Analisis dan Pengembangan Algoritma *Brute Force*, *Greedy*, dan Heuristik Berbasis Regex untuk Pemecahan Kata Sandi

Marvel Pangondian - 13522075
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung 40132, Indonesia
13522075@std.stei.itb.ac.id

***Abstrak*— Algoritma *brute force* sering digunakan sebagai metode untuk memecahkan kata sandi, namun metode ini seringkali memerlukan waktu yang sangat lama. Sebaliknya, algoritma *greedy* dan heuristik yang diperkenalkan dalam makalah ini adalah sebuah pendekatan yang dapat memecahkan kata sandi dengan lebih cepat, meskipun tidak selalu berhasil menemukan kata sandi yang benar. Dalam makalah ini, penulis mengusulkan pengembangan algoritma *brute force*, *greedy,* dan heuristik berbasis ekspresi reguler (regex) untuk pemecahan kata sandi. Algoritma yang diusulkan mengkombinasikan eksplorasi menyeluruh dari *brute force* dengan pendekatan heuristik yang didukung oleh regex untuk mengurangi ruang pencarian. Implementasi algoritma ini menunjukkan bahwa kombinasi teknik ini dapat mengoptimalkan proses pemecahan kata sandi dengan kompleksitas waktu yang lebih rendah dibandingkan dengan penggunaan *brute force* murni.**

***Kata Kunci*— Algoritma *brute force*; Algoritma *greedy*; Algoritma heuristik; Ekspresi Reguler; Pemecahan Kata Sandi**

I. PENDAHULUAN

Kata sandi merupakan hal penting untuk menjaga data privat seperti data pada email, media sosial, akun bank, dan sebagainya. Namun, dengan meningkatnya kekuatan komputasi dan teknik pemecahan kata sandi, penting untuk memahami dan mengembangkan algoritma yang dapat memecahkan kata sandi dengan cepat dan efisien supaya dapat mengembangkan keamanan terhadap algoritma tersebut. Algoritma yang sering digunakan dalam peretasan sandi adalah algoritma *brute force*. Algoritma *brute force* bekerja dengan mencoba setiap kemungkinan kombinasi karakter hingga menemukan kata sandi yang benar. Meskipun metode ini menjamin keberhasilan, ia memiliki kelemahan signifikan dalam hal waktu komputasi, terutama untuk kata sandi yang panjang dan kompleks.

Oleh karena itu, dibutuhkan algoritma lain yang dapat memecahkan kata sandi dengan lebih cepat. Salah satu algoritma tersebut adalah algoritma *greedy*. Dengan menggunakan kata sandi yang paling sering digunakan serta diurutkan berdasarkan nilai, yakni nilai entropinya, algoritma *greedy* dapat dengan cepat memecahkan kata sandi. Terdapat juga algoritma heuristik yang menggunakan nilai median entropi untuk mengukur nilai heuristiknya. Namun, kedua algoritma ini tidak menjamin menemukan kata sandi yang benar; jika kata sandi yang ingin dipecahkan tidak ada dalam kamus yang digunakan, maka kata sandi tersebut tidak akan berhasil diretas.

Untuk mendapatkan keuntungan dari algoritma *brute force, greedy,* dan heuristik serta mengurangi kekurangan ketiga algoritma tersebut, penulis mengusulkan pengembangan algoritma *brute force* dan heuristik berbasis ekspresi reguler (regex) untuk meningkatkan efisiensi pemecahan kata sandi. Pendekatan heuristik berbasis regex memungkinkan identifikasi pola tertentu dalam kata sandi yang dapat digunakan untuk mempersempit ruang pencarian. Dengan mengkombinasikan kekuatan eksplorasi menyeluruh dari *brute force* dan kecepatan dari pendekatan heuristik, diharapkan algoritma yang diusulkan dapat memecahkan kata sandi dengan lebih cepat dan efisien.

II. LANDASAN TEORI

A. Kata Sandi

Kata sandi menurut *Merriam-Webster* adalah sesuatu yang memperboleh seseorang untuk mendapatkan akses. Kata sandi dapat diartikan sebagai rangkaian karakter, angka, karakter spesial, atau kombinasi ketiga hal tersebut yang digunakan untuk memvalidasi pengguna dan menjaga data pengguna. Kata sandi yang kuat biasanya terdiri dari kombinasi huruf besar, kecil, angka, serta karakter spesial dengan panjang kata sandi tersebut minimal 8.

Kata sandi yang lemah, yaitu kata sandi yang pendek dan mudah ditebak[1], dapat menyebabkan peretasan sehingga keamanan data pengguna dapat terancam. Oleh karena itu, diperlukan kata sandi yang kuat untuk mengurangi risiko peretasan dan melindungi data pengguna.

Secara umum, kata sandi dibagi menjadi dua untuk memperbolehkan seseorang mendapatkan akses yakni:

1. Otentikasi Dua Faktor (*Two-Factor Authentication*)
   Otentikasi dua faktor (2FA) adalah metode keamanan yang membutuhkan dua jenis verifikasi untuk mengkonfirmasi identitas pengguna. Kombinasi ini biasanya mencakup sesuatu yang pengguna tahu (kata sandi) dan sesuatu yang pengguna miliki (token fisik atau kode OTP)[2].
2. Otentikasi Multi-Faktor (*MFA*)

Otentikasi multi-faktor (MFA) memperluas konsep 2FA dengan menggunakan lebih dari dua metode verifikasi[3]. MFA meningkatkan keamanan dengan menambahkan lapisan tambahan, yang membuatnya lebih sulit bagi pihak yang tidak sah untuk mengakses akun.

B. Algoritma Brute Force

Algoritma *brute force* adalah algoritma yang *straightforward* dalam memecahkan sebuah masalah[4]. Algoritma ini tidak mementingkan kompleksitas maupun efisiensi, hanya mementingkan pemecahan masalah saja.

Algoritma *brute force* dapat digunakan untuk menyelesaikan sebagian besar persoalan yang ada dengan pendekatan yang mudah dibuat dan dimengerti, hanya saja dalam sebagian besar situasi algoritma ini kurang efisien dan kurang layak diterapkan dikarenakan kompleksitas waktu dan kebutuhan komputasional yang besar.

Algoritma *brute force* adalah metode sederhana namun efektif untuk memecahkan kata sandi dengan mencoba setiap kemungkinan kombinasi karakter hingga menemukan kata sandi yang benar. Metode ini menjamin keberhasilan karena pada akhirnya akan menemukan kata sandi yang benar, tetapi memiliki kelemahan signifikan dalam hal waktu komputasi. Semakin panjang dan kompleks kata sandi, semakin lama waktu yang dibutuhkan oleh algoritma *brute force* untuk menemukan kata sandi tersebut. Oleh karena itu, meskipun *brute force* adalah metode yang pasti, ia sering kali tidak praktis untuk digunakan dalam pemecahan kata sandi yang kompleks.

