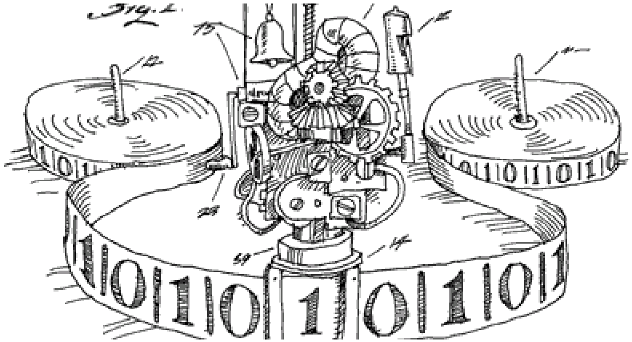
Reverse Engineering Informational Concepts

Introductions and Why?

Computational & Information Theory

- Before dealing with real systems, We need to go over some overarching concepts presents we can use in reverse engineering to find out information about systems before ever looking into them
- By understanding the underlying concepts of computers and information, it automatically gives a person a more innate understanding of a system that is difficult to grasp without a knowledge of the theory itself
- Theories we’re going over are the
  - Shannon’s Information Theory
  - Theory of Computation
  - And finally the work of Church-Turing Hypothesis



Information Theory

- Although Initially build for the Cryptographic field, Claude Shannon's information theory which is more of a unified ecosystem of the mathematics of information itself, but don't worry the actual math is outside of the scope of this conversation
- Information Theory itself should be viewed as Mathematical Approach so there are terms that may be used differently in this theory that have alternate meanings elsewhere
- The first theory we're approaching is know as the "**Source Coding** Theorem", in this context **Source** refers to a **Information Source** where as **Coding** Refers to the "Encoding"(Compression or Encryption) of information
  - ***"it is impossible to compress the data such that the code rate is less than than that of the shannon entropy of the source without being virtually certain that information loss will ocurr"***
- The Second Theory is the "Noisy Channel Theorem" in which
  - ***"(For any given degree of noise contamination of a communication channel) it is possible to communicate discrete data nearly error-free up to a computable maximum rate through said channel"***
- Thought's?

Shannon's channel coding theorem (simplified version)

Let C be the capacity of the channel. For the rate $R < C$, there exists

- a block code (N, K) (with $R = K/N < C$) and
- a decoding algorithm

such that the code error vanishes

$$\lim_{N\to\infty} P(\text{error}) = 0 \quad (\text{keeping } R = K/N < C)$$

Shannon Entropy of Information

- Proposed by Claude Shannon the concept of Information Entropy differs significantly from the physics concept of Entropy
- This type of *Informational Entropy* is effectively meta-information about the information itself where as it is literally a ”measure of uncertainty” in a given medium(channel)
- In addition for our purpose the term ”uncertainty” main be a bit vague; but realistically, it refer’s to the probabilistic of how the information communicates
- For most applications, this is simply used to disclose the state(Compression or Encryption) in which information exists but can be used to calculate more nuanced information if utilized carefully
- To put it simply “Shannon entropy is simply the calculation of how likely the symbols in a given set are to be similar(or dissimilar)”
- If this still makes no sense, It’ll make more sense once we put it into a practice

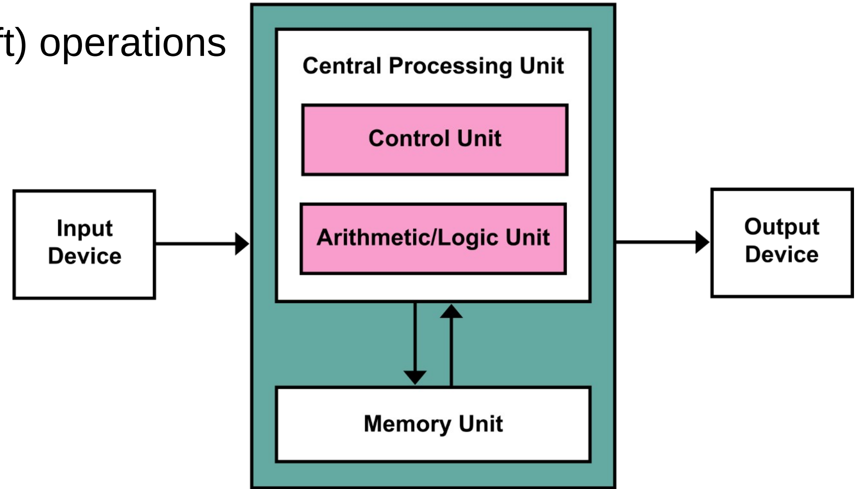
Model of Computation

- During the Genesis of both Analog and Digital computers, Computer Science was initially a field of mathematics in which the concepts of computers were interpreted as *abstract machines*
- There are multiple computational model group that exist but those the we’re concerned about is/are the Sequential Models
  - Register Machines(von Neumann Architecture)
    - Effectively the Basis of all Modern CPU’s
  - Turing Machines
    - Infinitely Computationally Complex
  - Finite-State Machine*
    - A Singe finite state(number) of a set of possible states(number’s) are functionally exclusive from each other
- While the Register machine for our purposes are the most significant there are components that are signifiant to us specifically...

