Reverse-Engineering

From YobiWiki

Contents

Static Analysis Tools

IDA Pro

Metasm (http://metasm.cr0.org/)

Metasm is a cross-architecture assembler, disassembler, compiler, linker and debugger.
It has some advanced features such as live process manipulation, GCC/Microsoft Visual Studio-compatible preprocessor, automatic backtracking in the disassembler (similar to "slicing"), C headers shrinking, linux/windows/remote debugging API interface, a C compiler/decompiler, a gdb-server compatible debugger, and various advanced features. It is written in pure Ruby, with no dependency.
Intel IA32 (16/32/64bits), MIPS, PPC. Ongoing: ARM
MZ, PE/COFF (32 and 64 bits), ELF (32 and 64 bits), Mach-O (incomplete) and UniversalBinary

REC Studio (http://www.backerstreet.com/rec/rec.htm)

- x86, x64
- Windows, Linux, Mac OS X
- HLA disassembler

Useful commands:

```
help
strings
calltree
showprocs
decompile /tmp/myprog.c
```

click on a function in the "Project" function list to HLA disass it

Hopper (http://www.hopperapp.com/)

- Intel (32 and 64bits), and ARM (ARMv6, ARMv7 and ARM64) processors
- Mach-O binaries (Mac and iOS), PE32/32+/64 Windows binaries and ELF binaries
- decompiler
- debugger
- patcher

Capstone (http://www.capstone-engine.org/)

- ARM, ARM64 (ARMv8), Mips, PowerPC, Sparc, SystemZ & Intel

Radare (http://radare.nopcode.org/y/)

The reverse engineering framework

- Book (https://maijin.github.io/radare2book/)
- Slides (http://rada.re/get/condret-r2talk.pdf)
- tuto (http://samsymons.com/blog/reverse-engineering-with-radare2-part-1/)

Bokken (https://inguma.eu/projects/bokken)

GUI
git repo (https://github.com/inguma/bokken) synced with mercurial repo

Amoco (https://github.com/bdcht/amoco)

Amoco is a python package dedicated to the (static) analysis of binaries
Very young but promising, seems easy to add an arch
With BBL symbolic execution

Miasm (https://code.google.com/p/miasm/)

Miasm is a a free and open source (GPLv2) reverse engineering framework. Miasm aims at analyzing/modifying/generating binary programs. * opening/modifying/generating PE/ELF 32/64 le/be using Elfesteem

- Assembling/Disassembling ia32/ppc/arm
- Representing assembly semantic using intermediate language
- Emulating using jit (dynamic code analysis, unpacking, ...)
- Expression simplification for automatic de-obfuscation
- ...

Medusa (https://github.com/wisk/medusa)

Medusa is a disassembler designed to be both modular and interactive. It runs on Windows and Linux

Snowman (http://derevenets.com/)

(was called SmartDec)
Native code to C/C++ decompiler
x86 and x86-64 architectures, ELF and PE file formats
IDA Pro & standalone versions, for Windows
Standalone i86 Windows version runs fine under Wine

Misc Static

Distorm (https://code.google.com/p/distorm/)

diStorm3 is really a decomposer, which means it takes an instruction and returns a binary structure which describes it rather than static text, this is great for advanced binary code analysis

PyPEELF (http://sourceforge.net/apps/trac/pypeelf)

PyPEELF is a multi-platform binary editor written in Python, wxPython and BOA Constructor. It allows you to manage binary data in PE32, PE32+ (x64) and ELF binary files.

PyPEELF uses pefile to manage PE32 and PE32+ files and pyelf to manage ELF files. Besides, it uses winappdbg and pydasm in some others features like Task Running Viewer and Disassembling files.

PyPEELF was designed for Reverse Engineers who want to edit or visualize binary file data in multi-platforms. That is why PyPEELF runs under Windows and Unix/BSD operating systems

Retargetable decompiler (http://decompiler.fit.vutbr.cz/)

Support ELF & PE for Intel x86, ARM, ARM+Thumb, MIPS, PIC32, and PowerPC architectures
Online decompilation service (http://decompiler.fit.vutbr.cz/decompilation/) available!

binwalk (http://binwalk.org/)

Binwalk is a fast, easy to use tool for analyzing and extracting firmware images.

PREF (http://dax89.comlu.com/?page_id=21)

Portable Reverse Engineering Framework
On github (https://github.com/Dax89/PREF)

```
apt-get install qtbase5-dev ...
qmake
make
```

Bindead (https://bitbucket.org/mihaila/bindead/wiki/Home)

A static analysis tool for binaries.
ELF/PE, x86/x64, IL RREIL, DBI PIN

Hexinator (https://hexinator.com/)

A powerful hexadecimal editor

```
sudo apt-key adv --keyserver pgp.mit.edu --recv-keys A04A6C4681484CF1
sudo apt-get install apt-transport-https
```

```
echo "deb [arch=amd64,i386] https://hexinator.com/downloads/ synalysis non-free" |sudo tee /etc/apt/sources.list.d/hexinator.list
sudo apt-get update
sudo apt-get install hexinator
```

MC-Semantics (https://github.com/trailofbits/mcsema)

A library for translating the semantics of native code to LLVM IR. McSema support translation of x86 machine code, including integer, floating point, and SSE instructions

joelpx/reverse (https://github.com/joelpx/reverse)

Reverse engineering (x86 / elf) to pseudo-C

Poor man's tools

File, -z to uncompress, -s to inspect non-files, e.g. /dev/sda1

```
file -k [-z] [-s] mybin
```

Strings

```
strings [-n min_length] -a -e [s|S|b|l|B|L] mybin
```

Android

Documentation

- Dalvik (http://source.android.com/devices/tech/dalvik/index.html) : bytecode, dex & VM instructions

Dex2jar (http://code.google.com/p/dex2jar/)

