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 Datum

Erhan Yilmaz

Thema der Bachelorarbeit

...

Stichworte

...

Kurzzusammenfassung

...

Erhan Yilmaz

Title of the paper

...

Keywords

...

Abstract

...

Inhaltsverzeichnis

1 Einleitung

…

1.1 Motivation

Beinahe täglich erreichen uns Nachrichten von Unternehmen, deren Datenbanken kompromittiert wurden. Die Angreifer haben es in solchen Fällen meistens auf die Benutzernamen und die Passwörter der Kunden abgesehen, denn diese sind Geld wert. [ISTR 2013]

Entwendete Kundendaten oder Passwörter führen bei Firmen zu erheblichen Schäden.
Solche Vorfälle verursachen, zusätzlich zum Image-Schaden, immense Kosten, da beispielsweise Kunden entschädigt werden müssen.
Falls auch Passwörter entwendet wurden, kann es zu Identitäts-Diebstählen kommen. Diese sind besonders fatal, da die Angreifer, durch die Entwendung der Passwörter, Zugriff auf Geld oder Konten der Kunden bekommen können.
So konnte im Jahr 2011 eine Hacker-Gruppe erfolgreich in die Datenbanken des Playstation-Networks eindringen und Accounts im siebenstelligen Bereich stehlen.
Schätzungen zufolge, hat dieser Angriff Sony bis zu 3.2 Milliarden US-Dollar gekostet. [ROW 2011]

Passwörter und andere sensible Datensätze zu schützen, hat für Unternehmen eine hohe Priorität, aus diesem Grund sollten Passwörter immer gehasht abgespeichert werden.
Kryptographische Hash-Funktionen werden benutzt, um Passwörter unlesbar zu machen.
Diese Hash-Funktionen sind nicht umkehrbar. Will ein Angreifer also den Klartext zu einem verschlüsselten Passwort lesen, muss er eine Zeichenfolge finden, welche einen identischen Hash-Code erzeugt.
Die Zeit in der ein beliebiger Datensatz (z.B. Passwort) lesbar gemacht werden kann, sollte im besten Fall nur durch Brute-Force-Verfahren möglich sein.

Durch den Handel mit Datensätzen werden hohe Kapitalerträge erzielt. Im Jahr 2012 erzielten E-Mail Zugangsdaten einen Wert von bis zu 20 US-Dollar pro Account. [ISTR 2013]
Aus der Sicht eines Angreifers ist es demnach wichtig einen Datensatz lesbar zu machen, da er diesen teurer verkaufen kann. Um den Klartext zu einem gehashten Passwort zu finden, werden Passwort-Cracking-Tools eingesetzt.
Da Kriminelle nicht rechtmäßig in den Besitz der Daten kommen, müssen Sie zum Verkauf der Daten auf Schwarzmärkte (z.B. TOR-Netzwerk) ausweichen.

Es gibt verschiedene Verfahren mit denen man versuchen kann, den Klartext zu einem Hash-Wert zu finden.
Vor dem Hintergrund der gegebenen Daten, und dem zur Verfügung stehenden Wissen über diese Daten, sind jedoch nicht alle Verfahren anwendbar.
So hebelt beispielsweise das „Salting“ die Möglichkeit aus, Rainbow-Tables zu nutzen:
Bei Rainbow-Tables handelt es sich um Datenstrukturen. Diese bestehen aus Ketten von Hash-Werten und den zugehörigen Passwörtern. Ziel ist es, durch die Nutzung der vorberechneten Tabellen, den Klartext zu einem Hash-Wert schneller finden zu können.
Durch das Anhängen einer zufälligen Zeichenfolge an das Passwort wird verhindert, dass identische Passwörter auf denselben Hash-Wert abbilden. Somit kann nicht von einem gegebenen Hash-Wert auf die Ursprungs-Zeichenfolge geschlossen werden. Anders ausgedrückt: die mittlere Zeit, welche benötigt wird um ein Passwort lesbar zu machen, wird erhöht. Die Anzahl der benötigten Versuche steigt dabei um den Faktor der Anzahl aller, vom Salt, annehmbaren Werte. [POU 2012]

Derzeit existieren diverse Passwort-Cracking-Tools am Markt. Sie unterscheiden sich unter anderem in Expertise, Funktionsumfang und Geschwindigkeit.
Aufgrund dieser Unterschiede ist die Nutzung eines einzelnen Password-Cracking-Tools nicht immer ideal. Es ist möglich, dass ein Teilproblem schneller oder eindeutiger durch ein anderes Produkt gelöst werden kann.
Die Lösung für dieses Problem besteht darin, die besten und aktuellsten Services unterschiedlicher Anbieter miteinander zu kombinieren, um so ein genaueres Ergebnis ableiten zu können.

Im Rahmen dieser Arbeit soll ein Tool, bzw. eine Simulationsumgebung entstehen, welches die folgenden funktionalen Eigenschaften erfüllt:

1. Nutzung einer simulierten oder realen Menge von Passwörtern als Wissensbasis
2. Aufbau von Szenarien zum Testen der Wissensbasis (i.e. cracken der Passwörter), mithilfe von festgelegten Verfahren und Ansätzen
3. Aufzeichnung und Aufbereitung der während des Tests gesammelten Daten

Das Tool wird als Framework benutzt. Dabei werden gewünschte externe Services als Plugins eingebunden. Das Framework bietet somit eine einheitliche Sicht auf eingebundene Services.

Ein Service bezeichnet eine bereits vorhandene externe Software (z.B. John the Ripper).
So können nach der Einbindung eines Services die angebotenen Angriffsmodi und Cracking-Verfahren benutzt werden.
Sollte sich also ein neues Verfahren am Markt etablieren, kann es mit einfachen Mitteln in die Service-Landschaft der bereits eingebundenen Plug-Ins integriert und genutzt werden.
Ziel ist ein Produkt, welches mit möglichst geringem Aufwand aktuell und dementsprechend performant gehalten werden kann. Deshalb ist es wichtig den Aufwand, für das Einbinden eines Services, möglichst gering zu halten. Dazu wird ein generisches Format angeboten, welches vom Tool interpretiert werden kann.

Darüber besitzt die Anwendung Wissen über die Domäne, so dass bestimmt Operationen angeboten werden können, ohne Wissen durch den Benutzer voraussetzen zu müssen.

1.2 Zielsetzung

Ziel dieser Arbeit ist eine Anwendung. Diese Anwendung dient dazu, eine Testumgebung aufzusetzen. Idealerweise ist es möglich alle Parameter dieser Umgebung selbst zu bestimmen.

Ausgehend von einem Datenbestand aus Klartext-Passwörtern, sollte man entscheiden können welche Hashing-Algorithmen zur Verschlüsselung genutzt werden sollen, ob salting angewendet werden soll und welche Möglichkeiten der Angreifer zum cracken der Wissensbasis besitzt (e.g. Rainbow-Tables, Hardware-Unterstützung, Wortlisten etc.).
Abschließend ist es möglich Cracking-Verfahren einzusetzen, um die Wissensbasis auf Sicherheit zu prüfen.
Dadurch können Erkenntnisse über die Schnelligkeit und Zuverlässigkeit unterschiedlicher Cracking-Verfahren gewonnen werden.
Weil die Umgebung beliebig angepasst werden kann, ist es möglich eine Art Katalog zu erstellen.
Dieser Katalog kann Kunden vorgelegt werden und besteht aus bereits vorgefertigten Strategien zur Bildung von Passwörtern, Sicherheits-Richtlinien zum Management von Passwörtern und den dafür notwendigen Kosten und Aufwendungen.
Der Katalog kann dazu genutzt werden, um Kunden Schwächen und Stärken in ihren Datenbanken aufzuzeigen. Zusätzlich kann demonstriert werden, welche Art von Maßnahme welchen Schutz liefert.
Dabei ist es auch denkbar, dass Unternehmen aufgrund Ihrer eigenen Forschung mit Hilfe des Tools Kataloge anlegen.
Langfristig betrachtet soll das Tool, durch die Aktualität der eingebundenen Services, seine Daseinsberechtigung sichern. Aufgrund wissenschaftlicher Entwicklung kann es passieren, dass bestimmte Verfahren nicht mehr den aktuellen Anforderungen entsprechen. In diesem Fall kann das Tool helfen, da ein, den Anforderungen entsprechendes, Produkt einfach eingebunden werden kann.

Es ist anzumerken, dass der Funktionsumfang des Tools direkt vom Funktionsumfang der eingebundenen Plug-Ins abhängt. Mit Ausnahme der Built-In Funktionalitäten, wie das Hashen von Passwörtern und der Erkennung angewendeter Hashing-Alogrithmen.

1.3 Methodik

Diese Arbeit wird in zwei Teilen bearbeitet.

Der erste Teil vermittelt die Grundlagen von Verschlüsselungs- und Hashing-Verfahren und gibt einen Überblick über Passwort-Studien und den aktuellen Markt der Passwort-Cracking-Tools. Zusätzlich wird auf die unterschiedlichen Ansätze zum Cracken von Passwörtern eingegangen.

Bezüglich der drei genannten Verfahren (Verschlüsselung, Hashing und Cracking), werden die Funktionsweise und die Vor- und Nachteile erklärt. Zuletzt wird anhand eines Beispiels der Einsatz erläutert.

Die Passwort-Studie beschäftigt sich mit der Sicherheit von Passwörtern. Dabei werden unterschiedliche Studien zu Rate gezogen, welche sich mit der der Analyse weltweit genutzter Passwörter befassen. In diesem Kapitel wird beschrieben, was ein starkes von einem schwachen Passwort unterscheidet und wie eine möglichst hohe Passwort-Sicherheit erreicht werden kann.
Des Weiteren soll anhand von Statistiken die Häufigkeit von genutzten Passwörtern aufgezeigt werden.

Die Markt-Studie gibt einen Überblick über ausgewählte Passwort-Cracking-Tools. Dabei soll die Nutzung und der Funktionsumfang beschrieben und auf mögliche Defizite hingewiesen werden.

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.

Zusätzlich 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 2 externe Services anbietet.

In der Test-Phase werden die angebotenen Services, auf einem dedizierten Server, auf Geschwindigkeit und Korrektheit geprüft. Dabei werden verschiedene Test-Szenarien erstellt und ausgeführt.
Die Auswertung erfolgt unter Berücksichtigung der Größe und Komplexität der Eingabeparameter, der Geschwindigkeit und der Menge der gefundenen korrekten Lösungen.
Abschließend werden die gewonnenen Erkenntnisse zusammengefasst und ein Fazit gezogen, welches sich kritisch mit der weiteren Nutzung des Tools auseinandersetzt.

2 Anforderungsanaylse

Um die an das End-Produkt gestellten Erwartungen (Anforderungen?) zu erfüllen, muss eine Anforderungsanalyse durchgeführt werden.

Dazu werden die Anforderungen an das System beschrieben. Ebenfalls müssen die funktionalen und nicht funktionalen 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 Pflichtenheft betrachtet werden.

Im Folgenden wird zwischen zwei Entitäten unterschieden:

1. Benutzer: der Benutzer der Software bedient das Graphical User Interface. Dabei wird weder der Zugriff auf die Source-Dateien noch Wissen über die Funktionsweise des Programms vorausgesetzt.
2. Plugin-Architekt: der Plugin-Architekt muss Wissen über bestimmte Funktionsweisen des Systems besitzen. Dieses Wissen kann genutzt werden um externe Dienste in das System einzubinden. Im speziellen muss die Kommunikation mit dem Framework und mindestens die Delegation eines Arbeitsauftrages an einen Prozess verstanden werden.

Anforderungen an das System:
Mit dem System können gewünschte Dienste zum cracken von Passwörtern eingebunden und benutzt werden, zusätzlich werden Funktionalitäten zur Erzeugung von Passwortdateien, zur Erkennung benutzter Hashing-Algorithmen und zur Delegation von generisch erzeugten Dienst-Aufträgen angeboten.
Das System besteht aus zwei Dienst-Paketen, welche unabhängig voneinander genutzt werden können. Unterschieden wird dabei zwischen zwei Arten der Nutzung:

1. Der Fokus liegt auf der Benutzung eines bestimmten Dienstes
2. Der Fokus liegt auf der Bearbeitung einer bestimmten Passwortdatei

Ein Plugin-Architekt kann ein bestimmtes Password-Cracking-Programm zur Nutzung durch das System bereitstellen. Dazu müssen bestimmte Protokolle zur Registrierung beim System eingehalten werden. Nach der Registrierung ist es möglich Befehle an die konkrete Instanz des Password-Cracking-Programms abzusetzen, welche dann ausgeführt werden. Das Ergebnis wird dann an das System zurückgeführt. Hier können alle möglichen Befehle abgesetzt und verarbeitet werden, welche auch von der konkreten Programm-Instanz verstanden werden. Zuletzt muss es möglich sein Log-Dateien zu erstellen, so dass Ergebnisse nicht verloren gehen.

