Malware-Analyse und Reverse Engineering

11: Verhaltensanalyse von Viren

8.6.2017

Prof. Dr. Michael Engel

Überblick

Themen:

- Verhaltensanalyse
- Systemaufruf-Traces mit lookahead

Verhaltensanalyse

Zwei Ansätze zur Erkennung von „Eindringlingen“ (*intrusion detection*):

- Erkennung von **Missbrauch** oder Muster (Signaturen) des Eindringens (intrusion signatures) werden verwendet, um die Funktionalität **bekannter** Malware zu verhindern

- Die Erkennung von Anomalien geht von einem **unbekannten Eindringling** aus, dessen Verhalten sich vom normalen beobachteten Systemverhalten unterscheidet => **Thema heute**

- Viele Erkennungssysteme kombinieren beide Ansätze

Verhaltensanalyse: Biologische Inspiration

Analogie zu natürlichem Immunsystem:

- Annahme (der Informatiker): Natürliche Immunsysteme unterscheiden zwischen Proteinsegmenten (Peptiden), die zum funktionierenden Körper gehören (***self***) und solchen, die von eindringenden und fehlerhaft funktionierenden Zellen erzeugt werden (***nonself***)
- Dieses Modell ist in der Biologie allerdings nicht unbedingt anerkannt… ☺

Verhaltensanalyse: von Biologie zu Informatik

HOCHSCHULE COBURG

Einsatz eines künstlichen Immunsystems für Computer erfordert

- Entscheidung, welche Daten oder Aktivitätsmuster des Systems zur Unterscheidung zwischen *self* und *nonself* verwendet werden
- Bestimmung, welche Datenströme von einem Schutzsystem (IDE = *Intrusion Detection System*) erkannt werden
- Entwicklung eines Bedrohungsmodells
- **Also:** Finden des Computer-Äquivalents zu Peptiden
  - Soll Sicherheitsverletzungen erkennen
  - Dabei wenig falsche Alarme als Reaktion auf übliche Änderung des Systemverhaltens (Upgrade, Betriebsmodus usw.) erzeugen

Verhaltensanalyse: Entwurfsprinzipien

HOCHSCHULE COBURG

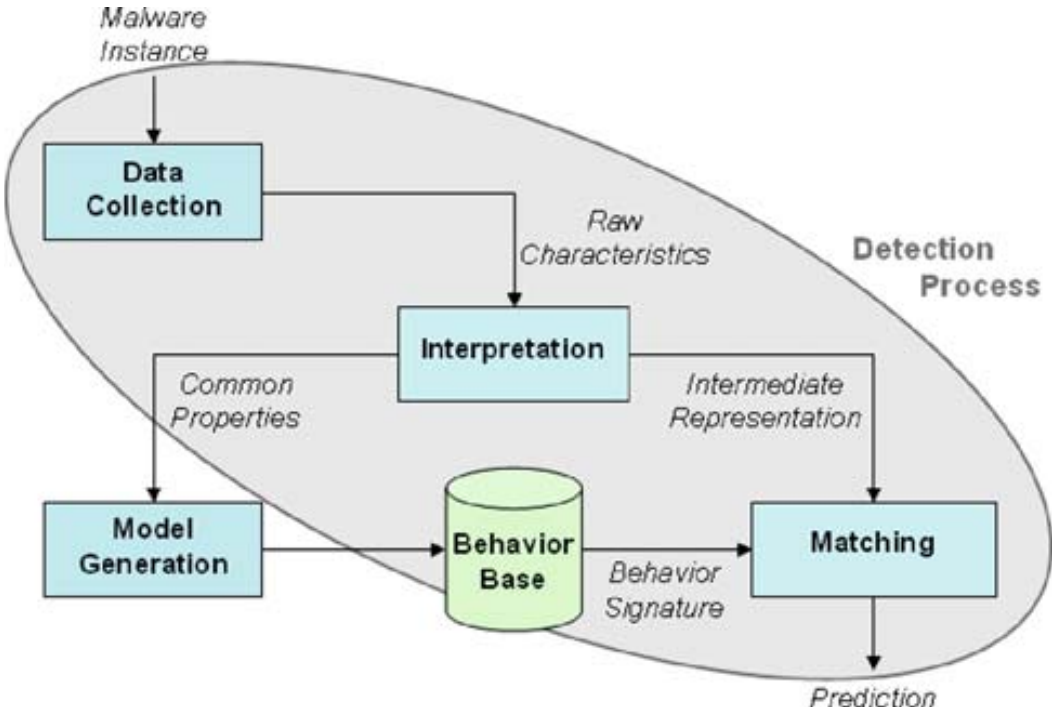
Anforderungen an das System:

- *Generischer Mechanismus:* erfasst große Vielfalt von Angriffen
  - Ist nicht 100% beweisbar sicher unter einem Angriffsmodell
- *Adaptierbar* bei Änderungen in zu schützendem System
  - Stellt Robustheit des Schutzsystems sicher
  - Analog zu Adaption an mutierende biologische Viren
- *Autonomie*: unabhängige Operation zur Laufzeit des Systems
- *Abgestufte Reaktion*: Biologische Systeme reagieren mit kleinen Aktionen auf kleine Abweichungen vom Verhalten, entsprechend mit großen auf größere
  - Software ist hier oft binär => lässt sich das anpassen?
- *Diversität* des Schutzes: Schutz vor verschiedensten Angriffen

Verhaltensanalyse von Viren

Vorgehensweise

- Sammeln von Daten
- Interpretation
- Erzeugen eines Modells für „korrektes“ (erwartetes) Verhalten
- Zur Laufzeit: Analyse des realen Verhaltens und Abgleich mit erwartetem

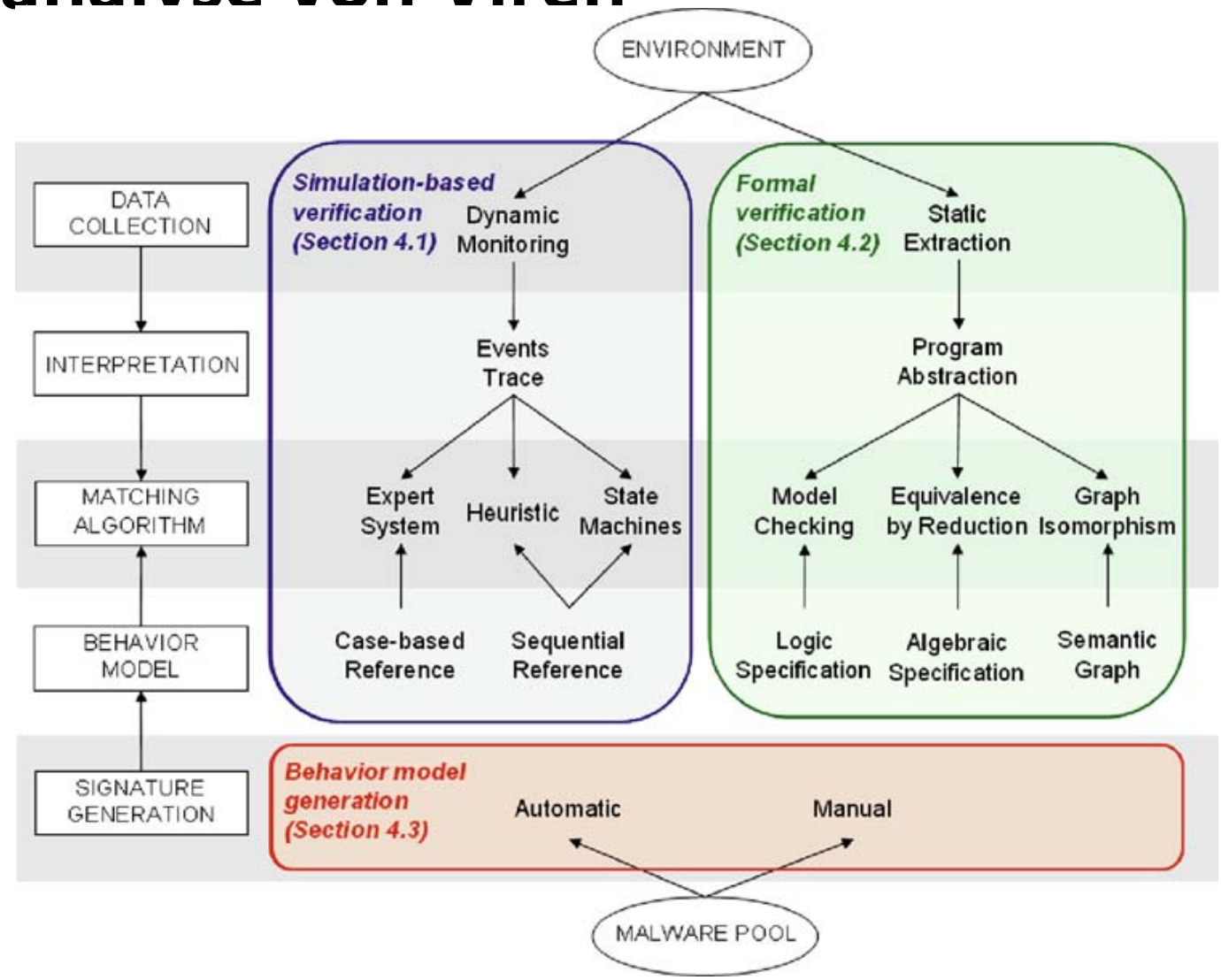


Verhaltensanalyse von Viren

Einordnung der Verhal-tensanalyse

Andere Ansätze:

- Formale Verifikation
- Simulations-basierte Verifikation



Ein „Selbstbewusstsein“ für Prozesse (1)

HOCHSCHULE COBURG

Forrest et al., *“A Sense of Self for Unix Processes”*, IEEE Symposium on Security and Privacy, 1996

Unterscheidung von „eigenem“ (erwarteten) und „fremdem“ (unerwartetem) Verhalten:

- *“Finden des Computer-Äquivalents zu Peptiden“*
- Große Menge verschiedener Ansätzen, Verhalten zu beobachten:
  - Kontrollfluss (schon gesehen)
  - Angelegte Dateien
  - Netzwerkverbindungen
- Aber auch “exotische” Systemparameter wie:
  - Elektrische Leistungsaufnahme
  - Abgestrahlte elektromagnetische Felder
  - Verhalten von Caches/TLBs, z.B. [4]
