

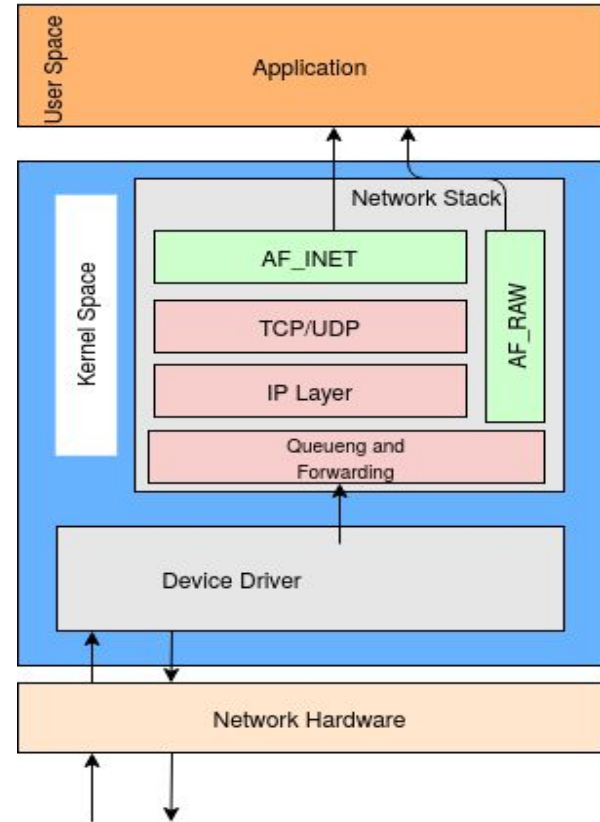
Network Security Tutorial



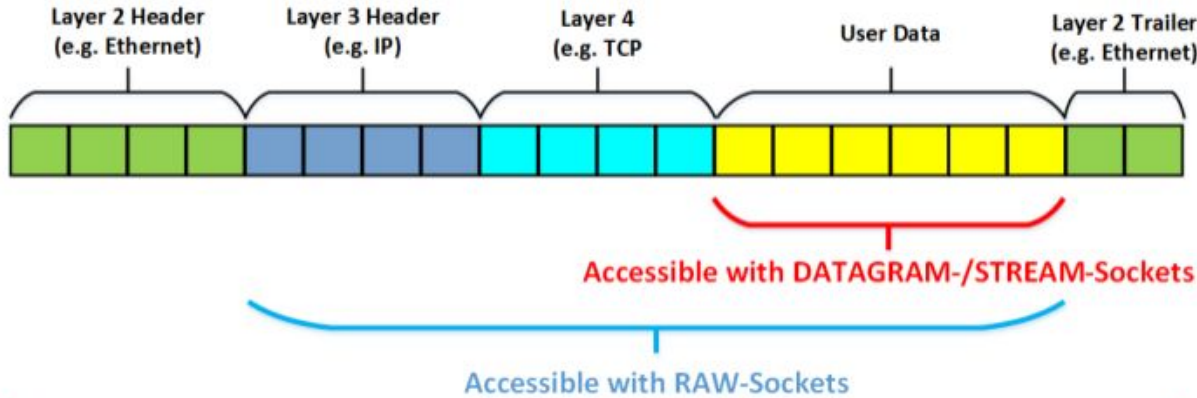
Raw Socket Programming

Raw Socket

- Directly access lower layer protocols in the network stack.
 - Link Layer - L2 (Ethernet)
 - Network Layer - L3 (IP)
 - Transport Layer - L4 (TCP/UDP/SCTP)
- Socket Creation
- Add relevant header information based on the protocol specified in the socket
- Protocols
 - IPPROTO_ICMP = 1
 - IPPROTO_TCP = 6
 - IPPROTO_UDP = 17
 - IPPROTO_SCTP = 132
 - IPPROTO_RAW = 255
 - IPPROTO_ESP = 50
 - IPPROTO_AH = 51



Raw Socket Programming



- Important Headers
 - `sys/socket.h`
 - `netinet/in.h`
 - `ethernet.h`
 - `netinet/ip.h`
 - `netinet/tcp.h`
 - `netinet/udp.h`