C. Teknik Exhaustive Search

Teknik *exhaustive search* adalah metode pencarian solusi menggunakan pendekatan *brute force* untuk berbagai masalah seperti masalah permutasi, kombinasi, atau subset dari suatu himpunan[4]. Teknik ini dilakukan dengan menghasilkan setiap solusi yang mungkin, kemudian disimpannya solusi terbaik saat itu juga (*best solution so far*), saat pencarian berakhir maka solusi terbaik yang dipilih akan dijadikan solusi persoalan tersebut.

Dalam konteks pencarian kata sandi, teknik ini melibatkan mencoba setiap kemungkinan kombinasi karakter hingga menemukan kombinasi yang cocok dengan kata sandi yang ingin dipecahkan.

Walaupun teknik ini tentu akan memberikan solusi sesuai, yakni kata sandi yang ingin dipecahkan, tetapi metode ini sering memerlukan waktu dan sumber daya yang sangat besar untuk menemukan solusi, terutama pada masalah dengan ruang solusi yang besar. Oleh karena itu akan dikembangkan algoritma gabungan *brute force* dan heuristik sehingga tidak seluruh kemungkinan perlu ditelusuri saat menggunakan teknik *exhaustive search*, metode ini dinamakan *pseudo-exhaustive search* yakni metode yang mengkombinasikan strategi heuristik dengan *completeness* dari algoritma *brute force* untuk mengoptimasi waktu komputasi.

D. Algoritma Greedy

Algoritma *greedy* adalah algoritma yang memecahkan masalah dengan mengambil solusi terbaik pada setiap langkah tanpa memperhatikan pengaruh solusi tersebut terhadap langkah - langkah berikutnya[5]. Dalam konteks pemecahan kata sandi, algoritma *greedy* menggunakan daftar kata sandi yang paling sering digunakan dan mengurutkannya berdasarkan nilai entropi kata sandi. Pendekatan ini memungkinkan pemecahan kata sandi dengan lebih cepat dibandingkan metode *brute force*. Namun, algoritma *greedy* tidak selalu menjamin keberhasilan; jika kata sandi yang ingin dipecahkan tidak ada dalam kamus yang digunakan, maka kata sandi tersebut tidak akan berhasil diretas.

E. Teknik Heuristik

Heuristik adalah pendekatan atau metode pemecahan masalah yang menggunakan teknik praktis atau aturan praktis yang berasal dari pengalaman untuk menemukan solusi yang cukup baik dengan cepat. Teknik-teknik ini dirancang untuk menyelesaikan masalah kompleks lebih efisien dibandingkan dengan metode tradisional yang menyeluruh, meskipun mungkin tidak selalu menghasilkan solusi yang optimal atau sempurna.

Dalam konteks pemecahan kata sandi, teknik heuristik dapat diterapkan untuk mempercepat proses pencarian dengan cara mengurangi ruang pencarian dan fokus pada area yang lebih mungkin mengandung solusi. Algoritma *heuristik* dan *greedy* walaupun secara konsep terlihat sama tetapi merupakan dua algoritma yang berbeda. Algoritma *heuristik* mencoba untuk menemukan solusi berdasarkan *rules of thumb*[10] untuk mengurangi ruang solusi, sedangkan algoritma *greedy* mencoba untuk mendapatkan solusi terbaik setiap langkahnya. Secara umum algoritma *greedy* termasuk *heuristik* tetapi tidak semua *heuristik* termasuk algoritma *greedy*.

F. Ekspresi Reguler

Ekspresi reguler, atau regex, adalah pola yang digunakan untuk mencocokkan rangkaian teks tertentu. Pola ini memungkinkan pencarian dan pengenalan karakter, kata, atau pola karakter dalam sebuah teks dengan cara yang efisien dan fleksibel[6].

Dalam konteks pemecahan kata sandi, ekspresi reguler akan digunakan pada fungsi heuristik algoritma *greedy* terutama dalam menghitung entropi sebuah kata sandi. Kata sandi memiliki nilai entropi/*randomness* yang menentukan kuat atau lemahnya kata sandi tersebut. Semakin besar nilai entropi sebuah kata sandi, maka semakin susah kata sandi tersebut untuk diretas.

Ekspresi reguler akan digunakan untuk menghitung nilai entropi setiap kata sandi pada kamus yang akan digunakan yang kemudian akan dijadikan representasi nilai tiap kata sandi tersebut ketika akan digunakan dalam algoritma *greedy*.

G. Entropi Kata Sandi

Entropi kata sandi adalah ukuran matematis yang digunakan untuk mengukur seberapa kuat atau acak sebuah kata sandi. Entropi dalam konteks kata sandi mencerminkan seberapa sulit kata sandi tersebut untuk ditebak atau dipecahkan melalui serangan *brute force* atau teknik lainnya. Semakin tinggi entropi kata sandi, semakin sulit kata sandi tersebut untuk ditebak.

Secara teori, entropi sebuah kata sandi dapat dihitung berdasarkan panjang dan variasi simbol dalam kata sandi tersebut. Untuk kata sandi berukuran L dengan variasi simbol

sebanyak N maka entropi E kata sandi adalah sebagai berikut[7]:

$$E = L * Log_2(N)$$

Untuk menilai variasi simbol pada kata sandi, dapat dilihat pada tabel berikut:

Tabel 2.1. Jenis karakter dan jumlah variasinya

No	*Jenis Karakter*	*Variasi Karakter (N)*
1.	Angka	10 (0 - 9)
2.	Huruf Kecil	26 (a - z)
3.	Huruf Besar	26 (A - Z)
4.	Karakter Istimewa	32 (! @ # etc)

semisal kata sandi "P1@ssWord", maka kata sandi tersebut memiliki variasi karakter angka, huruf kecil, huruf besar, dan karakter istimewa sehingga memiliki nilai N sebesar

$N\_full\_pattern = 10 + 26 + 26 + 32 = 94$

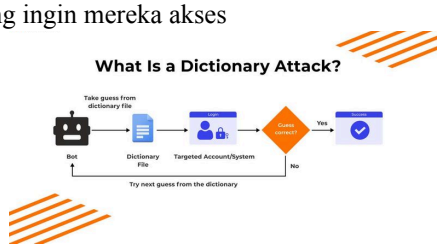
semakin panjang dan semakin bervariasi sebuah kata sandi, maka semakin besar nilai entropi dan semakin susah kata sandi tersebut untuk diretas

H. Dictionary Attack

Serangan kamus atau *dictionary attack* adalah metode yang digunakan dalam untuk meretas kata sandi, kunci enkripsi, atau mekanisme keamanan lainnya dengan memasukkan setiap kata dalam daftar yang telah ditentukan sebelumnya (kamus) yang berisi kemungkinan kata sandi. Metode ini memanfaatkan kecenderungan pengguna yang sering memilih kata-kata umum atau kata sandi sederhana, yang membuatnya lebih mudah bagi penyerang untuk menebaknya.

