Виконав:

Коломієць А.Ю.

Email: andriy.kolomiets.work@gmail.com

GitHub: https://github.com/andrew-kolomiets/

Практичне завдання:

«Робота з паролями та геш-значеннями»

Аргументація стійкості паролю

0123456789 - алфавіт складається з 10 цифр, якщо пароль складається з ***n*** цифр, то за правилом добутку, на кожну позицію можемо обрати одну з 10 цифр, складність перебору становить: $\boldsymbol{10^n}$

Abcdefghijklmnopqrstuvwxyz - алфавіт складається з 26 символів, аналогічно попередньому алфавіту, на кожній позиції паролю, що складається з ***n*** символів, може бути одна з 26 букв, і за правилом добутку маємо: $\boldsymbol{26^n}$

aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ - аналогічно міркуванням раніше: $(26_{маленькі\ літери}+26_{великі\ літери})^n = 52^n$

попередній варіант з додаванням цифр - аналогічно попереднім міркуванням: $(10_{цифр}+52_{великі\ та\ малі\ літери})^n = 62^n$

попередній варіант з додаванням спецсимволів - нехай кількість спецсимволів дорівнює ***s***, тоді: $(10_{цифр}+52_{великі\ та\ малі\ літери}+s)^n = (62+s)^n$

Як бачимо кількість можливих комбінацій зі збільшенням алфавіту зростає, але слід враховувати при цьому різні закономірності, числа можуть повторювати, якусь рекурентну послідовність, чи літери повторювати слова з тексту. Спецсимволи, теж можуть повторювати котрусь залежність, як азбука Морзе, що складається з крапок, може нести в собі якусь інформацію.

Якщо паролі побудовані зі змістовних фраз мови, є можливість проведення атак, що спираються на взаємозалежність знаків мови, тобто шляхом обчислення ентропії і мовиних залежностей, можна зменшити можливості перебору і таким чином знайти правильний пароль чи ключ. Яскравим прикладом є статистичні атаки на шифр Віженера звичайний і з автоключем, або Афінні шифри, де можна використовуючи індекс відповідності мови та частоти комбінацій літер зломати шифр.

Найкращим варіантом підбору пароля, використовувати всі симовли, які згадані в усіх алфавітах, і поява символу на кожній позиції має бути випадкова в ідеалі, або псевдовипадковою в крайньому випадку.

Пошук прообразу геш-функції

a03ab19b866fc585b5cb1812a2f63ca861e7e7643ee5d43fd7106b623725fd67

```
linux@linux-X505BP:~/Documents/1/rainbow-table-attack$ echo -n "123" | sha3sum -a 256
a03ab19b866fc585b5cb1812a2f63ca861e7e7643ee5d43fd7106b623725fd67  -
```

d182aed568b01fee105557a1d173791c798030db267cf94e17102b94dcbbda3c

```
linux@linux-X505BP:~$ echo -n "cryptography" | sha3sum -a 256
d182aed568b01fee105557a1d173791c798030db267cf94e17102b94dcbbda3c  -
```

7b6a784b05c64d2e669e026fc61296eca2ee8acd5112eb8ae5f16023809e203b

```
linux@linux-X505BP:~$ echo -n "Distributed Lab" | sha3sum -a 256
7b6a784b05c64d2e669e026fc61296eca2ee8acd5112eb8ae5f16023809e203b  -
```

Дослідження райдужних таблиць (rainbow table)

Райдужна таблиця (rainbow table) — це попередньо обчислена таблиця, що використовується для знаходження праобразів хеш-функції, для злому гешів паролів. Паролі зазвичай зберігаються не у формі звичайного тексту, а як хеш-значення.

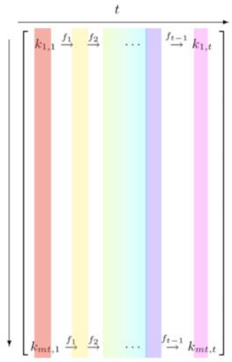
На просторах інтернету не розрізняють таблиці винайдені Філіпом Оксліном та Мартіна Геллмана, і тому я б назвав цей пункт, *дослідження райдужної таблиці* просто, бо в оригінальній статті лише <u>одна таблиця</u> використана, і там менша імовірність появи колізій, на відміну таблиць побудованих за алгоритмом Мартіна Геллмана, що помилково називають райдужними, де імовірність появи колізій гешфункцій набагато вища.

