

Malware-Analyse und Reverse Engineering

8: Statische und dynamische Analysen
zur Malware-Erkennung

11.5.2017

Prof. Dr. Michael Engel



Überblick

Themen:

- Statische Analysen zur Malware-Erkennung
- Dynamische Analysen zur Malware-Erkennung

Malware-Erkennung

Statische Analysen

- Signaturen
- Abstract Interpretation

Dynamische Analysen

- Anomalieerkennung
- Kontrollflussverfolgung

Statische Analysen

Vorige Vorlesung

- Statische Analyse von Programm-Quellcode zum Finden möglicher Sicherheitslücken

Jetzt

- Statische Analyse von Binärprogrammen, um festzustellen, ob ein Programm Malware enthält
- Statische Analyse:
 - Informationen über das Verhalten eines Programms ermitteln, ohne das Programm selbst auszuführen

Einfache Statische Analyse: Signaturen

Idee

- Malwarecode besteht aus bestimmter (Maschinen-)Befehlsfolge
- Wenn diese in Programmcode vorkommt $\Rightarrow$ Malware entdeckt
- Grundlage vieler einfacher Virens Scanner

Signatur

- Folge von Bytes im Programmcode
- Nicht notwendigerweise aufeinanderfolgend
 - Auch komplexere Muster möglich
- *Syntaktische* Methode – kein Verständnis des *Verhaltens* der Malware

Signaturen: Beispiel

„Stoned“-Virus von 1987

- Angeblich von einem Studenten in Neuseeland entwickelt
 - 1989: in Neuseeland und Australien verbreitet
 - 1990: Varianten auf der ganzen Welt verbreitet
- Bootsektor-Virus: befällt Bootsektoren von Disketten und Festplatten (unter MS-DOS!)
 - Kopiert sich beim Booten resident in den Hauptspeicher
 - Infiziert neu eingelegte Disketten (und neu angeschlossene Festplatten – die waren aber damals nicht im Betrieb wechselbar...)
- Malware-Funktionalität:
 - Bei jedem 8. Bootvorgang des PC (von befallener Diskette oder Festplatte) wird der Text „*Your PC is now Stoned!*“ ausgegeben

Stoned “in the wild”

Ein Virus von 1987 macht 20 Jahre später noch Ärger...



[Blog](#) [Bu](#)

Boot virus shipped on German laptops

Posted by  **Virus Bulletin** on  Sep 14, 2007

Aged malware installed on batch of Vista systems.

A consignment of laptops from German manufacturer *Medion*, sold through German and Danish branches of giant retail chain *Aldi*, have been found to be infected with the boot sector virus 'Stoned.Angelina', first seen as long ago as 1994 and last included on the official WildList in 2001.

According to German sources, anywhere between 10,000 and 100,000 systems may be infected, but as the machines apparently ship without floppy drives the virus is unlikely to spread. The systems come pre-installed with *Windows Vista* and *Bullguard* anti-virus, which will warn of the 'harmless' infection on booting the machines, but is unable to remove it.

Dump des “Stoned”-Virus

```

00000000 EA 05 00 C0 07 E9 99 00 00 51 02 00 C8 E4 00 80 Ê..À.é™..Q..Êä.€
00000010 9F 00 7C 00 00 1E 50 80 FC 02 72 17 80 FC 04 73 Ý.|...P€Ü.r.€Ü.s
00000020 12 0A D2 75 0E 33 C0 8E D8 A0 3F 04 A8 01 75 03 ..Ôu.3Ažø?....u.
00000030 E8 07 00 58 1F 2E FF 2E 09 00 53 D1 52 06 56 57 è..X..ÿ...SñR.vw
00000040 BE 04 00 B8 01 02 0E 07 BB 00 02 33 C9 8B D1 41 %.....*..3É<ñA
00000050 9C 2E FF 1E 09 00 73 0E 33 C0 9C 2E FF 1E 09 00 e.ÿ...s.3Ae.ÿ...
00000060 4E 75 E0 EB 35 90 33 F6 BF 00 02 FC 0E 1F AD 3B Nuãë5.3õž...ü...;
00000070 05 75 06 AD 3B 45 02 74 21 B8 01 03 BB 00 02 B1 .u...;E.t!...*...±
00000080 03 B6 01 9C 2E FF 1E 09 00 72 0F B8 01 03 33 DB .¶.e.ÿ...r...30
00000090 B1 01 33 D2 9C 2E FF 1E 09 00 5F 5E 07 5A 59 5B ±.30e.ÿ..._À.ZY[
000000A0 C3 33 C0 8E D8 FA 8E D0 BC 00 7C FB A1 4C 00 A3 A3Ažøüžø%.|û;L.£
000000B0 09 7C A1 4E 00 A3 0B 7C A1 13 04 48 48 A3 13 04 .|;N.£.|;...HH£..
000000C0 B1 06 D3 E0 8E C0 A3 0F 7C B8 15 00 A3 4C 00 8C ±.0àžA£.|...£L.£
000000D0 06 4E 00 B9 B8 01 0E 1F 33 F6 8B FE FC F3 A4 2E .N.'...3õ<þüöµ.
000000E0 FF 2E 0D 00 B8 00 00 CD 13 33 C0 8E C0 B8 01 02 ÿ......f.3AžA...
000000F0 BB 00 7C 2E 80 3E 08 00 00 74 0B B9 07 00 BA 80 *.|.€>...t.'...°€
00000100 00 CD 13 EB 49 90 B9 03 00 BA 00 01 CD 13 72 3E .f.ëI.'...°...f.r>
00000110 26 F6 06 6C 04 07 75 12 BE 89 01 0E 1F AC 0A C0 &õ.l...u.%%...-.A
00000120 74 08 B4 0E B7 00 CD 10 EB F3 0E 07 B8 01 02 BB t.'...f.ëó...*
00000130 00 02 B1 01 BA 80 00 CD 13 72 13 0E 1F BE 00 02 ..±.°€.f.r...%..
00000140 BF 00 00 AD 3B 05 75 11 AD 3B 45 02 75 0B 2E C6 ž...;u...;E.u...A
00000150 06 08 00 00 2E FF 2E 11 00 2E C6 06 08 00 02 B8 .....ÿ....A.....
00000160 01 03 BB 00 02 B9 07 00 BA 80 00 CD 13 72 DF 0E ...*...'.°€.f.rB.
00000170 1F 0E 07 BE BE 03 BF BE 01 B9 42 02 F3 A4 B8 01 ...%%.ž%. 'B.óµ...
00000180 03 33 DB FE C1 CD 13 EB C5 07 59 6F 75 72 20 50 .30hA£.ëA.Your p
00000190 43 20 69 73 20 6E 6F 77 20 53 74 6F 6E 65 64 21 C is now Stoned!
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Analyse des “Stoned”-Virus

Bootsektor = „Master Boot Record“ (MBR)

- Erster Sektor einer Diskette/Festplatte bei PCs, die mit klassischem BIOS (nicht UEFI) arbeiten: **512 Bytes**
- Wird von BIOS beim Start des PC von Diskette/Platte geladen
- Enthält Code, der mit Hilfe von BIOS-Funktionen (SW-Interrupts) Rest des OS lädt

Adresse		Funktion / Inhalt		Größe (Bytes)
hex	dez			
0x0000	0	Startprogramm (englisch <i>Bootloader</i>) (Programmcode)		440
0x01B8	440	Datenträgersignatur (seit Windows 2000)		4
0x01BC	444	Null (0x0000)		2
0x01BE	446	Partitionstabelle		64
0x01FE	510	55 _{hex}	Bootsektor-Signatur (wird vom BIOS für den ersten Bootloader geprüft)	2
0x01FF	511	AA _{hex}		
Gesamt:				512

Der “Stoned”-Bootsektor

