Institut für Informatik
der Technischen Universität München

Systementwicklungsprojekt

Entwicklung eines Mobilen Agenten zur Überprüfung der Sicherheit von Passwörtern

Bearbeiter: Stefan Heinz Thom

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Helmut Reiser
Harald Rölle

Abgabetermin: 15.05.2002

Inhaltsverzeichnis

Inhaltsverzeichnis 2
1 Motivation und Aufgabenstellung 3
2 Crack 5
2.1 Funktionsweise 5
2.2 Benutzung 6
2.3 Relevante Implementierungsdetails 6
3 MASA 7
3.1 Überblick 7
3.2 Implementierung eines MASA Agenten 7
4 Design 9
4.1 Parallelisierung 9
4.1.1 Verteilung nach Benutzern der Passwortdatei 9
4.1.2 Verteilung nach Wörterbüchern 9
4.1.3 Verteilung nach Regeln 10
4.1.4 Gewählte Parallelisierung 10
4.2 Verwaltung der Verteilung 10
4.3 Verteilung der Teilaufgaben 11
4.4 Gesamtablauf beider Agentenarten 11
4.5 Schnittstelle des MasterCrack 12
4.6 Schnittstelle des WorkerCrack 14
5 Der MasterCrack Agent 16
5.1 Einlesen der Passwortdatei un d der Wörterbücher 16
5.2 Bilden der Teilaufgaben 17
5.3 Erzeugen der WorkerCrack-Agenten und Übergabe ihrer Teilaufgabe 18
5.4 Sammeln der Ergebnisse der WorkerCrack-Agenten 18
5.5 Verwaltung der Aufenthaltsorte der erzeugten WorkerCrack-Agenten 18
5.6 Verwaltung der Agentensysteme, die Crack nicht ausführen können 19
5.7 Präsentation der Ergebnisse 19
6 Der WorkerCrack Agent 21
6.1 Parallelisierung von Crack 21
6.2 Erzeugen der Crack-Instanz 21
6.3 Erstellung des Berichtes 22
6.4 Migration 22
6.5 Feststellen und Berichten des Fortschritts 23
7 Leistungsbetrachtung 24
8 Benutzeranleitung 26
8.1 Installation 26
8.2 Benutzung 27
9 Zusammenfassung und Ausblick 30
9.1 Zusammenfassung 30
9.2 Ausblick 30
10 Literatur 31

1 Motivation und Aufgabenstellung

Passwörter sind nach wie vor eines der meistverbreitesten Mitteln zur Authentifizierung und Authorisation bei Computersystemen. Deshalb liegt es im Interesse des Benutzers und auch des Betreiber des Systems, dass diese Passwörter nicht von Dritten erraten werden.
Aus Sicht des Benutzer entstehen bei einem erratenen Passwort folgende negativen Folgen:

- Der Eindringling kann Leistungen benutzen für die der Benutzer zahlt
- Der Eindringling kann Daten des Benutzers auf dem Computersystem beschädigen
- Der Eindringling kann Daten ausspähen

Aus Sicht des Betreiber bzw. Anbieters entstehen diese negativen Folgen:

- Der Eindringling kann den Zugang zum weiteren Eindringen ins das System benutzen, z.B. root-Priviliegien erschleichen
- Missbrauch des Systems durch den Eindringling zu illegalen Zwecken, z.B. verteile Denial of Service Angriffe
- Probleme bei der Abrechnung der erschlichenen Leistungen, da der betroffene Benutzer diese nicht bezahlen will
- Imageverlust bei Bekanntwerden

Es ist daher in beiderseitigem Interesse „schwache“ Passwörter nicht zuzulassen. Ein schwaches Passwort ist ein Passwort das erraten werden kann. Ein Passwort kann erraten werden, wenn es sich aus dem Benutzernamen ableiten lässt oder der Name einer Person oder Sache ist.
Der Benutzer kann schwache Passwörter vermeiden, indem er einfach kein schwaches Passwort wählt.
Der Betreiber hat die Möglichkeit schwache Passwörter zu verhindern, indem er die Benutzer ihre Passwörter nicht wählen lässt, sondern sie erzeugen lässt. Aber dies selbst birgt Risiken in sich. So könnten sich im Programm, dass diese Passwörter erzeugt, Schwächen befinden, die es ermöglichen diese erzeugten Passwörter zu erraten. Auch können äussere Bedingungen dafür sorgen, dass eine Zuteilung von Passwörter nicht erwünscht oder möglich ist.
Wählen die Benutzer ihr Passwort selbst, hat der Betreiber nur die Möglichkeit diese auf schwache Passwörter zu überprüfen. Da die Passwörter nur verschlüsselt vorliegen, da die unverschlüsselte Speicherung der Passwörter ein viel höheres Risiko ist, bleibt in diesem Fall nur der Einsatz von sogenannten Passwort-Crackern, die nach dem Brute-Force Prinzip schwache Passwörter ausprobieren. Dabei hat der Betreiber Angreifern von aussen voraus, dass ihm das gesamte Passwortfile vorliegt und er daher die Namen der Benutzer kennt. Dieser Vorteil entfällt natürlich, falls der Angreifer über einen Benutzer-Account verfügt (d.h. ihn legal besitzt oder ihn auf andere Weise „gehackt“ hat).
Ein solcher Passwort-Cracker ist Crack. Crack benutzt Wörterbücher und die Namen der Benutzer, um in einem Passwortfile schwache Passwörter zu finden.
Ein Durchlauf von Crack durch ein Passwortfile kann allerdings sehr lange dauern, insbesondere wenn das Passwortfile gross ist, das heißt wenn viele Benutzer überprüft werden sollen, da viele mögliche Passwörter verschlüsselt und mit den Einträgen verglichen werden müssen. Daher bietet es sich an diese Rechenlast zu verteilen, um die Gesamtzeit zu verkürzen.
Als Plattform hierfür bietet sich MASA an. MASA steht für Mobile Agent System Architecture und bietet den für sie geschriebenen Agenten Kommunikation über Systemgrenzen und Mobilität. Daher bietet es sich an einen MASA-Agenten zu schreiben, der die Arbeit in Teilaufgaben aufteilt und dann auf den verfügbaren Agentensystem parallel mit Hilfe von Crack bearbeitet.

Kapitel 2 gibt einen Überblick über Crack: seine Funktionsweise, die Benutzung und die für die Entwicklung des Agenten relevanten Implementierungsdetails.
Kapitel 3 gibt eine kurze Einführung in MASA und wie man einen MASA-Agenten implementiert.
In Kapitel 4 wird das Gesamt-design beschrieben. Es werden die Möglichkeiten zur Parallelisierung, der Verwaltung und der Verteilung der Teilaufgaben beschrieben und welche Möglichkeiten gewählt wurden. Die dabei getroffenen Entscheidungen führten zur Implementierung zweier Agentenarten deren Zusammenspiel anhand des Gesamtablauf und der öffentlichen Schnittstellen erklärt wird..
Kapitel 5 beschreibt die Implementierung des sogenannten MasterCrack-Agenten. Er übernimmt die Erstellung und Verwaltung der Teilaufgaben, sowie die Präsentation der Ergebnisse.
Kapitel 6 beschreibt die Implementierung des sogenannten WorkerCrack-Agenten. Er bearbeitet die Teilaufgaben und meldet die Ergebnisse an den MasterCrack-Agenten.
Kapitel 7 enthält eine kurze Leistungsbetrachtung der Implementierung.
In Kapitel 8 wird die Installation und Benutzung der beiden Agenten beschrieben.
Kapitel 9 gibt eine Zusammenfassung sowie einen Ausblick.

2 Crack

Crack wurde von Alec Muffett geschrieben und ist ein Programm zum Aufspüren von schwachen Passwörtern in Unix oder Unix-ähnlichen Passwortdateien.
Das Programm lag mir in der Version 5.0a vor und ist unter der Adresse http://www.users.dircon.co.uk/~crypto/ erhältlich. Crack ist ein Programm zur Überprüfung der Sicherheit von Passwörtern in Unix-Passwortdateien, indem es Wörter aus Wörterbücher als Passwörter versucht und bei Übereinstimmung die betroffenen Passwörter als unsicher berichtet.

2.1 Funktionsweise

Crack generiert aus der zu bearbeitenden Passwortdatei zwei zusätzliche Wörterbücher. Das erste enthält Wörter die direkt aus den Benutzernamen abgeleitet wurden und das zweite enthält Wörter, die durch Permutation und Kombination von Teilen der Benutzernamen erstellt wurden.
Aus diesen und den vorhandenen Wörterbüchern werden nun durch konfigurierbare Regeln sogenannte „Guesses“, das heißt Versuche zum Erraten der Passwörter, erstellt. Diese Versuche können so verdoppelte Einträge, rückwärts geschriebene Einträge, Teile eines Eintrags oder Einträge, bei denen bestimmte Zeichen durch andere ersetzt wurden sein.
Diese so generierten Versuche werden dann verschlüsselt und mit den Einträgen in der Passwortdatei verglichen. Der dabei verwendete Verschlüsselungsalgorhitmus bzw. Implementierung kann konfiguriert werden und ist nicht Teil von Crack. Als Verschlüsselungsimplementierung wird „Libdes“ von Eric Young mitgeliefert, aber man kann auch andere Verschlüsselungsbibliotheken benutzen. „Libdes“ ist eine Implementierung von DES, die eine schnelle Version der traditionellen `crypt`-Funktion enthält. Diese hat neben dem zu verschlüsselnden Passwort einen zweiten Parameter, der salt genannt wird. Dieses salt beeinflusst das Ergebnis der Funktion, so dass bei gleichen Passwörtern aber verschiedenen salt verschiedene Ergebnisse erzeugt werden. Das verwendete salt wird durch die ersten beiden Zeichen des verschlüsselten Passworts angegeben und kann 4096 verschiedene Werte annehmen. Crack muss also für jeden Benutzer dieses salt auslesen, den Versuch mit diesem salt verschlüsseln und dann das Ergebnis der Verschlüsselung mit dem Eintrag in der Passwortdatei vergleichen. Stimmen das Ergebnis der Verschlüsselung und des Eintrag überein, so ist der Versuch das Passwort des entsprechenden Benutzers.
Hat Crack auf diese Weise ein Passwort erraten, kann es eine email an den betroffenen Benutzer senden, in dem ihm mitgeteilt wird, dass das von ihm gewählte Passwort schwach ist und er es ändern sollte. Ansonsten kann man sich jederzeit die erratenen Passwörter ausgeben lassen.
Die bereits enthaltenen Wörterbücher enthalten Namen von Gegenständen, Personennamen, Begriffe aus dem Computerbereich und StarTrek, sowie ausländische Wörter (da der Programmierer von Crack Engländer ist, sind hier andere Sprachen als Englisch gemeint). Auch andere leicht erratbare Passwörter, wie 12345, sind enthalten. Diese Wörterbücher liegen thematisch zusammengefaßt als Dateien im dwg Format in den Unterverzeichnissen `dict/1`, `dict/2` und `dict/3`. Dabei liegen die Wörterbücher mit den wahrscheinlichsten Passwörtern in `dict/1`und die Wörterbücher mit den unwahrscheinlichsten in `dict/3`. Dementsprechend werden die Wörterbücher in `dict/1` als erstes versucht. Da ein Durchlauf

von Crack sehr lange dauern kann, soll damit sicher gestellt werden, daß die schwächsten Passwörter möglichst schnell festgestellt werden.

