datoteku. Ovo je vrlo jednostavan primjer sa samo četirima riječima, ali za ulaznu listu riječi može se staviti i rockyou.txt datoteka te se primjenom pravila na njoj svim riječima zamijene velika i mala slova – što nije mali broj lozinki.

Naravno, u pravilima ne mora biti samo jedna funkcija, kao u gornjem slučaju u kojem je korištena samo funkcija „t“. Može ih biti nekoliko ili čak nekoliko desetaka stranica različitih funkcija. Mogu se ručno izmišljati razne varijante različitih funkcija, primjerice, dodaj sve brojeve 0 – 500 na kraj svakog reda i slično. No, takvo „ručno“ osmišljavanje i testiranje tih funkcija moglo bi se pretvoriti u mukotrpan posao. Taj posao nije nemoguć, ali može trajati neko vrijeme. U pomoć uskače *hashcat* koji prilikom instalacije dolazi s nekoliko desetaka različitih već gotovih pravila. Ta pravila u sebi sadrže provjerene funkcije koje se mogu koristiti prilikom napada. Pravila se nalaze u mapi „*rules*“. Ona se nalazi u instalacijskoj mapi u kojoj je instaliran *hashcat,* kao što je prikazano na sljedećoj slici.



Slika 3.23. *rules* mapa unutar *hashcat* mape

Vidljivo je da nastavci datoteka ovih pravila završavaju s .rule, ali to su i dalje samo tekstualne datoteke i ne moraju imati nastavak .txt. Jedno od poznatijih pravila je best64.rule koje se često koristi. *Hashcat* isto tako dozvoljava korištenje više od jednog pravila, ali pritom se u naredbi mora navesti svako od pravila. Primjerice:

```
hashcat64.exe moja_lista.txt -r moje_pravilo_1.txt -r moje_pravilo_2.txt -r moje_pravilo_3.txt --stdout
```

Na kraju ovog poglavlja kreirat će se pravilo za koje je određeno da će biti zadatak kao u Tablica 3.2.

Raščlanjeni svaki dio zadatka, tj. pravila, koje je potrebno kreirati za ovaj primjer objašnjen je ispod. Svaki uvjet koji je zadan bit će jedna linija funkcija u datoteci pravila nazvanoj moje_pravilo.txt. Ovo su bili uvjeti:

- Svaka riječ treba započinjati malim slovom i na kraju imati broj 123.

S obzirom na to da je poznata ulazna lista riječi (datoteka moja_lista.txt) i da ona sadrži isključivo riječi s malim slovima, neće biti potrebno koristiti funkciju „l“ (malo slovo L – engl. *lowercase*). Da bi se dobio broj 123 na kraju svake riječi, funkcija koju je potrebno koristiti glasi $1 $2 $3.

- Svaka riječ treba započinjati velikim slovom i na kraju imati broj 123.

Funkcija koja se koristi za prva velika slova je „c“ (engl. *capitalize*). Broj 123 na kraju svake riječi dobije se upotrebom $1 $2 $3.

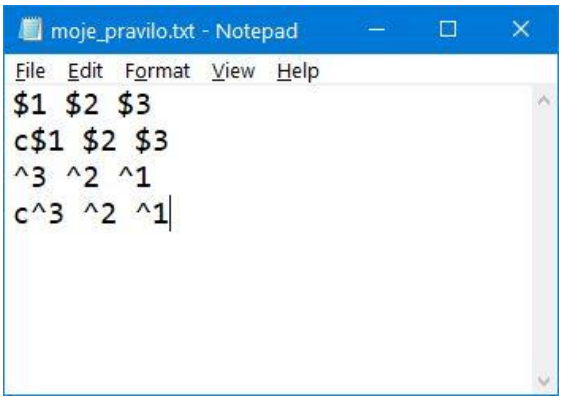
- Svaka riječ treba započinjati malim slovom i na početku imati broj 123.

Mala slova možemo opet zanemariti, s obzirom na to da je ulazna lista riječi načinjena od samo malih slova. Da bismo dobili broj 123 na početku riječi, potrebno je koristiti funkciju koja glasi ^3 ^2 ^1.

- Svaka riječ treba započinjati velikim slovom i na početku imati broj 123.

Funkcija „c“ za svako prvo veliko slovo i opet ^3 ^2 ^1 kako bi se dobio broj 123 na početku riječi.

Na donjoj je slici prikazana gotova pripremljena datoteka koja sadrži pravilo sa svim gornjim uvjetima.

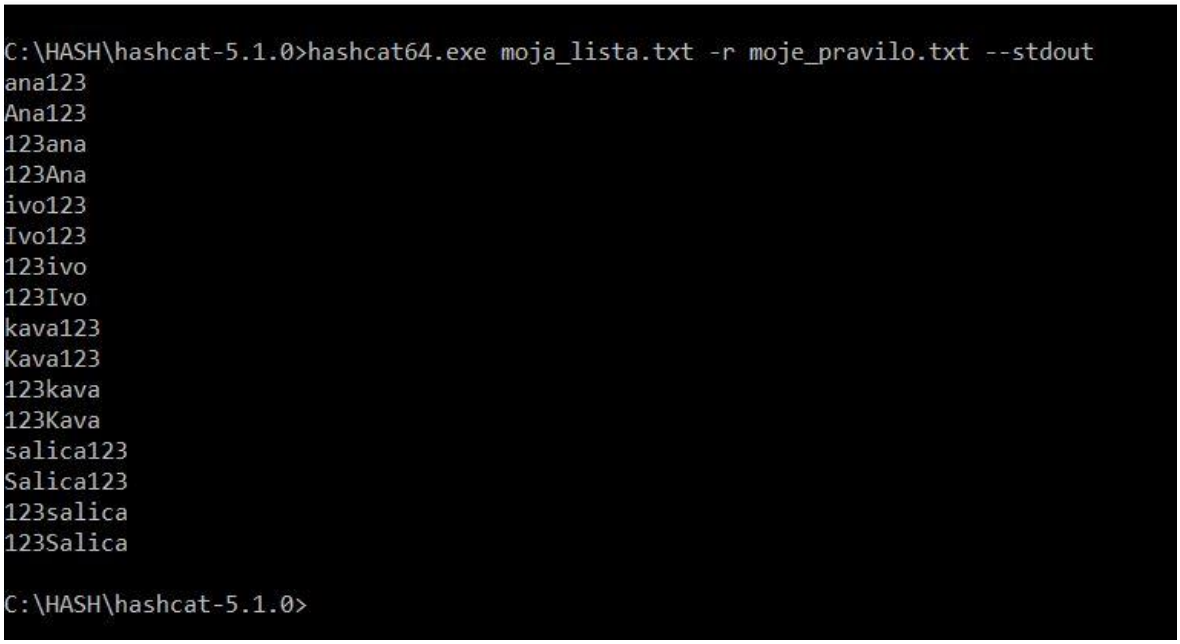


Slika 3.24. Kreirano pravilo u datoteci moje_pravilo.txt

Sada će se pokrenuti cijela naredba i provjeriti jesu li dobiveni ispravni kandidati za lozinke sa zadanim uvjetima. Naredba glasi:

```
hashcat64.exe moja_lista.txt -r moje_pravilo.txt --stdout
```

Ovako izgleda ispis koji se dobije na ekranu.



Slika 3.25. Ispis na ekranu lozinki kreiranih pravilom

Kao što je vidljivo, uspješno je kreirano pravilo. Mogućnosti kreiranja pravila su praktički neograničene i ovise jedino o mašti i snalažljivosti napadača na lozinke. Bitno je napomenuti da je većina te mašte i snalažljivosti već isprobana. Velik broj različitih korisnih permutacija pravila već se nalazi unutar samog *hashcata* u mapi „*rules*“. Kad god je moguće, bolje je koristiti već gotova pravila nego izmišljati nova. Naravno, postoje i slučajevi kada je korisno kreiranje vlastitih pravila.

U sljedećem i zadnjem poglavlju bit će obrađeni praktični primjeri probijanja lozinki s gore navedenim vrstama napada i prikazat će se kako to sve izgleda u praksi.

4. Probijanje lozinke

U ovom poglavlju bit će praktično prikazano koliko je vremena potrebno za probijanje lozinki različitih duljina uz pomoć različitih vrsta napada.

4.1. Određivanje vrste napada

U napadu će se koristiti i praktično pokazati tri vrste napada:

- napad sirovom snagom
- napad rječnikom
- napad korištenjem pravila.

Prvenstveni fokus napada bit će na *hashu* 7-zip datoteke, što znači da će se razne .7z datoteke zakriptirati različitim lozinkama i potom pokušati probijati. Isto tako, kako bi se bolje prikazalo koliko veliki utjecaj ima i sama jačina *hasha*, a 7-zip *hash* spada pod jake *hasheve* (tj. spore, teško se probijaju), usporedno će se (u nekim slučajevima, gdje to bude potrebno), koristiti i MD5 *hashevi*. Oni su puno brži za probijanje i trebali bi služiti kao izvrsna komparacija prilikom probijanja istih lozinki.

Svaki put bit će jasno naznačeno koji se točno *hash* koristi (MD5 ili 7-zip), koja je lozinka korištena, tj. koja se točno lozinka pokušava probiti i koja vrsta napada se koristi.

4.1.1. Duljina lozinke i njezin utjecaj

Već je objašnjeno da što je veća duljina lozinke, to ju je teže probiti. Da bi se uspješno napravilo testiranje i izmjerilo vrijeme, moraju se uzeti u obzir neki realni parametri, u okvirima koji su (potencijalno) ostvarivi. Zbog toga je odlučeno da će se promatrati tri različita ranga duljine lozinke:

- duljina lozinke do pet znakova (uključujući i peti)
- duljina lozinke od pet do deset znakova (uključujući i deseti)
- duljina lozinke od deset do petnaest znakova (uključujući i petnaesti).

Isto tako, lozinke smiju biti sastavljene isključivo od velikih i malih slova engleske abecede, brojeva i specijalnih znakova. Korištenje većeg seta znakova od toga praktički bi onemogućilo probijanje lozinke.

4.1.2. Generiranje slučajno odabranih lozinki

Za dio lozinki koji će biti korišten prilikom napada sirovom snagom koristit će se KeePass program te će se njime generirati slučajno odabrane lozinke određenih duljina. Prilikom napada rječnikom iskoristit će se engleske riječi, a za napade pravilima kreirat će se osobna lista riječi koja će biti upotrijebljena.

4.2. Usporedna analiza napada na različitim računalima

Da bi se dobila metrika vremena potrebnog za probijanje lozinke i napravila usporedna analiza, određeni su sljedeći uvjeti:

a. Prvo će se koristiti slabije računalo, a potom jače računalo. Ako slabije računalo probije lozinku, izmjerit će se koliko je vremena potrebno jačem računalu da tu istu lozinku probije.
b. Određeno vrijeme za probijanje lozinke je deset sati. Ako se unutar tog vremena lozinka probije, smatrat će se da je uspješno obavljeno probijanje lozinke. Ako slabije računalo ne uspije probiti lozinku unutar tog vremena, ali jače računalo uspije, isto će se smatrati da je lozinka uspješno probijena.

4.2.1. Specifikacije računala za probijanje lozinke

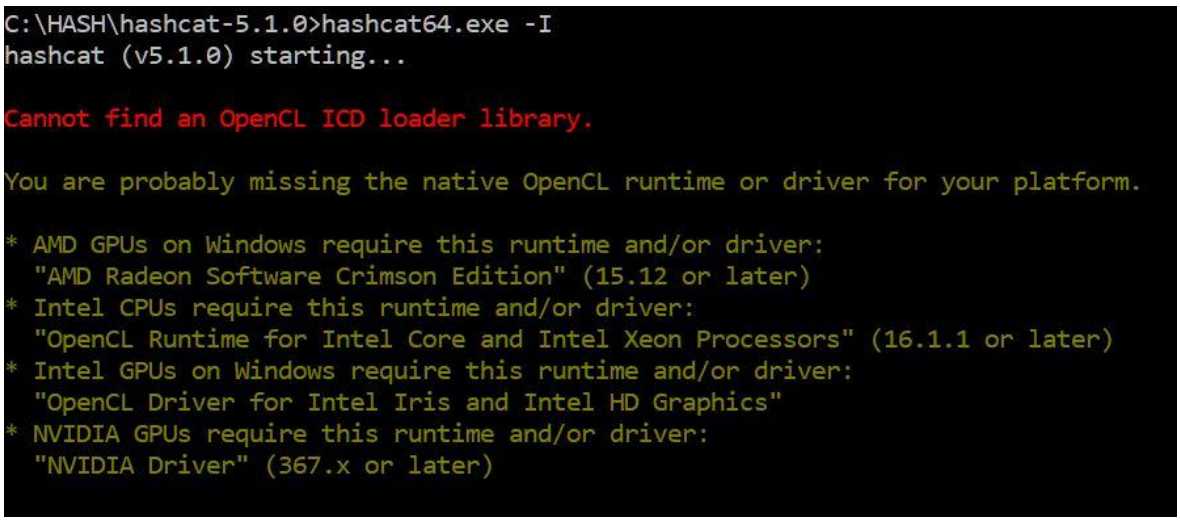
Koristit će se dva različita računala. Također, koristit će se *hashcat* od verzije 3.00 na više koji ima mogućnost upotrebe i GPU-a[37] prilikom procesiranja probijanja lozinki – što je u većini slučajeva brže od korištenja samo CPU-a. I GPU i CPU zahtijevaju OpenCL[38] okvir.

Da bi *hashcat* uspješno radio na računalu, potrebni su mu instalirani upravljački programi (engl. *drivers*) za OpenCL – kako za CPU, tako i za GPU. Nažalost, OpenCL podrška za različite procesore nije uvijek idealna, tako da nakon isprobavanja nekih od računala uopće nije bilo moguće koristiti *hashcat*.

Hashcat sam prepoznaje hoće li koristiti CPU ili GPU. U slučaju da su oba prepoznata, može se opcijom -D i opcijom --force koristiti ili jedan ili drugi. Naredba za prikaz uređaja koje *hashcat* prepoznaje je: `hashcat64 -I`.

[37] Izvor: https://en.m.wikipedia.org/wiki/Hashcat Datum: 28. 11. 2019.
[38] Izvor: https://en.m.wikipedia.org/wiki/OpenCL Datum: 28. 11. 2019.

Prilikom odabira i isprobavanja računala javljale su se razne greške, što je prikazano na donjoj slici.



Slika 4.1. Nemogućnost korištenja *hashcat* programa na nekim računalima

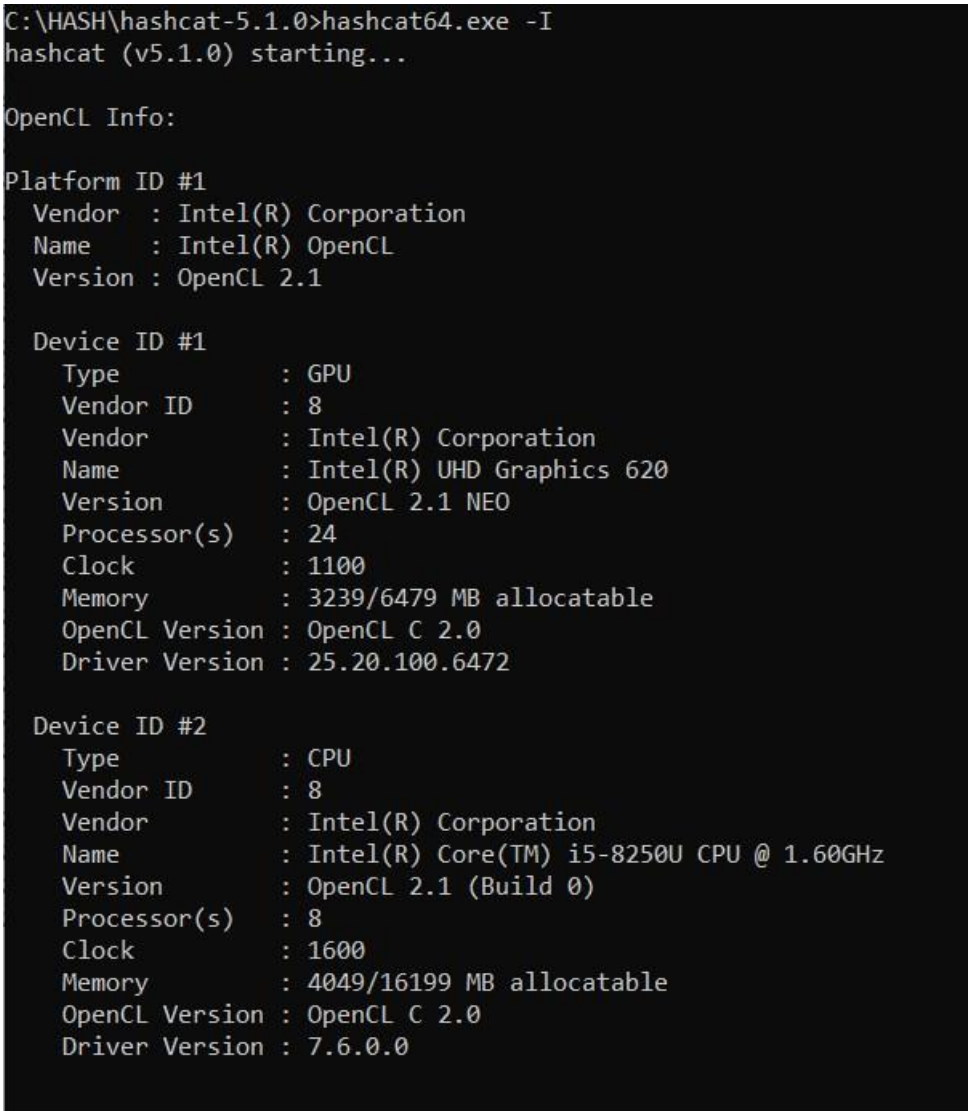
Čak i nakon instalacije potrebnih upravljačkih programa u gornjem primjeru na Slika 4.1. i dalje nije bilo moguće koristiti *hashcat*. U konačnici, pronađena su dva računala koja zadovoljavaju uvjete – jedno slabije i jedno jače računalo.

U donjoj tablici prikazane su specifikacije oba računala koja će se koristiti prilikom pokušaja probijanja lozinke.

