**MIB 구조를 이용한 패킷 캡쳐 프로그래밍 응용**

소프트웨어학과 201621654 박민지

소프트웨어학과 201720768 김수영

소프트웨어학과 201821225 구교현

**1.설계 목적**

본 보고서는 MIB 구조를 이용한 패킷 캡쳐 응용을 설계한 내용이다. MIB는 Management Information Base의 약자로서 망관리를 위해 사용되는 체계화된 관리 정보인데 자신의 장치의 상태와 네트워크 장치를 통해 받는 통계정보를 MIB이라는 트리 형태로 관리한다. 따라서 본 설계를 통하여 지금까지 수업과 실습에서 배워왔던 패킷 캡쳐와 생성에 대하여 복습하고 이를 응용하여 Socket 기반 Client-Server 프로그램을 연계 응용하여 네트워크 프로그래밍에 대한 이해를 높이고자 한다.

**2.시스템 구조 #1**

**2.1 프로토콜**

1)Syntax

**UDP (TCP CO connection 이전에 이뤄져야 하는 syntax)**

(1) agent UDP client 가 manager UDP server 에게 자신의 정보를 전달하는 syntax:

|  |
| --- |
| typedef struct UDP\_PROTO {  int type;  int port;  }UDP\_PROTO; |

새롭게 선언된 UDP\_PROTO 구조체 변수에 agent UDP client 의 코드에서 사용할 port number 정보를 (int)형식으로 그대로 받아 저장하고, 일반 연결을 의미하는 ‘1’ 숫자를 type 값에 넣어 구성을 한 이후에 이 구조체를 가리키는 구조체 포인터를 문자열로 변환한다. 이 문자열을 UDP\_PROTO 구조체 크기만큼 잘라 syntax를 구성한다.

(2) manager UDP server가 agent UDP client에게 잘 받았다고 전달하는 확인 syntax:

|  |
| --- |
| #define BUFSIZE 1500  char buf[BUFSIZE + 1]; //이 형식 계속 사용  ======================  sprintf(buf, "check\0"); |

“check\0” 문자열이 담긴 char[] 형태의 buf를 buf의 길이만큼 syntax 구성한다.

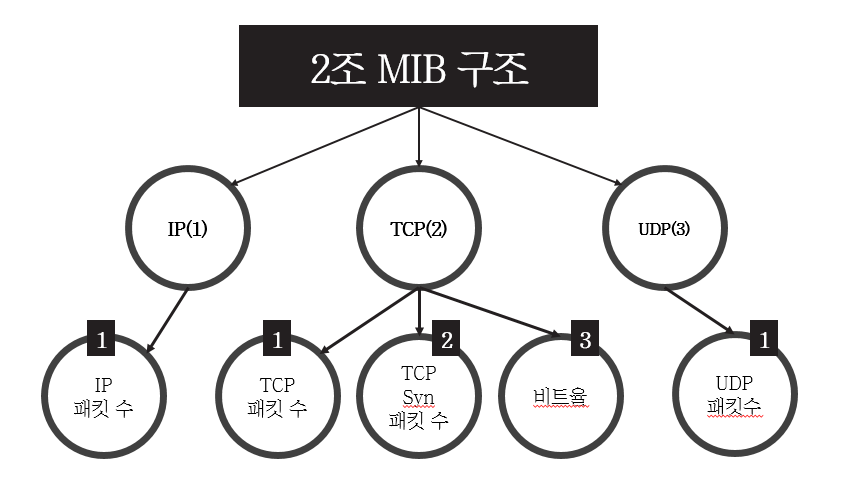
**TCP**

(1) manager TCP client 에서 agent TCP server로 보내는 syntax:

|  |
| --- |
| sprintf(buf, "request\0"); |

manager TCP client가 agent TCP server의 mib 정보를 얻어오기 위해서 요청하는 syntax로 “request\0”문자열이 담긴 char[] 형태의 buf를 buf의 길이만큼 syntax로 구성한다.

(2) agent TCP server가 mib tree 정보를 변형된 SNMPv2 PDU Format형태로 manager TCP client에게 보내는 syntax:



***[표1] 자체적으로 구성한 MIB 트리 구조***

1.1.1 : 총 capture 된 IP packet 수

1.2.1 : 총 capture 된 TCP packet 수

1.2.2 : agent에게 온 TCP Syn packet 수 (syn flooding 감지를 위해서)

1.2.3 : 비트율 (총 capture 된 전체 packet 수 / 5)(5초에 한번씩 snmp manager가 request)

1.3.1 : 총 capture 된 UDP packet 수

Manager와 Agents간의 핵심 동작을 설명하기 위해 팀에서 자체적으로 구성한 MIB 구조를 설명해보고자 한다. 위와 같은 구조는 Manager가 Agent의 상태를 알기 위하여 Agents에게 보낸 “Request”에 대한 정보 응답에 이용했다. Agents들이 Manager에게 전송하는 정보는 5개로 구성을 했는데 IP 패킷 수, TCP 패킷 수, TCP Syn 패킷 수, TCP 비트율, UDP 패킷 수이다. 예를 들어 전송하는 MIB 구조체의 code가 1.1.1로 설정되어있다면 IP 패킷수를 나타내고, 1.2.3라면 TCP의 비트율을 나타내는 것으로 Manager가 전송된 정보에 대해 알 수 있게끔 하였다. Manager는 이 트리구조를 통해 알게 된 정보를 화면에 Display한다. 이러한 과정에서 신뢰성 있는 정보를 받기 위해 Manager와 Agents는 서로 Connection Oriented된 TCP 프로토콜을 이용하였다.