- `socket(AF_INET, SOCK_RAW, IPPROTO_RAW)` -> For IP /L3 level - SEND only
- `socket (PF_PACKET, SOCK_RAW, htons (ETH_P_ALL))` -> For all ethernet type L2 packet processing
- `socket (PF_PACKET, SOCK_RAW, htons (ETH_P_IP))` -> For all ethernet type, IP Packets only at L2
- `socket (AF_INET, SOCK_RAW, IPPROTO_UDP)` -> For UDP only
- `socket (AF_INET, SOCK_RAW, IPPROTO_TCP)` -> For TCP only

Stack Layers Headers

```
struct ether_header
{
    u_int8_t  ether_dhost[ETH_ALEN];
    u_int8_t  ether_shost[ETH_ALEN];
    u_int16_t ether_type;
} __attribute__((__packed__));

/* Ethernet protocol ID's */
#define ETHertype_PUP      0x0200
#define ETHertype_SPRITE  0x0500
#define ETHertype_IP       0x0800
#define ETHertype_ARP      0x0806
#define ETHertype_REVARP   0x8035
#define ETHertype_AT       0x809B
#define ETHertype_AARP     0x80F3
#define ETHertype_VLAN     0x8100
#define ETHertype_IPX      0x8137
#define ETHertype_IPV6     0x86dd
#define ETHertype_LOOPBACK 0x9000
```

File: /usr/include/net/ethernet.h

```
struct iphdr
{
    #if __BYTE_ORDER == __LITTLE_ENDIAN
        unsigned int ihl:4;
        unsigned int version:4;
    #elif __BYTE_ORDER == __BIG_ENDIAN
        unsigned int version:4;
        unsigned int ihl:4;
    #else
        # error "Please fix <bits/endian.h>"
    #endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
};
```

File: /usr/include/linux/
/usr/include/netinet/ip.h

Stack Layers Headers

```
struct tcphdr {
    __be16 source;
    __be16 dest;
    __be32 seq;
    __be32 ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u16 res1:4,
        doff:4,
        fin:1,
        syn:1,
        rst:1,
        psh:1,
        ack:1,
        urg:1,
        ece:1,
        cwr:1;
#elif defined(__BIG_ENDIAN_BITFIELD)
    __u16 doff:4,
        res1:4,
        cwr:1,
        ece:1,
        urg:1,
        ack:1,
        psh:1,
        rst:1,
        syn:1,
        fin:1;
#else
#error "Adjust your <asm/byteorder.h> defines"
#endif
    __be16 window;
    __sum16 check;
    __be16 urg_ptr;
};
```

File: /usr/include/linux/tcp.h

```
struct udphdr
{
    __extension__ union
    {
        struct
        {
            u_int16_t uh_sport;
            u_int16_t uh_dport;
            u_int16_t uh_ulen;
            u_int16_t uh_sum;
        };
        struct
        {
            u_int16_t source;
            u_int16_t dest;
            u_int16_t len;
            u_int16_t check;
        };
    };
};
```

File: /usr/include/linux/
/usr/include/netinet/udp.h

Code Walkthrough

- Packet Snooping on Linux
- Receive all IP packets on a specific Ethernet interface
- Parse the packet
- Print all the IP header fields
- Identify the type of transport protocol
- Print specific transport protocol fields (TCP/UDP/SCTP)
- Build: **make**
- Final binary: **rawSocket**
- Execution: **sudo ./rawSocket <n/wk interfacename>**
- Example: **sudo ./rawSocket wlp2s0**

UDP Chat - Demo

- Execution: **sudo ./rawSocket <n/wk interfacename>**
- Example: **sudo ./rawSocket wlp2s0**
- Receive a UDP chat message and send a reply message on raw socket.

Task 1 - DNS Packet Parser

Using the RAW socket program shared, identify the DNS packets received on your interface. Print various fields of DNS like identification, query/response, type of query/response, number of different records (like question, answer, authority, resource).