A tool for converting Android's .dex format to Java's .class format
See also DeObfuscate jar with dex tool (https://code.google.com/p/dex2jar/wiki/DeObfuscateJarWithDexTool)

```
./d2j-dex2jar.sh myapp.apk
```

This returns a file myapp-dex2jar.jar
Then use Java decompilers: jad, jd-gui, cf below

enjarify (https://github.com/google/enjarify)

Similar to dex2jar but newer and supposed to handle cases where dex2jar was failing

Smali (http://code.google.com/p/smali/)

smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation

Examples:

- https://leonjza.github.io/blog/2015/02/09/no-more-jailbreak-detection-an-adventure-into-android-reversing-and-smali-patching/

Apktool (https://code.google.com/p/android-apktool/)

https://github.com/brutall/brut.apktool
It is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug smali code step by step. Also it makes working with app easier because of project-like files structure and automation of some repetitive tasks like building apk, etc.

```
apktool d myapp.apk
```

Apk Multi-Tool (http://apkmultitool.com/node/7)

Swiss knive (was Apk Manager)
Contains apktool, smali/baksmali etc
on Github (https://github.com/APK-Multi-Tool) for Linux release

- 9 decompile apk / 1 select apk / 9 decompile apk

GetStrings

Small script to prepare a sed script to inject resource strings into jad, to ease reversing.
Update: inject resource names for other resources than strings, will still be more explanatory than 0x7f123456
To be used e.g. after apktool / Apk Multi-Tool decompilation

```
#!/bin/bash
DECOMPILED_DIR=working/*apk/
cat $DECOMPILED_DIR/res/values/public.xml|grep "type=.string"|\
    sed 's/.*name="\?//;s/" id="\?/ /;s/"\? \/>//'|\
    awk --non-decimal-data '{print $2, int($2), $1}'\
    > getstring-pub
cat $DECOMPILED_DIR/res/values/strings.xml|grep '<string'|\
    sed 's/.*name="\?//; s/"\?>/ /;s/<\/string>//;s/#/\\#/g'\
    > getstring-str
join -1 3 -2 1 --nocheck-order getstring-pub getstring-str|\
    sed 's/[^ ]\+ \([[:alnum:]]\+\) [[:alnum:]]\+ \(.*\)/s#\1#"\2"#/'\
    > getstring-sed
rm getstring-pub getstring-str
cat $DECOMPILED_DIR/res/values/public.xml|grep "type="|\
    grep -v "type=.string"|\
    sed 's/.*type="\(.*\)" name="\(.*\)" id="\(.*\)" \/>/s#\3#\1:\2#/'\
    >> getstring-sed
```

SetStrings

```
find $1 -name "*.jad" -exec sed -i -f getstring-sed {} \;
```

Soot (http://www.sable.mcgill.ca/soot/)

Soot is a Java bytecode analysis and transformation framework, now supporting Dalvik too.
Get soot.jar

Help:

```
java -jar soot.jar --help|less
```

SootDisassembleApkToJimple.sh

```
#In case you don't have the right platform android.jar, you can force using another one, e.g.:
#FORCEJAR="-force-android-jar /path/to/android-sdk-linux_x86/platforms/android-17/android.jar"
java -jar soot.jar -allow-phantom-refs -android-jars /path/to/android-sdk-linux_x86/platforms -src-prec apk -process
```

SootAssembleJimpleToDex.sh

```
java -jar soot.jar -allow-phantom-refs -android-jars /path/to/android-sdk-linux_x86/platforms -src-prec jimple -proc
mv sootOutput/classes.dex .
```

Example

Example of reverse-engineering and modding APK with smali:

- in APK-Multi-Tool-Linux working dir:
  - Drop myapp.apk in place-apk-here-for-modding/
  - ./script.sh (and leave it always open in a separate window)
  - 9 decompile / 1 select myapp.apk / 9 decompile
  - ./getstrings
  - Copy apk to dex2jar working dir
  - Copy getstring-sed to jad working dir
- in dex2jar working dir:
  - ./d2j-dex2jar.sh myapp.apk
  - Copy myapp-dex2jar.jar to jad working dir (and/or jd-gui)
- in jad working dir:
  - ./unjar myapp-dex2jar.jar
  - ./setstrings.sh myapp-dex2jar
  - Analyse .jad file and understand what to modify
- in jd-gui working dir:
  - As alternative analysis can also be done with jd-gui directly on .jar file
- in APK-Multi-Tool-Linux working dir:

- In working/ find corresponding .smali file and modify it
- (in script.sh windows) 13 compile/sign/install

Lim Electronics' APK Decompiler (http://limelect.com/downloads/apk-shell-decompiler/)

A GUI wrapper for apktool.jar, dex2jar.bat and jad.exe on Windows.

Example 2

Example of reverse-engineering and modding APK with Soot / jimple

- in APK-Multi-Tool-Linux working dir:
  - Drop myapp.apk in place-apk-here-for-modding/
  - ./script.sh (and leave it always open in a separate window)
  - 1 extract apk
  - Copy apk to soot working dir
- in soot working dir:
  - ./SootDisassembleApkToJimple.sh myapp.apk
  - Analyse and modify sootoutput/*.jimple files
  - ./SootAssembleJimpleToDex.sh
  - Copy classes.dex to overwrite APK-Multi-Tool-Linux/out/classes.dex
- in APK-Multi-Tool-Linux working dir (in script.sh windows)
  - 3 zip apk / 2 regular app
  - 4 sign app
  - adb install place-apk-here-for-modding/repackaged-signed.apk

Dare (http://siis.cse.psu.edu/dare/index.html)

Dalvik Retargeting, a tool for converting Android’s .dex format to Java’s .class format

Retargeted .class:

```
./dare -d output_dir -e myapp.apk
```

Optimized retargeted .class: (using Soot, slow!)

```
./dare -o -d output_dir -e myapp.apk
```

Decompiled optimized retargeted .class: (using Soot, very slow!)

```
./dare -c -d output_dir -e myapp.apk
```

APKInspector (https://github.com/honeynet/apkinspector/)

The goal of this project is to help analysts and reverse engineers to visualize compiled Android packages and their corresponding DEX code. APKInspector provides both analysis functions and graphic features for the users to gain deep insight into the malicious apps
Still beta and inactive for a year.
GUI around other tools

Androguard (http://code.google.com/p/androguard/)

Reverse engineering, Malware analysis of Android applications … and more !

