STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití Kali Linux pro prolamování hesel

Vojtěch Žák
Pardubický kraj

Pardubice, 2023

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

Využití Kali Linux pro prolamování hesel

Using Kali Linux for password cracking

Jméno: Vojtěch Žák
Škola: DELTA – Střední škola informatiky a ekonomie, s.r.o., Ke Kamenci 151, 530 03 Pardubice
Kraj: Pardubický kraj
Konzultant: Mgr. Josef Horálek, Ph.D.

Pardubice, 2023

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Pardubicích dne 26. března 2023 .....

Vojtěch Žák

Poděkování

Chci poděkovat Mgr. Josefu Horálkovi, Ph.D. za odborný dohled a vedení práce. Dále bych chtěl poděkovat RNDr. Janu Koupilovi, Ph.D. za pomoc s organizací. Nakonec bych chtěl poděkovat rodině a kamarádům za jejich neustálou podporu.

Anotace

Předložená práce je zaměřena na problematiku hesel, jejich ukládání a prolamování. Hlavním cílem projektu je praktické využití volně dostupných nástrojů pro prolamování hesel, postupy její implementace za účelem ověření síly a odolnosti uživatelských hesel. Práce je koncipována jako sada specifických laboratorních úloh využívajících nástroje John the Ripper a Hashcat s možností jejich využití na platformách jakákoliv podporovaných operačních systémech.

Klíčová slova

prolamování hesel; ukládání hesel; Kali Linux; John the Ripper; Hashcat

Annotation

The presented work is focused on the issue of passwords, their storage and cracking. The main objective of the project is the practical use of freely available password cracking tools, its implementation procedures in order to verify the strength and robustness of user passwords. The work is designed as a set of specific laboratory tasks using John the Ripper and Hashcat tools with the possibility of using them on any supported operating system platforms.

Keywords

cracking passwords; storing passwords; Kali Linux; John the Ripper; Hashcat

Obsah

1 Úvod

Tato práce se zabývá prací s hesly a jejich prolamováním. Obsahuje studijní materiál, který teoreticky popisuje principy ukládání a prolamování hesel. Následně obsahuje popis postupu instalace Kali Linux. Nakonec obsahuje tři praktická cvičení, ve kterých si čtenář může vyzkoušet prolamování hesel v praxi.

Cílem práce je vytvořit sadu řešených úloh pro využití distribuce Kali Linux v oblasti prolamování hesel. Pro naplnění cíle a objasnění komplexní problematiky bezpečnosti hesel jsou v první kapitole teoreticky popsány metody manipulace s hesly a jejich ukládání. V druhé kapitole jsou postupně detailně popsány způsoby instalace Kali Linuxu. V poslední kapitole jsou tři praktická cvičení, která ukazují, jak se prolamují hesla v praxi.

V teoretické části je popsán algoritmus prohledávání textů hrubou silou a principy ukládání a správy hesel. Následně je popsána instalace a nastavení Kali Linuxu a modů potřebných pro prolamování hesel. V praktické části práce jsou vytvořeny a podrobně popsány minimálně tři řešené úlohy na prolamování hesel při využití Kali Linux. Tato práce je vytvořena jako doprovodný studijní materiál, jehož cílem je seznámit čtenáře se základními principy ukládání a prolamování hesel.

Jednotlivé laboratorní úlohy mají jednotnou strukturu, která vždy představuje definování problémů, jejich řešení, podrobný popis implementace a využití dílčích nástrojů a ukázkové step-by-step motivační řešení. U dílčích částí je poukázáno na možné nejčastější problémy, které se při řešení dané úlohy vyskytují.

2 Ukládání a prolamování hesel

2.1 Úvod

Když jsi v dnešní době chceme na internetu založit účet, zeptají se nás obvykle na dvě věci: e-mail a heslo. Pomocí e-mailu říkáme, na jaký účet se chceme přihlásit. Heslo zase slouží k tomu, abychom dokázali, že jsme skutečnými vlastníky daného účtu a nejedná se o neoprávněného uživatele.

Občas se ale stane, že útočníci ukradnou účty i s hesly. Jsou tato hesla nějak chráněna? Je vůbec možné je nějaký způsobem chránit? A jak vůbec se taková hesla prolamují?

2.2 Ukládání hesel

Abychom mohli zkontrolovat, zda uživatel zadal správné heslo, musíme si ho někde uložit. Existuje několik způsobů, jak ukládat hesla. Většina z nich je špatná a neměla by se používat. I přes to tu jsou i dnes zaznamenávány případy úniku databází hesel, které byly uloženy nesprávně.

Žádný způsob ukládání hesel není dokonalý. Každé heslo se dá prolomit. Problém je s náročností prolomení hesla, které se odvíjí od časové náročnosti funkce na výpočet. Například heslo převedené pomocí hashovací funkce MD5 je možné prolomit do několika hodin. Ale když použijeme Argon2 s vysokými nároky na výkon, můžeme se dostat i na několik miliard let.

2.2.1 Prostý text

Nejlehčí způsob, jak ukládat hesla, je ukládání beze změny.[3]

Chceme-li přidat účet, zapíšeme email a heslo v takové podobě, v jaké je uživatel bude zadávat. Jako například v tabulce č. 1. Pokud se uživatel bude chtít přihlásit, zadá email a heslo. Následně najdeme řádek se zadaným emailem a zkontrolujeme, zda zadané heslo odpovídá heslu v databázi.

ID	Email	Heslo
1	potr123@example.com	1234
2	Honza56@example.com	Slunicko99
3	Pavel8@example.com	Slunicko99
4	Trman13@example.com	fvriFQcKEq

Tabulka 1: Tabulka s hesly uložená v prostém textu

Výhody této metody jsou v jednoduchosti. Samotná kontrola, zda je zadané heslo správné či ne, spočívá pouze v porovnání textu. Zároveň je tato metoda nenáročná na výkon. Vyhledávání a porovnávání textu jsou základní operace, které má většina programovacích jazyků a databází. Navíc tyto operace jsou ještě optimalizovány, což může ve smyslu manipulace s hesly vytvářet bezpečnostní díry (např.: Timing attack).

Hlavní nevýhodou je, že samotná hesla nejsou nijak chráněna. Tudíž pokud útočník hesla ukradne, nemusí je nijak prolamovat.

2.2.2 Šifrování

Šifrování slouží k převedení nezabezpečených dat na data šifrovaná. Pro dešifrování dat (převedení dat na data čitelná) musíme znát nějakou znalost. Většinou se pro šifrování

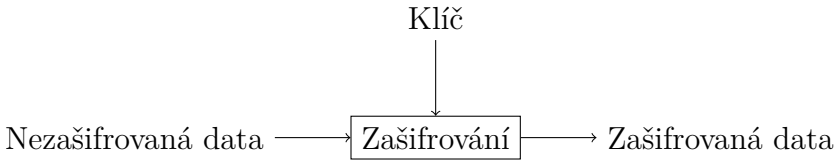
a dešifrování používají klíče (hesla).[3]

Šifrování existovalo již za Julia Caesara. Ten používal Caesarovu šifru. Tato jednoduchá šifra spočívá v posunutí písmenek abecedy o daný počet kroků. Caesarova šifra spadá do období klasické kryptografie, která trvala do poloviny 20. století. Šifry z tohoto období se vyznačují tím, že k šifrování stačí tužka a papír. Během první poloviny 20. století začali vznikat šifrovací stroje. Mezi ně patří například Enigma. K šifrování musel být použit šifrovací stroj, ale stále se šifrovala písmena, popř. další znaky. To změnily počítače a s nimi i nástup moderních šifer. Ty nepracují se znaky, ale s bity. To umožňuje šifrovat nejen text, ale jakákoliv data (např.: obrázky, zvuk, ...).

Existují dva hlavní typy moderních šifer: symetrické a asymetrické.

2.2.3 Symetrické

Symetrické šifry používají jeden klíč k šifrování i dešifrování. Tím se podobají šifrám z klasické kryptografie.



Obrázek 1: Symetrické zašifrování

Obrázek 2: Symetrické dešifrování

Mají dvě důležité vlastnosti: jsou rychlé a nejsou náročné na výpočetní výkon. Z těchto dvou důvodů se používají pro šifrování souborů.

Pokud bychom chtěli použít symetrické šifrování pro ukládání hesel, postupovali bychom následovně. Nejdříve musíme vybrat šifru, kterou budeme používat. V našem případě použijeme AES. Jedná se standardizovaný šifrovací algoritmus, který je široce používán. Konkrétně zvolíme AES-128, který bude vyžadovat klíč o velikosti 16 znaků. Náš klíč bude AhojJakSeMas1234. Je dobré připomenout, že moderní šifry pracují na úrovni bitů. Tudíž záleží na samotné implementaci, jaké kódování textu použijeme. V našem případě jsme pro převedení klíče použili kódování ASCII.

Nyní vezmeme příklad jednoduché databáze s hesly z předchozí kapitoly a zašifrujeme hesla pomocí AES.

