# Reverse Engineering

# Reverse Engineering



- Reverse Engineering ist ein Prozess
  - Prozess besteht aus verschiedenen Verfahren
- Analysiert ein bestehendes System
  - Schadsoftware, Virus, Backdoor
- Versucht die Funktionsweise der Software zu ermitteln

# Reverse Engineering Szenario



- Infizierter Rechner
- Schadsoftware löschen oder untersuchen ?
- RE nutzen um Schadsoftware zu verstehen

# Reverse Engineering IT-Forensik



- Beantwortet Fragen
  - Was wurde geloggt ?
  - Wohin wurden die Daten verschickt?
  - Gibt es Logs ?
  - Wurde etwas verschlüsselt ? Wie?

# Reverse Engineering IT-Forensik



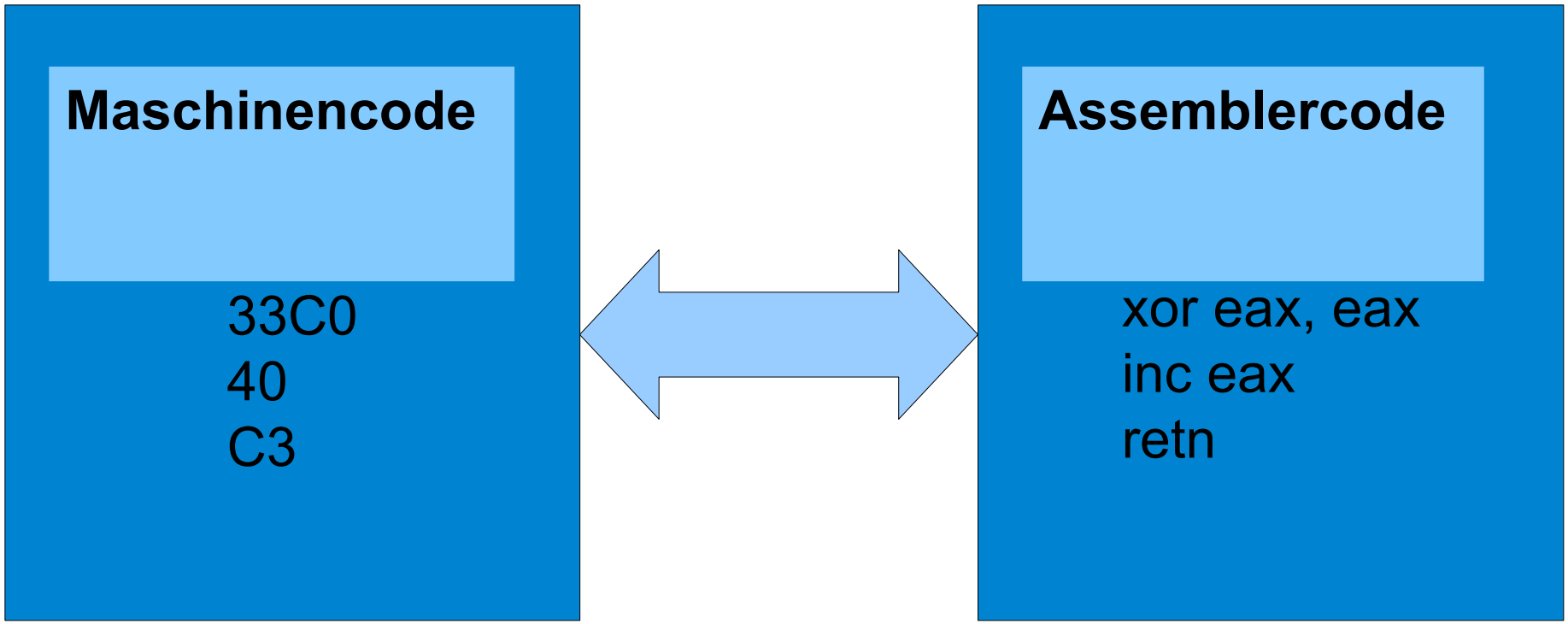
- Kann man nutzen um Tools zu bauen
  - Dateisystem Analyser
  - Dechiffrierer
  - Logfile Tool
  - Backdoors

# Reverse Engineering Verfahren



- Disassembling
  - Wandelt Maschinencode in Assemblercode um

# Disassembling



# Funktion in C

```
bool istZahlAGroesserZahlB(int zahlA, int zahlB) {
    if (zahlA > zahlB)
        return true;
    else
        return false;
}
```

