CMPSCI 630 Systems **Fall 2009**

Lecture 9-26

Lecturer: Emery Berger *Scribe: Jane Tangen*

Ontology recapitulates phylogeny:

- notion: evolution or organism follows the development of evolution.

This is not true currently, every computer system goes through the exact same evolutionary stages.

- Very first OS's: single user, single process, no protection.
  - Then added multiple users and multiple processes through time sharing.
  - Now protection is needed! Then more advanced protections!
    - File system protection (technically happened later), memory protection
    - VM's introduced: MMU (memory management unit)
- Now personal computers and PCs come around
  - Apple (1984) and IBM PC (1981)
  - Now single user, single process, no protection. Went back to beginning for ease and cheapness and to not repeat mistakes of the past
  - No MMU
    - Side note: current automotive and small IOT have no MMU . ARM has the option, but not added because that's expensive. People bet on hardware and software being done properly (scary).
  - Evoultion of MS-DOS:
    - story: previous os for 8080 called CP/M, with idea of lattices. Invented by Gary Kildall (idea of lattices and OS based on this). IBM wanted to buy an OS, so approached him and wanted him to do his thing on a new one (8088?). He said no, they couldn't agree. So IBM went to Microsoft. They didn't really do OS at that time, just BASIC, crappy interpreter (written by Gates and Allen). But Microsoft said they had an OS, and agreed as long as had MS sticker, thats good. Microsoft was like shit so they found guy that made OS for 8080 as hobby, gave MS all rights. Poor dude.
  - Macintosh Computers:
    - took a decade to separate processes from killing the whole damn thing.
    - Infinite loop of process and you would have to reboot the machine. This happened a lot.
    - Unix, which was owned by ATT, was incredibly expensive. Berkley made their own OS based off Unix called BSD. Apple eventually took this OS and built theirs on top of it.
  - Notion of "security through obscurity" existed at this time. Considered no security at all.
  - iPhones:

    * First ones had one user, one processor, no security.
    * If on a phone call, nothing else would work. Only one user process. For a while even the OS couldn't run in background.
    * Would literally exit one process to go to the other process.
    * Many people hacked the iPhone and found it was a stripped down unix.
    * Now: Multiple processes and memory protections and lots of security!

- What about Programming Languages?
  - Starts out as dynamically typed
    * This means types are associated with values. In statically types language, both values and variables have types.
    * LISP at first was dynamically typed language
      · Accidentally dynamically scoped as well. Normally we usually prefer statically/lexically scoped. Scope is about name resolution. Lexically is with globals and nesting with blocks (objects inside objects). Variables resolved by going out. This is what we normally use. Dynamic scope is terrible. Here is F(x) which refers to X, foo calls f(x), but here instead of referring to the inner scope, go to bar with has x that calls foo, but also baz that has x which calls foo, then foo's use of x will use the x in the call stack. It walks up the call stack. Not intended in LISP, eval was written incorrectly, stuck for 25 years. Not until Common LISP was this fixed.
  - Interpreter: early languages have EVAL.
    * EVAL takes string and runs it. Very powerful. Makes code hard to compile in advance, or compile at all into efficient code.
  - Garbage collection: Start with Reference Counting, which is not good but easy.
  - Why a language takes off: Random person wants to fix a problem, then others don't have a choice and have to use it.
  - This evolution happened again in In perl, ruby, python, JS... kind of java too (but defined by PL people, with lexical scope and real GC from the beginning.) Tends to happen: once finally takes off, people endeavor on decades long process to make faster with JIT compiler, type system, and gradual typing.
    * Python now has JIT (pypy)! Still has weird scoping.
    * TypeScript has block scoping!
  - What happened with JS though?
    * Before JS was Java applets. JAVA goal was to do this "safe": with no buffer overflows (check reads and writes), no dangling points, memory safe, type safe. Originally called Oak, made for toasters. Then Sun came out with HotJava browser, which was cool and could upload java, and do it safely? Java had bytecode, interpreter, GC, and connections were only like 9600 bps —¿ so bloated and slow connection. Startup time terrible. Netscape (with first graphical browser Mosaic that eventually became Mozilla) and then made Netscape Navigator.
    * Needed better solution: Brenden Eich was a PL person, not much time to make it, liked small talk, but bosses wanted it to look like Java, so he added curly braces (a "curly-brace language")
    * First called LiveScript, made in 10 days. Not fast, but it was better. Would just work. (1995)
    * From Emery's lab: Browsix (and before was Doppio)- can run normal OS programs in browser.

    - asm.js introduced: JS has monkey patching: redefining anything in the program is OK by JS. Not good for static analysis because anything can be redefined. So asm.js said only use a subset of JS that is easy to compile. So use a contract with a magic word at the beginning. Could set things to actually set a type instead of like an object thing and it became more fast.
    - Weird arguments over asm.js lead (eventually) to WebAssembly (WASM)! Has to be safe, efficient, and fast to parse. Parsing was thought to be solved (process is not much actual parsing) but now its a bottleneck. (Dev tools in chrome can actually show you whats happening, like where all the time is going. Kinda cool.)
    - Safety was a real problem. Banks are running IBM 360 code on VMs, Conventional ISA (instruction set architecture) has no memory protection, arbitrary structure (GOTOs) and hard to analyze. NaCL and PNaCL (ARM) was result of research for safe ISA. ActiveX was very bad for security. Emery is very upset about this.
    - Ok but WebAssembly: uses suites to test it called PolyBench because it was not a normal program. They rewrote benchmarks for polyhedral operations. Significant gap between native and non-native code performance.
- Register based vs stack based architecture: Registers use registers, and stack does push and pop. Stack-based is good for easy analysis.
- Attacks:
    - Return Oriented Programming (ROP) you can find any sequence of instructions that are before returns, so call one before return to make that instruction. Seizing control of jumps means taking over computer.
    - WASM tries to use CFI (control flow integrity) so that you cant just jump around.