  - *Angreifer nutzen diese für “side channel attacks” zur Analyse von Verhalten*

Ein „Selbstbewusstsein“ für Prozesse (2)

HOCHSCHULE COBURG

Forrest et al., *“A Sense of Self for Unix Processes”*, IEEE Symposium on Security and Privacy, 1996

Systemaufrufe als Datenbasis für Verhaltensanalyse

- Jedes Programm besitzt implizit eine Menge von Systemaufrufe, die es erzeugen kann
- Abfolge von Systemaufrufen im Kontrollfluss des Programms erzeugt *mögliche Folgen von Systemaufrufen*
- Die normale Ausführung eines Programms erzeugt eine *Teilmenge aller möglichen Abfolgen*
- Theoretische Anzahl von Abfolgen in realem (nicht trivialen) Programm ist sehr groß => *jede Ausführung kann zu bisher nicht beobachteter Abfolge von Systemaufrufen führen!*
- **Beobachtung:** lokale Anordnung von Systemaufrufen (kurze Folgen) ist sehr konsistent
  => Basis zur Bestimmung des “self”=“normales” Verhalten

Analyse von Systemaufruf-Traces mit Lookahead

Analyse kurzes Abfolgen von Systemaufrufen

- Normales Verhalten: definiert durch kurze Abfolgen von Systemaufrufen mit Längen 5, 6 und 11 Aufrufe
- Vorgehensweise: Aufbau einer separaten Datenbank für jeden zu beobachtenden Prozess
  - spezifisch für SW-Architektur, -Version, und -Konfiguration, lokalen Vorgabe und Benutzungsmuster
  - Durch große Variabilität zwischen Systemen entsteht eindeutige Definition des *"self"* für die meisten Systeme
  - Hinreichend große und stabile Datenbankinhalte werden zur Überwachung des jeweiligen Prozesses verwendet
  - Abfolgen in der Datenbank => normales Verhalten
  - Nicht gefundene Abfolgen => Verhaltensabweichung

