Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Erhan Yilmaz

Password Cracking as a Service - Ein Framework zur Integration externer Dienste

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Erhan Yilmaz

Password Cracking as a Service - Ein Framework zur Integration externer Dienste

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Hübner
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Abgegeben am 05.11.2014

Erhan Yilmaz

Thema der Bachelorarbeit
Password Cracking as a Service - Ein Framework zur Integration externer Dienste

Stichworte
Passwortsicherheit, Passwort-Cracking, Framework, Sicherheit, Hashing-Verfahren

Kurzzusammenfassung
Diese Arbeit handelt von dem Entwurf und der Implementierung eines Frameworks, welches die Integration externer Passwort-Cracking-Software über eine generische Schnittstelle möglich macht. Zusätzlich ist es möglich angebotene Dienste ohne Fach- oder Branchenwissen zu bedienen und zu benutzen.

Erhan Yilmaz

Title of the paper
Password Cracking as a Service - A Framework for the integration of external serivces

Keywords
Password Security, Password-Cracking, Framework, Security, Hashing-Procedures

Abstract
This thesis is about the concept and design of an application, which allows the integration of external passwort-cracking-software through usage of a generic interface. Additionally, it is possible to use and manage provided services without knowledge about password-cracking or without knowledge about usage of password-cracking-software.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Beinahe täglich erreichen uns Nachrichten von Unternehmen, deren Datenbanken kompromittiert wurden. Die Angreifer haben es in diesen Fällen häufig auf die Benutzernamen und die Passwörter der Kunden abgesehen, denn diese sind Geld wert. Im Jahr 2012 erzielten E-Mail Zugangsdaten einen Wert von bis zu 20 US-Dollar pro Account. [ISTR 2013]

Entwendete Kundendaten oder Passwörter führen bei Firmen zu erheblichen Schäden.
Solche Vorfälle verursachen immense Kosten, da Kunden möglicherweise entschädigt werden müssen.
So konnte im Jahr 2011 eine Hackergruppe erfolgreich in die Datenbanken des Playstation-Networks der Sony Corporation eindringen und Accounts im siebenstelligen Bereich stehlen. Schätzungen zufolge hat dieser Angriff Sony bis zu 3.2 Milliarden US-Dollar gekostet. [ROW 2011]

Der Schutz von Passwörtern und anderen sensiblen Daten sollte für Unternehmen also eine hohe Priorität haben. Aus diesem Grund sollten Passwörter immer als Hashwert abgespeichert werden. Falls es passiert, dass ein Angreifer erfolgreich Passwörter entwendet, kann er somit nicht die Passwörter als Klartext einsehen.
Um zu verhindern, dass entwendete Passwörter mit wenig Aufwand entschlüsselt werden, können Unternehmen in regelmäßigen Abständen Penetrationstests durchführen. Ein Teil dieser Tests ist die Untersuchung der Passwortsicherheit für alle Mitarbeiter mit Hilfe von Passwort-Cracking-Tools. Schwache Passwörter werden erkannt und können durch stärkere Passwörter ersetzt werden.
Solche Tests müssen von einem Experten durchgeführt werden, da nur er die nötige Expertise zur Bedienung eines solchen Tools besitzt. Erschwerend kommt hinzu, dass viele Passwort-Cracking-Tools am Markt existieren. Experten mit Fachkenntnis sind dementsprechend teuer.

Es wäre hilfreich, wenn Unternehmen eigene Möglichkeiten zum Testen Ihrer Passwörter zur Verfügung gestellt bekommen. Auf diese Weise kann die Sicherheit firmenintern gewährleistet werden, selbst wenn kein Experte verfügbar ist.

1.2 Zielsetzung

Um das in der Einleitung beschriebene Problem zu lösen, sollen allgemein anerkannte Passwort-Cracking-Tools unterschiedlicher Anbieter in einer Anwendung kombiniert werden. So ist es möglich für die jeweilige Problemstellung das geeignetste Passwort-Cracking-Tool auszuwählen. Die Bedienung der Anwendung soll auch branchenfremden Personen möglich gemacht werden.
Um eine komplette Arbeitspalette anbieten zu können, sollen Werkzeuge zur Erstellung von geeigneten Passwortdateien bereitgestellt werden. Mechanismen zur Identifikation der Inhalte von Passwortdateien sollen diese Palette vervollständigen.

Im Rahmen dieser Arbeit soll eine Anwendung entstehen, welche die folgenden funktionalen Eigenschaften erfüllt:

1. Integration und Nutzung unterschiedlicher Passwort-Cracking-Tools
2. Nutzung von Passwort-Cracking-Tools ohne Fachwissen
3. Nutzung von Funktionen zur Erstellung eigener Passwortdateien
4. Nutzung von Funktionen zur Identifikation von Hashing-Verfahren
5. Nutzung einer Menge von Passwörtern als Wissensbasis
6. Aufbau von Szenarien zum Testen der Wissensbasis (cracken der Passwörter), mithilfe der integrierten Passwort-Cracking-Tools
7. Aufzeichnung und Aufbereitung der während des Tests gesammelten Daten

Diese Anwendung soll beim Testen von Passwortsicherheit helfen. Ein weiterer Zweck der Anwendung ist es, den Vorgang der Passwortüberprüfung auch Personen möglich zu machen, welche kein Wissen über die Nutzung von Passwort-Cracking-Tools besitzen. Die Nutzung der eingebundenen Tools soll dabei möglichst einfach gehalten werden. Ein Benutzer muss dazu die gewünschte Vorgehensweise und die Passwortdatei angeben.
Darüber hinaus sollen Funktionalitäten zur Erstellung von Passwortdateien angeboten werden. Die Passwörter dieser Datei können, entsprechend den Angaben des Nutzers, verschlüsselt werden.
Ebenso soll es möglich sein eine Datei mit verschlüsselten Passwörtern zu benutzen, um die benutzten Hashing-Algorithmen zu identifizieren.
Sollte sich ein neues Verfahren oder Tool am Markt etablieren, kann es mit einfachen Mitteln in die Servicelandschaft der Anwendung integriert und genutzt werden.
Die Anwendung soll mit möglichst geringem Aufwand aktuell und performant gehalten werden. Deshalb ist es wichtig den Aufwand für das Einbinden eines Passwort-Cracking-Tools möglichst gering zu halten.

1.3 Methodik

Der erste Teil dieser Arbeit vermittelt die Grundlagen von Verschlüsselungs- und Hashing-Verfahren und gibt einen Überblick auf die unterschiedlichen Ansätze zum Cracken von Passwörtern. Zusätzlich wird eine Übersicht der in dieser Arbeit benutzten Passwort-Cracking-Tools gegeben.

Bezüglich der Grundlagen der Verschlüsselung, des Hashings und des Crackings werden die Funktionsweise sowie die Vor- und Nachteile erklärt. Zuletzt wird anhand bekannter Verfahren der Einsatz erläutert.

Der Überblick über genutzte Passwort-Cracking-Tools soll die Nutzung und den Funktionsumfang beschreiben und auf mögliche Defizite hinweisen.

Der zweite Teil dieser Arbeit beschäftigt sich mit dem Entwurf, der Implementierung und der Validierung des resultierenden Tools.

Anhand der Anforderungen und der Anwendungsfälle werden Systemoperationen (Schnittstellenoperationen) abgeleitet. Diese werden zum Entwurf des Komponentenschnitts und der Architektur genutzt.

Es muss ein Format zur Nutzung gewünschter Services definiert und in die Architektur integriert werden. Dabei kann auf bereits etablierte Formate (z.B. XML) zurückgegriffen werden.
Um die Machbarkeit zu zeigen, wird ein Prototyp implementiert, welcher bis zu zwei externe Passwort-Cracking-Tools anbietet.
Abschließend werden die gewonnenen Erkenntnisse zusammengefasst und ein Fazit gezogen, um sich kritisch mit der weiteren Nutzung der Anwendung auseinander zu setzen.

2 Theorie und Grundlagen

In diesem Kapitel werden Verfahren und Ansätze beschrieben, welche für das Verständnis dieser Bachelorarbeit erforderlich sind.
Nach einer kurzen Einführung in die Kryptographie und das Hashing im ersten Abschnitt, wird auf die Eigenschaften allgemeiner Verfahren eingegangen und die Funktionsweise erläutert. Daraufhin werden einige weit verbreitete Algorithmen im Detail beschrieben.
Im Abschnitt 2.3 Cracking und Cracking-Ansätze werden die verschiedenen Möglichkeiten zur Entschlüsselung von Passwörtern erläutert. Dabei wird auch auf Unterschiede zwischen den Ansätzen eingegangen.
Zuletzt werden die in dieser Arbeit genutzten Passwort-Cracking-Tools beschrieben.

2.1 Verschlüsselung und Kryptographie

Ursprünglich war die Kryptographie synonym mit der Verschlüsselung von Daten. In der heutigen Zeit ist die Verschlüsselung von Daten nur noch ein Teilbereich der Kryptographie. Dieses Kapitel beschäftigt sich mit der Verschlüsselung von Daten. [CRYE 2010, S. 23]

2.1.1 Einführung und Secret-Key Encryption

Allgemein geht es bei der Verschlüsselung darum, Daten nur bestimmten Parteien oder Personen zugänglich zu machen. Als generisches Beispiel möchte eine Person, Alice, eine Nachricht an eine andere Person, Bob, versenden.

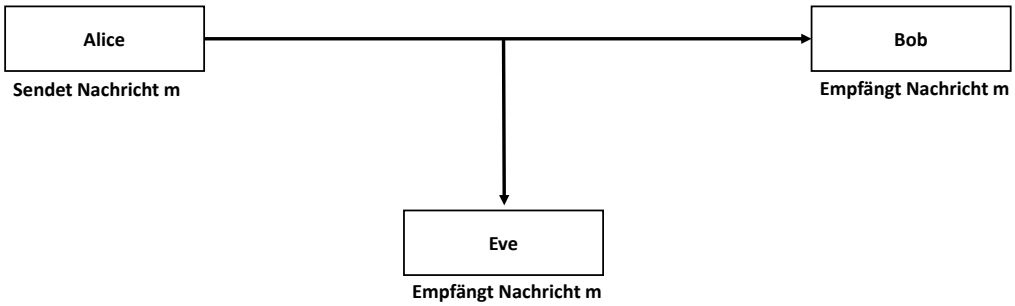

Abbildung 2.1: Beispiel für das Mithören von Nachrichten

In der Abbildung 2.1 ist zu erkennen, dass der Kommunikationskanal über den Alice und Bob Nachrichten austauschen nicht sicher ist, so dass eine dritte Person, Eve, jede Nachricht mitlesen kann. Das Ziel von Verschlüsselungsverfahren ist es, die Kommunikation zwischen Alice und Bob sicher zu machen, so dass Eve keinen Nutzen aus dem Mitlesen der Nachrichten ziehen kann.
Um dies zu erreichen müssen Alice und Bob bei ihrer Kommunikation ein Verschlüsselungsverfahren anwenden. Im folgenden Beispiel verwenden Alice und Bob ein symmetrisches Verschlüsselungsverfahren, wobei genau einen Schlüssel zur Chiffrierung und Dechiffrierung der Nachricht existiert.
Bei der Verwendung eines symmetrischen Verfahrens müssen sich Alice und Bob auf einen Schlüssel einigen, mit dem Sie Nachrichten verschlüsseln wollen. Dies muss über einen separaten Kommunikationskanal geschehen, auf welchen Eve keinen Zugriff hat, da ansonsten der Schlüssel mitgeschnitten werden kann. An dieser Stelle sei erwähnt, dass spezielle Verfahren zum Schlüsselaustausch existieren. Um das Beispiel nicht unnötig zu verkomplizieren, werden Schlüsselaustauschverfahren außen vor gelassen. [CRYE 2010, S. 24 ff]
Nachdem sich Alice und Bob auf einen gemeinsamen Schlüssel geeinigt haben, kann Alice mit dem Verschlüsseln der Nachricht beginnen. Sie benutzt eine Verschlüsselungsfunktion, um

zusammen mit der Nachricht im Klartext und dem Schlüssel eine verschlüsselte Nachricht zu erzeugen. Die Verschlüsselungsfunktion kann wie folgt beschrieben werden:

$$c = Enc_m(K_{Enc}, m)$$

Dabei wird eine Nachricht m zusammen mit dem Schlüssel K_{Enc} durch die Funktion $Dec_C()$ verschlüsselt. Das Resultat ist der Chiffretext c. Diesen chiffrierten Text kann Alice nun an Stelle des Klartexts versenden. Sobald Bob die Nachricht empfängt, kann er den Chiffretext entschlüsseln. Dies tut er mit der Entschlüsselungsfunktion:

$$m = Dec_c(K_{Dec}, c)$$

Der Schlüssel K_{Dec} ist in diesem Fall identisch zum Schlüssel K_{Enc}. Das Resultat der Entschlüsselungsfunktion ist der Klartext m, also die ursprüngliche Nachricht, welche Alice an Bob versendet hat.
Da Eve den Schlüssel nicht besitzt, ist sie nicht in der Lage den mitgeschnittenen Chiffretext zu entschlüsseln. [CRYE 2010, S. 24 ff]

Das obige Beispiel beschreibt ein symmetrisches Verschlüsselungsverfahren (auch Secret-Key Verfahren genannt), bei welchem genau ein Schlüssel zur Ver- und Entschlüsselung existiert. Im Weiteren wird ein asymmetrisches Verfahren beschrieben: die Public-Key Verschlüsselung.

2.1.2 Public-Key Encryption

Ein großes Problem bei symmetrischen Verfahren ist das Schlüsselmanagement. Sollte eine Person mit mehreren anderen Personen sicher kommunizieren wollen, muss mit jeder Person ein Secret-Key vereinbart werden. Sollten also fünf Personen miteinander kommunizieren wollen, müssen vier Schlüssel pro Person vereinbart und ausgetauscht werden. [CRYE 2010, S. 27]
Um dieses Problem teilweise umgehen zu können, wurde die Public-Key Verschlüsselung eingeführt. Im Gegensatz zum symmetrischen Verfahren existieren in der asymmetrischen Public-Key Verschlüsselung zwei Schlüssel pro Person: der öffentliche und der private Schlüssel (public und private Key). Der öffentliche Schlüssel ist allgemein zugänglich, der private Schlüssel ist geheim. Es ist wichtig anzumerken, dass der Schlüssel zum Verschlüsseln einer Nachricht nicht derselbe ist wie der Schlüssel zum Entschlüsseln einer Nachricht. [CRYE 2010, S. 28]
Das folgende Beispiel beschreibt einen Austausch von Nachrichten zwischen Alice und Bob, bei dem ein Public-Key Verfahren zur Verschlüsselung genutzt wird. Konkret wird eine Nachricht von Bob an Alice versendet. Zu diesem Zweck benutzt Bob den allgemein zugänglichen öffentlichen Schlüssel von Alice, um seine Nachricht zu verschlüsseln. Er benutzt eine Verschlüsselungsfunktion:

$$c = Enc_m(PK_{Alice}, m)$$

Um den Chiffretext c zu erzeugen, benutzt Bob den public Key PK_{Alice} von Alice. Der erzeugte Chiffretext kann nun an Alice versendet werden. [CRYE 2010, S. 28]
Beim Empfang der Nachricht benutzt Alice ihren geheimen privaten Schlüssel SK_{Alice}, um die Nachricht zu entschlüsseln. Dazu benutzt sie eine Entschlüsselungsfunktion:

$$m = Dec_c(SK_{Alice}, c)$$

Damit dieses Verfahren die richtigen Ergebnisse erzeugt, also der ursprüngliche Text durch Ver- und Entschlüsselung nicht verändert wird, muss für die Verschlüsselungs- und Entschlüsselungsfunktion folgendes gelten:

$$Dec_c\big(SK_X, Enc_m(PK_X, m)\big) = m, wobei\ Enc_m(PK_X, m) = c$$

Diese Formel sagt aus, dass eine mit dem öffentlichen Schlüssel einer Person verschlüsselte Nachricht nicht verändert werden darf, wenn der resultierende Chiffretext mit dem privaten Schlüssel derselben Person entschlüsselt wird. Dieses Verhalten muss auf Basis des erzeugten Schlüsselpaares sowie der Ver- und Entschlüsselungsfunktion erreicht und garantiert werden. Eine wichtige Vorrausetzung für die Sicherheit ist, dass der private geheime Schlüssel unter keinen Umständen aus dem öffentlichen Schlüssel berechenbar sein darf.
Ein Vorteil dieses Verfahrens ist das Schlüsselmanagement. Jede Person muss nun lediglich ein Schlüsselpaar zur Kommunikation erzeugen. [CRYE 2010, S. 28]
Aufgrund der Schlüssellänge des öffentlichen Schlüssels ist die Public-Key Verschlüsselung wesentlich komplexer als ein symmetrisches Verfahren und ist dementsprechend langsamer. [CRYE 2010]

2.1.3 Grundlage von Verschlüsselungsfunktionen

Wichtig für das Verständnis von Verschlüsselungsverfahren ist die Verrechnung des Schlüssels mit dem Klartext, so dass ein Chiffretext entsteht. Die Grundlage der meisten Verschlüsselungsverfahren basiert auf Transposition, Substitution, XOR-Operationen und Bitshift-Operationen. Dies bedeutet, dass ein Klartext und ein Schlüssel bitweise, auf Basis der oben genannten Operationen, miteinander verrechnet werden. Diese Verrechnung variiert von Verfahren zu Verfahren, so dass nur Teile des Schlüssels für bestimmte Verrechnungsschritte benutzt werden. Das Ergebnis dieser Operation ist die verschlüsselte Repräsentation des Klartexts. Besonders starke und sichere Verschlüsselungsverfahren benutzen alle Operationen mehrfach und auch in unterschiedlicher Reihenfolge. Das mehrfache Anwenden einer Funktion wird auch Runde genannt. [ACRY 1996]
Bei anderen Verfahren werden Nachrichten als Integer verarbeitet und mit einem Schlüssel verrechnet. Das asymmetrische RSA-Verfahren nutzt diese Vorgehensweise. [CRYE 2010, S. 209 ff]

2.1.4 Block Cipher und Block Cipher Modi

Einfach betrachtet sind Block Cipher Verschlüsselungsfunktionen mit einer bestimmten Blocklänge. Die Blocklänge bestimmt die Größe der zu verarbeitenden Daten einer Teilberechnung, die Länge des Schlüssels und des resultierenden Chiffretexts. Eine Block Cipher ist umkehrbar. Es ist möglich aus einem gegebenen Chiffretext und dem Schlüssel die Ursprungszeichenfolge herzuleiten. Dies hat zur Folge, dass abhängig von einem bestimmten Schlüssel keine unterschiedlichen Klartexte auf den gleichen Chiffretext abbilden. Bei der Nutzung einer Block Cipher müssen die zu verschlüsselnden Daten die exakte Länge des Schlüssels besitzen. Auch ein Vielfaches der Schlüssellänge ist möglich. Übliche Blocklängen sind 128 oder 256 Bit. [CRYE 2010, 43 ff]

Bei einem Block Cipher Modus handelt es sich um eine Verschlüsselungsfunktion, welche blockweise arbeitet. Im Gegensatz zur normalen Block Cipher kann man Daten beliebiger Länge verschlüsseln. Um dies zu erreichen, wird Padding eingesetzt. [CRYE 2010, S. 63]
Um Daten beliebiger Länge verarbeiten zu können, ist es nötig, eine Anpassung der Länge der Daten vorzunehmen, so dass sie vom Block Cipher Modus korrekt verarbeitet werden können. Die Länge der Daten muss ein Vielfaches der Schlüssellänge sein. Der Vorgang der Anpassung eines Datensatzes nennt sich Padding. Dabei muss eindeutig erkennbar sein, welche Bits dem ursprünglichen Klartext zuzuordnen sind und welche dem Padding. Andernfalls wäre keine Herleitung des Klartexts aus dem Chiffretext möglich. [CRYE 2010, S. 64 ff]
Es existieren unterschiedliche Regeln um korrektes Padding durchzuführen. Eine einfache Regel ist, dass ein Byte mit dem Wert 128 an den Klartext gehängt wird. Daraufhin wird solange mit Null-Bytes aufgefüllt, bis die Datenlänge ein Vielfaches der Blocklänge ist. [CRYE 2010, S. 64 ff]

2.1.5 One-Way Funktionen

One-Way Funktionen sind essentiell für Public-Key Verfahren. Durch diese Funktionen ist es möglich sichere private und öffentliche Schlüssel zu erzeugen.
Eine One-Way Funktion basiert auf der Annahme, dass ein Wert für eine Funktion *g(x)* leicht zu berechnen ist, wenn *x* gegeben ist. Sollte jedoch nur der Wert der Funktion *g(x)* gegeben sein, ist es um ein Vielfaches schwerer *x* zu berechnen. [ACRY 1996, Kapitel 2.3]
Streng mathematisch betrachtet existieren One-Way Funktionen nicht, sie können auch nicht erzeugt werden. Der Sicherheitsaspekt beruht dabei auf der Tatsache, dass es bisher keinen effizienten Weg gibt den Wert für *x* aus dem Wert von *g(x)* zu berechnen. [ACRY 1996, Kapitel 2.3]
In einem Anwendungsszenario sind einfache One-Way Funktionen nur in spezifischen Fällen einsetzbar, etwa zur Verifikation von Dateimanipulationen. Die Verschlüsselung von Daten ist damit nicht anwenderfreundlich möglich, da eine Entschlüsselung nicht vorgenommen werden kann. [ACRY 1996, Kapitel 2.3]
Aus diesem Grund existieren One-Way Funktionen mit Falltüren. Sie erfüllen die gleichen Voraussetzungen wie eine normale One-Way Funktion mit dem Unterschied, dass die

Berechnung von *x* aus *g(x)* möglich wird, wenn die Falltür-Information bekannt ist. [ACRY 1996, Kapitel 2.3]
Zusätzlich findet eine weitere Unterscheidung von One-Way Hashfunktionen statt: One-Way Hash Funktionen verarbeiten einen Datensatz beliebiger Länge, auch Pre-Image genannt, zu einem Hashwert fester Länge. [ACRY 1996, Kapitel 18.1]
Mathematisch kann dies folgendermaßen beschrieben werden:

$$h = H(m), wobei\ h\ der\ Länge\ x\ entspricht$$

In dieser Aussage beschreibt $H(m)$ die One-Way Hash Funktion, welche auf die Nachricht angewendet wird.
Es ist wichtig anzumerken, dass auch bei One-Way Hash Funktionen die Berechnung des Datensatzes aus seinem Hashwert nicht effizient möglich ist. [ACRY 1996, Kapitel 18.1]
Zuletzt müssen One-Way Hash Funktionen kollisionsresistent sein. Zu diesem Zweck produzieren Implementierungen dieser Funktionen Hashwerte mit einer Länge von mindestens 128 Bit. [ACRY 1996, Kapitel 18.1]

2.2 Verschlüsselungsverfahren

In diesem Kapitel werden zwei der bekanntesten Verschlüsselungsverfahren beschrieben, DES und RSA.

2.2.1 DES

DES bezeichnet den Data Encryption Standard. Es ist eine symmetrische Block Cipher mit einer Schlüssellänge von 56 Bits, exklusive acht Padding-Bits. Die Blocklänge beträgt ebenfalls 64 Bits.
Bei einem kompletten Durchlauf wird ein Klartext in 64 Bit Stücke aufgeteilt, wobei jeder dieser 64 Bit Blöcke in 2 Hälften, zu je 32 Bit, aufgeteilt wird. Die zwei Hälften werden im Sinne des Verfahrens als linke und rechte Hälfte betrachtet. Während der Aufteilung werden die Bits transpositioniert. Diese Transposition hat keine kryptographischen Auswirkungen und ist reine Definition der Autoren des Verfahrens. [CRYE 2010, S. 51 ff]
Jeder Durchlauf besteht aus 16 Runden, wovon jede Runde einen eigenen 48-Bit-Schlüssel besitzt. Jeder dieser Rundenschlüssel wird aus dem 56-Bit-Schlüssel gebildet, indem 48 der Bits ausgewählt werden.

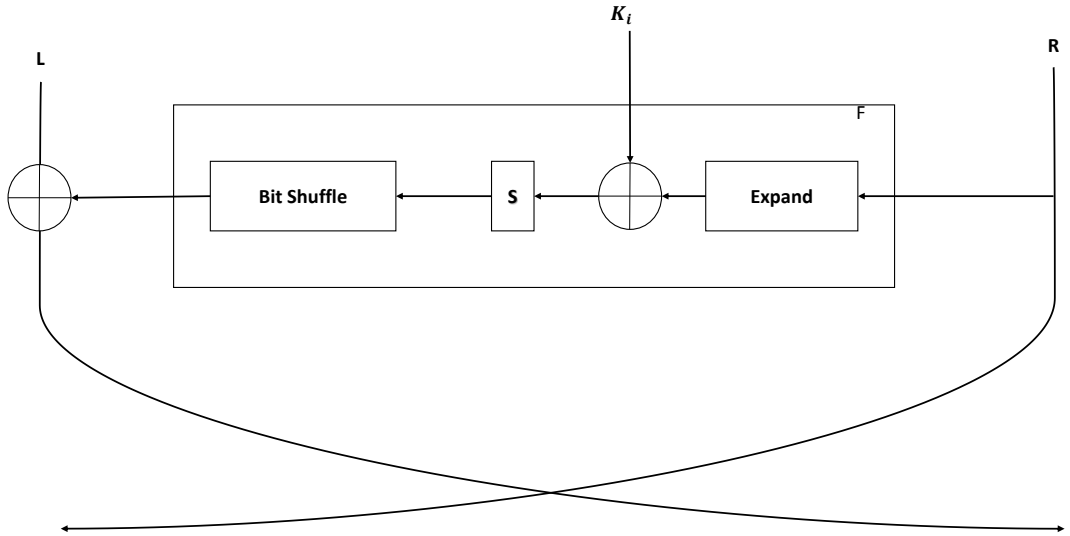

Abbildung 2.2: Anwendung einer Runde DES. [CRYE 2010, S. 51 Abbildung 3.1]

Die Abbildung 2.2 beschreibt eine Runde des DES-Verfahrens:
Das Formelzeichen $\oplus$ steht für bitweise Addition, das Symbol *S* bezeichnet eine Substitution. Das Symbol K_i steht für den 48-Bit-Rundenschlüssel. Der Kasten stellt die Verschlüsselungsfunktion F dar.
Wie zu erkennen ist, wird zunächst die rechte Hälfte expandiert. Dies bedeutet das bestimmte Bits der rechten Hälfte dupliziert werden, so dass 48 Bits vorhanden sind. Dieses Resultat wird mit dem 48-Bit-Rundenschlüssel bitweise addiert. Das Ergebnis dieser Operation wird mit Hilfe einer Substitutionstabelle auf 32 Bits reduziert und vermischt. Nach der Vermischung wird das Ergebnis mit der linken Hälfte bitweise addiert. Zuletzt werden die linke und die rechte Hälfte vertauscht, so dass die linke Hälfte nun als erster Parameter der nächsten Runde benutzt werden kann (also als neue rechte Hälfte). Dieser Vorgang wird 16 Mal wiederholt, das unterliegende Konzept nennt sich Feistel-Netzwerk. [CRYE 2010, S. 51 ff]
Die Entschlüsselung findet in identischer Reihenfolge statt, wenn der Tausch der Hälften in der letzten Runde ausgelassen wird. Ausnahme ist die Reihenfolge der Rundenschlüssel. Dieser Umstand ist einer der Stärken des DES, da sowohl die Ver- als auch Entschlüsselungsfunktion einfach implementiert werden können. [CRYE 2010, S. 52]

Einfaches DES ist in der heutigen Zeit schneller Rechner nicht mehr sicher, welches an der kurzen Länge des Schlüssels und der Datenblöcke liegt. Aus diesem Grund wurde eine Weiterentwicklung zu 3-DES, bzw. Triple-DES, angestrebt. Bei 3-DES wird das DES-Verfahren drei Mal hintereinander angewendet.
Allerdings wird die Weiterentwicklung von Experten als schwierig empfunden, da sich das DES-Verfahren nicht wie eine ideale Block Cipher verhält. [CRYE 2010, S. 54]

2.2.2 RSA

RSA ist ein asymmetrisches Verfahren, welches digitale Signaturen und Verschlüsselung anbietet. Das Verfahren basiert auf der Faktorisierung großer Zahlen und der einhergehenden One-Way Funktionalität. [CRYE 2010, S. 195]

Erzeugung der Schlüssel und die Verschlüsselungsfunktion

Ein wichtiger Bestandteil des RSA-Verfahren ist die Erzeugung des öffentlichen und privaten Schlüssels.

