Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

AW2 Ausarbeitung

Benjamin Jochheim

Leichtgewichtige Schadcodeerkennung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einführung

Der wachsende Markt für Mobile Endgeräte, wie Smartphones und Tablet PCs, hat dazu geführt das Anwender viele private Informationen auf diesen Geräten speichern. In den falschen Händen können diese Informationen sehr wertvoll sein. Das hat dazu geführt das es in letzter Zeit vermehrt Attacken auf mobile Endgeräte gegeben hat. In dieser Arbeit geht es um die Erkennung von Schadcode.

In McGraw und Morisett (2000) definieren die Autoren Schadcode als *„jeden Code der zu einem Softwaresystem hinzugefügt, geändert oder entfernt wurde um absichtlich Schaden zuzufügen oder die beabsichtigte Funktion eines Systems zu unterwandern“*. Auf traditionellen PCs gibt es verschiedene etablierte Methoden um Schadcode abzuwehren. Diese Methoden, wie z.B. Virenscanner können jedoch nicht in der selben Form auf mobile Endgeräte übertragen werden, weil diese Geräte im Vergleich zu PCs in vielerlei Hinsicht beschränkt sind. Im Vergleich zu PCs sind langsame Rechengeschwindigkeit, geringe Speicherressourcen, limitierte Akkuleistung nur einige der Herausforderungen die ein Abwehrsystem auf einem mobilen Endgerät meistern muss.

Im folgenden geht es darum Methoden zu betrachten die im Hinblick auf die o.g. Einschränkungen leichtgewichtig genug sind um auf einem mobilen Endgerät eingesetzt zu werden. Dabei geht es um die Erkennung einer immer häufiger auftretenden Art von Attacken, den „Embedded Shellcode“-Angriffen. Bei dieser Art von Angriffen wird Schadcode in Dateien versteckt die normalerweise keine Gefahr für ein Computersystem darstellen können (wie z.B. PDF-Dokumente). Der eingebettete Schadcode kommt beim öffnen der präparierten Datei unbeabsichtigt zur Ausführung, da der Angreifer Fehler in der Programmierung der Software ausnutzt die diese Dateien öffnet und interpretiert.

1.1 Problemstellung

Es sollen die Grundlagen geschaffen werden um ein leichtgewichtiges System zur Erkennung von eingebettetem Schadcode zu erstellen. Dabei wird davon ausgegangen das ein Schadcodefragment stets umgeben von regulären Daten ist. Wenn sich ausführbarer Code an stellen befindet an denen er nicht erwartet wird, wird die Annahme getroffen das es sich um Schadcode handelt.

Nach diesen Grundannahmen wird zur Erkennung von Schadcode ein System benötigt, welches an jeder beliebigen Stelle eines Datenstroms erkennen kann ob es sich um ausführbaren Binärcode handelt oder ob es sich um andere Daten handelt. Somit soll es sich bei dem Verfahren um einen *Zero-Knowledge-Ansatz* handeln, der nicht (wie z.B. Virenscanner) auf Signaturen basiert, sondern ohne vorherige Kenntnis eines bestimmten Angriffs nutzbar ist.

2 Verwandte Arbeiten

In diesem Kapitel werden verschiedene Verfahren vorgestellt, die ähnlich zu unserer Fragestellung sind.

2.1 Net-Entropy

Um Angriffe auf ein Netz möglichst früh zu erkennen haben sich „Intrusion Detection Systeme“ (IDS) bewährt. Ein IDS-System verhält sich meist passiv und zeichnet alle besonderen Ereignisse im Netz auf. Dabei gibt es auch Systeme welche das Verhalten der Netzwerkteilnehmer auf der Anwendungsschicht überwachen.

Ein großes Problem stellen dabei verschlüsselte Protokolle dar, die nicht ohne größere Hürden überwacht werden können. Um verschlüsselte Netzwerkkommunikation indirekt überwachen zu können nutzen die Autoren Olivain und Goubault-Larrecq in Olivain und Goubault-Larrecq (2006) Entropie-werte der Kommunikation. Erfahrung hat gezeigt das der Beginn einer verschlüsselten Kommunikation für Angriffe am anfälligsten ist. Das Net-Entropy Verfahren zeichnet verschiedene typischen Entropie-Verläufe einer Netzwerkkommunikation auf und kann anschließend warnen wenn sich starke Abweichungen ergeben, die auf einen Angriff hindeuten können. [1]

Die Kommunikation kann also nicht direkt mitgelesen werden, allerdings kann das ihr Verfahren ungewöhnliche Entropie-werte erkennen. Dadurch lassen sich verschiedene Angriffsarten wie Buffer-Overflow Angriffe trotz Verschlüsselung entdecken.