Dalam dunia nyata, *dictionary attack* dapat dilakukan melalui dua cara yaitu cara *online* atau *offline*[8].

1. *Dictionary attack* secara online dilakukan oleh seorang penyerang dengan identitas palsu, serangan ini lebih efektif jika penyerang memiliki daftar kata kunci yang kemungkinan besar digunakan sebagai kata sandi, tetapi dikarenakan batas percobaan yang ada ketika melakukan serangan *online* menyebabkan metode ini lebih lama dibandingkan dengan metode *offline.*
2. *Dictionary attack* secara *offline* tidak memiliki kelemahan serangan *online* yakni batas percobaan kata sandi, hanya saja metode ini lebih kompleks dibandingkan dengan metode *online* sebab penyerang perlu memiliki *file* simpanan kata sandi dari jaringan yang ingin mereka akses

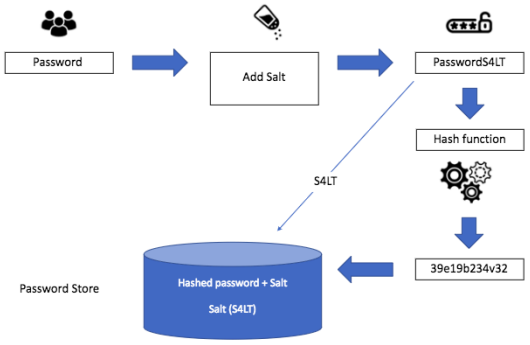


Gambar 2.1 Proses Dictionary Attack
Sumber: https://www.wallarm.com/what/dictionary-attack

I. Password Hashing

Dalam dunia nyata, kata sandi tidak disimpan dalam bentuk nyatanya melainkan dalam bentuk *hash*. *Password Hashing* adalah proses memasukan kata sandi kedalam sebuah *hashing algorithm* seperti *RSA* atau *SHA-256* sehingga kata sandi berubah dari *human readable plaintext* menjadi *unreadable text*[9]. Tujuan proses ini adalah untuk menghindari pencurian kata sandi yang oleh pihak - pihak yang tidak layak. Dengan proses *hashing*, data pengguna dapat dijaga dengan tingkat keamanan yang lebih dibandingkan dengan data yang disimpan dengan kata sandi yang tidak di *hash*.

Dalam penyimpanan kata sandi dalam *database*, terdapat proses tambahan yang dapat diterapkan yakni proses *add salt* pada kata sandi, proses ini dilakukan dengan menambahkan rangkaian karakter acak di awal atau akhir kata sandi sebelum proses *hash* , hal ini dilakukan untuk mengurangi kerentanan kata sandi pengguna terhadap peretasan.



Gambar 2.2 Proses penyimpanan kata sandi dalam database
Sumber: https://www.okta.com/blog/2019/03/what-are-salted-passwords-and-password-hashing/

III. ANALISIS PERMASALAHAN

A. Pemecahan Kata Sandi Dengan Brute Force

Salah satu cara untuk memecahkan kata sandi yang sudah dalam bentuk *hash* adalah dengan menghasilkan setiap kemungkinan kata sandi yang ada, lalu masing - masing kata sandi tersebut diubah ke dalam bentuk *hash* dan dibandingkan dengan hasil *hash* kata sandi tujuan. Pemecahan kata sandi menggunakan *brute force* bergantung kepada panjang dan variasi dari kata sandi tersebut, kata sandi yang dibatasi dengan angka saja akan lebih mudah dan lebih cepat diretas menggunakan algoritma *brute force* dibandingkan dengan kata sandi yang menggunakan alfabet. Semisal untuk kata sandi yang dibatasi oleh huruf kecil dan berukuran 5, maka jumlah kata sandi yang dapat dibentuk menggunakan *brute force* adalah 26^5 kata sandi.

$$\underline{\quad} \; \underline{\quad} \; \underline{\quad} \; \underline{\quad} \; \underline{\quad} = 26 \quad 26 \quad 26 \quad 26 \quad 26 = 26^5$$

Gambar 3.1 Ilustrasi mencari jumlah kemungkinan kata sandi
Sumber: Dokumen pribadi

Dapat dilihat bahwa algoritma *brute force* tidak optimal untuk memecahkan kata sandi yang panjang dan dengan variasi simbol yang besar, contoh pada gambar 3.1 merupakan banyaknya kata sandi yang harus dibuat untuk memecahkan kata sandi berukuran 5 karakter dan hanya terdiri dari huruf kecil, tetapi dalam dunia nyata panjang serta variasi dari kata sandi tidak akan diketahui sebab kata sandi yang disimpan dalam *database* berbentuk *hash*. Untuk itu, algoritma *brute force* akan mencoba menghasilkan semua kata sandi yang mungkin mulai dari ukuran 1 karakter sampai N karakter dimana N dalam makalah ini dibatasi berukuran 6 karakter. Kata sandi yang dihasilkan oleh *brute force* akan mengasumsikan *worst case* yakni kata sandi tujuan memiliki ukuran variasi simbol yang terbesar (huruf kecil, huruf besar, angka, dan karakter istimewa). Untuk kata sandi berukuran L dengan variasi simbol N dimana N adalah 94 (*N_full_pattern*) maka jumlah solusi (M) dalam skenario *worst case* yang perlu dihasilkan oleh algoritma *brute force* adalah

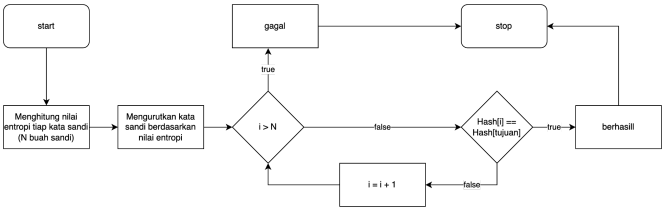
$$l = 1, m_1 = 94$$
$$l = 2, m_2 = 94^2$$
$$l = 3, m_3 = 94^3$$
$$\dots$$
$$l = L, m_L = 94^L$$
$$M = m_1 + m_2 + m_3 \dots + m_L$$
$$M = 94 + 94^2 + 94^3 + \dots + 94^L = \frac{94(94^L - 1)}{93}$$

Ketahui bahwa tidak semua kemungkinan yang dihasilkan oleh algoritma *brute force* dipakai pengguna, pengguna cenderung menggunakan kata sandi yang mudah diingat, sangat jarang pengguna menggunakan kata sandi kompleks seperti "LKLojhfaf@SF" yang merupakan salah satu contoh kata sandi yang dihasilkan oleh algoritma *brute force*, sehingga diperlukan cara lain untuk mencari kata sandi, cara lain tersebut adalah *dictionary attack* yang akan dikembangkan menggunakan algoritma *greedy* dan algoritma heuristik.