	Računalo broj 1 – slabije	Računalo broj 2 – jače
Procesor	Intel Core i5 8250U	AMD Ryzen 5 1600
Grafička	integrirana Intel HD 620	NVIDIA GeForce GTX 1060 3GB
Memorija	16 GB DDR3	64 GB DDR4

Tablica 4.1. Specifikacije računala

Na sljedećim slikama prikazani su uređaji koje *hashcat* prepoznaje na svakom od računala koje će se koristiti za probijanje lozinki. Na slabijem računalu prepoznati su i GPU i CPU, a na jačem samo GPU.



Slika 4.2. Računalo 1 – slabije



Slika 4.3. Računalo 2 – jače

Na slabijem računalu koristit će se opcija `-D 1` kako bi se forsiralo korištenje isključivo CPU jedinice, tj. procesora.

4.2.2. Određivanje snage *hasha*

Kako bismo bolje usporedili ova dva računala i stvarno dokazali da je jedno slabije od drugog, iskoristit će se ugrađena *hashcat* opcija koja služi za testiranje određivanja snage *hasha* (engl. *benchmark*). Opcija za to je `-b` (što označava *benchmark*). *Hashcat* ovom opcijom mjeri koliko brzo može računati *hasheve* po sekundi za određenu vrstu *hasha* (npr. za MD5 *hash* algoritam). Na sljedećim slikama prvo će biti prikazano slabije računalo i koliko brzo može računati MD5 i 7-zip *hash*. Također, bit će prikazane iste opcije korištene na jačem računalu i u komparacijskoj tablici bit će jasno pokazano koliko je jedno računalo slabije (tj. jače) od drugog, u odnosu na koliki broj *hasheva* može računati.

```
C:\HASH\hashcat-5.1.0>hashcat64.exe -m 0 -b -D 1
hashcat (v5.1.0) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported password length.
To disable the optimized kernel code in benchmark mode, use the -w option.

* Device #1: Intel's OpenCL runtime (GPU only) is currently broken.
             We are waiting for updated OpenCL drivers from Intel.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: Intel(R) Corporation
======================================
* Device #1: Intel(R) UHD Graphics 620, skipped.
* Device #2: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 4049/16199 MB allocatable, 8MCU

Benchmark relevant options:
===========================
* --opencl-device-types=1
* --optimized-kernel-enable

Hashmode: 0 - MD5

Speed.#2.........:   591.3 MH/s (14.04ms) @ Accel:1024 Loops:1024 Thr:1 Vec:8
```

Slika 4.4. Slabije računalo – MD5 *hash*

```
C:\HASH\hashcat-5.1.0>hashcat64.exe -m 11600 -b -D 1
hashcat (v5.1.0) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported password length.
To disable the optimized kernel code in benchmark mode, use the -w option.

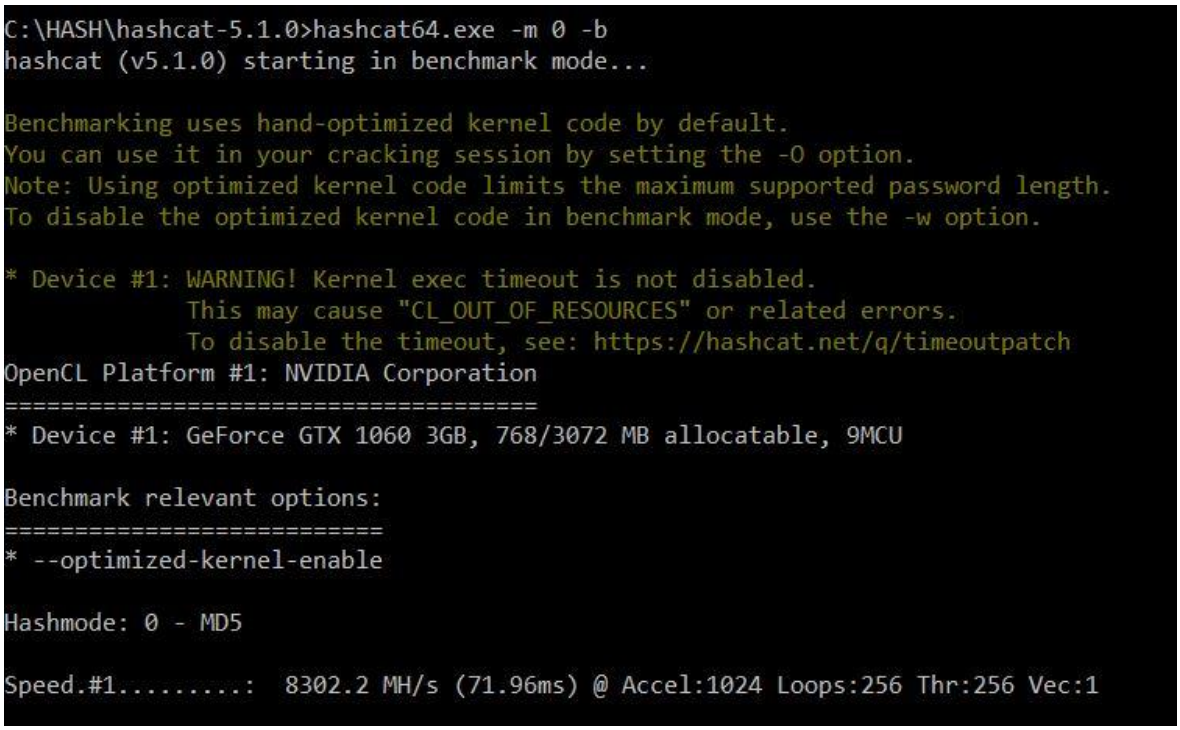
* Device #1: Intel's OpenCL runtime (GPU only) is currently broken.
             We are waiting for updated OpenCL drivers from Intel.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: Intel(R) Corporation
======================================
* Device #1: Intel(R) UHD Graphics 620, skipped.
* Device #2: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 4049/16199 MB allocatable, 8MCU

Benchmark relevant options:
===========================
* --opencl-device-types=1
* --optimized-kernel-enable

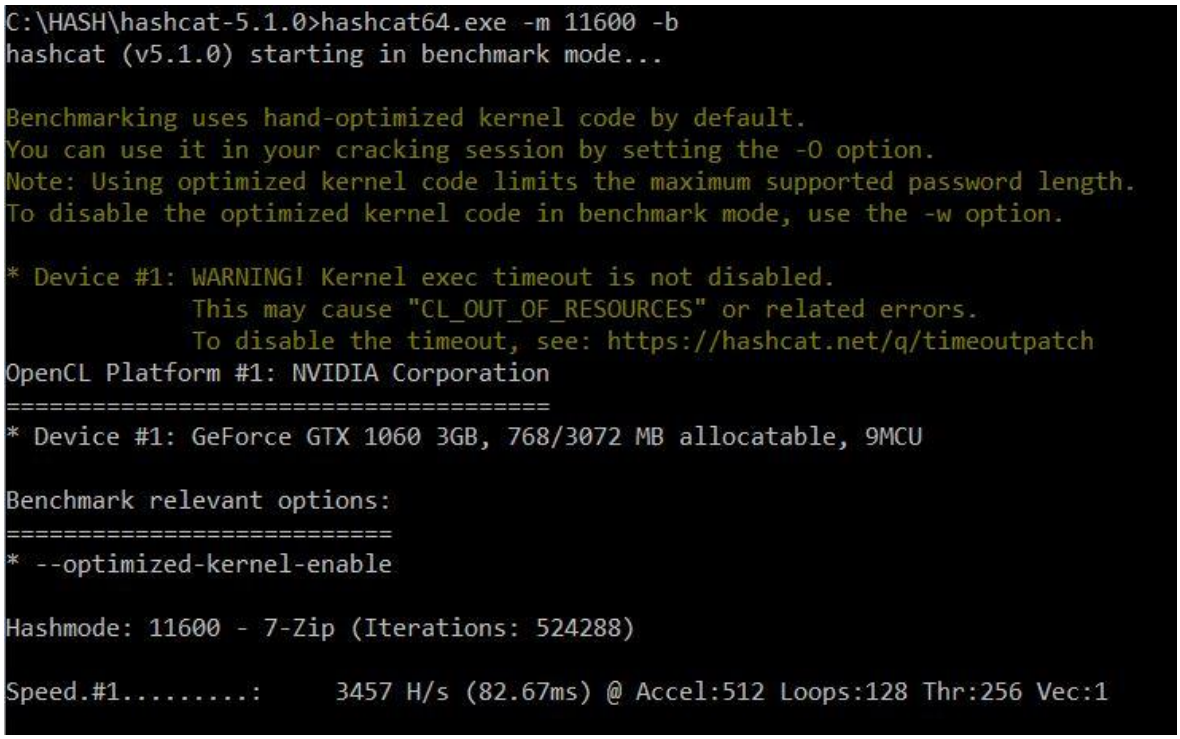
Hashmode: 11600 - 7-Zip (Iterations: 524288)

Speed.#2.........:       62 H/s (64.58ms) @ Accel:1024 Loops:256 Thr:1 Vec:8
```

Slika 4.5. Slabije računalo – 7-zip *hash*



Slika 4.6. Jače računalo – MD5 *hash*



Slika 4.7. Jače računalo – 7-zip *hash*

Komparativni prikaz bitnih brojeva s gornjih slika, tj. koliko brzo *hashcat* može računati *hasheve* na svakom od računala, prikazan je u donjoj tablici.

Vrsta *hash* algoritma	Brzina računanja *hasheva* – slabije računalo	Brzina računanja *hasheva* – jače računalo
MD5	591 MH/s	8302 MH/s
7-zip	62 H/s	3457 H/s

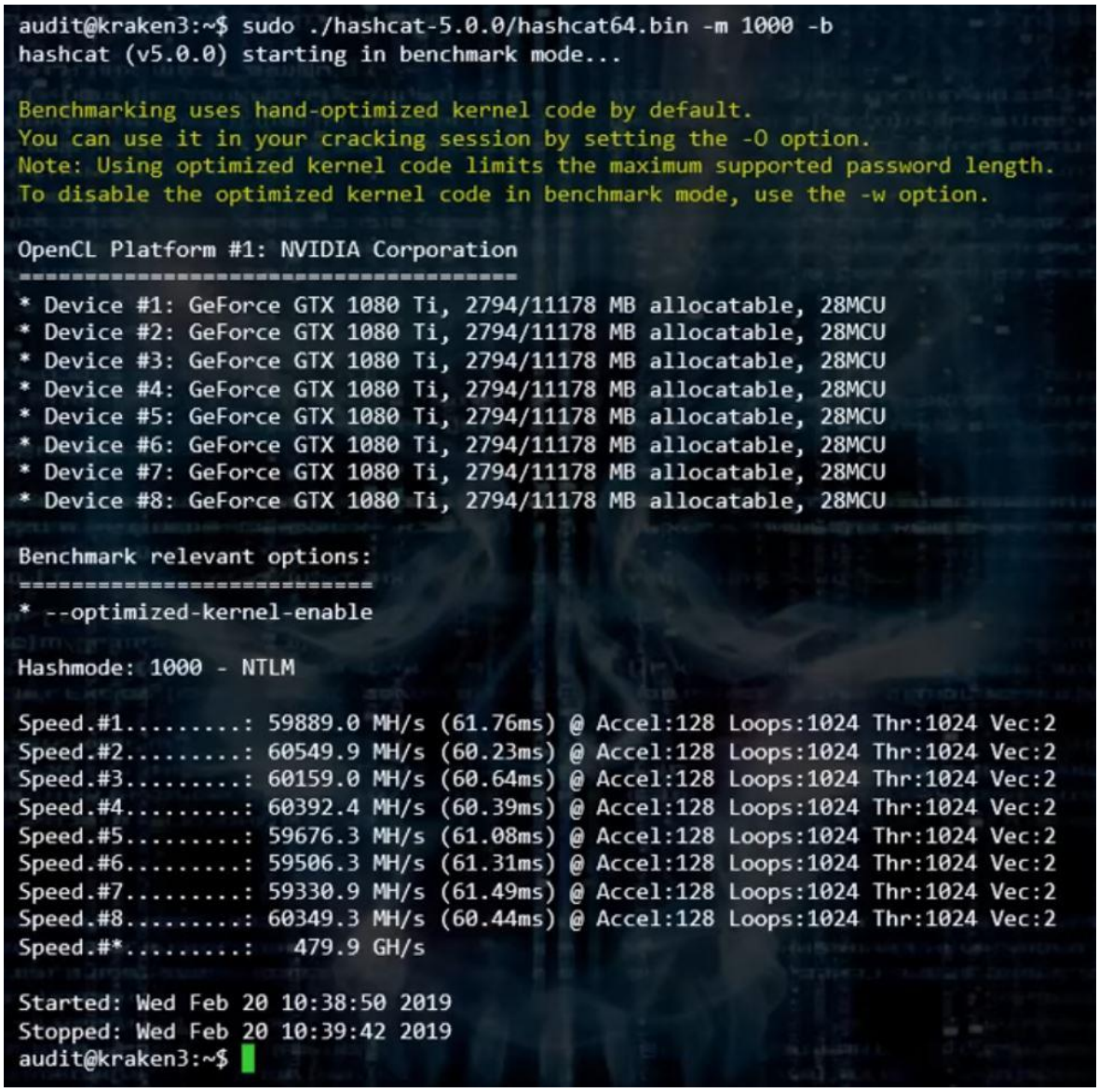
Tablica 4.2. Usporedni test brzine računanja *hasheva*

Vidljivo je da jače računalo može otprilike četrnaest puta brže računati MD5 *hasheve* te oko 55 puta brže računati 7-zip *hasheve*! Razlika je jasna i uočljiva te je sigurno dokazano da je jedno računalo slabije, a drugo jače za potrebe testiranja probijanja lozinki. Također, može se zaključiti da je teže probijati 7-zip *hasheve* naspram MD5.

4.2.3. Usporedbe s drugim sustavima

Postoje i specijalizirani *online* servisi koji omogućavaju uslugu probijanja lozinki.[39] Takvi servisi koriste računala sa snažnim grafičkim karticama koje su međusobno povezane i pružaju usluge probijanja jako velikog broja *hasheva*, zakriptiranih datoteka i slično. Jedna od takvih grupa profesionalaca, koji se bave sigurnošću i penetracijskim testiranjima[40], napravila je i mjerenja potrebna za probijanje NTLM *hasheva* s izuzetno snažnim računalom sastavljenim od 8 NVIDIA 1080 Ti grafičkih procesora[41]. Umjesto nekoliko dana, bilo im je potrebno nekoliko sati da pokušaju probiti NTLM *hasheve* od lozinki sastavljenih od osam i devet znakova. Na donjim slikama prikazana su testna mjerenja (engl. *benchmark*) njihovog računala u odnosu na dva računala korištena u ovom radu.

[39] Izvor: https://www.onlinehashcrack.com/ Datum: 30. 12. 2019.
[40] Izvor: https://www.shellntel.com/ Datum: 30. 12. 2019.
[41] Izvor: https://www.shellntel.com/blog/2019/2/19/how-to-build-a-2nd-8-gpu-password-cracker Datum: 30. 12. 2019.



Slika 4.8. *Shellntel benchmark* NTLM *hasheva*

Ukupna brzina je gotovo 480 GH/s, što je izuzetno velika brzina računanja *hasheva*.

Isti rezultati na računalima korištenim u ovom radu su sljedeći.

```
C:\HASH\hashcat-5.1.0>hashcat64.exe -m 1000 -b -D 1
hashcat (v5.1.0) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported password length.
To disable the optimized kernel code in benchmark mode, use the -w option.

* Device #1: Intel's OpenCL runtime (GPU only) is currently broken.
             We are waiting for updated OpenCL drivers from Intel.
             You can use --force to override, but do not report related errors.
OpenCL Platform #1: Intel(R) Corporation
======================================
* Device #1: Intel(R) UHD Graphics 620, skipped.
* Device #2: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 4049/16199 MB allocatable, 8MCU

Benchmark relevant options:
===========================
* --opencl-device-types=1
* --optimized-kernel-enable

Hashmode: 1000 - NTLM

Speed.#2.........:   961.0 MH/s (8.48ms) @ Accel:1024 Loops:1024 Thr:1 Vec:8

Started: Mon Dec 30 18:33:50 2019
Stopped: Mon Dec 30 18:34:07 2019
```

Slika 4.9. *Benchmark* NTLM *hasheva* na slabijem računalu

```
C:\HASH\hashcat-5.1.0>hashcat64.exe -m 1000 -b
hashcat (v5.1.0) starting in benchmark mode...

Benchmarking uses hand-optimized kernel code by default.
You can use it in your cracking session by setting the -O option.
Note: Using optimized kernel code limits the maximum supported password length.
To disable the optimized kernel code in benchmark mode, use the -w option.

* Device #1: WARNING! Kernel exec timeout is not disabled.
             This may cause "CL_OUT_OF_RESOURCES" or related errors.
             To disable the timeout, see: https://hashcat.net/q/timeoutpatch
OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1060 3GB, 768/3072 MB allocatable, 9MCU

Benchmark relevant options:
===========================
* --optimized-kernel-enable

Hashmode: 1000 - NTLM

Speed.#1.........: 14476.2 MH/s (82.56ms) @ Accel:1024 Loops:512 Thr:256 Vec:1

Started: Mon Dec 30 18:38:02 2019
Stopped: Mon Dec 30 18:38:11 2019
```

Slika 4.10. *Benchmark* NTLM *hasheva* na jačem računalu

Usporedne brzine računanja NTLM *hasheva* preglednije su prikazane u donjoj tablici.

	Shellntel	Slabije računalo	Jače računalo
NTLM *hash*	480 GH/s	0,96 GH/s	14,4 GH/s

Tablica 4.3. Brzine računanja NTLM *hasheva*

Vidljivo je da su brzine neusporedivo veće kada se koriste snažna računala s više zajedno povezanih GPU-a[42]. *Shellntelovo* računalo u odnosu na slabije korišteno u ovom radu je 480 puta brže. Za predodžbu, to bi značilo da lozinka za koju je potrebno 480 sati (dvadeset dana) da bude probijena, ovo računalo može probiti za samo sat vremena.

4.3. Mjerenje vremena

Sada će biti prikazano teorijsko znanje obuhvaćeno ovim radom na praktičnim, pravim primjerima. Izmjerit će se vrijeme potrebno za probijanje različitih lozinki koristeći prvenstveno 7-zip *hash*, a po potrebi koristit će se i MD5 *hash*. 7-zip *hash* će se generirati tako da se prvo zakriptira .7z datoteka određenom lozinkom i da se programom 7z2hashcat izvuče njezin *hash*. MD5 *hashevi* će biti generirani već prije spomenutim *online* MD5 generatorom.[43] Nasumično odabrane lozinke (engl. *random*) bit će, po potrebi, generirane KeePass programom.