ID	Email	Heslo
1	potr123@example.com	5twQ5uGZ9ZkxFZ9JoCeyZQ==
2	Honza56@example.com	uEWKo1vQAkEPiN5XOEaJAw
3	Pavel8@example.com	uEWKo1vQAkEPiN5XOEaJAw
4	Trman13@example.com	8ujeJ36gNZtCuepf5OMALg==

Tabulka 2: Tabulka s hesly zašifrované pomocí symetrické šifry

Kdybychom přidávali nového uživatele, před zapsáním hesla do databáze bychom ho zašifrovali. Při přihlášení máme dvě možnosti, jak zkontrolovat, jestli se heslo shoduje s heslem v databázi:

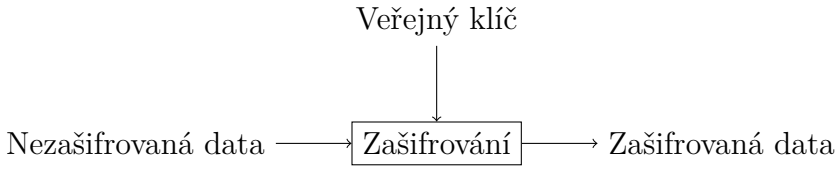
- Nejdříve vezmeme zašifrované heslo z databáze. Následně ho rozšifrujeme a dešifrované heslo porovnáme se zadaným heslem.
- Zadané heslo zašifrujeme a potom porovnáme se zašifrovaným heslem z databáze.

Výhoda oproti ukládání hesel v prostém textu je, že pokud nám někdo ukradne hesla, bude je muset dešifrovat. Tento způsob zabezpečení je při použití kvalitních šifer bezpečný.

Problém nastává v tom, že abychom mohli samotná hesla šifrovat, musíme někde mít také uložený klíč. Pokud útočník kromě hesel ukradne i tento klíč, tak je schopný všechny hesla dešifrovat.

2.2.4 Asymetrické

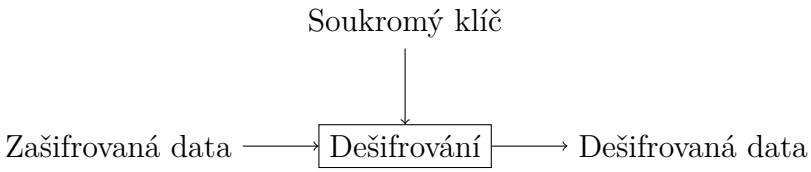
Asymetrické šifry používají dva klíče: veřejný klíč (šifrovací) a soukromý klíč (dešifrovací). Tento způsob šifrování nebyl dříve možný, protože většinou využívá složitou matematiku, která nejde bez počítače vypočítat.[4]



Obrázek 3: Asymetrické zašifrování

Oproti symetrickému šifrování je pomalejší a náročnější na výkon. Proto se používá pro malé soubory nebo zprávy. Typické použití je v kombinaci se symetrickými šiframi, kdy pomocí asymetrické šifry zašifrujeme klíč k symetrické šifře.

Postup zašifrování naší příkladové tabulky s hesly se bude podobat postupu u symetrického šifrování. Nejdříve vybereme konkrétní asymetrickou šifru. V našem případě použijeme



Obrázek 4: Asymetrické dešifrování

RSA. Následně si budeme muset vygenerovat veřejný a privátní klíč. Minimální doporučená délka klíče je 2048 bitů. Naše vygenerované klíče vypadají následovně[1]:

Veřejný klíč: MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCA ...

Privátní klíč: MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlA ...

Nyní vezmeme tabulku č. 1 a hesla zašifrujeme pomocí veřejného klíče[2].

ID	Email	Heslo
1	potr123@example.com	nY3Ha79oK8 ...
2	Honza56@example.com	Z/kyw96BGa ...
3	Pavel8@example.com	Z/kyw96BGa ...
4	Trman13@example.com	i88rqDXsU1 ...

Tabulka 3: Tabulka s hesly zašifrované pomocí asymetrické šifry

Kdybychom přidávali nového uživatele, před zapsáním hesla do databáze bychom ho zašifrovali veřejným klíčem. Při přihlášení máme stejně jako u symetrické šifry dvě možnosti, jak zkontrolovat, jestli se uživatel zadal správné heslo:

- Nejdříve vezmeme zašifrované heslo z databáze. Následně ho rozšifrujeme pomocí privátního klíče a dešifrované heslo porovnáme se zadaným heslem.
- Zadané heslo zašifrujeme pomocí veřejného klíče a potom porovnáme se zašifrovaným heslem z databáze.

Na rozdíl od symetrické šifry, kdy žádná metoda nebyla lepší, zde záleží, jakou metodu vybereme. Pokud budeme používat první způsob, budeme k manipulaci s hesly potřebovat jak veřejný, tak i privátní klíč. Zatímco u druhé metody potřebujeme pouze veřejný klíč a privátní klíč nemusíme ani generovat.

Při používání prvního způsobu je stejný problém jako u symetrické šifry. Pokud útočník ukradne jenom databázi hesel, hesla jsou chráněná. Avšak když ukradne i privátní klíč, který máme někde uložený, hesla může dešifrovat.

[1] Klíče byly zkráceny.
[2] Zašifrovaná hesla byla zkrácena.

Ale když budeme používat druhý způsob, kde nikdy privátní klíč nepoužíváme, útočník může ukrást pouze databázi a veřejný klíč. I přes to, že útočník ukradl klíč, kterým šifrujeme, tak nedokáže hesla dešifrovat, protože pro dešifrování se používá privátní klíč. Ten však nemáme nikde uložený, protože ho nepotřebujeme. Tato metoda se může zdát dostatečně bezpečná, což v praxi platí, avšak teoreticky je možné najít k veřejnému klíči klíč privátní. Pokud by tedy dokázal útočník získat privátní klíč touto metodou, mohl by opět dešifrovat všechny hesla.

2.2.5 Hashovací funkce

Hashovací funkce (též také jako hašovací funkce) je algoritmus pro převedení jakýkoliv vstupních dat na konstantně stejně dlouhý výstup. Tento výstup je označován jako otisk (fingerprint) nebo hash. To, jakým názvem výstup označíme, záleží na tom, kde hashovací funkci použijeme.[1, 3]

Využití hashovacích funkcí je široké. Příkladem mohou být hashovací tabulky, kontrolní součty, metody vyhledávání, nebo kryptografické hashovací funkce.

Jsou dvě typické vlastnosti hashovacích funkcí:

- Výstup o konstantní délce – nezáleží na délce vstupu, hashovací funkce vždy produkuje stejně dlouhý výstup
- Jednosměrnost – z výstupu hashovací funkce nelze získat původní vstup

Právě jednosměrnost je jeden z hlavních důvodů, proč se hashovací funkce (konkrétně kryptografické) používají pro hashování hesel. Když bychom si chtěli převést logiku hashovací funkce na šifrování[3], mohli bychom říct, že dokážeme zašifrovat jakoukoliv zprávu bez klíče. Zašifrovaná zpráva by následně nešla dešifrovat.

Obrázek 5: Ukázka teoretického jednosměrného šifrování

Nyní tuto teoretickou ukázku převedeme se správnými pojmy na graf, jak funguje hashovací funkce.

Obrázek 6: Princip hashovací funkce

[3]Tento převod slouží pouze jako vysvětlení jednosměrnosti hashovacích funkcí. Ostatní vlastnosti nejsou brány v úvahu.

Pokud bychom v tabulce č. 1 chtěli převést hesla na hash, budeme postupovat následovně. Nejdříve si vybereme hashovací funkci, kterou budeme používat. V naší ukázce použijeme MD5. Následně každé heslo převedeme pomocí této funkce na hash, který uložíme do tabulky. To samé bychom udělali, kdybychom přidávali nový účet.

ID	Email	Heslo
1	potr123@example.com	81dc9bdb52d04dc20036dbd8313ed055
2	Honza56@example.com	be8bf6952c87565b106c1c82706230db
3	Pavel8@example.com	be8bf6952c87565b106c1c82706230db
4	Trman13@example.com	5ba3922fc19674677cad5a0effc76bd2

Tabulka 4: Tabulka s hesly zahashované pomocí MD5

Kontrola při přihlášení funguje tak, že vezmeme zadané heslo a převedeme ho pomocí hashovací funkce na hash. Následně tento hash porovnáme s hashem z databáze.

Pokud by útočník ukradl naši databázi uživatelů, hesla zůstanou v bezpečí. To je způsobeno tím, že hashovací funkce jsou jednosměrné. Útočník teoreticky nemá žádnou možnost z hashů získat původní hesla.

V praxi však toto neplatí. Existují způsoby, jak hashe prolomit a získat heslo. Pro každý hash platí, že je prolomitelný pomocí útoku hrubou silou, záleží pouze na čase, jak dlouho tento útok potrvá, než hash prolomíme. Tento čas, který potřebujeme k získání původního hesla, se tudíž snažíme udělat co největší. Pokud prolomení hashe bude trvat dostatečně dlouho (déle než je lidský život), samotný útočník by dříve zemřel, než by hash prolomil.

Když si vezmeme tabulku č. 4, tak už samotný výběr typu hashovací funkce je špatný. MD5 má totiž spousty problémů. Ten největší je ten, že již byla několikrát prolomena, tudíž bylo doporučeno ji přestat používat. Avšak i přes to má neustále využití v oblasti hesel. Využívá se totiž jako příkladová hashovací funkce, když se ukazují příklady s prolamováním hesel. Zde se totiž její negativní vlastnosti (rychlost, paměťová nenáročnost, různé způsoby prolamování atd.) naopak hodí, protože pro ukázku není zrovna vhodné vybírat hashovací funkci, kterou by nešlo v rozumném čase prolomit.