B. *Pemecahan Kata Sandi Dengan Greedy*

Seperti sudah dibahas sebelumnya, pengguna cenderung menggunakan kata sandi yang mudah diingat, kumpulan kata sandi tersebut dapat disimpan ke dalam sebuah kamus yang kemudian akan digunakan untuk mencoba meretas kata sandi lain, proses ini dinamakan dengan *dictionary attack*, tetapi apa yang terjadi jika kamus yang digunakan itu berukuran sangat besar ? Tidak jarang kamus yang digunakan untuk melakukan *dictionary attack* berukuran 10 GB, bahkan terdapat kamus berukuran 60 GB seperti kamus yang disediakan pada web https://wiki.skullsecurity.org/index.php/Passwords. Diperlukan sebuah cara untuk menelusuri kamus dengan efektif, cara tersebut adalah dengan menggunakan algoritma *greedy*. Algoritma *greedy* berfungsi untuk menelusuri kamus pada *dictionary attack* dengan cara rakus dengan menargetkan kata sandi pada kamus yang yang memiliki nilai entropi yang paling terbesar, algoritma ini mengasumsikan bahwa kata sandi yang paling ketat merupakan kata sandi yang paling dekat dengan kata sandi tujuan, hal ini berdasarkan asumsi bahwa kata sandi yang dibuat pengguna ketika membuat sebuah akun harus kuat seperti minimal terdiri dari 8 karakter, 1 karakter kecil, 1 karakter besar, 1 angka, 1 karakter spesial. Langkah - langkah algoritma *greedy* adalah sebagai berikut:

1. Mencari nilai entropi setiap kata sandi dalam kamus yang digunakan.
2. Mengurutkan kata sandi berdasarkan nilai entropinya mulai dari yang terbesar sampai terkecil.
3. Melakukan komparasi antara hasil *hash* kata sandi dengan nilai entropi terbesar (solusi terbaik pada langkah saat ini) dengan *hash* kata sandi tujuan.
4. Jika kata sandi sesuai dengan kata sandi tujuan, maka algoritma diberhentikan.
5. Jika tidak sesuai dengan kata sandi tujuan, maka akan dilakukan komparasi menggunakan kata sandi terbaik berikutnya (kata sandi dengan nilai entropi terbesar berikutnya).
6. Langkah 3 - 5 diulang sampai semua kata sandi dalam kamus sudah ditelusuri.
7. Jika semua kata sandi sudah ditelusuri dan tidak ditemukan kata sandi tujuan maka algoritma gagal.



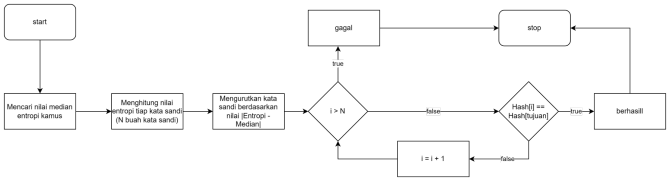
Gambar 3.2 Langkah - langkah algoritma greedy
Sumber: Dokumen pribadi

C. *Pemecahan Kata Sandi dengan Algoritma Heuristik*

Salah satu cara lain untuk memecahkan kata sandi adalah dengan pencarian berbasis heuristik. *Rule of thumb* yang akan digunakan adalah "kata sandi dengan nilai entropi paling umum (*closest to median*) adalah kata sandi yang paling dekat dengan kata sandi tujuan". Ketahui bahwa mustahil untuk mendapatkan nilai median kata sandi dari pengguna secara global sehingga untuk mendapat aproksimasi nilai median dapat melakukan analisis pada kamus yang digunakan. Penulis menggunakan kamus *rockyou.txt* yang tersedia pada https://github.com/praetorian-inc/Hob0Rules/tree/master. Nilai median akan digunakan untuk mengaproksimasi jarak tiap kata sandi dalam kamus dengan kata sandi tujuan. Asumsi disini adalah kata sandi yang dibuat pengguna secara umum memiliki nilai entropi yang umum juga. Berikut adalah langkah - langkah algoritma heuristik:

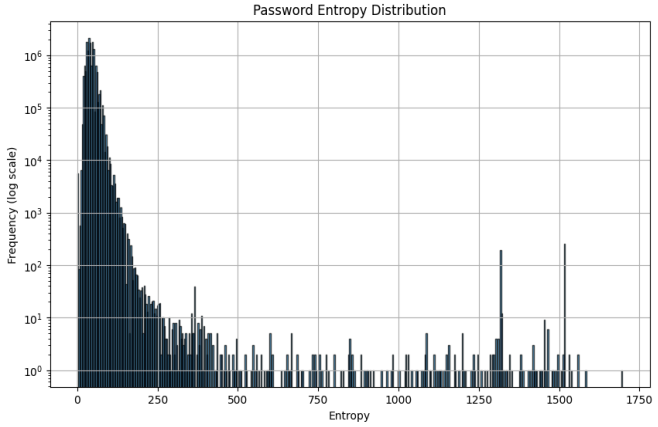
1. Mencari median nilai entropi berdasarkan kamus yang digunakan.
2. Mencari nilai entropi setiap kata sandi.
3. Mencari nilai heuristik setiap kata sandi sesuai dengan rumus *heuristic value* = | *entropy* - *median*|.
4. Mengkomparasi hasil *hash* kata sandi dengan nilai entropi paling umum dengan *hash* kata sandi tujuan.
5. Jika kata sandi sesuai dengan kata sandi tujuan, maka algoritma berhenti.
6. Jika tidak sesuai dengan kata sandi tujuan, maka akan melakukan komparasi dengan kata sandi berikutnya.

7. Langkah 4 - 6 diulang sampai semua kata sandi dalam kamus sudah ditelusuri.
8. Jika semua kata sandi sudah ditelusuri dan tidak ditemukan kata sandi tujuan maka algoritma gagal.



Gambar 3.3 Langkah - langkah algoritma heuristik
Sumber: Dokumen pribadi

Berdasarkan kamus *rockyou.txt* yang digunakan penulis, didapat distribusi nilai *entropi* sebagai berikut:



Gambar 3.4 Diagram distribusi entropi kata sandi dalam kamus rockyou.txt
Sumber: Dokumen pribadi

Berdasarkan hasil distribusi nilai entropi, didapat nilai median 41.3594 yang penulis akan bulatkan menjadi 41.36. Nilai median tersebut akan digunakan saat implementasi algoritma heuristik dan algoritma *hybrid*.