Relevant ist das Verfahren weil zum einen demonstriert wird das eine zuverlässige Erkennung von Inhalts-Mustern mittels Entropie möglich ist. Zum anderen zeigt es auch das Entropie auch für den Einsatz der Echtzeitüberwachung von Kommunikation geeignet ist.

[1] Auf http://www.lsv.ens-cachan.fr/~olivain/net-entropy/ gibt es eine Referenzimplementierung dieses Verfahrens

2.1.1 Entropie Schätzfunktion

Paninski stellt in Paninski (2004) eine Schätzfunktion für die Berechnung der Shannon-Entropie

vor (siehe 3.1.1). Mit der Schätzfunktion ist es möglich die Entropie eines Datenstromes abzuschätzen, ohne den gesamten Datenstrom zu kennen. Die Schätzfunktion wird von Net-Entropy verwendet um die Entropie von noch nicht erhaltenen Datenpaketen abzuschätzen. Zum einen ermöglicht die Schätzfunktion ein schnelleres reagieren von Net-Entropy. Zum anderen ermöglicht die Schätzfunktion auch eine einfachere Verarbeitung. Denn eine Alternative zur Schätzfunktion ist die Berechnung der Entropie mittels kleinen, sich überlappenden Fenstern, wie in 3.1.1 dargestellt. Die Verarbeitung mittels Fenstern hat den Nachteil das Daten (durch die Überlappung) mehrfach verarbeitet werden müssen.

2.2 Computer-Forensik

Im Zusammenhang mit Computer Forensik kommt häufig die Problemstellung auf das ein Forensiker viele Dateifragmente besitzt, diese jedoch keinen Dateien zuordnen kann. Ist eine Zuordnung nicht möglich lautet die die erste Fragestellung häufig:

Zur welchem Dateityp gehört dieses Fragment?

Das Problem tritt z.B. auf wenn Dateifragmente auf Festplatten wiederhergestellt werden deren Inhalt bereits zum Teil überschrieben wurde. Aber auch beim zusammensetzen von Netzwerk-Datenströmen treten ähnliche Fragestellungen auf. Daher wurden für diesen Bereich viele Verfahren entwickelt, die sich speziell mit der Klassifizierungsfrage von Dateifragmenten auseinandersetzen. Wie in Kapitel 1.1 erläutert sind für unsere Art der Malware-Erkennung nicht die frage des Dateityps entscheidend, sondern nur die Unterscheidung ob es sich um *ausführbaren* Binärcode oder *nicht-ausführbaren* Binärcode handelt.

In der Computer Forensik werden Daten meist „offline“ analysiert. Das bedeutet das ein zu analysierender Vorfall bereits passiert ist und die Analyse daher länger dauern kann. Im Gegensatz dazu sollen Verfahren auf mobilen Endgeräten möglichst zeitsparend und im Hinblick auf den benötigten Speicherplatz, effizient sein. Bei den im weiteren Vorgestellten Arbeiten wurde daher meist nicht auf das Laufzeitverhalten geachtet.

2.2.1 Verwandte Arbeiten der Computer-Forensik

Hier werden relevante Arbeiten aus dem Bereich der Computer Forensik vorgestellt. Die Arbeiten entstanden im vergangenen Jahrzehnt. Ein Manko bei den Arbeiten ist die Vergleichbarkeit bezgl. der Erkennungsgenauigkeit. Viele der Arbeiten bauen aufeinander auf, allerdings ist ein direkter Vergleich nicht immer sinnvoll möglich. Um ein Verfahren vergleichen zu können werden stets Testdaten benötigt. Nicht in jeder Arbeit wird angegeben ob und in welchem Umfang getestet wurde. Ein weiteres Problem sind die unterschiedlichen Größen der getesteten Dateifragmente. Es ist leicht einsehbar das bei einer steigend Fragmentgröße eine bessere Erkennung möglich ist. Am günstigsten wäre es wenn jede Arbeit mit einem gemeinsamen Menge an Testdateien auf gleich großen Fragmentgrößen arbeiten würde.