*a Finite-State Machine is used in a Turing Machine

Register Machine Specifics

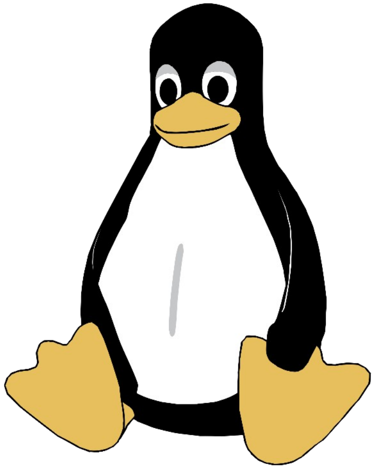
- There are multiple subsets of register machines in which we’re further refining out scope by utlizing the Von Neumann Arch. Model of a CPU
  - CAU(Central Arithmetic Unit)
  - CCU(Central Control)
  - “Memory”
  - I/O
- All of these aspects are crucial to implementing the concepts of abstract register machines; in which they require
  - Discrete (Functionally Infinite) Registers / C(Carry) / S(Shift) operations
  - Instruction Sets
  - State Tracking (IP)



That is a lot of information!
It will make more sense further on

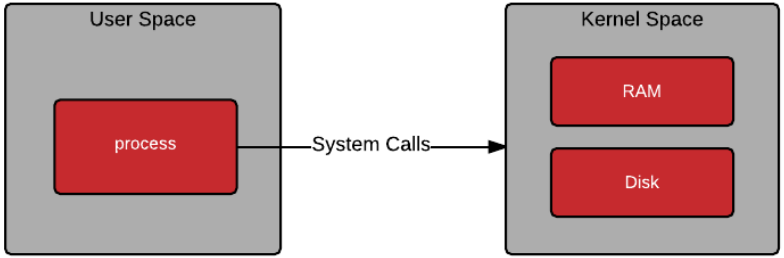
Model of Computation

- Using the previously held models as well as abstract register machine, we can form the basis of an operating system, as for this series we're only dealing with *nix operating systems
- So by applying the model of computation to something like a **file**, we can understand the basis of all Unix/Linux Systems, in which **ALL** components of the operating system can be accessed(rwx)
- So using this model in conjuction with the file concept, the basis of almost all modern computers are created
- Going forward we're fleshing out the concepts of Linux/Unix Operating Systems



*nix Operating Systems

- As stated before the most important concept of *nix operating systems is that **“Everything is a file”** from a more traditional text file, all the way to the very memory of the machine is and can be abstracted to a file
- Broadly, the operating system can be separated into two components the **Kernel** and the **Userspace**
  - The Kernel handles low level functions of the computer itself that is abstracted from the user including but not limited to as networking, cryptographic operations, and power management
  - The Userspace is defined as an area of operation where the actual user of the operating system’s program’s typically for the specific and intended function of providing utility to the user. So in turn the Userspace is laid over the kernel in order to interact with resources allocated by the kernel
  - For various reasons the user space is more restricted in freedom of operation than the kernel space



Speaking of files...

- Based off of the Distribution used, the arrangement of files on “disk” can very but vague follows mnemonic naming conventions
- As based off a root(‘/’) directory

/home	/usr	/etc	/root	/opt	/lib	/boot	/sbin	/bin	/var	/mnt	*media*	/tmp

Linux and Reverse Engineering

- One of the most confusing and difficult concepts about these systems is that they can be as simple or as complex as people want them to be, simply put at their core all it is a basis of abstract ideas given shape
- Once you understand and master these concepts computing and it’s concepts become open to you, as regardless of their parentage there are no computers that don’t rely on fundamental abstractions set forth by the mathematics and computer engineering laws
- This gives you a base to understand and in turn begin to reverse engineer any machine albeit that being much more complex and difficult than it seems
- In essence, when it comes to reverse engineering computers
  - You cannot escape Physics and Mathematics they fundamentally govern all computing concepts