Mit dem Fokus auf der Passwortdatei können nur Funktionen benutzt werden, welche auch tatsächlich als Dienst eingebunden sind. Insofern ist diese Nutzung abhängig vom Funktionsumfang der eingebundenen Dienste und einer eingeführten Abstraktions-Ebene.
Diese Ebene muss durch den Anwendungskern des Frameworks vorgegeben werden und darf nicht durch einen Plugin-Architekten manipuliert werden. Dazu werden mögliche Operationen im Vorfeld evaluiert. Die Evaluation bestimmt welche Operationen in den Funktionsumfang des Anwendungskerns aufgenommen werden.
In Abhängigkeit von einer bestimmten Passwortdatei, welche der Benutzer angeben muss, können bestimmte Dienste genutzt werden. Es können die notwendigen Parameter zur Erfüllung der Aufgabe angegeben werden. Besonders wichtig ist, an welche konkrete Programm-Instanz die Ausführung der Operation delegiert werden soll.
Die Nutzung des Diensts durch die gewünschte Programm-Instanz kann nur erfolgen, wenn der Plugin-Architekt zuvor die Nutzungsbedingungen erfüllt hat.
Da der Plugin-Architekt keine direkte Manipulation am Anwendungskern vornehmen darf, muss er dem Anwendungskern eine Datei oder einen Dateipfad übermitteln. Diese Datei, welche in einem bestimmten Format vorliegen muss, enthält Ausdrücke ähnlich den „Prepared Statements“ und kann vom Anwendungskern verarbeitet werden. Die Ausdrücke müssen die vom Anwendungskern geforderten Operationen in generischer Form beschreiben. Die Verarbeitung kann erfolgen, da der Anwendungskern feststellen kann welche Parameter, beispielsweise die Passwort-Datei oder die Wortliste, an welcher Stelle des Ausdrucks eingesetzt werden müssen. Der durch dieses Vorgehen geformte Ausdruck wird dann als Kommando-Objekt abgesetzt und ausgeführt.

Die Implementierten Dienste (built-in-Operationen) beziehen sich auf das Hashen der in der Passwortdatei enthaltenen Passwörter, der Erkennung von benutzten Hashing-Algorithmen und der Benutzung einer sogenannten „Click and Fire“-Operation.

Bei der Nutzung einer „Click and Fire“-Operation können bestimmte Funktionalitäten genutzt werden, ohne Wissen über die Funktionsweise oder die Syntax des benutzten Dienstes zu kennen. Dabei wird, durch den Benutzer, eine gewünschte Funktion ausgewählt. Daraufhin werden die, zur Erfüllung des Dienstes, benötigten Parameter gesetzt und ein gewünschtes Plugin ausgewählt, welches den Dienst ausführen soll.
Nach dem Start der „Click and Fire“-Operation wird aus den angegebenen Parametern ein Kommando abgeleitet und an das entsprechende Plugin gesendet, welches den Dienst ausführt und das Ergebnis an den Benutzer zurück reicht.
Dieses Vorgehen hilft vor allem unerfahrenen und branchenfremden Personen, da nicht über das mögliche Vorwissen des Benutzers spekuliert wird. Stattdessen wird eine Abstraktionsebene eingeführt, welches die Delegation des angeforderten Dienstes übernimmt.
Somit kann das System in einem konfigurierten Zustand ausgeliefert und nach einer Einführung genutzt werden.
Ein Nachteil der Abstraktion besteht darin, dass Dienste nicht in Ihrem vollen Umfang genutzt werden können, da eine generische Darstellung über alle Dienste ausgearbeitet werden muss, welche möglicherweise nicht den gleichen Detailgrad eines selbst erzeugten Kommandos erreichen kann. Dieser Umstand wirkt sich negativ auf die Performanz und die Vollständigkeit aus.
Auch hier muss es möglich sein Log-Dateien zu erstellen, so dass Ergebnisse nicht verloren gehen.
Aus dieser Beschreibung ergeben sich folgende funktionale und nichtfunktionale Anforderungen.

2.1 Funktionale und nichtfunktionale Anforderungen

2.1.1 Funktionale Anforderungen

Anforderungen an die Anwendung:

A1.1 Der Benutzer gibt eine von ihm erstellte Passwortdatei an, welche zeilenweise im Format „Username:Password“ vorliegt. Diese Datei kann vom Framework umgewandelt werden. Dabei kann der gewünschte Hash-Algorithmus und/oder ein Salt-Wert angegeben werden. Die Ausgabe hat das Format „Username:$Hash$Salt$HashedPasswd:1000:1000:,,#,:/home/etherpad:/bin/bash“.

A1.2 Der Benutzer kann eine Passwortdatei mit dem Format

„Username:$Hash$Salt$ HashedPasswd:1000:1000:,,#,:/home/etherpad:/bin/bash“ angeben. Das Framework kann versuchen alle benutzten Hash-Algorithmen der Datei zu erkennen. Das Ergebnis wird dann in aufbereiteter Form angezeigt.

A1.3 Der Benutzer kann 2 Grundfunktionalitäten im Framework nutzen. Den Standard-Modus der gewünschten Programm-Instanz („God Mode“), sowie den Standard Wortlisten-Modus der gewünschten Programm-Instanz. Dazu muss er lediglich die notwendigen Parameter angegeben (Passwortdatei, Wortliste, Programm-Instanz). Das Ergebnis wird in einer Logdatei gespeichert. In Bezug auf die Funktionsweise der Programm-Instanz darf keinerlei Kenntnis, des Benutzers, vorausgesetzt werden.

A1.4 Die Nutzung der in A1.1 und A1.2 beschriebenen Funktionalität findet unabhängig von registrierten Plugins statt. Sie ist also auch ohne registrierte Plugins nutzbar („built-in“).

A1.5 Zur Nutzung der in Anforderung A1.3 genannten Funktionalität wird ein generisches Format angeboten, welches streng eingehalten werden muss. Die Übergabe des Dateipfades der Datei in entsprechendem Format muss bei Registrierung verschickt werden.

A1.6 Sollte der Benutzer falsche oder nicht verständliche Parameter übergeben, wird eine entsprechende Fehlermeldung erzeugt.

A1.7 Der Benutzer kann Befehle an zuvor registrierte Plugins absetzen, dabei dürfen alle Befehle abgesetzt werden. Sollte der Befehl nicht verstanden werden, wird eine entsprechende Fehlermeldung angezeigt. Das Ergebnis und die Log-Dateien müssen gespeichert werden.

A1.8 Der Benutzer hat in der Anforderung A1.4 beschriebenen Funktionalität die Möglichkeit folgende Parameter zu setzen: Arbeits-Verzeichnis, Speicherort der Log-Datei und dem Ergebnis.

A1.9 Der Benutzer hat für alle Operationen und den daraus resultierenden Prozessen die Möglichkeit einen „kill“-Befehl abzusetzen der alle laufenden Prozesse der konkreten Programm-Instanz beendet.

A1.10 Es ist möglich Password-Cracking-Programme und deren angebotene Dienste über eine generische Schnittstelle verfügbar zu machen.

A1.11 Es ist möglich Informationen zu laufenden Prozessen und registrierten Programmen zu erhalten. Diese werden in geeigneter Form dargestellt.

A1.12 Jeder laufende Prozess hat eine eindeutige Identifikation. Diese wird durch den Anwendungskern vergeben.

A1.13 Der Benutzer kann eine erneute Registrierung aller bereits registrierten Programme fordern. Dazu werden die Daten zur Verwaltung der bisher registrierten Programme gelöscht. Wissen über frühere Zustände wird nicht gespeichert.

A1.14 Zur Kommunikation zwischen dem Anwendungskern und den Plugins sollen nur Nachrichten verschickt werden. Die Nachrichten bestehen dabei aus serialisierten Objekten.

A1.15 Die zur Kommunikation genutzten Nachrichten müssen alle relevanten Informationen zur Ausführung eines Prozesses enthalten.

A1.16 Bevor ein Dienst genutzt werden kann, muss eine Registrierung des entsprechenden Programms erfolgen.

A1.17 Die Nachricht zur Registrierung eines Programms hat ein wohldefiniertes Format und kann vom Anwendungskern verarbeitet werden.

A1.18 Jedes Plugin führt eine Datenstruktur über alle laufenden Prozesse, welche bei Anfrage herausgegeben wird.

A1.19 Jedes Plugin hat Informationen über sich selbst, welche bei Anfrage herausgegeben werden.

A1.20 Jedes Plugin kann feststellen, welche Prozesse aktuell arbeiten.

A1.21 Jedes Plugin kann Status-Informationen zu laufenden Prozessen abfragen. Diese können regelmäßig an den Anwendungskern verschickt werden, wo sie dann angezeigt werden.

A1.22 Die Status-Informationen in Anforderung A1.20 haben kein bestimmtes Format und werden ohne Manipulation in der GUI angezeigt.

A1.23 Nachrichten an den Anwendungskern werden Asynchron empfangen.

A1.24 Die Anzeige und Verarbeitung von Daten in der GUI muss Threadsafe sein, es darf zu keinen Abstürzen oder Fehlern kommen.

A1.25 Die GUI kann jederzeit neu geladen werden, so dass neu registrierte Programm-Instanzen benutzt werden können. Dabei dürfen vorher bereits angemeldete Programme und deren laufende Prozesse nicht unterbrochen werden. Ebenfalls darf die Anzeige nicht gelöscht werden und muss fortlaufen können. Auch das Logging darf nicht betroffen sein.

A1.26 Der Benutzer muss eine Nachricht über Erfolg oder Misserfolg erhalten, wenn er einen Dienst anfordert. Die Nachricht muss vom Plugin generiert und verschickt werden. Dazu müssen eventuelle Fehler abgefangen und gekapselt werden.

A1.27 Der Prozess, welcher einen Dienst ausführt, darf nur solange existieren, bis die Ausführung des Diensts beendet ist.

A1.28 Sollten aus bestimmten Operationen Dateien entstehen, dürfen diese Dateien alte Dateien nicht überschreiben.

A1.29 Antworten verschiedener Plugins dürfen nur in den entsprechenden Fenstern der jeweiligen Programm-Instanz, innerhalb der GUI, angezeigt werden.

A1.30 Sollte die Verarbeitung einer Datei nicht erfolgen können, muss eine entsprechende Fehlermeldung angezeigt werden. Die Ausführung des Anwendungskerns und der GUI darf nicht betroffen werden.

A1.31 Die Art und Weise der Prozess-Verwaltung liegt im Ermessen des Plugin-Architekten, solange keine Anforderungen verletzt werden.

A1.32 Eine fehlerhafte oder nicht verständliche Eingabe darf nicht zum Absturz der GUI oder des Anwendungskerns des Frameworks führen.

A1.33 Beim Beenden der GUI werden alle Informationen über registrierte Programm-Instanzen gelöscht und an alle laufenden Prozesse ein „kill“-Befehl gesendet. Beim neu starten der GUI müssen sich alle Plugins neu anmelden.

A1.34 Bei Nutzung einer „Click and Fire“-Operation (A1.3) müssen die gecrackten Passwörter angezeigt werden. Dieser Umstand kompensiert das Fehlende Fachwissen des Benutzers, beim Absetzen einer Status-Anfrage.

A1.35 Der Anwendungskern besitzt eine Komponente zum Senden und Empfangen. Jedes Plugin ist mit diesen beiden Komponenten verbunden. Die Identifikation eines Kommunikationskanals findet über den Namen statt.

A1.36 Der Benutzer kann Batch-Processing nutzen. Dabei ist es möglich eine bestimmte Passwort-Datei anzugeben, welche sich im Format „Username:Passwort“ befindet. Der Batch-Process übernimmt die Funktionalitäten der Anforderungen A1.1 und A1.3.

2.1.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen an die Anwendung

A2.1 Die Kommunikation zwischen Anwendungskern und Plugins ist nicht verschlüsselt.

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

A2.3 Die Kommunikation zwischen Anwendungskern und Plugins 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. Dementsprechend muss das, zu diesem Zweck genutzte, generische Format gewählt werden.

2.1.3 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 Plugin implementiert sein, dies erlaubt ebenfalls die Nutzung der in A1.3 beschriebenen Funktionalität. Über weitere Funktionalität wird keine Aussage getroffen.

A3.4 Plugins werden nicht vom Autor dieser Arbeit implementiert. Ausnahme: A3.3.