Sowohl der öffentliche als auch der private Schlüssel bestehen aus einem Zahlenpaar (e, N) für den öffentlichen und (d, N) für den privaten Schlüssel. Dabei steht N in beiden Paaren für denselben Wert und wird aus zwei großen Primzahlen erzeugt. Die Werte e und d bezeichnen Exponenten, welche im Rahmen des Verfahrens speziellen Voraussetzungen unterliegen und in Abhängigkeit der Faktoren von N erzeugt werden. [CRYE 2010, S. 196 und S. 200]

Bei der Erzeugung der Schlüssel werden zunächst zwei große Primzahlen p und q gesucht. Diese Zahlen werden basierend auf einem Wert K erzeugt, wobei K für die Länge der Primzahl steht. Zahlen werden mit einer Stelligkeit von K erzeugt und anschließend mit einem Primzahlentest geprüft. Zusätzlich ist darauf zu achten, dass p und q eine ähnliche Größenordnung besitzen. [CRYE 2010, S. 200 und S. 203 ff]

Wenn beide Zahlen erzeugt wurden, werden sie miteinander multipliziert und ergeben den Wert N. N ist doppelt so lang wie p und q. Nun ist es möglich die Werte für e und d zu suchen. Die Voraussetzung für beide Zahlen ist wie folgt:

$$e * d = 1\ (mod\ t), wobei\ t = kgV(p-1, q-1)$$

Dies bedeutet, dass der Wert t teilerfremd zum Wert e ist. Der Wert d ist dabei die Inverse zu $(e\ mod\ t)$. [CRYE 2010, S. 200]

Zur Verschlüsselung einer Nachricht m wird folgende Verschlüsselungsfunktion genutzt:

$$c = m^e\ (mod\ N)$$

Um aus dem Chiffretext c die ursprüngliche Nachricht zu erzeugen wird folgende Entschlüsselungsfunktion genutzt:

$$m = c^d\ (mod\ N)$$

Die Falltür-Information des RSA-Verfahrens zur Berechnung des privaten Schlüssels ist die Faktorisierung von N. Sollte sie bekannt sein, ist die Funktion mit einfachen Mitteln umkehrbar. [CRYE 2010, S. 195 und S. 200 ff]

Obwohl RSA ein Verschlüsselungsverfahren ist, wird diese Funktionalität in der Praxis kaum genutzt, um größere Dateien oder Nachrichten zu verschlüsseln. Dies liegt an der Begrenzung der Länge der zu verschlüsselnden Nachricht, welche an die Größe von N

gebunden ist. Um eine Verschlüsselung mit dem RSA-Verfahren auszuführen, müsste die Nachricht in N große Teile zerlegt werden. Auf jedem dieser Teile müsste die Verschlüsselungsfunktion aufgerufen werden. [CRYE 2010, S. 206]
In der Realität werden deshalb häufig Secret-Key Verfahren zur Verschlüsselung von Nachrichten genutzt. Der Schlüssel des genutzten Secret-Key Verfahrens wird mit dem RSA-Verfahren verschlüsselt und parallel zur verschlüsselten Nachricht versendet. [CRYE 2010, S. 206]

Digitale Signaturen mit RSA
Bei der Verwendung digitaler Signaturen werden Nachrichten mit dem privaten Schlüssel verschlüsselt. Das Ergebnis dieser Operation ist die digitale Signatur. Der Empfänger der Signatur kann die signierte Nachricht mit dem öffentlichen Schlüssel entschlüsseln. Sollte die Nachricht mit dem öffentlichen Schlüssel entschlüsselt werden können, ist sichergestellt, dass der Absender der Nachricht die Partei ist, für die sie sich ausgibt. Die Authentizität ist gewährleistet, da nur eine Partei den korrekten privaten Schlüssel besitzt mit dem die Nachricht verschlüsselt werden konnte. [CRYE 2010, S. 209 ff]
In der Realität werden Nachrichten nicht direkt mit dem privaten Schlüssel verschlüsselt. In der Regel wird die Nachricht gehasht und dann signiert. Dazu ist es notwendig, dass der genutzte Hashing-Algorithmus kollisionsresistent ist. Versendet werden der signierte Hashwert, der ursprüngliche Hashwert und die Nachricht. Wie im Abschnitt „Erzeugung der Schlüssel und Verschlüsselungsfunktion“ beschrieben ist, werden alle drei Nachrichten zuvor mit einem Secret-Key Verfahren verschlüsselt. [CRYE 2010, S. 209 ff]
Der Empfänger ist dann in der Lage den signierten Hashwert zu entschlüsseln und mit dem mitgelieferten Hashwert abzugleichen. Dieser Vorgang gewährleistet Authentizität und Integrität. [CRYE 2010, S. 209 ff]

2.3 Hashing

In diesem Kapitel werden Hashing-Verfahren beschrieben. Dazu werden zunächst die nötigen Sicherheitsmerkmale und Mechanismen zur Erstellung eines Hashwerts dargelegt. Daraufhin wird näher auf die Kompressionsfunktion eingegangen. Letztlich wird der Zusammenhang zur Passwortsicherheit erklärt.

2.3.1 Einführung

Hashfunktionen arbeiten als One-Way Funktionen. Dabei wird eine beliebig lange Eingabenachricht, auch Pre-Image genannt, in einen Hashwert mit einer bestimmten Länge umgewandelt. [ACRY1996, Kapitel 18.1]
Mathematisch lässt sich dies folgendermaßen ausdrücken:

$$h = H(M), wobei\ h\ der\ Länge\ x\ entspricht$$

Der Hashwert wird durch die Anwendung der One-Way Hashfunktion auf der Nachricht erzeugt. Die Länge des Hashwerts entspricht einer bestimmten Größe x, welche vom Verfahren abhängt und sich zwischen 128 und 1024 Bit befindet. [CRYE 2010, S. 77]
Die Nutzung von Hashfunktionen dient der Erzeugung eines einzigartigen Fingerabdrucks für eine Nachricht. [ACRY 1996, Kapitel 18.1]

Zusätzlich besitzt eine One-Way Hashfunktion weitere Merkmale:

1. Wenn eine Nachricht M gegeben ist, kann h leicht berechnet werden.
2. Wenn ein Hashwert h gegeben ist, ist es praktisch unmöglich ein M zu berechnen, so dass die Hashfunktion angewendet auf M einen identischen Hashwert erzeugt.
3. Wenn eine Nachricht M gegeben ist, ist es praktisch unmöglich eine Nachricht $M´$ zu finden, so dass beide Nachrichten einen identischen Hashwert erzeugen.

Um eine One-Way Hashfunktion in kryptographischen Szenarien einsetzen zu können, muss das Merkmal der Kollisionsfreiheit erfüllt sein. Dabei ist anzumerken, dass Kollisionsfreiheit in der Realität nicht existieren kann, da eine unendliche Menge von Eingaben auf einen Hashwert bestimmter Länge abgebildet wird. Die Grundlage der Sicherheit dieses Merkmals beruht darauf, dass Kollisionen existieren, aber nicht gefunden werden können. [ACRY 1996, Kapitel 18.1] [CRYE 2010, S. 78]
Um eine Kollision zu erzeugen sind durchschnittlich $2^{n/2}$ Schritte notwendig, wobei n der Länge des Hashwerts entspricht. [CRYE 2010, S. 78]
Die ideale Hashfunktion verhält sich wie eine zufällige Abbildung aller Eingabewerte auf alle möglichen Ausgabewerte. [CRYE 2010, S. 79]

2.3.2 Erzeugung von Hashwerten

Bei der Erzeugung eines Hashwerts aus einer Eingabe spielt die Kompressionsfunktion eine zentrale Rolle. Die Nachricht wird in Blöcke fester Größe aufgeteilt, wobei für den letzten Block auch Padding eingesetzt werden kann. Die Blöcke haben eine Länge von 512 Bits. Der letzte Block enthält auch einen Wert zur Identifikation der Länge der Eingabe. [CRYE 2010, S. 80]

Zur Erzeugung eines Hashwerts wird die Kompressionsfunktion auf die einzelnen Blöcke angewendet:

$$h_i = f(M_i, h_{i-1})$$

Jede Anwendung der Kompressionsfunktion erzeugt einen Hashwert, welcher zusätzlich zum nächsten Block der Nachricht als Eingabe für die Anwendung des nächsten Schritts genutzt wird. Das Endergebnis ist der Hashwert der Eingabe. Dieses Vorgehen wird als iteratives Verfahren bezeichnet. [CRYE 2010, S. 80]

2.3.3 Entschlüsselung und Pre-Image

Es ist nicht möglich einen Hashwert im Sinne eines Verschlüsselungsverfahrens zu entschlüsseln, da keine inverse Funktion existiert. Um das Pre-Image zu finden ist es notwendig einen Klartext zu finden, welcher auf denselben Hashwert abbildet. Aufgrund der Eigenschaft der Kollisionsfreiheit ist dieser Klartext die „Entschlüsselung“ des Hashwerts. Dieses Vorgehen benötigt im Durchschnitt $2^{n/2}$ Schritte, wobei n der Länge des Hashwerts entspricht. [CRYE 2010, S. 79]

2.3.4 Hashing und Passwortsicherheit

Die häufigste Form von Authentifikation basiert auf Passwörtern. Dabei werden die Passwörter nicht im Klartext gespeichert, weil ein erfolgreicher Angriff direkten Zugriff auf die Passwörter gewähren würde. Stattdessen werden Passwörter als Hashwert einer kryptografischen Hashfunktion gespeichert. Ein Angreifer wird dadurch um ein Vielfaches verlangsamt, weil er Ressourcen aufbringen muss, um die gehashten Passwörter zu entschlüsseln. [DUE 2013]

Zur Entschlüsselung von Hashwerten muss ein Angreifer eine große Menge potentieller Passwörter erstellen. Die Passwörter müssen verschlüsselt werden, um gegen den zu entschlüsselnden Hashwert geprüft zu werden. Bei einer Übereinstimmung zählt das Passwort als entschlüsselt. [DUE 2013]

Aufgrund des Fortschritts der heute genutzten Hardware ist die einmalige Anwendung bestimmter Hashing-Verfahren nicht mehr ausreichend. Um dieses Problem zu lösen werden Hashing-Verfahren in vielen Iterationen benutzt, wobei beispielsweise 1000 Iterationen des SHA-1 Algorithmus angewendet werden. Dieses iterative Vorgehen verlangsamt den Angreifer, aber auch den Authentifizierungsvorgang in gleichem Maße. [DUE 2013]

Um zusätzlichen Schutz für Passwörter zu bieten, werden Saltwerte genutzt. Bei Saltwerten handelt es sich um eine zufällige Folge von Zeichen, welche mit dem Passwort verrechnet werden. Dabei existieren unterschiedliche Regeln zur Verrechnung des Saltwerts mit dem Passwort. So kann beispielsweise eine Konkatenations-Operation ausgeführt werden. Nach der Verrechnung des Passworts mit dem Saltwert wird das Ergebnis gehasht und gespeichert. Der genutzte Saltwert muss gesondert gespeichert werden und ist pro Benutzer immer gleich. Sollte ein Angreifer Zugriff auf den gespeicherten Hashwert bekommen, wird die Anzahl der benötigten Versuche zum Entschlüsseln um den Faktor aller vom Saltwert annehmbaren Werte erhöht. Mit diesem Vorgehen wird die Benutzung vorberechneter Tabellen, sogenannter Rainbow-Tabellen, effektiv ausgeschaltet. [DUE 2013]

2.4 Hashing-Verfahren

In diesem Abschnitt werden zwei der bekanntesten Hashing-Verfahren beschrieben: das MD5-Verfahren und das SHA-Verfahren.

2.4.1 MD5

Das MD5-Verfahren ist eine Verbesserung des MD4-Verfahrens und erzeugt einen 128-Bit Hashwert.

Bei der Ausführung wird die Eingabe zunächst gepaddet, so dass sie 64 Bits kürzer ist als ein Vielfaches von 512. Konkret wird ein 1-Bit angehängt, woraufhin solange mit Nullen aufgefüllt wird, bis die gewünschte Länge erreicht ist. Die fehlenden 64 Bits werden mit der Länge der Eingabe aufgefüllt. Somit ist die Länge der Nachricht ein exaktes Vielfaches von 512. [ACRY 1996, Kapitel 18.5]

Zur weiteren Verarbeitung werden vier 32-Bit-Variablen, auch Kettenvariablen genannt, initialisiert:

$$A = 0x01234567$$
$$B = 0x89abcdef$$
$$C = 0xfedcba98$$
$$D = 0x76543210$$

Diese vier Variablen werden in die Variablen a, b, c und d kopiert. Daraufhin wird eine Schleife gestartet, die für jeden 512-Bit-Block der Eingabe durchläuft. Jeder Durchlauf besitzt vier Runden, wobei jede Runde 16 Operationen ausführt. [ACRY 1996, Kapitel 18.5]

Für jeden Durchlauf wird der 512-Bit-Block nochmals in 16 32-Bit-Blöcke aufgeteilt. Daraufhin werden drei der oben genannten vier Variablen miteinander verrechnet und das Ergebnis wird zur vierten Variable, zusammen mit einem der 16 Blöcke und einer Konstanten, hinzu addiert. Zuletzt wird das Ergebnis um eine definierte Größe rotiert und zu einer der 4 Variablen summiert. Dieses Ergebnis ersetzt eine der anderen Variablen. Zum Ende einer Runde werden die Variablen a, b, c, und d den Variablen A, B, C und D hinzu addiert, woraufhin der nächste 512-Bit-Block verarbeitet werden kann. Das Ergebnis des Algorithmus

wird aus der Konkatenation der vier Variablen A, B, C und D zusammen gesetzt. [ACRY 1996, Kapitel 18.5]

2.4.2 SHA-1

Der Secure Hash Algorithm (SHA) arbeitet ähnlich dem MD5-Algorithmus. Dabei wird, äquivalent zu MD5, ein Padding vorgenommen, so dass die Länge der Nachricht 64 Bits kürzer ist als ein Vielfaches von 512. Diese Bits werden dann mit einer Längenrepräsentation aufgefüllt. [ACRY 1996, Kapitel 18.7]
Daraufhin werden fünf Variablen initialisiert:

$$A = 0x67452301$$
$$B = 0xefcdab89$$
$$C = 0x98badcfe$$
$$D = 0x10325476$$
$$E = 0xc3d2e1f0$$

Nach der Initialisierung wird eine Schleife gestartet, die solange durchlaufen wird, bis kein 512-Bit-Block mehr vorhanden ist. Die fünf Variablen werden in die Variablen a, b, c, d und e kopiert. Die Schleife besteht aus vier Runden, jede Runde führt 20 Operationen aus. Es werden drei der oben genannten fünf Variablen miteinander verrechnet und das Ergebnis wird zu einer vierten Variable, zusammen mit dem nächsten der 16 Blöcke und einer Konstanten, hinzu addiert. [ACRY 1996, Kapitel 18.7]
Zuletzt werden Shift-Operationen vorgenommen. Bei der Shift-Operation wird der 512-Bit-Block transformiert, so dass aus 16 32-Bit-Blöcken 80 32-Bit-Datenwörter entstehen. Abschließend werden die Variablen a, b, c, d und e zu den jeweiligen fünf Variablen A, B, C, D und E summiert. Das Ergebnis ist die Konkatenation der fünf Variablen als 160 Bit Block. [ACRY 1996, Kapitel 18.7]
Auch der SHA-1 Algorithmus ist aufgrund seiner Ausgabelänge von 160 Bit nicht mehr sicher, da eine Kollision in durchschnittlich 2^{80} Schritten gefunden werden kann. [CRYE 2010, S. 82]

2.5 Cracking und Cracking-Ansätze

Der Vorgang der Entschlüsselung eines gehashten Passworts nennt sich Cracking. Dabei versucht ein Angreifer mit Hilfe verschiedener Ansätze den Klartext eines gehashten Passworts zu finden. Im Folgenden werden zwei Ansätze beschrieben: Die Brute Force-Attacke und die Dictionary-Attacke.

2.5.1 Brute Force-Attacke

Bei einer Brute Force-Attacke werden alle möglichen annehmbaren Werte eines Passworts getestet. Dabei wird eine Belegung gesetzt, gehasht und mit dem gehashten Passwort verglichen. Dieser Vorgang wird für alle Möglichkeiten eines Passworts bestimmter Länge wiederholt. Brute Force-Attacken bildet die simpelste Form der Entschlüsselung von Passwörtern, da keine Annahmen über mögliche Belegungen getroffen werden. Stattdessen werden alle Möglichkeiten durchprobiert.
Unter der Annahme, dass ein genutzter Hashing-Algorithmus so sicher ist, dass ein entsprechend gehashtes Passwort nur mit Brute Force gecrackt werden kann, werden 2^n Versuche zur Entschlüsselung benötigt. Dabei steht n für die Länge der des gehashten Passworts in Bit. Bei einem 64-Bit-Hashwert würde es 600.000 Jahre dauern ein Passwort zu entschlüsseln, wenn 1 Million Hashwerte pro Sekunde erzeugt und geprüft werden können. [ACRY 1996, Kapitel 7.4]

2.5.2 Dictionary-Attacke

Bei einer Dictionary-Attacke, auch Wörterbuch-Angriff oder Wortlisten-Attacke genannt, handelt es sich um eine angepasste Variante der Brute Force-Attacke. Dabei werden die „best guesses“ zuerst ausprobiert, anstatt alle möglichen Belegungen in numerischer Reihenfolge auszuprobieren. Der Angreifer nutzt dazu eine Liste der wahrscheinlichsten Wörter für das Passwort. Diese Liste enthält allgemeine und auf das Passwort zugeschnittene Wörter. Es können der Benutzername, Initialen des Opfers oder andere relevante Informationen über den Account aufgeführt sein. Allgemeine Einträge können Länder oder Tiernamen enthalten, auch Berufe oder populäre Aktivitäten oder Personen sind denkbar. Zusätzlich zum einfachen Testen der Wörter in der Wortliste können abweichende Formen eingesetzt werden. So kann ein Benutzername oder Tiername mit Zahlen oder anderen Informationen kombiniert werden, um eine größere Menge an Passwörtern abdecken zu können. Bei diesen Zusatzinformationen muss, wie bei den Passwörtern selbst, auf Groß- und Kleinschreibung geachtet werden.
Es gibt eine große Menge an möglichen Regeln zur Erweiterung der Passwortmenge, wobei auch kultur- und länderspezifische Eigenschaften bedacht werden können.
Eine Wörterbuch-Attacke eignet sich zur Entschlüsselung einer großen Menge von Passwörtern, da die Wahrscheinlichkeit größer ist auf schwache Passwörter zu treffen. [ACRY 1996, Kapitel 8.1]

2.6 Passwort-Cracking-Tools

In diesem Kapitel werden zwei Password-Cracking-Tools vorgestellt: John the Ripper und Hashcat. Zuvor wird der Unterschied zwischen Online- und Offline-Cracking-Tools beschrieben.

2.6.1 Online- und Offline-Cracking

Beim jedem Versuch ein Passwort zu entschlüsseln handelt es sich um einen Offline- oder Online-Angriff. Offline-Angriffe können ausgeführt werden, wenn Zugriff auf die Hashwerte der zu entschlüsselnden Passwörter besteht. Sollte dies nicht der Fall sein, muss eine Online-Attacke durchgeführt werden.
Bei der Ausführung eines Online-Angriffs wird versucht das Passwort zu einem bekannten Nutzernamen oder eine Kombination aus gültigem Nutzernamen und dem zugehörigen Passwort zu ermitteln. Solche Angriffe werden auf Loginformularen ausgeführt, beispielsweise auf einer Webseite oder einer Client-Software.
Diese Vorgehensweise dauert sehr lange, da Loginformulare Sperren für zu viele fehlgeschlagene Loginversuche verhängen können. Außerdem hängt die Geschwindigkeit bei Online-Angriffen von der Schnelligkeit der Internetverbindung ab. Erschwerend kommt hinzu, dass zu viele Loginversuche Auswirkung auf die Größe der Logdateien eines Webseiten-Betreibers haben und somit bemerkt werden können. Beispiele für Online-Cracking-Tools sind „Brutus“ und „THC Hydra“.
Bei einem Offline-Angriff besteht Zugriff auf die Passwortdatei. Der Angriff kann somit auf dem eigenen System ausgeführt werden. Die Geschwindigkeit bei einem Offline-Angriff wird nur durch die vorhandene Hardware begrenzt. John the Ripper und Hashcat sind Offline-Cracking-Tools. [BRUT 2014]

2.6.2 John the Ripper

John the Ripper ist ein Open Source Offline-Passwort-Cracker, der ursprünglich für auf Unix basierende Systeme gebaut wurde. Zum heutigen Zeitpunkt ist es auf mindestens zehn unterschiedlichen Betriebssystemen lauffähig.
Die Hauptfunktionalität von John the Ripper umfasst das Cracken von Passwörtern durch Brute Force- und Wörterbuch-Attacken. Das Hauptaugenmerk wurde auf Performanz gerichtet. John the Ripper unterstützt unter anderem DES-basierte Hashtypen, BSD-basierte Hashtypen sowie Kerberos und Windows-LM Hashtypen. Bei der Benutzung von John the Ripper können unterschiedliche Modi ausgewählt werden um ein Passwort zu entschlüsseln:

1. Wordlist-Mode: Beim Wordlist-Mode handelt es sich um einen Wörterbuch-Angriff. In diesem Modus können unter anderem Word-Mangling Regeln spezifiziert werden, die definierte Muster an Wörtern aus der Wirtliste produzieren und Diese überprüfen.
2. Single-Mode: Beim Single-Mode wird versucht das Passwort eines Benutzers zu entschlüsseln. Es werden spezielle, dem Benutzer zugeordnete, Informationen

genutzt. Diese umfassen den Benutzernamen selbst oder das Heim-Verzeichnis. Diese Informationen werden zusätzlich mit Word-Mangling Regeln versehen, so dass eine große Menge an Passwörtern geprüft werden kann.

3. Incremental-Mode: Beim Incremental-Mode handelt es sich um einen einfachen Brute Force-Angriff.

Zusätzlich zu diesen Modi ist es möglich weitere Optionen anzugeben, welche einen Hashing-Algorithmus spezifizieren oder die Word-Mangling Regeln beeinflussen. Ebenfalls kann John the Ripper parallelisiert eingesetzt werden, so dass eine effiziente Nutzung aller zur Verfügung stehenden Ressourcen stattfinden kann. [JDOC 2014]

2.6.3 Hashcat

Laut dem offiziellen Wiki ist Hashcat der schnellste CPU-basierte Passwort Cracker auf dem Markt. [HDOC 2014]

Hashcat bietet sechs unterschiedliche Angriffs-Modi an und unterstützt weit über 25 verschiedene Hashtypen. Es können beliebig komplexe Befehle abgesetzt werden, die Einfluss auf den Cracking-Prozess haben. Die Optionen umfassen die Nutzung von Wortlisten versehen mit Word-Mangling Regeln, die Spezifizierung der zur Verfügung stehenden Ressourcen, wie beispielsweise die Nummer der zu startenden Threads oder die Benutzung bestimmter Zeichensätze.

Bei der Benutzung von Hashcat ist die strenge Einhaltung der benötigten Formatierung vorausgesetzt, ohne die eine korrekte Verarbeitung nicht möglich ist. Diese Formatierung variiert dabei mit der Verarbeitung des Saltwerts im Passwort. Hashcat ist parallelisiert einsetzbar und deshalb ideal zum Cracken von Passwörtern geeignet.

Es ist wichtig anzumerken, dass unterschiedliche Versionen von Hashcat existieren, jede mit seiner eigenen Existenzberechtigung: so wird beispielsweise oclHashcat zum GPU-basierten Cracken benutzt. In dieser Arbeit wird das ursprüngliche Hashcat in der Version 0.47 eingesetzt. [HDOC 2014]

3 Anforderungsanalyse

Um die Anforderungen an das Endprodukt zu erfüllen, muss eine Anforderungsanalyse durchgeführt werden.
Dazu werden die Anforderungen an das System beschrieben. Ebenfalls müssen die funktionalen und nichtfunktionalen Anforderungen und die daraus resultierenden Anwendungsfälle festgehalten werden.
Die Anforderungen sind vollständig, identifizierbar, konsistent und einheitlich dokumentiert.
Das Ergebnis der Anforderungsanalyse kann als Grundstein für ein Lastenheft betrachtet werden.

