

Forum:[Hack Forums](#)

Thread: [Scan IP Ranges for SSDP](#)

DerekM - February 17, 2015, 3:39 pm

And i quoted it so you cannot edit it and take it out haha

Max Resist - February 17, 2015, 4:41 pm

(Feb 17, 2015 04:39 PM)DerekM Wrote: [\[CITE\]](#) Figured I would share with the community, only thing I ask is no PMs asking for help/support.

I saw a few copies floating around with some malware added, I removed it.

Usage: %s <ip range start (192.168.0.0)> <ip range end (192.168.255.255)> <outfile> <threads>
<scan delay in ms>

```
Code #include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <net/ethernet.h>
#include <netinet/udp.h>
#include <netinet/ip.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <sys/resource.h>
#include <unistd.h>
```

```
void process_packet(void *args, struct pcap_pkthdr *header, void *buffer);
```

```
struct buffer
{
    void *data;
    int size;
    struct buffer *next;
    struct buffer *prev;
};
struct buffer *head;
```

```
char *ipv4;
int processed,over,total,i,j;
struct sockaddr_in dest;
pthread_mutex_t buf_mutex = PTHREAD_MUTEX_INITIALIZER;
sem_t loop_sem;
int running_threads = 0;
volatile int found_srvs = 0;
```

```
volatile unsigned long per_thread = 0;
volatile unsigned long start = 0;
volatile unsigned long scanned = 0;
int sleep_between = 0;
volatile int bytes_sent = 0;
volatile unsigned long hosts_done = 0;
FILE *fd;
```

```
void *readthread()
{
    struct buffer *ourhead = head;
    struct sockaddr_in saddr;
    while(1)
    {
        sem_wait(&loop_sem);
        while(ourhead->data == NULL){ ourhead = ourhead->next; }
        pthread_mutex_lock(&buf_mutex);
        void *buf = malloc(ourhead->size);
        int size = ourhead->size;
        memcpy(buf, ourhead->data, ourhead->size);
        free(ourhead->data);
        ourhead->data = NULL;
        ourhead->size = 0;
        pthread_mutex_unlock(&buf_mutex);
        memset(&saddr, 0, sizeof(saddr));
        struct iphdr *iph = (struct iphdr*)(buf + sizeof(struct ethhdr));
        saddr.sin_addr.s_addr = iph->saddr;
        struct udphdr *udph = (struct udphdr*)(buf + sizeof(struct ethhdr) + sizeof(struct iphdr));
        if(ntohs(udph->source) == 1900)
        {
            int body_length = size - sizeof(struct ethhdr) - sizeof(struct iphdr) - sizeof(struct udphdr);
            fprintf(fd,"%s %d\n",inet_ntoa(saddr.sin_addr),body_length);
            fflush(fd);
            found_srvs++;
        }
        free(buf);
        processed++;
        ourhead = ourhead->next;
    }
}
```

```
void *flood(void *par1)
{
    running_threads++;
    int thread_id = (int)par1;
    unsigned long start_ip = htonl(ntohl(start)+(per_thread*thread_id));
    unsigned long end = htonl(ntohl(start)+(per_thread*(thread_id+1)));
    unsigned long w;
    int y;
    unsigned char buf[65536];
    strcpy(buf, "M-SEARCH *
HTTP/1.1\r\nHost:239.255.255.250:1900\r\nST:ssdp:all\r\nMan:\"\ssdp:discover\"\r\nMX:3\r\n\r\n");
    int sizeofpayload = 90;
```

```

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(1900);
sendto(sock,(char *)buf,sizeofpayload,0, (struct sockaddr *)&servaddr,sizeof(servaddr));
bytes_sent+=sizeofpayload;
scanned++;
hosts_done++;
usleep(sleep_between*1000);
}
close(sock);
running_threads--;
return;
}

```

```

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

```

```

void *printthread(void *argvs)
{
    char **argv = (char **)argvs;
    int threads = atoi(argv[4]);
    pthread_t thread;
    sleep(1);
    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("Starting Scan...\n");
    char *temp = (char *)malloc(17);
    memset(temp, 0, 17);
}

```

```

sprintf(temp, "Found");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Host/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "B/s");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "Running Thrds");
printf("%-16s", temp);
memset(temp, 0, 17);
sprintf(temp, "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);
exit(0);

```

```

}

```

```

int main(int argc, char *argv[ ])

```

```

{

```

```

    if(argc < 6){

```

```

        fprintf(stderr, "Invalid parameters!\n");

