TL;DR

Even without authentication bypass, passwords may be weaker than what the typical user thinks. Thankfully, there are certain measures that can be implemented to mitigate such attacks.

Introduction

My birthday is coming and that led me to think about the birthday paradox, passwords and how many ways a password can be "guessed" head-on (i.e. not via methods which bypass the need to know the password such as "passing the hash"). These are due to several mistakes that remain, even for enterprises that apply techniques such as account lockouts, complexity rules, etc. In general, users (referring to developers, typical corporate employees and management) think of themselves as reasonably secure and that they are different from the people who are hacked ("the other guys") . I will cover two mistakes in this post and provide some recommendations to reduce the risks of these mistakes.

Mistake 1 – Users assuming their password is too difficult

In an environment where users observe certain measures to make password guessing difficult – e.g. password rules and account lockout, users may think that their password is not easily obtainable.

This is a mistake.

Several passwords that appear clever and unique at first glance are actually well-known to attackers.

Here are a few from the UK National Cyber Security Centre's top 100,000 passwords:

- `3rJs1la7qE`
- `PE#5GZ29PTZMSE`
- `!~!1`
- `N0=Acc3ss`

The second measure users see - Account lockout - is also a defence which can be bypassed. Account lockout, as the name implies, locks an account for a period of time after a certain number of wrong password attempts are attempted. A technique known as password spraying can be used to prevent this lockout.

Password sprays are a method of attempting to gain access by iterating through a list of user names for a given password. Since some account lockouts are set to lockout only if a password is entered wrongly for a number of times within a certain time window or to lock an account for a short period of time after a number of incorrect attempts, the password spray technique attempts to bypass both problems. By going through a large number of user accounts, the attacker would hope that by the time the script that is performing the attack is performing the password attempt, the time window for either lockout or the attempt window is past. Just like password lists in a brute force attack, there are also common username list such as this link by the Github user "insidetrust" and another link by the Github user "danielmiessler". Several other account discovery techniques and account footprinting techniques can also be used.