```

00000000 EA 05 00 C0 07 00 00 00 00 51 02 00 C6 E4 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010 9F 00 7C 00 00 1E 50 80 FC 02 72 17 80 FC 04 73 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 12 0A D2 75 0E 33 C0 8E D8 A0 3F 04 A8 01 75 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 E8 07 00 58 1F 2E FF 2E 09 00 53 D1 52 06 56 57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 BE 04 00 B8 01 02 0E 07 BB 00 02 33 C9 8B D1 41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 9C 2E FF 1E 09 00 73 0E 33 C0 9C 2E FF 1E 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 4E 75 E0 EB 35 90 33 F6 BF 00 02 FC 0E 1F AD 3B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 05 75 06 AD 3B 45 02 74 21 B8 01 03 BB 00 02 B1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 03 B6 01 9C 2E FF 1E 09 00 72 0F B8 01 03 33 DB 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 B1 01 33 D2 9C 2E FF 1E 09 00 5F 5E 07 5A 59 5B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0 C3 33 C0 8E D8 FA 8E D0 BC 00 7C FB A1 4C 00 A3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0 09 7C A1 4E 00 A3 0B 7C A1 13 04 48 48 A3 13 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 B1 06 D3 E0 8E C0 A3 0F 7C B8 15 00 A3 4C 00 8C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0 06 4E 00 B9 B8 01 0E 1F 33 F6 8B FE FC F3 A4 2E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0 FF 2E 0D 00 B8 00 00 CD 13 33 C0 8E C0 B8 01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 BB 00 7C 2E 80 3E 08 00 00 74 0B B9 07 00 BA 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100 00 CD 13 EB 49 90 B9 03 00 BA 80 01 CD 13 72 3E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110 26 F6 06 6C 04 07 75 12 BE 89 01 0E 1F AC 0A C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120 74 08 B4 0E B7 00 CD 10 EB F3 0E 07 B8 01 02 BB 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000130 00 02 B1 01 BA 80 00 CD 13 72 13 0E 1F BE 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140 BF 00 00 AD 3B 05 75 11 AD 3B 45 02 75 0B 2E C6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150 06 08 00 00 2E FF 2E 11 00 2E C6 06 08 00 02 B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160 01 03 BB 00 02 B9 07 00 BA 80 00 CD 13 72 DF 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170 1F 0E 07 BE BE 03 BF BE 01 B9 42 02 F3 A4 B8 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000180 03 33 DB FE C1 CD 13 EB C5 07 59 6F 75 72 20 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190 43 20 69 73 20 6E 6F 77 20 53 74 6F 6E 65 64 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

**0x000-004: Sprungbefehl
“JMP 07C0:0005”**
(wird oft zur Erkennung
eines gültigen Bootsektors
verwendet)

Adresse		Funktion / Inhalt		Größe (Bytes)
hex	dez			
0x0000	0	Startprogramm (englisch Bootloader) (Programmcode)		440
0x01B8	440	Datenträgersignatur (seit Windows 2000)		4
0x01BC	444	Null (0x0000)		2
0x01BE	446	Partitionstabelle		64
0x01FE	510	55 _{hex}	Bootsektor-Signatur (wird vom BIOS für den ersten Bootloader geprüft)	2
0x01FF	511	AA _{hex}		
Gesamt:				512

0x1BE-0x1FD:
Partitionstabelle
(wird freigelassen
und später ergänzt)

**0x1FE/1FF: Signatur
des Bootsektors**
(wird später eingefügt)

Analyse: "Stoned"-Bootsektor

```

00000000 EA 05 00 C0 07 E9 99 00 00 51 02 00 C8 E4 00 80 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A 6A
00000010 9F 00 7C 00 00 1E 50 80 FC 02 72 17 80 FC 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 12 0A D2 75 0E 33 C0 8E D8 A0 3F 04 A8 01 75 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 E8 07 00 58 1F 2E FF 2E 09 00 53 D1 32 06 56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040 BE 04 00 B8 01 02 0E 07 BB 00 02 33 C9 8B D1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050 9C 2E FF 1E 09 00 73 0E 33 C0 9C 2E FF 1E 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 4E 75 E0 EB 35 90 33 F6 BF 00 02 FC 0E 1F AD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070 05 75 06 AD 3B 45 02 74 21 B8 01 03 BB 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080 03 B6 01 9C 2E FF 1E 09 00 72 0F B8 01 03 33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 B1 01 33 D2 9C 2E FF 1E 09 00 5F 5E 07 5A 59 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0 C3 33 C0 8E D8 FA 8E D0 BC 00 7C FB A1 4C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0 09 7C A1 4E 00 A3 0B 7C A1 13 04 48 48 A3 13 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 B1 06 D3 E0 8E C0 A3 0F 7C B8 15 00 A3 4C 00 8C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0 06 4E 00 B9 B8 01 0E 1F 33 F6 8B FE FC F3 A4 2E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0 FF 2E 0D 00 B8 00 00 CD 13 33 C0 8E C0 B8 01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0 BB 00 7C 2E 80 3E 08 00 00 74 0B B9 07 00 BA 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100 00 CD 13 EB 49 90 B9 03 00 BA 00 01 CD 13 72 3E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110 26 F6 06 6C 04 07 75 12 BE 89 01 0E 1F AC 0A C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120 74 08 B4 0E B7 00 CD 10 EB F3 0E 07 B8 01 02 BB 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000130 00 02 B1 01 BA 80 00 CD 13 72 13 0E 1F BE 00 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140 BF 00 00 AD 3B 05 75 11 AD 3B 45 02 75 0B 2E C6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150 06 08 00 00 2E FF 2E 11 00 2E C6 06 08 00 02 B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160 01 03 BB 00 02 B9 07 00 BA 80 00 CD 13 72 DF 0E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170 1F 0E 07 BE BE 03 BF BE 01 B9 42 02 F3 A4 B8 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000180 03 33 DB FE C1 CD 13 EB C5 07 59 6F 75 72 20 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190 43 20 69 73 20 6E 6F 77 20 53 74 6F 6E 65 64 21 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

```

00000000 EA0500C007 jmp 07C0:0005
; jump to set correct Code Segment
Set_Code_Segment:
00000005 E99900 jmp Stoned_Start
; initial address of stoned boot virus

```

BIOS lädt Bootsektor immer an Adresse 0x07C0:0000

Virus nutzt absolute Adresse aus, um Daten usw. zu referenzieren

Erkennung von “Stoned” (1)

Erkennen von Bytefolge, die eindeutig Virus identifiziert

```

00000000 EA 05 00 C0 07 E9 99 00 00 51 02 00 C8 E4
00000010 9F 00 7C 00 00 1E 50 80 FC 02 72 17 80 FC
00000020 12 0A D2 75 0E 33 C0 8E D8 A0 3F 04 A8 01
00000030 E8 07 00 58 1F 2E FF 2E 09 00 53 01 52 06
00000040 BE 04 00 B8 01 02 0E 07 BB 00 02 33 C9 88
00000050 9C 2E FF 1E 09 00 73 0E 33 C0 9C 2E FF 1E
00000060 4E 75 E0 EB 35 90 33 F6 BF 00 02 FC 0E 1F
00000070 05 75 06 AD 3B 45 02 74 21 B8 01 03 BB 00
00000080 03 B6 01 9C 2E FF 1E 09 00 72 0F B8 01 03
00000090 B1 01 33 D2 9C 2E FF 1E 09 00 5F 5E 07 5A 59 5B
000000A0 C3 33 C0 8E D8 FA 8E D0 BC 00 7C FB A1 4C 00 A3
000000B0 09 7C A1 4E 00 A3 0B 7C A1 13 04 48 48 A3 13 04
000000C0 B1 06 D3 E0 8E C0 A3 0F 7C B8 15 00 A3 4C 00 8C
000000D0 06 4E 00 B9 B8 01 0E 1F 33 F6 8E FE FC F3 A4 2E
000000E0 FF 2E 0D 00 B8 00 00 CD 13 33 C0 8E C0 B8 01 02
000000F0 BB 00 7C 2E 80 3E 08 00 00 74 0B B9 07 00 BA 80
00000100 00 CD 13 EB 49 90 B9 03 00 BA 00 01 CD 13 72 3E
00000110 26 F6 06 6C 04 07 75 12 BE 89 01 0E 1F AC 0A C0
00000120 74 08 B4 0E B7 00 CD 10 EB F3 0E 07 B8 01 02 BB
00000130 00 02 B1 01 BA 80 00 CD 13 72 13 0E 1F BE 00 02
00000140 BF 00 00 AD 3B 05 75 11 AD 3B 45 02 75 0B 2E C6
00000150 06 08 00 00 2E FF 2E 11 00 2E C6 06 08 00 02 B8
00000160 01 03 BB 00 02 B9 07 00 BA 80 00 CD 13 72 DF 0E
00000170 1F 0E 07 BE BE 03 BF BE 01 B9 42 02 F3 A4 B8 01
00000180 03 33 DB FE C1 CD 13 EB C5 07 59 6F 75 72 20 50
00000190 43 20 69 73 20 6E 6F 77 20 53 74 6F 6E 65 64 21
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Kein geeigneter Kandidat:

- *jeder Bootsektor enthält diese Bytefolge (oder eine sehr ähnliche)*

<http://www.stoned-vienna.com/analysis-of-stoned.html>

Erkennung von “Stoned” (2)