2.2 Anwendungsfälle

Anwendungsfall AF1:	**Benutzer startet die Anwendung und möchte ein Plugin nutzen**
Akteur:	Angestellter

Auslöser:	Benutzer möchte eine Passwortdatei cracken
Vorbedingungen	Anwendung ist installiert. Mindestens ein Plugin hat eine Registrierungsanfrage versendet und ist bereit zur Nutzung
Nachbedingungen	Anwendung ist gestartet und das Plugin kann einen angeforderten Dienst ausführen

Erfolgsszenario:

1. Der Benutzer startet die Anwendung.
2. Das System empfängt alle Nachrichten, welche an den Kommunikationskanal versendet wurden.
3. Eine Registrierungs-Anforderung wird empfangen und entsprechend weiter geleitet.
4. Das System erstellt ein generisches GUI-Element, welches mit den Daten aus der Registrierungs-Nachricht erstellt wird.
5. Das System hat Informationen über den Nachrichten-Kanal und das eingebundene Plugin entsprechend gespeichert. Der Name des 1:1 Kommunikationskanals entspricht dem Namen des Plugins.
6. Der Benutzer klickt den Menü-Punkt „Refresh GUI Plugins“.
7. Das System lädt alle neu erstellten GUI-Elemente in die GUI.
8. Der Benutzer klickt auf den Reiter mit dem Namen des gewünschten Plugins.
9. Der Benutzer muss festlegen an welchem Ort die Ergebnisse gespeichert werden sollen, das Arbeitsverzeichnis festlegen in dem die relevanten Daten liegen und das Zeit-Intervall für Status-Anfragen an laufende Prozesse angeben.
10. Der Benutzer kann nun einen Befehl absetzen und erhält das Ergebnis. Einmal als Anzeige in der GUI und als Datei.

Fehlerfälle:

a. Die Registrierungs-Anforderung kann nicht gelesen werden oder ist nicht verschickt worden.
b. Der Benutzer klickt nicht den Menü-Punkt „Refresh GUI Plugins“.
c. Der Benutzer gibt nicht alle notwendigen Parameter an.
d. Der Benutzer setzt einen falschen oder nicht verständlichen Befehl ab.
e. Der Benutzer besitzt nicht die Rechte eine Datei zu erstellen.+
f. Das Plugin ist nicht mehr vorhanden oder funktionsfähig.

Anforderungen: []

Anwendungsfall AF2:	**Benutzer erstellt eine Passwortdatei**
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 „Create File“.
2. Der Benutzer gibt eine Datei an, welche ein geeignetes Format hat. In dieser Datei befinden sich die *„Username:Password“*-Einträge.
3. Der Benutzer gibt den Hash-Algorithmus und den Salt an. Zusätzlich bestimmt er einen Speicherort für die Ergebnis-Datei.
4. Der Benutzer klickt auf den Menü-Button „Generate File“.
5. Das System erstellt eine Passwortdatei im geeigneten Format, am gewünschten Speicherort.

Fehlerfälle:

1. Die vom Benutzer gestellte Passwortdatei hat ein falsches oder nicht verständliches Format.
2. Der Benutzer gibt keinen gewünschten Hash-Algorithmus und/oder einen Speicherort an.
3. Der Benutzer besitzt nicht die Rechte eine Datei zu erstellen.

Anforderungen:[]

Anwendungsfall AF3:	**Benutzer möchte Informationen über eine bestimme Passwortdatei**
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 „Get File information“.
2. Der Benutzer gibt eine Datei im Unix verständlichen Format an.
3. Der Benutzer klickt auf den Menü-Button „start Session“.
4. Das System zeigt nach Ende des Prozesses Informationen zu genutzten Hashing-Algorithmen und der Menge der entsprechend gehashten Passwörter an. Es ist möglich, dass keine relevanten Informationen extrahiert werden konnten.

Fehlerfälle:

a. Die gestellte Datei hat ein falsches oder nicht verständliches Format oder enthält keine gehashten Passwörter.

Anforderungen:[]

Anwendungsfall AF4:	**Benutzer nutzt die mitgelieferten Anwendungs-Funktionalitäten**
Akteur:	Angestellter
Auslöser:	Der Benutzer hat keine Fachkenntnis, möchte aber die Passwörter einer Datei cracken
Vorbedingungen:	Die Passwortdatei hat ein Unix verständliches Format
Nachbedingungen:	Der Benutzer erhält die gleichen Daten wie bei der Nutzung des Plugins

Erfolgsszenario:

1. Der Benutzer klickt auf den Reiter „Click and Fire“.
2. Der Benutzer wählt einen von zwei möglichen Modi aus.
   2.1. „God Mode“:
   Der Benutzer gibt eine Passwortdatei an, sowie welches Plugin den Dienst ausführen soll. Der Benutzer klickt auf den Menü-Button „Fire“. Hier wird der Standard-Modus des gewünschten Plugins ausgeführt.
   2.2. „Wordlist-Mode“:
   Der Benutzer gibt eine Passwortdatei und eine Wortliste an. Zusätzlich gibt er an, welches Plugin den Dienst ausführen soll.
   Der Benutzer klickt den Menü-Button „Fire“. Hier wird der Standard Wortlisten-Modus des gewünschten Plugins ausgeführt.
3. Das System zeigt die Ergebnisse in der GUI an und stellt sie als Datei zur Verfügung. Die Ergebnisse der „Click and Fire“-Operation stimmen mit den Ergebnissen einer direkten Nutzung des entsprechenden Plugins 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.
b. Das Plugin ist mehr vorhanden oder kann den Dienst nicht ausführen.
c. Der Benutzer hat nicht das Recht Dateien zu erstellen.

Anforderungen:[]

Anwendungsfall AF5:	**Laufenden Prozess (eines angeforderten Diensts) beenden**
Akteur:	Angestellter
Auslöser:	Der Benutzer möchte einen angeforderten Dienst vorzeitig beenden
Vorbedingungen:	Es Existiert ein aktiver Prozess eines Diensts (keine notwendige Bedingung). Der Benutzer befindet sich im entsprechenden Reiter des Plugins
Nachbedingungen:	Für das entsprechende Plugin gibt es keine aktiven Prozesse mehr

Erfolgsszenario:

1. Der Benutzer klickt auf den Menü-Punkt „Kill all Processes“.
2. Das System versendet eine Nachricht zum Beenden aller aktiven Prozesse an das entsprechende Plugin. Alle Prozesse werden kontrolliert heruntergefahren.
3. Der Benutzer erhält eine Nachricht über den Erfolg der Operation.

Fehlerfälle:

a. Die Prozesse konnten nicht beendet werden.
b. Das Plugin ist nicht mehr vorhanden oder kann nicht Antworten.

Anforderungen:[]

Anwendungsfall AF5:	**Informationen über laufende Prozesse eines bestimmten Plugins abfragen**
Akteur:	Angestellter
Vorbedingungen:	Der Benutzer befindet sich im entsprechenden Reiter des Plugins. Es sind aktive Prozesse vorhanden (keine notwendige Bedingung)
Nachbedingungen:	Der Benutzer hat Wissen über Anzahl bisherige Laufzeit der laufenden Prozesse (keine?)

Erfolgsszenario:

1. Der Benutzer klickt auf den Menü-Punkt „Show running Processes“.
2. Das System sendet eine Anfrage über alle aktiven Prozesse an das entsprechende Plugin.
3. Das Plugin antwortet mit einer entsprechenden Nachricht, welche alle aktiven Prozesse identifiziert und ihre bisherige Laufzeit angibt.
4. Das System stellt die Informationen in geeigneter Form dar, in einem speziell dafür vorgesehenen Dialog-Fenster.

Fehlerfälle:

a. Die Nachricht kann nicht versendet werden.
b. Der Kommunikationskanal ist nicht mehr aktiv.
c. Das Plugin ist nicht mehr vorhanden oder kann nicht antworten.
d. Die Antwort vom Plugin ist nicht in einem lesbaren Format oder falsch serialisiert.

Anforderungen:[]

Anwendungsfall AF6:	**Überprüfen welche Plugins antworten**
Akteur:	Angestellter
Vorbedingungen:	Es ist mindestens ein Plugin registriert und mindestens einmal in die GUI geladen worden
Nachbedingungen:	-

Erfolgsszenario:

1. Der Benutzer klickt auf den Menü-Punkt „Test active Plugins“.
2. Es öffnet sich ein Dialog-Fenster, in welchem die bisher registrierten Plugins aufgelistet sind.
3. Das System sendet automatisch eine Heartbeat-Anfrage an jedes Plugin.
4. Das System wartet maximal 10 sekunden auf eine Antwort.
5. Das System signalisiert für jedes Plugin, ob eine Nachricht erhalten wurde.

Fehlerfälle:

a. Es ist noch kein Plugin registriert.
b. Das System kann keine Nachrichten versenden.
c. Der Kommunikationskanal ist nicht mehr vorhanden.
d. Das Plugin kann keine Nachrichten senden oder empfangen.

Anforderungen:[]

Anwendungsfall AF7:	**Batch-Processing**
Akteur:	Angestellter
Vorbedingungen:	Der Benutzer hat eine Passwortdatei in einem geeigneten Format
Nachbedingungen:	Der Batch-Prozess wurde ausgeführt. Die Ergebnisse des Batch-Prozesses stimmen überein mit der Nutzung der einzelnen Dienste in der gleichen Reihenfolge

Erfolgsszenario:

1. Der Benutzer klickt auf den Menü-Punkt „Batch Processing“.
2. Der Benutzer gibt eine Passwortdatei in einem lesbaren Format an.
3. Der Benutzer gibt an welcher Hashing-Algorithmus genutzt werden soll, optional den gewünschten Salt-Wert, sowie an welches konkrete Plugin der Dienst delegiert werden soll. Bei der Auswahl des Plugins kann der Benutzer auf die Funktionalitäten der Anforderung A1.3 zugreifen.
4. Der Benutzer klickt auf den Menü-Punkt „Start Batch Process“.
5. Das System führt nacheinander alle erforderlichen Anweisungen aus, um zum Ergebnis zu gelangen.
6. Das System zeigt Informationen über den Stand des Batch-Prozesses an und gibt eine Rückmeldung über Erfolg oder Misserfolg.
7. Nach Beendigung des Batch-Prozesses ist die Passwortdatei mit gehashten Passwörtern gespeichert. Die Ergebnisse des Batch-Prozesses (Status-Informationen und genutztes Plugin) ebenfalls.

Fehlerfälle:

a. Die angegebene Passwortdatei hat ein falsches oder nicht lesbares Format.
b. Der Benutzer gibt einen notwendigen Parameter nicht an.
c. Die Delegierung des Batch-Prozesses an ein Plugin findet nicht statt.
d. Das Plugin ist nicht mehr vorhanden oder kann keine Nachrichten senden oder empfangen.

Anforderungen: []
//nochmal lesen hier existiert eine Diskrepanz zwischen batch und click and fire

3 Entwurf

In diesem Kapitel wird die Umsetzung der Anforderungsanalyse beschrieben.
Zunächst werden die allgemeinen Entwurfselemente beschrieben, ohne konkret auf spezifische Teil-Komponenten einzugehen. Daraufhin werden die Teil-Komponenten des Entwurfs genauer betrachtet, um so ein tieferes Verständnis von der Funktionsweise der kompletten Anwendung zu erhalten.

Die Teil-Komponenten des Entwurfs beziehen sich auf den Anwendungskern, die Plugins, die Public-Klassen, sowie die Kommunikation zwischen Anwendungskern und Plugins.

3.1 Allgemeine Architektur

Die Haupt-Design-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 Kommunikations-Modul des Anwendungskerns, über eine Message-Queue versendet.
Das entsprechende Plugin verarbeitet den Befehl, um ihn dann innerhalb der Process Control Klasse auszuführen. Das Ergebnis wird ebenfalls in ein Kommando-Objekt gewickelt 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 Einbindung eines Plugins benutzt werden können. Zur Ausführung dieser Funktionalitäten existieren entsprechende Komponenten.