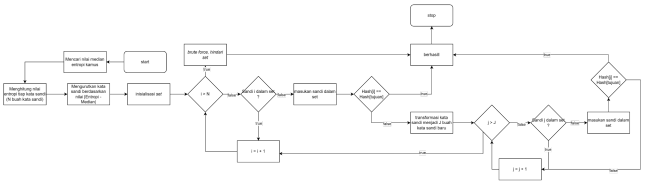
D. Ekspresi Reguler dalam Menghitung Nilai Entropi

Nilai entropi sebuah kata sandi secara dasar bergantung kepada variasi simbol pada kata sandi tersebut. Jenis simbol yang dapat terdapat pada kata sandi adalah huruf kecil, huruf besar, angka, dan karakter spesial sesuai dengan tabel 2.1. Untuk mendeteksi kehadiran simbol - simbol ini dapat melakukan pengecekan tiap karakter secara manual menggunakan *for loop* atau cara lain yang lebih efektif adalah dengan menggunakan ekspresi reguler. Dengan menggunakan ekspresi reguler, dapat mempermudah pengecekan kehadiran sebuah jenis simbol tertentu pada kata sandi.

E. Algoritma Hybrid

Diketahui bahwa algoritma *brute force* adalah metode yang pasti akan berhasil dalam meretas kata sandi, tetapi membutuhkan waktu komputasi yang sangat lama. Sebaliknya, algoritma *greedy* dan heuristik mampu meretas kata sandi dengan cepat, namun tidak selalu berhasil menemukan kata sandi yang benar. Oleh karena itu, diperlukan sebuah algoritma yang dapat menggabungkan kelebihan dari *brute force, greedy*, dan heuristik, sambil mengurangi kekurangan dari masing-masing algoritma tersebut. Algoritma tersebut adalah algoritma *hybrid*, yang menggunakan pendekatan brute force sekaligus mengurangi ruang solusi dengan menggunakan pendekatan heuristik. Algoritma ini juga menggunakan transformasi kata sandi dengan ekspresi reguler. Dalam dunia nyata, sebagian besar karakter pada kata sandi memiliki substitusi karakter lain[11]. Dengan menggunakan *common substitution* tersebut, algoritma dapat menghasilkan kata sandi yang lebih variasi dibandingkan dengan menggunakan kamus saja. Jika semua kata pada kamus dan hasil transformasi sudah ditelusuri tetapi tidak ada yang sesuai dengan kata sandi tujuan, maka akan digunakan algoritma *brute force* tetapi menghindari komparasi kata sandi yang sudah dilakukan sebelumnya. Langkah - langkah algoritma *hybrid* adalah sebagai berikut:

1. Mencari median nilai entropi berdasarkan kamus yang digunakan.
2. Mencari nilai entropi setiap kata sandi.
3. Mencari nilai heuristik setiap kata sandi sesuai dengan rumus *heuristic value* = | *entropy* - *median*|.
4. Menginisialisasi struktur data *set* yang akan menyimpan seluruh kata sandi yang sudah ditelusuri sebelumnya (termasuk transformasi kata sandi).
5. Melakukan pengecekan pada kata sandi dengan nilai heuristik terendah (paling dekat dengan median) saat itu, yakni mengecek apakah kata sandi tersebut sudah ditelusuri sebelumnya.
6. Jika sudah, kata sandi tersebut diabaikan dan lanjut ke kata sandi berikutnya.
7. Jika belum, melakukan komparasi hasil *hash* kata sandi tersebut dengan *hash* kata sandi tujuan.
8. Jika sama, algoritma diberhentikan.
9. Jika tidak, melakukan transformasi kata sandi tersebut menggunakan regex sehingga menghasilkan *list of transformed passwords.*
10. Mengiterasi pada setiap kata sandi yang sudah di transformasi dan mengecek satu per satu hasil *hash* nya dengan *hash* kata sandi tujuan.
11. Jika ada yang sama, algoritma berhenti.
12. Jika tidak ada, proses lanjut dengan kata sandi berikutnya.
13. Kata sandi hasil transformasi juga disimpan pada *set* untuk menghindari komparasi berulang.
14. Ulangi langkah 5 - 13 sampai seluruh kata sandi dalam kamus sudah ditelusuri.
15. Jika tidak ada kata sandi yang sesuai dengan kata sandi tujuan, lanjut dengan metode *brute force*.
16. Metode *brute force* dilakukan dengan memperhatikan *set* yang ada untuk menghindari komparasi berulang, hal ini mengurangi ruang solusi pada metode *brute force*.



Gambar 3.5 Langkah - langkah algoritma hybrid
Sumber: Dokumen pribadi

F. Ekspresi Reguler dalam Transformasi Kata Sandi

Pada algoritma *hybrid*, dilakukan penelusuran bukan hanya pada kata sandi dalam kamus, tetapi pada hasil transformasi setiap kata sandi tersebut. Hasil transformasi kata sandi dilakukan menggunakan ekspresi reguler. Ekspresi reguler digunakan untuk mendeteksi karakter tertentu dan menggantikan karakter tersebut dengan substitusi yang sesuai.

IV. IMPLEMENTASI

A. Implementasi Algoritma Brute Force

Implementasi *brute force* dilakukan melalui fungsi yang dapat memberikan setiap kombinasi kata sandi yang mungkin dihasilkan dengan sebuah *character_set* (karakter - karakter yang mungkin muncul pada sebuah kata sandi) serta pada pada panjang *max_length*.

```
def possible_combinations(characters, max_length):
    if max_length == 0:
        yield ''
    else:
        for char in characters:
            for smaller_combination in possible_combinations(characters, max_length - 1):
                yield char + smaller_combination
```

Dengan fungsi tersebut, *brute force* kemudian akan melakukan iterasi mulai dari *length = 1* sampai *length = 6*, alasan *length* dibatasi sampai 6 karena untuk kata sandi yang berukuran lebih dari 6, dalam skenario terburuk dibutuhkan ratusan hari (asumsikan setiap detik dapat melakukan 1 juta perbandingan).

```
def brute_force_password_cracker(hashed_target_password : string) -> Tuple[str, float]:

    print("Searching using brute force algorithm..")
    start_time = time.time()
    # set maximum length of password to 5, for 6 > passwords it will take days to compute
    for length in range(1, 6 + 1):
        for password_tuple in possible_combinations(CHARACTERS, length):
            password = ''.join(password_tuple)
            hashed_password = hash_password(password)
            if hashed_password == hashed_target_password:
                end_time = time.time()
                elapsed_time_seconds = end_time - start_time
                return password,elapsed_time_seconds


    end_time = time.time()
    elapsed_time_seconds = end_time - start_time
    return None, elapsed_time_seconds
```

B. Implementasi Algoritma Greedy

Algoritma *greedy* diimplementasikan menggunakan fungsi yang menghitung nilai entropi dari setiap kata sandi menggunakan ekspresi reguler.

```
def entropy(password: str) -> float:
    score = 0

    # Entropy calculation
    charset_size = 0
    if re.search(r"[A-Z]", password):
        charset_size += 26  # Uppercase letters
    if re.search(r"[a-z]", password):
        charset_size += 26  # Lowercase letters
    if re.search(r"\d", password):
        charset_size += 10  # Digits
    if re.search(r"[!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~]", password):
        charset_size += len("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")  # Special characters

    if charset_size > 0:
        entropy = len(password) * math.log2(charset_size)
        score += entropy

    return round(score, 4)
```