- Installation (https://code.google.com/p/androguard/wiki/Installation)
- Usage (https://code.google.com/p/androguard/wiki/Usage)
- Reverse Engineering Tutorial of Android Apps (https://code.google.com/p/androguard/wiki/RE)

Seems to be able to tackle also dynamically loaded code, native code, reflection code

Dexdump (http://code.google.com/p/dex-decomplier/)

Java .dex file format decompiler
Inactive since 2009

FlowDroid (http://sseblog.ec-spride.de/tools/flowdroid/)

FlowDroid is a context-, flow-, field-, object-sensitive and lifecycle-aware static taint analysis tool for Android applications

Mobile Sandbox (http://www.mobile-sandbox.com)

Provides online static analysis of malware images.

JEB Decompiler (https://www.pnfsoftware.com/index)

Commercial ($1000)
Decompile Android apps and obfuscated Dalvik bytecode

Misc Android Static

Online decompilation at http://www.decompileandroid.com/ (using dex2jar, jad, apktool, zip/unzip)

Simplify: Generic Android Deobfuscator (https://github.com/CalebFenton/simplify)

Java

JAD (http://www.varaneckas.com/jad)

Java Decompiler
To use on a jar (from dex2jar):

```
#!/bin/bash
JAD=$(pwd)/jad
ODIR=${1%.jar}
if [ "$ODIR" == "$1" ]; then
    echo "Error: expecting a file ending with .jar"
    exit 1
fi
7z x -o${ODIR} $1
for d in $(find ${ODIR}/com -type d); do
  echo Entering $d
  cd $d
  # Clean Android stuffs
  rm *\$*.class
  for c in *.class; do
    $JAD $c
    # Want to keep the .class or not?
    rm $c
  done
  cd -
done
```

```
./unjar myapp-dex2jar.jar
```

jadretro (http://jadretro.sourceforge.net/)

Helps converting Java 1.4, Java 1.5 or later classes so JAD gives better results

JadAlign (http://jad-align.sourceforge.net/)

Aligns java-files, which are decompiled by jad

```
java -jar JadHelper-0.0.1.jar myfile.java
```

No much effect on jad from dex

Jd-gui (http://jd.benow.ca/)

JD-GUI is a standalone graphical utility that displays Java source codes of “.class” files

binary-refactor (https://github.com/argan/binary-refactor)

Helper to manual de-obfuscate obfuscated jars

- rename class/packages in a jar
- match a jarjar-ed & obfuscated jar with a known jar,to find the 'same' classes
- bytecode dump(asm)
- class dependency graph

dirtyJOE (http://dirty-joe.com/)

Java Overall Editor is a complex editor and viewer for compiled java binaries (.class files)

PJB

- http://blog.soat.fr/tag/jvmhardcore/ (fr)
- http://www.kaourantin.net/source/pbjtools/dpbj.cpp

Online

- http://www.javadecompilers.com/

Procyon

- https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler

JEB2

- https://www.pnfsoftware.com/

ELF

```
man elf
```

readelf

```
readelf -a -g -t --dyn-syms -W mybin
```

elfedit

objdump

```
objdump -C -g -F -x -T --special-syms mybin
objdump -d -l -r -R -S mybin
objdump -D -l -r -R -S mybin
```

nm

```
nm -a -C -S -s --special-syms mybin
```

ldd

Shared library dependencies:

```
ldd -v mybin
```

PE

Pefile (https://code.google.com/p/pefile/)

A Python module to read and work with PE (Portable Executable) files, see usage examples (https://code.google.com/p/pefile/wiki/UsageExamples)

```
#!/usr/bin/env python
import sys, pefile
pe = pefile.PE(sys.argv[1])
pe.dump_info()
open('out.txt', 'w').write(pe.dump_info())
```

Can run under Linux

PEiD

Can run with Wine

PETools (http://pe-tools.sourceforge.net/)

Can run with Wine

Get-PEHeader (http://www.exploit-monday.com/2012/07/get-peheader.html)

A Scriptable In-memory and On-disk PE Parsing Utility

Resource Hacker (http://www.angusj.com/resourcehacker/)

Can run with Wine

Dependency Walker (http://www.dependencywalker.com)

Can run with Wine

PEview (http://wjradburn.com/software/)

Can run with Wine

DLL Export Viewer (http://www.nirsoft.net)

Can run with Wine
Under Wine, require absolute path to DLL so: click on gears, "load functions from the following DLL file", Browse

PEBrowse Pro (http://www.smidgeonsoft.prohosting.com/pebrowse-pro-file-viewer.html)

Can run with Wine

Explorer Suite (http://www.ntcore.com/exsuite.php)

- CFF Explorer: Allows also to modify a PE
- Signature Explorer
- PE Detective
- Task Explorer (32 & 64)

PE Insider (http://icerbero.com/peinsider/)

Windows Object Explorer 64-bit (https://github.com/hfiref0x/WinObjEx64)

Static protections

Packers

- http://www.openrce.org/reference_library/packer_database
- http://www.reverse-engineering.info/documents/33.html
- https://corkami.googlecode.com/files/packers.pdf
- UPX (http://upx.sourceforge.net/)

```
upx -d myfile
```

- http://www.woodmann.com/crackz/Packers.htm
- Crinkler (http://www.crinkler.net/) : some insane PE packing tool coming from the demoscene world.
- midgetpack (https://github.com/arisada/midgetpack) Midgetpack is a binary packer for ELF binaries. The curve25519 is the real advantage of midgetpack. In this mode, you do not provide any password or key. Instead, a key file is generated at packing time. This key file must be used every time you wish to use the binary. When you start the binary, it will give a challenge and expect a response.
- android-unpacker (https://github.com/strazzere/android-unpacker)