Erkennen von Bytefolge, die eindeutig Virus identifiziert

```

00000000 EA 05 00 C0 07 E9 99 00 00 51 02 00 C8 E4 00 80  .A.
00000010 9F 00 7C 00 00 1E 50 80 FC 02 72 17 80 FC 04 73  . .
00000020 12 0A D2 75 0E 33 C0 8E D8 A0 3F 04 A8 01 75 03  .Ou.
00000030 E8 07 00 58 1E 2E FF 2E 09 00 53 D1 32 06 56 57  .X.
00000040 BE 04 00 B8 01 02 0E 07 BB 00 02 33 C9 8B D1 41  . .
00000050 9C 2E FF 1E 09 00 73 0E 33 C0 9C 2E FF 1E 09 00  . .
00000060 4E 75 E0 EB 35 90 33 F6 BF 00 02 FC 0E 1F AD 38       
00000070 05 75 06 AD 3B 45 02 74 21 B8 01 03 BB 00 02 B1  . .
00000080 03 B6 01 9C 2E FF 1E 09 00 72 0F B8 01 03 33 DB  . .
00000090 B1 01 33 D2 9C 2E FF 1E 09 00 5F 5E 07 5A 59 5B  . .
000000A0 C3 33 C0 8E D8 FA 8E D0 BC 00 7C FB A1 4C 00 A3       
000000B0 09 7C A1 4E 00 A3 0B 7C       
000000C0 B1 06 D3 E0 8E C0 A3 0F       
000000D0 06 4E 00 B9 B8 01 0E 1F       
000000E0 FF 2E 0D 00 B8 00 00 CD       
000000F0 BB 00 7C 2E 80 3E 08 00       
00000100 00 CD 13 EB 49 90 B9 03       
00000110 26 F6 06 6C 04 07 75 12       
00000120 74 08 B4 0E B7 00 CD 10       
00000130 00 02 B1 01 BA 80 00 CD       
00000140 BF 00 00 AD 3B 05 75 11       
00000150 06 08 00 00 2E FF 2E 11       
00000160 01 03 BB 00 02 B9 07 00       
00000170 1F 0E 07 BE BE 03 BF BE       
00000180 03 33 DB FE C1 CD 13 EB       
00000190 43 20 69 73 20 6E 6F 77       
000001A0 00 00 00 00 00 00 00 00       
000001B0 00 00 00 00 00 00 00 00       
000001C0 00 00 00 00 00 00 00 00       
000001D0 00 00 00 00 00 00 00 00       

```

Bessere Signatur:

- *eigentlicher Code des Virus*
- *l nger, damit h here Wahrscheinlichkeit, dass Bytefolge eindeutig ist*

```

seg000:7C40 BE 04 00
seg000:7C40
seg000:7C43
seg000:7C43 next:
seg000:7C43 B8 01 02
seg000:7C46 0E
seg000:7C47 07
seg000:7C48
seg000:7C48 B8 00 02
seg000:7C48 33 C9
seg000:7C4D 8B D1
seg000:7C4F 41
seg000:7C50 2C
seg000:7C51 2E FF 1E 09 00
seg000:7C56 73 0E
seg000:7C58 33 C0
seg000:7C5A 9C
seg000:7C5B 2E FF 1E 09 00
seg000:7C60 4E
seg000:7C61 75 E0
seg000:7C63 EB 35

```

```

mov     si, 4           ; Try it 4 times
;
;
; CODE XREF: sub_7C3A+27↓j
; read one sector
mov     ax, 201h
push    cs
pop      es
assume  es:seg000
mov     bx, 200h        ; to here
xor     cx, cx
mov     dx, cx
inc     cx
pushf
call    dword ptr cs:9   ; int 13
jnb     short fine
xor     ax, ax
pushf
call    dword ptr cs:9   ; int 13
dec     si
jnz     short next
jmp     short giveup

```

Signaturen: Probleme

False Positives (*falscher Alarm*)

- Folge von Bytes „legaler“ Bestandteil eines Programms
 - Wie zwischen gut und böse unterscheiden?

False Negatives

- Virus ist vorhanden, wird aber nicht erkannt
- Ursache: kleine Änderung des Virus-Codes
 - Andere Adressen, Verwendung anderer Register, ...
- *Polymorphe Malware*
 - Viren, die sich bei jeder Infektion selbst verändern, um Signaturprüfung zu entgehen

Signaturen:

Beispiel für false positives

„Stoned“-Virus und Bitcoins

- Virus-Signatur von „Stoned“ wurde im Mai 2014 in die Blockchain von Bitcoin eingeführt
 - Blockchain = Daten, *kein ausführbares Programm*
 - Virens Scanner scannt Dateien auf Festplatte (+evtl. Inhalt des Hauptspeichers), kann nicht zwischen Code und Daten unterscheiden
- Folge: Microsoft Security Essentials erkannte Blockchain als Virus und wollte Datei löschen
 - Bitcoin-SW lädt Blockchain neu
 - Virens Scanner erkennt „Virus“ erneut...

<http://thehackernews.com/2014/05/microsoft-security-essential-found.html>

Weitere Probleme von Signaturen

Mit jedem neuen Virus steigt Größe der Signatur-Datenbank

- Aufwendigerer Scanvorgang – immer mehr Signaturen müssen überprüft werden

Signaturdatenbank muss aktuell gehalten werden

- Neu auftretende Malware wird nicht erkannt, da Hersteller von Virenscannern diese erst analysieren müssen
- Danach wird Signaturdatenbank aktualisiert
- ...und dann (irgendwann) verteilt...

Polymorphe Malware

Ziel: Umgehen der Erkennung durch Signaturen

- Änderung von Bytes in Code des Programms
- Absichtlich: *Mutation* oder *Obfuskation*
 - Einfügen von „**NOP**“: **0x90** = XCHG EAX, EAX oder anderen Instruktionen, die die Semantik nicht ändern
 - Ändern der Codereihenfolge durch Sprungbefehle
 - Ersetzen von (Folgen von) Maschineninstruktionen durch semantisch identische Instruktionen
- ...oder auch eher unabsichtlich durch Adressänderungen im Code, Compileroptimierungen bei „Update“ eines Virus usw.

Malware-Obfuskation:

Beispiel für Einfügen von NOPs (1)

Einfügen von „NOP“-Operationen

- Beibehaltung der Semantik
- aber: andere Signatur

Code aus Chernobyl/CIH-Virus:

Mihai Christodorescu and Somesh Jha,
“Static analysis of executables to detect
malicious patterns”, USENIX Security
Symposium - Volume 12, 2003

Original code

E8 00000000	call 0h
5B	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
50	push eax
0F01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
FA	cli
8B 2B	mov ebp, [ebx]

Obfuscated code

E8 00000000	call 0h
5B	pop ebx
8D 4B 42	lea ecx, [ebx + 45h]
90	nop
51	push ecx
50	push eax
50	push eax
90	nop
0F01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
90	nop
FA	cli
8B 2B	mov ebp, [ebx]

Malware-Obfuskation:

Beispiel für Einfügen von NOPs (2)

Andere Signatur benötigt

- Einfügen von NOPs ist aber an beliebigen Stellen möglich...

<i>Original code</i>	
E8 00000000	call 0h
5B	pop ebx
8D 4B 42	lea ecx, [ebx + 42h]
51	push ecx
50	push eax
50	push eax
0F01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
FA	cli
8B 2B	mov ebp, [ebx]

<i>Obfuscated code</i>	
E8 00000000	call 0h
5B	pop ebx
8D 4B 42	lea ecx, [ebx + 45h]
90	nop
51	push ecx
50	push eax
50	push eax
90	nop
0F01 4C 24 FE	sidt [esp - 02h]
5B	pop ebx
83 C3 1C	add ebx, 1Ch
90	nop
FA	cli
8B 2B	mov ebp, [ebx]

Signature

E800	0000	005B	8D4B	4251	5050
0F01	4C24	FE5B	83C3	1CFA	8B2B

New signature

E800	0000	005B	8D4B	4290	5150
5090	0F01	4C24	FE5B	83C3	1C90
FA8B	2B				

Flexible Erkennung von durch „NOP“ mutierten Varianten

Signatur wird zu *regulärem Ausdruck*

- Möglichkeit der Beschreibung optionaler und wiederholter Bytes und Bytefolgen notwendig
- Aufwendiger zu analysieren

Signature

E800	0000	005B	8D4B	4251	5050
0F01	4C24	FE5B	83C3	1CFA	8B2B

New signature

E800	0000	005B	8D4B	4290	5150
5090	0F01	4C24	FE5B	83C3	1C90
FA8B	2B				

An den mit (90)* markierten Stellen könnten möglicherweise eine oder mehrere NOP-Instruktionen zusätzlich auftauchen

Flexible Signatur

E800	0000	00(90)*	5B(90)*
8D4B	42(90)*	51(90)*	50(90)*
50(90)*	0F01	4C24	FE(90)*
5B(90)*	83C3	1C(90)*	FA(90)*
8B2B			

Malware-Obfuskation:

Beispiel für Code-Umsortierung

Code wird im Speicher umgeordnet

- Verwenden von Sprungbefehlen, um Kontrollfluss zu verändern

Original code

```
call 0h
pop ebx
lea ecx, [ebx+42h]
push ecx
push eax
push eax
sidt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

Code obfuscated through code transposition

```
call 0h
pop ebx
jmp S2
S3:  push eax
     push eax
     sidt [esp - 02h]
     jmp S4
     add ebx, 1Ch
     jmp S6
S2:  lea ecx, [ebx+42h ]
     push ecx
     jmp S3
S4:  pop ebx
     cli
     jmp S5
S5:  mov ebp, [ebx]
```

Malware-Obfuskation:

Verwendung semantisch identischer Instr.

Instruktionen werden ersetzt

- Semantik wird beibehalten
- z.B. push durch explizite SP-Operation + Store ersetzen

Original code

```
call 0h
pop ebx
lea ecx, [ebx+42h]
push ecx
push eax
push eax
sidt [esp - 02h]
pop ebx
add ebx, 1Ch
cli
mov ebp, [ebx]
```

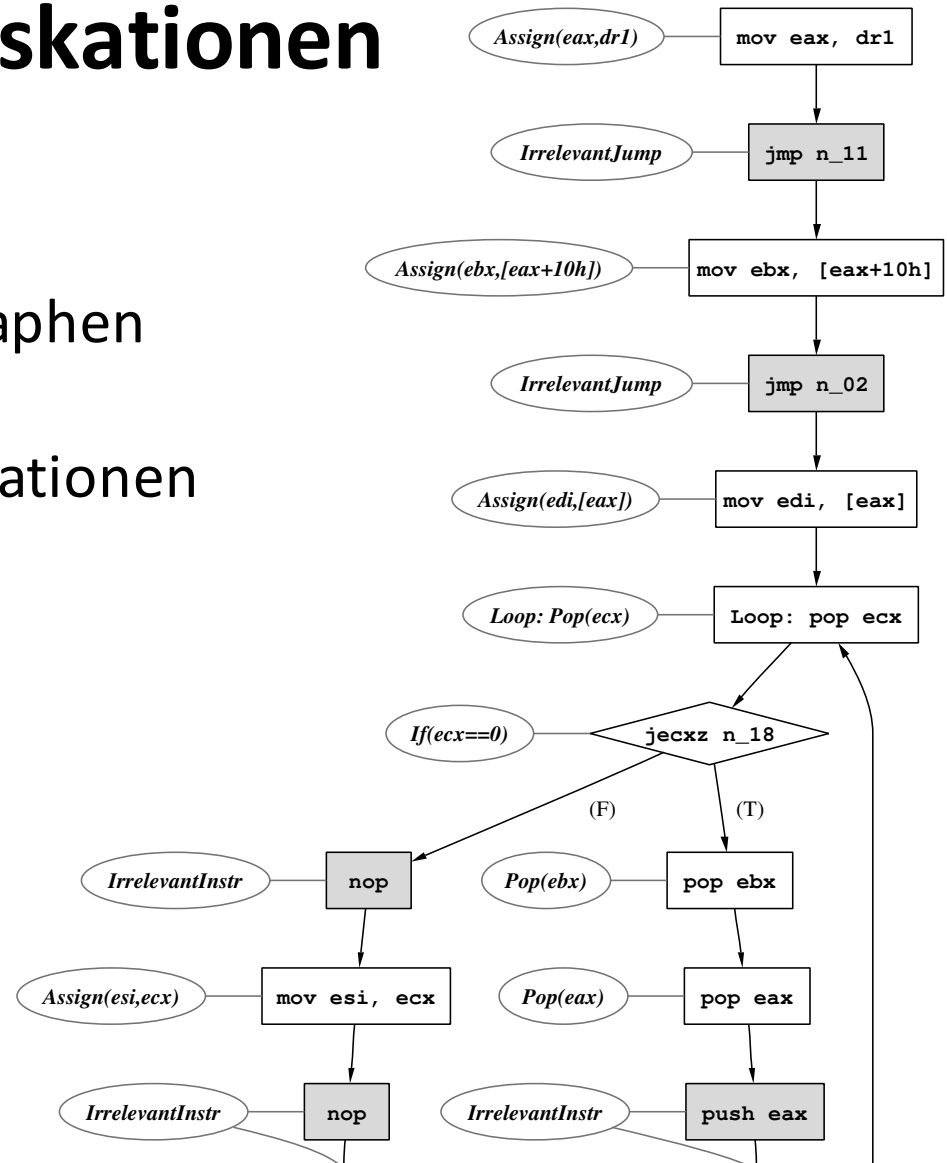
Code obfuscated through instruction substitution