Traces mit Lookahead: Aufbau einer Verhaltensdatenbank (1)

Aufbau einer Datenbank mit *“normalen”* Traces v. Systemaufrufen

„Sliding window“ über letzte *n* aufgezeichnete Systemaufrufe:

Aufruf-Trace: ❶ ❷ ❸ open, read, mmap, mmap, open, getrlimit, mmap, close

Bestimmung möglicher (“legaler”) Folgeaufrufe zu gegebenem Aufruf für Position 1–4, Betrachtung von bis zu drei Folgeaufrufen:

call	position 1	position 2	position 3
open	read	mmap	mmap
read	mmap	mmap	
mmap	mmap		

Traces mit Lookahead: Aufbau einer Verhaltensdatenbank (2)

„Sliding window“ über letzte *n* aufgezeichnete Systemaufrufe:

Aufruf-Trace: ❶ ❷ ❸ open, read, mmap, mmap, open, getrlimit, mmap, close

Gesamte Datenbank möglicher Systemaufruffolgen:

call	position 1	position 2	position 3
open	read, getrlimit	mmap	mmap, close
read	mmap	mmap	open
mmap	mmap, open, close	open, getrlimit	getrlimit, mmap
getrlimit	mmap	close	
close			

Traces mit Lookahead: Verhaltensanalyse (1)

HOCHSCHULE COBURG

Analyse von Prozessen

- Analog zur Ermittlung der “normalen” Abfolgemuster
- Neue Traces werden zur Laufzeit gesammelt (=> strace/ptrace) und mit Mustern in der Datenbank verglichen
- Window der Größe k + 1 wird über den neuen Trace bewegt
- Feststellen, ob die aktuelle Abfolge von Systemaufrufen von denen in der Datenbank abweicht
  - Einfachster Ansatz: Test auf Vorhandensein oder Abwesenheit einer legalen Abfolge

Traces mit Lookahead: Verhaltensanalyse (2)

HOCHSCHULE COBURG

call	position 1	position 2	position 3
open	read, getrlimit	mmap	mmap, close
read	mmap	mmap	open
mmap	mmap, open, close	open, getrlimit	getrlimit, mmap
getrlimit	mmap	close	
close			

Datenbank der Aufruf-abfolgen

Neuer erfasster Systemaufruf-Trace:

- Unterschied an einer Stelle:
  remap ➛ open

open, read, mmap, open, open, getrlimit, mmap, close

Erzeugt **4 Abweichungen** im Vergleich zur Datenbank:

- Auf open folgt kein open an Position 3
- Auf read folgt kein open an Position 2
- Auf open folgt kein open an Position 1 und
- Auf open folgt kein getrlimit an Position 2

Traces mit Lookahead: Verhaltensanalyse (3)

HOCHSCHULE COBURG

Anzahl der Abweichungen wird als Prozentsatz der maximal möglichen Anzahl von Abweichungen ermittelt

- Maximale Anzahl paarweiser Abweichungen für eine Abfolge der Länge *L* mit einem Lookahead von *k* beträgt:

$$k(L-k)+(k-1)+(k-2)+\ldots+1 = k(L-(k+1)/2).$$

Im Beispiel: $L = 8$, $k = 3$ und es gibt 4 Abweichungen

- Formel ergibt max. Datenbankgröße von 18 => 22% Abweichung
- Abweichungen sind einziger Parameter, der “normal” von “nicht normal” unterscheidet

Lookahead-Algorithmus implementierbar in O(N)

- N = Länge des Trace (Anzahl Systemaufrufe)
- Beispiel: Analyse von Traces mit ca. 1250 Systemaufrufen/Sekunde

Traces mit Lookahead: Evaluation (1)

HOCHSCHULE COBURG

Ist die vorgeschlagene Methode sinnvoll *und* realisierbar?

- Wie groß muss die Datenbank sein, um “normales” Verhalten beschreiben zu können?
- Welcher Bruchteil aller möglichen Systemaufruf-Abfolgen wird von der Datenbank “normaler” Abfolgen erfasst?
- Kann diese Definition von “normalem” Verhalten zwischen verschiedenen Arten von Programmen unterscheiden?
- Kann diese Definition von “normal” tatsächlich abweichendes Verhalten erkennen?