Visualization

- ..cantor.dust.. (https://sites.google.com/site/xxcantorxdustxx/) (Recon 2013 (http://recon.cx/2013/schedule/events/20.html) , BH 2012 demo release (https://media.blackhat.com/bh-us-12/Arsenal/Domas/_cantor.dust_.7z.zip)), Windows only?
- binglide (https://github.com/wapiflapi/binglide) , Python3
- Binwalk (http://binwalk.org/) (binwalk --3D), Linux, OSX
- VisualBinary (https://github.com/Spl3en/VisualBinary) , Windows, Linux
- BinVis (https://code.google.com/p/binvis/) , C#
- Binvis.io (http://binvis.io/#/) , online
- Senseye (https://github.com/letoram/senseye/wiki) , Dynamic Visual Debugging / Reverse Engineering Toolsuite, Linux, à la Cantor Dust

Source code

- Imagix4D (http://www.imagix.com/products/source-code-analysis.html)
- SciTools Understand (https://scitools.com/)

Dynamic Analysis Tools

IDA Pro

Metasm

Metasm has debugging capabilities too.

Intel PIN tools

- Official page (http://software.intel.com/en-us/articles/pintool)
  - User guide (http://software.intel.com/sites/landingpage/pintool/docs/61206/Pin/html/)
- Windows, Linux, Mac OS X, Android
- x86-32, x86-64 (only Intel platforms obviously)
- binary instrumentation

The best way to think about Pin is as a "just in time" (JIT) compiler. The input to this compiler is not bytecode, however, but a regular executable. Pin intercepts the execution of the first instruction of the executable and generates ("compiles") new code for the straight line code sequence starting at this instruction. It then transfers control to the generated sequence. The generated code sequence is almost identical to the original one, but Pin ensures that it regains control when a branch exits the sequence. After regaining control, Pin generates more code for the branch target and continues execution. Pin makes this efficient by keeping all of the generated code in memory so it can be reused and directly branching from one sequence to another. In JIT mode, the only code ever executed is the generated code. The original code is only used for reference. When generating code, Pin gives the user an opportunity to inject their own code (instrumentation).

- Program Record/Replay Toolkit (https://software.intel.com/en-us/articles/program-recordreplay-toolkit)
- PIN++ (https://github.com/SEDS/PinPP) , see paper (pdf) (http://sebox.cs.iupui.edu/PDF/GPCE2014-PinPP.pdf)
- New attempt to bind with Python: @ancat & @1blankwall1 at Shmoocon2015: slides (https://raw.githubusercontent.com/blankwall/Python_Pin/master/talk.pdf) , github (https://github.com/blankwall/Python_Pin)

Tracers:

- BinTrace (https://bitbucket.org/mihaila/bintrace/wiki/Home)

DynamoRIO (http://www.dynamorio.org/)

DynamoRIO is a runtime code manipulation system that supports code transformations on any part of a program, while it executes. DynamoRIO exports an interface for building dynamic tools for a wide variety of uses: program analysis and understanding, profiling, instrumentation, optimization, translation, etc. Unlike many dynamic tool systems, DynamoRIO is not limited to insertion of callouts/trampolines and allows arbitrary modifications to application instructions via a powerful IA-32/AMD64 instruction manipulation library. DynamoRIO provides efficient, transparent, and comprehensive manipulation of unmodified applications running on stock operating systems (Windows or Linux) and commodity IA-32 and AMD64 hardware.

For ARM, see also DynamoRIO-ARM (https://github.com/sfrankl85/DynamoRIO-ARM) and (dead?) DynamoRIO-for-ARM (https://github.com/j616/DynamoRIO-for-ARM)

TEMU (http://bitblaze.cs.berkeley.edu/temu.html)

The BitBlaze infrastructure provides a component, called TEMU, for dynamic binary analysis. TEMU is built upon a whole-system emulator, QEMU, and provides the following functionality:

- Dynamic taint analysis. TEMU is able to perform whole-system dynamic taint analysis. Marking certain information sources (e.g., keystrokes, network inputs, reads for certain memory locations, and function call outputs) as tainted, TEMU keeps track of the tainted information propagating in the system. This feature also provides a plug-in environment for dynamic symbolic execution, in which symbolic values are marked as tainted, and concrete values as untainted.
- OS awareness. Information about OS-level abstractions like processes and files is important for many kinds of analysis. Using knowledge of the guest operating system (Windows XP or Linux), TEMU can determine what process and module is currently executing, what API calls have been invoked (with their arguments), and what disk locations belong to which files.
- In-depth behavioral analysis. TEMU is able to understand how an analyzed binary interacts with the environment, such as what API calls are invoked, and what outstanding memory locations are accessed. By marking the inputs as tainted (i.e., symbolic), TEMU provides insights about how outputs are formulated from inputs.

