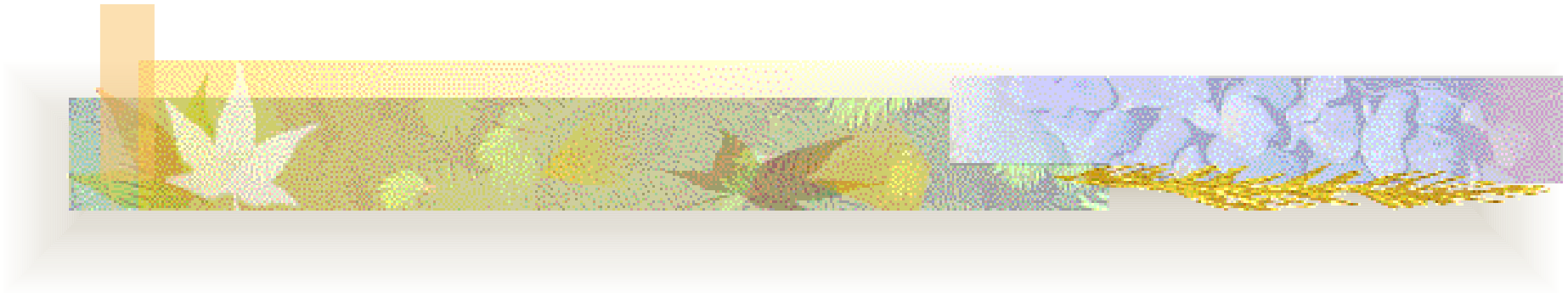
Guidelines



CSEN3103-Sec A, Prepared by Nilina Bera, CSE, HIT

Module-I:

Introduction and System Structure: Services and functions provided by operating systems, Types of OS (multitasking, multi user, real time, batch, interactive, time-sharing)

From **Galvin or related reference copy**: Systems calls, types of system calls, Operating system structure – monolithic and layered approach, microkernels.

Module-II:

Concept of a process, process states (life cycle), PCB, Threads, process scheduling, scheduling queues, scheduler, context switch, CPU-I/O burst cycle, You know and able to solve numerical problems from CPU scheduler – a) pre-emptive scheduling, FCFS, Shortest-job-first scheduling, SRTN, priority scheduling, RR. Draw the Gantt chart and find out average waiting time, average turnaround time and average weighted turnaround time for the scheduling algorithms as specified above.

Example:

Process	Burst Time (mills.)	Priority	Arrival Time (mills)
P1	9	5	0
P2	4	3	1
P3	5	1	2
P4	7	2	3
P5	3	4	4
Total	28		

Process synchronization: Slides discussion in class, Internal-I questions, From Galvin: the critical section problem, race condition, busy waiting, Peterson's solution

Peterson's Solution is a classical software based solution to the critical section problem.
In Peterson's solution, we have two shared variables:
boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
int turn : The process whose turn is to enter the critical section.

```
do {
        flag[i] = TRUE ;
        turn = j ;
        while (flag[j]  &&  turn == j) ;

            critial section

        flag[i] = FALSE ;

            remainder section

    } while (TRUE) ;
```

Peterson's Solution preserves all three conditions :

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson's Solution- a) It involves Busy waiting, b) It is limited to 2 processes.

Process Synchronization: Mutex locks, semaphores

What is Mutex?

The full form of Mutex is Mutual Exclusion Object. It is a special type of binary semaphore which used for controlling access to the shared resource. It includes a priority inheritance mechanism to avoid extended priority inversion problems. It allows current higher priority tasks to be kept in the blocked state for the shortest time possible. However, priority inheritance does not correct priority- inversion but only minimizes its effect.

Use of Semaphore

In the case of a single buffer, we can separate the 4 KB buffer into four 1 KB buffers. Semaphore can be associated with these four buffers. This allows users and producers to work on different buffers at the same time.

Use of Mutex

A mutex provides mutual exclusion, which can be either producer or consumer that can have the key (mutex) and proceed with their work. As long as producer fills buffer, the user needs to wait, and vice versa. In Mutex lock, all the time, only a single thread can work with the entire buffer.