|  |
| --- |
| typedef struct mibtreenode {  int degree;  float data;  struct mibtreenode\* child;  }Node;  typedef struct mibtree {  Node\* root;  }mibTree; |

***mib tree 구조체 형식***

|  |
| --- |
| typedef struct CodeAndValue {  char code[6];  float value;  }codeandvalue;  typedef struct mibStructure {  //pdu type response 보내는 입장이기 때문에 type 를 2로 지정  int pdu\_type = 2;  //request ID, error status, error index는 나중에 지정하기로 한다  codeandvalue array[5];  }; |

*변형된 SNMPv2 PDU 구조체 형식*

앞서 언급된 mib tree에 각자 저장된 정보를 각각 codeandvalue 구조체 배열인 array[]에 담고, 이 배열을 포함하는 변형된 SNMPv2 PDU 구조체 형식인 새로운 mibStructure 구조체를 구성한다. 이 syntax의 길이와 내용은 새롭게 구성된 mibStructure 구조체의 길이와 변환된 문자열이다.

2) Semantics (반환값 : 실패시 error (-1), 성공시 수신,송신 둘다 전달받고 전달하는 syntax의 길이) **UDP**

***(1)retval = sendto(sock, (char\*)params, sizeof(UDP\_PROTO), 0, (SOCKADDR\*)&serveraddr, sizeof(serveraddr));*** *(agent UDP client)*

agent 와 manager가 TCP CO 통신을 하기 위해서, agent UDP client에서 agent 의 port number와 type = 1를 넘겨준다. 여기서 type =1의 의미는 나중에 나타나지만 syn flooding alert message를 보내는 부분이 아닌 일반 udp 통신을 의미한다. 이 정보를 전달받는 소켓 주소 포인터 즉 목적지(manager)의 ip주소는 "127.0.0.1", port number 는 8000이다.

***(2)retval = recvfrom(sock, buf, sizeof(UDP\_PROTO), 0, (SOCKADDR\*)&clientaddr, &addrlen);***

*(manager UDP server)*

manager는 따라서 agent 의 정보를 다음과 같은 semantics를 통해 받아오고, buf에 저장해서 TCP connection 에 필요한 정보들을 parsing 하여 저장한다. 전달받는 정보는 UDP\_PROTO 구조체에서 문자열로 변환된것으로 다음과 같은 사이즈로 받는다.

***(3)retval = sendto(sock, buf, strlen(buf), 0, (SOCKADDR\*)&clientaddr, sizeof(clientaddr));***

*(manager UDP server)*

agent 의 정보를 받은 manager는 UDP server로서 잘 받았다는 syntax를 담은 buf를 agent UDP client 에 전달한다. 이는 error를 체크와 TCP connection 준비가 잘 되었다는 것을 확인하기 위한 semantics이다.

***(4)retval = recvfrom(sock, buf, BUFSIZE, 0, (SOCKADDR\*)&peeraddr, &addrlen);***

*(agent UDP client)*

이전에 지정한 peeraddr 를 통해서 manager UDP server로부터 잘 받았다는 syntax를 받고 error 체크를 하는 semantics.

**TCP**

***(1)retval = connect(sock, (SOCKADDR\*)&serveraddr, sizeof(serveraddr));*** *(manager TCP client)*

manager TCP client가 agent TCP server와 TCP통신을 하기 위해서 server에 connect를 맺으려는 try의 semantics로 TCPsocket으로 정해둔 sock를 통해서 이전에 설정해둔 server의 주소로 연결을 한다. 이때 지정해둔 serveraddr의 정보는 이전에 처음 recvform 한 udp 통신의 “*retval = recvfrom(sock, buf, sizeof(UDP\_PROTO), 0, (SOCKADDR\*)&clientaddr, &addrlen)*;“ 에서 받아온 agent TCP server의 port number와 ip 주소는 clientaddr의 ip주소이다.

***(2)retval = send(sock, buf, strlen(buf), 0);*** *(manager TCP client)*

manager TCP client는 agent 의 각 mib 정보를 받아오기 위해서 이전에 지정한 sock 로 request syntax를 보낸다.

***(3)msglen = recv(client\_sock, buf, BUFSIZE, 0);*** *(agent TCP server)*

agent 가 연결된 client 주소와 연결된 client\_sock를 통해 manager TCP client가 보낸 request syntax를 받아오는 semantics.

***(4)retval = send(client\_sock, (char\*)&sendMibStructure, sizeof(sendMibStructure), 0);*** *(agent TCP server)*

agent는 “packet capture”를 동시에 하면서 저장한 mib 정보들을 새롭게 구성한 pdu 형식으로 바꿔 TCP server의 역할로 manager TCP client에게 전달한다. request 받아온것에 대응해 reponse를 해주는 단계이다.