```
call 0h
pop ebx
lea ecx, [ebx+42h]
sub esp, 03h
sidt [esp - 02h]
add [esp], 1Ch
mov ebx, [esp]
inc esp
cli
mov ebp, [ebx]
```

Erkennung von Obfuskationen

Vorgehensweise:

- Erstellen des Kontrollflussgraphen des Maschinencodes
- Markieren „unnötiger“ Operationen (NOP, JMP, etc.)
- Vergleich mit bekanntem Kontrollflussgraphen (*Graph-Isomorphismus*)



Überblick

Obfuskationstechniken

Kombination der verschiedenen Techniken möglich

- Extrem hoher Erkennungsaufwand

<i>Original code</i>	<i>Code obfuscated through dead-code insertion</i>	<i>Code obfuscated through code transposition</i>	<i>Code obfuscated through instruction substitution</i>
call 0h pop ebx lea ecx, [ebx+42h] push ecx push eax push eax sibt [esp - 02h] pop ebx add ebx, 1Ch cli mov ebp, [ebx]	call 0h pop ebx lea ecx, [ebx+42h] nop (*) nop (*) push ecx push eax inc eax (**) push eax dec [esp - 0h] (**) dec eax (**) sibt [esp - 02h] pop ebx add ebx, 1Ch cli mov ebp, [ebx]	call 0h pop ebx jmp S2 S3: push eax push eax sibt [esp - 02h] jmp S4 add ebx, 1Ch jmp S6 S2: lea ecx, [ebx+42h] push ecx jmp S3 S4: pop ebx cli jmp S5 S5: mov ebp, [ebx]	call 0h pop ebx lea ecx, [ebx+42h] sub esp, 03h sibt [esp - 02h] add [esp], 1Ch mov ebx, [esp] inc esp cli mov ebp, [ebx]

**Kommerzielle Virens Scanner
erkannten mutierte
Varianten nicht!**

		Norton® Antivirus 7.0	McAfee® VirusScan 6.01	Command® Antivirus 4.61.2
Chernobyl	original	✓	✓	✓
	obfuscated	✗ ^[1]	✗ ^[1,2]	✗ ^[1,2]
z0mbie-6.b	original	✓	✓	✓
	obfuscated	✗ ^[1,2]	✗ ^[1,2]	✗ ^[1,2]
f0sf0r0	original	✓	✓	✓
	obfuscated	✗ ^[1,2]	✗ ^[1,2]	✗ ^[1,2]
Hare	original	✓	✓	✓
	obfuscated	✗ ^[1,2]	✗ ^[1,2]	✗ ^[1,2]

Polymorphe Malware:

Beispiel für Adreßänderungen

Verwendung einer statisch gelinkten Library

- Identischer Assembler-Quelltext
- Unterschiedliche absolute Adressen durch anderes Speicherlayout
- Komplexere Signaturerkennung notwendig

6A 58	push 58h
68 70 E4 40 00	push offset unk_40E470
E8 9A 04 00 00	call __SEH_prolog4
33 DB	xor ebx, ebx
89 5D E4	mov [ebp+var_1C], ebx
89 5D FC	mov [ebp+ms_exc.disabled], ebx
8D 45 98	lea eax, [ebp+StartupInfo]
50	push eax
FF 15 C0 B0 40 00	call ds:GetStartupInfoA
6A 58	push 58h
68 60 0A 55 00	push offset unk_550A60
E8 BB 05 00 00	call __SEH_prolog4
33 DB	xor ebx, ebx
89 5D E4	mov [ebp+var_1C], ebx
89 5D FC	mov [ebp+ms_exc.disabled], ebx
8D 45 98	lea eax, [ebp+StartupInfo]
50	push eax
FF 15 6C 11 51 00	call ds:GetStartupInfoA

Polymorphe Malware:

Beispiel für Compileroptimierungen

Compiler eliminiert nicht benötigte Maschineninstruktionen

- Oft identischer (z.B. C-)Quelltext
- Optimierungsphase des Compilers erkennt, dass einige erzeugte Maschineninstruktionen in Kontext nicht benötigt werden
- Entfernt diese aus Programm

55		push	ebp
8B EC		mov	ebp, esp
51		push	ecx
8B 45 08		mov	eax, [ebp+arg_0]
89 45 FC		mov	[ebp+var_4], eax
83 7D FC 09		cmp	[ebp+var_4], 9
0F 87 B0 00 00 00		ja	loc_4010C4
8B 4D FC		mov	ecx, [ebp+var_4]
FF 24 8D D8 10 40+		jmp	ds:off_4010D8[ecx*4]
55		push	ebp
8B EC		mov	ebp, esp
8B 45 08		mov	eax, [ebp+arg_0]
83 F8 09		cmp	eax, 9
0F 87 A7 00 00 00		ja	loc_4010B6
FF 24 85 C8 10 40+		jmp	ds:off_4010C8[eax*4]

Komplexere Statische Analyse: Abstrakte Interpretation

Idee

- Ermitteln der Semantik eines Programms, ohne es auszuführen
- Erkennung von Anomalien

Methode: Abstrakte Interpretation

- Informationen über das Verhalten von Programmen (Analyse der Semantik) erhalten, indem man von Teilen des Programms abstrahiert und die Anweisungen Schritt für Schritt nachvollzieht (Interpretation)

Patrick Cousot, Radhia Cousot: *“Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”*, Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1977

Abstrakte Interpretation

Vorgehensweise

- Konzentration auf Teilaspekte der Ausführung der Anweisungen
- Gezieltes Weglassen von Information (Abstraktion)
- Ergebnis: Näherung an die Programmsemantik
 - Kann für gewünschten Zweck vollkommen ausreichend sein
- Viele Eigenschaften von Programmen sind nicht berechenbar
 - Man kann kein Programm angeben, welches sie in endlicher Zeit mit endlichen Speicherressourcen für beliebige Programme berechnet (→ Halteproblem)
- Approximation (Weglassen einiger Informationen) ermöglicht einige weitere Analysen

Abstrakte Interpretation: Beispiel

Abstrakte Interpretation ist komplexer mathematischer Ansatz

- Die Grundprinzipien sind aber einfach
- Beispiel: Bestimmen des Vorzeichens von Variablen

Concrete		Abstract	
Semantics	State	Semantics	State
$x = \boxed{1};$	$\langle x, y, z, w \rangle = \langle 1, ?, ?, ? \rangle$	$x = \boxed{+};$	$\langle x, y, z, w \rangle = \langle +, ?, ?, ? \rangle$
$y = \boxed{-1};$	$\langle 1, -1, ?, ? \rangle$	$y = \boxed{-};$	$\langle +, -, ?, ? \rangle$
$z = x \boxed{*} y;$	$\langle 1, -1, -1, ? \rangle$	$z = x \boxed{*} y;$	$\langle +, -, -, ? \rangle$
$w = x \boxed{+} y;$	$\langle 1, -1, -1, 0 \rangle$	$w = x \boxed{+} y;$	$\langle +, -, -, \top \rangle$

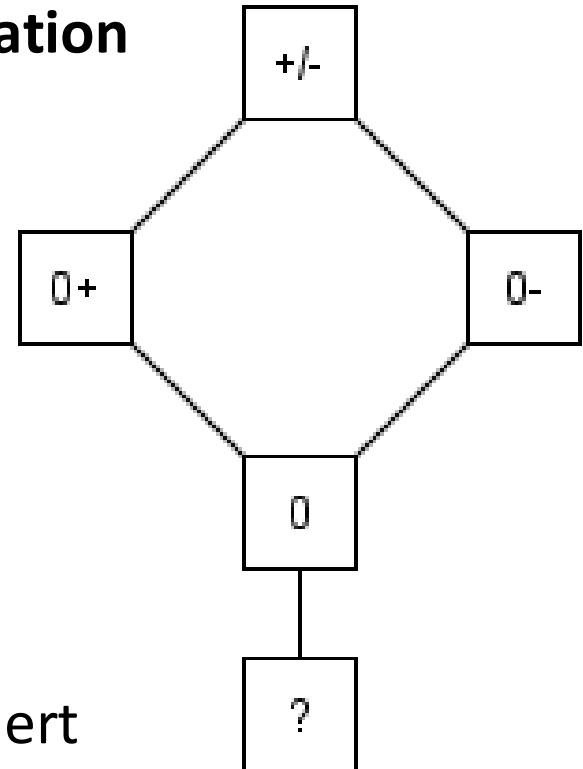


Ersetzen von **konkretem Zustand** durch **abstrakten Zustand**
und **konkreter Semantik** durch **abstrakte Semantik**

Abstrakte Interpretation: Zustandsabstraktion

Verschiedene Methoden der abstr. Interpretation verwenden unterschiedliche Abstraktionen

- Für Vorzeichenanalyse ist jede Variable:
 - unbekannt: positiv oder negativ (+/-)
 - positiv: $x \geq 0$ (0+)
 - negativ: $x \leq 0$ (0-)
 - null (0)
 - uninitialisiert (?)
- Alle weiteren Informationen werden ignoriert
 - Hier z.B. der konkrete Wert der Variablen



Abstrakte Interpretation: Semantikabstraktion (1)

Abstrakte Interpretation befolgt Vorzeichenregeln für die Multiplikation:

- „+“ * „+“ = „+“
- „-“ * „-“ = „+“
- „-“ * „+“ = „+“ * „-“ = „-“

*	?	0	0+	0-	+/-
?	+/-	0	+/-	+/-	+/-
0	0	0	0	0	0
0+	+/-	0	0+	0-	+/-
0-	+/-	0	0-	0+	+/-
+/-	+/-	0	+/-	+/-	+/-

- Hinweis: diese Regeln beziehen sich auf mathematische ganze Zahlen, im Computer repräsentierte Zahlen können überlaufen und unabsichtlich ihr Vorzeichen ändern!

Abstrakte Interpretation: Semantikabstraktion (2)

Abstrakte Interpretation benötigt Vorzeichenregeln für die Addition:

- „+“ + „+“ = „+“
 - „-“ + „-“ = „-“
 - „-“ + „+“ = „+“ + „-“ = „?“
- Beispiel:
 - $-5 + 5 \Rightarrow$ konkretes Ergebnis 0
 - $-6 + 5 \Rightarrow$ konkretes Ergebnis -1
 - $-5 + 6 \Rightarrow$ konkretes Ergebnis 1

+	?	0	0+	0-	+/-
?	+/-	+/-	+/-	+/-	+/-
0	+/-	0	0+	0-	+/-
0+	+/-	0+	0+	+/-	+/-
0-	+/-	0	0-	0+	+/-
+/-	+/-	0	+/-	+/-	+/-

Konkrete Anwendung der abstrakten Interpretation: Sprunganalyse

Herausfinden der Struktur einer compilierten „switch“-Anweisung

- Compiler verwenden Sprungtabellen für effizienten Code

```
switch(x)
{
    case 0: /* ... */ break;
    case 1: /* ... */ break;
    /* ... */
    case 9: /* ... */ break;
    default: /* ... */ break;
}
```

```
cmp     eax, 9           ; switch 10 cases
ja      loc_4010B6        ; default
jmp     ds:off_4010C8[eax*4] ; switch jump

off_4010C8 dd offset loc_401016
dd offset loc_401026
dd offset loc_401036
dd offset loc_401046
dd offset loc_401056
dd offset loc_401066
```

```
switch(x)
{
    case 0: case 2: case 4: case 6:
    case 8: printf("even\n"); break;

    case 1: case 3: case 5: case 7:
    case 9: printf("odd\n"); break;

    default: printf("other\n"); break;
}
```

```
cmp     eax, 9           ; switch 10 cases
ja      short loc_401129 ; default
movzx   eax, ds:index_table[eax]
jmp     ds:off_40113C[eax*4] ; switch jump