2.2 Benutzung

Die Benutzung von Crack ist einfach. Crack selbst ist ein Shellscript, in diesem müssen nur Optionen für das Kompilieren eingestellt werden, das heißt der Name des Compilers, Flags und der Pfad zur verwendeten Verschlüsselungsbibliothek. Bei der Verschlüsselungsbibliothek muss man sicherstellen, dass diese durch einen Aufruf von make im angegebenen Verzeichnis korrekt übersetzt wird.
Bei einer Passwortdatei im Standartformat reicht dann der Aufruf von „`Crack passwortdatei`“. Nach dem Starten läuft Crack nun im Hintergrund. Durch den Aufruf des Shellsscripts Reporter kann man dann jederzeit die bereits erratenen Passwörter abfragen.

2.3 Relevante Implementierungsdetails

Crack besteht aus mehreren Programmen, die durch das Shellscript Crack aufgerufen werden. Als erstes werden die Programme unter zu Hilfenahme von make compiliert. Die kompilierten Dateien werden in einem Unterverzeichnis `run/ARCHITECTURE` abgelegt. Die Wörterbücher werden nach `run/dict` kopiert, eventuell vorhandene doppelte Einträge entfernt und dann komprimiert. Danach werden die aus den Benutzernamen generierten Wörterbücher erstellt und im selben Verzeichnis abgelegt. Sodann wird die Passwortdatei eingelesen und in einem für den eigentlichen Passwortcracker mkcracker lesbaren Format als Datei I<Rechnername>.<Prozessnummer> im Unterverzeichnis `run` abgelegt. Mit dieser Datei als Eingabe wird dann mkcracker gestartet. Falls Crack mit der Option –fgnd gestartet wurde, wird diese Ausgabe von mkcracker über die Standartausgabe ausgebenen. Ohne diese Option läuft Crack nach dem es mkcracker gestartet hat im Hintergrund weiter und schreibt Ausgabe von mkcracker in eine Datei namens D<Rechnername>.<Prozessnummer> im Unterverzeichnis run.
Das Shellscript Reporter filtert aus dieser Datei die erratenen Passwörter und gibt diese über Standartausgabe an.
Wird Crack beendet, kann man die Arbeit fortsetzen lassen, indem man die Datei mit der Ausgabe des Passwortcrackers (also D<Rechnername>.<Prozessnummer>) umbenennt und dann Crack mit der Option –recover und der umbenannten Ausgabedatei als Passwortdatei startet, also „`Crack -recover run/<umbenannte Ausgabedatei>`“.
Der Passwortcracker arbeitet die oben erwähnten Regeln der Reihe nach ab und schreibt das Ergebnis jeweils in die Ausgabe. Durch das Einlesen dieser Ausgabe kann die neue Crackinstanz erkennen, bei welcher Regel die vorherige Instanz abgebrochen wurde und dort fortfahren.

3 MASA

3.1 Überblick

MASA steht für Mobile Agents System Architecture und wurde vom Institut für Informatik der Ludwig-Maximilians-Universitäts München als Plattform für flexibles IT-Managment entwickelt. Dabei implementiert MASA den MASIF-Standart der OMG [MASIF]. Eine ausführliche Beschreibung von MASA findet sich in [GHR99].
MASA besteht aus Agentensystemen und Agenten. Agentensysteme stellen die Ablaufumgebung für die Agenten bereit. Die Agenten laufen auf diesen Agentensystemen und können für die vielfältigsten Funktionen benutzt werden. Mobilen Agenten ist auch möglich zu migrieren, dass heißt das Agentensystem zu wechseln.
Agenten und Agentensysteme kommunizieren untereinander mittels CORBA. Implementiert wurde MASA in Java, damit ist eine gewisse Plattformunabhängigkeit erreicht.
Gesteuert wird MASA über in HTML-Seiten eingebettete Applets. Dieses Applets kommunizieren mit den von ihnen gesteuerten Instanzen ebenfalls über CORBA.

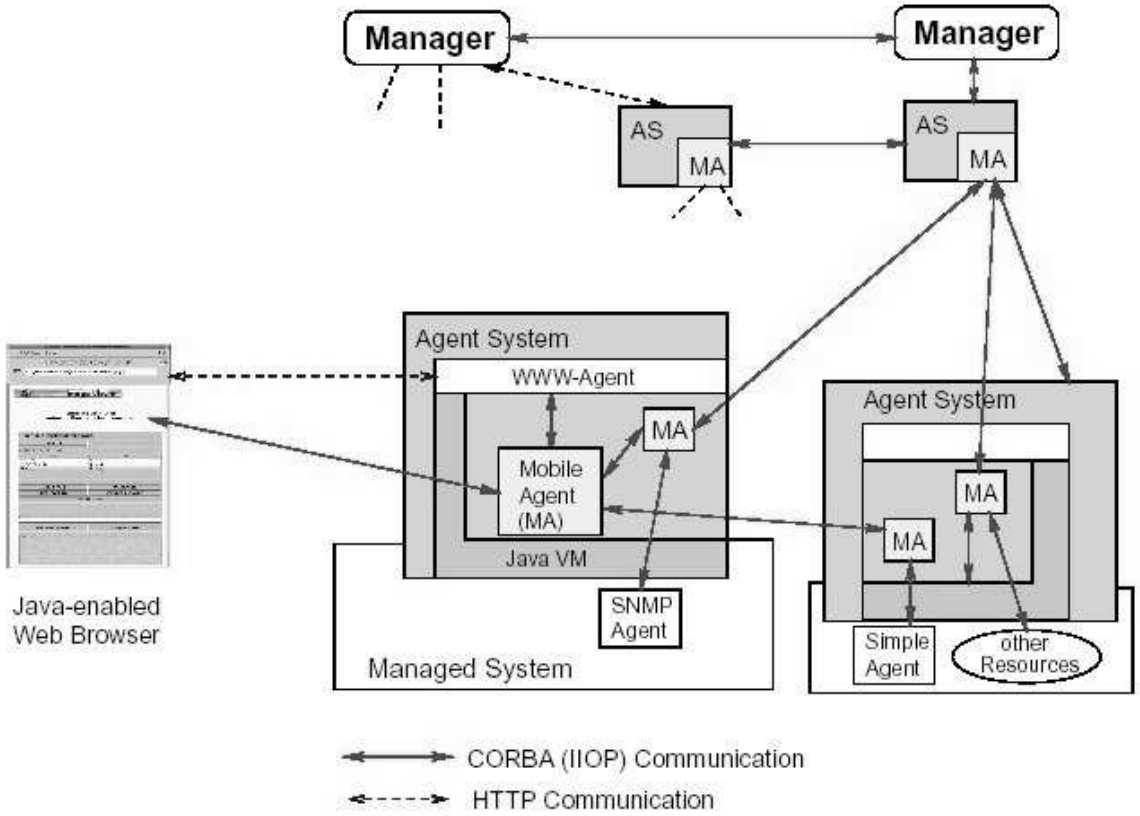

Abbildung 1: MASA Architektur aus GHR99

3.2 Implementierung eines MASA Agenten

Um einen neuen MASA-Agenten zu schreiben ruft man bei einem installierten MASA im system-Verzeichnis das Make-Ziel newagent auf. Dieses erzeugt ein neues Unterverzeichnis

im MASA-Verzeichnis mit dem Namen des neuen Agenten und legt in diesem Verzeichnis den Rumpf des Agenten und seines Applets ab.
Um den Agenten nun zu implementieren muss man drei Dateien vervollständigen. Im Verzeichnis `src/idl` muss man über die Datei <Agentname>.idl die Schnittstelle des Agenten in IDL definieren. Hierbei definiert man die Methoden, die der Agent anderen Agenten zur Verfügung stellt. Auch muss man hier die Methoden deklarieren mit denen das Agentenapplet den Agenten steuern soll.
Der Quellcode des eigentlichen Agenten befindet sich in der Datei <Agentenname><Typ des Agenten>.java im Verzeichnis `src/<Agentenname>`. In dieser Datei befindet sich bereits ein Rumpf der vervollständigt werden muss. Diese Javaklasse implementiert das Interface, dass aus der IDL Beschreibung der Schnittstelle generiert wird.
Im Verzeichnis `src/<Agentenname>/applet` befindet sich der Quellcode für das Applet des Agenten. Hier ist ebenfalls bereits ein Rumpf vorgegeben.

4 Design

In diesem Kapitel wird vorgestellt welche Möglichkeiten der Parallelisierung offen standen und welche Möglichkeit gewählt wurde. Ferner wird beschrieben wie die Verteilung verwaltet wird und wie die Teilaufgaben verteilt werden. Dies führte zu zwei verschiedenen Agenten und deren Zusammenspiel wird anhand des Gesamtablaufs und der Schnittstellen der beiden Agenten erklärt.

4.1 Parallelisierung

Zur Parallelisierung stehen bei Crack prinzipiell drei Möglichkeiten zur Verfügung: Das Verteilen der Benutzer aus der Passwortdatei, der Wörterbücher und der Regeln.
Im folgenden werden die Vor- und Nachteile dieser Möglichkeiten beschrieben.

4.1.1 Verteilung nach Benutzern der Passwortdatei

Bei dieser Möglichkeit wird die Passwortdatei in mehrere Stücke aufgeteilt und diese von den verschiedenen Crackinstanzen bearbeitet.
Dabei muss das Programm Crack selbst nicht modifiziert werden. Es muss nur mit dem jeweiligen Stück der Passwortdatei aufgerufen werden.
Bei dieser Art der Parallelisierung wird so gut wie keine Arbeit mehrfach geleistet, die bei einem nicht parallelisierten Lauf nur einmal gemacht wird. Nur die Generierung der „Guesses“ aus den Wörterbucheinträgen wird mehrfach gemacht. Auch bei der Verschlüsselung tritt keine Mehrarbeit auf, da wie im Kapitel 2 beschrieben durch das salt die „Guesses“ für jeden Benutzer neu verschlüsselt werden müssen. Damit hat diese Parallelisierung eine sehr gute parallele Effizienz.
Ein weiterer Vorteil ist die gute Skalierbarkeit, da die Motivation für die Parallelisierung von Crack die lange Laufzeit bei grossen Passwortdateien, sprich vielen Benutzern, ist und damit lassen sich aus der Passwortdatei viele Stücke bilden.
Desweiteren sind die gebildeten Stücke gleichmässig, das heißt die gebildeteten „Teilaufgaben“ haben keine grossen Unterschiede in der Grösse und somit der Last.

4.1.2 Verteilung nach Wörterbüchern

Hierbei werden die Wörterbücher unter den verschiedenen Crackinstanzen aufgeteilt, so dass jede Instanz einen anderen Satz von Wörterbüchern erhält.
Allerdings werden die Wörterbücher, die aus der Passwortdatei generiert werden, bei jeder Instanz generiert und bearbeitet. Insbesondere versuchen somit am Anfang sämtliche Crackinstanzen dieselben Kombinationen an denselben Benutzer.
Ein weiterer Nachteil ist die Ungleichmässigkeit der gebildeteten „Teilaufgaben“. Da die Wörterbücher thematisch zusammenhängende Wörter enthalten, sind sie nicht gleich gross und somit auch die daraus gebildeten Teilaufgaben.
Auch ist die Anzahl der Wörterbücher im Normalfall immer gleich. Damit ergibt sich ein Maximum für die Anzahl der gebildeten Teilaufgaben

Der Eingriff in das Crack-Programm ist nicht sehr gross. Bei der erzeugten Crack-Instanz müssen die Wörterbücher im Unterverzeichnis dict gegen die Wörterbücher der zu bearbeitenden Teilaufgabe ausgetauscht werden.