QIRA (https://code.google.com/p/qira/)

QEMU Interactive Runtime Analyser to do dynamic analysis as well as IDA does static analysis
Write-up example: ezhp (https://code.google.com/p/qira/wiki/ezhpQIRAwriteup)
QIRA now at https://github.com/BinaryAnalysisPlatform

Vdb/Vtrace (http://visi.kenshoto.com/viki/Vdb) / Vivisect (http://visi.kenshoto.com/viki/Vivisect)

- debugger, static analysis
- Windows, Linux, Android
- Intel, ARM

vtrace is a cross-platform process debugging API implemented in python, and vdb is a debugger which uses it
vivisect is a Python based static analysis and emulation framework

- vtrace script examples (https://github.com/pdasilva/vtrace_scripts)

S2 Dynamic tracer and decompiler for gdb (http://jolmos.blogspot.com.es/2014/09/s2-dynamic-tracer-and-decompiler-for-gdb.html)

Cryptoshark (https://github.com/frida/cryptoshark)

Self-optimizing cross-platform code tracer based on dynamic recompilation, powered by Frida and Capstone

Avatar (http://www.s3.eurecom.fr/tools/avatar/)

Avatar is an event-based arbitration framework that orchestrates the communication between an emulator and a target physical device. Avatar's goal is to enable complex dynamic analysis of embedded firmware in order to assist in a wide range of security-related activities including (but not limited to) reverse engineering, malware analysis, vulnerability discovery, vulnerability assessment, backtrace acquisition and root-cause analysis of known test cases.
ARM

Android

ADBI: Binary Instrumentation Framework for Android (http://mulliner.org/android/)

Slides here (http://mulliner.org/android/feed/binaryinstrumentationandroid_mulliner_summercon12.pdf)

- https://github.com/crmulliner/adbi

Dynamic Dalvik Instrumentation Framework for Android (http://mulliner.org/android/)

Slides here (http://mulliner.org/android/feed/mulliner_ddi_30c3.pdf)

- https://github.com/crmulliner/ddi

DroidScope (https://code.google.com/p/decaf-platform/)

DECAF(short for Dynamic Executable Code Analysis Framework) is a binary analysis platform based on QEMU. This is also the home of the DroidScope dynamic Android malware analysis platform. DroidScope is now an extension to DECAF
Slides here (https://www.usenix.org/sites/default/files/conference/protected-files/yan_usenixsecurity12_slides.pdf) and article here (https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final107.pdf)

DroidBox (https://code.google.com/p/droidbox/)

Android Application Sandbox

TaintDroid (http://appanalysis.org/)

Realtime Privacy Monitoring on Smartphones

Soot (http://www.sable.mcgill.ca/soot/)

Java, Dalvik (see here (http://www.bodden.de/2013/01/08/soot-android-instrumentation/) and here (http://www.abartel.net/dexpler/))

GameCIH

GameGuardian

Drozer (https://labs.mwrinfosecurity.com/tools/drozer/)

Comprehensive security and attack framework for Android
Interacts with Dalvik VM and explore applications attack surface (activities, content providers, services, etc).
Can also be used remotely à la Metasploit with exploits & payloads

AndBug (https://github.com/swdunlop/AndBug)

A Scriptable Debugger for Android's Dalvik Virtual Machine

Hooker (https://github.com/AndroidHooker/hooker)

Hooker is an opensource project for dynamic analysis of Android applications. This project provides various tools and applications that can be use to automaticaly intercept and modify any API calls made by a targeted application. It leverages Android Substrate framework to intercept these calls and aggregate all their contextual information (parameters, returned values, ...) in an elasticsearch database. A set of python scripts can be used to automatize the execution of an analysis in order to collect any API calls made by a set of applications.

Xposed (http://repo.xposed.info/module/de.robv.android.xposed.installer)

Changes app_process binary and hooks into all system or applications
Many modules
See also XDA forum (http://forum.xda-developers.com/xposed)

Cydia Substrate (http://www.cydiasubstrate.com)

Similar to Xposed but not via replacement of system components.
Hooks into Dalvik and native code

Misc Android Dynamic

- setpropex (https://docs.google.com/file/d/0B8LDObFOpzZqY2E1MTIyNzUtYTkzNS00MTUwLWJmODAtZTYzZGY2MDZmOTg1/edit) , as setprop but changes read-only properties by attaching to init via ptrace
- iSec Intent Sniffer (https://www.isecpartners.com/tools/mobile-security/intent-sniffer.aspx) and iSec Intent Fuzzer (https://www.isecpartners.com/tools/mobile-security/intent-fuzzer.aspx)
- On Tracking Information Flows through JNI in Android Applications (pdf) (http://www4.comp.polyu.edu.hk/~csxluo/NDroid.pdf) , A dynamic information flow tracing system for Android, sources here (https://github.com/0-14N/NDroid)
- ZjDroid (https://github.com/BaiduSecurityLabs/ZjDroid) Android app dynamic reverse tool based on Xposed framework (you'd better understand Chinese...)
- mkbreak (https://github.com/robertmillan/mkbreak) Generic exploit for master key vulnerability in Android
- Appie (https://manifestsecurity.com/appie/) Android Pentesting Portable Integrated Environment
- http://translate.wooyun.io/2015/06/17/android-logcat-security.html

iOS

iSpy (https://github.com/BishopFox/iSpy)

A reverse engineering framework for iOS

Java

Javasnoop (https://code.google.com/p/javasnoop/)

A tool that lets you intercept methods, alter data and otherwise hack Java applications running on your computer.

Flash

FlashHacker (https://github.com/ohjeongwook/FlashHacker)

ActionScript Bytecode instrumentation framework

ELF

ltrace/strace

Tracing library calls and system calls.
Getting a summary:

```
ltrace -f -S mybin 2>&1|grep '(.*)'|sed 's/(.*//'|sort|uniq -c
```

Getting more:

```
ltrace -f -i -S -n 4 -s 1024 mybin
```

ftrace (https://github.com/leviathansecurity/ftrace)