Use dig/your own DNS client to send the DNS query to your local configured virtual IP

E.g: Dig @192.168.1.22 rawsocket.tut

Add screenshots of your execution along with your source files. If you are adding a new source file add it to Makefile to include it for build.

The code should be successfully built with existing instructions only. There is no need to change the final binary name. If you do, add a readme in your submission.

Task 2 - DNS Client with RAW Socket

Using the RAW socket program shared, build your own DNS client to send a simple DNS A Record query. Use the DNS packet parser from Task 1 to print the query details at the other end.

Add screenshots of your execution along with your source files. Add a short readme in your submission explaining details of building and executing your client program.

Report Submission

- Prepare a report including Task1 and Task2 with specific details asked in individual tasks' slides. Add relevant screenshots of the execution showing the respective output.
- Submit it to google classroom

References

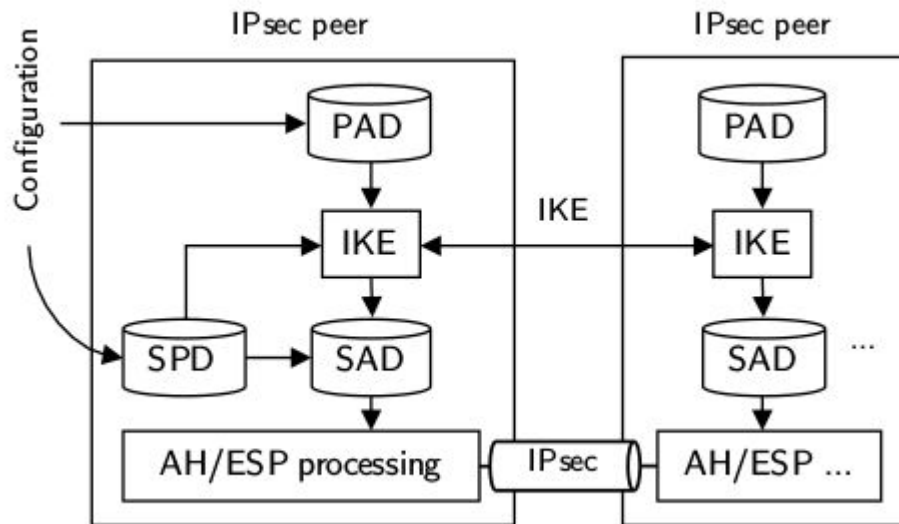
- \$man 7 raw
- Chapter 8 Security in Computer Networks from COMPUTER NETWORKING A Top-Down Approach Kurose and Ross
- Guide to IPsec VPNs Recommendations of the National Institute of Standards and Technology
- <https://wiki.openssl.org/index.php/EVP>
- <https://man7.org/linux/man-pages/man7/netdevice.7.html>
- <https://www.wireshark.org/>
- NOTE: A protocol of IPPROTO_RAW implies enabled IP_HDRINCL and **is able to send** any IP protocol that is specified in the passed header. **Receiving of all IP protocols via IPPROTO_RAW is not possible using raw sockets.**