Process Synchronization: Mutex locks, semaphores

Advantages of Semaphore: Here, are pros/benefits of using Semaphore:

- ✓ It allows more than one thread to access the critical section
- ✓ Semaphores are machine-independent.
- ✓ Semaphores are implemented in the machine-independent code of the microkernel.
- ✓ They do not allow multiple processes to enter the critical section.
- ✓ As there is busy waiting in semaphore, there is never a wastage of process time and resources.
- ✓ They are machine-independent, which should be run in the machine-independent code of the microkernel.
- ✓ They allow flexible management of resources.

Advantages of Mutex

- ✓ Here, are important pros/benefits of Mutex
- ✓ Mutexes are just simple locks obtained before entering its critical section and then releasing it.
- ✓ Since only one thread is in its critical section at any given time, there are no race conditions, and data always remain consistent.

Classic problems of synchronization - The Bounded-Buffer problem, The Readers-Writers problem, The Dining-Philosophers Problem.
Monitors

Deadlocks and starvation – slides + Galvin, Banker's Algorithm (numericals – Discussed in class in detail, very important for semester. You MUST understand how to identify if a set of processes with different allocation requests would operate in safe state. If not, why they would fall into deadlock? Example is shown in next slides:

- 5 processes P_0 through P_4; 3 resource types A (10 instances), B (5instances, and C (7 instances).
- Snapshot at time T_0:

	Allocation	Max	Need	Total
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	7 4 3	10 5 7
P_1	2 0 0	3 2 2	1 2 2	Allocated
P_2	3 0 2	9 0 2	6 0 0	7 2 5
P_3	2 1 1	2 2 2	0 1 1	Available
P_4	0 0 2	4 3 3	4 3 1	3 3 2

- The system is in a safe state since the sequence $< P_1, P_3, P_4, P_2, P_0>$ satisfies safety criteria

- Check that Request ≤ Available
  that is, (1, 0, 2) ≤ (3, 3, 2) ⇒ true.

	Allocation	*Max*	*Need*	*Available*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 3	7 4 3	**2 3 0**
P_1	2 0 0	3 2 2	**0 2 0**	
P_2	3 0 2	9 0 2	6 0 0	
P_3	2 1 1	2 2 2	0 1 1	
P_4	0 0 2	4 3 3	4 3 1	

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

Module-III: Storage Management (Memory + Virtual Memory), File Systems, I/O, Disk Management: **Slides as discussed in class (numericals)**, **Internal-II questions + Internal-II Guidelines** + Galvin.

- ✓ Hardware address protection with base and limit registers
- ✓ Address binding (compile time, load time, execution time)
- ✓ logical vs physical address space (need to know memory-address register, memory-management unit [MMU], relocation register)
- ✓ dynamic loading, dynamic linking
- ✓ swapping
- ✓ contiguous memory allocation → memory protection, memory allocation, fragmentation. Internal and external memory fragmentations. When is an external fragmentation said to occur? Can we use compaction to solve internal fragmentation problem? Justify your answer. (Hint: No, because internally fragmented space cannot be used by other processes.)

Suppose by mistake, during the processor design and development, the MMU forgot to check the segment limit. Explain some consequences of this defect. (Hint: A process can overwrite the physical memory allocated to others. If the kernel does not reside at the lower memory, applications can corrupt the kernel too.)

- ✓ Segmentation
- ✓ Paging

Module-III (continued)

- ✓ Virtual address space, demand paging, Page replacement (FIFO, LRU, Optimal – assignments given in slides + Internal-II), Thrashing, Page fault.
- ✓ Is there any difference between external and internal fragmentation?
- ✓ What is pure demand paging?

S.NO	INTERNAL	
1.	In internal fragm memory, blocks appointed to pro	zed memory method.
2.	Internal fragmer the method or p the memory.	n the method
3.	The solution of i fragmentation is	compaction,
4.	Internal fragmer memory is divid partitions.	memory is ased on the
5.	The difference b allocated and re memory is called internal fragmentation.	non- oo small to serve a new process, is called External fragmentation .

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

In pure demand paging, even a single page is not loaded into memory initially. Hence pure demand paging causes a page fault

When starting execution of a process with no pages in memory, the operating system sets the instruction pointer to the first instruction of the process, which is on a non-memory resident page, the process immediately faults for the page. After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This schema is pure demand paging.

