

# FIGHT WITH LINUX

Firmy@XDSEC

<https://github.com/firmianay>

firmianay@gmail.com

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# WHAT IS GNU/LINUX?



- Hello, this is Linus Torvalds, and I pronounce Linux as Linux!
- Inspired by the UNIX OS, the Linux kernel was developed as a clone of UNIX
- GNU was started in 1984 with a mission to develop a free UNIX-like OS
- Linux was the best fit as the kernel for the GNU Project
- Linux kernel was passed onto many interested developers throughout the Internet

# DISTRIBUTIONS



守序善良

我给社区做贡献，  
社区让我赚大钱



中立善良

这个世界需要一个开  
源的类Unix

slackware  
linux

混乱善良

这个世界所有人都应该  
能自己处理包依赖



守序中立

红帽是个遵守GPL的好公司



绝对中立



混乱中立

不会编译整个Linux的  
不是好程序员



守序邪恶

红帽让我抄，  
你能把我怎么滴？



中立邪恶

用户友好是狗屁，  
用户中心才王道



混乱邪恶

吃我unity啦!!

- Linux is basically a kernel, it was combined with the various software and compilers from GNU Project form an OS, called GNU/Linux
- Linux is a full-fledged OS available in the form of various Linux Distributions
- Archlinux, Ubuntu, Debian, RedHat, Fedora are examples of Linux distros
- Linux is supported by big names as IBM, Google, Sun, Oracle and many more
- <http://distrowatch.com/>

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# WHY USE LINUX?

- Powerful
  - Runs on multiple hardware platforms
  - Users like its speed and stability
  - No requirement for latest hardware
- Convenience
  - A consistent software environments that is completely machine independent
  - Every system will have a GNU toolchain to compile code for the resident platform!
- It's "free"
  - Licensed under GPL

# JUST FOR FUN



一蓑烟雨

10月8日 17:02

大家好，给大家介绍一下，这是我女朋友。@Linux

```
firmy@firmy-pc ~$ cowsay "My heart belongs to you."
  _____
< My heart belongs to you. >
  -----
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
              ||----w |
              ||


```

浏览255次



等39人觉得很赞



# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# SETUP YOUR VM

- What is a virtual machine?
  - Simply, a computer in your computer
  - Really, a (usually) segregated virtual environment that emulates real hardware
  - Virtual Box, VMware Workstation Pro/Player, QEMU
- Why we need a virtual machine?
  - Safety, reliability, consistency, it's easy
  - Keep the binary in a contained environment
  - Snapshots
- What's in a virtual machine?
  - Lots of tools: debuggers, disassemblers, analyzers, unpackers, compilers...
  - 迅雷, 百度云, QQ

# HOW TO USE LINUX?

- Environments
  - Arch Linux
  - GNU bash 4.4.12(1)
- Resources
  - 鸟哥的 Linux 私房菜
    - <http://linux.vbird.org/>
  - man bash

# BASIC COMMAND LINE

- `ls [path]`
  - list directory contents
  - `"ls -al /"`
    - `/bin, /boot, /dev, /etc, /home, /lib, /mnt, /proc, /root, /tmp, /usr`
- `cd [path]`
  - change the working directory
- `pwd`
  - print the path of current/working directory
- `cat [file]`
  - concatenate files and print on the standard output
  - `"less", "more"`

# BASIC COMMAND LINE

- `cp [file] [location]`
  - copy the file/directory to the location
- `mv [file] [location]`
  - move (rename) the file to the location
- `rm [file]`
  - remove the file or directory
  - never do “`sudo rm -rf /`”
- `vim`
  - command line text editors
  - “type `:quit<Enter>` to quit VIM”

# BASIC COMMAND LINE

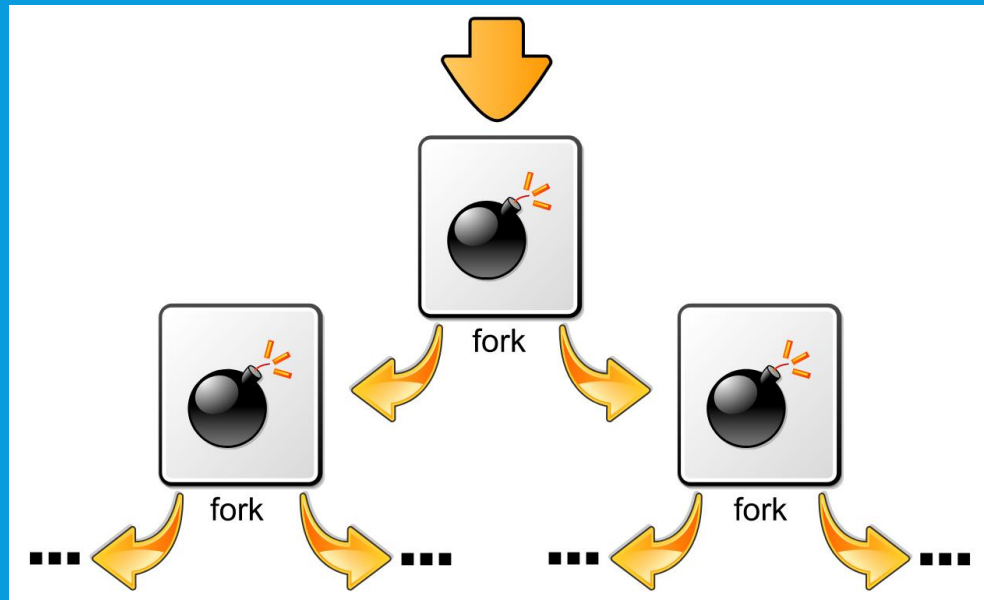
- `grep [pattern]`
  - print lines matching a pattern
- `find`
  - search for files in a directory hierarchy
- `man [command]`
  - an interface to the on-line reference manuals
- `apropos [whatever]`
  - search the manual page names and descriptions
- `[command] --help`
  - display help pages

# PIPES AND REDIRECTION

- Redirection
  - `/proc/[PID]/fd`
    - 0: stdin
    - 1: stdout
    - 2: stderr
  - “>”: take the standard output of the command on the left and redirects it to the file on the right
  - “>>”: take the standard output of the command on the left and appends it to the file on the right
  - “<”: takes the standard input from the file on the right and input into the program on the left
- Pipes - “|”
  - take the standard output of the program on the left and input into the program on the right

# JUST FOR SAD

- `:(){ :|:& };:`
  - Fork bomb
  - [https://en.wikipedia.org/wiki/Fork\\_bomb](https://en.wikipedia.org/wiki/Fork_bomb)





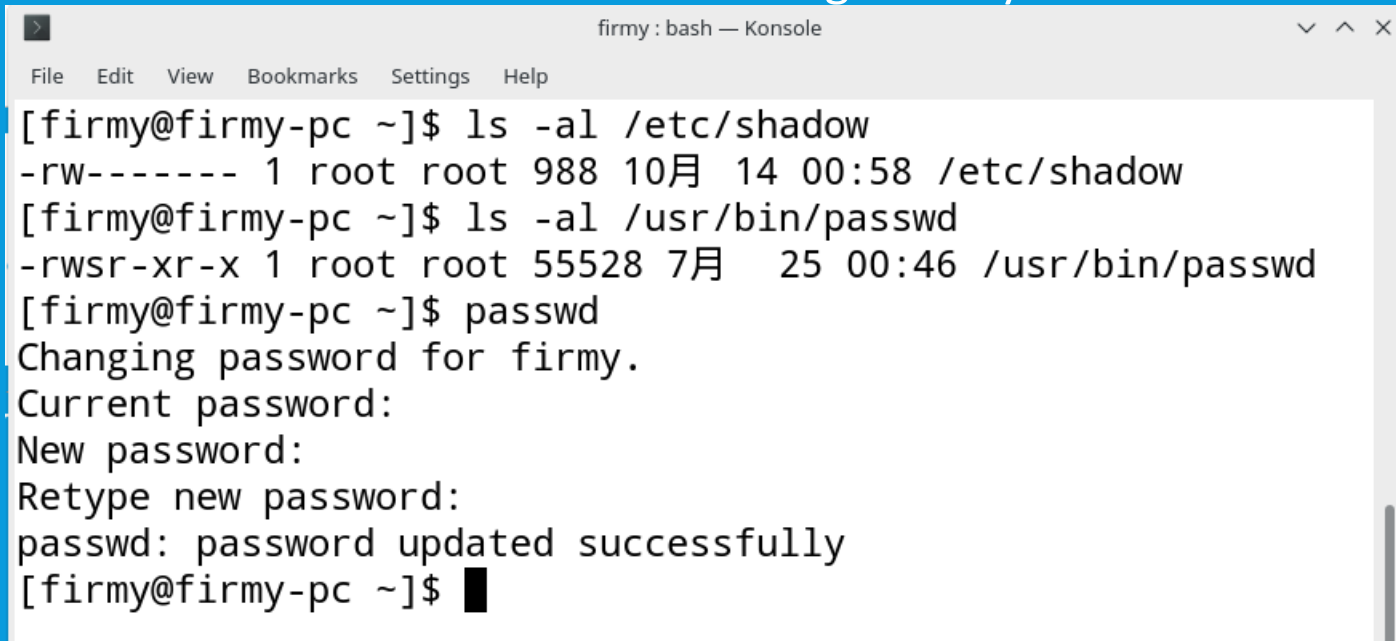
# LINUX FILE PERMISSIONS

- Owner, group
- Permissions set by owner/root
- Resolving permissions:
  - If user=owner, then owner privileges
  - If user in group, then group privileges
  - Otherwise, all privileges

1	2	3	4	5	6	7	8	9	10
File Type	User Permission			Group Permission			Other Permission		
	Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
d/l/s/p/-/c/b	r	w	e	r	w	e	r	w	e

# LINUX PROCESS PERMISSIONS

- Process (normally) runs with permissions of user that invoked process
  - “/etc/shadow” is owned by root
  - Users shouldn't be able to write to it generally



```
firmy : bash — Konsole
File Edit View Bookmarks Settings Help
[firmy@firmy-pc ~]$ ls -al /etc/shadow
-rw----- 1 root root 988 10月 14 00:58 /etc/shadow
[firmy@firmy-pc ~]$ ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 55528 7月 25 00:46 /usr/bin/passwd
[firmy@firmy-pc ~]$ passwd
Changing password for firmy.
Current password:
New password:
Retype new password:
passwd: password updated successfully
[firmy@firmy-pc ~]$
```

# LINUX PROCESS PERMISSIONS

## UID 0 is root

- Real user ID (RUID)
  - same as UID of parent (who started process)
- Effective user ID (EUID)
  - from set user ID bit of file being executed or due to sys call
- Saved user ID (SUID)
  - place to save the previous UID if one temporarily changes it

Also SGID, EGID, etc...

# EXECUTABLE FILES HAVE 3 SETUID BITS

- Setuid bit – set EUID of process to owner’s ID
- Setgid bit – set EGID of process to group’s ID
- sticky bit:
  - 0 means user with write on directory can rename/remove file
  - 1 means only file owner, directory owner, root can do so
- So, “passwd” is a setuid program
  - It runs at permission level of owner, not user that runs it

# EXECUTABLE LINKABLE FORMAT (ELF)

- Relocatable file
  - holds code and data suitable for linking with other object files to create an executable or a shared object file
  - a.o
- Executable File
  - holds a program suitable for execution
  - a.out
- Shared Object File
  - holds code and data suitable for linking in two contexts. First, the linker process it with other relocatable and shared files to create another object file. Second, the dynamic linker combines it with an executable file and other shared objects to create a process image
  - libc-2.25.so

# EXECUTABLE LINKABLE FORMAT (ELF)

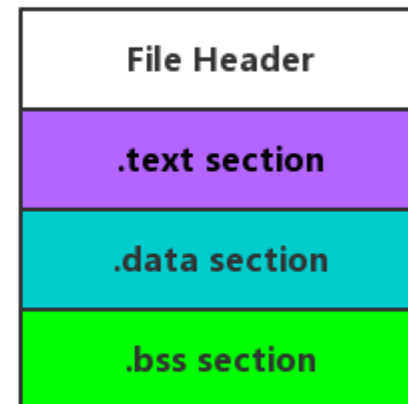
```
int global_init_var = 10;
int global_uninit_var;
void func(int sum) {
    printf("%d\n", sum);
}

void main(void) {
    static int local_static_init_var = 20;
    static int local_static_uninit_var;

    int local_init_val = 30;
    int local_uninit_var;

    func(global_init_var + local_init_val +
        local_static_init_var );
}
```

可执行文件/对象文件



# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

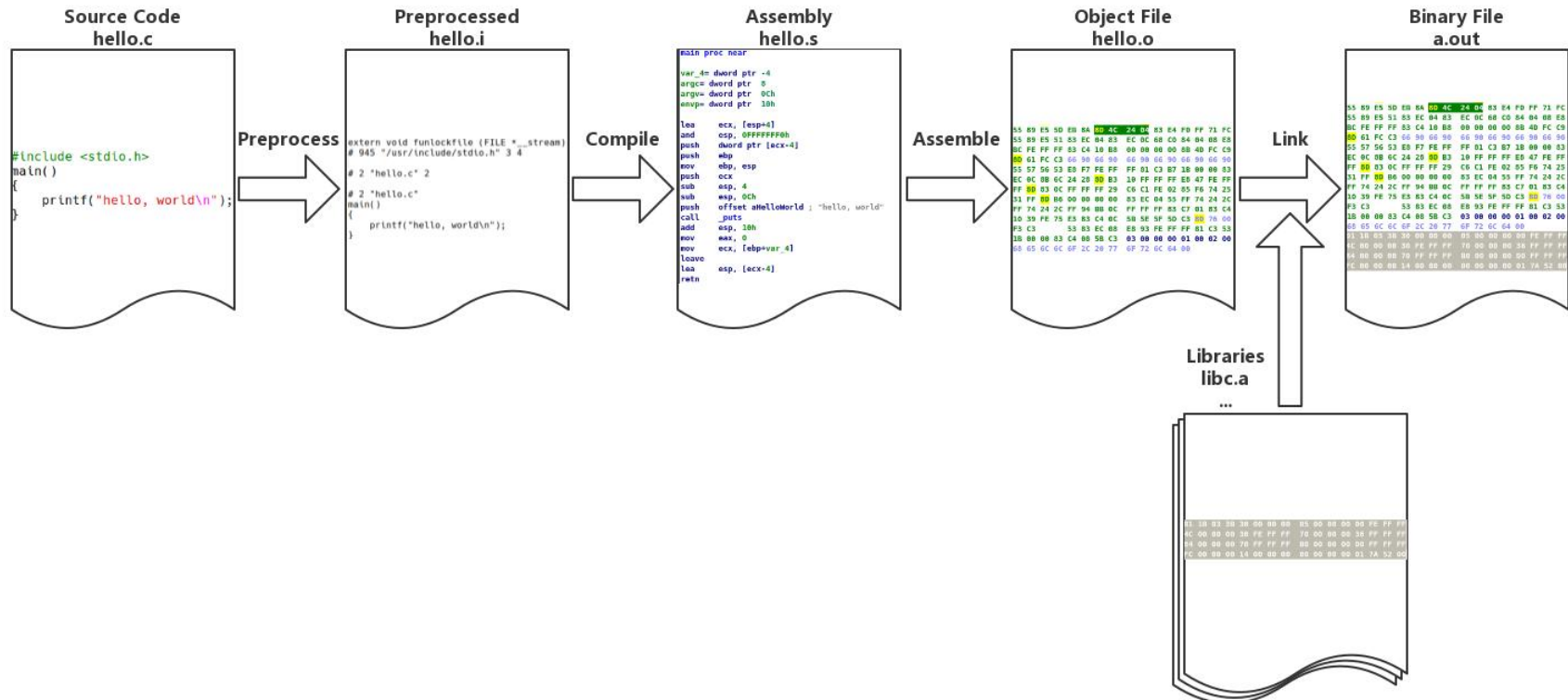
What?

Why?

How?

# FROM C CODE TO BINARY FILE

gcc -save-temps hello.c





# WHAT IS REVERSE ENGINEERING?

- The process of analyzing a subject system to
  - (i) identify the system's components and their inter-relationships and
  - (ii) create representations of the system in another form or at a higher level of abstraction
  
- From *New Frontiers of Reverse Engineering*  
[http://cypressosjsu.info/CS266/pdf/new\\_frontiers\\_of\\_reverse\\_engineering.pdf](http://cypressosjsu.info/CS266/pdf/new_frontiers_of_reverse_engineering.pdf)

# TERMINOLOGY

- Machine
  - A computer, server, sometimes refers to the actual CPU
- Binary
  - An executable such as an .EXE, ELF, Mach-O or other code containers that run on a machine
- Malware
  - A malicious binary meant to persist on a machine such as a Rootkit or Remote Access Tool (RAT)

# TERMINOLOGY

- Vulnerability
  - A bug in a binary that can be leveraged by an exploit
- Exploit
  - Specially crafted data that utilizes vulnerabilities to force the binary into doing something unintended
- 0day
  - A previously unknown or unpatched vulnerability that can be used by an exploit
  - An 0day can also be an exploit using the unpatched vulnerability
- Pwn/Pwning
  - In security, pwning commonly refers to vulnerability research, exploit development and sometimes luckily found a 0day

# APPLICATIONS

- Military or commercial espionage
- Software security analysis
- Bug digging and fixing
- Game external plugins
- Algorithm copy
- Saving money
- ...

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# WHY LEARN REVERSE ENGINEERING?

- Understanding of how programs really work
- It's a big challenge
- Almost non-existent in academia
- Few people have mastered
- Satisfy your curiosity
- Gain a sense of accomplishment
- Just for fun
- ...



# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# WHY DOES SOFTWARE HAVE VULNERABILITIES?

- Programmers are humans
  - Use tools
- Programmers often aren't security-aware
  - learn about common types of security flaws
- Programmers languages aren't designed well for security
  - Use better languages (Java, Python, ...)



# HOW TO DO REVERSE ENGINEERING?

## Static Analysis

- Disassembly, Decompile, Unpack, Deobfuscate
- Analyzing a binary without executing any code
- Can provide complementary insights to guide dynamic and advanced analysis
- Potential for more comprehensive assessment
- Lots of tools involved
- Safer

# HOW TO DO REVERSE ENGINEERING?

## Dynamic Analysis

- Debugging, Tracing, Memory dumping
- Analyze what happens when the binary is executed
- Are files made, processes created, websites contacted, files downloaded/executed, etc
- Show you the effect the binary has on the system/network
- Run binaries in a sandbox for safe

# EVASIONS AND OBFUSCATIONS

- To Defeat Static Analysis
  - Encryption (packing)
  - API and control-flow obfuscations
  - Anti-disassembly
- To Defeat Dynamic Analysis
  - Anti-debugging, anti-tracing, anti-memory dumping
  - VM detection, emulator detection

The main purpose of obfuscation is to slow down the security community

# REVERSE ENGINEERING PHASES

- Unpacking
  - The image of a running binary is often considered damaged: No known OEP. Imported APIs are invoked dynamically and the original import table is destroyed. Arbitrary section names and r/w/e permissions.
- Disassembly
  - Identification of code and data segments
  - Relies on the unpacker to capture all code and data segments

# REVERSE ENGINEERING PHASES

- Decompilation
  - Reconstruction of the code segment into a C-like higher level representation
  - Relies on the disassembler to recognize function boundaries, targets of call sites, imports, and OEP
- Program understanding
  - Relies on the decompiler to produce readable C code, by recognizing the compiler, calling conventions, stack frames manipulation, functions prologs and epilogs, user-defined data structures

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# WHAT WE CAN/WILL DO?

## attack

- Integer overflows
- Buffer overflows
- Format string vulnerabilities
- Heap overflows
- Return-Oriented Programming (ROP)
  - Sigreturn-Oriented Programming (SROP)
- Return-into-libc exploits

## protect

- Stack canary
- Non-eXecutable memory pages (NX)
  - Data Execution Prevention (DEP)
  - W xor X ( $W^X$ )
- Position Independent Executable (PIE)
- Address Space Layout Randomization (ASLR)
  - Position Independent Executables (PIE)

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

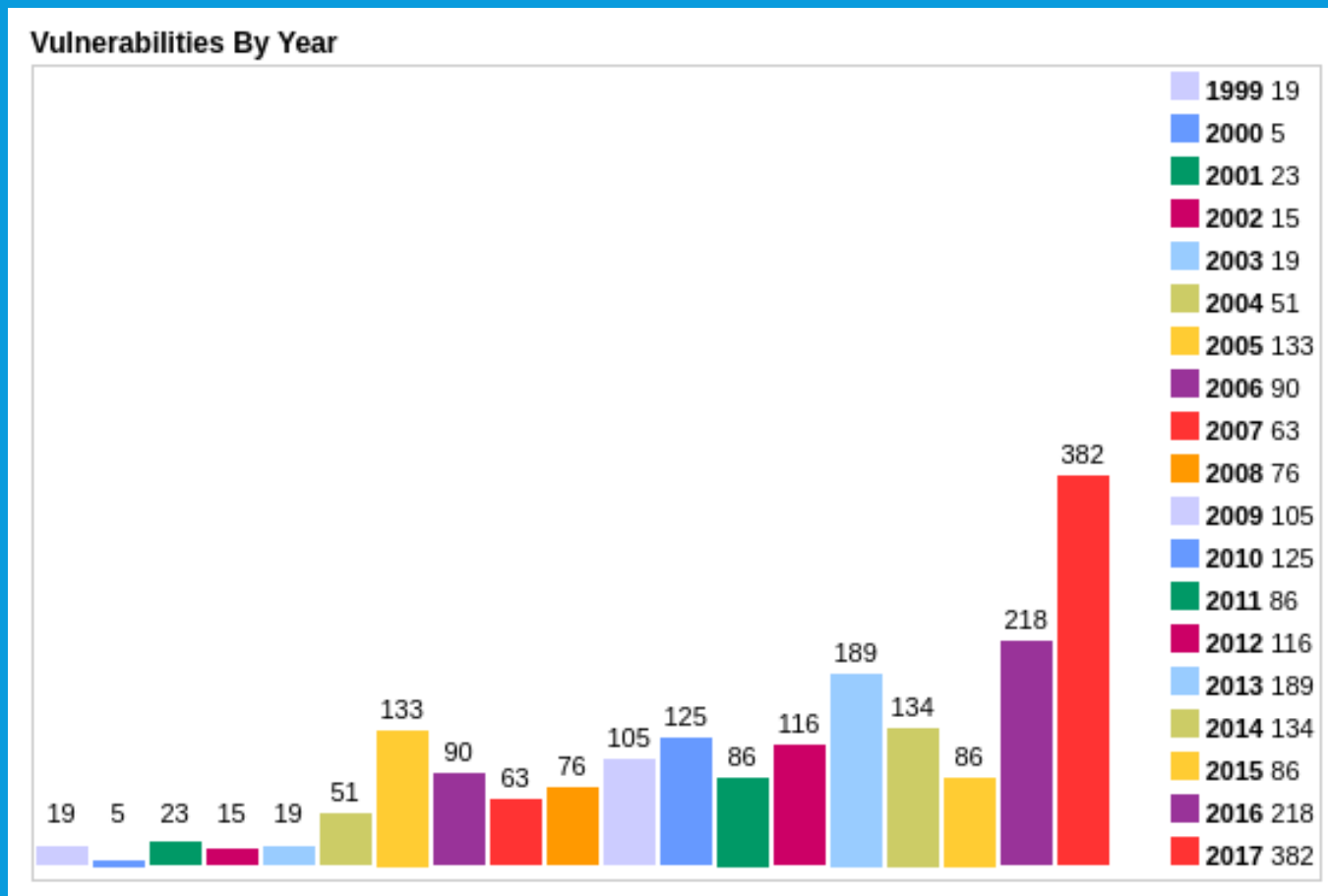
What?

Why?

How?

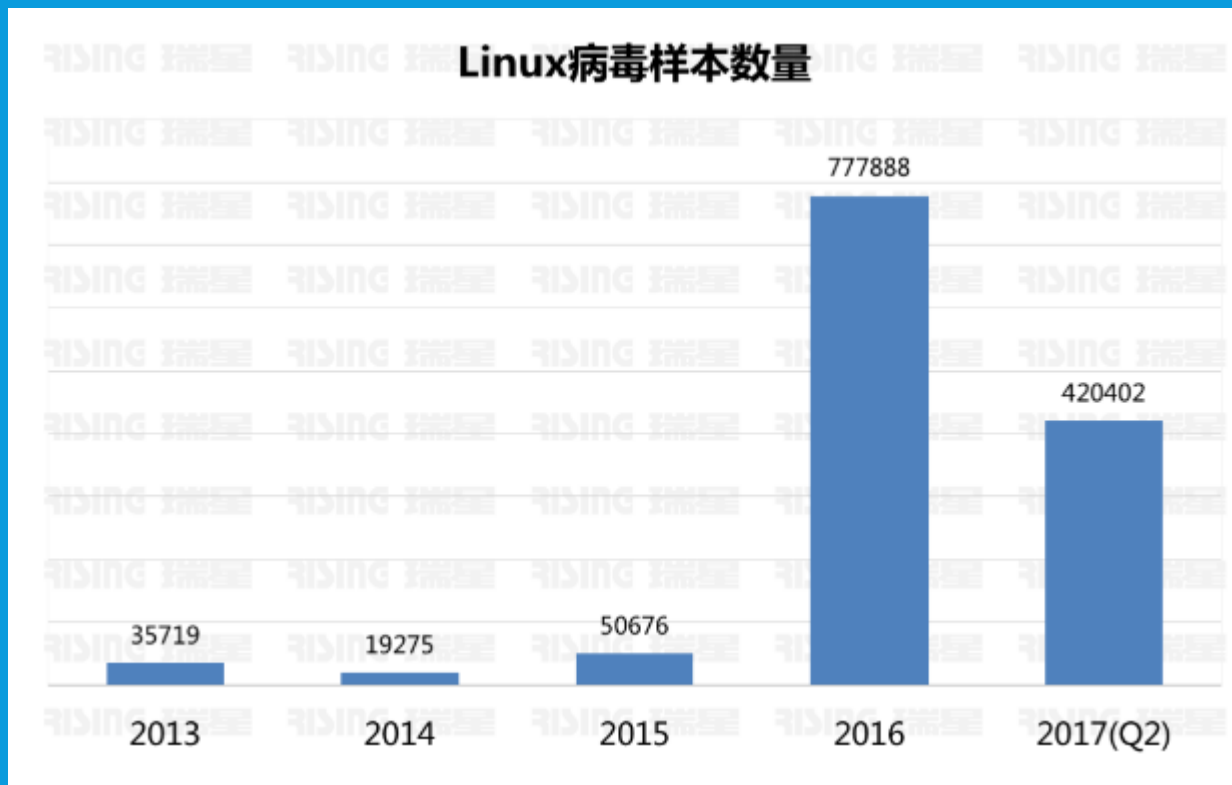


# WHY REVERSE ENGINEERING ON LINUX IS NEEDED?



<http://www.cvedetails.com/vendor/33/Linux.html>

# WHY REVERSE ENGINEERING ON LINUX IS NEEDED?



<http://www.sic.gov.cn/archiver/SIC/UpFile/Files/Default/20170807115920801889.pdf>

# OVERVIEW

- Linux
- Reverse Engineering
- Reverse Engineering on Linux

What?

Why?

How?

# LINUX TOOLS

## Hex editors / viewers

- wxHexEditor (GUI)
- xxd
  - “-i”: output in C include file style
  - “-g”: number of octets per group in normal output
  - “-l len”: stop after <len> octets
  - “-u”: use upper case hex letters
  - “xxd -g1”

# LINUX TOOLS

## ASCII readable hex

- strings
  - print the strings of printable characters in files
  - “-a --all”: scan the entire file, not just the data section [default]
  - “-t --radix={o, d, x}”: print the location of the string in base 8, 10 or 16
  - “-e --encoding={s, S, b, l, B, L}”: select size and endianness
    - s = 7-bit, S = 8-bit, {b, l} = 16-bit, {B, L} = 32-bit
- “strings -t x /lib32/libc-2.24.so | grep /bin/sh”
- “strings [executable] | grep -i upx”

# LINUX TOOLS

## File format on disk

- file
  - determine file type
  - “file -L [file]”: follow symlinks
- readelf
  - displays information about ELF files
  - “-h --file-header”: display the ELF file header
  - “-l --program-headers”: display the program headers
  - “-S --section-headers”: display the sections’ header
  - “-e --headers”: Equivalent to: -h -l -S
  - “-s --relocs”: display the relocations (if present)
  - “-d --dynamic”: display the dynamic section (if present)

# LINUX TOOLS

## Display information from object files

- objdump
  - “-d --disassemble”: display assembler contents of executable sections
  - “-R --dynamic-reloc”: display the dynamic relocation entries in the file
  - “objdump -d [executable] | grep -A 30 [function\_name]”
- ldd
  - print shared object dependencies

# LINUX TOOLS

## Tracing

- ltrace

- trace runtime library calls in dynamically linked programs
- “-f”: trace children (fork() and clone())
- “-p pid”: attach to the process with the process id
- “-S”: trace system calls as well as library calls

- strace

- trace system calls and signals
- “-o file”: send trace output to FILE instead of stderr
- “-c”: count time, calls, and errors for each syscall and report summary
- “-p pid”: trace process with process id, may be repeated



# LINUX TOOLS

## Debugger

- edb (GUI)
- gdb
  - GNU Debugger
  - A debugger for several languages, including C and C++
  - It allows you to inspect what the program is doing at a certain point during execution
  - Errors like “segmentation faults” may be easier to find with the help of gdb
  - peda / gef
    - “mv special ~/.gdbinit”

# GDB

- (b) break [file:]function
  - set a breakpoint at function (or file:function)
  - (i b) info breakpoints
    - show information about all declared breakpoints
- (r) run [arglist]
  - start your program (with arglist)
- (s) step
  - execute next program line; step into any function calls in the line
- (n) next
  - execute next program line; step over any function calls in the line
- (q) quit
  - exit from gdb

# GDB

- (bt) backtrace
  - display the program stack
- (p) print [expr]
  - print the value of an expression
- (c) continue
  - continue running your program
- (fin) finish
  - execute until selected stack frame returns
- (h) help [name]
  - show information about gdb command

# GDB

- x/NFU <addr>
  - examine memory
  - “N”: repeat count followed by a format letter and a size letter
  - “F”: format letters are o(octal), x(hex), d(decimal), u(unsigned decimal), t(binary), f(float), a(address), i(instruction), c(char), s(string) and z(hex, zero padded on the left)
  - “U”: size letter are b(byte), h(halfword), w(word), g(giant, 8 bytes)

# GDB

- `gcc -g hello.c`
  - the “-g” option will enable built-in debugging support
- `list [file:]function`
  - list specified function or line
- `edit [file:]function`
  - edit specified file or function

# GNU BINUTILS

- The GNU Binutils are a collection of binary tools
  - ld
    - the GNU linker
  - as
    - the GNU assembler
- Cross compile
  - i386, arm, mips, sparc, powerpc
  - amd64, aarch64, mips64, sparc64, powerpc64

# MORE TECHNIQUES AND TOOLS

- Fuzzing
  - An automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program.
  - AFL, LibFuzzer
- Symbolic Execution
  - Analyzing a program to determine what inputs cause each part of a program to execute
  - angr, Triton, S2E
- LLVM
  - Collection of modular and reusable compiler and toolchain technologies
  - clang
- Machine Learning

# RESOURCES

- Reverse Engineering for Beginners by Dennis Yurichev
  - <https://beginners.re/>
- Practical Reverse Engineering by Dang, Gazet, Bachaalany
- Hacking: The Art of Exploitation, 2<sup>nd</sup> Edition by Jon Erickson
- The Shell coder's Handbook: Discovering and Exploiting Security Holes, 2<sup>nd</sup> Edition by Chris Anley et al
- Secure Coding in C and C++, 2<sup>nd</sup> Edition by Robert C. Seacord



# CONTACTS

- Chao Yang
- Blog
  - Old: <https://firmianay.github.io>
  - New: <https://github.com/firmianay/blog/issues>
- Contact with me POLITELY
  - Tel:
  - QQ:
  - Telegram: @firmianay
- Contact me if you have any questions, or just want to talk with me 😊

一直学习就可以了

-- @Icemakr

# THANKS

2017.10.14