Reverse Engineering

Wie schau ich den Anwendungen unter die Haube?

Agenda

- Warum will ich Anwendungen unter die Haube schauen
- Ueberblick ueber Anwendungsarchitekturen
- Generelle Einfuehrung in RE
  - Android/Java
  - Binary / Linux, Windows, Mac OS/iOS
- Anti RE

Warum?

Sicherheit

- Finden von Schwachstellen
- Finden von BackDoors/Bypass
- Malware Analyse
- Kontrolle von Releases

Entwicklung

- Alte Software - keine Entwickler mehr
- API/Schnittstellen von propreritaerer Software
  Open Source Alternativen
- Neugier
- Algorithmen
- Industriespionage

Ueberblick Architekturen

Skriptsprachen

- Bash/Batch
- Perl/Python/Ruby/JavaScript
- Plain Text - einfach den Code lesen

- Kein Reversing von noeten – Code einfach lesbar.

Binaere Anwendungen

- Abhaengig von CPU Architektur – amd64/x86_64
- Abhaengig vom Betriebsystem
- Formate gute dokumentiert

- # file <xxx>

```
/usr/bin/ssh: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux
2.6.32, BuildID[sha1]=7dcdbd1e1d7a87e0e78392efdcd0d7f0bd65b4b2, stripped
```

Native Windows

- Portable Executeable (PE) (Win32/64)
- Historie in DOS – MZ Format – 16 bit

```
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 e8 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  73 b2 8d 21 37 d3 e3 72  37 d3 e3 72 37 d3 e3 72  |s..!7..r7..r7..r|
00000090  ea 2c 2c 72 36 d3 e3 72  ea 2c 2e 72 35 d3 e3 72  |.,,r6..r.,.r5..r|
```

Native Linux

- Executable and Linkable Format (ELF)
- ABI Version fuers Betriebsystem

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  f0 d0 00 00 00 00 00 00  |..>.............|
00000020  40 00 00 00 00 00 00 00  30 14 0b 00 00 00 00 00  |@.......0.......|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1d 00 1c 00  |....@.8...@.....|
00000040  06 00 00 00 05 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |@.......@.......|
00000060  f8 01 00 00 00 00 00 00  f8 01 00 00 00 00 00 00  |................|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
00000080  38 02 00 00 00 00 00 00  38 02 00 00 00 00 00 00  |8.......8.......|
00000090  38 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00  |8...............|
```

Native Mac OS/iOS

- Mach Object (Mach-O)
- Multi-architecture binaries

```
00000000  cf fa ed fe 07 00 00 01  03 00 00 80 02 00 00 00  |................|
00000010  12 00 00 00 d0 08 00 00  85 00 20 00 00 00 00 00  |.......... .....|
00000020  19 00 00 00 48 00 00 00  5f 5f 50 41 47 45 5a 45  |....H...__PAGEZE|
00000030  52 4f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |RO..............|
00000040  00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  19 00 00 00 c8 02 00 00  |................|
00000070  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000080  00 00 00 00 01 00 00 00  00 10 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
```

Virtuelle Maschine (VM)

- Laufzeitumgebung abhaengig vom Betriebssystem
- Anwendungen sind betriebsunabhaengig compiliert in Binary Formaten (Bytecode)
- .NET ist zwar im PE Format, aber …
- Oft in Archiven (jar, war)

```
00000000  ca fe ba be 00 00 00 34  00 1d 0a 00 06 00 0f 09  |.......4........|
00000010  00 10 00 11 08 00 12 0a  00 13 00 14 07 00 15 07  |................|
00000020  00 16 01 00 06 3c 69 6e  69 74 3e 01 00 03 28 29  |.....<init>...()|
00000030  56 01 00 04 43 6f 64 65  01 00 0f 4c 69 6e 65 4e  |V...Code...LineN|
00000040  75 6d 62 65 72 54 61 62  6c 65 01 00 04 6d 61 69  |umberTable...mai|
00000050  6e 01 00 16 28 5b 4c 6a  61 76 61 2f 6c 61 6e 67  |n...([Ljava/lang|
00000060  2f 53 74 72 69 6e 67 3b  29 56 01 00 0a 53 6f 75  |/String;)V...Sou|
00000070  72 63 65 46 69 6c 65 01  00 0f 48 65 6c 6c 6f 57  |rceFile...HelloW|
00000080  6f 72 6c 64 2e 6a 61 76  61 0c 00 07 00 08 07 00  |orld.java.......|
```

Alle Formate

- Header (statische Laenge)
- Verschiedene Sektionen/Untersektionen

- objdump –h <file)

```
 0 .interp       0000001c  0000000000000238  0000000000000238  00000238  2**0
                 CONTENTS, ALLOC, LOAD, READONLY, DATA
...
13 .text        00064592  0000000000009820  0000000000009820  00009820  2**4
                 CONTENTS, ALLOC, LOAD, READONLY, CODE
```

Action

Windows .NET/MONO

- iLSpy
- Laden – fertig

- DEMO

Android DEX/Java Class

- Anwendung vom Telefon laden (apk)
  adb shell pm list packages –f
  adb pull <xxx> .

- Umwandeln von DEX in Java (dex2jar)

- APK Tool (smali Code/XML Dateien)

- DEMO

Java Class

- JD-Core ist nicht so toll
- CFE geht super

- DEMO

ELF

- Disassemble mit Huerden

- String Liste:
- objdump -S -j .rodata <elf datei>

- Anzeigen von Assembler Code:
- objdump –d <elf datei>

- DEMO

Portable Executable

- Windows Debugger Tools
- IDA Pro (sehr teuer)

- DEMO

Mach-O

- “otool” kommt mit xcode

- # xcrun otool -v -s TEXT cstring <mach-o executrable>