CPU Instructions

- Instructions/Machine code is functionally the lowest possible abstraction(except for binary) for CPU function
- Instructions are made of Hexadecimal(0x0 – 0xf) Operation Codes(Opcodes) in which give a specific “Width” (Bits) are typically made up of at least a single instruction but often include other operands and arguments which are heavily reliant on the preceding instruction
- Based off the CPU the instruction set will differ on different architecture
  - ARM
  - X86
  - MIPS
- Understanding the basis of CPU instructions bridges the gap between traditional object oriented Languages, Intermediary languages, and Assembly languages

```
0000000000000000 3B10                  CMP EDX,DWORD PTR [RAX]
```

```
0000000000000000 0FA2                  CPUID
```

```
0000000000000000 FF                    INC DWORD PTR [RAX]
```

```
0000000000000000 F1                    ICEBP
```

```
00000000 8DBD                          LEA EDI,[EBP+00000NAN]
```

Instructions w/ registers

- To help for identification, almost all instructions are meant to have a phonetic/contraction name this is somewhat the same for any assembly
- Registers are effectively the same concept as they are in the previously mentioned machine they are sections of memory used to do intermediate interactions for the CPU
- Depending on the memory width the registers will have a different prefix letter
  - R(AX/BP/SP/IP)
  - E(AX/BP/SP/IP)
  - AL, ...
- The number of Registers depend on the Instruction set and Width

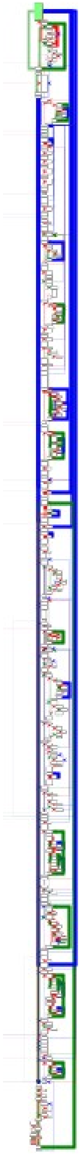
```
(gdb) info registers
eax 0x1 1
ecx 0xbffff064 -1073745820
edx 0x80483ed 134513645
ebx 0xb7fbe000 -1208229888
esp 0xbfffefc8 0xbfffefc8
ebp 0xbfffefc8 0xbfffefc8
esi 0x0 0
edi 0x0 0
eip 0x80483f0 0x80483f0 <main(int, char**)+3>
eflags 0x246 [ PF ZF IF ]
cs 0x73 115
ss 0x7b 123
ds 0x7b 123
es 0x7b 123
fs 0x0 0
gs 0x33 51
```

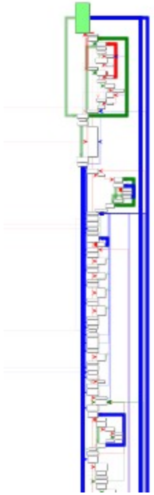
Why Create Different Architectures?

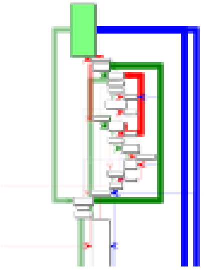
CISCV vs RISCV plus ISA?

Programmatic Aspects

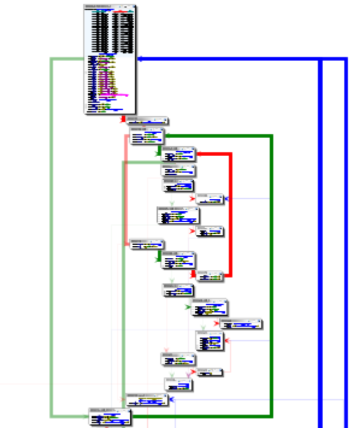
- Instructions are generated from Source Files, it is Assembled, Compiled ,and Linked so that the instructions can be interpreted by the CPU
- Since instructions are naturally difficult from humans to understand intuitively so there are some measures we use to help conceptualize the algorithmic and programmatic aspects used in programming
- Cyclomatic Complexity
  - A Emperical measurement of a progams complexity by measuring the linearity of Independent paths present in the code
  - Although not from the original, from a governmental perspective the McCabe Scale is useful
  - 1 - 10: Simple procedure, little risk
  - 11 - 20: More complex, moderate risk
  - 21 - 50: Complex, high risk
  - > 50: Untestable code, very high risk
- NSA Bsim Measurements

