Testiranje sigurnosti zaporki
Tehnička dokumentacija
Verzija 1.1

Studentski tim: Marin Mikulčić
Lara Brečić
Dominik Topić
Sven Sonicki

Nastavnik: prof. dr. sc. Marin Golub

Sadržaj

Tehnička dokumentacija

1. Opis alata Hashcat

Hashcat je otvoreni alat za oporavak zaporki i analizu hasheva. projekt se 2015. spojio s GPU varijantom oclHashcat te danas nudi jedinstveni program za CPU i GPU platforme. Hashcat je poznat po svojoj brzini radi koje je popularan među sigurnosnim istraživačima, testnim timovima i sistemskim administratorima. Kako alat simulira napade na hashirane zaporke, omogućava provjeru jačine zaporke kao i koliko je sigurno pohranjena zaporka.

Alat podržava više stotina algoritama za pohranu zaporki, od zastarjelih poput md4, md5 do modernih i složenih funkcija izvedenih iz SHA-1, SHA-2, PBKDF. Među njima su i PBKDF2-HMAC-SHA256 i Argon2id koji nisu klasični hashing algoritmi već se smatraju funkcijama za derivaciju ključa iz zaporke (engl. Password-Based Key Derivation Functions, skraćeno KDF).

Pri oporavku zaporki iz hasheva pomoću hashcata potrebno je definirati nad kojim algoritmom/funkcijom se pokušava otkriti zaporka. S obzirom da je hashcat alat naredbenog retka (engl. Command-line interface skraćeno CLI) za svaku operaciju potrebno ga je konfigurirati zastavicama/opcijama. Tip algoritma definira se pomoću zastavice/opcije „-m“ i broja (argumenta) koji definira o kojem se algoritmu/funkciji radi. U ovom projektu fokus je bio na sljedeće algoritme/funkcije:

SHA-1 – hashing algoritam koji zaporke pretvara u hasheve duljine 160 bita. U hashcatu se odabire pomoću „-m 100“. Osim samog algoritma hashcat nudi i podršku za SHA-1 zajedno sa saltom. Ako se radi o formatu SHA-1(zaporka.salt) lagano se bira pomoću „-m 110“, ako se radi o SHA-1(salt.zaporka) bira se pomoću „-m 120“, za SHA-1(salt.zaporka.salt) potrebno je odabrati „-m 130“

PBKDF2-HMAC-SHA256 – funkcija za derivaciju ključa iz zaporke je dizajnirana da bude spora kako bi značajno usporilo napade. Jedan je od funkcija koji se standardno koristi u modernim sustavima jer omogućava podešavanje broja iteracija i veličine soli kako bi sustav bio responzivan ali značajno usporio napade i time povećao sigurnost pohrane zaporki. U hashcatu se bira pomoću „-m 10900“.

Argon2id – također funkcija za derivaciju ključa iz zaporke, ali za razliku od PBKDF2-HMAC-SHA256 osim što je vremenski skupocjen je i memorijski skupocjen čime dodatno povećava sigurnost pohrane od masivnih napada ubrzanih GPU-ovima ili drugim ASIC napadima. Bira se pomoću zastavice „-m 34000“

2. Tehničke značajke alata Hashcat

Hashcat podržava nekoliko načina napada (attack modes) koje biramo zastavicom -a. U najnovijoj verziji dostupni su sljedeći načini napada:

1. Rječnički napad (dictionary / straight) „–a 0“. U ovom načinu program redom čita svaku riječ iz tekstualne datoteke i koristi je kao kandidat za zaporku. Kombinacije se mogu proširiti tzv. rules datotekama (-r rules.rule) koje primjenjuju transformacije (npr. mijenjanje slova iz malih u velika, dodavanje sufiksa, obrnut redoslijed itd.). Primjer za SHA-1: hashcat -m 100 -a 0 hashevi.txt words.txt -r rules/best64.rule.

2. Kombinator napad „–a 1“. Kombinator spaja dva rječnika – prvi s prefiksom, drugi sa sufiksom i generira sve moguće parove. Na taj način moguće je simulirati konstrukcije tipa ime + godina rođenja (npr. Marko + 2002) bez ručnog spajanja lista. Sintaksa: hashcat -m 100 -a 1 hashevi.txt dict1.txt dict2.txt.

3. Maskirani napad / napad grubom silom „–a 3“. Ovdje generator prolazi kroz sve kombinacije znakova unutar definirane maske. Maska se zadaje kao niz znakova i zamjenskih skupova – npr. ?l?l?l?d?d označava tri mala slova, pa dva broja (npr. abc12). Postoje ugrađene skupine: ?l (mala slova), ?u (velika slova), ?d (brojevi), ?s (specijalni znakovi) itd., a moguće je definirati i vlastite skupove -1, -2, -3, -4. Maskirani napad je učinkovit kada se zna struktura zaporke (duljinu, vrste znakova), jer značajno smanjuje prostor pretrage u odnosu na puni napad grubom silom.

4. Hibridni napadi „–a 6“ i „–a 7“. Hybrid kombinira rječnik i masku: -a 6 spaja riječ iz rječnika + masku, a -a 7 masku + riječ. Na primjer, hashcat -m 100 -a 6 hashovi.txt words.txt ?d?d pokušat će sve riječi iz words.txt i na kraj im dodati dvije znamenke.

5. Asocijacijski napad „–a 9“. Poznat i kao association/single način, pokušava svaki član rječnika u kombinaciji s pripadajućim hashom u istoj poziciji. Koristi se kada se gleda popis korisničkih podataka (npr. korisnička imena) i jednako dugačak popis hashova – svrha mu je brzo pronaći poklapanja. Ovaj način se može kombinirati s pravilima (-r) za dodatne transformacije.