4.1.3 Verteilung nach Regeln

Bei dieser Möglichkeit werden die Regeln, die bestimmen wie aus den Wörterbüchern die Versuche generiert werden, auf die Crackinstanzen verteilt.
Der Vorteil dieser Lösung ist, dass keine Arbeit mehrfach gemacht wird.
Allerdings ist die Implementierung sehr schwierig, da die Regeln nicht wie die Wörterbücher einfach in Dateien zusammengefasst werden, sondern durch eine spezielle Datei aufgerufen werden. Somit kann auch eine in einer späteren Crackversion mögliche Änderung an diesem Mechanismus, auch eine Änderung an der Implementierung der Aufteilung nötig machen.
Fernerhin gilt wie bei den Wörterbüchern, dass die Anzahl der Regeln dieselbe ist und damit ein Maximum für die Anzahl der gebildeten Teilaufgaben existiert.

4.1.4 Gewählte Parallelisierung

Aufgrund des grossen Eingriffs in die Wirkungsweise von Crack wurde von der Wahl der Verteilung der Regeln abgesehen.
Wegen des geringen Eingriffs in Crack, der gleichmässigen Teilaufgaben. der guten Skalierbarkeit und guten parallelen Effizienz wurde die Verteilung nach Benutzer favorisiert.
Nur wenn die Anzahl der verfügbaren Systeme grösser ist als die Anzahl der aus der Passwortdatei bildbaren Teilaufgaben, also mehr Systeme als Benutzer verfügbar sind, wird geprüft ob sich aus den Wörterbüchern mehr Teilaufgaben bilden lassen und falls dies so ist werden die Wörterbücher verteilt.

4.2 Verwaltung der Verteilung

Zur Verwaltung der Verteilung wurden zwei Möglichkeiten betrachtet. Die erste Möglichkeit sind für jede Crackinstanz gleichartige Agenten die sich über gemeinsames Medium über die Verteilung verständigen. Die zweite Möglichkeit ist die Aufteilung auf zwei verschiedene Agentenarten, wobei eine die Teilaufgaben erstellt und an die andere Agentenart verteilt, die diese dann mit Hilfe von Crack bearbeitet.
Aus folgenden Gründen wurden 2 Agentenarten implementiert:

- Der Benutzer muss nur einen Agenten starten. Bei gleichartigen Agenten hätte der Benutzer entweder auf jedem System manuell einen Agenten starten, oder ein Agent hätte sich als besonderer Agent auszeichnen müssen, der die anderen startet.
- Die Implementierung der Verteilung und die Implementierung der Bearbeitung wird getrennt. Somit kann eines der beiden leichter geändert werden, ohne den anderen Teil ebenfalls umzuschreiben.
- Es existiert eine zentrale Instanz, die die Ergebnisse präsentiert.
- Es hätte ein kompliziertes Protokoll entwickelt werden müssen, wie die Agenten die Aufgaben verteilen und was bei neuhinzugekommen Agenten passiert, ohne dass dadurch ein Vorteil erreicht würde.

Die eine Agentenart, genannt MasterCrack-Agent liest die Passwortdatei und die Wörterbücher ein, erstellt daraus die zu bearbeitetenden Teilaufgaben und verteilt sie an die mehrere Agenten der zweiten Agentenart, genannt WorkerCrack-Agenten. Diese bearbeiten mit Hilfe von Crack diese Teilaufgaben und teilen das Ergebnis ihrem MasterCrack-Agenten mit. Er präsentiert die gesammelten Ergebnisse dem Benutzer über sein Applet.

4.3 Verteilung der Teilaufgaben

Als nächstes stellte sich die Frage wieviele Teilaufgaben erstellt werden sollen und wie sie verteilt werden sollen.
Der MasterCrack-Agent könnte mehr Teilaufgaben als Systeme erstellen, dann soviel WorkerCrack-Agent wie Systeme erstellen und diesen die entsprechende Anzahl Teilaufgaben mitgeben. Nachdem ein WorkerCrack-Agent seine Teilaufgabe bearbeitet hat, erhält er eine neue Teilaufgabe und bearbeitet diese.
Als zweite Möglichkeit könnte der MasterCrack-Agent so viele Teilaufgaben erstellen, wie Systeme vorhanden sind und genau so viele WorkerCrack-Agenten erstellen, die diese Teilaufgaben bearbeiten und nach deren Bearbeitung dem MasterCrack-Agenten ihr Ergebnis mitteilen.

Die gewählte Methode ist die zweite Möglichkeit, d.h. es werden so viele Teilaufgaben erstellt wie Systeme vorhanden sind. Da bei der gewählten Weise der Parallelisierung erst zur Laufzeit entschieden wird, ob nach Benutzern oder nach Wörterbüchern verteilt wird, muss eine Teilaufgabe für einen WorkerCrack-Agenten die von diesem WorkerCrack-Agenten zu bearbeitende Passwortdatei und die zu verwendenden Wörterbücher enthalten, d.h. beides muss an den WorkerCrack-Agenten übertragen werden, der diese Teilaufgabe bearbeiten soll.
Bei einer Verteilung nach Benutzern, wird dann bei jeder Teilaufgabe der komplette Satz von Wörterbüchern mitübertragen, bei einer Verteilung nach Wörterbüchern entsprechend die gesamte Passwortdatei. Daher würde die erste Möglichkeit einen höheren Netzwerkverkehr erzeugen. Ferner werden alle Benutzer gleichzeitig bearbeitet, so dass schwächere Passwörter, entsprechend der Intention des Crack Autors, möglichst schnell gefunden werden. Dazu wurde im WorkerCrack-Agent eine Möglichkeit der Abfrage der bereits gefundenen Passwörter implementiert.

4.4 Gesamtablauf beider Agentenarten

Somit ergibt sich folgender Ablauf des Aufspürens schwacher Passwörter in einer gegebenen Passwortdatei:
Der MasterCrack-Agent liest die gegebene Passwortdatei, sowie die Wörterbücher einer angegebenen Crackinstallation ein. Anschliessend stellt er fest, wieviele Agentensysteme in seiner MASA-Region existieren und entscheidet dann über die Art der Verteilung.
Ist die Anzahl der Benutzer in der Passwortdatei grösser oder gleich der Anzahl der gefundenen Agentensystem, erstellt er die Teilaufgaben indem jede Teilaufgabe einen entsprechenden Bruchteil der Passwortdatei und den gesamten Satz der Wörterbücher erhält.
Andernfalls wird die Anzahl der Benutzer und die Anzahl der vorhandenen Wörterbücher verglichen. Ist die Anzahl der Benutzer höher, wird ebenfalls die oben genannte Methode

benutzt. Sind jedoch mehr Wörterbücher als Benutzer vorhanden, enthält jede gebildete Teilaufgabe die gesamte Passwortdatei und einen entsprechenden Bruchteil der Wörterbücher.
Die Anzahl der gebildeten Teilaufgaben entspricht dabei der Anzahl der gefundenen Agentensystemen, es sei denn es können nicht mehr gebildet werden, da nicht genügende Benutzer und Wörterbücher vorhanden sind.
Anschliessend erzeugt der MasterCrack-Agent soviele WorkerCrack-Agenten wie Teilaufgaben erstellt wurden und verteilt diese auf die gefundenen Agentensysteme. Die erstellten WorkerCrack-Agenten bearbeiten auf ihren Systemen ihre vom MasterCrack-Agenten zugeteilte Teilaufgabe. Der WorkerCrack-Agent der auf dem selben Agentensystem wie der MasterCrack-Agent arbeiten soll, benötigt dabei ein entsprechendes Signal, dass er mit der Arbeit starten soll. Die anderen WorkerCrack-Agenten erhalten dieses Signal durch die Migration.
Nach Abschluss der Bearbeitung ihrer Teilaufgabe berichten die WorkerCrack-Agenten ihr Ergebnis an den MasterCrack-Agenten.
Der MasterCrack-Agent sammelt die Berichte der WorkerCrack-Agenten und präsentiert dem Benutzer diese dann über sein Applet.
Um den Benutzer auch Zwischenergebnisse präsentieren zu können, melden die WorkerCrack-Agenten ihren Fortschritt und den Zwischenbericht in einem einstellbaren Zeitabstand an ihren MasterCrack-Agenten. Der Benutzer hat auch die Möglichkeit den MasterCrack-Agenten zu veranlassen, von allen WorkerCrack-Agenten einen aktuellen Zwischenbericht einzuholen.
Ferner speichert der MasterCrack-Agent auf welchem Agentensystem sich die einzelnen WorkerCrack-Agenten befinden, damit der Benutzer einen Überblick darüber hat, wo die einzelnen WorkerCrack-Agenten arbeiten. Dazu teilt ein WorkerCrack-Agent seinem MasterCrack-Agenten sein neues Agentensystem mit, wenn der WorkerCrack-Agent migiriert.
Da sich im voraus nicht herausfinden lässt, ob ein System fähig ist Crack auszuführen, besteht nur die Möglichkeit, dass der WorkerCrack-Agent versucht Crack auszuführen und falls dies nicht erfolgreich ist, es auf einem anderen System zu versuchen. Damit nicht jeder WorkerCrack-Agent selbst alle System versuchen muss, meldet jeder WorkerCrack-Agent falls er auf seinem System Crack nicht ausführen könnte. Der MasterCrack-Agent speichert dies und gibt dem WorkerCrack-Agenten den Namen eines Agentensystem, dass bisher noch nicht auf diese Weise gemeldet wurde. In diesem Fall arbeiten dann mehrere WorkerCrack-Agenten auf dem selben Agentensystem

4.5 Schnittstelle des MasterCrack

Da die Schnittstellen der MASA-Agenten in idl definiert werden, wird hier ebenfalls die idl Schnittstelle zur Erklärung benutzt.

Für die WorkerCrack-Agenten werden die folgende Methoden zur Verfügung gestellt:

- `boolean report_result(in long worker_number, in string result);` Durch Aufruf dieser Methode teilt der aufrufende WorkerCrack-Agent seinem MasterCrack-Agenten das Ergebnis seiner fertig bearbeitetenden Teilaufgabe mit. Der erste Parameter `worker_number` ist die Nummer des aufrufenden WorkerCrack-Agenten. Diese Nummer wurde dem WorkerCrack-Agenten zusammen mit seiner Teilaufgabe zugeteilt. Der zweite Parameter `result` beinhaltet das Ergebnis der Crackinstanz als String. Der zurückgegebene boolsche Wert gibt an ob der MasterCrack-Agent das

Ergebnis erfolgreich gespeichert hat. Eine nicht erfolgreiche Speicherung kann dann auftreten, wenn die `worker-number` nicht korrekt ist, oder ein WorkerCrack-Agent unter derselben Nummer bereits ein Ergebnis berichtet hat.