off_40113C dd offset loc_401109 ; DATA
dd offset loc_401119 ; jump
index_table
db      0, 1, 0, 1
db      0, 1, 0, 1
db      0, 1
```

Sprunganalyse (2)

Switch mit weit verteilten Werten

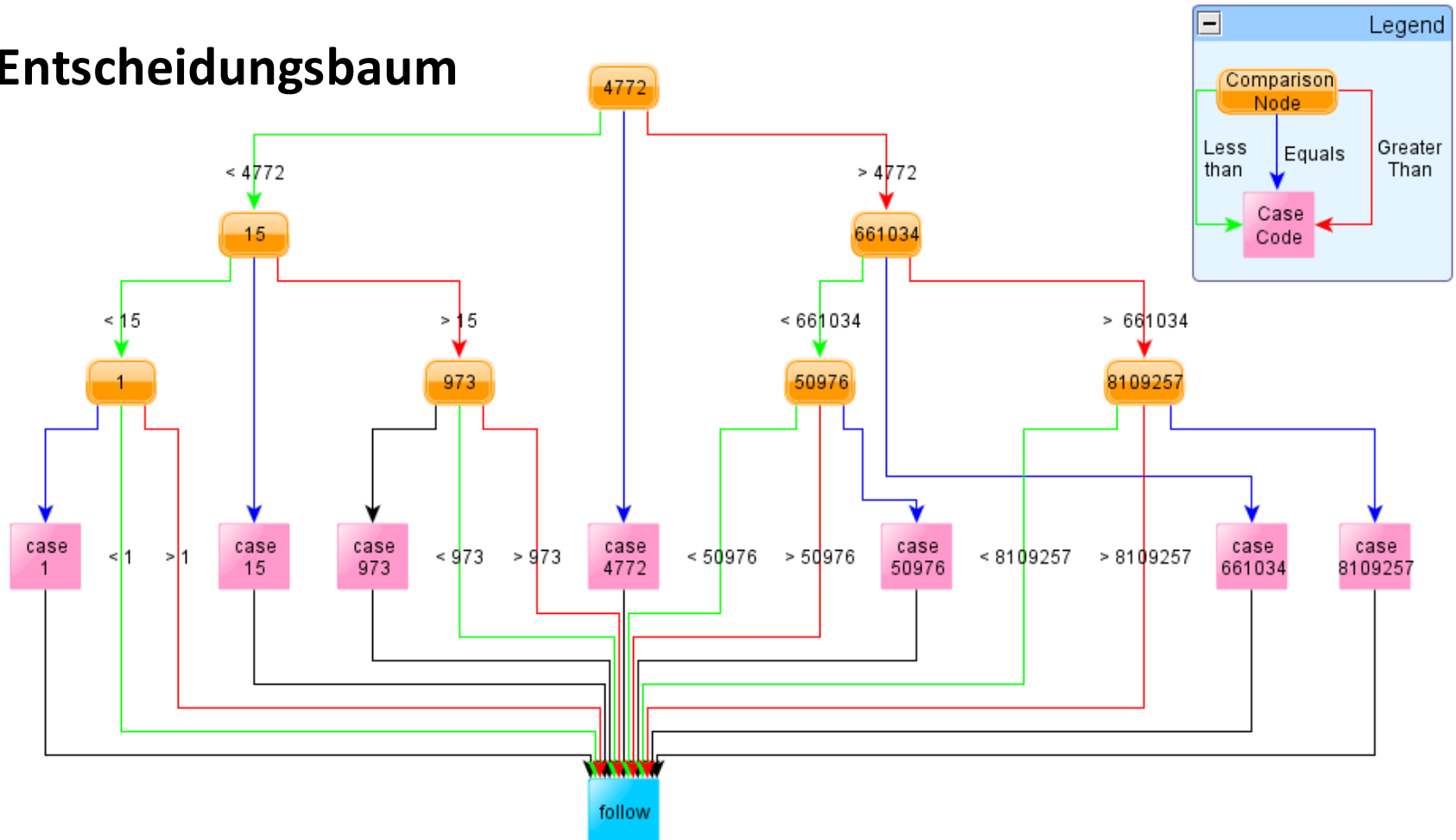
- Sprungtabelle wäre nur dünn besetzt (viele Nullen)
- Ersetzung durch Folgen von „if“-Abfragen ist ineffizient
 - Viele Abfragen und bedingte Sprünge notwendig: $O(N)$
 - stattdessen verwenden Compiler Entscheidungsbäume, Aufwand: $O(\log(n))$

```
switch(x)
{
    case 1: /*1*/ break;
    case 15: /*2*/ break;
    case 973: /*3*/ break;
    case 4772: /*4*/ break;
    case 50976: /*5*/ break;
    case 661034: /*6*/ break;
    case 8109257: /*7*/ break;
}
```

```
if(x == 1) /*1*/ else
if(x == 15) /*2*/ else
if(x == 973) /*3*/ else
if(x == 4772) /*4*/ else
if(x == 50976) /*5*/ else
if(x == 661034) /*6*/ else
if(x == 8109257) /*7*/;
```

Sprunganalyse (3)

Entscheidungsbaum



Sprunganalyse: Assemblercode (1)

Implementierung des Entscheidungsbaums

```
mov     eax, [ebp+arg_0]
cmp     eax, 11270h
jg      short loc_40167B
jz      short loc_40166B
cmp     eax, 3C3h
jg      short loc_401654
jz      short loc_401644
dec     eax
jz      short loc_401634
sub     eax, 11
jnz     loc_4016BE
push    offset a00000012
call    ds:__imp__printf
```

Vergleich mit Entscheidungswert

Größer? Springen!

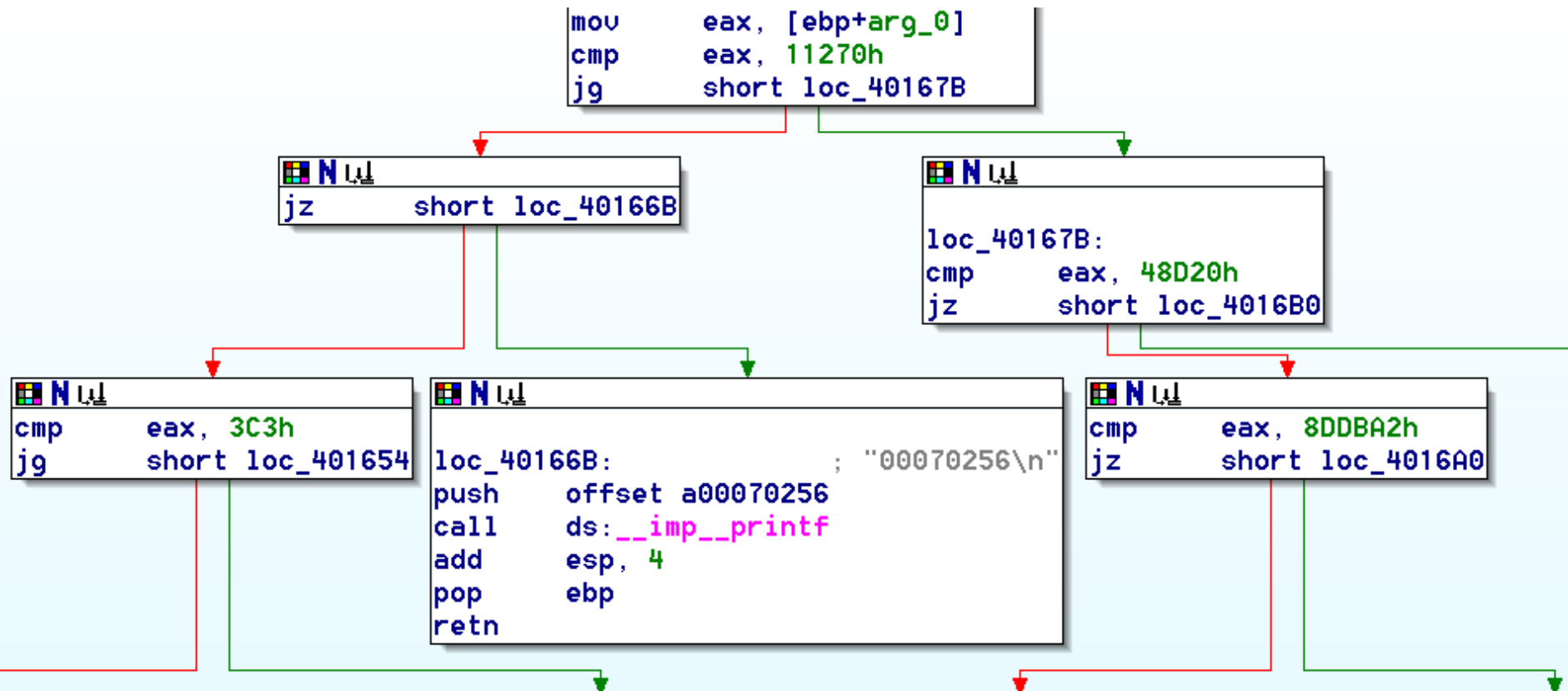
Gleich? Springen!

Sonst: weiter im Code,
nächsten Vergleich ausführen

Kleines Problem:
eax wird von markierten Instruktionen
verändert

Sprunganalyse: Assemblercode (2)

Kontrollflussdarstellung:



Abstraktion der Entscheidungen im switch-Statement

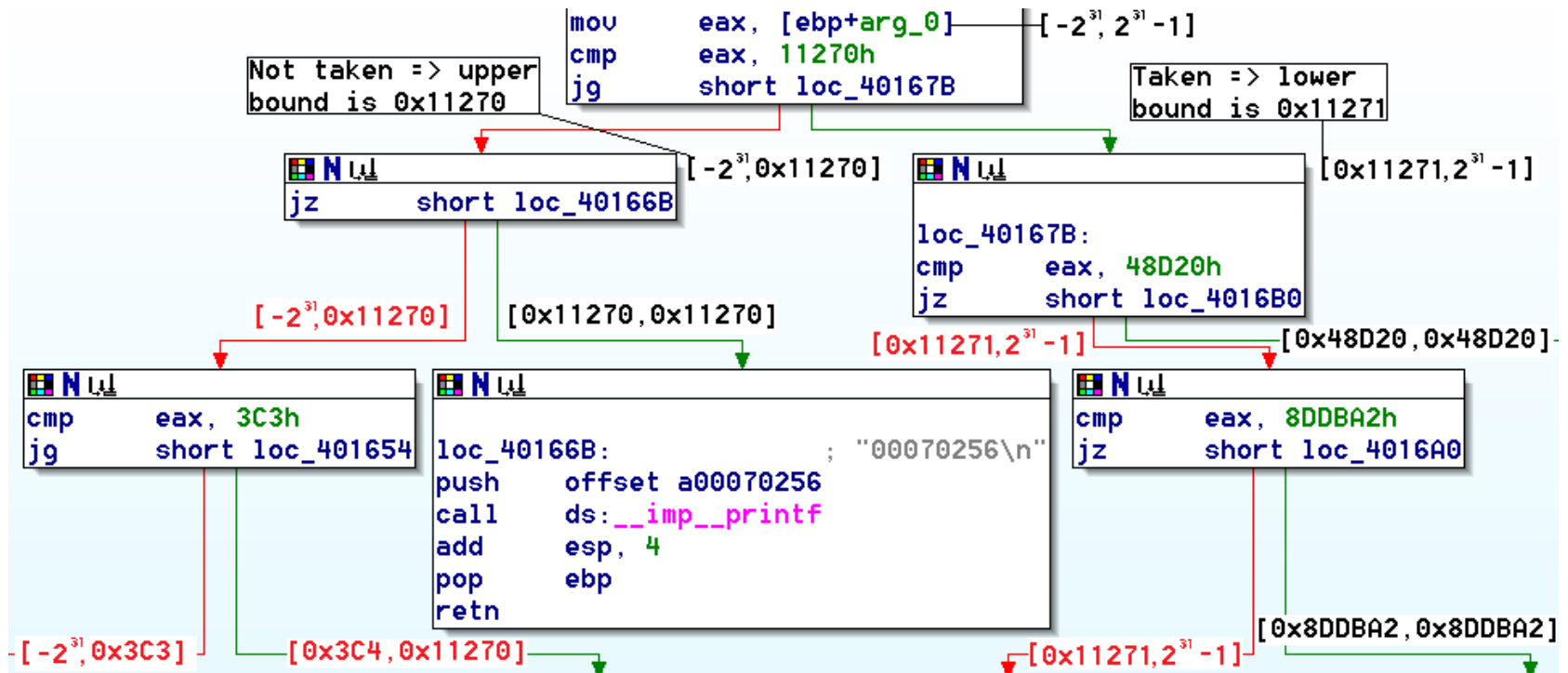
- **Erkenntnis:**
- wichtig: welche *Wertebereiche* führen zur Ausführung welches case-Falls?
- **Datenabstraktion:**
 - Intervalle $[l, u]$ mit $l \leq u$
- switch implementiert mit **sub**, **dec** und **cmp**-Instruktionen – alles Subtraktionen – und bedingten Sprüngen
- **Semantikabstraktion:**
 - Beibehalten der Subtraktionen
 - Verzweigung bei branch-Befehlen

