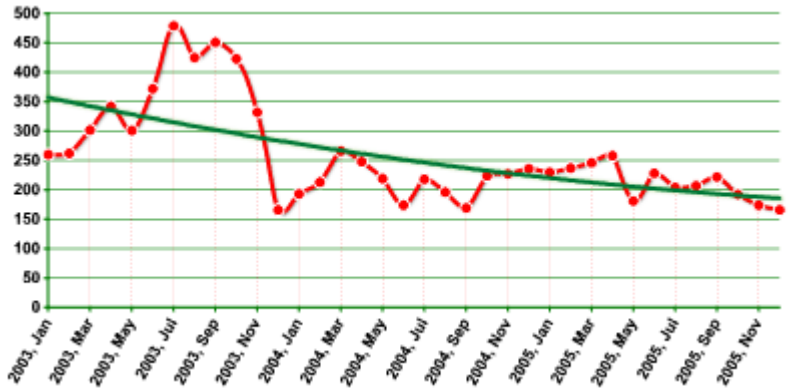
Malwareanalyse

Dr. Heiko Patzlaff
Siemens
Corporate Technology
IC CERT

Information &
Communications
Computer
Emergency
Response
Team

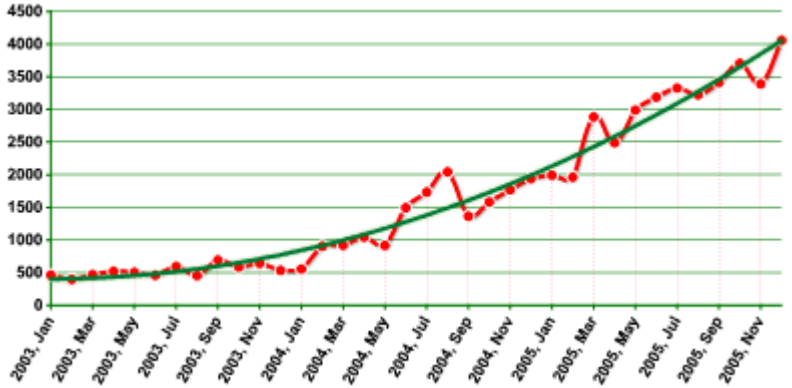
Warum ist Malwareanalyse so wichtig?

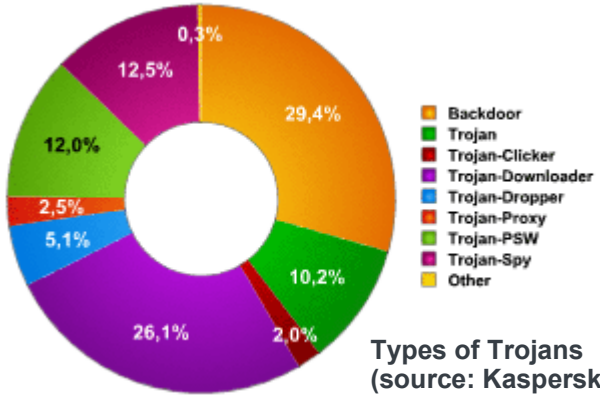
Malwarestatistiken



Number of Worms and Viruses (source: Kaspersky)

Number of Trojans (source: Kaspersky)





Types of Trojans (source: Kaspersky)

Information & Communications Computer Emergency Response Team

Bei Malware Vorfällen ist man an Fragen interessiert wie:

- Ist der Rechner kompromitiert?
- Wie wurde er kompromitiert?
- Wann wurde er kompromitiert?
- Welcher Schaden wurde angerichtet?
- Sind andere Computer betroffen?
- Wie kann die Malware entfernt werden?

Bei der *Analyse von Malware* hat man lediglich eine oder mehrere Dateien vorliegen **ohne kontextuelle Informationen**. Hier geht es um die Beantwortung von Fragen wie:

- Ist die Datei Malware?
- Wird sie von Anti-Virus erkannt?
- Wie verbreitet sie sich?
- Welche Schadfunktionen hat sie?
- Wie nistet sie sich in das System ein?

Aus der Beantwortung dieser und ähnlicher Fragen hofft man Rückschlüsse auf die Fragen der vorhergehenden Folie zu erhalten.

Malware Analyse hat enge Beziehungen zur '*compromise detection*', also der Erkennung von Infektionen.