- `boolean report_new_Location(in long workernumber,in CfMAF::Name newAS);` Mit dieser Methode teilt ein WorkerCrack-Agent seinem MasterCrack-Agenten mit, dass er nun auf einem neuen Agentensystem arbeitet. Der Parameter `worker_number` hat die oben beschriebene Semantik. Der zweite Parameter `newAS` ist der Name des Agentensystems auf dem der WorkerCrack-Agent nun arbeitet. Der zurückgegeben boolsche Wert gibt an, ob die Speicherung des neuen Agenstensystems erfolgreich war.
- `CfMAF::Name report_AS_as_not_workable_and_get_new_AS(in CfMAF::Name notWorkableAS);` Diese Methode realisiert das Melden eines Agentensystems, dass Crack nicht ausführen kann, und die Zurückgabe des Namens eines bisher nicht gemeldeten Agentensystem: Der Parameter `notWorkableAS` ist der Name des Agentensystem, auf dem der WorkerCrack-Agent Crack nicht ausführen konnte und zurückgegebene Wert ist der Name eines Agentensystem, dass noch nicht durch Aufruf dieser Methode gemeldet wurde und daher Crack vielleicht ausführen kann.
- Die Zwischenergebnisse werden dem MasterCrack-Agenten durch Aufruf der Methode `boolean report_intermediate_state(in long workernumber,in string intermediateReport, in string progress)` mitgeteilt. Der Parameter `intermediateReport` ist der Zwischenbericht des aufrufenden WorkerCrack-Agenten in der selben Form wie der Endbericht. Der Parameter `progress` ist die Anzeige des Fortschritts des WorkerCrack-Agenten in der selben Form wie sie von der Methode `string get_Progress()` WorkerCrack-Agenten geliefert wird.

Die folgenden Methoden sind für das Applet des MasterCrack-Agenten:

- Für die Anzeige der End- und Zwischenberichte existieren zwei Methoden, die sich in der Aktualität der zurückgelieferten Berichte unterscheidet. Beiden ist gemein, dass die Ergebnisse der WorkerCrack-Agenten konkatenieren, dabei Zwischenberichte mit einem Hinweis versehen, dass der betreffenden Agent noch nicht fertig ist, und es als String zurück geben. Bei `string get_results()` werden bei allen noch nicht beendeten WorkerCrack-Agenten die aktuellen Zwischenberichte abgefragt. `string get_intermediate_reports()` verwendet bei noch nicht beendeten WorkerCrack-Agenten die gespeicherten Zwischenberichte.
- Analog existieren für die Anzeige der Fortschritte und der Aufenthaltsorte der WorkerCrack-Agenten ebenfalls zwei Methoden. Beide generieren für jeden WorkerCrack-Agenten eine Zeile bestehend aus seinem Namen, seinem Fortschritt und dem Namen des Agentensystems auf dem er sich befindet. Das Format des Fortschritt ist das selbe wie bei der Methode `string get_Progress()` des WorkerCrack-Agenten Bei `string get_Worker_Status()`werden bei allen noch nicht beendeten WorkerCrack-Agenten der aktuelle Fortschritt abgefragt. `string get_Worker_intermediate_Status()` verwendet die gespeicherten Fortschritte.
- Falls der MasterCrack-Agent seine Aufgabe nicht erfüllen kann, muss der Benutzer informiert werden, woran es lag. Die geschieht über die Methode: `string getError();` Der zurückgegebene String soll den Fehlerstatus des MasterCrack-Agenten in Textform darstellen.
- Die Methode `string getcreatedWorkersNumber()` gibt die Anzahl der erzeugten WorkerCrack-Agenten in Textform an.

- `string getfinishedWorkerNumber()` liefert die Anzahl der WorkerCrack-Agenten, die mit der Bearbeitung ihrer Teilaufgabe fertig sind, als Text.
- `string getPasswordFileName()` liefert bei gestartetem Agent den Namen der Passwortdatei an die bearbeitet wird zurück, ist der Agent noch nicht gestartet wird der default-Wert aus den Agenten-Properties zurückgeliefert.
- `string getCrackDir()` liefert bei gestartetem Agenten den Namen des Verzeichnisses der Crackinstallation aus der die Wörterbücher verwendet wurden und bei noch nicht gestartetem den default-Wert aus den Agenten-Properties zurück.
- Ferner stellt der MasterCrack-Agent noch die Möglichkeit zur Verfügung per Applet gestartet zu werden: `:boolean start(in string inCrackDirName, in string inPasswordfileName);` Der erste Parameter CrackDirName gibt das Verzeichnis des Cracks an, aus dem der MasterCrack-Agent die Wörterbücher nehmen soll. Der zweite Parameter PasswordfileName gibt den Namen der zu bearbeitenden Passworddatei an.

4.6 Schnittstelle des WorkerCrack

Für den MasterCrack-Agenten werden vom WorkerCrack-Agenten folgende vier Methoden zur Verfügung gestellt:

- Wichtigste Methode hierbei ist natürlich die, mit der der MasterCrack-Agent dem WorkerCrack-Agenten seine Teilaufgabe mitteilt:

```
boolean receive_work_portion(
in CfMAF::Name MasterName,
in long myNumber,
in ContentOfFile passwordfile,
in WorkPortion dictionaries1,
in WorkPortion dictionaries2,
in WorkPortion dictionaries3,
in NameOfFiles namesOfDictionaries1,
in NameOfFiles namesOfDictionaries2,
in NameOfFiles namesOfDictionaries3 );
```

  Der erste Parameter MasterName gibt den Namen des MasterCrack-Agenten an, der diese Teilaufgabe verteilt hat und an den der Bericht zu liefern ist.
  Der zweite Parameter myNumber gibt die Nummer an, die der WorkerCrack-Agent angeben soll, wenn er den Bericht abliefert oder migriert.
  Der dritte Parameter passwordfile ist der Inhalt der Passwortdatei als eindimensionales Bytearray. Die Typangabe ContentOfFile ist dabei nur ein Alias für ein eindimensionales Bytearry.
  Die vierten bis sechsten Parameter dictionaries1 bis dictionaries3 sind die Wörterbücher. Diese werden als zeidimensionale Bytearray übergeben. Wobei eindimensionalen Bytearray auf die vom ersten Index verwiesen sind, der Inhalt der einzelnen Wörterbücher sind. Die Übergabe als 3 Parameter wurde gewählt um die Einteilung der Wörterbücher in den Verzeichnissen dict/1 bis dict/3 beizubehalten.
  Die siebten bis neunten Parameter namesOfDictionaries1 bis namesOfDictionaries3 geben die Namen der Wörterbücher an. NameOfFiles ist hierbei nur ein Alias für ein Stringarray. Die Zuteilung der Namen zu den entsprechenden Inhalten findet über den ersten Index statt.
- `boolean start();` Diese Methode stellt das Signal zum Starten der Bearbeitung dar, falls der WorkerCrack-Agent auf dem selben Agentsystem wie sein MasterCrack-

Agent arbeiten soll: Der zurückgelieferte boolsche Wert gibt an, ob der WorkerCrack-Agent starten konnte.

- Um die Abfrage der Zwischenstände zu realisieren stellt der WorkerCrack-Agent die Methode `string getReport()` zur Verfügung: Hierbei liefert der WorkerCrack-Agent die von vom Reporter-shellscript gelieferte Ausgabe als String zurück. Diese Methode wird auf vom Applet des WorkerCrack-Agenten benutzt, um den Stand des Agenten anzuzeigen.
- Den Fortschritt des WorkerCrack-Agenten liefert die Methode `string get_Progress();` Hierbei wird ein String der Form "currentRule: g Guessed: n/m" zurückgegeben, wobei g die Nummer der Regel ist die Crack gerade bearbeitet, n die Anzahl der Benutzer ist dessen Passwort erraten wurde und m die Anzahl der Benutzer der von diesem WorkerCrack-Agenten zu bearbeitetenden Passwortdatei ist.

Diese Methode ist für das Applet des WorkerCrack-Agenten:

- `string getError();` liefert den Fehlerzustand des WorkerCrack-Agenten an, damit es vom Applet angezeigt werden kann.

5 Der MasterCrack Agent

Wie bereits im Design Kapitel vorgestellt, hat der MasterCrack-Agent folgende Aufgaben;

- Einlesen der Passwortdatei und der zu verwendeden Wörterbücher
- Bilden der Teilaufgaben (dabei Entscheidung über die Strategie)
- Erzeugen der WorkerCrack-Agenten und Übergabe ihrer Teilaufgabe
- Sammeln der Ergebnisse der WorkerCrack-Agenten
- Verwalten der Aufenthaltsorte der erzeugten WorkerCrack-Agenten
- Verwalten der Agentensystem, die Crack nicht ausführen können
- Präsentation der Ergebnisse

5.1 Einlesen der Passwortdatei und der Wörterbücher

Da die Verteilung der Arbeit durch die Verteilung der Benutzer in der Passwortdatei oder der Verteilung der Wörterbücher realisiert werden, müssen Passwortdatei und Wörterbücher als erstes vom Dateisystem eingelesen werden.

Den Namen der Passwortdatei und den Pfad der Wörterbücher können dem MasterCrack-Agenten auf drei Wegen mitgeteilt werden. Die Auswahl welchen Weg der Agent benutzen soll, erfolgt über die Wahl des Konstruktors beim Erzeugen des Agenten.

- Die erste Möglichkeit ist die Angabe über die Properties des MasterCrack-Agenten. Der Agent liest nach dem Start die Properties ein, überprüft ob die angegeben Daten stimmen und liest die angegebene Passwordatei und die Wörterbücher der angegebenen Crackinstallation ein. Ist die Überprüfung nicht erfolgreich, kann man dem Agenten mit der dritte Möglichkeit korrigierte Angaben übergeben.
- Die zweite Möglichkeit ist Angabe über die Parameter des Konstruktors. Der Agent verwendet die Werte der Parametern des Konstruktors als Angaben des Namens der Passwortdatei und des Pfads der zu verwendeden Crackinstalltion. Diese werden wie bei der ersten Möglichkeit überprüft und bei erfolgreicher Überprüfung eingelesen. Ist diese Überprüfung erfolglos, kann ebenfalls über die dritte Möglichkeit diese Angaben korrigiert werden.
- Die dritte Möglichkeit ist die Angabe über das Applet des Agenten. Der Benutzer trägt die Angabe in das Applet des Agenten ein und sendet diese per Knopfdruck an den Agenten. Bei Empfang überprüft der Agent die Angaben und liest bei Erfolg Passwortdatei und Wörterbücher ein. Ist die Überprüfung erfolglos, wartet der Agent auf einen weiteren Aufruf mit Angaben.

Dementsprechend wurden drei Konstruktoren implementiert:

- Der erste Konstruktor besitzt keine Parameter und realisert die erste Möglichkeit.
- Der zweite Konstruktor besitzt einen Parameter vom Typ String, namens autostart. Dieser Parameter gibt an, ob der Agent nach dem Erzugen sofort mit seiner Arbeit beginnen soll. Erhält der Parameter den Wert „true“ so beginnt der Agent nach seiner Erzeugnung mit der Arbeit und zeigt damit das selbe Verhalten, wie der parameterlose Konstruktor. Erhält der Parameter den Wert „false“, so wartet der Agent auf den Aufruf der Methode start durch sein Applet, bevor er mit der Arbeit beginnt. Somit erhält man auf diese Weise Möglichkeit Drei.

- Der dritte Parameter besitzt drei Parameter vom Typ String. Der erste Parameter besitzt die selbe Semantik, wie der Parameter des zweiten Konstruktors. Die beiden anderen Parameter erhalten den Namen der Passwortdatei und den Pfad der Crackinstallation. Wählt man bei autostart „true“, so erhält man Möglichkeit Zwei.

