

ХЕШИРОВАНИЕ ПАРОЛЕЙ

Дисциплина: Криптографические методы защиты информации

Преподаватель: Миронов Константин Валерьевич

Поток: БПС-3, ИКТ-5

Учебный год: 2020/21



Общие сведения

Хеширование паролей

- Если пароли пользователей хранятся в системе напрямую, то доступ к системным файлам равносителен доступу к паролям
- Можно хранить в системе только хеш-функции от паролей
 - Пользователь вводит пароль
 - От пароля вычисляется хеш-функция, после чего пароль стирается
 - Хеш-функция сравнивается с хранящейся в таблице
- С помощью хеширования можно получить на основе пароля секретный ключ и потом использовать его для шифрования

Пользователь	Хэш-функция пароля
...	

Таблица проверки паролей

Содержание лекции

- **Атаки на хешированные пароли и защита от них**
 - **Взлом полным перебором**
 - **Таблицы предрассчитанных цепочек**
 - **Радужные таблицы**
 - **Соленые пароли**
- Специализированные функции для хеширования паролей



Таблица полного перебора

- У злоумышленника есть доступ к базе хешей; его задача – подобрать пароль, соответствующий известной хеш-функции
- Заданы хеш-функция от пароля $h=f(p)$ и два множества – множество возможных хеш-функций H и множество предполагаемых паролей P
- При взломе полным перебором необходимо рассчитывать $h(p)$ для любых p , пока не найдется совпадение
- Можно заранее рассчитать таблицу и искать по ней пароль, соответствующий хеш-функции

Пароль	Хэш-функция
...	...

- Недостаток – нужен большой объем памяти для хранения

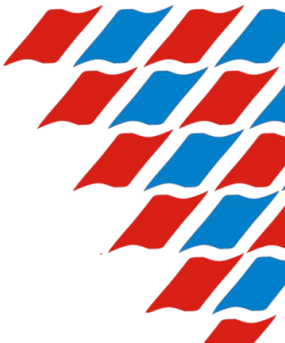


Таблица предрассчитанных цепочек

- Задается **функция редукции** $p=r(h)$
 - Аргумент – любой элемент H , значение – любой элемент P
 - Вероятность коллизии должна быть минимальна
- Берется некоторый пароль p_1
- Рассчитывается цепочка вида $\{p_1, h_1, p_2, h_2, \dots, p_n, h_n\}$, где каждая хеш-функция вычисляется от соответствующего пароля, а каждый пароль является функцией редукции от предыдущей хеш-функции:

$$p_1 \xrightarrow{\text{hash}} h_1 \xrightarrow{r} p_2 \xrightarrow{\text{hash}} h_2 \xrightarrow{r} \dots p_n \xrightarrow{\text{hash}} h_n$$

Таблица предрассчитанных цепочек

- Из цепочки удаляются все элементы кроме первого пароля и последней хеш-функции
- Составляется таблица из множества таких цепочек
- Необходимо по возможности охватить все множество P
- Поиск пароля p_x по известному $h = \text{hash}(p_x)$ в таблице

шаг 1. поиск h в таблице

шаг 2. если h найдена – перерассчитать цепочку, которая ей кончается, и взять из нее нужный пароль

шаг 3. иначе $h = \text{hash}(r(h))$

шаг 4. повторять шаги 1-3, пока h не найдется

Таблица предрассчитанных
цепочек

Пароль 1	Хэш-функция n
...	...

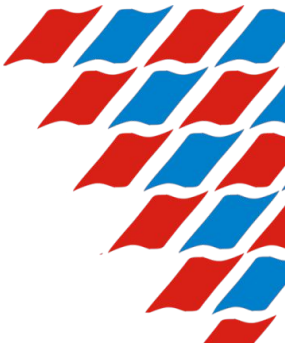


Таблица предрассчитанных цепочек

Недостаток – чувствительность к коллизиям функции редукции

- Разные цепочки могут сливаться друг с другом
- Таблица оказывается избыточной
- Часть паролей может оказаться неучтенной

p	h(p)	r(h(p))	h(r(h(p)))	r(h(r(h(p))))	h(r(h(r(h(p)))))
...	...	x1	y1	x2	y2
x1	y1	x2	y2	x3	y3
...	x1	y1



