Studienarbeit

Methoden und Verfahren zur Optimierung der Analyse von Netzstrukturen am Beispiel des AGN-Malware Crawlers

René Soller

Betreut durch
Prof. Dr. Klaus Brunnstein

Arbeitsbereich
Anwendungen der Informatik in
Geistes- und Naturwissenschaften
Universität Hamburg
Vogt-Kölln Straße 30
22527 Hamburg

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Optimierung und Bewertung der automatischen Suche von Webseiten, die maliziöse Inhalte zum Download anbieten. Hierbei wird vor allem der AGN-Malware-Crawler betrachtet. Außerdem wird die graphische Darstellung der Netzstruktur gefundener Seiten zum Zwecke der Analyse betrachtet, um eventuelle Gruppendependenzen zu visualisieren. Die Studienarbeit ist in einen theoretischen und einen praktischen Teil gegliedert: Im theoretischen Teil werden Möglichkeiten zur Optimierung des Malware-Crawlers gesucht und die zugrundeliegende Problematik diskutiert. Im praktischen Teil folgt dann die Erläuterung der Implementation einer DLL zur Beschleunigung des betrachteten Malware-Crawlers. Im Weiteren wird ein Programm beschrieben, das Virenmeldungen, die per E-Mail empfangen werden, analysiert und graphisch aufbereitet. Dieses Programm und die DLL befinden sich in der Anlage zur vorliegenden Arbeit.

Inhalt

Abbildungsverzeichnis

Abkürzungsverzeichnis

AGN	- Arbeitsbereich „Anwendungen der Informatik in Geistes- und Naturwissenschaften“
ASCII	- American Standard Code for Information Interchange
GUI	- Graphical Userinterface (Grafische Benutzeroberfläche)
HTML	- Hyper Text Markup Language
MWC	- Malware-Crawler
NAI	- Network Associates Inc.
URL	- Universal Resource Locator
VTC	- Virus Test Center
W3C	- World Wide Web Consortium
WWW	- World Wide Web

Legende

Text	- Standardtext
Text	- Hervorgehobener Text
Text	- Zitate
`Text`	- Programmcode oder HTML-Code
`Text`	- Links im HTML-Code
Text	- Links
Text	- Beispiel-Links (fiktiv, aber Existenz möglich)

1 Einleitung

In den letzten Jahren konnte man im Scannertest des VTC einen Anstieg der zum Test benutzten Viren, insbesondere an Makroviren, feststellen:

Table ES0: Development of threats as present in VTC test databases:

Test#	= File viruses= Number Viruses	Infected objects	= Boot Viruses= Number viruses	Infected objects	=Macro Viruses= Number viruses	Infected objects	== Malware == Number file	Malware macro	=ScriptViruses= Number viruses	Number objects
1997-07:	12,826	83,910	938	3,387	617	2,036	213	72	---	---
1998-03:	14,596	106,470	1,071	4,464	1,548	4,436	323	459	---	---
1998-10:	13,993	112,038	881	4,804	2,159	9,033	3,300	191	---	---
1999-03:	17,148	128,534	1,197	4,746	2,875	7,765	3,853	200	---	---
	+ 5	146,640	(VKIT+4*Poly)							
1999-09:	17,561	132,576	1,237	5,286	3,546	9,731	6,217	329	---	---
	+ 7	166,640	(VKit+6*Poly)							
2000-04:	18,359	135,907	1,237	5,379	4,525	12,918	6,639	260	---	---
	+ 7	166,640	(VKit+6*Poly)							
2000-08:	---	---	---	---	5,418	15,720	---	500	306	527
2001-04:	20,564	140,703	1,311	5,723	6,233	19,387	12,160	627	477	904
	+ 7	166,640	(Vkit+6*Poly)							

Remark: Before test 1998-10, an ad-hoc cleaning operation was applied to remove samples where virality could not be proved easily. Since test 1999-03, separate tests are performed to evaluate detection rates of VKIT-generated and selected polymorphic file viruses.

[AGN]

Folgendes Diagramm veranschaulicht dies:

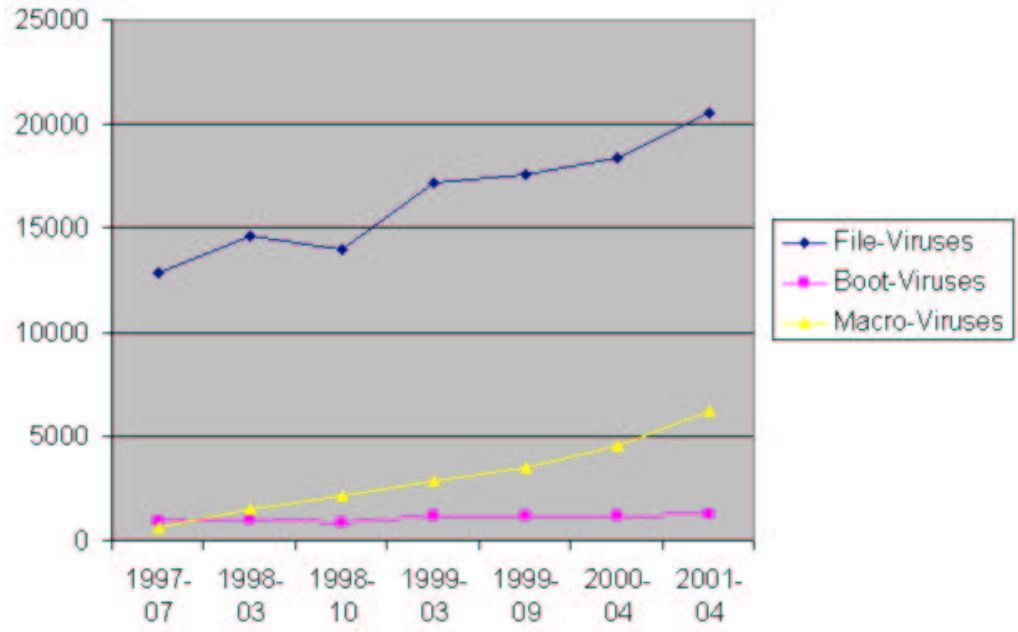

Abbildung 1 - Anstieg der getesteten Viren

Angesichts dieser steigenden Verbreitung von Computerviren ist es notwendig, Gegenmaßnahmen einzuleiten. Da ein Großteil der Anwender auch nach medienwirksamen Virenvorfällen wie dem Loveletter- oder AnnaKournikova-Virus keine Verhaltensänderung zeigt, und somit keine wirksamen Gegenmaßnahmen ergreift, müssen andere bzw. bessere und effizientere Wege gefunden werden, gegen Viren vorzugehen. Am besten wäre es, die Viren schon vor ihrer Verbreitung aufzufinden. Diesen Ansatz verfolgt der AGN-Malware-Crawler, ein Programm zum automatischen Auffinden von Viren im Internet, welches aus der Diplomarbeit von Sönke Freitag [MWC-00] hervorgegangen ist. Da es aufgrund der immensen Datenmenge im Internet nicht möglich ist, alle Webseiten automatisch abzusuchen (siehe hierzu eine Abschätzung in Kapitel 2.3), ist es notwendig, das Programm und die vorhandene Heuristik weitestgehend zu optimieren, um einen größtmöglichen Teil des relevanten WWW-Raums zu durchsuchen.

1.1 Ziele

Ziel dieser Arbeit ist es, Optimierungsmöglichkeiten der bestehenden Heuristik und anderer wichtiger Teile, wie z.B. der Stringoperationen, des AGN-Malware Crawlers aufzuzeigen und zu realisieren. Das Auffinden von Malware im WWW durch den MWC kann, um einer weiteren Verbreitung entgegenzuwirken, zur Schließung der betreffenden Webseite führen, soweit dies nach den Gesetzen des jeweiligen Landes möglich ist. Außerdem können Anti-Viren-Programme frühzeitig aktualisiert werden, falls bisher unbekannte Viren oder Varianten gefunden werden. Des Weiteren könnten Statistiken über die Verteilungswege der Malware erstellt und Zusammenhänge zwischen den Virenautoren ermittelt werden.

1.2 Grenzen des MWC

Ohne ein gewisses Maß an aktiver Mitarbeit seitens des Anwenders ist ein wirksames Vorgehen gegen Viren ausgeschlossen. Mit Hilfe des MWC ist es zwar bereits möglich, die Präsenz von Viren im Internet zu entdecken und die Virenscanner zu aktualisieren, so dass sie bereits beim ersten Auftreten eines neuen Virus wirksam schützen können, bevor mit der unkontrollierten Verbreitung Schaden angerichtet werden kann. Sofern kein Virenscanner eingesetzt wird, bzw. die Signaturen eines vorhandenen Virenscanners nicht oft genug auf den neuesten Stand gebracht werden, besteht aber auch weiterhin kein Schutz vor Computerviren.

Auch existieren eine Vielzahl von Gegenmaßnahmen, welche die Verbreiter von bösartiger Software gegen den Malware Crawler einsetzen können, so dass nicht alle Seiten, die Viren enthalten als solche erkannt werden. So ist es z.B. möglich,

Links nicht fest in HTML-Code einzubauen, sondern erst durch Java, Java-Script[1], Flash oder Shockwave generieren zu lassen, etwa durch das Zusammensetzen der URLs aus Einzelteilen oder durch kompliziertere Berechnung der Links. Außerdem ist es denkbar, dass der gesamte Text einer Seite als Bild dargestellt wird, so dass es erforderlich werden würdre, diese mit OCR[2]-Verfahren zu analysieren, was beim heutigen Stand der Technik noch ziemlich langsam und fehleranfällig ist, vor allem wenn versucht wird, dieses systematisch durch ausgefallene Schriftarten oder andere Tricks zu verhindern.

Selbst wenn alle diese Hürden überwunden würden, gäbe es immer noch die Sprachenabhängigkeit des Heuristik-Algorithmus. So kann die Heuristik zwar an verschiedene Sprachen angepasst werden, was aber gleichzeitig einen nicht unerheblichen Aufwand bedeuten würde. Weiterhin müssten auch andere Zeichensätze, wie etwa das griechische Alphabet oder chinesische Schriftzeichen, berücksichtigt werden. Die oben aufgeführten Punkte machen deutlich, dass es immer Wege geben wird, die Heuristik des Malware-Crawlers zu täuschen, es soll aber versucht werden, dieses auf ein Minimum zu reduzieren.

[1] Javascript ist eine Skriptsprache, die direkt in HTML-Dokumente eingebunden ist
[2] Optical Character Recognition, Verfahren zur Erkennung von Buchstaben aus Grafiken

2 Bestandsaufnahme

Der AGN-Malware Crawler ist ein Resultat der Diplomarbeit „Webbasiertes Auffinden maliziöser Software mit fortschrittlichen heuristischen Verfahren“ von Sönke Freitag [MWC-00]. Im Folgenden wird die Funktionsweise des Crawlers kurz beschrieben. Bei Bedarf findet der Leser ausführlichere Informationen zum Malware Crawler in der oben genannten Diplomarbeit, die im Internet auf der Webseite des Fachbereichs Informatik, Arbeitsbereich AGN der Universität Hamburg [AGN] eingesehen werden kann.

2.1 Kurze Beschreibung des Malware Crawlers

Der AGN-Malware Crawler arbeitet ähnlich wie ein Browser. Begonnen wird mit einer Startseite oder Startliste, aus der/denen alle Links, die Tags (siehe weiter unten) und der Text extrahiert werden. Dann werden die extrahierten Links nacheinander verfolgt und aus diesen Webseiten ebenfalls Tags, Text und Links nach Vorgabe der Crawl-Parameter extrahiert. Der Inhalt der Seite wird durch ein mehrstufiges heuristisches Verfahren bewertet und je nach Höhe dieses Heuristikwertes werden die enthaltenen Links oder die zuvor extrahierten Links weiterverfolgt. Die Links werden in einer Datenbank gespeichert, so dass mehrfache Seitenaufrufe (Endlos-Schleifen) vermieden werden.
Während des Crawlens werden alle Dateiverweise von Seiten mit einem genügend hohen Heuristikwert gespeichert. Diese Dateien werden später geladen und mit einem Virenscanner im Heuristik-Modus[3] nach verdächtigen Inhalten untersucht.

2.2 HTML

Um die im folgenden Abschnitt erklärte Extraktion von Text, Links und Tags (siehe 4.2) aus HTML-Dateien besser zu verstehen, werden hier kurz die grundlegenden Dinge über das HTML-Format angeführt.
HTML–Dokumente sind Textdateien, die bestimmte Formatierungsanweisungen enthalten. Um diese Anweisungen von normalem Text zu unterscheiden, sind sie in spitze Klammern (< und >) eingeschlossen. Sie werden als „Tags“ bezeichnet. Es gibt zwei verschiedene Arten von Tags:

1) Tags, die einen Abschnitt umschließen. Hierbei steht jeweils ein Tag vor und ein Tag hinter dem jeweiligen Abschnitt, wobei das zweite Tag das gleiche wie das erste ist, jedoch zur Unterscheidung noch einen Schrägstrich nach der ersten spitzen Klammer hat. So sieht die Formatierung einer Überschrift (= größere Schrift) folgendermaßen aus:

[3] Es wird nach Programmteilen gesucht, die Schaden anrichten könnten, oder die für die Replikation zuständig sind.

```
<H1> Ueberschrift </H1>
```

2) Alleinstehende Tags.
   So bewirkt beispielsweise
 einen Zeilenwechsel.

Dabei spielt es keine Rolle, ob die Tags in Groß- oder Kleinbuchstaben dargestellt werden. Es gibt einige Vorteile für Großbuchstaben (z.B. bessere Lesbarkeit) als auch für Kleinbuchstaben (bessere Komprimierbarkeit). Da es hier aber nur um die Analyse, nicht aber um das Erstellen geht, ist dies unerheblich und es müssen beide Versionen berücksichtigt werden.

Das Grundgerüst eines zu analysierenden HTML – Dokumentes sieht so aus:

```
<HTML>
	<HEAD>
		<TITLE> Titel der Webseite</TITLE>
	</HEAD>
	<BODY>
		Text der Webseite
	</BODY>
</HTML>
```

Man sieht, dass der gesamte Inhalt des Dokumentes in <HTML> und </HTML> eingeschlossen ist und dass Tags auch ineinander verschachtelt werden können. Mit <TITLE> und </TITLE> wird der Titel der Webseite markiert, der in der Fensterleiste des Browser angezeigt wird. Das Einrücken des Textes ist zwar nicht unbedingt erforderlich, jedoch möglich und dient hier nur der besseren Lesbarkeit. Die <HEAD>- und <BODY>-Tags kennzeichnen die zwei Hauptbereiche des Dokumentes: Den Kopfbereich und den Körper.

Weiterhin können Tags noch bestimmte Parameter enthalten, wie z.B. bei Bildern:

```
<IMG SRC="bild.jpg">
```

Das Tag ist hier <IMG>, der Parameter SRC ist das Bild „bild.jpg“. Auch die Links werden als Parameter realisiert.

Es gibt zahlreiche Tags mit den verschiedensten Parametern, die hier aber nicht alle erläutert werden sollen. Die wichtigsten (hauptsächlich solche zur Darstellung von Links) werden später im Abschnitt Link-Extraktion erklärt. Eine ausführliche Auflistung und Beschreibung aller Tags findet man bei [SelfHTML].
Außer den Tags gibt es noch einige Sonderzeichen, zu denen auch die Umlaute „ä“, „ö“ und „ü“ gehören sowie „ß“ und weitere sprachspezifische Zeichen wie „é“, „î“ usw.

Diese werden durch ein „&“-Zeichen, einen Kürzel (2 bis 4 Zeichen) für das Zeichen und abschließendes Semikolon dargestellt. Die Liste der häufigsten (und unterstützten) Sonderzeichen:

Zeichen	**HTML-Code**
ä	ä
ö	ö
ü	ü
Ä	Ä
Ö	Ö
Ü	Ü
ß	ß
[erzwungenes Leerzeichen]	
>	>
<	<
'	"
&	&
©	©
®	®

2.3 Filterung, Heuristiken

Ein Absuchen des gesamten WWW ist nicht möglich. Folgende Abschätzung macht dies deutlich:

Bei Google [Google] sind 1.610.476.000 Webseiten (Stand: 12.09.2001) indiziert. Es existieren also mindestens 1.610.476.000 Webseiten. Angenommen jede Webseite wäre nur 10kB groß. Dies bedeutete eine abzusuchende Menge von 16.104,76 GB. Man nehme weiter an, dass die maximale Übertragungsrate des Labornetzanschlusses (10MBit/s[4]) dauerhaft ausgenutzt werden kann. Beide Annahmen kombiniert kommen zu folgendem Ergebnis:

Zeit des Downloads = Datengröße [Bytes] / Übertragungsrate [Bytes/s]

= 16.104,76 GB / 10MBit/s (=1,25 MByte/s[5]) = 12.883.808 s = 149,11 Tage

Der Crawler benötigte zur Erledigung seiner Aufgabe 149,11 Tage.

Diese Berechnung erfolgt unter der Abstraktion einer unendlich kleinen Verarbeitungszeit, der Vernachlässigung langsamerer Gegenstellen und Internetverbindungen.

[4] Eigentlich 100MBit/s, die Firewall unterstützt aber nur 10 MBit/s
[5] Wobei dies eine rein theoretische Umrechnung ist. Realer wäre: 10MBit/s = 0,7 MByte/s

Weiterhin sind in diese Rechnung natürlich noch nicht das Downloaden, die Rechenzeit des Programms zur Analyse und Übertragungsengpässe sowie eventuelle Verzögerungen bzw. Datenverluste während der Übertragung eingegangen. Außerdem benötigte man selbst bei einer Komprimierung der URLs auf 16 Bytes mindestens 25,7 GB für die Speicherung der URLs (hinzu kommt jeweils noch der Heuristikwert, CRC usw.). Aus diesem Grund ist eine gute Heuristik notwendig, welche die Suche auf die relevanten Seiten begrenzt.

Die Heuristik ist mehrstufig, d.h. es werden nicht nur die HTML-Dokumente analysiert (Stufe I), sondern es wird auch bewertet, wenn eine Seite auf Seiten verweist, die in der „Badlist“ aufgeführt sind (Stufe II). In dieser „Badlist“ befinden sich Links auf Seiten, von denen bereits bekannt ist, dass sich auf ihnen maliziöse Inhalte befinden. Der URL der analysierten Seite selbst fließt in Stufe III in den Heuristikwert mit ein. Da derzeit häufig auf russischen[6] und slovakischen Webseiten maliziöse Inhalte zu finden sind, werden z.B. .ru - Domains höher bewertet als .net - Domains. In der vierten und letzten Stufe wird schließlich noch die Entfernung der Seite zu anderen Seiten anhand der Verzeichnistiefe einbezogen, da sich viele Webseiten mit maliziösen Inhalten Domains teilen, beispielsweise „www.domain.net/ virenautor1/index.html“ und „www.domain.net/virenautor2/ index.html“. In der ersten Stufe werden nicht einfach Schlüsselwörter gesucht und diesen Werte zugeordnet, die dann miteinander verknüpft werden. Vielmehr fließt auch die Nähe einzelner Schlüsselwörter zueinander mit in den Heuristikwert ein, so dass schon eine gewisse semantische Komponente vorhanden ist. Dazu erfolgt eine Aufteilung in Keywords und Qualifier, wobei der Malware-Crawler den Heuristikwert bei einigen Keywords anstatt zu erhöhen auch verringern kann, wie z.B. beim Wort „Virenscanner„.

[6] nicht zuletzt wegen der Gesetzeslage in Bezug auf Viren dort

3 Mögliche Optimierungen

Nach der Bestandsaufnahme in Kapitel 2 sollen an dieser Stelle mögliche Optimierungen aufgezeigt werden, um beim Crawlen und beim Downloaden zu besseren Resultaten zu gelangen. Optimierung bedeutet hierbei nicht nur das Erzielen einer besseren Trefferquote oder die erhöhte Relevanz gefundener Seiten, sondern auch eine Geschwindigkeitsverbesserung der Crawl- und Downloadzeiten. Dabei geht es sowohl um die Benutzerfreundlichkeit als auch um Stabilität und Geschwindikeit des Programms, sowie das Crawlen und die Heuristik.

3.1 Benutzerfreundlichkeit

Da auch eine erhöhte Benutzerfreundlichkeit zu besseren Ergebnissen führen kann, soll diese hier betrachtet werden.

3.1.1 GUI

Die grafische Benutzeroberfläche des Malware-Crawlers ist kompakt und gut gegliedert. Die wichtigsten Funktionen sind auf Seitenelementen untergebracht, zwischen denen schnell hin- und hergeschaltet werden kann. Es dauert sicherlich einige Zeit, um sich in die Bedienung einzuarbeiten. Danach ist es jedoch leicht, in den verschiedenen Seitenelementen zu navigieren.

Da der Malware-Crawler die meiste Zeit selbständig arbeitet, ist das GUI eher von geringerer Bedeutung. Man könnte aber folgende Punkte verbessern:

- Tooltip-Funktion hinzufügen
- Menüs anstatt Reiterelemente für weniger oft benötigte Funktionen, z.B. das Setup
- Entfernen des Startdialoges, in dem nur zwischen den drei Funktionen „Malware-Crawler“, „Filelist-Scan“, „Send SMTP-Mail“ und beenden gewählt werden kann und das als Ausgabefenster dient.

3.1.2 Reportformate

Um den Malware-Crawler bei seiner Arbeit zu überwachen, gibt der Crawler während des Crawlens laufend Meldungen über seinen Status per E-Mail und am Bildschirm aus. Die Gliederung ist sehr detailliert und kann bei fehlerhaften Seiten, Attacken gegen den MWC oder auch bei Bugs sehr hilfreich sein. Man sieht also bei jedem der protokollierten Ereignisse, welcher Programmteil gerade abgearbeitet wird. Zu jeder Meldung erscheint zusätzlich die genaue Uhrzeit des

Ereignisses. Bei Problemen läßt sich also sehr gut nachvollziehen, zu welcher Zeit welches Ereignis stattgefunden hat. Als weitere Optimierung wären die Darstellung von Statistiken oder das Erstellen von Laufzeitbetrachtungen denkbar.

3.2 Stabilität / Geschwindigkeit

Damit der Malware-Crawler längere Zeit laufen kann, muss eine ausreichende Stabilität sichergestellt sein. Außerdem sollte trotz der Heuristik ein möglichst großer Teil des Internets durch die Suche abgedeckt werden. Dazu ist es erforderlich, die Geschwindigkeit des Crawlens zu erhöhen. Eine Möglichkeit dazu ist ein Verteilen und Parallelisieren der Aufgaben.

3.2.1 Netzwerktauglichkeit

Um die Aufgaben des Malwarecrawlers zu verteilen, ist es zuerst notwendig, ihn netzwerkfähig zu machen. Es könnte beispielsweise einen Rechner für das Crawlen und einen für das Downloaden geben. Diese Rechner müssten natürlich miteinander kommunizieren, um die Aufgaben zu koordinieren. Hauptsächlich würde dies die Datenbank betreffen, auf die beide (oder mehrere) Rechner Zugriff bekommen müssten.

3.2.2 Parallelität der Verarbeitung

Auch die beiden Hauptaufgaben - Crawlen und Download - können aufgesplittet und auf verschiedenen Rechnern bearbeitet werden. Dabei muss sichergestellt werden, dass nicht die gleichen URLs bearbeitet werden. Denkbar wäre eine Komponente, die sich um die Aufgabenverteilung kümmert. Diese würde die zu untersuchenden URLs auf die einzelnen Rechner verteilen und die Ergebnisse verwalten. Die Verteilung könnte aber durch eine zentrale Datenbank realisiert werden, indem die verschiedenen Rechner einzelne URLs aus der Datenbank auslesen und markieren. Die Rechner würden sich die Arbeit also in gewisser Weise selbst zuteilen, indem sie sich die Links zu noch nicht bearbeiteten Seiten aus der zentralen Datenbank holen. Zugriffskonflikte müssten hierbei natürlich vermieden werden, was aber meistens bereits von den Datenbanksystemen übernommen wird.

3.2.3 Parallelität des Downloads

Ähnlich wie beim parallelen Verarbeiten kann auch das Downloaden verdächtiger Dateien mit mehreren Rechnern parallel erfolgen. Auch hier wird gemeinsam auf eine Datenbank zugegriffen.

Es muss beachtet werden, dass sowohl beim Downloaden als auch bei der parallelen Verarbeitung es nicht möglich ist, beliebig viele Rechner zusammen zu schalten, da der gemeinsame Zugang zum Internet als Flaschenhals-Effekt bleibt. Es ist also genau abzuwägen, bei wie vielen Rechnern die Leistungsgrenze der Internetanbindung erreicht wird.

3.3 Beschleunigung der Stringoperationen

Das Crawlen ist eine der wichtigsten Aufgaben des MWC. Die Stringoperationen in Visual FoxPro sind sehr langsam, da es sich um eine interpretierte Sprache (Pseudo-Code) handelt. Da beim Analysieren der HTML-Dateien Stringoperationen eine wesentliche Rolle spielen, kommt es zu einem Engpass. Es ist also notwendig, diese Operationen speziell für diese Aufgabe zu optimieren. Die als Anlage existierende Visual C++-DLL, die als ein Resultat aus der vorliegenden Arbeit entstanden ist, erledigt genau diese Aufgabe. Wie dies genau geschieht, wird später in Kapitel 4.2 beschrieben.

3.4 Heuristik

Der vom Malware-Crawler berechnete Heuristikwert liegt zwischen 0 (= irrelevant) und 9999 (= enthält sehr wahrscheinlich Malware-Verweise). Wenn man stichprobenweise untersucht, auf welchen Webseiten tatsächlich Malware zu finden ist, so erkennt man, dass auch Seiten mit einem relativ niedrigem Heuristikwert Malware enthalten können. Der frei einstellbare Grenzwert zu relevanten Seiten ist momentan auf 10 festgelegt. Um den Heuristikwert besser einstufen und abschätzen zu können, sollte die Grenze aber ungefähr in der Mitte des Wertebereichs liegen. Eventuell ist es erforderlich, die verschiedenen Stufen der Heuristik auf eine andere Art zu verknüpfen als es bisher der Fall ist. Zumindest sollte eine Bewertung und Studie zwischen Heuristikwert und tatsächlich gefundenen Viren erfolgen, um die Heuristik besser beurteilen zu können.

3.4.1 Einbeziehung von Metainformationen aus Suchmaschinen

Suchmaschinen liefern bei geeigneten Suchbegriffen (s. [MWC-00]) schon recht viele Viren-Sites. Dies könnte ausgenutzt werden, indem man die Suchergebnisse in die MWC-Suche einfließen lässt. Die von den Suchmaschinen gefundenen Links könnten als Ausganspunkte des Malware-Crawlers dienen, von denen aus dann weiter gecrawlt wird. Dies ist notwendig, da sonst nur Seiten gefunden werden, die auch gefunden werden wollen.

Es können zwar Suchmaschinen mit entsprechenden Suchbegriffen als Ausgangsseite gewählt werden[7], jedoch wären auch automatische Aufrufe denkbar.

3.4.2 Verarbeitung von Usenet-Eingaben des NAI-Crawlers

Network Associates Inc. (NAI) hat einen Crawler (Virus Patrol) entwickelt, der, im Gegensatz zum AGN, Malware-Crawler nicht das Internet, sondern das Usenet nach Viren durchsucht. Liest man die vom NAI-Crawler gelieferten Daten ein und bereitet sie graphisch auf, so erhält man einen guten Überblick darüber, welche Viren/Würmer/Trojaner wann am häufigsten auftraten. Dies wurde durch das weiter unten (4.1) erläuterte Programm „VPAnalyzer“ realisiert. Weiterhin wäre es interessant, die Ergebnisse dieses Programmes in Relation zu den vom MWC gefundenen Viren zu setzen.

3.4.3 Semantische Heuristiken

Versucht man dem HTML-Code einen Heuristikwert nicht nur mit Hilfe einer Wertetabelle mit möglichst vielen Wörtern zuzuordnen, sondern anhand der Bedeutungsanalyse von einzelnen Worten, kommt man zur semantischen Heuristik.

Als gutes Beispiel dient das Wort „Virus“. Es hat zwei grundlegend verschiedene Bedeutungen[8]:

Virus *<“Schleim, Saft, Gift“> das (auch: der); -, Viren:*
1. *kleinstes [krankheitserregendes] Partikel, das sich nur in lebendem Gewebe entwickelt.*
2. *Computerprogramm, das falsche oder zerstörerische Befehle in anderen Programmen auslöst.*

[Dud5, S. 848]

Es ist klar, dass nur Webseiten über Viren mit der Bedeutung der zweiten Definition relevant sind. Da allein aus dem Wort „Virus“ aber nicht geschlossen werden kann, um welche Bedeutung es sich handelt, muss der Kontext analysiert werden, um die Semantik zu ermitteln. Dies geschieht zwar schon teilweise durch Einbeziehung der Entfernung zwischen den Schlüsselwörtern in der ersten Stufe der Heuristik, könnte aber noch ausgefeilter erfolgen, z.B. durch Satzanalyse.

[7] für einige Suchmaschinen sind schon Startpunkte mit Keywords vordefiniert
[8] Dies ist keine wissenschaftlich korrekte Definition, sondern dient lediglich als Beispiel für die unterschiedliche Semantik

3.4.4 Interferenz-Heuristiken

Wenn feststeht, dass auf einer Webseite maliziöser Inhalt zum Download angeboten wird, dann verweisen auch von dort ausgehende Links mit einer hohen Wahrscheinlichkeit auf solche Seiten [Spertus98]. Dies wird umso wahrscheinlicher, je mehr Seiten, die Malware enthalten, gemeinsam auf eine weitere verweisen.

Abbildung 2 - Interferenz-Heuristik

4 Implementation

Im Rahmen einer Studienarbeit ist es nicht möglich, alle aufgeführten Möglichkeiten der Optimierung auszuführen. Jedoch wurden einige der wichtigsten Punkte implementiert, die nun im einzelnen erläutert werden sollen:

4.1 Graphische Darstellung der in Newsgroups aufgetretenen Viren/Würmer/Trojaner

Das unter der Leitung von Herrn Prof. Dr. Klaus Brunnstein entstandene Programm „VPAnalyzer“ stellt die vom NAI-Crawler (s. 3.4.2) gelieferten Daten graphisch dar.

Folgende Anforderungen sollten erfüllt werden:

- Die graphische Ausgabe des Programms soll zur Darstellung im Internet geeignet sein
- Die aufgetretenen Viren-Arten (bzw. Trojaner/Würmer) sollen unterschieden werden
- Die Ausgabe soll als Jahres- und Monatsübersicht erfolgen

Der erste Punkt wurde vom Autor der vorliegenden Arbeit dadurch gelöst, dass das Programmes als Ausgabe bereits fertige HTML-Dateien liefert.

4.1.1 Einlesen der Daten

Da die vom NAI-Crawler generierten Daten anfangs nicht dafür gedacht waren, maschinell ausgewertet zu werden, haben sie kein spezifiziertes Format. Dies machte das Einlesen und Interpretieren der Daten besonders schwer.
Die Daten eines Tages liegen in jeweils einer Datei, in dessen Name das Datum kodiert ist. Da sie dafür gedacht sind, per E-Mail versendet zu werden, liegen sie in Textform vor. Um die relevanten Daten zu extrahieren, sucht der VPAnalyzer nach bestimmten Strings (z.B. „Subject“), auf die dann die benötigten Informationen folgen, z.B. „VIRUS ALERT!“ oder „TROJAN ALERT!“, was für die Unterscheidung zwischen Viren/Würmern und Trojanern wichtig ist. Die extrahierten Daten werden in einer eigenen Datei gespeichert, um später, z.B. bei neuer Farbzuweisung, nicht erneut analysiert werden zu müssen. Aus diesem Grund ist im Programm zwischen dem Einlesen der Daten und dem Erstellen der Statistik getrennt worden. Neben diesen Funktionen kann im Hauptdialog auch der Zeitraum der Statistik eingestellt werden.

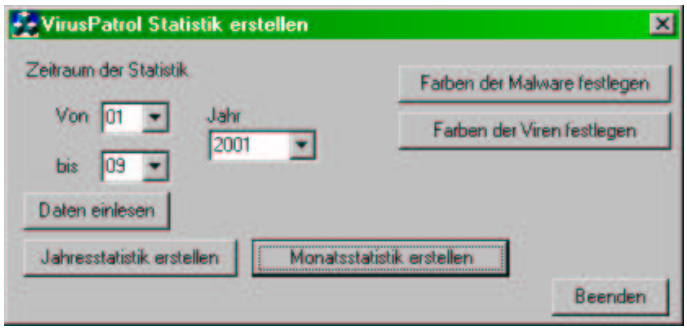

Abbildung 3 - VPAnalyzer Hauptdialog

4.1.2 Farbauswahl

In den resultierenden HTML-Dokumenten wird die Malware in je zwei Balkendiagrammen – Trojaner und Viren/Würmer - dargestellt. Die x-Achse stellt die Zeit dar (Tag oder Jahr), die y-Achse die Anzahl gefundener Malware. Die Balken der Malware sind farblich unterscheidbar. Die Farben sollen nicht einfach zufällig verteilt werden, vielmehr sollen zusammengehörige Gruppen - wie etwa Script-Viren - ähnliche Farben bekommen. Da das Programm diese Farben aber in der aktuellen Implementierung nicht selbständig zuordnen kann, muss dies vom Benutzer geschehen. Um dies zu erleichtern, werden alle Farbzuordnungen als Kästchen auf einem Farbkreis dargestellt. Daneben erscheint eine Liste der Malware der entsprechenden Kategorie. Später ist eine automatische Zuordnung der Farben anhand des Namens möglich.

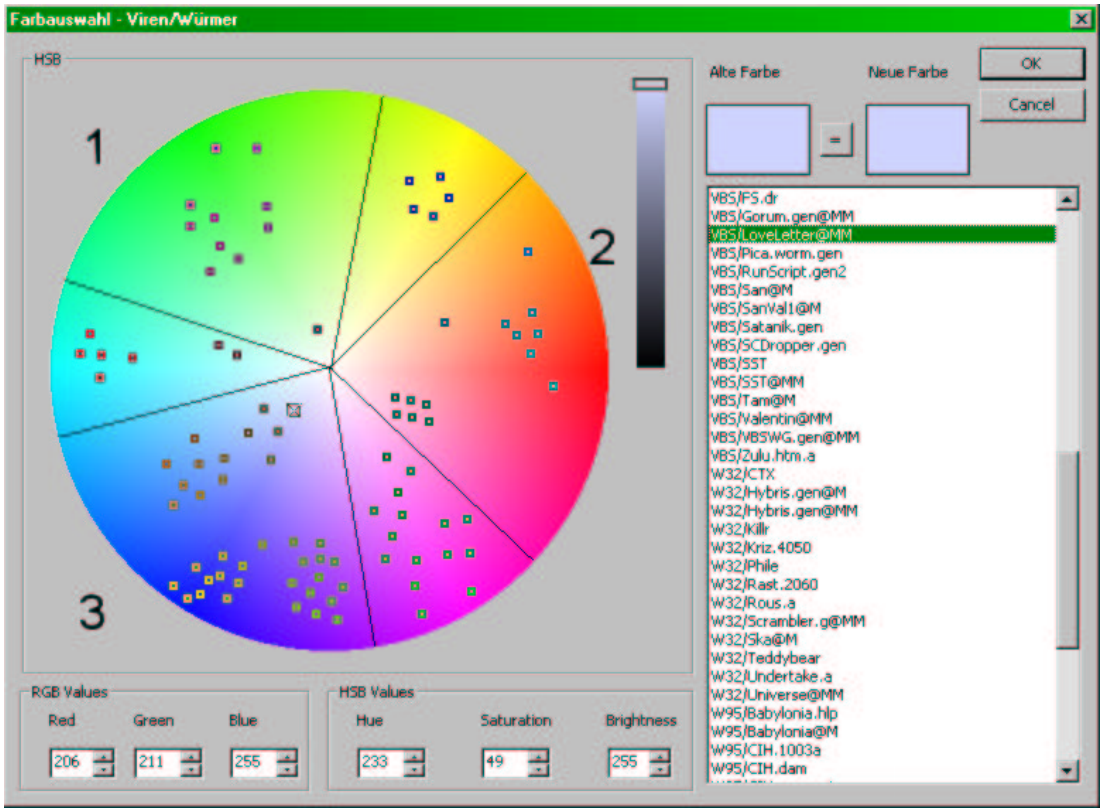

Abbildung 4 - VPAnalyzer Farbauswahl

Durch Klicken mit der linken Maustaste kann dem markierten Eintrag in der Liste eine Farbe zugeordnet werden. Mit der rechten Maustaste kann man den zu einem Kästchen auf dem Farbkreis gehörenden Eintrag aus der Liste auswählen.

Nun noch ein Beispiel einer entstandenen HTML-Datei:

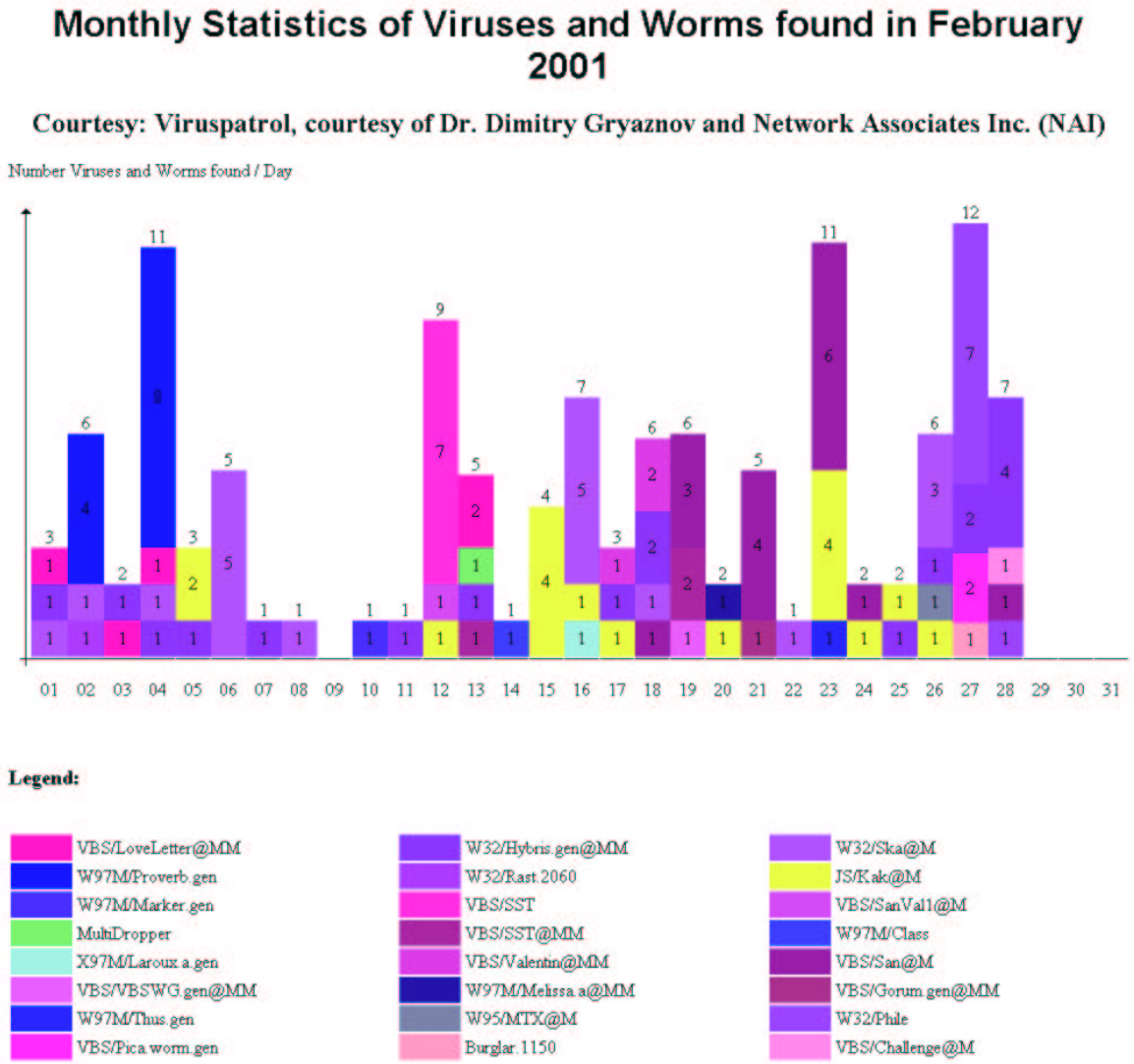

Abbildung 5 - Virenstatistik Februar 2001

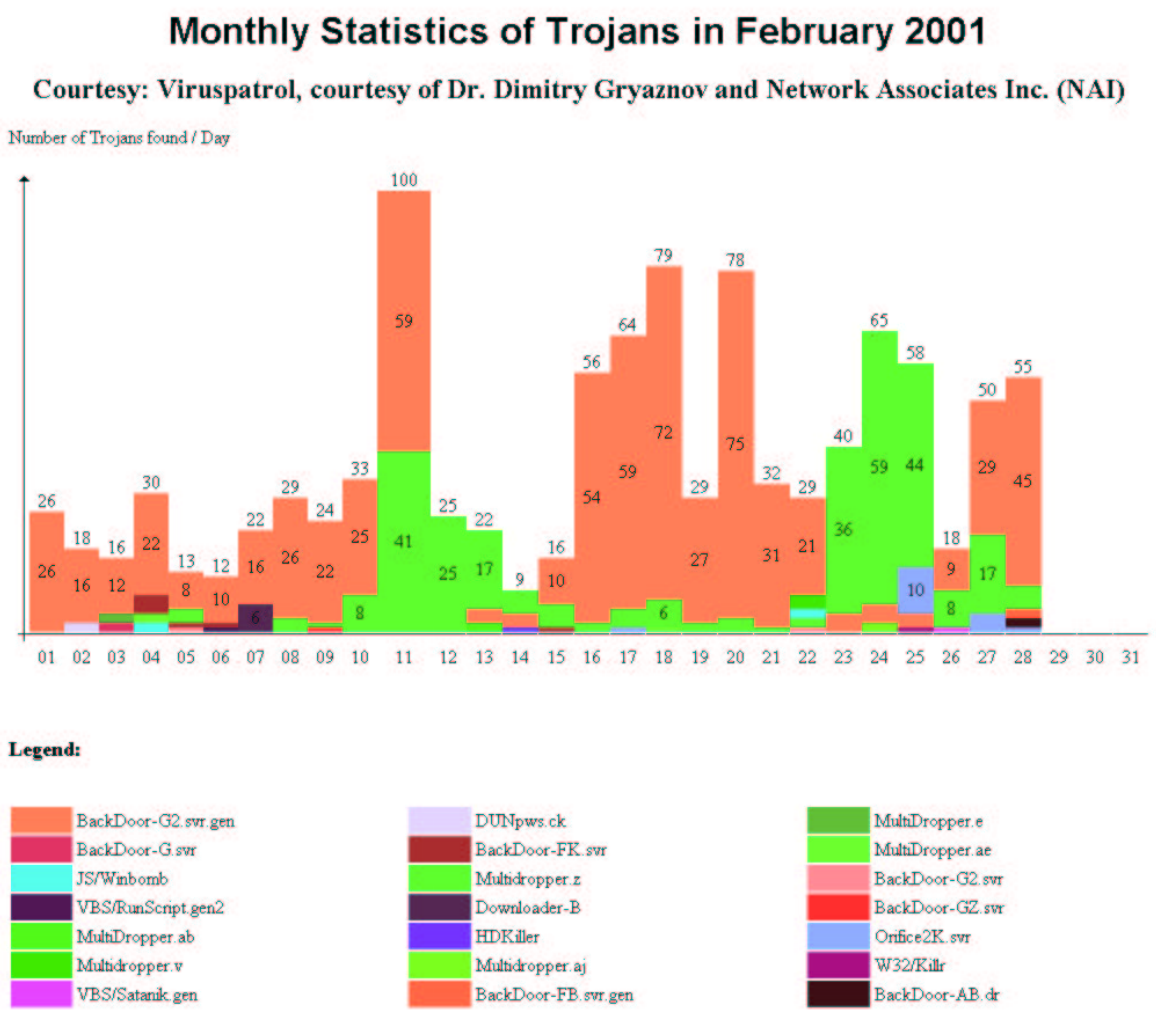

Abbildung 6 - Malwarestatistik Februar 2001

4.2 Crawling Optimierungen

Da Visual FoxPro in einigen sehr wichtigen Bereichen, z.B. Datenbankverwaltung ziemlich stark und deshalb kaum zu verbessern ist, empfiehlt es sich, nur die wesentlichen Teile auszulagern. Dies wurde durch eine Visual C++-DLL realisiert, die in das Visual FoxPro–Programm eingebunden wird. Die DLL ist für die Analyse der HTML-Dateien zuständig. Sie extrahiert den Text, die Links und die Tags und gibt das Ergebnis an den Malware-Crawler zurück, der dann die Eintragung in die Datenbank(en) und das Weiterverfolgen der Links übernimmt. Um in den nächsten Abschnitten die genaue Vorgehensweise besser veranschaulichen zu können, wird dies anhand eines einfachen Beispiels einer HTML-Datei geschehen. Diese Datei enthält alle wesentlichen Darstellungen von Links, sowie META-Tags, Kommentare und Javascript. Um die Tauglichkeit des Programms zu untermauern, sind einige Fehler in dieses Beispiel eingebaut, wie sie selbst bei beneziösen Webseiten mitunter vorkommen, da man sich auch hier nicht immer an die Konventionen des W3C [W3C] hält.

```
<HTML>
     <HEAD>
          <TITLE> Titel </TITLE>
     </HEAD>

<BODY onload="window.location='http://www.seite.de/'"> >
<H1>&Uuml;berschrift</H1>



<!—Abschnitt 1: Einige einfache Links <A HREF=... >Text</A> -->

<A HREF=“ http://www.irgendeinlink.de“>Link 1</A>
<A HREF=‘http://www.nocheinlink.de‘>Link 2</A>
<A HREF= http://www.nocheinlink.de/>Link 3</A>
<A HREF=‘page.html‘>Link 4</A>

<A HREF=“http://www.fehlerhaft.de> fehlerhafter Link</A>



<!-- Abschnitt 2: Areamaps -->

<IMG SRC=“bild.gif“ width=“10“ height=“10“ ismap usemap=“#map“
alt=”Bildbeschreibung”>
<MAP name=“map“>
   <AREA shape=“RECT“ coords=“0,0,50,50“ href=“link1.html“>
   <AREA shape=“RECT“ coords=“50,50,99,99“ href=“link2.html“>
</MAP>



<!—Abschnitt 3: Framesets -->

<FRAMESET framespacing=“0“ border=“false“ frameborder=“0“
cols=“10,*“>
     <FRAME name=“teil1“ src=“rand.html“ target=“_top“
scrolling=“auto“>
     <FRAME name=“teil2“ src=“inhalt.html“ scrolling=“auto“>
</ FRAMESET>



<!-- Abschnitt 4: Formulare -->

<FORM name=eintrag1 onsubmit=“link1.html“>
```

```
     <INPUT type=submit value=click>
</FORM>



<!-- Abschnitt 5: -->

<APPLET codebase="../„ code="applet.class" width="50"
height="50">
     <PARAM name=erstens value="link1.html">
     <PARAM name=zweitens value="link2.html">
</APPLET>


</BODY>
</HTML>
```

4.2.1 Textextraktion

Am einfachsten zu extrahieren ist der reine Text einer HTML-Datei. Es werden lediglich die Tags, also alles in spitzen Klammern, entfernt. Einige Besonderheiten müssen jedoch beachtet werden: Problematisch können beispielsweise Kommentare sein, die wiederum spitze Klammern enthalten. Man betrachte hierzu den ersten Kommentar des obigen Beispielcodes. Folglich ist es notwendig, zuerst die Kommentare zu entfernen. Außerdem müssen die Sonderzeichen (s. 2.2.1) in lesbare Zeichen umgewandelt werden.

4.2.2 Linkextraktion

Das Programm geht die zu untersuchende HTML-Datei schrittweise durch und extrahiert dabei die vorhandenen Links. Anhand welcher Merkmale erkannt wird, ob und um welche Links es sich handelt, wird nun im Einzelnen dargestellt.

4.2.2.1 Einfache Links

Die einfachste Art, Links in ein HTML-Dokument einzubauen, sind die <A>-Tags (Abschnitt 1 im Beispielcode). Der Parameter HREF gibt den URL an, auf den der Link zeigen soll; Der Text zwischen zwei <A>-Tags wird dann als Link auf diesen URL im Browser angezeigt. Es ist zu beachten, dass der URL mit doppelten, mit einfachen oder ganz ohne Anführungszeichen auftreten kann. Außerdem kann es vorkommen, dass Anführungszeichen am Anfang oder Ende vergessen werden.

4.2.2.2 Areamaps

Areamaps bieten die Möglichkeit, Links in Bilder zu integrieren, so dass bestimmte Bereiche eines Bildes als Links fungieren. Wie im Beispielcode ersichtlich (Abschnitt 2), wird mit dem <IMG>-Tag ein Bild angezeigt, dem mit dem usemap-Parameter eine Map (Karte) zugeordnet wird. In der Map sind dann mit <AREA>-Tags Bereiche spezifiziert, die in dem HREF-Parameter die einzelnen Links enthalten.

4.2.2.3 Framesets

Framesets bieten die Möglichkeit, das Browserfenster in verschiede Bereiche (z.B. Navigationsfenster und Hauptfenster) einzuteilen. Die URLs der einzelnen Bereiche sind im SRC-Parameter der <FRAME>-Tags zu finden (siehe Abschnitt 3).

4.2.2.4 Formulare

Formulare bieten normalerweise die Möglichkeit, Daten – wie z.B Name und E-Mailadresse - einzugeben und diese dann mit einem Submit-Button zu übertragen. Diese Möglichkeit wird auf Viren-Sites jedoch oft dazu mißbraucht, versteckte Links auf andere Seiten zu setzen. Dazu wird dem Submit-Button einfach ein Link als Parameter (Onsubmit-Parameter) übergeben, so dass der Button als Link dient (Abschnitt 4).

4.2.2.5 Java und Javascript

Um Java oder Javascriptcode komplett auszuwerten, müsste dieser ausgeführt werden, was nicht ganz ungefährlich ist, da dieser Code selbst maliziös sein könnte. Aus diesem Grund wird Java und Javascript nur eingeschränkt analysiert. So wird der Text nach Zeichenketten durchsucht, die in Anführungszeichen eingeschlossen sind und mit „.htm“ oder „.html“ enden. So werden auch Links gefunden, die sich im Javascript-Code befinden. Werden die Strings allerdings erst während der Ausführung des Codes generiert – etwa durch Zusammensetzen aus Teilstrings – ist dieses Verfahren machtlos.

