OKsystem

Entropie, hesla a jak uložit tajemství…

Ivo Rosol
ředitel vývojové divize

Agenda prezentace

- informace, míra informace, entropie
- optimální kódování
- entropie hesla a fráze
- útoky na uložení hesla – slovníkový útok, předpočítané vyhledávací tabulky, duhové tabulky, výpočetní síla paralelních GPU
- protiopatření – salt, výpočetně a paměťově náročné „hashovací“ funkce, složitější autentizační schémata
- doporučení pro hesla

Hesla v kryptografii

V mnoha aplikacích využívajících kryptografii je bezpečnost závislá na tajných textových hodnotách (heslech, přístupových frázích). Hesla se používají k více účelům, například:

- K autentizaci subjektu
- Ke kontrole autenticity a integrity zprávy
- Pro odvození šifrovacích klíčů

Hesla

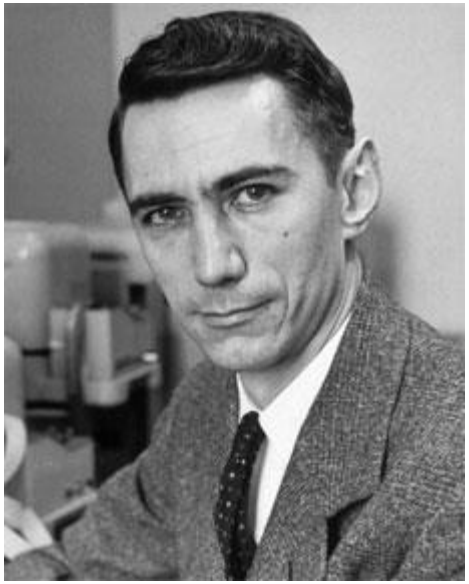
- Hesla jsou často vybírána z omezené množiny, mají nízkou entropii.
- Heslo samotné nejde přímo použít jako kryptografický klíč
- Pokud je heslo („sdílené tajemství“) uloženo v informačním systému/aplikaci, využívá se princip uložení vhodné transformace hesla, místo uložení přímé hodnoty.

Nelze mluvit o heslech a nezmínit informaci a entropii

Teorie informace

- Entropie a míra informace jsou stěžejní pojmy teorie informace, za jejíž zakladatele je považován Claude Shannon. Práce „A Mathematical Theory of Communication“, publikovaná roku 1948 je často považována za základ rigorózního matematického základu informace.
- Teorie informace se zabývá kvantitativní stránkou informace, tedy měřením, kódováním a zpracováním informace, nikoli jejím významem, tedy kvalitativní stránkou..

Claude Elwood Shannon



„We know the past
but cannot control it.

We control the future
but cannot know it“

O tom, že kvalita informace je složitý pojem:

zkusme zadat jméno Shannon do Googlu a vyhledat obrázek s fotografií

Kvantitativní definice informace

Shanonova definice

- Informace je míra množství neurčitosti nebo nejistoty o nějakém náhodném ději odstraněná realizací tohoto děje. Informace rozšiřuje okruh znalostí příjemce.

Ralph Hartley

- Hartleyova míra informace
- Hartleyův oscilátor
- Hartleyova transformace

Ralph Hartley, předchůdce a později spolupracovník Shannona. Zabýval se zejména radiový přenosem informace.

Hartleyova míra infomace

Předpoklad – zdroj poskytuje zprávy se stejnou pravděpodobností.

Požadované vlastnosti míry informace:

- Čím více různých zpráv poskytuje zdroj, tím je větší množství informace. Množství informace je tedy (nelineární) monotónně rostoucí funkcí počtu zpráv.
- Pokud zdroj poskytuje jedinou zprávu, jedná se o jistý jev, množství informace je 0
- Pokud dva nezávislé zdroje poskytují současně zprávy, pak množství informace z obou zdrojů je součtem informací jednotlivých zdrojů

Hartleyova míra infomace

Předpoklad – zdroj poskytuje zprávy se stejnou pravděpodobností

Požadované vlastnosti míry informace

- Čím vice různých zpráv poskytuje zdroj, tím je větší množství informace. Množství informace je tedy (nelineární) monotónně rostoucí funkcí počtu zpráv.
- Pokud zdroj poskytuje jedinou zprávu, jedná se o jistý jev, množství informace je 0
- Pokud dva nezávislé zdroje poskytují současně zprávy, pak množství informace z obou zdrojů je součtem informací jednotlivých zdrojů

Hartleyova míra infomace

s – počet symbolů abecedy
n – délka zprávy
I – množství informace

N – počet různých zpráv s délkou n, sestavených z abecedy o s symbolech:

$$N = s^n$$

Množství informace I je rostoucí funkcí počtu těchto zpráv (vlastnost 1):

$$I = f(N) = f(s^n)$$

Hartleyova míra infomace

Pokud jsou 2 nezávislé zprávy z 2 nezávislých zdrojů s délkami n1 a n2 (nebo je delší zpráva z jednoho zdroje s délkou n rozdělena na 2 části s délkami n1 a n2), pak z vlastnosti 3 (aditivita) plyne:

$I = I1 + I2$ – požadavek aditivity míry informace

$f(s^{n1+n2}) = f(s^{n1}) + f(s^{n2})$

$f(N1*N2) = f(N1) + f(N2)$

Tyto vlastnosti má funkce f – logaritmus

$I = f(N) = k*\log(N) = k*n*\log(s)$

kde k je škálovací konstanta

Důležité pozorování:

- Množství různých zpráv N zdroje musí růst exponencionálně, aby množství informace rostlo lineárně (big data).
- Toho se dá mnohem snáze dosáhnout délkou zprávy, než velikostí abecedy zprávy (hesla!).

Shannonova míra informace

Předpoklady

- Informace je ve zprávě obsažena, pokud odstraňuje u příjemce neurčitost (obsahuje něco nového pro příjemce).
- Příjemce musí znát předem množinu všech možných zpráv. Zdroj informace má volnost výběru z této množiny zpráv. U příjemce tedy existuje neurčitost o výběru a vyslání zprávy zdrojem informace.
- Různé zprávy vysílá zdroj s určitou (ne stejnou) pravděpodobností – na rozdíl od Hartleyho míry informace

Shannonova míra informace

Množství informace ve zprávě

Množství informace obsažené ve zprávě souvisí s pravděpodobností jejího výskytu.

Čím menší je pravděpodobnost výskytu zprávy, tím je méně pravděpodobné, že ji příjemce uhádne před přijetím a tím větší neurčitost u příjemce odstraní přijetí zprávy, tedy tím větší informaci zpráva přinesla příjemci.

Množství informace ve zprávě je tudíž nepřímo úměrné pravděpodobnosti, s jakou lze uhodnout zprávu před jejím přijetím:

$$I(X) = f(1/P(X))$$

Příklad:

Zpráva, „**v ruletě padlo 21**“ (P= 1/37) přinese více informace, než zpráva „**v ruletě padla červená**“ (P=18/37)

Shannonova míra informace

Množství informace ve zprávě

S použitím funkce logaritmu (viz Hartleyova míra informace)

$$I(X) = \log(1/P(X)) = -\log(P(X))$$

Shanonova míra informace je horní mez informace, která je dosažitelná při optimálním kódování.

Střední míra informace zdroje (entropie zdroje)

Střední množství informace zdroje, který generuje zprávy x1, x2,…xn s pravděpodobnostmi p1, p2,..,pn je dáno střední hodnotou informací všech zpráv zdroje:

$$H(X) = -\sum_{i=1}^{n} P(xi) * \log P(xi)$$

Entropie

Entropie se měří v bitech (pokud je použit $\log_2$)

Entropie je opačná veličina k informaci – přírůstek informace odpovídá úbytku entropie.

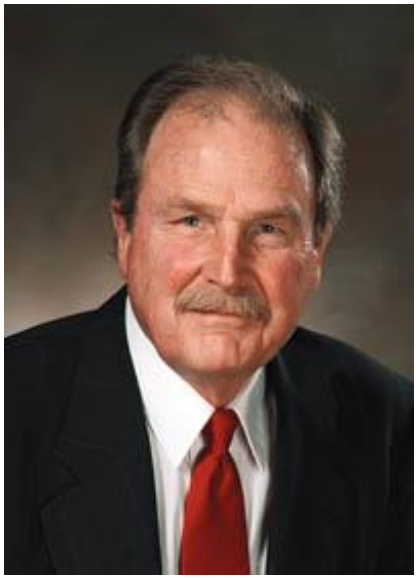
Informaci obsahuje pouze ta část zprávy, která je nepredikovatelná, neboť to, co je predikovatelné, nepřináší žádnou informaci (je to redundance zprávy). Jinak řečeno, přijetím zprávy (informace) se sníží neurčitost znalosti (entropie) příjemce

Entropie je současně průměrný počet bitů (vzorec pro výpočet entropie je výpočet střední hodnoty), nezbytných k zakódování zprávy při optimálním kódování (optimální kódování používá co nejméně bitů k zakódování zpráv).

Poznámka

Pojmu entropie bylo původně použito v termodynamice k vyjádření rozlišnosti nebo uspořádání stavů mikrofyzikálních systémů (např. atomů). Přírůstek entropie v termodynamice znamená posun z více uspořádaného do méně uspořádaného stavu fyzikálního systému. Tento jev je doprovázen energetickými změnami.

David Huffman



- Huffmanovo kódování

Prefixové kódování s variabilní délkou kódu pro bezeztrátovou kompresi

Huffmanův algoritmus

Seřadí se sloupec pravděpodobnosti zpráv (jevů) od největší k nejmenší.

