Reverse Engineering

Übung 1 - Gruppe 3

Ludwig Karpfinger	Armin Jeleskovic	Valentin Altemeyer
ludwig.karpfinger@hm.edu	a.jeleskovic@hm.edu	valentin.altemeyer@hm.edu

Aufgabe 1 Zip Datei entpacken

a) Zip Datei entpacken

UI Tool:

Das vorinstallierte UI Tool in Nautilus ist der Meinung, dass die vorliegende Datei passwortgeschützt ist. Das Öffnen ist nicht möglich.

CLI Tool:

```
$ unzip packed.zip
```

Die Fehlermeldung lautet *invalid compressed data to inflate*. Man könnte vermuten, dass dies an einer ungültigen Bytesquenz in der Dateistruktur liegt. Die Datei scheint nicht völlig unbrauchbar zu sein, da folgender Befehl ergibt, dass die zip File eine Textdatei namens Unpackme.txt enthält:

```
$ unzip -l packed.zip
```

b) ZIP reparieren

Eine ZIP-File folgt der Dateistruktur von PKZip[1].

Das *Extra Field* in einer Zip Datei ist vorhanden, um zusätzliche Informationen zu speichern. Es wird der Tipp gegeben, dass die Längenangabe im *extra field* auf 28 bytes gesetzt werden muss. Das Tool *hexedit* kann die Bytes anpassen.

Nun kann die Zip file entpackt werden. Obwohl das möglich ist, wird die Fehlermeldung *invalid zip file with overlapped components (possible zip bomb)* ausgegeben.

Dies konnte nicht gelöst werden.

c) Inhalt ausgeben

Der Inhalt lautet: *Wenn man es weiß ist es eigentlich ganz einfach.*

Aufgabe 2 Einstieg in ELF Reversing

a) main Methode in stripped File

Der Entry Point der Datei findet man durch den gdb Debugger heraus.

```
$ (gdb) info file
```

Der Entry Point lautet: *0x4012d1* Man kann daraus schlussfolgern, dass der Entry Point in *.text* Section liegt, da *.text* bei *0x0000000000401170* beginnt.

```
$ (gdb) set disassembly-flavor intel
$ (gdb) x/20i 0x0000000000401170
```

Es werden 20 Instructions des disassemblierten Codes angezeigt.

Es ist auffällig, dass an der Stelle *0x401198* ein call kommt und davor 3 register vorbereitet werden mit mov. Es wird wahrscheinlich die libc Funktion auf den Stack geladen. Das Programm muss nämlich dynamische Libaries aufrufen. Das bedeutet, dass die Adresse in dem Register *RDI* die Main Methode ist.

Die main Methode liegt also in *0x4012f3*.

Es wird als Entry Point eine Funktion aufgerufen, die mittels *puts* einen String zurückgibt. Der Entry Point der ELF Datei muss also bearbeitet werden. Stattdessen soll das Programm an der Stelle aufgerufen, an welcher der Stack für die main Methode geladen wird.

[1] `https://users.cs.jmu.edu/buchhofp/forensics/formats/pkzip.html`

Im Hexeditor werden die entsprechenden Bytes auf *0x401170* abgeändert.

Nun lässt sich die ELF Datei ausführen ohne Fehler.

b)

Mittels folgendem Befehl werden verlinkte Libaries angezeigt.

```
$ ldd password
```

Ausgabe:

```
linux-vdso.so.1
libcrypto.so.1.1
libc.so.6
libdl.so.2
libpthread.so.0
ld-linux-x86-64.so.2
```

Wir kennen nicht die Adresse der dynamischen Symbole während der Link-Zeit, aber während der Laufzeit. Für das dynamische Symbol wird eine Speicheradresse für die aktuelle Adresse reserviert, welche dann während der Laufzeit gefüllt wird.

Alle dynamisch gelinkten Symbole kann man mit folgendem Befehl sehen:

```
$objdump -R password
```

Ausgabe:

```
DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000403ff0 R_X86_64_GLOB_DAT  __gmon_start__
0000000000403ff8 R_X86_64_GLOB_DAT  __libc_start_main@GLIBC_2.2.5
0000000000404018 R_X86_64_JUMP_SLOT  printf@GLIBC_2.2.5
0000000000404020 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
0000000000404028 R_X86_64_JUMP_SLOT  strlen@GLIBC_2.2.5
0000000000404030 R_X86_64_JUMP_SLOT  MD5_Final@OPENSSL_1_1_0
0000000000404038 R_X86_64_JUMP_SLOT  MD5_Update@OPENSSL_1_1_0
0000000000404040 R_X86_64_JUMP_SLOT  sprintf@GLIBC_2.2.5
0000000000404048 R_X86_64_JUMP_SLOT  __stack_chk_fail@GLIBC_2.4
0000000000404050 R_X86_64_JUMP_SLOT  strcmp@GLIBC_2.2.5
0000000000404058 R_X86_64_JUMP_SLOT  MD5_Init@OPENSSL_1_1_0
0000000000404060 R_X86_64_JUMP_SLOT  __isoc99_scanf@GLIBC_2.7
```

Beispielsweise wird bei der call Funktion am Entry Point die libc libary auf der Adresse *0x403ff8* aufgerufen.

c) password

```
$ (gdb) info functions
```

Es wird die Funktion strcmp gefunden, welche Strings vergleicht. Es könnte also sein, dass diese Funktion genutzt wird, um die Passwörter zu vergleichen. Des Weiteren wurden noch MD5 Funktionen gefunden, welche das Passwort wahrscheinlich hashen.

Die strcmp Funktion wird gedebuggt.

```
$ (gdb) break 0x0000000000401140
```

Der Buchstabe 'e' wird eingegeben.

```
$ (gdb) info register
```

In den Registern müssten wohl das eigentliche Passwort sein und der Buchstabe 'e' in MD5 Hash.

D.h. man muss die Register einzeln als Strings ausgeben und mit einem MD5 Decrypter umkehren. Es wird der online MD5 Decrypter von `https://md5decrypt.net/en/#answer` verwendet.

Es kommt raus, dass das Passwort im *rdi* Register lag, welches mit folgendem Befehl ausgegeben wurde:

```
$ (gdb) x/s 0x4040a0
```

Aufgabe 3 ELF Reversing

a) Entry Point

```
$ readelf -h Execme
```

Es handelt sich um eine:

- 32 bit ELF File
- Header size 52 bytes
- Entry Point 0x804887f
- Section Header komplett stripped

b)

Der Grund für die Fehlermeldungen sind in erster Linie falsche Angaben im Header. Der Header gibt an, dass Section Headers vorhanden seien. Dies entspricht nicht der Wahrheit.

c)

Im ELF Header werden alle Referenzen, die auf einen Section Header zeigen, mit *0* überschrieben. Dies wird über das Tool namens hexedit bewerkstelligt.

d)

e)