Setelah mendapatkan nilai entropi dari setiap kata sandi, maka dilakukan pengurutan pada setiap kata sandi sesuai dengan nilai entropinya (pengurutan menurun), lalu melakukan perbandingan dengan kata sandi yang memiliki nilai entropi tertinggi saat itu sampai kata sandi yang sesuai berhasil didapatkan atau seluruh kata sandi dalam kamus sudah ditelusuri.

```
def greedy_password_cracker(hashed_target_password: str) -> Tuple[str, float]:

    print("Searching using greedy algorithm..")
    start_time = time.time()
    scored_passwords =  [(password, entropy(password)) for password in PASSWORD_DICTIONARY]


    # Sort passwords based on their heuristic score
    # Strong passwords have high score, weak passwords have low scores
    scored_passwords.sort(key=lambda x: x[1], reverse=True)


    for password, _ in scored_passwords:
        hashed_password = hash_password(password)

        # Greedy by comparing target password with current strongest password first
        # If not a match, then try comparing the next strongest password
        if hashed_password == hashed_target_password:
            end_time = time.time()
            elapsed_time_seconds = end_time - start_time

            return password, elapsed_time_seconds

    end_time = time.time()
    elapsed_time_seconds = end_time - start_time

    return None, elapsed_time_seconds
```

C. Implementasi Algoritma Heuristik

Algoritma heuristik menggunakan fungsi *heuristic* yang menghitung nilai heuristik setiap kata sandi yang ada dalam kamus.

```
def heuristic(password: str) -> float:
    score = 0
    median = 41.35
    score = entropy(password)
    return abs(score - median)
```

Berdasarkan nilai heuristik tersebut, kata sandi akan diurutkan (secara menaik), kemudian akan diiterasi satu per satu sampai ditemukan kata sandi yang sesuai

```
def heuristic_password_cracker(hashed_target_password: str) -> Tuple[str, float]:

    print("Searching using heuristic algorithm..")
    start_time = time.time()
```

```
    scored_passwords =  [(password, heuristic(password)) for password in
PASSWORD_DICTIONARY]

    # Sort passwords based on their heuristic score
    # Common passwords have scores close to 0
    scored_passwords.sort(key=lambda x: x[1], reverse=False)
    for password, _ in scored_passwords:
        hashed_password = hash_password(password)
        if hashed_password == hashed_target_password:
            end_time = time.time()
            elapsed_time_seconds = end_time - start_time
            return password, elapsed_time_seconds

    end_time = time.time()
    elapsed_time_seconds = end_time - start_time

    return None, elapsed_time_seconds
```

D. Implementasi Algoritma Hybrid

Algoritma *hybrid* menggunakan gabungan antara algoritma *brute force* dan heuristik untuk membuat algoritma memiliki keuntungan kedua algoritma dengan mengurangi kekurangannya. Alasan algoritma *greedy* tidak diimplementasikan dalam algoritma ini karena sedikit redundan sebab algoritma heuristik dan *greedy* memiliki konsep yang mirip. Dalam algoritma ini juga terdapat fungsi transformasi untuk menghasilkan kata sandi yang lebih bervariasi, transformasi ini diimplementasikan berdasarkan *common substitution*[11], tetapi hanya sebagian substitusi yang diimplementasikan untuk mengurangi kompleksitas program.

```
def apply_transformations(word):

    transformations = set()

    # Common substitutions
    substitutions = [
        (r"a", "@"),
        (r"e", "3"),
        (r"i", "1"),
        (r"o", "0"),
        (r"s", "$"),
        (r"g", "9"),
    ]

    for pattern, replacement in substitutions:
        transformed_word = re.sub(pattern, replacement, word)
        transformations.add(transformed_word)

    return list(transformations)
```

Dengan fungsi tersebut, algoritma *hybrid* akan menjalankan prosesnya sesuai dengan penjelasan pada bab III.

```
def hybrid_password_cracker(hashed_target_password: str) -> Tuple[str, float]:

    print("Searching using hybrid algorithm..")
    start_time = time.time()
    all_encountered_password = set()
    scored_passwords = load_heuristic_password()

    for password_not_altered, _ in scored_passwords:

        if (password_not_altered not in all_encountered_password):
            all_encountered_password.add(password_not_altered)
            hashed_password = hash_password(password_not_altered)

            if hashed_password == hashed_target_password:
                end_time = time.time()
                elapsed_time_seconds = end_time - start_time
                return password_not_altered,elapsed_time_seconds
        else:
            continue

        for password in apply_transformations(password_not_altered):

            if (password not in all_encountered_password):
                all_encountered_password.add(password)
            else:
                continue

            hashed_password = hash_password(password)
            if hashed_password == hashed_target_password:
                end_time = time.time()
                elapsed_time_seconds = end_time - start_time
                return password, elapsed_time_seconds

    print("Password not found in dictionary. Falling back to brute force")

    for length in range(1, 6 + 1):
        for password_tuple in itertools.product(CHARACTERS, repeat=length):
            if password not in all_encountered_password:
                password = ''.join(password_tuple)
                hashed_password = hash_password(password)
                if hashed_password == hashed_target_password:
                    end_time = time.time()
                    elapsed_time_seconds = end_time - start_time
                    return password,elapsed_time_seconds

    end_time = time.time()
    elapsed_time_seconds = end_time - start_time

    return None,elapsed_time_seconds
```

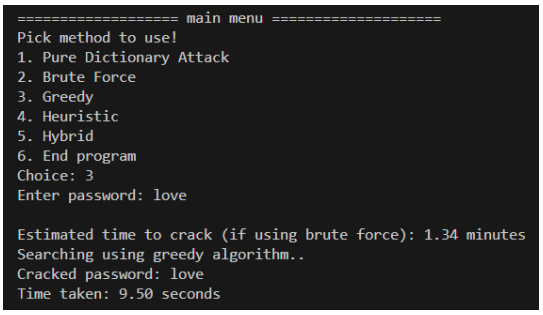
V. HASIL UJI COBA

Penulis telah melakukan uji coba terhadap berbagai algoritma, termasuk *brute force*, *greedy*, heuristik, dan *hybrid*. Sebagai tambahan, penulis juga melakukan perbandingan dengan metode *dictionary attack*. Tujuan dari perbandingan ini adalah untuk menilai efektivitas algoritma *greedy*, heuristik, dan *hybrid* dibandingkan hanya menggunakan *dictionary attack*. Untuk mempermudah uji coba, penulis melakukan dua macam percobaan yakni percobaan dengan kata sandi berukuran 4 karakter dan percobaan dengan kata sandi berukuran lebih dari 7 karakter, kedua percobaan ini dilakukan untuk mempermudah uji coba menggunakan algoritma *brute force* sebab untuk kata sandi berukuran lebih dari 5 karakter membutuh kan berjam - jam untuk diselesaikan dengan algoritma *brute force*. Untuk percobaan dengan ukuran kata sandi lebih dari 7 karakter akan dilakukan tanpa menggunakan algoritma *brute force*.