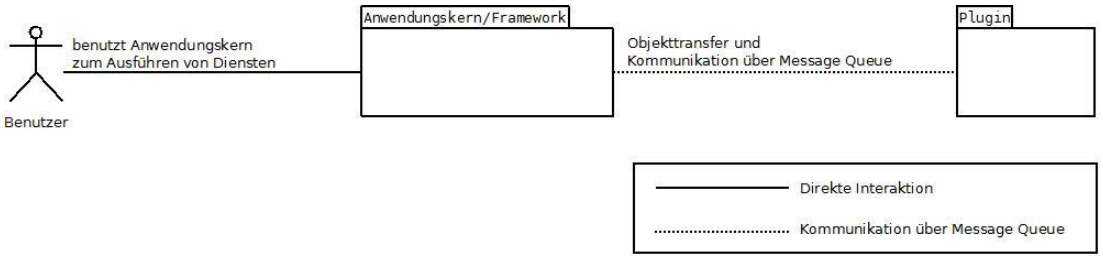


Abbildung 1: Kontext-Sicht der Anwendung

3.2 Entwurf Anwendungskern

Minimalistisch betrachtet ist der Anwendungskern eine Schaltzentrale mit erweitertem Fachwissen. Dies Bedeutet, dass die Hauptaufgabe in der Delegierung von Befehlen liegt. Zusätzlich dazu können Operationen zur Erstellung von Passwort-Dateien oder zur Weitergabe bestimmter Befehle genutzt werden. Ebenfalls können Informationen zu benutzten Hashing-Algorithmen einer Datei abgefragt werden.
Der Anwendungskern wird dabei von der GUI gesteuert und nimmt Befehle entgegen.

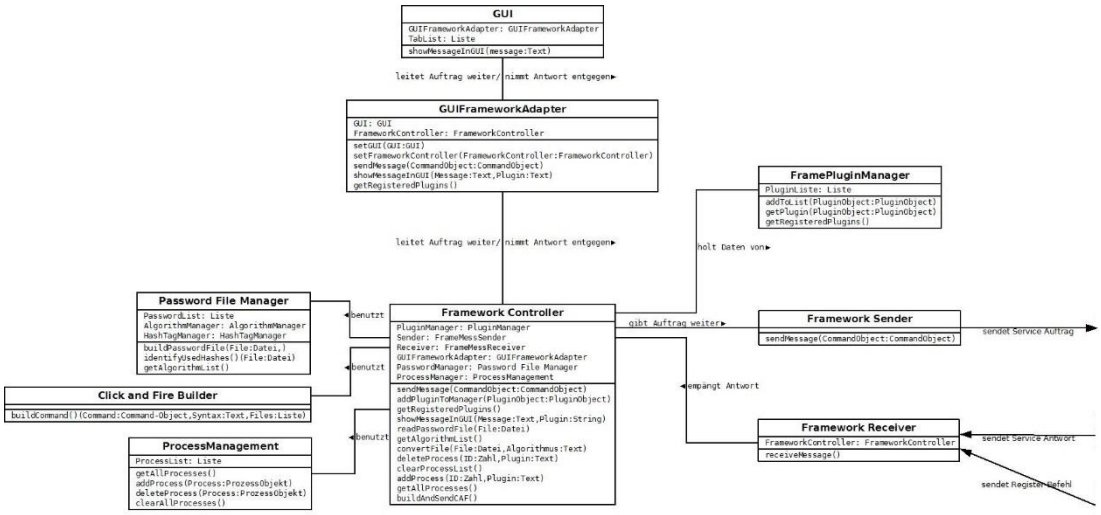


Abbildung 2: Fachliches Datenmodell des Anwendungskerns

GUIFrameworkAdapter:
Der GUIFrameworkadapter wird von der GUI initialisiert. Der Haupteinstiegspunkt der Anwendngs 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.

ProcessManagement:
Das ProcessManagement wird vom Framework Controller initialisiert. Das ProcessManagement kümmert sich um die Verwaltung aller aktiven Prozesse, wobei diese ebenfalls in der GUI angezeigt werden. Zur Verwaltung der aktiven Prozesse benutzt das ProcessManagement eine entsprechende Datenstruktur.
Zur eindeutigen Identifikation eines Prozesses wird eine Kombination aus ID und Namen des Plugins benutzt.
Die korrekte Verwaltung und Anzeige der Prozesse basiert dabei auf Informationen, welche von allen Plugins beigesteuert werden. Sobald ein Plugin mit der Ausführung eines Diensts beginnen konnte, wird ein Eintrag über diesen Prozess verfasst. Dieser Eintrag wird an die Datenstruktur des ProcessManagement 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. Der Controller übernimmt die Initialisierung der Kommunikations-Module und des Plugin-Managers. Zusätzlich kann er den Password File Manager nutzen, um Operationen auf Passwort-Dateien auszuführen.
Während der Laufzeit agiert der Framework-Controller als Schaltzentrale und koordiniert die angefragten Dienste des Benutzers. Befehle werden entgegen genommen, in ein Kommando-Objekt verpackt und an das Kommunikations-Modul übergeben.

Antworten, auf angefragte Dienste, werden ebenfalls als Kommando-Objekt, vom Kommunikations-Modul, entgegen genommen und verarbeitet. Die gelesene Antwort wird an den GUIFrameworkAdapter weiter gegeben.

Zur Verarbeitung einer Passwort-Datei hat der Framework Controller Zugriff auf den Password File Manager. Sollte ein entsprechender Befehl abgesetzt werden, nimmt der Controller den Dateipfad der Passwort-Datei, sowie die gewünschten Parameter zur Verarbeitung entgegen und übergibt die Parameter an den Password File Manager. Der Password File Manager lädt dann die Datei und führt die gewünschte Operation aus. Dies kann entweder die Formatierung eines Klartext-Passworts sein oder die Identifizierung benutzter Hashing-Algorithmen bei Passwort-Dateien mit bereits verschlüsselten Passwörten.
Bei der Formatierung einer Datei wird der Dateipfad der Ergebnis-Datei übergeben. Bei einer Identifikation werden die gesammelten Informationen zu der entsprechenden Datei 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, Passwort-Datei, Plugin und eventuelle weitere Parameter) werden vom Benutzer gestellt und vom GUIFramework-Adapter an den Framework Controller gegeben. Dort stellt der Framework Controller eine Anfrage an den Plugin-Manager, um die Syntax des Befehls im entsprechenden Modus korrekt zu bilden können. Dieser Befehl wird in ein Kommando-Objekt verpackt und an das Kommunikations-Modul übergeben.

Sollte ein Befehl zur Anzeige der bereits registrierten Plugins entgegen genommen werden, stellt der Framework Controller ebenfalls eine entsprechende Anfrage an seinen Plugin Manager. Der Plugin Manager führt eine Liste aller bereits registrierten, 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 Registrierungs-Nachrichten 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, einem bestimmten oder zu mehreren, Plugins zu erhalten. So ist es beispielsweise 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.
Die 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 Registrierungs-Nachricht 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 Kommunikations-Modul 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 Command-Object serialisiert und über eine Message-Queue an alle Plugins versendet. Das entsprechende Plugin deserialisert das Objekt und kümmert sich dann um die Ausführung des Diensts.
Die Ergebnisse der Ausführung werden dann, ebenfalls serialisiert, an den Framework Receiver gesendet. Der Framework Receiver deserialisiert das empfangene Objekt und leitet es, entsprechend seiner Instanz, an den Framework Controller weiter.
Da empfangene Kommando-Objekte lediglich Ergebnisse zu vorher angefragten Diensten enthalten können, kann die enthaltene Information direkt an die GUI weitergeleitet werden, wo sie in der entsprechenden Ansicht angezeigt werden.

Beim Empfang eines Registrierungs-Objekts extrahiert der Framework Sender das Plugin-Objekt, welches im Registrierungs-Objekt gekapselt ist und leitet dieses Objekt über den Framework Controller zum Framework Plugin Manager.

Password File Manager:
Der Password File Manager wird durch den Framework Controller initialisiert und hat zwei Aufgaben:

Die Formatierung einer, vom Benutzer bereit gestellten, Passwort-Datei in ein UNIX lesbares Format. Sowie die Darstellung von Informationen zu benutzten Hashing-Algorithmen in einer, vom Benutzer bereit gestellten, Passwort-Datei.

Zur Nutzung der Formatierungs-Funktionalität gibt der Benutzer eine Passwort-Datei im Format „Username:Password“ an, sowie den gewünschten Hashing-Algorithmus, mit dem die Passwörter gehasht werden sollen. Der Benutzer kann angeben, ob das Passwort zusätzlich zum hashen, mit einen Salt-Wert versehen werden soll. In diesem Fall wird der Salt-Wert 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 Ergebnis-Datei wird an den Framework Controller gegeben, welcher dann die Datei, zur Begutachtung, öffnet und eine Erfolgsmeldung in der GUI anstößt.

Sollte der Benutzer Informationen zu einer bestimmten Passwort-Datei wollen, erhält der Password File Manager einen entsprechenden Aufruf durch den Framework Controller.
Hier wird ebenfalls der Pfad einer Datei übergeben.
Der Inhalt der Datei wird dann Zeile für Zeile 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 mögliche Ergebnisse angezeigt.
Das Ergebnis enthält neben der Anzahl der gefundenen Hashing-Algorithmen auch die Anzahl der Einträge, welche 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, welcher den Inhalt dann an den GUIFrameworkAdapter reicht, damit es 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 dabei noch einen Schritt weiter und versteckt die Hintergrund-Prozesse, so dass ein Benutzer sich lediglich um die Angabe der korrekten Parameter kümmern muss.
Der Click and Fire Builder wird vom Framework Controller initialisiert, somit besitzt der Framework Controller eine Referenz auf den Click and Fire Builder.
Eine abgesetzte „Click and Fire“-Operation wird über die GUI und den GUIFrameworkAdapter bis zum Framework Controller gereicht.
Die vom Benutzer angegebenen Parameter (Modus, Passwort-Datei, 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 zunächst generische Schlüsselwörter, welche durch die konkreten Parameter ersetzt werden. Dieses Kommando kann nun in ein Command Object verpackt werden. Das Ergebnis ist dann ein komplettes Command Object, welches direkt an das richtige Plugin versendet werden kann. Der Dienst wird dann normal ausgeführt, das Ergebnis wird angezeigt.
Sollte die Click and Fire Operation fehlschlagen oder ein falscher Parameter übergeben worden sein, muss dies entsprechend kenntlich gemacht werden, idealerweise durch eine Fehlermeldung.
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, welche zur Ausführung einer „Click and Fire“ Operation benötigt werden, ist dem Kapitel „Click and Fire Syntax“ zu entnehmen.

3.3 Entwurf Plugin

Die Grundfunktion eines Plugins ist die Annahme und die Verarbeitung von Aufgaben oder Diensten. Diese Aufgaben werden vom Anwendungskern über die Message Queue versendet und von einem bestimmten 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 vom Plugin angeboten werden, um einen fehlerfreien Ablauf der Anwendung garantieren zu können. Die Aufgabenbereiche, welche zwingend abgedeckt werden müssen, 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, deswegen aber nicht weniger hilfreich für den Benutzer der Anwendung. So ist es beispielsweise dringend empfohlen, Status-Nachrichten 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 Architektur-Entscheidungen des Entwurfs mit Bedacht auf Performance, geringer Kopplung, Benutzer-Komfort und Modularisierung getroffen worden. Die Einhaltung und Nachahmung wird also nahe gelegt.

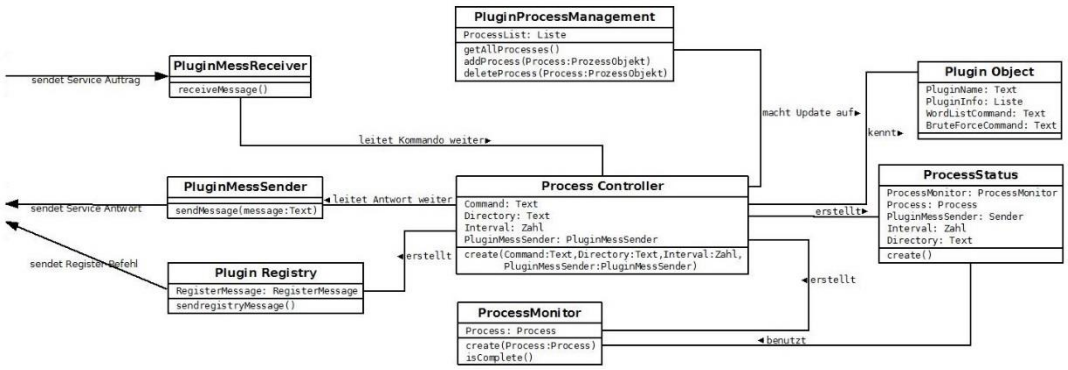


Abbildung 3: Fachliches Datenmodell eines möglichen Plugins

PluginProcessManagement:
Das PluginProcessManagement kümmert sich um die Verwaltung aller aktiven Prozesse seines Plugins und wird vom Plugin Message Receiver initialisiert.
Für jede empfangene Dienst-Aufforderung an das Plugin wird ein Verwaltungs-Objekt erstellt, welches eine Referenz auf den laufenden Prozess enthält. Dieses Objekt wird sowohl an den Anwendungskern, als auch an das PluginProcessManagement übergeben.
Das PluginProcessManagement benötigt das Objekt, falls der Anwendungskern eine Terminierung aller laufenden Prozesse veranlasst.
Der Anwendungskern benutzt das Objekt zur Anzeige aller laufenden Prozesse und fügt das Objekt in seine Verwaltungs-Struktur ein.
Wenn der laufende Prozess terminiert, wird eine Nachricht mit dem Referenz-Objekt des laufenden Prozesses, zusammen mit einer Aufforderung zum Löschen, an den Anwendungskern versendet. Der Anwendungskern löscht dann die Referenz auf den Prozess und korrigiert die GUI-Anzeige entsprechend.