[42] Izvor: Shiva Houshmand Yazdi; Analyzing Password Strength & Efficient Password Cracking; Florida State University Libraries (2011).
[43] Izvor: https://www.md5hashgenerator.com/.

4.3.1. Lozinke do pet znakova – napad sirovom snagom

Da bi se uspješno probila lozinka za koju znamo da ima do pet znakova (uključujući i peti), moraju se permutirati svi znakovi u toj lozinci te isto tako krenuti od prvog znaka pa sve do petog znaka. Opcije u *hashcatu* koje služe za povećavanje broja znakova (od – do) su `--increment-min` i `--increment-max` (uz obavezno opciju `-i` koja govori da će se koristiti inkrementiranje). Dakle, ako bi se primjerice tražile lozinke od dva do četiri znaka, `--increment-min` bi bio dva, a `--increment-max` bi bio četiri. *Hashcat* će u tom slučaju krenuti probijanje pokušavajući prvo znakove, aa, pa ab, ac, ad itd., te će potom proći i sva velika slova, sve brojeve i sve specijalne znakove. Nakon što sve izvrti, krenut će aaa, aab itd., sve do duljine četiriju znakova. Ako se stavi samo opcija `-i`, *hashcat* će automatski inkrementirati znakove počevši od 1, do duljine znakova određene maskom.

Primjer 1.

Za prvi najjednostavniji primjer lozinka će biti engleska riječ „**Cat**“ koja se sastoji od tri znaka – prvo slovo veliko i dva mala slova. Ovaj će primjer biti malo detaljnije opisan, a svi sljedeći primjeri bit će kraći uz fokus na mjerenje vremena probijanja lozinke.

Kreirana je datoteka Cat.txt i zakriptirana u 7-zip *file* koji se zove secret_file_1.7z. Programom 7z2hashcat64 dobiven je *hash* i spremljen je u datoteku secret_file_1_hash.txt. Ovo će i u svim daljnjim primjerima biti nomenklatura imenovanja datoteka (1, 2, 3 itd). Izlazna datoteka u koju će se spremati pronađena lozinka zove se secret_file_password.txt.

Dobiveni *hash* je sljedeći:

$7z$0$19$0$$8$687caa944740a7c60000000000000000$2874563440$64$58$3cc430272a9253605518fe668aa1af46a558c3d2e9bd80498be7afa6385cd32772e837155e6f7434527ae91f3a47d1ae9118ad5c9b4bfc4107256ff5f89d08f5

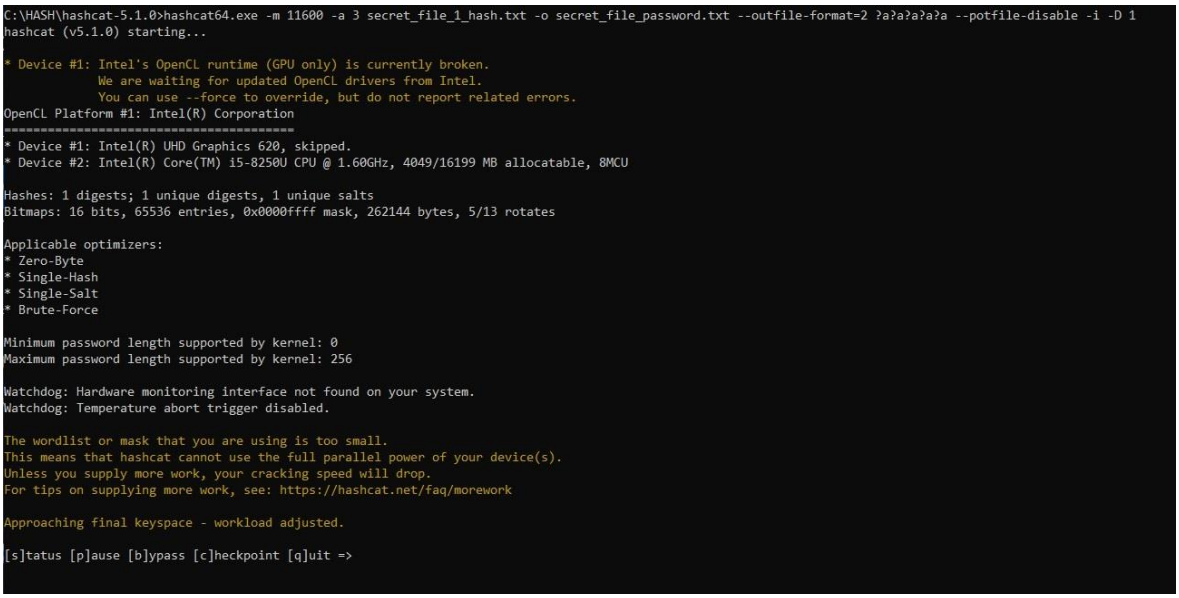
Cijela naredba kojom se probija ova lozinka (na slabijem računalu koje koristi CPU) napadom sirovom snagom je sljedeća:

```
hashcat64.exe -m 11600 -a 3 secret_file_1_hash.txt -o secret_file_password.txt --outfile-format=2 ?a?a?a?a?a --potfile-disable -i -D 1
```

?a – slovo „a“ u maski, označava „sve“ (engl. *all*), a pod time *hashcat* podrazumijeva sva velika i sva mala slova, brojeve 0 – 9 te sve specijalne znakove.

Duljina maske je pet, što je određeno opcijom ?a?a?a?a?a, te s obzirom na to da je korištena i opcija `-i`, *hashcat* će sam automatski krenuti od jednog znaka, sve dok ili ne nađe lozinku ili dok ne potroši cijeli prostor ključa koji je određen maksimalnom duljinom lozinke, u ovom slučaju od pet znakova.

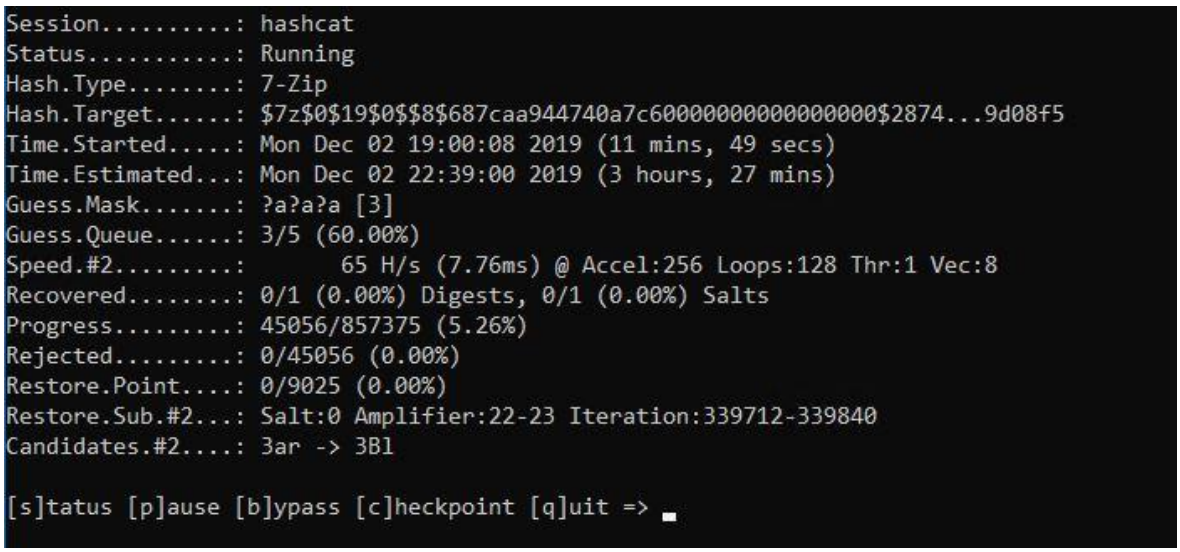
Na donjoj je slici prikazano kako izgleda početak probijanja ove lozinke.



Slika 4.11. Probijanje lozinke „Cat“ na slabijem računalu napadom sirovom snagom

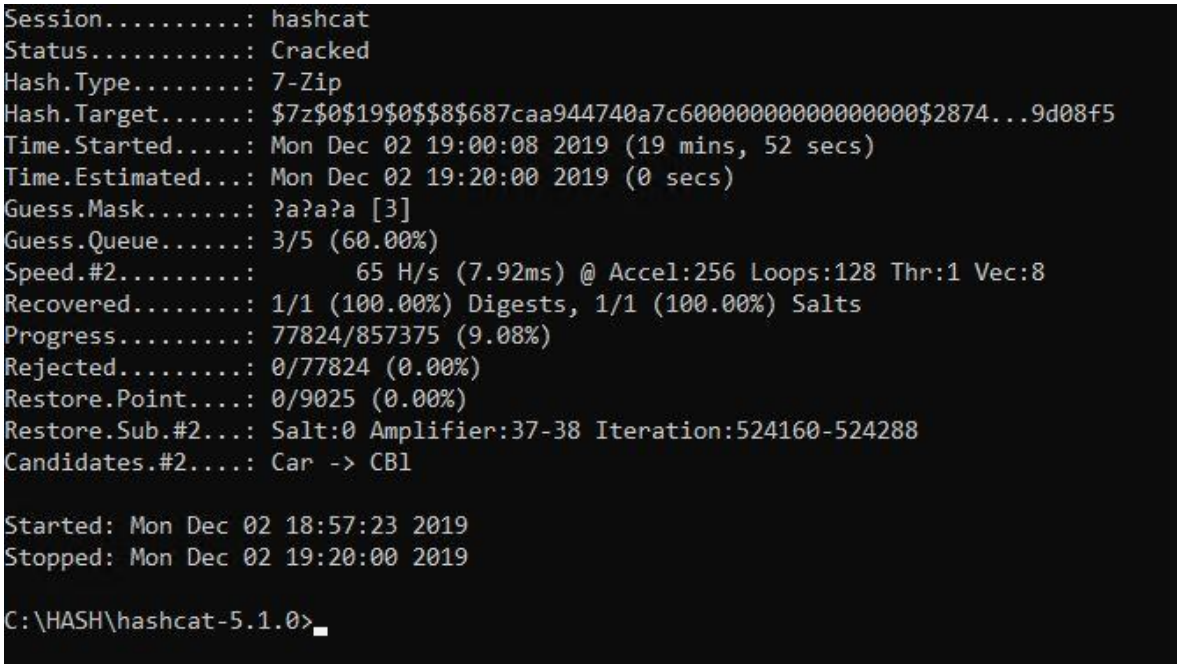
Procjena je da će slabije računalo uspješno probiti lozinku koristeći napad sirovom snagom. Naravno, s obzirom na to da je korištena riječ „Cat“, to prilikom napada sirovom snagom ne utječe na jednostavnost napada jer *hashcat* i dalje mora permutirati sve brojeve i sve specijalne znakove. Da je korišten napad rječnikom, stvar bi bila potpuno drugačija, što će i biti pokazano u sljedećem poglavlju.

Nakon otprilike jedanaest minuta koliko *hashcat* radi na probijanju ove lozinke, status je sljedeći.



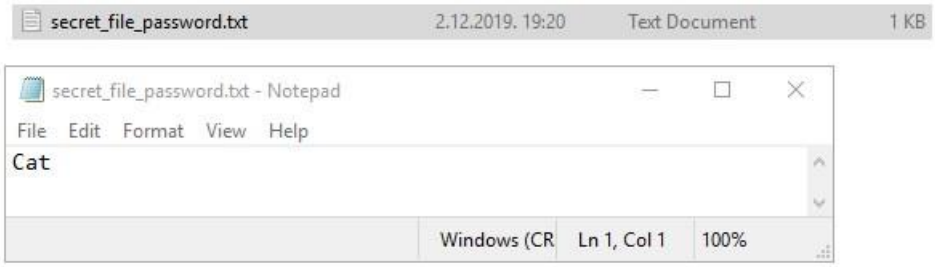
Slika 4.12. *Hashcat* radi na probijanju lozinke

Ukupno vrijeme *hascata* za probijanje lozinke iznosi nešto manje od 23 minute.



Slika 4.13. Uspješno probijena lozinka „Cat“ na slabijem računalu

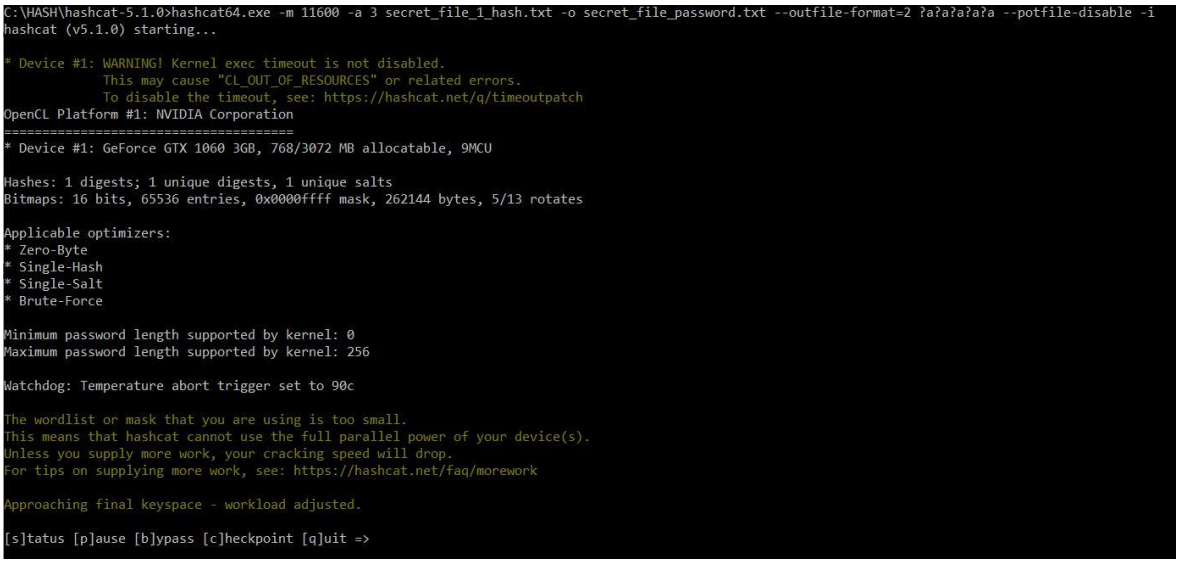
Kreirana je i ispravna datoteka u koju je spremljena pronađena lozinka.



Slika 4.14. Pronađena lozinka spremljena u datoteku

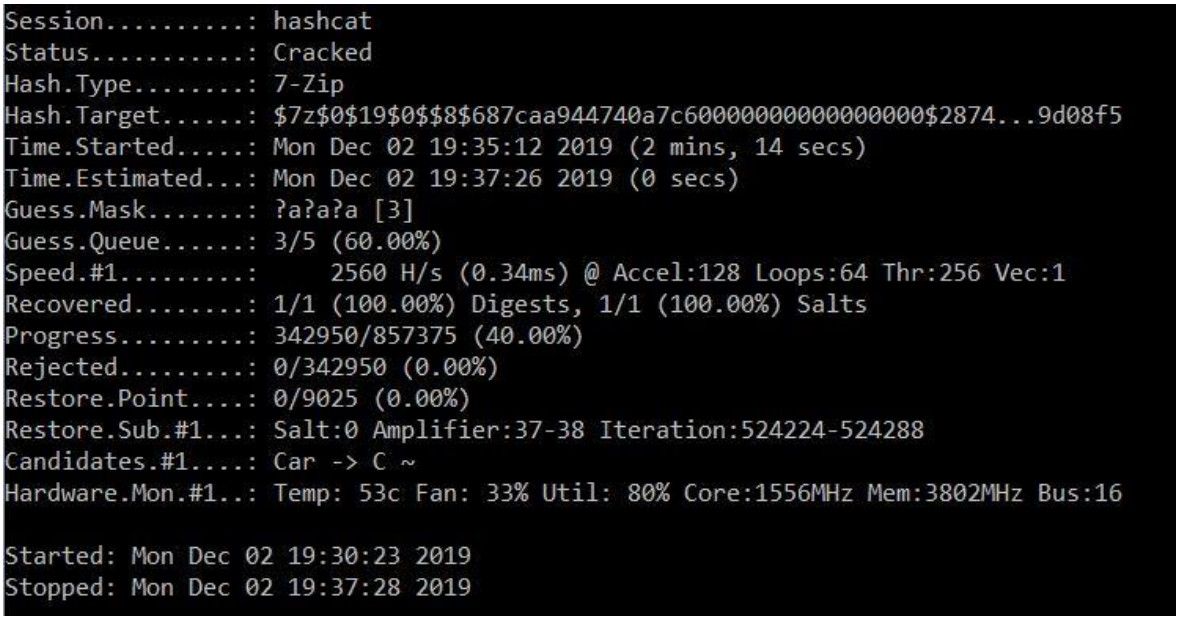
Slijedi prikaz potrebnog vremena da se ista ova lozinka probije na jačem računalu. Koristit će se isti *hash* – sve isto. Naredba koja će se koristiti na jačem računalu izgleda gotovo identično, jedino je izbačena opcija `-D 1` – ta opcija će se koristiti isključivo na slabijem računalu da se na njemu iskoristi CPU. Na jačem računalu koristit će se GPU.

Ovako izgleda početak probijanja lozinke na jačem računalu.



Slika 4.15. Probijanje lozinke „Cat“ na jačem računalu napadom sirovom snagom

Ukupno vrijeme za probijanje ove lozinke na jačem računalu iznosi 7 minuta i 5 sekundi.