Experimente:

- Analyse von sendmail unter SunOS 4.1.1/4.1.4 mit strace

Traces mit Lookahead: Evaluation (2)

HOCHSCHULE COBURG

Welche *Daten* zur Datenbank-Erstellung verwenden?

- Syscall-Folgen abhängig von Parametern
  - Kommandozeilenparameter, Eingaben, Dateien, …

- Entscheidung:
  - Erzeugen von künstlichen Eingaben, die alle normale Modi von sendmail abdecken?
  - oder reales Verhalten von sendmail beobachten und hoffen, dass dies alle Fälle von “normal” abdeckt?

Traces mit Lookahead: Evaluation (3)

HOCHSCHULE COBURG

Welche *Daten* zur Datenbank-Erstellung verwenden?

- Hier: 112 konstruierte Nachrichten
  - Enthalten so viel Variation wie möglich
  - Erzeugen kombinierte Tracelänge von > 1,5 Millionen syscalls
  - Für Windowgröße 6: Datenbank mit ~1500 Einträgen
    - Jeder Eintrag: ein Paar Systemaufrufe mit Lookahead-Wert
    - Eintrag = Paar von Systemaufrufen mit Lookahead-Wert
    - z.B. ist "read" legaler Nachfolger von "open" an Pos. 1

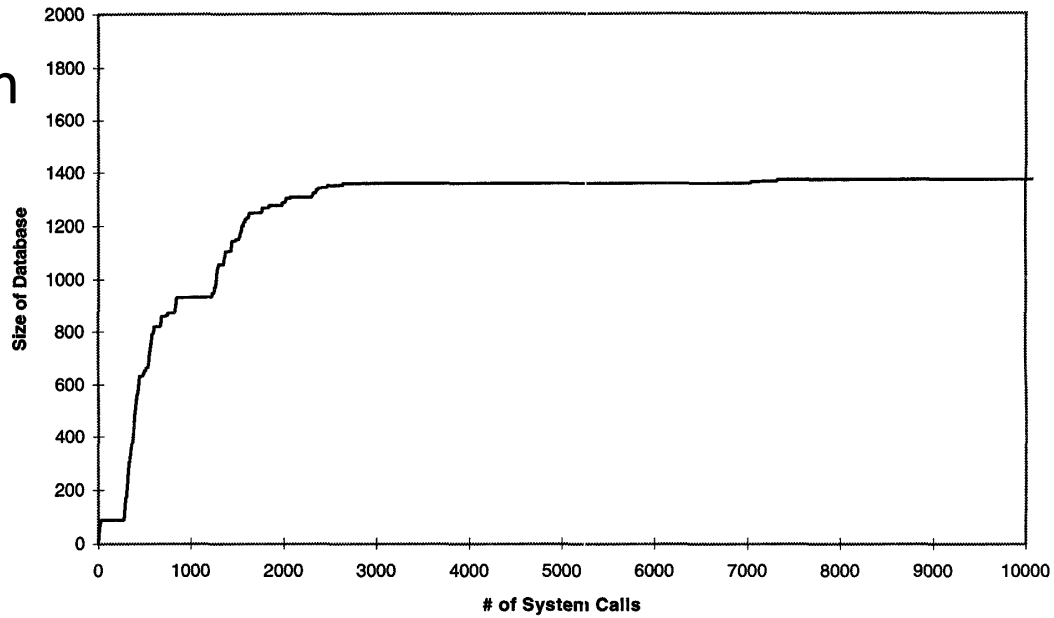
Type of Behavior	# of msgs.
message length	12
number of messages	70
message content	6
subject	2
sender/receiver	4
different mailers	4
forwarding	4
bounced mail	4
queuing	4
vacation	2
total	112

Traces mit Lookahead: Evaluation (4)

HOCHSCHULE COBURG

Einfluss verschiedener Parameter

- Einige der analysierten Parameter haben nur geringen oder keinen Einfluss auf das “normale” Verhalten:
  - Anzahl Nachrichten
  - Inhalt der Nachrichten
  - Betreffzeile
  - Absender/Empfänger
  - Warteschlangen
- Datenbankgröße stabil nach ca. 3000 erfassten Systemaufrufen