Abychom tuto chybu napravili, zvolíme bezpečnější hashovací funkci. Tentokrát vybereme Argon2i.

ID	Email	Heslo
1	potr123@example.com	f68247ae2eddaa4be491703ccc90a238
2	Honza56@example.com	fcdc071559d7c80c1851a9946dca2a5b
3	Pavel8@example.com	5bd5a60ecf84a434fb42129de3a37964
4	Trman13@example.com	b9628e9499f8980685ca2eedadbb21ee

Tabulka 5: Tabulka s hesly zahashované pomocí Argon2i

2.2.6 Solení

Jak si můžeme všimnout v tabulce č. 4, jsou to dva totožné hashe. Je to z toho důvodu, že daní uživatelé si dali stejné heslo. V tom nastává problém ve dvou případech.[3, 6]

První případ je, že útočník prolomí heslo prvního uživatele. Následně zkontroluje, jestli se daný hash v tabulce nenachází vícekrát. Pokud ano, tak tyto hashe nemusí prolamovat, protože už je jednou prolomil. Druhý případ je, když útočník použije pro prolomení hesel duhovou tabulku.

Těmto dvěma případům se dá předejít takzvaným solením. Vždy, když přidáváme nové heslo do tabulky, vygenerujeme zároveň náhodný řetězec znaků (sůl). Tento náhodný řetězec znaků přidáme k heslu, čímž ho posolíme. Až nyní dané heslo převedeme na hash. V tabulce si následně kromě hashe uložíme i sůl. Až budeme kontrolovat, jestli zadané heslo je správné, vezmeme zadané heslo, přidáme k němu sůl, převedeme na hash a porovnáme s hashem z databáze.

2.2.7 Pepření

Pepření je stejné jako solení, kromě toho, že se výsledné náhodné znaky neuloží vedle hesla do databáze, ale někam jinam. Ukládacím místem může být jiná databáze, tabulka, nebo hardware k tomu určený.[6]

Při ukradení záznamů s hesly je ukradena i sůl. Zatímco při pepření by došlo k ukradení pouze hesel bez pepře. Následné prolamování hesel by se stalo náročnější, dokonce i při dostatečné délce pepře v podstatě nemožným.

2.2.8 Iterace

Iterace (někdy taktéž opakování) slouží k zvýšení náročnosti vytváření hashe. Iterace funguje tak, že místo toho, abychom po vygerování hashe ho hned uložili, vezmeme hash a dáme ho jako vstup do té samé hashovací funkce. Pokud budeme mít například dvě iterace, tak po tom, co převedeme heslo na hash, vezmeme vygenerovaný hash a znova ho převedeme na hash, který už uložíme.[7]

Tímto cyklením dochází k tomu, že pokud výpočet hashe trvá například 2 vteřiny, při dvou iteracích se už bude jednat o 4 vteřiny. Takto sice zpomalujeme kontrolu hesel při přihlášení, ale zároveň navyšujeme časovou náročnost na prolomení.

2.3 Algoritmus vyhledávání hrubou silou

Už na počátcích počítačů bylo potřeba pracovat s textem. Mezi základní operace patří i vyhledávání v textu. V dnešní době se tato funkce bere jako samozřejmost. Avšak dříve neexistoval žádný algoritmus, který by dokázal vyhledávat v textu. Prvním takovým algoritmem se stalo vyhledávání hrubou silou. I přes to, že v dnešní době existují rychlejší algoritmy na vyhledávání v textu, logika tohoto algoritmu se přenesla do jiných oblastí, například jako jeden ze způsobů prolamování hesel.[29]

Samotný popis algoritmu je velmi jednoduchý. Máme dva texty: zkoumaný text a vzor. Zkoumaný text je text, ve kterém hledáme, zatímco vzor je text, který hledáme. Příkladem může být následující situace: máme text *Ahoj jak se máš*, to je zkoumaný text. V něm chceme vyhledat slovo *jak*, které tudíž bude vzorem. Postup vyhledávání by se dal popsat snadno tak, že postupně zkoušíme všechny možnosti, dokud vzor ve zkoumaném textu nenajdeme, nebo nevyzkoušíme všechny možnosti.

Vyhledávání si můžeme ukázat na našem předešlém příkladě. Máme zkoumaný text *Ahoj jak se máš* a hledaný vzor *jak*. Nyní začneme postupně z leva porovnávat znaky mezi zkoumaným textem a vzorem. Pokud se znaky budou shodovat, porovnáme další znak. Pokud se znaky shodovat nebudou, posuneme vzor o jeden charakter doprava a začneme porovnávat znovu od začátku. V našem příkladě začneme tak, že porovnáme první písmeno ze zkoumaného textu (A) s prvním písmenem ze vzoru (j). Vzhledem k tomu, že A a j nejsou stejné znaky, posuneme vzor o jeden charakter doprava. Nyní porovnáme ne první, ale druhý znak ze zkoumaného textu (h) opět s prvním znakem ze vzoru (j). Tyto znaky se znova neshodují, tudíž posuneme doprava. Situace se opakuje i se znakem o. V následujícím kroku nám však nastane shoda. Čtvrté písmeno ze zkoumaného textu (j) se shoduje se prvním písmenem ze vzoru (také j). V tomto případě vzor neposouváme doprava, ale porovnáme další charaktery. Vezmeme tudíž následující charakter ze zkoumaného vzoru (mezera) a druhý charakter ze vzoru (a). Tyto dva charaktery se neshodují, tudíž posuneme vzor doprava a pokračujeme v porovnávání. Mezera ze zkoumaného textu a j ze vzoru se neshodují, takže posouváme vzor doprava. Tady nám nyní postupně nastávají tři shody, šestý až osmý znak ze zkoumaného textu se shodují se všemi znaky ve vzoru. To znamená, že jsme našli vzor ve zkoumaném textu.

Pokud bychom ale vzali text *Mám se dobře* a v něm opět hledali vzor *jak*, došli bychom nakonec zkoumaného textu, aniž by se nám shodovaly všechny znaky. V tomto případě zkoumaný text hledaný vzor neobsahuje.

Tento algoritmus je velmi jednoduchý a jeho logika, zkusit všechny možnosti, co jsou možné, se přenesla i do jiných oblastí. Velkou nevýhodou je však jeho časová náročnost. Na počátcích tento problém byl vyvažován snadnou implementací a nenáročností na výkon. Jenže postupem času se prohledávaná data začala zvětšovat a tento algoritmus začal být nahrazován rychlejšími algoritmy na vyhledávání (např. algoritmus K-M-P nebo

Karp-rabinův algoritmus). V dnešní době se algoritmus vyhledávání hrubou silou v praxi v podstatě nepoužívá, protože i ty nejméně výpočetně výkonné počítače dokáží pracovat s náročnějšími, ale rychlejšími algoritmy.

2.4 Prolamování hesel

Představme si, že jsme útočník, a právě jsme získali databázi účtů. Vedle názvů účtů a e-mailů jsou tu i hesla. Ta jsou však uložena jako hash. A právě tento hash je poslední ochrana, před získáním hesel.[2]

Útoků, které se zaměřují na prolamování hesel, je spousta. Některé se dají použít na jakoukoliv hashovací funkci, některé jsou zase specifické na konkrétní hashovací funkce.

2.4.1 Prolamování hrubou silou

Tento typ útoku vychází z algoritmu vyhledávání hrubou silou. Princip útoku spočívá v postupném generovaní znaků a zkoušení, jestli jejich hash není totožný s tím, který chceme prolomit. Tento typ útoku má jednu velkou výhodu. Je totiž možné s ním prolomit všechny typy hashovacích funkcí, a tudíž dokážeme prolomit jakékoliv heslo. Naopak velkým problémem je velká časová náročnost.[27, 16]

Logiku si můžeme ukázat na následujícím příkladu. Máme hash hesla *1234*. Zároveň známe, jaká hashovací funkce byla použita. Pokud chceme začít prolamování hrubou silou, začneme postupně generovat znaky. Z nich vždy uděláme hash pomocí hashovací funkce, která byla použita na vygenerování hashe hesla *1234*, a porovnáme, jestli se shoduje s hashem hesla *1234*. Pro zjednodušení řekněme, že budeme generovat pouze čísla. Začněme s heslem *0*. Pro toto heslo vygenerujeme hash a zkontrolujeme, jestli se neshoduje s hashem hesla *1234*. Zjistíme, že se neshoduje, tak vygenerujeme další heslo, což je *1*. Znova vytvoříme hash a znova zkontrolujeme, zda se náhodou hashe neshodují. Takto pokračujeme, dokud nevygenerujeme heslo *1234*. Po vygenerování hashe a kontrole zjistíme, že vygenerovaný hash se shoduje s hashem, který chceme prolomit. Nyní jsme dokázali prolomit heslo.

Tento příklad byl velmi jednoduchý. Problém nastává, když budeme muset tímto způsobem prolomit heslo mnohem delší. S každým přidaným znakem se totiž počet kombinací hesel zvyšuje ne násobením nebo sčítáním, ale umocňováním.