```

```

        fprintf(stdout, "Usage: %s <ip range start (192.168.0.0)> <ip range end (192.168.255.255)>

```

```

<outfile> <threads> <scan delay in ms>\n", argv[0]);

```

```

        exit(-1);

```

```

    }

```

```

    fd = fopen(argv[3], "a");

```

```

    sleep_between = atoi(argv[5]);

```

```

    int num_threads = atoi(argv[4]);

```

```

const rlim_t kOpenFD = 1024 + (num_threads * 2);
struct rlimit rl;
int result;
rl.rlim_cur = kOpenFD;
rl.rlim_max = kOpenFD;
result = setrlimit(RLIMIT_NOFILE, &rl);
if (result != 0)
{
    perror("setrlimit_nofile");
    fprintf(stderr, "setrlimit_nofile returned result = %d\n", result);
}
bzero(&rl, sizeof(struct rlimit));
rl.rlim_cur = 256 * 1024;
rl.rlim_max = 4096 * 1024;
result = setrlimit(RLIMIT_STACK, &rl);
if (result != 0)
{
    perror("setrlimit_stack");
    fprintf(stderr, "setrlimit_stack returned result = %d\n", result);
}

signal(SIGINT, &sighandler);

pcap_if_t *alldevsp;
pcap_t *handle; //Handle of the device that shall be sniffed

char errbuf[100] , *devname , devs[100][100];
int count = 1 , n;

if( pcap_findalldevs( &alldevsp , errbuf ) )
{
    exit(1);
}

devname = alldevsp->name;
ipv4 = malloc(16);
bzero(ipv4, 16);
struct ifreq ifc;
int res;
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

if(sockfd < 0) exit(-1);
strcpy(ifc.ifr_name, devname);
res = ioctl(sockfd, SIOCGIFADDR, &ifc);
close(sockfd);
if(res < 0) exit(-1);
strcpy(ipv4, inet_ntoa(((struct sockaddr_in*)&ifc.ifr_addr)->sin_addr));
printf("Opening device %s for sniffing ... " , devname);
handle = pcap_open_live(devname , 65536 , 1 , 0 , errbuf);

if (handle == NULL)
{

```

```

fprintf(stderr, "Couldn't open device %s : %s\n" , devname , errbuf);
exit(1);
}
printf("Done\n");

sem_init(&loop_sem, 0, -1);
i = 1024*1000;
while(i--)
{
if(head == NULL)
{
head = (struct buffer *)malloc(sizeof(struct buffer));
bzero(head, sizeof(struct buffer));
head->data = NULL;
head->size = 0;
head->next = head;
head->prev = head;
} else {
struct buffer *new_node = (struct buffer *)malloc(sizeof(struct buffer));
bzero(new_node, sizeof(struct buffer));
new_node->data = NULL;
new_node->size = 0;
new_node->prev = head;
new_node->next = head->next;
head->next = new_node;
}
}

pthread_t prnthread;
pthread_create( &prnthread, NULL, &printthread, (void *)argv);
pthread_t redthread;
pthread_create( &redthread, NULL, &readthread, NULL);

pcap_loop(handle , -1 , process_packet , NULL);

return 0;
}

void process_packet(void *args, struct pcap_pkthdr *header, void *buffer)
{
int size = header->len;

//Get the IP Header part of this packet , excluding the ethernet header
struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
memset(&dest, 0, sizeof(dest));
dest.sin_addr.s_addr = iph->daddr;

if(iph->protocol == 17 && strcmp(inet_ntoa(dest.sin_addr), ipv4) == 0)
{
//toss into buffer
if(head->data != NULL) over++;
pthread_mutex_lock(&buf_mutex);
void *temp = malloc(size);

```

```
memcpy(temp, buffer, size);
head->data = temp;
head->size = size;
head = head->next;
pthread_mutex_unlock(&buf_mutex);
sem_post(&loop_sem);
total++;
}
}Nice c/p bro, i fucking love it :)
```

And i quoted it so you cannot edit it and take it out haha

0xDEAD10CC - February 17, 2015, 7:27 pm

OP Wrote:only thing I ask is no PMs asking for help/support

The only reason one person would write something like this is because they didn't write the code; you can't help with any of the issues people might be having. I wrote an SSDP implementation, I bet you don't understand how SSDP even works. I'm doubtful that you even edited any of this source as you claim.

Zettabit - February 17, 2015, 7:34 pm

"A few copies with malware"

You completely C+P'ed this whole thing. You didn't "remove" anything.