Jak działa Internet – DNS

W kolejnym artykule z serii „Jak działa Internet” przyjrzymy się systemowi DNS. Dowiemy się, jak on funkcjonuje oraz jakie są jego poszczególne elementy, a także poznamy narzędzia, które pomogą nam w rozwiązywaniu problemów związanych z tym systemem.

CZYM JEST DNS?

DNS (ang. *Domain Name System*) to system, który pozwala nam na zamianę nazw domen, jak np. programistamag.pl, na odpowiedni adres IP serwera, który hostuje dane [1].

Jeśli czytaliście poprzedni artykuł z serii, być może zastanawiacie się, dlaczego nie omawiamy dalej modelu OSI – protokołów TCP/IP czy HTTP. W wielu przypadkach nie musimy implementować własnych rozwiązań serwerowych. Często będziemy używać gotowych rozwiązań (jak chociażby hostingu przez pikapods.com czy gotowych obrazów dockerowych), których konfiguracja nie wymaga dogłębnej znajomości protokołów. Wystarczy podstawowa wiedza o adresowaniu i DNS, żeby postawić własny serwis online.

Dlatego najpierw przyjrzymy się właśnie systemowi nazw domenowych. W kolejnym artykule omówimy protokół HTTP, a następnie zagłębimy się w TCP/IP. Jeśli w międzyczasie będziemy potrzebować jakichś szczegółów z poszczególnych protokołów, będziemy wyjaśniać je na bieżąco.

Dla zainteresowanych – DNS z punktu widzenia modelu OSI jest aplikacją – działa na warstwie siódmej tego modelu. Co więcej, korzysta z różnych warstw transportu, w tym protokołu UDP [4], ale również z protokołu TCP, w szczególności, gdy używamy DoH (DNS over HTTPS), o czym opowiemy później.

Krótka historia DNS – plik hosts.txt

Mówiąc o DNS, należy chociaż pobieżnie wspomnieć o jego historii, zwłaszcza że jej szczątki znajdziemy i w dzisiejszych systemach komputerowych.

Kiedy Internet (wówczas ARPANET) był małą siecią z ograniczoną liczbą komputerów, system DNS nie był potrzebny, bo każdy komputer był w stanie przechowywać aktualną listę wszystkich adresów maszyn podłączonych do sieci.

Wpisy te były synchronizowane i zarządzane przez jedną organizację (Stanford Research Institute) [3]. W tamtych czasach, kiedy chciało się dodać wpis, należało zadzwonić na konkretny numer telefonu w godzinach pracy, po czym plik manualnie był aktualizowany przez administratora.

W miarę jak sieć się powiększała, rozwiązanie to przestawało być skalowalne. Jak to często bywa, potrzeba stała się matką wynalazków i tak ostatecznie powstał DNS.

W międzyczasie pojawiały się protokoły takie jak NetBIOS czy NetBEUI, które były używane jako systemy nazw wyłącznie w lokalnych sieciach (DNS sprawdza się zarówno w globalnej sieci – Internecie, jak i prywatnych sieciach LAN) [5].

Pliki *hosts.txt* można do dzisiaj znaleźć na naszych komputerach. Systemy operacyjne z rodziny UNIX mają plik */etc/hosts*, który zazwyczaj zawiera definicję nazw takich jak `localhost`, chociaż można go rozwinąć o własne nazwy.

Na systemach Windows plik ten znajdziemy w katalogu `c:\Windows\System32\drivers\etc` [6].

Blokowanie stron Internetowych za pomocą pliku hosts

Mając tę wiedzę, możemy już przejść do praktycznego jej zastosowania, chociaż nie będziemy jeszcze wykorzystywać systemu DNS.

Za pomocą pliku hosts oraz zdobytej wiedzy o adresowaniu możemy utworzyć listę witryn, które chcemy zablokować na naszym urządzeniu, przez nadanie im adresu `0.0.0.0`. Dodatkowo możemy nadać przyjazne nazwy urządzeniom w naszej lokalnej sieci, żeby nie zapamiętywać ich adresów IP.

Przykład takiego pliku znajduje się w Listingu 1.

Listing 1. Plik hosts blokujący dostęp do YouTube oraz dodający przyjazną nazwę dla interfejsu routera

```
127.0.0.1 localhost
# blokowanie youtube
0.0.0.0 youtube.com
0.0.0.0 www.youtube.com
# przyjazna nazwa dla panelu
# administracyjnego naszego routera
192.168.0.1 router.com
```

Skoro znamy już zarys historyczny i wiemy, co to jest plik hosts, przejdźmy do technikaliów DNS.