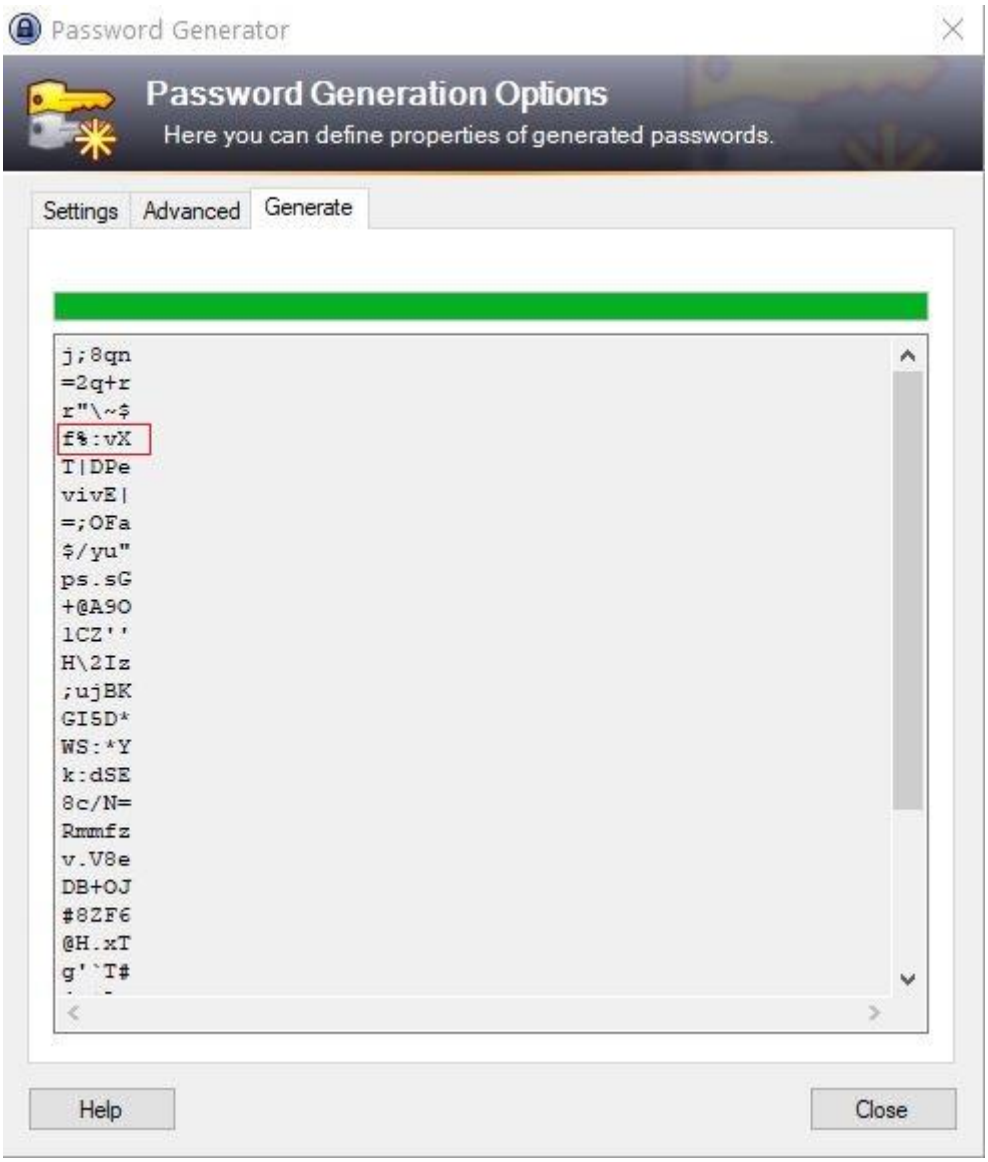
Slika 4.16. Uspješno probijena lozinka „Cat“ na jačem računalu

Prosječna brzina kojom su računani *hashevi* na ovom računalu je 2560 *hasheva* u sekundi.

U sljedećem primjeru bit će prikazano probijanje kompleksnije lozinke – ovoga puta od kompletnih pet znakova, a sama lozinka bit će nasumično generirana. Računalo će biti pušteno da radi tijekom noći, a nakon što se uspješno ili neuspješno probije lozinka, pokrenut će se isto probijanje na jačem računalu. Maksimalno vrijeme koje je određeno da bi demonstracija bila uspješna je deset sati.

Primjer 2.

Koristeći KeePass, odabrana slučajno generirana lozinka od pet znakova je: „**f%:vX**“.



Slika 4.17. Generirana slučajno odabrana lozinka

Hash koji je dobiven kriptiranjem .7z datoteke s gornjom lozinkom je sljedeći:

$7z$0$19$0$$8$74d3c506b9d26d6e0000000000000000$3181483066$80$74$c11ea24af542e1631c9a97e1b530f12c2ac288cf631a7bb7ddf36fb86ff5bb6046f533da8e87ea2f845e782e5ac836817fc953680ec57154f0692cb545c9cdca783a5151b5d063c7ec63b417da8367dd

Hashcat naredbe su gotovo identične, osim što se za *hash* datoteku koristi naziv secret_file_2_hash.txt.

```
Session..........: hashcat
Status...........: Quit
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$74d3c506b9d26d6e0000000000000000$3181...8367dd
Time.Started.....: Tue Dec 03 00:02:13 2019 (8 hours, 5 mins)
Time.Estimated...: Wed Dec 18 00:27:50 2019 (14 days, 16 hours)
Guess.Mask.......: ?a?a?a?a [4]
Guess.Queue......: 4/5 (80.00%)
Speed.#2.........:       63 H/s (7.66ms) @ Accel:256 Loops:128 Thr:1 Vec:8
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 1828864/81450625 (2.25%)
Rejected.........: 0/1828864 (0.00%)
Restore.Point....: 18432/857375 (2.15%)
Restore.Sub.#2...: Salt:0 Amplifier:38-39 Iteration:230784-230912
Candidates.#2....: Jesh -> J@ND

Started: Mon Dec 02 20:20:48 2019
Stopped: Tue Dec 03 08:08:02 2019
```

Slika 4.18. Lozinka „f%:vX“ nije probijena na slabijem računalu

Nažalost, nakon 11 sati i 47 minuta lozinka nije bila ni blizu probijanja te je zaustavljen rad *hashcata*. Slika 4.18. pokazuje da se nakon ovoliko vremena uspjelo doći tek do maske od četiri znaka. Procijenjeno vrijeme koje *hashcat* prikazuje je četrnaest i više dana, što uz uvjet koji je postavljen (unutar deset sati), čini ovaj pokušaj probijanja neuspješnim.

Probijanje iste lozinke sada će se pokušati jačim računalom te će rezultat biti vidljiv na sljedećoj slici.

```
Session..........: hashcat
Status...........: Quit
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$74d3c506b9d26d6e0000000000000000$3181...8367dd
Time.Started.....: Wed Dec 04 02:16:19 2019 (5 hours, 49 mins)
Time.Estimated...: Tue Dec 24 05:49:45 2019 (19 days, 21 hours)
Guess.Mask.......: ?a?a?a?a?a [5]
Guess.Queue......: 5/5 (100.00%)
Speed.#1.........:     4445 H/s (7.88ms) @ Accel:128 Loops:64 Thr:256 Vec:1
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 92897280/7737809375 (1.20%)
Rejected.........: 0/92897280 (0.00%)
Restore.Point....: 884736/81450625 (1.09%)
Restore.Sub.#1...: Salt:0 Amplifier:30-31 Iteration:358272-358336
Candidates.#1....: 7olel -> 7jM52
Hardware.Mon.#1..: Temp: 79c Fan: 45% Util:  0% Core:1873MHz Mem:3802MHz Bus:16

Started: Tue Dec 03 21:43:49 2019
Stopped: Wed Dec 04 08:05:25 2019
```

Slika 4.19. Lozinka „f%:vX“ nije probijena na jačem računalu

Nakon 10 sati i 21 minute lozinka i dalje nije probijena jačim računalom. I ovaj pokušaj probijanja lozinke je neuspješan. No, za razliku od slabijeg računala, na jačem je računalu maska uspjela doći do računanja svih pet znamenki.

S obzirom na dva neuspjela pokušaja u prethodnim primjerima, ovo je izvrsna prilika da se sada isproba MD5 *hash* i vidi je li njega moguće probiti. Naravno, lozinka je i dalje ista: „f%:vX“.

Primjer 3.

MD5 *hash* od „f%:vX“ lozinke je: bbb7c16ad34edf24ee240868f8364bf2

Ovaj *hash* će se spremiti u datoteku secret_file_3_hash.txt i prvo će se koristiti slabije računalo za pokušaj probijanja *hasha*. Vrijedi isti uvjet – *hash* mora biti probijen unutar deset sati da bi probijanje bilo uspješno. Pretpostavka je da će *hash* sigurno biti probijen, no to će biti vidljivo tek nakon mjerenja koje će biti prikazano na sljedećoj slici. Naredba za ovaj napad na MD5 *hash* koristi opciju `-m 0`, što označava MD5 i glasi ovako:

```
hashcat64.exe -m 0 -a 3 secret_file_3_hash.txt -o secret_file_password.txt --outfile-format=2 ?a?a?a?a?a --potfile-disable -i -D 1
```

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: bbb7c16ad34edf24ee240868f8364bf2
Time.Started.....: Wed Dec 04 17:59:54 2019 (58 secs)
Time.Estimated...: Wed Dec 04 18:00:52 2019 (0 secs)
Guess.Mask.......: ?a?a?a?a?a [5]
Guess.Queue......: 5/5 (100.00%)
Speed.#2.........:   113.4 MH/s (5.90ms) @ Accel:1024 Loops:95 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 6871080960/7737809375 (88.80%)
Rejected.........: 0/6871080960 (0.00%)
Restore.Point....: 72318976/81450625 (88.79%)
Restore.Sub.#2...: Salt:0 Amplifier:0-95 Iteration:0-95
Candidates.#2....: sf0z~ ->  JB9:

Started: Wed Dec 04 17:59:39 2019
Stopped: Wed Dec 04 18:00:53 2019
```

Slika 4.20. Uspješno probijena lozinka „f%:vX“ na slabijem računalu (MD5)

Razlika je izuzetno velika. *Hashcat* je uspio probiti ovu lozinku, napadom sirovom snagom, za 1 minutu i 14 sekundi. Pretpostavlja se da bi i jače računalo bez problema probilo istu lozinku. Slika 4.20. pokazuje da slabije računalo računa 113,4 milijuna MD5 *hasheva* po sekundi. Za usporedbu, kod 7-zip *hasha* ta se brojka na istom računalu kretala oko 65 *hasheva* po sekundi.

Uz jače računalo lozinka je probijena za samo deset sekundi.

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: bbb7c16ad34edf24ee240868f8364bf2
Time.Started.....: Sun Dec 15 22:25:44 2019 (3 secs)
Time.Estimated...: Sun Dec 15 22:25:47 2019 (0 secs)
Guess.Mask.......: ?a?a?a?a?a [5]
Guess.Queue......: 5/5 (100.00%)
Speed.#1.........:  3139.1 MH/s (5.51ms) @ Accel:128 Loops:95 Thr:256 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 6892093440/7737809375 (89.07%)
Rejected.........: 0/6892093440 (0.00%)
Restore.Point....: 72253440/81450625 (88.71%)
Restore.Sub.#1...: Salt:0 Amplifier:0-95 Iteration:0-95
Candidates.#1....: sT:f\ ->  ^M!>
Hardware.Mon.#1..: Temp: 39c Fan: 20% Util: 94% Core:1961MHz Mem:3802MHz Bus:16

Started: Sun Dec 15 22:25:38 2019
Stopped: Sun Dec 15 22:25:48 2019
```

Slika 4.21. Uspješno probijena lozinka „f%:vX“ na jačem računalu (MD5)

Jače računalo sposobno je obraditi 3139 milijuna MD5 *hasheva* u sekundi.

4.3.2. Lozinke od pet do deset znakova – napad rječnikom

Napad rječnicima uvelike olakšava probijanje lozinki preko pet znakova, iako nedostatak ovog napada je to da neće uvijek biti moguće pronaći lozinku. Razlog tome je da lozinka koju se pokušava probiti ne mora postojati u rječniku koji se koristi, osobito ako se koriste specijalni znakovi. Stoga, u ovom dijelu pokazivanja napada koristit će se pažljivo izabrane lozinke. Također, pokazat će se i hibridni oblik napada koji je samo bio spomenut, ali odnosi se na kombiniranje riječi iz rječnika s napadom maskom.

Primjer 4.

Odabrana lozinka za ovaj primjer bit će engleska riječ „**Wednesday1**“. Pretpostavimo da je netko odabrao riječ „srijeda“ (engl. *wednesday*) kao svoju lozinku te da bi ju „još bolje“ zaštitio, dodao je i broj 1 na kraju te riječi i stavljeno je prvo slovo veliko. Lozinka je duljine deset znakova, s time da nema specijalnih znakova. S obzirom na to da je praktički nemoguće probiti sirovom snagom 7-zip *hash* od toliko znakova (što je pokazano u primjeru broj 2), probat će se iskoristiti napad rječnikom, tj. listom riječi. Lista riječi koja će biti korištena je već poznata rockyou.txt datoteka koja sadržava velik broj lozinki koje su u nekom trenutku bile korištene. Namjerno nije prethodno pregledana rockyou.txt datoteka sadrži li lozinku „Wednesday1“ da bi primjer bio što je moguće bliže stvarnosti. Na taj način, ako datoteka i sadrži tu lozinku, to će odgovarati stvarnom pokušaju kako bi netko napao 7-zip datoteku koristeći napad rječnikom.

Naredba potrebna za ovaj napad pomoću slabijeg računala je sljedeća:

```
hashcat64.exe -m 11600 -a 0 secret_file_4_hash.txt C:\HASH\rockyou.txt -o secret_file_password.txt --outfile-format=2 --potfile-disable -D 1
```

U datoteci secret_file_4_hash.txt nalazi se 7-zip *hash* datoteke secret_file_4.7z koja je zakriptirana „Wednesday1“ lozinkom. *Hash* te datoteke je:

$7z$0$19$0$$8$a729c6c4aadb9b900000000000000000$3181483066$80$74$efd689ce95f74d263b20971ce6a1a8f476e7c9dd45d7f411d64caf1a26b5bd5940f4d1ebec6bd06825455c2dd8c527bce9aa450825a998d4fafec519317c8d0f5f63f54b3635672a16a6373788093be0

```
Session..........: hashcat
Status...........: Quit
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$a729c6c4aadb9b900000000000000000$3181...093be0
Time.Started.....: Tue Dec 03 20:48:10 2019 (11 hours, 19 mins)
Time.Estimated...: Sat Dec 07 12:09:42 2019 (3 days, 4 hours)
Guess.Base.......: File (C:\HASH\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#2.........:       46 H/s (11.99ms) @ Accel:256 Loops:128 Thr:1 Vec:8
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 1859584/14344384 (12.96%)
Rejected.........: 0/1859584 (0.00%)
Restore.Point....: 1859584/14344384 (12.96%)
Restore.Sub.#2...: Salt:0 Amplifier:0-1 Iteration:85760-85888
Candidates.#2....: dig089tuft870 -> diadems

Started: Tue Dec 03 20:48:06 2019
Stopped: Wed Dec 04 08:07:48 2019
```

Slika 4.22. Lozinka „Wednesday1“ nije probijena na slabijem računalu

Slabije računalo nakon 11 sati i 9 minuta nije uspjelo probiti lozinku koristeći napad listom riječi iz datoteke rockyou.txt. I ovaj je pokušaj neuspješan. Vidjet će se u nastavku je li jače računalo sposobno odraditi ovaj zadatak uspješno. Pokrenuta naredba je ista, osim što se ne koristi -D 1 opcija (forsiranje korištenja CPU-a) već je određeno da će se koristiti samo na slabijem računalu.

Uz iste uvjete, samo na jačem računalu, došlo se do sljedećeg rezultata.

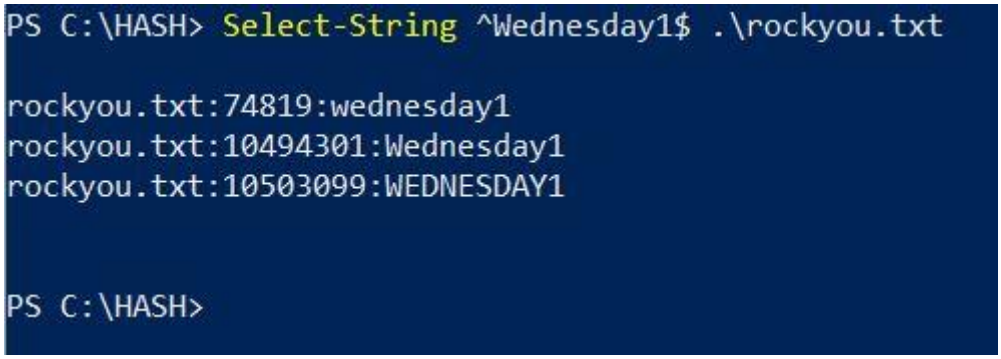
```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: 7-Zip
Hash.Target......: $7z$0$19$0$$8$a729c6c4aadb9b900000000000000000$3181...093be0
Time.Started.....: Wed Dec 04 21:17:28 2019 (3 hours, 52 mins)
Time.Estimated...: Thu Dec 05 01:10:07 2019 (0 secs)
Guess.Base.......: File (C:\HASH\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:      761 H/s (45.76ms) @ Accel:512 Loops:64 Thr:64 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 10616832/14344384 (74.01%)
Rejected.........: 0/10616832 (0.00%)
Restore.Point....: 10321920/14344384 (71.96%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:524224-524288
Candidates.#1....: ahki14 -> Sabo2008
Hardware.Mon.#1..: Temp: 66c Fan: 41% Util:  1% Core:1911MHz Mem:3802MHz Bus:16

Started: Wed Dec 04 21:17:24 2019
Stopped: Thu Dec 05 01:10:09 2019
```

Slika 4.23. Uspješno probijena lozinka „Wednesday1“ na jačem računalu

Jačem računalu trebalo je 3 sata i 52 minute da probije ovu lozinku koristeći rockyou.txt listu riječi. Ovaj primjer dobro ilustrira koliko snaga računala utječe na uspjeh probijanja lozinke.

Postavlja se pitanje je li *hashcat* stvarno pronašao pravu lozinku u listi riječi iz rockyou.txt datoteke, s obzirom na to da to još uvijek nije provjereno. Sljedeća *powershell* naredba daje odgovor na to pitanje.



Slika 4.24. Provjera postoji li „Wednesday1“ lozinka (rockyou.txt)

Vidljivo je da zaista postoji upravo ta riječ, ali nažalost slabije računalo nije uspjelo probiti tu lozinku koristeći standardni napad listom riječi. Ovo je idealna prilika da se iskoristi zadnji korak u metodologiji probijanja lozinki – prepravljanje (engl. *tweak*). Taj korak kaže da ako se ne pogodi lozinka iz prvog pokušaja, može se prepraviti napadačka tehnika i pokušati ponovo. Dat će se još jedna šansa slabijem računalu da se drugačijom tehnikom pokuša probiti ista lozinka. To će biti prikazano u sljedećem primjeru.

Primjer 5.