```
switch(x)
{
    case 1: /*1*/ break;
    case 15: /*2*/ break;
    case 973: /*3*/ break;
    case 4772: /*4*/ break;
    case 50976: /*5*/ break;
    case 661034: /*6*/ break;
    case 8109257: /*7*/ break;
}
```

```
mov     eax, [ebp+arg_0]
cmp     eax, 11270h
jg      short loc_40167B
jz      short loc_40166B
cmp     eax, 3C3h
jg      short loc_401654
jz      short loc_401644
dec     eax
jz      short loc_401634
sub     eax, 11
jnz     loc_4016BE
push    offset a00000012
call    ds:__imp__printf
```

Analyseergebnisse switch-Statement

- Keine initiale Information über arg_0 vorhanden
- Jeder Pfad durch Code schränkt mögliche Werte für arg_0 ein
- Wert an Blättern (case-Labels): Wert/einfacher Wertebereich



Dynamische Analysen

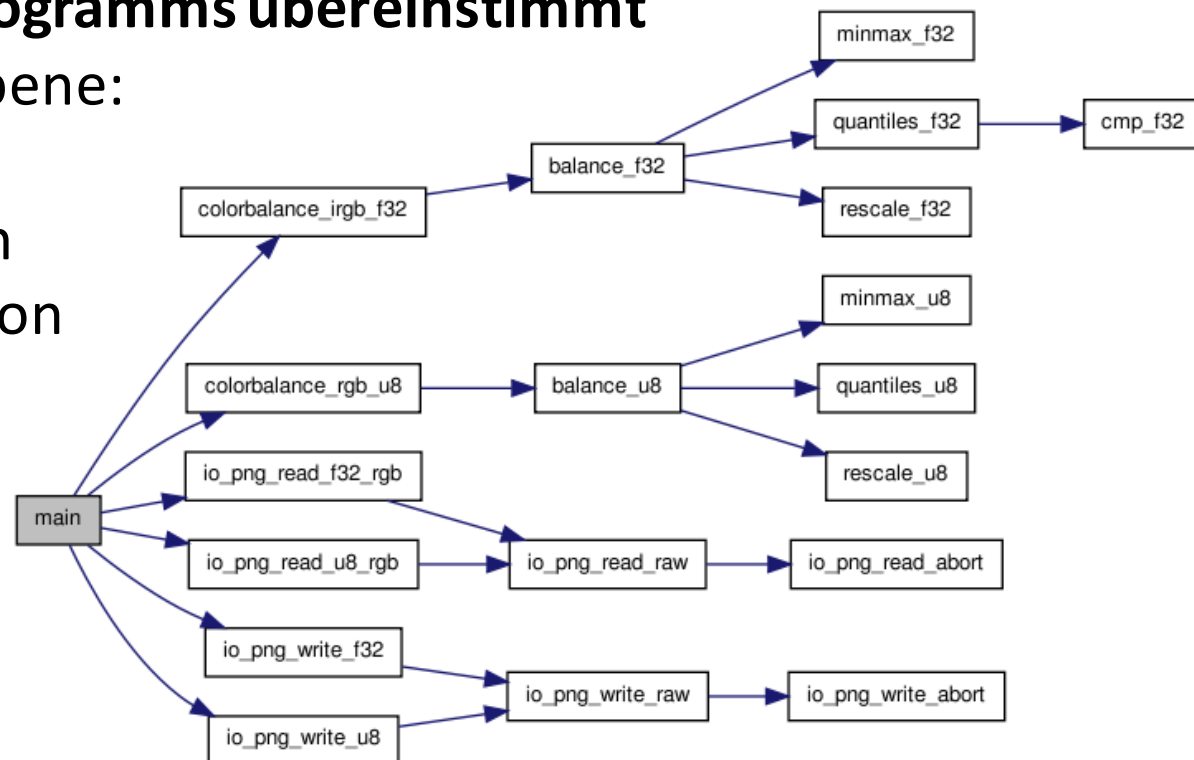
Überwachung des Programmverhaltens zur Laufzeit

- Statische Analysen benötigen Signaturen
 - Können neue und evtl. mutierte Malware nicht erkennen
- Idee der dynamischen Analyse:
 - Betrachten des Verhaltens eines Programms zur Laufzeit
- Semantische Methode – Analyse des *Verhaltens* eines Programms

Dynamische Analyse: Kontrollflussverfolgung

Idee: Überprüfen, ob tatsächlich stattgefundenener Kontrollfluss mit Semantik eines Programms übereinstimmt

- z.B. auf Funktionsebene:
Überprüfen von
Rücksprungadressen
bei Eintritt in Funktion
- Vergleich mit
Funktionsaufruf-
graph
- Entdeckt z.B.
manipulierte Rück-
sprungadressen



The graph illustrates the dependencies between various components of a software project. The nodes are arranged in a hierarchical manner, with 'Solve' at the top left and 'Htot' at the bottom right. The nodes are connected by arrows, indicating the flow of dependencies. Some nodes are highlighted with red borders, including 'WriteDat', 'WritePNG', 'WritePNG2', 'Hcubic', 'Hdemag', 'Helastic', 'Hexchani', 'Hexternal', and 'Htot_Init'.

