Projektarbeit Brute Force

Lennard Bernet, Lino Steiner, Luca Güttinger

Dipl. Informatiker Applikationsentwicklung

TEKO Zürich

Lösungsalgorithmen, Stefan Kessler

Inhaltsverzeichnis

Selbstständigkeitserklärung

Wir bestätigen hiermit, dass wir die vorstehende Projektarbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und sowohl wörtliche als auch sinngemäss verwendete Textteile, Grafiken oder Bilder kenntlich gemacht haben. Diese Arbeit ist in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt worden.

Ort, Datum Unterschrift/en

Oberglatt, 03.03.2025,
Lennard Bernet

Baltenswil, 03.03.2025,
Lino Steiner

Dorf, 03.03.2025,
Luca Güttinger

Initialisierung und Planung

Auftrag des Dozenten

Im Rahmen eines Unternehmensprojekts wurden kurze Schlüsselwörter aus Gross- und Kleinbuchstaben ohne Umlaute generiert. Diese wurden versehentlich als kryptografische MD5-Hashes in einer Datenbank gespeichert. Da die MD5-Hash-Funktion eine Einwegfunktion ist, können die ursprünglichen Werte nicht direkt rekonstruiert werden.

Die Aufgabe besteht darin, ein Tool zu entwickeln, das mittels der Brute-Force-Methode die ursprünglichen Schlüsselwörter aus den Hash-Werten ermittelt. Das Programm soll systematisch alle möglichen Kombinationen von Zeichen testen, bis das gesuchte Schlüsselwort gefunden ist.

Das Tool dient dazu, durch eine systematische Suche verlorene oder vergessene Schlüsselwörter zu rekonstruieren, zeigt aber auch gleichzeitig die Schwächen von MD5 als Hash-Funktion auf.

Funktionsweise des Tools

- Eingabe eines MD5-Hash-Werts.
- Festlegen der Schlüsselwortlänge (maximal 8 Zeichen).
- Auswahl, ob nur Kleinbuchstaben oder sowohl Klein- als auch Grossbuchstaben verwendet werden.
- Ausgabe der Anzahl der möglichen Kombinationen basierend auf der Eingabe.
- Generierung aller möglichen Kombinationen, Hashing mit MD5 und Vergleich mit der Eingabe.
- Falls ein Treffer gefunden wird, wird das ursprüngliche Schlüsselwort ausgegeben.
- Falls nach einem vollständigen Brute-Force-Durchlauf kein Schlüssel gefunden wird, erscheint eine entsprechende Nachricht.

MD5 Hash

MD5 steht für «Message-Digest Algorithm 5» und ist eine kryptografische Hash-Funktion. Diese Hashfunktion wurde ursprünglich 1991 von Ronald Rivest entwickelt. Es wird aus einer beliebigen Eingabe einen fixen 128-Bit-Hash-Wert erstellt. Dies entspricht auch einem 32-stelligen Hexadezimalzahl. Als Eingabewert kann entweder eine Zeichenkette von ein paar Charakteren oder ein ganzes File verwendet werden. Schlussendlich ist für jeden Input der Output im gleichen Format. [1]

Eigenschaften

Ein wichtiger Bestandteil dieser Hash-Funktion ist, dass der erstelle Hash-Wert nicht umkehrbar ist. Aus dem Hash-Wert kann also nicht wieder der Eingabewert gefunden werden. Diese Hash-Werte werden dadurch auch Signaturen oder eben Message Digests genannt.

Ein MD5-Hash hat einige eindeutige Eigenschaften. Diese wären:

- **Deterministisch**: Die gleiche Eingabe liefert immer denselben Hash-Wert.
- **Feste Länge**: Unabhängig von der Eingabegrösse hat das Ergebnis immer 128 Bit (32 Zeichen in hexadezimaler Darstellung).
- **Schnelle Berechnung**: MD5 kann sehr effizient berechnet werden, was sowohl ein Vorteil als auch eine Schwäche ist.
- **Schneeball-Effekt**: Kleine Änderungen in der Eingabe führen zu völlig unterschiedlichen Hash-Werten.