Для побудови таблиці користуємося поняттями *геш-фукції*, *функції редукції* (Мартін Геллман вперше її ввів, коли хотів побудувати атаку на DES, де треба було 64 бітні значення переганяти в 56 бітні, дана функція просто хеш перетворювала в скорочене текстове представлення, <u>це не функція знаходження праобразу</u>), та *ланцюг* (розглядаємо ланцюг фіксованої довжини, тобто однорідний ланцюг, є неоднорідні ланцюги райдужні, там атака набагато ефективніша, і кількість колізій, ще менша), що представляється наступним чином:

$$S_0 = t_0 \xrightarrow{h} h_0 \xrightarrow{r_0} t_1 \xrightarrow{h} h_1 \xrightarrow{r_1} \dots \xrightarrow{h} h_n = E_0$$

де S_0 деякий текст випадковий решта S_j теж довільні, E_0 геш значення деяке. І відбувається наступний процес побудови ланцюга: до $S_0 = t_0$ застосовуємо геш-функцію $hash(t_0) = h_0$ маємо на виході значення h_0, до якого застосовуємо функцію редукції $r_0(h_0) = t_1$, і так далі відбувається знову гешування ***hash(t_1) = h_1***, функція редукції ***r_1(h_1) = t_2***, доки не дійдемо, до ***h_n = E_0***. Це лише один ланцюг, таких ланцюгів треба будувати ***mn^2***. У Мартіна Геллмана було ***n*** таблиць розміру ***mn***. Весь ланцюг ми не зберігаємо на комп'ютері, лише пари ***(S_j,E_j)***.

$$\left|\begin{array}{c} S_0 = t_{0,0} \\ S_1 = t_{1,0} \\ \vdots \\ S_j = t_{j,0} \\ \vdots \\ S_m = t_{m,0} \end{array}\right| \begin{array}{ccccccccc} \xrightarrow{h} & h_{0,0} & \xrightarrow{r_0\circ h} & h_{0,1} & \xrightarrow{r_1\circ h} & \dots & \xrightarrow{r_{n-2}\circ h} & h_{0,n-1} & \xrightarrow{r_{n-1}\circ h} \\ \xrightarrow{h} & h_{1,0} & \xrightarrow{r_0\circ h} & h_{1,1} & \xrightarrow{r_1\circ h} & \dots & \xrightarrow{r_{n-2}\circ h} & h_{1,n-1} & \xrightarrow{r_{n-1}\circ h} \\ & \vdots & & \vdots & & \vdots & & \vdots & \\ \xrightarrow{h} & h_{j,0} & \xrightarrow{r_0\circ h} & h_{j,1} & \xrightarrow{r_1\circ h} & \dots & \xrightarrow{r_{n-2}\circ h} & h_{j,n-1} & \xrightarrow{r_{n-1}\circ h} \\ & \vdots & & \vdots & & \vdots & & \vdots & \\ \xrightarrow{h} & h_{m,0} & \xrightarrow{r_0\circ h} & h_{m,1} & \xrightarrow{r_1\circ h} & \dots & \xrightarrow{r_{n-2}\circ h} & h_{m,n-1} & \xrightarrow{r_{n-1}\circ h} \end{array} \left|\begin{array}{c} h_{0,n} = E_0 \\ h_{1,n} = E_1 \\ \vdots \\ h_{j,n} = E_j \\ \vdots \\ h_{m,n} = E_m \end{array}\right|$$



До кожної колонки вищезображеної матриці застосовується лише одна окрема функція редукції, для того, щоб якщо виникне колізія, далі ланцюги не повторювали свої частини ланцюга однаково. Якщо такі функції розфарубувати в різні кольори, вийде райдуга, звідки і походить назва.

Після побудови райдужної таблиці потрібно знайти бінарним пошуком геш значення ***y = hash(x)*** в останньому стовпчику, де представлені ***E_j,*** якщо такий геш знайдено, тоді ***x = t_{n-1}*** і є нашим шуканим паролем чи праобразом, в іншому випадку перевіряємо настпуне чи ***hash(r_{n-1}(y))*** входить в останній стовпчик з ***E_j,*** тобто ми вважаємо, що дане ***y*** зустрічається в попередньому стовпці і не досягло стовпчика ***E_j*** тому ми так робимо**,** і так далі аналогічно, якщо не знайшли на попередньому кроці шукаємо далі поки не вичерпано всі буде стовпчики. Якщо ми знаходимо збіг, ми будуємо ланцюжок на основі відповідної початкової точки ***S***$_j$; ми зупиняємось, коли отримуємо ***$t_{j,x}$*** , який має властивість ***hash($t_{j,x}$)= y,*** де ***$t_{j,x}$ = x.***

Райдужні атаки застосовуються, для того, щоб знайти праобраз або колізую геш-функції, що дозволить провести атаку на систему, котра зберігає лише геш-значення паролів, без "солі" (сіль дозволяє протидіяти такій атаці). Навіть, якщо це не пароль користувача, а випадкова фраза, яка дає однаковий геш, в такому разі атака теж успішна, оскільки при авторизації, порівнюються геші, і система видасть даний пароль за справжній.

Райдужні таблиці є практичним прикладом просторово-часового компромісу: вони використовують менше часу комп'ютерної обробки та більше пам'яті, ніж атака грубої сили, яка обчислює хеш за кожної спроби, але більше часу обробки та менше пам'яті, ніж проста таблиця, яка зберігає хеш усіх можливих паролів. Тому мета такої атаки, скоротити час, і використати по можливості пам'ять в тих обсягах, яку ми її маємо.

Література

1. Zang Y. *Rainbow Tables*. Accessed April 27, 2023. https://umm-csci.github.io/senior-seminar/seminars/fall2019/zang.pdf

2. Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. *Lecture Notes in Computer Science*. Published online August 17, 2003:617-630. doi: https://doi.org/10.1007/978-3-540-45146-4_36