**Halting Problem**

TESTHALT on input arbitrary P, I

Construct a new program Z:

If TESTHALT(I) == true loop forever Then, Calling Z(Z)

**Rice‘s Theorem**

Any functional property (semantic & non-triviality) X of a program cannot be perfectly decidable!

Assume TESTX can decide program Q on input J has X. For P terminate on I: Construct program T on input J:

•Run P on input I.

•Clear all memory (restart machine).

•Run Z on J, where Z is an arbitrary program that actually has property X. Use the output of Z as the output of T.

•Feed T, J to TESTX and output the result as the final result of TESTHALT.

**There is no fully automated software testing without a trade-off!**

**Complete**: report all errors

**Sound**: reports no false alarms

**7+1 Kingdoms**

1. Input Validation and Representation Problems -Maliciously Crafted Inputs (Buffer overflows, Cross site scripting attacks, SQL injection) -White listing approach for input validation

2. API Abuse -Implicit Assumptions about Communication Partners to Behave as Agreed (Identify expensive services and launch a distributed denial of service attack, Spoofing GPS data to order a pizza to a wrong location) -Can partly be dealt with using cryptography

3. Problems when Using Security Features/Tools -Correct Combination of Correct Security Mechanisms (Using bad or outdated cryptography, Requiring too weak authentication techniques)

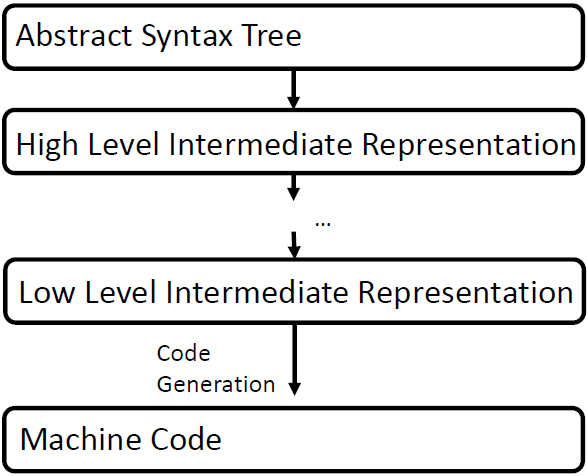
4. Parallelism and Consistency Problems -Time and State (Keeping databases consistent) -Semaphores, Transactions in database management systems

5. Error Handling/Output Problems -Error Handling is Error Prone (Error message discloses if first half of password is correct, Error type may give attacker valuable information)

6. Code Quality Problems (unpredictable behavior, easier for attacker, Mistakes hard to spot)

7. Encapsulation/Isolation Problems (No separation of authenticated data vs. unauthenticated data, different users’ data, output reveals secret.)

8. Dependence on Environment (DNS, TLS Tunnels, Good Randomness)



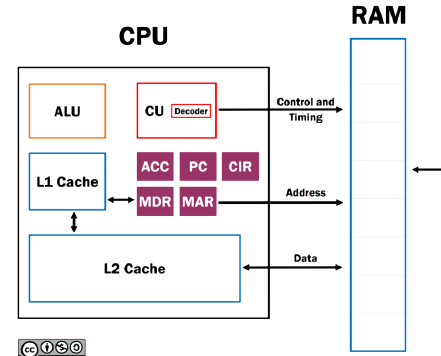
**Compilers add code:** Code to save processor registers before jumping to subroutine, Start-up code to include libraries / initialize runtime; remove code: Loop unrolling or inlining, Remove variable names

Set of all possible machine instructions is called instructions set architecture (**ISA**)

Since binary code can be disassembled, a common attacker model is that a program is available as assembler code to the attacker

**CPU:** registers, control unit, ALU.

**Von Neumann Architecture**

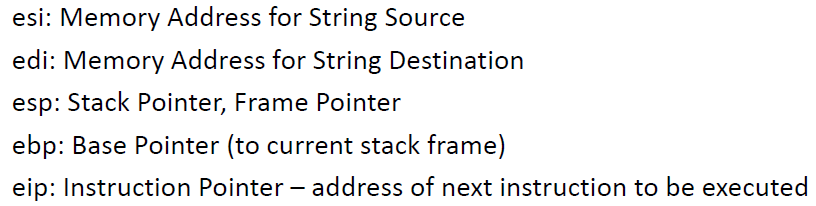


**Program:** a sequence of instructions

**Executing a program:** Processor reads next instruction from memory and executes it with the corresponding data (**Von Neumann Cycle**): Fetch, Decode, Fetch Operands, Execute.