Hier in einer Tabelle einige Beispiele für solche MD5-Hashes:

Eingabe	MD5-Hash
hello	5d41402abc4b2a76b9719d911017c592
Hello	8b1a9953c4611296a827abf8c47804d7
password	5f4dcc3b5aa765d61d8327deb882cf99
123456	e10adc3949ba59abbe56e057f20f883e

Einsatzbereich

Hash-Funktionen haben in der Geschichte einen breiten Einsatzbereich. Sie werden zum Beispiel gebraucht, um Dateibeschädigungen oder unbeabsichtigte Veränderungen bei grossen Dateisammlungen zu erkennen. Dabei prüft man den bestehenden Hashwert der Datei, und prüft den mit dem zweiten Hashwert der angeblich gleichen Datei. Stimmen die beiden Hashwerte nicht überein weiss man, dass sich die Dateien unterscheiden.

[1] (okta, 2024)

Eines der bekanntesten Einsatzbereiche für solche Hashes ist für Passwörter, die auf einer Datenbank abgespeichert werden. Anstatt Passwörter im Klartext zu speichern, werden sie gehasht um sie vor Angriffe zu schützen.

Weiterhin gibt es heutzutage eine ganze Reihe an Beispielen, wo eben genau solche Hashes verwendet werden.

- Digitale Zertifikate: SSL-Zertifikate nutzen Hash-Funktionen, um sichere Verbindungen im Internet zu ermöglichen.
- E-Mail-Sicherheit: Mit Hashing kann die Authentizität von E-Mails geprüft werden.
- Virenscanner: Antivirenprogramme nutzen Hashes, um bekannte Schadsoftware zu identifizieren.
- Digitale Wasserzeichen: Hash-Funktionen schützen digitale Medieninhalte.

Diese vielfältigen Anwendungen zeigen die zentrale Rolle von Hashing-Algorithmen in der modernen IT-Sicherheit und Datenverarbeitung. [2]

Schwächen

Obwohl MD5 ursprünglich für kryptografische Zwecke entwickelt wurde, gilt es heute als unsicher. Die wesentlichen Probleme sind:

- **Kollisionen**: Zwei unterschiedliche Eingaben können denselben MD5-Hash ergeben. Dies wurde erstmals 2004 demonstriert, wodurch MD5 als unsicher für digitale Signaturen oder Passwortspeicherung eingestuft wurde.
- **Brute-Force-Anfälligkeit**: Aufgrund der schnellen Berechnung kann MD5 leicht durch Brute-Force-Angriffe oder Wörterbuchangriffe gebrochen werden.
- **Rainbow-Table-Angriffe**: Es existieren grosse vorgefertigte Tabellen mit Hash-Wert-Zuordnungen für gängige Passwörter, wodurch eine Entschlüsselung oft ohne Brute-Force-Suche möglich ist.

Aufgrund der bekannten Schwachstellen gilt MD5 heute als unsicher für sicherheitskritische Anwendungen wie das Hashing von Passwörtern. Daher werden stattdessen moderne Algorithmen wie SHA-256 oder bcrypt bevorzugt.[3]

[2] (Spasojevic, 2024)
[3] (Spasojevic, 2024)

Zielformulierung

Das Ziel des Projekts ist die Entwicklung eines Brute-Force-Tools, das gehashte MD5 Schlüsselwörter entschlüsseln kann. Dazu müssen die Benutzenden verschiedene Eingaben tätigen. Am Schluss erhalten die Benutzenden Informationen über das Resultat. Dabei soll die Interaktion mit den Benutzenden einfach und nachvollziehbar sein, Eingaben sauber validiert und die Berechnung möglichst effizient sein. Auf die Erstellung einer grafischen Oberfläche (GUI) wird dabei bewusst verzichtet.

