

Forum: [Hack Forums](#)

Thread: [RIPv1 Amp Scanner Source \(Attack script source in a previous post\)](#)

L3GIT_ - September 11, 2015, 5:23 am

Hi HF today i'm releasing RIPv1 Amp Scanner source!

[illegible]

```

unsigned char buf[65536];
memset(buf, 0x01, payloadsize);
int sizeofpayload = payloadsize;
int sock;
if((sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))<0) {
    perror("cant open socket");
    exit(-1);
}
for(w=ntohl(start_ip);w<htonl(end);w++)
{
    struct sockaddr_in servaddr;
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=htonl(w);
    servaddr.sin_port=htons(xport);
    sendto(sock,payload,size,0, (struct sockaddr *)&servaddr,sizeof(servaddr));
    bytes_sent+=size;
    scanned++;
    hosts_done++;
}
close(sock);
running_threads--;
return;
}

void sighandler(int sig)
{
    fclose(fd);
    printf("\n");
    exit(0);
}

void *recievethread()
{
    printf("\n");
    int saddr_size, data_size, sock_raw;
    struct sockaddr_in saddr;
    struct in_addr in;

    unsigned char *buffer = (unsigned char *)malloc(65536);
    sock_raw = socket(AF_INET , SOCK_RAW , IPPROTO_UDP);
    if(sock_raw < 0)
    {
        printf("Socket Error\n");
        exit(1);
    }
    while(1)
    {
        saddr_size = sizeof saddr;
        data_size = recvfrom(sock_raw , buffer , 65536 , 0 , (struct sockaddr *)&saddr , &saddr_size);
        if(data_size < 0 )
        {
            printf("Recvfrom error , failed to get packets\n");

```

```

        exit(1);
    }
    struct iphdr *iph = (struct iphdr*)buffer;
    if(iph->protocol == 17)
    {
        unsigned short iphdrlen = iph->ihl*4;
        struct udphdr *udph = (struct udphdr*)(buffer + iphdrlen);
        unsigned char* payload = buffer + iphdrlen + payloadsize;
        if(ntohs(udph->source) == xport)
        {
            int body_length = data_size - iphdrlen - payloadsize;

            if (body_length > 40)

            {
                found_srvs++;

                fprintf(fd,"%s %d\n",inet_ntoa(saddr.sin_addr),body_length);
                fflush(fd);

            }

        }
    }

}
close(sock_raw);

}

int main(int argc, char *argv[ ])
{
    if(argc < 6){
        fprintf(stderr, "Invalid parameters!\n");
        fprintf(stdout, "RIP Scanner\nUsage: %s <ip range start (192.0.0.0)> <ip range end (198.255.255.255)> <outfile> <threads> <scan delay in ms>\n", argv[0]);
        exit(-1);
    }
    fd = fopen(argv[3], "a");
    sleep_between = atoi(argv[5]);

    signal(SIGINT, &sighandler);

    int threads = atoi(argv[4]);
    pthread_t thread;

    pthread_t listenthread;
    pthread_create( &listenthread, NULL, &recievethread, NULL);

    char *str_start = malloc(18);
    memset(str_start, 0, 18);
    str_start = argv[1];

```

```

char *str_end = malloc(18);
memset(str_end, 0, 18);
str_end = argv[2];
start = inet_addr(str_start);
per_thread = (ntohl(inet_addr(str_end)) - ntohl(inet_addr(str_start))) / threads;
unsigned long toscan = (ntohl(inet_addr(str_end)) - ntohl(inet_addr(str_start)));
int i;
for(i = 0; i < threads; i++){
    pthread_create(&thread, NULL, &flood, (void *) i);
}
sleep(1);
printf("Scan in Progress \n");
char *temp = (char *)malloc(17);
memset(temp, 0, 17);
sprintf(temp, "Reflectors");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "IP/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Bytes/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Threads");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Percent Done");
printf("%s", temp);
printf("\n");

char *new;
new = (char *)malloc(16*6);
while (running_threads > 0)
{
    printf("\r");
    memset(new, '\0', 16*6);
    sprintf(new, "%s|%-15lu", new, found_srvs);
    sprintf(new, "%s|%-15d", new, scanned);
    sprintf(new, "%s|%-15d", new, bytes_sent);
    sprintf(new, "%s|%-15d", new, running_threads);
    memset(temp, 0, 17);
    int percent_done = ((double)(hosts_done) / (double)(toscan)) * 100;
    sprintf(temp, "%d%%", percent_done);
    sprintf(new, "%s|%-15s", new, temp);
    printf("%s", new);
    fflush(stdout);
    bytes_sent = 0;
    scanned = 0;
    sleep(1);
}
printf("\n");
fclose(fd);
return 0;

