Passwords

The Good, The Bad and The Ugly

or

How To Secure Your Passwords

Adir Abraham

adir@computer.org

What is a password?

- Secret characters used for authentication or access approval
- May be machine generated, allocated by the provider of the service or user-generated.

Problems with storing passwords

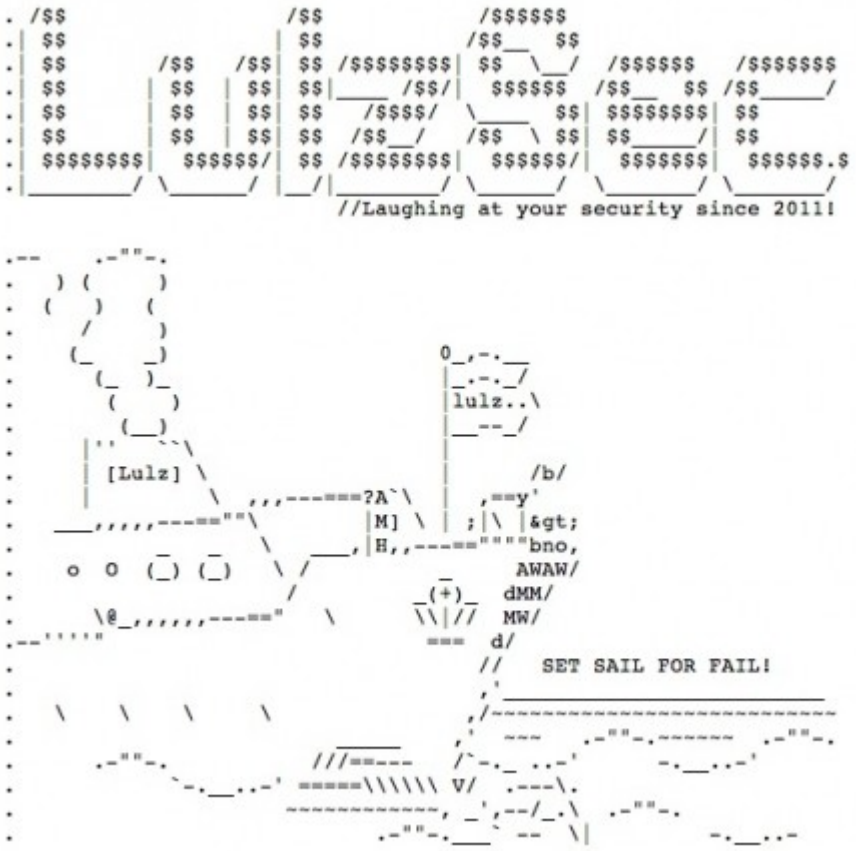
- Many users – lots of troubles
- Simple passwords (used by majority)
- Data is physically accessible
- Data is virtually accessible

Storing passwords in a server

- Simple authentication method – checking if both user and password are correct by saving user-password list in the database
- Many times, these user-password lists are saved as plain text in big companies.

Some Success Stories

- Sony 77M users' database (including passwords, e-mails, home addresses and date of birth) was fully accessible as plain text



Some Success Stories

- 0xOmar was a KSA hacker who published credit card numbers of tens thousands of Israelis
- These numbers were (illegally) stored in many Israeli websites. They were fully matched as plain text with their users, and passwords.

Danger for both the users and the companies

- Many users use the same (simple or complicated) password in many sites.
- Which means that if one site is hacked, you have to change your password in many other sites as well.
- What about secret questions and answers?
- What if the website owner doesn’t even know that his site was hacked?

The Solution: encrypt your passwords

- We need to find a way to be able to authenticate the real password of a specific user, without storing the password as plain text.
- If you hack my site or attempt to hack, you will not be able to get my users (and passwords) list.