Anforderungen an die Anwendung
Die Grundlage der Anwendung ist die Integration und Nutzung von Passwort-Cracking-Tools. Es soll dabei möglich sein Passwort-Cracking-Tools so einzubinden, dass sämtliche Funktionen benutzbar werden. Die Nutzung von Funktionen der Passwort-Cracking-Tools soll auch möglich sein, wenn der Benutzer kein Wissen über die Vorgehensweise oder die Syntax besitzt. Deshalb muss die Anwendung die Bildung und Ausführung von Funktionen übernehmen. Der Benutzer soll lediglich die gewünschte Funktion sowie die zur Ausführung benötigten Parameter angeben. Die Ausführung durch die Anwendung muss die gleichen Ergebnisse erzeugen wie eine direkte Ausführung durch das Passwort-Cracking-Tool. Auch das Starten mehrerer Instanzen eines oder mehrerer Passwort-Cracking-Tools zum Entschlüsseln von Passwörtern soll möglich sein.
Um dem Benutzer weitere Hilfestellung zu bieten soll es möglich sein, Klartext-Passwörter gemäß den Angaben zu verschlüsseln. So erzeugte Dateien sollen an ein Passwort-Cracking-Tool weiter gegeben werden können. Das Format der vom Benutzer spezifizierten Datei darf nicht komplex sein, da kein Fachwissen vorausgesetzt wird. Um die Ausgabedatei verarbeiten zu können, muss die Anwendung die Formatierung entsprechend der Voraussetzungen des benutzten Passwort-Cracking-Tools handhaben.
Zusätzlich soll es möglich sein die verschlüsselten Passwörter einer spezifizierten Datei zu identifizieren, so dass die genutzten Hashing-Verfahren erkennbar werden.
Da mehrere Cracking-Vorgänge gestartet werden können, ist es erforderlich eine Verwaltung aller aktiven Prozesse vorzunehmen. Alle gestarteten Prozesse sollen klar voneinander unterscheidbar sein. Die Verwaltung beinhaltet auch, dass Prozesse auf Wunsch beendet werden können.

Aus dieser Beschreibung ergeben sich folgende funktionale und nichtfunktionale Anforderungen:

3.1 Funktionale und nichtfunktionale Anforderungen

3.1.1 Funktionale Anforderungen

Anforderungen an die Anwendung:

A1.1 Der Benutzer kann eine Passwortdatei im Format „Username:Password“ angeben. Diese Datei kann von der Anwendung in ein Unix-verständliches Format umgewandelt werden.

A1.1.a Bei der Umwandlung einer Datei kann der gewünschte Hashing-Algorithmus angegeben werden.

A1.1.b Bei der Umwandlung einer Datei kann angegeben werden, ob ein Saltwert erzeugt und benutzt werden soll.

A1.1.c Die Ausgabedatei befindet sich im Format „Username:Password:UID:GID:Info:HomeDir:Shell“.

A1.2 Der Benutzer kann eine Passwortdatei im Format „Username:Password:UID:GID:Info:HomeDir:Shell“ angeben. Die Anwendung kann versuchen alle benutzten Hashing-Algorithmen der Datei zu erkennen.

A1.2.a Das Ergebnis der Erkennung wird in aufbereiteter Form angezeigt.

A1.3 Der Benutzer kann zwei Grundfunktionalitäten eingebundener Passwort-Cracking-Tools nutzen. Es kann der Brute Force-Modus sowie der Wortlisten-Modus des gewünschten Tools genutzt werden.

A1.3.a Das Ergebnis wird in einer Logdatei gespeichert.

A1.3.b In Bezug auf die Funktionsweise des Passwort-Cracking-Tools darf keinerlei Kenntnis des Benutzers vorausgesetzt werden.

A1.4 Passwort-Cracking-Tools können zur Verwendung durch die Anwendung bereitgestellt werden.

A1.5 Sollte der Benutzer falsche oder nicht verständliche Parameter übergeben, wird eine entsprechende Fehlermeldung erzeugt.

A1.6 Der Benutzer kann Befehle an eingebundene Passwort-Cracking-Tools absetzen.

A1.6.a Alle Befehle dürfen abgesetzt werden.

A1.6.b Sollte der Befehl nicht verstanden werden, muss eine entsprechende Fehlermeldung erzeugt werden.

A1.6.c Das Ergebnis und die Logdateien müssen gespeichert werden.

A1.7 Der Benutzer hat die Möglichkeit Speicherort der Logdateien und Speicherort der Ergebnisse festzulegen.

A1.8 Der Benutzer hat für alle Operationen und den daraus resultierenden Prozessen die Möglichkeit alle laufenden Prozesse zu beenden.

A1.9 Es ist möglich Informationen zu laufenden Prozessen von eingebundenen Passwort-Cracking-Tools zu erhalten.

A1.10 Jeder laufende Prozess muss eindeutig identifizierbar sein.

A1.11 Der Benutzer kann eine erneute Registrierung aller bereits eingebundenen Passwort-Cracking-Tools fordern.

A1.11.a Angesammelte Daten zur Verwaltung der registrierten Tools werden gelöscht.

A1.12 Der Benutzer kann einsehen welche Passwort-Cracking-Tools eingebunden und lauffähig sind.

A1.13 Bei der Kommunikation der Anwendung mit Passwort-Cracking-Tools müssen alle notwendigen Informationen zur korrekten Ausführung mitgeliefert werden.

A1.14 Bevor ein Passwort-Cracking-Tool genutzt werden kann, muss eine Registrierung erfolgen.

A1.15 Die Registrierung eines Passwort-Cracking-Tools erfolgt durch ein wohldefiniertes Protokoll.

A1.16 Jedes eingebundene Passwort-Cracking-Tool übernimmt eine Verwaltung seiner aktiven Prozesse.

A1.17 Jedes eingebundene Passwort-Cracking-Tool besitzt Informationen über sich selbst.

A1.18 Jedes eingebundene Passwort-Cracking-Tool kann feststellen, welche eigenen Prozesse aktuell arbeiten.

A1.19 Jedes eingebundene Passwort-Cracking-Tool kann Statusinformationen zu eigenen Prozessen abfragen.

A1.20 Die Statusinformationen in Anforderung A1.20 haben ein bestimmtes Format.

A1.21 Die Kommunikation zwischen der Anwendung und Passwort-Cracking-Tools darf nicht blockiert werden und muss jederzeit stattfinden können.

A1.22 Der Benutzer muss eine Nachricht über Erfolg oder Misserfolg erhalten, wenn er einen Dienst eines Passwort-Cracking-Tools anfordert.

A1.23 Sollten aus bestimmten Operationen Dateien entstehen, dürfen existierende Dateien nicht überschreiben werden.

A1.24 Sollte die Verarbeitung einer Datei nicht erfolgen können, muss eine entsprechende Fehlermeldung angezeigt werden. Die Ausführung der Anwendung darf nicht betroffen werden.

A1.25 Falscheingaben des Benutzers dürfen nicht zum Absturz der Anwendung führen.

A1.26 Bei Terminierung der Anwendung werden alle laufenden Prozesse beendet.

A1.27 Beim erneuten Starten der Anwendung müssen sich alle Passwort-Cracking-Tools anmelden.

3.1.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen an die Anwendung

A2.1 Die Kommunikation zwischen Anwendung und eingebundenen Passwort-Cracking-Tools ist nicht verschlüsselt.

A2.2 Die Anwendung hat für alle Fehlerfälle eine entsprechende Fehlermeldung. Verschiedene Fehler können die gleiche Fehlermeldung erzeugen.

A2.3 Die Kommunikation zwischen Anwendungskern und eingebundenen Passwort-Cracking-Tools soll minimal gekoppelt sein.

A2.4 Die Architektur soll erweiterbar und flexibel sein, so dass möglichst wenige Einschränkungen für zukünftige Erweiterungen entstehen.

A2.5 Die Anwendung soll zu jeder Zeit stabil laufen und nicht abstürzen.

A2.6 Die Implementierung weiterer Grundfunktionalitäten soll möglichst einfach erfolgen.

A2.7 Die Anzeige und Verarbeitung von Daten in der GUI muss Threadsafe sein. Es darf zu keinen Abstürzen oder Fehlern kommen.

3.1.3 Abgrenzungen

A3.1 Es wird auf Linux Ubuntu 12.04 LTS und Java 1.7 entwickelt. Nutzung auf anderen Plattformen oder Veröffentlichungen kann zu Fehlern oder unerwartetem Verhalten führen

A3.2 Die Server-Umgebung, in welcher die Anwendung läuft, ist bereits implementiert und funktionsfähig.

A3.3 Die Anwendung wird mit der in Anforderung A1.1 und A1.2 beschriebenen Funktionalität ausgeliefert. Zusätzlich werden John the Ripper und Hashcat als Passwort-Cracking-Tools integriert sein. Dies erlaubt die Nutzung der in A1.3 beschriebenen Funktionalität. Über weitere Funktionalität wird keine Aussage getroffen.

3.2 Anwendungsfälle

Anwendungsfall AF1:	**Benutzer startet die Anwendung und möchte ein Passwort entschlüsseln**
Akteur:	Angestellter
Auslöser:	Benutzer möchte eine Passwortdatei cracken
Vorbedingungen:	Die Anwendung ist installiert. Mindestens ein Passwort-Cracking-Tool hat eine Registrierungsanfrage versendet und ist bereit zur Nutzung
Nachbedingungen:	Das Passwort-Cracking-Tool versucht das Passwort zu entschlüsseln

Erfolgsszenario:

1. Der Benutzer startet die Anwendung.
2. Der Benutzer klickt den Menüpunkt „Refresh GUI Plugins“.
3. Der Benutzer klickt auf den Reiter mit dem Namen des gewünschten Passwort-Cracking-Tools.
4. Der Benutzer muss festlegen an welchem Ort die Ergebnisse gespeichert werden sollen, das Arbeitsverzeichnis festlegen in dem die relevanten Daten liegen und das Zeitintervall für Statusanfragen an laufende Prozesse angeben.
5. Der Benutzer kann einen verständlichen Befehl absetzen und erhält das Ergebnis.

Fehlerfälle:

a. Kein Passwort-Cracking-Tool ist eingebunden. Ein Passwort-Cracking-Tool muss eingebunden werden.

b. Der Benutzer klickt nicht den Menüpunkt „Refresh GUI Plugins“. Weiter bei Punkt 2.
c. Der Benutzer gibt nicht alle notwendigen Parameter an. Weiter bei Punkt 4.
d. Der Benutzer setzt einen falschen oder nicht verständlichen Befehl ab. Weiter bei Punkt 5.
e. Der Benutzer besitzt nicht die Rechte eine Datei zu erstellen. Der Benutzer muss Rechte zur Erstellung einer Datei anfordern und zugeteilt bekommen.
f. Das eingebundene Passwort-Cracking-Tool ist nicht mehr vorhanden oder funktionsfähig. Das Passwort-Cracking-Tool muss neu gestartet werden.

Anforderungen: [1.4, 1.5, 1.6, 1.6.a, 1.6.b, 1.6.c, 1.12, 1.13, 1.21, 1.22, 1.24, 1.25]

Anwendungsfall AF2:	**Ein Benutzer möchte eine Passwortdatei erstellen**
Akteur:	Angestellter
Auslöser:	Der Benutzer braucht eine Passwortdatei im entsprechenden Format zur Weiterbearbeitung
Vorbedingungen:	Die Passwortdatei hat die richtige Formatierung und kann gelesen werden
Nachbedingungen:	Eine neue Datei in einem Unix verständlichen Format wurde erstellt

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Password File Management“.
2. Der Benutzer gibt eine Datei an, die sich in einem geeigneten Format befindet. In dieser Datei befinden sich die „Username:Password“-Einträge.
3. Der Benutzer gibt den Hashing-Algorithmus an. Außerdem kann er entscheiden, ob ein Saltwert genutzt werden soll. Zusätzlich bestimmt er einen Speicherort für die Ergebnisdatei.
4. Der Benutzer klickt auf den Menüpunkt „Generate File“.
5. Das System erstellt eine Passwortdatei im Unix verständlichen Format.

Fehlerfälle:

1. Die vom Benutzer gestellte Passwortdatei hat ein falsches oder nicht verständliches Format. Das korrekte Format muss eingehalten werden.
2. Der Benutzer gibt keinen gewünschten Hashing-Algorithmus und/oder einen Speicherort an. Weiter bei Punkt 3.
3. Der Benutzer besitzt nicht die Rechte eine Datei zu erstellen. Der Benutzer muss Rechte zur Erstellung einer Datei anfordern und zugeteilt bekommen.

Anforderungen:[1.1, 1.1a, 1.1.b, 1.1.c, 1.5, 1.7, 1.23, 1.24, 1.25]

Anwendungsfall AF3:	**Ein Benutzer möchte Informationen über eine bestimme Passwortdatei erfahren**
Akteur:	Angestellter
Auslöser:	Der Benutzer möchte wissen, welche Hashing-Algorithmen in einer bestimmten Datei benutzt wurden
Vorbedingungen:	Die gestellte Datei hat ein verständliches Format
Nachbedingungen	Informationen über die Datei werden angezeigt

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Password File Management“.
2. Der Benutzer gibt eine Datei im verständlichen Format an.
3. Der Benutzer klickt auf den Menüpunkt „Identify“.
4. Das System zeigt Informationen zu genutzten Hashing-Algorithmen und der Menge der entsprechend gehashten Passwörter an. Es ist möglich, dass keine relevanten Informationen extrahiert werden.

Fehlerfälle:

a. Die gestellte Datei hat ein falsches oder nicht verständliches Format oder enthält keine gehashten Passwörter. Das korrekte Format muss eingehalten werden.

Anforderungen:[1.2, 1.2.a, 1.5, 1.7, 1.23, 1.24, 1.25]

Anwendungsfall AF4:	**Ein Benutzer möchte eine Passwortdatei entschlüsseln, besitzt aber kein Fachwissen (Nutzung von Click and Fire Operationen)**
Akteur:	Angestellter
Auslöser:	Der Benutzer hat keine Fachkenntnis, möchte aber die Passwörter einer Datei entschlüsseln
Vorbedingungen:	Die Passwortdatei hat ein Unix verständliches Format
Nachbedingungen:	Der Benutzer erhält die gleichen Daten wie bei der Nutzung des Passwort-Cracking-Tools

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Click and Fire“.
2. Der Benutzer wählt einen von zwei möglichen Modi aus.
   2.1. „Brute Force-Mode“:
   Der Benutzer gibt eine Passwortdatei an sowie welches Passwort-Cracking-Tool den Dienst ausführen soll. Der Benutzer klickt auf den Menüpunkt „Fire“.
   Hier wird der Brute Force-Modus des gewünschten Passwort-Cracking-Tools ausgeführt.
   2.2. „Wordlist-Mode“:
   Der Benutzer gibt eine Passwortdatei und eine Wortliste an. Zusätzlich gibt er an, welches Passwort-Cracking-Tool den Dienst ausführen soll.
   Der Benutzer klickt den Menüpunkt „Fire“. Hier wird der Wortlisten-Modus des gewünschten Passwort-Cracking-Tools ausgeführt.
3. Das System zeigt die Ergebnisse in der GUI an und stellt sie als Datei zur Verfügung. Die Ergebnisse der Operation stimmen mit den Ergebnissen einer direkten Nutzung des entsprechenden Passwort-Cracking-Tools überein.
4. Nach Beendigung des Durchlaufs und währenddessen werden Informationen über gecrackte Passwörter angezeigt.

Fehlerfälle:

a. Die vom Benutzer gestellten Dateien haben ein falsches oder nicht verständliches Format. Das korrekte Format muss eingehalten werden.
b. Das eingebundene Passwort-Cracking-Tool ist nicht mehr vorhanden oder kann den Dienst nicht ausführen. Das Passwort-Cracking-Tool muss neu gestartet werden.
c. Der Benutzer hat nicht das Recht Dateien zu erstellen. Der Benutzer muss Rechte zur Erstellung einer Datei anfordern und zugeteilt bekommen.

Anforderungen:[1.3, 1.3.a, 1.3.b, 1.4, 1.5, 1.8, 1.13, 1.21, 1.22, 1.23, 1.24, 1.25]

Anwendungsfall AF5:	**Ein Benutzer möchte alle laufenden Prozesse (zuvor angeforderter Dienste) beenden**
Akteur:	Angestellter
Auslöser:	Der Benutzer möchte angeforderte Dienste vorzeitig beenden
Vorbedingungen:	Es existieren aktive Prozesse. Der Benutzer befindet sich im entsprechenden Reiter des Passwort-Cracking-Tools
Nachbedingungen:	Alle aktiven Prozesse sind beendet

Erfolgsszenario:

1. Der Benutzer klickt auf den Menüpunkt „Kill all Processes“.
2. Das System versendet Nachrichten zum Beenden aller aktiven Prozesse an alle eingebundenen Passwort-Cracking-Tools. Alle Prozesse werden kontrolliert heruntergefahren.
3. Der Benutzer erhält eine Nachricht über den Erfolg der Operation.

Fehlerfälle:

a. Die Prozesse konnten nicht beendet werden. Das eingebundene Passwort-Cracking-Tool ist möglicherweise fehlerhaft Implementiert oder Nachrichten können nicht mehr empfangen werden. Implementierung korrigieren oder eingebundenes Passwort-Cracking-Tool neu starten.
b. Das eingebundene Passwort-Cracking-Tool ist nicht mehr vorhanden oder kann nicht Antworten. Das Passwort-Cracking-Tool muss neu gestartet werden.

Anforderungen:[1.8, 1.9, 1.10, 1.13, 1.16, 1.18, 1.21]

Anwendungsfall AF6:	**Ein Benutzer möchte Informationen über laufende Prozesse abfragen**
Akteur:	Angestellter
Vorbedingungen:	Es sind aktive Prozesse vorhanden
Nachbedingungen:	Der Benutzer hat Wissen über Anzahl und bisherige Laufzeit der aktiven Prozesse

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Process Management“.
2. Das System zeigt eine Liste aller aktiven Prozesse an.
3. Das System stellt die Informationen in geeigneter Form dar.

Fehlerfälle:

a. Der Kommunikationskanal ist nicht mehr aktiv. Die Anwendung muss neu gestartet werden.

Anforderungen:[1.9, 1.10, 1.13, 1.16, 1.18, 1.19, 1.21]

Anwendungsfall AF7:	**Ein Benutzer möchte überprüfen, welche Passwort-Cracking-Tools noch lauffähig sind**
Akteur:	Angestellter
Vorbedingungen:	Es ist mindestens ein Passwort-Cracking-Tool registriert und mindestens einmal in die GUI geladen worden
Nachbedingungen:	-

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Test Plugins“.
2. Es öffnet sich ein Dialog-Fenster, in welchem die bisher eingebundenen Passwort-Cracking-Tools aufgelistet sind.
3. Das System sendet automatisch eine Anfrage an jedes eingebundene Passwort-Cracking-Tool.
4. Das System wartet maximal 10 Sekunden auf eine Antwort.
5. Das System signalisiert für jedes Passwort-Cracking-Tool, ob eine Nachricht erhalten wurde.

Fehlerfälle:

a. Es ist noch kein Passwort-Cracking-Tool registriert. Ein Passwort-Cracking-Tool muss eingebunden werden.
b. Das System kann keine Nachrichten versenden. Anwendung muss neu gestartet werden.
c. Der Kommunikationskanal ist nicht mehr vorhanden. Die Anwendung muss neu gestartet werden.
d. Das eingebundene Passwort-Cracking-Tool kann keine Nachrichten senden oder empfangen. Das Passwort-Cracking-Tool muss neu gestartet werden.

Anforderungen:[1.11, 1.11.a, 1.12, 1.13, 1.21, 1.27]

4 Entwurf

In diesem Kapitel wird die Umsetzung der Anforderungsanalyse beschrieben.
Zunächst wird eine Architekturauswahl getroffen, woraufhin die allgemeinen Entwurfselemente beschrieben werden. Daraufhin werden die Teilkomponenten des Entwurfs genauer betrachtet, um so ein tieferes Verständnis von der Funktionsweise der kompletten Anwendung zu erhalten.
Die Teilkomponenten des Entwurfs beziehen sich auf den Anwendungskern, die eingebundenen Passwort-Cracking-Tools, die öffentlichen Klassen sowie die Kommunikation zwischen Anwendungskern und eingebundenen Tools.

4.1 Auswahl der Architektur

Die wichtigste Anforderung an die Anwendung ist die Integration von Passwort-Cracking-Tools. Das Hauptaugenmerk dieser Anforderung liegt auf der Integration aller Passwort-Cracking-Tools. Zudem ist es notwendig die Funktionstüchtigkeit der Anwendung nicht durch neue Verfahren oder Cracking-Software einzuschränken, so dass eine langfristige Nutzung möglich wird.
Die Grundlage soll ein unabhängiger Anwendungskern sein, der durch die Integration von Passwort-Cracking-Tools nicht beeinflusst wird. Der Anwendungskern sollte hauptsächlich koordinative Aufgaben übernehmen, wobei die integrierten Passwort-Cracking-Tools sämtliche Informationen bereitstellen müssen, um für einen korrekten Ablauf zu garantieren. Zu diesem Zweck muss der Anwendungskern eine Verwaltung aller Informationen vornehmen, welche von den integrierten Passwort-Cracking-Tools verlangt werden. Mit Hilfe der gespeicherten Informationen soll es möglich sein, einen vom Benutzer angefragten Dienst in ein Kommando umzuwandeln. Das Kommando soll vom gewünschten Passwort-Cracking-Tool verstanden und ausgeführt werden. Die Umwandlung des Kommandos erfolgt mit den spezifizierten Parametern des Benutzers.
Damit ein Passwort-Cracking-Tool ein Kommando erhalten und verarbeiten kann, wird es in eine Softwarekomponente gekapselt. Diese Komponente übernimmt sämtliche Aufgaben außer die konkrete Ausführung des Prozesses.
Die Kommunikation zwischen dem Anwendungskern und des gekapselten Passwort-Cracking-Tools soll lose gekoppelt sein. So kann sichergestellt werden, dass kein Anwendungskern-Internes Wissen nach Außen bekannt sein muss. Ideal für diesen Zweck ist die Kommunikation mittels Nachrichten, wodurch auch eine asynchrone Verarbeitung möglich wird.

Sobald ein Benutzer einen Dienst eines bestimmten Password-Cracking-Tools benutzen möchte, gibt er die dafür erforderlichen Parameter an. Mit Hilfe dieser Parameter und den im Anwendungskern gespeicherten Informationen wird ein Kommando erzeugt. Dieses Kommando wird an die Softwarekomponente des gewünschten Passwort-Cracking-Tools versendet. Die Komponente übernimmt die Delegierung des Kommandos an das Passwort-Cracking-Tool.
Die beschriebene Vorgehensweise und Handhabung ist typisch für eine Framework Architektur. Der Anwendungskern ist die Framework Komponente. Die Softwarekomponenten, welche die Passwort-Cracking-Tools kapseln, sind die angebundenen Plugins. Die Kommunikation findet über eine Message Queue statt.
Eine Framework Architektur erfüllt somit alle beschriebenen Anforderungen und soll als Basis für den Entwurf und die Implementierung der Anwendung dienen.

4.2 Allgemeine Architektur

Die Hauptdesign-Entscheidungen der Architektur basieren auf der Nutzung als Framework. Dabei residieren sowohl der Anwendungskern als auch die Plugins auf demselben System.
Der Anwendungskern agiert als Schaltzentrale mit erweitertem Fachwissen. Plugins dienen der Ausführung von Befehlen. Die Kommunikation zwischen dem Anwendungskern und seinen Plugins findet über eine Message Queue statt.
Abgesetzte Befehle werden in ein Kommando Objekt verpackt und serialisiert. Daraufhin wird das Objekt vom Kommunikationsmodul des Anwendungskerns über eine Message Queue versendet.
Das entsprechende Plugin verarbeitet den Befehl, damit er ausgeführt werden kann. Das Ergebnis wird ebenfalls in ein Kommando Objekt gekapselt und serialisiert versendet. Der Anwendungskern nimmt das Ergebnis in Empfang, verarbeitet es und zeigt es in der GUI an.
Des Weiteren besitzt der Anwendungskern Funktionalitäten, welche ohne die Integration eines Plugins benutzt werden können. Zur Ausführung dieser Funktionalitäten existieren entsprechende Komponenten. Die Abbildung 4.1 beschreibt die Kontextsicht der Anwendung sowie die Kommunikation zwischen Anwendungskern und Plugins.

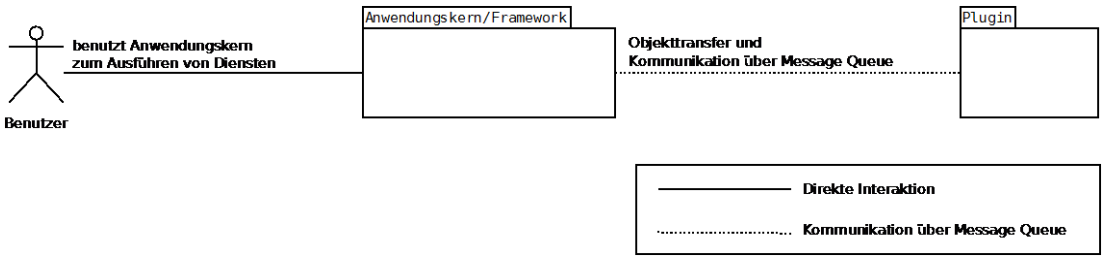

Abbildung 4.1: Kontextsicht der Anwendung

4.3 Entwurf Anwendungskern

Minimalistisch betrachtet, ist der Anwendungskern eine Schaltzentrale mit erweitertem Fachwissen. Die Hauptaufgabe liegt in der Delegierung von Befehlen. Zusätzlich können Operationen zur Erstellung von Passwortdateien oder zur Weitergabe bestimmter Befehle genutzt werden. Und es können Informationen zu benutzten Hashing-Algorithmen einer Datei abgefragt werden.
Der Anwendungskern wird dabei von der GUI gesteuert und nimmt Befehle entgegen.