JAK DZIAŁA DNS?

Trzeba wiedzieć, że DNS jest hierarchiczny. Składa się z trzech komponentów – resolvera z cache'em, serwerów nazw oraz klienta. Klient wysyła zapytanie DNS z nazwą domeny w formacie FQDN do resolvera. Ten natomiast korzysta z wielu serwerów nazw, aby dla danego zapytania znaleźć odpowiednią odpowiedź w bazie danych. Krok po kroku przyjrzymy się kolejnym komponentom, które wymieniliśmy.

FQDN

Zacznijmy od nazw domen, tzw. FQDN (ang. *Fully Qualified Domain Name*), i przeanalizujmy ich budowę. Pomoże nam to zrozumieć organizację serwerów nazw [7].

Listing 2. Przykładowa nazwa FQDN

```
search.brave.com.
```

Zwyczajowo, kiedy używamy przeglądarek internetowych, pomijamy końcową kropkę, jednak pełna nazwa FQDN ją zawiera i ma ona swoje znaczenie.

Jak możemy się domyślać, nie istnieje jeden serwer DNS, który obsługuje wszystkie zapytania z całego świata. Takich serwerów jest wiele i są ułożone w hierarchiczną strukturę, która pozwala na odpowiednie rozdzielenie obsługi zapytań. Struktura ta przypomina strukturę nazwy FQDN.

Kropki zazwyczaj oddzielają tzw. strefy. Hierarchia DNS zaczyna się zawsze od korzenia (ang. *root*). Sam korzeń hierarchii nie ma nazwy, stąd w nazwie FQDN po końcowej kropce nie znajdziemy już żadnej nazwy.

Korzenie DNS zawierają informacje o domenach najwyższego poziomu (ang. TLD – Top Level Domain) [9]. Wszystkie nazwy domen najwyższego poziomu można znaleźć w bibliografii [8], a zarządzane są przez IANA (Internet Assigned Numbers Authority) [10]. W Listingu 2 nazwą domenową najwyższego poziomu jest nazwa `com`.

Następnie mamy nazwę domenową drugiego (ang. SLD – Second Level Domain) i trzeciego poziomu. Są to odpowiednio nazwy `brave` oraz `search` z Listingu 2. Nazwę `search` będziemy również nazywać poddomeną lub subdomeną [11].

W ramach domeny możemy tworzyć wiele poddomen (jak w przypadku google.com: `ads.google.com`, `calendar.google.com`, `images.google.com`) oraz zagnieżdżać kolejne poziomy subdomen (znalezione przeze mnie chociażby `vcenter-np.corp.google.com` czy `vcenter-pp.twd.corp.google.com`). W praktyce wykupujemy nazwę SLD, a w ramach jej konfiguracji możemy stworzyć praktycznie dowolną ilość poddomen.

Serwery nazw i strefy

Serwery DNS przechowują dane o domenach. Dane te są zgrupowane w strefy, a podstawową jednostką danych (wpisem w strefie) jest rekord (rekordy opisane są dokładniej w sekcji „Rekordy stref”). Strefy grupują rekordy powiązane z pojedynczą domeną.

Serwery DNS często obsługują zapytania o rekordy ze stref, których nie mają. W takim przypadku przekierowują zapytanie do innych serwerów DNS (nazywane później w artykule delegacją).