First Step: Hash Functions

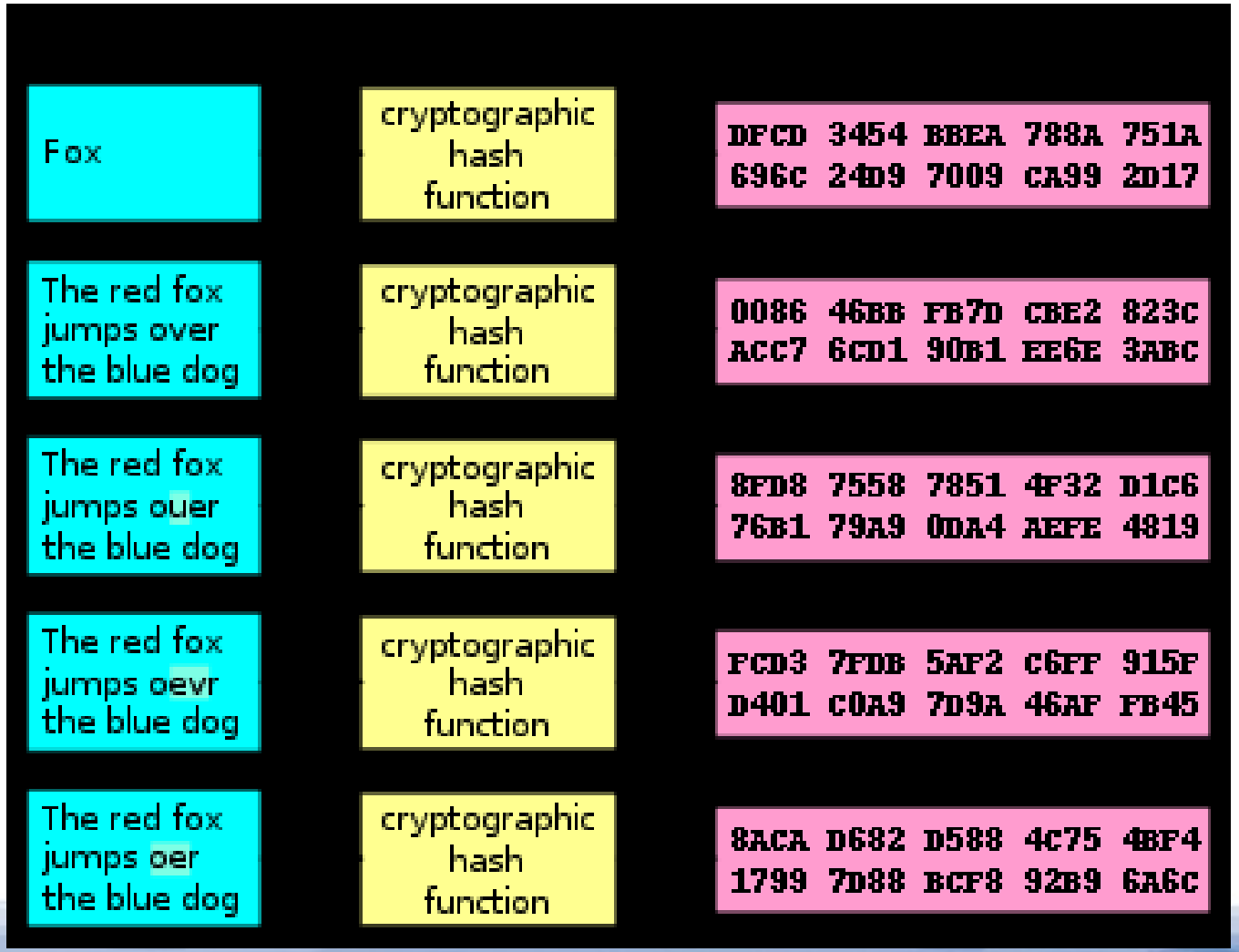
- Hash function is a function that maps large data sets called keys, to smaller data sets of fixed length.
- Since the data set is smaller, the image is smaller, hence collisions (i.e. Two different keys which get the same value) may occur.
- Hash functions uses one-way-function, which is a function that is “easy” to compute on every input, but “hard” to invert given the image (data) of a random input.