***(5)retval = recv(sock, buf, BUFSIZE, 0);*** *(manager TCP client)*

manager TCP client는 앞서 agent TCP server가 send() 한 여러 통신 상태를 담은 pdu형식 문자열을 받는다. 이후에 특정 mib tree를 기반으로 이 정보를 parsing 해서 개인 창에 display 해준다.

**2.2 프로토콜 구현**

**1)매니저 구현**

먼저 main 함수에서 UDP Socket을 열고 연결을 맺는다. UDP에 연결 요청이 있을 경우(type==1)에는 CreateThread() 함수를 이용하여 TCP Client에 관련된 Thread를 생성한다. type==0인 경우에는 뒤의 시스템구조#2 Attack 부분에서 추가적으로 설명한다. CreateThread 함수를 사용하는 방법에 대해 자세히 설명하자면, TCP\_client라는 함수를 호출하고, params라는 UDP\_PROTO 구조체를 인자로 넘겨준다. 이때 UDP\_PROTO 구조체인 params는 LPVOID 형태로 캐스팅하여 주소 형태로 넘겨주게 된다. 이를 구현한 코드는 아래와 같다.

|  |
| --- |
| else {  UDP\_PROTO params;    params.type = recvbuf->type;  strcpy(params.ip, recvbuf->ip);  params.port = recvbuf->port;  hThread = CreateThread(NULL, 0, TCP\_client, (LPVOID)&params, 0, &ThreadId);  if (hThread == NULL)  err\_display((char\*)"error: failure of thread creation!!!");  else  CloseHandle(hThread);  } |

CreateThread()를 통해 호출된 DWORD WINAPI 형식의 TCP\_Client 함수는 Agents에게 “request” 메시지를 전송하고 MIB에게 정보를 받고 display하는 역할을 수행한다. 이러한 작업을 수행하기 위해서 먼저 TCP Socket을 오픈한 후, Agents와 연결을 맺는 connect()를 수행하게 된다. while문을 통해서 실제 TCP Client의 작업이 이뤄지는데 send()를 통해 “request” 메시지를 5초마다 보낸다. 이때, 5초는 Sleep(5000)을 통해 5초동안 잠들었다가 다시 수행할 수 있게끔 하였다. 만약, Agents에게 정보를 recv()하게 되면 연결된 TCP sender의 IP 주소와 Port 번호를 각각 params->ip와 prams->port를 통해 출력하였다. 받은 정보는 미리 제작해둔 mibStructure 구조체 형태로 받아서 출력한다. 자세한 형태는 위의 시스템 구조에서 언급한 트리 형태의 모양으로 인식하여 분류해서 display한다. 자세한 출력 형태는 아래 코드와 같다.

|  |
| --- |
| mibStructure\* input;  input = (mibStructure\*)malloc(sizeof(mibStructure));  input = (mibStructure\*)buf;  printf("| IP | \t1.IP PACKET \t:\t %.1f\t|\n", input->array[0].value); //IP 패킷 수 출력  printf("| TCP | \t1. TCP PACKET \t:\t %.1f\t|\n", input->array[1].value); //TCP 패킷 수  printf("| | \t2. Syn PACKET \t:\t %.1f\t|\n", input->array[2].value); //TCP Syn 패킷 수  printf("| | \t3. BIT RATE \t:\t %.1f\t|\n", input->array[3].value); //TCP 바이트율  printf("| UDP | \t1.UDP PACKET \t:\t %.1f\t|\n", input->array[4].value) ;//UDP 패킷 수 |

**2)디바이스(agent) 구현**

main()

TCP server의 역할을 하기 위해 listen socket를 열어두고 자신의 port 정보를 입력받게 해서 server address를 구성한다. TCP 통신 준비 이후에, 디바이스는 packet capture를 동시다발적으로 해야하기 때문에 thread를 생성한다.

|  |
| --- |
| hThread = CreateThread(NULL, 0, Capture, NULL, 0, &ThreadId); |

다음은 각 디바이스의 통신 status를 저장하기 위한 mib tree를 생성한다.

|  |
| --- |
| //parsing 전에 정보를 저장할 tree 선언  //새로운 tree 생성  tree\_for\_mib = newTree();  tree\_for\_mib->root = newNode(0);  //IPnode 대임  ParentInsertChild(tree\_for\_mib->root, 0);  //이러면 root 의 degree가 하나 올라가게 된다.  //TCP node 대입  ParentInsertChild(tree\_for\_mib->root, 0);  //UDP node 대입  ParentInsertChild(tree\_for\_mib->root, 0); |

이때 ParentInsertChild()함수를 구현해, 각 root의 degree정보에 따라 비어 있는 child에 값을 넣어주는 역할을 하게 한다. 이후에 TCP connection이 이뤄지자마자 manager가 request를 보내기 때문에 다시 ParentInsertChild()함수를 이용해 각 mib tree에 비어있는 value값을 넣어 전체 mib tree 구조를 가지게 한다.