Eingabe

Sobald das Programm gestartet wird, werden die Benutzenden aufgefordert, verschiedene Angaben in die Konsole einzugeben. Die erste Eingabe ist der MD5-Hash-Wert, vom Schlüsselwort, dass gesucht werden soll. Zusätzlich wird die gewünschte Zeichenanzahl des gesuchten Schlüsselworts festgelegt. Anschliessend wird angegeben, ob das Schlüsselwort ausschliesslich Kleinbuchstaben oder einer Kombination aus Gross- und Kleinbuchstaben besteht. Die Eingaben der Benutzenden werden dabei anhand sinnvoller Kriterien validiert. Beispielsweise muss für den MD5-Hash eine 32-stellige Zeichenkette aus Hexadezimalzeichen eingegeben werden. Die Zeichenlänge muss eine positive Ganzzahl und maximal 8 sein, und für die Angabe, ob das Quellwort nur Klein- oder zusätzlich auch Grossbuchstaben beinhaltet, muss entweder. «lower» oder «both» eingegeben werden.

Ausgabe

Falls sich ein erfolgreiches Resultat finden lässt, wird dieses unmittelbar auf der Konsole ausgegeben, um die Benutzenden über den entschlüsselten Wert zu informieren. In der Ausgabe wird der Gesuchte Begriff im Klartext sowie die Zeitdauer für die Entschlüsselung angezeigt.

Falls nach einem Durchlauf aller möglichen Kombinationen kein passendes Resultat gefunden werden konnte, wird eine entsprechende Meldung ausgegeben. Dies bedeutet das der eingegebene MD5-Hash zwar syntaktisch korrekt ist, also aus 32 Hexadezimalstellen besteht, aber dennoch nicht valid ist.

Zusätzlich wird dem Benutzer die Gesamtanzahl der möglichen Kombinationen angezeigt. Dies dient als Indikator für die Rechenintensität. Diese Information kann dabei helfen, mögliche Optimierungsansätze in der Brute-Force-Logik zu erkennen, insbesondere im Hinblick auf die Laufzeiteffizienz und die effektive Nutzung von Ressourcen.

Brute-Force-Berechnung

Sobald die Eingaben durch die benutzende Person gemacht wurden, wird mit der Brute-Force-Methode das Ergebnis gesucht. Das Programm soll alle möglichen Kombinationen (kartesisches Produkt) innerhalb der vorgegebenen Zeichenanzahl

systematisch testen. Für jede Kombination wird der entsprechende MD5-Hash berechnet und mit der Eingabe verglichen. Sobald das Ergebnis gefunden wurde, wird dies den Benutzenden auf dem Terminal angezeigt.

Technische Umsetzung

Das Programm wird in der Programmiersprache Python umgesetzt. Dabei wird die **`hashlib`**-Bibliothek zur Berechnung der MD5-Hashes verwendet. Im Modul wird mit der Web-Tigerjython-Umgebung gearbeitet. In dieser Umgebung ist diese Bibliothek jedoch standardmässig nicht vorhanden. Ein Code-Editor oder eine Python-Entwicklungsumgebung (IDE, z. B. PyCharm) oder Tigerjython müssen daher verwendet werden.

Projektablaufplanung

Einführung

Die Entwicklung des Tools zur Entschlüsselung von MD5-Hashes mithilfe von Brute-Force wird in mehreren Phasen durchgeführt, um eine strukturierte Vorgehensweise sicherzustellen. Durch eine iterative Herangehensweise können technische Herausforderungen gezielt angegangen und Effizienzsteigerungen nach und nach umgesetzt werden. Das Ziel ist ein inhaltlich sinnvoll aufgebautes Programm, das eine effiziente und zuverlässige Entschlüsselung von MD5-Hashes ermöglicht.