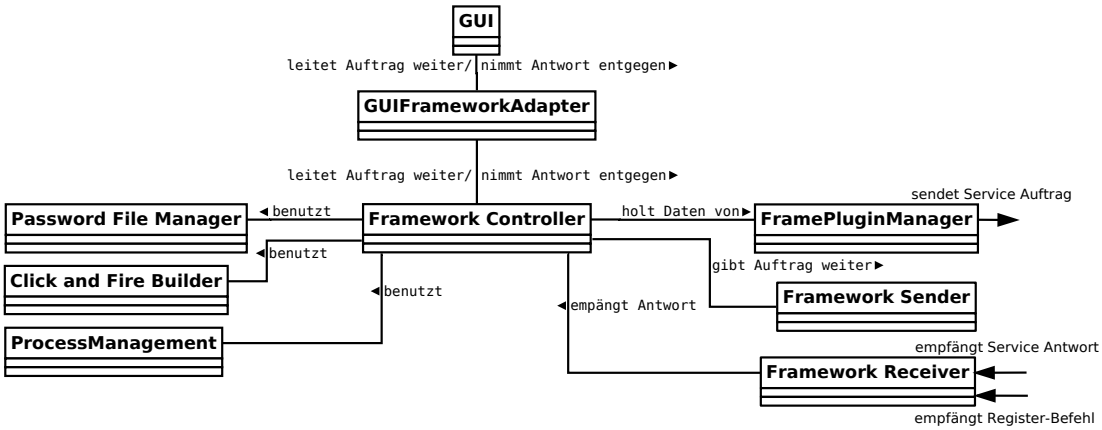

Abbildung 4.2: Fachliches Datenmodell des Anwendungskerns

Die Abbildung 4.2 ist eine unvollständige Darstellung des fachlichen Datenmodells des Anwendungskerns. Eine vollständige Version kann im Anhang eingesehen werden. Basierend auf diesem Modell folgt nun die Beschreibung der einzelnen Komponenten.

GUIFrameworkAdapter:
Der GUIFrameworkadapter wird von der GUI initialisiert. Der Haupteinstiegspunkt der Anwendung ist somit die GUI. Die Hauptaufgabe während der Laufzeit ist die Kontrolle der Kommunikation zwischen der GUI und dem Anwendungskern.
Der Anwendungskern wird durch die Erstellung eines Framework Controllers initialisiert, woraufhin Befehle entgegen genommen und verarbeitet werden können.

Process Management:
Das Process Management wird vom Framework Controller initialisiert. Es übernimmt die Verwaltung aller aktiven Prozesse. Zur Verwaltung der aktiven Prozesse benutzt das Process Management eine entsprechende Datenstruktur.
Zur eindeutigen Identifikation eines Prozesses wird eine Kombination aus ID und Namen des Plugins benutzt.
Sobald ein Plugin mit der Ausführung eines Diensts beginnt, wird ein Eintrag über diesen Prozess verfasst. Dieser Eintrag wird an die Datenstruktur des Process Managements gegeben und zusammen mit einer Aufforderung zum Hinzufügen des Prozesses an den

Anwendungskern verschickt. Der Anwendungskern benutzt diesen Eintrag zur Verwaltung und Anzeige aller aktiven Prozesse.
Nach Terminierung des Plugin-Prozesses wird die Referenz des Prozesses zusammen mit einer Aufforderung zum Löschen an den Anwendungskern verschickt. Der Anwendungskern kann nun den Eintrag aus der Datenstruktur löschen und das GUI Element entsprechend neu laden.

Framework Controller:
Der Framework Controller ist die zentrale Komponente des Anwendungskerns. Er übernimmt die Initialisierung der Kommunikationsmodule und des Plugin Managers. Zusätzlich kann er den Password File Manager nutzen, um Operationen auf Passwortdateien auszuführen.
Während der Laufzeit agiert der Framework Controller als Schaltzentrale und koordiniert die angefragten Dienste des Benutzers. Befehle werden vom Framework Controller entgegen genommen, in ein Kommando Objekt verpackt und an das Kommunikationsmodul übergeben.
Antworten auf angefragte Dienste werden ebenfalls als Kommando Objekt entgegengenommen und verarbeitet. Die gelesene Antwort wird an den GUIFrameworkAdapter weiter gegeben.

Zur Verarbeitung einer Passwortdatei hat der Framework Controller Zugriff auf den Password File Manager. Sollte ein entsprechender Befehl abgesetzt werden, nimmt der Controller den Dateipfad der Passwortdatei sowie die gewünschten Parameter zur Verarbeitung entgegen und übergibt die Parameter an den Password File Manager. Der Password File Manager lädt die Datei und führt die gewünschte Operation aus. Dies kann entweder die Formatierung eines Passworts im Klartext oder die Identifizierung benutzter Hashing-Algorithmen bei Passwortdateien mit bereits verschlüsselten Passwörtern sein.
Bei der Formatierung einer Datei wird der Dateipfad der Ergebnisdatei übergeben. Bei einer Identifikation werden die gesammelten Informationen angezeigt.
Die erstellte Datei wird vom Framework Controller mit Hilfe des Dateipfads geöffnet, so dass der Benutzer sie begutachten kann.

Um die Click and Fire Funktionalität entsprechend benutzen zu können, besitzt der Framework Controller eine Referenz auf den Plugin Manager. Die für eine Click and Fire Operation benötigten Parameter (Modus, Passwortdatei, Plugin und eventuelle weitere Parameter) werden vom Benutzer gestellt und vom GUIFrameworkAdapter an den Framework Controller gegeben. Dort stellt der Framework Controller eine Anfrage an den Plugin Manager, um die Syntax des Befehls korrekt bilden zu können. Dieser Befehl wird in ein Kommando Objekt gekapselt und an das Kommunikationsmodul übergeben.

Falls ein Befehl zur Anzeige der bereits registrierten Plugins entgegengenommen wird, stellt der Framework Controller eine entsprechende Anfrage an seinen Plugin Manager. Der Plugin Manager führt eine Liste aller bereits registrierten Plugins, welche an den Controller gegeben

wird. Diese Liste wird vom Framework Controller an den GUIFrameworkAdapter gereicht, so dass die entsprechenden GUI Elemente gebaut und angezeigt werden können. Von diesem Zeitpunkt an können Befehle an Plugins übergeben werden.

Framework Plugin Manager:
Der Framework Plugin Manager ist für die Verwaltung aller registrierten Plugins zuständig. Zu diesem Zweck werden alle Registrierungsnachrichten der Plugins, beim Empfang durch den Framework Receiver, direkt an den Framework Plugin Manager gegeben. Der Framework Plugin Manager hält eine entsprechende Datenstruktur zur Verwaltung.
Der Framework Plugin Manager kann nur Anfragen und Operationen vom Framework Controller entgegen nehmen.
Es ist mit Hilfe des Managers möglich, Anfragen an seine Datenstruktur zu stellen, um so Informationen zu Plugins zu erhalten. So ist es möglich die Syntax der Befehle für eine Operation abzufragen, um den Click and Fire Builder korrekt nutzen zu können. Auch Informationen zu den Plugins, beispielsweise der Name, können auf diese Weise abgefragt werden.
Alle Ergebnisse werden direkt an die aufrufende Instanz des Framework Controllers zurückgegeben.
Das Hinzufügen eines Plugins zum Plugin Manager wird durch die Übergabe eines entsprechenden Plugin Objekts erreicht. Dieses Plugin Objekt ist in der Registrierungsnachricht des Plugins gekapselt.
Der Framework Plugin Manager wird durch den Framework Controller initialisiert.

Framework Sender/Framework Receiver:
Zusammen bilden der Framework Sender und der Framework Receiver das Kommunikationsmodul des Anwendungskerns. Beide Instanzen werden vom Framework Controller initialisiert.
Angefragte Dienste oder Befehle werden vom Framework Controller an den Framework Sender übergeben. Hier wird das Kommando Objekt serialisiert und über eine Message Queue an alle Plugins versendet. Das entsprechende Plugin deserialisert das Objekt und übernimmt die Ausführung des Diensts.
Die Ergebnisse der Ausführung werden ebenfalls serialisiert und an den Framework Receiver gesendet. Der Framework Receiver deserialisiert das empfangene Objekt und leitet es, entsprechend seiner Instanz, an den Framework Controller weiter.
Die vom Anwendungskern empfangenen Kommando Objekte können nur Ergebnisse zu vorher angefragten Diensten enthalten. Aus diesem Grund wird die enthaltene Information direkt an die GUI weitergeleitet.

Beim Empfang eines Registrierungsobjekts extrahiert der Framework Sender das Plugin Objekt. Dieses Objekt wird über den Framework Controller an den Framework Plugin Manager gereicht.

Password File Manager:
Der Password File Manager wird durch den Framework Controller initialisiert und hat zwei Aufgaben:
Die Formatierung einer Passwortdatei in ein UNIX lesbares Format und die Erkennung von benutzten Hashing-Algorithmen einer Passwortdatei.

Zur Nutzung der Formatierung gibt der Benutzer eine Passwortdatei im Format „Username:Password“ an. Außerdem wird der gewünschte Hashing-Algorithmus angegeben, mit dem die Passwörter gehasht werden sollen. Der Benutzer kann angeben, ob das Passwort mit einen Saltwert versehen werden soll. Falls ja, wird der Saltwert für den Benutzer generiert.
Das Ergebnis ist eine von UNIX-Systemen lesbare Datei im Format Username:Password:UID:GID:Info:HomeDir:Shell. Der Dateipfad der Ergebnisdatei wird an den Framework Controller gegeben.

Falls der Benutzer Informationen zu einer bestimmten Passwortdatei möchte, erhält der Password File Manager einen entsprechenden Aufruf durch den Framework Controller.
Der Inhalt der Datei wird zeilenweise begutachtet. Für jedes erkannte Muster wird ein Eintrag angelegt, welcher den benutzten Hashing-Algorithmus beschreibt. Dabei ist es auch möglich, dass keine eindeutige Lösung gefunden werden kann. Für diesen Fall werden entweder keine oder mehrere Ergebnisse angezeigt.
Das Ergebnis enthält neben der Anzahl der gefundenen Hashing-Algorithmen auch die Anzahl der Einträge, die keinem Algorithmus zugeordnet werden konnten.
Ebenfalls werden die Ergebnisse in einer Datei fest gehalten, so dass sie zu einem späteren Zeitpunkt eingesehen werden können. Nach Beenden der Operation wird der Inhalt der Datei an den Framework Controller gegeben, damit der Inhalt in der GUI angezeigt werden kann.

Click and Fire Builder:
Mit Hilfe des Click and Fire Builders ist es möglich bestimmte Dienste anzufragen oder einen einfachen Befehl abzusetzen. Es wird kein Fachwissen zum angefragten Dienst oder seiner Ausführung vorausgesetzt. Die Anwendung geht noch einen Schritt weiter und versteckt die Hintergrundprozesse, so dass ein Benutzer sich lediglich um die Angabe der korrekten Parameter kümmern muss.
Der Click and Fire Builder wird vom Framework Controller initialisiert.
Eine abgesetzte Click and Fire Operation wird über den GUIFrameworkAdapter bis zum Framework Controller gereicht.
Die vom Benutzer angegebenen Parameter (Modus, Passwortdatei, Plugin und eventuelle weitere Dateien) werden genutzt, um eine Anfrage an den Framework Plugin Manager zu stellen.
Der Aufruf an den Framework Plugin Manager gibt als Ergebnis die benötige Syntax zur Ausführung des angefragten Diensts zurück.

Die erhaltene Syntax enthält generische Schlüsselwörter, welche durch die konkreten Parameter ersetzt werden. Dieses Kommando kann nun in ein Kommando Objekt verpackt werden. Das Ergebnis ist ein komplettes Kommando Objekt, welches an das richtige Plugin versendet werden kann. Der Dienst wird normal ausgeführt, das Ergebnis wird angezeigt.
Falls die Click and Fire Operation fehlschlägt oder ein falscher Parameter übergeben worden sein sollte, muss dies durch eine Fehlermeldung kenntlich gemacht werden.
Es ist wichtig anzumerken, dass unterschiedliche Platzhalter Symbole existieren, um die Einsetzung der Parameter an der richtigen Stelle garantieren zu können. Die Syntax der Strings, die zur Ausführung einer Click and Fire Operation benötigt werden, ist dem Kapitel „Click and Fire Syntax“ zu entnehmen.

4.4 Entwurf Plugin

Die Grundfunktion eines Plugins ist die Annahme und die Verarbeitung von Aufgaben und Diensten. Diese Aufgaben werden vom Anwendungskern über die Message Queue versendet und von einem Plugin ausgeführt.
Die Architektur eines Plugins ist nicht vollständig festgelegt, große Teile der Architektur liegen im Ermessen des Plugin-Architekten.
Beim Entwurf oder der Implementierung eines Plugins müssen bestimmte Funktionalitäten angeboten werden, um einen fehlerfreien Ablauf der Anwendung garantieren zu können. Diese Aufgabenbereiche müssen zwingend abgedeckt werden und unterteilen sich in Empfangen und Senden, Registrierung, Erkennung von Nachrichtentypen, Delegation von Aufgaben, Ausführung von Aufgaben und Rückführung der Ergebnisse.
Sämtliche weitere Funktionalitäten sind optional und zur korrekten Ausführung der Anwendung nicht notwendig. So ist es beispielsweise dringend empfohlen, Statusnachrichten zu aktiven Prozessen zu versenden oder eine allgemeine Verwaltung aller aktiven Prozesse zu entwerfen.

Es ist anzumerken, dass solange die Ausführung der notwendigen Funktionalitäten korrekt abläuft, die Architektur aus Sicht des Anwendungskerns irrelevant ist.
Im Folgenden sind die Architekturentscheidungen des Entwurfs mit Bedacht auf Performance, geringer Kopplung, Benutzerkomfort und Modularisierung getroffen worden. Die Einhaltung und Nachahmung wird nahe gelegt.

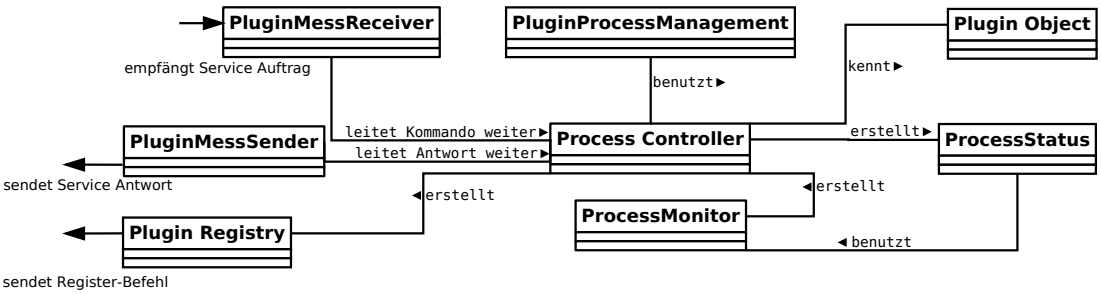

Abbildung 4.3: Fachliches Datenmodell eines möglichen Plugins

Die Abbildung 4.2 ist eine unvollständige Darstellung des fachlichen Datenmodells eines Plugins. Eine vollständige Version kann im Anhang eingesehen werden. Basierend auf diesem Modell folgt nun die Beschreibung der einzelnen Komponenten.

Plugin Process Management:
Das Plugin Process Management kümmert sich um die Verwaltung aller aktiven Prozesse seines Plugins und wird vom Plugin Message Receiver initialisiert.
Für jede empfangene Dienstaufforderung an das Plugin wird ein Verwaltungsobjekt erstellt. Dieses Objekt enthält eine Referenz auf den laufenden Prozess und wird sowohl an den Anwendungskern als auch an das Plugin Process Management übergeben.
Das Plugin Process Management benötigt das Objekt, falls der Anwendungskern eine Terminierung aller laufenden Prozesse veranlasst.
Der Anwendungskern benutzt das Objekt zur Anzeige aller laufenden Prozesse und fügt es in seine Verwaltungsstruktur ein.
Sobald der laufende Prozess terminiert, wird eine Nachricht an den Anwendungskern versendet. Die Nachricht besteht aus dem Referenz Objekt des laufenden Prozesses und einer Aufforderung zum Löschen. Der Anwendungskern löscht die Referenz auf den Prozess und korrigiert die GUI entsprechend.

Plugin Message Receiver:
Der Plugin Message Receiver dient dem Empfangen von Nachrichten. Darüber hinaus wird über den Plugin Message Receiver das Plugin initialisiert. Der Plugin Message Receiver initialisiert den Plugin Message Sender und veranlasst das Versenden einer Registrierungsaufforderung an den Anwendungskern. Gekapselt in der Registrierungsaufforderung ist das Plugin Objekt des Plugins. Die Erstellung des Objekts wird vom Plugin Message Receiver übernommen.
Nach Erstellung aller benötigten Instanzen meldet sich der Plugin Message Receiver bei der Message Queue des Anwendungskerns an und geht in den Empfangszustand über.
Sobald Nachrichten empfangen werden, muss geprüft werden, ob die Nachricht an die eigene Instanz adressiert wurde. Falls der Adressat ein beliebiges anderes Plugin ist, wird die Nachricht verworfen und auf den Erhalt der nächsten Nachricht gewartet.
Ist die Nachricht für das Plugin bestimmt, wird der Auftrag aus dem empfangenem Kommando Objekt extrahiert. Mit Hilfe dieses Auftrags wird eine Instanz des Process Controllers erstellt. Der Process Controller muss eine Referenz auf den Plugin Message Sender erhalten. Die Ausführung des Auftrages muss asynchron geschehen, da keine Aussage über die Terminierung des Process Controllers getroffen werden kann.
Nach der Erteilung des Auftrages begibt sich der Plugin Message Receiver wieder in den Empfangszustand.

Plugin Message Sender:
Der Plugin Message Sender wird vom Plugin Message Receiver initialisiert und ist zuständig für das Versenden von Nachrichten an den Anwendungskern. Zusätzlich ist Dieser zuständig für das Versenden einer Registrierungsaufforderung an den Anwendungskern. Dies geschieht mit Hilfe eines Registrierungsobjekts, welches der Plugin Message Sender vom Plugin Message Receiver übergeben bekommt.
Wenn der Process Controller einen Dienst ausführt, verschickt er neben der Ausgabe des Diensts auch Statusnachrichten. Beide Nachrichtentypen werden an den Plugin Message Sender gegeben. Von hier aus werden sie an den Anwendungskern versendet.
Zum Versenden von Nachrichten an den Anwendungskern muss sich der Plugin Message Sender bei der Message Queue des Anwendungskerns anmelden.

Process Controller:
Eine Instanz des Process Controllers dient der Ausführung von genau einem erteilten Auftrag. Dabei ist die Lebenszeit jeder Instanz an die Ausführung des Auftrages gebunden.
Eine Instanz des Process Controllers wird initialisiert, sobald eine Nachricht vom Anwendungskern empfangen wird. Der Plugin Message Receiver übergibt dem Process Controller den Auftrag und die Instanz des Plugin Message Senders.
Zunächst werden die zur Ausführung des Auftrages benötigten Parameter (Arbeitsverzeichnis, Statusintervall und Speicherort für Dateien) gesetzt. Daraufhin startet der Process Controller den Arbeitsprozess sowie einen Prozess der die Terminierung des Arbeitsprozesses beobachtet, den Process Monitor. Zusätzlich wird ein Process Writer gestartet. Dieser sendet in regelmäßigen Abständen Status Nachrichten zum laufenden Prozess an den Anwendungskern. Alle drei oben beschriebenen Komponenten müssen asynchron ausgeführt werden, da ansonsten die Ausführung des Arbeitsprozesses blockiert werden kann.
Jede vom Arbeitsprozess generierte Nachricht wird in ein Kommando Objekt gekapselt und an den Anwendungskern verschickt. Der Adressat des Objekts stimmt mit dem Namen des Plugins überein, so dass das Ergebnis in der entsprechenden Ansicht der GUI angezeigt werden kann.
Sollte die Ausführung des empfangenen Befehls nicht möglich sein, wird eine entsprechende Fehlermeldung erzeugt und an den Anwendungskern versendet.

Process Monitor:
Der Process Monitor ist eine Hilfsklasse und wird vom Process Controller initialisiert. Der Process Monitor erhält bei Erstellung eine Referenz auf den aktiven Arbeitsprozess und beobachtet, ob der dieser terminiert ist oder nicht.
Der Process Monitor wird vom Process Writer benutzt, um festzustellen, ob eine Status Anfrage an den Arbeitsprozess gesendet werden kann. Die Statusnachricht gibt Aufschluss über Laufzeit, Versuche pro Minute und geschätzten Arbeitsfortschritt. Die genaue Syntax der Statusnachricht ist nicht vorgegeben und kann entweder vom unterliegenden Programm übernommen oder vom Plugin-Architekten selbst generiert werden.

Process Status:
Der Process Status hat die Aufgabe Statusnachrichten zu einem laufenden Prozess abzufragen und an den Anwendungskern zu verschicken.
Der Process Status wird vom Process Controller initialisiert und ist an die Lebenszeit des Arbeitsprozesses des Process Controllers gebunden. Er hält Referenzen auf den Arbeitsprozess, den zugehörigen Process Monitor und dem Plugin Message Sender. Der Process Status muss asynchron arbeiten, um die Ausführung des Arbeitsprozesses nicht unnötig zu verzögern. Aus diesem Grund benötigt Process Status eine eigene Referenz auf den Plugin Message Sender.
In einem bestimmten Intervall werden Statusnachrichten generiert und an den Anwendungskern verschickt. Die Generierung dieser Nachrichten ist vom unterliegenden Programm abhängig. Hier ist es notwendig Fachwissen über das genutzte Programm zu besitzen, um die Erstellung von Statusnachrichten auslösen zu können.
Es ist möglich ohne Process Status zu arbeiten, es wird aber nicht empfohlen. Cracking Sessions können sehr lange dauern. Die Generierung von Statusnachrichten hilft bei der Beobachtung des Fortschritts des Arbeitsprozesses. Auf diese Weise kann festgestellt werden, ob der Arbeitsprozess terminiert ist und ob das Plugin noch lauffähig ist.

4.5 Entwurf Public Klassen

Dieses Kapitel beschreibt Klassen, welche der gesamten Anwendung bekannt sind. Diese Klassen können als Helferklassen betrachtet werden. Sie werden bei der Kommunikation und der Verwaltung benötigt.

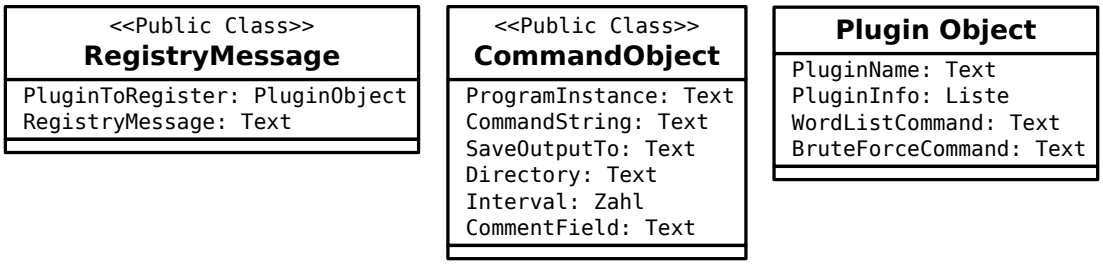

Abbildung 4.4: Fachliches Datenmodell des Public Klassen

Die Abbildung 4.2 ist eine Darstellung der öffentlichen Klassen. Die folgende Beschreibung bezieht sich auf Nutzung und Verarbeitung innerhalb des Programmablaufs.

Command Object:
Das Command Object wird als Transportobjekt zur Kommunikation zwischen dem Anwendungskern und seinen Plugins genutzt.
Zur eindeutigen Identifikation des Empfängers und der korrekten Ausführung des Befehls besitzt das Command Object mehrere Instanzvariablen. Diese Variablen beziehen sich auf den eindeutigen Namen des Plugins (z.B. „John the Ripper“ oder „Hashcat“), dem auszuführenden Befehl, einem Arbeitsverzeichnis, einem Speicherort für Dateien und dem Zeitintervall für die Statusnachricht. Zusätzlich existiert ein Kommentarfeld für beliebige Strings. Im Fall der Prozessverwaltung wird das Kommentarfeld benötigt, um festzustellen welche Prozesse noch aktiv sind.
Bei Anfrage eines Diensts wird der konkrete Befehl in ein Command Object gekapselt und an den Framework Controller übergeben. Hier wird es über den Framework Sender serialisiert und an alle registrierten Plugins versendet.
Alle Plugins, die das Command Object empfangen, deserialisieren es und prüfen anhand des eindeutigen Namens, ob das Objekt für sie bestimmt ist. Nur der Adressat des Command Objects darf das Objekt verarbeiten, alle anderen Plugins müssen es verwerfen.
Beim Empfang durch ein Plugin wird das Command Object einer Instanz des Process Controllers übergeben. Hier werden alle benötigten Parameter zur Ausführung gesetzt und der Arbeitsprozess wird gestartet.
Sämtliche Ergebnisse werden in ein Command Object verpackt und an den Anwendungskern verschickt. Wichtig zu beachten ist, dass der Name des Command Objects dem Namen des Plugins entsprechen muss, um eine eindeutige Zuordnung garantieren zu können. Der Speicherort, das Arbeitsverzeichnis, das Kommentarfeld sowie das Zeitintervall sind bei einer Antwort irrelevant.

Registry Object:
Das Registry Object wird als Transportobjekt benutzt und dient der Registrierung eines Plugins beim Anwendungskern.
Sobald ein Plugin alle notwendigen Instanzen initialisiert hat, wird ein Registry Object erstellt, serialisert und an den Anwendungskern verschickt. Von diesem Zeitpunkt an besitzt der Anwendungskern eine Referenz auf das entsprechende Plugin. Die Referenz liegt in Form eines Plugin Objects vor und wird vom Registry Object gekapselt.
Nach der Registrierung erstellt der Anwendungskern einen neuen Reiter in der GUI, welcher benutzt werden kann, um Befehle an das Plugin abzusetzen. Zusätzlich wird das Plugin in die Liste der Programme für Click and Fire Operationen eingefügt. Diese Liste wird im entsprechenden Reiter der GUI angezeigt. Beide GUI Elemente werden mit Hilfe der Daten aus dem Plugin Object erstellt.
Der Registrierungsvorgang bedarf keiner Erfolgsmeldung.