**Pointers:** load addresses of memory units into registers.

**Memory mapping:** each hardware component is provided with a dedicated memory area. Via manipulation of the memory units in these areas, the CPU can control the hardware.



rax reads and stores all 64 bits of rax; eax 32; ax 16; al 8

**.txt** executable code (the program);

**.data** initialized data; **.bss** uninitialized data; **Heap**; **Unused memory; Stack**

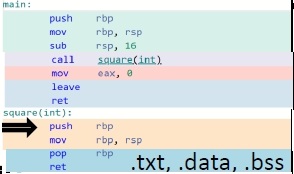
The basic mechanism would only allow a linear control flow! (aligned, no nesting)

**PUSH:** sp -1, store X. **POP:** return stored sp, sp +1, **CALL:** push pc +1 on stcak, load called func into pc. (store ret addr). **RET:** pop as ret addr, store in pc. (continue the caller).

**Nested func call: s**tore (on the stack) the address of the next command to be executed (PC+1) by the caller.

**stack frame/ activation record**: a dedicated area in the stack that stores local values belonging to that function

**BP** will hold a central reference address in the stack. **<BP:** input variables of the current function. **>BP:** local variables of the calling function. **stack walking:** Moving from the current stack frame to previous via loading older and older base pointers.



leave: mov rsp , rbp; pop rbp

**buffer overflow** =memory corruption attacks (wider sense)

**Attack:** - inputs more characters than the variables are supposed to hold - inserts malicious assembler code into stack - overwrites return address to point to malicious code

**Code Injection** (challenge: without null bytes), **Arc Injection** (jump to another position), **Return-to-libc** (call func in standard lib)

**Return oriented Programming：**jump to a location that is ended by a ret. Jumps called gadgets.

**4 Observations:** Buffer overflow; Overwriting return address; Malicious codes executable; Return address map to malicious codes;

**Countermeasures:** Range checks for variables (type safe memory language, fat pointer) -efficiency; check integrity (stack canaries) -attacker can output stack; Malicious code not executable (Read-only and write-only memory organization); Make address organization unpredictable (Address Space Layout Randomization)

**Fat pointers:** store the length of the memory and check at runtime if bounds are legal.

**Concurrency:**

**Amdahl’s Law:**



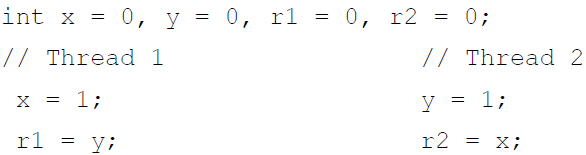
Time-of-check-time-of-use Error

**Concurrency property:** At least two control flows must be executing concurrently.

**Shared object property:** A shared race object must be accessed by both of the concurrent flows.

**Change state property:** At least one of the control flows must alter the state of the race object.

**Dekker’s Example**



R1 and r2 can be both 0 if reordered.

**Race window:** a code segment that accesses the race object in a way that opens a window of opportunity during which other concurrent flows could “race in” and alter the race object.

**Critical Section:** A race window protected by a lock or by a lock free mechanism is called a critical section.

Reason: 1, Timing dependent and manifest sporadically. 2, Race conditions depend on the Meta-Context that is not immediately related and often are non-local.

**Locks based Approaches: Mutex (Mutual execution), Read-write Locks, Semaphores, Condition Variables** (used for signaling ordering between threads. They allow a thread to voluntarily release a lock and wait for a condition to be satisfied.)

**Atomic Operations:** indivisible and uninterruptible. **Barrier Synchronization**, **Message Passing:** through inter process communication (IPC).

**Bugs:** Atomicity Violation (/Lock

based synchronization), Order Violation (/Condition Variables)

Deadlocks: Mutual exclusion. •Hold and wait: Threads hold resources allocated to them while waiting for additional resources. •No preemption: Resources cannot be forcibly removed from threads that are holding them. • Circular wait: There exists a circular chain of threads such that each thread holds one or more resources requested by the next thread.

**Bug Depth:** number of ordering constraints a schedule has to satisfy.

Typically, X high depth.

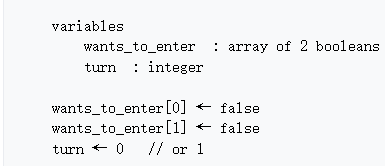
Finding: Using delay statement

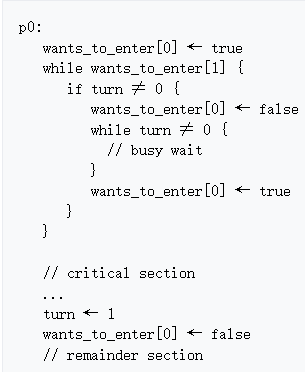
Probabilistic Analysis, (improved) Exhaustive Testing.

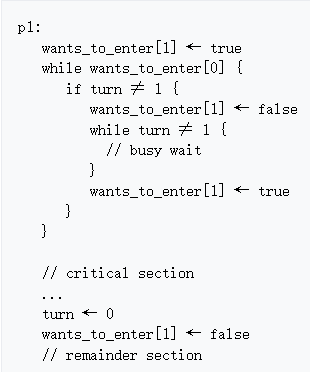
Make sure that invariant is fulfilled. Less expensive, false positive.