```

graph TD
    Solve[Solve] --> WriteSet[WriteSet]
    Solve --> Htot_Energy[Htot_Energy]
    Solve --> CalcEnergy[CalcEnergy]
    Solve --> Htot[Htot]
    Solve --> WriteLogPid[WriteLogPid]
    Solve --> Vec3VolAvg[Vec3VolAvg]
    Solve --> MpHext[MpHext]
    Solve --> WriteLogPNode[WriteLogPNode]
    Solve --> WriteLogPNodeInit[WriteLogPNodeInit]
    Solve --> WriteLogData[WriteLogData]
    Solve --> WriteLogInit[WriteLogInit]
    Solve --> Hexternal_hext[Hexternal_hext]
    Solve --> RenormVec[RenormVec]
    Solve --> RHSfunction[RHSfunction]
    Solve --> calc_dMdt[calc_dMdt]
    Solve --> CheckIterationLLG_Init[CheckIterationLLG_Init]
    Solve --> read_one_line[read_one_line]
    Solve --> EquilCheck[EquilCheck]
    Solve --> adjtol[adjtol]
    Solve --> adjtol[adjtol]
    Solve --> PNodeRelnit[PNodeRelnit]
    Solve --> PNodeRelnit[PNodeRelnit]
    Solve --> myTSStepPNode[myTSStepPNode]
    Solve --> CheckIterationLLG[CheckIterationLLG]
    Solve --> DistortVec[DistortVec]
    Solve --> WriteLog[WriteLog]
    Solve --> get_scalf_hstep[get_scalf_hstep]
    Solve --> Hext_ho_hstep[Hext_ho_hstep]
    Solve --> Cart2Sphere[Cart2Sphere]
    Solve --> EminiSolve[EminiSolve]
    Solve --> CheckIterationEmini[CheckIterationEmini]
    Solve --> keepsolving[keepsolving]
    WriteSet --> WriteAVS[WriteAVS]
    WriteSet --> WriteDat[WriteDat]
    WriteSet --> WritePNG[WritePNG]
    WriteSet --> WritePNG2[WritePNG2]
    WriteSet --> SynchronizedFastFPrintf[SynchronizedFastFPrintf]
    Htot_Energy --> CalcEnergy
    CalcEnergy --> Htot[Htot]
    CalcEnergy --> Helastic_Energy[Helastic_Energy]
    CalcEnergy --> Hexchani_Energy[Hexchani_Energy]
    CalcEnergy --> Hexternal_Energy[Hexternal_Energy]
    CalcEnergy --> Hcubic_Energy[Hcubic_Energy]
    CalcEnergy --> Hdemag_Energy[Hdemag_Energy]
    Htot --> Hcubic[Hcubic]
    Htot --> Hdemag[Hdemag]
    Htot --> Helastic[Helastic]
    Htot --> Hexchani[Hexchani]
    Htot --> Hexternal[Hexternal]
    Htot --> Htot_Init[Htot_Init]
    WriteLogPid --> WriteLogPidInit[WriteLogPidInit]
    Vec3VolAvg --> MpHext
    MpHext --> WriteLogPNodeInit
    WriteLogPNode --> WriteLogPNodeInit
    WriteLogData --> WriteLogInit
    WriteLogInit --> Hexternal_hext
    Hexternal_hext --> RenormVec
    RenormVec --> RHSfunction
    RHSfunction --> calc_dMdt
    PNodeRelnit --> RenormVec
    PNodeRelnit --> adjtol
    adjtol --> adjtol
    adjtol --> EquilCheck
    EquilCheck --> CheckIterationLLG
    CheckIterationLLG --> myTSStepPNode
    myTSStepPNode --> CheckIterationLLG
    CheckIterationLLG --> DistortVec
    DistortVec --> WriteLog
    WriteLog --> get_scalf_hstep
    get_scalf_hstep --> Hext_ho_hstep
    Hext_ho_hstep --> Cart2Sphere
    Cart2Sphere --> EminiSolve
    EminiSolve --> CheckIterationEmini
    CheckIterationEmini --> keepsolving
    keepsolving --> WriteSet
    WriteSet --> WriteAVS
    WriteAVS --> SynchronizedFastFPrintf
    SynchronizedFastFPrintf --> Htot_Energy
    Htot_Energy --> Htot
    Htot --> Hcubic
    Htot --> Hdemag
    Htot --> Helastic
    Htot --> Hexchani
    Htot --> Hexternal
    Htot --> Htot_Init
    
```

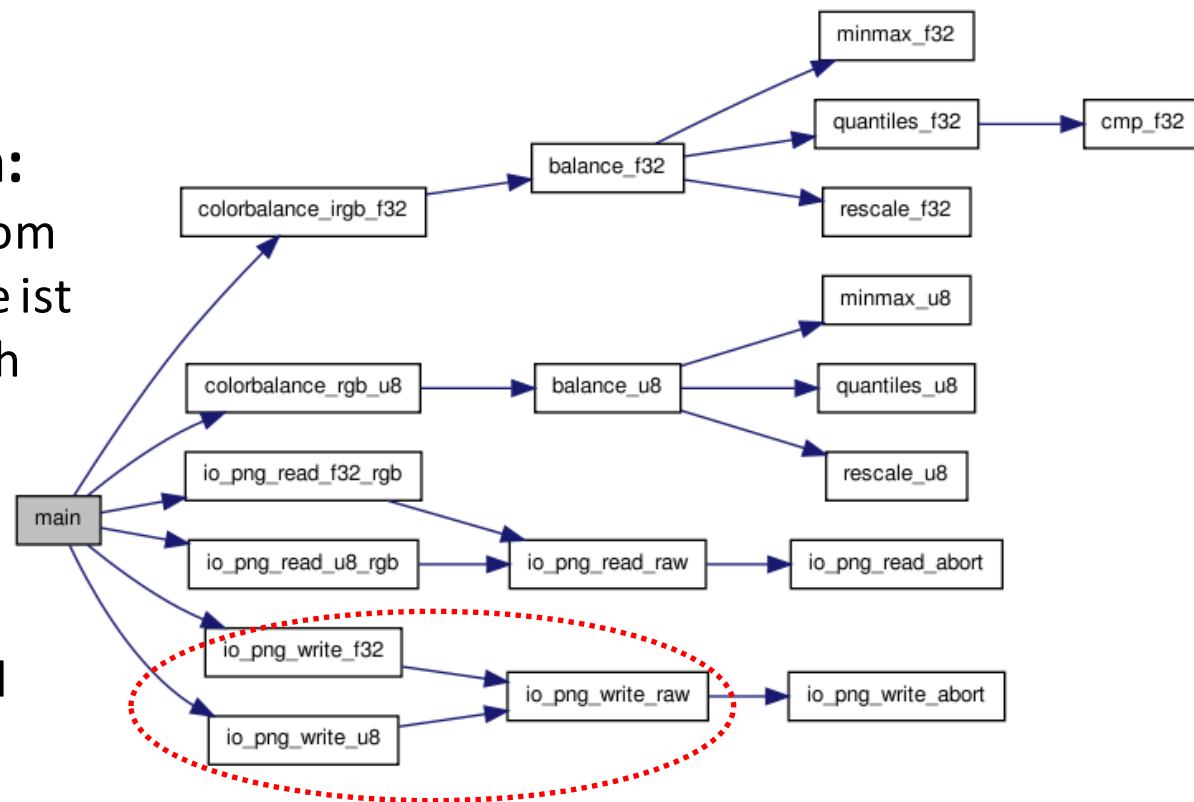
Implementierung der Kontrollflussverfolgung

Idee: Erstellen einer Repräsentation des Aufrufgraphen

- Automatisch durch Compiler erstellbar

Bei Eintritt in Funktion:

- Überprüfen, ob Kante vom Aufrufer (Returnadresse ist auf dem Stack!) in Graph enthalten ist
- z.B. darf (kann?) „io_png_write_raw“ nur von „io_png_write_f32“ und „io_png_write_u8“ aufgerufen werden



Probleme der Kontrollflussverfolgung

Dynamischer Code

- Verwendung von Funktionszeigern/Sprungtabellen:
Compiler kann zur Übersetzungszeit keine vollständige Liste der Funktionen erstellen, die zum Aufruf einer Funktion f führen
 - Also ist der Aufrufgraph unvollständig
- Lösung:
 - Zeigeranalyse wäre erforderlich... **x**

Fazit

Statische Methoden zur Erkennung von Malware

- Signaturchecks sind problematisch
 - Mutierende Malware, mangelnde Aktualität
- Aufwendigere statische Methoden existieren
 - Abstrakte Interpretation

Dynamische Analysen zur Erkennung von Malware

- Kontrollflussanalyse durch Überwachung von Funktionsaufrufen
- mehr: nächste Woche...