Plugin Object:
Das Plugin Object dient der Identifikation eines Plugins und enthält alle zur Ausführung einer Operation benötigten Daten. Zu diesem Zweck enthält das Plugin Object den eindeutigen Namen des Plugins und die Syntax für Click and Fire Operationen. Zusätzlich lässt sich anhand der Instanzvariablen ablesen, welche Click and Fire Operationen unterstützt werden.
Während der Initialisierung des Plugins werden alle benötigten Komponenten gestartet. Daraufhin wird ein Registry Object erstellt, das ein Plugin Object kapselt. Dieses Objekt wird serialisiert und an den Anwendungskern verschickt.
Beim Empfang durch den Anwendungskern wird das Registry Object deserialisert. Daraufhin wird das Plugin Object entnommen und am den Framework Controller gegeben.
Der Framework Controller gibt das Plugin Object an den Framework Plugin Manager, wo es zur Verwaltung in eine entsprechende Datenstruktur eingefügt wird.
Bei Aktualisierung der GUI wird die Liste der registrierten Plugins zurückgegeben. Diese Liste wird benutzt, um die GUI entsprechend zu aktualisieren. Bei Aktualisierung wird ein Reiter für das Plugin erstellt. Zusätzlich wird das Plugin in die Liste der Click and Fire fähigen Operationen eingefügt, sollte es die entsprechende Operation unterstützen.
Das Format der Strings zur Ausführung einer Click and Fire Operation ist dem Kapitel „Click and Fire Syntax“ zu entnehmen.

4.6 Entwurf Message Queue

Die Message Queue ist der Kommunikationskanal zwischen dem Anwendungskern und seinen Plugins. Zu versendende Nachrichten werden über die Message Queue geleitet.
Während der Initialisierung des Anwendungskerns werden Instanzen des Framework Senders und des Framework Receivers erstellt. Diese kümmern sich um die Initialisierung der eigenen Message Queues. Zur Laufzeit existieren zwei Message Queues, eine Queue wird zum Senden, die andere zum Empfangen von Nachrichten genutzt.
Zu verschickende Nachrichten werden von Transportobjekten gekapselt und an den Framework Sender übergeben. Hier werden sie serialisiert und über die Message Queue versendet.
Inspiriert vom Adress-Resolution-Protocol (RFC 826), werden Nachrichten an alle registrierten Plugins versendet. Zu diesem Zweck besitzt jedes Plugin ein Sende- und Empfangsmodul. Beide sind an die entsprechenden Message Queues des Anwendungskerns angebunden.
Nach Empfang und Deserialisierung des Objekts muss festgestellt werden, ob das Objekt den eigenen Namen als Empfänger enthält. Falls dies nicht der Fall ist, muss das Objekt verworfen werden, ansonsten kann der angeforderte Dienst ausgeführt werden.
Während und nach der Ausführung des Diensts werden alle Ergebnisse in ein Objekt verpackt und an das Sendemodul des Plugins gegeben. Hier werden die Objekte serialisiert und mit Hilfe der Message Queue übertragen.
Das vom Anwendungskern empfangene Objekt wird deserialisiert und bis zur GUI hochgereicht.

Die Übertragung von Registrierungsaufforderungen geschieht äquivalent, aber ohne eine Anzeige innerhalb der GUI. Stattdessen wird mit Hilfe des in der Registrierungsaufforderung enthaltenen Plugin Objects ein Eintrag in der Verwaltungskomponente vorgenommen. Anhand der Daten in der Verwaltungskomponente können die erforderlichen GUI Elemente erstellt werden.

4.7 Passwortdatei Formate

Ein großer Fokus der Anwendung liegt auf Passwortdateien. Ein Benutzer kann seine eigenen Passwortdateien zur Verfügung stellen oder eine durch die Anwendung erstellen lassen.
Wenn ein Benutzer eine eigene Passwortdatei benutzen möchte, muss Diese im gleichen Format wie die Unix /etc/passwd Datei vorliegen.
Die Unix /etc/passwd Datei enthält Informationen zu Benutzern und Passwörtern, welche während eines Loginvorgangs benötigt werden.
Beim Login gibt der Benutzer das Passwort zu einem Account an. Mit Hilfe des Account Namens wird der entsprechende Eintrag in der /etc/passwd Datei herausgesucht. Dieser Eintrag enthält eine Referenz auf den korrespondierenden Eintrag in der /etc/shadow Datei, wo das gehashte Passwort verwaltet wird.
Das vom Benutzer eingegebene Passwort wird mit Hilfe der Daten aus der /etc/passwd Datei gehasht und mit dem gespeicherten Hashwert verglichen. Falls beide Zeichenfolgen übereinstimmen, wird der Benutzer eingeloggt.
Die Trennung von Accountdaten und zugehörigem Passwort findet aus Sicherheitsgründen statt. Die Passwörter befinden sich in der /etc/shadow Datei, da sie hier nur vom Super User einsehbar sind. Dadurch wird der direkte Zugriff auf Passwörter erschwert.
Im normalen Betrieb müssen diese beiden Dateien zusammen geführt werden, um eine geeignete Passwortdatei zu erhalten. [UMR 2014]
Um ein besseres Verständnis zu schaffen, wird im Folgenden davon ausgegangen, dass diese Dateien bereits zusammen geführt worden sind.

Format der kombinierten /etc/passwd:
Die kombinierte /etc/passwd Datei ist ein Textdokument und enthält einen Eintrag pro Zeile. Jeder Eintrag beschreibt einen Benutzer-Account des Systems. Dabei besteht jeder Eintrag aus sieben Feldern, welche mit einem Doppelpunkt getrennt werden. Die Reihenfolge der Felder ist relevant.
Zunächst die Beschreibung eines generischen Eintrags:

Username:Password:UID:GID:Info:HomeDir:Shell

Username:
Der Username ist der String , mit dem sich der Benutzer beim System anmeldet. Dieser Name muss eindeutig sein. Falls doppelte Einträge vorhanden sind, wird der erste gefundene Eintrag verwendet.

Password:
Das Password Feld beschreibt Informationen über das Passwort des Benutzers. Es können Informationen über den genutzten Hashing-Algorithmus und den benutzten Saltwert abgefragt werden. Außerdem kann der Hashwert des Passworts abgelesen werden. Falls dieses Feld leer ist, wird kein Passwort für den Login benötigt. Sollte dieses Feld ein kleingeschriebenes „x“ enthalten, ist das gehashte Passwort im korrespondierenden Eintrag der /etc/shadow Datei zu finden.

UID:
Dieses Feld beschreibt die eindeutige Kennung des Benutzers. Die UID 0 wird an den Super User vergeben. Die Zahlen 1-1000 werden an die Systembenutzer vergeben.

GID:
Die Gruppen-ID des Benutzers.

Info:
Dieses Feld enthält Informationen zum Benutzer. Es ist unter anderem möglich den Namen oder die Adresse des Benutzers zu hinterlegen. Einzelne Informationssegmente werden mit einem Komma getrennt. Auch jeglicher anderer Text kann in dieses Feld eingetragen werden.

HomeDir:
Dieses Feld enthält das Heimatverzeichnis eines Benutzers. Dieses Feld zeigt an, in welchem Verzeichnis sich ein Benutzer befindet, nachdem er eingeloggt wurde. Es ist beschrieben durch einen absoluten Pfad.

Shell:
Dieses Feld beschreibt den absoluten Pfad einen Programms. Dieses Programm wird nach dem Login gestartet. In der Regel handelt es sich hierbei um eine Shell, es kann jedoch auch jedes andere ausführbare Programm sein, solange das Programm einen Eintrag in der /etc/shells Datei besitzt. Der Default-Parameter des Shell Feldes ist „/bin/sh“.

Ein konkreter Eintrag in einer Passwortdatei kann wie folgt aussehen:

tAnderson:$1$$5f4dcc3b5aa765d61d8327deb882cf99:1001:1000:Thomas Anderson,Whitehaven Apts:/home/tAnderson:/bin/sh

Der Name des konkreten Benutzers ist „tAnderson”. Das Passwort wurde mit dem MD5-Algorithmus gehasht (Zahlencode 1) und besitzt keinen Saltwert. Der Hashwert des Passworts lautet „5f4dcc3b5aa765d61d8327deb882cf99“. Der Benutzer hat die eindeutige Kennung 1001 und die Gruppenkennung 1000. Weitere Informationen beschreiben den vollen Namen, „Thomas Anderson“, und die Adresse, „Whitehaven Apts“. Sein Heimatverzeichnis ist „/home/tAnderson“, nach dem Loginvorgang wird eine Shell gestartet.

Wie beschrieben, kann eine Passwortdatei in diesem Format von allen unter Ubuntu ausführbaren Passwort-Cracking-Programmen gelesen werden. Somit ist es möglich diese Passwortdatei als Parameter für eine Cracking-Session zu benutzen, um so die Passwörter lesbar zu machen.

Falls der Anwender keine solche Datei besitzt oder nicht weiß, wie man eine solche Datei erstellen kann, ist es möglich eine geeignete Passwortdatei erstellen zu lassen. Dazu muss der Benutzer eine Textdatei mit einem Eintrag pro Zeile erstellen. Jeder Eintrag besteht aus zwei Feldern, die den Accountnamen und das Passwort im Klartext beschreiben. Getrennt werden die Felder durch einen Doppelpunkt:

Username:Passwort

Username und Passwort verhalten sich äquivalent zu den oben genannten Feldern.
Somit ergibt sich als konkreter möglicher Eintrag:

tAnderson:superPasswort1

Der Name des Benutzers ist „tAnderson“, sein gewünschtes Passwort ist „superPasswort1“. Nach Angabe dieser Datei kann der Benutzer die entsprechende Operation anstoßen und erhält als Ergebnis eine neue Datei. Diese hat das Format der /etc/passwd Datei kombiniert mit der /etc/shadow Datei.
Das Ergebnis kann benutzt werden, um beliebige Passwörter auf Stärke zu prüfen, in dem es an ein Plugin weiter gegeben wird. Entweder mit Hilfe des Click and Fire Builders oder händisch durch Weitergabe an ein Plugin.
Es ist zusätzlich möglich einen Saltwert zu benutzen. Dieser Saltwert darf nicht vom Benutzer vorgegeben werden, sondern muss durch das System generiert werden.

4.8 Click and Fire Syntax

Click and Fire Operationen werden benutzt, um einen Dienst anzufragen. Dabei ist es nicht notwendig Wissen über die Funktionsweise des ausgeführten Diensts zu besitzen. Es reicht, alle notwendigen Parameter zu setzen. Die Ausführung wird delegiert, die Ergebnisse werden angezeigt.
Nach der Auswahl eines bestimmten Diensts durch den Benutzer, wird eine Liste aller Plugins angezeigt, welche diese Operation unterstützen. Ebenfalls werden die entsprechenden GUI Elemente erzeugt, um die Parameter angeben zu können. Nachdem die Parameter gesetzt und der Dienst angestoßen wurde, erhält der Frame Plugin Manager eine Anfrage. Diese Anfrage gibt als Ergebnis einen String zurück. Mit Hilfe dieses Strings wird das gewünschte Kommando erzeugt.
Die Syntax des Strings besteht aus einer Zeile und enthält den exakten Befehl des angefragten Diensts. Anstelle von Parametern enthält der String Schlüsselwörter. Die Schlüsselwörter werden durch konkrete Werte ersetzt und bilden so einen ausführbaren Befehl.

Im Umfang dieser Arbeit werden zwei Click and Fire Operationen angeboten: Der Brute Force-Modus und der Wortlisten-Modus eines Plugins. Bei beiden Modi kann exakt vorausgesagt werden, welche Parameter zur korrekten Ausführung gesetzt werden müssen. Aus diesem Grund existieren drei Platzhalter Symbole:
Der String **„PASSD“** steht an der Stelle des Pfads für die Passwortdatei.
Der String **„WORDL“** steht an der Stelle des Pfads für die Wortliste.
Der String **„HASHT“** steht an der Stelle des Hashtypen. Der Hashtyp ist eine Abbildung des Hashing-Verfahrens auf eine Zahl.
Die folgenden Strings sind das Ergebnis einer Anfrage an ein Plugin (John the Ripper und Hashcat) im Wortlisten-Modus, bevor die konkreten Parameter eingesetzt wurden:

john --wordlist=WORDL PASSD

./hashcat.bin –a 0 –m HASHT PASSD WORDL

Im nächsten Schritt werden die Platzhalter Symbole durch die konkreten Parameter ersetzt. Durch Angabe eines Arbeitsverzeichnisses ist es nicht notwendig, die absoluten Pfade einer Datei anzugeben. Um ein besseres Verständnis zu schaffen, werden sie trotzdem beschrieben.
Das Ergebnis sieht folgendermaßen aus:

john --wordlist=/home/user/jtr1.8.0/run/myWordList
/home/user/jtr1.8.0/run/myPassList

./hashcat.bin –a 0 –m 1800 /home/user/hashcat/myPassList
/home/user/hashcat/myWordList

Oder unter Angabe des Arbeitsverzeichnisses:

john --wordlist=myWordList myPassList

./hashcat.bin –a 0 –m 1800 myPassList myWordList

Nach Erstellung des Kommandos, wird ein Command Object erzeugt und verschickt. Das enthaltende Kommando wird vom Plugin ausgeführt.
Die Erweiterung dieser Funktionalität um weitere Schnittmengenoperationen unterschiedlicher Passwort-Cracking Anbieter ist mit vertretbarem Aufwand möglich. Zur Erweiterung müssen die erforderlichen GUI Elemente erstellt werden, um alle nötigen Parameter angeben zu können. Zusätzlich ist es notwendig die Implementierung des Plugin Objects so zu verändern, dass die erforderlichen Felder der Schnittmengenoperation erfasst werden können. Sollte ein neuer Modus eingeführt werden, muss ein neues Feld erstellt werden, welche die Syntax der unterliegenden Operation, inklusive Platzhalter-Symbolen,

enthält. Möglicherweise müssen zu diesem Zweck neue Schlüsselwörter eingeführt werden. Zuletzt muss die Methode angepasst werden, um den Zusammenbau des Kommandos auf Grundlage der Schlüsselwörter zu erweitern.

4.9 Ablauf des Programms

Dieses Kapitel beschäftigt sich mit den Abläufen innerhalb der laufenden Anwendung. Dabei werden die im Hintergrund laufenden Aufrufe der Anwendung dargestellt und erläutert. Insbesondere geht es um die Mechanismen bei der Nutzung von Plugins, der Erstellung von Passwortdateien und der Kommunikation des Anwendungskerns mit den Plugins.

Initialisierung des Anwendungskerns:
Abbildung 4.5 beschreibt den Initialisierungsvorgang beim Starten des Anwendungskerns:

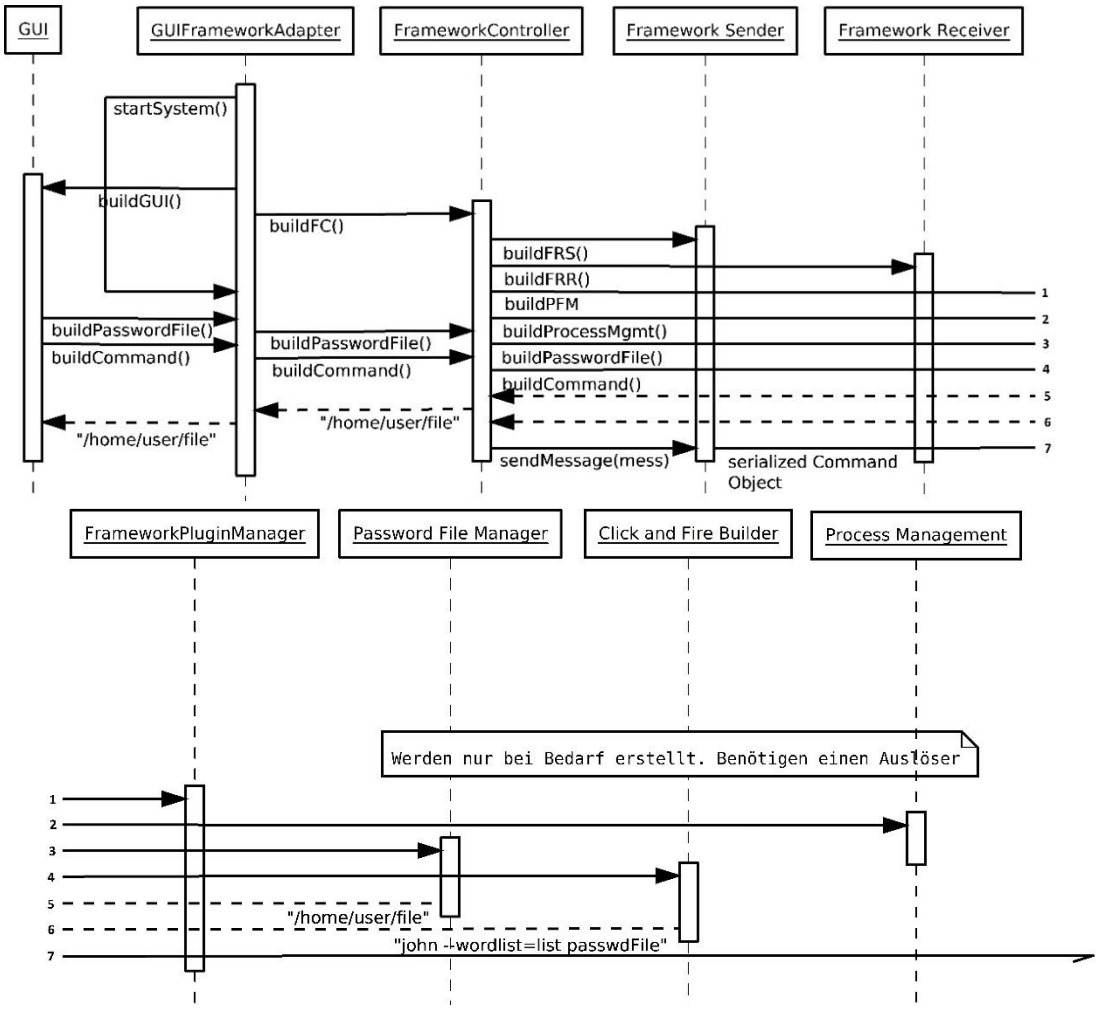

Abbildung 4.5: Sequenzdiagramm der Initialisierung des Anwendungskerns

Der Anwendungskern wird durch den GUIFrameworkAdapter angestoßen. Der GUIFrameworkAdapter erstellt die GUI und den Framework Controller. Der Framework Controller initialisiert alle weiteren Instanzen zur Ausführung von Operationen. Konkret handelt es sich um den Framework Sender und Receiver und den Framework Plugin Manager. Da der Framework Controller die wichtigsten Referenzen selber erstellt, werden alle Aufträge von ihm erteilt und verteilt.
Der Framework Receiver erstellt bei Initialisierung eine Message Queue und geht in den Empfangs-Modus über. Hier wartet der Receiver auf potenzielle Nachrichten von Plugins.
Der Framework Sender meldet sich bei der vorhandenen Message Queue an und erwartet von diesem Zeitpunkt an Sendeaufträge vom Framework Controller.
Die Erstellung des Anwendungskerns ist mit der Initialisierung des Framework Plugin Managers abgeschlossen.
Die Instanzen des Password File Managers und des Click and Fire Builders werden erst erstellt, wenn sie benötigt werden. Die Initialisierung dieser beiden Instanzen muss durch den Benutzer erfolgen. Der Benutzer agiert also als Auslöser für die Erstellung. Nachdem sie ihre Funktion erledigt haben, werden die Referenzen verworfen und bei der nächsten Erteilung eines Auftrages werden neue Instanzen erstellt.
Sowohl beim Password File Manager als auch beim Click and Fire Builder werden jeweils Aufträge erteilt, welche nach Abarbeitung das Ergebnis an den Framework Controller zurück geben. Ergebnisse des Password File Managers werden an die GUI gegeben. Ergebnisse des Click and Fire Builders werden in ein Command Object verpackt und zur Ausführung an das Plugin gesendet.

Registrierung eines Plugins:
Abbildung 4.6 beschreibt den Ablauf der Registrierung eines Plugins beim Anwendungskern. Unten im Bild ist die Legende zum Diagramm zu finden.

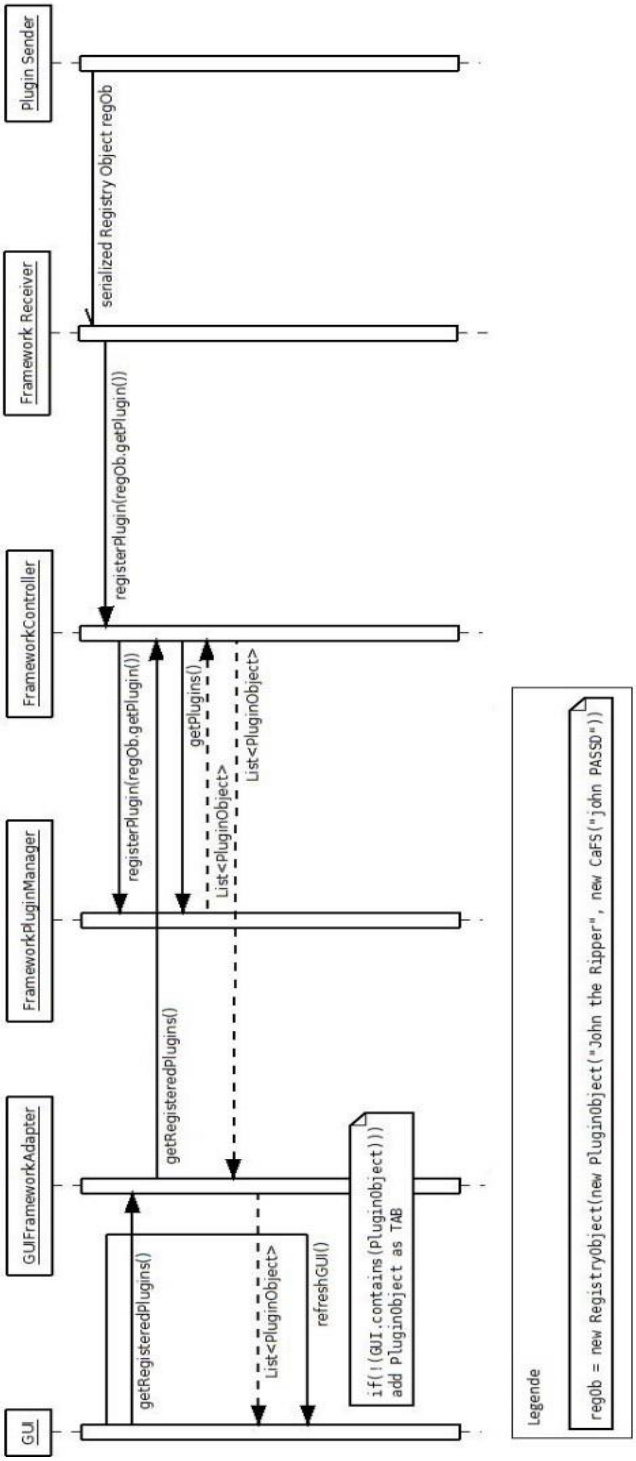

Abbildung 4.6: Sequenzdiagramm der Registrierung eines Plugins

Die Registrierung eines Plugins beginnt mit dem Versenden eines Registry Objects. Dieses Registry Object wird vom Plugin versendet und vom Framework Receiver empfangen.
Nach der Deserialisierung wird überprüft, ob es ein Registry Object ist. Im korrekten Fall enthält das Registry Object ein Plugin Object. Dieses Plugin Object wird an den Framework Controller übergeben. Der Framework Controller übergibt das Plugin Object an den Framework Plugin Manager, bei dem es verwaltet wird. Das Plugin ist nun angemeldet.
Zur Vervollständigung der kompletten Operation muss das Plugin dem Benutzer zur Verfügung gestellt werden. Dafür ist es nötig, dass der Benutzer die GUI aktualisiert. Die Aktualisierung stößt eine Anfrage beim Framework Controller an, welcher den Aufruf an den Framework Plugin Manager delegiert. Der Aufruf kehrt mit einer Liste aller angemeldeten Plugins zurück. Diese Liste wird vom Framework Controller an die GUI weitergeleitet, bei dem ein Abgleich mit den bereits erstellten GUI Elementen stattfindet. Nun kann der Benutzer ein angemeldetes Plugin in vollem Umfang nutzen.

Anforderung eines Diensts durch ein Plugin:
Das erste Diagramm in Abbildung 4.7 bezieht sich auf die Seite des Anwendungskerns, das zweite Diagramm auf die Seite des Plugins. Abbildung 4.71 enthält die Legende zu Abbildung 4.7.