Sada će se pokušati drugačije pristupiti ovom napadu rječnikom / listom riječi. Očito je da unutar deset sati slabije računalo nije sposobno obraditi četrnaest milijuna riječi, koliko ima rockyou.txt datoteka. Zato će se sada koristiti druga datoteka kao rječnik – datoteka words.txt preuzeta s interneta.[44] U njoj se nalaze engleske riječi, ali ih ima oko 466 tisuća, što je manje od rockyou.txt datoteke. Isto tako, po prvi put će se koristiti jedan oblik *hashcat*

[44] Izvor: https://github.com/dwyl/english-words Datum: 4. 12. 2019.

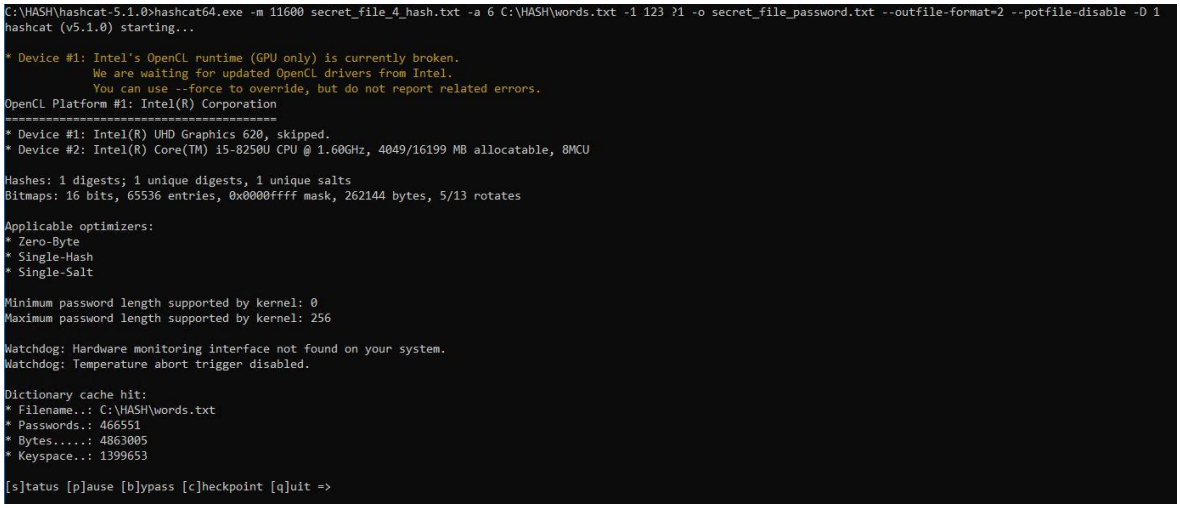
napada koji se naziva hibridni napad.[45] U ovom konkretnom slučaju, to je hibridni napad korištenjem rječnika i maske. Iskoristit će se words.txt rječnik te maskom odrediti da se na kraj svake riječi u rječniku dodaju brojevi 1, 2 i 3. Opcija u hibridnom napadu za dodavanje nekog znaka na kraj svake riječi u rječniku je opcija `-a 6`. Konkretna naredba, u kojoj se dodaje broj 1 – 3 na kraju, glasi:

```
hashcat64.exe -m 11600 secret_file_4_hash.txt -a 6 C:\HASH\words.txt -1 123 ?1 -o secret_file_password.txt --outfile-format=2 --potfile-disable -D 1
```

Datoteka words.txt isto tako ni u ovom primjeru neće biti prvotno provjerena sadrži li lozinku. U ovom slučaju prva pretpostavka je da s obzirom na to da je to rječnik engleskih riječi, on bi trebao sadržavati riječ „Wednesday“ – pisano prvim slovom velikim jer se ta riječ tako piše u engleskom jeziku. Druga pretpostavka je da lozinka koja se pokušava probiti ima broj u rasponu 1– 3 na kraju (jednu znamenku). To je bit prepravljanja načina napada – potrebno je biti maštovit i snalažljiv. Iako je ovo jednostavan primjer, i dalje je vrlo realan jer dosta ljudi stavlja samo jedan broj na kraj svoje lozinke. Obično se stavi samo broj 1, ali u ovom primjeru proširit će se da može uključivati i brojeve 2 i 3. U sljedećem poglavlju, vezanom za napade pomoću pravila, vidjet će se da se lako mogu permutirati velike količine raznih brojeva i znamenki na kraj riječi, na početak riječi, između itd. Naravno, to povećava i vrijeme napada. Ali, sada je fokus na ovom hibridnom napadu rječnikom za koji postoji mogućnost da ipak probije lozinku „Wednesday1“.

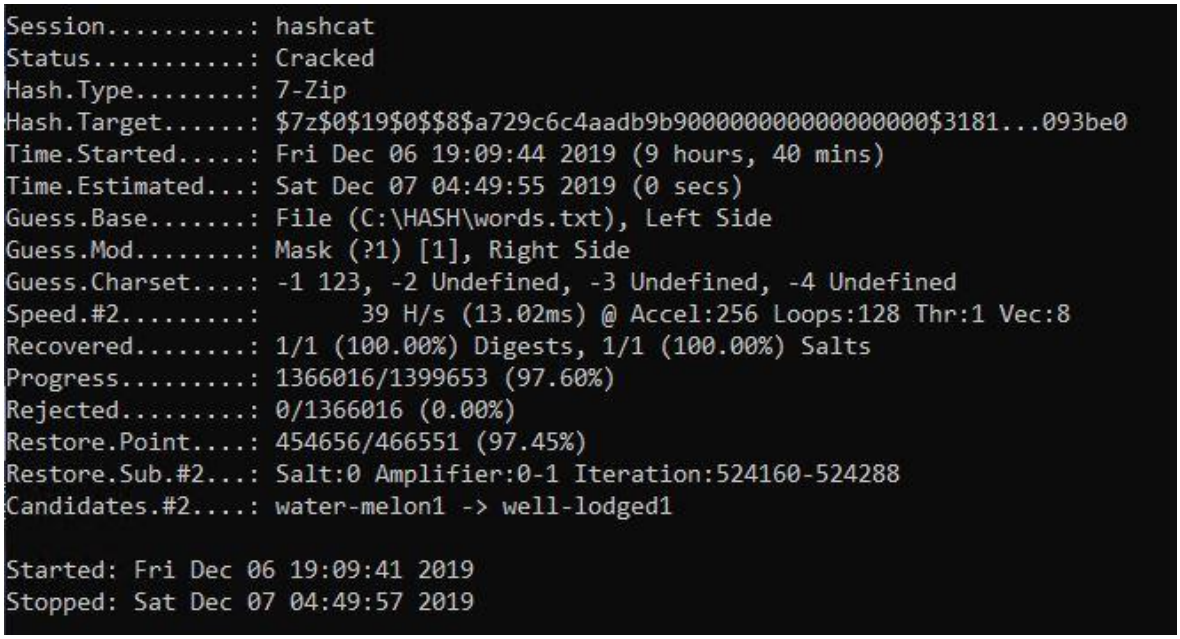
[45] Izvor: https://hashcat.net/wiki/doku.php?id=hybrid_attack Datum: 4. 12. 2019.

Na slabijem računalu pokrenuta je gornja naredba.



Slika 4.25. Pokretanje hibridnog *hashcat* napada

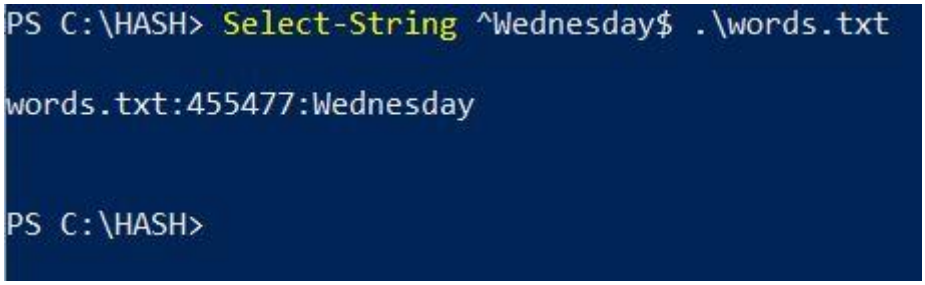
Nakon 9 sati i 40 minuta, rezultat je sljedeći.



Slika 4.26. Uspješno probijena lozinka „Wednesday1“ na slabijem računalu

Pogodak! Ovoga puta lozinka je probijena. Uz rječnik i masku bilo je potrebno nešto manje od deset sati za probijanje lozinke. Iako je bilo korišteno slabije računalo, vidljivo je iz primjera da način kojim se napada lozinka ima utjecaj na konačni uspjeh.

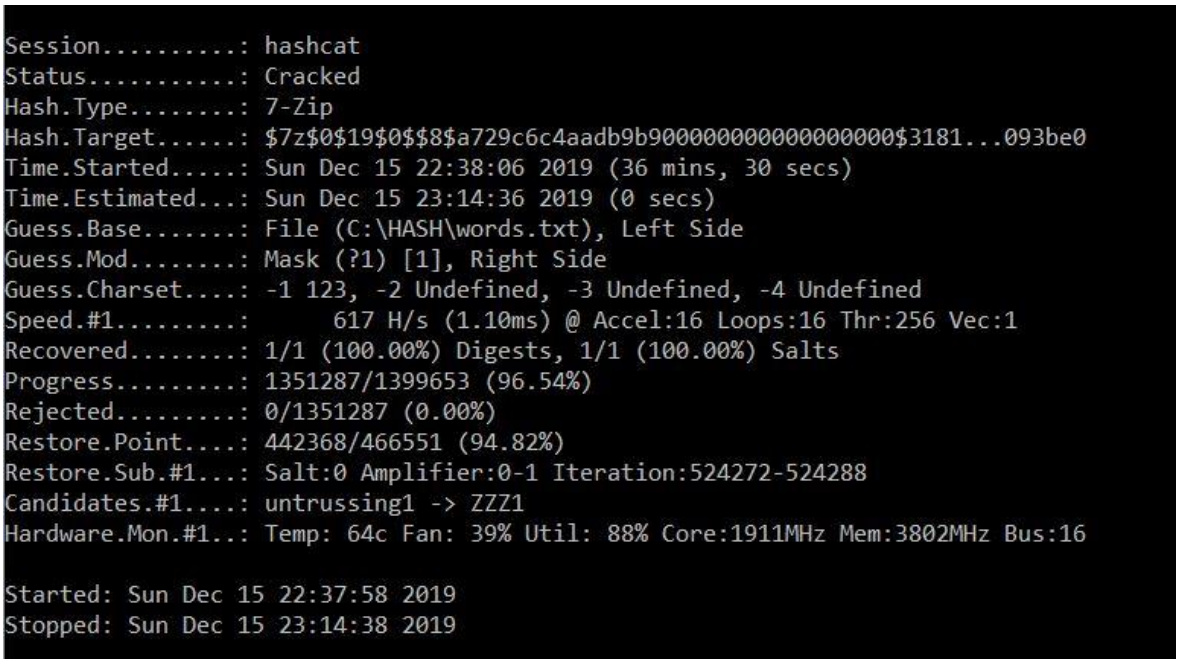
Na kraju je još potrebno provjeriti sadrži li zaista words.txt riječ „Wednesday“:



Slika 4.27. Provjera postoji li riječ „Wednesday“ (words.txt)

Provjera je pokazala da sadrži točno tu riječ, identično pisanu s prvim slovom velikim. *Hashcat* je, dakle, odlično obavio svoj posao probijanja lozinke na ovom primjeru sa slabijim računalom – ovaj put koristeći hibridni napad rječnikom.

Na jačem računalu vrijeme potrebno za probijanje ove lozinke iznosilo je 36 minuta i 40 sekundi.



Slika 4.28. Uspješno probijena lozinka „Wednesday1“ na jačem računalu

4.3.3. Lozinke od deset do petnaest znakova – napad korištenjem pravila

Napad korištenjem pravila je praktički jedina vrsta napada kojom se može (potencijalno) uspješno probiti lozinka veća od deset znakova – pod uvjetom da zadovoljava neki oblik kompleksnosti, npr. da koristi velika i mala slova, specijalne znakove itd. To je u ovim korištenim primjerima i zadovoljen uvjet, s obzirom na to da se koristi kompleksniji set znakova koji to sadrži. I u ovom napadu koristit će se pažljivo birane lozinke, ali kad god to bude moguće, bit će što bliže situacijama u stvarnom životu.

Fiktivna pretpostavka je da neka osoba (u daljnjem tekstu Marko) koristi relativno dugačke lozinke za svoje potrebe te nema samo jednu lozinku, već ih koristi desetak. Isto tako, svakih trideset dana Marko mora promijeniti sve svoje lozinke, ali s obzirom na to da mu ih nije uvijek lako zapamtiti, on dodaje razne znakove na kraj ili na početak lozinke. Također, prilikom mijenjanja lozinki, umjesto da ju kompletno promijeni, Marko je odlučio da mu je lakše ako samo poveća npr. zadnji broj te mu doda 1. Nakon nekog vremena, on zaboravi neku svoju lozinku te mora proći kroz postupak „Zaboravio sam lozinku“ i odgovarati na sigurnosna pitanja da bi ju uspio resetirati. U jednom trenutku Marko je odlučio sve svoje lozinke zapisati u tekstualnu datoteku te tu tekstualnu datoteku zakriptirati u 7-zip, sa svojom nekom već poznatom lozinkom. Nažalost, duže vrijeme nije imao potrebu otvarati tu .7z datoteku i u jednom ju trenutku više nije mogao otvoriti, a unutra je imao sve svoje bitne lozinke. Sada mu više ne pomaže, niti može negdje stisnuti link „Zaboravio sam lozinku“. Što god je isprobao upisati, nije bila lozinka kojom može otvoriti 7-zip datoteku.

Gore navedeni slučaj jedna je od realnih situacija u kojoj se netko može naći. Upravo u sličnoj situaciji našao se i autor ovog rada te je zbog navedene poteškoće i pokušaja probijanja lozinke i nastao ovaj rad. Kako se može pristupiti probijanju ovakvih lozinki, bit će prikazano u sljedećim primjerima. Za ovu svrhu koristit će se *hashcat* napad korištenjem pravila. Direktno pokušati probiti takvu lozinku, koristeći napad sirovom snagom, zahtijevalo bi vrijeme koje realno nije ostvarivo. S druge strane, napad rječnicima isto tako ne pomaže, s obzirom na to da riječi koje su se koristile za ove lozinke ne pripadaju nijednom poznatom rječniku. Ali, sam korijen riječi tih lozinki je poznat i moguće ga se prisjetiti. Korištenjem pravila unutar *hashcat* programa moglo bi se pokušati probiti takvu lozinku. U

donjoj tablici definirane su sve lozinke kojima je potencijalno zakriptirana 7-zip datoteka secret_file_6.7z.

Lozinka	Duljina lozinke
NovaGodina2019	14
#NovaGodina2018	15
B1jelaK@va123	13
BijelaK@va124	13
Dinosaur22	10
commodore64	11
C0mm0d0re128	12
!40bb1T0n!	10
St0lnaLAMPA#123	15
St0lnaLAmPA#234	15
!!Pa$$w0rd!!	12
GelzaKosu123!	13

Tablica 4.4. Lozinke za primjer napada korištenjem pravila

Riječi koje će biti spremljene u prilagođenu (engl. *custom*) listu riječi nazvanu moja_lista_6.txt prikazane su u Tablica 4.5.

moja_lista_6.txt
novagodina
bijelakava
dinosaur
commodore
hobitton
stolnalampa
password
gelzakosu

Tablica 4.5. Lista riječi moja_lista.txt

S obzirom na to da su lozinke u ovom primjeru poznate, razmatrat će se sljedeći slučaj. Da bismo pristupili probijanju lozinki iz Tablica 4.4., kreirat će se po jedna 7-zip datoteka zakriptirana sa svakom od lozinki. Iz svake 7-zip datoteke izvući će se njezin 7-zip *hash* te će se svi *hashevi* spremiti u datoteku secret_file_6_hashes.txt. Već je prije spomenuto da *hash* datoteka koja se koristi može sadržavati jedan ili više *hasheva* te će se sada koristiti više *hasheva* odjednom.

Ono što će *hashcat* raditi je sljedeće: za svaku riječ koja se dobije s pravilom koje će se koristiti izračunat će se 7-zip *hash* te će se usporediti sa svim *hashevima* spremljenima u secret_file_6_hashes.txt datoteci. Dakle, nije bitno je li u toj datoteci jedan ili više *hasheva*. Ako je samo jedan, kao u svim prijašnjim primjerima, onda se uspoređuje sa samo tim *hashom*. Ako ih je više, uspoređuje se sa svima. Ali, u ovom slučaju, to i nije toliki problem s obzirom na to da lista riječi koja će se koristiti sadrži izuzetno mali broj riječi. Sama pravila koja će se koristiti, iz te će liste riječi kreirati velik broj novih riječi kojima će se izračunavati *hash*, ali to je i dalje jako mala brojka u odnosu na primjerice rockyou.txt listu riječi.

Ako lozinka ne bude pronađena, može se koristiti i kombinacija pravila. To se radi tako da se opcije `-r` u *hashcatu* dodaju jedna za drugom, npr. `-r prvo_pravilo.txt -r drugo_pravilo.txt`. Nije potrebno koristiti samo jedno pravilo, može se i više njih odrediti jedno za drugim. Ono što se dogodi takvim kombiniranjem pravila je to da se svako pravilo, u svakoj datoteci pravila, kombinira sa svakim pravilom iz druge (ili drugih) datoteka pravila. Ovdje treba biti oprezan jer može doći do situacije da više nema slobodne memorije računala koja može obrađivati tako velike podatke, osobito ako se koriste velike liste riječi.

U svim sljedećim primjerima prvo će se koristiti jače računalo da se vidi mogu li uopće pravila koja će se koristiti probiti ove lozinke. U slučajevima gdje lozinka bude probijena, isti test će se napraviti i na slabijem računalu tako da se može dobiti predodžba o odnosu snage računala i uspješnosti probijanja lozinke. Ako lozinka (ili više njih) ne bude probijena jačim računalom, sa slabijim se neće ni pokušavati.