**Invariants: Lock Sets:** Any access to shared memory is protected by locks mechanism. **Data races**: Desirable: No concurrent read write to same memory location. **Happens-before-checking:** No unordered reads and writes to the same memory location.

Dekker’s algorithm:



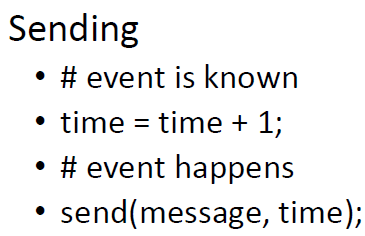


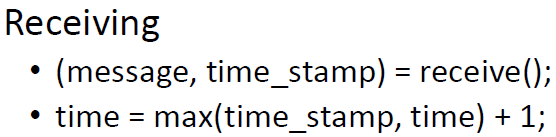


**Algorithms**: Token-based (sequence number), Non-token based (communicate, timestamps, logical clock), Quorum-based. X out of order.

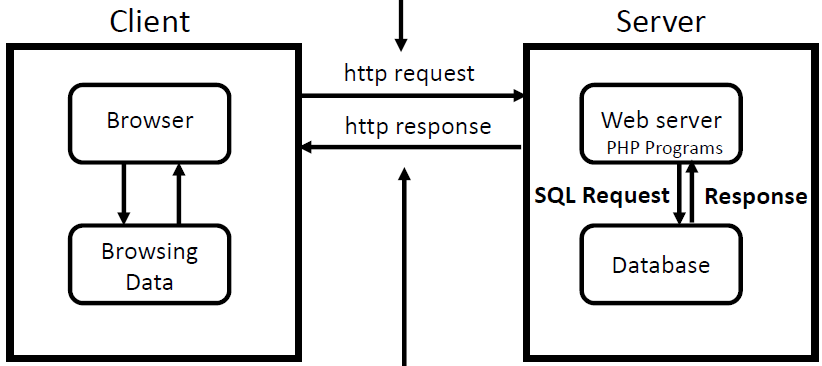
**Required Properties:** No Deadlock, No Starvation, Fairness, Fault Tolerance.

**Lamport‘s Timestamps**





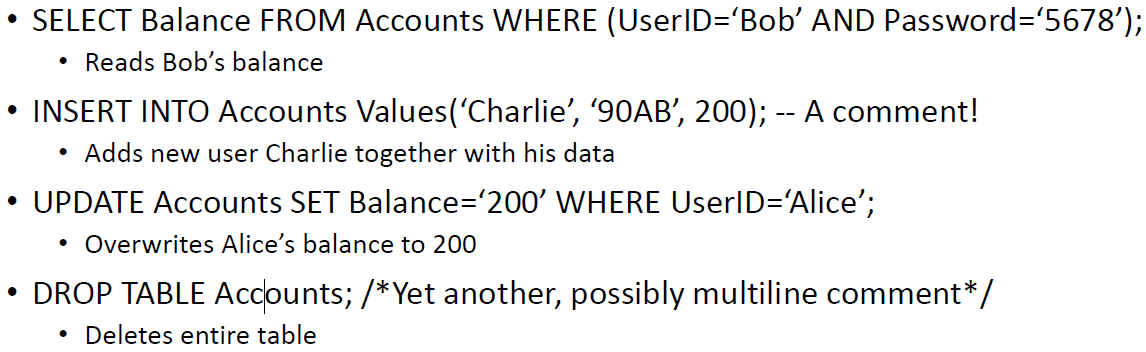
**Quorum-based:** Each site requests only a subset of sites which is called a quorum. Any two subsets of sites or Quorum contains a common site. This common site is responsible to ensure mutual exclusion.



Ephemeral State, Long Term State

**GET**: URL contains all information required to process request, request to read only. **POST**: may change data, input data in explicit data fields, e.g. posting to forum.

**DBMS** implement ACID transactions that guarantee validity of data even in the face of problems like errors, power failures. Atomic, Consistent (state), Isolation (Concurrently like in order). Durability (persistent).



http response may contain dynamically generated files that are usually output by a PHP program that runs on the server and communicates with the database.

**Blind SQL**

Boolean based blind SQL injection: Attacker manipulates conditions to get a true or false response of the application. • Time based blind SQL injection: Attacker introduces time delays in the SQL query to infer true or false conditions.

Root: attackers can construct input values that would be evaluated to modify the SQL command.

**Countermeasures:** •Input Validation, Whitelisting (rejection), Blacklisting (delete unwanted characters), Escaping, Prepared Statements.

**Further Techniques:** Use Restrictive Acccess Control Policy to Important Tables, Encrypt Sensitive Data.