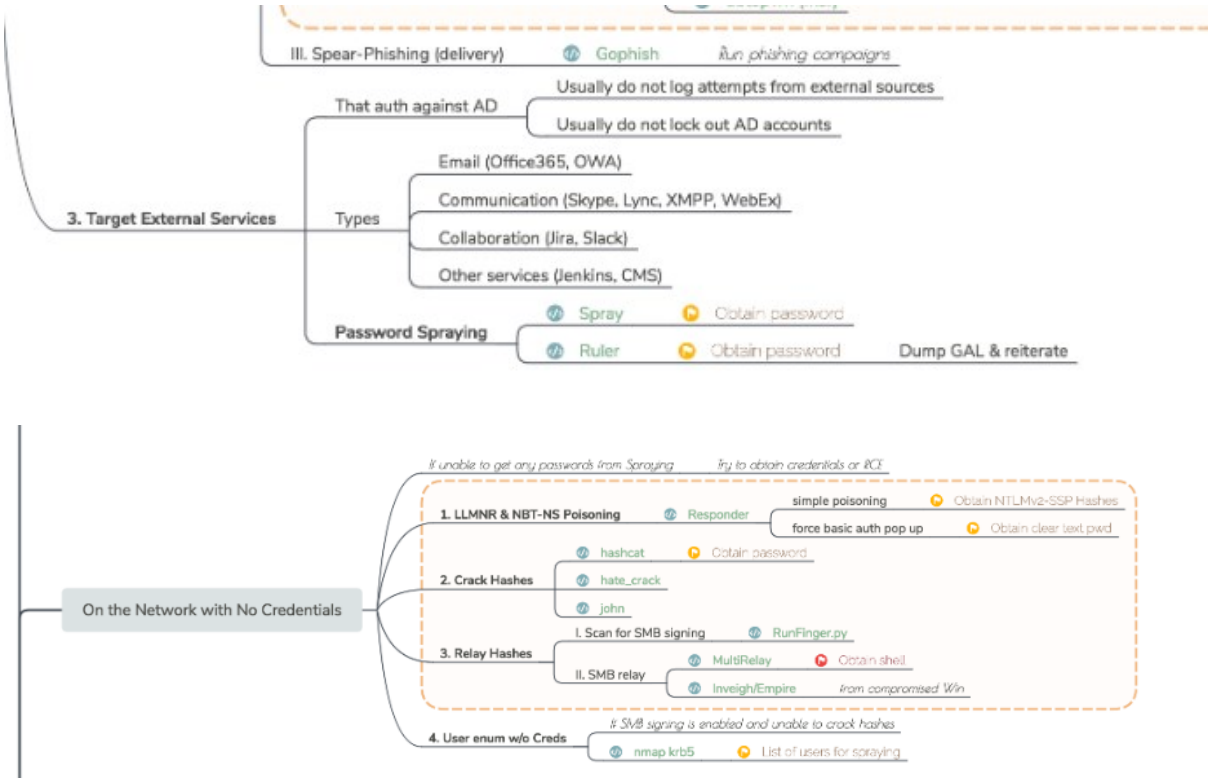

Figure: Red Teaming Mind Map from The Hacker Playbook 3

A common password cracking tool, hydra, offers some options to attack targets using password spraying. The following command tries one password across a list of user accounts:

```
hydra -L default_logins.txt -p test ftp://192.168.1.1/
```

Another tool by Trustwave, spray, can also be used to perform the attack. A typical command to attack SMB service is as follows:

```
spray.sh –smb 10.10.10.10 usernames.txt passwords.txt 1 35
SPIDERLABS
```

These two tools can simply use the common user names obtained online or through the account footprinting techniques to have a list of user names to perform the password spray attempts.

Mistake 2 – Applications not keeping passwords safe

Organizations and the quick development lifecycle of many companies' applications, especially internal ones, mean that there are many with poor storage and treatment of passwords. Just early this year, it was revealed that Facebook stored passwords without "hashing" them.

Even if they did store them with hashes, there is no telling how secure this "hashing" process might be. Hashing performs a one-way transformation on the submitted password, turning the password into another sequence of characters, which we can call "hashed password". By design, a "one-way"

hashing algorithm is intended to be irreversible. However, there is nothing to say that an attacker cannot repeat all the "one-way" operations repeatedly. An insecure hash would be quickly broken by an attacker with a decent amount of computing power. For example, a 8 GPU system can guess 2.003 x 10^{11} possible passwords using the weak MD5 hash compared to 3.49 x 10^{6} of the better scrypt hash.

This is not helped by posts on developer's Q&A sites that provide advice to use MD5 hashing without salt:

> You can either use
>
> ```
> var md5 = new MD5CryptoServiceProvider();
> var md5data = md5.ComputeHash(data);
> ```
>
> or
>
> ```
> var sha1 = new SHA1CryptoServiceProvider();
> var sha1data = sha1.ComputeHash(data);
> ```
>
> To get `data` as byte array you could use
>
> ```
> var data = Encoding.ASCII.GetBytes(password);
> ```
>
> and to get back string from `md5data` or `sha1data`
>
> ```
> var hashedPassword = ASCIIEncoding.GetString(md5data);
> ```

A number of implementations may not have the correct "salt" length as well. A salt is added to the hashing process to force their uniqueness, increase their complexity without increasing user requirements. The use of different password "salt" would also make the case of multiple users using the same password less obvious if the entire password database was compromised.

Depending on the size of the application, the number of bytes allocated for the "salt" may be insufficient. For example, a salt of seven bytes is minimally sufficient for 100 million "salts" (which should allow every user to change passwords & corresponding salt for a few times over the lifetime of the account). A "rough" calculation that can be used is base on assuming a lifetime change "limit" of 10 password changes. If there are a projected maximum of 10 million users, the calculation of the minimum "salt" length would be:

1. 10 million users multiple by 10 password changes = 100 million
2. Solve for x in the equation $2^{x} = 10 \times 10^{7}$
3. The number x is the number of bits of the salt. Multiply by 2.
4. In this example, the number should be about 54 (round up).
5. Using ASCII, the minimum number of characters is 8 characters.

What we can do

All of the following should be done:

1. Use “slow” hashes with long random “salts” when storing passwords. An article on how to do this for ASP.NET is Troy Hunt’s article here.
2. Implement long hashes that are at least 16 characters, as per OWASP guidelines.
3. Implement IP whitelisting for internal applications.
4. Use Multi Factor Authentication, like using Yubikey
5. Mandate long passwords, if passwords must be used
6. Keep password safe or don’t have them (e.g. use dynamic secrets)