Protecting binaries

Andrew Griffiths
andrewg@felinemenace.org

Introduction

- Who am I?
- 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 usually portable to other operating systems.

Defence in depth

- What is the threat model?
  - What problems are you trying to solve?
    - Casual copying
    - Determining who leaked a copy
    - Determined crackers
  - Once you know what you're doing you can work out a plan of attack
- Work out what is suitable and feasible to implement.
  - Code obfuscation / Watermarking / Licensing types
  - Re-evaluate what you've done in case its broken.

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)

- Idealistic defence in depth for binaries
  - When pieces are removed / tampered with, 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.

- “He who fights with monsters should look to it that he himself does not become a monster. And when you gaze long into an abyss, the abyss also gazes into you.”

  Quote by Nietzsche.

Watermarking

- Why watermark?
  - Watermarking does not prevent against fraudulent attempts to obtain the software.
- Fragile vs Robust watermarks
- Visible vs Invisible watermarks
- Watermarking values
  - Counter
  - Code constructs / code ordering
  - Data initialisation values
- Tamperproofing

Obfuscation

- Assembly level
  - 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?
  - What is its file size impliciations? Run-time costs?

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)

- Basic block 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.

Obfuscation (cont)

- Disadvantages to obfuscation
  - Performance impact
    - Breaks optimised code
    - May not be suitable if customers are having problems with their binaries, as its now harder for you to debug as well if there is nothing to go on but a core dump.

License scheme implementation

- Effort needed to implement
  - May be worthwhile to use a (decent) commercail offering.
- 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.

License schemes (cont)

- A small checksum can be used to ensure that keys have been entered without typos. (see http://bopedia.com/en/wikipedia/i/is/isbn.html for the ISBN implementation)
- 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)
  - Byte Code or Just in Time
- Advantages
  - Increased analysis time
  - Can be really convulted
- Disadvantages
  - A lot of custom development may need to be done, depending what you want to implement.
  - Can be a lot of effort to debug

“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

Code signing?

- What is signed code?
  - Verifies that code has not been modified after signing.
    - Cannot verify the code is safe, though “if you trust the vendor”, and assuming that companies haven't had certificates fraudently obtained, which almost has (had?) happened with Microsoft due to Verisign.
      - http://pcworld.about.com/news/Mar222001id45284.htm
  - Microsoft Authenticode and Java are prime examples of where signed code has been used.
- It may or may not be useful depending on curiminstance, and how it is implemented.

Packed code

- What is packed code?
- Common traits of packed code:
  - Decompression stub
  - Anti-debugging code
- Can usually be “dumped” to disk in order to carry out an investigation.
  - Both using core files (and reconstructing)
  - Proper process dumpers that can resume execution

Packed code (cont)

- **Dumpers are reasonably rare in the UNIX environment**
  - **Phrack article (http://www.phrack.org/show.php?p=63&a=12)**
  - **Cryopid (http://cryopid.berlios.de/)**

General signal handling

- sigaction() can be used to set signal handlers.
  - kill -l to list signals which can intercepted (except KILL. man 7 signal for more information.)
  - This can be used to achieve things similar to Shared Exception Handling under Windows.
  - Using SA_INFO allows you to get to the instruction context when a fault occured.
    - /usr/include/sys/ucontext.h
    - See struct ucontext.uc_mcontext, and the structure mconstruct
  - You can manipulate the registers (such as EIP, etc) to change control flow / make changes.

Random new trick

- Multiple pages referring to same data
  - Same shared memory segment mapped at multiple places
  - Can be used to modify pages at different addresses and hopefully confuse people trying to analyse the code
  - I need to sit down and work out some proper sample code.

Implementation problems

- Hardened kernels and decrypting code
  - Such as:
    - PAX (http://pax.grsecurity.net)
    - Grsecurity (http://www.grsecurity.net)
  - Main problem
    - Not allowing execution of code where you want
    - Not allowing arbitary page permission changing
  - Can possibly be worked around using the system
    - Not a real problem for legitimate pieces of software
    - Dependant on setup, the binary can be “marked” so it doesn't cause any problems.

Implementation problems (cont)

- Some areas can be worked around, depending if there are certain ACL's not in place.
- Otherwise, decrypting code and writing it to memory isn't possible.
  - This is mainly a problem for malicious pieces of code
  - You can use other techniques such as obfuscation to help hide what code the code is doing.

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
  - 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.
- It will be interesting to see what happens in the future with Pallidum / NGSCB / etc.

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.