Traces mit Lookahead: Evaluation (5)

HOCHSCHULE COBURG

Ist das aufgezeichnete Verhalten *eindeutig* für sendmail?
Untersuchung, ob andere Unix-Prozesse gleiche Systemaufruf-Folgen aufweisen

Signifikante Anzahl von abweichenden Folgen

- ~5–32% für Folgen der Länge 6
- Klar(?): andere Prozesse führen andere Aktionen als sendmail durch

	5		6		11	
Process	%	#	%	#	%	#
sendmail	0.0	0	0.0	0	0.0	0
ls	6.9	23	8.9	34	13.9	93
ls -l	30.0	239	32.1	304	38.0	640
ls -a	6.7	23	8.3	34	13.4	93
ps	1.2	35	8.3	282	13.0	804
ps -ux	0.8	45	8.1	564	12.9	1641
finger	4.6	21	4.9	27	5.7	54
ping	13.5	56	14.2	70	15.5	131
ftp	28.8	450	31.5	587	35.1	1182
pine	25.4	1522	27.6	1984	30.0	3931
httpd	4.3	310	4.8	436	4.7	824

Traces mit Lookahead: Evaluation (6)

HOCHSCHULE COBURG

Kann *abweichendes Verhalten* festgestellt werden?

Angriff auf sendmail über verschiedene Sicherheitslücken:

- *sunsendmailcp*: verwendet sendmail-Option, die eine Mail an eine Datei anhängt. Kann wichtige Systemdateien abändern (hier: /. rhosts für Fernzugriff => Shell mit root-Rechten)
- *syslog attack script*: Ausnutzen eines Fehlers im syslog- interface zur Erzeugung eines Pufferüberlaufs in sendmail
- *decode alias attack*: Decodierung von Attachments über externes Programm uudecode. Dieses akzeptiert absolute Pfadnamen in uuencodeter Datei
  - Wenn Datei “bar.uu” als Originalpfad “/home/foo/.rhosts” angibt, wird diese Datei angelegt oder überschrieben

Traces mit Lookahead: Evaluation (7)

HOCHSCHULE COBURG

Analyse verschiedener anomaler Situationen:

- Erfolgreiches Eindringen (sunsendmailcp, syslog, decode, lprcp)
- Erfolg*lose* Versuche (sm565a und sm5x)
  - Sicherheitslücken waren bereits gepatcht
- Seltene Fehler (forward-Schleife)

Anomaly	5 %	5 #	6 %	6 #	11 %	11 #
sunsendmailcp	3.8	72	4.1	95	5.2	215
syslog:						
remote 1	3.9	361	4.2	470	5.1	1047
remote 2	1.4	111	1.5	137	1.7	286
local 1	3.0	235	4.2	398	4.0	688
local 2	4.1	307	3.4	309	5.3	883
decode	0.3	21	0.3	24	0.3	57
lprcp	1.1	7	1.4	12	2.2	31
sm565a	0.4	27	0.4	36	0.6	89
sm5x	1.4	110	1.7	157	2.7	453
forward loop	1.6	43	1.8	58	2.3	138

Forward-Schleife:

Email address	`.forward` file
foo@host1	bar@host2
bar@host2	foo@host1

Traces mit Lookahead: Einschränkungen

HOCHSCHULE COBURG

Analyse durch Lookahead leidet unter Beschränkungen:

- Nur ein (kleiner) Teilaspekt des Prozessverhaltens betrachtet
- Nur die Systemaufrufe an sich werden aufgezeichnet
  - Keine Parameter der Aufrufe
  - Kein Zeitverhalten
  - Keine Analyse von zwischen den Systemaufrufen ausgeführten Befehlsfolgen
  - …
- Dennoch bereits so gute Ergebnisse erzielbar