Analyse der MD5-Hash-Funktion

Zu Beginn des Projekts steht eine umfassende Auseinandersetzung mit der MD5-Hash-Funktion, um deren API für das geplante Brute-Force-Tool zu analysieren. Geeignete Methoden für die Umsetzung des Projekts müssen ausgesucht werden. Um zu vermeiden, dass Funktionalität implementiert wird, die bereits in der Bibliothek zur Verfügung steht, ist es wichtig, dass die Dokumentation genau gelesen wird.

Programmarchitektur

Im Anschluss wird die Programmarchitektur festgelegt: Nutzerinteraktion, Eingabeverwaltung und die Brute-Force-Logik. Ein frühzeitig angelegtes Git-Repository gewährleistet nachvollziehbare Versionsverwaltung und unterstützt die organisierte sowie kollaborative Weiterentwicklung des Codes. Ein Git-Repository wird auch in der Reflexion des Projekts hilfreich sein, um den iterativen Prozess der Entwicklung des Tools besser nachvollziehen zu können.

Umsetzung

Ein zentraler Meilenstein ist die Nutzung von Pythons hashlib-Bibliothek zur MD5-Berechnung. Zur Überprüfung der Machbarkeit wird ein Proof of Concept (PoC) entwickelt, der grundlegende Funktionen wie die Erzeugung und den Vergleich von Hashes testet. Die Erkenntnisse aus dem PoC fliessen direkt in die Optimierung und den Ausbau der Brute-Force-Logik ein.

Die Brute-Force-Implementierung erfordert eine leistungsfähige Generierung sämtlicher Zeichenkombinationen. Es wird die Benutzerinteraktion optimiert: Eine Eingabeprüfung mittels eines regulären Ausdrucks stellt sicher, dass nur valide MD5-Hashes (32-stellige Hexadezimalzeichen) verarbeitet werden, um Fehlerquellen frühzeitig auszuschliessen.

Refactoring und Abschluss

In der anschliessenden Refactoring-Phase wird der Code vereinfacht und standardisiert. Überflüssige Codeteile werden entfernt, Funktionen logisch neu organisiert und die Lesbarkeit durch konsistente Namensgebung verbessert. Dies reduziert Komplexität und erhöht die langfristige Wartbarkeit.

Den Abschluss bildet die Erstellung einer detaillierten Dokumentation, die Funktionsweise, technische Grundlagen und Erweiterungsoptionen erklärt. Dank des iterativen Ansatzes entsteht ein optimiertes Tool, das MD5-Entschlüsselung effektiv und benutzerfreundlich zur Verfügung stellt.

Realisierung

Programmstruktur

Brute-Force Angriffe sind je nach Passwortlänge und Zeichenumfang ein sehr zeitaufwändiges und mühsames vorgehen. Man kann zum Beispiel Multithreading nutzen, um die Berechnungen zu beschleunigen. Weitere Methoden sind die Kombination aus Brute-Force und Wörterbuchangriffen, wobei in diesem Projekt Wörterbuchangriffe explizit von der Aufgabe ausgeklammert wurden.

Benötigte Importe

Zu Beginn des Programms werden mehrere Bibliotheken importiert, die für die Funktionalität des Programms erforderlich sind.

`hashlib` stellt verschiedene Hash-Funktionen bereit, darunter den MD5-Algorithmus, der zur Berechnung von Hash-Werten genutzt wird. In diesem Programm wird **`hashlib.md5()`** verwendet, um den MD5-Hash eines gegebenen Strings zu berechnen und diesen mit einem zuvor bereitgestellten Hash zu vergleichen.[4]

`itertools` bietet effiziente Iteratoren für Aufgaben mit verschiedenen Kombinationen. Hier wird **`itertools.product()`** genutzt, um alle möglichen Zeichenkombinationen einer bestimmten Länge zu generieren. Diese werden anschliessend gehasht und mit dem gesuchten Hash verglichen.[5]