2.4.2 Duhová tabulka

Za myšlenkou duhových tabulek stojí nápad, že je zbytečné neustále generovat ty samé hashe pro ta samá hesla. Mnohem lepší je vygenerovat si pro dané heslo hash a tuto dvojici si uložit. Při prolamování hesel budeme pouze hledat, zda se daný hash vyskytuje v naší

duhové tabulce. Pokud ano, tak k tomuto hashi máme uložené i odpovídající heslo.[5]

Tento typ útoku má však jednu velkou nevýhodu. Počet kombinací hashů je většinou tak velký, že fyzicky není dostatek místa na uložení všech hesel a jejich hashů. Navíc se proti tomuto útoku dá bránit velmi jednoduše za pomoci dostatečně dlouhé soli. Tu totiž nemůžeme předem tušit, a tudíž pravděpodobně ani v naší duhové tabulce nebude daný hash.

2.4.3 Útok maskou

Útok maskou je upravená verze útoku hrubou silou. V tomto útoku se hlavně zneužívají lidské neduhy ve vytváření hesel. V útoku maskou negenerujeme všechny možné kombinace znaků, ale pomocí masky řekneme, na jakou pozici se mají jaké znaky generovat. Příkladem může být například to, že hodně lidí dává na konec hesla číslo a na začátek velké písmeno. Proto vytvoříme masku, kde na začátku bude vždy velké písmeno a na konci číslo. Na ostatní pozice necháme generovat všechny znaky.[27, 21]

Tímto útokem si sice snížíme počet hesel, která dokážeme prolomit. V našem předchozím příkladě by se jednalo o hesla, která by začínala jiným znakem, než velkým písmenem a končila by jiným znakem než číslicí. Na druhou stranu však snižujeme počet kombinací, které budeme muset vygenerovat, a tudíž v kratším čase prolomíme více hesel. Zbytek neprolomených hashů budeme muset prolomit jiným způsobem, nebo můžeme vyzkoušet jinou masku.

2.4.4 Slovníkový útok

Hesla, která si lidé volí, se opakují. Někteří lidé si nedokážou nebo nechtějí pamatovat velké množství složitých hesel, a tak si zvolí jedno, které následně používají na více místech zároveň. Tohoto nešvaru zneužívá právě slovníkový útok.[28, 16]

Slovníkový útok ke svému fungování potřebuje takzvaný slovník. Jedná se o soubor, který obsahuje list hesel. Tato hesla jsou často dvojího druhu. Buď se jedná o často používaná hesla, jako například *12345678* nebo *password*, nebo se jedná o list ukradených hesel z nějaké databáze uživatelů.

Samotný útok probíhá stejně, jako prolamování hrubou silou, pouze místo generování hesel postupně načítáme hesla ze slovníku. Tento typ útoku se většinou používá jako první, protože je velmi rychlý a dokáže prolomit ty nejprimitivnější hesla, která by jiným útokem mohla být prolomena za mnohem delší čas.

2.4.5 Slovníkový útok s pravidly

Pokud se podíváme na nejpoužívanější hesla na světě, najdeme zde například heslo *password*. To lze jednoduše prolomit slovníkovým útokem. Avšak pokud si uživatel zvolí heslo *PASSWORD1*, tak i přes to, že je velmi podobné předešlému heslu, ho slovníkový útok pravděpodobně nedokáže prolomit. Přitom by stačilo pouze heslo *password* zvětšit a přidat na konec jedničku. To právě umožňují pravila.[20]

Slovníkový útok s pravidly kromě slovníku navíc potřebuje sadu pravidel. Tato pravidla určují, co se s heslem ze slovníků má stát. Možností, co lze dělat, je spousty. Například zvětšování a zmenšování písmen, přidávání číslic a speciálních znaků na konec a na začátek, obrácení hesla atd. Tato pravidla se dají navzájem různě kombinovat.

Pomocí tohoto útoku se dají prolomit ta hesla, která se podobají v nějakém ohledu heslu ze slovníku, avšak nejsou zcela stejná.

2.4.6 Zneužití chyb

Tento typ útoku zahrnuje všechny známé chyby, které se objevily u různých hashovacích funkcí. Všeobecně platí, že čím více času uplynulo od chvíle, kdy daná hashovací funkce byla vydána, tím pravděpodobnější je, že se u ní objevila nějaká chyba.

Chyby můžeme dělit na tři kategorie: kritické, nekritické a teoretické. Kritické chyby umožňují útočníkovi zjistit původní data rychleji, než by to trvalo prolamováním hrubou silou. Pokud se taková chyba u hashovací funkce najde, hashovací funkce se stává prolomenou a není doporučeno ji již dále používat. Nekritické chyby jsou takové, které sice útočníkovi umožňují zjistit původní data, ale jsou pomalejší než prolamování hrubou silou. V tomto případě se hashovací funkce nestává prolomenou, ale je dobré si rozmyslet, jestli ji budeme nadále používat. Poslední jsou teoretické chyby, které sice fungují teoreticky, ale nejdou implementovat do praxe. Pokud je takováto chyba odhalena, hashovací funkce se nestává prolomenou a není větší důvod ji přestat používat. Musíme si však dávat pozor, zda ona překážka, která brání v implementaci nebyla překonána.

3 Instalace Kali Linux

3.1 Úvod

Pokud chceme začít s praktickým prolamováním hesel, budeme potřebovat k tomu určené nástroje. Ty můžeme stáhnout na jakýkoliv podporovaný systém. Avšak existují i operační systémy, které jsou zaměřené na etické hackování. Mezi ně se řadí i Kali Linux.

Kali Linux (často zkracováno pouze na Kali, dříve BackTrack Linux) je operační systém zaměřený na penetrační testování. Obsahuje mnoho nástrojů na testování zabezpečení a etické hackování. Mezi ně patří například nástroje na skenování počítačové sítě, reverzní inženýrství, ale i na prolamování hesel. Navíc předpřipravené prostředí přináší různé výhody. Příkladem může být třeba složka s již předpřipravenými slovníky pro slovníkový útok.[8]

Od verze 2023.1 je dostupná ke stáhnutí i nová distribuce nazvaná Kali Purple. Tato distribuce se místo útoku zaměřuje na obranu. Kromě jiných nástrojů, přináší i jinou strukturu v samotném fungování operačního systému. I když se jedná o speciální verzi Kali, a tudíž má podobné základní funkce, pro naše účely se tolik nehodí.

V následující sekci jsou popsány nejčastější možnosti zprovoznění Kali ve verzi 2023.1. Pokud je dostupná novější verze, je doporučeno postupovat podle aktuálního návodu na instalaci. Ten je dostupný na adrese `https://www.kali.org/docs/`. Zde jsou dostupné i ostatní typy instalací.

3.2 Příprava

Před samotnou instalací budeme ve většině případů potřebovat tři věci: obraz Kali, disk a program na zapisování obrazu na disk.

Abychom mohli Kali zprovoznit, budeme ho nejdříve muset stáhnout. Tyto obrazy[4] jsou většinou jako soubor s příponou *.iso*. Všechny dostupné verze se dají stáhnout z oficiálních stránek Kali na adrese `https://www.kali.org/get-kali/`.[9]

Další věcí, co budeme potřebovat, je disk, z kterého budeme Kali spouštět. Doporučený je flash disk, avšak můžeme použít i SD kartu. Minimální velikost disku pro většinu instalací je 4 až 8 GB. Doporučená velikost je 16 GB a více. Rychlost disku není nijak specifikována. Dopředu musíme počítat s tím, že obsah celého disku bude smazán a to včetně oddílů. Proto vždy před použitím disku pro instalaci musíme soubory na něm zkopírovat na jiný disk.

Poslední věcí je program na zápis obrazů na disk. V našem případě použijeme program Rufus, který je ke stažení na adrese `https://rufus.ie/en/`. Současná verze je 3.21. Pokud by vyšla nová verze a nejednalo by se o majoritní verzi (např.: 4.0), měl by postup zůstat podobný.[15]

[4]Obraz je kopie disku. V tomto případě se jedná o kopii Kali.

3.3 Live USB

Live USB je speciální metoda instalace operačního systému. Ten totiž neinstalujeme na počítač, ale na přenosný disk.[13] To má dvě hlavní výhody:

- Nijak nezasahuje do disku počítače – protože operační systém je spouštěný přímo z přenosného disku, nemá důvod nijak sahat na disk v počítači.
- Je přenosný – operační systém můžeme zapnout z jakéhokoliv podporovaného zařízení.

Kali má speciální obraz pro Live USB pojmenovaný jako *Live Boot*. Ten má ještě tři podverze:

- Kali – klasická verze Kali, která obsahuje základní nástroje.
- Everything – obsahuje všechny nástroje a je o hodně větší než klasická verze.
- Weekly Image – neotestovaná verze vydávána každý týden, která obsahuje nejnovější funkce.