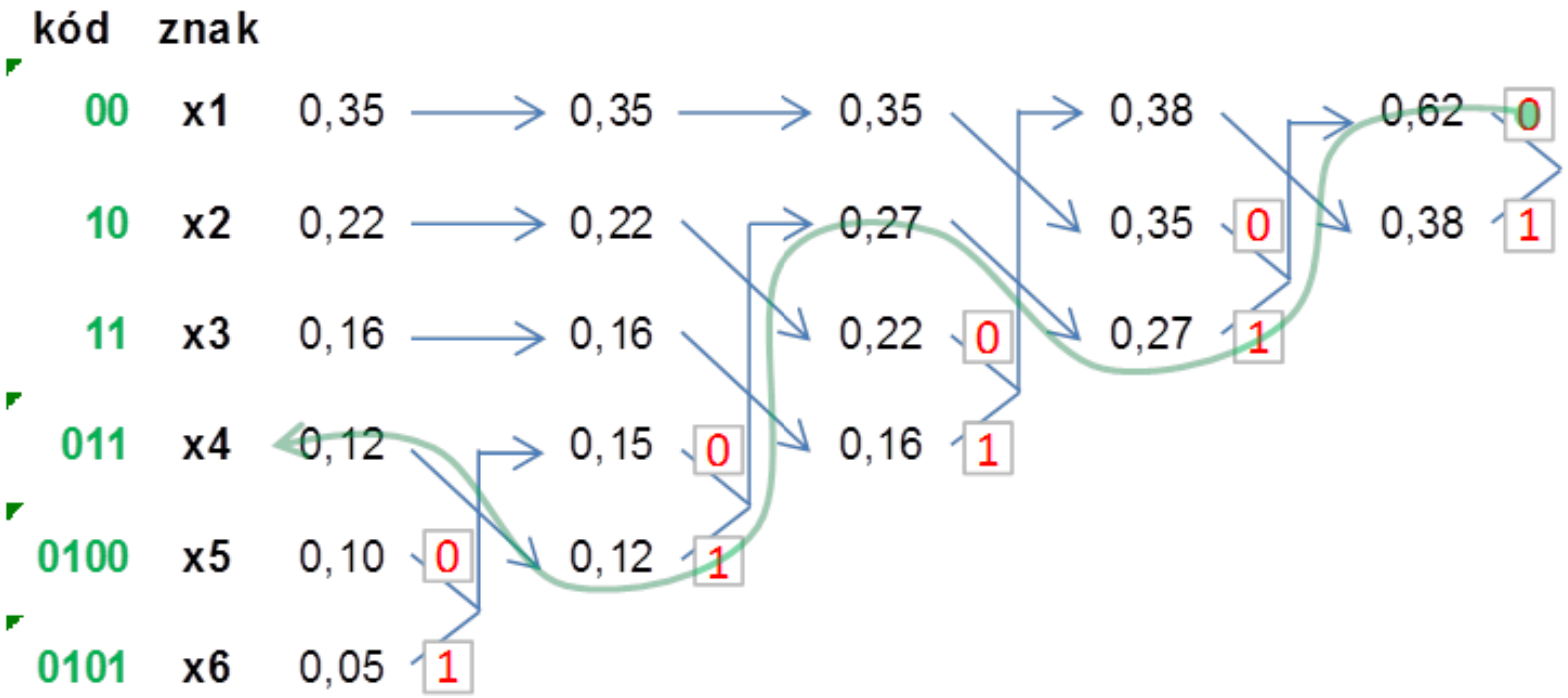
Dvěma posledním zprávám (s nejmenší pravděpodobností výskytu) se přiřadí bit kódování s hodnotou 0 a 1.

Sečtou se pravděpodobnosti dvou posledních zpráv (součet odpovídá pravděpodobnosti jevu, že nastane jedna, nebo druhá zpráva) a výsledný součet se zatřídí do nového sloupce.

Postupem zdola nahoru se postupně slučují dva jevy s nejnižší pravděpodobností (přičemž to mohou být již dříve sloučené jevy), až v sloupci zbydou poslední dva jevy, kterým se přiřadí kódovací bit s největší váhou s hodnotami 0 a 1.

Při pohledu od konce je zřejmé, že se nevyšší bit kódování přiřadí s hodnotou 0 elementárnímu jevu s nejvyšší pravděpodobností a s hodnotou 1 složenému jevu (tedy jevu, že nastane kterýkoli ze zbývajících jevů s nižší pravděpodobností) s pravděpodobností danou součtem pravděpodobností elementárních a složených jevů z předchozího postupu

Příklad - kódování



Příklad – průměrná délka kódu

Znak	P(xi)	-P(xi)*log(P(xi;2))	
x1	0,35	-0,53010	
x2	0,22	-0,48057	
x3	0,16	-0,42302	
x4	0,12	-0,36707	
x5	0,10	-0,33219	
x6	0,05	-0,21610	
ΣP(xi)*log(P(xi;2))		2,34905	bitů/znak

Entropie hesla

Entropie hesla je míra nepredikovatelnosti hesla.

