Bruteforcing Passwords

After having success at `username enumeration`, an attacker is often just one step from the goal of bypassing authentication, and that step is the user's password. Passwords are the primary, when not the only one, security measure for most applications. Despite its popularity, this measure is not always perceived as important by both end-users and administrators/maintainers. Therefore, the attention it receives is usually not enough. Wikipedia has a page that lists the most common passwords with a leaderboard for every year starting from 2011. If you have a quick look at this table you can see that people are not so careful.

Password Issues

Historically speaking, passwords suffered from three significant issues. The first one lies in the name itself. Very often, users think that a password can be just a word and not a phrase. The second issue is that users mostly set passwords that are easy to remember. Such passwords are usually weak or follow a predictable pattern. Even if a user chooses a more complex password, it will usually be written on a Post-it or saved in cleartext. It is also not that uncommon to find the password written in the hint field. The second password issue gets worse when a frequent password rotation requirement to access enterprise networks comes into play. This requirement usually results in passwords like `Spring2020`, `Autumn2020` or `CompanynameTown1`, `CompanynameTown2` and so forth.

Recently NIST, the National Institute of Standards and Technology refreshed its guidelines around password policy testing, password age requirements, and password composition rules.

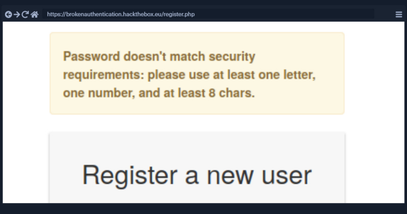
The relevant change is:

```
Verifiers SHOULD NOT impose other composition rules (e.g., requiring mixtures of different character types
```

Finally, it is a known fact that many users reuse the same password on multiple services. A password leak or compromise on one of them will give an attacker access to a wide range of websites or applications. This attack is known as `Credential stuffing` and goes hand in hand with wordlist generation, taught in the Cracking Passwords with Hashcat module. A viable solution for storing and using complex passwords is password managers. Sometimes you may come across weak password requirements. This usually happens when there are additional security measures in place. An excellent example of that is ATMs. The password, or better the PIN, is a just sequence of 4 or 5 digits. Pretty weak, but lack of complexity is balanced by a limitation in total attempts (no more than 3 PINs before losing physical access to the device).

Policy Inference

The chances of executing a successful brute force attack increase after a proper policy evaluation. Knowing what the minimum password requirements are, allows an attacker to start testing only compliant passwords. A web app that implements a strong password policy could make a brute force attack almost impossible. As a developer, always choose long passphrases over short but complex passwords. On virtually any application that allows self-registration, it is possible to infer the password policy by registering a new user. Trying to use the username as a password, or a very weak password like `123456`, often results in an error that will reveal the policy (or some parts of it) in a human-readable format.



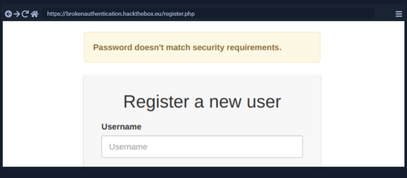
Policy requirements define how many different families of characters are needed, and the length of the password itself.

Families are:

- lowercase characters, like `abcd..z`
- uppercase characters, like `ABCD..Z`
- digit, numbers from `0 to 9`
- special characters, like `./.?!` or any other printable one (`space` is a char!)

It is possible that an application replies with a `Password does not meet complexity requirements` message at first and reveals the exact policy conditions after a certain number of failed registrations. This is why it is recommended to test three or four times before giving up.

The same attack could be carried on a password reset page. When a user can reset her password, the reset form may leak the password policy (or parts of it). During a real engagement, the inference process could be a guessing game. Since this is a critical step, we are providing you with another basic example. Having a web application that lets us register a new account, we try to use `123456` as a password to identify the policy. The web application replies with a `Password does not match minimum requirements` message. A policy is obviously in place, but it is not disclosed.



We then start guessing the requirements by registering an account and entering a keyboard walk sequence for the password like `Qwertyiop123!@#`, which is actually predictable but long and complex enough to match standard policies.