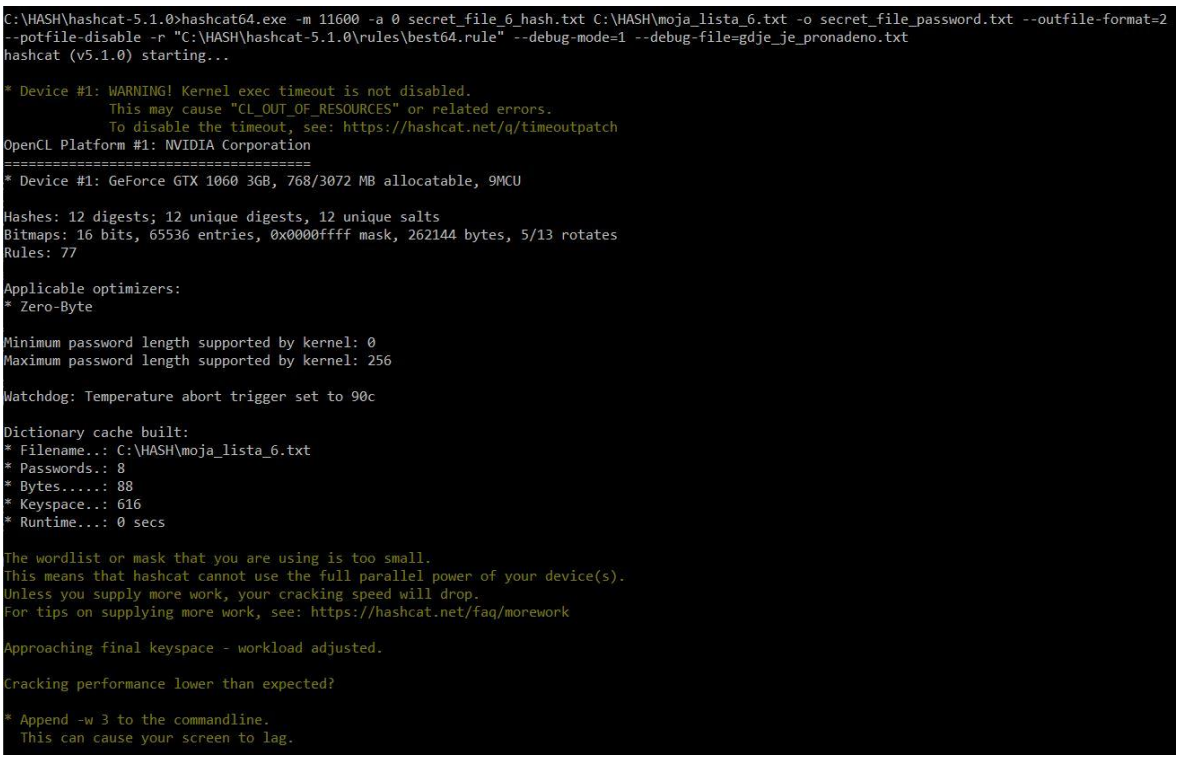
Primjer 6.

Koristit će se pravilo best64.rule, kao prvo i osnovno pravilo za probijanje lozinki. Pokazat će se i jedna nova opcija *hashcata*, a to je opcija *debug*. Ona služi za logiranje pravila koje je pronašlo lozinku (ako je pronađena) i taj se log sprema u neku datoteku. U ovom će se primjeru koristiti opcije *--debug-mode=1* i *--debug-file=gdje_je_pronadeno.txt* koje govore *hashcatu* da svaki put kada uspješno pronađe lozinku ispiše pravilo s kojim je pronađena u datoteku gdje_je_pronadeno.txt.

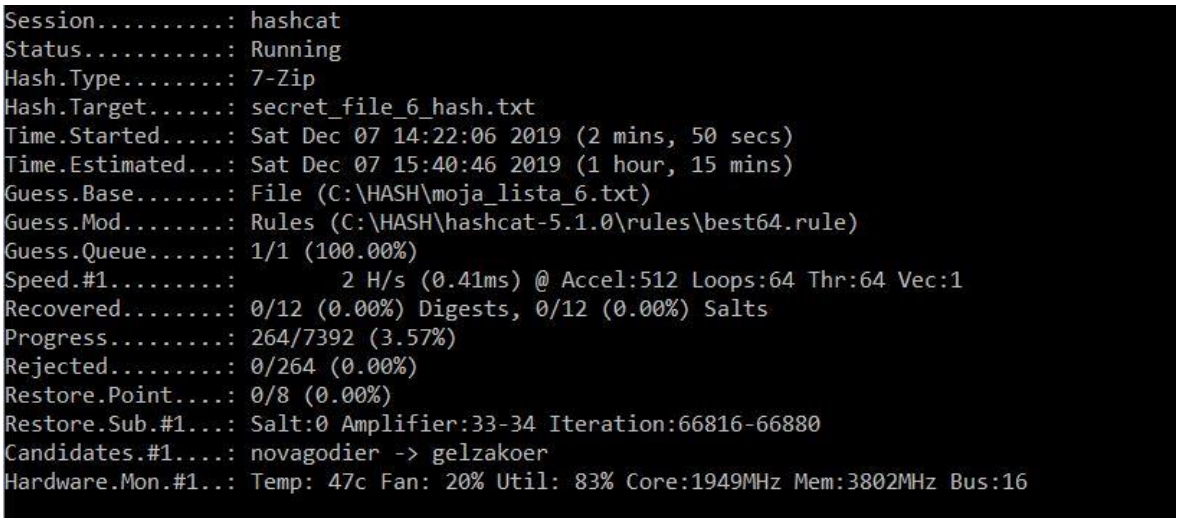
Kompletna konačna naredba koja se koristi za prvi napad pravilom na jačem računalu glasi ovako:

```
hashcat64.exe -m 11600 -a 0 secret_file_6_hash.txt C:\HASH\moja_lista_6.txt -o secret_file_password.txt --outfile-format=2 --potfile-disable -r "C:\HASH\hashcat-5.1.0\rules\best64.rule" --debug-mode=1 --debug-file=gdje_je_pronadeno.txt
```

Naredba je poprilično velika, ali zorno prikazuje neke od opcija koje ima *hashcat*. Ovako izgleda pokretanje te naredbe na jačem računalu.



Slika 4.29. Početak napada korištenjem pravila best64.rule



Slika 4.30. *Hashcat* je krenuo probijati lozinke koristeći best64.rule pravilo

Pod „Candidates.#1“ vidi se trenutna permutacija potencijalne lozinke koju *hashcat* koristi pravilom best64.rule. Procijenjeno vrijeme za probijanje je 1 sat i 15 minuta.

Nakon 1 sata i 4 minute *hashcat* je iscrpio sve mogućnosti i nije pronađena nijedna lozinka. Status je „Iscrpljeno“ (engl. *Exhausted*).

```
Session..........: hashcat
Status...........: Exhausted
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Sat Dec 07 14:22:06 2019 (1 hour, 4 mins)
Time.Estimated...: Sat Dec 07 15:26:18 2019 (0 secs)
Guess.Base.......: File (C:\HASH\moja_lista_6.txt)
Guess.Mod........: Rules (C:\HASH\hashcat-5.1.0\rules\best64.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:        2 H/s (0.12ms) @ Accel:512 Loops:64 Thr:64 Vec:1
Recovered........: 0/12 (0.00%) Digests, 0/12 (0.00%) Salts
Progress.........: 7392/7392 (100.00%)
Rejected.........: 0/7392 (0.00%)
Restore.Point....: 8/8 (100.00%)
Restore.Sub.#1...: Salt:11 Amplifier:76-77 Iteration:524224-524288
Candidates.#1....: nodina -> gkosug
Hardware.Mon.#1..: Temp: 53c Fan: 34% Util: 56% Core:1936MHz Mem:3802MHz Bus:16

Started: Sat Dec 07 14:22:02 2019
Stopped: Sat Dec 07 15:26:19 2019
```

Slika 4.31. Neuspješno pronađena lozinka

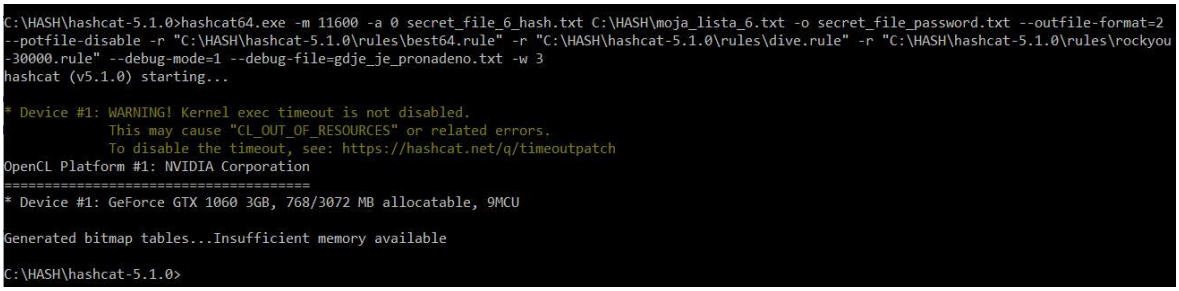
Pravilo best64.rule samo po sebi nije dovoljno da bi se pronašla lozinka. U sljedećem primjeru dodat će se još pravila da bi se prostor ključa povećao te da bi se pronašla barem jedna lozinka.

Primjer 7.

I dalje će se koristiti isključivo samo jače računalo, ali ovoga puta uključit će se još jedna opcija unutar *hashcata* – opcija -w 3. Opcija -w služi za određivanje profila opterećenja napada (engl. *workload profile*), tj. što je veće opterećenje, to se više koriste resursi procesora ili grafičke kartice za probijanje lozinke. Profil opterećenja označen brojem 3 pojačava opterećenje na grafičku karticu na opciju „jako“ (engl. *high*). Pravila koja će se kombinirati zajedno s pravilom best64.rule bit će isto tako *hashcat* ugrađena pravila. Koristit će se dodatno još dva pravila, a to su dive.rule i rockyou-30000.rule. Ovo bi trebalo znatno opteretiti GPU koji se koristi na jačem računalu.

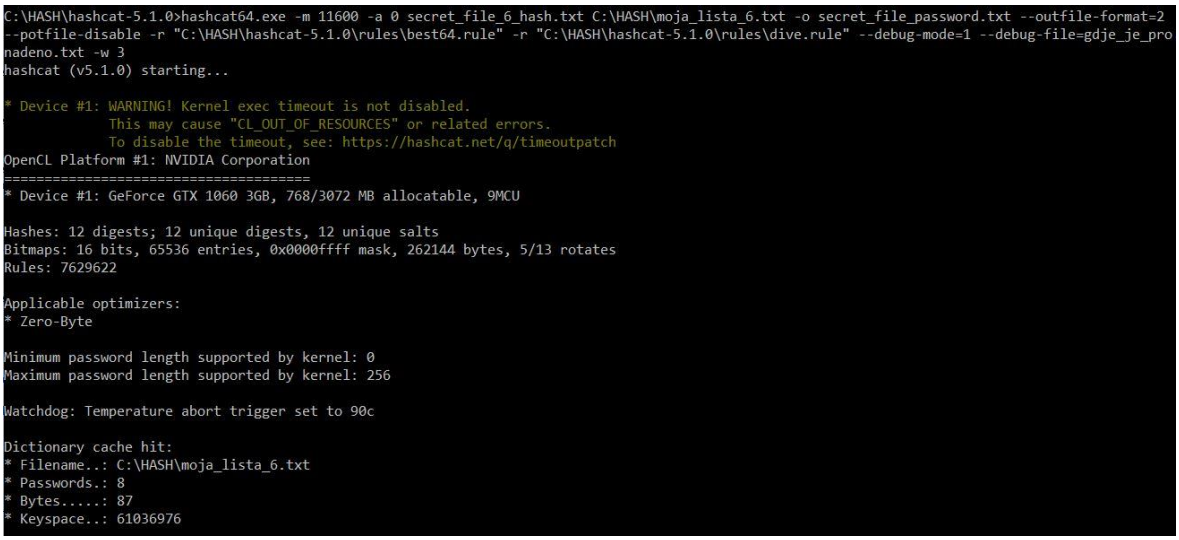
Naredba prikazana na slici sada izgleda ovako:

```
hashcat64.exe -m 11600 -a 0 secret_file_6_hash.txt C:\HASH\moja_lista_6.txt -o secret_file_password.txt --outfile-format=2 --potfile-disable -r "C:\HASH\hashcat-5.1.0\rules\best64.rule" -r "C:\HASH\hashcat-5.1.0\rules\dive.rule" -r "C:\HASH\hashcat-5.1.0\rules\rockyou-30000.rule" --debug-mode=1 --debug-file=gdje_je_pronadeno.txt -w 3
```



Slika 4.32. Nedovoljno memorije za pokretanje napada

Nažalost, nema dovoljno memorije da bi se pokrenuo ovakav napad. Izbacit će se jedno pravilo te će se pokušati koristiti samo best64.rule u kombinaciji s dive.rule.



Slika 4.33. Uspješno pokrenut napad dvama pravilima

Sada je napad uspješno pokrenut. Donja slika prikazuje trenutni status s procijenjenim vremenom.

```
Session..........: hashcat
Status...........: Running
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Sat Dec 07 15:53:42 2019 (1 min, 31 secs)
Time.Estimated...: Fri Apr 29 18:36:39 2033 (13 years, 144 days)
Guess.Base.......: File (C:\HASH\moja_lista_6.txt)
Guess.Mod........: Rules (C:\HASH\hashcat-5.1.0\rules\best64.rule, C:\HASH\hashcat-5.1.0\rules\dive.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:        2 H/s (2.37ms) @ Accel:1024 Loops:256 Thr:64 Vec:1
Recovered........: 0/12 (0.00%) Digests, 0/12 (0.00%) Salts
Progress.........: 152/732443712 (0.00%)
Rejected.........: 0/152 (0.00%)
Restore.Point....: 0/8 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:19-20 Iteration:353792-354048
Candidates.#1....: Novagodina13 -> Gelzakosu13
Hardware.Mon.#1..: Temp: 46c Fan: 20% Util: 94% Core:1949MHz Mem:3802MHz Bus:16

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

Slika 4.34. Procijenjeno vrijeme uspješnosti napada

Vidljivo je da *hashcat* procjenjuje vrijeme potrebno za ovaj napad na trinaest godina. To, naravno, nije moguće jednostavno ostvariti, ali pustit će se da napad radi punih deset sati te će se nakon toga vidjeti je li uspješno pronađena barem jedna lozinka.

```
Session..........: hashcat
Status...........: Quit
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Sat Dec 07 15:53:42 2019 (10 hours, 15 mins)
Time.Estimated...: Sun Jun 16 17:24:44 2030 (10 years, 191 days)
Guess.Base.......: File (C:\HASH\moja_lista_6.txt)
Guess.Mod........: Rules (C:\HASH\hashcat-5.1.0\rules\best64.rule, C:\HASH\hashcat-5.1.0\rules\dive.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:        2 H/s (1.39ms) @ Accel:1024 Loops:256 Thr:64 Vec:1
Recovered........: 0/12 (0.00%) Digests, 0/12 (0.00%) Salts
Progress.........: 79520/732443712 (0.01%)
Rejected.........: 0/79520 (0.00%)
Restore.Point....: 0/8 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:9940-9941 Iteration:115712-115968
Candidates.#1....: nona3 -> geu3
Hardware.Mon.#1..: Temp: 54c Fan: 33% Util: 84% Core:1936MHz Mem:3802MHz Bus:16

Started: Sat Dec 07 15:53:36 2019
Stopped: Sun Dec 08 02:08:52 2019
```

Slika 4.35. Lozinka nije pronađena korištenjem dvaju pravila

Nakon 10 sati i 15 minuta nije pronađena nijedna lozinka uz kombinaciju dvaju pravila – best64.rule i dive.rule.

S obzirom na to da je gornji napad u sedmom primjeru bio izrazito zahtjevan i da nije dao nikakve rezultate, u sljedećem primjeru pokrenut će se još jedan napad na istom jačem računalu, ali s izmijenjenim pravilima. Pravilo best64.rule zamijenit će se pravilom koje zamjenjuje velika i mala slova. *Hashcat* standardno pri instalaciji dolazi s pet zamjenjivih (engl. *toggle*s) pravila, koja se nalaze u *rules* mapi. Pravila se zovu toggles1.rule, toggles2.rule i tako dalje, a zadnje pravilo glasi toggles5.rule. Svaki broj označava koliko će se prvih slova promijeniti iz velikog u malo i obratno. Vidljivo je da je to ograničeno na prvih pet slova. Kako su lozinke u ovom slučaju poznate, prepravit će se ovo pravilo da uključuje prvih četrnaest znakova te ako sadrže slova, bit će zamijenjena velika i mala slova. Da se ne bi ručno kreiralo ovo pravilo, iskoristit će se već postojeće napisano pravilo koje je slobodno dostupno na internetu.[46] To pravilo dolazi u datoteci toggles-lm-ntlm.rule, koja će biti preimenovana u toggle14.rule za potrebe sljedećeg primjera.

Drugo pravilo koje će se koristiti zamijenit će dive.rule pravilo i bit će ručno kreirano. To će se pravilo sastojati od sljedećeg uvjeta.

- Dodat će se brojevi od 1 do 4 na kraj svake riječi.

Datoteka koja će sadržavati ovo pravilo nazvana je ZP_custom.rule. Da bi se uspješno kreirala ova datoteka, pod ovim uvjetom – znamenke od 1 do 4 na kraju svake riječi, potrebno je dodati jako veliki broj kombinacija koje sadržavaju brojeve. Za to će se koristiti program *maskprocessor*.[47] Naredbe kojima je to postignuto su sljedeće.

```
C:\HASH\hashcat-5.1.0\maskprocessor-0.73>mp64.exe $?d > ZP_custom.rule

C:\HASH\hashcat-5.1.0\maskprocessor-0.73>mp64.exe $?d$?d >> ZP_custom.rule

C:\HASH\hashcat-5.1.0\maskprocessor-0.73>mp64.exe $?d$?d$?d >> ZP_custom.rule

C:\HASH\hashcat-5.1.0\maskprocessor-0.73>mp64.exe $?d$?d$?d$?d >> ZP_custom.rule

C:\HASH\hashcat-5.1.0\maskprocessor-0.73>
```

Slika 4.36. Ručno kreiranje pravila za dodavanje znamenki na kraj svake riječi

Ovim naredbama kreirano je pravilo koje zadovoljava gornji uvjet.

[46] Izvor: https://blog.didierstevens.com/2016/07/16/tool-to-generate-hashcat-toggle-rules/ Datum: 7. 12. 2019.
[47] Izvor: https://hashcat.net/wiki/doku.php?id=maskprocessor Datum: 7. 12. 2019.

Treće pravilo mora zadovoljavati sljedeći uvjet:

- Dodat će se do dva specijalna znaka na početak i na kraj svake riječi.

Postoji veći broj specijalnih znakova i zbog toga će se koristiti tri najčešća koja se koriste u lozinkama, a to su specijalni znakovi ljestve/*hashtag* (#), uskličnik (!) i znak minus (-).

Naredbe kojima će kreirati ovo novo pravilo, spremljeno u datoteku ZP_specijalni.rule, su sljedeće.