Angriff auf Trace-basierte Methoden: Mimikri

D. Wagner and D. Dean, *"Intrusion detection via static analysis"*, IEEE Symposium on Security and Privacy 2001

Idee: Einfügen vieler sinnloser (und funktionsloser) Systemaufrufe

- Verwirrung stiften...
- Einfügen von "unwirksamen" System-aufrufen: Aufrufe ohne Seiteneffekte durch Ignorieren von Rückgabewerten oder Manipulation von Parametern
- Ermöglicht es dem Angreifer, eine Angriffsfolge innerhalb einer „legalen" Folge zu verbergen, indem mit „unwirksamen" Aufrufen aufgefüllt wird
- Erfordert detaillierte Kenntnisse des Verhaltens und der verwendeten Analysemethode

```
read() write() close() munmap() sigprocmask() wait4()
sigprocmask() sigaction() alarm() time() stat() read()
alarm() sigprocmask() setreuid() fstat() getpid()
time() write() time() getpid() sigaction() socketcall()
sigaction() close() flock() getpid() lseek() read()
kill() lseek() flock() sigaction() alarm() time()
stat() write() open() fstat() mmap() read() open()
fstat() mmap() read() close() munmap() brk() fcntl()
setregid() open() fcntl() chroot() chdir() setreuid()
lstat() lstat() lstat() lstat() open() fcntl() fstat()
lseek() getdents() fcntl() fstat() lseek() getdents()
close() write() time() open() fstat() mmap() read()
close() munmap() brk() fcntl() setregid() open() fcntl()
chroot() chdir() setreuid() lstat() lstat() lstat()
lstat() open() fcntl() brk() fstat() lseek() getdents()
lseek() getdents() time() stat() write() time() open()
getpid() sigaction() socketcall() sigaction() umask()
sigaction() alarm() time() stat() read() alarm()
getrlimit() pipe() fork() fcntl() fstat() mmap() lseek()
close() brk() time() getpid() sigaction() socketcall()
sigaction() chdir() sigaction() sigaction() write()
munmap() munmap() munmap() exit()
```

Angriff auf *wu-ftpd:* Nur unterstrichene Syscalls sind Teil des Angriffs

Angriff: Kurze Traces

K. Tan and R. Maxion, *“"Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector”,* IEEE Security and Privacy 2002

Idee: Mindestlänge v. Traces für wirksame Erkennung notwendig

- Verwendung sehr kurzer Traces
- Mindest-Tracelänge unterschritten => nicht erkennbar
- Frage: welche sinnvolle Funktionalität lässt sich in kurzen Aufruffolgen realisieren?

Bewertung der Virenanalyseansätze

Signaturbasiert

- Einfache statische Methode
- Signaturen müssen aktuell sein: neue Viren nicht erkannt
- Polymorphe Viren umgehen einfache Signaturchecks
- Komplexe Checks zur Laufzeit sehr aufwendig (Entschlüsseln...)

Verhaltensbasiert

- Erfolgreichster Ansatz: Traces von Systemaufrufen
- Definition eines „normalen“ Verhaltens: was ist Abweichung?
- Effizienzte Auswertung zur Laufzeit
- Genaue Ursache (welches Virus?) des Problems nicht bekannt

Fazit

Verhaltensbasierte Analysen

- Aufwand zur Laufzeit
- Einfache Muster (Systemaufrufe) bereits sehr nützlich
- Komplexere Analysen => nächste Vorlesung

- 100% zuverlässige Virenerkennung ist unmöglich!
  - Siehe Paper von Evans

Referenzen

1. G. Jacob, H. Debar, E. Filiol, "*Behavioral detection of malware: From a survey towards an established taxonomy*", Journal in Computer Virology 4(3):251-266, August 2008
   - Überblick über >50 Verhaltensanalysen aus Forschung und Industrie
2. S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, "*A sense of self for Unix processes*", Proceedings 1996 IEEE Symposium on Security and Privacy, Oakland, CA, 1996, pp. 120-128
   - Erste Ideen zur Verhaltensanalyse von Unix-Prozessen (+Folgepapers)
3. D. Evans, "*On the Impossibility of Virus Detection*", https://www.cs.virginia.edu/~evans/virus/
   - Theoretisches Paper, das zeigt, dass Viruserkennung eigentlich unmöglich ist…
4. C. Percival, *"Cache missing for fun and profit",* BSDCan 2005, Ottawa