- Pro teoreticky přesný výpočet entropie by bylo nutné znát rozložení pravděpodobnosti jednotlivých znaků v hesle a kódovat znaky podle jejich pravděpodobnosti různým počtem bitů.
- Je běžné přijmout nepřesný předpoklad, že jsou všechny znaky stejně pravděpodobné (přestože v praxi jsou písmena a číslice v hesle použita častěji než speciální znaky, bude skutečná entropie nižší než spočtená).

Příklady entropie hesla

Příklad výpočtu entropie číselného hesla o délce N znaků

- Abeceda hesla má 10 znaků (0,1,…9), všechny znaky se vyskytují se stejnou pravděpodobností, p_i=0,1, heslo má N znaků. Pravděpodobnost každého hesla o délce N znaků je $1/10^N$, tedy 10^{-N}
- Entropie pro heslo o délce N znaků, pouze číslice H = N*log(10)/log(2) = N*3,32.
- Číselné heslo má entropii 3,32 bitů/znak, tedy běžný 4 ciferný PIN má entropii 13,3 bitů

Entropie pro různé množiny znaků

- Číslice – 10 znaků, entropie 3,32 bitů/znak
- Malá písmena bez diakritiky – 26 znaků, entropie 4,7 bitů/znak
- Velká písmena bez diakritiky – 26 znaků, entropie 4,7 bitů/znak
- Malá i velká písmena bez diakritiky – 52 znaků, entropie 5,7 bitů/znak
- Malá i velká písmena bez diakritiky a číslice – 62 znaků, entropie 5,95 bitů/znak

Doporučení pro hesla

- Je zřejmé, že pokud jsou všechny znaky hesla stejně pravděpodobné, entropie roste lineárně s délkou hesla, ale pouze logaritmicky s počtem znaků v množině.
- Pro mnohé může být překvapující, že podstatně větší dopad na entropii hesla má jeho délka, než počet znaků v použité množině, neboť je „zažito“, že heslo má obsahovat kombinaci malých a velkých písmen, číslic a tisknutelných speciálních znaků.
- Zejména speciální znaky činí potíž při zadávání na různých klávesnicích (i plnohodnotných), o to víc pak na mobilních zařízeních. Mnohem pohodlnější způsob dosažení potřebné entropie spočívá v zadání dostatečně dlouhého hesla složeného pouze z (malých) písmen.
- Minimální počet znaků pro takové heslo je 15 znaků (cca 70 bitů entropie)

Passphrase

- Často se má za to, že pro uživatele je jednodušší zadat relativně snadno zapamatovatelnou frázi, než „nesmyslné“ a obtížně zapamatovatelnou změť znaků hesla.
- V případě použití „věty“ skládající se ze slov, je třeba entropii počítat jinak – slovníkem zde nejsou náhodně vybrané znaky (které by dávaly obrovskou entropii), ale slova daného jazyka.

Passphrase

Kolik slov má angličtina?

Druhé vydání 20 svazkové Oxford Eglish Dictionary obsahuje 171 000 slov, z toho 47 000 zastaralých.

http://www.oxforddictionaries.com/words/how-many-words-are-there-in-the-english-language

Passphrase - angličtina

- V angličtině, s cca 171 000 slovy, za zjednodušujícího předpokladu, že všechna slova mají stejnou pravděpodobnost, by mělo každé slovo (nezávisle na své délce !) entropii 17,4 bitů/slovo.
- Takže věta „quick errors fox pleasure“ by měla entropii 4 * 17,4 = 69,5 bitů, nikoli 25 * 4,75 = 118,8 bitů. (délka včetně mezer je 25 znaků, malá písmena a mezera má 27 znakovou abecedu, s 4,75 bitů/znak).
- Ve skutečnosti bude entropie passphrase nižší, neboť všechna slova nejsou stejně pravděpodobná – pouze několik set slov se používá relativně běžně, většina ostatních spíše výjimečně.

Passphrase – reálná čeština

Kolik slov má čeština?
Slovník spisovné češtiny obsahuje cca 48 000 hesel, akademický slovník 200 000 slov.

- Každé slovo v češtině může mít určitý počet tvarů, v průměru 13 tvarů (slovo dobrý má údajně 42 tvarů) http://nase-rec.ujc.cas.cz/archiv.php?art=3932
- Lidé s VŠ vzděláním znají cca 60 000 slov, ostatní cca 20 000 slov.
- V češtině 1000 nejběžnějších slov pokrývá 65% textu.