Plugin Message Receiver:
Der Plugin Message Receiver dient dem Empfangen von Nachrichten, welche vom Anwendungskern verschickt werden. Darüber hinaus wird über den Plugin Message Receiver das Plugin initialisiert. Der Plugin Message Receiver initialisiert dabei den Plugin Message Sender und veranlasst das Versenden einer Registrierungs-Aufforderung an den Anwendungskern. Gekapselt in der Registrierungs-Aufforderung ist das Plugin-Objekt des Plugins, welches auch vom Plugin Message Receiver erstellt wird.
Nach Erstellung aller benötigten Instanzen und der Anmeldung meldet sich der Plugin Message Receiver bei der Message Queue des Anwendungskerns an und geht in den Empfangs-Zustand über.
Sollten nun Nachrichten empfangen werden, muss zunächst 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 Command-Object 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 man keine Aussage über die Terminierung des Process Controllers treffen 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 der Plugin Message Sender zuständig für das Versenden einer Registrierungs-Aufforderung an den Anwendungskern, dies geschieht mit Hilfe eines Registry-Objects, welches der Plugin Message Sender vom Plugin Message Receiver übergeben bekommt.
Zum Versenden von Nachrichten an den Anwendungskern, muss sich der Plugin Message Sender bei der Message Queue des Anwendungskerns anmelden.
Wenn der Process Controller einen Dienst ausführt, verschickt er neben der Ausgabe des Diensts auch Status-Nachrichten. Beide Nachrichten-Typen werden an den Plugin Message Sender gegeben, welcher die Nachrichten an den Anwendungskern versendet.

Process Controller:
Eine Instanz des Process Controllers dient genau der Ausführung eines erteilten Auftrags vom Anwendungskern. 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 (Arbeits-Verzeichnis, Status Intervall und Speicherort für Dateien) gesetzt. Daraufhin startet der Process Controller den Arbeits-Prozess, sowie einen Prozess der die Terminierung des Arbeits-Prozesses beobachtet, den Process-Monitor. Zusätzlich wird ein ProcessWriter gestartet, welcher in regelmäßigen Abständen Status-Nachrichten, zum laufenden Prozess, an den Anwendungskern versendet. Alle drei oben beschriebenen Kompnenten müssen asynchron ausgeführt werden, da ansonsten die Ausführung des Arbeits-Prozesses blockiert werden kann.
Jede vom Arbeits-Prozess generierte Ergebnis-Nachricht wird in ein Command-Object gekapselt und über den Plugin Message Sender 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. Die Ausführung des Befehls

ist nicht möglich, falls der Befehl falsche Parameter enthält oder Schreibfehler vorhanden sind.

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

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

3.4 Entwurf Public Klassen

Dieses Kapitel beschreibt Klassen, welche der gesamten Anwendung bekannt sind. Diese Klassen können als Helfer-Klassen betrachtet werden, welche bei der Kommunikation und der Verwaltung benötigt werden. Der Plugin-Architekt muss die entsprechenden Interfaces der Klassen implementieren, um sie benutzen zu können.

Abbildung 4: Fachliches Datenmodell des Public Klassen

Command Object:
Das Command-Object wird als Transportobjekt genutzt, im speziellen zur Kommunikation zwischen dem Anwendungskern und seinen Plugins.
Zur eindeutigen Identifikation des Empfängers und der korrekten Ausführung des Befehls besitzt das Command-Object mehrere Instanz-Variablen. 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 Zeit-Intervall für die Status-Nachricht. Zusätzlich existiert ein Kommentar-Feld, welches beliebige Strings enthalten kann. Im Fall der Prozessverwaltung wird das Kommentar-Feld 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 das Command-Object über den Framework Sender serialisiert und an alle registrierten Plugins versendet.
Alle Plugins, welche das Command-Object empfangen, deserialisieren das Objekt 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 das Objekt verwerfen.
Beim Empfang durch das Plugin wird das Command-Object deserialisiert und einer Instanz des Process Controllers übergeben. Hier werden alle benötigten Parameter zur Ausführung gesetzt und der der Arbeits-Prozess wird gestartet.
Sämtliche Ergebnisse, auch die Ergebnisse eventuell erzeugter Status-Nachrichten, werden wiederum in ein Command-Object verpackt und, serialisert, 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. Das Ergebnis wird im Command-Feld des Command-Objects mitgeführt. Der Speicherort, das Arbeitsverzeichnis, das Kommentar-Feld sowie das Zeit-Intervall sind bei einer Antwort irrelevant.
Das zu implementierende Interface dieser Klasse umfasst die Getter- und Setter-Methoden der Instanz-Variablen. Weiter Funktionalität ist nicht vorgesehen.

Registry Object:

Das Registry-Object wird als Transport-Objekt benutzt und dient der Registrierung eines Plugins beim Anwendungskern, so dass vom Plugin angebotene Dienste ausgeführt werden können.
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, welches vom Registry-Object gekapselt wird.
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 eingefügt, welche eine „Click and Fire“-Operation ausführen können. Diese Liste wird im entsprechenden Reiter der GUI angezeigt. Beide GUI-Elemente werden mit Hilfe der Daten aus dem Plugin-Object erstellt.
Der Registrierungs-Vorgang bedarf keiner Erfolgsmeldung vom Anwendungskern an das Plugin.

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 die „Click and FIre“-Operationen. Zusätzlich lässt sich anhand der Instanz-Variablen ablesen, welche „Click and Fire“-Operationen unterstützt werden.
Während der Initialisierung des Plugins werden alle benötigten Plugin-Instanzen gestartet, daraufhin wird ein Registry Object erstellt, welches ein Plugin Object kapselt. Dieses Objekt wird serialisert und an den Anwendungskern verschickt.
Beim Empfang durch den Anwendungskern, wird das Registry Object deserialisert. Daraufhin wird das Plugin Object entnommen und am dem 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, so dass das Plugin für „Click and FIre“-Operationen benutzt werden kann, sollte es die entsprechende Operation unterstützen.
Die Syntax der Strings, welche zur Ausführung einer „Click and Fire“ Operation benötigt werden, ist dem Kapitel „Click and Fire Syntax“ zu entnehmen.

3.5 Entwurf Message Queue

Die Message Queue ist der Kommunikations-Kanal 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. Sie kümmern sich um die Initialisierung der eigenen Message-Queues. Zur Laufzeit existieren also zwei Message-Queues, eine Queue

wird zum Senden, die andere zum Empfangen von Nachrichten genutzt. Nach Beenden der Initialisierungs-Phase können Nachrichten versendet und empfangen werden.
Zu verschickende Nachrichten werden von Transport-Objekten gekapselt und an den Framework Sender übergeben. Hier werden sie serialisiert und über die Message-Queue versendet.
Inspiriert vom Adress-Resolution-Protocol [QUELLE], werden Nachrichten an alle registrierten Plugins versendet. Zu diesem Zweck besitzt jedes Plugin ein Sende- und Empfangs-Modul, welche an die entsprechenden Message-Queues des Anwendungskerns gebunden werden.
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 wiederum in ein Objekt verpackt und an das Sende-Modul des Plugins gegeben. Hier werden die Objekte serialisiert und, mit Hilfe der Message-Queue, übertragen.
Das vom Anwendungskern empfangene Objekt wird dann deserialisiert und bis zur GUI hochgereicht, um dort angezeigt zu werden.
Die Übertragung von Registrierungs-Aufforderungen geschieht äquivalent, jedoch ohne eine Anzeige innerhalb der GUI. Stattdessen wird, mit Hilfe des in der Registrierungs-Aufforderung enthaltenen Plugin-Objects, ein Eintrag in der Verwaltungs-Komponente vorgenommen. Anhand der Daten in der Verwaltungs-Komponente können dann die erforderlichen GUI-Elemente erstellt werden.

3.6 Passwort-Datei Formate

Ein großer Fokus der Anwendung liegt auf Passwort-Dateien. Ein Benutzer kann seine eigenen Passwort-Dateien stellen oder eine Passwort-Datei durch die Anwendung erstellen lassen.
Wenn ein Benutzer eine eigene Passwort-Datei benutzen möchte, muss diese Datei im gleichen Format wie die Unix /etc/passwd Datei vorliegen.
Die Unix /etc/passwd Datei enthält Informationen zu Benutzern und, indirekt, Passwörtern, welche während eines Login-Vorgangs benötigt werden.
Beim Login gibt der Benutzer das Passwort zu einem bestimmten 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 dann, mit Hilfe der Daten aus der /etc/passwd Datei, gehasht und mit dem gespeicherten Hash verglichen. Sollten beide Zeichenfolgen übereinstimmen, wird der Benutzer eingeloggt.
Die Trennung von Account-Daten 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 deshalb zusammen geführt werden, um eine geeignete Passwort-Datei zu erhalten. [UMR 2014]
Um ein besseres Verständnis zu schaffen wird im Folgenden davon ausgegangen, dass diese Dateien bereits zusammen geführt worden sind.
Das Format, in dem die /etc/passwd Datei vorliegt, wird von allen Passwort-Cracking-Programmen verstanden, welche auch unter Ubuntu 12.04 LTS laufen. Somit ist es notwendig dieses Format streng einzuhalten.

Format der kombinierten /etc/passwd:
Die kombinierte /etc/passwd Datei ist ein Text-Dokument 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 Name, mit welchem 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. Man bekommt Informationen über den genutzten Hash-Algorithmus, den benutzten Salt-Wert und kann den Hash-Wert des Passworts ablesen. Sollte dieses Feld leer sein, bedeutet dies, dass kein Passwort zum Login benötigt wird. 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. In der Regel werden die Zahlen 1-1000 an die System-Benutzer vergeben.

GID:
Die Gruppen-ID des Benutzers.

Info:
Dieses Feld enthält Informationen zu dem Benutzer. Es ist unter anderem möglich den Namen oder die Adresse des Benutzers zu hinterlegen. Einzelne Informations-Segmente

werden mit einem Komma getrennt. Auch jeglicher anderer Text kein in dieses Feld eingetragen werden.

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

Shell:
Dieses Feld beschreibt den absoluten Pfad einen Programms, welches direkt nach dem Login gestartet wird. 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 Passwort-Datei kann also 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 Salt-Wert. Das Passwort lautet „5f4dcc3b5aa765d61d8327deb882cf99“. Der Benutzer hat die eindeutige Kennung 1001 und die Gruppen-Kennung 1000. Weitere Informationen beschreiben den vollen Namen, „Thomas Anderson“, und die Adresse, „Whitehaven Apts“, des Benutzers. Sein Heimat-Verzeichnis ist „/home/tAnderson“, nach dem Login-Vorgang wird eine Shell gestartet.

Wie bereits beschrieben, kann eine Passwort-Datei mit diesem Format von allen, unter Ubuntu ausführbaren, Passwort-Cracking-Programmen gelesen werden. Somit ist es möglich diese Passwort-Datei 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 Passwort-Datei erstellen zu lassen. Dazu muss der Benutzer lediglich eine Text-Datei erstellen, welche ebenfalls einen Eintrag pro Zeile enthält. Jeder Eintrag besteht dabei aus zwei Feldern, welche den Account-Namen und das Passwort im Klartext beschreibt. 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 Datei, welche dann im Format der /etc/passwd Datei, kombiniert mit der /etc/shadow Datei, vorliegt.
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 eine konkrete Plugin-Instanz.
Es ist zusätzlich möglich einen Salt-Wert zu benutzen. Dieser Salt-Wert darf aber nicht vom Benutzer vorgegeben werden, sondern muss durch das System generiert werden.
Die Ergebnis-Datei enthält dann den Salt-Wert als zusätzliches Feld.

