Research Group “Embedded Malware”
Prof. Dr. Thorsten Holz
Dipl.-Inf. Ralf Hund
Dipl.-Inf. Carsten Willems

Vorlesung Programmanalyse
Herbst-/Wintersemester 2011/2012
Ruhr-Universität Bochum

3. Übung

24. November 2011
Abgabe der Hausaufgaben: per Moodle bis zum 08. Dezember 2011 - 09:00 Uhr
Bei Fragen und Problemen können Sie sich per E-Mail/Moodle an die Tutoren wenden.

Aufgaben

Aufgabe 1: Rekonstruktion von C-Code (1 Punkt)

Rekonstruieren Sie den C-Code, der folgendes Assembler-Äquivalent hat:

```
00411280  push    ebp
00411281  mov     ebp,esp
00411283  sub     esp,48h
00411286  push    ebx
00411287  push    esi
00411288  push    edi
00411289  mov     dword ptr [ebp-4],0
00411290  mov     byte ptr [ebp-5],0
00411294  mov     eax,dword ptr [ebp+8]
00411297  add     eax,dword ptr [ebp-4]
0041129A  movsx   ecx,byte ptr [eax]
0041129D  test    ecx,ecx
0041129F  jne     004112A7
004112A1  cmp     dword ptr [ebp-4],0Ah
004112A5  jle     004112C0
004112A7  call    externe_funktion
004112AC  movzx   ecx,byte ptr [ebp-5]
004112B0  add     ecx,eax
004112B2  mov     byte ptr [ebp-5],cl
004112B5  mov     eax,dword ptr [ebp-4]
004112B8  add     eax,1
004112BB  mov     dword ptr [ebp-4],eax
004112BE  jmp     00411294
004112C0  pop     edi
004112C1  pop     esi
004112C2  pop     ebx
004112C3  mov     esp,ebp
004112C5  pop     ebp
004112C6  ret
```

Aufgabe 2: Rekonstruktion von Datentypen I (1 Punkt)

Rekonstruieren Sie die einzelnen Datentypen der über `[ebp+8]` übergebenen C-Struktur! Beachten Sie dabei, dass es mehrere mögliche Lösungen gibt, wobei Sie lediglich eine mögliche Lösung abgeben müssen.

```
  mov    eax,  [ebp+8]
  mov    ecx,  0
  mov    cl,   [eax]
  mov    edx,  [ebp+8]
  inc    edx
  mov    dword ptr[edx], ecx
  mov    edx,  [ebp+8]
  movzx  eax,  word ptr [edx+5]
  add    ecx,  eax
  mov    eax,  [ebp+8]
  mov    cx,   [eax+7]
  cmp    ecx,  0Ah
  jz     _loc2

_loc1:
  mov    eax,  [ebp+8]
  mov    edx,  0
  add    eax,  09h
  mov    [eax], dl
  ret

_loc2:
  mov    eax,  [ebp+8]
  mov    ecx,  [eax+0Ah]
  xor    ebx,  ebx
  cmp    ecx,  100h
  jbe    _loc1
  mov    eax,  [ebp+8]
  mov    [eax+0Eh], bl
  ret
```

Aufgabe 3: Rekonstruktion von Datentypen II (1 Punkt)

Erklären Sie in kurzen Worten, welche Art von Datenstrukturen in den folgenden Beispielen verwendet werden und begründen Sie dieses. Auch hier sind mehrere richtige Lösungen möglich.

1.
```
Address     Command
00402413    mov     [ebp-38h], 0
0040241A    jmp     00402425h

0040241C    mov     eax, [ebp-38h]
0040241F    add     eax, 1
00402422    mov     [ebp-38h], eax

00402425    cmp     [ebp-38h], 0Ch
00402429    jge     00402437h
```

```
0040242B    mov     eax, [ebp-38h]
0040242E    mov     ecx, [ebp-38h]
00402431    mov     [ebp+eax*4-2Ch], ecx
00402435    jmp     0040241Ch
```

2.
```
Address     Command
00401300    MOV EBX,DWORD PTR SS:[EBP+8]
00401303    MOV BYTE  PTR DS:[EBX+9],0Fh
00401307    MOV DWORD PTR DS:[EBX+4],0CCCh
0040130E    MOV WORD  PTR DS:[EBX+1],07Bh
00401313    MOV EBX,DWORD PTR DS:[EBX+0A]
00401316    CMP EBX,0
00401319    JNE SHORT 00401303h
```

Aufgabe 4: PEB / Anti-Debugging (2 Punkte)

Manchmal wird die Analyse eines Binaries durch Anti-Debugging Techniken erschwert. Eine Möglichkeit, `IsDebuggerPresent()`, haben Sie bereits in Aufgabe 6 der letzten Übung kennen gelernt. Schreiben Sie nun selbst ein Programm, welches erkennt, **ob ein Debugger geladen** ist und eine entsprechende Meldung ausgibt. Überprüfen Sie dafür die Werte `BeingDebugged` und `NtGlobalFlag` innerhalb des PEB (Process Environment Block), die Rückschlüsse auf die Präsenz eines Debuggers erlauben. Zusätzlich soll der Inhalt der **Kommandozeile incl. der Parameter** ausgegeben werden.

Greifen Sie bitte direkt auf den PEB zu, die Verwendung von `IsDebuggerPresent()` und ähnlichen Funktionen, sowie des Parameters `lpCmdLine`, ist nicht erlaubt! Bitte nutzen Sie Assembler (Vorlage: PEB_Tests.asm) oder C++ (Vorlage: PEB_Tests.cpp) und laden Sie Ihren Quelltext, sowie Ihre ausführbare *.exe-Datei getrennt bei Moodle hoch. Bitte nutzen Sie keine Archive (*.zip/*.rar).

Hinweise:

- Die Adresse des PEB bekommen Sie mit `mov eax, FS:[30h]`.
- Mit dem Befehl `ASSUME FS:NOTHING` in Ihrem Assembler-Quelltext unterdrücken Sie die Fehlermeldung von MASM, die wegen des direkten Zugriffs auf das FS-Register entsteht, mit `ASSUME FS:ERROR` schalten Sie das normale Verhalten wieder ein.
- Schauen Sie sich den Inhalt vom PEB im OllyDebug an, in dem Sie zur ermittelten Adresse navigieren (Strg+G), auf ihren Inhalt (im "Hex dump") rechtsklicken, dann "Decode as structure..." und schließlich "PEB_WINXP" auswählen.
- Ein Doppelklick auf die erste Adresse innerhalb der angezeigten Struktur wechselt zwischen absoluten und relativen Adressen. (Stichwort "Indirekte Adressierung" ⇒ siehe Aufgabe 4 der letzen Übung)
- Nach dem Sie die interessanten Werte gefunden haben, lassen Sie sie von Ihrem Programm ausgeben und beobachten Sie ihre Unterschiede innerhalb und außerhalb eines Debuggers.
- Die Kommandozeile findet sich unter "PEB→ProcessParameters→CommandLine", wobei "ProcessParameters" eine Struktur vom Typ "RTL_USER_PROCESS_PARAMETERS" ist. Da es ein Unicode-String ist, können Sie ihn mit `wprintf()` ausgeben.

Aufgabe 5: PE-Dateien (3 Punkte)

Schreiben Sie ein C-Programm zum Parsen von PE-Dateien. Die folgenden Teilaufgaben sollen dabei erfüllt werden:

- Erkennung, ob es sich um eine gültige DLL oder EXE-Datei handelt (MagicBytes)
- Ausgabe der ImageBase und des Entrypoints
- Ausgabe der Namen der importierten Bibliotheksmodule
- Ausgabe der Namen/Ordinalzahlen der importierten Funktionen

Benutzen Sie keine vorgefertigten Bibliotheken zum Auslesen! Bitte laden Sie sowohl Ihr Programm als auch den Quelltext des Programmes getrennt in Moodle hoch. Bitte nutzen Sie keine Archive (*.zip/*.rar). Im Material finden Sie ein Programmgerüst (*peparser.cpp*), welches Sie für diese Aufgabe verwenden können.

Hinweis: Wenn Sie Visual Studio benutzen, können Sie die bereits vorhandenen PE-Strukturen `IMAGE_DOS_HEADER`, `IMAGE_NT_HEADERS` und `IMAGE_IMPORT_DESCRIPTOR` verwenden.

Aufgabe 6: CrackMe (3 Punkte)

Bei dem Programm `RegMe.exe` handelt es sich um ein etwas schwereres Crackme. Analysieren Sie wie gewohnt das Programm und versuchen Sie die Software „zu registrieren“, indem Sie ein gültiges Keyfile in das Ausführverzeichnis legen. Laden Sie zum Lösen der Aufgabe Ihr Keyfile hoch!

Abgabe der Übungen

Die Hausaufgaben sind bis zur übernächsten Woche in Moodle einzureichen. Sie erreichen die Moodle-Seite per `http://moodle.rub.de`: Loggen Sie sich in das System ein und wählen Sie unter *Meine Kurse* die Veranstaltung *Programmanalyse WS 11/12* aus.