Die Methoden der '*compromise detection*' lassen sich grob klassifizieren:

- Erkennung anhand der Eigenschaften
- Erkennung anhand des Verhaltens
- Erkennung anhand des Kontexts
- andere Methoden ('whitelisting', Korrelationen, etc.)

Statische Analyse - Überblick

1. Dateiformaterkennung
2. Packererkennung
3. Compilererkennung
4. heuristische Virenerkennung
5. Funktionalitätserkennung (strings, exports, UUIDs)

Im folgenden soll das Augenmerk auf einer möglichen Automatisierung liegen.

Dateiformaterkennung

Binärdateien: Signaturen im Dateiheader

z.B.

- **0x4d 0x5a** ('MZ') – DOS/Windows EXE
- **0x4d 0x53 0x46 0x54** ('MSFT') – MS Type Library
- **0xca 0xfe 0xba 0xbe** – Java class file

Textdateien: lexikalische und syntaktische Analyse. Insbesondere Entscheidung ob ausführbares Script vorliegt.

Dateiformaterkennung - Scripte

Zur Analyse können häufig die Parser der Scriptinterpreter verwendet werden. Dazu Kapselung des unbekannten Textinhaltes und Auswertung durch den Scriptparser.

Beispiel javascript:

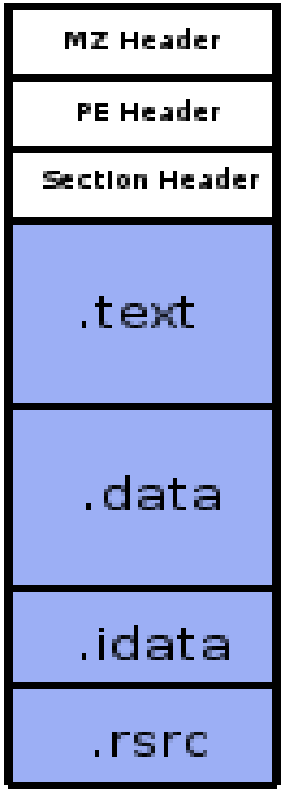
```
function _mytest() {
        <originaler Dateiinhalt>
}
```

Test z.B. mittels Windows Scripting Host

```
cscript.exe //B //E:jscript <modifizierte Datei> || echo Kein JScript
```

Schwierigkeiten bei BAT/CMD Dateien.

Einschub: PE-File Format



Packererkennung

Dinge die wir erkennen wollen:

Packers, Encryptors, Obfuscators, Archivers, Binders

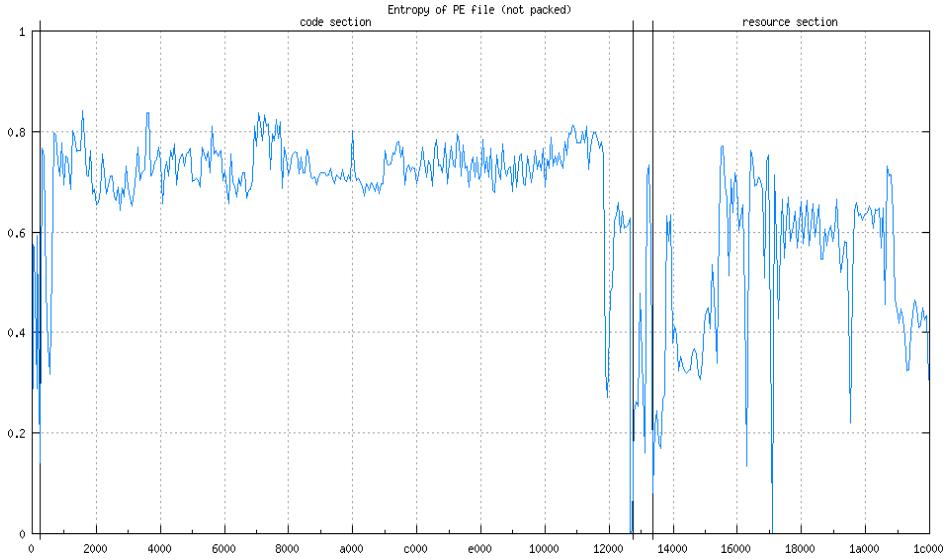
Spezifische Erkennung anhand von *entrypoint* Signatur oder anderen Dateimerkmalen (Dateistruktur, Sektionsnamen, Imports, etc.).