3.7 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 konkrete Ausführung wird dann delegiert, die Ergebnisse werden daraufhin angezeigt.
Nach der Auswahl eines bestimmten Diensts, durch den Benutzer, wird eine Liste aller Plugins angezeigt, welche die 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, wodurch das gewünschte Kommando zusammen gebaut werden kann.
Die allgemeine Syntax dieses Strings besteht aus einer Zeile, welche wiederum die exakte Form des angefragten Diensts enthält. Stellen innerhalb des Strings, welche Parameter enthalten, werden durch Platzhalter-Symbole ersetzt.
Im Umfang dieser Arbeit werden zwei Click and Fire Operationen angeboten: Der Standard-Modus und der Wortlisten-Modus eines Plugins. Bei beiden Modi kann man exakt voraussagen, 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 Passwort-Datei.
Der String **„WORDL“** steht an der Stelle des Pfads für die Wortliste.
Der String **„HASHT“** steht an der Stelle des Hash-Typen, welcher den jeweiligen Hash-Typen mit einer entsprechenden Zahl belegt.
Die folgenden Strings sind das Ergebnis einer Anfrage an ein Plugin (John the Ripper und Hashcat) im Wortlisten-Modus, noch 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, welche durch den Benutzer bereits angegeben wurden.
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 korrekten Arbeitsverzeichnisses:

john --wordlist=myWordList myPassList

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

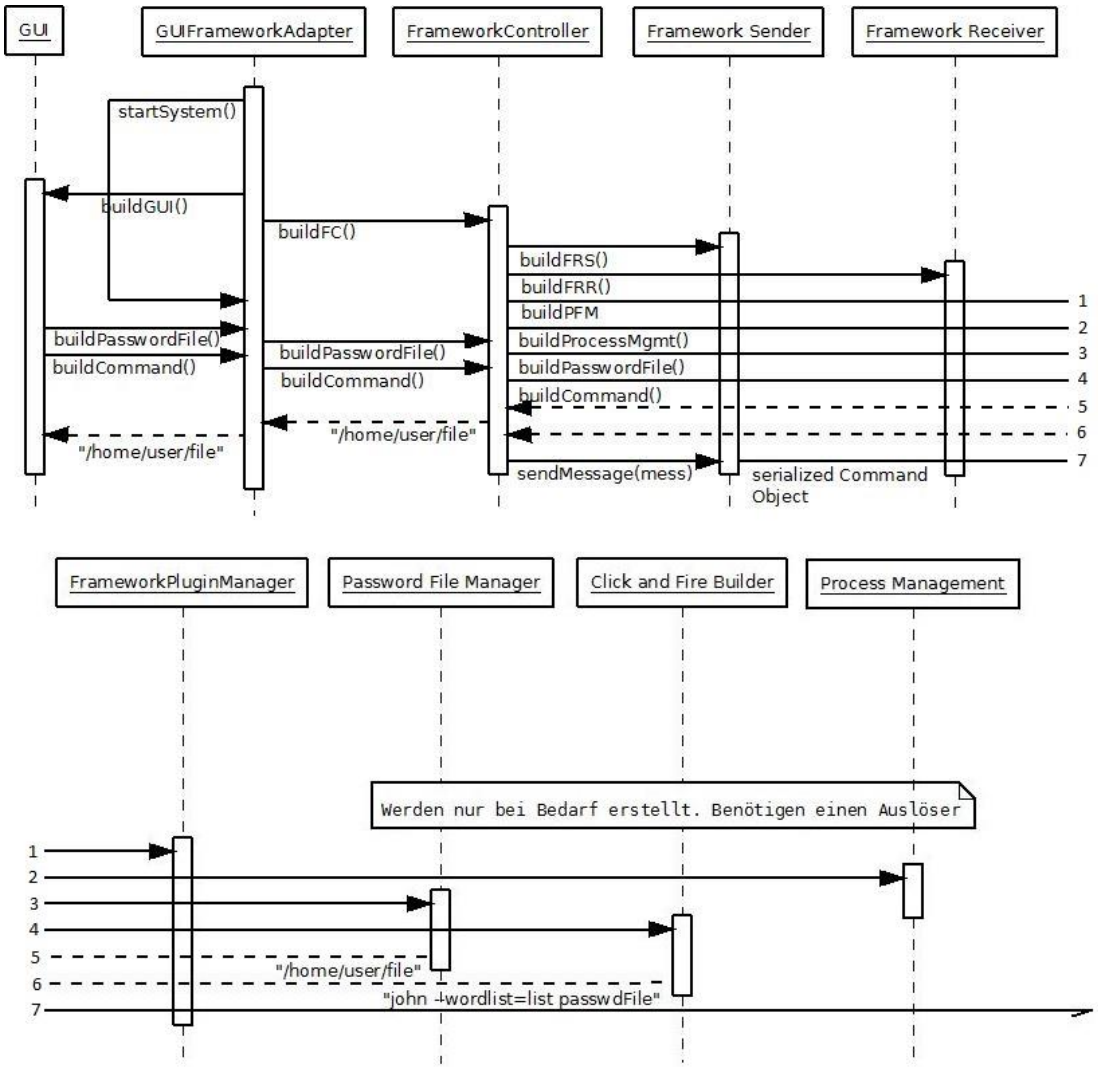
Nach Erstellung des Kommandos, wird, wie üblich, ein Command Object erstellt und verschickt. Das enthaltende Kommando wird dann vom Plugin ausgeführt.
Die Erweiterung dieser Funktionalität um weitere Schnittmengen-Operationen unterschiedlicher Passwort-Cracking Anbieter ist, mit vertretbarem Aufwand, 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 Schnittmengen-Operation erfasst werden können. Sollte also 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 Platzhalter-Symbole eingeführt werden. Zuletzt muss die Methode angepasst werden, welche den Zusammenbau des Kommandos auf Grundlage der Platzhalter-Symbole übernimmt.

3.8 Ablauf des Programms

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

Initialisierung des Anwendungskerns:

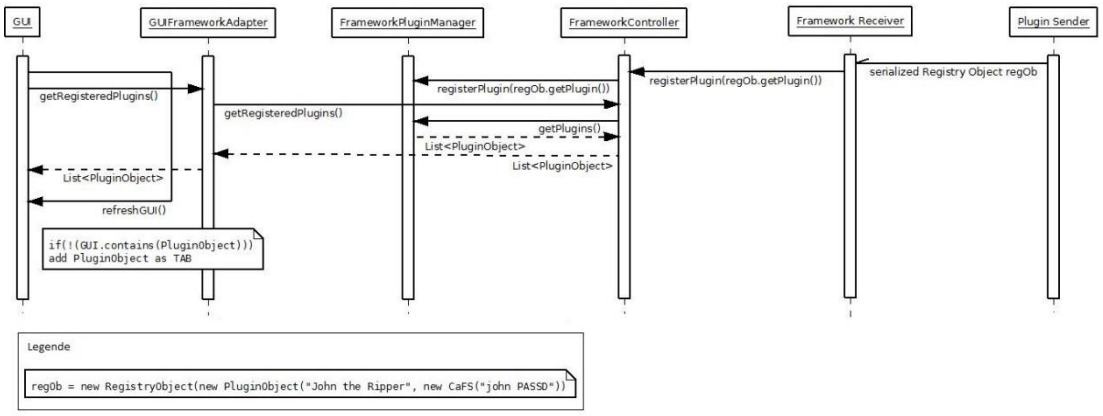
Dieses Diagramm beschreibt den Initialisierungs-Vorgang beim Starten des Anwendungskerns.



Der Anwendungskern wird durch den GUIFrameworkAdapter angestoßen. Der GUIFrameworkAdapter erstellt hierbei die GUI und den Framework Controller. Der Framework Controller spielt von diesem Zeitpunkt an die zentrale Rolle, indem er alle weiteren benötigten Instanzen zur Ausführung aller möglichen Operationen initialisiert. Konkret handelt es sich dabei 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. Sämtliche Operationen laufen somit über den Framework Controller.

Der Framework Receiver erstellt bei Initialisierung eine Message Queue und geht dann direkt in den Empfangs-Modus über. Hier wartet der Receiver auf potenzielle Nachrichten, welche von Plugins gesendet werden.
Der Framework Sender meldet sich bei der, nun vorhandenen, Message 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 gebraucht 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 des Password File Managers oder des Click and Fire Builders 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.

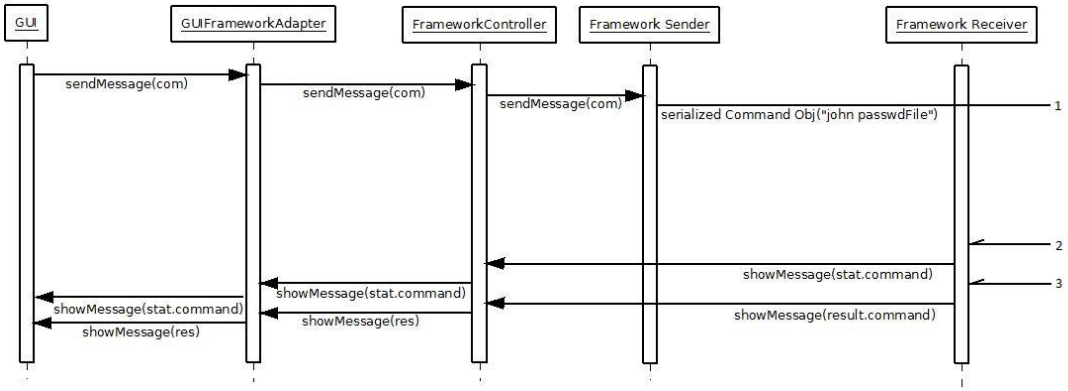
Registrierung eines Plugins:
Dieses Diagramm beschreibt den Ablauf der Registrierung eines Plugins beim Anwendungskern. Unten im Bild erkennt man die Legende zum Diagramm.

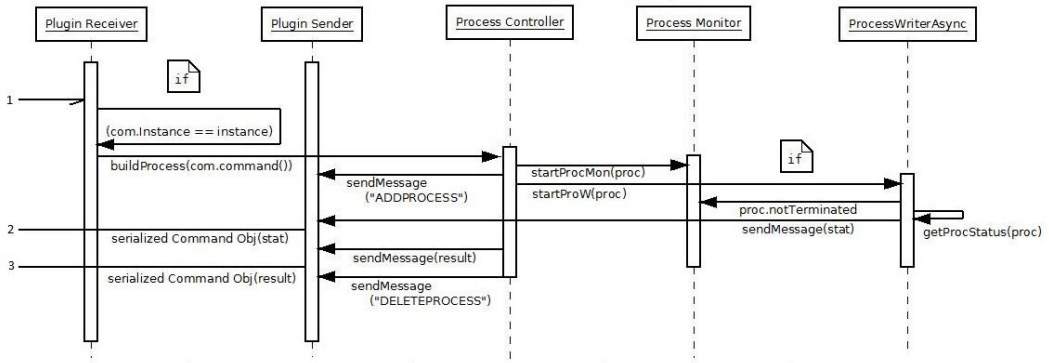


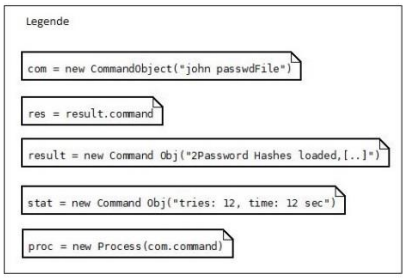
Die Registrierung eines Plugins beginnt mit dem Versenden eines Registry Objects. Dieses Registry Object wird vom Plugin versendet und vom Framework Receiver empfangen. Nach der Deserialisierung wird überprüft, ob es tatsächlich ein Registry Object ist. Im korrekten Fall enthält das Registry Object ein Plugin Object. Dieses Plugin Object wird an den Framework Controller übergeben. Der Framework Controller übergibt das Plugin Object an den Framework Plugin Manager, wo es verwaltet wird. Das Plugin ist nun angemeldet.

Zur Vervollständigung der kompletten Operation muss das Plugin nun 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 der 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, wo ein Abgleich mit den bereits erstellten GUI-Elementen stattfindet. Für alle neuen Plugins werden dann entsprechende GUI-Elemente erstellt. Nun kann der Benutzer ein angemeldetes Plugin in vollem Umfang nutzen.

Anforderung eines Diensts durch ein Plugin:
Aufgrund der Größe dieses Schritts wird das Laufzeit-Diagramm in zwei Teilen dargestellt. Das erste Diagramm bezieht sich auf die Seite des Anwendungskerns, das zweite Diagramm auf die Seite des Plugins. Die letzte Abbildung beschreibt die Legende.

