**Hackers Types**

**1.White Hat Hackers**

White hat hackers are types of hackers who’re professionals with expertise in cybersecurity. They are authorized or certified to hack the systems. These White Hat Hackers work for governments or organizations by getting into the system. They hack the system from the loopholes in the cybersecurity of the organization

**2.Black Hat Hackers**

Black hat hackers are also knowledgeable computer experts but with the wrong intention. They attack other systems to get access to systems where they do not have authorized entry. On gaining entry they might steal the data or destroy the system

**3.Grey Hat Hackers**

The intention behind the hacking is considered while categorizing the hacker. The Gray hat hacker falls in between the black hat hackers and white hat hackers. They are not certified, hackers. These types of hackers work with either good or bad intentions. The hacking might be for their gain.

**4.Green Hat Hackers**

Green hat hackers are types of hackers who’re learning the ropes of hacking. They are slightly different from the Script Kiddies due to their intention. The intent is to strive and learn to become full-fledged hackers. They are looking for opportunities to learn from experienced hackers.

**5.Blue Hat Hackers**

Blue Hat Hackers are types of hackers who’re similar to Script Kiddies. The intent to learn is missing. They use hacking as a weapon to gain popularity among their fellow beings. They use hacking to settle scores with their adversaries. Blue Hat Hackers are dangerous due to the intent behind the hacking rather than their knowledge.

**6.Red Hat Hackers**

Red Hat Hackers are synonymous with Eagle-Eyed Hackers. They are the types of hackers who’re similar to white hackers. The red hat hackers intend to stop the attack of black hat hackers. The difference between red hat hackers and white hat hackers is in the process of hacking through intention remains the same. Red hat hackers are quite ruthless while dealing with black hat hackers or counteracting with malware. The red hat hackers continue to attack and may end up having to replace the entire system set up.

**Attacks**

### **1. Malware**

Some forms of malware are designed to extort the victim in some way. Perhaps the most notable form of malware is Ransomware – a program designed to encrypt the victim’s files and then ask them to pay a ransom in order to get the decryption key.

### **2. Phishing**

A Phishing attack is where the attacker tries to trick an unsuspecting victim into handing over valuable information, such as passwords, credit card details, intellectual property, and so on.

### **3. Man-in-the-middle attack (MITM)**

A man-in-the-middle attack (MITM) is where an attacker intercepts the communication between two parties in an attempt to spy on the victims, steal personal information or credentials, or perhaps alter the conversation in some way.

### **4.Drive-by Attack**

A ‘drive-by-download’ attack is where an unsuspecting victim visits a website which in turn infects their device with malware. The website in question could be one that is directly controlled by the attacker, or one that has been compromised.

### **5. Password Attack**

A password attack, as you may have already guessed, is a type of cyber-attack where an attacker tries to guess, or “crack” a user’s password. There are many different techniques for cracking a user’s password, although an explanation of these different techniques is beyond the scope of this article.

### **6. Zero-day exploit**

A zero-day exploit is where cyber-criminals learn of a vulnerability that has been discovered in certain widely-used software applications and operating systems, and then target organizations who are using that software in order to exploit the vulnerability before a fix becomes available.

**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. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

**Syntax**

**interface** <interface name>

{

// declare constant fields

// declare methods that abstract

// by default.

}

**Lambda Expression in Java**

Lambda expression provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

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.

Java lambda expression is treated as a function, so compiler does not create .class file.

Java lambda expression is consisted of three components.

**1) Argument-list:** It can be empty or non-empty as well.

**2) Arrow-token:** It is used to link arguments-list and body of expression.

**3) Body:** It contains expressions and statements for lambda expression.

# **#Pragma Directive in C/C++**

This directive is a special purpose directive and is used to turn on or off some features. This type of directives is compiler-specific i.e., they vary from compiler to compiler. Some of the #pragma directives are discussed below:

**1.#pragma startup and #pragma exit**:

These directives help us to specify the functions that are needed to run before program startup (before the control passes to main ()) and just before program exit (just before the control returns from main ()).

**2.#pragma warn Directive**:

This directive is used to hide the warning messages which are displayed during compilation. This may be useful for us when we have a large program and we want to solve all the errors before looking on warnings then by using it we can focus on errors by hiding all warnings. we can again let the warnings be visible by making slight changes in syntax.

**3.#pragma GCC poison**:

This directive is supported by the GCC compiler and is used to remove an identifier completely from the program. If we want to block an identifier then we can use the **#pragma GCC poison** directive.

4. **#pragma GCC dependency**:

The #pragma GCC dependency allows you to check the relative dates of the current file and another file. If the other file is more recent than the current file, a warning is issued. This is useful if the current file is derived from the other file, and should be regenerated.

5. **#pragma GCC system header**:

This pragma takes no arguments. It causes the rest of the code in the current file to be treated as if it came from a system header.

6. **#pragma once**:

The #pragma once directive has a very simple concept. The header file containing this directive is included only once even if the programmer includes it multiple times during a compilation. This is not included in any ISO C++ standard. This directive works similar to the #include guard idiom. Use of #pragma once saves the program from multiple inclusion optimization.