`re`: Die **`re`**-Bibliothek dient zur Verarbeitung von regulären Ausdrücken und wird in diesem Programm genutzt, um die Benutzereingaben zu validieren.[6]

`string`: Die **`string`**-Bibliothek stellt vordefinierte Zeichensätze bereit, darunter **`string.ascii_lowercase`** für Kleinbuchstaben und **`string.ascii_letters`** für Gross- und Kleinbuchstaben. Diese Zeichen werden genutzt, um die möglichen Passwörter für die Brute-Force-Attacke zu generieren.[7]

`time` wird zur Messung der Programmlaufzeit verwendet. Durch Aufrufe von **`time.time()`** vor und nach dem Brute-Force-Versuch kann die Dauer der Entschlüsselung ermittelt und die Effizienz des Programms bewertet werden.[8]

Hilfsfunktionen

Im Anschluss folgen zwei zentrale Hilfsfunktionen: **`get_character_set(case)`** gibt je nach gewählter Einstellung entweder nur Kleinbuchstaben oder sowohl Klein- als

[4] (Python Software Foundation, 2025)
[5] (Python Software Foundation, 2025)
[6] (Python Software Foundation, 2025)
[7] (Python Software Foundation, 2025)
[8] (Python Software Foundation, 2025)

auch Grossbuchstaben zurück. Diese Zeichenmenge wird später für die Generierung der Schlüsselwort-Kombinationen genutzt.

Die zweite Funktion, **`calculate_md5(text)`**, erzeugt aus einem gegebenen Text den entsprechenden MD5-Hash, indem sie die **`hashlib.md5()`**-Funktion verwendet und das Ergebnis als hexadezimale Zeichenkette zurückgibt.

Kernfunktion

Das Herzstück des Programms ist die Klasse MD5Cracker, die den eigentlichen Brute-Force-Angriff implementiert. Beim Initialisieren (**`__init__`**) speichert die Klasse den Ziel-Hash, die vom Benutzer angegebene Länge des Schlüsselworts sowie den zu verwendenden Zeichensatz. Zusätzlich wird die Anzahl der möglichen Versuche berechnet, indem die Länge des Zeichensatzes mit der angegebenen Wortlänge potenziert wird. Diese Anzahl wird dem Benutzer angezeigt, um eine Vorstellung vom Umfang des Suchraums zu geben.

Die Methode **`crack()`** übernimmt die eigentliche Brute-Force-Suche. Sie generiert alle möglichen Zeichenkombinationen mit der **`itertools.product()`**-Funktion, die alle Variationen für die festgelegte Länge durchläuft. Für jede Kombination wird der MD5-Hash berechnet und mit dem Zielwert verglichen. Falls eine Übereinstimmung gefunden wird, gibt das Programm das gefundene Schlüsselwort aus und beendet die Suche. Falls nach dem Durchprobieren aller Kombinationen kein Treffer erzielt wird, erfolgt eine entsprechende Fehlermeldung.

Benutzereingabe

Die Benutzerinteraktion erfolgt in der **`main()`**-Funktion. Dabei werden die Eingaben durch separate Funktionen verarbeitet:

`get_input_hash()`: Diese Funktion fordert die Benutzenden auf, einen gültigen MD5-Hashwert einzugeben. Die Eingabe wird mit einem regulären Ausdruck überprüft, um sicherzustellen, dass sie genau 32 hexadezimale Zeichen enthält. Falls die Eingabe ungültig ist, werden die Benutzenden erneut zur Eingabe aufgefordert.

`get_length_str()`: Diese Funktion fragt die Länge des ursprünglichen Schlüsselworts ab. Es wird sichergestellt, dass nur positive Ganzzahlen akzeptiert werden die kleiner als 9 sind. Falls eine ungültige Eingabe erfolgt, wird eine Fehlermeldung ausgegeben und die Eingabe wiederholt.