3. Upute za korištenje alata Hashcat

Alat hashcat je dostupan na većini operacijskih sustava. Na operacijskim sustavaima osnovanim na Linuxu najjednostavnije ga je instalirati pomoću respektivnog upravitelja paketa (apt, pacman, dnf), za Windows operacijske sustave najjednostavnije je preuzeti 7zip arhivu sa službenog GitHuba alata i za macOS najjednostavnije je pomoću njegovog upravitelja paketa homebrew (brew).

```
hashcat bad_sha1.txt -o result.txt -a 3 -m 100 -w 3 --increment --increment-min 6 --increment-max 8 -O
'?a?a?a?a?a?a?a?a' --status --status-timer 10
hashcat (v7.1.2) starting

CUDA API (CUDA 12.2)
====================
* Device #01: NVIDIA GeForce RTX 2080 Ti, 10836/11009 MB, 68MCU
* Device #02: NVIDIA GeForce RTX 2080 Ti, 10848/11011 MB, 68MCU

OpenCL API (OpenCL 3.0 CUDA 12.2.149) - Platform #1 [NVIDIA Corporation]
========================================================================
* Device #03: NVIDIA GeForce RTX 2080 Ti, skipped
* Device #04: NVIDIA GeForce RTX 2080 Ti, skipped

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 55

Hashes: 2400 digests; 2400 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Precompute-Init
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Brute-Force
* Raw-Hash

Watchdog: Temperature abort trigger set to 90c

Host memory allocated for this attack: 6459 MB (92385 MB free)

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

Session..........: hashcat
Status...........: Running
Hash.Mode........: 100 (SHA1)
Hash.Target......: bad_sha1.txt
Time.Started.....: Tue Dec 16 22:28:08 2025 (7 secs)
Time.Estimated...: Tue Dec 16 22:28:29 2025 (14 secs)
Kernel.Feature...: Optimized Kernel (password length 0-55 bytes)
Guess.Mask.......: ?a?a?a?a?a?a [6]
Guess.Queue......: 1/3 (33.33%)
Speed.#01........: 17014.7 MH/s (94.30ms) @ Accel:48 Loops:1024 Thr:512 Vec:1
```

```
Speed.#02.....: 16508.1 MH/s (97.18ms) @ Accel:48 Loops:1024 Thr:512 Vec:1
Speed.#*.....: 33522.8 MH/s
Recovered.....: 577/2400 (24.04%) Digests (total), 577/2400 (24.04%) Digests (new)
Remaining.....: 1823 (75.96%) Digests
Recovered/Time...: CUR:N/A,N/A,N/A AVG:N/A,N/A,N/A (Min,Hour,Day)
Progress.....: 251584315392/735091890625 (34.22%)
Rejected.....: 0/251584315392 (0.00%)
Restore.Point....: 25067520/81450625 (30.78%)
Restore.Sub.#01..: Salt:0 Amplifier:4096-5120 Iteration:0-1024
Restore.Sub.#02..: Salt:0 Amplifier:1024-2048 Iteration:0-1024
Candidate.Engine.: Device Generator
Candidates.#01...: tE%"rj -> :{mXU5
Candidates.#02...: /p1:p. -> IZzum.
Hardware.Mon.#01.: Temp: 54c Fan: 42% Util: 99% Core:1770MHz Mem:6800MHz Bus:16
Hardware.Mon.#02.: Temp: 49c Fan: 37% Util: 99% Core:1725MHz Mem:6800MHz Bus:16

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

Session.....: hashcat
Status.....: Running
Hash.Mode.....: 100 (SHA1)
Hash.Target.....: bad_sha1.txt
Time.Started.....: Tue Dec 16 22:28:08 2025 (18 secs)
Time.Estimated...: Tue Dec 16 22:28:30 2025 (4 secs)
Kernel.Feature...: Optimized Kernel (password length 0-55 bytes)
Guess.Mask.....: ?a?a?a?a?a?a [6]
Guess.Queue.....: 1/3 (33.33%)
Speed.#01.....: 17005.8 MH/s (94.81ms) @ Accel:48 Loops:1024 Thr:512 Vec:1
Speed.#02.....: 16520.1 MH/s (97.41ms) @ Accel:48 Loops:1024 Thr:512 Vec:1
Speed.#*.....: 33525.9 MH/s
Recovered.....: 782/2400 (32.58%) Digests (total), 782/2400 (32.58%) Digests (new)
Remaining.....: 1618 (67.42%) Digests
Recovered/Time...: CUR:N/A,N/A,N/A AVG:N/A,N/A,N/A (Min,Hour,Day)
Progress.....: 591951101952/735091890625 (80.53%)
Rejected.....: 0/591951101952 (0.00%)
Restore.Point....: 61833216/81450625 (75.91%)
Restore.Sub.#01..: Salt:0 Amplifier:8192-9025 Iteration:0-1024
Restore.Sub.#02..: Salt:0 Amplifier:3072-4096 Iteration:0-1024
Candidate.Engine.: Device Generator
Candidates.#01...: 3<vMaE ->  ~I@n)
Candidates.#02...: MGO!@? -> dI/V`U
Hardware.Mon.#01.: Temp: 58c Fan: 46% Util: 99% Core:1740MHz Mem:6800MHz Bus:16
Hardware.Mon.#02.: Temp: 52c Fan: 40% Util:100% Core:1680MHz Mem:6800MHz Bus:16

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

U ovom primjeru se vidi ispis alata hashcat za napad grubom silom na hasheve zaštičene SHA-1 algoritmom. U samom pozivu je definirano da se radi o SHA-1, burte force napadu povečavajući broj znakova za pokušaj od 6 do 8. maska se sastoji od ?a što označava da na svaku poziciju treba pokušati svaki mogući simbol, slovo i broj. Definirano je da ispiše napredak svakih deset sekundi radi čega se u ispisu dobivaju detalji napretka.

Na početku vidimo da hashcat bira kojim putem će komunicirati sa dostupnim grafičkim karticama koje će koristiti kako bi značajno ubrzao napad i postavlja optimizacije kojima bi dodatno ubrzao napad. Dalje svakih deset sekundi ispisuje nad kojom hash datotekom radi, vrijeme početka i kolko je prošlo od tog početka, vrijeme očekivanog kraja i koliko još treba do kraja, koji kernel koristi, koju masku koristi (u slučaju napada grubom silom ili napada koji kombiniraju rječnik i masku), ako ima neki oblik iteracija u ovom slučaju povećavanje maske od 6 do 8 znakova, brzinu kojom provjerava hasheve, koliko ih je uspio otkriti iz datoteke hasheva u odnosu na koliko ih je u datoteci, koliko je kandidata provjerio i koliko ih još treba provjerit, kod koliko kandidata je došlo do greške, pozicije na kojima je spremljeno stanje napada u slučaju da je potrebno nastaviti napad ako stane, prikaz trenutnih kandidata i informacije o hardveru.

4. Djelotvornost alata Hashcat u eksperimentu

U eksperimentu su korištene tri liste zaporki pomoću kojih su stvorene hash datoteke korištene u napadima.

Prva lista je skup nasumičnih znakova ispod minimalne norme (barem 8 znakova od kojih su barem jedno veliko slovo, barem jedno malo slovo, barem jedan broj i barem jedan simbol) ukupno se u listi nalazilo 2400 zaporki. U nastavku će hash datoteke nastale iz te liste biti označene sa „bad“.

Druga lista se sastoji od 1000 najkorištenijih zaporki (isto je većina njih ispod minimalne norme). U nastavku će hash datoteke nastale iz te liste biti označene sa „common“.

Treća lista se sastoji od 1000 zaporki nasumično generiranih od 16 do 20 znakova koji svaki ima barem jedno veliko i malo slovo, broj i simbol. To je približno zaporkama koje generiraju čuvari zaporki. U nastavku će hash datoteke nastale iz te liste biti označene sa „locker“.

Glavna razlika između različitih lista zaporki je to da nisu jednako osjetljive na iste tipove napada. U tablici je prikazano koliko je zaporki pronađeno u najboljem slučaju napada. Svi napadi su mogli trajati maksimalno 12 sati nakon čega bi se napad prekinuo jer bi neki od napada trajali ekstremno dugo.

Hashing algoritmi	bad	common	locker
SHA-1	1847	985	-
SHA-1 + salt	667	986	-
PBKDF2-HMAC-SHA256	0	703	-
Argon2id	0	979	-

4. 1. tablica uspješno pronađenih zaporki po listi zaporki i algoritmu

Za hash datoteke iz liste bad najbolje je radio napad grubom silom koji generira sve moguće kombinacije zaporki i pomoću njih provjerava ima li podudaranja sa nekim od hasheva. Ovdje također možemo vidjeti da salt osim što štiti zaporke od rainbow table napada značajno usporava bilo koji oblik napada grubom silom ili rječničkog napada jer je za svakog kandidata potrebno generirati zaseban hash kako bi se moglo provjeriti podudaranje. U ovom slučaju svakog kandidata je bilo potrebno hashirati 2400 puta da bi se provjerilo podudaranje. Broj koliko je puta bilo potrebno hashirati svakog kandidata je padalo jer bi za svakog otkrivenog kandidata broj potrebnih hasheva po kandidatu padao jer nema potrebe provjeravati podudaranje za već otkrivenu zaporku.