Радужная таблица

- Более устойчивая к коллизиям модификация таблицы предрассчитанных цепочек
- Задаются различные функции редукции $r_1 \dots r_{n-1}$
- Слияние цепочек происходит только если коллизия функции редукции происходит на одном и том же звене цепочки
- Это слияние можно обнаружить в процессе составления таблицы
 - Если хеш-функция, лежащая в конце составленной цепочки, совпадает с какой-либо хеш-функцией из таблицы, значит имела место коллизия
 - В таком случае цепочка не записывается в таблицу

$$p_1 \xrightarrow{\text{hash}} h_1 \xrightarrow{r_1} p_2 \xrightarrow{\text{hash}} h_2 \xrightarrow{r_2} p_3 \xrightarrow{\text{hash}} h_3 \xrightarrow{r_3} \dots \xrightarrow{r_{n-1}} p_n \xrightarrow{\text{hash}} h_n$$

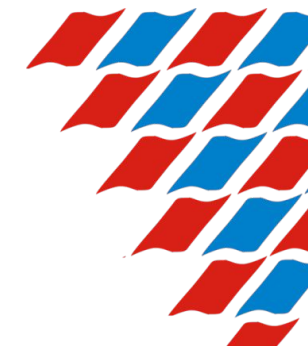
Пароль 1	Хэш-функция n
...	...

Соленые пароли

- Криптографическая соль [salt] – случайный код, который используется для обеспечения непредсказуемости при хешировании паролей
 - Либо $h = \text{HASH}(p || s)$
 - Либо в стандарте хеш-функции указан дополнительный параметр $h = \text{HASH}(p, s)$
 - В качестве соли может использоваться вектор инициализации
- В файле проверки паролей хранятся значения соли и хеш-функции пароля для каждого пользователя

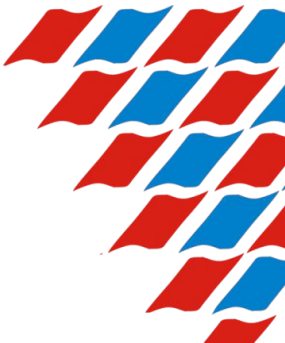
пользовате ль	соль	хэш- функция
...

- Преимущества:
 - Соль защищает от перебора паролей при помощи радужных таблиц
 - Если у пользователя есть два аккаунта с одним и тем же паролем, хеш-функция от этих паролей будет разная
- Безопасная длина соли ~ 128 бит



Содержание лекции

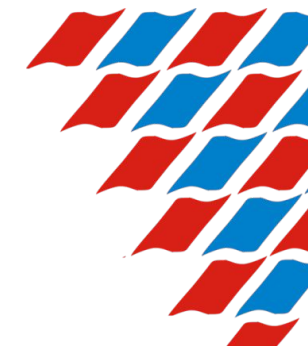
- Атаки на хешированные пароли и защита от них
- **Специализированные функции для хеширования паролей**
 - **Общие сведения**
 - **PBKDF2**
 - **Scrypt**



Общие сведения

KDF

- **Функции формирования ключа** [Key Derivation Function, KDF] – специализированные хеш-функции для получения ключей симметричного шифрования на основе произвольных данных
- Также применяются для:
 - хеширования паролей
 - контроля целостности записей в распределенных реестрах
- Если обычные ХФ должны по возможности вычисляться быстро, то вычисление KDF должно быть ресурсоемким и трудно распараллеливаемым
- Обычные ХФ часто вычисляются от больших файлов, KDF как правило от небольших объемов данных



Password-based KDF, RSA Laboratories, 2000

Параметры:

$K = \text{PBKDF2} (\text{PRF}, P, S, c, kLen)$

- PRF [pseudorandom function] «псевдослучайная функция» – используемый в рамках PBKDF2 алгоритм вычисления имитовставки (как правило, HMAC)
- P - Пароль
- S – соль
- c – количество раундов (рекомендуется от 1000)
- kLen – длина ключа в байтах