Die hier vorgestellten Verfahren arbeiten nach Cluster-Methoden. Dabei werden vorab viele Dateien eines Dateityps gesammelt und nach einem bestimmten Schema Merkmale [2] extrahiert. Wenn die Art der Merkmale gut gewählt wurde, bilden unterschiedliche Dateitypen Häufungen (Cluster) im Merkmalsraum. Anhand verschiedener Distanzmetriken kann die nähe zu bekannten Dateitypen im Merkmalsraum erkannt werden.

In McDaniel und Heydari (2003) stellen die Autoren Mc Daniel et al. zwei für uns relevante Verfahren zur Dateityp-Identifizierung vor. Es handelt sich dabei um Verfahren mit denen sich der Dateityp einer ganzen Datei erkennen lässt. Kleine Ausschnitte von Dateien wurden in der Arbeit nicht betrachtet. Das erste Verfahren *Byte Frequency Analysis* (BFA) berechnet für jeden Dateityp die Häufigkeitsverteilung der Bytes für eine Datei. Als Fingerabdruck wird somit ein *Histogramm* der Byte-Häufigkeiten erstellt. Das zweite Verfahren *Byte Frequency Cross-correlation* (BFC), verbessert BFA um die Korrelation zwischen den Häufigkeiten der einzelnen Byte-werte. Haben zwei Byte-werte eine gleich hohe Häufigkeit haben sie auch eine hohe Korrelation.

Um BFA auch auf Dateifragmente anwenden zu können erzeugten die Autoren Li et al. in Li u. a. (2005), pro zu erkennendem Dateityp, eine Vielzahl gleich großer Abschnitte. Für diese Abschnitte erstellten sie eine Cluster-Analyse [3] und konnten so mehrere Cluster-Zentren pro Dateityp finden. Die verschiedenen Cluster-Zentren pro Dateityp repräsentieren die unterschiedlichen Bereiche einer Datei, wie z.B. bei JPEG-Dateien erst die unkomprimierten Metadaten, gefolgt vom komprimiertem Inhalt. Die Autoren zeigten das dieses Verfahren im Header-Bereich einer Datei sehr hohe Erkennungsraten aufweist. Leider sinkt die Erkennungsrate sehr stark bei den darauf folgenden Dateiabschnitten.

In Karresand und Shahmehri (2006) stellen die Autoren eine Methode, Oscar genannt, zur Dateifragment-Erkennung vor. Oscar erreicht speziell bei JPEG-Dateien eine sehr hohe Er-

[2] Diese Merkmale werden in verschiedenen Arbeiten auch als Datei-„Fingerprints“ bezeichnet
[3] mittels k-Means-Algorithmus

kennungsrate. Die Oscar-Methode verwendet zur Dateierkennung die Durchschnittliche sowie die Standard-Abweichung der Häufigkeitsverteilung verschiedener Dateitypen.

Die Autoren des Papers Hall und Davis (2007) mit dem Titel „Sliding Window Measurement for File Type Identification“ erstellen Datei-Fingerabdrücke indem sie die LZW-Kompression Welch (1984) als Kriterium die Komprimierbarkeit [4] heranziehen. Dabei nutzen sie kleine Fenster um eine Funktion der Komprimierbarkeit,ähnlich dem im Abschnitt 3.1.1 gezeigten, für eine Datei aufzustellen. Nach dem erstellen der Kompressions-Funktion kann der so erzeugte „Fingerabdruck“ mit gespeicherten verglichen werden. Das Verfahren eignet sich nur zum Vergleich ganzer Dateien, jedoch nicht für Dateifragmente. In der Arbeit vergleichen die Autoren die LZW-Komprimierbarkeit mit der Entropie-Funktion. Dabei zeigt sich das die Entropie-Funktion stets sehr ähnlich der LZW-Komprimierbarkeit aussieht. Ihre Messungen führten sie mit einem wandernden Fenster (Sliding Window) aus, das ähnlich auch beim STFT-Verfahren in Abschnitt 3.1.2 angewandt wird.

Mit Erbacher und Mulholland (2007) stellen die Autoren Erbacher und Mulholland eine große Auswahl von 13 verschiedenen Statistischen Maßeinheiten für die Erkennung von Dateitypen vor. Um die Messgrößen anzuwenden nutzen Sie überlappende Fenster. Mittels dieser Fenster stellen sie Messungen an und generieren Funktionen der Byte-Position für eine Datei. Dabei stellen sie fest das Fenstergrößen zwischen 256byte und 1Kb am geeignetsten sind um Dateitypen zu erkennen.