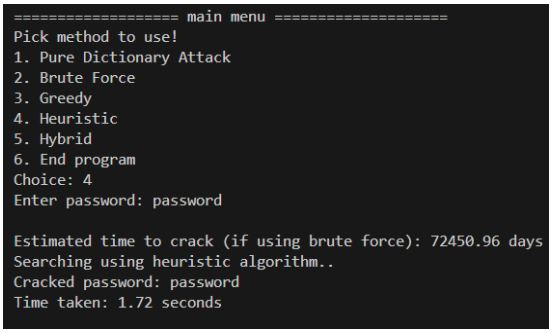
```
================== main menu ====================
Pick method to use!
1. Pure Dictionary Attack
2. Brute Force
3. Greedy
4. Heuristic
5. Hybrid
6. End program
Choice: 2
Enter password: P@sS

Estimated time to crack (if using brute force): 1.34 minutes
Searching using brute force algorithm..
Cracked password: P@sS
Time taken: 1.24 minutes
```

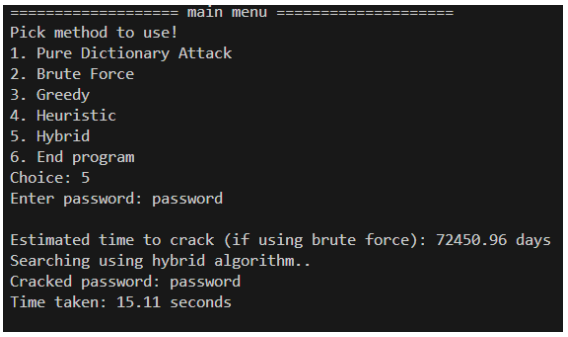
Gambar 4.1 Percobaan dengan brute force
Sumber: Dokumen pribadi



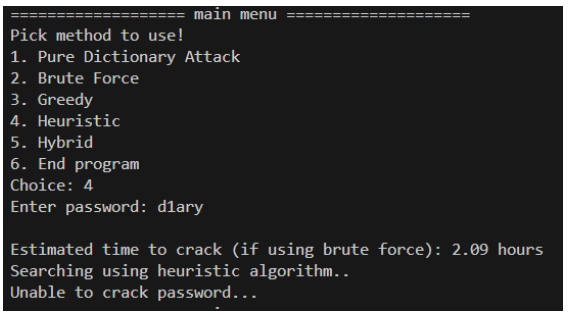
Gambar 4.2 Percobaan dengan greedy
Sumber: Dokumen pribadi



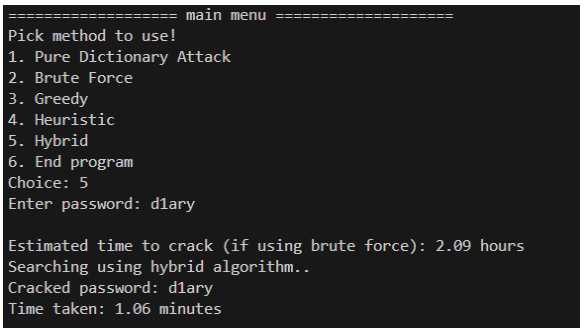
Gambar 4.3 Percobaan dengan heuristik
Sumber: Dokumen pribadi



Gambar 4.4 Percobaan dengan hybrid
Sumber: Dokumen pribadi



Gambar 4.5 Percobaan dengan heuristik yang gagal
Sumber: Dokumen pribadi



Gambar 4.6 Percobaan hybrid yang berhasil walaupun kata tidak ada dalam kamus
Sumber: Dokumen pribadi

Percobaan pertama adalah percobaan menggunakan kata sandi berukuran 4 karakter untuk menguji kemampuan algoritma lain dibandingkan dengan algoritma *brute force*, kata sandi yang penulis pilih diusahakan se-acak mungkin, penulis mendapatkan kata sandi yang diujikan dari *website* - *website* beragam. Terdapat 25 kata sandi yang penulis ujikan, berikut adalah hasil percobaan pertama:

Tabel 4.1 Hasil percobaan dengan kata sandi berukuran 4 karakter

Metode	**Waktu Total**	**Banyaknya kata sandi yang berhasil diretas**	***Rata - rata proses***
Dictionary Attack	63.11 detik	18	3.51 detik
Brute Force	1351.75 detik	25	54.07 detik
Greedy	173.10 detik	18	9.62 detik
Heuristic	154.64 detik	18	8.59 detik
Hybrid	1140.67 detik	25	45.63 detik

Berdasarkan percobaan pertama, didapatkan bahwa *dictionary attack* memiliki rata-rata waktu proses terendah, sedangkan *brute force* memiliki rata-rata waktu proses terbesar. Untuk percobaan pertama ini, *dictionary attack* lebih cepat dibandingkan dengan metode *greedy* dan heuristik. Hal ini dikarenakan kata sandi yang berukuran 4 karakter secara umum memiliki nilai entropi yang rendah dan nilai heuristik yang tinggi (nilai entropi yang jauh dari median nilai entropi), sehingga kata sandi tersebut cenderung ditelusuri pada akhir metode *greedy* dan heuristik. Ditemukan juga bahwa algoritma *brute force* dan *hybrid* berhasil meretas seluruh sandi yang diujikan. Algoritma *hybrid* memiliki waktu rata-rata proses yang lebih lambat dibandingkan algoritma *greedy*, heuristik, dan *dictionary attack* karena algoritma *hybrid* tetap mencoba meretas kata sandi tujuan meskipun seluruh kata dalam kamus sudah ditelusuri. Perbedaan waktu antara algoritma *hybrid* dan *brute force* terlihat sedikit, tetapi ini dikarenakan algoritma *hybrid* yang harus menelusuri seluruh kamus terlebih dahulu sebelum lanjut dalam metode *brute force* sehingga algoritma *hybrid* memiliki waktu tambahan dibandingkan dengan *brute force*. Namun, secara keseluruhan, semakin besar kamus yang digunakan dan semakin besar kata sandi yang ingin diretas maka algoritma *hybrid* akan jauh lebih efektif dibandingkan dengan *brute force*.

Pada percobaan kedua, penulis menggunakan kata sandi berukuran 8 karakter atau lebih untuk menguji kemampuan algoritma *greedy*, heuristik, dan *hybrid* terhadap serangan *dictionary attack*. Perlu diketahui bahwa dalam proses percobaan, penulis menemukan banyak situasi di mana kata sandi tidak bisa diretas menggunakan kamus yang digunakan. Dalam situasi tersebut, algoritma *hybrid* tentu akan mencoba menggunakan metode *brute force*. Namun, karena ukuran sandi yang sangat besar, waktu untuk meretas kata sandi tersebut dapat mencapai harian bahkan tahunan.