Das Einlesen selbst geschieht über die Methoden des Javapackages „Java.io“. Die Passwortdatei wird dann in einem einfachen bytearray Attribut gespeichert. Die Wörterbücher werden in insgesamt 6 Attributen des Agenten gespeichert. Pro Verzeichnis werden zwei Attribute verwendet. Das erste ist ein zweidimensionales Bytearray. Der Inhalt eines Wörterbuchs wird als einfaches bytearray gespeichert und die Wörterbücher eines Verzeichnisses werden zum zweidimensionalen array zusammengefasst. Das jeweilige zweite Attrbut ist ein Stringarray, dass die zu den Inhalten der Wörterbücher gehörenden Namen des Wörterbuches speichert.

5.2 Bilden der Teilaufgaben

Um zu entscheiden nach welcher Strategie verteilt wird, braucht der Agent folgende Informationen:

- Anzahl der verfügbaren Agentensysteme
- Anzahl der Benutzter in der Passwortdatei
- Anzahl der Wörterbücher

Die Anzahl der Agentensysteme fällt bei der Suche nach verfügbaren Agentensystemen an. Durch den Aufruf der entsprechenden Methoden erstellt der MasterCrack-Agent ein Array mit den IORs der Agentensysteme in der MASA-Region. Diese IORs benutzt er um von jedem Agentensystem den Namen zu erfragen. Diese Namen speichert der Agent in einem Array. Dieses Array dient dann später dazu die WorkerCrack-Agenten zu verteilen, sowie festzuhalten auf welchen Systemen Crack nicht ausführbar ist.
Die Anzahl der Benutzer in der Passwortdatei, stellt der Agent fest indem er die Anzahl der Zeilen in der Passwortdatei feststellt, da jeder Benutzer genau eine Zeile in der Passwortdatei einnimmt.
Die Anzahl der Wörterbücher erhält der Agent durch Addieren der Arraylängen der drei Attribute, die die Namen der Wörterbücher speichern.

Nun entscheidet der Agent über die Verteilung. Sind mehr Agentensysteme als Benutzer vorhanden, werden die Teilaufgaben durch Verteilen der Benutzer gebildet. Das heißt eine Teilaufgabe besteht aus den gesamten Wörterbüchern und einem Teil der Benutzer. Andernfalls werden die Anzahl der Benutzer und die Anzahl der Wörterbücher verglichen. Sind mehr Benutzer vorhanden werden diese ebenfalls verteilt, aber es werden dann nur soviele Teilaufgaben wie vorhandene Benutzer gebildet. Ansonsten wird eine Teilaufgabe aus der gesamten Passwortdatei und einem Teil der Wörterbücher gebildet. Dabei wird die Einteilung der Wörterbücher entsprechend der Verzeichnisse nicht aufrecht erhalten. Sind genügend Wörterbücher vorhanden, werden soviele Teilaufgaben gebildet wie Agentensysteme. Ansonsten werden soviele Teilaufgaben wie möglich gebildet.
Die gebildeten Teilaufgaben werden in den Attributen des Agenten gespeichert. Für die Passwortdatei ist dies ein zweidimensionales Bytearray, wobei der erste Index der Numer der Teilaufgabe entspricht und die bytearray auf die darin verwiesen wird der Inhalt der aufgeteilten Passwortdatei ist. Für den Inhalt der Wörterbücher sind dies entsprechend drei dreidimensionale Bytearrays. Der erste Index ist die Nummer der Teilaufgabe, der zweite die

Nummer des Wörterbuchs und die dritten ergeben den Inhalt des Wörterbuches. Für die Namen der Wörterbücher ist dieses Attribut ein zweidimensionales Stringarray, der erste Index ist wie gehabt die Nummer der Teilaufgabe, der zweite die Nummer des Wörterbuches.

5.3 Erzeugen der WorkerCrack-Agenten und Übergabe ihrer Teilaufgabe

Mit Hilfe der vom Agentensystem zur Verfügung gestellten Methoden werden soviele WorkerCrack-Agent erzeugt wie Teilaufgaben zur Verfügung stehen. Über die WorkerCrack Methode `receive_work_Teilaufgabe` werden den erzeugten WorkerCrack-Agenten ihre Teilaufgabe übergeben. Anschliessend werden die WorkerCrack-Agenten auf eines der gefundenen Agentensysteme migriert, wo sie dann ihre Teilaufgabe bearbeiten. Da einer der erzeugten WorkerCrack-Agenten auf dem selben Agentensystem wie der MasterCrack-Agent arbeiten soll, muss dieser WorkerCrack-Agenten mit einem Signal gestartet werden. Dies geschieht durch den Aufruf der Methode `start` dieses Agenten.

5.4 Sammeln der Ergebnisse der WorkerCrack-Agenten

Hat einer WorkerCrack-Agent seine Teilaufgabe fertig bearbeitet, teilt er seinem MasterCrack-Agenten das Ergebnis mit, indem er die Methode `report_result` aufruft. Dabei werden die Nummer der Protion sowie das Ergebnis in Form eines Strings übergeben. Die gesamten Ergebnisse speichert der MasterCrack-Agent in einem Attribut vom Typ Stringarray. Berichtet ein WorkerCrack-Agent das Ergebnis der Bearbeitung seiner Teilaufgabe, so speichert der MasterCrack-Agent den übergebenen String einfach unter dem ebenfalls übergebenen Index in diesem Attribut ab. Ferner wird in einem boolsches array diese Teilaufgabe als bearbeitet markiert.
Ferner speichert der MasterCrack-Agent die Zwischenberichte und Fortschrittsangaben, die die WorkerCrack-Ageneten in regelmässigen Abständen melden. Hierzu rufen die WorkerCrack-Agenten die Methode `report_intermediate_state(in long workernumber,in string intermediateReport, in string progress)` auf. Der MasterCrack-Agent speichert dann die Zwischenberichte und Fortschritte in zwei Array an dem durch `workernumber` gegebenen Index.

5.5 Verwaltung der Aufenthaltsorte der erzeugten WorkerCrack-Agenten

Die Speicherung der Aufenthaltsorte der erzeugtenWorkerCrack-Agenten erfolgt in einem Attribut vom Typ Namearray. Dieses wird beim Erzeugen der WorkerCrack-Agenten mit den Namen der Agentensystem intialisiert, auf die der MasterCrack-Agent die erzeugten WorkerCrack-Agenten migriert. Migriert einer WorkerCrack-Agenten, so ruft er auf dem neuen Agentensystem die Methode `report_new_Location` seines MasterCrack-Agenten mit seiner Protionsnummer und dem Namen des Agentensystems, auf dem er sich nun befindet, auf. Der MasterCrack-Agent speichert dann unter dem gegeben Index den Namen des neuen Agentensystems.

5.6 Verwaltung der Agentensysteme, die Crack nicht ausführen können

Für diese Aufgabe benötigt der MasterCrack-Agent drei Attribute. Zum einen ein Array mit den Namen der gefundenen Agentensysteme, wobei dieses Array auch benutzt wird um die erzeugten WorkerCrack-Agenten auf die Agentensystem zu verteilen, und ein Array von boolschen Werten. Dabei hat der Wert „false“ die Bedeutung, dass auf dem Agentensystem, dessen Namen im selben Index des ersten Attributs steht, Crack nicht ausgeführt werden kann. Umgekehrt bedeutet ein Wert von „true“ nicht, dass das Agentensystem Crack ausführen kann, sondern nur, dass bisher kein WorkerCrack-Agent berichtet hat, dass dieses Agentsystem Crack nicht ausführen kann.
Teilt ein WorkerCrack-Agent dem MasterCrack-Agent durch Aufruf der Methode `report_AS_as_not_workable_and_get_new_AS` mit, dass er Crack auf diesem Agentensystem nicht ausführen kann, so sucht der MasterCrack-Agent nach dem Index des Agentensystems in dem ersten Attribut und setzt dann im Booleanarray an diesem Index den Wert auf false.
Um dann nicht allen WorkerCrack-Agenten die diese Methode aufrufen nicht das selbe Agentensystem zurückliefern, dass Crack vielleicht ausführen kann, besitzt der MasterCrack-Agent ein drittes Attribut vom Typ Integer, welches mit 0 initalisiert wird. Beginnend beim Wert dieses Attributs sucht der MasterCrack-Agent im boolschen Array nach einem Index dessen Wert „true“ ist. Erreicht er das Ende des Array fängt er von vorne an. Er liefert dann den Wert dieses gefundenen Index des ersten Attributs zurück. Das dritte Attribut erhält dann diesen um eins erhöhten Wert. Auf diese Weise wird beim nächsten Mal die Suche nach dem diesmal zurückgegeben Agentensystem begonnen. So wird eine einigermassen gleichmässige Verteiltung der WorkerCrack-Agenten erreicht. Findet er keinen solchen Index, da sämtliche Werte des Booleanarray „false“ sind, beudeutet dies, dass kein gefundenes Agentensystem fähig ist Crack auszuführen. In diesem Fall liefert der MasterCrack-Agent einen leeren Namen zurück um dem WorkerCrack-Agenten zu signalisieren, dass es kein Agentensystem gibt, dass Crack ausführen kann.

5.7 Präsentation der Ergebnisse

Die Präsentation der Ergebnisse, sowie des aktuellen Status geschieht über das Applet des MasterCrack-Agenten. Das Applet zeigt dies über 2 Textfelder an. In dem erstem Textfeld wird angezeigt auf welchen Agentensystemen die erzeugten WorkerCrack-Agenten sind, sowie eine Anzeige wieviel Passwörter die einzelnen WorkerCrack-Agenten erraten haben. Das zweite Textfeld zeigt die Berichte der WorkerCrack-Agenten an. Ferner zeigt das Applet an wieviele WorkerCrack-Agenten erzeugt wurden und wieviele bereits ihre Teilaufgabe bearbeitet haben. Alle diese Felder des Applet werden von den zurückgegebenen Werten der entsprechenden Methode des MasterCrack-Agenten gefüllt.
Zur Anzeige der erzeugten WorkerCrack-Agenten über `getcreatedWorkersNumber` wird einfach das Attribut zurückgegeben, in der die Anzahl der erzeugten Teilaufgaben gespeichert wird.
Über `getfinishedWorkerNumber` wird angeben wieviel WorkerCrack-Agenten bereits ihre Teilaufgabe bearbeitet haben, dafür wird im entsprechen Attribut gezählt, wieviel Teilaufgaben als bearbeitet markiert wurden.

`get_Worker_Status` bzw. `get_Worker_intermediate_Status` liefert die Aufenthaltsorte der erzeugten WorkerCrack-Agenten sowie eine Schätzung ihres Fortschritts. Dazu wird für jeden WorkerCrack-Agenten eine Zeiler erzeugt. Diese besteht aus dem Namen des WorkerCrack-Agenten als Text, den Namen des Agentensystems als Text, diesen erhält der MasterCrack-Agent aus seinem Attribut, sowie dem String den der jeweilige Agent beim Aufruf seiner Methode `get_Progress` zurückliefert. Bei `get_Worker_Status` alle noch nicht beendeten WorkerCrack-Agenten neu abgefragt. Diese Methode wird beim Start des Applets und bei Drücken des Refresh-Buttons verwendet. Bei `get_Worker_intermediate_Status` werden die gespeicherten Fortschrittsberichte verwendet, sie wird verwendet wenn der MasterCrack-Agent das Applet über ein Event informiert, dass ein WorkerCrack-Agent einen neuen Zwischenbericht geliefert hat.
Die gesammelten Berichte liefern die Methoden `get_results` und `get_intermediate_reports`. Hierbei werden die gesammelten Berichte der bereits beendeten WorkerCrack-Agenten aneinandergehängt. Bei den noch nicht beendeten WorkerCrack-Agenten wird bei `get_results` durch den Aufruf der Methode `getReport` ein neuer Zwischenbericht eingeholt und ebenfalls angehängt, bei `get_intermediate_reports` werden die gespeicherten Zwischenberichte verwendet. Die Auswahl welche Methode aufgerufen wird, ist die selbe wie bei der Statusanzeige.