`get_input_case()`: Diese Funktion bestimmt, ob das Schlüsselwort ausschliesslich aus Kleinbuchstaben («lower») oder aus Gross- und Kleinbuchstaben («both») bestehen darf. Ungültige Eingaben werden mit einer Fehlermeldung abgefangen, und die Benutzenden werden zur erneuten Eingabe aufgefordert.

Lösung zum Auftrag

Um den Ablauf des Programms besser nachzuvollziehen, wird im Folgenden die Reihenfolge der einzelnen Schritte beschrieben.

1. Das Programm startet und fordert die Benutzenden zur Eingabe eines MD5-Hashwerts auf.
2. Die Benutzenden geben eine Zeichenlänge ein, die bestimmt, wie lang der ursprüngliche Schlüssel war, dieser darf maximal 8 Zeichen beinhalten.
3. Die Benutzenden entscheiden, ob der Schlüssel nur Kleinbuchstaben («lower») oder auch Grossbuchstaben («both») enthalten soll.
4. Die Klasse MD5Cracker wird mit der Eingabe der Benutzenden initialisiert.
5. Die Anzahl der möglichen Kombinationen wird berechnet und den Benutzenden angezeigt.
6. Das Programm beginnt mit dem Brute-Force-Versuch, indem alle möglichen Zeichenkombinationen der angegebenen Länge generiert und gehasht werden.
7. Falls eine Übereinstimmung mit dem eingegebenen Hashwert gefunden wird, wird der Schlüssel ausgegeben.
8. Falls kein Treffer erzielt wird, wird eine entsprechende Fehlermeldung angezeigt.
9. Die Gesamtzeit für den Brute-Force-Prozess wird berechnet und ausgegeben.

Zum Schluss stellt die Bedingung **`if __name__ == "__main__": main()`** sicher, dass das Programm nur dann gestartet wird, wenn es direkt ausgeführt wird. Falls es in ein anderes Skript importiert wird, bleiben die Funktionen und Klassen verfügbar, ohne dass das Hauptprogramm automatisch startet.

Abschluss

Reflexion Weg zum Ziel

Die Entwicklung des Brute-Force-Tools zur Entschlüsselung von MD5-Hashes war ein iterativer Prozess, der sowohl technologische als auch methodische Herausforderungen mit sich brachte. In dieser Reflexion betrachten wir die wesentlichen Schritte, Herausforderungen und die gewonnenen Erkenntnisse aus der Umsetzung.

Ausgangslage und Zielsetzung

Das Projekt begann mit der klar definierten Aufgabe, ein Tool zu entwickeln, das mittels Brute-Force-Angriffen kurze, gehashte Zeichenketten rekonstruieren kann. Python war als Programmiersprache vorgegeben, wodurch auf eine breite Palette an Bibliotheken zurückgegriffen werden konnte, insbesondere **`hashlib`** für die MD5-Berechnung und **`itertools`** zur effizienten Generierung von Zeichenkombinationen.

Technische Herausforderungen und Iterationen

Ein erster grundlegender Ansatz bestand darin, alle möglichen Zeichenkombinationen einer gegebenen Länge zu generieren, zu hashen und mit dem Ziel-Hash zu vergleichen. Dies war eine naive Implementierung, die sich jedoch als ineffizient herausstellte, insbesondere für längere Zeichenketten.

Daher wurden mehrere Optimierungsschritte durchgeführt:

- **Effiziente Eingabeverarbeitung:** Die Eingabevalidierung wurde verbessert, um fehlerhafte Hashwerte oder ungültige Zeichenlängen frühzeitig abzufangen.
- **Reduktion der Komplexität:** Spätere Refactorings haben den Code gestrafft, indem unnötige Komplexität reduziert wurde, beispielsweise durch die Entfernung von Multiprocessing, das in diesem Fall keinen signifikanten Geschwindigkeitsvorteil brachte.
- **Laufzeitanalyse:** Die Implementierung einer Zeitmessung gab wertvolle Einblicke in die Skalierbarkeit des Programms und zeigte, dass Brute-Force-Versuche über eine gewisse Zeichenlänge hinaus exponentiell ineffizient wurden.