다음은 while문을 구성해 manager와 TCP connection 을 이뤄지게 한다. 다시 while문안에서 manager TCP client와 agent TCP server와의 data 통신이 이뤄지는 부분을 구현을 하고 manager TCP client에서 request가 도착하면, pdu format의 syntax를 담는 mibStructure 데이터 형식의 구조체를 선언, 이전에 생성한 mib tree의 정보를 각각 code, value로 담는다.

|  |
| --- |
| mibStructure sendMibStructure;  sendMibStructure.pdu\_type = 2;  memcpy(sendMibStructure.array[0].code, "1.1.1", 6);  sendMibStructure.array[0].value = tree\_for\_mib->root->child[0].child[0].data;  memcpy(sendMibStructure.array[1].code, "1.2.1", 6);  sendMibStructure.array[1].value = tree\_for\_mib->root->child[1].child[0].data;  memcpy(sendMibStructure.array[2].code, "1.2.2", 6);  sendMibStructure.array[2].value = tree\_for\_mib->root->child[1].child[1].data;  memcpy(sendMibStructure.array[3].code, "1.2.3", 6);  sendMibStructure.array[3].value = tree\_for\_mib->root->child[1].child[2].data;  memcpy(sendMibStructure.array[4].code, "1.3.1", 6);  sendMibStructure.array[4].value = tree\_for\_mib->root->child[2].child[0].data; |

구성한 sendMibStructure를 send()를 통해 manager에 전달하면서, 한번의 TCP 통신을 완료한다. 이때 response를 보낸 이후 mib tree에 들어가는 통계 자료를 reset 즉 =0으로 데이터를 초기화한다.

Capture()

main()에서 생성한 thread는 Capture함수로 packet capture를 이 안에 구현한다. 디바이스는 계속해서 packet capture를 진행하고 있기 때문에 이 부분이 가능하게 만들어준다. interface 를 선택받게 하고 adapter를 지정한 이후에 다시한번 CreateThread()를 통해 “UDP\_call” thread함수를 실행한다. 이 부분은 디바이스와 manager가 TCP통신을 하기전에 connection을 도와주는 역할을 한다. 이후에 pcap\_loop를 통해 packet capture를 시작한다.

|  |
| --- |
| hThread = CreateThread(NULL, 0, UDP\_call, (LPVOID)&params, 0, &ThreadId); |

UDP\_call()

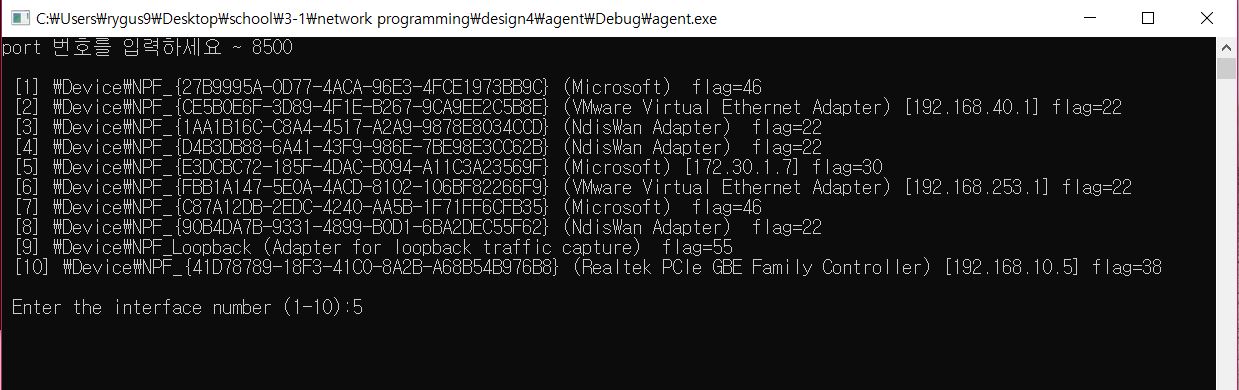
일반 echo UDP 통신 형식을 이용해서 각각 sendto(), recvfrom()를 잘 활용해 TCP connection를 위한 정보를 보낸다.

get\_stat()

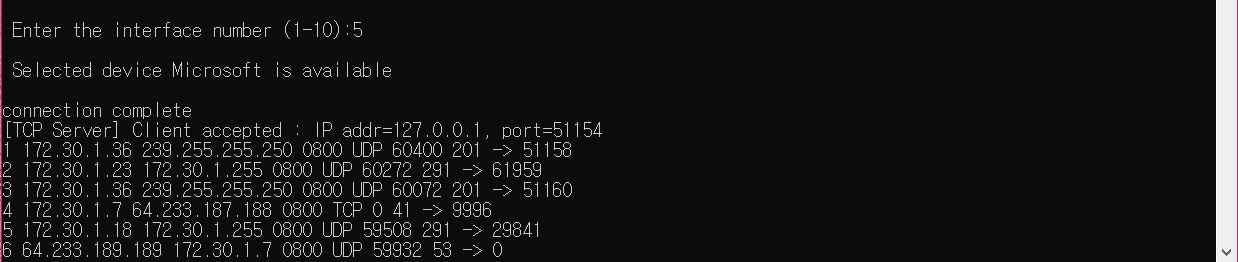
packet capture를 loop를 통해 계속해서 실행을 하는데 mib tree를 구성하기 위해서 capture를 진행하면서 net\_ip\_count (ip packet 개수), trans\_tcp\_count(tcp packet 개수), if\_tcp\_syn(이 디바이스에게 오는 tcp syn 개수), tot\_captured\_pk\_num (전체 packet 개수), trans\_udp\_count(udp packet 개수)를 각 조건에 맞게 parsing 한다. 이후에 global 로 선언한 tree\_for\_mib(mib tree)에 각 정보를 ParentInsertChild()함수를 이용해서 넣는다.