# Funktion im Disassembler



```
sub_401000      proc near               ; CODE XREF: _wmain+7↓p

arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch

                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+arg_0]
                cmp     eax, [ebp+arg_4]
                jle     short loc_401011
                mov     al, 1
                jmp     short loc_401013
; ---------------------------------------------------------------------------
                jmp     short loc_401013
; ---------------------------------------------------------------------------

loc_401011:                             ; CODE XREF: sub_401000+9↑j
                xor     al, al

loc_401013:                             ; CODE XREF: sub_401000+D↑j
                                        ; sub_401000+F↑j
                pop     ebp
                retn
sub_401000      endp
```

# Disassembling

- Keine Wiederherrstellung des Funktionsnamen oder der Parameternamen
- Manuelle Nachbearbeitung notwendig

# Funktion im Disassembler



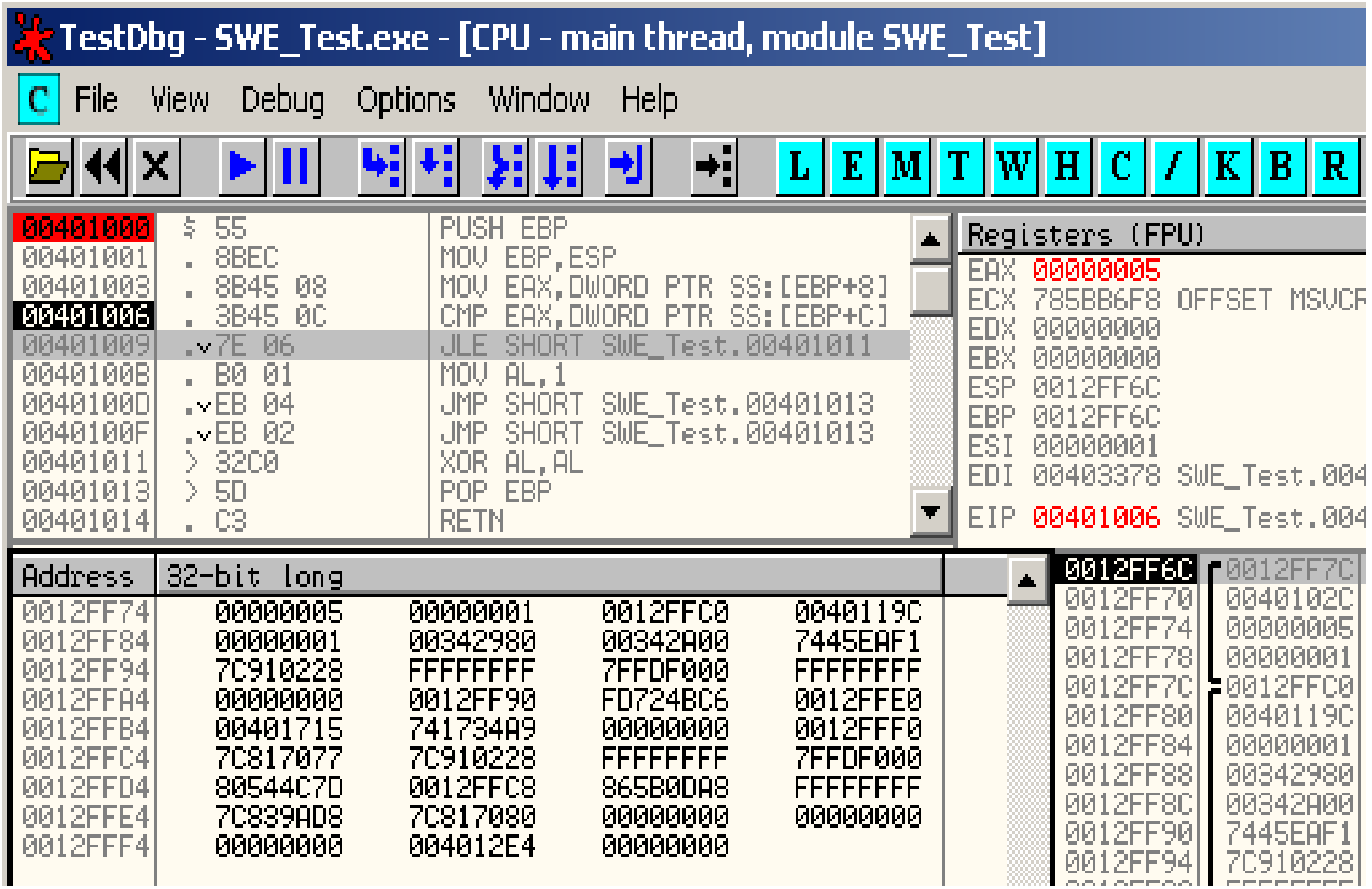
```
bool __cdecl istZahlAGroesserZahlB(int, int) proc near ; CODE XREF:

zahlA           = dword ptr  8
zahlB           = dword ptr  0Ch

                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+zahlA]
                cmp     eax, [ebp+zahlB]
                jle     short ist_nicht_groesser
                mov     al, 1
                jmp     short spring_zurueck
; ---------------------------------------------------------------------------
                jmp     short spring_zurueck
; ---------------------------------------------------------------------------

ist_nicht_groesser:                     ; CODE XREF: istZahlAGroesse
                xor     al, al

spring_zurueck:                         ; CODE XREF: istZahlAGroesse
                                        ; istZahlAGroesserZahlB(int,
                pop     ebp
                retn
bool __cdecl istZahlAGroesserZahlB(int, int) endp
```