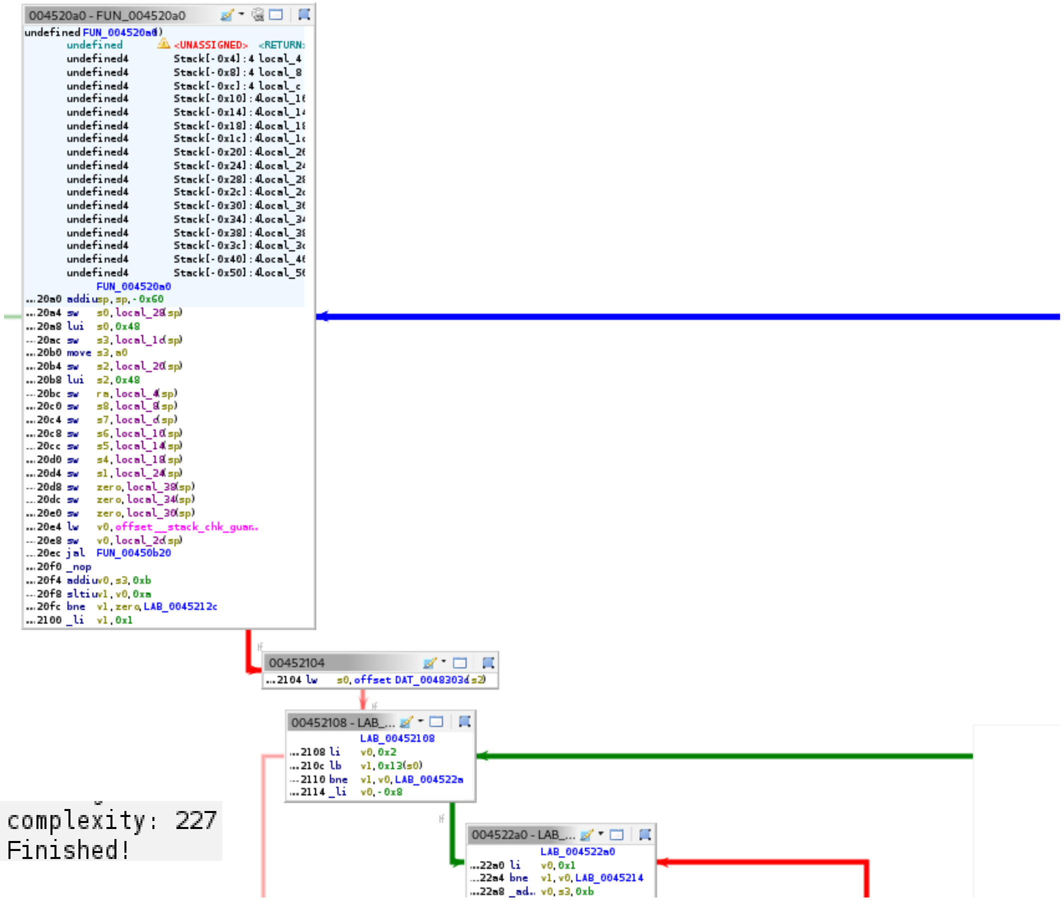






Cyclomatic Complexity



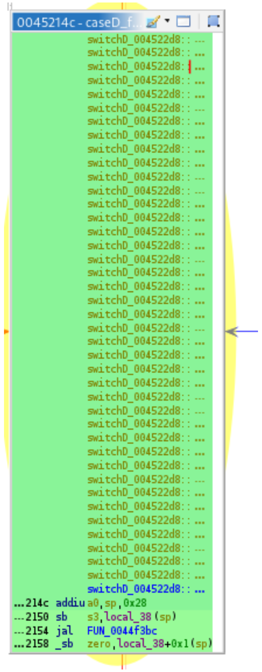


This Function(busybox switch) has over 368 Vertices amd CC of 227

```
ComputeCyclomaticComplexity.java> complexity: 227
ComputeCyclomaticComplexity.java> Finished!
```

Cyclomatic Complexity

- That would mean and incredibility unstable function right? Not Exactly.
- Sometimes massive switch Control Structures cause the CC to skyrocket but realistically,the switch functional is a manageable level of complexity



NSA Ghidra Bsim

- The concept of a Bsim database was not invented by the NSA but was largely developed under the umbrella of Ghidra, in which Bsim Databases are a collection of functions and sub routine vectors that are utilized by a given binary in such a way that they can be compared across multiple programs
- This allows us to establish connections to programs that would not typically be observable, for example establishing differences between versioned products or Malware Authors sharing code
- From a high level the Bsim database processes functions based off of the arch independents **Vectors** such as function arguments, datatypes, relative locations and a lot of other factors
- Effectively this then allows us to generate a “closeness” to another compared function, in which the closer the calculated “closeness” is to 1 the more similar a function is ex. “0.95/ 95% similiarity ”

BSim Search Results - [server: testsb.mv.db, function: FUN_004520a0, Similarity: 0.7, Confidence: 0.0]

Function Matches - 1 results

Stat...	Similarity	Co...	Function Name	Matching Function N...	Exe Name	Ingest D...	Location
	1.000	1503.218	FUN_004520a0	FUN_004520a0	busybox	Sep 9, 20...	004520a0

Filter:

Executables - 1 results

Exe Name	Ingest Date	Function Count	Confidence
busybox	Sep 9, 2025	1	1503.218

Questions?