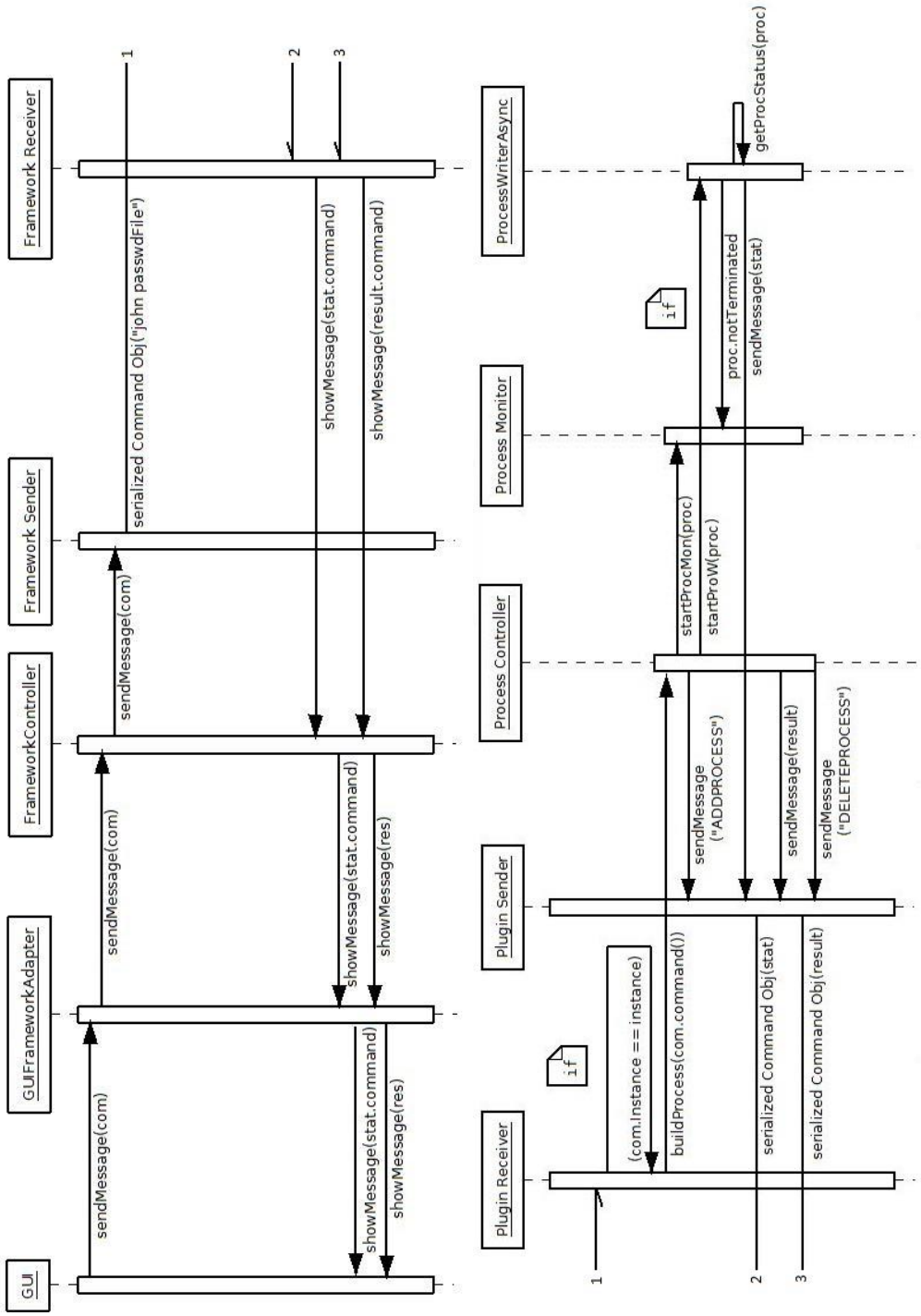

Abbildung 4.7: Sequenzdiagramm der Anforderung eines Diensts durch den Anwendungskern

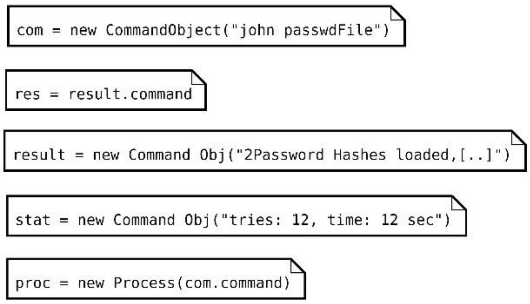

Abbildung 4.71: Legende zu Abbildung 4.7

Die Nutzung eines Diensts wird durch den Benutzer ausgelöst. Dabei wird die entsprechende Sicht in der GUI benutzt, um ein Kommando an ein bestimmtes Plugin abzusetzen. Das Kommando wird in ein Command Object gekapselt und über den GUIFrameworkAdapter an den Framework Controller gegeben. Der Framework Controller reicht das Objekt an den Framework Sender, wo es serialisiert und versendet wird.
Das entsprechende Plugin empfängt das Objekt und deserialisert es unter der Bedingung, dass das Plugin auch der Adressat ist. Daraufhin wird das im Objekt gekapselte Kommando extrahiert. Der Kommando String wird zusammen mit weiteren Parametern an den Process Controller gegeben. Der Process Controller erzeugt mit Hilfe des Kommadostrings einen Arbeitsprozess und führt den Dienst aus. Daraufhin wird ein Process Monitor gestartet, welcher die Terminierung des Arbeitsprozesses beobachtet. Zu diesem Zweck erhält der Process Monitor eine Referenz auf den Arbeitsprozess.
Nach der Erstellung des Process Monitor wird ein Process Writer gestartet. Dieser kümmert sich um das Versenden von Statusinformationen zum Arbeitsprozess und besitzt eine Referenz auf Selbigen.
Der Process Monitor und der Process Writer arbeiten Hand in Hand. Dies bedeutet, dass der Process Writer in regelmäßigen Abständen beim Process Monitor anfragt, ob der Arbeitsprozess noch aktiv ist. Sollte dies der Fall sein, wird der Arbeitsprozess aufgefordert seinen Status zurück zu geben. Der erhaltene String wird in ein Command Object gekapselt und an den Plugin Sender gegeben. Dort wird er serialisiert und zurück an den Anwendungskern versendet. Der Framework Receiver empfängt das Command Object und deserialisert es. Daraufhin wird die Statuszeile aus dem Objekt extrahiert und an den Framework Controller gegeben. Dieser leitet das Ergebnis bis zur GUI weiter.
Nach Terminierung des Arbeitsprozesses wird sein Ergebnis auf die gleiche Weise wie die Statuszeile verpackt, versendet, empfangen und in der GUI angezeigt.

Nutzung des Click and Fire Builders:
Abbildung 4.8 beschreibt den Ablauf bei der Nutzung des Click and Fire Builders:

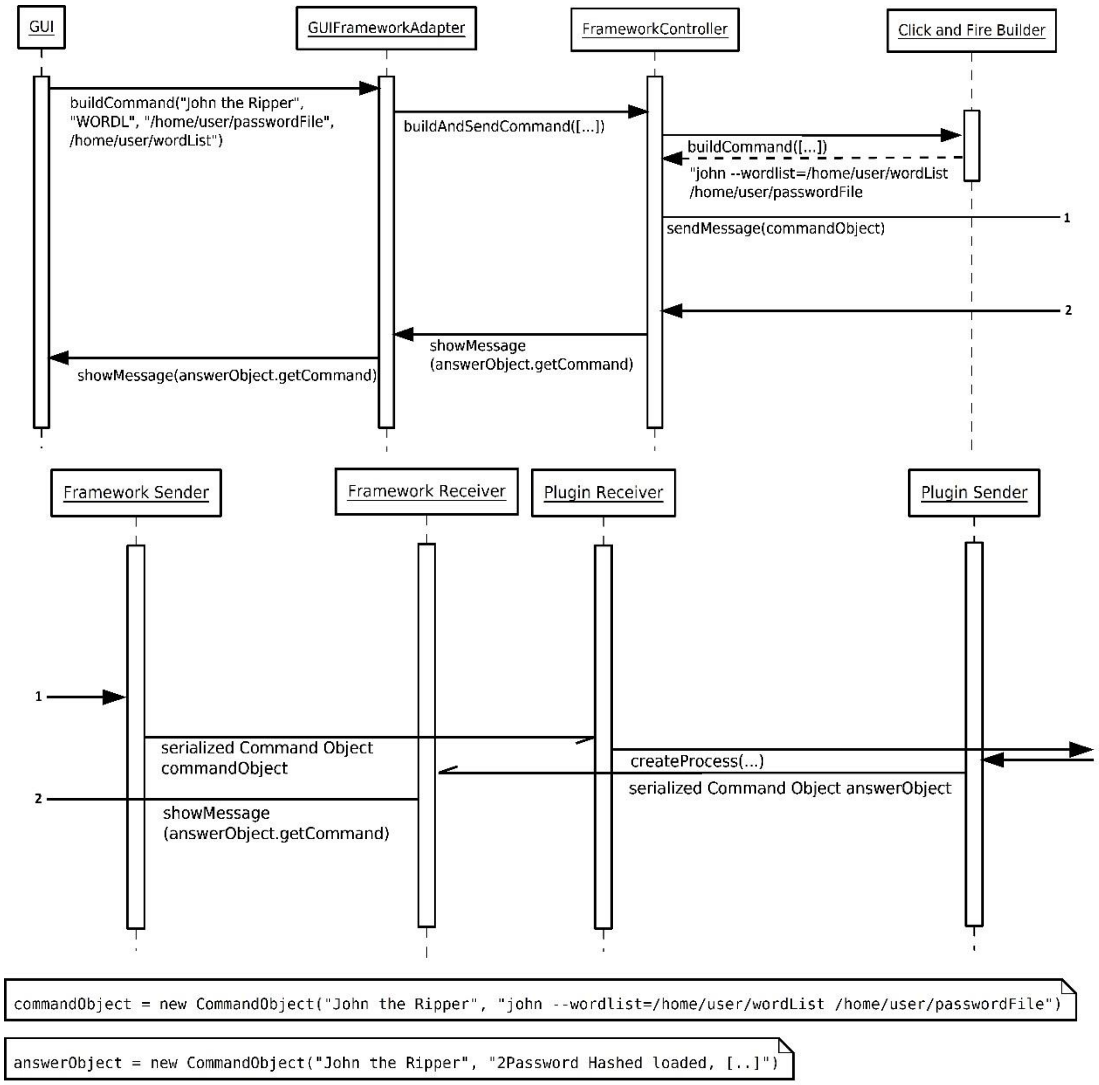

Abbildung 4.8: Sequenzdiagramm der Bildung eines Kommandos durch den Click and Fire Builder

Die Nutzung des Click and Fire Builders wird durch den Benutzer ausgelöst. Dabei wird im entsprechenden GUI Element eine Click and Fire Operation ausgewählt und die benötigten Parameter werden gesetzt. Nach dem Anstoß der Operation findet ein Aufruf an den GUIFrameworkManager statt. Das Ziel ist ein aus den Parametern abgeleitetes korrektes Kommando zu erhalten. Der Aufruf wird über den Framework Controller an den Click and Fire Builder delegiert. Hier wird aus den gestellten Parametern ein entsprechendes

Kommando erstellt und an den Framework Controller zurückgegeben. Dieses Kommando wird vom Framework Controller in ein Command Object verpackt und mit einem Empfänger versehen. Zusätzlich werden alle zur Ausführung des Diensts benötigten Parameter gesetzt. Diese Umschließen die im Entwurf des Command Objects beschriebenen Parameter.
Der Framework Controller gibt das erstellte Objekt an den Framework Sender, wo es serialisiert und versendet wird. Der Empfänger führt das im Objekt gekapselte Kommando aus. Die im Kapitel 3.3 beschriebenen Instanzen werden ebenfalls erzeugt. Die pluginseitig erzeugten Ergebnisse werden mit Hilfe des gleichen Vorgangs verpackt und serialisiert versendet.
Der Framework Receiver empfängt das Objekt und deserialisert es. Daraufhin wird das Ergebnis an den Framework Controller gegeben und über den GUIFrameworkAdapter an die GUI delegiert.

Formatierung einer Passwortdatei durch Password File Manager:
Abbildung 4.9 beschreibt den Ablauf der Formatierung einer Passwortdatei durch den Anwendungskern:

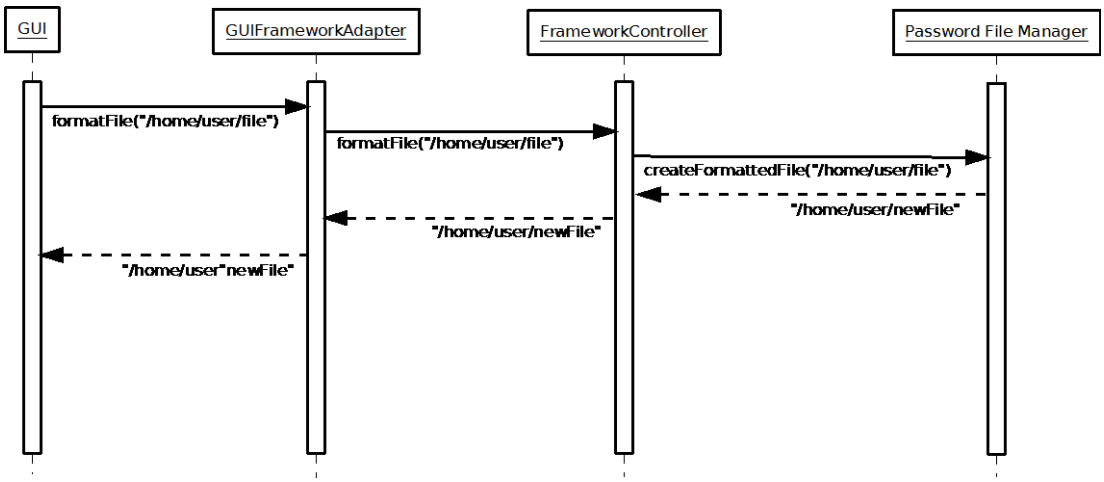

Abbildung 4.9: Sequenzdiagramm der Formatierung einer Datei

Der Ablauf einer Formatierung wird durch den Benutzer ausgelöst. Dabei gibt der Benutzer eine von zwei Dateiarten an, die sich im Format unterscheiden. Aufgrund des gestellten Formats wird ein Aufruf an den GUIFrameworkAdapter ausgelöst. Dieser Aufruf wird über den Framework Controller an den Password File Manager delegiert. Hier findet eine Formatierung gemäß Kapitel 3.6 statt. Der Dateipfad, der aus dieser Formatierung entstehenden Datei, wird zurück gereicht an den Framework Controller. In der GUI wird der Dateipfad zusammen mit einer Erfolgsmeldung angezeigt.

5 Implementierung

In diesem Kapitel geht es um die Implementierung eines Prototyps nach dem in Kapitel 3 beschriebenen Entwurf.
Die im Entwurf beschriebenen Komponenten können aus einer oder mehreren Klassen bestehen. Außerdem ist es möglich, dass bestimmte Komponenten in bereits bestehende Komponenten aufgenommen wurden, da die Implementierung dadurch vereinfacht werden konnte.
Die Implementierung wird mit Java Version 1.7 vorgenommen, das Betriebssystem ist Ubuntu 12.04LTS. Die Lauffähigkeit auf anderen Betriebssystemen kann nicht garantiert werden. [JAVA 2014] [UMR 2014]
Im Folgenden wird zwischen der Implementierung des Frameworks und der Implementierung des Plugins unterschieden. Die Kommunikation der Anwendung wird im Kapitel Message Queue beschrieben. Zuletzt folgt ein Kapitel, welches Schwierigkeiten und Hürden bei der Implementierung von Programmen als Plugins aufzeigt.

5.1 Implementierung des Frameworks

5.1.1 API des Frameworks

Framework Controller:
Der Framework Controller ist die Hauptkomponente und die Schnittstelle des Frameworks. Durch die GUI werden Operationen angestoßen, welche über den GUIFrameworkAdapter an den Framework Controller delegiert werden. Der Framework Controller dient als Schaltzentrale mit erweitertem Fachwissen. Sämtliche im Framework Controller aufgerufenen Methoden werden an die zuständigen Komponenten delegiert. Die Aufrufreihenfolge bis zur ausführenden Komponente kann der entsprechenden Methodenbeschreibung entnommen werden. Dazu ist anzumerken, dass Methoden aus anderen Komponenten nur Erwähnung finden, wenn sie nicht auf denselben Namen verweisen, wie er im Framework Controller benutzt wird. So existiert die Methode sendMessage() aus dem Framework Controller lediglich zur Delegierung eines Aufrufs an den Framework Message Sender. Deshalb wird diese Methode nur im Zusammenhang mit dem Framework Controller beschrieben und nicht im Zusammenhang mit dem Framework Message Sender. Hingegen wird der Aufruf der convertFile() Methode delegiert an readPasswordFile(), eine Methode des Password File Managers. Dementsprechend wird die Methode readPasswordFile() zusätzlich im Kapitel Password File Manager beschrieben.

In Bezug auf das erweiterte Fachwissen besitzt der Framework Controller Methoden, die Kenntnis von der Funktionsweise bereits eingebundener Plugins besitzen. Diese Methoden dienen der Formatierung in ein für Plugins verständliches Format sowie für die Generierung korrekter Befehle zur Ausführung von Aufgaben. Diese Methoden sind nur innerhalb der Komponente benutzbar und können nicht von außen aufgerufen werden.
Der Framework Controller bietet folgende Methoden:

void sendMessage(CommandObject commandObject)
Diese Methode dient dem Versenden von Nachrichten an ein Plugin. Dabei ist die gewünschte Nachricht innerhalb des Command Objects gekapselt.
Der Aufruf dieser Methode wird an den Framework Message Sender delegiert. Dort wird das Command Object serialisiert und über die Message Queue an alle Plugins versendet. Plugins erkennen den Empfänger der Nachricht anhand des Adressaten des Command Objects.

void addPluginToManager(PluginObject pluginObject)
Diese Methode fügt ein Plugin Object in die Liste der registrierten Plugins ein. Die Liste wird vom Framework Plugin Manager geführt. Ab dem Zeitpunkt des Einfügens kann das Plugin in der GUI verfügbar gemacht werden.
Der Aufruf dieser Methode wird durch den Empfang einer Registrierungsaufforderung durch den Framework Message Receiver ausgelöst. Der Aufruf wird über den Framework Controller an den Framework Plugin Manager weiter delegiert. Dort wird eine Einfügeoperation in die entsprechende Datenstruktur ausgeführt.

List<PluginObject> getRegisteredPlugins()
Diese Methode gibt eine Liste aller registrierten Plugins zurück. Sie dient zur Aktualisierung der GUI, um Plugins zur Benutzung verfügbar zu machen. Der Aufruf wird durch den Benutzer ausgelöst und vom Framework Controller an den Framework Plugin Manager delegiert. Im Framework Plugin Manager wird die entsprechende Datenstruktur als Ergebnis zurückgegeben.
Um keine doppelten Reiter in der GUI zu erzeugen, muss vor dem Hinzufügen der Liste eine Überprüfung durchgeführt werden.

void showMessageInGUI(String message, String pluginName)
Diese Methode wird aufgerufen, wenn eine Nachricht empfangen und in der GUI angezeigt werden soll. Der Parameter pluginName dient der Identifikation des korrekten GUI Elements. Der Parameter message enthält die Nachricht, die in der GUI angezeigt werden soll.
Der Aufruf der Methode im Framework Controller findet durch den Framework Message Receiver statt. Dieser Aufruf wird über den GUIFrameworkAdapter an die GUI delegiert. Dort wird das entsprechende GUI Element mit der im message-Parameter enthaltenen Nachricht aktualisiert.

void convertFile(File file, String algorithm)
Diese Methode dient der Konvertierung einer Datei mit Passwörtern im Klartext in eine neue Datei. Die neue Datei enthält gehashte Passwörter. Der Parameter file identifiziert die zu konvertierende Datei. Der Parameter algorithm beschreibt den Hashing-Algorithmus zur Verschlüsselung der Klartext Passwörter.
In diesem Fall ist es nicht möglich einen nicht existierenden algorithm-Parameter anzugeben, da die Auswahl des Benutzers auf vorhandene Algorithmen beschränkt ist.
Der Aufruf dieser Methode wird an den Password File Manager delegiert, bei dem eine überladene Methode buildPasswordFile() aufgerufen wird. Die Funktionsbeschreibung der buildPasswordFile() kann dem Implementierungskapitel des Password File Managers entnommen werden.

Set<HashtagDataType> readPasswordFileReturnHashTags(File file)
Diese Methode liest die spezifizierte Datei ein, welche verschlüsselte Passwörter enthält. Als Rückgabewert liefert sie eine Menge von identifizierten Hashing-Algorithmen. Die Identifizierung der genutzten Algorithmen wird von einem externen Dienst übernommen.
Der Aufruf dieser Methode wird an den Password File Manager delegiert. Innerhalb des Password File Managers wird die readPasswordFile()-Methode aufgerufen. Daraufhin werden gesammelte Daten mit Hilfe des im Password File Manager gekapselten Hashtag Managers zurückgeführt. Die vollständige Funktionsweise der readPasswordFile() kann dem Implementierungskapitel des Password File Managers entnommen werden.

void addProcess(Integer id, String pluginName)
Diese Methode fügt eine Prozessreferenz in die Liste der laufenden Prozesse ein. Die Referenz wird von einem Plugin empfangen, sobald ein Prozess gestartet wurde.
Der Aufruf dieser Methode wird vom Framework Controller an den Process Manager delegiert. Dort wird ein Referenzobjekt generiert, welches beide Parameter enthält. Dieses Referenzobjekt wird in die entsprechende Datenstruktur eingefügt. Bei Aktualisierung des GUI Elements wird der neue Prozess angezeigt.
Zusammen ergeben beide Parameter eine eindeutige Referenz, so dass immer eine Zuordnung stattfinden kann. Somit entstehen keine doppelten Bezeichnungen für verschiedene Prozesse. Der Parameter id wird vom Plugin vergeben.

void deleteProcess(Integer id, String pluginName)
Diese Methode löscht eine Prozessreferenz aus der Liste der laufenden Prozesse. Die Referenz wird empfangen, sobald der Prozess pluginseitig terminiert. Der Aufruf dieser Methode wird vom Framework Controller an den Process Manager delegiert. Der Process Manager löscht mit Hilfe der gegebenen Parameter die Prozessreferenz aus der Datenstruktur.
Aufgrund der eindeutigen Zuordnung mittels beider Parameter kann eine korrekte Löschung erfolgen.

void clearProcessList()
Der Aufruf dieser Methode wird vom Benutzer ausgelöst und beendet alle aktiven Prozesse von Plugins.
Der Aufruf dieser Methode wird vom Framework Controller an den Process Manager delegiert. Durch den Aufruf wird das Versenden einer Kill-Nachricht an alle registrierten Plugins ausgelöst. Die Plugins beenden alle laufenden Prozesse und versenden eine Nachricht für jeden beendeten Prozess. Jede so vom Anwendungskern empfangene Nachricht hat den Aufruf der deleteProcess()-Methode zur Folge.

void buildAndSendCAFCommand(String mode, String pluginName, HashMap<String, String> info, String hashType, String passwordFilePath, String wordListFilePath, String directory)
Diese Methode wird aufgerufen, wenn der Benutzer eine Click and Fire Operation ausführt. Sie kümmert sich um die korrekte Generierung des Kommandos mit Hilfe des Plugin Managers und um das Versenden des daraus resultierenden Command Objects. Die Parameter mode, pluginName, hashType, passwordFilePath und wordListFilePath werden zur Generierung des Kommandos genutzt. Der Parameter directory wird vom Plugin zur Ausführung des Prozesses genutzt.

Methoden des Framework Controllers mit eingeschränkter Sichtbarkeit:
String buildCommandStringForCAF(PluginObject pluginObject, String mode, Integer hasType, String wordListPath, String passwordListPath)
Diese Methode kümmert sich um die Generierung des korrekten Befehls zur Ausführung eines Prozesses beim Plugin.
Zunächst wird überprüft, welche Art von Befehl generiert werden soll. Dies geschieht mit Hilfe des Parameters mode. Daraufhin wird der generische Befehl des korrespondierenden Plugin Objects entnommen und durch die konkreten Parametern ersetzt. Der Rückgabewert der Methode ist der ausführbare Befehl.

void buildHashcatFileFromType(Integer hastype, File hashcatFile, String passwordListPath)
Diese Methode formatiert eine Passwortdatei, so dass die resultierende Datei von einem Hashcat Plugin verarbeitet werden kann.
Zunächst wird aufgrund des Paramters hashType geprüft, in welchem Format die resultierende Datei vorliegen muss. Daraufhin wird jede Zeile der Datei passwordListPath entsprechend formatiert und in die Datei hashcatFile eingetragen. Dies geschieht mit Hilfe des Password File Managers, welcher die Methode readPasswordFileReturnPasswdDataTypeListe() benutzt. Die Funktionsweise dieser Methode kann dem Implementierungskapitel des Password File Managers entnommen werden.

Password File Manager:
Der Password File Manager dient der Verarbeitung von Passwortdateien. Die Methoden des Password File Managers beschränken sich auf das Erstellen neuer Passwortdateien und das Identifizieren von genutzten Hashing-Algorithmen.
Das Vorgehen bei der Benutzung des Password File Managers ist immer gleich: Der Password File Manager wird erstellt, woraufhin ein Aufruf stattfinden kann. Die Operationen generieren neue Dateien oder liefern Ergebnisse zu den spezifizierten Dateien. Somit ist es notwendig für jede neue Passwortdatei einen eigenen Password File Manager anzulegen.

void buildPasswordFile(File file, PasswdAlgorithmDataType dataType, boolean enableSalt)
Diese Methode dient der Konvertierung einer Datei mit Passwörtern im Klartext in eine neue Datei mit gehashten Passwörtern. Dabei identifiziert der Parameter file die Passwortdatei mit dem Klartext. Der Parameter dataType ist das Referenzobjekt, welches den gewünschten Hashing-Algorithmus enthält. Zudem kann der Benutzer entscheiden, ob ein Saltwert an das Passwort angehängt werden soll, bevor der Hashing-Algorithmus ausgeführt wird.
Die neu erstellte Datei hat den Namen der alten Datei mit einem angehängten „.out“. Somit kann auch aus einer übergeordneten Instanz vorausgesagt werden, welchen Namen die neue Datei haben wird.

void readPasswordFile(File file)
Diese Methode dient dem Einlesen von Passwortdateien in den Password File Manager. Die Benutzung dieser Methode trifft keine Aussage über das weitere Vorgehen des Benutzers. Sie ist Stand Alone und hat keine Auswirkung, wenn sie nicht mit weiteren Methoden kombiniert wird. Dies beruht auf der Funktionsweise des Password File Managers, bei dem zunächst eine Instanz gestartet wird. Nach der Instanziierung muss eine Passworddatei spezifiziert werden. Im Falle der readPasswordFile()-Methode wird eine Datei für die weitere Verarbeitung mit Hilfe des HashTagManagers vorbereitet.
Zum einen kann auf Einträge innerhalb der Datei zugegriffen werden, welches zeilenweise geschieht.
Zum anderen ist es möglich Informationen zu einer Datei zu erhalten. Dabei werden die genutzten Hashing-Algorithmen geprüft.

List<PasswdDataType> readPasswordFileReturnPasswdDataTypeList(File file)
Diese Methode dient dem Einlesen von Passwortdateien in den Password File Manager. Der Unterschied zur readPasswordFile()-Methode ist der Rückgabewert. Während die readPasswordFile()-Methode dem Benutzer die Freiheit lässt über das weitere Vorgehen zu entscheiden, wird bei der readPasswordFileReturnPasswdDataTypeList()-Methode direkt eine Liste von PasswdDataTypes zurückgegeben. In dieser Liste korrespondiert jeder Eintrag mit einer Zeile der eingelesenen Datei.
Daraufhin ist es möglich auf jeden Eintrag innerhalb der Datei zuzugreifen und sogar auf Teilbereiche des Eintrags zuzugreifen. Die Datei muss in einem verständlichen Format vorliegen.

5.1.2 Codebeispiele des Frameworks

Die Komponenten wurden, wie im Entwurf beschrieben, in Java umgesetzt. Eine Ausnahme ist der Click and Fire Builder, welcher mit einfachen Mittel in den Framework Controller integriert werden konnte.
Zusätzlich anzumerken ist, dass Codebeispiele aus den Kommunikationsmodulen (Framework Message Sender und Receiver, sowie Plugin Message Sender und Receiver) im Kapitel „Implementierung der Message Queue“ beschrieben werden.