**2.3 동작 결과**

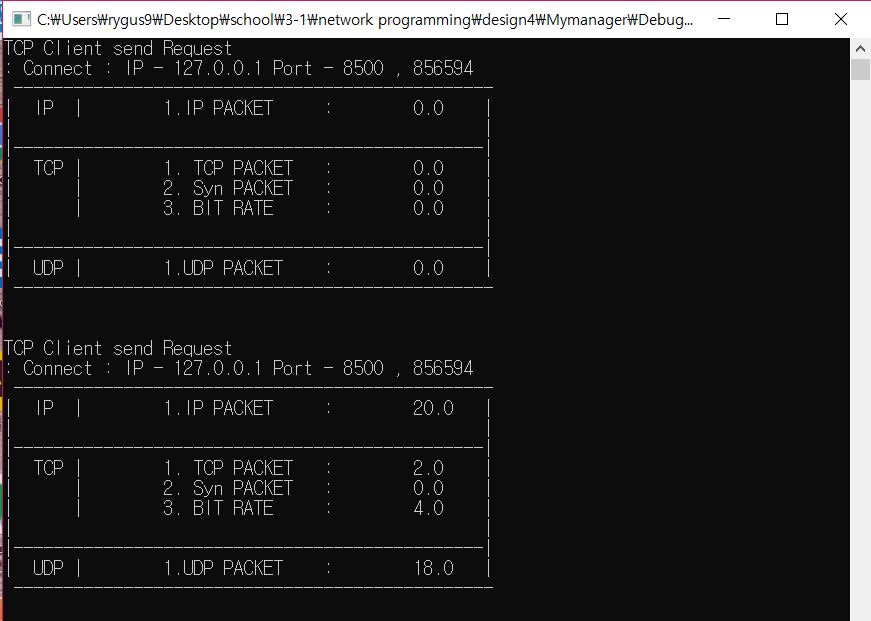
1. agent를 실행시키기 이전에 먼저 manager를 실행시켜야 한다. 다음은 에이전트 시작할 때 사진이다. 자신의 TCP 서버 포트를 정하고 (8500) 캡처할 디바이스를 정한다.



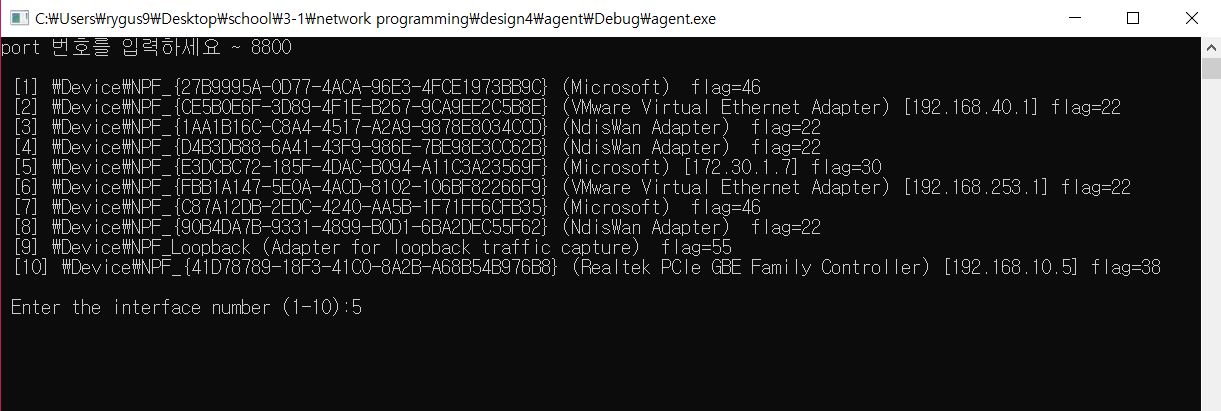
1. 장치를 선택하면 agent는 바로 manager와 연결을 시도한다. 연결에 성공하면 connection complete 메시지를 띄우고 이후 캡처를 계속 진행한다.



