binutils Binary inspection tools

START HERE

`file <bin>`
What kind of binary

`strings -n 8 <bin>`
Printable text

`readelf -h <bin>`
ELF header

`ldd <bin>`
Shared libraries

`nm -C <bin>`
List symbols

`addr2line -e <bin> -f <addr>`
Address to source line

`objdump -d -C <bin>`
Disassemble code

`size <bin>`
Text, data, bss sizes

Source cc -g -c → `compiles to` → **Object .o** nm, objdump → `ar / ld` → **Archive .a / ELF** readelf, size → `loaded as` → **Process** ldd, addr2line

CONCEPTS

Most tools read a file and never run it, so they are safe on untrusted binaries. ldd is the exception: it may execute the file.

Add -C to demangle C++ names; readelf, objdump and nm all accept it (readelf -C only for symbols).

Stripped binaries have no .symtab: use nm -D or readelf --dyn-syms for the dynamic symbols that remain.

Source lines and local variable names need a build with -g; without it addr2line only knows functions, if that.

SYMBOLS (NM)

`nm -C <file>`
List symbols, demangled

`nm -D <file>`
Dynamic symbols of a shared lib

`nm -u <file>`
Undefined symbols it needs

`nm --defined-only <file>`
Only symbols it provides

`nm -S --size-sort -C <file>`
Biggest symbols last, with size

`nm -n <file>`
Sort by address

`nm -A <file>...`
Prefix each line with file name
Handy with shell globs: find which .o or .a member defines a symbol.

`nm -g <file>`
External (global) symbols only

`nm -l <file>`
Add source file and line

`nm --print-armap <lib.a>`
Archive index: symbol to member

nm type letters

Letter	Meaning
`T / t`	code, global / local
`D / d`	initialised data
`B / b`	zero-filled bss
`R / r`	read-only data
`U`	undefined, resolved later
`W / w`	weak symbol

ELF STRUCTURE (READELF)

`readelf -h <elf>`
ELF header: type, arch, entry

ELF STRUCTURE (READELF) (cont.)

`readelf -S <elf>`
Section headers

`readelf -l <elf>`
Program headers (segments)

`readelf -d <elf>`
Dynamic section: NEEDED, RPATH

`readelf -s -W <elf>`
Symbol tables, wide lines

`readelf --dyn-syms <elf>`
Dynamic symbols only

`readelf -r <elf>`
Relocations

`readelf -n <elf>`
Notes such as the build ID

`readelf -p .comment <elf>`
Dump a section as strings
Shows the compiler that built the file.

`readelf -x .rodata <elf>`
Hex dump of a section

`readelf --debug-dump=info <elf>`
DWARF debug info

`readelf -a <elf>`
Everything at once

`readelf -W <elf>`
Do not wrap wide output

DISASSEMBLY (OBJDUMP)

`objdump -d <file>`
Disassemble code sections

`objdump -d -C <file>`
Disassemble, demangled names

`objdump -d -M intel <file>`
Intel syntax instead of AT&T

`objdump -D <file>`
Disassemble all sections, even data

`objdump -S <file>`
Interleave source (needs -g)

`objdump -d --no-show-raw-insn <file>`
Hide the opcode bytes

`objdump -d --disassemble=<sym> <file>`
Only one function
Older releases lack the =<sym> form; use -d and grep -A there.

`objdump -d --start-address=<a> --stop-address=<b> <file>`
Only an address range

`objdump -h <file>`
Section headers with sizes

DISASSEMBLY (OBJDUMP) (cont.)

`objdump -t <file>`
Symbol table

`objdump -T <file>`
Dynamic symbol table

`objdump -s -j <sect> <file>`
Hex dump of one section

`objdump -p <file>`
Private headers, NEEDED libs

`objdump -r -d <obj.o>`
Show relocations inside code

STRINGS AND SIZES

`strings <file>`
Printable strings, 4+ chars

`strings -n <len> <file>`
Set minimum string length

`strings -a <file>`
Scan the whole file (default)
Default since 2.14; -d scans only data sections.

`strings -t x <file>`
Prefix each with hex offset

`size <file>`
Text, data, bss totals

`size -A <file>`
Per-section sizes

`c++filt <name>`
Demangle a C++ name

ADDRESSES AND LIBRARIES

`addr2line -e <elf> <addr>`
Address to file:line

`addr2line -e <elf> -f -C -i <addr>`
Add function, demangle, inlines

`addr2line -e <lib.so> -a <off>`
Print address, for shared lib offsets
PIE and .so crashes log runtime addresses: subtract the load base first.

`ldd <bin>`
Shared libs it needs

`readelf -d <bin> | grep NEEDED`
Direct dependencies, no execution
Safe alternative to ldd on untrusted files.

`objdump -p <bin> | grep -E 'NEEDED|RPATH|RUNPATH'`
Libs and search paths

`file <bin>`
Type, arch, stripped or not

STRIP, COPY, ARCHIVES

`strip <file>`
Remove all symbols and debug info

`strip --strip-debug <file>`
Keep symbols, drop debug info

`strip -o <out> <file>`
Write result to another file

`objcopy --only-keep-debug <elf> <elf>.dbg`
Split debug info into a file

`objcopy --add-gnu-debuglink=<elf>.dbg <elf>`
Point stripped binary at it

`objcopy -O binary <elf> <out.bin>`
Raw image of the sections

`ar t <lib.a>`
List archive members

`ar rcs <lib.a> <obj.o>...`
Create or update static lib

`ar x <lib.a> [<member>]`
Extract members

`ar tv <lib.a>`
Members with size and date

RECIPES

Which library defines a symbol?

```
# search every static lib in the tree
nm -A -C --defined-only $(find . -name '*.a') | grep ' T .*parse_args'
# or dynamic: shared libs exporting it
for f in /usr/lib/x86_64-linux-gnu/*.so*; do nm -D --defined-only "$f" 2>/dev/null | grep -q ' T malloc$' && echo "$f"; done
```

Resolve a crash address

```
# from a crash log: binary+0x1a2b3
addr2line -e ./app -f -C -i 0x1a2b3
# find the surrounding code
objdump -d -C -S --start-address=0x1a280 --stop-address=0x1a2c0 ./app
```

Split debug info before shipping

```
objcopy --only-keep-debug app app.dbg
strip --strip-debug --strip-unneeded app
objcopy --add-gnu-debuglink=app.dbg app
readelf -n app | grep 'Build ID'
```

Find what bloats a binary

```
size -A app | sort -k2 -n -r | head
nm -S --size-sort -C -r app | head -20
# then see who pulls it in
readelf -d app | grep NEEDED
```