dehashr

Anforderungsanalyse

|  |  |
| --- | --- |
| **Autor(en):** | Kris |
| **Version:** | 2.0 |
| **Erstellt am:** | 3. Dezember 2020 |
| **Letzte Änderung:** | 9. Dezember 2020 |
| **Speicherort:** | Dehashr |

Inhalt

[1.1 Systemidee 3](#_Toc58422704)

[Die wichtigsten Funktionen sind: 3](#_Toc58422705)

[1.2 Management Summary 3](#_Toc58422706)

[1.3 Projektteam 3](#_Toc58422707)

[1.4 Glossar 3](#_Toc58422708)

[2 Ausgangslage (IST), Problembereiche 3](#_Toc58422709)

[2.1 Beschreibung der Ausgangslage 3](#_Toc58422710)

[3 Ziele (SOLL) 4](#_Toc58422711)

[3.1 Beschreibung der Ziele 4](#_Toc58422712)

[3.2 Produktperspektive, Nutzen 4](#_Toc58422713)

[3.3 Abgrenzung 4](#_Toc58422714)

[4 Anforderungsanalyse 5](#_Toc58422715)

[4.1 Identifizierung der Akteure 5](#_Toc58422716)

[4.2 Anforderungskatalog 6](#_Toc58422717)

[A: Funktionale Anforderungen 6](#_Toc58422718)

[NF: Nichtfunktionale Anforderungen 6](#_Toc58422719)

[5 Systemablaufmodelle (Aktivitäten) 7](#_Toc58422720)

[5.1 Aktivität «XY» 7](#_Toc58422721)

[6 Risiko-Analyse 8](#_Toc58422722)

[6.1 Risikokatalog 8](#_Toc58422723)

[7 Anhang 9](#_Toc58422724)

[7.1 Termine 9](#_Toc58422725)

Einleitung

## Systemidee

### Die wichtigsten Funktionen sind:

* Programm ist abrufbar im Terminal (Programm ist eine CLI Applikation)
* Man kann die folgenden kommandozeilen-argumente mitgeben:

1. Den input Hash
2. Das Textfile der wordlist / Bruteforce methode

## Management Summary

Mit diesem Programm können sie Hashes cracken

## Projektteam

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Kürzel** | **Funktion** | **Kontakt (Telefon, Mail...)** |
| Kris Huber | KH | Entwickler/Projektleiter | 076 317 80 11 kris.n.huber@gmail.com |

## Glossar

Die im Projekt verwendeten Fachbegriffe und Abkürzungen in alphabetischer Reihenfolge.

|  |  |
| --- | --- |
| **Begriff** | **Beschreibung** |
| C99 | C Programmier Sprache |
| CLI | Command line interface |
| Command-Line Argumente | Benutztereingaben über die Konsole |

# Ausgangslage (IST), Problembereiche

In diesem Kapitel wird die Ausgangslage beschrieben und es werden die Problembereiche identifiziert, die sich in der heutigen Situation zeigen.

## Beschreibung der Ausgangslage

Ich habe im Basislehrjahr die möglichkeit ein Projekt zu erstellen, das Thema kann ich frei wählen. Ich habe mich für das Thema ein Hash Cracken in C entschieden, da es mir spass macht so etwas zu entwickeln.

# Ziele (SOLL)

In diesem Kapitel werden die übergeordneten Ziele beschrieben, die mit dem zu entwickelnden System erreicht werden sollen.

## Beschreibung der Ziele

Einen Funktionierenden Hash Cracker den man in der Konsole bedienen kann.

## Produktperspektive, Nutzen

Finden sie ihr vergessenes passwort heraus mit dieser Software!

## Abgrenzung

Ich möchte mich an die UNIX Philosophie halten, heisst keine unnötige Features einbauen.

# Anforderungsanalyse

Die Anforderungen an das zu entwickelnde System definieren alle zu erfüllenden Eigenschaften oder die zu erbringende Leistung, sowie allfällige technische Vorgaben und weitere Forderungen des Kunden im Zusammenhang mit dem (den) zu erstellenden Produkt(en).

## Identifizierung der Akteure

Akteur ist ein Hacker der diese Software als eines seiner Werkzeuge benutzt. Akteur ist ein Informatik spezialist und kennt sich gut mit Computern aus.

* **Hacker:** Kann dehashr ausführen und seine kopie von dehashr modifizieren.



## Anforderungskatalog

Der Anforderungskatalog ist eine priorisierte Liste, die alles enthält, was im zu entwickelnden Produkt enthalten sein soll. Es wird unterschieden zwischen funktionalen und nicht-funktionalen Anforderungen.

Jede Anforderung ist in Alltagssprache, in der Form einer "User-Story", formuliert, mit einer ID zur späteren Identifizierung versehen sowie mittels 3 verschiedener Kategorien priorisiert: 1 = hohe Prorität, 2 = mittlere Priorität, 3 = keine Priorität. Diese drei Prioritäten repräsentieren die Verpflichtungen "must", "should" und "nice to have".

Im Verlauf der Zeit können neue Anforderungen hinzukommen und/oder bestehende Anforderungen können wegfallen. Anforderungen, die wegfallen, sind im Dokument zu belassen und als ~~gestrichen~~ zu markieren.