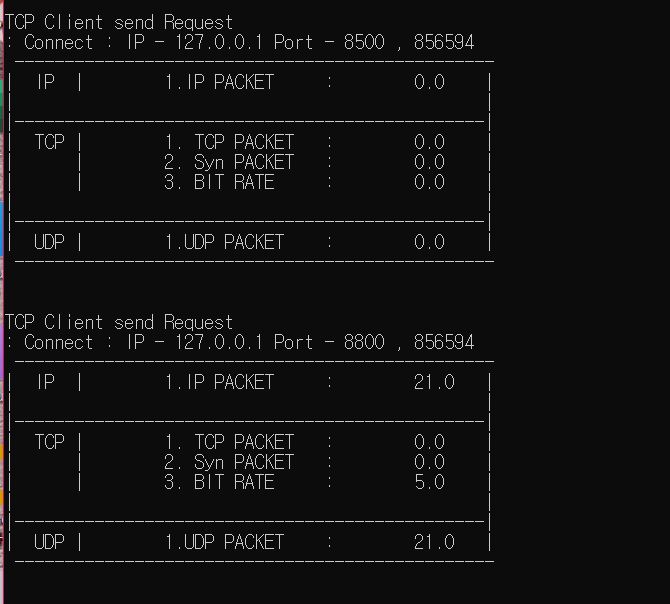
1. manager는 agent의 연결을 기다리다 agent와 연결되면 agent에게 5초 간격으로 정보를 요청하고 응답 받은 정보를 디스플레이한다.



1. 시스템 구조 #1 처럼 여러 개의 agent와 통신하는 상황을 만들기 위해 agent를 하나 더 실행한다. 기존 agent와는 다른 8800 포트로 실행시켰다.



1. manager는 각각의 connection에 쓰레드를 부여하여 각각의 agent와 통신하도록 한다. agent 사이의 정보는 해당 response를 주소를 띄워 구분한다.



**3. 시스템구조#2 - Attack**

3.1 프로토콜

1)Syntax

**UDP**

(1) agent UDP client 에서 syn flooding attack를 manager UDP server에 알리는 syntax:

설계 #1에서 언급한 UDP\_PROTO 구조체 형식을 이용해서 type 에는 0의 값을 넣어 이 syntax는 syn flooding attack를 받은 디바이스에서 전송하는것을 알리고, 처음에 받아온 agent의 port number를 저장해 이 구조체를 가리키는 포인터를 문자열로 변환, UDP\_PROTO 구조체의 길이만큼을 syntax로 지정한다.

(2) manager UDP server에서 attack 정보를 잘 받았다고 알려주는 echo 형식의 syntax:

“check\0”의 문자열을 담은 buf를 buf의 길이만큼 자른것까지 syntax 구성.

**TCP**

(1) attacker 에서 agent를 대상으로 한 많은 양의 동일한 syntax:

|  |
| --- |
| struct ethhdr {  unsigned char h\_dest[ETH\_ALEN];  unsigned char h\_source[ETH\_ALEN];  unsigned short h\_proto;  };  struct iphdr {  unsigned char ihl : 4;  unsigned char version : 4;  unsigned char tos;  u\_short tot\_len;  u\_short id;  u\_short frag\_off;  u\_char ttl;  u\_char protocol;  u\_short check;  unsigned int saddr;  unsigned int daddr;  };  struct tcphdr {  u\_short p\_source;  u\_short p\_dest;  u\_long seq\_num;  u\_long ack\_num;  unsigned char zero1 : 4;  unsigned char thl : 4;  unsigned char syn : 6;  unsigned char zero2 : 2;  u\_short window;  u\_short checksum;  u\_short point;  unsigned char tcp\_option[TCP\_OPT\_LEN];  };  u\_char msg[1500]; |

ethernet header, IP header, TCP header 구조를 바탕으로 보낼 패킷 버퍼인 msg에 value를 대입하는 방식으로 msg를 생성한다. syntax는 이 packet를 담고 있는 msg와 길이는 미리 지정한 TOTLEN 66만큼이다.

2)Semantics

**공격**

우선 attack를 당하는 상황을 구현하기 위해서는 attacker가 syn 패킷을 만들어서 보내는 과정이 필요하다. 이는 단순히 공격하고자 하는 지점의 ip와 port를 가진 syn 패킷을 많이 만들면 이 패킷이 알아서 공격 지점으로 몰리게 된다.

**UDP**

***(1)retval = sendto(sock, (char\*)params, sizeof(UDP\_PROTO), 0, (SOCKADDR\*)&serveraddr, sizeof(serveraddr));*** *(agent UDP client)*

agent에 attack이 들어와 syn 패킷이 많이 검출된다면 manager의 UDP server에게 공격당하고 있음을 알리기 위해 패킷을 만들어 보낸다. Syntax는 위에서 manager와 agent 사이에 연결할 때와 똑같지만 type 값에 0을 대입해서 syn flooding attack이라는 점을 알린다.

***(2)retval = recvfrom(sock, buf, sizeof(UDP\_PROTO), 0, (SOCKADDR\*)&clientaddr, &addrlen);***

(Manager UDP server)

agent로 부터 attack 당하고 있다는 패킷을 수신 받는 부분이다. 이 부분은 위의 call 과정과 같지만 이후 type 값을 기준으로 분기 처리를 해서 수행해야 할 동작을 다르게 하였다.

이후 server에서 패킷을 잘 받았다는 의미로 check 패킷을 보내고 client는 이를 받아 패킷이 잘 갔음을 확인한다. 이 과정은 시스템 구조 #1의 UDP semantic에서 check 하는 것과 동일하게 작동한다.

