Date _____ Day _____

CHAPTER # 11

→ Main cause of Software Vulnerabilities
  ↳ Poor progamming practices are a major source of Security flaws
    ↳ Design flaws
    ↳ Coding Errors
    ↳ Incorrect assumptions about i/p & env

→ Fixing bugs early is cheaper & easier than after release.

→ Common Dangerous Coding Issues (check everything → datatyp, length, format, range)

① Unvalidated Input
  ↳ Failure to properly check & sanitize user input, allowing attackers to inject malicious data or cmds

② Buffer Overflows
  ↳ Occur when a program writes more data to a buffer than it can hold, potentially overwriting memory & executing malicious code

③ Injection Flaws
  ↳ Happen when untrusted input is interpreted as cmds or queries (eg: SQL, OS cmds), allowing attackers to alter program behaviour

Date _____ Day _____

④ Cross-Site Scripting (XSS)

↳ Allow attackers to inject malicious scripts into web pages viewed by other users, leading to data theft or session hijacking

⑤ Improper Error Handling

↳ Reveals sensitive sys or application information through err msgs that can aid attackers in exploitation

→ **Recommendations**

→ Reduce impact using resilient sys architectures

→ Stopping vulnerabilities before they occur by using improved methods for specifying & building software

→ Finding vulnerabilities before they can be exploited by using better & more efficent testing techniques

→ **Software Quality & Reliability Vs Software Security**

↳ "Software Quality & reliability" focus on accidental failures caused by random inputs, unexpected interaction or coding mistakes

↳ These failures follow a probability distribution & are addressed using structured design & testing

↳ Testing targets common inputs & typical errors to reduce how often failures occur, not necessarily eliminate all bugs.

→ Use whitelisting rather than blacklisting, means default should be denied.
→ Centralize validation logic
→ fail safely

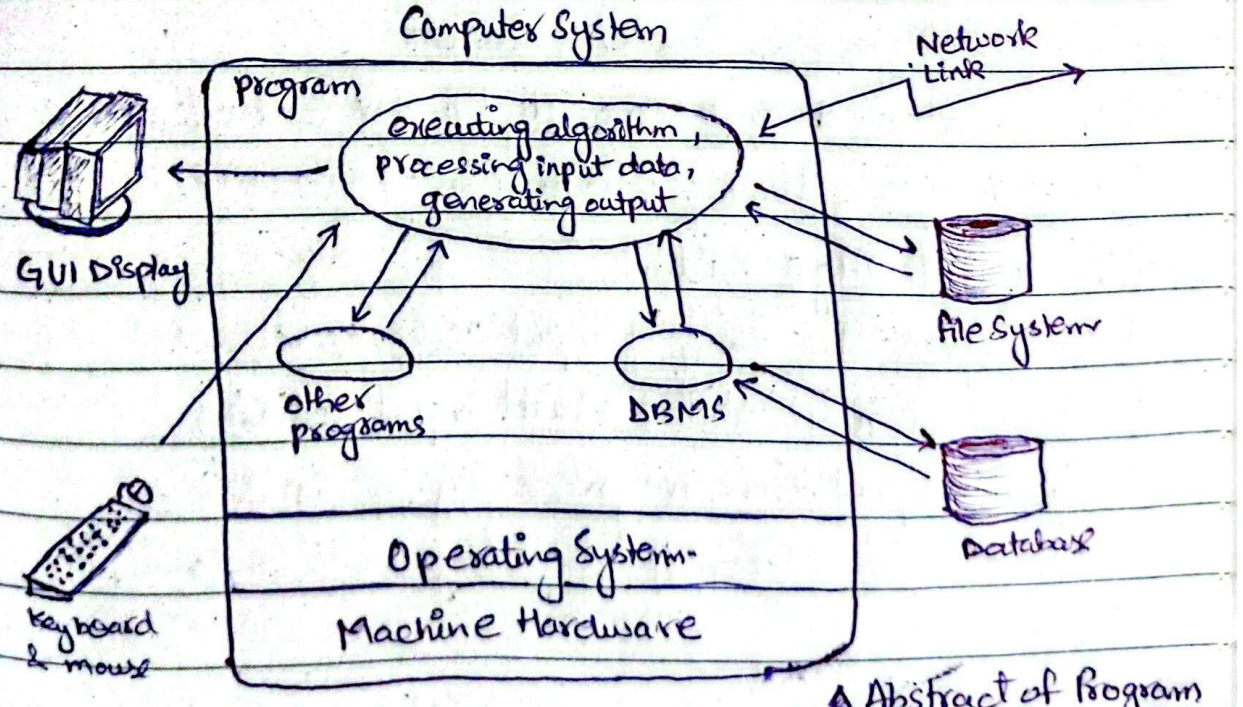
Date _____ Day _____

↳ "Software Security" focuses on intentional exploitation, where attackers deliberately target specific bugs

↳ Attacker choose inputs that are abnormal & unexpected, so such bugs often escape normal testing

