Advanced Programming 2020 – 2021

Programming Assignment

January 19, 2021

Introduction

There is always a chance of hacked nodes being present in a computer network. These nodes can attack the network in different ways. For example, there are passive attacks in which the data that is sent over the network is sniffed (aka eavesdropping). Traffic sniffing is especially a problem in wireless networks where malicious nodes have easy access to the network (as opposed to wired networks). An active and more aggressive attack can be the modification of the content of a data packet. In such an attack, the content of the received packets can no longer be trusted. There are dozens of attacks that frequently crop up in computer network literature: jamming, black hole attack, sink hole attack, replay attack, flooding, …

Assignment

In this assignment, you are expected to program a hacked access point that has a certain probability of attacking the packets that it receives. In this simulation, you will use three distinct types of packets. A packet can contain different types of data: booleans, integers, chars, etc. The access point is infected with two different attacks:

- Sniffing attack: the contents of the packet are monitored. In this simulation, this means that the contents of the packet are written to the screen;
- Modification attack: the contents of the packet are modified;

In the modification attack, the following happens with the different types of packets:

- Boolean packet: the boolean value is inverted;
- Integer packet: the sign of the integer is reversed;
- Any other type of packet: the contents of the packet are written to the screen and nothing is modified;

Implementation

The following is required for enabling an extensible and flexible implementation of this simulation:

- There is a Packet<T> class that represents the packet. The contents of the packet can be printed with the << operator. *Hint: The easiest way to overload an operator into a templated class is to put the method body in the class definition*.

- There are two different attacks that can be performed in the node. So you need an abstract base class Attack<T> with the following derived classes for the specific attacks:

    - SniffingAttack<T> which writes the type and data of the Packet to the screen

    - ModificationAttack<T> which changes the data of the Packet depending on the type (i.e., see above) and also writes this change to the screen

  When the Attack<T> type call operator is applied to a Packet, the attack is executed.

- The infected access point is represented by the AccessPoint<T, SIZE> class where the SIZE represents the maximum size of Packet<T> queue. The two possible attacks are created at the initialization of the node and kept in an appropriate container. The AccessPoint<T, SIZE> is initialized with the probability of a packet being attacked. A packet is therefore only attacked with a certain probability. When an attack is performed on a packet, a random choice is made between the two attacks.

  A packet can be added to the AccessPoint<T, SIZE> at any time via the << operator (see example below).

- Two alias declarations up_packet<T> and up_attack<T> must be used throughout the program. The first represents a std::unique_ptr to a Packet<T>, the second is similar but refers to an attack.

- Define a print function that will be passed to the run method of the access point. The function is executed just before the attack is applied and prints the number of packets involved (note: this must be kept in the function!) and the original value of the packet. See the main and terminal example for more information.

Example

The implementation of the individual classes should allow the following code in the main function (it should not be changed):

```
using namespace std;

...
void print (...) {
	...
}

int main() {
	try {
		// Initialize with 0.8 attack probability.
		network::AccessPoint<int, 5> ap(0.8);

		// Infect the access point.
		ap.infect(network::up_attack<int>(new
network::ModificationAttack<int>()));
		ap.infect(network::up_attack<int>(new network::SnifferAttack<int>()));

		// Add the packets.
		ap << network::up_packet<int>(new network::Packet<int>(1.0));
		ap << network::up_packet<int>(new network::Packet<int>(2.0));
		ap << network::up_packet<int>(new network::Packet<int>(3.0));
		ap << network::up_packet<int>(new network::Packet<int>(4.0));
		ap << network::up_packet<int>(new network::Packet<int>(5.0));

		// Run, possibly attack the packets.
		ap.run(print <int >);
	} catch (exception &e) {
		std::cout << "Fatal error: " << e.what() << std::endl;
	}

	return 0;
}
```

You can also find this main code in the main.cpp file on Blackboard. In that file you will also find example of `bool` and another type of package. Your program must work for all those types, according to the specification.

Output

The output of the program should look something like this:

```
Added packet with value 1 to access point.
Added packet with value 2 to access point.
Added packet with value 3 to access point.
Added packet with value 4 to access point.
Added packet with value 5 to access point.
Packet 0 being handled has value 1.
Packet 1 being handled has value 2.
```

```
Packet 2 being handled has value 3.
Packet 3 being handled has value 4.
Modified int packet now has value −4.
Packet 4 being handled has value 5.
Sniffed int packet has value 5.
```

Remarks

Pay attention to the following (very important) things:

- Implement “minimal” code to build the structure of the discussed entities. So use your time to implement the requested features first. Things that you do not have to write (e.g., unnecessary constructors, methods synthesized by the compiler, other methods not used, …) are best omitted.

- Use exception handling for error handling. Among other things, you must be able to handle the following errors (the error message can be used as a string in the *_error):

    - runtime_error: More packages are added to the node than allowed

    - logic_error: When the access point is running, no attacks are present

- “Package” the classes in the network namespace.

- Use the standard library as needed, useful and / or highly recommended.

- Provide your code with clear but concise comments (in English!!) and **write your name and role number in each file**.

Good luck!