**3.2 프로토콜 구현**

1)매니저 구현

위에서 언급했던 것과 같이 먼저 main 함수에서 UDP Socket을 열고 연결을 맺는다. Attack에 대한 매니저의 역할은 간단하다. Attack에 대한 부분을 알림받고 display해주기만 하면 된다. 만약, 입력받은 type이 0으로 들어온다면 Trap에 관한 부분이기 때문에 “Syn Flooding Attack!!”이라는 정보를 프린트하고, 추가적으로 어느 곳에서 Trap 경고 메시지가 들어왔는지 확인하기 위해서 IP 주소와 Port 번호를 함께 기재해서 display한다. 자세한 코드는 아래와 같다.

|  |
| --- |
| UDP\_PROTO\* recvbuf;  recvbuf = (UDP\_PROTO\*)buf;  if (recvbuf->type == 0) {  printf("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");  printf("| Syn Flooding Attack !! |\n");  printf("| from IP addr : %s , Port : %d |\n", inet\_ntoa(clientaddr.sin\_addr), ntohs(clientaddr.sin\_port));  printf("|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n\n");  } |

2)디바이스 구현

디바이스의 경우 mib를 만들기 위해 캡처하는 과정에서 syn 패킷 개수를 캡처 하면서 syn 패킷의 개수를 체크한다. 특정 개수 이상일 경우 새로운 쓰레드를 만들고 UDP client socket 를 열어 매니저에게 자신이 공격당하고 있음을 알린다. 밑에는 순서대로 syn 패킷을 체크해서 특정 개수 이상일 경우 매니저에게 UDP client socket을 열어 경고 메시지를 보내는 코드다. 참고로 alert은 Syn flooding attack 공격 메시지를 한 번만 보내기 위해 만든 변수이다. UDP alert 함수는 간단히 매니저로 UDP\_PROTO를 따르는 type 값은 0이고 port 값은 자신의 port인 패킷을 보낸다. 이에 대한 코드는 아래와 같다.

|  |
| --- |
| if (ntohs(tcp->dport) == port) { if\_tcp\_syn++; }  //Syn flooding check 단순하게 그냥 갯수로 측정  if (if\_tcp\_syn > =1500 && alert == 0) {  UDP\_PROTO\* params = (UDP\_PROTO\*)malloc(sizeof(UDP\_PROTO));  params->port = port;  params->type = 0;  hThread = CreateThread(NULL, 0, UDP\_alert, (LPVOID)params, 0, &ThreadId);  if (hThread == NULL)  err\_display((char\*)"error: failure of thread creation!!!");  else  CloseHandle(hThread);  alert = 1;  } |

3)기타

시스템 구조 #2에서는 시스템 구조 #1과 달리 시나리오에 attacker 가 존재한다. attacker는 agent에게 다량의 tcp syn 패킷을 보내 syn flooding attack을 유도하는 역할을 하는 프로그램이다. attacker 프로그램 구현에서 만들어지는 패킷은 실제로 tcp connection이 이루어지는 패킷을 참고하여 만들었다. 그래서 수업에서는 다루지 않은 tcp option 필드가 존재한다. option field에는 selective ack 설정과 mss 관련 설정, window size 관련 설정이 있어서 이를 그대로 참고하였다. 패킷을 만들 때 자신의 source ip와 source port는 랜덤 함수를 이용해서 프로토콜 내에서 랜덤하도록 설정하였다. ip 를 랜덤하는 과정이 좀 빈약한데 실제로는 4자리 숫자 전부를 랜덤화 해야 한다. 마지막 자리만 랜덤화한 이유는 테스트 하는 환경이 무선랜 안이라 기기 간 구분에 마지막 자리에서만 이루어질 것이라 생각했기 때문이다.

|  |
| --- |
| tcph->p\_source = htons((unsigned short)(rand()%10000 + 50000));  iph->saddr = inet\_addr("172.30.1.1"); **// sourc IP address**  ((unsigned char\*)&(iph->saddr))[3] = rand() % 254 + 1; |