```
Contents of (__TEXT,__cstring) section
0000000100000fa6  Hello World!\n
```

Mach-O

- Anzeigen von Assembly code
- # xcrun otool -v -t a.out

```
(__TEXT,__text) section
_main:
0000000100000f50        pushq   %rbp
0000000100000f51        movq    %rsp, %rbp
0000000100000f54        subq    $0x20, %rsp
0000000100000f58        leaq    0x47(%rip), %rax
0000000100000f5f        movl    $0x0, -0x4(%rbp)
0000000100000f66        movl    %edi, -0x8(%rbp)
0000000100000f69        movq    %rsi, -0x10(%rbp)
0000000100000f6d        movq    %rax, %rdi
0000000100000f70        movb    $0x0, %al
0000000100000f72        callq   0x100000f84
0000000100000f77        xorl    %ecx, %ecx
0000000100000f79        movl    %eax, -0x14(%rbp)
0000000100000f7c        movl    %ecx, %eax
0000000100000f7e        addq    $0x20, %rsp
0000000100000f82        popq    %rbp
0000000100000f83        retq
```

- C vs Objective-C

```
# xcrun nm -nm a.out
                 (undefined) external _NSFullUserName (from Foundation)
                 (undefined) external _NSLog (from Foundation)
                 (undefined) external _OBJC_CLASS_$_NSObject (from libobjc)
                   (undefined) external _OBJC_METACLASS_$_NSObject (from libobjc)
                 (undefined) external ___CFConstantStringClassReference (from CoreFoundation)
                 (undefined) external __objc_empty_cache (from libobjc)
                 (undefined) external _objc_autoreleasePoolPop (from libobjc)
                 (undefined) external _objc_autoreleasePoolPush (from libobjc)
                 (undefined) external _objc_msgSend (from libobjc)
                 (undefined) external dyld_stub_binder (from libSystem)
0000000100000000 (__TEXT,__text) [referenced dynamically] external __mh_execute_header
0000000100000e80 (__TEXT,__text) external _main
0000000100000f00 (__TEXT,__text) non-external -[Foo run]
0000000100001138 (__DATA,__objc_data) external _OBJC_METACLASS_$_Foo
0000000100001160 (__DATA,__objc_data) external _OBJC_CLASS_$_Foo
```

- # Anzeigen vom laden dynamischer Bibliotheken

```
# (export DYLD_PRINT_LIBRARIES=; ./a.out )
dyld: loaded: /Users/jens/build/reversing_macho/m-sample/./a.out
dyld: loaded: /System/Library/Frameworks/Foundation.framework/Versions/C/Foundation
dyld: loaded: /usr/lib/libSystem.B.dylib
dyld: loaded: /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
dyld: loaded: /usr/lib/libobjc.A.dylib
dyld: loaded: /usr/lib/libauto.dylib
dyld: loaded: /usr/lib/libextension.dylib
...
dyld: loaded: /System/Library/Frameworks/ServiceManagement.framework/Versions/A/ServiceManagement
dyld: loaded: /usr/lib/libxslt.1.dylib
2016-09-03 09:49:47.863 a.out[21751:2161310] Jens Muecke
```

Portable Executable

- Anzeigen vom laden dynamischer Bibliotheken

```
# (export DYLD_PRINT_LIBRARIES=; ./a.out )
dyld: loaded: /Users/jens/build/reversing_macho/m-sample/./a.out
dyld: loaded: /System/Library/Frameworks/Foundation.framework/Versions/C/Foundation
dyld: loaded: /usr/lib/libSystem.B.dylib
dyld: loaded: /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
dyld: loaded: /usr/lib/libobjc.A.dylib
dyld: loaded: /usr/lib/libauto.dylib
dyld: loaded: /usr/lib/libextension.dylib
...
dyld: loaded: /System/Library/Frameworks/ServiceManagement.framework/Versions/A/ServiceManagement
dyld: loaded: /usr/lib/libxslt.1.dylib
2016-09-03 09:49:47.863 a.out[21751:2161310] Jens Muecke
```

Firmware Image

- Binwalk ist dein Freund
- Extrahieren von Filesystems/Binary/bekannten Dateiformaten

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
104448        0x19800         U-Boot version string, "U-Boot 1.1.3 (Sep 21 2015 - 10:10:04)"
106096        0x19E70         xz compressed data
127264        0x1F120         uImage header, header size: 64 bytes, header CRC: 0x1941DFF, created:
2016-03-21 03:02:14, image size: 1864874 bytes, Data Address: 0x80000000, Entry Point: 0x8000C310, data
CRC: 0x57F98D02, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name:
"Linux Kernel Image"
127328        0x1F160         LZMA compressed data, properties: 0x5D, dictionary size: 33554432 bytes,
uncompressed size: 4813872 bytes
1992202       0x1E660A        JFFS2 filesystem, little endian
```

Anti RE Techniken

Obfuscator

- ProGuard
- allatori
- klassmaster
- ClassGuard
- yGuard
- stunnix

Binary Executable Packer

- aspack
- UPX
- mpress

Anti Debugging

- BOOL WINAPI IsDebuggerPresent(void);
- Self Debugging
- Checksumme
- Pruefen auf VMs/Sandbox
- Pruefen auf DLLs
- Rogue instructions
  - INT3
  - INT 2Dh - Breakpoint Exception
  - “Trap flag”

Danke fuers zuhoeren

Q&A

(jens@nons.de/jensmuecke@kryptonsecurity.com