Module-IV: Security + Protection – Slides sent + Internal-II question + guidelines given as follows:

- ✓ Concepts of Bug of a program vs flaw of the same program [use you experience to define both bug and flaw], OTP, Trap door, Logic bomb, Trojan Horse, Virus, Worm, Zombie.
- ✓ What is man-in-the middle attack? What computer programs are vulnerable to this kind of attack?
- ✓ How is malicious behaviour different from incidental behaviour? [Hint: Malicious behaviour attempts to read or destroy sensitive data, or disrupt the system operation intentionally. Incidental behaviours need not be intentional and usually arise from within the system, but their consequences are as serious as those of malicious behaviours are]
- ✓ What is security? How is it different from protection? Explain some goals of computer security. [Hint: Security is a set of policies to ensure a safe and trusted environment for computer users, and protection is a collection of mechanisms to implement the security policies in the system. In essence, security is an end goal and protection is a means to attain that goal.]
- ✓ In a UNIX system, suppose a process first executes a new program P1 (of owner U1) whose *setuid bit* is set. The process then executes another program P2 of that owner. What would be the *euid* of the process at that point if P2 (a) has or (b) has not *setuid bit* on? [Hint: (a) euid is the uid of U1; (b) euid is the uid of the process owner.]

Module-IV: Security + Protection

- ✓ In UNIX systems, how can we allow a group of users to read and write a particular file, but they cannot delete the file? Rest of users of the system can only read the file.

[Hint: The file will have only read and write permission for the group, and only read permission for the universe. The container directory will not have write permission for the group and the universe.]

- ✓ When can we say a system is secured?

[Hint: If the system can guarantee that all its resources are always used as per their specifications, then the system is secured.]

- ✓ In the domain of security, what are threat and attack?

[Hint: A threat is a potential for the possibility of a security violation. An attack is an attempt to break a security protection.]

Module-IV: Security + Protection

✓What are the goals of protection? Explain how access matrix can be implemented using global table and access list.

[Hint: discussed in slide → 14.CSEN3103-Protection-n-Security-to-students..]

Useful link for access matrix, global table and access list:

https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/14_Protection.html

✓ For each of the following security attacks, say if public key encryption can help prevent the attack. (Justify your answer.) [Question is given in Internal-II, I have tried to get some links on these attacks and public key cryptography. Please try to search on net of public key infrastructure can prevent each of the attacks mentioned below.]

i.Abuse of valid privileges

ii.Denial of Service attack (https://ieeexplore.ieee.org/document/995148 --> public key infrastructure can prevent DoS attack)