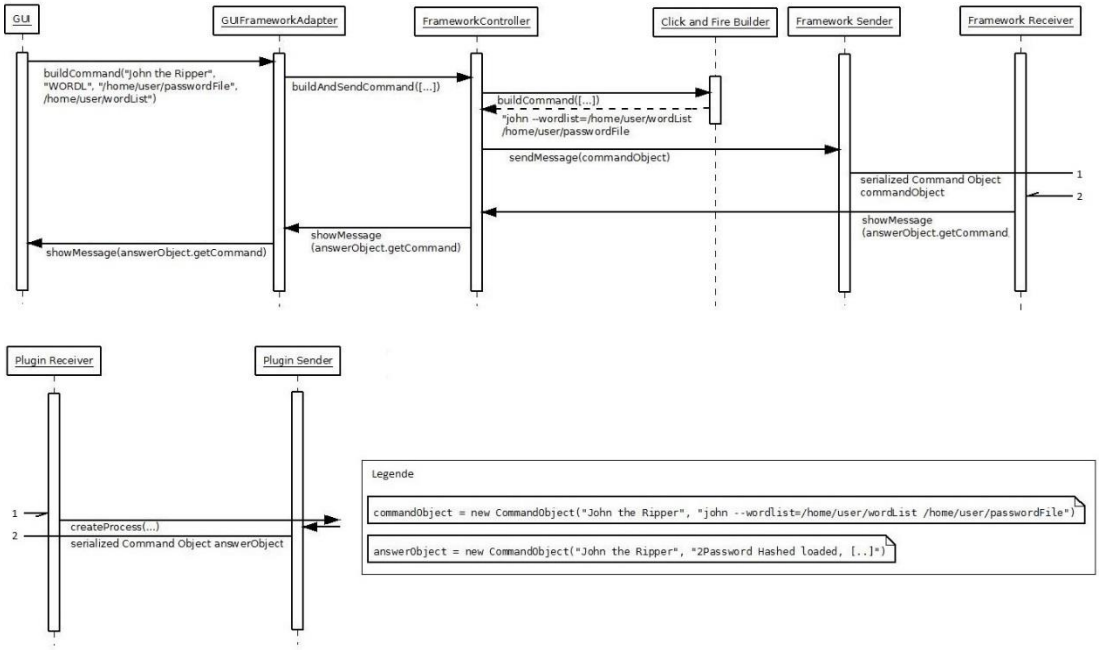






Die Nutzung eines Diensts durch ein Plugin wird durch den Benutzer ausgelöst. Dabei benutzt der Benutzer die entsprechende Sicht in der GUI, um ein Kommando an ein bestimmtes Plugin abzusetzen. Das Kommando wird in ein Command Object verpackt und über den GUIFrameworkAdapter an den Framework Controller 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 Kommando, welches im Objekt enthalten ist, extrahiert. Der Kommando-String wird, zusammen mit den anderen Parametern, an den Process Controller gegeben. Der Process Controller erzeugt mi Hilfe des Kommado-Strings einen Arbeits-Prozess, welcher den Dienst ausführt. Daraufhin wird ebenfalls ein Process Monitor gestartet, welcher die Terminierung desArbeits-Prozesses beobachtet. Zu diesem Zweck erhält der Process Monitor eine Referenz auf den Arbeits-Prozess.
Nach der Erstellung des Process Monitor wird ein ProcessWriter gestartet. Dieser kümmert sich um das Versenden von Status-Informationen zum Arbeits-Prozess und besitzt ebenfalls eine Referenz auf Selbigen.
Der Process Monitor und der ProcessWriter arbeiten Hand in Hand. Dies bedeutet, dass der ProcessWriter in regelmäßigen Abständen abfragt, ob der Arbeits-Prozess noch aktiv ist. Sollte dies der Fall sein, wird der Arbeits-Prozess aufgefordert seinen Status preis zu geben. Der aus dieser Aufforderung erhaltene String wird dann direkt in ein Command Object verpackt 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 das Ergebnis, die Status-Zeile, aus dem Objekt extrahiert und an den Framework Controller gegeben, welcher das Ergebnis wiederum bis zur GUI weiter reicht. In der GUI wird der Status des Arbeits-Prozesses dann angezeigt.
Nach Terminierung des Arbeits-Prozesses wird sein Ergebnis ebenfalls, auf die gleiche Weise wie die Status-Zeile, verpackt, versendet, empfangen und in der GUI angezeigt.

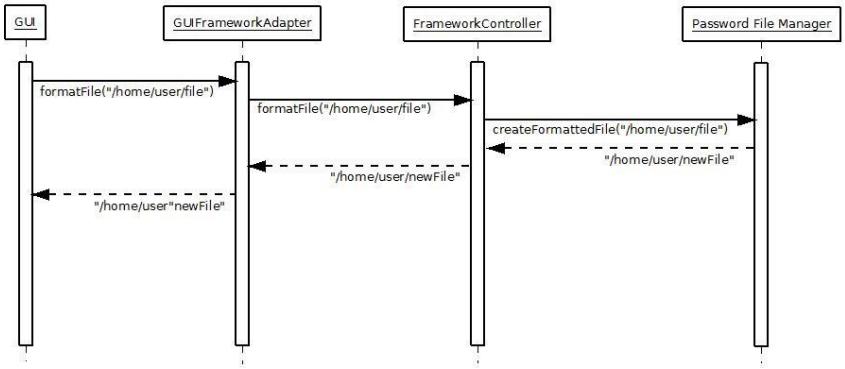
Nutzung des Click and Fire Builders:
Dieses Diagramm beschreibt den Ablauf bei der Nutzung des Click and Fire Builders.



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, mit dem Ziel ein aus den Parametern abgeleitetes korrektes Kommando zu erhalten. Der Aufruf wird wiederum 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, welche vom Benutzer angegeben wurden. Diese Umschließen die im Entwurf des Command Objects beschriebenen Parameter.
Der Framework Controller gibt das erstellte Objekt nun an den Framework Sender, wo es serialisiert und versendet wird. Der Empfänger führt dann das, im Objekt enthaltene, Kommando aus. Die im Kapitel 3.3 beschriebenen Instanzen werden ebenfalls erzeugt. Die pluginseitig erzeugten Ergebnisse werden dann, 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, welcher es dann über den GUIFrameworkAdapter an die GUI delegiert. In der GUI wird das Ergebnis dann, im entsprechenden Element, angezeigt.

Formatierung einer Passwort-Datei durch Password File Manager:

Dieses Diagramm beschreibt den Ablauf der Formatierung einer Passwort-Datei durch den Anwendungskern.



Der Ablauf einer Datei-Formatierung wird durch den Benutzer ausgelöst. Dabei gibt der Benutzer eine von zwei Datei Arten an, welche sich im Format unterscheiden. Aufgrund des gestellten Formats wird ein entsprechender Aufruf von der GUI 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 dann zurück gereicht an der Framework Controller, welcher den Dateipfad über den GUIFrameworkAdapter wieder zurück an die GUI gibt. In der GUI wird der Dateipfad, zusammen mit einer Erfolgsmeldung, angezeigt.

4 Implementierung

In diesem Kapitel geht es um die Implementierung eines Prototyps nach dem in Kapitel 3 beschriebenen Entwurf. Es werden alle Funktionalitäten implementiert, mit Ausnahme von Fehlerfällen, welche nur Teilweise implementiert sind. Zusätzlich muss noch ein geeigneter

Logging-Machanismus implementiert werden. Die GUI ist zum jetzigen Zeitpunkt noch nicht Threadsafe.
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 konkrete Implementierung dadurch vereinfacht werden konnte.
Die Implementierung wird mit Java Version 1.7 vorgenommen, das Betriebssystem ist Ubuntu 12.04LTS. Für die Lauffähigkeit auf anderen Betriebssystemen kann nicht garantiert werden.
Im Folgenden wird zwischen der Implementierung des Frameworks und der Implementierung des Plugins unterschieden. Zuletzt folgt ein Kapitel, welches Schwierigkeiten und Hürden bei der Implementierung von Programmen als Plugins aufzeigt.

4.1 Implementierung des Frameworks

4.1.1 API des Frameworks

Framework Controller:
Der Framework Controller ist die Haupt-Komponente 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 hierbei als Schaltzentrale mit erweitertem Fachwissen. Sämtliche im Framework Controller aufgerufenen Methoden werden dann an die Komponente delegiert, welche für die Ausführung der angeforderten Aufgabe zuständig ist. Die Aufrufreihenfolge, bis hin zur ausführenden Komponente, kann der entsprechenden Methoden-Beschreibung 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 ist beispielsweise die Methode sendMessage() aus dem Framework Controller lediglich eine Methode zur Delegierung an den Framework Message Sender und wird nur im Zusammenhang mit dem Framework Controller, nicht aber mit dem Framework Message Sender, beschrieben. Hingegen bildet die Methode convertFile(), aus dem Framework Controller, ab auf die Methode readPasswordFile(), aus dem Password File Manager, also wird die Methode readPasswordFile() zusätzlich im Kapitel Password File Manager beschrieben.
In Bezug auf das erweiterte Fachwissen besitzt der Framework Controller Methoden, welche Kenntnis von der Funktionsweise bereits eingebundener Plugins (Hashcat und John the Ripper) 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 CommandObjects gekapselt.
Der Aufruf dieser Methode wird an den Framework Message Sender delegiert. Dort wird das CommandObject serialisiert und über die Message Queue an alle Plugins versendet. Plugins erkennen den Empfänger der Nachricht anhand des Adressaten des CommandObjects.

void addPluginToManager(PluginObject pluginObject)
Diese Methode fügt ein PluginObject 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 zur Benutzung verfügbar gemacht werden.
Der Aufruf dieser Methode wird durch den Empfang einer Registrierungs-Aufforderung durch den Framework Message Receiver ausgelöst. Der Aufruf wird, über den Framework Controller, an den Framework Plugin Manager weiter delegiert. Dort wird eine Einfüge-Operation in die entsprechende Datenstruktur ausgeführt.

List<PluginObject> getRegisteredPlugins()
Diese Methode hat als Rückgabewert eine Liste aller registrierten Plugins. 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 **pluginName**-Parameter dient der Identifikation des korrekten GUI-Elements. Der **message**-Parameter 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 dann 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 Klartext-Passwörtern, in eine neue Datei, welche gehashte Passwörter enthält. Der **file**-Parameter identifiziert die zu konvertierende Datei. Der **algorithm**-Parameter beschreibt den Hashing-Algorithmus, welcher benutzt wird, um die Klartext-Passwörter zu verschlüsseln.

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, wo eine überladene Methode **buildPasswordFile()** aufgerufen wird. Die Funktions-Beschreibung der **buildPasswordFile()** kann dem Implementierungs-Kapitel 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 Hashing-Algorithmen, welche innerhalb der Datei identifiziert werden konnten. Die Identifizierung der genutzten Algorithmen wird von einem externen Dienst übernommen. Die Aufbereitung der gesammelten Informationen wird von der GUI übernommen.
Der Aufruf dieser Methode wird an den Password File Manager delegiert. Innerhalb des Password File Managers wird die **readPasswordFile()**-Methode aufgerufen. Daraufhin werden die gesammelten Daten mit Hilfe des, im Password File Manager gekapselten, Hashtag Managers zurückgeführt. Die vollständige Funktionsweise der **readPasswordFile()** kann dem Implementierungs-Kapitel des Password File Managers entnommen werden.

void addProcess(Integer id, String pluginName)
Diese Methode fügt eine Prozess-Referenz 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 Referenz-Objekt generiert, welches die beide Parameter enthält. Dieses Referenz-Objekt wird dann 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 eindeutige Zuordnung stattfinden kann. Somit können keine doppelten Bezeichnungen für verschiedene Prozesse entstehen. Der **id**-Parameter wird dabei vom Plugin vergeben.

void deleteProcess(Integer id, String pluginName)
Diese Methode löscht eine Prozess-Referenz 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 dann, mit Hilfe der gegebenen Parameter, die Prozess-Referenz aus der Datenstruktur.
Aufgrund der eindeutigen Zuordnung mittels beider Parameter, kann eine korrekte Löschung erfolgen. Es ist nicht möglich, das eine empfangene Referenz nicht existiert, da die Referenz bereits zu einem früheren Zeitpunkt empfangen worden sein muss.

void clearProcessList()
Der Aufruf dieser Methode wird vom Benutzer ausgelöst und beendet alle aktiven Prozesse.

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 dann eine Nachricht für jeden beendeten Prozess. Jede vom Anwendungskern empfangene Nachricht verursacht den Aufruf der **deleteProcess()**-Methode.

void buildAndSendCAFCommand(String mode, String pluginName, HashMap<String, String> info, String hashType, String passwordFilePath, String wordListFilePath, String directory)
Diese Methode wird augerufen, 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 zur Ausführung des Befehls. Die Parameter **mode**, **pluginName**, **hashType**, **passwordFilePath** und **wordListFilePath** werden dabei zur Generierung des Kommandos genutzt. Der Parameter **directory** wird, pluginseitig, 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 **mode**-Parameters. Daraufhin wird der generische Befehl des korrespondierenden PluginObjects entnommen und mit den konkreten Parametern ersetzt. Der Rückgabewert der Methode ist der ausführbare Befehl.