6 Der WorkerCrack Agent

6.1 Parallelisierung von Crack

Aufgabe des WorkerCrack-Agenten ist die Bearbeitung der vom MasterCrack-Agenten zugewiesenen Teilaufgabe mit Hilfe von Crack. Dazu muss der WorkerCrack-Agent eine Crackinstanz so erzeugen, dass sie seine Teilaufgabe bearbeitet und die anderen WorkerCrack-Agenten nicht stört. Wie im Kapitel über Crack beschrieben, legt Crack bei Start im Unterverzeichnis run einige Dateien an, die es während der Laufzeit benutzt. Zwar sind die Dateien im run Verzeichnis selbst mit der Prozessnummer gekennzeichnet, aber es wäre möglich dass ein anderer WorkerCrack-Agent auf einem anderen Agentensystem, dass sich das Dateisystem mit aktuellen Agentensystem teilt, die selbe Prozessnummer erhält. Ferner sind die Dateien im Unterverzeichnis run/dict in dem die komprimierten Wörterbücher sowie die aus der Passwortdatei genererierten Wörterbücher gespeichert sind nicht mit einer Prozessnummer gekennzeichnet. Es ist also nicht möglich Crack einfach mit der Passwortdateiportion aufzurufen. Ausserdem lässt sich auf diese Weise die Verteilung der Wörterbücher nicht realisieren, da die Wörterbücher aus dem Unterverzeichnis dict im run-Verzeichnis zusammenfasst und komprimiert. Bei einer Verteilung nach Wörterbüchern würde so der zeitlich letzte Agent stets die Wörterbücher der vor ihm gestarteten Agenten überschreiben.
Daher muss Crack dazu gebracht werden ein bestimmtes run Verzeichnis zu benutzen, sowie ein bestimmtes Verzeichnis für die Wörterbücher. Da sich diese Verzeichnisse nicht konfigurieren lassen, bleibt nur ein Kopie von Crack in ein nur für einen WorkerCrack-Agenten bestimmtes Verzeichnis zu installieren. Dazu reicht es ein lauffähig installiertes Crack zu kopieren. Das Crack Shellscript compiliert und installiert Crack falls nötig. Um Platz zu sparen wurden die Verzeichnisse src und run/bin nur verlinkt. Dadurch wird Crack nur neu compiliert, falls es von einer Architektur gestartet wurde, von der es bisher noch nie gestartet wurde.

6.2 Erzeugen der Crack-Instanz

Der WorkerCrack-Agent beginnt mit der Bearbeitung, falls eine der folgenden Begebenheiten eintritt:

- Der Agent wurde migriert und hat seine Teilaufgabe erhalten
- Der Agent hat seine Teilaufgabe erhalten und seine Methode `start()` wurde aufgerufen

Als erstes muss der WorkerCrack-Agent eine Crackinstallation finden. Dazu gibt der Benutzer in den Properties des Agenten eine Reihe von möglichen Pfade an, in denen Crack installiert sein kann. Die Angabe eines Pfades reicht dabei nicht aus, da möglicherweise nicht alle Systeme das selbe Dateisystem benutzen und daher Crack auf verschiedenen Systemen an verschiedenen Orten installiert sein kann.
Hat der Agent ein Verzeichnis gefunden, in dem Crack installiert ist, so versucht er in diesem Verzeichnis ein Unterverzeichnis der Form crackRunN anzulegen, wobei N eine natürliche Zahl ist. Dabei beginnt er mit N=0, das Überprüfen ob das Verzeichnis existiert und das Anlegen muss dabei natürlich atomar geschehen, damit nicht ein anderer WorkerCrack-Agent in der Zeit zwischen Existenzprüfung und Anlegen ebenfalls zum Schluss kommt, er könne

ein Verzeichnis mit dem selben Namen verwenden. Dies wird erreicht indem mit der Methode mkdir() der Klasse File versucht wird das Verzeichnis anzulegen. Diese Methode garantiert Atomizität und liefert nur „true“ zurück, wenn das Verzeichnis wirklich angelegt wurde. Existiert bereits ein Verzeichnis dieses Namen liefert die Methode „false“ zurück.
Danach kopiert der WorkerCrack Agent das installierte Crack bis auf die Wörterbücher und das run- und src-Verzeichnis in dieses angelegte Verzeichnis. Dann wird ein Link auf das src-Verzeichnis des installierten Crack erzeugt. Das run Verzeichnis wird ebenfalls erstellt und in diesem wird ein Link auf das run/bin-Verzeichnis des installierten Crack erzeugt.
Dazu wird ein systemabhängiges Kopier- und Linkprogramm benutzt dessen Namen ebenfalls über die Properties angegeben wird. Da Crack nur auf Unixsystemen läuft, wurde hier auf Plattformunabhängigkeit verzichtet. Der WorkerCrack-Agent benutzt ausserdem die systemabängigen Löschprogramme sowie das Compilierhilfprogramm Make. Für das Ausführen von Crack benötigt der WorkerCrack-Agent ebenfalls den Namen einer Shell. Alle diese Angaben werden dem Agenten über das Propertyfile gemacht.
In dieses Verzeichnis werden dann die Wörterbücher an die entsprechende Stelle geschrieben. Diese Wörterbücher wurden dem Agenten vom MasterCrack-Agenten über den Aufruf der Methode `receive_work_Teilaufgabe` übergeben und den Attributen des Agenten gespeichert. Die Passwortdatei wird ebenfalls in das Verzeichnis geschrieben.
Anschliessend überprüft der Agent ob er Crack bereits einmal gestartet hat und daher dafür sorgen muss, dass Crack nicht nochmals von vorne anfängt. Dieser Fall wird später besprochen. Bei einem Neustart führt der Agent mit Hilfe der Shell das Crackshellscript mit der vom Agenten geschriebenen Passwortdatei aus. Dabei wird Crack mit Option –fgnd gestartet, diese Option sorgt dafür, dass Crack im Vordergrund ausgeführt. Auf diese Weise beendet sich die Shell erst, wenn Crack seine Arbeit beendet hat. Somit kann der Agent feststellen, wann Crack mit der Bearbeitung fertig ist. Allerdings läuft dann die Ausgabe des mkcracker Programms statt in eine Datei im run Unterverzeichnis über die Standartausgabe. Das heißt das Schreiben in diese Datei muss der Agent selbst übernehmen.
Stellt der Agent fest, dass sich die Shell erfolgreich beendet hat, bedeutet dies, dass Crack mit der Bearbeitung fertig ist. Der Agent erstellt dann den Bericht und meldet dies seinem MasterCrack-Agenten

6.3 Erstellung des Berichtes

Für die Erstellung des Berichtes sowohl des Endberichtes als auch der Zwischenberichte wird das Shellscript Reporter benutzt. Dieses Shellscript filtert aus der Ausgabedatei des mkcracker Programms die Daten für den Bericht heraus. Daher muss der Agent auch die Ausgabe von mkcracker in eine Datei schreiben, die das Reporter Shellscript bearbeitet. Das Reporterscript bearbeitet dabei alle Dateien im Unterverzeichnis run, die mit einem grossen D beginnen. Der Agent schreibt die Ausgabe von mkcracker daher in eine Datei namens Dmasa ins Unterverzeichnis run. Allerdings ist das Reportershellscript im Gegensatz zum Crackshellscript nur korrekt lauffähig, wenn es im Crackverzeichnis aufgerufen wird. Daher muss der Agent ein Shellscript erzeugen, dass erst in das Verzeichnis, in dem der Agent Crack ausführt, wechselt und dann das Reporter-Shellscript aufruft.

6.4 Migration

Muss der WorkerCrack-Agent migieren, weil er Crack nicht ausführhren kann, so meldet er sein augenblickliches Agentensystem seinem MasterCrack-Agenten und erhält dann den Namen eines neuen Agentensystems. Erhält er dabei einen leeren Namen, so schickt seinem MasterCrack-Agenten als Bericht einen String mit dem Inhalt, dass kein System gefunden wurde, dass Crack ausführen konnte, sowie die Fehlermeldungen.
Damit Crack bei einer Migration, während eines laufenden Cracks, nicht jedesmal neu starten muss, muss der Agent den Status von Crack in einer Form speichern. Dies geschieht indem der Agent bei einer Migration die Dateien in den Unterverzeichnissen run und run/dict in seinen Attributen speichert. Auf dem neuen Agentensystem schreibt er dieses Dateien aus den Attributen wieder in diese Verzeichnisse. Anschliessend benennt er die Datei Dmasa, die die Ausgabe der vorherigen Ausführung von Crack beinhaltet in resume um. Anschliessen wird Crack mit der Option –recover und der Datei resume statt der Passwortdatei gestartet. Crack parst dann die Ausgabe in der Datei resume und kann so die Bearbeitung der Teilaufgabe fortsetzen, statt neu anzufangen.

6.5 Feststellen und Berichten des Fortschritts

Wie bereits in der Schnittstellenbeschreibung beschrieben, soll die Methode `get_Progress` einen Fortschritt als String zurückgeben. Dies geschieht in der Form „current Rule: <aktuelle Regelnummer> Guessed: <erratene Benutzer> / <gesamte Benutzer>“.
Die Anzahl der zu bearbeitenden Benutzer stellt der WorkerCrack-Agent auf dieselbe Weise fest, wie der MasterCrack-Agent, indem er die Zeilen in der Passwortdatei zählt. Die Anzahl der gecrackten Benutzer stellt der Agent fest, indem er die Ausgabe des Reporter-Shellscripts parst. Für jeden erratenen Benutzer erzeugt dieses Shellscript eine Zeile die mit „Guessed“ beginnt. Durch zählen dieser Zeilen stellt der Agent fest, wie viele Passwörter erraten wurden.
Die Nummer der Regel die Crack gerade bearbeitet wird durch das Parsen der Crack Ausgabe festgestellt. Crack gibt wenn es eine neue Regel bearbeitet eine Zeile der Form O:<Zahl>:<Nummer der Regel> aus. Wird eine solche Zeile erkannt, wird die Nummer der Regel in einem Attribut des Agenten gespeichert.
Ein seperater Thread berichtet diesen Fortschritt und einen aktuellen Zwischenbericht in regelmässigen Abständen an den MasterCrack-Agenten durch Aufruf der Methode `report_intermediate_state` des MasterCrack-Agenten.