Cryptographic Hash Function

- Since naive mapping can be done quite easily, we will want a cryptographic hash function which will manipulate the value s.t. any change will make a change in the hash value itself.
- The hash result is based on a one-way-function, hence it is “hard” to guess what the source is.
- Examples for such functions are MD5, SHA-1, SHA-2 and SHA-3 which was recently declared.

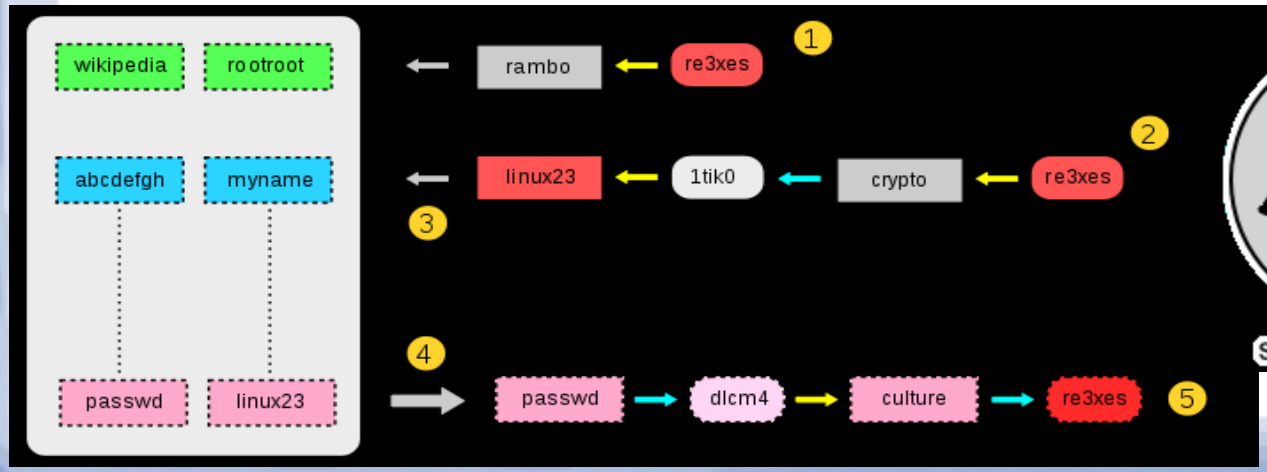
Cryptographic Hash Function

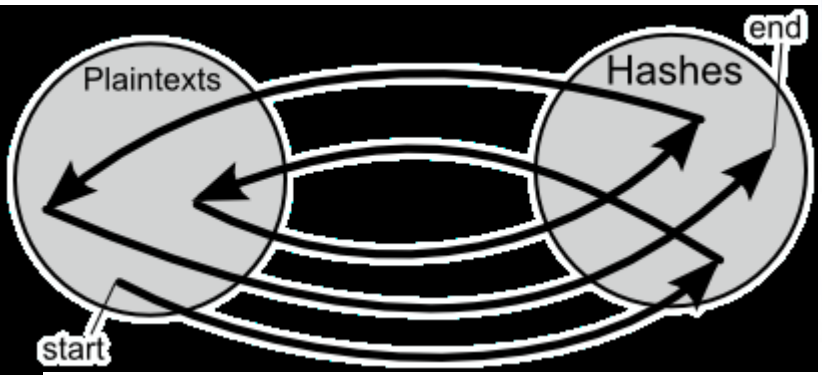
- Example (notice the changes in a single character)



But Hash Functions Are Not Enough

- Hash Functions will make a hard time for the cracker, but today using enough (distributed) power, you can easily hack 8-characters password, which is considered quite strong.
- Since guessing might take a long time, crackers use lists which already have a string and its hash result, for low amount of characters.
- Crackers also use rainbow tables which include pre-computed tables. In this list, we will check for the closest value in a series of reductions until we find a match.





Second Step: Use SALT

- SALT is a static, (very) long string which is chosen by the admin.
- It is concatenated to each password before using the hash function.
- It is a secret. Only the admin should know it. The user only has to remember his password. Suppose the user's password is 8 characters long and the SALT is 100 characters long – this makes a 108 characters "password", while 100 of them are not known to anyone else (including the user).
- This way, it makes using the methods described in the first step to hack the system, impractical.