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

Password File Manager:
Der Password File Manager dient der Verarbeitung von Passwort-Dateien. Die Methoden des Password File Managers beschränken sich auf das Erstellen neuer Passwort-Dateien und das Identifizieren von genutzten Hashing-Algorithmen innerhalb von Passwort-Dateien.

Das Vorgehen bei der Benutzung des Password File Managers ist dabei immer gleich: Der Password File Manager wird erstellt, wobei eine Datei spezifiziert werden muss. Daraufhin ist es möglich unterschiedliche Operationen auf dem Password File Manager aufzurufen. Die Operationen generieren neue Dateien oder liefern Ergebnisse zu den spezifizierten Dateien. Somit ist es notwendig, für jede neue Passwort-Datei, einen eigenen Password File Manager anzulegen.

void buildPasswordFile(File file, PasswdAlgorithmDataType dataType, boolean enableSalt)
Diese Methode dient der Konvertierung einer Datei mit Klartext-Passwörtern, in eine neue Datei, welche gehashte Passwörter enthält. Dabei identifiziert der **file**-Parameter die Passwort-Datei mit dem Klartext. Der **dataType**-Parameter ist das Referenz-Objekt, welches den gewünschten Hashing-Algorithmus enthält. Zudem kann der Benutzer entscheiden, ob ein Salt-Wert generiert werden soll, welcher an das Passwort angehängt wird, 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, weshalb kein Rückgabewert nötig ist.

void readPasswordFile(File file)
Diese Methode dient dem Einlesen von Passwort-Dateien in den Password File Manager. Nach dem Aufruf der Methode können unterschiedliche weitere Operationen ausgelöst werden. Zum einen kann auf Einträge innerhalb der Datei zugegriffen werden, dies geschieht Zeilenweise (mit dem PasswdDataType).
Zum anderen ist es möglich Informationen zu einer Datei zu erhalten, beispielsweise, welche Hashing-Algorithmen möglicherweise genutzt wurden, welches mit dem HashtagManager bewerkstelligt wird.
Die Benutzung dieser Methode trifft keine Aussage über das weitere Vorgehen des Benutzers, sie ist sozusagen „Stand-Alone“ und hat keine Auswirkung, wenn sie nicht mit weiteren Methoden kombiniert wird.

List<PasswdDataType> readPasswordFileReturnPasswdDataTypeList(File file)
Diese Methode dient dem Erstellen einer Passwort-Datei, welche von Hashcat verarbeitet werden kann.
Sie liest eine spezifizierte Passwort-Datei ein. Daraufhin ist es möglich auf jeden Eintrag innerhalb der Datei zuzugreifen und sogar auf Teilbereiche des Eintrags zuzugreifen. Dies ist nur möglich, wenn die Datei in einem verständlichen Format vorliegt. Der Rückgabewert der Methode ist eine Liste, wobei jeder Eintrag in der Liste mit einer Zeile der Datei korrespondiert. Mit Hilfe des PasswdDataType kann nun auf Teilbereiche eines Eintrags zugegriffen werden. Dies ist notwendig, um die Erstellung einer verarbeitbaren Datei zu gewährleisten.

4.1.2 Code-Beispiele des Frameworks

Die Komponenten wurden, wie im Entwurf beschrieben, in Java umgesetzt. Ausnahme bildet dabei der Click and Fire Builder, welcher mit einfachen Mittel in den Framework Controller aufgenommen werden konnte.
Zusätzlich anzumerken ist, dass Code-Beispiele aus den Kommunikations-Modulen (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 bildet alle weiteren notwendigen Instanzen und wartet daraufhin auf Nachrichten oder Befehle vom Benutzer.
Folgend der 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;
    run();
}
```

Hierbei ist die run()-Methode zu beachten, welche den Anwendungskern in den Empfangs- und Arbeitszustand führt. Dies geschieht durch den Start eines neuen Threads, welcher sich um den Empfang von Nachrichten kümmert, so dass der Haupt-Thread Befehle vom Benutzer entgegen nehmen kann.

Datei Konvertierung:
Die Datei Konvertierung wird durch den Aufruf der convertFile()-Methode ausgelöst. Dabei wird aus dem **algorithm**-Parameter ein Referenz-Objekt abgeleitet, welches 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 Namens-Konvention bezüglich neu erstellter Dateien, kann die Ergebnis-Datei an einer übergeordneten Stelle verarbeitet werden. Aus diesem Grund besitzt die Methode keinen Rückgabewert.
Folgend 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 können sonst, aufgrund der
    //Datenverwaltung, zu falschen Ergebnissen fuehren.
    _passwdManager = new PasswdManager();
    _passwdManager.buildPasswordFile(file, pADT, "", false);
}
```

In diesem Beispiel wird der Aufruf der buildPasswordFile()-Methode des Password File Managers ohne Salt-Wert ausgeführt, dementsprechend ist der String Parameter, des Salt-Werts, leer und der Boolean-Wert false.

Bildung von „Click and Fire“-Befehlen:
Bei der Bildung von „Click and Fire“-Befehlen wird zunächst eine Anfrage an den Framework Plugin Manager gestellt. Gesucht wird dabei das PluginObject des Empfänger-Plugins. Dieses PluginObject wird benötigt, um den Befehl zum Ausführen korrekt bilden zu können.
Daraufhin wird überprüft, ob ein entsprechendes PluginObject gefunden werden konnte. 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, welche von Hashcat vorausgesetzt wird.
Sollte der Befehl an Hashcat adressiert sein, muss die vorhandene Passwort-Datei entsprechend formatiert und an Stelle der alten Passwort-Datei 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, Click and Fire Syntax, beschrieben.
Zuletzt wird der konkrete Befehl, zusammen mit allen weiteren Parametern, in ein CommandObject 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.
Folgend der Code für die buildAndSendCAFCommand()-Methode:

```
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();
            buildHashcatPWFileFromType(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);
    }
}
```

Wichtig anzumerken ist, dass zum jetzigen Zeitpunkt kein Logging-Mechanismus implementiert ist. 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.

Formatierung der Passwort-Datei für Hashcat:
Dabei wird zunächst die Passwort-Datei mit der Methode readPasswordFileAndReturnPasswdDataTypeList() eingelesen. Als Rückgabewert erhält man eine Liste mit PasswdDatatypes. Jeder PasswdDatatype repräsentiert eine Zeile aus der Passwort-Datei.
Nun wird durch die Liste iteriert. Bei der Iteration wird jede Zeile aus der Passwort-Datei entsprechend des **hashType**-Parameters formatiert und in eine neue Datei geschrieben. Aufgrund der strengen Namens-Konventionen bezüglich neu erstellter Dateien, wird kein Rückgabewert benötigt, um die neue Datei aus einer übergeordneten Instanz zu finden.
Folgend der Code der buildHashcatPWFileFromType()-Methode:

```
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) {
    }
}
```

Bildung des Befehls aus dem PluginObject:

Bei der Methode buildCommandStringForCAF() wird zunächst geprüft, in welchem Modus der Befehl ausgeführt werden soll, da jeder Modus eine potenziell unterschiedliche Syntax besitzt, um korrekt ausgeführt werden zu können.

Daraufhin wird der generische Befehl des mitgelieferten PluginObjects abgerufen und die enthaltenen generischen Parameter werden durch die konkreten Parameter ersetzt. Der Rückgabewert der Methode ist der korrekt gebildete Befehl, welcher nur noch in das CommandObject gekapselt werden muss:

```
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;
}
```

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

2. Prozessverwaltungs-Nachrichten: 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“.

4.2.1 Nachrichtenverarbeitung des Plugins

Registrierungs-Aufforderung:
Nach der Initialisierung versendet das Plugin einer Registrierungs-Aufforderung an den Anwendungskern. Diese Nachricht kapselt ein PluginObject und dient der Verwaltung der Plugins, so dass der Anwendungskern weiß, welche Plugins existieren und wie die generischen Strings zur Verarbeitung von „Click and Fire“-Befehlen aussehen.

Befehls-Nachrichten:
Nachdem ein Plugin initialisiert wurde verschickt es eine Registrierungs-Aufforderung und begibt sich in den Empfangs-Zustand.
Sollte nun eine Befehls-Nachricht empfangen werden, wird eine ProcessControl-Instanz gestartet, welche alle benötigten Parameter enthält. Diese Instanz führt den Befehl aus.
Zusätzlich wird der laufende Prozess in die ProcessManagement Komponente eingetragen und eine „ADDPROCESS“ Nachricht, zusammen mit einem Referenz-Objekt auf den Prozess, wird an den Anwendungskern verschickt. Dies führt beim Anwendungskern dazu, dass der laufende Prozess in die Framework Process Management Komponente eingetragen wird.

Prozessverwaltungs-Nachrichten:
Nachdem ein Plugin initialisiert wurde verschickt es eine Registrierungs-Aufforderung und begibt sich in den Empfangs-Zustand.
Beim Empfang einer Prozess-Verwaltungs-Nachricht wird zunächst der Command-String geprüft, welcher das Kommando „KILLPROCESS“ enthalten muss. Daraufhin wird die Methode killAllProcesses() des Plugin Process Managers aufgerufen, welche sich um die Terminierung aller laufenden Prozesse kümmert.
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 Referenz-Objekt auf den zu löschenden Prozess. Auf diese Weise kann der Anwendungskern eindeutig feststellen, welcher Prozess beendet wurde.

4.2.2 Code-Beispiele des Plugins

Empfangs-Schleife:
Bei der Empfangs-Schleife 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.
Folgend nun der Code der Empfangs-Schleife des Plugin Message Receivers:

```
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("Hashcat")) {
        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")){
        //ProcessControl sendet bei Terminierung eine DELETEPROCESS-Nachricht
        _processManager.killAllProcesses();
    }
    else {
        //Objekt ist an ein anderes Plugin adressiert
    }
}
```

Die statische Methode create() der Process Control 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().

Prozess-Verarbeitung:
Nach dem Start einer Instanz der Process Control-Klasse wird die run()-Methode aufgerufen, welche sich um die korrekte Verarbeitung des Prozesses kümmert. Dabei wird die run()-Methode als neuer Thread gestartet, um den Haupt-Thread 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 dann als Liste an den Java ProcessBuilder gegeben. Daraufhin wird das Arbeitsverzeichnis gesetzt und der Prozess wird gestartet.
Nachdem der Prozess, innerhalb des ProcessBuilders, gestartet wurde, werden benötigte Verwaltungs-Prozesse getstartet. Bei Diesen handelt es sich um einen Überwachungs-Prozess zur Terminerung des Arbeits-Prozesses, sowie um den Status-Prozess, welcher Status-Nachrichten zum Arbeits-Prozess abfragt und diese an den Anwendungskern verschickt. Beide brauchen dafür eine Referenz auf das ProcessBuilder-Objekt. Zusätzlich dazu benötigt der Status-Prozess Referenzen auf den Überwachungs-Prozess, um feststellen zu können wann der Prozess terminiert, und den Plugin Message Sender, um die Status-Nachrichten versenden zu können.
Nun können die benötigten Reader und Writer für den ProcessBuilder generiert werden. Welche benötigt werden, um den Output zu lesen und Nachrichten an den ProcessBuilder zu senden.
Daraufhin wird eine blockierende „while(true)“-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 Prozess-Referenz, an den Anwendungskern versendet und der Prozess aus dem Plugin Process Manager gelöscht.
Folgend der Code der run()-Methode der Process Control-Klasse:

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

    //Befehl übergeben, Directory setzen
    ProcessBuilder proB = new ProcessBuilder(formattedText);
    proB.directory(new File(_directory));
    proB.redirectErrorStream(true);
    Process process;
    try {
        //Prozess starten
        process = proB.start();
        _process = process;
        //Terminierungsüberwachung des Prozesses
        ProcessMonitor proM = ProcessMonitor.create(process);
        processMonitor = proM;
        //Status Abfrage
        ProcessWriterAsync.create(proM, process, _plugMessSender,
            _directory, _interval, _cafFlag);

        //schreibender Zugriff
        OutputStreamWriter osw = new OutputStreamWriter(process.getOutputStream());
        BufferedWriter bw = new BufferedWriter(osw);
        //lesender Zugriff
        InputStreamReader isr = new InputStreamReader(process.getInputStream());
        BufferedReader br = new BufferedReader(isr);

        String line;
        //Prozess output
        while ((line = br.readLine()) != null) {
            _plugMessSender.sendMessage("Hashcat", 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();
    }
}
```

4.3 Implementierung der Message Queue

…

4.4 Mögliche Probleme beim Einbinden von Programmen

…

5 Literaturverzeichnis

[ISTR 2013] Symantec Corporation
Internet Security Threat Report 2013
Volume 18

[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

[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 Sysmposium

[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

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

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

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

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 _____ _____