Tracing inner execution flow as well

Lib preloading

```
#define _GNU_SOURCE

#include <dlfcn.h>
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
#include <stdio.h>
#include <time.h>


// Kill nanosleep()
int nanosleep(const struct timespec *req, struct timespec *rem){
    printf("\n==== In our own nanosleep(), I dunnah want sleep\n");
    return 0;
}

// Kill usleep()
int usleep(useconds_t usec){
    printf("\n==== In our own usleep(), I dunnah want sleep\n");
    return 0;
}

// Fix time()
time_t time(time_t *t){
    printf("\n==== In our own time(), will return 1380120175\n");
    return 1380120175;
}

// Fix srand()
void srand(unsigned int seed){
    printf("\n==== In our own srand(), will do srand(0)\n");
    void (*original_srand)(unsigned int seed);
    original_srand = dlsym(RTLD_NEXT, "srand");
    unsigned int myseed = 0;
    return (*original_srand)(myseed);
}

#if 0
// Kill rand()
int rand(void){
    printf("\n==== In our own rand(), will return 0\n");
    return 0;
}
#else
// Intercept rand()
int rand(void){
    int (*original_rand)(void);
    original_rand = dlsym(RTLD_NEXT, "rand");
    int r = (*original_rand)();
    printf("\n==== In our own rand(), will return %04X\n", r);
    return r;
}
#endif
```

```
gcc -fPIC -shared -Wl,-soname,patch -o patch.so patch.c -ldl
export LD_PRELOAD=patch.so
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

ldpreloadhook (https://github.com/poliva/ldpreloadhook)

a quick open/close/ioctl/read/write/free symbol hooker

injectso (http://stealth.openwall.net/local/)

- x86-32, x86-64, ARM (since v0.52)

scanmem (https://code.google.com/p/scanmem/)

scanmem is a simple interactive debugging utility for linux, used to locate the address of a variable in an executing process. This can be used for the analysis or modification of a hostile process on a compromised machine, reverse engineering, or as a "pokefinder" to cheat at video games.

- Linux/Android
- with a GUI since v0.13: GameConqueror

GDB

Enable binary writing, here changing a conditional jump to unconditional jump:

```
gdb -write -silent --args mycode 1 2 3
...
(gdb) set {unsigned char}0x400123 = 0xeb
(gdb) disassemble 0x400123 0x400124
0x400123 jmp 0x...
```

or injecting NOPs:

```
(gdb) set {unsigned char}0x400123 = 0x90
```

Extensions

Stephen Bradshaw ha swritten some extensions to have more useful gdb info when debugging stripped binaries, closer to what you get with OllyDbg. See:

- http://www.thegreycorner.com/2013/10/my-python-gdb-extensions.html
- http://www.thegreycorner.com/2014/03/gdb-extensions-110.html

GUI

- Voltron (https://github.com/snarez/voltron) is an unobtrusive debugger UI for hackers
- SchemDBG (https://github.com/hexgolems/schem) is a backend agnostic debugger frontend that focuses on debugging binaries without access to the source code

ERESI (http://www.eresi-project.org/)

The ERESI Reverse Engineering Software Interface is a multi-architecture binary analysis framework with a domain-specific language tailored to reverse engineering and program manipulation.

PE

Process Monitor (http://technet.microsoft.com/en-us/sysinternals/bb896645)

Process Explorer (http://technet.microsoft.com/en-us/sysinternals/bb896653)

RegShot (http://sourceforge.net/projects/regshot/)

Computes diff between two registry snapshots

HeapMemView (http://www.nirsoft.net)

OllyDbg

PE32-only dynamic disassembler and debugger: http://ollydbg.de/.
Version 1.1 is historically widespread, version 2.0 is re-written from scratch, still considered as beta by some.
Support software and hardware breakpoint, binary patching and repacking, symbol analysis, advanced instruction pattern search, trace with conditional breaking, etc.

ImmDbg

There is also a patched version of OllyDbg with advanced python scripting ability called Immunity Debugger:
http://www.immunityinc.com/products-immdbg.shtml
Expect some OllyDbg plugins to not work properly with ImmDbg.
Plugins:

- Mona (http://redmine.corelan.be/projects/mona) , a debugger plugin / Exploit Development Swiss Army Knife

WinAppDbg (https://github.com/MarioVilas/winappdbg)

The WinAppDbg python module allows developers to quickly code instrumentation scripts in Python under a Windows environment.

Tracer.py (https://brundlelab.wordpress.com/2012/08/19/small-and-cute-execution-tracer/)

Based on WinAppDbg, finds interesting bits in trace by dichotomy signal/noise

- run first time and try everything but not the interesting stuff -> use noise option
- then run again and try interesting stuff -> use signal option

WTFDLL.py (https://github.com/carlosgprado/Python-to-the-rescue/blob/master/WTFDLL.py)

Find libraries loaded at runtime and the functions called

x64_dbg (http://x64dbg.com/#start)

An open-source x64/x32 debugger for windows.

Cuckoo Sandboxing (http://www.cuckoosandbox.org/)

Currently only supporting Windows binaries.
Cuckoo Sandbox is a malware analysis system. You can throw any suspicious file at it and in a matter of seconds Cuckoo will provide you back some detailed results outlining what such file did when executed inside an isolated environment. Cuckoo generates a handful of different raw data which include:

- *Native functions and Windows API calls traces*
- *Copies of files created and deleted from the filesystem*
- *Dump of the memory of the selected process*
- *Full memory dump of the analysis machine*
- *Screenshots of the desktop during the execution of the malware analysis*
- *Network dump generated by the machine used for the analysis*

Reven (http://www.tetrane.com/)

- technology overview (http://lifeat.tetrane.com/2014/11/reven-technology-overview.html)

Unknown price

Protections

- http://www.openrce.org/reference_library/anti_reversing
- https://corkami.googlecode.com/files/cm.pdf
- ptrace e.g. on iOS (http://www.coredump.gr/articles/ios-anti-debugging-protections-part-1/)
- sysctl, e.g. on iOS (http://www.coredump.gr/articles/ios-anti-debugging-protections-part-2/)
- ELF obfuscation (http://h4des.org/blog/index.php?/archives/346-ELF-obfuscation-let-analysis-tools-show-wrong-external-symbol-calls.html) let analysis tools show wrong external symbol calls
- Obfuscator-LLVM (https://github.com/obfuscator-llvm/obfuscator/wiki)
  - Deobfuscation: recovering an OLLVM-protected program (http://blog.quarkslab.com/deobfuscation-recovering-an-ollvm-protected-program.html)
- x86obf code virtualizer (http://chaplja.blogspot.in/2015/02/x86obf-code-virtualizer-released-for.html) , 32bit PE files (EXE and DLLs), source code (http://chaplja.blogspot.in/2015/02/x86obf-source-code.html)

Patching

- IDA Pro DB patching
- IDA Pro & Fentanyl
- Radare

Fuzzing

- AFL (http://lcamtuf.coredump.cx/afl/)
  - Introduction to Fuzzing in Python with AFL (https://alexgaynor.net/2015/apr/13/introduction-to-fuzzing-in-python-with-afl/)

Z3

- Using Z3 theorem prover to prove equivalence of some bizarre alternative to XOR operation (http://blog.yurichev.com/node/86)
- https://github.com/0vercl0k/z3-playground/blob/master/hackingweek-reverse400_z3.py

Exploitation

Tools

- Ropper (http://scoding.de/ropper/) , rop gadget finder and binary information tool, based on Capstone
- ROPgadget (http://shell-storm.org/project/ROPgadget/) , supports ELF/PE/Mach-O format on x86, x64, ARM, PowerPC, SPARC and MIPS architectures
- ROPshell (http://ropshell.com/) , online, supports ELF/PE/Mach-O format on x86, x64, ARM
- pwntools (https://github.com/pwnies/pwntools)
- PEDA (https://code.google.com/p/peda/) : Python Exploit Development Assistance for GDB (x86/x64)
- GEF (https://github.com/hugsy/re-stuff/blob/master/gef.py) : GDB enhanced features - multi-arch

(x86/x64/mips/ppc/arm)
- Hexcellents notes (http://koala.cs.pub.ro/hexcellents/wiki/kb/exploiting/home)
- ROP on ARM (pdf) (https://dl.dropboxusercontent.com/u/2595211/ROP_ARMEXP.pdf) by Xipiter / dontstuffbeansupyournose
- Framing Signals a return to portable shellcode: article (http://www.ieee-security.org/TC/SP2014/papers/FramingSignals-AReturntoPortableShellcode.pdf) , slides (https://minemu.org/srop_slides_sp2014.pdf)
- BARF (https://github.com/programa-stic/barf-project) : A multiplatform open source Binary Analysis and Reverse engineering Framework

Mitigation techniques

Some are taken from excellent Android Hacker's Handbook

Hardening the Heap

Hardened version of dlmalloc? Alternatives?

This can be done with LD_PRELOAD, e.g. with tcmalloc (http://goog-perftools.sourceforge.net/doc/tcmalloc.html)