Parallele Verarbeitung

Ein früherer Versuch, die Performance durch parallele Verarbeitung zu steigern, um die Berechnung von Hashwerten effizienter zu gestalten, zeigte gemischte Ergebnisse. Dabei wurden verschiedene Methoden getestet, darunter asynchrone Hash-Berechnungen und verschiedene Varianten der Parallelisierung:

Ansatz	Beschreibung	Geschwindigkeitsgewinn	Probleme
Hashes asynchron berechnen	Nutzung von asynchronen Funktionen zur parallelen Berechnung der Hashwerte.	Verlangsamt	Erhöhter Speicherverbrauch, schwierigere Fehlerbehandlung. Viel Overhead.
Pro Buchstaben ein Thread	Aufteilung der Arbeit so, dass jeder Thread für eine bestimmte Buchstabenposition zuständig ist.	Geringer bis moderater Gewinn	Hoher Synchronisationsaufwand, ineffiziente Lastverteilung.
Single-Threaded	Lineare Berechnung ohne parallele Verarbeitung.	Basiswert	Stabil, aber langsam für grosse Eingaben.

Die Analyse ergab, dass asynchrone Berechnungen und eine strikte Aufteilung der Threads auf Buchstabenpositionen zwar theoretisch vorteilhaft waren, aber in der Praxis durch den hohen Verwaltungsaufwand ineffizient wurden. Letztendlich wurden alle parallelen Ansätze verworfen, da der zusätzliche Overhead die Vorteile oft zunichtemachte.

Reflexion der Methodik

Ein wesentlicher Lerneffekt war die Erkenntnis, dass Brute-Force-Methoden in der realen Welt begrenzte Anwendungsfälle haben. Während die Implementierung technisch interessant und lehrreich war, zeigte sich auch deutlich die Schwäche von MD5 als Hash-Algorithmus, insbesondere im Hinblick auf Kollisionen und die leichte Berechenbarkeit durch spezialisierte Hardware.

Durch die iterative Optimierung des Codes wurde ein besseres Verständnis für die Herausforderungen der Algorithmen-Effizienz entwickelt. Die Balance zwischen Lesbarkeit, Performance und Komplexität war ein wichtiger Aspekt bei der Weiterentwicklung des Programms.

Fazit

Das Projekt verdeutlichte die Grenzen und Möglichkeiten von Brute-Force-Methoden zur Passwortentschlüsselung. Während das Tool erfolgreich funktionierte, bestätigte die Umsetzung die theoretischen Annahmen: MD5 ist für sicherheitskritische Anwendungen ungeeignet. Gleichzeitig hat die Auseinandersetzung mit paralleler Verarbeitung, Komplexitätsreduktion und Performance-Analysen wertvolle Einblicke in die Softwareentwicklung und IT-Sicherheit geliefert.

Ausblick

Das aktuelle MD5-Cracker-Programm verwendet eine Brute-Force Methode zur Entschlüsselung von Hashes. Dabei gibt es mehrere Möglichkeiten, die Effizienz und Geschwindigkeit des Programms zu steigern. Nachfolgend werden einige vielversprechende Ansätze vorgestellt.

Andere Programmiersprache / Assembler

Momentan ist das Programm in Python implementiert, was eine hohe Lesbarkeit und einfache Erweiterbarkeit bietet. Allerdings könnte die Performance erheblich verbessert werden, wenn die rechenintensiven Teile in einer niedrigeren Programmiersprache wie C oder sogar Assembler (ASM) geschrieben würden. Insbesondere könnte eine direkte Speicher- und Prozessorsteuerung durch ASM die Geschwindigkeit des Hashing-Vergleichs optimieren. Alternativ wäre eine Implementierung in Rust eine Option, da Rust sowohl hohe Performance als auch Speicher-Sicherheit bietet.