```
REM ------------------------------------------------------
REM Adding 2 special characters to the end and beginning of each word
REM ------------------------------------------------------

REM Jedan znak na početku
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" "^?1" > ZP_specijalni.rule

REM Jedan znak na kraju
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" $?1 >> ZP_specijalni.rule

REM Jedan znak na početak i jedan na kraj
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" "^?1$?1" >> ZP_specijalni.rule


REM Dva znaka na početku
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" "^?1^?1" >> ZP_specijalni.rule

REM Dva znaka na kraju
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" $?1$?1 >> ZP_specijalni.rule

REM Dva znaka na početku i dva znaka na kraju
C:\HASH\hashcat-5.1.0\maskprocessor-0.73\mp64.exe --custom-charset1="#!-" "^?1^?1$?1$?1" >> ZP_specijalni.rule
```

Slika 4.37. Ručno kreiranje pravila za specijalne znakove

Ne smije se zaboraviti dodati znak dvotočka (:) u ovu datoteku jer to pokriva i slučaj da se ne dodaje nijedan specijalni znak.

Četvrto pravilo koje će se koristiti je *leetspeak*[48] pravilo, koje radi permutaciju za pojedina slova u riječima. Primjerice, malo slovo „a“ bit će zamijenjeno znakom „@“. Slovo „i“ bit će zamijenjeno brojem „1“ i slično. *Leetspeak* pravila već postoje i dolaze zajedno s *hashcatom*, a nalaze se u *rules* mapi. Također, u ovom primjeru ručno će se kreirati *leetspeak* pravilo i spremiti u datoteku ZP_leet.rule.

[48] Izvor: https://www.howtogeek.com/443390/what-is-leet-speak-and-how-do-you-use-it/ Datum: 7. 12. 2019.

Ručno kreirano ZP_leet.rule pravilo sadrži sljedeće opcije (svaka u svojoj liniji):

so0, se3, sa@, si1, si!, sl1, ss$. Prvo slovo „s“ označava zamjenu (engl. *substitute*) i govori *hashcatu* da zamijeni prvi znak s drugim.

Nakon kreiranja svih potrebnih pravila pokrenuta je naredba koja će sve kombinacije svih lozinki spremiti u datoteku SVE_LOZINKE_TEST.txt. Naredba glasi ovako:

```
hashcat64.exe -a 0 C:\HASH\moja_lista_6.txt -o
SVE_LOZINKE_TEST.txt --outfile-format=2 --potfile-disable -r
"C:\HASH\hashcat-5.1.0\rules\ZP_leet.rule" -r
"C:\HASH\hashcat-5.1.0\rules\ZP_custom.rule" -r
"C:\HASH\hashcat-5.1.0\rules\toggles14.rule" -r
"C:\HASH\hashcat-5.1.0\rules\ZP_specijalni.rule" --stdout
```

Opcija `--potfile-disable` ovdje nije potrebna, ali ne smeta. Nakon pokretanja naredbe *hashcat* je odmah javio sljedeću poruku.



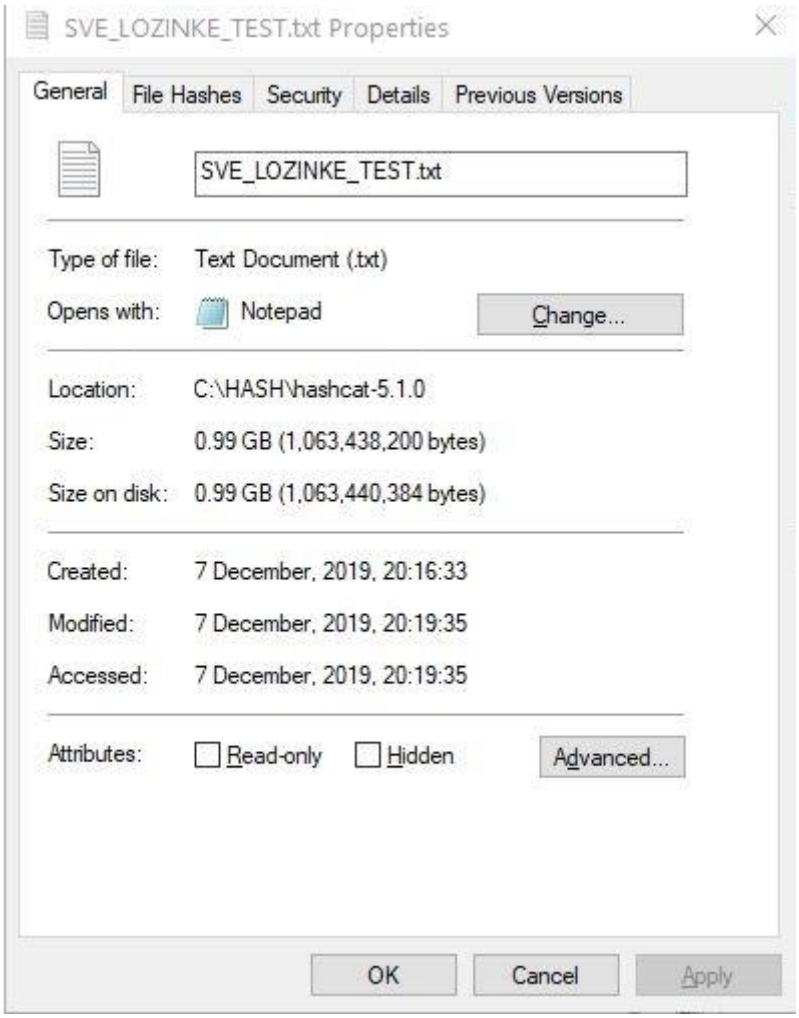
Slika 4.38. Nedostatak memorije

Potrebno je smanjiti prostor ključa, tj. količinu lozinke koju će ova pravila obrađivati. Prvo je smanjeno pravilo toggles14.rule tako da koristi samo prvih sedam znamenki i nazvano je toggles7.rule. *Hashcat* i dalje javlja poruku o nedostatku memorije. Izbačena su neka *leet* pravila i sada sadržavaju samo so0, sa@, si1, si1 i ss$. No, i dalje nema memorije. Očito je da se pokušalo koristiti previše podataka i da čak ni ovo jače računalo nema dovoljno memorije za takvu obradu. Potom su izbačeni svi četveroznamenkasti brojevi iz ZP_custom.rule pravila, što bi trebalo znatno smanjiti procesiranje. No, ni to nije pomoglo. Tek kada se koristilo samo toggles1.rule pravilo, dakle, samo da zamjenjuje prvo veliko slovo u malo i obratno, tek je tada *hashcat* uspio obraditi naredbu.



Slika 4.39. Uspješno kreirana SVE_LOZINKE_TEST.txt datoteka

Ova datoteka velika je oko 1 GB i sadrži 61 272 000 riječi, što je prikazano na Slika 4.40 i na Slika 4.41.



Slika 4.40. Veličina SVE_LOZINKE_TEST.txt datoteke

```
C:\HASH\hashcat-5.1.0>type SVE_LOZINKE_TEST.txt | find "" /c /v
61272000

C:\HASH\hashcat-5.1.0>
```

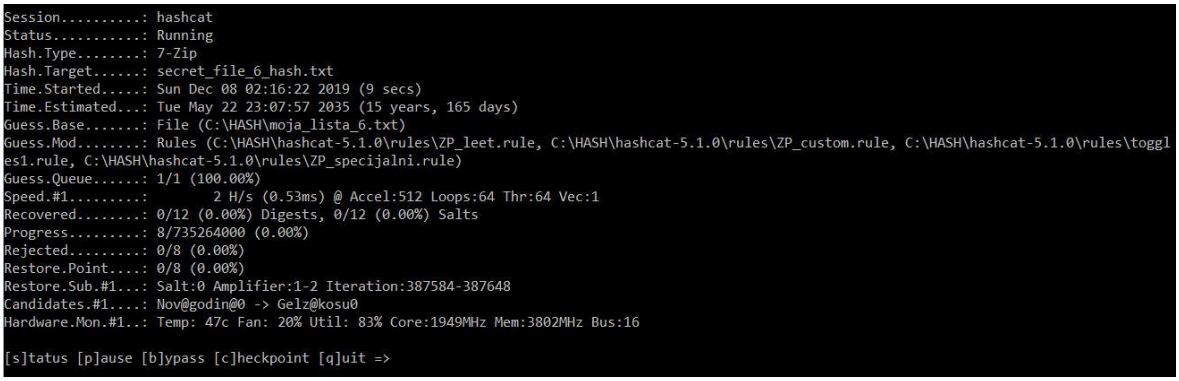
Slika 4.41. Ukupan broj riječi u SVE_LOZINKE_TEST.txt datoteci

Kada bi se koristio napad rječnikom, tj. listom riječi, ova bi lista riječi imala skoro četiri puta više riječi od rockyou.txt datoteke.

Na kraju ovog primjera pokrenut će se naredba za napad koristeći ova četiri pravila. Pretpostavka je da se neće uspjeti probiti lozinka unutar deset sati, ali pokrenut će se napad kako bi se vidjelo koliko je procijenjeno vrijeme za cijeli napad. Puna naredba sada glasi:

```
hashcat64.exe -m 11600 -a 0 secret_file_6_hash.txt C:\HASH\moja_lista_6.txt -o secret_file_password.txt --outfile-format=2 --potfile-disable -r "C:\HASH\hashcat-5.1.0\rules\ZP_leet.rule" -r "C:\HASH\hashcat-5.1.0\rules\ZP_custom.rule" -r "C:\HASH\hashcat-5.1.0\rules\toggles1.rule" -r "C:\HASH\hashcat-5.1.0\rules\ZP_specijalni.rule" --debug-mode=1 --debug-file=gdje_je_pronadeno.txt -w 4
```

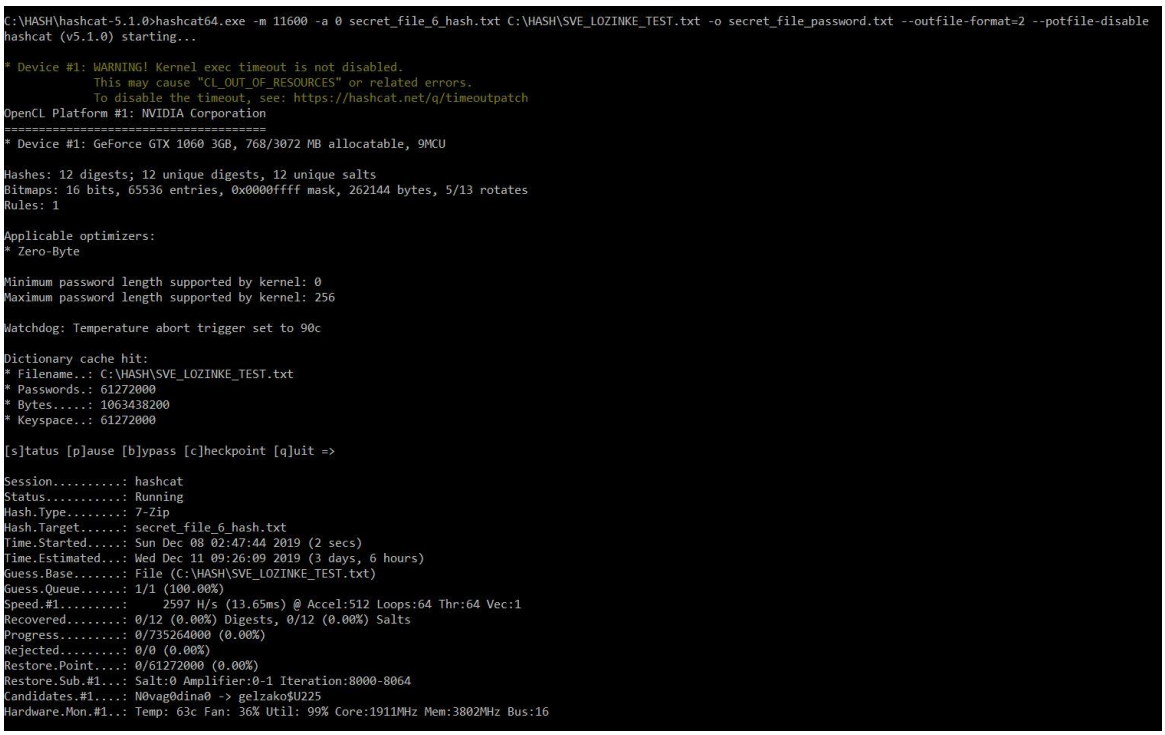
Ako se pogleda status nakon pokretanja ove naredbe, odmah je jasno da neće biti moguće koristiti ova pravila za probijanje lozinke. Procijenjeno vrijeme je petnaest i više godina, što je dovoljan indikator da se zaustavi napad.



Slika 4.42. Neuspješan pokušaj korištenja četiriju pravila

Umjesto toga, probat će se iskoristiti napad listom riječi koja je kreirana uz pomoć ovih pravila (datoteka SVE_LOZINKE_TEST.txt). Iako sljedeći napad spada pod vrstu napada rječnikom / listom riječi, on je direktno nastao kao posljedica kombiniranja četiriju različitih pravila. Kao što je već prije spomenuto, potrebne su snalažljivost i mašta. Nekad to znači i skretanje s puta ili u ovom slučaju odustajanje od jedne vrste napada i pokušaj drugom vrstom napada. Ali, još uvijek se prati metodologija probijanja lozinki i ovo je korak

prepravljanja. Nakon pokretanja naredbe za napad listom riječi, vidljivo je na Slika 4.433. da je procijenjeno vrijeme manje i da *hashcat* sada računa puno više *hasheva* po sekundi.



Slika 4.43. Promjena napada iz napada pravilima u napad listom riječi

Kako lista riječi u ovom napadu sadrži preko 61 milijun riječi, napad će potrajati. Iako možda napad neće pronaći sve lozinke unutar deset sati, ovaj put će se pustiti da napad izvršava znatno duže. Ako to i bude slučaj (da napad ne bude gotov unutar deset sati), proglasit će se neuspješnim, ali na kraju će biti vidljivo ukupno potrebno vrijeme za ovakav napad.

Nakon što je prošlo malo više od deset sati, pregledan je trenutni status ovog napada. Vidi se da je *hashcat* do sada uspio probiti dvije lozinke unutar deset sati.

```
Session..........: hashcat
Status...........: Running
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Sun Dec 08 02:47:44 2019 (10 hours, 13 mins)
Time.Estimated...: Wed Dec 11 08:57:17 2019 (2 days, 19 hours)
Guess.Base.......: File (C:\HASH\SVE_LOZINKE_TEST.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     2181 H/s (15.19ms) @ Accel:512 Loops:64 Thr:64 Vec:1
Recovered........: 2/12 (16.67%) Digests, 2/12 (16.67%) Salts
Progress.........: 95256576/735264000 (12.96%)
Rejected.........: 0/95256576 (0.00%)
Restore.Point....: 7667712/61272000 (12.51%)
Restore.Sub.#1...: Salt:11 Amplifier:0-1 Iteration:176768-176832
Candidates.#1....: -n0vag0dina856! -> -gelzako$u081!
Hardware.Mon.#1..: Temp: 73c Fan: 43% Util: 60% Core:1898MHz Mem:3802MHz Bus:16
```

Slika 4.44. Probijene dvije lozinke

Sve probijene lozinke spremaju se u datoteku secret_file_password.txt. Dvije trenutno probijene lozinke su „C0mm0d0re128“ i „commodore64“. Napad će se pustiti da radi sve dok ne prođe 50 %, iako je procijenjeno vrijeme napada još gotovo tri dana. Time će se dobiti realnija situacija koliko bi ovaj napad točno trajao i bit će prikazane i sve ostale lozinke (ako budu pronađene). Ovaj napad smatra se uspješnim za sve lozinke koje će biti pronađene.

Nakon nepunih dva dana i petnaest sati vidljivo je da su i dalje pronađene samo te dvije lozinke. Napad će sada biti zaustavljen.

```
Session..........: hashcat
Status...........: Quit
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Sun Dec 08 02:47:44 2019 (2 days, 14 hours)
Time.Estimated...: Thu Dec 12 17:54:03 2019 (2 days, 0 hours)
Guess.Base.......: File (C:\HASH\SVE_LOZINKE_TEST.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     1534 H/s (16.18ms) @ Accel:512 Loops:64 Thr:64 Vec:1
Recovered........: 2/12 (16.67%) Digests, 2/12 (16.67%) Salts
Progress.........: 415531008/735264000 (56.51%)
Rejected.........: 0/415531008 (0.00%)
Restore.Point....: 34504704/61272000 (56.31%)
Restore.Sub.#1...: Salt:5 Amplifier:0-1 Iteration:157056-157120
Candidates.#1....: #!n0vag0dina352!# -> #!gelzAko$u687!!
Hardware.Mon.#1..: Temp: 66c Fan: 41% Util:  0% Core:1898MHz Mem:3802MHz Bus:16

Started: Sun Dec 08 02:47:40 2019
Stopped: Tue Dec 10 17:39:26 2019
```

Slika 4.45. Napad zaustavljen nakon dva dana i petnaest sati

Primjer 8.

Na kraju, u zadnjem primjeru, smanjit će se intenzitet napada i koristit će se dva pravila – best64.rule i toggle1.rule. Jedna od lozinki koja se koristi i koja je poznata je lozinka „Dinosaur22“. Prvo slovo je veliko. Ovakav način napada trebao bi probiti tu lozinku kombinirajući ova dva pravila.

Naredba glasi:

```
hashcat64.exe -m 11600 -a 0 secret_file_6_hash.txt C:\HASH\moja_lista_6.txt -o secret_file_password.txt --outfile-format=2 --potfile-disable -r "C:\HASH\hashcat-5.1.0\rules\best64.rule" -r "C:\HASH\hashcat-5.1.0\rules\toggles1.rule" --debug-mode=1 --debug-file=gdje_je_pronadeno.txt -w 3
```

Nakon otprilike tri sata i trideset minuta lozinka je probijena, a cjelokupni napad pušten je da traje do kraja. Nije pronađena nijedna druga lozinka.