7 Leistungsbetrachtung

Wie im Abschnitt über die gewählte Parallelisierung im Design Kapitel (4.1) erwähnt, hat die gewählte Parallelisierung durch Verteilung der Benutzer bzw. der Wörterbücher eine gute parallele Effizienz. Durch das salt bei der `crypt`-Funktion muss jeder Versuch für jeden Benutzer neu verschlüsselt werden, dadurch wird bei der Parallelisierung bei der Verschlüsselung keine Arbeit von mehreren Instanzen geleistet, die bei einer nicht-parallelisierten Instanz nur einmal geleistet würde. Bei der Verteilung nach Benutzern wird lediglich die Generierung der Versuche aus den Wörterbüchern mehrfach geleistet, diese Arbeit ist aber im Vergleich zum Verschlüsseln äusserst gering. Bei der Verteilung nach Wörterbüchern werden allerdings die aus der Passwortdatei generierten Wörterbücher von jeder Instanz bearbeitet, deshalb wird diese Verteilung nur verwendet, wenn sich aus der Passwortdatei weniger Teilaufgaben bilden lassen als Systeme vorhanden sind.

Der zusätzliche Aufwand der betrieben werden muss um Crack zu parallelisieren ist nicht sehr gross und muss nur beim Starten geleistet werden. Dieser Mehraufwand findet im MasterCrack-Agenten statt. Es ist die Suche nach Agentensystemen, das Einlesen der Wörterbücher und der Passwortdatei, das Bilden der Teilaufgaben, das Erstellen der WorkerCrack-Agenten und das Verteilen der Teilaufgaben an die erzeugten WorkerCrack-Agenten. Hinzu kommt die Zeit für die Migration der WorkerCrack-Agenten auf ihr Ziel-Agentensystem. Bis auf das Einlesen der Wörterbücher und der Passwortdatei, ist der Mehraufwand proportional zur Anzahl der verfügbaren Agentensysteme. Allerdings ist er nur in sehr geringem Masse von der Arbeitslast, sprich der Anzahl der Benutzer der Passwortdatei und des Umfangs der Wörterbücher abhängig.
Ein zusätzlicher konstanter Mehraufwand fällt bei den WorkerCrack-Agenten an. Dies ist der Aufwand um die Crackinstallation zu kopieren und eventuell neu zu compilieren. Dieser Aufwand hängt dabei weder von der Arbeitslast der zugeteilten Teilaufgabe, noch der Anzahl der gebildeten Teilaufgaben ab.

Als nächster Aspekt wird die Ausnutzung der vorhandenen Rechenkapazität der zur Verfügung stehenden Systeme betrachtet. Hierbei ist nur die Verwaltung der Verteilung ausschlaggebend. Da diese statisch ist und immer gleich grosse Teilaufgaben erstellt, werden die vorhandenen Rechenkapazitäten nur bei einer absolut homogenen Verteilung der Rechenkapazitäten auf die Systeme voll ausgenutzt. Bei einer heterogenen Verteilung wird die Gesamtlaufzeit durch das langsamste System verzögert, während die restlichen Kapazitäten brach liegen. Dieser Effekt wird jedoch bei einer grossen Anzahl von System abgemildert, da der Anteil der Arbeit den das langsamste System leisten muss bei mehr Systemen immer kleiner wird, während der Leistungsunterschied zum schnellsten System gleich bleibt.

Bei einer stichprobenhaften Messung der Laufzeiten ergaben sich folgende Werte:
Getestet wurde mit einer Passwortdatei aus 11 Benutzern und 10 Wörterbüchern. Dabei standen drei Systeme zur Verfügung:

- pcheger3: Ein Pentium 3 (Katmai) mit 500 Mhz, 878 Mbyte RAM und Linux 2.4.14.
- pcheger14: Ein Pentium 3 (Katmai) mit 1000 Mhz, 878 Mbyte RAM und Linux 2.4.14.
- sunheger1: Eine Ultra Sparc 450 mit 1Gbyte RAM und SunOS 5.8.

Es wurde dabei folgendes gemessen:

- Auf allen Systemen wurde die Zeit gemessen, die Crack alleine für die Bearbeitung der gesamten Passwortdatei benötigt.
- Auf pcheger3 und pcheger14 wurde die Zeit gemessen, die die Agenten brauchen um jeweils die gesamte Passwortdatei zu bearbeiten.
- Die Zeit die pcheger3 und pcheger14 benötigen um die Passwortdatei zusammen zu bearbeiten.
- Die Zeit die alle drei Systeme benörtigen um die Passwortdatei zusammen zu bearbeiten.

Dabei ergaben sich folgende Zeiten (Die Messungen wurden dabei lediglich minutengenau durchgeführt):

	pcheger3	pcheger14	sunheger1
Crack, einzeln	32 min	11 min	4 h 48 min
Agenten, einzeln	33 min	12 min	-
Agenten, zusammen auf 2 Systemen	18 min	8 min	-
Agenten, zusammen auf allen Systemen	14 min	5 min	3 h 13 min

Hierbei sieht man, dass sich die besprochene mangelnde Fähigkeit die Rechenkapazitäten bei heterogener Verteilung bei wenigen Systemen und grossem Unterschied sehr stark auswirkt.
Auch wenn die gesamte Passwortdatei länger bearbeitet wird, als sie auf dem schnellesten System alleine gebraucht hätte, hat es doch Vorteile es auf mehreren Systemen bearbeiten zu lassen. So ist wenn das schnellste System fertig ist. ein Teil der Passwortdatei komplett bearbeitet, während bei der einzelnen Bearbeitung die gesamte Datei nur zum Teil bearbeitet wurde. Ausserdem ist das schnellere System im Vergleich zum Einzellauf früher wieder für andere Arbeiten frei.

8 Benutzeranleitung

8.1 Installation

Vor der Installations der Agenten müssen folgende Vorraussetzungen erfüllt sein:

- Es muss eine lauffähige Masainstallation vorhanden sein
- Es muss ein lauffähiges, installiertes Crack so vorhanden sein, dass von mindestens einem Agentensystem darauf zugegriffen werden kann

Vor dem Übersetzen der Quelltexte muss in der Datei Makefile.DEF `MASA_INSTALL_PATH` auf das installierte Masa gesetzt werden. Ferner muss, falls das Agenten Gegenstück nicht im selben Verzeichnis installiert wird, im Unterverzeichnis src/idl ein Link auf die IDL-Datei des jeweils anderen Agenten erzeugt werden. Das heißt am Beispiel des WorkerCrack-Agenten: Ist der MasterCrack-Agent nicht im Verzeichnis MasterCrack im darüberliegenden Verzeichnis installiert, muss im Verzeichnis src/idl ein Link namens MasterCrack.idl erzeugt werden, der auf die Datei MasterCrack.idl des MasterCrack-Agenten verweist. Ansonsten erzeugt das Makefile diesen Link automatisch.
Ist dies erfolgt können beide Agenten durch Eingabe von „make proddir“ und „make“ übersetzt werden.
Bevor beide Agenten jedoch endgültig installiert werden können, muss ihre Properties sowie ihre Policywishlist-Dateien konfiguriert werden.
Soll der MasterCrack-Agent fähig sein, ohne weitere Einstellungen sofort nach dem Erzeugen zu starten, so müssen in der Datei genPropertiesFile.MasterCrack im Unterzeichnis bin über die Properties die zu bearbeitende Passwortdatei, sowie der Pfad der Crackinstallation deren Wörterbücher benutzt werden sollen, angegeben werden.
Dies geschieht durch Setzen der Properties crackDir sowie passwordfile. Als Beispiel:

```
${MASA_PACKAGE_thisagent}.crackDir=/proj/fagent/users/thom/c50a/
${MASA_PACKAGE_thisagent}.passwordFile=/proj/fagent/users/thom/c50a/passtes
t
```

Ferner müssen dem MasterCrack-Agenten durch editierern der Datei genPolicyWishlist.MasterCrack im selben Verzeichnis, die Leserechte auf die Passwortdatei, sowie das gesamte Verzeichnis der Crackinstalltion inklusive deren Unterverzeichnis. Dies geschieht über die Filepermissions des Javapackages io. Als Beispiel:

```
permission java.io.FilePermission \"/proj/fagent/users/thom/c50a/-\",
\"read\";
```

Der WorkerCrack-Agent benötigt mehr Properties. Er benötigt mögliche Pfade für das systemabhängige Kopierprogramm, das Compilierhilfsprogramm make, eine Shell. das Löschprogramm, das Link-Programm und eine Crackinstallation. Da diese für die verschiedenen Agentensystem verschieden sein können, kann bei jeder Property mehre Möglichkeiten angeben werden. Dazu werden an die Namen der Properties Zahlen angehängt.
Als Beispiel:

```
${MASA_PACKAGE_thisagent}.copyLocation0=/bin/cp
${MASA_PACKAGE_thisagent}.makeLocation0=/soft/bin/make
${MASA_PACKAGE_thisagent}.bashLocation0=/bin/bash
${MASA_PACKAGE_thisagent}.rmLocation0=/bin/rm
${MASA_PACKAGE_thisagent}.lnLocation0=/bin/ln
${MASA_PACKAGE_thisagent}.crackDir0=/proj/fagent/users/thom/c50a/
${MASA_PACKAGE_thisagent}.crackDir1=/users/thom/c50a/
```

Benötigen die Crackinstallationen zusätzliche Optionen für den Start, so werden diese dem Agent über die Property additionalCrackOptions mitgeteilt.

Die Property `reportInterval` bestimmt das Intervall zwischen den Fortschrittsberichten an den MasterCrack-Agenten. Sie wird als Integer-Literal angegeben und gibt die Anzahl der Minuten zwischen zwei Fortschrittsberichten an.
In der Datei genPolicyFile.WorkerCrack müssen dem WorkerCrack-Agenten Lese- und Ausführrechte für die Systemprogramme erteilt werden. Für die Verzeichnisse (inklusive aller Unterverzeichnisse) der Crackinstallationen benötigt der Agent Lese-, Schreib-, Lösch- und Ausführrechte. Als Beispiel:

```
permission java.io.FilePermission \"/bin/cp\", \"read\";
permission java.io.FilePermission \"/bin/cp\", \"execute\";
permission java.io.FilePermission \"/soft/bin/make\", \"read, execute\";
permission java.io.FilePermission \"/bin/bash\", \"read, execute\";
permission java.io.FilePermission \"/bin/rm\", \"read, execute\";
permission    java.io.FilePermission    \"/proj/fagent/users/thom/c50a/-\",
\"read, execute, write,delete \";
permission    java.io.FilePermission    \"/proj/fagent/users/thom/c50a/\",
\"read, execute, write,delete \";
permission java.io.FilePermission \"/ users/thom/c50a/\", \"read, execute,
write,delete \";
permission java.io.FilePermission \"/ users/thom/c50a/-\", \"read, execute,
write,delete \";
```

Dabei ist zu beachten, dass bei den Crack-Verzeichnissen auch die Rechte am Verzeichnis selbst erteilt wird. Die Angabe `Crack-Verzeichnis/-` bezieht sich nur auf alle Verzeichnisse und Dateien unter diesem Verzeichnis, aber nicht das `Crack-Verzeichnis` selbst. Daher müssen die Rechte an diesem Verzeichnis selbst noch extra erteilt werden.