1 000 slov	10 bitů/znak	7 slov/70 bitů entropie
20 000 slov	14,3 bitů/znak	4,9 slov/70 bitů entropie
48 000 slov	15,5 bitů/znak	4,5 slova/70 bitů entropie
60 000 slov	15,9 bitů/znak	4,4 slova/70 bitů entropie

Pozorování:
není nutné používat velký slovník, podstatný je počet slov

Password hashing

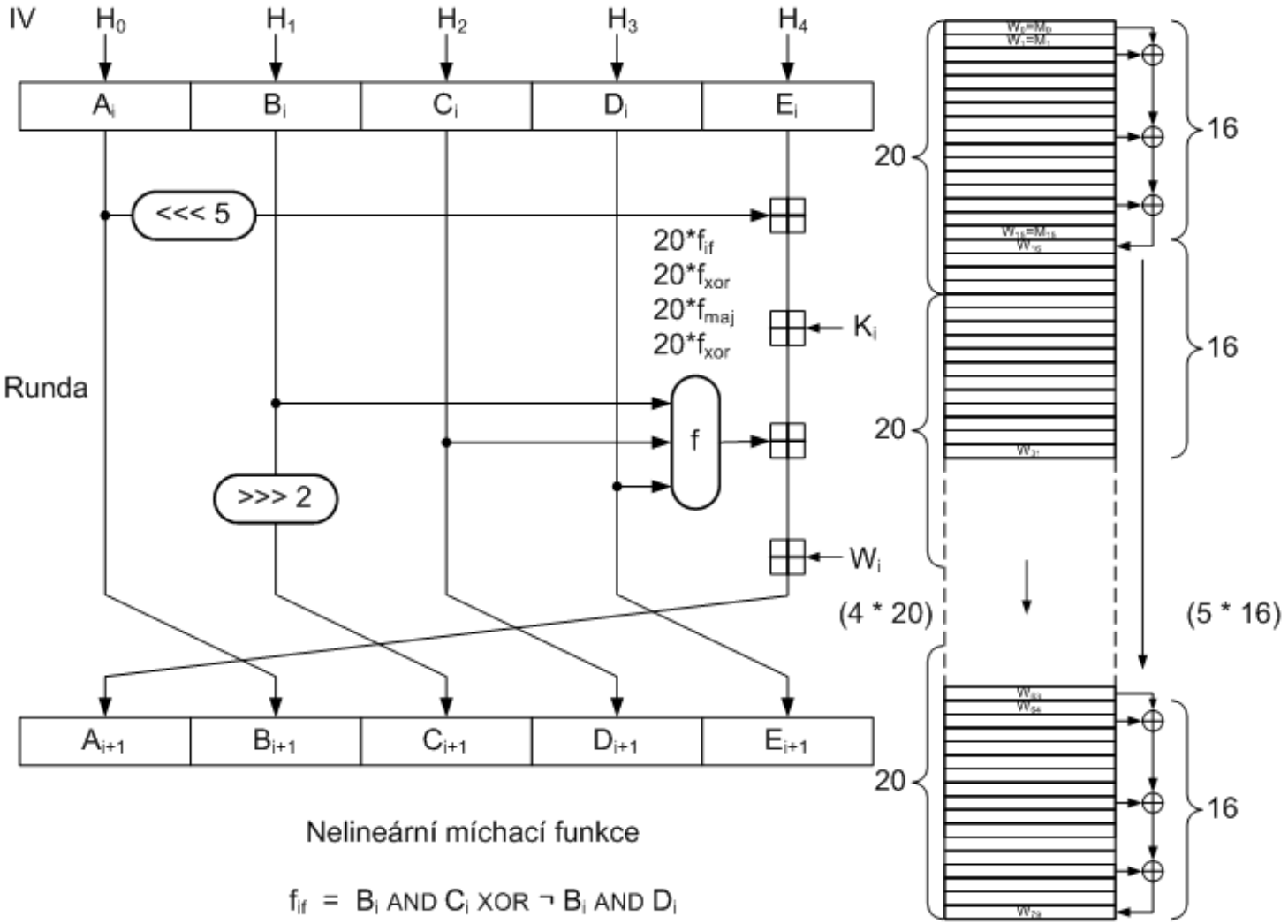
- Hashe hesel se používají v mnoha systémech, zejména ve spojitosti s autentizací heslem nebo symetrickým klíčem – autentizace k desktopovým i mobilním systémům, web aplikacím, šifrovacím systémům disků apod.
- Příklad konstrukce hashovací funkce SHA1 (princip SHA256 je obdobný, pro výklad složitější). Před hashováním se zpráva opatří zarovnáním (100000….00000Délka)