Generische Packererkennung:

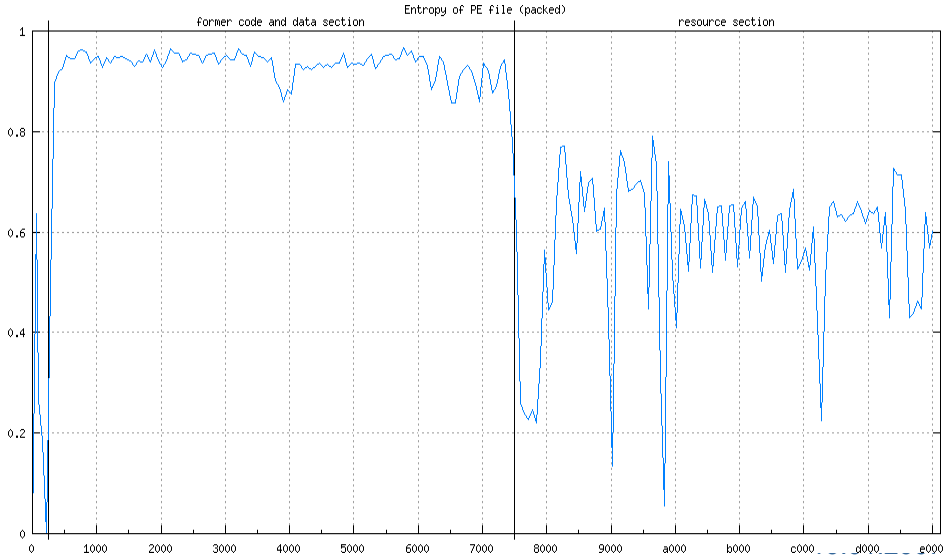
- schreibbare Codesection
- wenige Imports
- statistische Methoden

Verwendung der Entropy zur Erkennung gepackter Dateien

calc.exe



calc.exe
(gepackt)



Compilererkennung

Dinge die man erkennen will:

- Typ: EXE, DLL, ActiveX
- Programmiersprache: Code (C, C++, Pascal) oder Pseudocode (J++, .NET, P-Code)
- Compiler/Linker, Framework: VC++, Delphi, VisualBasic, MinGW, Digital Mars, FreePascal, COM
- Laufzeitbibliothek: msvcrt, MFC, ATL

Methoden zur Compilererkennung

Compiler- und Frameworkerkennung indirekt über Laufzeitbibliothek oder Linker:

- Laufzeitbibliothek: Signatur am '*entrypoint*'
- Linker: '*linker-fingerprinting*' mit Hilfe von Feldern im PE Header
  - 'linker version'
  - stack reserve, stack commit
  - heap reserve, heap commit

Beispiel: Googlesuche nach
'Signature: 00004550 Linker Version: 1.03'
liefert Links zu Binärdateien übersetzt mit LCC Compiler.

Viruserkennung

Viren modifizieren Dateien auf spezifische Art. Diese Modifikationen lassen sich im allgemeinen generisch erkennen.

Originaldatei soll sich nicht auffällig verhalten → Möglichkeiten der Arten der Infektion sind begrenzt.

Wo sitzt der virale Code?

- am Dateianfang (prepender)
- am Dateiende (appender)
- im Slackspace
- im Hostcode (eg. ZMist)

