**Types of attacks**

1. Malware: Malware breaches a network through a vulnerability, typically when a user clicks a dangerous link or email attachment that then installs risky software.
2. Phishing: Phishing is the practice of sending fraudulent communications that appear to come from a reputable source, usually through email. The goal is to steal sensitive data like credit card and login information or to install malware on the victim’s machine. Phishing is an increasingly common cyber threat.
3. Man in the middle attack: Man-in-the-middle (MitM) attacks, also known as eavesdropping attacks, occur when attackers insert themselves into a two-party transaction. Once the attackers interrupt the traffic, they can filter and steal data.
4. Denial of service attack: A denial-of-service attack floods systems, servers, or networks with traffic to exhaust resources and bandwidth. As a result, the system is unable to fulfill legitimate requests. Attackers can also use multiple compromised devices to launch this attack. This is known as a [distributed-denial-of-service (DDoS) attack](https://www.cisco.com/c/en/us/products/security/what-is-a-ddos-attack.html).
5. SQL Injection: A Structured Query Language (SQL) injection occurs when an attacker inserts malicious code into a server that uses SQL and forces the server to reveal information it normally would not. An attacker could carry out a SQL injection simply by submitting malicious code into a vulnerable website search box.
6. DNS Tuning: DNS tunneling utilizes the DNS protocol to communicate non-DNS traffic. It sends HTTP and other protocol traffic over DNS. For malicious use, DNS requests are manipulated to ex filtrate data from a compromised system to the attacker’s infrastructure. It can also be used for command and control callbacks from the attacker’s infrastructure to a compromised system.

**INTERFACE**

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [abstraction](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Jav](https://www.javatpoint.com/inheritance-in-java)a

It is used to achieve total abstraction. Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance. We can not create object of interface

interface printable{

void print();

}

class A6 implements printable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

A6 obj = new A6();

obj.print();

}

}

**LAMBDA**

A Java lambda expression is thus a function which can be created without belonging to any class. A Java lambda expression can be passed around as if it was an object and executed on demand.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

**Interface** Drawable{

**public** **void** draw();

}

**public class** LambdaExpressionExample2{

**public** **static** **void** main(String[] args){

**int** width=10;

//with lambda

Drawabled2=()->{

System.out.println("Drawing"+width);

};

d2.draw();

}

}

**THIS** keyword

Using ‘this’ keyword to refer current class instance variables. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

**class**A{

**void** m(){System.out.println("hello m");}

**void** n(){

System.out.println("hello n");

//m();//sam as this.m()

**this**.m();

}

}

**class** TestThis4{

**public static void** main(String args[]){

A a=**new** A();

a.n();

}}