Bei Java-Code ist es noch schwieriger, die Links zu extrahieren, da der Code in externe Dateien ausgelagert ist. Allerdings werden Links erkannt, die als Parameter an diese Programme übergeben werden (<PARAM>-Tag mit VALUE-Parameter, siehe Abschnitt 5 im Beispielcode).

Der im Code angegebene ONLOAD-Parameter im <BODY>-Tag ist eine sehr einfaches aber wirksames Javascript-Beispiel. Es bewirkt eine Weiterleitung zur angegebenen Seite „http:\\www.seite.de“.

4.2.2.6 Zusammenfassung der Linkarten

Um den Programmcode nicht unnötig mit Spezialfällen der Linkarten zu komplizieren, wurde eine möglichst allgemeine Lösung gesucht. In allen Fällen befinden sich die Links in den Parametern der einzelnen Tags. Es wurde also eine Liste mit Tags aufgestellt und jedem Tag ein Parameter zugeordnet, in dem jeweils die Links zu finden sind:

Tag	Parameter des Tags, in dem die Links zu finden sind
<A>	HREF
<AREA>	HREF
<FRAME>	SRC
<FORM>	ONSUBMIT
<BODY>	ONLOAD[9]
<FORM>	ACTION
<PARAM>	VALUE

Damit ist der Code besonders kurz, effektiv und zudem noch leicht erweiterbar.
Das Kernstück der DLL sieht demnach folgendermaßen aus:

```
...

case '>': //schließende spitze Klammer beim durchgehen erreicht

   //Hat es sich um das Ende eines Kommentare gehandelt?
   if (i>=2 && RetString[i-1]=='-' && RetString[i-2]=='-')
   {
    // Evtl. '--' am Ende wieder entfernen
    if (t>=2 && RetString[t-1]=='-' && RetString[t-2]=='-') t-=2;
    blnKommentar = false;
    lnTagcount = 0;
    break;
   }


    // Tags wie <BR> etc. umsetzen
    lnTagcount = 0;     // Achtung: Ersetzung muß kürzer
                        // oder gleichlang sein wie das Tag!!

    if (lngTagLen>0)    // Tag gespeichert ?
    {
```

[9] hier müsste zusätzlich noch der führende Text (`window.location=`) und die einfachen Anführungszeichen entfernt werden

```
 // In Großbuchstaben umwandeln
 _strupr(lcHilf1);

 // Links extrahieren
 char *begr;
 for (j=0;j< DEFINED_LINKTAGS;j++)
 {

   // In aLinkTags sind die relevanten Tags
   // In aLinkTagParams die zugehörigen Parameter

   if (_strnicmp(lcHilf1, aLinkTags[j],
         strlen(aLinkTags[j])) == 0)
   {
     begr = strstr( lcHilf1, aLinkTagParams[j]);
     if (begr != NULL)
     LinkExtract( RetString, RetString2, begr - lcHilf1
          + strlen(aLinkTagParams[j]));
   }
 }

// Meta-Tags extrahieren
if (_strnicmp( lcHilf1, "<META ", 6 ) == 0)
{
  begr = strstr( lcHilf1, "CONTENT=");
  if (begr != NULL)
  {
    long lngLaenge,indexAnfang = begr - lcHilf1 + 8;
    lngLaenge = Extract( lcHilf1 , indexAnfang );

    // String aus RetString holen (nicht aus  lcHilf1,
    // da dieser uppercase ist)

    strncpy(&RetString[t], &RetString[indexAnfang],
      lngLaenge );

    t += lngLaenge;

    RetString[t++] = 32; // Leerzeichen anhängen
    }
   }


   // Anderes Tag

   // Keine 4 aufeinanderfolgenden CR's zulassen
```

```
        if ((lcLastchar!=10) && (lnCrcount<3))
        {

        cou   = 0;
        lcHilf1[lngTagLen]=0; // Nullstring anhängen
        while (cou<DEFINED_TAGS)
        {
          if (!_stricmp(lcHilf1, aTags[cou]))  // Übereinstimmung
          {
            if (lcLastchar == 13)
              lcLastchar = 10;
            else
              lcLastchar = 13;

            lnCrcount++;
            RetString[t++] = 13; // Zeilenumbruch
            RetString[t++] = 10;

            cou = DEFINED_TAGS; // Schleife beenden, Tag gefunden
          }

          cou++;         // Nächsten String vergleichen

        }// endwhile


        // Tag ohne CR => leerzeichen
        if (cou != (DEFINED_TAGS+1) &&
          (!t || (RetString[t-1]!=' ')))
            RetString[t++] = 32; // Leerzeichen anhängen

        } //endif keine 4 aufeinanderfolgende CRs

      // Hilfsfeld neu initialisieren
      _strset( lcHilf1 , '#');

      lngTagLen = 0;
      lcOut = 9;
    } else lcOut=9;    // endif n>0

  break;
...
```

4.2.3 Tag-Extraktion

Die meisten Tags dienen der Formatierung des im HTML-Dokument enthaltenen Textes. Diese Tags müssen in der Regel nicht extrahiert werden. Es gibt aber

noch eine andere Art von Tags, auf die hier gesondert eingegangen wird: Metatags.

Metatags sind Tags, in denen im HTML-Dokument nicht sichtbare, Eigenschaften enthalten sind. Dies sind unter anderem Angaben zum Autor, sowie eine Kurzbeschreibung des Seiteninhaltes. Da davon ausgegangen werden kann, dass man Angaben zum Autor auf Virenseiten vermissen wird, sind besonders Metatags von Interesse, die Seitenbeschreibungen, Stichwörter u.ä. enthalten.

Weiterhin gibt es die Möglichkeit, mit dem ALT-Parameter Bildern einen Text zuzuordnen, der anstatt des Bildes angezeigt wird, falls dieses nicht geladen werden kann[10]. Dieser Text enthält häufig eine Beschreibung des Bildes.

4.2.4 Javascript-Analyse

Eine vollständige Analyse des Javascript-Code ist sehr aufwendig, bzw. fast unmöglich. Es müsste ein Simulator programmiert werden, der den Code ausführt und das Ergebnis analysiert. Dabei besteht das Risiko, dass der Javascript-Code selbst maliziös ist. Es könnten Endlosschleifen eingebaut sein, die den Computer zum Stillstand bringen. Es ist auch nicht möglich, vorher zu erkennen, ob ein Programm hält oder endlos läuft (Halteproblem [theo, S. 110]).

[10] Beim Internet Explorer wird dieser Text auch beim Überfahren des Bildes mit der Maus angezeigt

5 Ausblicke

Durch die entstandene DLL ist eine erhebliche Beschleunigung des Malware-Crawlers erzielt worden. Das Analysieren der HTML-Dokumente ist um Faktor 10-100 beschleunigt worden. Dennoch gibt es weitere Möglichkeiten, den Malware-Crawler zu verbessern.

5.1 Visualisierung der Gruppendependenzen in Graphen

Um gezielter nach Malware zu suchen, ist es vorteilhaft, mehr über die Netzstruktur von Webseiten, die maliziöse Software enthalten, zu erfahren. Dazu werden die vom MWC gecrawlten Webseiten als Graph dargestellt.

5.1.1 Vorhaben

Die vom Malware-Crawler gelieferten Daten in Form einer Textdatei werden eingelesen und analysiert. Auf den Aufbau und Inhalt dieser Datei und auf genauere Details wird später noch eingegangen. Die vom MWC besuchten URLs werden graphisch als Kreise dargestellt, die Verlinkung zwischen diesen Seiten als Pfeile zwischen den Kreisen. Außerdem wird noch der Heuristikwert, den der Malware-Crawler liefert, berücksichtigt. So werden die Kreise in verschiedenen Farbnuancen gezeichnet. Die entstehende Netzstruktur soll dann auf Gruppierungen in Bezug auf Farbe und Position untersucht werden.

5.1.2 Probleme

Das erste Problem bei der Darstellung der URLs auf dem Bildschirm ist die Position an der diese angezeigt werden. Diese könnte man z.B. aus dem URL-String generieren, so dass man auch die Domain-Endung (.de, .com usw.) in die Graphik einbezieht. Dies würde aber nur eine grobe Rasterung ergeben und hätte nicht den gewünschten Effekt der Visualisierung.

Die URLs sollen nicht nur einfach möglichst realistisch aufgrund des Standortes oder des Inhaltes dargestellt werden, wie es bei [MAP] der Fall ist. Vielmehr sollen die Webseiten nach einem bestimmten Algorithmus geordnet werden, der die Beziehungen der URLs untereinander berücksichtigt. So sollen URLs, die stärker miteinander verlinkt sind, näher beieinander stehen als solche, die nicht oder nur kaum miteinander verlinkt sind. Es soll also von der geographischen Position der Seite abstrahiert werden.

Des Weiteren sollten die Seiten einer Website nicht einzeln dargestellt werden, sondern als Ganzes. Das wirft das Problem auf, zu entscheiden, ob zwei Seiten zu derselben Website gehören oder nicht.[11]

[11] Vgl. Beispiel Kapitel 5.1.7

Ein weiteres Problem entsteht durch die große Datenmenge. Da der Malware-Crawler über einen längeren Zeitraum Daten sammelt und URLs verfolgt, fällt eine nicht unerhebliche Datenflut an. Diese Daten werden außerdem nicht besonders komprimiert, da dies zu weiteren Performanceeinbußen seitens des Malware-Crawlers sowie des Analyseprogrammes führen würde.

5.1.3 Prognose

Zu erwarten sind Formatierungen einzelner URLs zu zusammenhängenden Gruppen, die untereinander mehr oder weniger stark verlinkt sind. Eventuell ergibt sich eine farbliche Aufteilung der Gruppen. Hierbei wären Gruppen von URLs mit sehr hohem Heuristikwert besonders zu betrachten, da diese einen ersten Anhaltspunkt zur weiteren Optimierung des Malware-Crawlers und zur weiteren Analyse liefern.