Assignment- IPSec

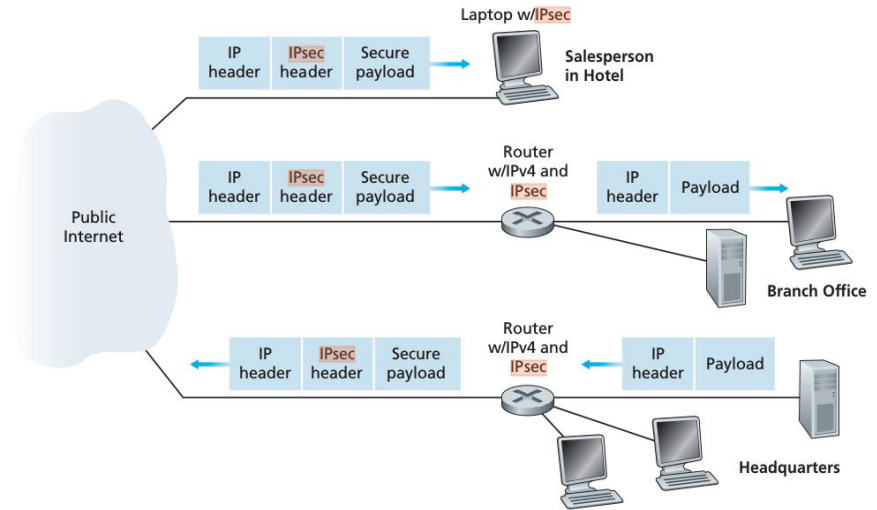
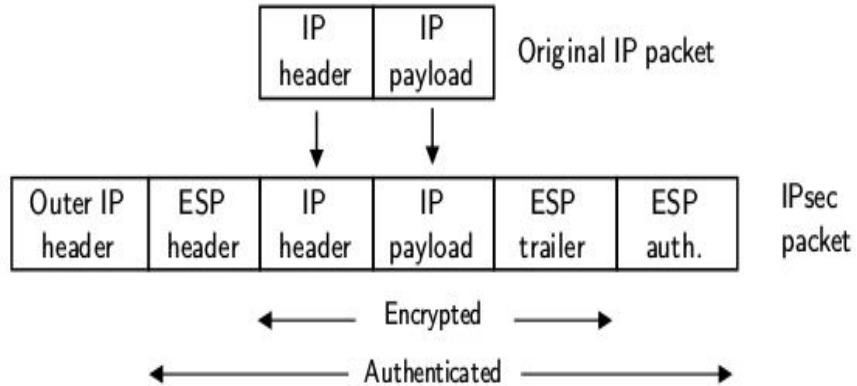
- IP security protocol
- Used for network layer security
- Ensures private communications over public networks.
- Typically used to host VPNs
 - Secure communications mechanism for data and control information transmitted between networks.
 - Protection Mechanisms
 - Confidentiality,
 - Integrity,
 - Data origin authentication,
 - Replay protection
- Secures IP Datagrams between
 - Site to Site
 - Host to Site
 - Host to Host

IPSec Peer Communication

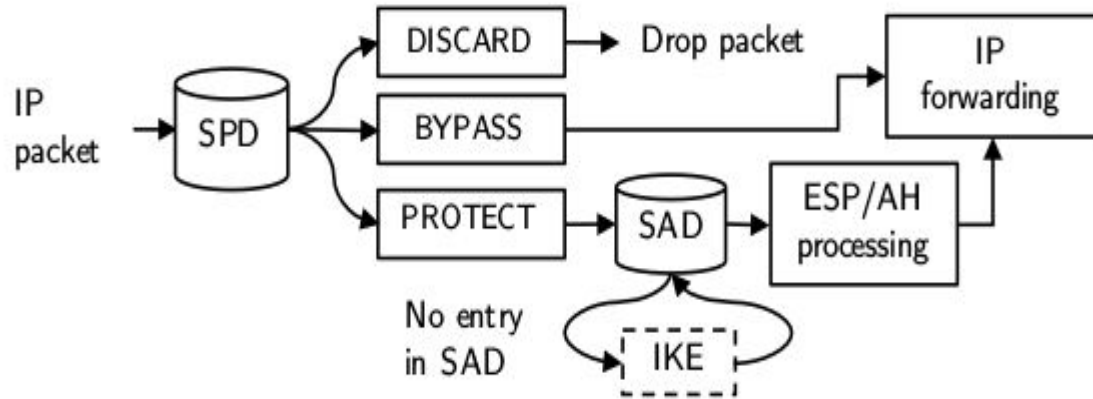
- Secure Association (SA) between IPSec Peers
- IKE: Internet Key Exchange before secure exchange of data on IPSec
 - Two phases
 - Key Exchange and session key establishment
 - SA creation
 - Successor: Internet Key Exchange v2
- IPSec Modes
 - Tunnel Mode
 - Transport Mode
- IPSec Protocols
 - ESP
 - AH
- IPSec Databases
 - SPD
 - SAD