Bonus Step: Use Double Salt

- Once someone knows the SALT of your system, it becomes an easier mission (using similar methods mentioned at the first step) to get your users and passwords list.
- If your systems are very sensitive and you can Trust No One, you can use Double Salt.
- Double Salt uses an extra Salt, a dynamic one. Each user ID gets its own 2nd (dynamic) Salt, making it even harder to guess since you have to guess the right Salt for the right user.
- In addition to that, splitting the hashing process to another server (where it will be computer), may also help, since another server has to be cracked in order to get the rest of the list (suppose the passwords list are in one server and users list is in another server)

Bonus Step: Use Key Stretching

- When someone tries to hack our system, it can take a lot of resources from our server, adding extra computational operations leading to Denial of Service.
- Regular cryptographic hash functions allow the attacker to get the results quickly, hence result in Denial of Service.
- Suppose we can “slow down” the hashing operation, now, the attacker will have to wait for years until he gets the results.

Bonus Step: Use Key Stretching

- Using bcrypt which is based on BLOWFISH, allows us to add a “slow down” timer. Suppose each guess of user takes at least 3 seconds, guessing 1000 times one user will take him 3000 seconds (for each user).

  For example, in PHP:

- *$hashed_input = crypt($input, '$2a10'.$system_salt);*
- *$2a to choose bcrypt, $10 is 2^10*
- This way we cause delaying and keep our system safe.
- If we delay it too much, we may damage the user experience, cause a Denial of Service of our own system.

Which Cryptographic Hash Function to Use?

- The most popular hash functions nowadays are MD5, SHA-1, SHA-2, Whirlpool, bcrypt, scrypt and PBKDF2.
- The rule is to use the function which makes the most difficulties to the attacker. Today, MD5 and SHA-1 have weaknesses which make them exploitable, while SHA-2 and Whirlpool are considered safe, according to the National Institute of Standards and Technology (NIST)
- PBKDF2, bcrypt, and scrypt all use large random "salt" values to make sure that each user's password is hashed uniquely. Attacking 100 password hashes will take 100 times longer than attacking one hash. Attacking a million will take a million times longer, etc. With SHA-2, the attacker can try to crack thousands or millions of hashes at the same time with very little slow down.

Simple HOWTOs

- Hashing the passwords is easily done using one SQL query. For example:
- Mysql:

  UPDATE users SET password = SHA2(CONCAT('myRandomS4lt', password, user_id), 512);

- Microsoft SQL SERVER:

  UPDATE users SET password = HASHBYTES('SHA2_256', CONCAT('myRandomS4lt', password, user_id));

- In this example we use a static SALT (user_id) and a dynamic one (“myRandomS4lt) for each user.

Simple HOWTOs

- Bcrypt and scrupt are not supported by SQL, hence we will have to do some conversion, using PHP, for example:

```
 // Get users details from DB
$results = $con->query("SELECT user_id, password FROM users");
$users = $results->fetch_all(MYSQLI_ASSOC);


   // Set the hashed password for every user
foreach ($users as $user)
{
   $plain_pass = $user['password'];
      // Hash the password using the static salt
  // and the user ID as dynamic salt
   $hashed_pass = crypt(    $plain_pass, '$2a$10$'
...
```

In Short

- Don’t use Plain Text passwords (including in papers)
- Use cryptographic hash functions which are considered safe (SHA-2 family, Whirlpool) and don’t use cryptographic hash functions which are considered weak (SHA-1, MD5)
- Use SALT and/or Double SALT
- Consider using key stretching to slow down the hashing operation, giving an attacker a hard time while trying to crack your system.

Use The Guide

- The lecture slides are based on the passwords security guide of the Israeli Internet Society (ISOC-IL), available for both developers and managers here: http://www.isoc.org.il/techinfo/ref_pas_guide.html
- In that guide you get a more detailed explanation, including more code examples and references to more useful sites.
- Feel free to share that guide (and these lecture slides), based on CC BY 2.5
- Enjoy protecting your environment :-)

Questions?