Doporučená je klasická verze Kali. Obsahuje nástroje, které budeme potřebovat, nezabírá tolik místa, a navíc je nejvíce stabilní. Po stažení tohoto obrazu, vložíme přenosný disk do počítače a zapneme Rufus.[10, 11] V něm budeme postupovat následovně:

1. Ze seznamu *Device* vybereme disk, na kterém chceme Live USB.
2. V sekci *Boot selection* klikneme na tlačítko *SELECT*. Otevře se nám Průzkumník souborů, ve kterém vybereme ISO soubor Kali.
3. V sekci *Partition scheme* zvolíme, jaký typ oddílu Rufus vytvoří na disku. Pokud máme v počítači BIOS, tak zvolíme možnost *MBR*. Pokud bychom měli UEFI, zvolíme *GPT*.
4. Nyní klikneme na tlačítko *START*.
5. Může se stát, že vyskočí dialogové okno ISOHybrid. V tomto případě můžeme zvolit, v jakém módu bude obraz na disk zapsán. Zvolíme možnost *ISO* a potvrdíme.
6. Pokud vyskočí upozornění, že Rufus potřebuje stáhnout dodatečné soubory, požadavek odsouhlasíme.

Nyní musíme počkat, než Rufus zkopíruje všechny soubory na disk. Postup je vidět v ukazateli průběhu v sekci *Status*. Až Rufus dokončí všechny práce, lišta bude kompletně zelená a bude v ní napsáno *READY*. Nyní se z našeho přenosného disku stalo Live USB.

Abychom nastartovali Kali, budeme muset restartovat počítač a jít do BIOS (nebo UEFI) nastavení. V něm budeme potřebovat nastavit pár věcí:

1. Vypnout Fast Boot – ten může bránit v načtení externího disku před startem operačního systému.
2. Vypnout Secure Boot – ten nedovolí spustit nic, co není digitálně podepsané. V našem případě by to mohlo způsobovat problémy.
3. Nastavit bootovací zařízení na disk s Kali – většinou se jedná o list, ve kterém posuneme disk na první místo.
4. Nakonec nastavení uložíme a odejdeme z nastavení.

Následně se počítač může i několikrát restartovat, aby se změny aplikovaly. Po chvíli by se měla zobrazit bootovací obrazovka Kali. V ní zvolíme možnost *Live system*. Až se nám nastartuje operační systém Kali Linux, už můžeme začít například prolamovat hesla.

3.4 Čistá instalace

Pokud nechceme neustále spouštět Kali z přenosného disku a chceme ho používat jako klasický operační systém, musíme ho nainstalovat na počítač. Nejsnazší je čistá instalace. Při ní se smaže celý disk a naformátuje se na instalaci Kali. Proto, než se rozhodneme pro tuto instalaci, musíme být rozhodnutí, že skutečně nebudeme potřebovat současný operační systém, který běží na počítači. Když už se tak rozhodneme, musíme vytvořit zálohu všech souborů, které na disku máme. Pokud máme v počítači více fyzických disků, tak se naformátuje pouze ten, na který budeme operační systém instalovat. Ostatní disky by měly zůstat nedotčené.[12]

Abychom splnili minimální požadavky budeme potřebovat 20 GB velký disk a 2 GB paměti RAM. Doporučená velikost paměti RAM je 8 GB a více.

Počáteční postup je stejný jako u vytváření Live USB. Pouze budeme potřebovat jiný obraz Kali. Nyní totiž budeme stahovat soubor z kategorie *Installer Images*. Jsou zde čtyři možnosti:

- Installer – klasická verze Kali, která obsahuje základní nástroje.
- Weekly – neotestovaná verze vydávána každý týden, která obsahuje nejnovější funkce.
- Everything – obsahuje všechny nástroje a je o hodně větší než Installer.
- NetInstaller – stejný jako Installer, neboť jsou všechny balíčky jsou staženy z internetu během instalace, nezabírá tolik místa na přenositelném disku.

Doporučená verze je Installer. Po stažení postupujeme stejně jako u Live USB až do chvíle, kdy se nám zobrazí bootovací obrazovka Kali. V ní už neuvidíme možnost *Live system*, protože z disku už neumožňuje nastartovat operační systém. Zvolíme možnost *Graphical install*. Možnost *Install* má stejný postup, ale je v textovém rozhraní. Načte se nám instalace Kali, kde budeme pokračovat následovně:

1. Jazyk – zvolíme jazyk, ve kterém bude instalace i následně nainstalovaný operační systém. Doporučená je angličtina, ale můžeme klidně zvolit češtinu.

2. Geografická poloha – pokud jsme za jazyk zvolili češtinu, automaticky se nám zvolí Česko. Pokud jsme ale zvolili angličtinu, musíme zvolit *Other*, následně zvolit *Europe* a nakonec nastavit možnost *Czechia*. Pak se objeví obrazovka s upozorněním, že vybraná oblast neodpovídá zvolenému jazyku. Toto znemožňuje automaticky vybrat „kódování“ a musíme ho vybrat sami. Zvolíme *united states* (popřípadě pro Britskou angličtinu *united kingdom*).

3. Klávesnice – vybereme buď anglické nebo české rozvržení.

4. Síť – máme dvě možnosti připojení:

   - Ethernet kabel (eth) – pokud máme zapnuté DHCP na routeru, nemusíme nic nastavovat. Jestli ale používáme statické IP adresování, budeme muset všechny informace nastavit ručně.
   - Wifi (wlan) – z listu zvolíme Wifi, na kterou se chceme připojit. Pak zvolíme, jaké zabezpečení bezdrátová síť používá a zadáme heslo. Pokud máme zapnutou službu DHCP, IP adresa a ostatní informace se nám nastaví samy. V opačném případě je budeme muset nastavit ručně.

   Následně zadáme, jak se naše zařízení bude v síti jmenovat. To je ve výchozím nastavení nastaveno na *kali*, ale můžeme si zvolit vlastní název. Nakonec nastavíme výchozí adresu. Tu ve většině případů necháme prázdnou.

5. Vytvoření účtu – kromě root účtu Kali vyžaduje ještě jeden uživatelský účet. Prvně musíme vyplnit naše jméno, které se bude zobrazovat. Nemusí se však jednat o naše vlastní jméno, můžeme zadat jakýkoliv název. Následně vyplníme název našeho účtu, kterým se budeme přihlašovat. Název by měl začínat malým písmenem anglické abecedy a obsahovat pouze číslice a písmena z anglické abecedy. Nakonec si zvolíme heslo.

6. Disk – z listu vybereme možnost, že chceme použít celý disk. Pak vybereme disk, na který chceme operační systém nainstalovat. Následně zvolíme, že chceme všechny soubory na jednom oddílu. Poté se nám zobrazí souhrn změn. Zvolíme, že chceme ukončit upravování a změny zapsat na disk. Nakonec potvrdíme naše rozhodnutí.

7. Software – nyní budeme volit dvě věci:

- Desktopové prostředí – můžeme si zvolit ze tří možností: Xfce, GNOME a KDE Plasma. Pokud nevíme, které prostředí zvolit, necháme výchozí možnost.
- Kolekce nástrojů – v tomto listu zaškrtneme možnosti *top10* a *default*.

8. GRUB – pokud se nás instalace zeptá, jestli chceme nainstalovat GRUB na hlavní disk, zvolíme že ano.

Nyní máme již vše nainstalované, potom, co dáme pokračovat, se počítač restartuje. V tuto chvíli musíme jít do nastavení BIOSu (nebo UEFI). Zde vrátíme nastavení do původního stavu, vytáhneme přenosný disk z počítače, uložíme změny a odejdeme. Nyní se nám restartuje počítač a potom nastartuje GRUB, ve kterém zvolíme Kali Linux. Následně se stačí přihlásit do účtu, který jsme vytvořili během instalace.

4 Praktická cvičení

4.1 Prolamování hrubou silou

Požadavky

Program	Verze
Hashcat	6.2.6
John the Ripper	1.9.0

Použité soubory

Název	Popis
password.hash	Soubor obsahuje MD5 hash hesla *password*

4.1.1 Úvod

Prolamování hrubou silou je nejzákladnější metoda na prolamování hesel. Spočívá v generování postupných kombinací znaků. Zároveň se jedná o jeden z nejpomalejších způsobů. Proto se používá jako poslední možnost.

4.1.2 John the Ripper

V programu John the Ripper je prolamování hrubou silou nazýváno jako *Incremental mode*. Pokud bychom nechali automatický mód (nezvolili žádný mód), tak John the Ripper použije *Incremental mode* jako poslední možnost.[16, 17, 18, 19]

Incremental mode zapneme pomocí `--incremental`. Následně ještě předáme informaci, že se jedná o MD5 hash přes argument `--format`. Nakonec napíšeme název souboru, který hash obsahuje.

```
john --incremental --format=Raw-MD5 password.hash
```

Po spuštění se vypíše informace, že se načetl jeden hash typu Raw-MD5. Následně se spustí samotné prolamování. V terminálu není žádný výstup. Ten se zobrazí až poté, co nějaký hash prolomíme. Vypsaný řádek bude obsahovat hash a vedle něho prolomené heslo. V našem případě se vedle hashe zobrazí *password*. Po prolomení všech hesel se program vypne a budeme moci zadat další příkaz.

Pokud by trvalo prolamování příliš dlouho, můžeme program zastavit pomocí klávesové zkratky *Ctrl + C* nebo zmáčknout klávesu *Q*. Program se zastaví a hesla, která jsme už prolomili, si zapamatuje.

4.1.3 Statistiky

Během prolamování můžeme zmáčknout klávesu *S*, pomocí které se nám ukáže statistika aktuální situace. Vypíše se nám jeden řádek s několika informacemi.