PBKDF2

- Длина ключа может быть различной
- Ключ разбивается на блоки с такой же длиной, как у имитовставки
- Каждый i -й блок $K[i]$ вычисляется по следующему алгоритму:

$U[0] = S || i$; // суммарная длина S и i должна быть равна длине имитовставки

$K[i] = 0$;

для j от 1 до c

$U[j] = \text{PRF}(P, U[j-1])$; // P трактуется как ОТ, а U - как ключ имитовставки

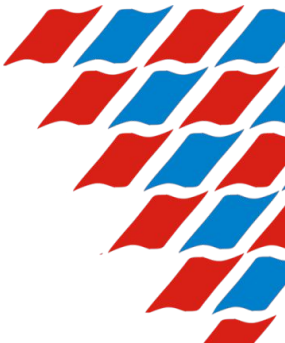
$K[i] = \text{xor}(K[i], U[j])$;

конец



PBKDF2

- WPA-PSK:
 - PBKDF2(HMAC-SHA2, P, ssid, 4096, 256)
где ssid – идентификатор точки доступа
- Р 50.1.111 2016
 - PBKDF2(HMAC-Стрибог-512, P, S, >1000, 256)
- Злоумышленник не может распараллелить вычисление PBKDF2, если длина выходного ключа меньше, чем длина блока
- Однако можно параллельно вычислять хеши для разных вариантов пароля



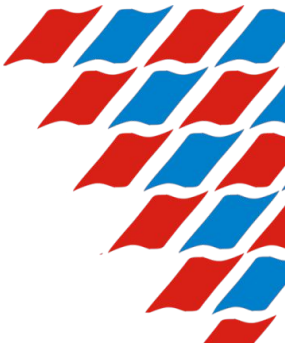
scrypt

- Колин Персиваль, 2009
- Для быстрого вычисления требуется большой объем оперативной памяти, поэтому распараллеливание на ПЛИС или GPU не дает значительного преимущества
- В рамках стандарта используются:
 - PBKDF2 на основе HMAC от SHA256
 - алгоритм Salsa20/8 (Salsa20 с количеством раундов 8 вместо 20)

- Параметры:

$$K = \text{scrypt}(P, S, c, r, p, kLen)$$

- P – пароль
- S – соль
- c – количество раундов
- r – размер блока в кибибитах (1 Киб = 128 Б = 1024 б)
- p – степень параллельности
- kLen – длина ключа



Алгоритм:

```
V = PBKDF2 (P, S, 1, p*r*128)  \\ V - массив из p блоков по 128*r байт
для i от 0 до p-1 выполнить V[i]=MIX(V[i], c)
K = PBKDF2 (P, V, 1, kLen)
```

Функция $X = \text{MIX}(V, c)$:

```
X = V
для i от 0 до  $(2^c) - 1$  выполнить
    V[i] = X
    X = BlockMIX(X)
конец
шаг 3. для i от 0 до  $(2^c) - 1$  выполнить
    j = X mod  $2^c$ 
    X = BlockMIX(X)
конец
```



Функция $A = \text{BlockMIX}(B)$:

```
// B - массив 64-битных блоков
```

```
r = length(B) / 128
```

```
X = B[2*r-1]
```

```
для i от 0 до (2^c)-1 выполнить
```

```
    X = Salsa20_8(X xor B[i])
```

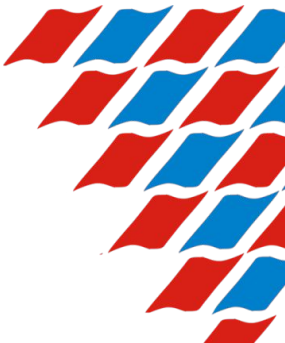
```
    Y[i] = X
```

```
конец
```

```
A = Y[0] || Y[2] || ... || Y[2*r-2] || Y[1] || Y[3] || ... || Y[2*r-1]
```

Выбор параметров алгоритма:

- Требуемый объем оперативной памяти для быстрого вычисления $128*r*c$ байт
- Рекомендуемая версия параметров $c=16384$ $r=8$ $p=1$
- «Криптовалютная» (слабая) версия параметров $c=1024$ $r=1$ $p=1$



Темы докладов

- **bcrypt**
- **Argon2**