Kompresní funkce 80 slov na 5 slov

80 rund míchání

Expanze 16 slov na 80 slov

$W_i = W_{i-3}$ XOR W_{i-8} XOR W_{i-14} XOR W_{i-16} <<< 1



Nelineární míchací funkce

$f_{if} = B_i$ AND C_i XOR ¬ B_i AND D_i

$f_{maj} = B_i$ AND C_i XOR B_i AND D_i XOR B_i AND C_i

$f_{xor} = B_i$ XOR C_i XOR D_i

OKsystem

Slabiny hashovací funkce

- Standardní hashovací funkce jsou vhodné a bezpečné pro hashování relativně dlouhých zpráv.
- V případě velmi krátkých zpráv (typickým případem jsou hesla s nízkou entropií) nejsou samotné hashovací funkce dostatečně bezpečné proti útoku na nalezení vzoru hrubou silou.
- Hrubá síla může spočívat v předpočítání tabulek v podmnožině možných hesel (slovníkový útok), lookup table s omezenou délkou a různě omezenými znakovými sadami, nebo výpočtem rainbow tables, které redukují velikost tabulek pomocí uchování pouze počátečních a koncových hodnot zřetězení vzorů a hashů, nebo finálně hrubou výpočetní silou masivně multiparalelního počítače s mnoha GPU, případně ASIC čipy.

Útoky na hash hesla

Lookup table

- předpočítaná tabulka hodnot hashovací funkce. Tabulka mapující hesla dlouhá 7 znaků, z množiny 95 znaků má 95 na 7 různých hodnot, potřebuje cca 2000TB diskové kapacity.
- Efektivnější na spotřebu paměti jsou předpočítané výstupy jednosměrné funkce pro předem určené velké množiny vstupů - často používaná slova, permutace jejich znaků a jejich kombinace s čísly a speciálními znaky. Tato hesla pocházejí ze slovníků, slovníků slabých hesel, knih, wikipedie, honneypots apod.

Slovníkový útok

Příklad:

Slovník různých funkcí pro hashování hesla obsahující všechna slova z Wikipedie + jejich modifikace online:

https://crackstation.net/

Výpočet různých typů hashů

http://www.hashemall.com/

Vyzkoušet libovolné slovo z Wikipedie, na konec přidat 123 a předvést, že se za 1 – 3 sec. nalezne vzor.

Rainbow table

Pro redukci paměti k uložení celé tabulky se používá redukovaná tabulka, obsahující hesla a zřetězené hodnoty jejich hashů, tzv Rainbow table.

RT ke stažení pro hesla 7 – 10 znaků:

http://project-rainbowcrack.com/table.htm

Princip Rainbow Table

- Pro konstrukci RT potřebujeme hashovací fukci H a redukční funkce R.
- Začneme s určitou množinou hesel o dané délce (např. 6 znaků) a vypočítáme řetězec P->H1->R1->H2->R2……Hn->Rn.
- Na každou hodnotu Hi aplikujeme redukční funkci s výstupem Ri, která mapuje hash do množiny hesel zvolené délky
- Uloží se pouze počáteční heslo P a koncová redukce Rn.
- Pokud chceme invertovat nějaký hash, spočítáme pro něj zřetězení postupnou aplikací H a R a pokaždé kontrolujeme, zdali se neobjevila některá z uložených koncových hodnot Rn. Pokud ano, tak hledané heslo je předchozí hodnota hashe ve zřetězení, které spočítáme na základě uloženého počátečního hesla.
- Prodlužování zřetězení vede k snižování paměťových nároků na tabulku, ale současně zvyšuje potřebný výpočetní výkon.

Útok výpočtem hrubou silou

K útokům lze využít vysoký výkon grafických karet (1000 x výkonnější než CPU). Nalezení hesla z hashe hrubou silou, je efektivní pro hesla o délce 8 znaků.

Útok jako služba:
http://www.onlinehashcrack.com/hash-cracking.php
do maximálně 10 znaků (sec. až 4 dny), do 8 znaků zdarma

Salted hash

- Účinnou obranou proti Lookup tables a Rainbow tables je salt („osolení“ heše), kdy se místo SHA-256(heslo) počítá SHA-256(salt + heslo). Salt je náhodná hodnota, která se uloží v otevřené podobě společně s hodnotami SHA-256(salt + heslo), aby bylo možné ověřit heslo. Protože je salt náhodné, nelze předpočítat Rainbow tables pro všechny jeho hodnoty zřetězené s hesly ze slovníku, a to ani pro relativně krátké a slabé heslo.
- Salt ovšem nechrání proti útoku výpočtem, tedy ani před slovníkovým útokem (chrání jen proti předpočítaným tabulkám), protože hodnota salt je útočníkovi většinou známá a pokud je použito heslo s nízkou entropií, lze vždy zaútočit výpočetním výkonem.
- Metoda stará cca 30 let, použitá primárně k odlišení stejných hesel různých uživatelů, respektive stejných uživatelů na různých systémech.
- Salt je náhodná hodnota, která je zřetězená s heslem (je lhostejno, zdali je heslo před solí, nebo za ní). Lze i kombinovat několik hodnot salt v prefixu, uprostřed a v postfixu

