Check Point®
SOFTWARE TECHNOLOGIES LTD.

Workshop in Information Security

Building a Firewall within the Linux Kernel

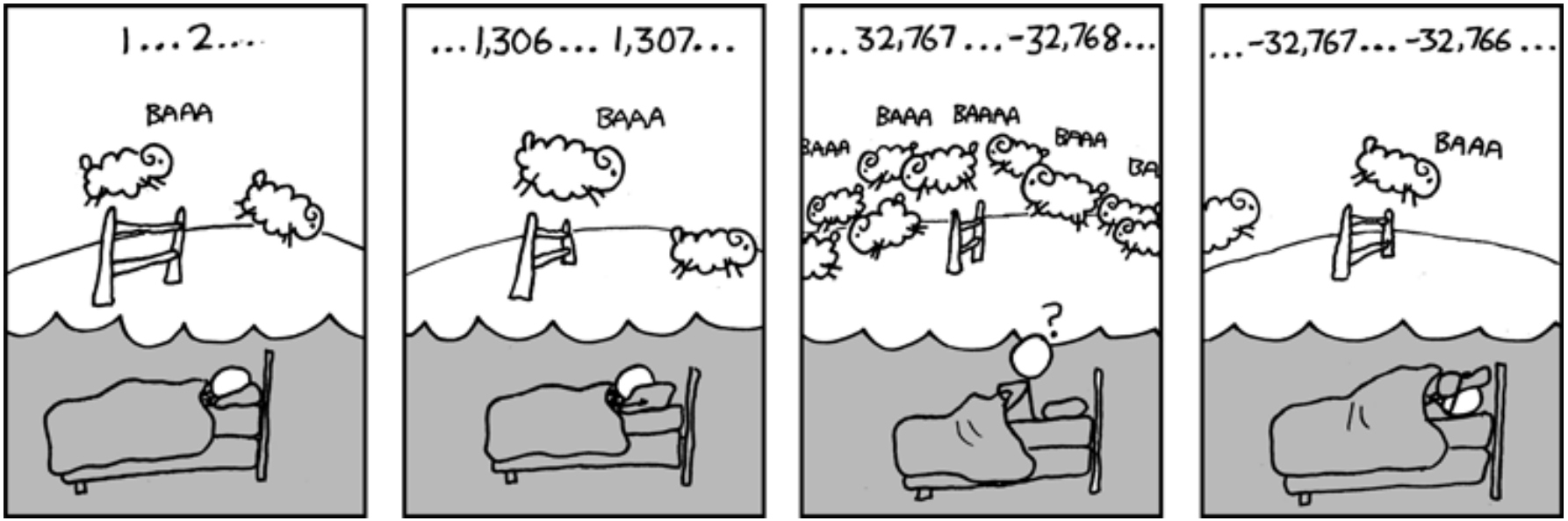
C Sockets, Software Vulnerabilities, Linux Kernel Modules.

Lecturer: Eran Tromer

Teaching assistant: Ariel Haviv

Advisor: Assaf Harel

Can't Sleep (xkcd.com/571)



1 C Sockets

2 Software Vulnerabilities

3 Intro to Linux Kernel Modules

1 **C Sockets**

2 Software Vulnerabilities

3 Intro to Linux Kernel Modules

Sockets

- The server:
  - Creates a socket.
  - Binds it to a port.
  - Listens (sets a maximum queue size of incoming connections)
  - Accepts (each accept call returns a new conversation)
- The client:
  - Creates a socket
  - Connect to a listening server
- Then they can start sending and receiving data.
- At the end, both sides close the socket.

Sockets

- The client and the server are actually two executables, one at each end of the connection.
- They can sit on the same machine and communicate through localhost (AKA: IPv4 127.0.0.1), or on two different continents.
- They can be written in two different programming languages, as long as they both use the socket API.
- We will use C because of its weaknesses.

Endianity

- How would you code the integer 0x004B89AF using 4 bytes?
  - Big Endian: 0x00, 0x4B, 0x89, 0xAF
  - Little Endian: 0xAF, 0x89, 0x4B, 0x00
- The net is coded in Big Endian, Intel machines are Little Endian, etc… Everyone chooses his endianity. We don't want to really worry about it when we write our code.
- When we read a packet, its encoded in net's endianity. But if we use a little endian machine?
  - Host to Net Long or Short: htonl, htons.
  - Net to Host Long or Short: ntohl, ntohs.
  - What about 1 byte values (like char)?

C Sockets

- Enough theory, get your hands dirty:
  - Include “sys/socket.h”, “sys/types.h” and friends.
  - Use all the wonderful structs & API defined there.
  - Don’t forget endianity! (htons, ntohl etc…)
  - Compile the server, and the client. On two different machines.
  - Start producing traffic…
- References:
  - Read man pages (socket, bind, listen etc…)
  - Beej's Guide to Network Programming
  - Your favorite search engine.

1 C Sockets

2 **Software Vulnerabilities**

3 Intro to Linux Kernel Modules

Software Vulnerabilities

- A common mistake is to think that by writing the code of the software, you know you will never get bad input from the other side of the conversation.
- Someone can send you a hand-crafted packet with bad input – and BOOM.
- If you don't check the input, and it's bad input:
  - You might crash due to segmentation fault. That's the better scenario.
  - In a worse scenario, you don't crash:
    - You mess up data in another part of your program.
    - Someone can execute code on your machine.
    - You unknowingly expose sensitive data.

Buffer Overflows

- Buffer Overflow – The most common vulnerability on the net.
- Caused by accessing unallocated, uninitialized, unwanted or un-something-else memory.
- Can be easily accomplished in C/C++, that are too permissive regarding array indexing and memory access.
  - E.g. no run-time check for array out of bounds access.

Secure Code

- A secure code:
  - Cautiously checks the input, before using it (like getting an offset in array from the user)
  - Uses safe functions (e.g. strncpy vs strcpy).
- That's exactly what we will "forget" to do, in order to see an attack in action.
- There are more types of attacks out there except buffer overflows.
  - Actually most of them are caused by lack of input checking.

1 C Sockets

2 Software Vulnerabilities

3 **Intro to Linux Kernel Modules**

What is a Kernel Module

- What is a kernel module? (wiki definition)
  - An object file that contains code to extend the running kernel, or so-called base kernel, of an operating system.
- What is a kernel module? (my definition)
  - A modular piece of code and data structures, that can be plugged in and out of kernel space.
- Be careful (or happy?)… When you insert a module into the kernel, you have access to all kernel code and memory.
  - That's why only the super user can do it.

Building the Module

- The purpose – eliminate the need to re-compile the kernel every time you need to add/remove a specific feature.
- A Makefile that adapts itself to current kernel.
  - Look it up!
- insmod and rmmod the module in and out the kernel.
- Initialization function that is called when the module enters the kernel.
- Cleanup function that is called when the module is removed from the kernel.

Our Kernel Module – The Firewall!

- What will we do with our kernel module? (spoilers ahead)
  - Register a char device, to communicate with the user space (AKA: the real world).
    - Make sysfs virtual files to get and set module values.
    - Use the mmap API to expose large chunks of data from kernel space.
  - Register our own functions (AKA: hooks) with the netfilter API, to issue verdicts on packets going in/out/through our linux box.
  - Maybe juggle some kernel threads, that will help us complete deferred or a-synchronic tasks.
- When our module is removed, it will clean up all this mess, as if it was never there.

Writing C for the Kernel

- The kernel API is a different one than the familiar user-space one.
  - Can't include <stdio.h>, or any other glibc header.
  - But <kernel.h> offers some nice utilities
    - e.g. min_t(type, x, y), swap(a, b)
  - And there are many more: kfifo.h, slab.h, kthread.h, wait.h…
- A few tips for debugging:
  - Use 'printk' to keep track of what's happening.
  - Use 'dmesg' to see the latest news from the kernel, and from your module playing around there.

References

- Further reference:
  - Linux Device Drivers, Third Edition
    - An excellent free e-book, contains all you need and don't need to know about kernel modules.
    - Written for kernel 2.6, but not a lot changed since.
  - Kernel Headers and Documentation
    - On your machine
      - e.g. /usr/src/linux-headers-`uname -r`/include/linux/ip.h
    - On the net
      - LXR or any other cross-reference site.
      - http://kernel.org/doc/Documentation/
    - The hardest to read, but probably the most useful.
  - Your favorite search engine.