iii.Listener or eavesdropper attack

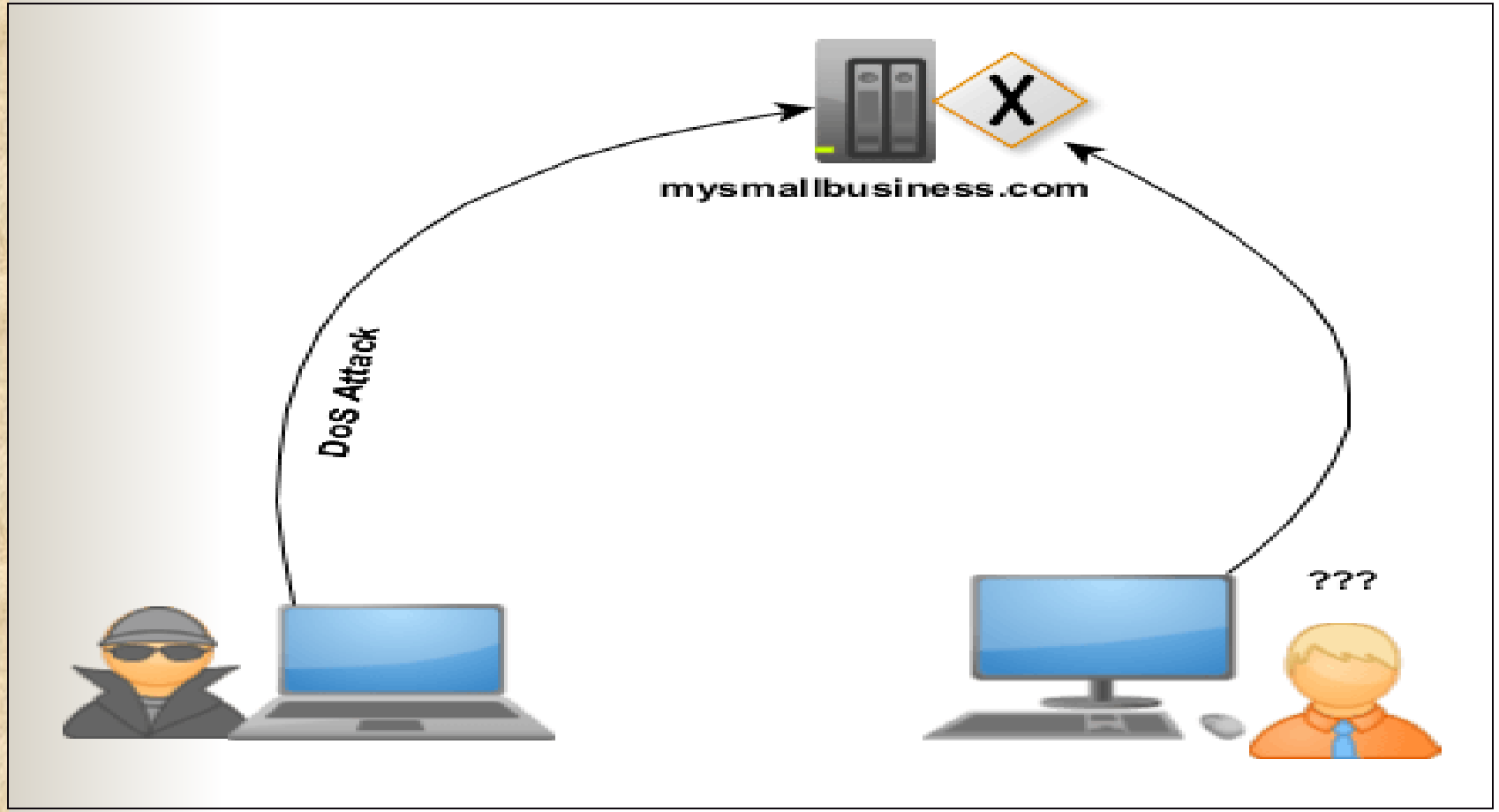
iv.Buffer overflow attack

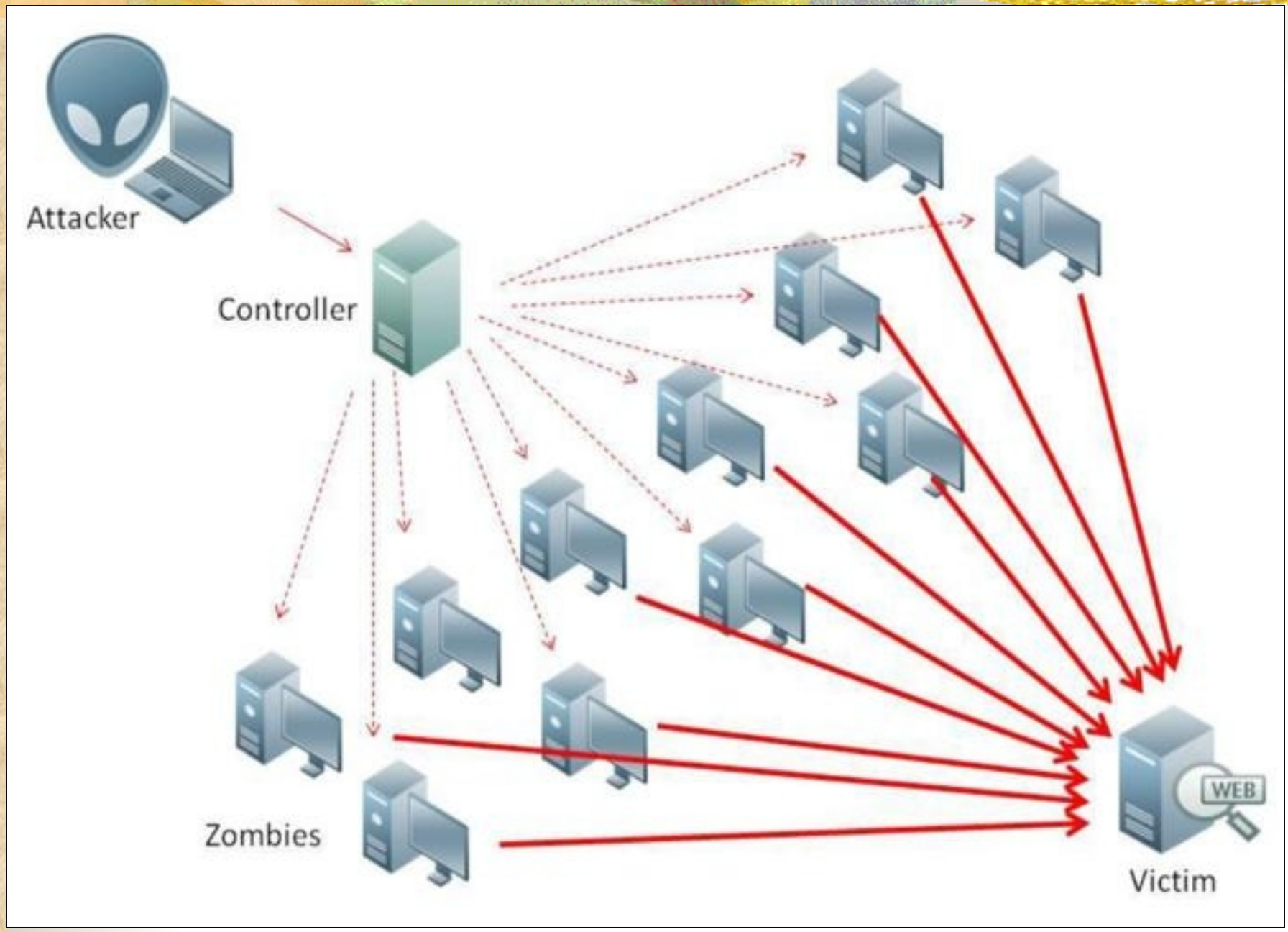
Module-IV: Abuse of valid privileges

- ✓Abuse of privilege attacks include any attack in which a person, company, or organization abuses the access privileges you have given them in order to steal data. This abuse of trust results in someone using valid login credentials to access data that they shouldn’t, or to access and distribute data against your wishes.
- ✓Abuse of privilege attacks can happen as an inside attack (disgruntled employee), outside attack (hacker obtains your login credentials), or through a company that you are working with. For example, if you are using a cloud service, and the cloud service is compromised, someone could access your data and information.
- ✓Abuse of privilege attacks are difficult to protect against, but you can work to defend yourself by choosing trading and technological partners wisely, vetting employees, and carefully controlling who has access to critical information.

Module-IV: Denial of Service attack

- A **denial-of-service** (**DoS**) is any type of **attack** where the attackers (hackers) attempt to prevent legitimate users from accessing the **service**.
- In a **DoS attack**, the attacker usually sends excessive messages asking the network or server to authenticate requests that have invalid return addresses.
- Useful link: https://hub.packtpub.com/10-types-dos-attacks-you-need-to-know/




Attacker
Controller
Zombies
WEB
Victim

Module-IV: Listener or eavesdropper attack

Eavesdropping attack, also known as sniffing or snooping **attack**, happens when an unauthorized party steals, modifies or deletes essential information that is transmitted between two electronic devices.