První údaj `g` určuje celkový počet prolomených hesel. Následuje čas, jak dlouho daný útok už běží. Údaj `g/s` ukazuje počet prolomených hesel za vteřinu. `P/s`, `c/s` a `C/s` zobrazují počet hesel vyzkoušených za jednu vteřinu. A na konci řádku jsou aktuálně zkoušená hesla.

4.1.4 Vypsání prolomených hesel

Abychom vypsali hesla, která John the Ripper prolomil, použijeme možnost `--show`. Zároveň musíme specifikovat typ hashe. Většinou se jedná o stejný typ, jako jsme zvolili u samotného prolamování.

```
john --show --format=Raw-MD5 password.hash
```

Ve výstupu se nám zobrazí hashe a vedle nich odpovídající hesla, stejně jako u samotného prolamování.

4.1.5 Vymazání prolomených hesel

Prolomená hesla si John the Ripper ukládá do souboru *john.pot*. Pokud bychom chtěli znovu spustit příkaz na prolamování, program by se vypnul s hláškou, že daný hash je již prolomen. Abychom donutili program, aby znova prolomil již prolomené heslo, musíme soubor *john.pot* smazat. Soubor můžeme najít pomocí programu `find`.[22]

```
find / -name john.pot
```

Program nám vypíše všechny cesty, kde se soubor nachází. Můžeme soubor smazat manuálně (např. přes příkaz `rm`), nebo přidáme do přechozího příkazu argument `-delete`.

```
find / -name john.pot -delete
```

Nyní se nám nevypíše žádný výstup do konzole. Pro kontrolu, jestli se soubory skutečně smazaly, zadáme znova příkaz bez `-delete`. V konzoli by se neměla zobrazit žádná cesta.

4.1.6 Hashcat

Hashcat měl útok hrubou silou, avšak vývojem se z něho stal útok maskou. Neznamená to však, že by nešel zvolit, pouze se z něho stal speciální případ jiného útoku.[26, 27]

Typ útoku vybíráme pomocí argumentu `--attack-mode`. Tomu následně předáme typ útoku pomocí čísla. V našem případě chceme zvolit útok maskou, která má číslo `3`. Následně musíme specifikovat typ hashe, pomocí kterého je heslo hashované. Heslo v souboru *password.hash* je hashované pomocí hashe MD5. Proto předáme argumentu `--hash-type` parametr `0`. Potom zadáme název souboru, který hash obsahuje. Nakonec specifikujeme, jaké charaktery se mají použít a jak dlouhé hádané heslo může být. Abychom použili útoku hrubou silou, musíme použít všechny charaktery (*?a*), a zadáme ho tolikrát, kolik odhadujeme, že prolamované heslo má znaků.

```
hashcat --attack-mode 3 --hash-type 0 password.hash
   ?a?a?a?a?a?a?a?a
```

Na rozdíl od John the Ripper je Hashcat mnohem „ukecanější“. To může být výhoda i nevýhoda. Ve velkém množství výpisu se těžko orientuje, zato v něm najdeme více informací, které mohou být užitečné.

Hashcat nejdříve zobrazí informace o útoku, který se spustil. Prvně se ukáže, na kterých zařízeních se daný hash bude vypočítávat a jakou API použije. Většinou si Hashcat zvolí procesor s OpenCL API. Pokud je ale dostupná grafická karta, bude preferovat ji. V případě grafických karet od společnosti NVIDIA umí Hashcat využít CUDA rozhraní.

Vybrání zařízení provádí hashcat automaticky sám. Pokud bychom si chtěli zvolit, jaké zařízení má použít, použijeme argument `--backend-devices` nebo `--opencl-device-types`. Ve většině případů to není nutné, naopak by mohly nastat nějaké obtíže.

Po informaci o zařízeních se vypíše minimální a maximální délka podporovaných hesel, které umí Hashcat prolomit. Pokud by prolamované heslo bylo mimo hranice, Hashcat nedokáže dané heslo prolomit.

Údaj *Hashes* udává, kolik hashů bylo načteno, kolik z nich je jedinečných a kolik jedinečné soli bylo načteno. V našem případě by všude měla být jednička.

Následuje seznam optimalizací, které byly použity. Tyto optimalizace značně zrychlují hashování, nijak však neovlivňují samotný princip útoku.

Hashcat má vestavěnou funkci sledování teploty zařízení. Tento údaj se nazývá *Watchdog*. Pokud by byla překročena daná teplota, tak se Hashcat zastaví. K tomuto okamžiku však často nedochází.

A jako poslední ve výpisu je požadovaná velikost paměti, kterou Hashcat potřebuje. Pravidlem je, že čím více hesel najednou chceme prolomit, tím více paměti budeme potřebovat.

Nyní začne samotné prolamování. Stejně jako v John the Ripper se nám začnou vypisovat prolomená hesla. V našem případě se vypíše pouze jeden řádek s následujícím textem: `5f4dcc3b5aa765d61d8327deb882cf99:password`.

Během toho, co Hashcat prolamuje hesla, můžeme zmáčknout několik kláves. Nejdůležitější jsou dvě: *S* a *Q*. Pomocí klávesy *Q* ukončíme program. Pomocí klávesy *S* vypíšeme aktuální statistiky.

4.1.7 Statistiky

Při prolomení všech hesel, při stisku klávesy *S* nebo při ukončení vypisuje Hashcat *status* neboli statistiky.

V těchto statistikách je spousta informací. Nejdůležitější však jsou: *Time Estimated*, *Recovered* a *Progress*. Ostatní informace nejsou zbytečné, ale slouží buď v případě, kdy máme spuštěných více útoků najednou, nebo se dané informace dají odvodit.

Time Estimated (předpokládaný čas) předpovídá datum a čas, kdy budou vyzkoušeny všechny možnosti. Tento čas je vypočítán pomocí vydělení zbývajícími možnostmi rychlostí generování hashů. Kromě datumu a času je v závorce uveden zbývající čas od doby, kdy se statistika vypsala. Důležité je si uvědomit, že tento čas se může měnit, většinou z důvodu změny rychlosti generování hesel. To již závisí na samotném hardwaru. Například se postupně rychlost může snižovat s tím, jak se zařízení zahřívá.

Recovered (obnoveno nebo prolomeno) udává počet prolomených hesel. Jsou zde dvě kategorie: total a new. Total je počet všech prolomených a načtených hesel. New je počet hesel, která nebyla nalezena v souboru obsahujícím již prolomená hesla. První číslo před lomítkem je počet prolomených hesel, druhé číslo je celkový počet hesel.

Progress (postup) udává počet hesel, která byla již vyzkoušena. Pokud provádíme útok

hrubou silou nebo útok maskou, bude se jednat o počet kombinací znaků. V případě, že bychom prováděli slovníkový útok, bude se jednat o počet slov ve slovníku.

4.1.8 Vypsání prolomených hesel

Pro vypsání prolomených hesel použijeme možnost `--show`, stejně jako u John the Ripper. Zároveň specifikujeme typ hashe pomocí argumentu `--hash-type`.

```
hashcat --show --hash-type 0 password.hash
```

Program nám vypíše hashe a vedle nich odpovídající hesla.

4.1.9 Vymazání prolomených hesel

Stejně jako u John the Ripper si Hashcat již prolomená hesla ukládá do souboru. Proto pokud bychom chtěli znovu prolamovat již prolomený hash, musíme smazat tento soubor. Postup je stejný jako u John the Ripper, pouze místo souboru *john.pot* chceme smazat *hashcat.potfile*.

```
find / -name hashcat.potfile -delete
```

4.2 Útok maskou

Požadavky

Program	Verze
Hashcat	6.2.6
John the Ripper	1.9.0

Použité soubory

Název	Popis
password.hash	Soubor obsahuje MD5 hash hesla *password*
mask.hcmask	Soubor obsahuje masku pro Hashcat

4.2.1 Úvod

Prolamování hrubou silou má jeden zásadní problém a tím je rychlost. Proto vznikla takzvaná maska, se kterou omezíme znaky, které se mohou na danou pozici dosadit.

John the Ripper a Hashcat mají každý svoji definici, jak masku zapisovat. Je však pár základních možností, které mají společné. V následující tabulce jsou vypsané ty, které jsou společné a nejvíce se používají.

Maska	Charaktery
?l	abcdefghijklmnopqrstuvwxyz
?u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d	0123456789
?s	speciální charaktery
?a	všechny ASCII znaky

4.2.2 John the Ripper

John the Ripper má několik způsobů, jak složit masku. Ale základem je vždy argument `--mask`. Do něho se jako parametr dávají definice použitelných charakterů.[16, 17, 18, 21]

První způsob, jak zapsat masku je, že určíme pro každé místo použitelné charaktery. V našem příkladě chceme generovat hesla dlouhá osm znaků a použitelné charaktery, na všech místech budou malá písmena anglické abecedy. Proto jako parametr argumentu `--mask` zadáme `?l?l?l?l?l?l?l?l`. Je lepší masku dát do apostrofů, aby nenastaly problémy se speciálními znaky. Ty by mohly být špatně interpretovány příkazovým řádkem nebo samotným programem.