Nutzung der GPU anstatt/zusätzlich zur CPU

Das derzeitige Programm nutzt ausschliesslich die CPU für die Brute-Force-Berechnungen. GPUs sind jedoch für parallelisierbare Aufgaben, wie Hash-Berechnungen, wesentlich besser geeignet. Bibliotheken wie CUDA (für NVIDIA-GPUs) oder OpenCL ermöglichen eine drastische Beschleunigung des Hashing-Prozesses, indem viele Kombinationen gleichzeitig getestet werden. Programme wie hashcat[9] nutzen diese Technik bereits erfolgreich. Eine Implementierung in PyCUDA[10] könnte eine erste Optimierung darstellen.

Nutzung von Lookup Tabellen für Schlüsselwörter

Eine effektive Alternative zur Brute-Force-Methode ist die Verwendung von vorberechneten Hash-Werten in Form von Lookup Tables. Diese Tabellen enthalten eine grosse Anzahl von vorab berechneten Hashes für bestimmte Schlüsselwort-Muster. Da im gegebenen Szenario ausschliesslich kurze Schlüsselwörter mit Gross-

[9] (hashcat, 2025)
[10] (Python Software Foundation, 2025)

und Kleinbuchstaben verwendet wurden, können gezielt Lookup Tables für diese Kombinationen erstellt werden.

Durch das Speichern der Hashes für alle möglichen Schlüsselwörter bis zu einer bestimmten Länge in einer Datenbank, kann das Programm Hash-Werte schnell nachschlagen, anstatt sie erneut zu berechnen. Dies spart erheblich Rechenzeit im Vergleich zur klassischen Brute-Force-Methode.

Ein praktischer Ansatz wäre, die Lookup Tables einmalig für alle möglichen Kombinationen bis zu einer Länge von 6 bis 8 Zeichen zu generieren und in einer effizienten Datenbankstruktur abzulegen. Anschliessend kann das Programm anstelle einer vollständigen Berechnung einfach eine Lookup-Operation ausführen, um das ursprüngliche Schlüsselwort zu bestimmen.

Fazit

Durch eine Kombination aus leistungsfähigeren Programmiersprachen und intelligenten Optimierungsmethoden wie Lookup Tables könnte das aktuelle MD5-Cracker-Programm erheblich verbessert werden. Dies würde nicht nur die Performance steigern, sondern auch die praktische Anwendbarkeit erweitern.

Quellenverzeichnis

hashcat. (1. März 2025). *hashcat*. Abgerufen am 1. März 2025 von https://hashcat.net/hashcat/

okta. (29. August 2024). *okta.com*. Von What is MD5?: https://www.okta.com/identity-101/md5/ abgerufen

Python Software Foundation. (1. März 2025). *hashlib — Secure hashes and message digests*. Abgerufen am 1. März 2025 von https://docs.python.org/3/library/hashlib.html

Python Software Foundation. (28. Februar 2025). *itertools — Functions creating iterators for efficient looping*. Abgerufen am 28. Februar 2025 von https://docs.python.org/3/library/itertools.html

Python Software Foundation. (1. März 2025). *pycuda 2025.1*. Abgerufen am 1. März 2025 von https://pypi.org/project/pycuda/

Python Software Foundation. (1. März 2025). *re — Regular expression operations*. Abgerufen am 1. März 2025 von https://docs.python.org/3/library/re.html

Python Software Foundation. (1. März 2025). *string — Common string operations*. Abgerufen am 1. März 2025 von https://docs.python.org/3/library/string.html

Python Software Foundation. (1. März 2025). *time — Time access and conversions*. Abgerufen am 1. März 2025 von https://docs.python.org/3/library/time.html

Spasojevic, A. (16. August 2024). *Phoenixnap*. Von Was ist MD5?: https://phoenixnap.de/Glossar/MD5-Message-Digest-Algorithmus abgerufen