```
LD_PRELOAD="/usr/lib/libtcmalloc.so"
```

Protecting against Integer Overflows

- Protected calloc?
- Hardened library for safe integer operations: safe_iop (https://code.google.com/p/safe-iop/)

Preventing Data Execution

Set stack (and heap) as non-executable.
Kernel marks stack as executable unless it finds a GNU_STACK program header without executable flag set.
To insert non-exec statement:

```
flag: -znoexecstack
```

To test:

```
/usr/sbin/execstack -q myprog
```

- "?": myprog has no GNU_STACK -> stack is executable
- "-": stack non-executable
- "X": stack executable

Same:

```
readelf -a myprog|grep -A1 GNU_STACK
```

- present? with RW or RWE?

Same:

```
cat /proc/123/maps|grep -E '(stack|heap)'
```

- rw or rwx?

To modify existing bin:

```
/usr/sbin/execstack -s myprog # set executable stack
/usr/sbin/execstack -c myprog # clear
```

Max nr of process IDs

```
/sbin/sysctl kernel.pid_max
```

Traditionally 32768

```
/sbin/sysctl -w kernel.pid_max=4194303
```

ptrace

```
/sbin/sysctl kernel.yama.ptrace_scope
```

To allow ptrace:

```
/sbin/sysctl -w kernel.yama.ptrace_scope=0
```

Address Space Layout Randomization

Bin needs to be compiled position-independent:

```
CFLAGS: -fPIE
LDFLAGS: -pie
```

To test:

```
readelf -h myprog | grep Type:
```

- DYN? position-independent
- EXEC? Not position-independent

or

```
readelf -d myprog | grep TEXTREL
```

Global settings

```
/sbin/sysctl kernel.randomize_va_space
/sbin/sysctl -w kernel.randomize_va_space=2
```

- 0 – No randomization. Everything is static.
- 1 – Conservative randomization. Shared libraries, stack, mmap(), VDSO and heap are randomized.
- 2 – Full randomization. In addition to elements listed in the previous point, memory managed through brk() is also randomized.

To disable it locally (in a bash and its children)

```
setarch `uname -m` -R /bin/bash
```

On 32 bit systems "ulimit -s unlimited" disables the randomization of the mmap()-ing

Protecting the Stack

ProPolice stack protection is enabled by using

```
flags: -fstack-protector
```

Format String Protections

Enabled by using

```
flags: -Wformat-security -Werror=format-security
```

Beware compiler cannot detect all corner cases
See also _FORTIFY_SOURCE=2 for runtime protection against %n

Read-Only Relocations

Partial relro enabled by using

```
flags: -Wl,-z,relro
```

To test:

```
readelf -h myprog|grep RELRO
```

- GNU_RELRO? Partial relro protection present

Full relro enabled by using

```
flags: -Wl,-z,relro -Wl,-z,now
```

To test:

```
readelf -d myprog|grep NOW
```

- flags NOW? Full relro protection present

Access Control Mechanisms

SELinux

(anti-)anti-debug

- The "Ultimate" Anti-Debugging Reference (http://pferrie.host22.com/papers/antidebug.pdf)
- TitanHide (http://mrexodia.cf/reversing/2015/02/05/TitanHide/) Opensource ring0 Windows x64 anti-anti-debug driver

Static source code analysis and fortifying

cppcheck

```
cppcheck --quiet --check-config .
cppcheck --xml --xml-version=2 --std=posix --std=c99 \
  --enable=style,performance,portability,information,unusedFunction \
  -I include --force --inconclusive .
```

flawfinder