Protože útok maskou se bere jako vylepšení útoku hrubou silou, ostatní argumenty jsou stejné, jako v útoku hrubou silou. To znamená, že specifikujeme *Incremental mode* pomocí `--incremental` a zvolíme typ hashe. V našem případě se jedná o MD5 hash, proto jako parametr argumentu `--format` dáme `Raw-MD5`.

```
john --incremental --format=Raw-MD5
   --mask='?l?l?l?l?l?l?l?l' password.hash
```

Druhý způsob je, že v argumentu `--mask` určíme pouze povolené charaktery a délku určíme pomocí argumentů `--min-length` a `--max-length`. Abychom docílili toho samého, co v prvním způsobu, do `--mask` dáme parametr `?l`, do `--min-length` parametr `8` a do `--max-length` také `8`.

```
john --incremental --format=Raw-MD5 --mask='?l'
   --min-length=8 --max-length=8 password.hash
```

Tento způsob má výhodu v „pružnosti“. Představme si, že máme prolomit hesla, o kterých víme, že jsou dlouhá 8 až 16 znaků. Kdybychom používali první způsob (jenom `--mask`), museli bychom postupně manuálně přidávat do masky další znaky. Pokud bychom ale použili druhý způsob, stačí jenom dát do argumentu `--min-length` hodnotu `8` a do `--max-lenght` číslo `16`. Maska se nám bude automaticky sama natahovat.

4.2.3 Hashcat

Hashcat měl útok hrubou silou, ale nahradil ho právě útokem maskou. Proto neexistuje žádný argument, do kterého by se maska zadávala.[26, 27]

V Hashcatu se maska zadává na konec příkazu hned za souborem s hashem. V našem případě, kde chceme generovat hesla dlouhá osm znaků a pouze z malých písmen anglické abecedy, zadáme nakonec `?l?l?l?l?l?l?l?l`. Zároveň musíme zvolit útok maskou. To uděláme tak, že argumentu `--attack-mode` dáme hodnotu `3`. Dále musíme specifikovat typ hashe v `--hash-type`. V našem případě se jedná o MD5, tudíž zadáme `0`. Před masku nesmíme zapomenout zadat soubor obsahující hash. V našem příkladě se jedná o soubor *password.hash*.

```
hashcat --attack-mode 3 --hash-type 0 password.hash
   ?l?l?l?l?l?l?l?l
```

Pokud bychom chtěli masku natahovat automaticky, musíme přidat argument `--increment`. Ten změní chování generování hesel tak, že maska se postupně od jednoho znaku bude natahovat do délky masky, kterou jsme zadali.

```
hashcat --attack-mode 3 --hash-type 0 --increment
   password.hash ?l?l?l?l?l?l?l?l
```

Hashcat navíc umí načíst soubor s maskami. Tento soubor má většinou příponu *.hcmask*. Díky tomu můžeme zadat více masek najednou do jednoho příkazu. Struktura samotného souboru je velmi jednoduchá. Na každém řádku je vždy jedna maska. Příkladový soubor *mask.hcmask* vypadá následovně:

```
?l
?l?l
?l?l?l
?l?l?l?l
?l?l?l?l?l
?l?l?l?l?l?l
?l?l?l?l?l?l?l
?l?l?l?l?l?l?l?l
```

Cesta k tomuto souboru se zadává místo samotné masky, to znamená na konec příkazu. Pokud použijeme v našem příkladě soubor *mask.hcmask*, příkaz bude stejný jako v předchozím příkladě s argumentem `--increment`.

```
hashcat --attack-mode 3 --hash-type 0 password.hash
   mask.hcmask
```

4.3 Slovníkový útok

Požadavky

Program	Verze
Hashcat	6.2.6
John the Ripper	1.9.0

Použité soubory

Název	Popis
password.hash	Soubor obsahuje MD5 hash hesla *password*
wordlist.txt	Soubor obsahuje 10 nejpoužívanějších hesel

4.3.1 Úvod

Slovníkový útok spočívá ve vyzkoušení hesel z nějakého seznamu, například již prolomených hesel. Je totiž pravděpodobné, že heslo, které se snažíme prolomit, bylo použito vícekrát tím samým člověkem nebo je populární mezi lidmi. Jako příklad můžeme uvést naše příkladové heslo *password*. Zaprvé se jedná o běžné anglické slovo a zadruhé se jedná o jedno z nejpoužívanějších hesel na světě. Právě seznamy nejpoužívanějších hesel jsou typickým příkladem slovníku pro slovníkový útok.

4.3.2 Wordlists

V Kali Linux je speciální příkaz a složka, které nám dávají předpřipravené slovníky z různých zdrojů. Mezi ně patří třeba Wifite, Metasploit nebo samotný John the Ripper. Avšak nejvíce používaný je rockyou. Jedná se o slovník hesel ukradených společnosti RockYou. Na rozdíl od slovníků například od společnosti John the Ripper, které jsou tvořeny uměle z více zdrojů, slovník rockyou je vyextrahovaná databáze hesel reálných uživatelů jedné firmy.[14]

Všechny tyto slovníky jsou v Kali Linux umístěny ve složce */usr/share/wordlists/*. Abychom vypsali všechny slovníky, které jsou dostupné, můžeme buď použít příkaz `ls` nebo použijeme vestavěný příkaz `wordlists`[5]. Tento příkaz nepřijímá žádné parametry a můžeme ho spustit z jakékoli složky.

```
wordlists
```

Příkaz nás přesune do složky obsahující slovníky. Zároveň nám vypíše list doporučených slovníků. Vedle samotného názvu je vedle nich i vypsaná cesta k nim. Pokud tento příkaz

[5]Tento příkaz je dostupný pouze v Kali Linux.

spouštíme poprvé, na posledním řádku dostaneme otázku, jestli chceme rozbalit *rockyou.txt*. Ten je totiž zkomprimovaný, aby nezabíral tolik místa na disku. My chceme dekomprimovat rockyou slovník, takže napíšeme *Y* a zmáčkneme *Enter*. Následně se nám vypíše ten samý list s výjimkou toho, že nám přibyl nový soubor *rockyou.txt*.

Můžeme si všimnout, že příkaz nás přesunul do jiné složky, než v které jsme příkaz spouštěli. Pro navrácení stačí zadat *exit*, díky kterému „vyskočíme“ ven z příkazu `wordlists`.

4.3.3 John the Ripper

U programu John the Ripper stačí pouze specifikovat cestu ke slovníku, aby se přepnul do *Wordlist mode*. V našem případě chceme využít soubor *rockyou.txt*, který se nachází ve složce */usr/share/wordlists/*. Cesta ke slovníku se udává do argumentu `--wordlist`. Následně pokud známe typ hashe, předáme ho do `--format`. Nakonec dáme cestu k souboru s hesly, která chceme prolomit.[16, 17, 18]

```
john --format=Raw-MD5
   --wordlist=/usr/share/wordlists/rockyou.txt password.hash
```

John the Ripper nijak nemanipuluje a neoptimalizuje slovník, který mu dáme. To má výhodu v rychlejším nastartování útoku. Nevýhoda nastává při samotném prolamování hesla, kdy při špatném slovníku může dojít k poklesu rychlosti, a tudíž prodloužení času prolamování. Abychom se tomuhle vyhnuli, musíme slovník optimalizovat.

Optimalizace slovníku spočívá hlavně ve dvou úkonech. První je seřazení hesel. Pokud hesla seřadíme, program na prolamování hesel může využít různých optimalizací na vypočítávání dalšího hashe hesla ve slovníku a tím zvýšit rychlost prolamování. Slovník seřadíme pomocí nástroje `sort`. Tomu dáme cestu k souboru, který chceme seřadit. Následně řekneme, že výstup se má zapsat do nového souboru. To uděláme pomocí symbolu `>`.[23]

```
sort wordlist.txt > wordlist_sort.txt
```

Druhý krok je zbavení se duplicitních řádků. Některé slovníky mohou obsahovat ta samá hesla vícekrát. To může být způsobeno například spojením více slovníků do jednoho. Tato duplicitní hesla nám avšak zpomalují rychlost prolamování, protože se dané heslo vypočítává a zkouší znova, což je zbytečné. K deduplikaci využijeme nástroj `uniq`. Tomu stejně jako u příkazu `sort` dáme cestu k souboru a zapíšeme výstup do nového souboru.[24]

```
uniq wordlist_sort.txt > wordlist_sort_uniq.txt
```

Tyto dva příkazy můžeme spojit do jednoho. Nástroj `sort` přijímá totiž argument `--unique`. Tím zařídíme, aby se řadily pouze ty řádky, které ještě nejsou ve výstupu. Navíc můžeme místo symbolu `>` použít argument `--output`, kde zadáme cestu k výstupnímu souboru.

```
sort --unique --output=wordlist_sort_uniq.txt wordlist.txt
```

4.3.4 Hashcat

U Hashcat musíme vždy zvolit typ útoku pomocí argumentu `--attack-mode`. My chceme zvolit *Dictionary attack*, takže mu dáme parametr `0`. Jako vždy musíme také zvolit typ hashe, který prolamujeme a cestu k souboru, který hash obsahuje. Úplně nakonec přijde cesta ke slovníku.[26, 28]

```
hashcat --attack-mode 0 --hash-type 0 password.hash
   /usr/share/wordlists/rockyou.txt
```

Hashcat na rozdíl od John the Ripper si slovník načte a provede na něm různé optimalizace. To sice zpomalí start útoku, ale zase samotné prolamování hesel bude rychlejší.