```
Session..........: hashcat
Status...........: Exhausted
Hash.Type........: 7-Zip
Hash.Target......: secret_file_6_hash.txt
Time.Started.....: Tue Dec 10 18:05:28 2019 (13 hours, 9 mins)
Time.Estimated...: Wed Dec 11 07:14:37 2019 (0 secs)
Guess.Base.......: File (C:\HASH\moja_lista_6.txt)
Guess.Mod........: Rules (C:\HASH\hashcat-5.1.0\rules\best64.rule, C:\HASH\hashcat-5.1.0\rules\toggles1.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:        2 H/s (0.45ms) @ Accel:1024 Loops:256 Thr:64 Vec:1
Recovered........: 1/12 (8.33%) Digests, 1/12 (8.33%) Salts
Progress.........: 110880/110880 (100.00%)
Rejected.........: 0/110880 (0.00%)
Restore.Point....: 8/8 (100.00%)
Restore.Sub.#1...: Salt:11 Amplifier:1154-1155 Iteration:524032-524288
Candidates.#1....: nodina -> gkosug
Hardware.Mon.#1..: Temp: 54c Fan: 34% Util: 86% Core:1936MHz Mem:3802MHz Bus:16

Started: Tue Dec 10 18:05:23 2019
Stopped: Wed Dec 11 07:14:39 2019
```

Slika 4.46. Lozinka „Dinosaur22“ probijena

U datoteci gdje_je_pronadeno.txt pojavila se i linija koja govori koje je pravilo iskorišteno kako bi lozinka bila pronađena. Zapis je ovakav: $2 $2 T0.

Zaključak

Analizom i mjerenjima obavljenim u ovom radu došlo se do nekoliko bitnih činjenica vezanih za probijanje lozinka. Te se činjenice mogu razvrstati u pet različitih podjela koje se međusobno nadopunjavaju i isprepliću, a jasno pokazuju odnose između uspješnosti probijanja lozinki.

Navedene podjele su:

- duljina lozinke
- kompleksnost lozinke (koriste li se specijalni znakovi, nepoznate riječi i sl.)
- vrsta *hasha*
- snaga računala
- vrsta napada.

Logično se nameće spoznaja da što je duljina lozinke veća, to će ju teže biti probiti. Ali, to ništa ne znači ako je ta ista lozinka obična riječ iz nekog rječnika. Primjerice, nikakvu sigurnost neće predstavljati ako je odabrana lozinka riječ „otorinolaringologija“ koja je duljine dvadeset znakova. *Hashcat*, uz pomoć napada rječnikom, bez problema može pronaći tu lozinku s obzirom na to da se ona nalazi u hrvatskom rječniku.

Kompleksnost lozinke kao druga podjela ovdje ima važnu ulogu. Ako se izmijene neka slova u toj riječi ili ako se dodaju brojevi i specijalni znakovi umjesto nekih slova, napad rječnikom će apsolutno biti bezuspješan. U tom slučaju može se pokušati napad sirovom snagom, ali kao što je pokazano u ovom radu, takav napad na lozinku duljine dvadeset znakova može potrajati poprilično dugo.

Također, pokazano je da je vrsta *hasha* izuzetno važna u uspješnosti probijanja lozinke. Neki *hashevi*, kao što je MD5, su lakši za probijanje, dok je 7-zip *hash*, koji je bio analiziran u ovom radu, izrazito teško probiti. 7-zip datoteke zakriptirane su s AES-256 enkripcijom i može se sa sigurnošću zaključiti da je takvu enkripciju, uz korištenje dobre lozinke, praktički nemoguće probiti.

Za uspješno probijanje lozinki od velike je važnosti i snaga samog računala kojim se pokušava probiti lozinka. *Hashcat* može koristiti i CPU računala, ali isto tako i GPU, što može biti puno snažnije.

Zadnja podjela, koja govori o uspješnosti ili neuspješnosti probijanja lozinki, odnosi se na samu vrstu napada koji se koristi. Taj dio ovisi o izboru samog napadača. Dok jedna vrsta napada, npr. napad sirovom snagom, neće u realnom vremenu moći probiti lozinku zaštićenu AES-256 enkripcijom, istu tu lozinku napad rječnikom ili napad pravilom može bez problema probiti. Za uspješno probijanje lozinke potrebno je odrediti i ispravan način napada, a to dolazi s iskustvom samog napadača.

U Tablica 0.1. prikazane su sve lozinke obrađene u ovom radu na sažet i lako razumljiv način.

Lozinka	**Duljina lozinke**	**Kompleksnost lozinke**	**Vrsta *hasha***	**Snaga računala**	**Vrsta napada**	**Vrijeme**	**Uspješnost probijanja**
test	4	mala	7-zip	jače	napad maskom (četiri znamenke)	1 min i 7 s	DA
test	4	mala	7-zip	jače	napad listom riječi (rockyou.txt)	5 min i 29 s	DA
test	4	mala	7-zip	jače	napad listom riječi (english3.txt)	5 min i 18 s	DA
Cat	3	mala	7-zip	slabije	napad maskom (inkrementirajući)	22 min i 37 s	DA
Cat	3	mala	7-zip	jače	napad maskom (inkrementirajući)	7 min i 5 s	DA
f%:vX	5	velika	7-zip	slabije	napad maskom (inkrementirajući)	10+ sati	NE
f%:vX	5	velika	7-zip	jače	napad maskom (inkrementirajući)	10+ sati	NE
f%:vX	5	velika	MD5	slabije	napad maskom (inkrementirajući)	1 m i 14 s	DA
Wednesday1	10	srednja	7-zip	slabije	napad listom riječi (rockyou.txt)	10+ sati	NE
Wednesday1	10	srednja	7-zip	jače	napad listom riječi (rockyou.txt)	3 h i 52 m	DA
Wednesday1	10	srednja	7-zip	slabije	hibridni (lista riječi words.txt + maska)	9 h i 40 m	DA
NovaGodina2019	14	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
#NovaGodina2018	15	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
B1jelaK@va123	13	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
BijelaK@va124	13	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
Dinosaur22	10	srednja	7-zip	jače	hibridni (pravila i liste riječi)	< 10 sati	DA
commodore64	11	srednja	7-zip	jače	hibridni (pravila i liste riječi)	< 10 sati	DA
C0mm0d0re128	12	velika	7-zip	jače	hibridni (pravila i liste riječi)	< 10 sati	DA
!40bb1T0n!	10	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
St0lnaLAMPA#123	15	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
St0lnaLAmPA#234	15	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
!!Pa$$w0rd!!	12	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE
GelzaKosu123!	13	velika	7-zip	jače	hibridni (pravila i liste riječi)	> 10 sati	NE

Tablica 0.1. Sažet prikaz svih korištenih lozinki

7-zip *hash* i lozinke do duljine pet znakova moguće je probiti napadom sirovom snagom. Ali, sve više od pet znakova postaje puno teže. Za lozinke duljine od pet do deset znakova moraju se koristiti neke druge vrste napada, kao što je napad rječnikom, a za lozinke preko deset znakova (isto 7-zip *hash*), mora se inventivno koristiti neki oblik napada korištenjem pravila.

Dokle god je lozinka kreirana na „ispravan“ način, šanse da se uopće probije su minimalne. Taj „ispravan“ način kreiranja lozinke je i glavni cilj ovog rada – da pruži čitatelju na pravim i stvarnim primjerima temeljna znanja o kreiranju dobre lozinke te pojam koliko bi vremena bilo potrebno da se neka lozinka probije ovisno o njezinoj duljini.

Nakon svih obrađenih pokušaja probijanja lozinke putem *hashcat* programa zaključak je: kompleksne lozinke duljine preko deset znakova zaštićene adekvatnom enkripcijom ne mogu biti jednostavno probijene niti sa snažnim računalima današnjice. Upravo to neka i bude smjernica u kreiranju sigurne lozinke.

Popis kratica

MD5	*Message Digest 5*	algoritam sažetka 5
SHA	*Secure Hashing Algorithm*	sigurni algoritam sažimanja
CRC	*Cyclic Redundancy Check*	ciklička kontrola pogrešaka
AES	*Advanced Encryption Standard*	napredni enkripcijski standard
Wiki	*Wikipedia*	slobodna enciklopedija
PIN	*Personal Identification Number*	osobni identifikacijski broj
CPU	*Central Processing Unit*	središnja procesorska jedinica
GPU	G*raphics Processing Unit*	grafička procesorska jedinica
OpenCL	*Open Computing Language*	otvoreni računalni jezik
AMD	*Advanced Micro Devices*	napredni mikroуređaji
DDR	*Double Data Rate*	dupla podatkovna brzina
MH	*Mega hash*	milijun *hasheva*
GB	*Gigabyte*	milijardu bajtova
NTLM	*New Technology LAN Manager*	LAN menadžer nove tehnologije

Popis slika

Slika 2.1. Prikaz *powershell* naredbe za ispis *hasha* datoteke 5

Slika 2.2 Korištenje MD5 algoritma u *powershell* naredbi 5

Slika 2.3. Opcija za generiranje lozinke unutar KeePass programa 7

Slika 2.4. Opcije za generiranje lozinke 8

Slika 2.5. HashTab program 9

Slika 2.6. HashTab opcije odabira algoritama 10

Slika 2.7. Komprimiranje i enkripcija test.txt datoteke 15

Slika 2.8. Prikaz novokreirane test.7z datoteke 16

Slika 2.9. Zakriptirana 7-zip datoteka ne može se otvoriti bez lozinke 16

Slika 2.10. Prikaz naredbe za dobivanje *hasha* 7-zip datoteke 17

Slika 2.11. Prikaz datoteka korištenih za dobivanje *hasha* 17

Slika 3.1. Preuzimanje *hashcat* programa sa službene stranice 20

Slika 3.2. Struktura *hashcat* mape 20

Slika 3.3. Prikaz nekih od početnih opcija *hashcat* `--help` naredbe 21

Slika 3.4. Izračunavanje prostora ključa 24

Slika 3.5. Početak probijanja testne lozinke korištenjem napada maske 30

Slika 3.6. Trenutni status *hashcat* radnje 31

Slika 3.7. Nastavak probijanja lozinke 32

Slika 3.8. Lozinka pronađena 32

Slika 3.9. Riječ „test“ odgovara *hashu* 32

Slika 3.10. Lozinka spremljena u poddatoteci 33

Slika 3.11. Lozinka probijena za jednu minutu i sedam sekundi 34

Slika 3.12. Pronađena lozinka u tekstualnom formatu 34

Slika 3.13. Broj riječi (redova) u rockyou.txt datoteci 36

Slika 3.14. Riječ „test“ unutar rockyou.txt datoteke 36

Slika 3.15. Primjer napada rječnikom 37

Slika 3.16. Probijena lozinka „test“ pomoću rockyou.txt datoteke 38

Slika 3.17. Rječnik english3.txt sadrži riječ „test“ 39

Slika 3.18. Probijanje lozinke putem english3.txt rječnika 39

Slika 3.19. Probijena lozinka „test“ pomoću english3.txt datoteke 40

Slika 3.20. *Hashcat* poruka prilikom korištenja malih rječnika 40

Slika 3.21. Probijena lozinka „test“ pomoću english3.txt datoteke uz korištenje optimiziranog kernela 41

Slika 3.22. Primjer novokreiranog pravila „t“ 45

Slika 3.23. *rules* mapa unutar *hashcat* mape 46

Slika 3.24. Kreirano pravilo u datoteci moje_pravilo.txt 47

Slika 3.25. Ispis na ekranu lozinki kreiranih pravilom 48

Slika 4.1. Nemogućnost korištenja *hashcat* programa na nekim računalima 52

Slika 4.2. Računalo 1 – slabije 53

Slika 4.3. Računalo 2 – jače 54

Slika 4.4. Slabije računalo – MD5 *hash* 55

Slika 4.5. Slabije računalo – 7-zip *hash* 55

Slika 4.6. Jače računalo – MD5 *hash* 56

Slika 4.7. Jače računalo – 7-zip *hash* 56

Slika 4.8. *Shellntel benchmark* NTLM *hasheva* 58

Slika 4.9. *Benchmark* NTLM *hasheva* na slabijem računalu 59

Slika 4.10. *Benchmark* NTLM *hasheva* na jačem računalu 59

Slika 4.11. Probijanje lozinke „Cat“ na slabijem računalu napadom sirovom snagom 62

Slika 4.12. *Hashcat* radi na probijanju lozinke 63

Slika 4.13. Uspješno probijena lozinka „Cat“ na slabijem računalu 63

Slika 4.14. Pronađena lozinka spremljena u datoteku 64
Slika 4.15. Probijanje lozinke „Cat“ na jačem računalu napadom sirovom snagom 64
Slika 4.16. Uspješno probijena lozinka „Cat“ na jačem računalu 65
Slika 4.17. Generirana slučajno odabrana lozinka 66
Slika 4.18. Lozinka „f%:vX“ nije probijena na slabijem računalu 67
Slika 4.19. Lozinka „f%:vX“ nije probijena na jačem računalu 67
Slika 4.20. Uspješno probijena lozinka „f%:vX“ na slabijem računalu (MD5)..... 68
Slika 4.21. Uspješno probijena lozinka „f%:vX“ na jačem računalu (MD5)..... 69
Slika 4.22. Lozinka „Wednesday1“ nije probijena na slabijem računalu 71
Slika 4.23. Uspješno probijena lozinka „Wednesday1“ na jačem računalu 71
Slika 4.24. Provjera postoji li „Wednesday1“ lozinka (rockyou.txt) 72
Slika 4.25. Pokretanje hibridnog *hashcat* napada..... 74
Slika 4.26. Uspješno probijena lozinka „Wednesday1“ na slabijem računalu 74
Slika 4.27. Provjera postoji li riječ „Wednesday“ (words.txt) 75
Slika 4.28. Uspješno probijena lozinka „Wednesday1“ na jačem računalu 75
Slika 4.29. Početak napada korištenjem pravila best64.rule 80
Slika 4.30. *Hashcat* je krenuo probijati lozinke koristeći best64.rule pravilo..... 80
Slika 4.31. Neuspješno pronađena lozinka 81
Slika 4.32. Nedovoljno memorije za pokretanje napada 82
Slika 4.33. Uspješno pokrenut napad dvama pravilima 82
Slika 4.34. Procijenjeno vrijeme uspješnosti napada 83
Slika 4.35. Lozinka nije pronađena korištenjem dvaju pravila 83
Slika 4.36. Ručno kreiranje pravila za dodavanje znamenki na kraj svake riječi 84
Slika 4.37. Ručno kreiranje pravila za specijalne znakove 85
Slika 4.38. Nedostatak memorije..... 86
Slika 4.39. Uspješno kreirana SVE_LOZINKE_TEST.txt datoteka..... 86

Slika 4.40. Veličina SVE_LOZINKE_TEST.txt datoteke 87
Slika 4.41. Ukupan broj riječi u SVE_LOZINKE_TEST.txt datoteci 87
Slika 4.42. Neuspješan pokušaj korištenja četiriju pravila 88
Slika 4.43. Promjena napada iz napada pravilima u napad listom riječi 89
Slika 4.44. Probijene dvije lozinke 90
Slika 4.45. Napad zaustavljen nakon dva dana i petnaest sati 90
Slika 4.46. Lozinka „Dinosaur22“ probijena 91

Popis tablica

Tablica 2.1. CRC32 *hash* funkcija 2

Tablica 2.2. Primjer MD5 *hash* izlaznih vrijednosti 11

Tablica 3.1. Izmišljene lozinke od korijena riječi 42

Tablica 3.2. Kreiranje prvog pravila 43

Tablica 3.3. Primjer nekih od pravila *hashcat* programa 44

Tablica 4.1. Specifikacije računala 52

Tablica 4.2. Usporedni test brzine računanja *hasheva* 57

Tablica 4.3. Brzine računanja NTLM *hasheva* 60

Tablica 4.4. Lozinke za primjer napada korištenjem pravila 77

Tablica 4.5. Lista riječi moja_lista.txt 78

Tablica 0.1. Sažet prikaz svih korištenih lozinki 93

Literatura

[1] PICOLET, J. *Hash Crack: Password Cracking Manual*. CreateSpace Independent Publishing Platform, 2017.

[2] ENGEBRETSON, P. *The Basics of Hacking and Penetration Testing*. Elsevier Inc, 2013.

[3] WIKIPEDIA, https://en.wikipedia.org/wiki/Main_Page, listopad. 2019.

[4] HASHCAT, https://hashcat.net/hashcat/, studeni. 2019.

[5] YAZDI, S.H. *Analyzing Password Strength & Efficient Password Cracking*. Florida State University Libraries, 2011.

[6] 7-ZIP, https://www.7-zip.org/, studeni. 2019.

[7] KEEPASS, https://keepass.info/, studeni. 2019.

[8] HASHTAB, http://implbits.com/products/hashtab/, studeni. 2019.

[9] JOHN THE RIPPER, https://www.openwall.com/john/, studeni. 2019.

[10] LEETSPEAK, https://www.howtogeek.com/443390/what-is-leet-speak-and-how-do-you-use-it/, prosinac. 2019.

[11] TOGGLE RULES, https://blog.didierstevens.com/2016/07/16/tool-to-generate-hashcat-toggle-rules/, prosinac. 2019.

[12] ENGLISH3 WORDLIST, http://www.gwicks.net/textlists/english3.zip, studeni. 2019.

[13] ROCKYOU.TXT, https://www.kaggle.com/wjburns/common-password-list-rockyoutxt, studeni. 2019.

[14] MIT LICENSE, https://opensource.org/licenses/MIT, listopad. 2019.

[15] INITIALIZATION VECTOR, https://whatis.techtarget.com/definition/initialization-vector-IV, prosinac. 2019.

[16] PASSWORD VS PASSPHRASE, https://www.passworddragon.com/password-vs-passphrase, prosinac. 2019.

[17] SHA2, https://cheapsslsecurity.com/blog/what-is-sha2-and-what-are-sha-2-ssl-certificates/, listopad. 2019.

[18] CONFIDENTIALITY IN THE WORKPLACE, https://www.hiscox.co.uk/business-blog/the-importance-of-confidentiality-in-the-workplace/, prosinac. 2019.

[19] BENCHMARK HASHCAT, https://www.onlinehashcrack.com/tools-benchmark-hashcat-gtx-1080-ti-1070-ti.php, listopad. 2019.

[20] ENCODING VS ENCRYPTION, https://stackoverflow.com/questions/4657416/difference-between-encoding-and-encryption, listopad. 2019.

[21] PASSWORD HABITS, https://www.helpnetsecurity.com/2019/05/13/people-password-habits/, prosinac. 2019.



ALGEBRA
VISOKO UČILIŠTE

PROBIJANJE LOZINKI PUTEM hashcat PROGRAMA

Pristupnik: Zlatko Paleka, 0112005683

Mentor: prof. Robert Petrunić