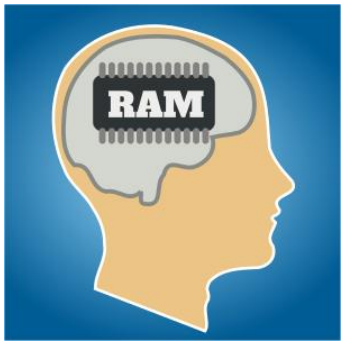
Resolução de nomes

Autor: Patricio Domingues
Atualizado: 2025

Identificar recursos

- O encaminhamento IP faz uso de endereços IPs (IPv4 e IPv6)
- Contudo, o ser humano tem uma capacidade (bastante) limitada para lidar com números (IPs, NIF, BI, etc.)
- Nomes surgem como alternativa aos endereços IPs
  - Ser humano tem mais aptidão para organizar/lembrar nomes
  - Nomes permitem associações (e.g, PT=Portugal, ipleiria=Politécnico de Leiria)



DNS

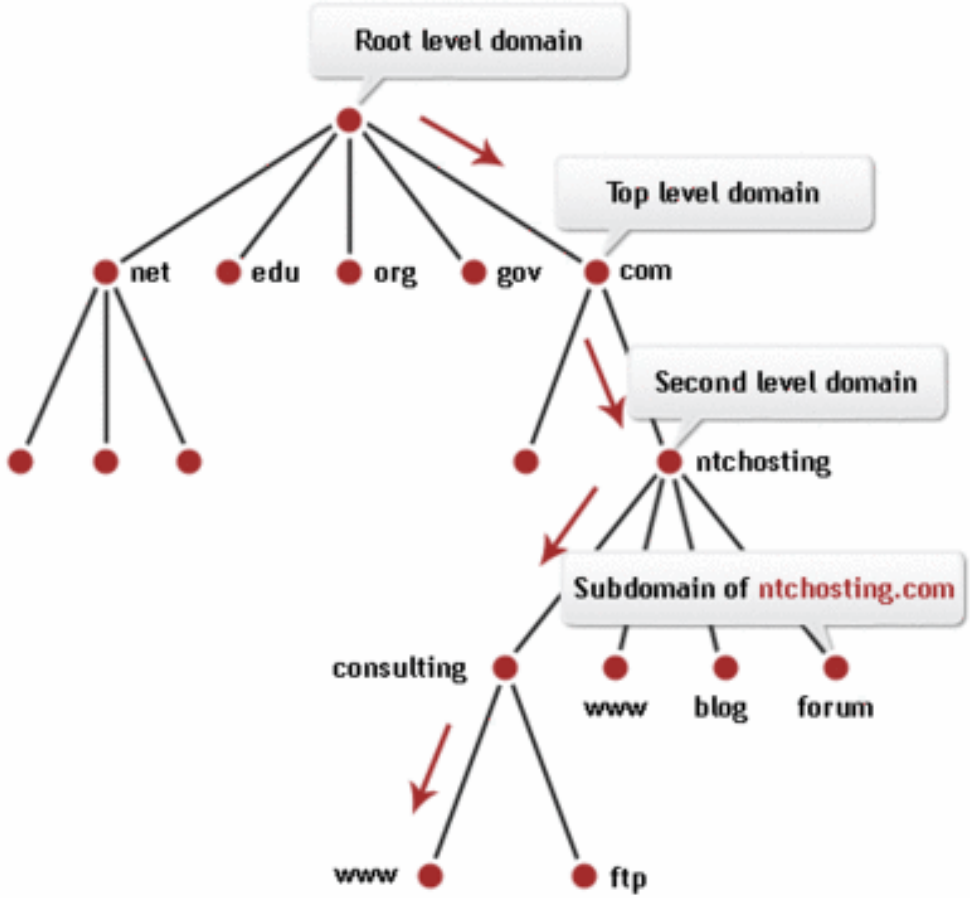
- DNS - Domain Name System
  - Empregue principalmente para mapear nomes para endereços IPs e vice-versa
    - Também tem papel importante no sistema de email (registos MX, TXT e protocolos DKIM, SPF, DANE, ARCS)
  - Nome pode ser *nome simples* ou *nome completo* (*Fully Qualified Domain Name – FQDN*)
- O DNS é uma base de dados distribuída
  - Cada domínio é responsável pela sua presença no DNS
    - Exemplo
      - Politécnico de Leiria é responsável pelo domínio `ipleiria.pt`

Observações sobre o DNS…

- Base de dados distribuída com milhões de milhões de registos
- DNS lida com trilhões de consultas/dia:
  - Número de Leituras >> número de escritas
    - O serviço 1.1.1.1 (cloudfare) trata cerca de 1.3×10^{12} pedidos DNS por dia
- Desempenho é muito importante
  - Quase todas as transações da Internet interagem com o DNS
- Sistema distribuído
  - Milhões de organizações diferentes responsáveis pelos seus registos e bom funcionamento
  - Distribuição física e organizacional

Hierarquia DNS (#1)

- Nó raiz tem nome "." (ponto)
- nome de domínio
  - lista de etiquetas de nós ligadas por pontos e lidas de baixo para cima
- Um domínio é um segmento (sub-árvore da hierarquia)
  - Cada instituição dona de um domínio é responsável pelo respetivo DNS
- Exemplo: domínio ipleiria.pt.
  - Ponto no fim é (usualmente) facultativo
  - Ponto no fim refere-se à zona raíz "root zone" (.)



Zona raiz

- Zona raiz
  - Identificada pelo ponto (dot)
  - Função é:
    1) Responder aos pedidos DNS da zona raiz
    2) Responder aos pedidos DNS para domínios de topo
       - TLD: top-level domain
- Clientes DNS
  - Permitem efetuar pedidos DNS
    - Testes à infraestrutura
  - Windows: `nslookup`
  - Unix: `host,dig`
    - `Também existe o nslookup`

- Domínios da zona raiz

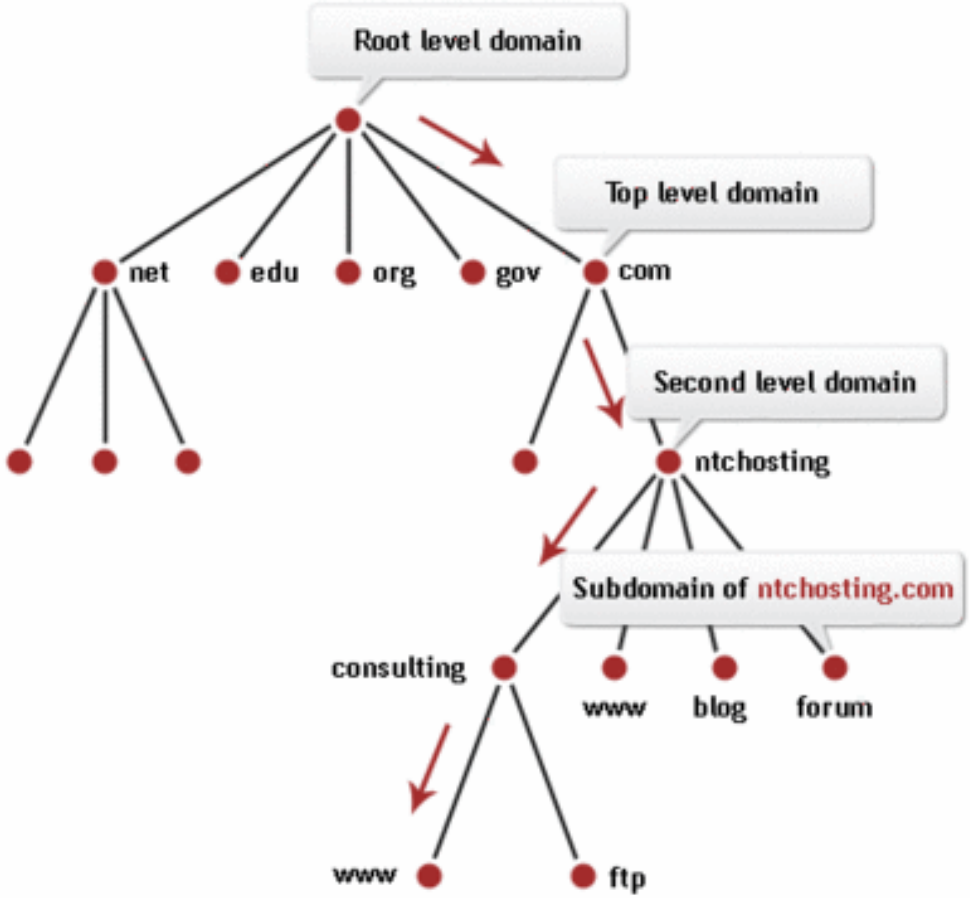
  `nslookup -type=NS .`

```
C:\>nslookup -type=NS .
Server:  dc3c2.ipleiria.pt
Address:  172.20.1.12

Non-authoritative answer:
(root)  nameserver = h.root-servers.net
(root)  nameserver = i.root-servers.net
(root)  nameserver = j.root-servers.net
(root)  nameserver = k.root-servers.net
(root)  nameserver = l.root-servers.net
(root)  nameserver = m.root-servers.net
(root)  nameserver = a.root-servers.net
(root)  nameserver = b.root-servers.net
(root)  nameserver = c.root-servers.net
(root)  nameserver = d.root-servers.net
(root)  nameserver = e.root-servers.net
(root)  nameserver = f.root-servers.net
(root)  nameserver = g.root-servers.net

h.root-servers.net      internet address = 198.97.190.53
j.root-servers.net      internet address = 192.58.128.30
```

```
C:\Users\user>nslookup www.facebook.com 8.8.8.8
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
Name:    star-mini.c10r.facebook.com
Addresses:  2a03:2880:f152:82:face:b00c:0:25de
          157.240.212.35
Aliases:  www.facebook.com
```

Hierarquia DNS (#2)

- Limites
  - Hierarquia
    - máximo de 127 níveis
  - Comprimento do nome individual de cada nó
    - 63 octetos
  - Comprimento do nome completo
    - 255 octetos
  - Nome completo
    - Fully qualified Domain Name (FQDN)
- Porquê octetos e não caracteres?
  - DNS suporta Unicode
    - IDN: International Domain Name
  - Exemplos: السعودية
    - (TLD, significa Saudi)



8 caracteres, 16 octetos

Interação com o DNS (#1)

- No Linux, o utilitário ***host*** permite interagir com o sistema DNS (cliente DNS)

```
$ host ipleiria.pt
ipleiria.pt has address 194.210.216.8
ipleiria.pt mail is handled by 10 mx02.ipleiria.pt.
ipleiria.pt mail is handled by 10 mx01.ipleiria.pt.
$ host www.google.com
www.google.com has address 216.58.210.228
www.google.com has IPv6 address 2a00:1450:4006:804::2004

$ host www.gmail.com 8.8.8.8              # servidor DNS 8.8.8.8
www.gmail.com is an alias for mail.google.com.
mail.google.com is an alias for googlemail.l.google.com.
googlemail.l.google.com has address 216.58.210.229
googlemail.l.google.com has IPv6 address 2a00:1450:4006:803::2005
```

Interação com o DNS (#2)

- No Windows, o utilitário ***nslookup*** permite interagir com o sistema DNS

```
$ nslookup -type=ANY ipleiria.pt. 9.9.9.9
Server:  dns.quad9.net
Address:  9.9.9.9
Non-authoritative answer:
ipleiria.pt     internet address = 194.210.216.28
ipleiria.pt     nameserver = ns2.ipleiria.pt
ipleiria.pt     nameserver = ns.ipleiria.pt$ host www.google.com
```

```
$ nslookup –type=A ead.ipleiria.pt 8.8.4.4
Server:  google-public-dns-b.google.com
Address:  8.8.4.4
Non-authoritative answer:
Name:    ead.ipleiria.pt
Address:  194.210.216.35
```

Pedidos DNS - exemplos

- Pedido dos endereços IPv6 de www.Linux.org, usando o servidor DNS 8.8.4.4

```
C:\Users\user>nslookup -type=AAAA www.linux.org 8.8.4.4
Server:  dns.google
Address:  8.8.4.4

Non-authoritative answer:
Name:    www.linux.org
Addresses:  2606:4700:3030::681b:a7db
          2606:4700:3031::681b:a6db
          2606:4700:3030::ac43:99d2
```

- Pedido dos endereços IP de www.Linux.org, usando o cliente "host" e o servidor DNS 1.1.1.1

```
user@so-vm:~$ host www.linux.org 1.1.1.1
Using domain server:
Name: 1.1.1.1
Address: 1.1.1.1#53
Aliases:

www.linux.org has address 104.27.166.219
www.linux.org has address 104.27.167.219
www.linux.org has address 172.67.153.210
www.linux.org has IPv6 address 2606:4700:3030::681b:a7db
www.linux.org has IPv6 address 2606:4700:3031::681b:a6db
www.linux.org has IPv6 address 2606:4700:3030::ac43:99d2
```

- Pedido dos servidores de nomes do domínio .pt usando o nslookup e o servidor DNS 9.9.9.9

```
C:\Users\user>nslookup -type=NS pt. 9.9.9.9
Server:  dns9.quad9.net
Address:  9.9.9.9

Non-authoritative answer:
pt      nameserver = d.dns.pt
pt      nameserver = ns2.nic.fr
pt      nameserver = c.dns.pt
pt      nameserver = g.dns.pt
pt      nameserver = h.dns.pt
pt      nameserver = e.dns.pt
pt      nameserver = a.dns.pt
pt      nameserver = ns.dns.br
pt      nameserver = b.dns.pt
```

Servidores da zona raiz (“.”)

- Obter a lista dos 13 servidores de raiz

```
nslookup -type=NS .
(root)  nameserver = f.root-servers.net
(root)  nameserver = g.root-servers.net
(root)  nameserver = c.root-servers.net
(root)  nameserver = b.root-servers.net
(root)  nameserver = d.root-servers.net
(root)  nameserver = a.root-servers.net
(root)  nameserver = m.root-servers.net
(root)  nameserver = h.root-servers.net
(root)  nameserver = i.root-servers.net
(root)  nameserver = j.root-servers.net
(root)  nameserver = k.root-servers.net
(root)  nameserver = l.root-servers.net
(root)  nameserver = e.root-servers.net
```

PROTOCOLO DNS

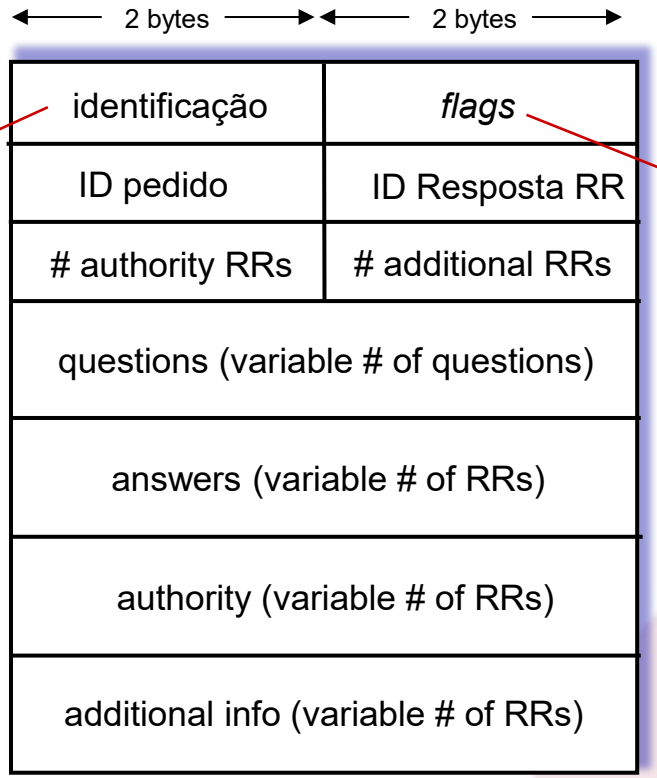
Protocolo DNS

- O protocolo aplicacional DNS recorre, preferencialmente, ao protocolo de transporte UDP
- Porto de servidor DNS
  - 53 em UDP e TCP
- Protocolo DNS segue modelo cliente/servidor
- Pedidos/respostas no protocolo DNS estão limitados a 512 octetos
  - Garantir que não existe fragmentação na camada IP
  - Fragmentação na camada IP deteriora significativamente a entrega de UDP
- Caso a resposta do servidor ultrapasse os 512 octetos, servidor DNS envia:
  - resposta parcial (<= 512 octetos)
  - Indicação que a resposta completa pode ser obtida pelo cliente ligando-se via protocolo de transporte TCP
- TCP também pode ser empregue quando a rede local está a bloquear o UDP/53

DNS protocol messages

- As consultas DNS e respetivas mensagens de resposta, uso o mesmo formato de mensagem
  - As mensagens são do tipo binário

Cabeçalho da mensagem:
identification: 16 bits ID para consulta, a resposta à consulta usa o mesmo ID



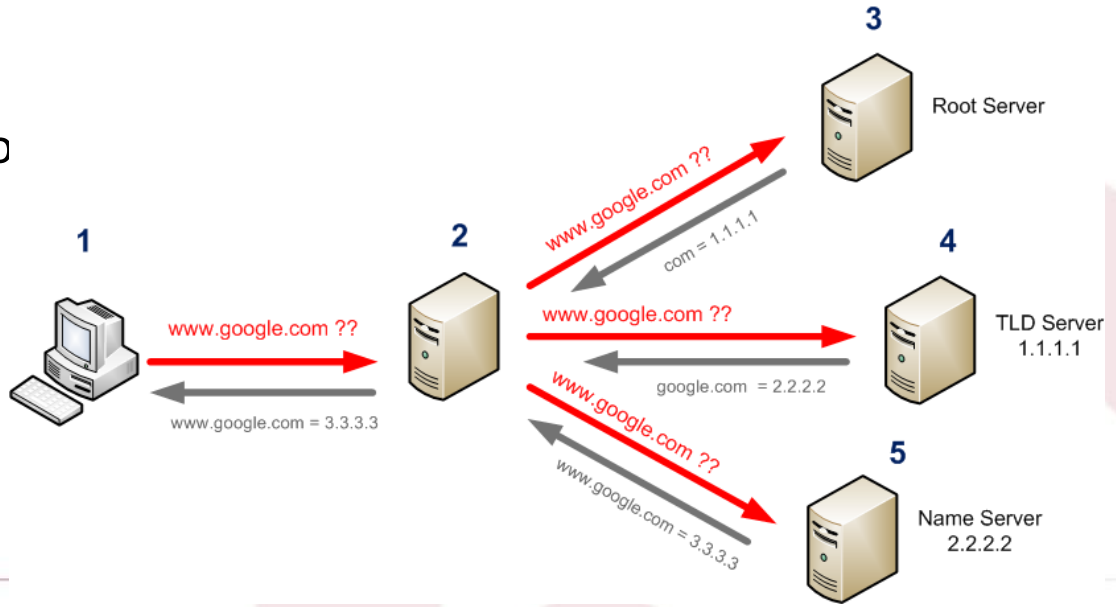
Cabeçalho da mensagem:
flags:
- Consulta ou resposta
- Solicita recursividade
- Recursividade disponível
- Resposta autoritária

Adaptado de:
Computer Networking: A Top-Down Approach
8th edition, Jim Kurose, Keith Ross, Pearson, 2020

Exemplo: resolução de nomes

- Cliente (e.g., navegador) procura IP
  do nome www.google.com
  1. SO contacta servidor de nomes local
  2. DNS local contacta servidor do domínio de raiz (e.g., a.root-servers.net)
     - Objetivo: obter endereço de **www.google.com** ou do servido de nomes .com
  3. Servidor *root* responde com endereço servidor DNS de **.com**

4. DNS local contacta servidor DNS **.com** pedindo IP de **www.google.com**
5. Servidor .com devolve endereços dos servidores de nomes **google.com**
6. DNS local contacta servidor de nomes **google.com** pedindo IP de **www.google.com**

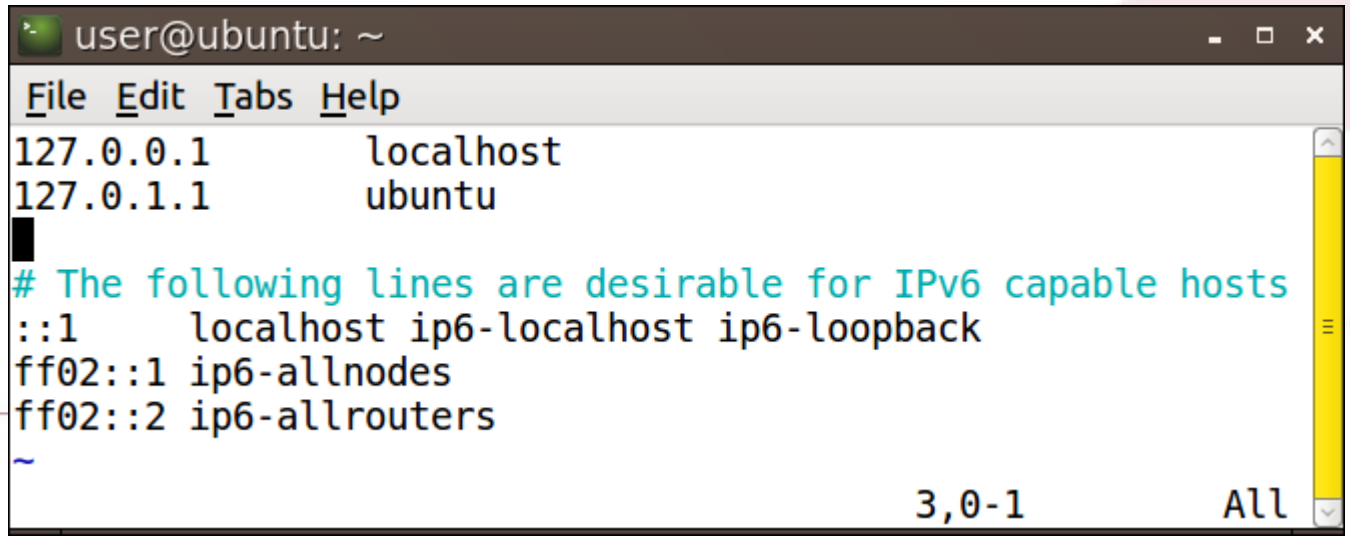


Linux: Ficheiro `hosts`

- Ficheiro estático que contém mapeamento entre nomes e endereços IP
  - /etc/hosts (Linux)
- Muitas vezes apenas contém localhost
  - `127.0.0.1`
  - `::1` (IPv6)

- Ficheiro tem prioridade sobre a resolução de nomes...
- Pode ser empregue para *DNS hijacking*
- Exemplo

  127.0.0.1 www.google.com
  - Google deixa de estar acessível



Windows: `hosts` e *cache*

- Localização em sistemas Windows
  - Varia consoante a versão
  - Vista, W7, W8 e W10

  –%**WinDir**%\System32\Drivers\Etc

```
# hosts (windows 7/8/10)
# localhost name resolution
# is handle within DNS itself.
# 127.0.0.1 localhost
# ::1 localhost
```

- Edição requer permissão de administrador

- Windows tem cache DNS
  - *Serviço dnscache*
- `sc query dnscache`

```
SERVICE_NAME: dnscache
 TYPE               : 30  WIN32
 STATE              : 4  RUNNING
    (NOT_STOPPABLE, NOT_PAUSABLE,
                IGNORES_SHUTDOWN)
 WIN32_EXIT_CODE    : 0  (0x0)
SERVICE_EXIT_CODE  : 0  (0x0)
CHECKPOINT         : 0x0
 WAIT_HINT         : 0x0
```

- Conteúdo da cache DNS

  `ipconfig /displaydns`
- Limpar a cache DNS

  `ipconfig /flushdns`
  - **Nota:** a cache DNS do Windows é sempre carregada com o conteúdo do ficheiro **hosts**

Servidores DNS

- Alguns servidores DNS "conhecidos"
  - Responsabilidade da google
    - 8.8.8.8
    - 8.8.4.4
  - *Responsabilidade da cloudfare*
    - 1.1.1.1 e 1.0.0.1
    - One.one.one.one
  - *Quad Nine*
    - 9.9.9.9 ("quad9")
    - 149.112.112.112 (secundário)
    - Filtra "threat list"
    - Ver: http://bit.ly/2BqSf14

https://twitter.com/arlojamesbarnes

- Lista de servidores DNS públicos
  - http://bit.ly/2py3ax1

Importância DNS: *hijack* a banco

- Fraude online: redirecionamento de todos serviços *online* de um banco!

ANDY GREENBERG SECURITY 04.04.17 10:52 AM

HOW HACKERS HIJACKED A BANK'S ENTIRE ONLINE OPERATION

WIRED

THE TRADITIONAL MODEL of hacking a bank isn't so different from the old-fashioned method of robbing one. Thieves get in, get the goods, and get out. But one enterprising group of hackers targeting a Brazilian bank seems to have taken a more comprehensive and devious approach: One weekend afternoon, they rerouted all of the bank's online customers to perfectly reconstructed fakes of the bank's properties, where the marks obediently handed over their account information.

Researchers at the security firm Kaspersky on Tuesday described an unprecedented case of wholesale bank fraud, one that essentially hijacked a bank's entire internet footprint. At 1 pm on October 22 of last year, the researchers say, hackers changed the Domain Name System registrations of all 36 of the bank's online properties, commandeering the bank's desktop and mobile website domains to take users to phishing sites. In practice, that meant the hackers could steal login credentials at sites hosted at the bank's legitimate web addresses. Kaspersky researchers believe the hackers may have even simultaneously redirected all transactions at ATMs or point-of-sale systems to their own servers, collecting the credit card details of anyone who used their card that Saturday afternoon.

"Absolutely all of the bank's online operations were under the attackers' control for five to six hours," says Dmitry Bestuzhev, one of the Kaspersky researchers who analyzed the attack in real time after seeing malware infecting customers from what appeared to be the bank's fully valid domain. From the hackers' point of view, as Bestuzhev puts it, the DNS attack meant that "you become the bank. Everything belongs to you now."

- Criaram certificado digital
  - Let's Encrypt
- E ainda infetaram clientes com *malware*: roubo de credenciais

Fonte: https://www.wired.com/2017/04/hackers-hijacked-banks-entire-online-operation/

API DE ACESSO À RESOLUÇÃO DE NOMES

API Posix

- É primeiro estudada a agora obsoleta API "**gethostbyname**" e "**gethostbyaddr**"

```
DESCRIPTION
       The gethostbyname*(), gethostbyaddr*(), herror(), and hstrerror()
       functions are obsolete. Applications should use getaddrinfo(3),
       getnameinfo(3), and gai_strerror(3) instead.
```

Obsoleto: uso desaconselhado

- API "gethost…" ainda subsiste em código legado
- Atualmente, é aconselhada a utilização da API POSIX **getaddrinfo/getnameinfo**

Função **gethostbyname** (#1)

- Função gethostbyname

```
struct hostent *gethostbyname(const char *name);
```

- Devolve
  - NULL se erro
    - Mas **não** usa o **errno**, mas sim **h_errno**
    - Variável **h_errno** recebe código numérico descritivo do erro
    - Função **const char *hstrerror(int err);** permite obter string descritiva do erro
  - Permite obter os endereços IPs associados a um determinado nome

struct hostent >>

Obsoleto: uso desaconselhado
(apenas IPv4, IPV6 nalguns casos)

Função **gethostbyname** (#2)

- Função gethostbyname

```
struct hostent *gethostbyname(const char *name);
```

```
struct hostent {
        char  *h_name; /* official name of host */
        char **h_aliases;  /* alias list */
        int    h_addrtype; /* host address type */
        int    h_length;   /* length of address */
        char **h_addr_list;/* list of addresses */
}
```

- `h_name:` nome oficial da máquina
- `h_aliases:` vetor de nomes alternativos, terminado por ponteiro NULL
- `h_addrtype:` tipo de endereço (AF_INET ou AF_INET6)
- `h_length:` tamanho do endereço em bytes
- `h_addr_list:` vetor de ponteiros para endereços de rede (formato de rede), terminado por NULL

Obsoleto: uso desaconselhado (apenas suporta IPv4)

Função **gethostbyaddr (#1)**

- struct hostent ***gethostbyaddr**(const void *addr, socklen_t len, int family);
  - Operação de *reverse-lookup* (designação anglo-saxónica)
  - Recebe um endereço **IPv4** (formato rede) e devolve, se existir, uma `struct hostent` com o nome correspondente
  - O endereço IPv4 deve encontrar-se no formato de rede
    - Não se pode especificar um endereço *dotted decimal*
  - Nota: implementação no Linux suporta IPv6

Exemplo >>

Obsoleto: uso desaconselhado

Função **gethostbyaddr (#2)**

```
#include <…>
int main(int argc, char **argv){
    struct in_addr ip;
    struct hostent *hp;
    if( argc != 2){
            fprintf(stderr,"%s <IP-address>\n", argv[0]);
            exit(1);
    }
    char *ipstr = argv[1];
    if(!inet_aton(ipstr, &ip)){
            fprintf(stderr,"can't parse IP address %s", ipstr);
            exit(1);
    }
    if((hp=gethostbyaddr((const void*)&ip,sizeof ip,AF_INET))==
NULL){
            fprintf(stderr, "no name associated with %s", ipstr);
            exit(1);
    }
    return 0;
}
```

gethostbyaddr IPv4 - exemplo

```
#include <...>
int main(int argc, char **argv){
        struct in_addr ip;
        struct hostent *hp;
        if( argc != 2){
                fprintf(stderr,"%s <IP-address>\n", argv[0]);
                exit(1);
        }
        char *ipstr = argv[1];
        if(!inet_aton(ipstr, &ip)){
                fprintf(stderr,"can't parse IP address %s", ipstr);
                exit(1);
        }
        if((hp=gethostbyaddr((const void*)&ip,sizeof ip,AF_INET))== NULL){
                fprintf(stderr, "no name associated with %s", ipstr);
                exit(1);
        }
        return 0;
}
```

Converte ASCII-dotted-decimal para formato de rede

IPv4 em formato de rede

Código completo: https://pastebin.com/CTSaf0Kw

gethostbyaddr IPv6 - exemplo

```c
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv){
        struct in6_addr ip6;
        struct hostent *hp;
        if( argc != 2){
                fprintf(stderr,"%s <IP-address>\n", argv[0]);
                exit(1);
        }
        char *ipstr = argv[1];
        if (!inet_pton(AF_INET6, ipstr, &ip6)){
                fprintf(stderr,"can't parse IP address %s", ipstr);
                exit(1);
        }
        if((hp=gethostbyaddr((const void *)&ip6,sizeof ip6,AF_INET6))==NULL){
                fprintf(stderr, "no name associated with %s", ipstr);
                exit(1);
        }
        printf("name associated with %s is %s\n", ipstr, hp->h_name);
        return 0;
}
```

Código completo: https://pastebin.com/qjdjxkCp

Versões reentrantes (#1)

- As versões originais de **`gethostbyname`** and **`gethostbyaddr`** **não** são reentrantes
  - Endereço da `struct hostent` devolvido corresponde a um zona de memória do tipo `static`
- Alternativas reentrantes (extensão `GLIBC`)
  - `int gethostbyname_r(const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop);`
  - `int gethostbyaddr_r(const void *addr, socklen_t len, int type, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop);`

Funcionamento das versões reentrantes >>

Obsoleto: uso desaconselhado

Versões reentrantes (#2)

- `int gethostbyname_r(const char *name, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop);`
- `int gethostbyaddr_r(const void *addr, socklen_t len, int type, struct hostent *ret, char *buf, size_t buflen, struct hostent **result, int *h_errnop);`
  - Após a chamada, **`result`** aponta para o resultado em caso de sucesso (NULL em caso de insucesso)
  - O utilizador passa:
    - (endereço de) `struct hostent` (`ret`) que é preenchida em caso de sucesso
    - buffer (**`buf`**) temporário com tamanho (**`buflen`**) suficiente para a escrita
      - Endereço apontado por **buf** é empregue com zona de memória para uso (temporário) da função
    - Caso **`buf`** não tenha tamanho suficiente, a função devolve **`buf`**, sendo necessário chamar a função com um **`buf`** de maior capacidade

Obsoleto: uso desaconselhado

Função getservbyname

- `struct servent *getservbyname(const char *name, const char *proto);`
- Devolve dados referentes a um serviço identificado pelo nome (e.g., ftp, NTP, ssh,…), i.e., *get service entry by name*
  - http://www.iana.org/assignments/port-numbers
- Devolve uma `struct servent`

```
struct servent {
        char  *s_name;          /* official service name */
        char **s_aliases;       /* alias list */
        int    s_port;          /* port number */
        char  *s_proto;         /* protocol to use */
    }
```

Parâmetro **buf**: Zona de memória passada pela entidade chamante para guardar os campos strings da `struct hostent` (s_name, s_aliases,s_proto)

- Existe uma versão reentrante
  - `int getservbyname_r(const char *name, const char *proto, struct servent *result_buf, char *buf, size_t buflen, struct servent **result);`

Obsoleto: uso desaconselhado

Função getservbyport

- `struct servent *getservbyport(int port, const char *proto);`
  - *Get service entry by port*
  - Procura um serviço correspondente ao porto indicado por «port »
  - Devolve (ponteiro para) struct servent caso o porto indicado corresponda a um serviço
- Versão reentrante
  - `int getservbyport_r(int port, const char *proto, struct servent *result_buf, char *buf, size_t buflen, struct servent **result);`

Obsoleto: uso desaconselhado

GETADDRINFO

Função getaddrinfo (#1)

- Função que…
  - Suporta IPv4 e IPv6
  - Suporta identificação serviço para *porto* e *vice-versa*
  - Substituição de várias funções da API anterior
  - Devolve estruturas de endereço do tipo `struct addrinfo`

```
struct addrinfo {
        int              ai_flags;
        int              ai_family;
        int              ai_socktype;
        int              ai_protocol;
        socklen_t        ai_addrlen;
        struct sockaddr *ai_addr;
        char            *ai_canonname;
        struct addrinfo *ai_next;
};
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
```

- `int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);`

Forma correta de resolução de nomes – API POSIX

Função getaddrinfo (#2)

- `int getaddrinfo(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);`
  - **node**: nome ou endereço IP da máquina
  - **service**: porto (e.g. "80") ou nome do serviço IANA (/etc/services) (e.g., "http", "smtp", etc.)
  - **hints**: estrutura `addrinfo` preenchida com informação relevante para o pedido
- Função devolve através do ponteiro '**res**', uma lista ligada de estruturas de endereços **`struct addrinfo`**
- O espaço alocado por **`getaddrinfo`** deve ser libertado com **`freeaddrinfo`**
  - `void freeaddrinfo(struct addrinfo *res);`

Função getaddrinfo (#3)

- int **getaddrinfo**(const char *node, const char *service, const struct addrinfo *hints, struct addrinfo **res);
- Exemplo: obter dados para efetuar *bind*

```
int status; struct addrinfo hints;
struct addrinfo *servinfo;          // Irá apontar para os resultados
memset(&hints, 0, sizeof hints);  // Esvazia estrutura
hints.ai_family = AF_UNSPEC; // indiferente: IPv4/IPv6 AF_INET/AF_INET6
hints.ai_socktype = SOCK_STREAM;  // socket TCP
hints.ai_flags = AI_PASSIVE;      // IP local preenchido pela função

if ((status = getaddrinfo(NULL, "3490", &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
        exit(1);
}
// servinfo aponta para uma lista ligada de um ou
// mais nós struct addrinfo
(…)
// ... Liberta lista ligada criada pela função getaddrinfo
freeaddrinfo(servinfo);
```

Fonte: http://beej.us/guide/bgnet/output/print/bgnet_A4.pdf

Função getaddrinfo (#4)

- A *struct addrinfo*
- *#include <netdb.h>*

```
 struct addrinfo {
int              ai_flags;    /* AI_PASSIVE, AI_CANONNAME,… */
int              ai_family;   /* AF_XXX */
int              ai_socktype; /* SOCK_XXX */
int              ai_protocol; /* 0 or IPPROTO_xxx */
socklen_t        ai_addrlen;  /* tamanho de ai_addr */
struct sockaddr *ai_addr;     /* ptr. p/ endereço */
char            *ai_canonname; /* ptr. nome canonical */
struct addrinfo *ai_next;     /* próx. Elemento lista ligada */
};
```

struct addrinfo

Element	Possible values	Comment
	AI_PASSIVE	Get address to use bind().
	AI_CANONNAME	Fill `ai_canonname`.
	AI_NUMERICHOST	Prevent name resolution on that address.
	AI_ALL	IPv6 and IPv4-mapped.
`ai_flags`	AI_V4MAPPED_CFG	Accept IPv4-mapped if kernel supports.
	AI_ADDRCONFIG	Only if any address is assigned.
	AI_V4MAPPED	Accept IPv4-mapped IPv6 address.
	AI_DEFAULT	AI_V4MAPPED_CFG | AI_ADDRCONFIG (special recommended flags for getipnodebyname).
`ai_family`	AF_INET	IPv4 address family.
	AF_INET6	IPv6 address family.
`ai_socktype`	SOCK_STREAM	TCP transport.
	SOCK_DGRAM	UDP transport.
	SOCK_SEQPACKET	SCTP transport.
`ai_protocol`	IPPROTO_IPV4	IPv4 protocol.
	IPPROTO_IPV6	IPv6 protocol.
`ai_addrlen`		Length of struct pointed by `ai_addr`.
`ai_canonname`		Hostname string.
`ai_addr`		IP Address itself.
`ai_next`		Pointer to next structure in the linked list of addresses.

Fonte: http://bit.ly/2g76nS7

Função getaddrinfo (#5)

- Tratamento de erros
  - A função **`getaddrinfo`** devolve um código numérico diferente de zero quando ocorre um erro
  - Pode ser obtida uma *string* descritiva do erro através da função **gai_strerror**:
    - `const char *gai_strerror(int errorcode);`
- Libertar recursos
  - A função **`getaddrinfo`** aloca recursos que o programa chamante tem que libertar
    - Função **`freeaddrinfo`**
      - `void freeaddrinfo(struct addrinfo *res);`

```c
/** showip.c -- show IP addresses for a host given on command line * Beej's Guide to Network Programming */
int main(int argc, char *argv[]) {
struct addrinfo hints, *res, *p;
int status;
char ipstr[INET6_ADDRSTRLEN];
if (argc != 2) {
        fprintf(stderr,"usage: showip hostname\n");
        return 1;
 }
 memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC; // AF_INET or AF_INET6 to force version
hints.ai_socktype = SOCK_STREAM;
if ((status = getaddrinfo(argv[1], NULL, &hints, &res)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status)); return 2;
}
printf("IP addresses for %s:\n\n", argv[1]);
for(p = res;p != NULL; p = p->ai_next) {
        void *addr; char *ipver; // get the pointer to the address itself,
        // different fields in IPv4 and IPv6:
        if (p->ai_family == AF_INET) {
                // IPv4
                struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
                addr = &(ipv4->sin_addr);
                ipver = "IPv4";
        } else {// IPv6
                struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
                addr = &(ipv6->sin6_addr);
                ipver = "IPv6";
        }
        // convert the IP to a string and print it
        inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
        printf(" %s: %s\n", ipver, ipstr);
}
freeaddrinfo(res); // free the linked list
return 0;
}
```

SOCK_STREAM ou SOCK_DGRAM
(se não for inicializado, devolve 3x o mesmo endereço)

- Exemplo
  - Mostra endereços IP (IPv4 e/ou IPv6) do nome indicado pela linha de comando
- Código fonte completo:

http://pasted.co/c7d02059

execução

Exemplo

```
user@so-vm:~$ ./showip_Inicializado.exe www.google.com
IP addresses for www.google.com:

 IPv4: 216.58.201.164
 IPv6: 2a00:1450:4003:80b::2004
user@so-vm:~$ ./showip_NaoInicializado.exe www.google.com
IP addresses for www.google.com:

 IPv4: 216.58.201.164
 IPv4: 216.58.201.164
 IPv4: 216.58.201.164
 IPv6: 2a00:1450:4003:80b::2004
 IPv6: 2a00:1450:4003:80b::2004
 IPv6: 2a00:1450:4003:80b::2004
```

ai_socktype inicializado

Com **ai_socktype <u>não</u>** inicializado (valor zero) getaddrinfo produz 3 registos

GETNAMEINFO()

Função **getnameinfo**

- Função complementar de **getaddrinfo**
  - Obtém dados em formato string referentes a um endereço
  - Endereço pode ser de cliente ou servidor
- Recebe uma estrutura de endereço preenchida (parâmetros **sa** e **salen**) e devolve:
  - String a descrever a máquina (parâmetros **host** e **hostlen**)
  - String a descrever o serviço (parâmetros **serv** e **servlen**)

- `int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host, socklen_t hostlen, char *serv, socklen_t servlen, int flags);`
  - Porque `socklen_t` e não `size_t` para `servlen`?
    - Por razões de compatibilidade com API socket BSD, o tipo de dado "servlen" deve ser um inteiro sem sinal de 32 bits. Contudo, dependendo da plataforma, size_t (inteiro sem sinal) poderá ter 32 ou 64 bits

EXEMPLO GETADDRINFO/GETNAMEINFO

getaddrinfo.c (#1)

```
#include <…>
char *proto_to_str(int proto){
        if( proto == IPPROTO_TCP ){
                return "IPPROTO_TCP";
        }else if( proto == IPPROTO_UDP ){
                return "IPPROTO_UDP";
        }else{
                return "unknown protocol";
        }
        return NULL;
}
int main(void){
         char *servname = "443";      // https service
         struct addrinfo hints;
         struct addrinfo *res_ptr;
         char *nodename = "www.ipleiria.pt";
         memset(&hints, 0, sizeof(hints) ) ;
         hints.ai_flags = AI_ALL;
         hints.ai_family = AF_INET;
```

Uso de **getaddrinfo** para obtenção de endereço IP e protocolos suportados do nome "**www.ipleiria.pt**"

Exemplo: getaddrinfo.c (#2)

```
(continuação)
        int ret = getaddrinfo(nodename,servname,&hints,&res_ptr);
        if( ret != 0 ){
                fprintf(stderr,
                        "can't getaddrinfo '%s'\n",gai_strerror(ret));
                exit(1);
        }
        /* Mostra conteúdo da lista ligada de nós addrinfo */
        struct addrinfo *p;
        char host_S[256]; // para ser preenchido por getnameinfo
        char serv_S[256]; // para ser preenchido por getnameinfo
        size_t host_S_len = sizeof(host_S);
        size_t serv_S_len = sizeof(serv_S);

(continua)
```

Exemplo: getaddrinfo.c (#3)

```
(continuação)
        // Itera a lista ligada de struct addrinfo
        for(p = res_ptr; p != NULL; p = p->ai_next) {
                printf("p->ai_family=%d\n", p->ai_family);
                printf("p->ai_socktype=%d\n",p->ai_socktype);
                printf("p->ai_protocol=%s\n",
                                        proto_to_str(p->ai_protocol));
                int flags = 0;
                getnameinfo(p->ai_addr,p->ai_addrlen,
                                host_S,host_S_len,
                                serv_S,serv_S_len,flags);
                printf("host_S='%s'\n",host_S);
                printf("serv_S='%s'\n",serv_S);
                printf("---------------------------------------\n");
        }
        // Liberta recursos
        freeaddrinfo(res_ptr);
        return 0;
}
```

Exemplo: getaddrinfo.c (#4)

- Saída

```
./a.exe
p->ai_family=2
p->ai_socktype=1
p->ai_protocol=IPPROTO_TCP
host_S='194.210.216.28'
serv_S='https'
-----------------------------------------------
p->ai_family=2
p->ai_socktype=2
p->ai_protocol=IPPROTO_UDP
host_S='194.210.216.28'
serv_S='https'
-----------------------------------------------
p->ai_family=2
p->ai_socktype=3
p->ai_protocol=unknown protocol
host_S='194.210.216.28'
serv_S='https'
-----------------------------------------------
```

Exemplo - getnameinfo

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <sys/types.h>
4.  #include <sys/socket.h>
5.  #include <netdb.h>
6.
7.  int main(void) {
8.    struct sockaddr_in addr;
9.    char host[NI_MAXHOST], service[NI_MAXSERV];
10.
11.   // Preenche o endereço do socket com informações
12.   memset(&addr, 0, sizeof(addr));
13.   addr.sin_family = AF_INET;
14.   addr.sin_port = htons(5000);
15.   addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
16.
17.   // Obtém informações sobre o socket
18.   if (getnameinfo((struct sockaddr*)&addr, sizeof(addr), host, NI_MAXHOST, service, NI_MAXSERV,
19.                                                          NI_NUMERICHOST | NI_NUMERICSERV) != 0)
    {
20.     fprintf(stderr, "getnameinfo failed\n");
21.     return 1;
22.   }
```

Flags que indicam a informação pretendida
NI_NUMERICHOST: end. IP
NI_NUMERICSERV: porto

EXEMPLO: CLIENTE

Exemplo de cliente (#1)

```
// Cliente que recorre ao getaddrinfo para obter estrutura
// de endereços que são empregues para ligar ao servidor
// Exemplo: socket TCP para www.example.com no porto 80 (http)
// IPv4 or IPv6
int sockfd;
struct addrinfo hints, *servinfo, *p;
int rv;
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC; // Usar AF_INET ou AF_INET6 para IPv4/v6
hints.ai_socktype = SOCK_STREAM;
if ((rv=getaddrinfo("www.example.com","http",&hints,&servinfo))!= 0){
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    exit(1);
}
(continua)
```

Exemplo de cliente (#2)

```
// Itera resultados e tenta ligação até obter sucesso
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("socket");
        continue;
    }
    if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        perror("connect");
        close(sockfd);
        continue;
    }
    break; // Ligação estabelecida
}
(continua)
```

Campos do nó struct addrinfo preenchidos por getaddrinfo

Exemplo de cliente (#3)

(continuação)

```
if (p == NULL) {
    // looped off the end of the list with no connection
    fprintf(stderr, "failed to connect\n");
    exit(2);
}
freeaddrinfo(servinfo); // liberta lista ligada de struct addrinfo
```

Fonte:
http://beej.us/guide/bgnet/output/html/multipage/getaddrinfoman.html

EXEMPLO: SERVIDOR

Exemplo de servidor (#1)

```
// Servidor – aguarda ligações TCP no porto 9999, no IP local
// (IPv4 ou IPv6)
int sockfd;
struct addrinfo hints, *servinfo, *p;
int rv;
memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;     // qualquer IPv4/IPv6
hints.ai_socktype = SOCK_STREAM; // TCP
hints.ai_flags = AI_PASSIVE;     // Usa endereço IP local
if ((rv = getaddrinfo(NULL, "9999", &hints, &servinfo)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    exit(1);
}
```

(continua)

Exemplo de servidor (#2)

```
// itera lista ligada de resultado, tentando bind até conseguir
for(p = servinfo; p != NULL; p = p->ai_next) {
    if ((sockfd = socket(p->ai_family, p->ai_socktype,
            p->ai_protocol)) == -1) {
        perror("socket");
        continue;
    }
    if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sockfd);
        perror("bind");
        continue;
    }
    break; // aqui? Bind efetuado com sucesso
}
```

(continua)

Campos do nó struct addrinfo preenchidos por getaddrinfo

Exemplo de servidor (#3)

(continuação)

```
if (p == NULL) {
    // looped off the end of the list with no successful bind
    fprintf(stderr, "failed to bind socket\n");
    exit(2);
}

freeaddrinfo(servinfo); // liberta lista ligada de nós addrinfo
```

Fonte:
http://beej.us/guide/bgnet/output/html/multipage/getaddrinfoman.html

BIBLIOGRAFIA

Bibliografia

- “Unix Network Programming: The Sockets Networking API”, Volume 1, 3rd edition, 2003, 1024 pages, Addison-Wesley. ISBN: 0-13-141155-1
- man 7 ip
- man 7 ipv6
- man gethostbyname
- man gethostbyaddr
- man getaddrinfo
- “Basic Socket Interface Extensions for IPv6”, RFC 3493, Fev. 2003 (https://www.ietf.org/rfc/rfc3493.txt)
- *Beej's Guide to Network Programming - Using Internet Sockets*, Brian “Beej Jorgensen” Hall, 2016 (http://beej.us/guide/bgnet/)