IK2213 - Network Services and Internet-based Applications
Assignment 3 - Packet Bouncer

Ganapathy Raman Madanagopal, Tien Thanh Bui
SVN repository: ttbu

1. Introduction

In this project, we have developed a packet bouncer for redirecting packets between multiple clients and a server. Instead of communicating directly to the server, the clients will communicate with the bouncer instead, and the bouncer will act as a client to communicate with the server. The packet bouncer supports ICMP, TCP and FTP active connection. The source code can be found in svn repository *ttbu.*

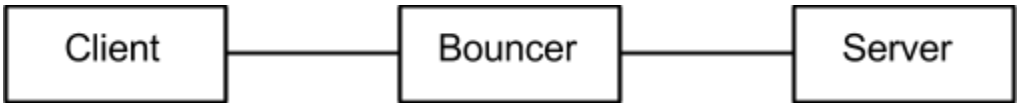

Figure 1 - Topology

2. Problems & Our Solutions

The packet bouncer has the four main tasks: capturing packets destined to the bouncer, validating the packets, modifying the packets so that the clients and the server do not know they are communicating with the other, and sending packets to their destination. This section will describe how we realize the tasks.

2.1. Capturing packets

Packets were captured on *tap0* interface of the bouncer host using libpcap library. The bouncer was initialized so that only ICMP and TCP packets were captured.

2.2. Validating packets

Each captured packet will be validated and will be dropped if it is invalid. The IP header will be validated first.

In IP header the following fields will be validated - IP Header Length, Destination Address (is the packet is destined to bouncer), IP header checksum, IP version, Evil bit set or not and Time-to-Live field.

To find the protocol in the payload, protocol field in the IP header is scrutinized. Based on the protocol present in the IP payload, the flow branches into two main branches: ICMP and TCP.

2.2.1. ICMP

When the payload contains ICMP packets, then ICMP header is also validated for the ICMP checksum, and ICMP code. If the packet passes every validation check, then it passed to the next stage of packet modification in the bouncer.

2.2.2. TCP

If the payload was TCP then the Transport level (TCP) checksum is first validated. Also the Destination port of the packet is validated to check whether is packet is destined to bouncer port. To support multiple TCP connections, a data structure (Stack) was used to store the details of the TCP sessions. Basically, Clients IP address (Source Address), Source Port, Dummy Port (which act as Client Source Port when the packet is modified and forwarded to Server).

Further, the TCP segment is inspected. When the packet arrives from the client, the details in the packet are searched against the existing connections in the stack. If a match is found, then the packet details are modified specific to that connection.

If the packet is from the client and contains SYN field, a new connection request has arrived from client and the above mentioned details are noted and added to the stack.

If the packet is from Server on the dummy port specified, then the existing connection list is searched and if the match is found the packet is forwarded to the modifying stage.

If the TCP payload was FTP, then the following additional work around is done before send modifying stage. Else, the packet is sent to modifying stage:

- **Control Packet**
    1. If the TCP payload was FTP, and the packet is from Client then packet payload is searched for “PORT” command. If it contains PORT, the client's data port is extracted from the packet and stored in the corresponding connection list in the stack for future communication.
    2. If the packet is from server on the bouncer’s dummy port, the connection details are searched in the stack, if found forwarded to the modifying stage.
- **Data Packet**
    1. If the packet is from client on the data port of the bouncer, the existing TCP connections are searched for find the match for the Clients data port, if found forwarded to the modifying stage and will modified and sent to port 20 of the server.
    2. If the packet is from server, on the data dummy port then the stack is searched to find the clients IP address and its corresponding data port and forwarded to modifying stage.

If any of the above mentioned stage fails, then the packet is dropped and an error is displayed on the bouncer console for more debugging.

2.3. Modifying packets

As mentioned, the packets that are validated reach this stage, where the packets are modified before sending to the next stage.

1. If the packet is from Client, then based on the result obtained in search of the existing connections in the stack, the IP source is modified to bouncer’s IP, Destination IP to server’s IP, Source port to bouncer's dummy port and Destination port is changed to 21 for control packets and 20 for the data packets.

2. If the packet is from Server, then based on the result obtained in search of the existing connections in the stack, the IP source is modified to bouncer's IP, Destination IP to server's IP.

After making the above modification, a new checksum is calculated on each layer, if the packet contains TCP segment a new TCP header checksum is calculated or if the packet is ICMP, a new ICMP header checksum is calculated.

Finally new IP header checksum is calculated and forwarded to the final stage.

2.4. Sending packets

Finally the a new socket (binded based on destination IP and destination port) will opened to connect for communication. On successful opening of the socket, the modified packet is send out of the packet over the network.

3. Discussion & Conclusion

Our packet bouncer service performs basic bouncer/NAT quite well. However, it is limited by the fact that currently bouncer can works only with ICMP, TCP and FTP packets. Support for UDP and other application handling hasn't been implemented. We will try to improve it in the future. However, although only basic functionalities were developed, this project has given us a solid background in the areas of Packet capturing using LibPcap, Socket Programming and Data Structures.