Password Hashing in Django

Adrian Kosmaczewski

2021-07-23

This technique can be useful when migrating applications from Django to ASP.NET or PHP, keeping usernames and passwords intact.

Passwords in Django are stored in two ways (in the `core_vipassuser` table of the database):

1. When accounts have been created without a valid password, or using Facebook or any other OAuth2 provider, the passwords are stored like this: `!mafzMhEywOfhrFMvFJ16JhB1uQiAvHRaN4KuEqfg` which is a random string prefixed with `!`. This is shown in the `make_password()` function in django.contrib.auth.hashers when the `password` parameter is `None`. This prevents usage of the system with an empty password.

```
UNUSABLE_PASSWORD_PREFIX = '!'
UNUSABLE_PASSWORD_SUFFIX_LENGTH = 40

def is_password_usable(encoded):
    """
    Return True if this password wasn't generated by
    User.set_unusable_password(), i.e. make_password(None).
    """
    return encoded is None or not encoded.startswith(UNUSABLE_PASSWORD_PREFIX)

def make_password(password, salt=None, hasher='default'):
    if password is None:
        return UNUSABLE_PASSWORD_PREFIX + get_random_string(UNUSABLE_PASSWORD_SUFFIX_LENGTH)
```

OAuth2 tokens are stored in the `social_auth_usersocialauth` table.

2. Django passwords are hashed using PBKDF2 with test vectors here. There are implementations of this algorithm in several languages. Hashed passwords in a Django DB look like this: `pbkdf2_sha256$36000$5LjfzfBwQAVI$sbEcyHm7a27GFKOgOOymu+mauqVLhS2QKQE4yLk8B9Y=` where the following elements are separated by `$`:

    1. `pbkdf2_sha256` indicates the hashing algorithm
    2. `36000` is the number of iterations

3. `5LjfzfBwQAVI` is the salt
4. `sbEcyHm7a27GFKOgOOymu+mauqVLhS2` is the actual hash.

This format is referred as `<algorithm>$<iterations>$<salt>$<hash>` in the Django documentation:

> By default, Django uses the PBKDF2 algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST. This should be sufficient for most users: it's quite secure, requiring massive amounts of computing time to break.

Validating Django Users from PHP

Using that information, the following code takes a password (given by the user) and creates the same hash as in point 2.4 above; that can be used to validate that the user is effectively whoever they claim to be.

```php
<?php
hash_pbkdf2('sha256', 'password', 'salt', 36000, 32, true);
```

The `hash_pbkdf2()` function is available since PHP 5.5.

Validating Django Users from C

```c
using System;
using System.Text;
using System.Security.Cryptography;

namespace App
{
    class Program
    {
        static string PBKDF2(string password, string salt, int iterations, int size)
        {
            byte[] saltBytes = Encoding.UTF8.GetBytes(salt);
            using (Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password, saltBytes, i
            {
                byte[] resultBytes = pbkdf2.GetBytes(size);
                return Convert.ToBase64String(resultBytes);
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine(PBKDF2("password", "salt", 36000, 32));
        }
    }
}
```

More information

Password management in Django, and GitHub - defuse/password-hashing: Password hashing code., where the code above was adapted from.