APÉNDICE B

CÓDIGO DE IMPLEMENTACIÓN DE LOS ATAQUES EN OMNET++

OMNET++ es un simulador de redes cuya plataforma de programación para los elementos que forman parte de la misma, es C++, dicha plataforma la establece uno de los programas complementarios del simulador, VisualStudio 2008 de Microsoft.

A continuación se describirá el proceso de programación para la implementación de los ataques en la simulación.

*Inicialmente se declaran las librerias ***stdio.h, string.h, omnetpp.h*** y ***MessageTesis_m.h*** las cuales contienen la información necesaria para crear el tráfico en la red.

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "MessageTesis_m.h"
```

*Se crea la clase SourceBase2 como el módulo en el cual se generará el tráfico, declarando inicialmente los mensajes que contienen los ataques ***ping, dos, ddos, xss y mitm***. Dichas variables se declaran como enteros para que posteriormente sean contados durante el registro de los eventos y posteriormente se grafiquen en las estadísticas. Dicha rutina es programada para los dos generadores de ataques.

```
class SourceBase2 : public cSimpleModule
{
      protected:
            std::string jobName;
            std::string ip;
            int pingCounter;
            int dosCounter;
            int ddosCounter;
            int xssCounter;
            int mitmCounter;

            int attackCounter;

            bool is_ping;
            bool is_dos;
            bool is_ddos;
            bool is_xss;
            bool is_mitm;

      protected:
            virtual void initialize();
            virtual MessageTesis *generateMessage();
```

```
            virtual void finish();
};

class Source_Attack2 : public SourceBase2
{
    private:
        simtime_t startTime;
        simtime_t stopTime;
        int numAttacks;
    protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};
```

*Se crea e inicializa un Segundo SourceBase en el cual se declaran los ataques como parámetros, lo que permitirá establecer una comparación con los mensajes que contiene datos normales (***Data***).

```
void SourceBase2::initialize()
    {
        is_ping = par("isPing");
        is_dos = par("isDos");
        is_ddos = par("isDdos");
        is_xss = par("isXSS");
        is_mitm = par("isMitM");

        pingCounter = 0;
        dosCounter = 0;
        ddosCounter = 0;
        xssCounter = 0;
        mitmCounter = 0;
        attackCounter = 0;

        jobName = par("jobName").stringValue();
        ip = par("ip").stringValue();
        if (jobName == "")
            jobName = getName();
    }
```

*Para la creación de mensajes se estructura una rutina en SourceBase2, la cual permitirá establecer al usuario las características de generación de los ataques, ya sea de manera infinita (-1) o estableciendo un número entero positivo, así como la activación de cada ataque (***true o false***).

```
MessageTesis *SourceBase2::generateMessage()
{
    bool valido = true;
    int type_msg = -1;
    int dest_msg = -1;
    std::string destination_ip;
    char msgname[20] = "";

    do
```

```
{
  type_msg = intuniform(0,4);
  switch(type_msg)
  {
              case 0:
                    if(is_ping)
                          valido = false;
                    break;
              case 1:
                    if(is_dos)
                          valido = false;
                     break;
              case 2:
                    if(is_ddos)
                          valido = false;
                     break;
              case 3:
                    if(is_xss)
                          valido = false;
                     break;
              case 4:
                    if(is_mitm)
                          valido = false;
                    break;
  }
}
while (valido);

switch(type_msg)
```

*Una vez establecidos los valores para la generación de los ataques, se establece una serie de conmutaciones (switch-break) para imprimir en la onomatopeya que el ataque llegó a su destino, esto se observa de manera gráfica durante la simulación.