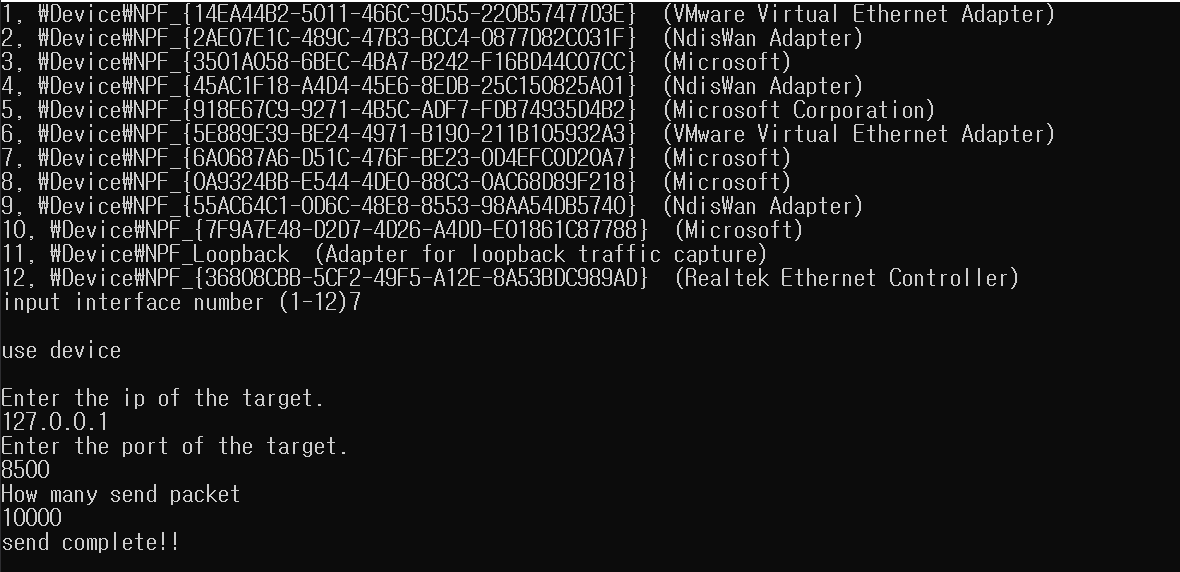
패킷 생성 과정은 실습 시 진행했던 UDP 패킷 만드는 과정에서 만들어야 하는 패킷의 구조만 다를 뿐 과정은 동일하기 때문에 생략한다.

입력 받은 ip와 port를 적용해서 패킷을 만들고 입력 받은 횟수만큼 패킷을 만들어 패킷을 보내는게 attacker 프로그램의 전부다. attack\_ip와 attack\_port는 각각 입력받은 ip와 입력받은 port다.

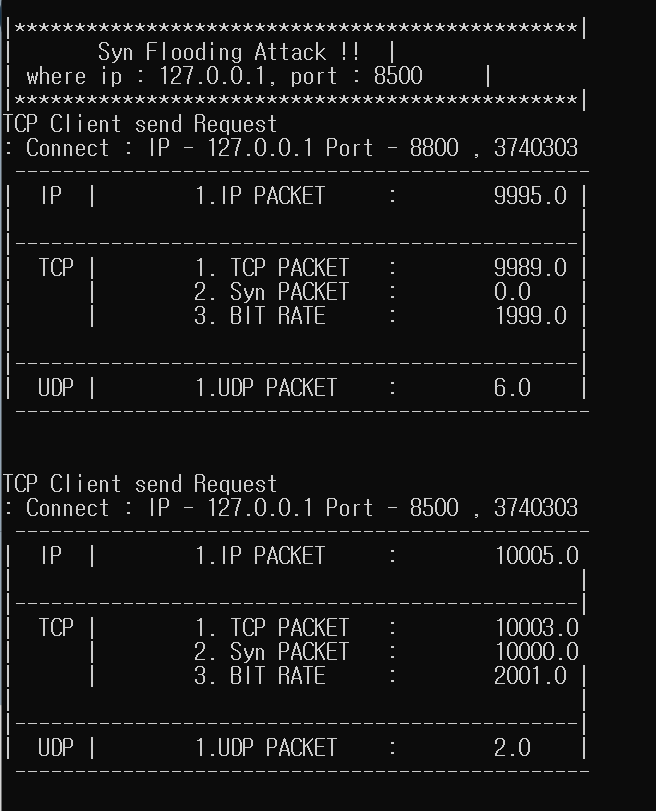
|  |
| --- |
| printf("\n얼마나 패킷을 보낼 까요?");  scanf("%d", &syn\_pkt);  //이 부분을 for문 으로 묶어서 패킷 여러 개 보내기 하자.  for (i = 0; i < syn\_pkt; i++) {  offset = 0;  memset(msg, 0, sizeof(msg));  build\_Ethernet\_Header(msg, ETH\_P\_IP, &offset);  build\_IP\_Header(msg, &offset, attack\_ip);  build\_TCP\_Header(msg, &offset, attack\_port);  pcap\_sendpacket(adhandle, msg, TOTLEN);  } |

3.3 동작 결과

1. 공격자 프로그램은 특정 ip와 특정 port를 선택하여 선택한 주소로 원하는 갯수만큼 syn 패킷을 생성해서 보낼 수 있다.



2. manager 프로그램은 해당 agent의 신호를 받아 어느 agent에서 syn flooding attack 이 발생했는지 디스플레이 해준다.



**4. 프로그램 내역**

Manager/Agent/Attacker를 구현한 cpp 파일을 압축한 ZIP 내역은 아래와 같다.

R04\_201720768\_snmpAssignment.zip

|  |
| --- |
| agent.cpp  agent.sln  attacker.cpp  manager.cpp  manager.sln  TCPgeneration.sln |

빌드 시 주의할 점은 빌드하는 순서인데 ⓵manager cpp파일 ⓶ agent cpp파일 (Port 번호와 번호 선택) ⓷ attacker cpp파일 순으로 컴파일 해야 한다. 또한, attack시, Port 번호를 입력하는 창이 있는데 port가 겹치지 않게 너무 낮은 port를 피하고 패킷을 전송하는 양은 1500개 이상으로 추천한다.

추가적으로 TCPgeneration.sln 는 attacker에 해당하는 파일이다.