5.1.4 Darstellung

Die graphische Darstellung der Webseiten wird durch farbige Punkte (Kreise) auf dem Bildschirm realisiert. Jeder Punkt repräsentiert eine Webseite. Die Links zwischen diesen Webseiten werden als Pfeile zwischen den Punkten dargestellt. Die Farben der Punkte stellen den Heuristikwert dar. Die Farbskala reicht von Grün (geringer Heuristikwert) über Gelb (= mittlerer Heuristikwert) bis Rot (= hoher Heuristikwert), wobei die Farben fließend ineinander übergehen.

Da schon Webseiten mit relativ geringem Heuristikwert Viren enthalten, sollte die Farbverteilung ggf. nicht linear sein[12]. Man kann im Programm den Schwellwert für relevante Webseiten eingeben, an der sich dann die Farbgebung der Punkte orientiert. Der Heuristikwert reicht von 0 (wahrscheinlich Virenfrei) bis 9999 (enthalten sehr wahrscheinlich Viren). Bei einem Schwellwert von x stuft das Programm nun die Farben folgendermaßen ab:

0 bis x: Grün bis Gelb
x bis 9999: Gelb bis Rot

Bei einem Grenzwert von 100 sähe die Farbverteilung z.B. so aus:

Abbildung 7 - Farbskala

Es folgt eine genau Beschreibung über den Aufbau und Inhalt der Daten, die vom Malware-Crawler geliefert werden. Dann werden die einzelnen Schritte vom

[12] z.B. bietet sich eine logarithmische Farbskala an

Einlesen der Daten über das Umsortieren der einzelnen Punkte, welche die Websites repräsentieren, bis zur endgültigen Anzeige erläutert.

5.1.5 Daten des MalwareCrawlers

Die Daten werden in einer Textdatei vom Malware-Crawler geliefert. Diese Textdatei ist in einzelne Blöcke eingeteilt, wobei jeder Block einem Link entspricht. Die dazugehörigen URLs ergeben sich automatisch aus den Start- und Zielpunkten der Links. Der genaue Inhalt und Aufbau der einzelnen Blöcke sieht wie folgt aus:

Größe	Inhalt	Bemerkung
254 Bytes	URL	aufgefüllt mit ASCII 20h
254 Bytes	Start-URL (Von-URL)	aufgefüllt mit ASCII 20h
4 Bytes	Heuristikwert	Dezimalziffern als ASCII Zeichen (0-9999)
10 Bytes	Datum	TT.MM.JJJJ (mit Trennzeichen)
10 Bytes	Scandatum	TT.MM.JJJJ (mit Trennzeichen)
10 Bytes	Checksumme	CRC
10 Bytes	Entfernung zum Ursprungs-Suchknoten in Links	
254 Bytes	reserviert für spätere Erweiterungen	
2 Bytes	Zeilenumbruch	CHR(13)+CHR(10)

Gesamtgröße eines Blocks: 808 Bytes

5.1.6 Interne Verwaltung der Daten

Die eingelesenen Daten werden intern als verkettete Objekte gespeichert. Jedes Objekt hat folgende Eigenschaften:

- URL (String, inklusive der auffüllenden ASCII 20h)
- Heuristikwert (long, -1 für keinen)
- eingehende Links (Liste)
- ausgehende Links (Liste)
- die anderen Daten (Datum, Checksumme usw.) werden noch nicht benutzt und daher auch nicht gespeichert.

Die Einträge der Liste mit den ausgehenden Links sind nicht direkt aus den Daten, die vom Malware-Crawler kommen, ablesbar. Vielmehr wird der URL beim Eintragen der eingehenden Links gleichzeitig in dessen Objekt als ausgehender Link eingetragen. Diese Redundanz ist zur schnelleren Berechnung nötig, da so einzelne Pfade besser verfolgt werden können.

Die –1 als Heuristikwert stellt einen Spezialfall dar, der durchaus vorkommen kann. Dies kann man sich bereits anhand des einfachen Beispiels mit nur einem

Block vor Augen führen: Zuerst wird ein Objekt erstellt und die eingelesene URL und der Heuristikwert eingetragen. Weiterhin wird die „Von-URL“ in die Liste der eingehenden Links gesetzt. Dann wird allerdings ein *zweites* Objekt erstellt, und zwar mit der „Von-URL“. Hierfür ist dann kein Heuristikwert vorhanden. Punkte ohne Heuristikwert werden in der Grafik weiß dargestellt.

5.1.7 Zusammenfassen von Seiten zu einer Webseite

Die Grundidee ist, die große Datenmenge zu komprimieren. Dies geschieht, indem man Seiten (z.B. www.webseite.de/eins.html und www.webseite.de/verzeichnis1/zwei.html), die zur gleichen Webseite (in diesem Fall also www.webseite.de) gehören, zusammenzufassen und als einen Punkt darzustellen. Es wäre kein Problem, die URL nach dem ersten Schrägstrich (/) bzw. Backslash (\ ) abzuschneiden. Probleme gibt es allerdings bei Freehostern, bei denen die einzelnen Webseiten der Mitglieder lediglich in verschiedenen Unterverzeichnissen liegen. Diese Webseiten würden dann alle zu einer zusammengefasst werden, z.B. würden die beiden voneinander unabhängigen Webseiten www.geocities.com/user/index.html und www.geocities.com/AnOtherUser/index.html als eine, nämlich www.geocities.com, dargestellt. Eine Trennung ist aber unbedingt notwendig, da es durchaus (wahrscheinlich überwiegend) harmlose Webseiten bei Freehostern gibt, die nichts mit Viren zu tun haben und auch nicht mit Malware in Verbindung gebracht werden möchten. Es ist daher nötig, eine Liste zu erstellen, die alle URLs enthält, die erst nach dem zweiten (evtl. auch dritten usw.) Schrägstrich bzw. Backslash trennen.

5.1.8 Positionsberechnung der Punkte

Definition:
Eine *Verbindung* ist ein eingehender oder ausgehender Link

Um eine Ausgangsbasis zu haben, wird zunächst allen Punkten eine Zufallsposition zugeteilt. Dann werden alle Punkte der Reihe nach durchgegangen und jedem, abhängig von Verbindungen zu anderen Punkten, eine neue Position zugeteilt. Da andere Punkte, von denen ein Punkt abhängt, in der späteren Berechnung ihre Position ebenfalls ändern können, sind mehrere Durchläufe nötig, um die endgültige Position zu bestimmen (Ausnahme: Punkte mit genau einer Verbindung, siehe unten).

5.1.8.1 Punkte mit genau einer Verbindung

Zuerst werden alle Punkte mit genau einer Verbindung betrachtet. Diese werden an den Punkt, mit dem sie verbunden sind „herangezogen“ (d.h. die Richtung bleibt und die Entfernung wird auf einen festgelegten Wert reduziert). Es ist

offensichtlich, dass dieser Schritt als letztes ausgeführt wird, da sich die Position des verbundenen Punktes bei weiteren Berechnungen ändern könnte. Außerdem können Punkte mit nur einer Verbindung bei vorhergehenden Betrachtungen außer acht gelassen werden.

Kommen wir jetzt zur genauen Berechnung der neuen Position eines Punkte mit genau einer Verbindung:

Seien Ax und Ay die Koordinaten des Punktes (im folgenden A genannt) mit nur einer Verbindung und der Punkt (im folgenden B genannt), mit dem dieser Punkt verbunden ist, sei zur einfacheren Berechnung der Ursprung. Die Hin- und Rücktransformation ist trivial.
Sei (x ; y) die neue Position von A.

Die gewünschte Entfernung der beiden Punkte sei z.
Die Ausgangsentfernung d berechnet sich wie folgt:

$d^2 = Ax^2 + Ay^2$

Es soll also $z^2 = x^2 + y^2$ gelten.

Außerdem soll die Richtung des Punktes A zu B nicht verändert werden, d.h. (x;y) muß auf der Geraden AB (=g) liegen.

g= v A , mit v als Unbekannte

Also bekommen wir die drei Gleichungen:

$x = v\,Ax$ (1)
$y = v\,Ay$ (2)
$z^2 = x^2 + y^2$ (3)

Aus (1) folgt $x^2 = v^2\,Ax^2$ und aus (2) $y^2 = v^2\,Ay^2$.
Also ist

$z^2 = v^2\,Ax^2 + v^2\,Ay^2 = v^2\,(Ax^2 + Ay^2)$

$$\Rightarrow v = \sqrt{\frac{z^2}{Ax^2 + Ay^2}} = z\sqrt{\frac{1}{Ax^2 + Ay^2}}$$

Die vollständige Berechnung im Code (mit Transformation) sieht demnach folgendermaßen aus (mit verkürzten Variablennamen und ohne Typumwandlungen) :

```
Ax = Ax - Bx
Ay = Ay - By
v = sqrt( z * z / (Bx * Bx + By * By) );
Ax = Bx +v*Ax;
Ay = By +v*Ay;
```

5.1.8.2 Punkte mit genau zwei oder mehr Verbindungen

Bereits bei Punkten mit genau zwei Verbindungen gestaltet sich das Umsortieren schwieriger. Es ist nicht möglich, den Punkt einfach in die Mitte der beiden Punkte, mit denen er verbunden ist, zu setzen. Dies wird am Beispiel eines Dreiecks (drei Punkte, bei denen jeder Punkt mit den anderen beiden verbunden ist) deutlich:

Zuerst wird ein Punkt (Punkt 1) in die Mitte der beiden anderen (Punkte 2 und 3) verschoben. Es entsteht eine Linie.

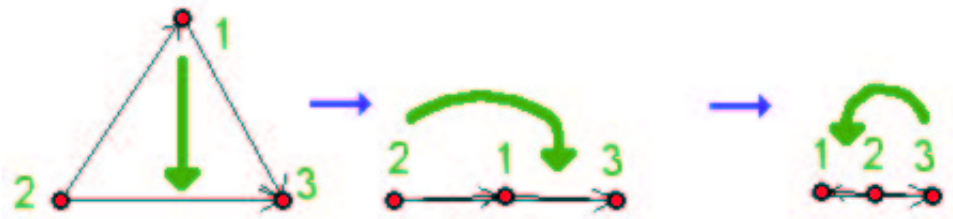

Abbildung 8 - Umsortierung eines Dreiecks

Bei jedem Durchgang näherten sich die Punkte weiter an und konvergierten schließlich gegen einen Punkt.