Framework Initialisierung:
Der Anwendungskern wird durch die Generierung eines Framework Controllers initialisiert. Dieser erzeugt alle weiteren Instanzen und wartet daraufhin auf Nachrichten oder Befehle vom Benutzer.
Folgend zeigt die Abbildung 5.1 den Konstruktor des Framework Controllers:

```
public FrameworkController(GUIFrameworkAdapter adapter) throws IOException {
   this._frameSender = new FrameMessSender();
   this._frameReceiver = new FrameMessReceiver(this);
   this._pluginManager = new FramePluginManager();
   this._passwdManager = new PasswdManager();
   this._adapter = adapter;
   startReceiveThread();
}
```

Abbildung 5.1: Konstruktor des Framework Controllers

Hierbei ist die startReceiveThread()-Methode zu beachten, welche den Anwendungskern in den Empfangs- und Arbeitszustand führt. Dies geschieht durch die Erzeugung eines neuen Threads zum Empfang von Nachrichten, so dass der Hauptthread Befehle vom Benutzer entgegen nehmen kann.

Datei Konvertierung:
Die Datei Konvertierung wird durch den Aufruf der convertFile()-Methode ausgelöst. Dabei wird aus dem Parameter algorithm ein Referenzobjekt abgeleitet, das vom Password File Manager verarbeitet werden kann. Daraufhin wird ein neuer Password File Manager gebildet und die benötigten Parameter werden an die ausführende Methode übergeben. Aufgrund der strikten Namenskonvention bezüglich neu erstellter Dateien, kann die Ergebnisdatei an einer übergeordneten Stelle verarbeitet werden. Aus diesem Grund besitzt die Methode keinen Rückgabewert.
Folgend zeigt die Abbildung 5.2 die convertFile()-Methode des Framework Controllers:

```
public void convertFile(File file, String algorithm) throws IOException,
        NoSuchAlgorithmException, InterruptedException, IllegalAccessException {
    String[] algorithmLabelAndTag = algorithm.split("/");
    PasswdAlgorithmDataType pADT = new PasswdAlgorithmDataType(algorithmLabelAndTag[1],
        algorithmLabelAndTag[0]);
    //Muss neu belegt werden, um die Ueberschreibung aelterer Dateien nicht zu
    //gefaehrden.
    //Auch doppelte Accounts aus verschiedenen Dateien koennen sonst, aufgrund der
    //Datenverwaltung, zu falschen Ergebnissen fuehren.
    _passwdManager = new PasswdManager();
    _passwdManager.buildPasswordFile(file, pADT, "", false);
}
```

Abbildung 5.2: Die convertFile()-Methode

In diesem Beispiel wird der Aufruf der buildPasswordFile()-Methode des Password File Managers ohne Saltwert ausgeführt. Dementsprechend ist der String Parameter des Saltwerts leer und der Boolean-Wert ist false.

Bildung von Click and Fire Befehlen:
Bei der Bildung von Click and Fire Befehlen wird eine Anfrage an den Framework Plugin Manager gestellt. Gesucht wird das Plugin Object des Empfänger Plugins. Dieses Plugin Object wird benötigt, um den Befehl zum Ausführen korrekt bilden zu können.
Es wird überprüft, ob ein entsprechendes Plugin Object existiert. Sollte dies der Fall sein, muss geprüft werden, ob der Befehl an das Hashcat Plugin versendet werden soll. Dies muss aufgrund der besonderen Formatierung geschehen, die von Hashcat vorausgesetzt wird.
Sollte der Befehl an Hashcat adressiert sein, muss die vorhandene Passwortdatei entsprechend formatiert und an Stelle der alten Passwortdatei versendet werden. Falls nicht, muss keine Formatierung vorgenommen werden.
Anschließend wird die Methode buildCommandStringForCAF() ausgeführt. Diese Methode kümmert sich um die Ersetzung der Parameter im generischen Befehl durch die konkreten Parameter. Dieses Vorgehen wird genauer im Kapitel 3.7 beschrieben.
Zuletzt wird der konkrete Befehl zusammen mit allen weiteren Parametern in ein Command Object gekapselt und über den Framework Message Sender an alle Plugins versendet. Das Setzen des CAF-Flags auf true ist notwendig, um festzustellen, von welchem GUI Element der Befehl abgesetzt wurde.
In Abbildung 5.3 wird die buildAndSendCAFCommand()-Methode des Framework Controllers beschrieben:

```
public void buildAndSendCAFCommand(String mode, String name, HashMap<String,String>
    info, Integer hashType, String passwordList, String wordList, String
    directory) throws IOException, NoSuchAlgorithmException,
    InterruptedException, IllegalAccessException {

    PluginObject pluginToSend = _pluginManager.getPluginFromName(name);
    if (pluginToSend != null) {
        if (name.equals("Hashcat")) {
            File hashcatFile = new File(passwordList + ".hashcat");
            PrintWriter writer = new PrintWriter(hashcatFile);
            writer.print("");
            writer.close();
            buildHashcatPWFileFromMode(hashType, hashcatFile, passwordList);
            passwordList = hashcatFile.getAbsolutePath();
        }
        String command = buildCommandStringForCAF(pluginToSend, mode, hashType,
            wordList, passwordList);
        CommandObject commandObj = new CommandObject(name,
                directory, command, true, true, "saveOutputTo", "saveStatTo", 5);
        commandObj.setCAFFLag(true);
        sendMessage(commandObj);
    }
}
```

Abbildung 5.3: Methode zur Bildung von Click and Fire Kommandos mit anschließendem Versenden

Zu diesem Zeitpunkt ist kein Logging-Mechanismus implementiert. Dementsprechend besitzen die Parameter saveOutputTo und saveStatusTo keine Bedeutung.

Um die Methode buildAndSendCAFCommand() vollständig zu verstehen ist, es notwendig, den Code für die Methoden buildHashcatPWFileFromType() und buildCommandStringForCAF() zu erklären. Beide Methoden werden in den Kaptiteln „Formatierung der Passwortdatei für Hashcat“ und „Bildung des Befehls aus dem Plugin Object“ erklärt.

Formatierung der Passwortdatei für Hashcat:
Bei diesem Vorgang wird die Passwortdatei mit der Methode readPasswordFileAndReturnPasswdDataTypeList() eingelesen. Der Rückgabewert ist eine Liste mit PasswdDatatypes. Jeder PasswdDatatype repräsentiert eine Zeile aus der Passwortdatei.
Nun wird durch die Liste iteriert. Bei der Iteration wird jede Zeile aus der Passwortdatei entsprechend des Parameters hashType formatiert und in eine neue Datei geschrieben. Aufgrund der strengen Namenskonventionen bezüglich neu erstellter Dateien wird kein Rückgabewert benötigt, um die neue Datei aus einer übergeordneten Instanz zu finden.
Es folgt Abbildung 5.4, welche den Code der buildHashcatPWFileFromType()-Methode zeigt:

```
private void buildHashcatPWFileFromType(Integer hashType, File hashcatFile, String
    passwordList) throws IOException, FileNotFoundException, NoSuchAlgorithmException,
    IllegalAccessException, InterruptedException {

    _passwdManager = new PasswdManager();
    try {
        List<PasswdDataType> passwdDataTypeList =
            _passwdManager.readPasswordFileAndReturnPasswdDataTypeList(
            new File(passwordList));
        for (PasswdDataType data : passwdDataTypeList) {
            try (PrintWriter out = new PrintWriter(new BufferedWriter(
                new FileWriter(hashcatFile, true)))) {
            System.out.println("HASHTYPE: " + hashType);
            switch (hashType) {
                case 0:
                    out.println(data.getPassword().getPassword().replace("$", ""));
                    break;
                case 10:
                    out.println(data.getPassword().getPassword().replace("$", "")
                        + ":" + data.getPassword().getSalt().replace("$", ""));
                    break;

                    [...]

                case 1800:
                    out.println(data.getPassword().getHash());
                    break;
                case 9999:
                    out.println(data.getPassword().getPassword().replace("$", ""));
                    break;
                default:
                    out.println(data.getPassword().getHash());
                    break;
            }
        } catch (IOException e) {
        }
    }
    } catch (IllegalArgumentException ex) {
    }
}
```

Abbildung 5.4: Methode zur Einhaltung der vorausgesetzten Hashcat-Formatierung

Zu diesem Zeitpunkt ist diese Formatierung nur für Hashcat zugänglich. Für jedes weitere integrierte Passwort-Cracking-Tool muss die Formatierung pluginseitig ausgeführt werden. Dabei ist es denkbar eine zusätzliche Verwaltungskomponente einzuführen, welche eine Liste aller Plugins mit abweichenden Formaten führt. Falls ein Befehl an ein entsprechendes Plugin abgesetzt wird, kann eine Formatierung sattfinden. Dazu wäre es ebenfalls notwendig eine Repräsentation aller relevanten Formate eines Plugins mitzuführen.

Bildung des Befehls aus dem Plugin Object:
Bei der Methode buildCommandStringForCAF()wird zunächst geprüft, in welchem Modus der Befehl ausgeführt werden soll. Jeder Modus besitzt eine unterschiedliche Syntax, um korrekt ausgeführt werden zu können.
Daraufhin wird der generische Befehl des mitgelieferten Plugin Objects abgerufen und die enthaltenen generischen Parameter werden durch die konkreten Parameter ersetzt. Der Rückgabewert der Methode ist der korrekt gebildete Befehl, der in das Command Object gekapselt werden muss. Dieser Vorgang wird in der Abbildung 5.5 beschrieben.

```
private String buildCommandStringForCAF(PluginObject po, String mode,
        Integer hashType, String wordList, String passwordList) {
    String resultCommand = "";
    if (mode.equals("BruteForce")) {
        String bruteForceComm = po.getBruteForceCommand();
        //replace Keywords
        String hashTypeReplaced = bruteForceComm.replace("HASHT", hashType.toString());
        String wordListReplaced = hashTypeReplaced.replace("WORDL", wordList);
        resultCommand = wordListReplaced.replace("PASSD", passwordList);
    } else if (mode.equals("WordList")) {
        String wordListCommand = po.getWordListCommandName();
        //replace Keywords
        String hashTypeReplaced = wordListCommand.replace("HASHT", hashType.toString());
        String wordListReplaced = hashTypeReplaced.replace("WORDL", wordList);
        resultCommand = wordListReplaced.replace("PASSD", passwordList);
    } else {
    }
    return resultCommand;
}
```

Abbildung 5.5: Der Zusammenbau eines Kommandos

Identifikation von genutzten Hashing-Algorithmen

Bei der Identifikation von genutzten Hashing-Algorithmen benutzt der Password File Manager den HashtagManager. Der HashtagManager dient der Verwaltung aller Zeilen der Passwortdatei.

Sollte eine Identifikationsoperation angestoßen werden, wird die spezifizierte Passwortdatei eingelesen. Daraufhin wird ein Python Script benutzt, um jede Zeile der Datei zu identifizieren. Die Ergebnisse werden zur Verwaltung an den HastagManager gegeben. Von dort aus sind die Identifikationsdaten abrufbar. Mit Hilfe dieser Daten kann nun eine Aufbereitung durchgeführt werden.

Folgend zeigt die Abbildung 5.6 den Code der getHashTag()-Methode. Diese übernimmt die Identifikation eines Hashwerts einer Zeile der Passwortdatei. Zur Identifikation wird ein Python Script genutzt:

```
public List<HashTagDataType> getHashTag(String hash) throws IOException, InterruptedException,
        IllegalAccessException {
    File hashTagFile = new File("/PCaaS_Framework_src/Framework/tools/HashTag.py");
    String[] command = {"python", hashTagFile.getPath(), "-sh", hash};
    ProcessBuilder processBuilder = new ProcessBuilder(command);

    Process process = null;
    process = processBuilder.start();

    InputStream inputStream = process.getInputStream();
    InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
    BufferedReader bufferedReader = new BufferedReader(inputStreamReader);

    String line = null;
    List<HashTagDataType> hashTagList = new ArrayList<HashTagDataType>();

    while ((line = bufferedReader.readLine()) != null) {
        HashTagDataType hashTagDataType = processHashTag(line);
        if (hashTagDataType != null) {
            hashTagDataTypes.add(hashTagDataType);
            hashTagDataType.incrementCounter();
            hashTagList.add(hashTagDataType);
        }
    }
    int exitValue = Integer.MIN_VALUE;
    exitValue = process.waitFor();
    if (exitValue == 0) {
        return hashTagList;
    } else {
        throw new InterruptedException(this.getClass().getSimpleName() + ": Error with exit
            code by ProcessBuilder.");
    }
}
```

Abbildung 5.6: Methode zur Identifikation genutzter Hashing-Algorithmen

Der String Parameter der Methode ist der zu identifizierende Hashwert. Alle notwendigen Parameter zur Ausführung der Identifikation werden in der command Liste gehalten und an den Java Process Builder übergeben. Der Prozess wird gestartet und die Ergebnisse werden zur Verwaltung an den HashtagManager übergegeben. Der Rückgabewert der Methode ist eine Liste aller möglicherweise genutzten Hashing-Algorithmen für den spezifizierten Hashwert.

5.1.3 Framework Tests

Das Framework wurde sowohl Komponenten- als auch Integrationstests unterzogen. Die Komponententests umfassen das korrekte Senden und Empfangen von Nachrichten und das korrekte Management aller Referenzobjekte für Plugins und Prozesse. Bei Testen der Kommunikation wurde eine Nachricht entsprechend der Funktionsweise des Plugins serialisiert und über den Plugin Message Sender versendet. Der Empfang dieser Nachricht wird durch eine entsprechende Empfangsklasse simuliert, da der reale Empfang durch den Anwendungskern stattfindet. Beim Testen der Verwaltungsklassen wurden alle auf der entsprechenden Datenstruktur ausführbaren Operationen getestet, zu diesem Zweck wurden benötigte Referenzobjekte erstellt.
Der Integrationstest des Frameworks umfasst die korrekte Bildung und Ausführung eines Kommandos beginnend mit dem Framework Controller. Dabei wurde ein Command Object basierend auf vorgegebenen Parametern erstellt und vom Framework Message Sender

versendet. Der Empfang dieser Nachricht wird von einer entsprechenden Empfangsklasse simuliert, da der reale Empfang der Nachricht durch ein Plugin stattfindet.

5.2 Implementierung des Plugins

Plugins besitzen keine Schnittstelle wie es beim Anwendungskern der Fall ist. Stattdessen werden sie direkt an die Message Queue des Anwendungskerns gebunden und befinden sich in einem ständigen Empfangszustand. Die Schnittstelle beschränkt sich somit auf Nachrichten die empfangen werden. Dabei existieren zwei Arten von Nachrichten, welche ein Plugin verarbeiten kann:

1. Befehlsnachrichten: Bei diesen Nachrichten handelt es sich um Aufforderungen zur Prozessausführung. Die empfangenen Nachrichten sind an das Plugin adressiert und enthalten ein Kommando, welches als Prozess verarbeitet werden kann.

2. Prozessverwaltungsnachrichten: Bei diesen Nachrichten handelt es sich um eine Aufforderung alle aktiven Prozesse zu beenden. Die empfangene Nachricht ist adressiert an „Process Management“ und enthält als Kommando den Text „KILLPROCESS“.

5.2.1 Nachrichtenverarbeitung des Plugins

Registrierungsaufforderung:
Nach der Initialisierung versendet das Plugin eine Registrierungsaufforderung an den Anwendungskern. Diese Nachricht kapselt ein Plugin Object und dient der Verwaltung der Plugins, so dass dem Anwendungskern bekannt ist, welche Plugins existieren und wie die generischen Strings zur Verarbeitung von Click and Fire Befehlen aussehen.

Befehlsnachrichten:
Nachdem ein Plugin initialisiert wurde, verschickt es eine Registrierungsaufforderung und begibt sich in den Empfangszustand.
Falls eine Befehlsnachricht empfangen wird, wird eine Process Control Instanz gestartet. Diese Instanz führt den Befehl aus.
Der laufende Prozess wird in die ProcessManagement Komponente eingetragen. Eine „ADDPROCESS“ Nachricht wird zusammen mit einem Referenzobjekt des Prozesses an den Anwendungskern verschickt. Dies führt beim Anwendungskern dazu, dass der laufende Prozess in die ProcessManagement Komponente des Frameworks eingetragen wird.

Prozessverwaltungsnachrichten:
Nachdem ein Plugin initialisiert wurde, verschickt es eine Registrierungsaufforderung und begibt sich in den Empfangszustand.
Beim Empfang einer Prozessverwaltungsnachricht wird der Command String geprüft, welcher das Kommando „KILLPROCESS“ enthalten muss. Daraufhin wird die Methode killAllProcesses() des Plugin Process Managers aufgerufen. Diese übernimmt die Terminierung aller laufenden Prozesse.
Bei der Terminierung eines Prozesses versendet die entsprechende Process Control Instanz eine Nachricht an den Anwendungskern. Diese Nachricht enthält neben einer „DELETEPROCESS“ Mitteilung auch das Referenzobjekt des zu löschenden Prozess. Auf diese Weise kann der Anwendungskern eindeutig feststellen, welcher Prozess beendet wurde.

5.2.2 Codebeispiele des Plugins

Empfangsschleife:
Bei der Empfangsschleife handelt es sich um eine while(true)-Schleife mit blockierendem Empfang durch die Message Queue. Auf diese Weise ist aktives Warten vermieden worden.
Beim Empfang einer Nachricht löst sich die Blockade, so dass die empfangene Nachricht deserialisert werden kann. Daraufhin wird die Instanz der Nachricht überprüft, um feststellen zu können an welches Plugin die Nachricht adressiert ist. Zuletzt wird der Inhalt der Nachricht geprüft, um den entsprechenden Arbeitsschritt einzuleiten.

```
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    CommandObject commandObj = null;
    try {
        ByteArrayInputStream bis = new ByteArrayInputStream(
                delivery.getBody());
        ObjectInputStream ois = new ObjectInputStream(bis);
        commandObj = (CommandObject) ois.readObject();
        ois.close();
        bis.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    }
    if (commandObj.getProgramInstance().equals("John the Ripper")) {
        System.out.println(" [x] Received '" + commandObj.getCommand() + "'");
        ProcessControl pc = ProcessControl.create(commandObj.getCommand(),
            commandObj.getDirectory(), commandObj.getInterval(), pluginMessSender,
            commandObj.getCAFFlag(), _processManager);

        //Verwaltung der aktiven Prozesse
        _processManager.addProcessToList(pc);
        pluginMessSender.sendMessageWithComment("Process Management", "ADDPROCESS", false,
            _processManager.getStringFromProcess(pc));

    } else if(commandObj.getProgramInstance().equals("Process Management")
            && commandObj.getCommand().equals("KILLPROCESSES")){
        //ProcssControl sendet eine DELETEPROCESS-Nachricht bei Terminierung
        _processManager.killAllProcesses();
    } else {
        //Objekt ist an ein anderes Plugin adressiert
    }
}
```

Abbildung 5.7: Empfang und Verarbeitung von Nachrichten

Die Abbildung 5.7 beschreibt den Vorgang der Nachrichtenverarbeitung. Die statische Methode create() der Process Control Klasse ist eine Fabrik-Methode, so dass kein direkter Zugriff auf den Konstruktor möglich ist.
Der blockierende Aufruf der Message Queue steht in der zweiten Zeile: consumer.nextDelivery().

Prozessverarbeitung:
Nach dem Start einer Instanz der Process Control Klasse wird die run()-Methode aufgerufen. Diese Methode kümmert sich um die korrekte Verarbeitung des Prozesses. Dabei wird die run()-Methode als neuer Thread gestartet, um den Hauptthread des Plugins nicht zu blockieren. Jeder gestartete Prozess wird mit Hilfe des Java ProcessBuilders gekapselt und ausgeführt. [JAVADOC]

Die run()-Methode beginnt mit der Formatierung des Befehls in einzelne Parameter. Diese Parameter werden als Liste an den Java ProcessBuilder gegeben. Daraufhin wird das Arbeitsverzeichnis gesetzt und der Arbeitsprozess wird gestartet.
Nach dem Start werden benötigte Verwaltungsprozesse angestoßen. Bei Diesen handelt es sich um einen Überwachungsprozess zur Terminerung des Arbeitsprozesses sowie um den Statusprozess, welcher Statusnachrichten zum Arbeitsprozess abfragt und Diese an den Anwendungskern verschickt. Beide Prozesse brauchen dafür eine Referenz auf das ProcessBuilder Objekt.
Der Statusprozess benötigt Referenzen auf den Überwachungsprozess und den Plugin Message Sender. Diese werden benötigt, um Nachrichten zu versenden und um die Terminierung des Arbeitsprozesses festzustellen.
Nun können die benötigten Reader und Writer für den ProcessBuilder generiert werden. Diese werden benötigt, um den Output zu lesen und um Nachrichten an den ProcessBuilder zu senden.
Daraufhin wird eine blockierende while-Schleife gestartet, welche bis zur Terminierung des ProcessBuilders läuft (Der ProcessBuilder beendet mit einer null terminierten Zeile).
Nachdem der ProcessBuilder terminiert ist, wird eine „DELETEPROCESS“ Nachricht zusammen mit der Prozessreferenz an den Anwendungskern versendet. Zuletzt wird der Prozess aus dem Plugin Process Manager gelöscht.
Folgend beschreibt Abbildung 5.8 den Code der run()-Methode der Process Control Klasse:

```
@Override
public void run() {
    //Argumentliste bauen
    String text = _jtRArguments;
    String[] formattedText = text.split(" ");

    //Befehl uebergeben, Directory setzen
    ProcessBuilder proB = new ProcessBuilder(formattedText);
    proB.directory(new File(_directory));
    proB.redirectErrorStream(true);
    Process process;
    try {
        process = proB.start();
        _process = process;
        //TerminierungsÃ¼berwachung des Prozesses
        ProcessMonitor proM = ProcessMonitor.create(process);
        _processMonitor = proM;
        //StatusAbfrage
        ProcessWriterAsync.create(proM, process, _plugMessSender, _directory, _interval,
                _cafFlag);

        //schreibender Zugriff
        OutputStream os = process.getOutputStream();
        OutputStreamWriter osw = new OutputStreamWriter(os);
        BufferedWriter bw = new BufferedWriter(osw);

        //lesender Zugriff
        InputStream is = process.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        String line;
        System.out.printf("Output of running %s is:\n", Arrays.toString(formattedText));

        //Prozess Output
        while ((line = br.readLine()) != null) {
            _plugMessSender.sendMessage("John the Ripper", line, _cafFlag);
        }
        //DELETEPROCESS-Nachricht an das Framework, Prozess ist terminiert
        this._plugMessSender.sendMessageWithComment("Process Management", "DELETEPROCESS",
            false, _processManager.getStringFromProcess(this));
        //loeschen des Prozesses aus dem Process Manager
        _processManager.deleteProcessFromList(this);
        bw.close();

    } catch (IOException e) {
        System.out.println("Exception in ProcessConrol, Stream is closed due to killed
            Process");
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

Abbildung 5.8: Start, Ausführung und Verwaltung eines angefragten Diensts

5.2.3 Plugin Tests

Mitgelieferte Plugins werden Komponententests unterzogen. Ein Integrationstest ist aufgrund des Zusammenspiels aller Komponenten bereits durch den Komponententest impliziert. Die Tests umfassen die korrekte Ausführung der Kommunikation und das korrekte Management aller Referenzobjekte für Prozesse.

Beim Testen der Kommunikation werden serialisierte Command Objects über den Plugin Message Sender versendet. Der Empfang dieser Objekte wird durch eine entsprechende Empfangsklasse simuliert, da der reale Empfang durch den Anwendungskern stattfindet.

Beim Testen der Verwaltungsklassen wurden alle auf der entsprechenden Datenstruktur ausführbaren Operationen getestet, zu diesem Zweck wurden benötigte Referenzobjekte erstellt.

5.3 Implementierung der Message Queue

Die Message Queue ist der Kommunikationskanal zwischen dem Anwendungskern und den Plugins. Sie arbeitet mit Nachrichten. Zur Implementierung der Message Queue wird RabbitMQ verwendet. [RABB 2014]
Um die Message Queue dem Entwurf nach implementieren zu können, müssen Regeln bei Versenden und Empfangen von Nachrichten erfüllt sein:

1. Wenn der Anwendungskern eine Nachricht versendet, muss diese Nachricht von jedem Plugin empfangen werden, solange das Plugin empfangsbereit ist. Dies bedeutet, dass eine Nachricht erst gelöscht werden darf, wenn alle Plugins die Nachricht empfangen haben oder jedes Plugin eine eigene Kopie der Nachricht empfangen hat.

2. Versendet ein Plugin eine Nachricht, kann die Nachricht erst gelöscht werden, falls der Anwendungskern sie empfangen hat. Diese Regel muss auch in dem Fall erfüllt sein, wenn der Anwendungskern nicht existiert.

Beide Anforderungen können mit einfachen Mitteln gelöst werden: In der Implementierung existieren zwei verschiedene Message Queues. Aus der Perspektive des Anwendungskerns existiert eine Message Queue zum Versenden und Eine zum Empfangen von Nachrichten.

Sendende Message Queue:
Die sendende Message Queue dient dem Versenden von Nachrichten an alle Plugins. Um zu erreichen, dass alle Plugins eine bestimmte Nachricht erhalten, wird die Message Queue an einen Exchange-Server angebunden. Der Exchange-Server wird von RabbitMQ angeboten. Sämtliche vom Anwendungskern versendeten Nachrichten werden von der Message Queue an den Exchange-Server weitergeleitet. Somit hat die Message Queue selbst keine Informationen über aktive Plugins und kein Wissen darüber, wie viele Empfänger eine Nachricht hat.
Der Exchange-Server, welcher nun die Nachricht besitzt, befindet sich im Fanout-Modus. In diesem Modus bekommt jedes Plugin, dass an den Exchange-Server angebunden ist, eine Kopie der Nachricht. Plugins können so entscheiden, ob eine Nachricht verworfen oder benutzt werden soll. Alle Plugins müssen bei der Initialisierung an den Exchange-Server angebunden werden, um Nachrichten empfangen zu können.