→ Defensive or Secure Programming

- Design software to continue functioning under attack or fail gracefully
- Never assume anything — validate all assumptions
- Check all inputs, error states & execution conditions
  ↳ It might be difficult to hard
    ↳ Incr code size & dev time
    ↳ often conflicts with business pressure for fast delivery



▲ Abstract of Program

Date _____ Day _____

• WRITTING SAFE PROGRAM CODE

→

→ Programs process input data using an algorithm to solve a problem

→ In procedural languages (eg: C), the algo is a sequence of steps manipulating input data

→ Program execution ultimately means machine instr. running on a processor, using memory & registers

→ From Software security point, key concerns are:

↳ Correctness of the algo for the intended problem

↳ Accurate translation from high-level code to machine instructions

↳ Valid & meaningful manipulation of data in memory & registers.

→ Correct Algorithm Implementation

↳ If an algo doesnot correctly handle all valid cases and inputs, it may introduce bugs

↳ Causes of Algo Failure

① Incomplete handling of input cases or variants

② Incorrect assumptions about input validity

③ Poor interpretation or processing of valid input

④ Weak implementation of security critical-logic

Date _____ Day _____

(1) Example: Netscape Random No. Generator

↳ Early Netscape browsers used a weak random number generator for SSL session keys

↳ The RNG was seeded with predictable information

↳ Attackers could predict session keys & decrypt secure web traffic

↳ Fix: Re-implemented Random No. Generator using sufficent unpredictable entropy

(2) Example: TCP Session Spoofing / Hijacking

↳ Attack goal: Impersonate a trusted host to gain unauthorized access

↳ Exploits predictable TCP Initial Sequence No. (ISN)

↳ Attacker guesses ISN & sends forged ACK packets

↳ Server assumes a valid connection & grants trusted access

↳ Hijack variant injects pkts during an active legitimate session

↳ Key Issues:

- ISNs used as both identifiers & authenticator
- Many TCP/IP implementations used predictable algos for ISNs

↳ Fix:

- Use randomized ISNs

Date _____ Day _____

→ Developers often add debug features during development.
↳ Can leak sensitive info
↳ & bypass authentication / security checks
↳ Example: sendMail included a DEBUG cmd in production which was exploited by Morris Internet Worm.

→ Interpreters must strictly enforce language semantics & security rules
↳ Early JVM implementation failed to do so
↳ Resulted in unauth access to sys resources

→ Ensuring that Machine Language Corresponds to Algorithm
↳ Source code correctness alone is insufficient for full security
↳ Compiler themselves can be attack vectors
↳ Detecting such attacks requires:
- Careful comparision of src & machine code
- Analysis accross many files & binaries
↳ Trusting compiler can undermine all higher level security measures, it can bypass
↳ Authentication checks ↳ Access controls
↳ Security Validations

Date _____ Day _____

→ This level of security is mainly req in high-assurance system like (military, critical infrastructure)

→ Machine code validation is ~~rarely~~ rarely done due to cost & complexity.

" Even a perfectly secure algo can be compromised if the compiler generates malicious or incorrect code "

→ Correct Interpretation Of Data Values

↳ All computer data is stored as binary bit, groped into bytes, words or larger units

↳ Incorrect interpretation of data can lead to security flaws or crashes

↳ Strongly Typed VS Loosely Typed languages

↳ Strongly Typed

- Restrict operations based on data type
- Reduce risk of invalid data manipulation

↳ Loosely Typed

- Allow flexible interpretation of data
- Permit casting b/w integers & pointers
- Increase risk of programming err
  - → buffer overflows
  - → memory corruption
  - → crashes or attacker controlled behavior

Date _____ Day _____

→ Correct Use Of Memory

↳ Programs that handle unknown data sizes often use dynamic memory (heap)

↳ Memory must be allocated when needed and released after use

↳ Attackers can exploit memory leaks to cause DDoS

↳ Older Language (eg: C)

- No built in dynamic memory management
- Difficult to track memory usage in large programs
- Memory leaks common & hard to debug

↳ Safer Languages (Java, C++)

- Provide automatic memory management
- Introduce some performance overhead
- Produce more reliable programs
- Strongly recommended to reduce memory issues

→ Preventing Race Condition with Shared Memory

↳ Shared memory may be accessed by multiple processes or threads

↳ Without proper synchronization, shared data can be corrupted, overwritten, lost due to overlapping access.

↳ A race condition occur when multiple processes compete for uncontrolled access to a shared resource

Date _____ Day _____

↳ Prevention requires correct selection & use of synchronization primitives

↳ Incorrect sync can cause deadlocks hard to recover from ↵

↳ Careful design & partioning are required

↳ Attackers can exploit deadlock to cause ΔDoS

- INTERACTING WITH THE OPERATING SYSTEM & OTHER PROGRAMS

→ Programs execute under the control of an OS, not in isolation

→ The OS manages & mediates access to system resources among running programs

→ When program runs, the OS creates an execution env. for the process.

