Protecting binaries

Andrew Griffiths
andrewg@felinemenace.org

Introduction

- This presentation is meant to be useful for people of all skill levels. Hopefully everyone will get something out of this presentation.
- This talk focuses on strategies, and mindsets, not products.
- Technical details will mainly refer to Linux unless otherwise specified, although the concepts are portable to other operating systems.

Defence in depth

- Determining the threat model / what problems you're trying to solve
  - Casual copying
  - Determining who leaked a copy
  - Determined crackers
- Determining what measures will be suitable to avoid problems, and be feasible to implement
  - Code obfuscation
  - Watermarking
  - Licensing types

Defence in depth (cont)

- Defence requires you to think like an attacker, and how to best defend them from offence.
- Offensive measures?
  - Chang Yu said: "Knowing the enemy enables you to take the offensive, knowing yourself enables you to stand on the defensive." He adds: "Attack is the secret of defense; defense is the planning of an attack."
- As opposed to just displaying a message when something has gone bad, wouldn't it be better to mislead an attacker and waste some of their time/resources?

Defence in depth (cont)

- Standard implementations
  - Can usually be analysed separately
  - Lends itself to individual pieces being analysed, without impacting the rest of the binary.

- Idealistic defence in depth for binaries
  - When pieces are removed, it impacts the correct operation of other parts of the binary.
  - Layers are tightly integrated so that everything must be considered at once.
  - Assumes layers will be broken.

Watermarking

- Why watermark?
  - Watermarking does not prevent against fraud.
- Fragile vs Robust watermarks
- Visible vs Invisible watermarks
- Watermarking values
  - Counter
  - Code constructs / code ordering
  - Data initialisation values
- Tamperproofing?

Obfuscation

- Source code
- Assembly level
  - Junk code?
    - Not unlike what viruses have contained (f.e Junkcomp)
    - Not really applicable in this case. (Preventing signatures / on access detection)
  - Various aspects to obfuscation
    - Code layout
    - Data obfuscation
    - Control obfuscation
    - Preventative

Obfuscation (cont)

- Potency
  - How hard is it to analyse by a human
- Resiliency
  - Protection against:
    - Attackers effort to write the un-obfuscator
    - The program attempting to un-obfuscater
- Cost
  - What impact does implementing the measures involve?

Obfuscation (cont)

- Control flow obfuscation
  - Opaque conditionals
    - Used to mislead attackers, increase their workload, decrease what can be done automatically
    - Control flow
      - Absolutely trivial example: xor eax, eax ; jnz 0xaddy
      - Usually a lot more involved.
  - “rewriting” instruction context
    - Determine context of the registers
      - If they're important to that section of code you're analysing
      - The relationship to other pieces of nearby code

Obfuscation (cont)

        - Insert new instructions that modify the unimportant registers / memory locations
            - Usually there is just mov's, shifts, add / sub etc.
            - If you add a section in memory and load/store from it, the analysis tools now have to do a lot more work in order to remove those constructs, if its possible at all (depending on how its implemented). This is because the program now looks a lot more like a proper program behaviour.
        - Usually done before the program is compiled completely (ie, operates on object files).
        - Makes analysis by humans harder
    - Loops
- Data obfuscation
    - Converting static data to functions

Obfuscation (cont)

- Inserting more cross-references
- Inserting new functions into object orientated classes
- Adding new data to structures, loading / storing to it.
- Convert variables to classes, and have functions which do the various operators on it, such as multiplication, addition.

- Code layout obfuscation
  - Basic blocks
    - Re-ordering of instructions
    - Independent obfuscation
      - Blocks need to converge in the end

Obfuscation (cont)

- Register usage example
- mov eax, 1
- mov ebx, 2
- add eax, ebx
-
- mov eax, 1 and mov ebx, 2 would be the first basic block.
- add eax, ebx would be the second basic block.

– Code flow reduction
  - Switch tables

– Disadvantages to obfuscation
  - Performance impact
  - Time to implement

License scheme implementation

- Effort needed to implement
- If they are not meant to have certain pieces of code, don't compile it in. If they aren't meant to have some data, don't include it in the distribution.
- Combine the license aspect with the program aspect, so that attempting to break the license implementation has flow on effects to the correct operation with the program.
  - Use license information for logic and data choices.

License schemes (cont)

- Small checksums can be used to ensure people have not mistyped a license code without giving anything away about the correctness of the key.
- In general, do not sanity-check the license data, just use it for it's respective operations.
- Think like an attacker, find your weak spots, and patch them.

Virtual Machines

- What are they?
  - Java, .NET assembly (CLR)
  - Either:
    - Completely byte code driven
    - Or translates to CPU for native execution (JIT)
- Increases analysis time, as they have to fully understand what the VM is doing.
  - A lot of custom development may need to be done, depending what you want to implement.
- Disadvantages

Virtual Machines

- Only needs to be analysed once, so it loses its effectiveness.
  - Can be improved limitedly by randomising what bytes map to what instructions, how the instruction is made up, and how parameters are accessed.
  - The VM instructions to be executed could configure the VM, making it a bit harder to analyse.

“Bastardising” the file format

- Generally aims to:
  - Cause an analysis application to behave unexpectedly, while the Operating system loads it fine
  - be exploited / caused to crash
  - generate incorrect output
- Standard arms race
  - Only effective for a while.
  - Can be useful against tools widely used but not currently actively supported by their author (Ollydbg v1 for example)

“Bastardising” the file format

- Disadvantages
  - Portability
    - Different OS releases (Win 98 vs Win NT)
    - Emulator programs, such as WINE.
  - Sometimes its useful to debug your own programs
  - Some AV's make pick up on the changes

Summary

- Use multiple layers of protections that rely on each other
- Don't check values for consistency / correctness, just use them straight away
- Learn to attack your own implementation, in order to identify weaknesses
  - Perhaps keep an eye out on various reverse engineering forums / cracking forums.
  - Realise when and where to focus your efforts.
- Have fun in the process :)

Summary (cont)

- Given enough time, skill and resources, pretty much everything can be broken.

Questions?

Thanks for attending

If you have any feedback, please contact me.

andrewg@felinemenace.org

Thanks to all the FM and PTP people.

Bonus slide
(don't worry if you don't get these)

- gcc dmeiswrong.c -o dmeiswrong
- 13:21 < nemo> buf = malloc(size * 12);
- </3
- http://church.felinemenace.org
- rm -rf diary.of.pike
- It's ok, $ACTIVITY isn't for everyone.
- IPv6-compatible Poodles
- Melting fish
- “This is your warning shot.”
- Sometimes you hurt me.
- In internet it's everytime
- Deaths of civilisations.