Dengan masalah tersebut, penulis memutuskan melakukan percobaan menggunakan kata-kata yang terdapat pada kamus atau kata-kata yang dapat diselesaikan oleh algoritma *hybrid* dengan transformasi kata. Percobaan tetap dilakukan secara acak dengan kata sandi yang diuji diambil dari berbagai website. Berikut adalah hasil percobaan kedua:

Tabel 4.2 Hasil percobaan dengan kata sandi berukuran 8 karakter atau lebih

Metode	**Waktu Total**	**Banyaknya kata sandi yang berhasil diretas**	***Rata - rata proses***
Dictionary Attack	106.17 detik	18	5.90 detik
Greedy	87.59 detik	18	4.87 detik
Heuristic	28.19 detik	18	1.57 detik
Hybrid	190.34 detik	25	7.61 detik

Berdasarkan percobaan kedua, dapat disimpulkan bahwa algoritma heuristik dan *greedy* memiliki waktu proses yang lebih pendek dibandingkan dengan serangan kamus (*dictionary attack*). Hal ini dikarenakan kata sandi dengan ukuran 8 karakter atau lebih umumnya memiliki nilai entropi yang tinggi serta nilai heuristik yang rendah (nilai entropi kata sandi mendekati median nilai entropi) sehingga kata sandi tersebut akan ditelusuri terlebih dahulu oleh algoritma *greedy* dan heuristik. Dari percobaan ini dapat dilihat bahwa metode heuristik merupakan metode yang lebih optimal dibandingkan dengan metode *greedy*. Hal ini disebabkan sebagian besar pengguna tidak membuat kata sandi dengan entropi yang terlalu besar, sehingga nilai median entropi merupakan nilai terbaik dalam menentukan kata sandi mana yang harus ditelusuri terlebih dahulu.

Algoritma *hybrid* memiliki waktu proses terbesar di antara keempat algoritma tersebut, tetapi algoritma *hybrid* berhasil meretas seluruh kata sandi yang diujikan. Berdasarkan hasil percobaan 1 dan 2, dapat disimpulkan bahwa algoritma *hybrid* menggabungkan keuntungan dari algoritma *brute force* dan heuristik sambil mengurangi kelemahan dari kedua algoritma tersebut. Walaupun algoritma *hybrid* tidak secepat serangan kamus biasa, kemampuannya untuk meretas kata sandi meskipun sandi tersebut tidak ada di kamus merupakan *trade-off* yang layak dipertimbangkan.

Catatan : untuk percobaan 1 dan 2, dapat diulangi dengan menjalankan program test.py pada *repository*.

VI. KESIMPULAN

Pemecahan kata sandi merupakan tantangan yang signifikan dalam bidang keamanan siber. Algoritma *brute force*, meskipun pasti berhasil menemukan kata sandi yang benar, membutuhkan waktu komputasi yang sangat lama. Sebaliknya, algoritma *greedy*, heuristik, dan serangan kamus (*dictionary attack*) memiliki waktu pemrosesan yang lebih cepat tetapi tidak selalu berhasil menemukan kata sandi yang benar. Oleh karena itu, dikembangkan algoritma *hybrid* yang menggabungkan keuntungan dari algoritma *brute force* dan algoritma heuristik, sekaligus mengurangi kekurangan dari masing-masing algoritma tersebut.

Melalui serangkaian percobaan, terbukti bahwa algoritma *hybrid* mampu menjadi solusi yang layak untuk pemecahan kata sandi. Algoritma ini efektif dalam mengurangi ruang solusi, sehingga meningkatkan efisiensi dan kecepatan dalam menemukan kata sandi yang benar. Dengan demikian, algoritma *hybrid* menawarkan pendekatan yang lebih seimbang antara kepastian hasil dan waktu komputasi yang dibutuhkan. Diharapkan penerapan algoritma *hybrid* dapat memberi sedikit kontribusi dalam peningkatan keamanan siber, dengan menyediakan metode yang lebih efisien untuk pemecahan kata sandi.

VII. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada berbagai pihak yang telah memberikan dukungan dan bantuan selama proses penulisan makalah ini. Pertama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena atas nikmat dan rahmat-Nya, penulis dapat menyelesaikan makalah berjudul “Analisis dan Pengembangan Algoritma Brute Force, Greedy, dan Heuristik Berbasis Regex untuk Pemecahan Kata Sandi” dengan baik.

Selain itu, penulis juga ingin menyampaikan terima kasih kepada dosen pembimbing, Bapak Dr. Ir. Rinaldi Munir, M.T., yang telah memberikan bimbingan dan arahan selama penulis mengikuti mata kuliah Strategi Algoritma. Bimbingan beliau sangat membantu dalam memahami konsep-konsep yang penting dan relevan untuk penyusunan makalah ini.

Penulis juga mengucapkan terima kasih kepada keluarga dan teman-teman yang selalu memberikan dukungan moral dan motivasi. Tak lupa, penulis menyampaikan apresiasi kepada semua pihak yang telah menyediakan sumber referensi yang digunakan dalam makalah ini. Sumber-sumber ini sangat berharga dalam menyusun argumen dan memperkaya isi makalah.

LINK VIDEO YOUTUBE

https://youtu.be/yarhrrqpUjM

LINK REPOSITORY

https://github.com/MarvelPangondian/Simple-Password-Cracker

REFERENSI

[1] https://www.sciencedirect.com/topics/computer-science/weak-password diakses pada 8 Juni 2024

[2] https://www.sailpoint.com/identity-library/two-factor-authentication/ diakses pada 8 juni 2024

[3] https://www.ibm.com/topics/multi-factor-authentication diakses pada 8 Juni 2024

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir//Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf diakses pada 8 Juni 2024

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir//Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf diakses pada 8 Juni 2024

[6] https://www.sciencedirect.com/topics/computer-science/regular-expression diakses pada 8 Juni 2024

[7] C. N. K. Babu, M. Rajendiran and B. S. Ibrahim, "Random password entropy and crack time calculation for South Asian languages," 2014 International Conference on Science Engineering and Management Research (ICSEMR), Chennai, India, 2014, pp. 1-4, doi: 10.1109/ICSEMR.2014.7043593.

[8] *https://www.wallarm.com/what/dictionary-attack* diakses pada 9 Juni 2024

[9] https://www.okta.com/blog/2019/03/what-are-salted-passwords-and-password-hashing/ diakses pada 9 Juni 2024

[10] H. Fu, J. Liu, Z. Han and Z. Shao, "A Heuristic Task Periods Selection Algorithm for Real-Time Control Systems on a Multi-Core Processor," in IEEE Access, vol. 5, pp. 24819-24829, 2017, doi: 10.1109/ACCESS.2017.2768559.

[11] https://cloud.google.com/solutions/modern-password-security-for-users.pdf diakses pada 10 Juni 2024

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Marvel Pangondian
13522075