Zároveň si na rozdíl od John the Ripper musíme dávat pozor na velikost RAM paměti, kterou načtený slovník bude zabírat. Pokud by totiž slovník byl příliš velký, Hashcat by se mohl odmítnout spustit. Aby se Hashcat spustil, budeme mu muset rozdělit slovník na více kusů a postupně mu je dávat. Nástroj na rozdělení souboru na více částí se v Linuxu nazývá `split`. Tomu předáme argument `--line-bytes` s hodnotou, na jak velké kusy chceme slovník rozdělit. V našem příkladě ho chceme rozdělit na soubory, které budou mít maximální velikost 1 GB. Následně napíšeme cestu k souboru, který chceme rozdělit. Nakonec můžeme napsat předponu pro vygenerované soubory.[25]

```
split --line-bytes=1GB /usr/share/wordlists/rockyou.txt
   rockyou_
```

Vygenerované soubory budou pojmenovávány abecedním stylem. To znamená, že nakonec souboru se vždy přidá číslo souboru, ale v abecedním pořadí.

```
rockyou_aa
rockyou_ab
rockyou_ac
...
```

5 Závěr

Tato práce se zabývala prací s hesly a jejich prolamováním. Vytvořil jsem studijní materiál, který teoreticky popisuje principy ukládání a prolamování hesel. Následně jsem popsal postup instalace Kali Linux. Nakonec jsem vytvořil tři praktická cvičení, ve kterých si čtenář může vyzkoušet prolamování hesel v praxi.

Cílem práce bylo vytvořit sadu řešených úloh pro využití distribuce Kali Linux v oblasti prolamování hesel. Pro naplnění cíle a objasnění komplexní problematiky bezpečnosti hesel jsem v první kapitole teoreticky popsal metody manipulace s hesly a jejich ukládání. V druhé kapitole jsem detailně postupně popsal způsoby instalace Kali Linuxu. V poslední kapitole jsem zdokumentoval tři praktická cvičení, která ukazují, jak se prolamují hesla v praxi.

V teoretické části byl popsán algoritmus prohledávání textů hrubou silou a principy ukládání a správy hesel. Následně byla popsána instalace a nastavení Kali Linuxu a modů potřebných pro prolamování hesel. V praktické části práce byly vytvořeny a podrobně popsány minimálně tři řešené úlohy na prolamování hesel za využití Kali Linux. Tato práce byla vytvořena jako doprovodný studijní materiál, jehož cílem je seznámit čtenáře se základními principy ukládání a prolamování hesel.

Jednotlivé laboratorní úlohy mají jednotnou strukturu, která vždy představuje definování problémů, jejich řešení, podrobný popis implementace a využití dílčích nástrojů a ukázkové step-by-step motivační řešení. U dílčích částí bylo poukázáno na možné nejčastější problémy, které se při řešení dané úlohy vyskytují.

Reference

[1] Hashing Algorithms and Security – Computerphile. *YouTube* [online]. Copyright © 2023 Google LLC [cit. 2023–03–18]. Dostupné z: https://www.youtube.com/watch?v=b4b8ktEV4Bg

[2] Password Cracking – Computerphile. *YouTube* [online]. Copyright © 2023 Google LLC [cit. 2023–03–18]. Dostupné z: https://www.youtube.com/watch?v=7U-RbOKanYs

[3] How NOT to Store Passwords! – Computerphile. *YouTube* [online]. Copyright © 2023 Google LLC [cit. 2023–03–18]. Dostupné z: https://www.youtube.com/watch?v=8ZtInClXe1Q

[4] Public Key Cryptography – Computerphile. *YouTube* [online]. Copyright © 2023 Google LLC [cit. 2023–03–18]. Dostupné z: https://www.youtube.com/watch?v=GSIDS_lvRv4

[5] ParthDutt. Understanding Rainbow Table Attack. *GeeksforGeeks: A computer science portal for geeks* [online]. GeeksforGeeks. 10 Feb, 2023 [cit. 2023–03–18]. Dostupné z: https://www.geeksforgeeks.org/understanding-rainbow-table-attack/

[6] Robert Lin. Salt & Pepper: Spice up your hash! *Medium: Where good ideas find you.* [online]. Medium. Oct 27, 2016 [cit. 2023–03–18]. DostupnÃľz:https://medium.com/@berto168/salt-pepper-spice-up-your-hash-b48328caa2af

[7] Alex Biryukov, Daniel Dinu, a Dmitry Khovratovich. Argon2: the memory–hard function for password hashing and other applications. *GitHub: Let's build from here* [online]. Copyright © 2023 GitHub, Inc. Mar 25, 2017 [cit. 27.03.2023]. Dostupné z: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf

[8] *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux [cit. 2023–03–18]. Dostupné z: https://www.kali.org/

[9] Get kali | Kali Linux. *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux [cit. 2023–03–18]. Dostupné z: https://www.kali.org/get-kali/

[10] g0tmi1k. Making a Kali Bootable USB Drive on Windows. *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux, 2022–Dec–30 [cit. 2023–03–18]. Dostupné z: https://www.kali.org/docs/usb/live-usb-install-with-windows/

[11] gamb1t. Making a Kali Bootable USB Drive. *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux, 2023–Mar–14 [cit. 2023–03–18]. Dostupné z: https://www.kali.org/docs/installation/create-bootable-media/

[12] gamb1t. Installing Kali Linux. *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux, 2023–Mar–13 [cit. 2023–03–18]. Dostupné z: https://www.kali.org/docs/installation/hard-disk-install/

[13] Re4son. Which Image Should I Download? *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux, 2022–Nov–22 [cit. 2023–03–18]. Dostupné z: https://www.kali.org/docs/introduction/what-image-to-download/

[14] Wordlists. *Kali Linux: Penetration Testing and Ethical Hacking Linux Distribution* [online]. Kali Linux, 2023–Mar–08 [cit. 2023–03–18]. Dostupné z: https://www.kali.org/tools/wordlists/

[15] *Rufus: Create bootable USB drives the easy way* [online]. Rufus [cit. 2023–03–18]. Dostupné z: https://rufus.ie/en/

[16] John the Ripper's cracking modes. *Openwall: bringing security into open computing environments* [online]. Openwall, 2013/05/29 17:57:56 [cit. 2023–03–18]. Dostupné z: https://www.openwall.com/john/doc/MODES.shtml

[17] John the Ripper's command line syntax. *Openwall: bringing security into open computing environments* [online]. Openwall, 2016/01/21 05:10:00 [cit. 2023–03–18]. Dostupné z: https://www.openwall.com/john/doc/OPTIONS.shtml

[18] John the Ripper usage examples. *Openwall: bringing security into open computing environments* [online]. Openwall, 2019/05/19 15:10:04 [cit. 2023–03–18]. Dostupné z: https://www.openwall.com/john/doc/EXAMPLES.shtml

[19] John the Ripper FAQ. *Openwall: bringing security into open computing environments* [online]. Openwall, 2019/05/19 15:10:04 [cit. 2023–03–18]. Dostupné z: https://www.openwall.com/john/doc/FAQ.shtml

[20] Wordlist rules syntax. *Openwall: bringing security into open computing environments* [online]. Openwall, 2017/05/14 12:16:07 [cit. 2023–03–18]. Dostupné z: https://www.openwall.com/john/doc/RULES.shtml

[21] john/MASK at bleeding–jumbo · openwall/john. *GitHub: Let's build from here* [online]. Github, Feb 18, 2022 [cit. 2023–03–18]. Dostupné z: https://github.com/openwall/john/blob/bleeding-jumbo/doc/MASK

[22] GNU Findutils 4.9.0. *The GNU Operating System and the Free Software Movement* [online]. GNU Operating System [cit. 2023–03–18]. Dostupné z: https://www.gnu.org/software/findutils/manual/html_mono/find.html

[23] sort: Sort text files. *The GNU Operating System and the Free Software Movement* [online]. GNU Operating System [cit. 2023–03–18]. Dostupné z: https://www.gnu.org/software/coreutils/manual/html_node/sort-invocation.html

[24] uniq: Uniquify files. *The GNU Operating System and the Free Software Movement* [online]. GNU Operating System [cit. 2023–03–18]. Dostupné z: https://www.gnu.org/software/coreutils/manual/html_node/uniq-invocation.html

[25] split: Split a file into pieces. *The GNU Operating System and the Free Software Movement* [online]. GNU Operating System [cit. 2023–03–18]. Dostupné z: https://www.gnu.org/software/coreutils/manual/html_node/split-invocation.html

[26] hashcat. *hashcat: advanced password recovery* [online]. hashcat [cit. 2023–03–18]. Dostupné z: https://hashcat.net/wiki/doku.php?id=hashcat

[27] Mask Attack. *hashcat: advanced password recovery* [online]. hashcat [cit. 2023–03–18]. Dostupné z: https://hashcat.net/wiki/doku.php?id=mask_attack

[28] Dictionary Attack. *hashcat: advanced password recovery* [online]. hashcat [cit. 2023–03–18]. Dostupné z: https://hashcat.net/wiki/doku.php?id=dictionary_attack

[29] WRÓBLEWSKI, Piotr. Algoritmy. Brno: Computer Press, 2015. ISBN 978–80–251–4126–7.