```
public void sendMessage(CommandObject message) {
    Connection connection;
    try {
        connection = _factory.newConnection();
        Channel channel = connection.createChannel();

        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        byte[] bytes;
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        try {
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(message);
            oos.flush();
            oos.reset();
            bytes = baos.toByteArray();
            oos.close();
            baos.close();
        } catch (IOException e) {
            Logger.getLogger("bsdlog").error("Unable to write to output stream",e);
        }

        channel.basicPublish(EXCHANGE_NAME, "", null, bytes);
        System.out.println(" [x] Sent '" + message + "'");

     } catch (IOException e1) {
         e1.printStackTrace();
     }
}
```

Abbildung 5.9: Das Versenden einer Nachricht

Abbildung 5.9 beschreibt die sendMessage()-Methode des Anwendungskerns, welche die Anbindung an den Exchange-Server übernimmt.
Der Parameter _factory ist eine Connection-Factory von RabbitMQ und erzeugt neue Verbindungen. Innerhalb der Verbindung wird ein Kanal erzeugt, der an einen Exchange-Server angebunden wird. Nun kann die Nachricht serialisiert und versendet werden.
Es ist anzumerken, dass nur bei der ersten Ausführung der Methode ein Exchange-Server erzeugt wird. Jeder weitere Aufruf bemerkt, dass bereits ein aktiver Exchange-Server mit identischem Namen (EXCHANGE_NAME) existiert und benutzt Diesen.

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
String exchangeQueueName = channel.queueDeclare().getQueue();
channel.queueBind(exchangeQueueName, EXCHANGE_NAME, "");

System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(exchangeQueueName, true, consumer);

while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
```

Abbildung 5.10: Initialisierung eines Message Queue Exchange Servers

Abbildung 5.10 stellt die Anbindung eines Plugins an die Message Queue dar.
Hierbei wird eine direkte Anbindung zu einem Exchange-Server erzeugt. Um die Anbindung an den korrekten Exchange-Server sicher zu stellen, muss das Plugin den Namen des Exchange-Servers kennen.
Wie im obigen Beispiel wird ein Exchange-Server erzeugt. Falls bereits ein Server vorhanden ist, wird Dieser Verbindung genutzt. Bei der erstmaligen Erzeugung des Exchange-Servers ist es für ein Plugin nicht relevant, ob auf der anderen Seite ein Nachrichten Sender existiert oder nicht. Es erzeugt den Exchange-Server und bindet sich normal an. Wenn der Anwendungskern nach dem Plugin gestartet wird, bindet sich er sich an den vom Plugin erzeugten Exchange-Server an.

Empfangende Message Queue:
Die empfangende Message Queue dient dem Empfang aller von Plugins versendeten Nachrichten. Die Erzeugung eines Exchange-Servers, wie bei der sendenden Message Queue, ist nicht nötig. Das Modul des Anwendungskerns, welches die Nachrichten entgegen nimmt, ist dabei das konsumierende Ende des Kommunikationskanals. Da wir sicher davon ausgehen können, dass nur ein Anwendungskern existiert, kann eine Nachricht einfach empfangen und dann gelöscht werden.
Alle Plugins binden sich bei der Initialisierung an das sendende Ende des Kommunikationskanals. Jede Instanz kann Nachrichten an die Message Queue übergeben, welche dann vom Anwendungskern empfangen werden.

```
public void sendMessage(String instance, String message, boolean cafFlag) throws IOException {
    _connection = _factory.newConnection();
    _channel = _connection.createChannel();
    _channel.queueDeclare(QUEUE_NAME, false, false, false, null);

    CommandObject commandObj = new CommandObject(instance, "",
            message, true, true, "saveOutputTo", "saveStatusTo", 5);
    commandObj.setCAFFLag(cafFlag);

    byte[] bytes;
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    try {
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(commandObj);
        oos.flush();
        oos.reset();
        bytes = baos.toByteArray();
        oos.close();
        baos.close();
    } catch (IOException e) {
        Logger.getLogger("bsdlog").error("Unable to write to output stream",e);
    }
        _channel.basicPublish("", QUEUE_NAME, null, bytes);
}
```

Abbildung 5.11: Das Versenden einer Nachricht durch ein Plugin

Abbildung 5.11 zeigt die sendMessage()-Methode des Plugins, welche sich an den Kommunikationskanal anbindet und eine Nachricht versendet.
Wie bereits im Abschnitt „Sendende Message Queue“ beschrieben, wird ein Sendekanal erstellt, wenn keiner vorhanden ist. Andernfalls wird der vorhandene Kanal benutzt.

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false, false, false, null);

QueueingConsumer consumer = new QueueingConsumer(channel);
channel.basicConsume(QUEUE_NAME, true, consumer);

while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
```

Abbildung 5.12: Anbindung an die Message Queue

Äquivalent zur Abbildung 5.11, zeigt Abbildung 5.12 den empfangenden Teil des Anwendungskerns.
Bei Fragen zu einzelnen Schritten der Erstellung einer Message Queue kann die RabbitMQ-Webseite zu Rate gezogen werden. [RABB 2014]

5.4 Anmerkungen zum Einbinden von Programmen

Die Anwendung wird mit zwei implementierten Programmen geliefert, diese sind John the Ripper und Hashcat.
Während der Implementierung der Programme als Plugins tauchten Hürden auf, die nicht vorhersehbar waren. Es gab Probleme bei der Prozessausführung, dem Abfragen von Statusnachrichten, dem korrektem Lesen der Ergebnisse aus den Streams und der von den Programmen geforderten Formatierung der Passwortdateien. In diesem Kapitel werden die Lösungen für die zeitintensivsten Probleme beschrieben.

Hashcat:
Da Hashcat ein Plattform übergreifendes Programm ist, benötigt es ein bestimmtes Format zur Verarbeitung von Passwörtern und Saltwerten. Zusätzlich muss ein numerischer Hashtyp angegeben werden. Der Hashtyp wird durch das genutzte Hashing-Verfahren und die Verarbeitung des Saltwerts bestimmt.
Folgend ein Auszug aus dem offiziellen Hashcat Wiki:

Hash-Mode	Hash-Name	Example
0	MD5	8743b52063cd84097a65d1633f5c74f5
10	md5($pass.$salt)	01dfae6e5d4d90d9892622325959afbe:7050461
20	md5($salt.$pass)	f0fda58630310a6dd91a7d8f0a4ceda2:4225637426
30	md5(unicode($pass).$salt)	b31d032cfdcf47a399990a71e43c5d2a:144816

Abbildung 5.13: Auszug aus der Format-Tabelle von Hashcat [HDOC 2014]

In Abbildung 5.13 ist zu erkennen, dass für ein MD5 gehashtes Passwort ohne Saltwert der Hashtyp null benutzt werden muss. Falls ein Salt verwendet worden ist, muss der Hashtyp zehn oder zwanzig benutzt werden, je nachdem auf welche Weise der Saltwert angewendet wurde.

Zusätzlich ist erkennbar, dass die vorausgesetzte Formatierung zur Verarbeitung PASSWORD:SALT sein muss. Dieses Format ist von einer Linux-Passwd formatierten Datei nicht vorgesehen. Die zu crackende Passwortdatei muss vor der Übergabe an Hashcat entsprechend formatiert werden.
Für Hashcat ist dieses Problem mit Hilfe des Password File Managers gelöst worden. Über den Password File Manager ist es möglich eine Passwortdatei einzulesen. Nach dem Einlesen kann auf jeden Eintrag innerhalb der Datei zugegriffen werden. So kann jeder Eintrag umgeformt werden.
Genaue Informationen über die Formatierung kann dem Abschnitt „Formatierung der Passwortdatei für Hashcat“ aus dem Kapitel 4.1.2 entnommen werden.

John the Ripper:
Bei der Implementierung von John the Ripper als Plugin kam es zu Problemen bei der Abfrage von Statusnachrichten und dem korrekten Lesen von Ergebnissen aus den benutzten Streams.
Sobald ein Arbeitsprozess gestartet wird, instanziiert das Plugin einen neuen Prozess. Dieser Prozess versendet Stausabfragen zum laufenden Arbeitsprozess an den Anwendungskern.
Bei der Benutzung von John the Ripper innerhalb eines Terminals können Satusnachrichten zu einem laufenden Arbeitsprozess durch das Betätigen der ENTER-Taste abgefragt werden. Der Status wird innerhalb desselben Terminals, während der Ausführung, angezeigt.
Dieses Verhalten ist mit dem Java Process Builder nicht simulierbar. Es nicht möglich, die Eingabe einer bestimmten Taste als Auslöser für die Generierung einer Statusnachricht an den Arbeitsprozess zu übergeben. Auch erste Versuche den Befehl [john --status] an den Arbeitsprozess zu übergeben zeigten keine Resultate und es stellte sich heraus, dass fundierteres Wissen über interne John the Ripper Prozesse notwendig ist. [JDOC 2014]
Die Umgehung dieses Problems stellte sich als zeitintensiver heraus, als ursprünglich angenommen und beinhaltete die Recherche des Quellcodes von John the Ripper.
Das Problem konnte gelöst werden, indem ein Kill-Signal abgesetzt wird. Das Kill-Signal in Linux ist kein Befehl zur Terminierung eines Prozesses. Stattdessen ist es ein Befehl zur Interprozesskommunikation. Ein Kill-Signal wird an ein Prozess versendet und spezifiziert einen Zahlenwert. Dieser Wert veranlasst die Aktualisierung von Konfigurations- und Statusdateien. Auf diese Weise kann ein aktueller Status eingelesen und versendet werden. [UMR 2014]
Abbildung 5.14 zeigt das Versenden eines Kill-Signals und das Einlesen der Statuszeile.

```
while (!(_pm.isComplete())) {
    if (statusTime + interval <= System.nanoTime()) {
        Process statusProcess = statusProcessBuilder.start();
        BufferedReader brStatErr = new BufferedReader(
            new InputStreamReader(statusProcess.getErrorStream()));
        BufferedWriter bwStat = new BufferedWriter(
            new OutputStreamWriter(statusProcess.getOutputStream()));

        new ProcessBuilder("kill", "-1", "" + _workProcessPID).start().waitFor();

        String line;
        while ((line = brStatErr.readLine()) != null) {
            //sendet zurueck an das Framework
            _sender.sendMessage("John the Ripper", line, _cafFlag);
        }
```

Abbildung 5.14: Anfrage von Status Nachrichten zu einem laufenden Prozess

Die erste while-Schleife terminiert, sobald der Arbeitsprozess (_pm) beendet wird. Innerhalb dieser Schleife bestimmt der Parameter interval, wie oft eine Anfrage an den Arbeitsprozess abgesetzt wird.
Es wird ein Kill-Prozess mit Hilfe des Java Process Builders gestartet, der ein Signal an den Arbeitsprozess sendet. Dieses Signal (-1) veranlasst den Arbeitsprozess seine Statusdatei zu aktualisieren und den aktualisierten Status als Ergebnis zurück zu geben. Das Ergebnis wird an den ErrorStream des Prozesses übergeben und nicht an den InputStream.
Um die Linux-Prozess-ID (_workProcessPID) des Arbeitsprozesses zu erhalten, waren ebenfalls gesonderte Schritte notwendig. Dabei wurde aufgrund einer Referenz auf den Arbeitsprozess die zugehörige ID ermittelt. Die konkrete Ausführung wird in Abbildung 5.15 dargestellt.

```
public static int getUnixPID(Process process) throws Exception {
    if (process.getClass().getName().equals("java.lang.UNIXProcess")) {
        Class cl = process.getClass();
        Field field = cl.getDeclaredField("pid");
        field.setAccessible(true);
        Object pidObject = field.get(process);
        return (Integer) pidObject;
    } else {
        throw new IllegalArgumentException("Needs to be a UNIXProcess");
    }
}
```

Abbildung 5.15: Anfrage einer Prozess-ID für Linux Betriebssysteme

Bei Hashcat existiert dieses Problem nicht, da Hashcat automatisch Statusnachrichten zu einer laufenden Cracking-Session versendet.

5.5 Mögliche Fehlerquellen bei der Ausführung auf fremden Maschinen

1. Arbeitsverzeichnis: Die Anwendung muss im korrekten Arbeitsverzeichnis gestartet werden. Alle benötigten Dateien werden bei der Initialisierung über relative Pfade angesteuert. Ein relativer Pfad beginnt immer im Ordner PCaaS_Framework_src.

2. Ausführbare Dateien für das Plugin: Ein Plugin arbeitet immer mit einer ausführbaren Datei zusammen (Beispiel: „./hashcat.bin“). Falls ein Prozess nicht ausführbar ist, muss das entsprechende Plugin Object angepasst werden, so dass die korrekte Datei ausgeführt wird.

3. Anwendungskern und Plugins: Anwendungskern und Plugins müssen auf demselben System residieren. Dieser Umstand folgt daraus, dass die benutzte Message Queue unter „localhost“ gestartet wird.

4. Installation von Passwort-Cracking Tools und die Path-Variable: Je nachdem wie die Passwort-Cracking Tools installiert sind, ist eine Anpassung der Plugin Objects notwendig. Sollte eine ausführbare Datei der Path-Variablen des Host-Systems hinzugefügt worden sein, muss kein absoluter Pfad angegeben werden.

5. Tests: Zur korrekten Ausführung der Tests sollten alle Message Queues geleert werden, da ansonsten bereits enthaltene Nachrichten konsumiert werden können, welche einen negativen Testausgang hervorrufen.

6. Hashing-Algorithmen: Passwort-Cracking-Tools können unterschiedliche Hashing-Algorithmen verarbeiten und benötigen teilweise ein bestimmtes Format für eine konkrete Implementierung eines Algorithmus. Eine entsprechende Meldung („No password hashes loaded“ oder „Skipping Line [x]“) kann also unterschiedliche Ursachen haben. Bevor Änderungen am Quellcode vorgenommen werden, sollten die beschriebenen Ursachen geprüft werden.

6 Zusammenfassung

Das Ziel dieser Arbeit war der Entwurf und die Implementierung eines Frameworks zur Benutzung von Passwort-Cracking-Tools. Zusätzlich sollte es möglich sein, generische Befehle zur Benutzung von Passwort-Cracking-Tools zu formulieren.
Zu diesem Zweck wurde eine Anforderungsanalyse durchgeführt, so dass die Anforderungen an den Entwurf extrahiert werden konnten. Auf Basis der Anforderungen konnte eine Architektur festgelegt werden, welche eine Zusammenarbeit des Anwendungskerns und den zugehörigen Plugins möglich macht. Um eine möglichst geringe Kopplung zwischen dem Anwendungskern und den Plugins zu erreichen, wurde eine nachrichtenbasierte Kommunikation eingesetzt. Darüber hinaus konnte aufgezeigt werden, wie eine den Anforderungen entsprechende Kapselung und Ausführung von angefragten Diensten und den daraus resultierenden Prozessen konzipiert werden kann. Auch das Erzeugen generischer Befehle auf Grundlage einfacher Parameter, ohne Vorwissen über die Benutzung von Cracking-Software, konnte erreicht werden.
Der Aufwand zur Erweiterung der Software durch weitere Plugins wurde minimal gehalten. Zudem wurde ein Graphical User Interface ausgearbeitet, um die Bedienung der Anwendung zu erleichtern.
Durch diese prototypische Implementierung konnte gezeigt werden, dass das Ziel dieser Arbeit mit vorhandener Technologie erreicht werden kann.

6.1 Fazit

Die in der Zielsetzung beschriebenen Anforderungen an die Anwendung konnten zum Großteil erfüllt werden. Die Integration von Passwort-Cracking-Tools ist durch die Kapselung als Plugin möglich. So können Aufträge an ein Cracking-Tool über eine Message Queue erteilt und ausgeführt werden.
Die Implementierung des Click and Fire Builders erlaubt die Nutzung der Anwendung ohne Fachwissen. Der Benutzer muss in diesem Fall die benötigten Parameter angeben, woraufhin die Erzeugung eines korrekten Auftrags von der Anwendung übernommen wird.
Die Nutzung von Cracking-Tools zum Entschlüsseln von Passwörtern ist durch die Kapselung als Plugin bereits impliziert und ohne Beschränkungen möglich.
Aufgrund einer prototypischen Implementierung war es nicht möglich sämtliche Anforderungen abzudecken.
Um die Bedienung der Anwendung noch einfacher zu machen, sollte die Angabe der Parameter soweit automatisiert werden, dass der Benutzer einzig die Passwortdatei angeben muss. Zu diesem Zweck muss eine von der Anwendung bereitgestellte Wortliste eingeführt

werden. Auch die Erkennung der Hashtypen der Passwortdatei muss dann automatisiert stattfinden. Die Methoden, welche eine solche Erkennung möglich machen, sind bereits voll einsatzfähig. Zu diesem Zeitpunkt muss die Operation noch vom Benutzer selbst angestoßen werden.
Die Implementierung eines Logging-Systems ist zu diesem Zeitpunkt abwesend. Fehlerfälle und das Exceptionhandling konnten nicht einwandfrei abgedeckt werden, aus diesem Grund sind fehlerhafte Eingaben oder Parameter nicht optimal auf Fehlerfälle abgebildet. Die Erzeugung von generischen Befehlen ist beschränkt auf zwei Basisoperationen. Diese wurden aus der Schnittmenge aller begutachteten Cracking-Tools extrahiert. Mit etwas mehr Zeit und Ressourcen ist es möglich Diese zu erweitern.
Die Implementierung der Unix Crypt Funktionalitäten konnte nicht nachgestellt werden. Die Anwendung ist in der Lage korrekte Hashwerte für MD5sum und SHA512sum zu bilden. Die Verarbeitung als Ersatz zur /etc/passwd Datei ist nicht möglich. Um die korrekte Verarbeitung zu gewährleisten, muss recherchiert werden, welche Mechanismen beim Hashing von Passwörtern in Linux benutzt werden. Eine mögliche Fehlerquelle ist die Anzahl der von Linux angewendeten Runden bei der Erzeugung des Hashwerts eines Passworts. Von Java angebotene Hashing-Algorithmen funktionieren einwandfrei.
Die Anwendung zeigt auf, dass Erweiterungen und Weiterentwicklungen einfach umsetzbar sind. Diese sind durch eine geeignete Architektur und Modularisierung ebenso zeitsparend.

Bei einer Recherche fällt auf, dass Produkte mit dem Schwerpunkt dieser Arbeit nicht existieren, die Problemstellung jedoch allgegenwärtig ist. Unternehmen und Firmen ist es mit dieser Anwendung möglich, Teile Ihrer eigenen Passwortsicherheit zu begutachten. Auch die mehrfache Überprüfung pro Zeitintervall ist einfach und verursacht keine zusätzlichen Kosten. Ein weiterer Vorteil ist die eingeführte Abstraktionsebene durch den Click and Fire Builder, welche die Bedienung selbst branchenfremden Personen möglich macht, da alle zur Überprüfung relevanten Teile des Vorgangs durch entsprechende GUI Ansichten bedient werden können. Ziel der Abstraktionsebene ist, den Fokus der Bedienung auf die Passwortdatei selbst zu lenken. Durch diesen Fokus kann der gesamte Vorgang der Überprüfung intuitiv gelenkt und die Bedienung erleichtert werden.
Ein weiterer Vorteil besteht in der Framework Architektur, wodurch es jedem Kunden möglich ist seine eigenen Plugins zu definieren. So können bereits genutzte Passwort-Cracking-Tools in ein Plugin gekapselt und ohne Umstellung genutzt werden. Dies erleichtert nicht nur den Umstieg auf ein neues Produkt, sondern erhöht auch die Akzeptanz durch die Mitarbeiter.

6.2 Ausblick

Die Erweiterung der Anwendung kann auf folgende Punkte reduziert werden: Erweiterung des Funktionsumfangs der Abstraktionsebene, Erweiterung des Funktionsumfangs der Hash-Erkennungs- und Hash-Verschlüsselungsmechanismen, bessere Modularisierung und Ausarbeitung eines Nachrichtenprotokolls.

Bei der Erweiterung des Funktionsumfangs ist eine weitere Analyse aller vorhandenen Passwort-Cracking-Tools notwendig. Ziel sollte die Extrahierung weiterer Schnittmengenoperationen sein, wodurch die Abstraktionsebene verbessert werden kann.
Die Erkennung von Hashing-Algorithmen ist nicht immer einwandfrei und genau, deswegen werden für die meisten Fälle mehrere Algorithmen als Lösung vorgeschlagen. Die Verbesserung der Erkennung ist ein eigenes Forschungsgebiet und kann nur mit einer Menge Aufwand erreicht werden. Eine Erwähnung sollte es trotzdem finden.
Die angebotenen Hashing-Verfahren zur Verschlüsselung von Passwörtern im Klartext können erweitert werden. Momentan beschränkt sich die Auswahl auf Verfahren, welche von Java-Bibliotheken angeboten werden.
Die Verbesserung der Modularisierung kann durch reinen Ressourcenaufwand erreicht werden. Dabei sollten Befugnisse und Aufgaben besser aufgespalten werden. So ist beispielsweise der Message Receiver neben dem Empfangen von Nachrichten, auch für das Verteilen von Nachrichten verantwortlich.
Zu diesem Zeitpunkt können Plugins nur in Java implementiert werden. Dies liegt an der Kommunikation mittels serialisierter Java Objekte. Sollte eine Erweiterung auf alle Programmiersprachen erforderlich sein, ist es notwendig ein Kommunikationsprotokoll einzuführen. Dieses Protokoll kann dann sprachunabhängig eingesetzt werden, weil es ohne Objekte arbeitet.
Die zur Kommunikation eingesetzten Java Objekte haben alle den gleichen Aufbau. So besitzt jedes Plugin Object genau ein Feld für die Erzeugung des Brute Force-Kommandos und genau ein Feld für die Erzeugung des Wortlisten-Kommandos. Dieser Umstand kann nicht verändert werden ohne den Quellcode anzupassen.
Sollte ein Protokoll eingeführt werden, ist es möglich eine variable Anzahl an generischen Kommandos zu übermitteln. Um dieses Ziel zu erreichen, müsste der Inhalt einer Konfigurationsdatei versendet werden, welche alle gewünschten generischen Kommandos enthält. Diese generischen Kommandos können den gleichen Aufbau folgen, der in dieser Arbeit beschrieben ist. Je nach Auswahl des Plugins kann die entsprechende Liste mit ausführbaren Kommandos angezeigt werden. Und auf Basis des ausgewählten Kommandos können dann die benötigten Parameter beschrieben werden. Der Benutzer würde die Parameter angeben und den Dienst anfordern. Durch diese Erweiterung ist es möglich, dass jede Person mit Zugriff auf die Konfigurationsdatei seine eigenen Kommandos beliebiger Komplexität erstellen und benutzen kann.
Ebenso ist es denkbar die Anwendung als Web-Applikation anzubieten, so dass Benutzer über ein Web-Interface Dateien zur Entschlüsselung bereitstellen. Mit ausreichend verfügbarer Rechenleistung kann eine Entschlüsselung zentralisiert stattfinden. Benutzer würden ihre Ergebnisse schneller erhalten und die frei gewordenen Ressourcen können für andere Ziele genutzt werden. Um die Interessen der Benutzer zu wahren, muss dabei sicher gestellt sein, dass der Übertragungskanal sicher ist.

Abschließend sei anzumerken, dass die Marktrelevanz dieser oder ähnlicher Anwendungen nur steigen kann. Welche Probleme in diesem Zusammenhang auftauchen und wie sie bewältigt werden können, wurde in dieser Arbeit erläutert.

7 Abbildungs- und Tabellenverzeichnis

Abbildungsverzeichnis:

8 Literaturverzeichnis

[ACRY 1996] Bruce Schneier
Applied Cryptography, Second Edition: Protocols, Algorithms and Source-Code in C
Published by John Wiley & Sons Inc. in 1996

[ALV 2014] Alvin Alexander
Java Exec - execute a system command pipeline in Java
Code for System Command Executor and Threaded Stream Handler
http://alvinalexander.com/java/java-exec-system-command-pipeline-pipe

[BRUT 2014] Brutus Online Cracker Website
http://www.securiteam.com/tools/2QUQ2PPRPG.html
Usage of Online Cracking Tools

[CAS 2012] Nunzio Cassavia, Elio Masciari
"Efficient MD5 Hash Reversing using DEA Framework for sharing Computational Resources"
Proceedings of the 16th International Database Engineering & Applications Symposium

[CRYE 2010] Niels Ferguson, Bruce Schneier, TadaYoshi Kohno
Cryptography Engineering: Design Principles and Practical Applications
Published by Wiley Publishing Inc. in 2010

[DUE 2013] Markus Dürmuth
"Useful Password Hashing: How to waste Computing Cycles with Style"
NSPW' 13, September 9-12, 2013, Banff, AB, Canada

[HDOC 2014] Hashcat Wiki
https://hashcat.net/wiki/
Official Hashcat Knowledge Base

[ISTR 2013] Symantec Corporation
Internet Security Threat Report 2013
Volume 18
http://www.symantec.com/security_response/publications/threatreport.jsp

[JAVA 2014] Java SE7 Documentation
http://docs.oracle.com/javase/7/docs/api/
Java Platform, Standard Edition 7 API Specification

[JDOC 2014] Openwall Project
http://www.openwall.com/john/doc/
Official John the Ripper Documentation

[KAHE 2014] Password File Manager Module
https://www.xing.com/profile/Kai_Henken
Advisor for Password File Management

[POU 2012] Poul-Henning Kamp
“LinkedIn Password Leak: Salt their Hide”
http://queue.acm.org/detail.cfm?id=2254400
Queue – Performance, Volume 10 Issue 6, June 2012

[RABB 2014] Rabbit Message Queue
http://www.rabbitmq.com/documentation.html
Official Rabbit Message Queue Documentation

[ROW 2011] Dale C. Rowe, Barry M. Lunt, Joseph J. Ekstrom
“The Role of Cyber-Security in Information Technology Education”
Proceedings of the 2011 conference on Information technology education

[SMEE 2014] Smeege Sec Hashtag: Password Hash Identification
http://www.smeegesec.com/2013/11/hashtag-password-hash-identification.html
Smeege Sec Security Research & Development

[UMR 2014] Dustin Kirkland
Ubuntu Manpage Repository
http://manpages.ubuntu.com
Official Ubuntu Documentation

9 Anhang

Im Anhang dieser Arbeit befindet sich eine DVD.

Inhalt der DVD:

- Bachelorarbeit im PDF-Format
- Quellcode des Prototypen
- Abbildungen im PNG-, JPG- und EMF-Format
- Netbeans IDE
- RabbitMQ Bibliotheken
- Fachliches Datenmodell der Anwendung
- Passwortdateien zu Testzwecken
- Bugliste zum Quellcode

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____