2.3 Antiviren Programme

Ein naheliegender Schritt wäre es sich die Funktionsweise heutiger Antivirenprogramme (AV) anzusehen. Die seit langem von den Virenscannern eingesetzten Signaturen laufen (wie eingangs in Kapitel 1.1 erwähnt) dem Zero-Knowledge Ansatz zuwider. Es gibt allerdings einige neue Entwicklungen die derzeit noch nicht von den führenden Virenscanner-Herstellern eingesetzt werden, aber Potential für zukünftige Produkte haben. Die Verfahren sollen hier kurz in dynamische und statische Verfahren eingeteilt werden.

2.3.1 Dynamische Verfahren

Es gibt Ansätze das Verhalten von Programmcode zu verstehen. Ein Ansatz sind sog. API-Hooks. Dabei überwacht ein Programm während der Ausführung einen teil der API-Schnittstellen des Betriebssystems. Anhand von vorab festgelegten Aufrufmustern kann erkannt werden ob der Code schädlich ist. Dabei kann das Programm vor Ausführung schädli-

[4] Es wird die Anzahl der Änderungen am Kompressions-Wörterbuch als Komprimierbarkeits Maß verwendet.

cher Programmteile angehalten werden. In Bai u. a. (2009) stellen die Autoren ein Verfahren das nach diesem Muster vorgeht vor.

2.3.2 Statische Verfahren

Zu den statischen Verfahren gehören auch die eingangs erwähnten Signatur-Verfahren die hier aber keine Rolle spielen sollen.

Bedeutung von Code Verstehen

Ein anderer statischer Ansatz ist es, den Code offline zu analysieren. Dabei wird aus dem ausführbaren Code ein Graph erstellt (im wesentlichen der Umgekehrte Weg eines Compilers). Die erzeugten Graphen stellen die Semantik des Codes dar und können auf schädlichen Code untersucht werden. Eines dieser Verfahren wird in Lee u. a. (2010) vorgestellt. Eine Variation wird in Park u. a. (2010) vorgestellt. Dort wird der Aufruf-Graph nicht aus dem Code direkt erzeugt, sondern es wird der zu untersuchende Codeabschnitt in einer Sandbox [5] ausgeführt. Zeitgleich werden die API-Aufrufe innerhalb der Sandbox überwacht. Die Autoren zeigen das dieses Verfahren genauer sein kann als eine Überprüfung des Codes, ohne ihn auszuführen.

Um AV Software zum umgehen, verwenden viele Viren *Code Obfuscation* Verfahren die eine Signaturerkennung sehr erschwert. Auch eine Erkennung über Aufruf-Graphen ist damit nicht mehr möglich. Daher sind AV Hersteller dazu übergegangen Merkmale zu verwenden die sich auch nach einer Verschleierung des Codes nicht ändern.

Ein Autor(S.J. Stolfo), der auch das BFA-Verfahren in Abschnitt 2.2.1 verbessern konnte, stellte in Schultz u. a. (2001) ein Verfahren namens *strings* vor.

Ein weiteres erfolgreiches Verfahren ist *KM* in J.Z. Kolter (2004), welches u.a. als Statistisches Maß die importierten APIs innerhalb von DLL [6] Dateien nahm.

Die Verfahren unterscheiden sich in ihren Ansätzen haben jedoch gemeinsam das sie statistische Merkmale von kleinen Datenblöcken extrahieren, diese Merkmale durch einen Klassifikator in bestimmte Klassen einteilen lassen und anschliessend die *Reihenfolge* der Klassifikationen des Datentstromes als Signatur nutzen.

[5] eine virtuelle Maschine in der der Code dem System nicht schaden kann.
[6] Dynamic Link Library, das Windows Dateiformat für dynamisch linkbare Bibliotheken

Locality-sensitive hashing

Ein Erweiterung der statischen Signaturen die für die weitere Arbeit interessant sein könnte, ist LS-hashing, eine besondere Gruppe von Hash-Verfahren.

Unter Locality-sensitive hashing (LSH) versteht man Hash-Verfahren die bei zwei nahezu identischen Daten zwei Hashes liefern die „nahe“ bei einander sind. Damit können z.B. Ähnlichkeiten zwischen Dateien ermittelt werden. Ein bekannter Vertreter ist das *MinHash* Broder (1997) Verfahren. Im Zusammenhang mit Signaturen werden bei den im Umlauf befindlichen Virenscannern auch ähnliche Techniken angewandt. Es ließ sich nicht in Erfahrung bringen mit welchen Verfahren die Virenscanner-Hersteller arbeiten.