```
flawfinder --quiet --dataonly --singleline --followdotdir .|sort -k 2 -r|less
flawfinder --immediate --dataonly --inputs --followdotdir .
```

gcc/clang

```
export CFLAGS="-Wall -g -O2 -Wextra -pipe -funsigned-char -fstrict-aliasing -Wchar-subscripts -Wundef -Wshadow -Wcast-align -Wwri
```

Clang:

```
export CFLAGS="-Wunreachable-code"
export CFLAGS="$CFLAGS -fno-omit-frame-pointer -D_FORTIFY_SOURCE=2 -fstack-protector"
export LDFLAGS="$LDFLAGS -fsanitizer=address -fno-omit-frame-pointer -D_FORTIFY_SOURCE=2 -fstack-protector"
```

Other resources

Books

- The IDA Pro Book, 2nd Edition by Chris Eagle (http://shop.oreilly.com/product/9781593272890.do)
- Reverse Engineering Code with IDA Pro by Dan Kaminsky et al (http://shop.oreilly.com/product/9781597492379.do)
- Practical Malware Analysis by Michael Sikorski (http://shop.oreilly.com/product/9781593272906.do)
- Reversing: Secrets of Reverse Engineering by Eldad Eilam (http://www.amazon.com/Reversing-Secrets-Engineering-Eldad-Eilam/dp/0764574817)
- Crackproof Your Software by Pavol Cerven (http://shop.oreilly.com/product/9781886411791.do)
- Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection (http://www.amazon.com/Surreptitious-Software-Obfuscation-Watermarking-Tamperproofing/dp/0321549252)
- Wikibooks Subject:Software_reverse_engineering (https://en.wikibooks.org/wiki/Subject:Software_reverse_engineering)
  - x86 Disassembly (https://en.wikibooks.org/wiki/X86_Disassembly) , x86 Assembly (https://en.wikibooks.org/wiki/X86_assembly)
  - Reverse Engineering (https://en.wikibooks.org/wiki/Reverse_Engineering)

- Reverse Engineering for Beginners (http://yurichev.com/RE-book.html) , free, by @yurichev

ELF

- Linux x86 Reverse Engineering (pdf) (http://www.exploit-db.com/wp-content/themes/exploit/docs/33429.pdf) Shellcode Disassembling and XOR decryption
- Introduction to Reverse Engineering Software in Linux (http://www.ouah.org/RevEng/)

ARM

- ARM instruction set (pdf) (http://simplemachines.it/doc/arm_inst.pdf) , slides

Android

- Big lists of tools
  - http://www.nyxbone.com/malware/android_tools.html
  - http://wiki.secmobi.com/tools:android_reversing_analysis
  - http://wiki.secmobi.com/tools:android_dynamic_analysis
  - https://androidsecuritywiki.com/
  - http://www.droidsec.org/wiki/
- Techbliss (http://techbliss.org/) forum: Android, IDA,...
- How To Decode ProGuard’s Obfuscated Code From Stack Trace (http://blog.simplyadvanced.net/android-how-to-decode-proguards-obfuscated-code-from-stack-trace/)
- Live Memory Forensics on Android with Volatility (pdf) (https://www1.informatik.uni-erlangen.de/filepool/publications/Live_Memory_Forensics_on_Android_with_Volatility.pdf)
- Dalvik and ART Part 1 (DEX) (http://newandroidbook.com/files/Andevcon-DEX.pdf) & Part 2 (ART) (http://newandroidbook.com/files/Andevcon-ART.pdf) , slides
- Reverse engineering & Rebuilding 3rd party, closed, binary Android apps (APK’s) using APKtool (http://s4mpl3d.me/?p=162) , tutorial

iOS

- Misc RE resources for OSX & iOS (http://samdmarshall.com/re.html)
- Pentesting iOS applications (http://www.slideshare.net/jasonhaddix/pentesting-ios-applications)
- iOS applications reverse engineering (pdf) (http://media.hacking-lab.com/scs3/scs3_pdf/SCS3_2011_Bachmann.pdf) : slides

Intel

- corkami (https://code.google.com/p/corkami/) , x86 oddities (https://code.google.com/p/corkami/wiki/x86oddities?show=content&wl=en) etc
- Intel Intrinsics Guide (https://software.intel.com/sites/landingpage/IntrinsicsGuide/)
- Intel® 64 and IA-32 Architectures Software Developer Manuals (http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html)

Misc Resources

- Reverse-Engineering on StackExchange (http://reverseengineering.stackexchange.com/)
- OpenRCE (http://www.openrce.org/)
- Hex Blog (http://www.hexblog.com/)
- http://www.reverse-engineering.info
- Automating RE with Python (slides) (http://brundlelab.files.wordpress.com/2013/09/automating-re-with-python.pdf) by Carlos Prado
- Literature review (https://github.com/REMath/literature_review) , big list of tools
- Python Arsenal for Reverse Engineering (http://pythonarsenal.erpscan.com/) , big list of tools
- OpenSecurityTraining (http://opensecuritytraining.info/Training.html) , x86, x86-64, ARM, RE, malware RE,... tutorials, slides, videos
- Reverse Engineering Tutorials (http://thelegendofrandom.com/blog/sample-page)
- http://deroko.phearless.org/rce.html
- http://www.woodmann.com/collaborative/tools/index.php/Category:RCE_Tools

Going Maths

- http://www.reddit.com/r/remath
- http://www.reddit.com/r/ReverseEngineering/comments/smf4u/reverser_wanting_to_develop_mathematically/
- A Bibliography of Papers on Symbolic Execution Technique and its Applications (https://sites.google.com/site/symexbib/)

SDR

- Signal Identification Guide (http://www.sigidwiki.com/wiki/Signal_Identification_Guide)
- ARTEMIS (http://www.rtl-sdr.com/artemis-free-signal-identification-software/) signal identification software (Windows)