# Reverse Engineering Verfahren



- Debuggen
  - Untersucht Programm zur Laufzeit
  - Hilft beim Verständnis des Codes
  - Neue Erkenntnisse nutzen für weitere Kommentare im Disassembler

# Debugger


TestDbg - SWE_Test.exe - [CPU - main thread, module SWE_Test]
File View Debug Options Window Help
L E M T W H C / K B R
00401000 $ 55 PUSH EBP
00401001 . 8BEC MOV EBP,ESP
00401003 . 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
00401006 . 3B45 0C CMP EAX,DWORD PTR SS:[EBP+C]
00401009 .v7E 06 JLE SHORT SWE_Test.00401011
0040100B . B0 01 MOV AL,1
0040100D .vEB 04 JMP SHORT SWE_Test.00401013
0040100F .vEB 02 JMP SHORT SWE_Test.00401013
00401011 > 32C0 XOR AL,AL
00401013 > 5D POP EBP
00401014 . C3 RETN
Registers (FPU)
EAX 00000005
ECX 785BB6F8 OFFSET MSVCR
EDX 00000000
EBX 00000000
ESP 0012FF6C
EBP 0012FF6C
ESI 00000001
EDI 00403378 SWE_Test.004
EIP 00401006 SWE_Test.004
Address 32-bit long
0012FF74 00000005 00000001 0012FFC0 0040119C
0012FF84 00000001 00342980 00342A00 7445EAF1
0012FF94 7C910228 FFFFFFFF 7FFDF000 FFFFFFFF
0012FFA4 00000000 0012FF90 FD724BC6 0012FFE0
0012FFB4 00401715 741734A9 00000000 0012FFF0
0012FFC4 7C817077 7C910228 FFFFFFFF 7FFDF000
0012FFD4 80544C7D 0012FFC8 865B0DA8 FFFFFFFF
0012FFE4 7C839AD8 7C817080 00000000 00000000
0012FFF4 00000000 004012E4 00000000
0012FF6C 0012FF7C
0012FF70 0040102C
0012FF74 00000005
0012FF78 00000001
0012FF7C 0012FFC0
0012FF80 0040119C
0012FF84 00000001
0012FF88 00342980
0012FF8C 00342A00
0012FF90 7445EAF1
0012FF94 7C910228

# Funktion im Disassembler



```
bool __cdecl istZahlAGroesserZahlB(int, int) proc near ; CODE XREF: _main+7↓p

zahlA           = dword ptr  8
zahlB           = dword ptr  0Ch

                push    ebp             ; wert von ebp seichern
                mov     ebp, esp        ; esp in ebp speichern
                mov     eax, [ebp+zahlA] ; zahlA nach eax ablegen
                cmp     eax, [ebp+zahlB] ; zahlA mit zahlB vergleichen
                jle     short ist_nicht_groesser ; springe fals ZahlA <= ZahlB
                mov     al, 1           ; true nach al legen
                jmp     short spring_zurueck ; ebp wieder herstellen
; ---------------------------------------------------------------------------
                jmp     short spring_zurueck ; ebp wieder herstellen
; ---------------------------------------------------------------------------

ist_nicht_groesser:                     ; CODE XREF: istZahlAGroesserZahlB(int
                xor     al, al          ; false auf al legen

spring_zurueck:                         ; CODE XREF: istZahlAGroesserZahlB(int
                                        ; istZahlAGroesserZahlB(int,int)+F↑j
                pop     ebp             ; ebp wieder herstellen
                retn                    ; zurueck springen
bool __cdecl istZahlAGroesserZahlB(int, int) endp
```

# Reverse Engineering Verfahren



- Dekompilierung
  - Übersetzt Assemblercode in höhere Programmiersprache
  - Dekompilierter Code spiegelt den Orginalcode nur logisch wieder

# Dekompilierter Code

```
bool __cdecl istZahlAGroesserZahlB(int zahlA, int zahlB)
{
  return zahlA > zahlB;
}
```

# Reverse Engineering



- Fragen?