```



```

memset(buf, 0x01, payloadsize);
int sizeofpayload = payloadsize;
int sock;
if((sock=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))<0) {
    perror("cant open socket");
    exit(-1);
}
for(w=ntohl(start_ip);w<htonl(end);w++)
{
    struct sockaddr_in servaddr;
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr=htonl(w);
    servaddr.sin_port=htons(xport);
    sendto(sock,payload,size,0, (struct sockaddr *)&servaddr,sizeof(servaddr));
    bytes_sent+=size;
    scanned++;
    hosts_done++;
}
close(sock);
running_threads--;
return;
}

```

```

void sighandler(int sig)
{
    fclose(fd);
    printf("\n");
    exit(0);
}

```

```

void *recievethread()
{
    printf("\n");
    int saddr_size, data_size, sock_raw;
    struct sockaddr_in saddr;
    struct in_addr in;

    unsigned char *buffer = (unsigned char *)malloc(65536);
    sock_raw = socket(AF_INET , SOCK_RAW , IPPROTO_UDP);
    if(sock_raw < 0)
    {
        printf("Socket Error\n");
        exit(1);
    }
    while(1)
    {
        saddr_size = sizeof saddr;
        data_size = recvfrom(sock_raw , buffer , 65536 , 0 , (struct sockaddr *)&saddr , &saddr_size);
        if(data_size < 0 )
        {
            printf("Recvfrom error , failed to get packets\n");
            exit(1);
        }
    }
}

```

```

    }
    struct iphdr *iph = (struct iphdr*)buffer;
    if(iph->protocol == 17)
    {
        unsigned short iphdrlen = iph->ihl*4;
        struct udphdr *udph = (struct udphdr*)(buffer + iphdrlen);
        unsigned char* payload = buffer + iphdrlen + payloadsize;
        if(ntohs(udph->source) == xport)
        {
            int body_length = data_size - iphdrlen - payloadsize;

            if (body_length > 40)

            {
                found_srvs++;

                fprintf(fd,"%s %d\n",inet_ntoa(saddr.sin_addr),body_length);
                fflush(fd);

            }

        }
    }

    close(sock_raw);
}

int main(int argc, char *argv[ ])
{
    if(argc < 6){
        fprintf(stderr, "Invalid parameters!\n");
        fprintf(stdout, "RIP Scanner\nUsage: %s <ip range start (192.0.0.0)> <ip range end (198.255.255.255)> <outfile> <threads> <scan delay in ms>\n", argv[0]);
        exit(-1);
    }
    fd = fopen(argv[3], "a");
    sleep_between = atoi(argv[5]);

    signal(SIGINT, &sighandler);

    int threads = atoi(argv[4]);
    pthread_t thread;

    pthread_t listenthread;
    pthread_create( &listenthread, NULL, &recievethread, NULL);

    char *str_start = malloc(18);
    memset(str_start, 0, 18);
    str_start = argv[1];
    char *str_end = malloc(18);

```

```

memset(str_end, 0, 18);
str_end = argv[2];
start = inet_addr(str_start);
per_thread = (ntohl(inet_addr(str_end)) - ntohl(inet_addr(str_start))) / threads;
unsigned long toscan = (ntohl(inet_addr(str_end)) - ntohl(inet_addr(str_start)));
int i;
for(i = 0; i < threads; i++){
    pthread_create( &thread, NULL, &flood, (void *) i);
}
sleep(1);
printf("Scan in Progress \n");
char *temp = (char *)malloc(17);
memset(temp, 0, 17);
sprintf(temp, "Reflectors");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "IP/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Bytes/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Threads");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Percent Done");
printf("%s", temp);
printf("\n");

char *new;
new = (char *)malloc(16*6);
while (running_threads > 0)
{
    printf("\r");
    memset(new, '\0', 16*6);
    sprintf(new, "%s|%-15lu", new, found_srvs);
    sprintf(new, "%s|%-15d", new, scanned);
    sprintf(new, "%s|%-15d", new, bytes_sent);
    sprintf(new, "%s|%-15d", new, running_threads);
    memset(temp, 0, 17);
    int percent_done = ((double)(hosts_done) / (double)(toscan)) * 100;
    sprintf(temp, "%d%%", percent_done);
    sprintf(new, "%s|%-15s", new, temp);
    printf("%s", new);
    fflush(stdout);
    bytes_sent = 0;
    scanned = 0;
    sleep(1);
}
printf("\n");
fclose(fd);
return 0;
}

```

You know you cannot call this a release , right?