3 Ansätze für ein Verfahren zur Erkennung von Schadcode

Das in Schmidt u. a. (2011) vorgestellte Verfahren soll hier kurz erläutert werden. Um einen Datenstrom effizient verarbeiten zu können wird mit sich verschiebenden Fenstern gearbeitet.

Auf einen Eingabe-Datenstrom wird zuerst wird die Entropie wie in 3.1.1 angewandt. Je 4 aufeinanderfolgenden Entropiewerten wird eine Fouriertransformation (wie in 3.1.2) ausgeführt, die dann ein Entropiespektrum ergeben. Anhand des Entropiespektrums lässt sich in den den meisten Fällen erkennen ob es sich um ausführbaren Code oder um andere Daten handelt.

Die Klassifikation (siehe 3.1.3) des so erhaltenen Entropiespektrums scheint mit künstlichen Neuronalen Netzen sinnvoll lösbar zu sein.

3.1 Techniken

Um ein effizientes System zur Erkennung von Schadcode zu konzipieren werden verschiedene Techniken eingesetzt die in diesem Kapitel kurz erläutert werden. Dabei wird kurz vorgestellt wie die jeweilige Technik funktioniert und an welcher Stelle sie eingesetzt werden soll.

3.1.1 Entropie

Entropie nach Shannon Shannon (1948) ist ein Maß für Unsicherheit. Es kann auch als ein Maß für die komprimierbarkeit gelten. Die Entropie nach Shannon muss immer im Zusammenhang mit der Alphabetgröße betrachtet werden. Im folgenden geht es um Alphabete mit einer Größe von einem Byte.

Die in hier angewandte Entropie (Formel 3.1.1) H liegt somit immer im Bereich von 0 bis 8.

$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i)$

Die Variable X_i ist in unserem Fall eine Zufallsvariable mit 256 möglichen Werten, aus dem Datenblock X. Wobei $p(X_i)$ die relativen Häufigkeiten in X widerspiegeln. Das Ergebnis liegt immer im Bereich von 0 bis 8.

Die Entropie kann für eine ganze Datei berechnet werden und liefert dann genau einen Entropie-wert. Um den Entropieverlauf festzustellen verwenden wir ein Fenster das über die Datei bewegt wird und jeweils für einen kleinen Abschnitt die Entropie berechnet. Das Fenster überlappt mit den vorhergehenden Fenster um eine festgelegte Anzahl von Byte. Durch diese Methode ist es möglich eine „Entropie-Funktion“ einer Datei darzustellen. Anhand der Entropiefunktion kann man Abschnitte wie beispielsweise Header und Nutzdaten einer Datei sehr gut erkennen.

3.1.2 Frequenzanalyse

Die in Abschnitt 3.1.1 gezeigte Entropie kann auch als digitales Signal verstanden werden. Im Bereich Signalverarbeitung ist die Fouriertransformation (FT) ein etabliertes mathematisches Verfahren um ein periodisches Signal vom Zeitbereich in den Frequenzbereich zu transformieren. Nach einer FT lassen sich die Frequenzanteile eines Signals sehr gut erkennen.

Wir werden später die FT auf die Entropie-Funktion anwenden um diese näher zu analysieren. Dabei werden wir dieselben Fenstertechniken anwenden wie in Abschnitt 3.1.1 beschrieben. Das bedeutet das die FT nicht auf ein vollständiges Signal angewendet wird, sondern nur auf einen kleines wanderndes Fenster. Dieses Vorgehen nennt sich auch *short term fourier transform*, kurz STFT und berechnet sich mit der folgenden Formel:

$$STFT[H](m,\omega) = \sum_{n=-\infty}^{\infty} X(n)W(n-m)e^{i\omega n}.$$

Dabei muss bei der STFT ein Kompromiss zwischen hoher Zeitlicher Auflösung (bei kleinen Fenstergrößen) und hoher Frequenzauflösung (bei großen Fenstern) eingegangen werden Meyer und Mildenberger (2002).