Salted hash

- Salt je možné považovat za index do mnohem větší tabulky klíčů odvozených z hesla, a jeho hodnotu není nutné utajovat. Útočník sice může zkonstruovat slovník, pro který jsou odvozeny klíče, nicméně ale musí zkombinovat slovník se salt, aby získal všechny možné hodnoty klíčů. Tím se eliminuje použití předpočítaných lookup tables a rainbow tables.
- Doporučená minimální délka salt pro hashování hesla je 16 byte. Předpokládá se, že hodnota salt je známá útočníkovi.
- Z tohoto důvodu jsou v současné době Rainbow tables již málo účinné, místo toho se k útokům na hashe používá vysoký výkon grafických karet.
- Existují názory, že bezpečnost lze dále zlepšit, pokud salt nebude jednoduše „po ruce“, například bude uložena v souboru (ne v databázi), na jiném serveru, zašifrovaná nějakým klíčem, zřetězená neznámým způsobem s heslem atd.

Hashovací funkce s klíčem

Hashovací funkce s klíčem jsou speciálním případem MAC (Message Authentication Code) funkcí, které ověřují autenticitu a integritu zprávy.

Způsoby návrhu hashovací funkce s klíčem:

- Špatně
  MAC = H(K || m)
  náchylné na „length extension attack“, útočník bez znalosti klíče je schopen přidat na konec zprávy cokoli a dopočítat MAC hashováním H(MAC || cokoli)
- Trochu lépe
  MAC = H(m || K)
  kolize hashovací funkce vede na kolizi MAC
- Přijatelně
  MAC = H(K || m || K)
- Správně
  H(K || H(K || message))

HMAC

HMAC(K,m) = H(K’ ⊕ opad || H((K’ ⊕ ipad) || m))

- opad je konstanta o délce vstupního bloku hashovací funkce (512) = 0x5c5c5c….5c5c
- ipad je konstanta o délce vstupního bloku hashovací funkce (512) = 0x363636….3636
- Ipad a opad modifikují „vnitřní a vnější“ klíč, aby měli co nejméně společných bitů
- K’ je klíč odvozený z K, doplněním 0 zprava na délku bloku, nebo H(K), pokud je K delší než blok
- HMAC je (mimo jiné) používána v PBKDF2

Výpočetně náročné „hashovací“ funkce

Myšlenka spočívá ve skutečnosti, že výpočetní náročnost (daná například počtem iterací) není podstatná pro legitimního ověřovatele, který provede výpočet pouze pro zaslaný autentizační materiál, ale je neschůdná pro útočníka, který musí výpočet provést pro všechny uvažované hodnoty hesla.

Využití těchto funkcí je dvojí:

- „hashování“ hesel (key stretching) (password based message authentication)
- Odvození symetrického klíče z hesla s nízkou entropií (password-based encryption)

Protože výpočetní náročnost lze řešit převedením na paměťovou náročnost (off-line předpočítání), nebo časovou náročnost masivní paralelizací (mnoho GPU), je ideální, pokud funkce umožňuje parametricky nastavit časovou náročnost, paměťovou náročnost (RAM nebo předpočítáním) a omezit paralelizaci počtem vláken.

PBKDF2

Password Based Key Derivation Function 2 – PKCS#5 v2, RFC 2898

PBKDF2 je výpočetně náročná jednosměrná (pseudonáhodná) funkce, podobná hashovací funkci, jejíž parametrem je typ pseudonáhodné funkce (HMAC, hash nebo šifra), vstupem je heslo p (libovolný řetězec), salt s (alespoň 64 bitů), počet iterací c a požadovaná délka výstupního klíče dkLen a výstupem odvozený kryptografický klíč DK.

DK = PBKDF2 (P, S, c, dkLen)

PBKDF2

- L = dkLen \ hLen +1 (kolikrát se vejde délka výstupu pseudonáhodné funkce – např. HMAC do délky výstupního klíče)

Začíná se s hodnotou i = 1

U_1 = PRF(P,S || INT(i)), kde INT32(i) je 4 bajtově zakódovaný index iterace i = 1 .. c

U_2 = PRF(P || U_1)

U_3 = PRF(P || U_2)

…

U_C = PRF(P || U_{c-1})

Dále se všechny U „xorují“ a tím se získá i-tý blok Ti výstupního klíče (na začátku 1. blok):

Ti = F(P,S,C,i) = $U_1 \oplus U_2 \oplus U_3 \oplus \ldots \oplus U_C$

Spočítají se vektory T_1, T_2, …T_L a zřetězí do výstupního klíče:

DK = T_1 || T_2 || … || T_L

PBKDF2

Poznámky:

- Na rozdíl od hashovací funkce probíhá výpočet ve velkém množství iterací (v řádu 10 000 - 100 000), aby se zabránilo útoku na výpočet vzoru hrubou silou. Salt a počet iterací nemusí být utajeny.
- Volbou délky klíče DKLen můžeme vyprodukovat více kratších klíčů (výstup se rozseká na délky klíčů)
- Různé hodnoty salt umožní generovat libovolné množství klíčů z jednoho hesla.
- Počet iterací je parametrem PBKDF2, takže výpočetní náročnost lze škálovat. Slabinou PBKDF2je to, že její implementace vyžaduje relativně málo paměti a je proto vhodná pro výpočet na GPU. Tuto bezpečnostní slabinu odstraňuje bcrypt, případně scrypt, který má libovolně nastavitelnou paměťovou náročnost.
- Pozor – PBKDF2 nelze považovat (a použít jako náhradu) za hashovací funkci, vzhlede ke skutečnosti, jak zachází s HMAC(SHA), kdy pro HMAC klíče delší než velikost bloku hashovací funkce se použije hash klíče, místo samotné hodnoty klíče. Z toho plyne rovnost:
- PBKDF2_HMAC_SHA(heslo) == PBKDF2_HMAC_SHA(SHA(heslo))
- Pro každé heslo delší než blok SHA existuje výše uvedená kolize.

Bcrypt

Funkce je založena na šifrovacím algoritmu Blowfish, je implicitní funkcí pro „hashování“ hesla (shadow file) v řadě distribucí Linuxu.

DK = bcrypt(cost, salt, password)

Bcrypt využívá modifikovanou, výpočetně náročnou přípravu klíče, kde v mnoha cyklech (2^{cost}) střídá sůl a klíč k nastavení subklíčů

http://bcrypt.sourceforge.net/

Argon2

- V roce 2013 byla vyhlášena otevřená soutěž Password Hashing Competition, viz https://password-hashing.net/ . Soutěže se zúčastnilo 24 kandidátů, na konci 2014 byl výběr zúžen na 9 finalistů a v polovině roku 2015 byl vyhlášen vítězný algoritmus **Argon 2**. Algoritmus lze parametrizovat pomocí:
- Časové náročnosti (Time cost)
- Paměťové náročnosti (memory cost)
- Počtu vláken, který omezí možnost paralelizace

Hashovat na klientu, nebo na serveru?

Existují argumenty pro a proti oběma možnostem. Společným předpokladem je autentizace v rámci SSL.

- Nemusí být bezpečné posílat heslo v otevřeném tvaru prostřednictvím SSL na server. Jednak kvůli útokům na SSL, jednak kvůli MitM útokům na serveru.
- Nemusí být bezpečné „hashovat“ na klientu a posílat přímo použitelný autentizátor prostřednictvím SSL na server
- Navrhuje se „hashovat“ na obou stranách komunikace – jednak na klientu (pre-hashing), s využitím zřetězení výzvy serveru, hesla a náhodně zvolené odezvy klienta, výsledek se pošle v SSL kanálu na server, který provede hash zřetězení klientského autentizátoru se solí a výsledek porovná s uloženou hodnotou. Samozřejmě je správné používat místo hashe některou funkci (PBKDF2, bcrypt, scrypt apod.).

Další zesílení

Existují racionálně vypadající návrhy na zesílení bezpečnosti „hashování“ hesla:

- dynamický počet iterací hashovací funkce, v závislosti na hodnotě hesla
- uložení soli jinde než „hashe“ hesla

K zapamatování na závěr

- Upřednostňujeme délku hesla před množinou znaků v hesle (lineární x logaritmický růst entropie), heslo složené z písmen by mělo mít alespoň 12 (15 znaků)
- Jakékoliv slovo je ve slovníku, přidat „zesílení“ typu 123 nakonec nebo počáteční velké písmeno nepomůže
- Passphrase nemá obrovskou entropii danou počtem znaků ale pouze entropii danou počtem slov, kterých musí být alespoň 5
- K uložení hesla nelze použít samotný hash hesla, nezbytná je sůl (proti rainbow tables) a použití výpočetně/paměťově náročné jednosměrné funkce typu PBKDF2, bcrypt, scrypt (proti GPU)
- Ověření hesla kombinací výpočtu na klientu i serveru je lepší, než výpočet pouze na jediné straně
- Přidání dalšího faktoru, i jednoduchého, je lepší, než trvat na silném heslu, které je nutno často zadávat

Nebo ještě jednodušeji

Citace (neznámý autor)

„Proč používáš tak dlouhý heslo ?“

„Protože krátký je k ničemu“

Dotazy?

Ivo Rosol
ředitel vývojové divize
OKsystem a.s.
rosol@oksystem.cz