Danach kann die Installation der Agenten durch Eingabe von “make install” beendet werden,

8.2 Benutzung

Um die Agenten zu benutzen wird einfach der MasterCrack-Agent über das Applet eines Agentensystem gestartet. Dabei muss beachtet werden, dass von diesem Agentensystem die gewünschte Passwortdatei sowie die Crackinstallation zugreifbar ist.
Dabei hat man die Wahl zwischen 3 Konstruktoren:
Bei Wahl des ersten Konstruktors beginnt der MasterCrack-Agent sofort die Arbeit mit den in den Properties angegebenen Daten.
Der zweite Konstruktor bietet die Auswahl, ob der Agent sofort beginnen soll oder auf das Startsignal seines Applets warten soll. Hat der Parameter den Wert „false“ wartet der Agent, bei „true“ beginnt er sofort mit der Arbeit. Kann der Agent den Parameter nicht parsen, so wartet er.
Der dritte Konstruktor bietet zusätzlich zu dieser Auswahl die Wahl der Passwortdatei und der Crackinstallation über den Konstruktor. Der zweite Parameter ist der Pfad der Crackinstallation, der dritte der Name der Passwortdatei (inklusive Pfad).

Nach dem Start des MasterCrack-Agenten kann man sich über sein Applet die Ergebnisse ansehen bzw. den Agenten starten.

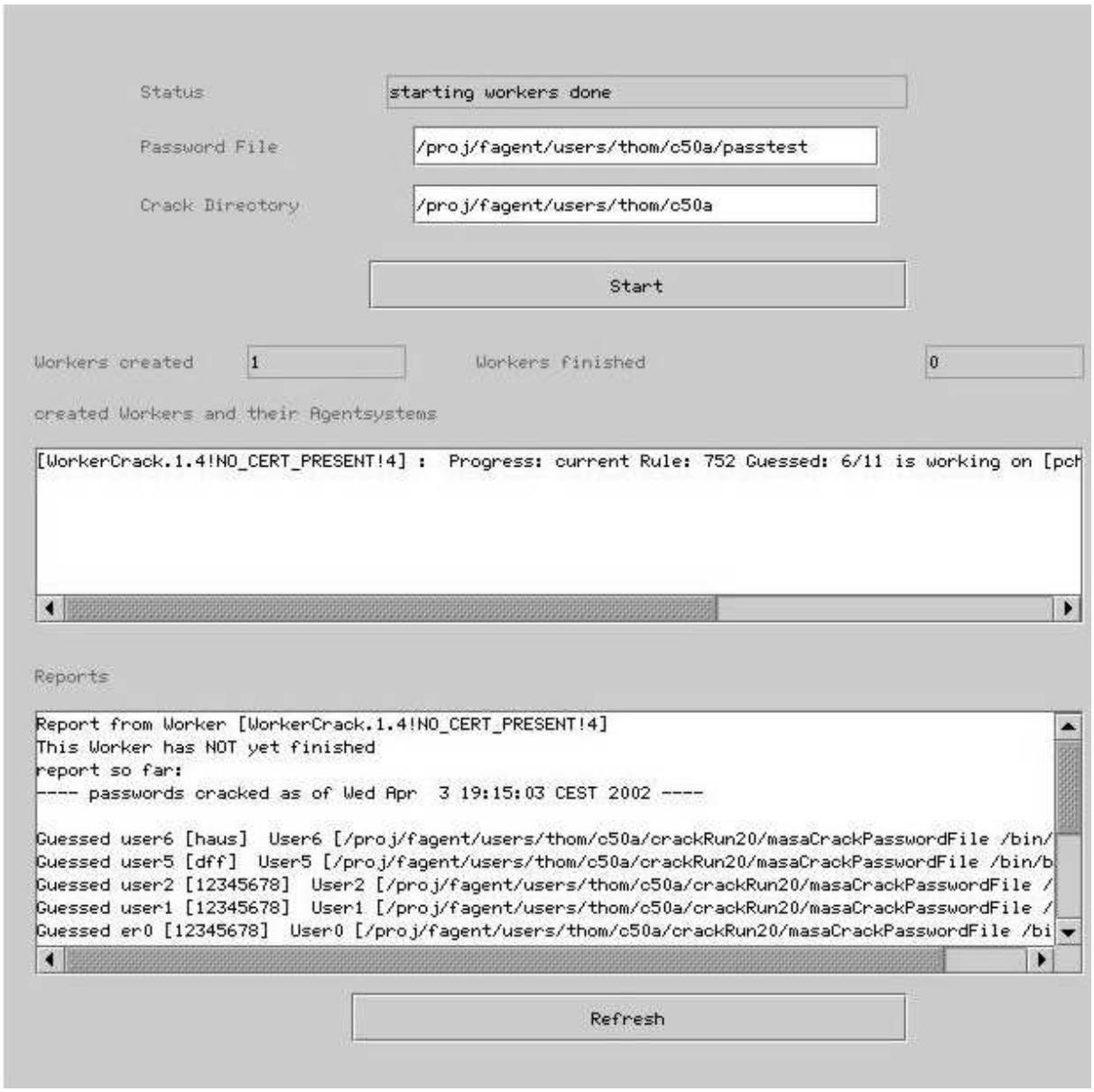

Abbildung 2: Applet des MasterCrack-Agenten

Dabei haben die Elemente des Applets folgende Bedeutung (von oben nach unten):
Im Status stehen die Meldungen des Agenten, insbesondere wird bei hier eine Fehlermeldung angezeigt, falls Dateien nicht gefunden wurden.
Darunter sind die Pfade der Passwortdatei und der Crackinstallation angegeben. Wurde der Agent noch nicht gestartet, stehen hier die Werte aus den Properties als Vorschläge.
Mit dem Start-Button wird der Agent gestartet. Der Agent versucht dann die in den beiden oberen Feldern eingegeben Werte zu bearbeiten. Ist der Agent bereits gestartet, hat eine weitere Betätigung des Start-Buttons keine weiteren Auswirkungen.
Hat der Agent mit der Arbeit begonnen, ist die Anzahl der erzeugten WorkerCrack-Agenten darunter zu lesen. Daneben ist die Anzahl der WorkerCrack-Agenten angegeben, die ihre Teilaufgabe fertig bearbeitet haben.
Im Textfeld darunter sind die Namen der erzeigten WorkerCrack-Agenten aufgeführt, ihr Fortschritt in Form der Nummer der Regel, die sie gerade bearbeiten und von wie vielen Benutzern das Passwort erraten wurde und am Ende der Zeile auf welchem Agentensystem sich der WorkerCrack-Agent befindet.

Das unterste Textfeld zeigt die End- oder Zwischenberichte der erzeugten WorkerCrack-Agenten an.
Alle Felder werden aktualisiert, wenn das Applet startet, ein WorkerCrack-Agent das Agentensystem wechselt, ein WorkerCrack-Agent seine Teilaufgabe fertig bearbeitet hat und wenn ein WorkerCrack-Agent einen Fortschrittbericht liefert. Durch das Betätigen des Refresh-Buttons können die Felder jederzeit aktualisiert werden. Dabei werden alle WorkerCrack-Agenten neu abgefragt.

Das Applet des WorkerCrack-Agenten wird im Normalfall nicht benötigt. Es zeigt lediglich einen Status-Text und den Zwischenbericht an. Beide Felder müssen dabei manuell durch Klicken des Refresh-Button aktualisiert werden.

Nachdem alle WorkerCrack-Agenten ihre Arbeit beendet haben, bleibt der MasterCrack-Agent bestehen. Um eine neue Passwortdatei zu untersuchen, muss ein neuer MasterCrack-Agent erzeugt werden.
Wird der MasterCrack-Agent beendet, bevor die WorkerCrack-Agenten fertig sind, laufen die WorkerCrack-Agenten weiter. Sind sie dann mit ihrer Arbeit fertig, geben sie auf der MASA-Systemkonsole eine Fehlermeldung aus, die den Bericht der bearbeitenden Teilaufgabe enthält.

9 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Arbeit kurz zusammen und beschreibt welche Verbesserungen noch gemacht werden können.

9.1 Zusammenfassung

Da Passwörter immer noch eines der meistverbreitestenden Mitteln der Authentifizierung und Authorisierung sind, haben Betreiber und Benutzer eines IT-Systems Interesse daran, dass sich diese Passwörter nicht leicht erraten lassen. Crack ist ein Programm eine Passwortdatei auf solche leicht zu erratenden Passwörter zu untersuchen, erzeugt dabei allerdings eine grosse Rechenlast. Daher wurde diese Last mit Hilfe von MASA, einer Plattform für verteiltes Management, durch Agenten auf mehrere System verteilt. Dazu werden von einem zentralen Agenten, MasterCrack genannt, die Last auf mehreren Teilaufgaben verteilt und diese werden dann von mehreren sogenannten WorkerCrack-Agenten bearbeitet. Dabei werden entweder die Wörterbücher oder Benutzer aus der Passwortdatei verteilt. Die WorkerCrack-Agenten bearbeiten diese Teilaufgaben indem sie auf ihrem System Crack mir ihrer Teilaufgabe aufrufen. Haben sie ihre Teilaufgabe fertig bearbeitet, melden sie das Ergebnis an ihren MasterCrack-Agenten, der dem Benutzer die gesammelten Ergebnisse über sein Applet präsentiert.

9.2 Ausblick

Raum für Verbesserungen ist vor allem an zwei Stellen: Am Applet des MasterCrack-Agenten und an der besseren Ausnutzung von Rechenkapazität bei grosser Heterogenität der Rechenkapazitäten der Agentensysteme.
Das Applet des MasterCrack-Agenten könnte man um diese Funktionen erweitern:

- Bei der Anzeige der Aufenthaltsorte auf Mausklick die Webseite des Agentensystems auf dem sich der ausgewählte WorkerCrack-Agent befindet öffnen.
- Die WorkerCrack-Agenten direkt vom Applet aus steuern, z.B. eine Migration veranlassen.

Der Ausgleich der verschiedenen Rechenkapazitäten ist allerdings schwieriger, da sich bei den gefundenen Agentensystem bisher keine Information über die Rechenkapazität abrufen lassen. Möglichkeiten wären vor dem Starten eine dritte Art Agent vorrauszuschicken, die die Rechenkapazität testen und überprüfen ob Crack auf diesem Systemlauffähig ist. Dann könnte für jedes System eine entsprechend grosse Teilaufgabe erstellt werden..
Oder der MasterCrack-Agent könnte langsame WorkerCrack-Agenten auf Agentensystem migrieren, auf denen WorkerCrack-Agenten liefen, die bereits fertig sind.
Ebenfalls denkbar ist die WorkerCrack-Agenten selbst „intelligenter“ zu machen, indem sie etwa selbstständig migrieren falls ihr Agentensystem stark belastet ist oder sie über den MasterCrack-Agenten erfahren, dass die anderen WorkerCrack-Agenten viel weiter sind.

10 Literatur

[GHR99] Boris Gruschke, Stephen Heilbronner, Helmut Reiser: „Mobile Agent System Architecture“, Technischer Bericht 9902, Ludwigs-Maximilians-Universität München, Aug 1999

[MASIF] „Mobile Agent System Interoperability Specification”, OMG TC Document orbos/98-03-09, Object Management Group, März 1998

Norman Hendrich: „Java für Fortgeschrittene“, Springer-Verlag Berlin Heidelberg, 1997

Sun Microsystems, “JavaTM 2 Platform, Standard Edition, v 1.3.1 API Specification”, http://java.sun.com/j2se/1.3/docs/api/index.html

Alec Muffet,“Alec Muffet’s Homepage”, http://www.users.dircon.co.uk/~crypto/

W. Huber : “Paralleles Rechnen, Eine Einführung“, R. Oldenburg Verlag, München, 1997