https://www.ilantus.com/blog/eavesdropping-attack-ensure-they-dont-listen-to-you/

Module-IV: Buffer overflow attack

A **buffer** is a temporary area for data storage. When more data (than was originally allocated to be stored) gets placed by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding.
In a **buffer-overflow attack,** the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.
Attacker would use a buffer-overflow exploit to take advantage of a program that is waiting on a user's input. There are two types of buffer overflows: stack-based and heap-based. Heap-based, which are difficult to execute and the least common of the two, attack an application by flooding the memory space reserved for a program. Stack-based buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack: memory space used to store user input.

Module-IV: public key encryption

- ✓ Public-key encryption is a cryptographic system that uses two keys -- a *public key* known to everyone and a *private* or *secret key* known only to the recipient of the message.
- ✓ ***Example:*** *When John wants to send a secure message to Jane, he uses Jane's public key to encrypt the message. Jane then uses her private key to decrypt it.*
- ✓ An important element to the public key system is that the public and private keys are related in such a way that *only the public key can be used to encrypt messages* and *only the corresponding private key can be used to decrypt them*. Moreover, it is virtually impossible to deduce the private key if you know the public key.
- ✓ **Public key cryptography** remains the most secure protocol (over **private key cryptography**) because users never need to transmit or reveal their **private keys** to anyone, which lessens the chances of **cyber** criminals discovering an individual's secret **key** during the transmission.