Ukládání a prolamování hesel

Vojtěch Žák

2022–2023

Obsah

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 neopráněného 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 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.

2.1 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 Šifrování

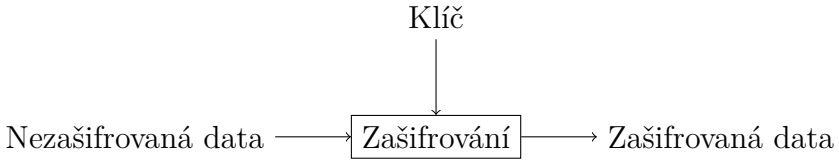
Šifrování slouží k převedení nezabezpečených dat na data šifrovaná. Pro dešifrování dat (převedení dat na data čitelná) musíme znát nějakou znalost. Většinou se pro šifrování a dešifrování používají klíče (hesla).[3]

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, ...).

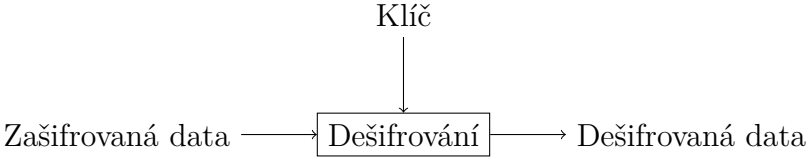
Existují dva hlavní typy moderních šifer: symetrické a asymetrické.

2.2.1 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:

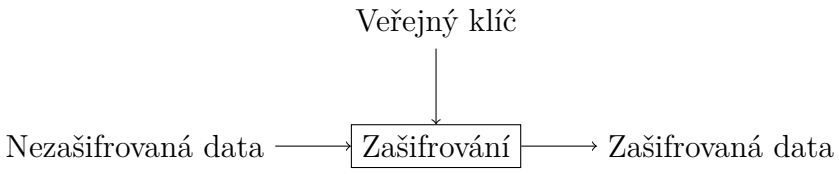
- Nejdříve vezmeme zašifrované heslo z databáze. Následně ho rozšifrujeme a dešifrované heslo porovnáme se zadaným heslem.
- Zadané heslo zašifrujeme a potom porovnáme se zašifrovaným heslem z databáze.

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.

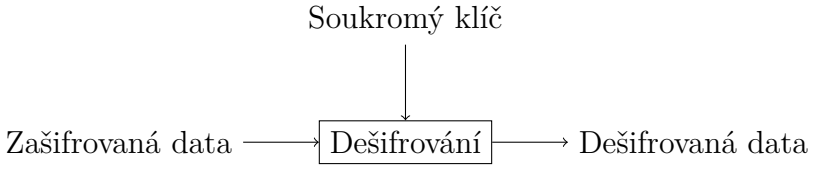
Problém nastává v tom, že abychom mohli samotná hesla šifrovat, musíme někde mít také uložený klíč. Pokud útočník kromě hesel ukradne i tento klíč, tak je schopný všechny hesla dešifrovat.

2.2.2 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



Obrázek 4: Asymetrické dešifrování

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 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.

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

- 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.

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.

2.3 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.

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

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

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

2.3.1 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.

2.3.2 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.

2.3.3 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.

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.[13]

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.

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.

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.[11, 8]

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.

4.1.1 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.

4.1.2 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.[11, 10]

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.

4.1.3 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.[12, 8]

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.

4.1.4 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.[9]

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.

4.2 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.

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] 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

[9] 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

[10] 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

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

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

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