IPSec Packet Structure



IPSec Packet Processing



Helper Exercise 1 - OPENSSL APIs

- Demonstration of confidentiality with client-server execution
- Libraries: crypto, pthread and boost_system
- openssl version 1.1.1
- Build
 - `g++ client.cpp -o oclient -lcrypto -lpthread -lboost_system`
 - `g++ server.cpp -o oserver -lcrypto -lpthread -lboost_system`
- APIs from <https://wiki.openssl.org/index.php/EVP>
- Dependency Installation
 - `sudo apt-get install libssl-dev`
 - [Boost Installation](#)

IPSec Packet Capture

Apply a display filter ... <Ctrl>-

No.	Time	Source	Destination	Protocol	Length	Info
1	0.869090000s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	1344	Aggressive
2	0.190369003s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	484	Aggressive
3	0.107428349s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	228	Aggressive
4	0.17143127s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	124	Transaction (Config Mode)
5	0.17174059s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	148	Transaction (Config Mode)
6	0.24211732s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	116	Transaction (Config Mode)
7	0.242343569s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	116	Transaction (Config Mode)
8	0.242451797s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	228	Transaction (Config Mode)
9	0.396649784s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	276	Transaction (Config Mode)
10	0.316930204s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	668	Quick Mode
11	0.380869159s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	140	Informational
12	0.381231432s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	244	Quick Mode
13	0.381475644s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	180	Quick Mode
14	0.382618361s	192.168.1.16	ipsecvp1.11th.ac.in	ISAKMP	132	Informational
15	0.400000000s	ipsecvp1.11th.ac.in	ipsecvp1.11th.ac.in	ESP	128	ESP (SPI=0x2827dea4)
16	0.446493478s	ipsecvp1.11th.ac.in	192.168.1.16	ISAKMP	132	Informational
17	4.34676339s	192.168.1.16	ipsecvp1.11th.ac.in	ESP	128	ESP (SPI=0x2827dea4)
18	12.278806088s	192.168.1.16	ipsecvp1.11th.ac.in	ESP	128	ESP (SPI=0x2827dea4)
19	13.97224403s	192.168.1.16	ipsecvp1.11th.ac.in	ESP	144	ESP (SPI=0x2827dea4)
20	14.161689954s	ipsecvp1.11th.ac.in	192.168.1.16	ESP	200	ESP (SPI=0xe44a4a7a)
21	27.894857263s	192.168.1.16	ipsecvp1.11th.ac.in	ESP	128	ESP (SPI=0x2827dea4)

Frame 15: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface 0

Linux cooked capture

Internet Protocol Version 4, Src: 192.168.1.16 (192.168.1.16), Dst: ipsecvp1.11th.ac.in (193.232.241.5)

User Datagram Protocol, Src Port: 54064, Dst Port: 4500

UDP Encapsulation of IPSec Packets

Encapsulating Security Payload

ESP SPI: 0x2827dea4 (673781540)

ESP Sequence: 1

0000 00 04 00 01 00 06 f2 ca 2b 21 22 00 00 00 00+1....
0010 45 00 00 70 51 8c 40 00 40 11 ce 4a c0 a0 01 10 E p0 0 0 3...
0020 67 e8 f1 05 d3 30 11 84 00 5c 7d Ba 28 27 de a4 g...0... \...
0030 00 00 00 01 26 39 31 50 e8 0b c5 50 ba 39 31 80 ...4...
0040 32 1c e8 f9 ca f5 9e 7a b8 da 51 9e 45 e8 e0 43 2...z...Q...C
0050 30 2a 76 ce 21 ac f3 fd 2e 9d 85 b7 0c 9a f6 0*zv...
0060 94 39 ec 05 06 bd d3 a1 3a 41 e0 ce ff 1e e0 82 9...A...
0070 b6 0f 97 4d ba e5 e2 1f fb 50 87 45 32 ce 3f 5f ...W...P E2 ?_

- Wireshark Filters
 - ESP
 - ISAKMP

Helper Exercise 2 - IPSec Packet Format

Capture traffic on Wireshark for 1-2 minutes, when connecting to IITH VPN. Observe the involved IPSec related packet exchange by placing appropriate packet filters.

- IKE Packets
- IPSec packet format
- Data encrypted