→ Programs require appropriate access to resources to function correctly.

→ Enviornment Variables

↳ String values inherited by a process from its parent

↳ They are stored in process memory and can affect program behaviour

↳ A new program may override, or modify env variables

↳ Modified variables are passed to child processes

Date _____ Day _____

↳ Env variables are a src of untrusted input and must be validated

↳ Security Risks

- Common attack goal: Privilege escalation by undermining privileged programs
- Well-known risky variables: → specifies set of directories to search for any given cmd
  - PATH – controls command lookup
  - IFS – defines word separators in shell script
  - LD_LIBRARY_PATH – contrds dynamic library loading

Ex # 01

```
#!/bin/bash
user=`echo $1 | sed 's/@.*$//'`
grep $user /var/local/accounts/ipaddrs
```

→ is script mein do functions use horahe hai "grep" & "sed" ye dono ek directory mein define hote hai.
Us directory ki value $PATH mein store hoti hai.
Shell uses PATH variable to locate those then run them.
But if attacker manipulated $PATH value and put the directory were malicious code is defined then that code will get executed.

↳ To prevent this use absolute path for the functions like in 2nd example

↳ Or reset the value of $PATH variable before script execution

Date _____ Day _____

Ex #02:

```
#!/bin/bash
PATH = "/sbin:/bin:/usr/sbin:/usr/bin"
export PATH
user = `echo $1 | sed 's/@.*$//'`
grep $user /var/local/accounts/ipaddrs
```

↳ Here we have reset the PATH

Problem due to IFS variable:

↳ This variable tells shell that has to break a single line into words
default value "space", "tab", "newline"

If the default IFS value is used then the ~~above~~ cmd :

PATH = "_____" will simple run as an assignment command ~~because there are~~

↳ If IFS = "=" ⇒ (this tells the shell the break a line if "=" occurs

so now the cmd which gets executed as PATH "_____"

function args of that func

↳ Now attacker can define it's own function with name "PATH" & put it in the directory, so now the malicious code gets executed with root permission.

Date _____ Day _____

→ Recommended Prevention:
  ↳ Prefer changing group privileges, not superuser privileges
  ↳ Reset all critical env variables
  ↳ Use a compiled wrapper program to:
    ↳ Set safe env variables
    ↳ then execute the script
    ↳ Ex: Apache suexec wrapper

Ex #03 : Dynamic Linking Risk
  ↳ Most sys use dynamic linking for shared libraries
  ↳ LD_LIBRARY_PATH specifies library search directori
  ↳ Attacker can:
    - Provide malicious libraries
    - Force program to load them first
    - Execute attacker code with program privileges
  ↳ Mitigations
    - Use static linking (memory cost)
    - OS may block LB_LIBRARY_PATH for privilege programs

Date _____ Day _____

→ Using Appropriate, Least Privileges

↳ Poivilege escalation occurs when an attacker gains higher sys rights by exploiting a vulnerable privileged program.

↳ program should run with only the min poivileges req to perform their func

↳ Sometimes privileged programs run with elevated rights to access restricted sys resources

↳ Granting Group privileges preferred over user privileges

↳ User privilege hides the identity of the user

↳ Group privilege improve accountability

↳ Excessive file ownership incr. damage if a privilege program is compromised

↳ Web defacement usually occurs because web servers are given unecessary write permissions

↳ Root privileges should be used briefly & dropped as soon as possible.

↳ Modular design limits the impact of security breaches

↳ Sandboxing restricts compromised programs from affecting the entire system.

Date _____ Day _____

→ System Calls & Standard Library Functions

↳ Program use system calls & std library func for common operations

↳ Programmers make assumption about these operation

↳ Deleting a file normally doesnot erase data. It only ~~removed~~ detached filename & data block.

↳ Therefor it can be recovered by attackers

↳ To prevent this file should be overwritten multiple times with diff bit patterns before deletion.

```
patterns = [10010101O, 01011101O, ---]
open file for writting
for each pattern
    seek to start of file
    overwrite file contents with pattern
close file
delete file
```

(a) Initial file shredding program

↳

Vulnerabilities:

① Program assumes new data is written to same disk block In reality, the OS may allocate new blocks, leaving old data untouched

② Writting data does not immediately write to the disk. Data first stored in application buffers & may never reach disk unless ~~spe~~ explicitly flushed

Date _____ Day _____

③ Even after flushing application buffer, OS uses file sys buffers & may delay or reorder disk writes.

```
pattern = [10010110l, 00110111, ---]
open file for ~~writting~~ update
for each pattern
    seek to start of file
    overwrite file contents with pattern
    flush application write buffers
    sync file sys write buffers with device
close file
remove file
```

(b) Improved program

↳ Open the file for update, not just writting
Flush App buffer after each overwrite
Synchronize file sys buffers with the storage device

→ Further Problems

① Disk Controllers may discard repeated writes to same block

② Flash storage may write data to new blocks instead of overwritting

Remaing from Slides ☺