Hinter einer Anforderung kann ihn eckigen Klammern in der Form [Z#] eine Zusatzinformation hinterlegt werden und mit [F#] bzw. [F#, F#, …] können eine oder mehrere offene Fragen referenziert werden.

### A: Funktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| **ID** | **Anforderung** | **Prio** |
| A001 | Als Hacker möchte einen Hash als input geben. | 1 |
| A002 | Als Hacker möchte ich verschiedene Argumente mitgeben können, um z.B. den hash algorythmus auszuwählen. (Z1) | 1 |
| A003 | Als Hacker möchte ich das Resultat erhalten. (der gecrackte Hash) | 1 |
| A004 | Als Hacker möchte ich mindestens den Hash algorythmus MD5 Auswählen können. | 1 |
| A005 | Dehashr sollte die anzahl CPU kerne herausfinden und dadurch mit mehreren Threads den cracking prozess ausführen. | 2 |
| A006 | Als Hacker möchte ich die folgenden Modi auswählen können:   * Bruteforce (Standart, wird bei A001-A004 implementiert) * Wordlist | 2 |

**Zusatzinformationen:**

(Z1) folgende parameter:

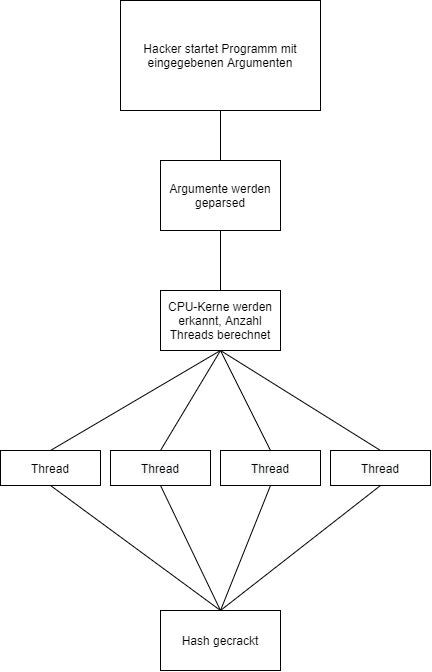
### NF: Nichtfunktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| **ID** | **Anforderung** | **Prio** |
| NF001 | In diesem Programm ist die schnelligkeit am wichtigsten. | 1 |
| NF002 | Die Dateien sollen gut strukturiert sein. (Quellcode) | 1 |
| NF003 | Die Software soll auch auf windows verfügbar sein. | 2 |
| NF004 | Die Software soll auch für ARM prozessoren verfügbar sein. | 2 |
| NF005 |  |  |

# Systemablaufmodelle (Aktivitäten)

Dieses Kapitel zeigt die wichtigsten und/oder komplexesten funktionalen Anforderungen in ihrem Ablauf. Dazu werden die einzelnen Aktivitätsschritte detailliert analysiert. Für die Darstellung der einzelnen Aktivitätsschritte werden Aktivitätsdiagramme nach UML verwendet.

## Programmablauf



# Risiko-Analyse

Ziel der Risiko-Analyse ist es, die Kontrolle zu haben über die wesentlichen Projektrisiken während des Projektverlaufs sowie die Identifizierung der grössten respektive der schwerwiegendsten Risiken, welche dann bei der Implementation als erstes angegangen/getestet werden (Greatest Risk First).

## Risikokatalog

Legende:

* **w** = Wahrscheinlichkeit des Auftretens: 1 = null, 2 = klein, 4 = eher klein 6=mittel 8=hoch, 10 = sehr hoch, >50%
* **b** = Bedeutung der Auswirkungen: 1 keine, 3 minim, 6 signifikant, 8 schwer, 10 fatal, Abbruch
* **p** = Prävention, Behebung, Vermeidung: 1 gratis, 3 leicht, 4 gut 90%, 6 mittel, 8 schlecht,10 unvermeidbar

Das grösste Risiko steht am Anfang des Katalogs, das geringste am Ende.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nr: Risiko** | Mehrere Threads bringen nichts. | **w \* b \* p** | 2 \* 6 \* 6 (72) |
| Ursache | **Jeder Thread beginnt das Cracken an der gleichen stelle.** | | |
| Folgen | **Der Cracking Prozess verlangsamt sich.** | | |
| Massnahmen | **Jeder thread sollte an anderen Stellen beginnen** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nr: Risiko** | Bediehnung zu kompliziert | **w \* b \* p** | 2 \* 6 \* 6 (72) |
| Ursache | **zu viele Argument Optionen** | | |
| Folgen | **Benutzer versteht bedienung des Programmes nicht.** | | |
| Massnahmen | **Argument Optionen simple halten** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Nr: Risiko** | Das compilieren von Source schlägt fehl. | **w \* b \* p** | 2 \* 6 \* 6 (72) |
| Ursache | **Eine Library fehlt, Make nicht installiert etc.** | | |
| Folgen | **Das compilieren von Source schlägt fehl.** | | |
| Massnahmen | **Alle Dependencies in dem README aufschreiben und GitHub Issues im Auge behalten** | | |

# Anhang

## Termine

|  |  |  |
| --- | --- | --- |
| **Datum** | **Ziel, Beschreibung, Meilenstein** | **Verantwortlich** |
| 13.12.2016 | Fertigstellung Iteration 1, Prototyp dem Kunden zeigen (Sitzung) |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |