**The Kernel**

**The main core of a system**

Written in both C and ASM

Has 2 spaces:

* User
* Kernel

Manages:

* Hardware
* Memory
* CPU
* Devices

The Kernel Space

* Protected space that manages everything from memory, hardware and core services of kernel
* Very little restrictions once space is accessed
  + Exploitations do exist that allows you to get from user space to kernel space easily
* Has full access to all hardware and memory
  + Even the protected memory space which is reserved specifically for the kernel
  + Modifying data in here when itys not supposed to be can cause the machine to immediately crash]

The User Space

* The common space used by most applications and user
* Use syscalls to reach into kernel space depending on its task
  + Or to kernel interface first, then to kernel
* This is to know what context the app is being ran in

**The Kernel Stack**

A set of software and hardwar that allow the kernel to operate

* Syscall interface
  + User space reaches out to kernel space
  + Read files or perform networking task
* Memory/process mgmt.
* Network stack
* Virtual File System
  + Sits on top of physical one
* Devices
* Architecture dependant code

**Syscall Interface**

Layer of abstraction between hardware and user space

* Many different hardware out there that need abstraction

Allows for user to make a function call without reaching directly into kernel space

* More secure than allowing a user to reach into the kernel

**Memory & Process Management**

Handles all processes in the system

Interfaces with processer when handling threads

Manages all memory on the system, physical and virtual

**Network Stack**

Controls and manages how machine interfaces with the network hardware

Handled in stack:

* Application
* TCP/IP/Ethernet
* Driver
* NIC
* Physical Medium

**Virtual FS**

Interface used so kernel can abstract direct hardware interactions on the drive for the file system

* Useful to avoid any corruptions, or issues cause by not doing things in a uniform way
* If things are done differently, each app would need to know how to process things individually
* With a virtual FS, we can just tell the app to allow the vFS to handle sorting and putting it into blocks etc.

**Devices**

Core of the kernel code

Interacts directly with hardware

Provides code to directly interface with all of the hardware

All syscalls and kernel operations go through a device driver

* Drivers are required to interface with hardware

**Architecture Dependant Code**

The code of the kernel may change depending on the architecture of the system

During the install, the installer will load the required code for the identified architecture of the system at that time

* Every install you do for an OS will put ADC on the system, based on what’s installed on the system at that time

The drivers take up the majority of the ADC

**Loadable Kernel Modules (LKM)**

LKMs allow you to add kernel code to the system without having to recompile and update the kernel, restarting the system

* Can load and extend kernel code whilst still running

This allows full access to the kernel space

This can be used to hide persistence

* LKM can be used as a root kit
  + Reverse shell
  + Create a user every time it starts up
  + Force things to not show up like open ports