Man kann dies auf zwei Arten vermeiden: Erstens ist es möglich, lediglich einen Durchlauf auszuführen, was aber, je nach Reihenfolge der Punkte, in unterschiedlichen Resultaten enden könnte.

Zweitens könnte man die Prozedur bei einem Mindestabstand der Punkte beenden. Dies ist wohl die sinnvollere Methode, aber auch die etwas kompliziertere in der Umsetzung.

Schon diese erste Sortierung bringt eine erhebliche Ordnung in die anfangs zufällig verteilten Punkte, wie die folgenden Abbildungen zeigen:

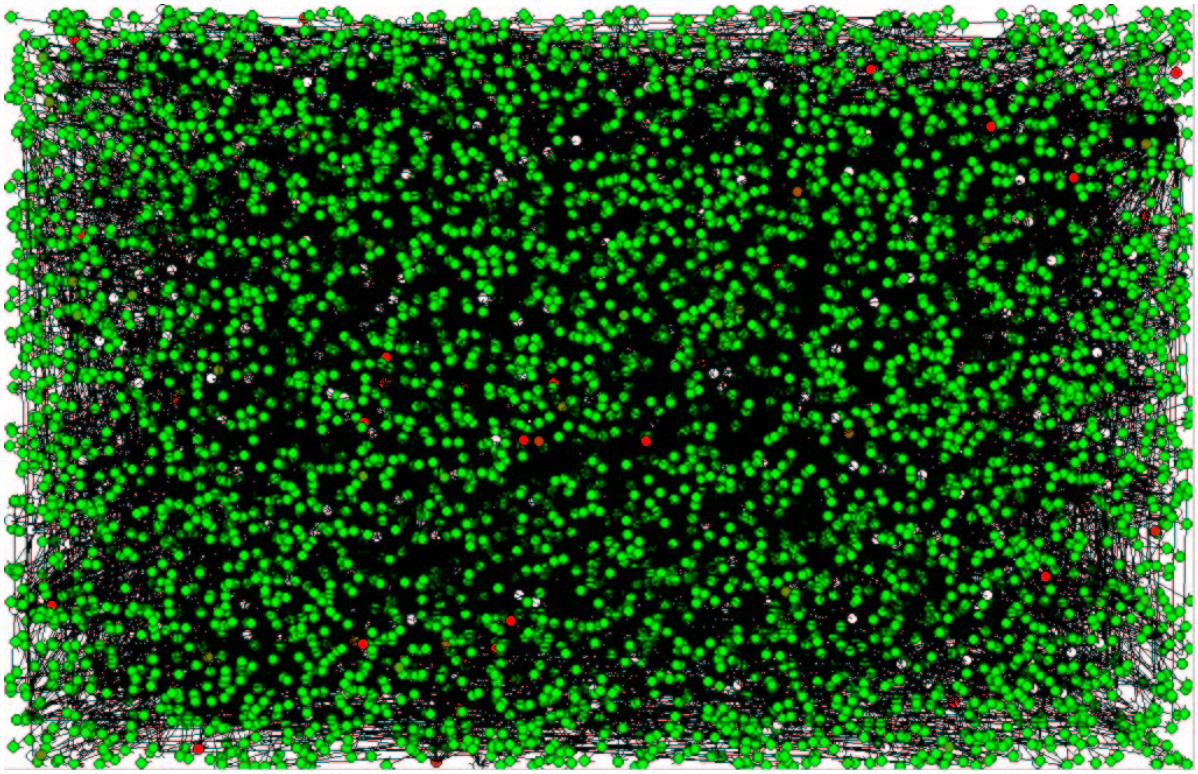

Abbildung 9 - Zufällige Verteilung der Punkte

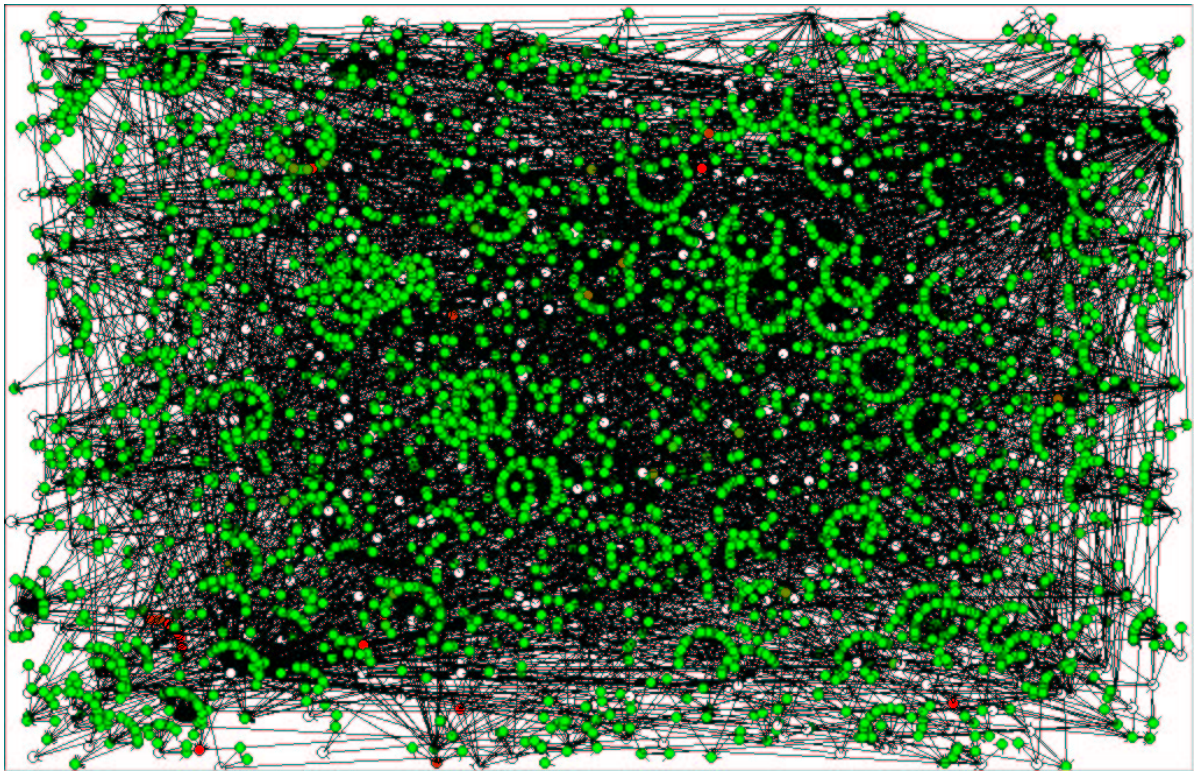

Abbildung 10 - Eine erste Sortierung der Punkte

Dies ist jedoch nur ein erster kleiner Ansatz und es wäre möglich, komplexere Algorithmen für das Ordnen von Punkten mit zwei oder mehr Verbindungen zu entwickeln, bzw. einen vollkommen anderen Ansatz für die Berechnung zu wählen. Es könnte auch die Richtung der Links oder der Heuristikwert mit in die Analyse einbezogen werden.

5.2 Agenten bei Providern

Eine andere Möglichkeit zum Optimieren des Crawlens wäre, diese Aufgabe auf mehrere Agenten zu verteilen.

Definition:
Mobile Agenten sind autonome Programme, die sich innerhalb von Rechnernetzen bewegen und dabei im Auftrag eines Benutzers Dienste verrichten. [Agents]

Es könnten Agenten bei Providern eingesetzt werden, die eigenständig nach Webseiten mit maliziösem Code suchen und ihre Ergebnisse an den Malware-Crawler liefern. Dieser könnte dann mit dem Download und der Überprüfung der Dateien auf den entsprechenden Webseiten beginnen, wobei auch dies schon von den Agenten erledigt werden könnte. Außerdem wäre es vorstellbar, dass die Agenten untereinander kommunizieren und ihre (Teil-) Ergebnisse austauschen.

5.3 Macro-Programmierbarkeit des Crawlers (Metasprache für Crawleralgorithmen)

Es ist denkbar, den Malware-Crawler dahingehend zu modifizieren, dass nicht ein festgelegter Algorithmus ausgeführt wird, um die Links zu verfolgen, sondern dies variabel zu gestalten. So könnte man das Crawlen von einem in einer Metasprache geschriebenen Programm abhängig machen. Diese Metasprache könnte etwa Befehle enthalten, die auf die Datenbank mit relevanten Wörtern zugreifen und mit denen aus dem HTML-Dokument extrahierten in Beziehung setzen kann.

So wäre es sehr einfach, schnell und unkompliziert die Heuristik zu ändern. Anstatt den Malware-Crawler-Code zu ändern, bräuchte man nur ein in der Metasprache geschriebenes Programm einzuspielen.

Anhang A: Quellenverzeichnis

[Agents] http://aifbpontus.aifb.uni-karlsruhe.de/telepraktikum/Ws0001/Thema1a/Mobility/text.html

[AGN] http://agn-www.informatik.uni-hamburg.de/
Webseite des Fachbereichs AGN

[Dud5] Band 5 Duden, Fremdwörterbuch, 6. Aufl. 1997 Mannheim; Wien; Zürich. Dudenverlag 1997

[MWC-00] Sönke Freitag: Webbasiertes Auffinden maliziöser Software mit fortschrittlichen heuristischen Verfahren, Diplomarbeit Universität Hamburg, Fachbereich AGN / Informatik, 7/2000

[Google] http://www.google.com/
Internetsuchmaschine

[Inf] Taschenbuch der Informatik, Uwe Schneider / Dieter Werner; 3. Aufl. 2000; Fachbuchverlag Leipzig im Carl Heiser Verlag, München Wien

[MAP] http://maps.map.net/
Grafische Darstellung von Webseiten anhand ihrer inhaltlichen und lokalen Relevanz

[SelfHTML] http://www.netzwelt.com/selfhtml/
Einführung in die Erstellung und den Aufbau von HTML-Dateien

[Spertus98] http://www.mills.edu/ACAD_INFO/MCS/SPERTUS/Thesis/thesis.html
ParaSite: Mining the Structural Information on the World-Wide Web

[theo] Theoretische Informatik – Grundlagen und praktische Anwendungen; Werner Brecht; Vieweg 1995

[W3C] www.w3.org World Wide Web Consortium

Eidesstattliche Erklärung

Ich versichere, die vorliegende Studienarbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Hamburg, den

_____
René Soller