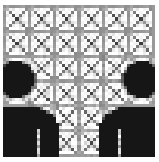
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich AGN



Baccalaureatsarbeit

Analyse und Weiterentwicklung der Testbed-Erstellung im aVTC-Projekt

Stefan Heimann

April 2003

Betreuer: Prof. Dr. Klaus Brunnstein

Inhaltsverzeichnis

1. Einleitung

Diese Baccalaureatsarbeit entstand im Rahmen des Anti-Virus Test Center[1] (aVTC-) Projektes des Arbeitsbereiches "Anwendungen der Informatik in Geistes- und Naturwissenschaften" (AGN) am Fachbereich Informatik der Universität Hamburg. Ziel der Arbeit ist die Analyse der bestehenden Verfahren im aVTC zur Generierung von Testbeds, die Erarbeitung eines Konzeptes zur Verbesserung jener und daraus resultierend die Erstellung eines Programms, welches dieses Konzept implementiert.

Kapitel 1 ist eine Einführung in das Forschungsgebiet Malware. Hier werden die wichtigsten Begriffe, die in dieser Arbeit verwendet werden, definiert. Ebenso wird die Arbeit des aVTC kurz vorgestellt und das CARO-Namensschema erläutert.

Im 2. Kapitel wird das aktuelle Verfahren zur Erstellung von Testbeds anhand des Heureka 3 Tests analysiert. Hierzu wird die Struktur der Testbeds gezeigt, der generelle Ablauf der Erstellung erklärt, sowie die dabei entstehenden Probleme aufgezeigt.

Das 3. Kapitel stellt das neue Konzept zur Verbesserung der aVTC Testbed Erstellung und daraus abgeleitet die Anforderungen an ein neues Programm zur Umsetzung dieser vor. Dafür wurde ein eigenes Schema speziell zur Identifikation von Malware-Samples entwickelt, welches durch die verschiedenen Module des Programms, genannt aVTC-Testbed-Tool, verwendet werden wird. Gleichermaßen werden in diesem Kapitel die Funktionen der einzelnen Module selbst vorgestellt.

Kapitel 4 liefert die Dokumentation zum aVTC-Testbed-Tool. Es werden hier die Parametrisierung, die erzeugten Dateien bzw. Listen und die Funktionsweise erklärt.

Das letzte Kapitel 5 bietet einen kurzen Ausblick auf die weitere Verwendung des Tools in Hinblick auf eine nachfolgende Diplomarbeit.

1.1. Definitionen

Es folgen nun einige Definitionen zu den wichtigsten Begriffen, die in dieser Arbeit verwendet werden.

[1] früher auch bekannt als "Virus Test Center" (VTC)

1.1.1. allgemeine Definitionen

Die Begriffe, die in diesem Abschnitt definiert werden, sind allgemein gebräuchlich, wenn es um das Thema Malware (Definition s.u.) geht. Allerdings werden diese Begriffe von verschiedenen Personen teilweise sehr unterschiedlich definiert.

Malware Der Begriff Malware ist die Kurzschreibweise für "malicious software" und beschreibt die gesamte Klasse an Software, die als schädlich einzustufen ist.

Software wird als Malware bezeichnet, wenn sie intentional dysfunktional ist oder hinreichende Beweise, zum Beispiel durch die Beobachtung des Verhaltens zur Laufzeit, vorliegen, daß die Software die Benutzung oder das Verhalten von Nicht-Malware absichtlich negativ beeinflussen kann.

Intentional dysfunktional nennt man Software, deren tatsächliche Gesamtfunktionalität mit Absicht von der formal oder auch informal spezifizierten Gesamtfunktionalität abweicht, also mindestens eine verborgene und damit meist ungewollte Funktionalität enthält.

Virus Ein Virus[2] oder genauer Computervirus ist eine spezielle Form von Malware, die sich mindestens dreimal durch die Ausführung jeweils eines virusfremden, ausführbaren Objektes einer bestimmten Plattform auf einem Einzelsystem repliziert.

Ein Computervirus infiziert also ausführbare Objekte und repliziert sich dadurch selbst auf Einzelsystemen, indem er sich selbst in irgendeiner Form in ein ausführbares Objekt, auch "Wirt" genannt, einfügt und dieses so modifiziert, daß bei dessen Ausführung auch der Virus ausgeführt wird. Die Replikation des Virus kann daher das Verhalten oder die Benutzung der ursprünglichen Software, nicht zuletzt durch den unnützen Ver- oder Gebrauch von Betriebsmitteln, negativ beeinflussen. Zusätzlich kann ein Virus eine Schadfunktion enthalten, die unter bestimmten Bedingungen ausgeführt wird.

In-The-Wild (ITW) Die Menge an bekannten Viren ist mittlerweile sehr groß und vielfältig geworden, jedoch nur ein verhältnismäßig kleiner Teil davon ist wirklich weit verbreitet. Einige Viren befinden sich nur noch in Malware-Datenbanken von Anti-Malware-Labors oder wurden vom Viren-Autor (noch) nicht verbreitet. Die Wildlist Organisation gibt daher für jeden Monat Listen von bekannten, verbreiteten Viren, die für diesen Monat "In-The-Wild" oder in "freier Wildnis" vorkommen, heraus, um diese Viren als akute Bedrohung, und daher als besonders wichtig, von den "Zoo-Viren", welche kaum verbreitet sind, abzugrenzen. Manche Anti-Malware-Hersteller haben sich darauf spezialisiert, ITW-Viren zu erkennen. Man sollte jedoch beachten, daß viele Viren bereits lange vorher bekannt ("In-Zoo") sind, bevor sie "In-The-Wild" vorkommen.

[2] Häufig wird der Begriff "Virus" oder "Viren" auch stellvertretend für den Begriff "Malware" benutzt, da es sich um die geläufigste Art von Malware handelt.

Anti-Malware-Produkte Anti-Malware-Produkte[3] sind Programme, die Daten oder andere Software vor jeglicher Art von Malware schützen sollen. Dies soll primär durch das Erkennen, Blockieren und Entfernen von Malware erreicht werden, wobei nicht jedes Anti-Malware-Produkt jede Art von Malware erkennen kann.

Scanner Scanner sind ein wichtiger Bestandteil von Anti-Malware-Produkten, der hauptsächlich für die Erkennung von Malware zuständig ist. Die Erkennung erfolgt anhand von Malware-Signaturen, die an die Scan-Engine angepaßt sind. Zusätzlich werden heuristische Verfahren eingesetzt, um bestimmte Malware und deren Abarten generisch erkennen zu können. Grundsätzlich sind zwei Arten von Scannern zu unterscheiden: "On-Demand-" und "On-Access-Scanner".

On-Demand-Scanner werden, wie der Name schon sagt, bei Bedarf per Hand oder periodisch per Skript benutzt, um Objekte auf Malware zu prüfen. Ein solcher Scanner kann auch eingesetzt werden, um alle Objekte, z.B. Dateien, auf Malware zu untersuchen. Hauptsächlich diese Art von Scannern wird im aVTC-Labor getestet (s. 1.2).

On-Access-Scanner laufen die ganze Zeit im Hintergrund und sollen Objekte beim Zugriff scannen und wenn nötig blockieren.

Sample Samples[4] sind Objekte, die Malware oder maliziösen Code enthalten, und stellen eine Instanz bzw. ein Exemplar bestimmter Malware dar.

Bei Viren handelt es dabei um infizierte Objekte, bei anderer Malware größtenteils um das maliziöse Programm selbst.

Testbed Ein Testbed oder eine Testmenge ist eine Menge von Samples oder Malware-Objekten, auf der Anti-Malware-Produkte getestet werden.

Testbeds sind meistens Dateisysteme, die auf die unterschiedlichsten Arten strukturiert sein können[5]. Allgemein gilt: je umfangreicher ein Testbed, desto aussagekräftiger ist der Test. Umfangreich bedeutet in diesem Zusammenhang auch vielfältig, denn die Anti-Malware-Produkte sollen möglichst in allen Malware-Kategorien[6] alle möglichen Instanzen bestimmter Malware erkennen können.

In den folgenden Kapiteln bezeichnet der Begriff "Testbed" ein einzelnes Testbed, das eine Teilmenge der gesamten Testmenge darstellt, und enthält Malware einer bestimmten Malware-Kategorie. Der Begriff "Gesamt-Testbed" steht für die gesamte Testmenge eines Tests.

[3] Oft wird auch der Begriff "Anti-Viren-Programme" benutzt
[4] dt.: Musterdateien
[5] Eine genauere Erläuterung der Testbed-Struktur des aVTC befindet sich in Kapitel 2
[6] siehe auch Definitionen in 1.1.2

1.1.2. spezielle Definitionen in dieser Arbeit

Die Definitionen dieses Abschnittes werden hauptsächlich vom Autor im Rahmen dieser Arbeit benutzt. Dabei handelt es sich hauptsächlich um die Unterteilung und Klassifizierung von Malware.

Malware-Typ Malware läßt sich aufgrund der verschiedenen grundsätzlichen Plattformen, auf denen sie ausgeführt werden kann, in verschiedene Typen unterteilen.

"Boot"-Malware wird über die Boot-Sektoren von Festplatten und Disketten ausgeführt. Daher handelt es sich hier fast ausschließlich um Viren.

"File"-Malware ist vor allem vom Betriebssystem und dessen Dateisystem abhängig. Sie benutzt die ausführbaren Dateien des entsprechenden Betriebsystems, um ihre schädliche Wirkung zu entfalten.

"Macro"-Malware ist in einer bestimmten Makrosprache geschrieben oder infiziert Dokumente, die Makros enthalten können. Makrosprachen bieten die Möglichkeit, in Dokumenten zusätzlich für die Anwendung ausführbaren Code, genannt Makro, einzubetten, der dann manuell oder automatisch ausgeführt werden kann. Dieser ausführbare Makro-Code kann nicht nur den Inhalt von Dokumenten der Anwendung verändern, sondern meist auch auf Betriebssystemfunktionen zurückgreifen. Macro-Malware ist betriebssystemunabhängig, da sie für jede Anwendung, die die entsprechende Makrosprache interpretieren kann, ausführbar ist.

"Script"-Malware nutzt eine spezielle Skriptsprache. Ein Skript ist nicht eingebettet in ein Dokument, wie ein Makro, muß aber ebenso interpretiert werden und liegt nicht in kompiliertem Byte-Code vor. Auch Skripte können fast immer Funktionen des Betriebsystems ausführen.

Es gibt auch Malware, die eigentlich zu mehr als einem Typ gehört, beispielsweise Viren, die sowohl Dateien als auch Boot-Sektoren infizieren. Diese Viren nennt man "Multi-Partite-Viren".

Malware-Klasse Unabhängig von dem Malware-Typ kann Malware nach Replikation und Systemart in Klassen unterteilt werden.

Replikation bezeichnet die Fähigkeit von Malware, sich selbst zu replizieren. Viren zum Beispiel sind selbstreplizierend, Trojaner[7] hingegen sind nicht selbstreplizierend.

Die Systemart gibt unabhängig von der Replikation an, ob Malware in Einzelsystemen oder vernetzten Systemen ausgeführt werden kann. Zur Malware, die auf Einzelsystemen ausgeführt werden kann, gehören beispielsweise die Klassen Virus und Trojaner, während die Klassen Wurm und Hostile Applet[8] ein Netzwerk benötigen, um wie gedacht zu funktionieren.

[7] auch als "Trojanisches Pferd" bekannt

[8] dt.: feindlicher Agent : nutzt die Möglichkeit der Einbettung von ausführbarem Inhalt in Web-Seiten

Weitere Informationen zur Klassifizierung von Malware und Malware-Klassen sind bei [Bontchev 1998] und [Kittel 2000] zu finden.

Malware-Plattform Dieser Begriff hat einen direkten Bezug zum Begriff "Malware-Typ". Während der Begriff Malware-Typ die grundsätzliche Ausführungsplattform bezeichnet, gibt der Begriff Malware-Plattform die konkrete Ausführungsplattform einer bestimmten Malware an. Von der Malware-Plattform läßt sich ohne weiteres auf den Malware-Typ schließen.

Ein Virus der in der Makrosprache für Word 97 verfaßt wurde gehört demnach zum Malware-Typ "Macro" und wird der Malware-Plattform "W97M" zugeteilt.

Malware-Kategorie Dieser Begriff bezeichnet nach bestimmten Kriterien, wie Malware-Typ und Malware-Klasse, eine Teilmenge der gesamten Malware und wird zur Einteilung von einzelnen Testbeds benötigt. So beschreibt beim aVTC die Kategorie "Script-ITW" Script-Viren und -Würmer, die für den Zeitrahmen des Tests In-The-Wild waren.

"Nonreplikativ"-Malware Mit nonreplikativer Malware wird in dieser Arbeit Malware bezeichnet, die nicht zur Klasse der Viren und Würmer gehört, also nicht replizierend ist. Der Grund dafür ist die Verwechslungsgefahr mit dem Begriff "Malware" in Zusammenhang mit Testbeds des aVTC (siehe auch 2.1).

Malware-Familie Die Malware-Familie ist die Bezeichnung für bestimmte Malware, wie zum Beispiel "Loveletter" für alle Varianten des Loveletter-Virus. Die Bezeichnung kann beliebig gewählt werden, sollte jedoch nicht dem Wunsch des Malware-Autors entsprechen, und darf nicht auf eine reale Person, Firma oder ähnliches hinweisen.

Variante Die Variante kennzeichnet eine Untergruppe einer Familie. Varianten werden fortlaufend alphabetisch mit bis zu zwei Buchstaben bezeichnet. Die erste Variante einer Familie hat immer die Variantenkennung "A", wird daher nicht immer mit angegeben. Für jede neue Variante wird eine noch nicht benutzte Variantenkennung vergeben. Dabei sind die Variantenkennungen nicht umbedingt chronologisch sortiert, da es vorkommen kann, daß eine falsch vergebene Variantenkennung wieder frei wird, nachdem bereits weitere vergeben wurden. Bei der Erkennung von Malware kann als Variante "GEN" angegeben sein, wenn die Variante sich aufgrund von generischer Erkennung nicht exakt bestimmen läßt.

infektiöse Länge Besonders bei File- und Boot-Viren ist es zum Teil üblich, als besonderes Charakteristikum innerhalb einer Familie die Länge des infektiösen Teils zusätzlich mit anzugeben. Der infektiöse Teil ist der Teil, welcher für die Replikation verantwortlich ist. Anhand dessen kann entschieden werden, ob und welcher Variante Malware innerhalb einer Familie zuzuordnen ist. Stimmt eine bestimmte Malware in keinem Fall zu mindestens 60% im infektiösen Teil mit bereits existierenden Varianten überein, stellt diese Malware eine neue Variante innerhalb der Familie dar.

(D-)Evolution Hierbei handelt es sich wiederum um eine Untergruppe einer Variante, die durch einen Programmierfehler neu aus einer Variante entstanden ist, aber genau deswegen keine eigene Variante darstellt. Dies kann bei replizierender Malware auftreten.

Sprachkennung Makrosprachen wurden früher zum Teil, gerade weil sie für Anwender gedacht sind, in die entsprechende Landessprache übersetzt. Das hatte im Negativen wie im Positiven zur Folge, daß Makros, die in einer bestimmten Landessprache geschrieben wurden, zu einem Interpreter für Makros einer anderen Landessprache inkompatibel waren.

Die Sprachkennung "en" für Englisch ist Standard und wird nicht angegeben.

Malware-Identifikation Hierbei handelt es sich um die "Gesamt-Identifikation" bestimmter Malware bzw. ihrer Samples und setzt sich aus einzelnen "Identifikationsteilen" oder Teildaten zusammen. Eine gängige Identifikation bei der Erkennung von Macro- oder Script-Malware besteht aus den Identifikationsteilen :

- Malware-Plattform
- Malware-Familie
- Variante

und möglicherweise dem Zusatz "@MM" für Mass-Mailer (siehe dazu auch die Namenskonvention von CARO).

Man kann weitere Informationen, wie zum Beispiel die Malware-Klasse, als Identifikationsteile in die Gesamt-Identifikation mit aufnehmen (siehe eigenes Namensschema in Kapitel 3).

Die Scanner und Hilfs-Tools, die bei der Erstellung der Testbeds zur Identifizierung der Samples zum Einsatz kommen, werden in dieser Arbeit allgemein als "Identifikatoren" bezeichnet.

Zero-Byte-Files Damit werden in dieser Arbeit Dateien bezeichnet, welche die Dateilänge null besitzen, demnach also keine Daten enthalten. Näheres dazu unter 2.3.

Datenbasis Unter einer Datenbasis[9] ist in dieser Arbeit eine recht einfach gehaltene Form der Datenhaltung gemeint, die sich vor allem durch das Fehlen eines Verwaltungssystems auszeichnet. Diese kann durch Text-Dateien wie Listen oder hierarchisch auch durch Dateisysteme repräsentiert sein. Eine Sammlung von Kollektionen oder eine bzw. mehrere Textdateien, in denen Daten in einer bestimmten Form enthalten sind, kann in diesem Zusammenhang als Datenbasis bezeichnet werden. Dieser Begriff soll der Abgrenzung zu dem Begriff "Testbed" einerseits und dem Begriff "Datenbank" andererseits dienen. Mit dem Begriff "Datenbank" soll ein "echtes" Datenbank-System (DBS), zum Beispiel eine Relationale Datenbank, mit einer Datenbank (DB) und einem Datenbank-Verwaltungssystem (DBVS) gemeint sein.

[9] nicht zu verwechseln mit der Bedeutung des englischen Wortes "database"

In dieser Arbeit ist mit dem Begriff "Testbed-Datenbasis" das Dateisystem mit den entpackten Kollektionen gemeint, welches die Samples, die in die verschiedenen Testbeds einsortiert werden sollen, enthält.

1.2. Der aVTC-Test

Das Anti-Virus Test Center (aVTC) testet unter der Leitung von Prof. Brunnstein Anti-Malware-Produkte auf ihre maximal mögliche Erkennungsrate von Malware, die in Form von Samples in verschiedenen Testbeds einsortiert ist. Dazu senden Anti-Malware-Hersteller, die am Test teilnehmen wollen, dem aVTC ihre Produkte zu einem bestimmten Zeitpunkt zu, und teilen die optimalen Parameter und Konfigurationseinstellungen für die maximal mögliche Erkennung von Malware mit. Der Test findet ebenso unter verschiedenen Betriebssystemen statt, für die die Anti-Malware-Hersteller ihre Produkte zusenden können.

1.2.1. Der Heureka-Test

Die Heureka-Tests sind relativ neue Tests des aVTC und sollen über die Fähigkeit der Anti-Malware-Produkte, zur Zeit ihrer Einreichung unbekannte, also neue Malware, erkennen zu können, Auskunft geben. Auch wenn der Test "Heureka-Test" genannt wird, handelt es sich hier allerdings nicht um einen Test, der einzig die Qualität der heuristischen Methoden der Produkte ermitteln soll, denn in diesem Fall müßte man die Produkte so konfigurieren, daß sie nur ihre Heuristik einsetzen, um Malware zu erkennen. Es werden jedoch die Konfigurationen und Parameter des vorigen sogenannten "großen Tests" übernommen. Aber der Heureka-Tests liefert eine wichtige Aussage in Hinblick auf die Notwendigkeit zur Aktualisierung der Signaturen und Scan-Engines der Produkte.

Der Test wird zur Zeit nur unter Windows 2000 durchgeführt. Die Testbeds werden für zwei aufeinanderfolgende Zeiträume von je drei Monaten nach dem Gesamt-Testbed-Zeitraum des letzten "großen Tests" erstellt und enthalten meistens nur neue Malware.

Weitere Informationen zu den Tests des aVTC befinden sich in [Seedorf 2002] und [Messerschmidt 2002].

1.3. CARO-Namenskonventionen

CARO[10] ist eine informelle Organisation, die gegründet wurde, um Malware zu analysieren, klassifizieren und zu benennen. Ebenso bietet sich den Mitgliedern von CARO die Möglichkeit, Informationen über Malware und Malware-Samples auszutauschen. Um die einheitliche Identifikation von Viren und später anderer Malware zu ermöglichen, wurde

[10] Computer Antivirus Research Organisation

ein Namensschema erstellt, welches mit neuen Typen und Klassen von Malware immer wieder erweitert wurde.

VMacro ist eine Untergruppe von CARO, die sich speziell über alle Dinge, die Macro- und Script-Malware betreffen, austauschen. Wichtige Aufgaben von VMacro sind die Analyse, Klassifizierung und Benennung neuer Macro- und Script-Malware.

Das generelle CARO-Namensschema sieht folgendermaßen aus:

```
<Klasse>''://''<Plattformen>''/''<Familie>''.''<Variante>[<Devolution>]
  ['' .''<Zusatz>]['':''<Sprachkennung>][''@''m[m]]
```

Zeichenerklärung: <> : variabel; "" : konstant; [] : optional; {} : Menge; | : exklusives oder; + : mindestens einmal oder mehrmals; * : optional oder mehrmals

Wertebereiche der einzelnen Namensteile :

<Klasse> : virus, worm, intended, trojan, joke, generator, dropper (, ...)

<Plattformen> : Für Multipartite-Malware besteht die Möglichkeit, eine Menge von Plattformen in geschweiften Klammern ("{ }") anzugeben[11] :

<Plattform> | "{"<Plattform 1>, <Plattform 2>(, <Plattform n>)*"}"

<Familie> : beliebig

<Variante> : {A-Z}(1,2) (A, ... ,Z,AA, ... ,AZ,BA, ... ,ZZ)

<Devolution> : {1-9}{0-9}* (1,2,3, ...)

<Zusatz> : zusätzliche Informationen zur Malware : beliebig

<Sprachkennung> : {a-z}(2) (de, fr, tw, jp, nl, it, ...)

@MM : [Mass] Mailer

Erklärungen zu den einzelnen Namensteilen sind bei den Definitionen in Abschnitt 1.1.2 zu finden.

Einige Beispiele für Malware-Identifikationen aus dem Heureka 3 Test nach CARO-Namensschema :

```
virus://W97M/AntiSocial.S
virus://W97M/NotHere.C:Ru
worm://W97M/Ping.P@mm
intended://X97M/Divi.AO
```

[11] einige etablierte Abkürzungen für Plattformen befinden sich in A.1; S. 38)

2. Analyse der bestehenden Testbed-Erstellung anhand des Heureka 3 Tests

2.1. Struktur der Testbeds

Für den Heureka-Test werden in der Regel Testbeds für zwei Zeiträume erstellt, die jeweils drei Monate umfassen. Beim Heureka 3 Test handelt es sich um die Monate November bis Januar und Februar bis April des Jahres 2002, wobei Produkte getestet werden, die bis zum 17.12.2001 für den "großen Test" 2002 eingeschickt und bereits auf dem großen Gesamt-Testbed für den Zeitraum bis Oktober 2001 getestet wurden.

Die Testbeds der Heureka-Tests enthalten in der Regel nur Samples von Malware, die in dem entsprechenden Zeitraum neu erschienen ist. Für jeden Zeitraum werden jeweils sechs Testbeds erstellt, die Malware-Samples bestimmter Kategorien enthalten :

- "Macro-Zoo"
- "Macro-Malware[1]"
- "Script-Zoo"
- "Script-Malware[1]"
- "Macro-ITW"
- "Script-ITW"

Die Testbeds sind aus historischen Gründen und Gründen der Handhabbarkeit, nach der Einteilung in Macro und Script, in "Zoo", "Malware" und "ITW" aufgeteilt. "Zoo" enthält nur replizierende Malware[2], während "Malware" jede andere Klasse an Malware, die "In-Zoo" ist, enthält. Diese Einteilung ist dadurch zu erklären, daß bei den frühen Tests nur auf Viren getestet wurde und erst später weitere Malware-Klassen in den Test aufgenommen wurden, wie der Begriff Malware auch erst viel später entstand. Vor allem durch die Weiterentwicklung vernetzter Systeme entstanden ganz neue Malware-Klassen.

[1] Der Begriff "Malware" entspricht hier nicht der üblichen Definition von Malware (siehe weiteren Text)
[2] Viren und Würmer

Dabei war das aVTC auch das erste Anti-Malware-Test-Center, das Trojanische Pferde in den Test aufnahm. Trotzdem ist die Verwendung des Begriffs Malware bei den Testbeds irreführend. Nachfolgend wird dafür stellvertretend der Begriff "Nonreplikativ" verwendet. "Macro-Nonreplikativ" steht nun für "Macro-Malware", also nicht replikative Macro-Malware.

Für den Heureka 3 Test wurde zusätzlich ein Testbed "ITW-Neu" für den Zeitraum November bis Januar erstellt, das nur neue ITW-Malware enthält. Dafür enthalten die übrigen ITW Testbeds die gesamte ITW-Malware für den entsprechenden Malware-Typ und Zeitraum. Diese Abweichung von den vorigen Heureka-Tests wurde beschlossen, da in den beiden Zeiträumen nur sehr wenige Viren bzw. Würmer für Macro und Script neu ITW kamen.

Die Struktur der Testbeds selbst ist, außer zwischen ZOO bzw. ITW und Nonreplikativ, identisch. Die Nonreplikativ-Testbeds enthalten auf oberster Ebene zusätzlich eine weitere Einteilung der nonreplikativen Malware in die verschiedenen Malware-Klassen, wie zum Beispiel "Trojaner", "Kit" und "Dropper".

Die Struktur darunter entspricht wieder der der ZOO- bzw. ITW-Testbeds und ist hierarchisch aufgebaut. Als erstes werden die verschieden Malware-Plattformen, wie beispielsweise "VBS" und "IRC" für Script oder "W98M" und "X97M" für Macro, unterschieden. Eine Ebene weiter befinden sich die Familien, darunter die einzelnen Varianten der Familien, eventuell mit Zusatzinformationen, und noch eine Ebene weiter schließlich die Samples zu den entsprechenden Varianten.

So befinden sich Samples des Trojaners "Seeker" für die Plattform "Java Script" in der Variante "E", der zwischen November und Januar 2002 neu erschien, im Testbed Script-Nonreplikativ im Verzeichnis :

```
\TROJAN\JVS\SEEKER\E
```

Die Samples besitzen alle ein einheitliches Format, welches die Auswertung der Test-Reports später erheblich vereinfacht[3]. Dieses Format sieht folgendermaßen aus :

```
<Plattform>''_''<Sample-Nummer>''_''.<Sample-Endung>
```

Die Sample-Nummer ist dreistellig und fängt mit "000" an. Das Dateiformat des Samples sollte an der Sample-Endung zu erkennen sein.

bis

Die Verzeichnisstrukturen von zwei Heureka 3 Testbeds sind im Anhang A.2 aufgelistet.

[3] Eine ausführliche Beschreibung zur Auswertung der Test-Reports befindet sich in [Messerschmidt 2002]

2.2. Ablauf der Testbed-Erstellung

2.2.1. Sammlung der Samples

Ein wesentlicher Teil zur Erstellung von Testbeds ist der Bezug von Samples, um so mehr, da das aVTC selbst keine replikative Malware reproduziert. Wie bereits bei der Definition von "Testbed" angedeutet, ist der Test um so aussagekräftiger, je mehr Samples sich in den Testbeds befinden, denn je größer die Testmenge, desto wahrscheinlicher, daß spezielle Samples nicht als Malware erkannt oder nicht der richtigen Malware-Familie zugeordnet werden. Zusätzlich werden die Unterschiede bei den Ergebnissen der einzelnen Anti-Malware-Produkte deutlicher sichtbar.

Jeden Monat bekommt Prof. Brunnstein verschlüsselt und gepackt Malware-Kollektionen per E-Mail von folgenden Quellen zugesandt :

- CARO
- McAfee von Network Associates International
- Kaspersky Lab.
- Symantec
- Wildlist Org.

Dazu kommen täglich Zusendungen einzelner Samples neuer Malware aus verschiedenen Quellen. Diese Zusendungen werden von Prof. Brunnstein protokolliert, um das Erscheinungsdatum neuer Malware festzuhalten. Dies ist vor allem wichtig für den Heureka-Test, da bei diesem Test nur neue Malware in die Testbeds aufgenommen werden soll. Von der Wildlist Org. kommen nur Samples für replikative Malware, die in dem entsprechenden Monaten ITW ist.

2.2.2. Sortierung der Samples für die Heureka 3 Testbeds

Nachdem die Malware-Kollektionen entpackt wurden, werden erst mit der neuesten Scanner-Version von F-MacroW alle Kollektionen gescannt, um den größten Teil an Macro-Malware zu identifizieren. Dann werden die erstellten Reports benutzt, um die identifizierte Macro-Malware in eine Urform des fertigen Gesamt-Testbeds zu verschieben, wozu das "Malware Database Utility" von Jens Gallion verwendet wird.

Dieses Tool legt zuerst eine Grundverzeichnisstruktur des Gesamt-Testbeds an. Im zweiten Schritt werden aus den Reports die einzelnen Teile, die man zur Einsortierung in diese Struktur, wie zum Beispiel Plattform, Familie und Variante, und zum Verschieben des Samples benötigt, herausgefiltert. Wichtig ist hierbei, den korrekten Sample-Pfad zum Verschieben zu extrahieren und die einzelnen Teile der Scanner-Identifikation richtig zu erkennen. Es wird dann geprüft, ob die Verzeichnisstruktur für diese bestimmte Malware bereits existiert, wenn nicht wird sie angelegt, und dann das Sample mit einem noch nicht

vergebenen Sample-Namen verschoben. Man erhält durch diese Vorgehensweise einen bereits einsortierten Teil und einen noch einzusortierenden Rest.

Diese Schritte können mit verschiedenen Scannern, die sich zur Identifikation für bestimmte Plattformen eignen, auf den jeweiligen Rest angewandt werden. Für die Identifikation der Samples zur Erstellung der Heureka 3 Testbeds wurden hauptsächlich diese Scanner verwendet:

- F-MacroW : primär für Macro-Malware
- F-Prot (FPR[4]) : primär für Script-; sekundär für Macro-Malware
- McAfee VirusScan (SCN[4]) : sekundär für Script-Malware

Der Rest an Samples, die von keinem Scanner zuverlässig identifiziert wurden oder deren Identifikation Probleme bereitete, müssen von Hand in die Testbeds einsortiert werden. Eine Möglichkeit besteht darin, Samples verschlüsselt und als jeweils ein Attachment pro Sample per E-Mail an eine Einrichtung namens "Minimavis" zu schicken, um von dort eine Identifikation zu erhalten. Minimavis setzt zwischen 20 bis 25 Scanner automatisch auf ein Sample an und schickt dann das Ergebnis zurück. Zusätzlich kann eine manuelle Analyse der enthaltenen Malware erfolgen, dessen Ergebnis der Absender dann ebenfalls erhält. Diese Möglichkeit ist zu aufwendig und steht im aVTC nur Prof. Brunnstein persönlich zur Verfügung. Eine weitere Möglichkeit ist, bei anderen Malware-Experten von CARO nachzufragen. Auch dies ist Prof. Brunnstein vorbehalten.

Sind alle Samples in die Testbeds einsortiert, müssen noch einige Konsistenzprüfungen durchgeführt werden. Es macht zum Beispiel wenig Sinn, identische Samples mehrfach in demselben Testbed zu haben. Die identischen Samples müssen daher aus dem Testbed gelöscht werden. Dieser Doppelte Suchlauf wird mit dem Tool RFW (Roose's File Weeder) durchgeführt, das speziell für die Sortierung und Pflege von Samples entwickelt wurde und ein paar nützliche Funktionalitäten aufweist. RFW kann vor allem Verzeichnisse rekursiv mittels Checksumme und Byte-weisem Vergleich auf doppelte, identische Dateien untersuchen, die dann in einer Logdatei aufgelistet werden. Dazu kann eine Datenbasis aufgebaut werden, die sich zum Vergleich mit einem Verzeichnis verwenden läßt. Es können auf diesem Wege auch neue Dateien in die Datenbasis aufgenommen werden.

Bei einem Doppelten Suchlauf über einem Testbed muß darauf geachtet werden, ob sich die doppelten Samples in demselben Verzeichnis befinden. Befinden sich identische Samples jedoch in unterschiedlichen Verzeichnissen, kann das auf eine unterschiedliche Benennung für dieselbe Malware hindeuten. Die Samples in den betroffenen Verzeichnissen müssen in diesem Fall dann abermals gescannt werden, und es ist zu prüfen, welche Benennung zu CARO konform ist, wonach die falsch einsortierten Samples entsprechend verschoben werden. Ebenso müssen Zero-Byte-Files aus den Testbeds entfernt werden.

Eine weitere Möglichkeit zur Prüfung der Konsistenz ist der Vergleich von Testbeds untereinander, wobei auch hier die Suche nach identischen Dateien zum Einsatz kommt.

[4] aVTC Kürzel für diesen Scanner

Beim Heureka-Test soll dadurch verhindert werden, daß sich dieselbe Malware in Testbeds für unterschiedliche Zeiträume befindet, da Malware ja nur in einem Zeitraum neu aufgetaucht sein kann.

Zuletzt muß sichergestellt werden, daß keine Attribute, wie "Schreibschutz", "Versteckt", "System" oder "Archiv", für Samples und Verzeichnisse gesetzt sind, die die Testergebnisse der Scanner negativ beeinflussen können. Während eines Tests wird allerdings schreibender Zugriff auf die Testbeds verwehrt.

2.3. Probleme bei der Testbed-Erstellung

Die Testbed-Erstellung wird durch einige generelle wie auch spezifische Probleme erschwert. Ein grundlegendes Problem in dem Gebiet Malware ist die uneinheitliche Benennung dieser. Daher liefern unterschiedliche Malware-Scanner zumeist unterschiedliche Identifikationen für ein und dieselbe Malware, die sich dann auch noch mit der Zeit ändern können. Auch das Report-Format der Scanner ist meist sehr unterschiedlich, ohne gute Dokumentation schwer zu analysieren und ändert sich bei einigen Anti-Malware-Produkten ständig. Es müßte eine einzige, zentrale und unabhängige Instanz für die Benennung von Malware geben, an die sich alle, die mit Malware zu tun haben, halten und wenden können. CARO bietet zwar die Möglichkeit für solch eine Instanz, doch leider halten sich viele Malware-Experten nicht daran, da CARO nur eine informelle Organisation ist. Dieses Problem ist vor allem bei File- und Boot-Malware existent, weil die Benennung erst nach und nach durch die Erfahrung bei der Analyse und Klassifizierung von Malware entstanden ist und verbessert werden konnte. Die Probleme waren beim Erscheinen der ersten Macro- und Script-Malware schon eher bekannt, wodurch sich gleich zu Beginn durch VMacro ein besseres Benennungsschema bei diesen Malware-Typen etablieren konnte. Doch auch hier besteht das Problem der Mehrfachbenennung von Malware, insbesondere bei den Familiennamen und der Variantenzuordnung. Eine genauere Beschreibung dieser Problematik und eine mögliche Lösung für VMacro lassen sich bei [Stradt 2002] finden.

Dadurch besteht bei den Sample-Kollektionen ein weiteres Problem, denn die Samples sind kaum nach einem einheitlichen Schema sortiert. Selbst verschiedene Vertretungen ein und derselben Quelle können unterschiedliche Sortierungsschemata für Samples aufweisen. Besonders die Sample-Kollektionen von CARO sind aufgrund der verschiedenen, einzelnen Bezugsquellen nicht gut sortiert. Zudem befinden sich in allen Kollektionen oft genug auch Dateien, die keinerlei Malware enthalten und auch Zero-Byte-Files, in denen sich überhaupt keine Daten befinden. Die Zero-Byte-Files sind einerseits ein Problem für die Scanner bei der Sortierung und andererseits für RFW, beim Finden doppelter Samples.

Eine damit verwandte Schwierigkeit sind die Dateiendungen, die bekanntlich ein bestimmtes Dateiformat darstellen oder zumindest eingrenzen sollen. Da zur Sicherheit möglichst verhindert werden soll, daß Samples einer Kollektion noch ausführbar sind,

werden die ursprünglichen Dateiendungen meist modifiziert, damit sie vom Betriebssystem nicht mehr ausgeführt oder einer Anwendung zugeordnet werden können. Leider folgt diese Umbenennung meist auch keinem einheitlichen Schema. Dateiendungen werden nicht so sehr beachtet und eigentlich sollte ein Scanner auch unabhängig von der Dateiendung Malware in Dateien erkennen können, doch für einige Scanner stellt die Dateiendung trotzdem noch ein wichtiges Merkmal einer Datei dar und könnte zur Vermeidung von fälschlicherweise produzierten Fehlalarmen, sogenannten "false positives", genutzt werden. Um realistischere Testergebnisse zu erhalten, werden die Sample-Endungen bei der Testbed-Erstellung im aVTC wieder zur originalen Endung umbenannt. Es existieren zwar, außer bei CARO, Konventionen für die Sample-Endungen, werden aber meist nicht konsequent festgeschrieben oder durchgesetzt, und teilweise hat die Sample-Endung nichts mehr mit der ursprünglichen Endung zu tun. Daher erfolgte die Wiederumbenennung oft per Handarbeit.

Das CARO-Namensschema kann für die Identifikation von Malware gut eingesetzt werden. Trotzdem sind hier bestimmte Spezialfälle nicht berücksichtigt. Einer davon ist, daß Malware einer bestimmten Plattform andere maliziöse Objekte erzeugen kann, die aber einer anderen Plattform angehören, beispielsweise eine W97M-Malware, die eine EXE-Datei erzeugt. Die Malware ist also nicht auf ein Objekt einer bestimmten Plattform beschränkt, sondern kann aus mehreren Komponenten mehrerer Plattformen bestehen. Solch eine Malware ist nicht nur "multipartite", sondern auch "multicomponent". Eine schwierige Streitfrage in diesem Fall ist, welcher Plattform und welchem Typ die Malware zuzuordnen ist. Die Information über das Dateiformat einer Komponente sollte daher erhalten bleiben, weswegen die Datei-Endung zur genauen Identifikation dazugehören sollte.

Die Benennung von Samples und Verzeichnissen kann auch, vor allem für den Test von Scannern, zu Problemen führen, wenn diese Sonderzeichen enthält. Wie schon angedeutet, ist dies bereits bei der Sortierung der Kollektionen schwierig, da die unsortierten Samples diese Sonderzeichen durchaus enthalten können, was es schwieriger macht, den Sample-Pfad zum Verschieben des Samples aus den Reports der zur Sortierung eingesetzten Scanner zu extrahieren. Als Sonderzeichen sind alle Zeichen anzusehen, die nicht in der Zeichenmenge {a-z,A-Z,0-9,"-","_",","} enthalten sind, wobei mehrere Punkte (".") in Dateinamen oder auch nur einer in Verzeichnisnamen ebenfalls zu Schwierigkeiten führen können. Beim aVTC wird daher versucht, das alte 8.3-Format von DOS einzuhalten und keine Sonderzeichen zu verwenden.

Zum Schluß ist die wachsende Menge an Malware und damit auch gleich die entsprechend wachsende Anzahl an Samples ein nicht zu unterschätzendes Problem. Während die allerersten Tests des aVTC nur ein paar hundert Samples enthielten, liegt die aktuelle Menge bei über 200.000 Samples für die "großen Tests". Zudem gibt es mittlerweile eine deutlich gestiegene Anzahl an Malware-Klassen. Konnte die Menge an Samples früher noch per Hand sortiert werden, ist dies heute nicht mehr durchführbar, selbst die halbautomatische Sortierung stößt da schnell an ihre Grenzen. Demnach sollte eine weitestgehend automatische Sortierung und Archivierung der Samples das Ziel sein.

3. Konzept für ein neues Testbed-Sortierungs- und -Erstellungs-Tool

Das folgende Kapitel beschäftigt sich mit dem Konzept für ein Testbed-Sortierungs- und -Erstellungs-Tool, welches nicht nur für Macro- und Script-Testbeds, sondern auch, mit einigen Anpassungen, für File-Malware konzipiert sein soll. Es soll ebenfalls die Möglichkeit bieten, neue Scanner oder auch andere Programme modulartig hinzuzufügen. Das Prinzip ist an das des Programms von Michel Messerschmidt zur Auswertung von Tests, beschrieben in [Messerschmidt 2002], angelehnt. Bei der Testbed-Sortierung muß jedoch die Identifikation der Malware, die die Scanner ausgeben, wesentlich genauer analysiert werden.

Da sich Scanner-Report-Formate häufig ändern und zwischen verschiedenen Scannern abweichen, ist es nötig, aus einem Scanner-Report die wichtigsten Informationen zu extrahieren und nach einem bestimmten, gut weiter zu verarbeitenden Format normiert in eine Datei abzuspeichern. Bei dem Auswertungsprogramm des aVTC werden dazu für jeden Scanner sogenannte "Keys" angelegt. Das sind Module, die den Scanner-Report abarbeiten und die daraus gewonnen Daten auf verschieden Dateien verteilen, zum Beispiel eine für als Malware erkannte Samples und eine weitere für gescannte, aber nicht als infiziert gemeldete Samples. Die für jeden Scanner einzeln erzeugten Dateien können dann von anderen Modulen ausgewertet werden, um die Testergebnisse zu erhalten.

Beim aVTC-Testbed-Tool werden nach und nach Dateien durch verschiedene Module bearbeitet, bis zur Erstellung der Testbeds. Dies folgt einem "Pipeline"-Prinzip, wobei die Ausgabe eines Moduls die Eingabe für ein anderes Modul darstellt. Dadurch wird vor allem die Identifikation für die zu sortierenden Samples immer weiter verfeinert, bis schließlich eine Datei erzeugt ist, aus der ein oder mehrere Testbeds erzeugt werden können. Diese Vorgehensweise bringt einige Vorteile mit sich.

Ein Vorteil ist die Vermeidung des Umgangs mit den zu sortierenden Samples, wodurch die versehentliche Ausführung von Malware verhindert wird. Die Samples bleiben praktisch bis zur eigentlichen Erstellung des Testbeds unberührt. Die inkrementelle Erstellung der Testbeds durch Verschieben der Samples soll durch eine inkrementelle Erstellung einer Liste, aus der dann die Testbeds erzeugt werden, abgelöst werden. Trotzdem besteht weiterhin die Möglichkeit, die Testbeds inkrementell durch Verschieben zu erstellen. Doch

diese Vorgehensweise birgt die Gefahr, durch einen fehlerhaften Schritt das Testbed in einen inkonsistenten Zustand zu bringen, was immer dann mit erheblichen Aufwand zur Bereinigung der Inkonsistenz verbunden ist, wenn der fehlerhafte Schritt nicht dokumentiert bzw. geloggt und daher vielleicht gar nicht nachvollziehbar ist, oder die Bereinigung selbst nur von Hand durchgeführt werden kann. Vermeiden kann man dieses Problem nur durch die Möglichkeit zur automatischen Bereinigung oder durch die konsequente Sicherung nach jedem Schritt, was entweder viel Speicherplatz verbraucht, oder, wenn die Sicherungen nur eine bestimmte Zeit, zum Beispiel für drei Arbeitsschritte, aufbewahrt wird, die Gefahr mit sich bringt, zu spät einen fehlerhaften Schritt zu bemerken, wodurch ab dann gemachte Arbeitsschritte wiederholt werden müßten oder noch schlimmer der konsistente Zustand verloren gegangen ist. Listen bieten dagegen den Vorteil, daß sie sich leicht und schnell sichern lassen. Außerdem können Listen vor der eigentlichen Erstellung noch einmal manuell geprüft und vor allem auch verändert werden. Die manuelle Korrektur bzw. Ergänzung wird damit im Konzept des aVTC-Testbed-Tools berücksichtigt.

Ein weiterer Vorteil der immer weiteren Verfeinerung der Identifikationen der Samples, ist die Möglichkeit, Teildaten der Identifikationen, die in unterschiedlichen Scannern verteilt vorhanden sind, zu einer einzigen und vollständigen zusammenzufügen. FMacroW liefert beispielsweise keine Daten zur Malware-Klasse, aber meistens sehr genaue Daten zur Plattform, Familie und Variante für Macro-Malware. McAfee-VirusScan liefert Daten zur Malware-Klasse, erkennt manche Malware jedoch generisch, oder verwendet eigene Familienbezeichnungen. Daher werden beim aVTC-Testbed-Tool für jeden Identifikator priorisiert je nach Malware-Typ Teildaten der Identifikation zu einer vollständigen End-Identifikation zusammengefügt. Fehlende Teildaten der Identifikation, die jedoch aus anderen Teildaten erschlossen werden können, werden danach ergänzt.

Ein Nachteil des Verfahrens ist, daß die gesamte Testbed-Datenbasis mit jedem Identifikator, der verwendet werden soll, identifiziert werden muß, während bei der iterativen Einsortierung durch Verschieben von Samples in die Testbeds nur der Teil, der noch nicht einsortiert wurde, mit dem nächst niedriger priorisierten Identifikator gescannt werden muß. Dieser Nachteil kann sich vor allem bei einer großen Testbed-Datenbasis von mehreren Giga-Byte bemerkbar machen.

3.1. Eigenes Schema für die Identifikation von Malware-Samples

In 2.3 wurde schon darauf hingewiesen, daß die gängige Identifikation von Malware nicht unbedingt zweckmäßig ist für die Sortierung von Samples. In dem Namensschema von CARO ist die Information über den Malware-Typ beispielsweise nur implizit über die Malware-Plattform enthalten, und sollte für eine automatische Verarbeitung explizit angegeben sein.

Das hier vorgestellte Namensschema ist stark an das von CARO angelehnt und wurde für die Bedürfnisse zur Sortierung und Erstellung von Testbeds erweitert. Wichtig ist einmal, daß das Format sich gut für die textuelle Verarbeitung eignet, was bei vielen Report-Formaten leider nicht der Fall ist, da die einzelnen Identifikationsteile oft kontextabhängig an verschiedenen Stellen enthalten sind und keine Markierung vorgenommen wird. So werden oft weitere Identifikationsteile mit demselben Seperator, meistens einem ".", kontextabhängig hinzugefügt. Ein einfaches Beispiel für "ein" Identifikationsformat eines Scanners:

```
<Plattform>''/''<Familie>[''.''<Variante>][''.dr''[''.''<Dropper-Variante>]]
```

Ohne Probleme, von Sonderzeichen abgesehen, lassen sich aus diesem Format die Daten <Plattform> und <Familie> extrahieren. Probleme bereitet der optionale Identifikationsteil "dr", der Malware als Malware-Typ "Dropper" auszeichnen soll. Leider ist "dr" auch in dem Wertebereich für die Variante enthalten. Das führt zu einem Problem, da die Variante nicht unbedingt mit angegeben werden muß[1]. So kann ein Sample bestimmter Malware fälschlicherweise der Variante "dr" zugeordnet werden, wenn die Testbed-Erstellung automatisch über die Identifikation im Report erfolgt.

Ein weiteres Kriterium für das eigene Namensschema ist die gute Übertragbarkeit auf die Dateisystemstruktur der Testbeds, die gleichzeitig ausgewogen sein soll. Auch die Sample-Endung soll enthalten sein.

Um künftig Sample-Listen vergleichen zu können, ist die Angabe einer Sample-Länge und zweier Checksummen ebenfalls vorgesehen. Für die Generierung von ITW-Testbeds wird ebenfalls die Information benötigt, wann Malware ITW war bzw. kam.

Das eigene Namensschema bzw. -format für das aVTC-Testbed-Tool sieht so aus :

```
<Typ>'':/''<Klasse>''/''<Plattformen>''/''<Familie>''/''
  <Variante>[''_''<infektiöse Länge>]
  [''_d_''<Devolution>][''_l_''<Sprachkennung>][''__''<Zusätze>]
  ''/.''<Endung>
  ''|''[<Sample-Länge>]'';''[<CRC 1>]'';''[<CRC 2>]
  ''||''[<ITW Monat 1>('','''<ITW Monat n>)*]
```

Zeichenerklärung: <> : variabel; "" : konstant; [] : optional; {} : Menge; | : exklusives oder; + : mindestens einmal oder mehrmals; * : optional oder mehrmals

[1] <Variante> = A

Erklärungen und Wertebereiche :

<Typ> : {A-Z}+

<Klasse> : {A-Z}+

<Plattformen> : Dieser Teil enthält das Plattformkürzel für die Plattform der Malware. Allgemein wird Malware nur einer Plattform zugeordnet, doch es gibt auch Malware, die einer Reihe von Plattformen zugeordnet wird. In diesem Fall können diese getrennt durch "," aufgeführt werden.

<Plattform 1>(","<Plattform n>)* : {A-Z,0-9}+ (","{A-Z,0-9}+)*

<Familie> : {A-Z,0-9,"_","-"}+

<Variante> : Die Angabe "GEN" zeigt an, daß die Malware generisch erkannt wurde. Es können vermutete Varianten, getrennt durch "-",mit angegeben sein.

{A-Z}(1,3) | ("GEN"("-"{A-Z}(1,3))*)

<Devolution> : {1-9} {0-9}*

<Sprachkennung> : {A-Z}(2)

<infektiöse Länge> : Mit "x" kann ein variabler Teil der infektiösen Länge gekennzeichnet werden.

{1-9,"x"} {0-9,"x"}*

<Zusätze> : Der Zusatz beinhaltet zusätzliche Informationen zur Malware. Der gängigste Zusatz für Macro- und Script-Malware ist "MM" für Mass-Mailer. Es können beliebig viele Zusatzinformationen, markiert durch "__"angefügt werden. Ein besonderer Zusatz ist die Dropper-Varianten-Angabe und wird speziell durch "_drv_" angezeigt.

```
''__''<Zusatz 1>(''__''<Zusatz n>)* :

  ''__''{A-Z,0-9}+ (''__''{A-Z,0-9})*
  ['']_drv_''<Dropper-Variante>] (''__''{A-Z,0-9})*
```

<Endung> : Sample-Endung, die das Datei-Format kennzeichnen soll. Da maliziöser Code in anderen Dateiformaten, wie zum Beispiel HTML, eingebunden sein kann, entspricht die Sample-Endung nicht unbedingt der Standard-Endung der entsprechenden Plattform.

{A-Z,0-9}(2,3)

<Sample-Länge> : Die Sample-Länge ist eine Angabe, die zum Vergleich von Samples herangezogen werden.

{1-9} {0-9}* ["."{0-9}*]

<CRC (1|2)> : Noch viel stärker als die Sample-Länge, liefern Checksummen eine Möglichkeit zum Vergleich von Samples, um identische Samples nicht doppelt in ein Testbed einzusortieren. Zur Sicherheit sollten jedoch mindestens zwei Checksummen verwendet werden. Checksummen werden auch von RFW verwendet und in einer Liste ausgeben.

{A-F,0-9}(8)

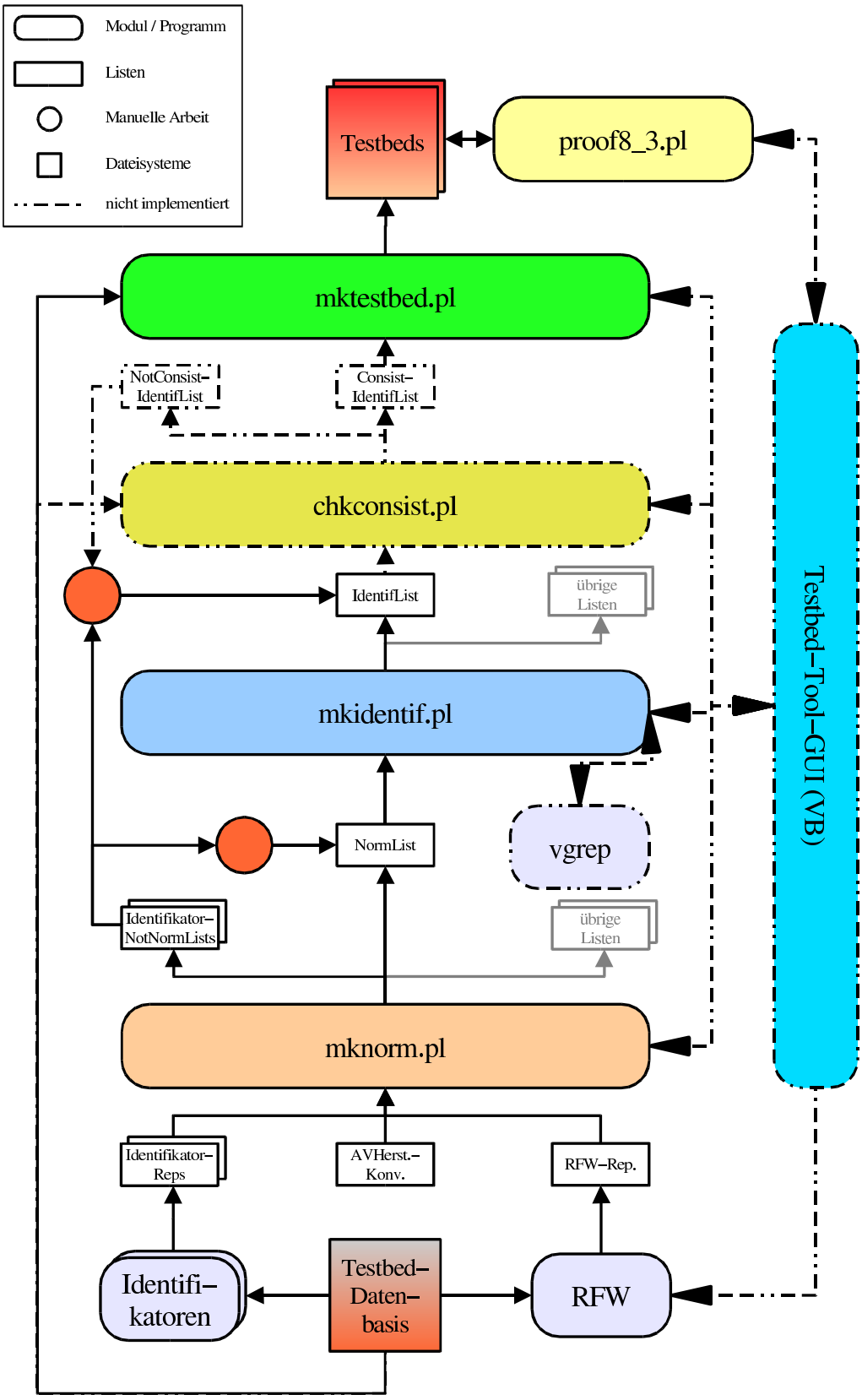
<ITW Monat 1>(","<ITW Monat n>)* : Gibt an, in welchen Monaten die identifizierte Malware ITW war. Dies ist für die Generierung der ITW-Testbeds wichtig. Dabei steht "*" für Malware, die in dem entsprechenden Monat ITW war und "+" für Malware, die in dem Monat neu in die Wildlist aufgenommen wurde.

<ITW Monat> : {"*","+"} {1-9}{0-9}(3) {01-12}

Die wichtigsten Teildaten für die Testbed-Erstellung, die für jede vollständige Identifikation und Sortierung in ein Testbed vorhanden sein müssen, sind <Typ> und <Klasse>, zur Zuordnung zu einem Testbed, <Plattformen>, <Familie>, <Variante> und <Endung>. Die anderen Teildaten können, wie auch oben erkenntlich, je nach Malware vorhanden sein.

3.2. Module des aVTC-Testbed-Tools

Dieser Abschnitt beschreibt die Funktionen der Hauptmodule des aVTC-Testbed-Tools und soll dadurch einen Überblick über diese vermitteln. Im folgenden Kapitel 4 wird dann detaillierter auf die Benutzung der Module eingegangen.


Modul / Programm
Listen
Manuelle Arbeit
Dateisysteme
nicht implementiert
Testbeds
proof8_3.pl
mktestbed.pl
NotConsist–IdentifList
Consist–IdentifList
chkconsist.pl
IdentifList
übrige Listen
mkidentif.pl
NormList
vgrep
Identifikator–NotNormLists
übrige Listen
mknorm.pl
Identifikator–Reps
AVHerst.–Konv.
RFW–Rep.
Identifi–katoren
Testbed–Daten–basis
RFW
Testbed–Tool–GUI (VB)

3.2.1. Normierungsmodul

Dieses Modul hat die Aufgabe, aus den verschiedenen Reports der verwendeten Scanner und Hilfs-Tools eine nach dem Namensschema des aVTC-Testbed-Tools normierte Liste der Samples und ihrer Identifikationen zu erstellen. Bei den Identifikationen ist zusätzlich der identifizierende Scanner bzw. das Hilfs-Tool zur Verarbeitung durch das End-Identifikationsmodul mit angegeben. Ein wesentlicher Bestandteil des Normierungsmoduls sind die sogenannten "Keys", mit welchen durch die Verwendung von regulären Ausdrücken der Sample-Pfad in der Testbed-Datenbasis und die Teildaten aus den Reports extrahiert werden. Für jeden Scanner und jedes verwendete Tool, muß ein solcher Key erstellt und bei Änderungen entsprechend angepaßt werden. Ein zusätzlicher Key ist speziell dafür entworfen, aus einer einfachen Sample-Liste die echten Sample-Endungen der Samples in den Kollektionen nach den Konventionen der entsprechenden Kollektionsquelle zu identifizieren. Allerdings müssen sich die Samples in bestimmten Subverzeichnissen, benannt nach der Kollektionsquelle, befinden. Es ist auch möglich, einen Key zu erstellen, der ebenfalls aus einer einfachen Dateiliste der Testbed-Datenbasis zusätzliche Teildaten aus der Benennung der Verzeichnisse und Samples extrahieren kann.

Alle Sample-Identifikationen, die nicht normiert werden konnten, werden in andere Listen geschrieben, die im nächsten Kapitel erklärt werden.

3.2.2. End-Identifikationsmodul

Nachdem jede Identifikation eines Samples in die normierte Form gebracht wurde, hat das End-Identifikationsmodul die Funktion, eine Liste mit einer einzigen, vollständigen End-Identifikation des Samples zu erstellen. Zusätzlich können Sample-Pfade mit unterschiedlichen Vor-Pfaden zusammengefaßt werden. Dies ist nötig, falls die Testbed-Datenbasis einmal auf ein anderes Laufwerk bzw. in ein anderes Verzeichnis verschoben wurde und daher die Sample-Pfade bis zu einem bestimmten Verzeichnis abweichen, obwohl dasselbe Sample identifiziert wurde.

Das Identifikations-Tool bietet einige Möglichkeiten zur Parametrisierung. Es ist zum Beispiel möglich, nur Samples, die sich innerhalb eines Pfades in einem bestimmten Verzeichnis befinden, auszuwählen, ebenso wie die zu identifizierenden Typen ausgewählt werden können. Mit welchem Gewicht die einzelnen Teildaten der Identifikationen zu einer End-Identifikation zusammengefügt werden sollen, wird zum einen anhand einer Prioritätsvergabe für die Identifikatoren und zum anderen durch die Angabe nicht zu berücksichtigender Identifikatoren geregelt.

Bei der Prioritätsvergabe werden vorrangig die Teildaten des Identifikators mit der höchsten Priorität in die End-Identifikation aufgenommen. Alle niederpriorisierten Identifikatoren können nur Teildaten zur End-Identifikation beisteuern, die kein höher priorisierter Identifikator liefern konnte. Identifikatoren, die keine Priorität erhalten haben, können Teildaten bei Mehrheitsgleicheit beisteuern. Es werden also nur mehrheitlich

identische Teildaten aller nicht priorisierten Identifikatoren der End-Identifikation hinzugefügt. Wenn zum Beispiel drei nicht priorisierte Identifikatoren jeweils ein Teildatum zu einer bestimmten Stelle der End-Identifikation liefern, muß eines bei zwei Identifikatoren identisch sein, um aufgenommen werden zu können. Denkbar wäre auch, daß alle Identifikatoren dasselbe Teildatum liefern müssen. Der Ausschluß von Identifikatoren muß ebenfalls möglich sein, um unzuverlässige Einzel-Identifikationen von vornherein ausschließen zu können.

Ein Tool namens "vgrep" könnte eine zusätzliche Hilfe bei der Erstellung einer End-Identifikation sein. Dieses Tool ähnelt einer Suchmaschiene, der man Identifikationen von Scannern übergeben kann, um dann eine Liste von Identifikationen anderer Scanner zu erhalten. Leider sind die Identifikationen selten eindeutig, da sie sich auch ändern können. Im aVTC wurde bereits der Versuch unternommen, mittels vgrep eindeutige Identifikationen zu erhalten und danach Testbeds zu sortieren. Die Ergebnisse waren jedoch teilweise, obwohl logisch nachvollziehbar, nicht korrekt. Es muß noch genauer geprüft werden, ob sich vgrep als Hilfe zur vollständigen Identifizierung eignet.

Bevor die zusammengefügten End-Identifikationen in die Liste aufgenommen werden, werden wenn möglich fehlende Teildaten durch Analyse der bereits vorhandenen ergänzt und danach eine Prüfung über die Vollständigkeit der End-Identifikation durchgeführt.

Auch dieses Modul schreibt Samples, deren End-Identifikation nicht erstellt werden konnte, je nach Grund in andere Listen.

3.2.3. Konsistenzprüfungsmodule

Bevor Testbeds erstellt werden können, muß vor allem die Konsistenz der End-Identifikationen überprüft werden. Wird bei der Synthese der End-Identifikation im End-Identifikationsmodul vor allem die Vollständigkeit und Belegung in Hinblick auf gegenseitigen Ausschluß der einzelnen Teildaten der End-Identifikation geprüft, was eher einer Syntaxprüfung entspricht, muß die End-Identifikation hier vor allem semantisch überprüft werden. Da die Liste von End-Identifikationen gegebenenfalls auch per Hand verändert werden kann, stellt das Konsistenzprüfungsmodul die letzte Instanz zur Prüfung vor der automatischen Erstellung der Testbeds dar.

Die Konsistenzprüfung soll in zwei Teile aufgeteilt werden. Der erste Teil überprüft die End-Identifikation auf Inkonsistenzen. Diese ist für die Teildaten <Typ>, <Klasse>, <Plattformen>, <Devolution>, <Sprachkennung> und <Zusätze> durchzuführen, da sie teilweise in Verbindung miteinander stehen. Zum Beispiel paßt der Typ "Macro" nicht zur einzigen Plattformen-Angabe "W32", ebenso gehört Malware mit dem Zusatz "MM" nicht zur Klasse "Trojaner", und die Angabe einer Dropper-Variante steht im Wiederspruch zur Klasse "Virus". Manche dieser Inkonsistenzen lassen sich automatisch beseitigen, andere wiederum erfordern manuelle Korrektur. Ebenso soll die Existenz der Sample-Pfade und Samples selbst sichergestellt werden, damit bei der Erstellung der Testbeds die zu kopierenden Samples auch wirklich vorhanden sind, und es sollen kei-

ne identischen Samples in ein Testbed aufgenommen werden, was die Benutzung von zuverlässigen Checksummen erforderlich macht.

Der zweite Teil der Konsistenzprüfung soll ein erstelltes Testbed auf Inkonsistenzen prüfen und diese wenn möglich korrigieren. Dies betrifft die Einhaltung der Testbed-Struktur und des 8.3-Dateisystem-Formates, das Fehlen von leeren Verzeichnissen und Zero-Byte-Samples, sowie die Rücksetzung der Verzeichnis- und Datei-Attribute.

3.2.4. Testbed-Generierungs- und -Pflegemodul

Dieses Modul ist hauptsächlich für das fehlerfreie Kopieren bzw. Verschieben der Samples aus der Testbed-Datenbasis in die Testbeds zuständig. Dabei soll ein Log erstellt werden, welches es möglich machen soll, dem Testbed-Sample das Testbed-Datenbasis-Sample zuordnen zu können.

Weitere Anforderungen können die automatische Erstellung der Pack-Testbeds für die "großen Tests" und die Zusammenstellung von Sample-Kollektionen, die unter Umständen auch gleich gepackt werden können, sein.

3.2.5. aVTC-Testbed-Tool-GUI

Die aVTC-Testbed-Tool-GUI soll, wie der Name schon sagt, eine grafische Oberfläche zur Bedienung des aVTC-Testbed-Tools unter Windows bieten. Die einzelnen Module sollen von hier aus mit den verschiedenen Parametern aufgerufen und damit gesteuert werden. Fehler und Logs sollen angezeigt werden können, und die GUI soll den komfortablen Umgang bzw. die komfortable manuelle Bearbeitung der verschieden, erstellten Listen ermöglichen.

4. Dokumentation für das aVTC-Testbed-Tool

In diesem Kapitel werden die zur Zeit der Abgabe dieser Arbeit implementierten Module des aVTC-Testbed-Tools, alle in der vorkompilierenden Skriptsprache PERL geschrieben, in ihrer Benutzung erläutert, sowie die erzeugten Listen näher erklärt. Leider sind noch nicht alle Module vollständig in ihren Funktionalitäten implementiert und einige Konfigurationseinstellungen sind relativ fest in die Module integriert. Hier besteht Bedarf an der externen Speicherung solcher Konfigurationen in Konfigurationsdateien. Die Benutzung von vgrep als Identifikations-Hilfe war zwar anfangs vorgesehen, wurde jedoch bisher nicht in das Modul zur End-Identifizierung eingebunden. Bei Erstellung dieser Arbeit kamen außerdem noch einige Ideen hinzu und das Namensschema des aVTC-Testbed-Tools wurde dabei ebenfalls modifiziert und muß noch in den Implementationen geändert werden. Was aber vor allem noch fehlt ist die Implementation zur Konsistenzprüfung der End-Identifikationen und zur Grafischen Benutzerobefläche (GUI), sowie der Ausschluß von Identifikatoren bei der End-Identifizierung.

Der Ablauf zur Erstellung von Testbeds mit den aktuell implementierten Modulen sieht folgendermaßen aus :

Normalisierung der Testbed-Datenbasis : Aus den in 2.3 genannten Problemen mit Verzeichnisstrukturen, die sich außerhalb der Norm des 8.3-Formates bewegen, ist es empfehlenswert, in Zukunft vor dem Scannen bzw. Identifizieren mit den Identifikatoren, die Testbed-Datenbasis mit proof8_3.pl zu normalisieren. So werden von vornherein Probleme mit ungewöhnlichen Sample-Pfaden und leeren Dateisystemobjekten vermieden. Zusätzlich sollten alle Dateiattribute der Samples und der Verzeichnisse in der Testbed-Datenbasis zurückgesetzt werden, um auch hier Komplikationen zu vermeiden. Danach wäre es sehr sinnvoll, mit RFW alle identisch doppelten Samples aus der Testbed-Datenbasis zu entfernen. Vielleicht sollte dem sogar ein zweiter Normalisierungslauf mit proof8_3.pl folgen, um leere Verzeichnisse aus der Testbed-Datenbasis zu löschen.

Scannen der Testbed-Datenbasis : Nun wird die Testbed-Datenbasis mit allen zur Sortierung zu verwendeten Identifikatoren gescannt bzw. identifiziert. Die Wahl der Anzahl der Identifikatoren sowie der Identifikatoren selbst gibt den Ausschlag dafür, wie lange diese Phase andauert. RFW sollte stets zur Generierung von Checksummen für die Samples mit verwendet werden.

Normierung der Reports : Bevor die Normierung tatsächlich durchgeführt wird, sind die Key-Module der Identifikatoren und der Key, der die Dateiendungskonventionen der Kollektionsquellen enthält, entsprechend anzupassen. Dann wird nacheinander mit den Reports aller Identifikatoren die Norm-Liste für die Testbed-Datenbasis erzeugt. Die übrigen Daten aus dem Report, insbesondere die aus der NotNorm-Liste, müssen von Hand entweder in die Norm-Liste oder auch später in die Identif-Liste eingefügt werden.

End-Identifizierungs-Liste erstellen : Für unterschiedliche Malware-Typen kann mit unterschiedlichen Prioritäten für die verwendeten Identifikatoren nun eine Liste der End-Identifikationen der Samples erstellt werden. In diese Liste können dann von Hand nicht automatisch verarbeitete Informationen eingefügt werden.

Generierung der Testbeds : Aus der Liste mit den End-Identifikationen werden dann die verschiedenen Testbeds generiert, wobei die Samples aus der Testbed-Datenbasis entweder kopiert oder verschoben werden.

Testbed-Konsistenzprüfung : Zum Schluß wird die Konsistenz der generierten Testbeds überprüft. proof8_3.pl spürt eventuelle Abweichungen vom 8.3-Format auf. Obwohl sich doppelte Samples eigentlich nicht in den Testbeds befinden können, sollte dies mit RFW trotzdem überprüft werden. Außerdem kann mit RFW ein Vergleich unter den verschiedenen Testbeds oder mit Testbeds voriger Tests durchgeführt werden. Dann sollten die Attribute der Samples und der Verzeichnisse wenn nötig zurückgesetzt werden.

4.1. Normierung : mknorm

Das Normierungsmodul besteht generell, wie fast jedes folgende Modul, aus einem Hauptmodul und einem Modul für das Einlesen und die Konvertierung der Hauptliste. Zusätzlich gibt es noch ein paar kleinere Module, die von allen hier vorgestellten Modulen genutzt werden, denn sie enthalten nützliche Methoden, die immer wieder benötigt werden. Das Modul "texttools.pm" enthält Methoden zur Bearbeitung von Zeichenketten. "filehandling.pm" beinhaltet Methoden für den Umgang mit Dateien, zum Beispiel eine Datei-Backup-Methode. Wie schon angedeutet, werden für das Normierungsmodul sogenannte "Keys" benötigt, um das Report-Format entsprechend in eine normierte Form überführen zu können.

Das Hauptmodul zur Normierung der unterschiedlichen Identifikationsformate benötigt als Parameter in dieser Reihenfolge :

Identifikator-Kürzel : Dieses Kürzel gibt an, welcher Key verwendet werden soll, um den gegebenen Report auszuwerten. Wenn möglich werden die aVTC üblichen dreistelligen Kürzel verwendet. Das Key-Modul wird daraus folgendermaßen abgeleitet :

"mknorm_"<Kürzel>"_key.pm"

Report-Datei : Der relative Pfad zur auszuwertenden Report-Datei.

Norm-Datei : Relativer Pfad zur verwendeten Norm-Datei. Ist eine Norm-Datei bereits vorhanden, wird sie mit der Datei-Endung ".old" umbenannt. Die alt und neu normierten Identifikationen für die Samples wird in die angegebene Datei gespeichert.

Beispiele zum Aufruf :

```
mknorm.pl fmw 021\F-MacroW.rep h3_021_norm.lst

mknorm.pl fpr 021\F-Prot.rep h3_021_norm.lst

mknorm.pl vsc 021\VScan.log h3_021_norm.lst
```

Existiert die angegebene Norm-Datei bereits, wird der Inhalt in einen Hash eingelesen. Die Speicherung in einem Hash ist recht komfortabel und soll bei der Normierung der angegebenen Report-Datei verhindern, daß vollkommen identische Identifikationen vom selben Identifikator für ein Sample mehrmals in die Norm-Liste bzw. Norm-Datei aufgenommen werden.

Nun wird jede Zeile aus der Report-Datei ausgelesen und normiert, falls die Zeile eine Identifikation enthält. Die Auswertung der Zeile erfolgt in dem entsprechenden Key-Modul, welches im nächsten Abschnitt erklärt wird. Die erhaltenen Daten werden in dem Hash gespeichert und nach der Auswertung der Report-Datei in die Norm-Datei geschrieben.

Alle Zeilen der gegebenen Report-Datei, aus denen keine normierte Identifikation oder ein Sample-Pfad extrahiert werden konnte, werden in andere Dateien abgespeichert :

NotNorm-Datei : Diese Datei enthält alle Zeilen, welche zwar als Identifikationszeilen für Samples erkannt wurden, jedoch nicht einem bekanntem Format entsprechen. Ist der Identifikator-Key dem Report-Format gut angepaßt, sollte diese Datei nicht erzeugt werden.

UnrelIdentif-Datei : enthält alle Zeilen, bei denen eine unzuverlässige Identifikation für ein Sample gefunden wurde. Ein Beispiel für so eine Identifikation wäre "may be a virus" gefolgt von einer Identifikation.

NoIdentif-Datei : Ähnlich wie bei der UnrelIdentif-Datei. Die Zeilen enthalten jedoch keinerlei Identifikations-Daten. Beispiel: "new virus".

SBO-Datei : Der Name ist vom Auswertungsprogramm übernommen und bedeutet: "Scannt, But Ok". Die Datei enthält Zeilen, bei denen Samples zwar gescannt, aber nicht als Malware gemeldet wurden, was allerdings auch wahr sein kann.

Rest-Datei : enthält alle sonstigen Zeilen, die nicht einer anderen Liste zugeordnet wurden. Normalerweise sollten das unwichtige Zeilen, wie Statistiken sein.

4.2. Das Identifikator-Key-Gerüst

Da es nötig sein wird, die Keys neuen Formaten anzupassen oder neue Keys zu erzeugen, wird hier der grundlegende Aufbau eines solchen Keys erläutert. Im Anhang B.1.1 befindet sich dazu der Quellcode, bei dem auch am Anfang der Umgang mit Regulären Ausdrücken von Perl erklärt wird.

Der Key besteht hauptsächlich aus einfachen "if"-Abfragen, wobei jede Zeile des gegebenen Reports an den Key übergeben wird, und versucht wird, mit einem Regulären Ausdruck zu vergleichen. Hier findet vor allem die Zuteilung zu den verschieden Listen statt.

Ist der Vergleich erfolgreich und dadurch auch eine Zuteilung, werden die Daten durch Klammerung adressiert an die entsprechenden Variablen übergeben oder die gesamte Zeile in die entsprechende Datei geschrieben, wenn keine Identifikation gefunden wurde. Die Daten können im Falle einer bekannten Identifikation dann noch im selben Key-Modul konvertiert werden. Hier können Identifikator-spezifische Besonderheiten der Norm angepaßt werden. Zum Beispiel können Plattformen wie "Word97Macro" in "W97M" konvertiert werden oder zusätzliche Daten extrahiert bzw. Daten weiter auf die verschiedenen Variablen aufgeteilt werden.

4.3. End-Identifizierung : mkidentif

Der generelle Aufbau dieses Moduls ist dem von mknorm.pl ziemlich ähnlich. Auch hier wird die bestehende Identif-Datei gesichert und ausgelesen. Zu den regulären können auch optionale Parameter angegeben werden. Diese regeln hauptsächlich die Auswahl der abzuarbeitenden Samples und der Malware-Typen, die in die Identif-Liste bzw. -Datei aufgenommen werden sollen, sowie die Prioritäten der Identifikatoren.

Folgende Parameter müssen bzw. können angegeben werden :

Norm-Datei : Die vom Normierungs-Modul erstellte Liste wird in diesem Modul weiterverarbeitet.

Identif-Datei : Genau wie beim Normierungs-Modul kann die Liste der Identifikationen inkrementell aufgebaut werden. Diese Datei enthält nach der Bearbeitung eine Liste von Samples und deren vollständige Identifikation.

optional: "-prior="<Scannerkürzel 1>,...,<Scannerkürzel n> : Diese Angabe ist optional und gibt eine Liste der priorisierten Scanner an und zwar in der Reihenfolge der Angabe. Der höchstpriorisierte Scanner steht am Anfang der Aufzählung, der zweit höchste an zweiter Stelle usw..

optional: "-workpath="<Verzeichnis> : gibt ein Verzeichnis an, das sich im Pfad der Samples befinden muß. Der Pfad davor wird abgeschnitten. So kann geregelt wer-

den, welche Samples verarbeitet werden und ein Verschieben der Testbed-Datenbasis in einen anderen Pfad kann damit ausgeglichen werden.

optional: "-tbs="<Typ 1>,...,<Typ n> : eine Auflistung von abzuarbeitenden Malware-Typen. Nur die angegebenen Typen werden in die Identif-Datei mit aufgenommen.

Beispiel zum Aufruf :

```
mkidentif.pl h3_021_norm.lst h3_021_identif.lst "-prior=fmw,fpr"
"-workpath=heur3-db.021" "-tbs=macro"

mkidentif.pl h3_021_norm.lst h3_021_identif.lst "-prior=fpr,vsc"
"-workpath=heur3-db.021" "-tbs=macro,script"
```

In Verbindung mit der Prioritätenoption kann mit der Typenauswahloption für jeden Typ eine bestimmte Priorisierung vorgenommen werden. Da bereits vollständig identifizierte Samples nicht noch einmal bearbeitet werden, lassen sich so inkrementell mehrere Typen in eine Identif-Datei aufnehmen. Ein erster Aufruf kann beispielsweise mit den Optionen "-tbs=macro" und "-prior=<Scanner 1>,<Scanner 2>" erfolgen. Es wird nur Macro-Malware in die Identif-Datei aufgenommen und mit der angegebenen Priorisierung vollständig identifiziert. Der zweite Aufruf mit "-tbs=script,macro" und "-prior=<Scanner 3>,<Scanner 2>" erzeugt eine Identif-Datei mit identifizierter Macro- und Script-Malware. Die zweite Priorisierung beeinflußt nicht die bereits voll identifizierte Macro-Malware, da bereits voll identifizierte Malware nicht mehr bearbeitet wird.

Bei der Verarbeitung werden wieder andere Dateien außer der Identif-Datei erzeugt, die unterschiedliche Samples oder Identifikationen enthalten :

NotNorm-Datei : Diese Liste enthält Samples, bei denen mindestens eine Identifikation nicht der Norm entspricht. Ebenso werden End-Identifikationen beim Einlesen und Schreiben der Identif-Datei, die nicht normgerecht sind, in dieser Datei abgespeichert. Die entsprechenden Identifikationen werden durch "NotNorm" gekennzeichnet.

NotIdentif-Datei : Die End-Identifikationen in dieser Datei entsprechen zwar der Norm, sind jedoch nicht vollständig. Hier müssen noch wichtige Teildaten ergänzt werden.

Rest-Datei : Diese Datei beinhaltet alle Samples und Identifikationen, die nicht verarbeitet wurden. Ist die Workpath-Option verwendet worden, sind hier diejenigen Samples aufgelistet, die nicht im Arbeitspfad sind, deren Pfad also nicht den angegebenen Pfadteil enthält. Ebenso werden hier die Samples, die nicht den zu verarbeitenden Malware-Typen, durch die Typenauswahloption angegeben, angehören, abgespeichert.

Die Liste der End-Identifikationen, wird wenn bereits vorhanden, ausgelesen und gespeichert. Es werden unter Verwendung der Workpath-Option Identifikationen, deren

zugehörige Sample-Pfade identisch sind, zusammengeführt. Wenn die Prioritätenoption angegeben ist, werden die aufgeführten Identifikatoren mit der entsprechenden Priorität versehen.

Die eigentliche Arbeit liegt aber in der Erstellung einer vollständigen End-Identifikation. Nachdem alle einzelnen Identifikationen syntaktisch geprüft und zur Weiterverarbeitung in einzelne Teildaten zerlegt wurden, werden nun die Teildaten priorisiert in die initialisierte End-Identifikation eingefügt. Es wird nur ein Teildatum eingefügt, wenn die entsprechende Stelle in der End-Identifikation leer ist, also kein höher priorisierter Identifikator dieses Teildatum bereits lieferte.

Den in der Norm-Datei erkannten und nicht als priorisiert angegebenen Identifikatoren wurde keine Priorität zugeteilt. Für jede einzelne, unbelegte End-Identifikationsstelle wird nun versucht, eine mehrheitliche Übereinstimmung bei den nicht priorisierten Identifikatoren zu finden. Dabei werden leere Teildaten nicht mitgezählt, so daß wenn nur ein Identifikator an einer bestimmten Stelle ein Teildatum liefert, dieses versucht wird zu übernehmen. Auch hier werden Teildaten der nicht priorisierten Identifikatoren nur übernommen, wenn die betreffende Stelle der End-Identifikation noch leer ist.

Die End-Identifikationen werden danach komplettiert. Hier werden implizite Teildaten an den entsprechend leeren Stellen eingefügt. Beispielsweise kann der Malware-Typ durch die Malware-Plattform, sofern es sich um eine einzige handelt, bestimmt werden. Ebenso läßt die Plattform zum Teil auf die Sample-Endung schließen. Die Komplettierung erfolgt durch implikative Regeln, die recht einfach zu erweitern sind.

Bevor die komplettierten End-Identifikationen in die Identif-Datei geschrieben werden, werden sie nochmals einer Prüfung auf Vollständigkeit unterzogen. Dabei wird geprüft, ob bestimmte, wichtige Stellen bzw. Teildaten der End-Identifikation leer sind. Es ist sogar möglich, Stellen auf wechselseitigen Ausschluß zu überprüfen.

Aus der Identif-Datei können bereits Testbeds erzeugt werden.

4.4. Testbed-Generierung : mktestbed

Die zur Zeit der Abgabe dieser Arbeit aktuelle Version des Testbed-Generierungs-Moduls erzeugt eine einfache Batch-Datei, die ausgeführt werden kann, um die entsprechenden Testbeds zu erzeugen. Dies hat den Vorteil, die Batch-Datei noch einmal überprüfen zu können, bevor die Testbeds generiert werden. Leider ist eine Konvertierung in das 8.3-Format in diesem Modul bisher nicht implementiert. In Zukunft werden die Dateisystemfunktionen von Perl genutzt und alle im Konzept vorgestellten Funktionalitäten implementiert werden.

Folgende Parameter dienen der Steuerung des Moduls :

Identif-Datei : enthält neben den Sample-Pfaden die End-Identifikationen, die in das Dateisystem des jeweiligen Testbeds übertragen werden.

Batch-Datei : Die Befehle in dieser Datei werden der Reihe nach zur Generierung der Testbeds abgearbeitet. Die Ausgaben der Befehle und eine Zuordnung des Testbed-Datenbasis-Sample zum Testbed-Sample werden bei Ausführung in eine Log-Datei geschrieben.

Zielverzeichnis : Hier wird das Zielverzeichnis angegeben, in welchem alle Testbeds generiert werden.

optional: "-move" : Standardmäßig werden die Samples aus der Testbed-Datenbasis in die Testbeds kopiert. Diese Option bewirkt, daß die Samples statt dessen in die Testbeds verschoben werden.

Beispiel zum Aufruf :

```
mktestbed.pl h3_021_identif.lst h3_021_tbs.bat h3_tbs.021
```

Auch dieses Modul erzeugt außer der Batch-Datei andere Dateien, die auf Problemfälle bei den End-Identifikationen hinweisen können :

NotNorm-Datei : Wiederum enthält diese Datei End-Identifikationen, die nicht normgerecht sind.

NotTestbed-Datei : Bei dem Inhalt dieser Datei handelt es sich um Samples, die aus irgendeinem Grund nicht in das entsprechende Testbed kopiert bzw. verschoben werden konnten. Momentan ist dies nur bei einem Nummer-Überlauf der Fall, der weiter unten erklärt wird.

Rest-Datei : Wie bei dem End-Identifizierungs-Modul werden hier Samples aufgelistet, die nicht bearbeitet wurden. Dies ist für den Fall vorgesehen, daß nur bestimmte Testbeds erzeugt werden sollen. Der Rest der Samples wird dann hier aufgelistet.

Log-Datei : Diese Datei wird beim Ausführen der Batch-Datei erzeugt und ermöglicht die Überprüfung der Kopier- bzw. Verschiebungsoperationen.

Erst werden die Daten aus der Identif-Datei gelesen. Danach wird der Pfad für das Testbed-Sample errechnet. Der Sample-Name folgt dem aVTC-Format :

```
<Plattform>''_''<dreistellige Sample-Nummer>''_.''<Sample-Endung>
```

Dabei wird die kleinste freie, dreistellige Sample-Nummer vergeben, die bei "000" beginnt. Ist keine Nummer mehr frei, so ist ein Nummer-Überlauf eingetreten und das Sample wird nicht in die Batch-Datei geschrieben. Wenn es keine Probleme gab, wird eine Zuordnung des Quellpfades zum Zielpfad, also des Testbed-Datenbasis-Sample-Pfades zum Testbed-Sample-Pfad gefolgt von dem Kopierbefehl in die Batch-Datei geschrieben. Ist die Verschieben-Option gewählt, wird zusätzlich ein Befehl zum Löschen des Testbed-Datenbasis-Sample angefügt.

4.5. Testbed-Überprüfungs-Tool : proof8_3

Dieses Tool überprüft die Struktur eines Testbed auf das 8.3-Format. Es stellt somit einen Teil der Konsistenzprüfung von Testbeds dar. Als Eingabeparameter wird die Pfadangabe des zu überprüfenden Verzeichnisses bzw. Laufwerkes benötigt.

Zusätzliche Optionen sind :

"-ps="<Pfadseperator> : Standardmäßig wird als Seperator der einzelnen Ebenen eines Pfades der bei Windows übliche "Backslash" verwendet. Diese Option bietet die Möglichkeit, einen anderen Pfadseperator zu verwenden, was speziell für Unix-Systeme gedacht ist.

"-nodemo" : Normalerweise überprüft das Tool das angegebene Ziel und schreibt das Problem nur in eine Log-Datei, ohne das Ziel zu verändern. Mit dieser Option wird angegeben, daß die Problemlösungen gleich durchzuführen sind. Muß ein Objekt umbenannt oder gelöscht werden, wird dies sofort ausgeführt.

Beispiel zum Aufruf :

```
proof8_3.pl h3_tbs.021
```

Das Tool nimmt folgende Prüfungen vor bzw. besitzt folgende Funktionalitäten :

Spezialzeichen : Es wird für jeden Dateinamen, jede Dateiendung und jedes Verzeichnis geprüft, ob es Sonderzeichen enthält. Wenn dem so ist, werden die Sonderzeichen jeweils durch "_" ersetzt. Dabei wird allerdings nicht berücksichtigt, das Verzeichnisse mit Endungen, was zwar nicht üblich aber möglich ist, ein Problem darstellen können. Eine Verzeichnisbezeichnung wie "verzeichnis.001" wird daher wie eine Dateibezeichnung gehandhabt.

Zeichenlängen : Ein Objekt sollte mit maximal acht Zeichen im Namen und drei Zeichen in der Endung bezeichnet sein. Ist dies nicht der Fall, werden überzählige Zeichen jeweils abgeschnitten.

Numerierung bei gleicher Bezeichnung : Soll ein Objekt umbenannt werden, so kann es vorkommen, daß die neue Bezeichnung bereits existiert. Besteht dieses Problem, muß die neue Bezeichnung so abgeändert werden, daß eine noch nicht vergebene Bezeichnung gewählt wird. Angelehnt an Windows, werden die letzten Zeichen des Namens durch "_"<Nummer> ersetzt, wobei die nächste freie Nummer, beginnend mit 1, gesucht wird.

Zero-Byte-Dateien löschen : Dateien mit der Dateilänge null (0) werden aufgezeigt bzw. gelöscht.

leere Verzeichnisse löschen : Verzeichnisse, die keine Dateien oder Unterverzeichnisse beinhalten werden ebenfalls aufgezeigt oder gelöscht.

Datei- und Unterverzeichnisstatistik : Am Ende der Prüfung wird eine Statistik der gezählten, nicht Zero-Byte-Dateien und nicht leeren Verzeichnisse angezeigt. Wird das Tool im "nicht Demo"-Modus ausgeführt, werden gescheiterte Versuche der Löschung von Objekten entsprechend berücksichtigt.

Das Programm durchläuft dabei alle Unterverzeichnisse des angegebenen Verzeichnisses rekursiv. Wird ein Unterverzeichnis im aktuellen Verzeichnis gefunden, wird die Hauptmethode rekursiv mit dem Unterverzeichnis als Eingabe aufgerufen. Rückgabewerte sind die Anzahl der Dateien und der Unterverzeichnisse. Wird eine Datei im aktuellen Verzeichnis gefunden, wird dieses überprüft und die Dateianzahl entsprechend erhöht. Wurden alle Objekte im aktuellen Verzeichnis überprüft, wird erst anhand der Dateianzahl und Unterverzeichnisanzahl ermittelt, ob das aktuelle Verzeichnis leer ist, und wenn nicht wird die die Namensprüfung an ihm durchgeführt.

5. Ausblick

Bevor die Implementierung des aVTC-Testbed-Tools in Angriff genommen wurde, sollte sich die Baccalaureatsarbeit eigentlich mit einem Konzept zur Erstellung einer Malware-Datenbank befassen. Damit sollten Samples im aVTC effizient verwaltet werden können. Die Generierung der Heureka 3 Testbeds, bei der ich mithelfen wollte, um einen praxisbezogenen Einblick zu gewinnen, war jedoch erheblich aufwendiger als geplant. Außerdem funktionierte das bis dahin verwendete Malware-Datenbank-Tool von Jens Gallion nicht mehr richtig, da sich das Report-Format eines Scanners geändert hatte. Daher beschloß ich, ein neues Tool zur Generierung von Testbeds zu entwickeln und auch meine Arbeit darüber zu schreiben.

Die Idee, eine Malware Datenbank für das aVTC aufzubauen, ist immer noch vorhanden. Eine Liste konsistenter End-Identifikationen von Samples könnte zum Beispiel zur Initialisierung einer solchen Datenbank verwendet werden. Die Vision wäre es, Testbeds zu generieren, indem man einfach nur die Generierungsparameter, wie zum Beispiel Zeitraum und Malware-Typ, an eine Applikation übergibt, welche dann aus der Malware-Datenbank heraus ein Testbed erzeugt. Eine weitere Applikation sollte die einfache Erweiterung der Malware-Datenbank ermöglichen, wobei neue Samples mit der bestehenden Datenbank und der darin enthaltenen Samples geprüft und identifiziert werden sollten.

Literaturverzeichnis

[Kittel 2000] Kittel, Martin; Tićak, Mario: "Viren und Malware - Eine Einführung", 2000, Universität Hamburg

[Bontchev 1998] Dr. Bontchev, Vesselin: "Methodology of Computer Anti-Virus Research", Dissertation, 1998, Universität Hamburg

[Brunnstein 1999] Prof. Dr. Brunnstein, Klaus: "From AntiVirus to AntiMalware Software and Beyond: Another Approach to the Protection of Customers from Dysfunctional System Behavior", Artikel, 1999, 22. National Information Systems Security Conference

[List-Of-Known-Macro-Viruses 09/2001] Dr. Bontchev, Vesselin;Prof. Dr. Brunnstein, Klaus: "Macro Virus List (PC + Macintosh)", List, 2001, Universität Hamburg

[Messerschmidt 2002] Messerschmidt, Michel: "Analyse der aVTC-Tests und Verbesserung der Testauswertung", Baccalaureatsarbeit, 2002, Universität Hamburg

[Seedorf 2002] Seedorf, Jan: "Verfahren zur Qualitätsbestimmung der Erkennung von bösartiger Software", Diplomarbeit, 2002, Universität Hamburg

[Stradt 2002] Stradt, Michael: "Konzeption und Implementation einer Datenbankanwendung für die Benennung bösartiger Software einschl. Benachrichtigungsdienst", Diplomarbeit, 2002, Universität Hamburg

[Wildlist] The Wildlist Organization International, `http://www.wildlist.org`

A. Anhang

A.1. Abkürzungsverzeichnis

Das Verzeichnis enthält hauptsächlich Abkürzungen, die bei der Identifizierung von Malware gebräuchlich sind :

- A97M Access 97 Macro (Plattform)
- APM Ami Professional Macro (Plattform)
- INTD Intended (Typ)
- @[M]M (= __MM) [Mass] Mailer (Zusatz-Info)
- JS (= JVS) Java Script (Plattform)
- O97M Office 97 Macro (Plattform)
- P98M Project Plan 98 Macro (Plattform)
- PP97M Power Point 97 Macro (Plattform)
- SEC_RISK Security Risk (Typ)
- VBS Visual Basic Script (Plattform)
- VCON (= KIT) Virus Construction Kit (Typ)
- W2M Word 2 Macro (Plattform)
- W97M Word 97 Macro (Plattform)
- WM Word Macro für Versionen 5-7 (Plattform)
- WBS Word Basic Script (Plattform)
- X97M Excel 97 Macro (Plattform)
- XF Excel Formula (Plattform)
- XM Excel Macro (Plattform)

A.2. Verzeichnisstruktur von zwei Heureka 3 Testbeds

A.2.1. Macro Zoo bis Januar 2002

```
H:\A97M\ACCREST\A
H:\W97M\ANTISOCI\S
H:\W97M\ASSILEM\I
H:\W97M\ASSILEM\J
H:\W97M\ASTIA\BU
H:\W97M\ASTIA\BV
H:\W97M\AYAM\A__MM
H:\W97M\BABLAS\DY
H:\W97M\BLECK\L
H:\W97M\BLOCKOUT\B
H:\W97M\BOTTRA\D
H:\W97M\BOTTRA\E
H:\W97M\BUENDIA\E
H:\W97M\CANDLE\E
H:\W97M\CANDLE\F
H:\W97M\CANDLE\G
H:\W97M\CANDLE\H
H:\W97M\CODEMAS\B
H:\W97M\DEATH\C
H:\W97M\DED\S
H:\W97M\DED\T
H:\W97M\DED\U
H:\W97M\DED\V
H:\W97M\DEEDEE\A__MM
H:\W97M\DEEDEE\B__MM
H:\W97M\DEEDEE\C__MM
H:\W97M\DELDOC\B
H:\W97M\DEST\F
H:\W97M\DEST\G
H:\W97M\DEST\H
H:\W97M\DOCCOPY\D
H:\W97M\DOCCOPY\E
H:\W97M\EIGHT941\Z
H:\W97M\ETHAN\EN
H:\W97M\FF\L
H:\W97M\FIFTEEN\A
H:\W97M\FLOP\C
H:\W97M\FOOTER\AB
```

```
H:\W97M\GABE\A
H:\W97M\GABE\B
H:\W97M\GABE\C
H:\W97M\GABE\GEN-A-C
H:\W97M\GAMLET\D
H:\W97M\GOODDAY\C
H:\W97M\HOPE\AH
H:\W97M\INTRUDED\B
H:\W97M\IRA\B
H:\W97M\IRA\C
H:\W97M\IRA\D
H:\W97M\KOP\I-DE
H:\W97M\LILY\E
H:\W97M\LILY\F
H:\W97M\MANDIR\A
H:\W97M\MARKER\JU
H:\W97M\MARKER\JV
H:\W97M\MARKER\JW
H:\W97M\MARKER\JX
H:\W97M\MARKER\JY
H:\W97M\MARKER\JZ
H:\W97M\MARKER\KA
H:\W97M\MARKER\KB
H:\W97M\MARKER\KC
H:\W97M\MARKER\KE
H:\W97M\MARKER\KG
H:\W97M\MARKER\KH
H:\W97M\MARKER\KK
H:\W97M\MARKER\KL
H:\W97M\MELISSA\BP__MM
H:\W97M\MESMOTH\A
H:\W97M\MINIMAL\AL
H:\W97M\MINIMAL\BY
H:\W97M\MIRAT\B
H:\W97M\MIRAT\C
H:\W97M\MIRAT\D
H:\W97M\MIRAT\E
H:\W97M\MIRAT\F
H:\W97M\MTRUE\C
H:\W97M\MTRUE\D
H:\W97M\MYCO\A
H:\W97M\MYNA\BB
H:\W97M\MYNA\BC
H:\W97M\NAGEM\H
```

```
H:\W97M\NID\C
H:\W97M\NINEL\A
H:\W97M\NOSTYLE\F
H:\W97M\NOTHERE\C-RU
H:\W97M\OPEY\AX
H:\W97M\OPEY\BB
H:\W97M\OZWER\R
H:\W97M\OZWER\S
H:\W97M\PACOL\A
H:\W97M\PACOL\B
H:\W97M\PANGGIL\B
H:\W97M\PANTHER\T
H:\W97M\PECAS\C
H:\W97M\PERVE\A
H:\W97M\PETMAN\A
H:\W97M\PING\O__MM
H:\W97M\PING\P__MM
H:\W97M\PRI\AC
H:\W97M\SABA\A
H:\W97M\SELIUQ\C
H:\W97M\SHEPMAH\L
H:\W97M\SHEPMAH\M
H:\W97M\SHORE\M
H:\W97M\SHORE\N
H:\W97M\SKAARJ\B
H:\W97M\SMAC\O
H:\W97M\SR\I
H:\W97M\TERMINA\A
H:\W97M\THESEC\K
H:\W97M\THESEC\L
H:\W97M\THESEC\M
H:\W97M\THESEC\N
H:\W97M\THESEC\O
H:\W97M\THUS\EY
H:\W97M\THUS\FK
H:\W97M\THUS\FL
H:\W97M\THUS\FM
H:\W97M\THUS\FN
H:\W97M\THUS\FO
H:\W97M\TUPCA\A
H:\W97M\TWOPEY\B
H:\W97M\VANHACK\A
H:\W97M\VDPAUSE\A
H:\W97M\VMPCK1\EF
```

```
H:\W97M\WRENCH\T
H:\W97M\WRENCH\U
H:\W97M\WRENCH\V
H:\W97M\YAPP\A
H:\WM\IMPOSTER\H
H:\X97M\ADN\F
H:\X97M\ANIS\A
H:\X97M\BARISADA\AE
H:\X97M\BREP\A
H:\X97M\CODEMAS\B
H:\X97M\DIVI\AN
H:\X97M\DIVI\AP
H:\X97M\ELLAR\B
H:\X97M\ELLAR\C
H:\X97M\ELLAR\D
H:\X97M\FONTIME\A
H:\X97M\HOPPER\AG
H:\X97M\LAROUX\OJ
H:\X97M\LAROUX\OK
H:\X97M\LAROUX\OL
H:\X97M\LAROUX\OM
H:\X97M\MANALO\L
H:\X97M\SLACKER\E
```

A.2.2. Macro Malware bis Januar 2002

```
I:\INTENDED\W97M\PEXAS\B
I:\INTENDED\W97M\PRI\AD
I:\INTENDED\W97M\ZINA\A
I:\INTENDED\W97M\ZINA\B
I:\INTENDED\W97M\ZINA\C
I:\INTENDED\W97M\ZINA\D
I:\INTENDED\X97M\DIVI\AO
I:\INTENDED\X97M\YAWN\J
I:\KIT\W97M\NTVCKA\A
I:\KIT\W97M\SATZ\A
I:\TROJAN\X97M\VELTMAR\A
I:\TROJAN\X97M\VELTMAR\B
I:\TROJAN\X97M\VELTMAR\C
```

B. Anhang

Die Quellcodes der implementierten Module, die sich in diesem Anhang befinden sollten, wurden, aufgrund der großen Seitenanzahl (S. 44 - 108) und der schlechten Lesbarkeit überlanger Zeilen, auf der beiliegenden Compact Disk (CD) zur Ansicht gespeichert.