Suppose that the web application accepts such passwords as valid. Now let's decrease complexity by removing special characters, then numbers, then uppercase characters, and decreasing the length by one character at a time. Specifically, we try to register a new user using `Qwertyiop123`, then `Qwertyiop!@#`, then `qwertyiop123`, and so forth until we have a matrix with the minimum requirements. While testing web applications, also bear in mind that some also limit password length by forcing users to have a password between 8 and 16 characters. This process is prone to error, and it is also possible that some combinations will not be tested while others will be tested twice. For this reason, it is recommended to use a table like this to keep track of our tests:

Tried	Password	Lower	Upper	Digit	Special	>=8chars	>=20chars
Yes/No	`qwerty`	X					
Yes/No	`Qwerty`	X	X				
Yes/No	`Qwerty1`	X	X	X			
Yes/No	`Qwertyu1`	X	X	X		X	
Yes/No	`Qwerti!`	X	X	X	X		
Yes/No	`Qwerty1!`	X	X	X	X	X	
Yes/No	`QWERTY1`		X	X			
Yes/No	`QWERTI!`		X	X	X		
Yes/No	`QWERTY1!`		X	X	X	X	
Yes/No	`Qwerty!`	X	X		X		
Yes/No	`Qwertyuiop12345!@#$%`	X	X	X	X	X	X

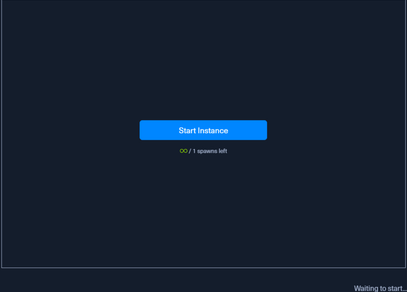
Within a few tries, we should be able to infer the policy even if the message is generic. Let us now suppose that this web application requires a string between 8 and 12 characters, with at least one uppercase and lowercase character. We now take a giant wordlist and extract only passwords that match this policy. Unix `grep` is not the fastest tool but allows us to do the job quickly using POSIX regular expressions. The command below will work against rockyou-50.txt, a subset of the well-known `rockyou` password leak present in SecLists. This command finds lines have at least one uppercase character (`'[[:upper:]]'`), and then only lines that also have a lowercase one (`'[[:lower:]]'`) and with a length of 8 and 12 chars (`^.{8,12}$`) using extended regular expressions (-E).

```
Govardhan Gujji22@htb[/htb]$ grep '[[:upper:]]' rockyou.txt | grep '[[:lower:]]' | grep -E '^.{8,12}$' 
416712
```

We see that starting from the standard `rockyou.txt`, which contains more than 14 million lines, we have narrowed it down to roughly 400 thousand. If you want to practice yourself, download the PHP script here and try to match the policy. We suggest keeping the table we just provided handy for this exercise.

Perform an Actual Bruteforce Attack

Now that we have a username, we know the password policy and the security measures in place, we can start brute-forcing the web application. Please bear in mind that you should also check if an anti-CSRF token protects the form and modify your script to send such a token.



Waiting to start...

Questions

Answer the question(s) below to complete this Section and earn cubes!

Cheat Sheet

Target: Click here to spawn the target system!

+ 1 Using rockyou-50.txt as password wordlist and htbuser as the username, find the policy and filter out strings that don't respect it. What is the valid password for the htbuser account?

Submit your answer here... | Submit | Hint

← Previous | Next →

Cheat Sheet
Resources
Go to Questions

Table of Contents

Broken Authentication
What is Authentication
Overview of Authentication Methods
Overview of Attacks Against Authentication

Login Bruteforcing
Default Credentials
Weak Bruteforce Protections
Bruteforcing Usernames
Bruteforcing Passwords
Predictable Reset Token

Password Attacks
Authentication Credentials Handling
Guessable Answers
Username Injection

Session Attacks
Bruteforcing Cookies
Insecure Token Handling

Skill Assessment
Skill Assessment - Broken Authentication

My Workstation
OFFLINE
Start Instance
OO / 1 spawns left