Za hash datoteke nastale iz liste common, najbolje se pokazao rječnički napad. Rječnički napad bio je izveden pomoću dobro poznatog rječnika „rockyou.txt“. Iako izgleda da je u situaciji sa SHA-1 + salt pronađena jedna zaporka više to nije situacija. Lista najkorištenijih zaporki korištena za generiranje ovih hash datoteka je imala jedan duplikat (ista zaporka upisana dva puta). Hashcat pri pokretanju napada provjerava hash datoteku za duplikate kako bi dodatno ubrzao izvođenje. Razlog zašto ista zaporka nije uklonjena iz drugih hash datoteka je u tome što se zbog različitih saltova razlikuju i hashevi iako su generirani iz iste zaporke. Za SHA-1 i SHA-1 + salt vrijeme potrebno da se provjere svi kandidati iz rockyou.txt je bilo ispod 30 sekundi dok su i PBKDF2-HMAC-SHA256 i Argon2id zaustavljeni nakon 12 sati. Iz ovoga vidimo da je opasnije imati zaporku koja se nalazi u nekom od rječnika nego imati zaporku koja ne zadovoljava minimalnu normu u potpunosti. Bez obzira na to i dalje je opasno ako zaporka ne zadovoljava barem minimalnu normu. Neočekivani ishod ovdje je taj da je pronađeno više zaporki zaštićenih Argon2id-om nego PBKDF2-HMAC-SHA256. Ponavljanjem napada nad Argon2id i PBKDF2-HMAC-SHA256 zanimljivo je da je količina pronađenih zaporki Argon2id u svim pokušajima bila slična dok je količina pronađenih zaporki za PBKDF2-HMAC-SHA256 značajno varirala od 335 do 703.

Za hash datoteke nastale iz liste locker, zaključkom iz prethodne dvije liste je da jedini smisleni napad bi bio napad grubom silom. No kako je minimalna duljina zaporke u tim datotekama 16 to bi i na jačem hardveru trajalo ekstremno dugo, a da se dobije ikakav značajan broj otkrivenih zaporki. Osim toga Hashcat se nije uspio pokrenut jer je duljina zaporke bila prevelika s obzirom da je potrebno provjeriti na svakoj poziciji i velika i mala slova, brojeve i simbole. Iz ovoga vidimo da bi bilo dobro da se podigne minimalna norma na veći broj znakova kako bi se neutralizirali već postojeći optimizirani alati, ali i samo povećanje broja znakova neće imati dovoljno veliki efekt jer će i dalje velika količina zaporki moć biti otkrivena pomoću rječničkih napada i kombinacije rječnika i generatora. Zato je bitno da su zaporke zaštićene algoritmima koji su dizajnirani da budu dovoljno spori kako bi se i u takvim slučajevima moglo značajno usporiti vrijeme potrebno za otkrivanje zaporki čime bi bilo preskupo raditi ikakav oblik napada bez značajne količine skupog hardvera.

5. Opis alata John the Ripper

John the Ripper je besplatan i otvoren izvorni alat za audite sigurnosti zaporki i obnovu zaporki, razvijen za različite operacijske sustave. Ovaj alat podržava širok spektar tipova kripto sažetaka i šifri, uključujući one korištene za zaporke na Unix sustavima (Linux, *BSD, Solaris, AIX, QNX i drugi), macOS, Windows, web aplikacije (npr. WordPress), grupne aplikacije (npr. Notes/Domino), kao i baze podataka (SQL, LDAP, itd.). Dodatno, John the Ripper omogućava obrada snimaka mrežnog prometa (npr. Autentifikacija u Windows mreži, WPA-PSK za WiFi), šifriranih privatnih ključeva (npr. SSH, GnuPG, novčanici za kriptovalute), datotečni sustav (npr. macOS .dmg, Windows BitLocker), arhiva (ZIP, RAR, 7z) i dokumenata (PDF, Microsoft Office).

Iako je John the Ripper prvobitno razvijen za Unix operacijske sustave, alat je sada dostupan na petnaest različitih platformi, uključujući Unix (s posebnim verzijama za različite arhitekture), DOS, Win32, BeOS, i OpenVMS. Zbog svoje svestranosti i široke primjene, John the Ripper je jedan od najčešće korištenih programa za testiranje sigurnosti zaporki. On integrira različite tehnike za razbijanje zaporki u jednom paketu, automatski prepoznaje tipove kripto sažetaka i pruža prilagodljive opcije za napade na zaporke.

Glavne karakteristike

• Podrška za razne algoritme: John the Ripper podržava brojne algoritme za kripto sažetke zaporki, uključujući stare i moderne metode poput DES, MD5, SHA1, bcrypt, Argon2id, i PBKDF2. Ova podrška omogućava alatima da obrade širok spektar zaporki i kripto sažetaka.

• Visoka efikasnost: Alat je optimiziran za visoke performanse i brzinu, koristeći tehnike kao što su paralelizacija putem višestrukih jezgri i GPU akceleracija, čime značajno poboljšava brzinu obrade.

• Multiplatformska podrška: John the Ripper je dostupan na različitim operacijskim sustavima, uključujući Linux, macOS, i Windows, što omogućava njegovu primjenu u različitim okruženjima.

• Raznovrsne strategije napada: Alat podržava različite strategije napada, uključujući napade s rječnicima, napade grubom silom i napade bazirane na pravilima, što ga čini izuzetno fleksibilnim u različitim sigurnosnim testiranjima.

Upotreba Alata

Upotreba alata John the Ripper obuhvaća različite tehnike napada na kripto sažetke zaporki. Najčešće korištene metode napada uključuju:

• Napad s rječnikom: Ovaj napad koristi tekstualne datoteke (rječnike) koji sadrže riječi koje bi mogle biti korištene kao zaporke. Svaka riječ u rječniku se enkriptira i uspoređuje s kripto sažetkom zaporke. Ovaj pristup je efikasan za razbijanje jednostavnih zaporki koje se temelje na uobičajenim riječima ili frazama.

Primjer upotrebe:

Ako se analizira datoteka koja sadrži kripto sažetke zaporki, npr. Pass.txt, sljedeća naredba se koristi za pokretanje napada:

john –wordlist=password.lst pass.txt

Ova naredba koristi rječnik password.lst za pokušaj dekriptiranja zaporki iz datoteke pass.txt.

• Napad grubom silom (inkrementalni napad): Ovaj tip napada pokušava sve moguće kombinacije znakova, uspoređujući kripto sažetak svakog pokušaja s originalnim kripto sažetkom. Iako je izuzetno učinkovit za zaporke koje nisu obuhvaćene u rječnicima, napad grubom silom može biti vrlo spor i zahtijeva značajne računalne resurse.

Vrste Napada

John the Ripper pruža niz različitih vrsta napada, koji omogućavaju prilagodbu napada specifičnim uvjetima:

• Markov režim: Temelji se na istraživanjima Arvinda Narayanana i Vitaly Shmatikova za generiranje kandidata zaporki. Ovaj način napada koristi statističke metode za predviđanje zaporki na temelju vjerojatnosti.

• Mask režim: Koristi korisnički definirane obrasce za generiranje kandidata zaporki. Ovaj način napada omogućava specifične prilagodbe u stvaranju zaporki, poput duljine, specijalnih znakova i sl.

• Single crack režim: U ovom načinu napada, alat koristi informacije poput korisničkog imena, punog imena ili putanje direktorija za generiranje mogućih kandidata zaporki.

• Subsets režim: Generira kandidate zaporki na temelju složenosti, prioritetizirajući duže zaporke i manje uobičajene kombinacije.

• Regex režim: Eksperimentalni način koji koristi korisnički definirane regularne izraze (regex) za generiranje kandidata zaporki, slično kao mask mode, ali s većim naglaskom na specifične obrasce.

• Mode stacking: Kombinira mask mode ili regex mode s drugim napadima, čime omogućava složenije i efikasnije napade na zaporke.

6. Instalacija i postavljanje alata John the Ripper

Ovaj odjeljak pruža detaljan vodič za instalaciju John the Ripper alata s GitHub repozitorija, uključujući sve potrebne korak za pripremu radnog okruženja, preuzimanje izvornog koda, prevođenje i pokretanje alata.

Priprema okruženja

Prvi korak u instalaciji John the Ripper alata je provjera postoji li odgovarajuća infrastruktura i potrebne biblioteke na sustavu kako bi se omogućilo uspješno prevođenje i izvršavanje alata. Preporučene komponente za većinu Linux distribucija, uključujući one temeljene na Ubuntu i Debian sustavima, obuhvaćaju sljedeće:

• Git – vanjski kontrolni alat koji omogućava preuzimanje izvornog koda alata s GitHub repozitorija

• Make – alat za automatizaciju procesa izgradnje, koji omogućava prevođenje izvornog koda prema definiranim uputama

• GCC ili drugi C prevoditelj – prevoditelj za programski jezik C, neophodan za prevođenje izvornog koda koji je temeljen na ovom jeziku

• Build Essential – paket koji sadrži osnovne alate i biblioteke potrebne za izgradnju i prevođenje softverskih paketa na sustavima temeljenim na Debianu

Za instalaciju navedenih paketa na Linux sustavima temeljenim na Ubuntu ili Debian distribucijama, potrebno je pokrenuti sljedeće naredbe u terminalu:

```
sudo apt update
sudo apt install git build-essential gcc make
```

Preuzimanje koda s GitHub-a

Nakon pripreme okruženja, sljedeći korak j kloniranje najnovije verzije John the Ripper alata s GitHub repozitorija. Sljedeća naredba će preuzeti cijeli izvorni kod za John the Ripper u direktorij pod nazivom JohnTheRipper:

```
git clone https://github.com/magnumripper/JohnTheRipper.git
```

Priprema za prevođenje

Nakon preuzimanja izvornog koda alata John the Ripper, sljedeći korak je priprema sustava za prevođenje. U ovom koraku potrebno je osigurati da su na sustavu instalirane sve relevantne biblioteke i specifični alati koji omogućuju uspješno izvođenje postupka prevođenja.

Za osnovnu verziju alata John the Ripper, koja koristi samo CPU za obradu, postupak prevođenja može se pokrenuti pomoću sljedeće naredbe:

make

Ova naredba pokreće postupak prevođenja na temelju Makefile datoteke, koja sadrži upute za izgradnju alata. Važno je napomenuti da vrijeme potrebno za prevođenje može varirati ovisno o računalnim resursima i performansama sustava. U prosjeku, proces može trajati nekoliko minuta, no na sustavima s manjim resursima ili starijom hardverskom opremom, postupak može trajati duže.

2.5. Instalacija s podrškom za specifične algoritme

Ako želite koristiti dodatne algoritme ili akceleraciju pomoću GPU-a, potrebno je izgraditi Jumbo verziju John the Ripper alata, koja uključuje mnoge dodatne optimizacije i podršku za različite algoritme. Za ovo je potrebno instalirati dodatne biblioteke, uključujući:

• OpenSSL – za podršku kriptografskim algoritmima

• CUDA – za podršku NVIDIA GPU akceleracije

Za izgradnju verzije s CUDA podrškom, sljedeće naredbe mogu se koristit:

```
make clean
```

```
make -s generic64-cuda
```

Ove naredbe će ukloniti prethodno prevedene datoteke i prevesti alat s CUDA podrškom.

Pokretanje John the Ripper-a

Nakon što je proces prevođenja alata John the Ripper uspješno završen, preporučuje se izvršiti osnovnu provjeru ispravnosti instalacije. Ova provjera omogućava korisnicima da se uvjere da je alat ispravn instalirano i da može ispravno izvršavati onovne funkcije. Za testiranje ispravnosti, potrebno je pokrenuti sljedeću naredbu u terminalu:

```
./john ––test
```

Ova naredba inicira internu provjeru performansi alata, koja uključuje testiranje osnovnih operacija, kao što su brzina i funkcionalnost napada na zaporke. Rezultati ovog testa omogućit će korisniku da potvrdi ispravnost instalacije te da procijeni performanse alata u odnosu na računalne resurse.

Ukoliko test završi bez greške, to je pokazatelj da je alat ispravno instaliran i spreman za daljnje korištenje. Ako se pojave greške, korisnici će morati ponovno pregledati proces instalacije i provjeriti konfiguracijske postavke.

Priprema datoteke sa sažecima zaporki

Za potrebe eksperimenta, generirana je datoteka koja sadrži podatke korisnika sustava. Ova datoteka je stvorena kombiniranjem sadržaja iz sustavnih datoteka /etc/password i /etc/shadow pomoću sljedeće naredbe:

```
sudo unshadow /etc/passwd /etc/shadow > mojipass.txt
```

Naredba unshadow spaja podatke iz passwd i shadow datoteka, stvarajući novu datoteku mojipass.txt koja sadrži sažetke zaporki svih korisnika na sustavu.
Pokretanje alata John the Ripper za probijanje zaporki

Nakon pripreme datoteke sa sažetcima zaporki, pokrenuta je naredba za razbijanje zaporki pomoću alata John the Ripper:

```
run/john mojpass.txt
```

Ova naredba omogućava alatu John the Ripper da pročita sažetke iz datoteke mojipass.txt i pokuša ih probiti koristeći unaprijed definirani skup algoritama i tehnika napada.

Izvedeni rezultati

Po završetku procesa probijenja zaporki, John the Ripper je izvijestio o uspjehu u probijanju svih zaporki iz datoteke mojipass.txt. Ukupno je probijeno deset zaporki, bez preostalih neuspješnih pokušaja. Brzina probijanja zaporki tijekom eksperimenta bila je približno 200 zaporki u sekundi. Ovi rezultati potvrđuju učinkovitost alata u probijanju jednostavnijih zaporki, kao i njegovu sposobnost brze obrade velike količine informacija. Potpuni rezultati mogu se vidjeti u nastavku, gdje je jasno da je alat uspio obraditi sve sažetke u vrlo kratkom vremenskom razdoblju:

Zaključak eksperimenta

Provedeni eksperiment pokazuje da je John the Ripper alat uspješno izvršio razbijanje svih zaporki sadržanih u datoteci mojpass.txt. S obzirom na brzinu obrade (oko 200 zaporki u sekundi), može se zaključiti da alat učinkovito razbija jednostavne zaporke koje koriste slabe ili uobičajene uzorke. Dodatno, rezultati eksperimenta potvrđuju visoku efikasnost alata u kontekstu brzine obrade, kao i njegovu primjenjivost u realnim scenarijima testiranja sigurnosti zaporki.

7. Djelotvornost alata John the Ripper u eksperimentu

Nakon početnog eksperimenta nad ograničenim brojem lokalnih korisnika, provedena su sustavnija ispitivanja alata John the Rippera uz korištenje unaprijed pripremljenog skupa tekstualnih datoteka (engl. Wordlist) koje sadrže kandidate za zaporke.

Za potrebe analize odabrane su tri wordliste različite razine složenosti označene kao bad, common i locker:

Skup bad_naziv_algoritma.txt tekstualnih datoteka predstavljaju zaporke koje ne zadovoljavaju uobičajene sigurnosne standarde. Riječ je o zaporkama duljine 5 do 8 znakova koje sadrže kombinaciju velikih i malih slova te barem jednog broja i barem jednog specijalnog znaka. U svakoj datoteci tipa bad_naziv_algoritma.txt nalazi se 2400 takvih zaporki.

Skup common_naziv_algoritma.txt obuhvaća zaporke koje se nalaze među 1000 najčešće korištenih zaporki u praksi. Svaka datoteka tipa common_naziv_algoritma.txt sadrži 1000 zaporki generiranih iz navedene liste najkorištenijih zaporki, čime se simulira realan scenarij u kojem korisnici odabiru jednostavne i predvidljive zaporke.

Skup locker_naziv_algoritma.txt sastoji se od zaporki koje su oblikom i složenošću približne zaporkama koje generiraju aplikacije za upravljanje zaporkama (engl. Password manager). Duljina ovih zaporki kreće se između 16 i 20 znakova te uključuje kombinaciju velikih i malih slova, brojeva i specijalnih znakova. Svaka datoteka tipa locker_naziv_algoritma.txt sadrži 1000 takvih složenih zaporki.

Svaka od ovih datoteka testirana je pojedinačno tijekom 12 sati, tijekom kojih su se analizirali rezultati izvedbe alata John the Ripper na različitim skupovima zaporki. Ovakva podijela omogućuje ispitivanje ponašanja odabranih algoritama za kriptiranje zaporki na jasno razgraničenim razinama sigurnosne složenosti.

Slaganje rezultata

U ovom poglavlju prikazani su rezultati testiranja različitih algoritama za razbijanje zaporki korištenjem alata John the Ripper. Tablica sadrži broj uspješno razbijenih zaporki za četiri različita kriptografska algoritma: SHA1, Salted-SHA1, Argon2i i PBKDF2-HMAC-SHA256, za tri različite skupine zaporki: bad, common i locker.

Svaka skupina zaporki označava različite vrste zaporki s različitim stupnjevima složenosti. Na temelju tablice, možemo primijetiti značajne razlike u učinkovitosti alata za razbijanje zaporki, ovisno o korištenom algoritmu i vrsti zaporki. Na primjer, SHA1 algoritam je postigao visoki broj uspješno razbijenih zaporki, dok su algoritmi poput Argon2i i PBKDF2-HMAC-SHA256 imali znatno niže rezultate, što je očekivano zbog njihove jače zaštite.

	SHA1	Salted-SHA1	Argon2i	PBKDF2-HMAC-SHA256
bad (2400)	1329	17	-	-
common (1000)	1000	944	405	556
locker (1000)	-	-	-	-

7. 1. tablica uspješno pronađenih zaporki po listi zaporki i algoritmu

Na temelju dobivenih rezultata vidljivo je da alat John the Ripper postiže vrlo visoku učinkovitost pri napadu na kripto sažetke generirane slabijim algoritmima i/ili za jednostavnije skupove zaporki. Za skup common (1000 najčešće korištenih zaporki) razbijeno je svih 1000 zaporki za algoritam SHA1, dok je za Salted-SHA1 razbijeno 944 zaporki, odnosno gotovo cijeli skup. Isti skup zaporki pokazao je znatno veću otpornost kod suvremenijih algoritama: za Argon2i razbijeno je 405, a za PBKDF2-HMAC-SHA256 556 zaporki, što i dalje potvrđuje djelotvornost alata, ali uz znatno veći stupanj zaštite u odnosu na klasični SHA1.

Skup bad pokazao je dodatno da i relativno jednostavne sigurnosne mjere, poput soljenja kripto sažetaka, značajno povećavaju otpornost: od 2400 zaporki razbijeno je 1329 za SHA1, dok je za Salted-SHA1 uspješno razbijeno tek 17 zaporki. Napokon, skup locker, koji oponaša zaporke generirane alatima za upravljanje zaporkama, ostao je u potpunosti nerazbijen u promatranom vremenskom razdoblju, neovisno o algoritmu. Ovakvi rezultati potvrđuju da složene, nasumično generirane zaporke u kombinaciji sa suvremenim algoritmima za izračun kripto sažetaka pružaju znatno višu razinu sigurnosti u odnosu na često korištene i jednostavne zaporke.

Zaključno, provedena tehnička analiza alata John the Ripper pokazala je da se radi o iznimno fleksibilnom i učinkovitom alatu za ispitivanje sigurnosti zaporki, osobito u kombinaciji s unaprijed pripremljenim skupovima zaporki različite složenosti. Korištenjem triju jasno definiranih skupova (bad, common i locker) te različitih algoritama za kriptiranje zaporki, dobiven je uvid u ponašanje alata pri napadu na zaporke koje odražavaju tipične korisničke obrasce (jednostavne i često korištene zaporke) u odnosu na zaporke generirane u skladu s preporučenim sigurnosnim praksama.