3.1.3 Klassifikation

Ein Klassifikator ist ein Stück Software, das Eingabedaten eine zugehörige Klasse zuordnet. Ein Beispiel für den Einsatz ist die Zeichenerkennung im OCR-Bereich. Dabei werden aus

Eingabebildern (Pixelmustern) der einzelnen Buchstaben die zugehörigen Klassen (Buchstaben des Alphabets) zugeordnet. Nach Eingabe eines Pixelmusters kann der Klassifikator die zugehörige Klasse feststellen. Wenn ein Klassifikator eine gute *Generalisierung* aufweist ist eine korrekte Klassenzuordnung auch bei „verzerrten“ Eingabedaten möglich.

Die Anzahl an Eingabewerten des Klassifikators wird als *Dimension* bezeichnet. Bei den meisten Praxisrelevanten Klassifikatoren ist die Dimension sehr groß, so dass eine direkte Programmierung der Klassengrenzen nicht mehr trivial möglich ist. Daher werden Techniken aus dem Bereich der Künstlichen Intelligenz benötigt. Ein KI-Verfahren aus dem Bereich Maschinenlernen sind die künstlichen Neuronalen Netze (Artificial Neural Networks, ANN).

In seiner Masterthesis Harris (2007) stellt Harris fest, das sich künstliche Neuronale Netze (ANNs) gut für die Dateityp-Identifikation nutzen lassen. Dabei stellt er auch fest das ANNs nicht besser als andere Lernverfahren sind.

4 Abgrenzung und Diskussion

In Kapitel 2 wurden verschiedene relevante Arbeiten die zur Erkennung von Schadsoftware beitragen können vorgestellt. Hier soll eine Abgrenzung gegenüber anderen Arbeiten stattfinden und Ziele für das weitere Vorgehen definiert werden.

4.1 Aufbereitung der Daten

Das Problem wurde auf die Unterscheidung von Ausführbarem und nicht Ausführbarem Code eingeschränkt. Ausführbarer Code sieht auf verschiedenen Prozessorarchitekturen natürlich unterschiedlich aus. Dabei besteht die bislang noch unbestätigte Hoffnung das es möglich ist ausführbaren Code anhand seiner Struktur zu erkennen. Diese Struktur sollte mit Hilfe von Statistischen Maßzahlen soweit aufbereitet werden das sie automatisch erkennbar wird.

Bei der Erkennung von Byte-folgen wird schnell klar das eine einfache Datenbank mit Bytefolgen die auf ausführbaren Code hindeutet nicht praktikabel ist und wie eingangs erläutert auch kein Zero-Knowledge-Ansatz wäre.

Bei der Verwendung von statistischen Maßzahlen sollten die in Erbacher und Mulholland (2007) vorgestellten Maßzahlen daraufhin geprüft werden in wie fern Sie brauchbar sind. Dabei sind zwei Faktoren besonders interessant: Lässt sich die Maßzahl

- schnell und effizient berechnen?
- inkrementell berechnen?

Weil *Datenströme* auf Schädlichkeit geprüft werden sollen, wird schnell klar das es eine Anwendung der Fenstertechnik mit Überlappung geben wird (Ähnlich der in 3.1.1 vorgestellten). Eine Möglichkeit der inkrementellen Berechnung (z.B. in der Berechnungsweise eines *Moving Average Filters*) einer Maßzahl wäre daher wünschenswert, da nur so ein wiederholtes Berechnen von Ergebnissen vermieden wird.

Die AV-Methoden aus 2.3.2 sehen vielversprechend aus, wobei einige wie *KM* ausscheiden, da Sie ein tieferes Verständnis (im sinne von parsen) der vorliegenden Dateien voraussetzen.

Es wäre günstig wenn solch eine tiefe Analyse von Code vermieden werden kann, damit das Verfahren auf möglichst vielen Platformen und Prozessorarchitekturen einsetzbar bleibt.

Welche Maßzahl verwendet werden soll, sollte experimentell ermittelt werden. Eventuell ist es sinnvoll mehrere hintereinander angewandte Maßzahlen zu nutzen. In 3 wird ein Verfahren vorgestellt, das als Teil des SKIMS-Projekts [1] in die nähere Auswahl für einen Schutz für Mobiltelefone kommt und in Schmidt u. a. (2011) beschrieben wird. Dieses Verfahren nutzt die Entropie sowie sowie eine Anschließende Fourier-Analyse.