```
{
        case 0:
              sprintf(msgname, "Ping Muerte 65.535");
              pingCounter++;
              break;
        case 1:
              sprintf(msgname, "DOS");
              dosCounter++;
              break;
        case 2:
              sprintf(msgname, "DDoS");
              ddosCounter++;
              break;
        case 3:
              sprintf(msgname, "XSS");
              xssCounter++;
              break;
        case 4:
              sprintf(msgname, "MitM");
              mitmCounter++;
              break;
}
```

*Debido a que cada submódulo o elemento establecido en la red cuenta con una IP, se declaran las direcciones a las cuales los ataques pueden tomar como destino, de manera que

se ejecute el ataque. Dichas direcciones se pueden cotejar con el direccionamiento de la figura 3.1.1 del capitulo 4.

```
        dest_msg = intuniform(0,11);

        switch(dest_msg)
        {
                case 0:
                destination_ip = "10.24.150.29";
                break;
          case 1:
                destination_ip = "10.24.150.31";
                break;
          case 2:
                destination_ip = "10.24.147.19";
                break;
          case 3:
                destination_ip = "10.24.147.20";
                break;
          case 4:
                destination_ip = "10.24.147.21";
                break;
          case 5:
                destination_ip = "10.24.147.22";
                break;
          case 6:
                destination_ip = "10.24.148.13";
                break;
          case 7:
                destination_ip = "10.24.148.15";
                break;
          case 8:
                destination_ip = "10.24.148.16";
                break;
          case 9:
                      destination_ip = "10.24.150.12";
                      break;
                case 10:
                      destination_ip = "10.24.150.13";
                      break;
                case 11:
                      destination_ip = "10.24.150.15";
                      break;
        }

        MessageTesis *msg = new MessageTesis(msgname);
        msg->setIp_source(ip.c_str());
        msg->setIp_destination(destination_ip.c_str());
        return msg;}
    void SourceBase2::finish()
    {

    }

    Define_Module(Source_Attack2);
```

*Cuando los ataques han llegado a su destino comienza el conteo de los mismos parámetros, para que más adelante en los resultados se puedan observar en el tiempo en que fueron

generados los ataques "***StartTime***", El registro del tiempo si la simulación se detiene con "***stopTime***" y el conteo de los número de ataques en cada uno de los elementos que fue afectado por alguno de ellos con "***numAttacks***".

```
void Source_Attack2::initialize()
      {
            SourceBase2::initialize();
            startTime = par("startTime");
            stopTime = par("stopTime");
            numAttacks = par("numAttacks");

            scheduleAt(startTime, generateMessage());
      }

      void Source_Attack2::handleMessage(cMessage *msg)
      {
            if ((numAttacks < 0 || numAttacks > attackCounter) && (stopTime
< 0 || stopTime > simTime()))
            {
                  scheduleAt(simTime() +
par("interArrivalTime").doubleValue(), msg);
                  MessageTesis *msg = generateMessage();
                  EV << "Se creo el ataque:  " << attackCounter << " de
tipo: " << msg -> getName() << "\n";
                  send(msg, "out");
                  attackCounter++;
            }
      }
```

B1.1 Snort Rule del IDS diseñado

Como se menciona anteriormente, la programación para realizar la simulación de la red establecida es en base a la plataforma C++. A continuación se muestra el diseño de la regla de Snort para el IDS diseñado a través del este lenguaje de programación.

```
void IDS::initialize()
```

Encabezado de la regla (Rule Header)	Opciones de regla (Rule options)

`alert tcp any !IP address->`
`(content: "filename=\"PING 65.535 Bytes\;\`
`nocase; msg: "Ping de la muerte";)`
	`check_and_cast<Message.`
	`1. Block message from IP address.`
	`2.`
	`if((strcmp("DOS", Ping 6.535Bytes, DDoS, Xss, ttmsg -> getName())==0)&& blockDos, Ping, DDoS, Xss)`
	`{`
	`EV<<"El IDS detuvo un ataque DOS";`
	`conDos = conDos + 1;`
	`bubble("DOS detenido");`
	`delete msg;`
	
	`2. msg->setHopCount->Pasa el dato }`

```
      {
            blockDos = par("blockDos");
            blockDdos  = par("blockDdos");
            blockPing = par("blockPing");
            blockXSS = par("blockXSS");
```