Rezultati ispitivanja, pri čemu je svaka datoteka bila izložena napadu tijekom unaprijed zadanog vremenskog razdoblja, potvrđuju očekivanu razliku u otpornosti između jednostavnijih i kriptografski snažnijih zaporki, kao i između zastarjelih i suvremenih algoritama za izračun kripto sažetaka. Time je demonstrirano da alat John the Ripper može poslužiti kao pouzdan mehanizam za procjenu razine sigurnosti zaporki u kontroliranom okruženju te kao potpora pri izradi smjernica za sigurno upravljanje zaporkama u stvarnim sustavima.

8. Opis alata Wfuzz

Wfuzz je alat otvorenog izvora namijenjen fuzzingu web aplikacija, koji se koristi u sigurnosnom testiranju s ciljem otkrivanja skrivenih resursa, funkcionalnosti i potencijalnih ranjivosti. Alat djeluje tako da generira i šalje velik broj HTTP zahtjeva prema ciljnoj web aplikaciji, pri čemu sustavno mijenja dijelove zahtjeva koristeći unaprijed definirane rječnike ili generirane uzorke.

Iako je Wfuzz primarno razvijen kao alat za fuzzing, u praksi se vrlo često koristi i za napade grubom silom na web autentifikacijske mehanizme, primjerice za ispitivanje otpornosti login formi koje koriste korisničko ime i zaporku. U takvim scenarijima Wfuzz omogućuje automatizirano isprobavanje velikog broja kombinacija vjerodajnica te analizu odgovora poslužitelja kako bi se identificirale uspješne autentifikacije.

Wfuzz se izvršava iz naredbenog retka te je namijenjen isključivo za rad nad web aplikacijama. Njegov osnovni princip rada temelji se na korištenju posebne ključne riječi FUZZ, koja se unutar URL-a, parametara, zaglavlja ili tijela HTTP zahtjeva dinamički zamjenjuje vrijednostima iz rječnika ili drugim definiranim skupovima podataka.

Alat podržava širok raspon funkcionalnosti, uključujući:

- fuzzing URL-ova i direktorija
- fuzzing GET i POST parametara
- fuzzing HTTP zaglavlja i kolačića
- testiranje autentifikacijskih mehanizama
- korištenje jednog ili više rječnika istovremeno
- filtriranje i skrivanje odgovora prema duljini, statusnom kodu ili sadržaju
- fuzzing REST API sučelja
- korištenje proxy poslužitelja

Zbog svoje fleksibilnosti i mogućnosti preciznog filtriranja odgovora, Wfuzz je pogodan alat za testiranje web aplikacija u kontroliranom okruženju te za procjenu sigurnosti autentifikacijskih sustava.

9. Tehničke značajke alata Wfuzz

Osnovna funkcionalnost Wfuzza temelji se na generiranju HTTP zahtjeva u kojima se jedan ili više dijelova zahtjeva dinamički mijenjaju pomoću FUZZ oznake. Svaka instanca FUZZ oznake povezana je s jednim rječnikom ili skupom podataka, čime se omogućuje paralelno testiranje više varijabli unutar istog zahtjeva.

9.1 Fuzzing mehanizmi

Wfuzz omogućuje fuzzing sljedećih elemenata HTTP zahtjeva:

- URL putanje – npr. otkrivanje skrivenih direktorija i endpointa
- GET parametara – fuzzing vrijednosti unutar URL-a
- POST parametara – fuzzing podataka poslanih kroz forme
- HTTP zaglavlja – uključujući User-Agent, Authorization i Cookie
- Kolačića (cookies) – ispitivanje sesijskog upravljanja

U kontekstu napada grubom silom na web autentifikaciju, Wfuzz se najčešće koristi za fuzzing POST parametara koji sadrže korisničko ime i zaporku.

9.2 Filtriranje i analiza odgovora

Jedna od ključnih tehničkih značajki Wfuzza je mogućnost filtriranja HTTP odgovora kako bi se smanjila količina nerelevantnih rezultata. Filtriranje se može temeljiti na:

- HTTP statusnom kodu (npr. 200, 302, 401, …)
- duljini odgovora
- broju linija ili riječi u odgovoru
- sadržaju odgovora (string matching)

Ova funkcionalnost posebno je korisna kod napada grubom silom, gdje se uspješna autentifikacija često razlikuje od neuspješne po duljini odgovora, statusnom kodu ili prisutnosti specifičnih poruka.

9.3 Podrška za hashiranje i enkodiranje

Wfuzz podržava različite oblike enkodiranja i hashiranja podataka koji se šalju u HTTP zahtjevima. To omogućuje testiranje aplikacija koje zaporke ne šalju u čistom tekstu, već ih prije slanja obrađuju kriptografskim funkcijama.

U kontekstu ovog rada promatra se hashiranje, pri čemu Wfuzz omogućuje korištenje ugrađenih hash funkcija za transformaciju kandidata zaporki prije slanja zahtjeva. Time je moguće simulirati ponašanje klijentskih aplikacija koje koriste hashirane vrijednosti zaporki tijekom autentifikacije. Wfuzz ima ugrađene funkcije za hashiranje:

- SHA-1
- SHA-256
- SHA-512
- MD5

9.4 Način rada i zastavice

Wfuzz se pokreće iz naredbenog retka korištenjem različitih zastavica koje definiraju:

- ciljni URL -u
- rječnike za fuzzing -z
- skrivanje povratnih kodova --hc
- broj uzastopnih konekcija -t
- kašnjenje između zahtjeva -s

Kombinacijom ovih zastavica moguće je precizno definirati način izvođenja fuzzing ili napada grubom silom, prilagođen specifičnoj web aplikaciji.

10. Instalacija i korištenje alata Wfuzz

Instalacija alata Wfuzz je vrlo jednostavna, sve što je potrebno napraviti je pokrenuti naredbu:

pip install wfuzz

Nakon instalacije, alat je dostupan izravno iz naredbenog retka.

Korištenje alata započinje definiranjem ciljnog URL-a i mjesta na kojem će se koristiti FUZZ oznaka. Rječnici koji sadrže kandidate za fuzzing koriste se kao ulazni podaci, a rezultati se analiziraju na temelju odgovora poslužitelja.

Primjer naredbe koja je korištena za dobivanje rezultata eksperimenta:

wfuzz -c -w wordlist.txt -t 20 -d "username=admin&password=FUZZ" -u http://10.0.2.15:5000/login --hc 401

Primjer ispisa alata:

```
********************************************************

* Wfuzz 3.1.0 - The Web Fuzzer                         *

********************************************************


Target: http://10.0.2.15:5000/login

Total requests: 2


=====================================================================

ID           Response   Lines    Word       Chars       Payload

=====================================================================


000000001:   401        0 L      2 W        12 Ch       "aaaaaa"

000000002:   200        0 L      2 W        16 Ch       "aaaaab"


Total time: 0

Processed Requests: 2

Filtered Requests: 0

Requests/sec.: 0
```

11. Djelotvornost alata Wfuzz u eksperimentu

Alat Wfuzz je vrlo specifičan za web aplikacije te nije u potpunosti fokusiran na brzinu kao što su Hashcat ili John the Ripper, zbog ovoga u eksperimentu vrlo često se dobivaju brzine u stotinama zahtjeva u sekundi. Naravno ovo nije loša stvar jer u svijetu web aplikacija brzine veće od 100 zahtjeva u sekundi nisu nimalo potrebne zbog vrlo velike raproširenosti Captcha izazova, ograničavanja brzine/zahtjeva korisnika i DDoS mitigacije, koje vrlo često limitiraju korisnike na 5-15 zahtjeva u sekundi.

Zbog ovih malih brzina (~500 zahtjeva u sekundi) vidimo da bi za objektivno slabu zaporku koja se sastoji od 6 malih slova engleske abecede trebali proći 26^6 zaporki te bi pri brzini od 500 zahtjeva u sekundi ovo trajalo 617831 sekundi, to jest malo više od jednoga tjedna kako bi iscrpili sve mogućnosti.

Iz ovoga razloga alat nije pokrenut u obliku napada grubom silom, nego je u fokusu eksperimenta bio napad rječnikom (eng. dictionary attack). U napadu rječnikom je korišten poznati rockyou.txt rječnik sa vrlo često korištenim zaporkama.

Zbog male brzine alata u usporedbi sa Hashcat i John the Ripper te drugih tehničkih problema koji će biti spomenuti kasnije, je smanjena veličina datoteke rockyou.txt s 14 344 392 linija na 25 000 linija te je onda datoteka common s 1000 najčešće korištenih zaporki smanjena na 901 šifri koje se nalaze u prvih 25 000 linija rockyou.txt datoteke.

Server side hashing	Nehashirano	MD5	Argon2id
Common(901)	269 (3874s)	208 (2363s)	70 (2142s)

11. 1. tablica uspješno pronađenih zaporki i vrijeme potrebno

Dobiveni brojevi su broj zaporki koje su pronađene te vrijeme trajanja eksperimenta. Eksperiment je bio postavljen da sliči korištenju alata u pravom svijetu gdje web aplikacija prima ne hashirane zaporke te ih hashira i uspoređuje za hashevima u bazi podataka. U eksperimentu je alat pokrenut bez hash algoritma te je hashiranje napravljeno na strani servera.

Eksperiment je pozivao proces(dretvu) Wfuzz nad serverom te je pri uspješnoj prijavi promijenjena zaporka te je ponovno pozvan novi proces(dretva) Wfuzz. Pri obavljanju eksperimenta se ukazao problem kada je jedan proces Wfuzz-a trajao ~5 minuta, tada se iznenadno zaustavio taj proces i skripta ne bi mogla nastaviti jer bi proces došao do indeksa ~50 000, te ne bi mogao nastaviti dalje od tog indeksa, ovo je također bio problem i u finalnoj verziji eksperimenta, no nije odlučeno da se rječnik još više smanji jer bi tražene šifre činile prevelik podskup samog rječnika i time bi se smanjila povezanost eksperimenta i samog korištenja alata van eksperimenta. Ako bi se pojavila šifra sa indeksom većim od 100 000 unutar rockyou.txt rječnika, onda bi eksperiment ušao u dead-lock jer ne bi mogao dobiti uspješan login. Ovo je glavni razlog za smanjenje common i rockyou.txt rječnika. Važno je napomenuti da ovo nije problem alata, jer alat pokrenut direktno iz naredbenog retka vrlo lako prelazi rječnike veličine milijuna linija, nego je ovo vjerojatno problem sa pokretanjem alata iz Python skripte.

Ovim eksperimentom smo pokazali da je alat djelotvoran za probijanje vrlo često korištenih i default zaporki, no zbog malog broja zahtjeva u sekundi potrebno je odbaciti Wfuzz kao alat za napad grubom silom pri testiranju web aplikacija. Također je važno napomenuti da je alat uvelike ograničen brzinom odgovora poslužitelja, zaštitnim mehanizmima poput rate limitinga i CAPTCHA sustava, no bi mogao biti od koristi u testiranju otpornosti na automatizirane napade.

12. Opis alata RainbowCrack

RainbowCrack je alat za probijanje hasheva zaporki baziran na konceptu „Rainbow tablica“ koje je prvo osmislio Philippe Oechslin. Ideja tih tablica je da se korištenjem heurističkih algoritama optimizira standardni time-memory tradeoff te se time dobije znatno ubrzanje pri probijanju hasheva. Tablica se sastoji od dugih lanaca čiji su elementi nizova znakova. Početni niz lanca se odlučuje nasumično iz prostora mogućih zaporki. Početni niz se korištenjem hash funkcije pretvara u sljedeći element koji se onda korištenjem posebne redukcijske funkcije vraća u prostor mogućih zaporki. Ponavljanjem ovog procesa se dobije jedan lanac. Rainbowcrack po svakom lancu traži zadani hash i kad ga pronađe se vrati jedno mjesto kako bi pročitao znakovni niz od kojeg je nastao čime je taj hash dešifriran.

Alat prije početka probijanja zahtjeva stvaranje potrebne tablice. One se generiraju posebno za odabrani hash algoritam i za odabrani prostor ključeva odnosno zaporki. Generiranje tablice izrazito je vremenski i prostorno zahtjevno. Kada je tablica spremna, korisnik može efikasno pretražiti je sa zadanim hashom.

RainbowCrack je danas zastarjeli alat. U vrijeme nastanka bio je uspješan jer algoritmi za koje je namijenjen su relativno jednostavni (poput MD5) te je koncept bio nov i zaštite protiv ovakve vrste napada nisu bile razvijene. Alat je kasnije naslijedio RainbowCrack Improved with Multithreading koji je znatno moćniji zbog uporabe indeksiranih tablica i mogućnosti višedretvene obrade, unatoč tome, sveukupne sposobnosti alata su vrlo ograničene. Noviji hash algoritmi su svi otporni zbog uporabe salt-a ili kompleksnosti samih hash funkcija što znatno otežava izradu pripadne tablice i pretragu iste.

Dostupne funckionalnosti RainbowCrack-a:

- generiranje Rainbow tablica s dostupnim algoritmima (LM, NTLM, MD5, SHA-1, SHA-256), ovo podrazumijeva ne indeksirane, sporije tablice
- sortiranje i optimiziranje postojećih tablica
- konverzija tablica između različitih verzija RainbowCrack-a i različitih formata platformi
- spajanje manjih tablica u veće te razdvajanje
- ispis detaljnih podataka o tablici te verifikacija tablica i detekcija loših lanaca
- uporaba tablica za dešifriranje više hasheva istovremeno pomoću efikasne pretrage
- moguće je koristiti i tablice koje nisu lokalno generirane nego su preuzete s raznih izvora

RainbowCrack Improved with Multithreading uključuje sve navedeno, ali s dodanom mogućnosti višedretvene izvedbe što mnogostruko ubrzava rad. Niti jedna varijanta ovog alata ne koristi GPU što je veliko ograničenje.

Sveukupno, ovaj alat je danas spor i neefikasan, alati poput hashcat-a puno su brži i lakši za upotrijebit te ne zahtjevaju ogromne diskove za pohranu tablica. Za neke algoritme i dalje je funkcionalan i uspijeva u razumnom vremenu, no za algoritme već poput SHA-1, pretrage traju vrlo dugo za ikakve značajne parametre zaporki. Svrha alata više je za demonstraciju koncepta Rainbow tablica nego za praktičnu uporabu zbog visokog vremenskog zahtjeva.

13. Instalacija i uporaba alata RainbowCrack

Instalacija alata je jednostavna. Može se preuzeti s github stranice alata ili sa službene stranice projekta.

Paket uključuje zasebne programe za sve navedene funkcije. Pokreću se jednostavno s naredbama u terminalu.

Za kratki pregled uporabe:

```
rcrack --help
```

Generiranje tablice:

```
rtgen ntlm ascii 1 6 0 2400 1000000
```

Pripadni argumenti su: hash algoritam (ntlm), charset (ascii), duljina zaporke min/max (1, 6), indeks tablice (0), duljina lanaca (2400), broj lanaca (1000000)

Sortiranje tablice:

```
rtsort .
```

Inspekcija tablice:

```
rtdump *.rt
```

Pretraga hash-a:

```
rcrack . -h 5f4dcc3b5aa765d61d8327deb882cf99
```

Ovdje je točkom zadan trenutni direktorij kao putanja do Rainbow tablice te je hash zadan u naredbenoj liniji. Postoje i dodatne opcije koje su ovdje upotrebljive:

- `-l` uporaba datoteke s hashevima
- `-o` datoteka u koju se spremaju rezultati
- `-t` datoteka s tablicom umjesto direktorija
- `-s` završetak rada nakon prvog pronađenog rezultata
- `-p` ispis samo plaintext rezultata
- `-v` detaljni ispis
- `-c` maksimalni broj tablica za uporabu
- `-f` forsiraj obradu iako su tablice nekompatibilne
- `-n` isključuje neke interne optimizacije
- `-i` ispis o učitanim tablicama
- `-q` reduciran ispis

RainbowCrack Improved with Multithreading koristi iste naredbe ali sa sufiksom i_mt (npr. rcracki_mt)

Dodatno, pri pokretanju rcrack programa moguće je specificirati sesiju opcijom -s <ime-sesije> čime se napredak sprema u .session datoteku u radnom direktoriju. Ako je radnja prekinuta, proces se može nastaviti pokretanjem te iste naredbe bez gubitka napretka.

14. Tehničke značajke alata RainbowCrack

RainbowCrack se bazira na *time-memory tradeoff-u* koji koristi Rainbow tablice za optimizaciju probijanja hasheva. Alat je brži od standardnog *tradeoff-a*, ali zbog toga nema garantiran uspjeh. Pri generiranju lanaca može se dogoditi kolizija zbog koje se ukupni broj ključeva u tablici smanjuje. Zbog ovoga je konačni algoritam heuristički te tablica ne uspije uvijek pronaći zaporku za pripadni hash.

Model rada:

- Koriste se unaprijed generirane tablice koje su posebno pripremljene za brzu pretragu
- U zamjenu za ogroman diskovni prostor i vrijeme provedeno na predračun dobije se brz pronalazak rješenja
- Pregled tablice je deterministički, u prijašnjoj fazi se eliminira potreba za nasumičnim pogađanjem

Podržani hash algoritmi:

- LM
- NTLM
- MD5
- SHA-1
- SHA-256

Generiranje i upravljanje tablicama:

- Potpuna kontrola nad specifikacijama generirane tablice (algoritam, charset, duljina zaporki, duljina lanaca, broj lanaca)
- Sortiranje i optimizacija tablica
- Spajanje i razdvajanje tablica
- Pregled metapodataka tablica i verifikacija integriteta

Karakteristike performanse:

- Isključivo koristi CPU
- Vrlo intenzivna uporaba diska zbog same pretrage izrazito velikih tablica
- Brza pretraga nakon svih priprema
- Generiranje tablica izrazito sporo te zahtjeva velik prostor
- Djelotvornost ovisi najviše o vrsti pohrane (HDD, SSD) te zatim o brzini procesora

15. Djelotvornost alata RainbowCrack u eksperimentu

U eksperimentu su korištene tri liste hasheva.

Prva lista sadrži 2400 zaporki automatski generiranih koje ne zadovoljavaju minimalne zahtjeve sigurnosti, sadrže premalo znakova, nemaju varijaciju velikog/malog slova ili ne sadrže specijalne znakove. Time su ove zaporke izrazito nesigurne te se očekuje znatan uspjeh u probijanju. Datoteka s ovim zaporkama zove se „bad_sha1.txt“.

Druga lista sadrži tisuću najčešćih zaporki. One su nesigurne jer se u napadima često koristi rječnik koji ih sadrži. Osim što su česte, mnoge ne zadovoljavaju ranije navedene uvjete sigurnosti. RainbowCrack ovdje nema nikakvi prednost jer je ograničen što se tiče funkcionalnosti na samo jednu vrstu napada. Datoteka s ovim zaporkama zove se „common_sha1.txt“.

Treća lista sadrži tisuću zaporki nalik na zaporke generirane raznim menađerima zaporki, one sadrže 16 do 20 znakova te koriste i mala i velika slova i specijalne znakove. Alat ovdje neće moći ništa jer Rainbow tablice za zaporke s više od deset znakova su izrazito skupe i vremenski i prostorno. Datoteka s ovom listom zove se „locker_sha1.txt“.

Jedini algoritam od pripremljenih koje RainbowCrack može prihvatiti je SHA-1 zbog svoje zastarjelosti. Algoritmi poput Argon2id su sasvim nemogući zbog svoje kompleksnosti koja daleko nadmašuje sposobnosti Rainbow tablica te zbog uporabe salt-a koji sasvim onemogućuje njihov koncept.

Datoteka	bad	common	locker
SHA-1	1224 (12.5 h)	964 (2.2 h)	0

15. 1. tablica uspješno pronađenih zaporki za SHA-1 po listama zaporki

Dobiveni brojevi su uspješno probijeni hashevi, a u zagradama stoji vrijeme nakon kojeg je alat završio rad. Za ovaj eksperiment korištena je tablica koja sadrži sve ključeve napisane malim slovima, brojevima i znakom razmaka te ne sadrži više od osam znakova ukupno.

Takva tablica naravno nije sadržavala ključeve s velikim slovima, zbog toga je otprilike 50% zaporki iz „bad_sha1.txt“ datoteke sasvim nemoguće obradit, a iz „common_sha1.txt“ datoteke tek par desetaka. Iz datoteke „locker_sha1.txt“ nije niti jedna zaporka bila u opsegu tablice tako da alat nije ništa mogao.

Vrijeme priloženo za dvije uspješne operacije se dijeli na tri odvojene faze: faza pristupa disku, faza kriptoanalize i faza predračuna. Vrijeme potrošeno na pristup disku je najkraće, ono zauzima otprilike 400s u oba pokretanja jer ovisi samo o veličini tablice. Vrijeme potrebno za kriptoanalizu i predračun je mnogostruko duže jer ovise o broju hasheva, specifikacijama tablice, ali i o broju promašaja jer alat i za hasheve koji nisu sadržani u tablici mora proći kroz ove faze te bezvezno troši vrijeme kroz cijelu obradu, zbog ovoga uočavamo ogromnu razliku između dva pokretanja jer iako su imala isti red broja hasheva, prvo pokretanje imalo je skoro 1200 promašaja, a drugo ni pedeset.

Zajedno s već navedenim, eksperiment je i pokrenut sa znatno većom tablicom koja sadrži i ključeve s velikim slovima, no vrijeme obrade je znatno premašilo opseg eksperimenta, nakon 24 sata, alat je završio predračun za samo prva dva hasha.

Na temelju rezultata, ogromnih vremenskih i prostornih resursa potrebnih za upotrebu ovog alata te izrazitu limitiranost pri odabiru hash algoritma zaključujemo da je RainbowCrack izrazito neučinkovit alat.

16. Pregled algoritama za hashiranje, alata i zaporki

Hashing algoritmi	Vremenska težina	Memorijska težina	Trenutna razina sigurnosti
SHA-1	Vrlo niska	Vrlo niska	Niska, zastarjelo
SHA-1 + salt	Vrlo niska	Vrlo niska	Niska, zastarjelo
PBKDF2-HMAC-SHA256	Srednja – visoka (podesiva)	niska	Dobra, prihvatljiva
Argon2id	Srednja – visoka (podesiva)	Visoka (podesiva)	Vrlo dobra, preporučena

16. 1. tablica karakteristika korištenih algoritama za hashiranje

Tip zaporke	Utjecaj na vrijeme	Trenutna razina sigurnosti
Zaporke ispod minimalne norme (8 znakova, velika i mala slova, brojevi simboli)	Brzo vrijeme pogađanja, pogotovo pomoću offline napada	Niska
Često korištene zaporke	Najbrže vrijeme pogađanja, najefikasniji su rječnički napadi	Vrlo niska
Zaporke formata koje generiraju čuvari zaporki	Najdulje vrijeme pogađanja	Visoka

16. 2. tablica karakteristika zaporki korištenih u eksperimentu

Mogućnosti	John the ripper	Hashcat	RainbowCrack	Wfuzz
SHA-1	+	+	+	+
SHA-1 + salt	+	+	-	-
SHA-256	+	+	-	+
MD5	+	+	+	+
Bcrypt	+	+	-	-
PBKDF2-HMAC-SHA256	+	+	-	-
Argon2id	+	+	-	-
Višedretvenost	+	+	+	+
Koristi GPU	+	+	-	-
Koristi rječnik	+	+	-	+
Kontekstno pogađanje	+	+	-	-
GUI	-	-	-	-

16. 3. tablica mogućnosti korištenih alata

Brzina u hash/sek	John the ripper	Hashcat	RainbowCrack	Wfuzz
Nešifrirani tekst	10 450 000	-	-	576
SHA-1	11 246 000	39 564 100 000	~1000	376
SHA-1 + salt	19 952 000	39 363 500 000	-	-
SHA-256	11 494 000	14 720 700 000	-	355
MD5	21 760 000	109 200 000 000	~700	330
PBKDF2-HMAC-SHA256 (600000)	25	9 279	-	-
Argon2id (m=131072, t=3, p=1)	15	458	-	-

16. 4. tablica zabilježenih brzina u eksperimentu ovisno o algoritmu

hardver	John the ripper	Hashcat	RainbowCrack	Wfuzz
procesor	AMD Ryzen 7 4800H	i9-9980XE	i7-8750H	VM – 2 core (HOST: i5-11335G7)
Količina memorije	24 GB	96 GB	10 GB	4 GB
Grafička kartica	Radeon graphics	2 x RTX 2080-ti	-	-

16. 5. tablica hardvera korištenog u eksperimentu za pojedini algoritam