4.1.1 weiteres Vorgehen

Egal welches Verfahren zum Einsatz kommt, im weiteren Verlauf der Arbeit müssen unbedingt umfangreiche Tests mit real vorkommenden Daten stattfinden. Dazu wird es sinnvoll zu sein sich einen großen Testdatensatz aufzubauen mit Dateitypen die typischerweise auf mobilen Endgeräten vorkommen (wie z.B. PDF,MS-Office,ARM-Code-Dateien). Um eine möglichst große Auswahl zu haben sollten diese Dateien möglichst aus verschiedenen Quellen stammen. Mit diesen Testdaten können die vorgeschlagenen Messgrößen auf Zuverlässigkeit geprüft werden.

Besonders wichtig ist die Zuverlässigkeit, besonders im Hinblick auf Falschmeldungen (nicht-Ausführbarer Code, der als Ausführbar gemeldet wird).

Nach dem Aufbau der Testdatensätze soll zuerst das Verfahren Schmidt u. a. (2011) getestet und auf seine Zuverlässigkeit sowie Praxistauglichkeit geprüft werden. Im Zusammenhang mit diesem Verfahren muss der Paninski Entropie-Abschätzer aus 2.1 unbedingt getestet werden. Eine weitere Verbesserung könnte die Nutzung von *Wavelets* anstatt der Fourieranalyse sein. Durch den Einsatz von Wavelets könnte die Frequenzauflösung besser werden.

4.2 Automatische Erkennung in den aufbereiteten Daten

In den Arbeiten zur Computer Forensik, sowie den AV Arbeiten in denen eine automatische Erkennung von Dateifragmenten bzw. Malware-Erkennung durchgeführt wird, werden verschiedenste Lernverfahren von künstlichen Neuronalen Netzen, bis zu Support Vector Machines genutzt. Das verwendete Lernverfahren scheint für den Erfolg meist nicht den entscheidenden Unterschied zu geben. Daher wird hier nicht der Fokus der weiteren Arbeit liegen.

[1] SKIMS: Schichtenübergreifendes kooperatives Immunsystem für mobile, mehrseitige Sicherheit

Für die Praktische Anwendung scheinen hier eher gute Laufzeiteigenschaften wichtig zu sein, damit das Gesamtsystem performant bleibt.

4.3 Grenzen

Eine in Diskussionen häufig gestellte Frage um dieses Thema war

Kann das Verfahren umgangen werden, wenn der Angreifer von solch einem Detektionsmechanismus Kenntnis erlangt?

Meine persönliche Meinung ist ein klares „Ja“. Denn Wettrüsten zwischen AV Herstellen und Virenprogrammierern zeigt ganz klar das es einen Wettkampf zwischen beiden Parteien gibt der mit ständig neuen Ideen geführt wird.

Allerdings kann solch ein Verfahren die Latte für Virenprogrammierer wieder etwas höher hängen. Manipulationsversuche die ich mit speziell mit dem Verfahren aus 3 vorstellen könnte wäre ein Manipulationsversuch an der Entropie. Die Entropie sinkt wenn auf einem Abschnitt nur wenig Änderungen passieren. Daher wäre ein naiver Ansatz sich wiederholende Füllanweisungen in den Code einzufügen die Semantisch keine Auswirkungen haben, wie z.B. die X86-Assembler Anweisung *NOP* [2]. Dadurch wird der Code aber unweigerlich größer, was einen Angriff in manchen Situationen erschweren kann.

[2] No Operation, die CPU wartet einen Taktzyklus, hat aber keine weiteren Auswirkungen

Literaturverzeichnis

[Bai u. a. 2009] BAI, Lili ; PANG, Jianmin ; ZHANG, Yichi ; FU, Wen ; ZHU, Jiafeng: Detecting Malicious Behavior Using Critical API-Calling Graph Matching. In: *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*. Washington, DC, USA : IEEE Computer Society, 2009 (ICISE '09), S. 1716–1719. – URL http://dx.doi.org/10.1109/ICISE.2009.494. – ISBN 978-0-7695-3887-7

[Broder 1997] BRODER, Andrei Z.: On the Resemblance and Containment of Documents. In: *In Compression and Complexity of Sequences (SEQUENCES 97*, IEEE Computer Society, 1997, S. 21–29

[Erbacher und Mulholland 2007] ERBACHER, Robert F. ; MULHOLLAND, John: Identification and Localization of Data Types within Large-Scale File Systems. In: *Systematic Approaches to Digital Forensic Engineering, IEEE International Workshop on* 0 (2007), S. 55–70. ISBN 0-7695-2808-2

