Cracking and reverse-engineering

Reverse engineering

- **Work out what programs do, and how**
- **Many applications**
  - Debugging is sort of “reverse-engineering” a bug
  - Analysing code flow to find “features” (Technical Minecraft)
  - Finding ways of disabling malware
  - Extracting the exploit used by worm to patch it
- **Very attractive skill to employers**

Binary Patching / Cracking

- **Modifying a program without the source code**
- **Many uses**
  - Extract useful parts without rewriting from scratch
  - Fix bugs on unmaintained software
  - Add features to software (i.e. modding games)
  - Can be used in software piracy
- **Be careful with terminology**
  - Cracking often means DRM bypass for piracy
  - “Binary patching” is the more employable term

DANGER WARNING PLEASE NO BREAK LAW!!!

- **DON’T crack DRM-infested programs**
  - Digital Rights Management
  - Bypassing DRM (even without sharing): illegal in the US, definitely dodgy in UK
  - Sharing cracked programs: violates copyright law everywhere
- **MAYBE examine legit programs you have rights to use**
  - Potential copyright issues if you write something that competes with owner (programs, manuals, etc)
  - Gives you useful practice
  - Useful insight into a PREVIOUSLY KNOWN co-operative and appreciative target
    - I’ve been in this situation before
- **DO examine your own programs**
  - Can teach you a lot about optimisation and low-level code
  - Teaches you what certain assembly blocks mean

Strings

- **Easy to recover plaintext data**
  - Grep for flags!
- **You can use `strings` to get strings from a binary**
  - Grep through the result, or just manually search
- **Anything that can display text works**
  - I have seen this solved with notepad

Strings demo

- [strings trivial]

Memory dump

- **Don’t always just store flag in plaintext**
- **Reverse engineering and patching is hard**
- **Grepping for flags is easy**
  - Set a breakpoint where the flag is stored in plaintext
  - Search for the flag, or print it if you know where it is

Memory dump demo

- **[cutter demo]**
- **[gdb demo]**
  - `search-pattern` with gef
  - `dump memory` with base gdb

Reverse engineering

- **Sometimes it’s a bit more complicated**
  - Data not stored in plaintext
  - You want an entire function, not just some data
- **For flags/keys**
  - It must check the data somehow
  - The secret is in the code

Reverse engineering rules

- **Rule 1: If you don’t understand it, it’s probably not important**
  - No-one cares what FYL2XP1 does
  - What the hell even is PHMINPOSUM?
  - Just look at jumps, calls and movs
- **Rule 2: Avoid looking at assembly whenever possible**
  - Assembly is a Lovecraftian aberration that slowly drives all who lay eyes upon it to insanity
  - Use decompilation where possible
  - Look at control flow graphs
- **Rule 3: focus on the important parts**
  - Modern software has tens of thousands of functions
  - Most of them are never used
  - Most of the rest do things you don’t care about
  - Only examine functions you directly need to understand

Reverse engineering tools

- **Real programmers use objdump -d**
  - Works for really small software that human brains can comprehend
  - Good luck with multi-million instruction binaries
- **Some people use Ghidra/IDA pro/binary ninja**
- **I find Cutter the most useful**
  - Supports decompilation
  - Supports binary patching
  - Experimental support for debugging (a bit rubbish)
  - A bit dodgy and crashes occasionally
  - Looks cool

Reverse engineering demo

- [cutter ez]

Binary patching – extracting information

- **Find the thing you want**
- **Find the things before it stopping you**
- **Disable them**
  - NOP: replace the instruction with no-ops
  - Reverse jump: invert the condition
  - Conditional → unconditional jumps: don’t check the condition

Binary patching - demo

- [demo → print flag]
- [ez → say win]

Advanced binary patching

- **Sometimes you don’t have everything**
  - Program only loads flag in chunks
- **If the program checks character by character, you can easily brute force**
  - Theoretically timing attacks work (but take ages!)
  - Patching is easier!
- **Standard approach: get the program to exit with the index of first incorrect character**

Advanced binary patching - demo

- [cutter ez]

Reversing checkers

- **In general you can’t work out what input a program accepts**
  - Literally a restatement of the halting problem
- **For easy things, we can do it by hand**
- **For harder things, we use software (usually Z3)**
- **Z3 can be controlled by most languages, but generally people use Python**
- **Won’t give full docs of Z3 here (google it!)**

Reversing checkers - demo

- [cutter harder]
- [nano harder-hax.py]
- [python3 harder-hax.py]

PLEASE DO CTF CHALLENGES :(((