Experiment 8

Learning Objective: Write a program to implement Salami Attack and provide a condition to prevent it.

Tools: C/C++/Java/Python

Theory:

A salami attack merges bits of seemingly inconsequential data to yield powerful results. For example, programs often disregard small amounts of money in their computations, as when there are fractional pennies as interest or tax is calculated. Such programs may be subject to a salami attack, because the small amounts are shaved from each computation and accumulated else where such as in the programmer's bank account.

Examples of Salami Attacks

The classic tale of a salami attack involves interest computation. Suppose your bank pays 6.5 percent interest on your account. The interest is declared on an annual basis but is calculated monthly. If, after the first month, your bank balance is $102.87, the bank can calculate the interest in the following way. For a month with 31 days, we divide the interest rate by 365 to get the daily rate, and then multiply it by 31 to get the interest for the month. Thus, the total interest for 31 days is 31/365*0.065*102.87 = $0.5495726. Since banks deal only in full cents, a typical practice is to round down if a residue is less than half a cent and round up if a residue is half a cent or more. However, few people check their interest computation closely, and fewer still would complain about having the amount $0.5495 rounded down to $0.54, instead of up to $0.55. Most programs that perform computations on currency recognize that because of rounding, a sum of individual computations may be a few cents different from the computation applied to the sum of the balances.

What happens to these fractional cents? The interest program merely had to balance total interest paid to interest due on the total of the balances of the individual accounts.

Why Salami Attacks Persist

Computer computations are notoriously subject to small errors involving rounding and truncation, especially when large numbers are to be combined with small ones. Rather than document the exact errors, it is easier for programmers and users to accept a small amount of error as natural and unavoidable. To reconcile accounts, the programmer includes an error correction in computations. Inadequate auditing of these corrections is one reason why the salami attack may be overlooked.

Usually the source code of a system is too large or complex to be audited for salami attacks, unless there is reason to suspect one. Size and time are definitely on the side of the malicious programmer.

Implementation:

```
public class salami {
   public static void main(String[] args) {
      double eve, bob;
      bob = 45200.75;
      System.out.println("amount before interest is credited " + bob);
      double interest = 0.06 * bob;
      bob += interest;
      System.out.println("amount after interest is credited " + bob);
      eve = bob - (int) bob;
      bob = (int) bob;
      System.out.println("Eve gets " + eve);
      System.out.println("Balance after attack " + bob);
   }
}
```

```
amount before interest is credited 45200.75
amount after interest is credited 47912.795
Eve gets 0.7949999999982538
Balance after attack 47912.0
```

Learning Outcomes: The student should have the ability to

LO1: Design preventive measure for Salami Attack

LO2: Implement Salami Attack

LO3: Understand how to execute Salami Attack and prevent it

Course Outcomes: Upon completion of the course students will be able to understand about Salami Attack and it's prevention.

Conclusion: Through this experiment we learned how Salami Attack is executed and how we can prevent it.

Name:

Class: SE-CSE

Roll No.:

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				