Viruserkennung – 'Appender' Viren

'entrypoint' in letzter Sektion: klares Zeichen einer Modifikation

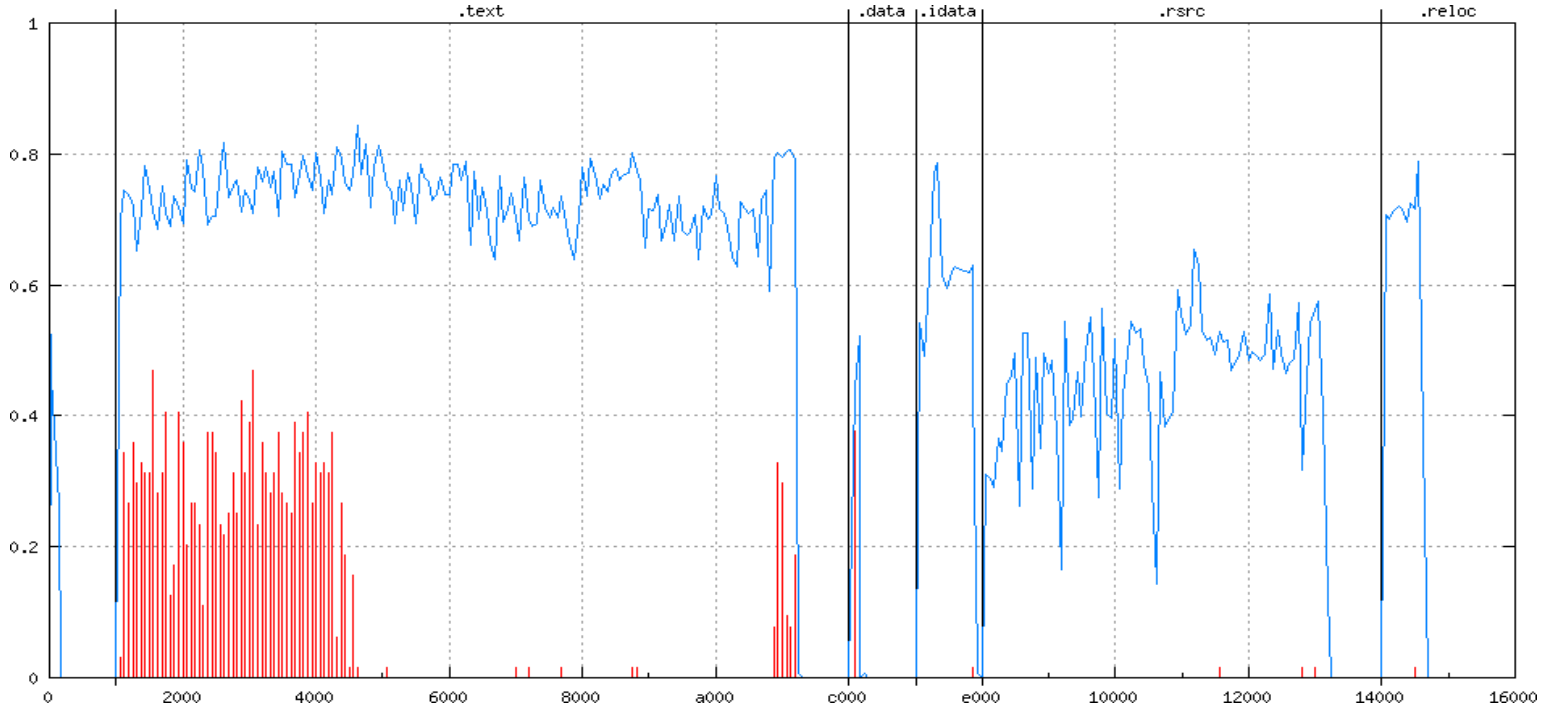
Für EPO (entry-point obscuring) Viren andere Methoden notwendig:

Letzte Sektion in PE-Dateien hat im allgemeinen eine feste Struktur (.rsrc, .reloc, .idata, .debug) → Angehängter Code lässt sich leicht identifizieren.

Generische Virenerkennung

Viren sind mobiler Code und enthalten keine externen Addressreferenzen

Beispiel: ZMist



Nutzung Externer Funktionalität

- Windows API
  - statische Imports (import section)
  - dynamische Imports (LoadLibrary, GetProcAddress)
  - dynamische Imports (int2e Interface)
- COM Interface

Strings

```
!This program cannot be run in DOS
    mode.
.text
`.rdata
@.data
.rsrc
@.reloc
SVt
{tR=
u[h0c
Ph,c
tDVh
@_^[
f90t-f
WWWP
t[9=
SVW3
Ph,c
t%WSV
_^[]
SVWP
WVSt
1f9E
WVSt
ChooseFontW
GetOpenFileNameW
5(545G5O5T5b5
62686t6z6
 1$1(1,1014181<1@1D1L1T1\1d1l1t1|
exe\notepad.dbg
.exe
```

Problem: zu viele nutzlose Information

Lösung:

- Filterung mit whitelist
- Filterung mit API Datenbank
- Filterung mit n-grams

Danke für Ihre Aufmerksamkeit!