[Hall und Davis 2007] HALL, Gregory A. ; DAVIS, Wilbon P.: *Sliding Window Measurement for File Type Identification*. http://www.mantech.com/cfia2/SlidingWindowMeasurementforFileTypeIndentification.pdf. 2007

[Harris 2007] HARRIS, Ryan M.: *Using artificial neural networks for forensic file type identification*, Purdue University, Diplomarbeit, 05 2007. – URL http://web.ics.purdue.edu/~rmharris/Thesis.pdf

[J.Z. Kolter 2004] J.Z. KOLTER, M.A. M.: Learning to detect malicious executables in the wild. In: *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, ACM, 2004, S. pp. 470–478

[Karresand und Shahmehri 2006] KARRESAND, Martin ; SHAHMEHRI, Nahid: Oscar - File Type Identification of Binary Data in Disk Clusters and RAM Pages. In: *SEC'06*, 2006, S. 413–424

[Lee u. a. 2010] LEE, Jusuk ; JEONG, Kyoochang ; LEE, Heejo: Detecting metamorphic malwares using code graphs. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA : ACM, 2010 (SAC '10), S. 1970–1977. – URL http://doi.acm.org/10.1145/1774088.1774505. – ISBN 978-1-60558-639-7

[Li u. a. 2005] LI, Wei-Jen ; WANG, Ke ; STOLFO, Salvatore J.: Fileprints: Identifying File Types by n-gram Analysis. In: *IEEE Information Assurance Workshop*, 2005

[McDaniel und Heydari 2003] MCDANIEL, Mason ; HEYDARI, M. H.: Content Based File Type Detection Algorithms. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9*. Washington, DC, USA : IEEE Computer Society, 2003 (HICSS '03), S. 332.1–. – URL http://dl.acm.org/citation.cfm?id=820756.821828. – ISBN 0-7695-1874-5

[McGraw und Morisett 2000] MCGRAW, G. ; MORISETT, G.: Attacking malicious code: A report to the Infosec Research Council, IEEE, 2000, S. 33–35

[Meyer und Mildenberger 2002] MEYER, M. ; MILDENBERGER, O.: *Grundlagen der Informationstechnik: Signale, Systeme und Filter ; mit 33 Tabellen*. Vieweg, 2002. – URL http://books.google.com.ag/books?id=_Ep59DSiNEcC. – ISBN 9783528039318

[Olivain und Goubault-Larrecq 2006] OLIVAIN, Julien ; GOUBAULT-LARRECQ, Jean: Detecting Subverted Cryptographic Protocols by Entropy Checking / Laboratoire Spécification et Vérification, ENS Cachan, France. URL http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2006-13.pdf, Juni 2006 (LSV-06-13). – Research Report. 19 pages

[Paninski 2004] PANINSKI, Liam: Estimating Entropy on m Bins Given Fewer than m Samples. In: *IEEE Transactions on Information Theory* 50 (2004), Nr. 9, S. 2200–2203

[Park u. a. 2010] PARK, Younghee ; REEVES, Douglas ; MULUKUTLA, Vikram ; SUNDARAVEL, Balaji: Fast malware classification by automated behavioral graph matching. In: *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. New York, NY, USA : ACM, 2010 (CSIIRW '10), S. 45:1–45:4. – URL http://doi.acm.org/10.1145/1852666.1852716. – ISBN 978-1-4503-0017-9

[Schmidt u. a. 2011] SCHMIDT, Thomas C. ; WÄHLISCH, Matthias ; JOCHHEIM, Benjamin ; GRÖNING, Michael: WiSec 2011 Paper: Context-adaptive Entropy Analysis as a Lightweight Detector of Zero-day Shellcode Intrusion for Mobiles. In: *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)* (2011). – Accepted for Publication

[Schultz u. a. 2001] SCHULTZ, M.G. ; ESKIN, E. ; ZADOK, E. ; STOLFO, S.J.: Data mining methods for detection of new malicious executables. In: *IEEE Symposium on Security and Privacy* (2001), S. pp. 38–49

[Shannon 1948] SHANNON, C. E.: A Mathematical Theory of Communication. In: *Bell System Technical Journal* 27 (1948), July, October, S. 379–423, 623–656

[Welch 1984] WELCH, T. A.: A Technique for High-Performance Data Compression. In: *Computer* 17 (1984), June, S. 8–19. – URL http://dl.acm.org/citation.cfm?id=1319729.1320134. – ISSN 0018-9162