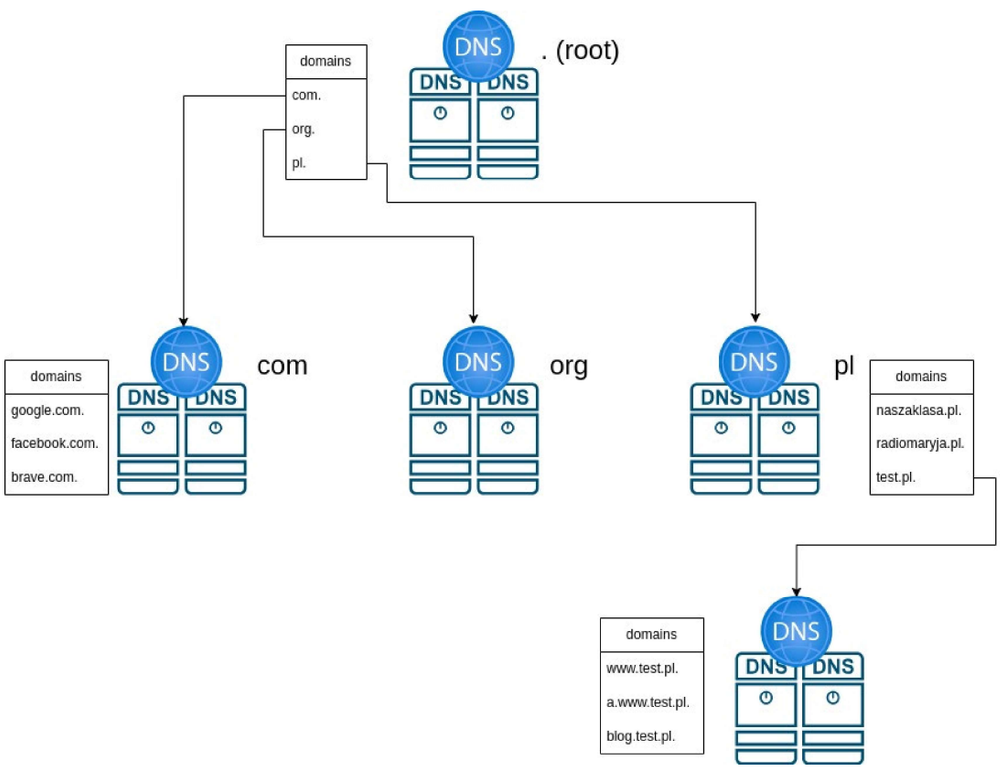
Aby lepiej zrozumieć hierarchiczność DNS oraz stref, spójrzmy na Rysunek 1.

Na samej górze widzimy korzeń drzewa DNS. Nie przechowuje on jednak informacji o wszystkich nazwach domen na świecie.

Zamiast tego przechowywane są one na innych serwerach. Warto również zwrócić uwagę, jak ważna jest redundancja danych. Istnieje wiele serwerów obsługujących domenę root (mamy 13 adresów IP dla serwerów root, a do każdego z adresu IP jest przypisane wiele maszyn obsługujących ruch. Każdy z serwerów IP ma swoją nazwę domenową – [a-m].root-servers.net [20]). Rozwiązanie takie jest ważne, aby nie obciążać zbytnio jednego serwera, jak również po to, żeby zwiększyć dostępność serwisu (gdy jedna z maszyn stanie się niedostępna, druga może przejąć jej rolę).

Korzeń drzewa DNS deleguje obsługę informacji o domenach TLD do innych serwerów. W przypadku TLD widzimy, że jeden serwer DNS odpowiada za jeden TLD – np. tylko .com. lub .pl.. Nieco inaczej jest w przypadku domeny test.pl.

Strefa ta zawiera informacje nie tylko o domenach z odpowiadających jej poziomu zagnieżdżenia, jak `www.test.pl`, ale także o bardziej



Rysunek 1. Strefy oraz serwery DNS

zagnieżdżonych domenach, jak `a.www.test.pl`. Rozwiązanie takie jest wygodne, kiedy obsługiwany ruch jest ograniczony, a przechowywanych danych mało.

Chociaż wydaje się, że informacje w bazie danych DNS są mocno statyczne, to w rzeczywistości nie zawsze tak jest. Serwery na poziomie tego ze strefy *.test.pl mogą dynamicznie generować subdomeny. Ma to zastosowanie chociażby w serwisach DDNS, gdzie zamiast statycznego, posiadamy jedynie dynamiczny adres IP [21].

Resolvery DNS

Resolvery DNS pośredniczą między klientem a serwerami DNS. Od klienta otrzymują one zapytanie (zazwyczaj z nazwą domenową), następnie korzystają z serwerów DNS, często odpytując wiele serwerów, aby ostatecznie móc dostarczyć odpowiedź klientowi (zazwyczaj zawierającą adres IP).

Dzięki resolverom działanie DNS jest znacznie szybsze, głównie ze względu na *cache*, który one implementują. Rozwiązanie jednej nazwy domenowej do adresu IP zazwyczaj wymaga wysłania co najmniej kilku zapytań. Jeśli odpowiedź na zapytanie znajduje się w cache'u, wystarczy jedynie pojedyncze zapytanie do resolvera, znajdującego się zazwyczaj w niewielkiej odległości od klienta.

Sprawdźmy przebieg komunikacji z użyciem klienta, resolvera oraz serwerów DNS.

Prześledźmy, co dzieje się na tym schemacie. Przede wszystkim klient DNS wysyła zapytanie do resolvera. Adres resolvera DNS jest konfigurowany manualnie lub automatycznie przez protokół (np. dhcp).

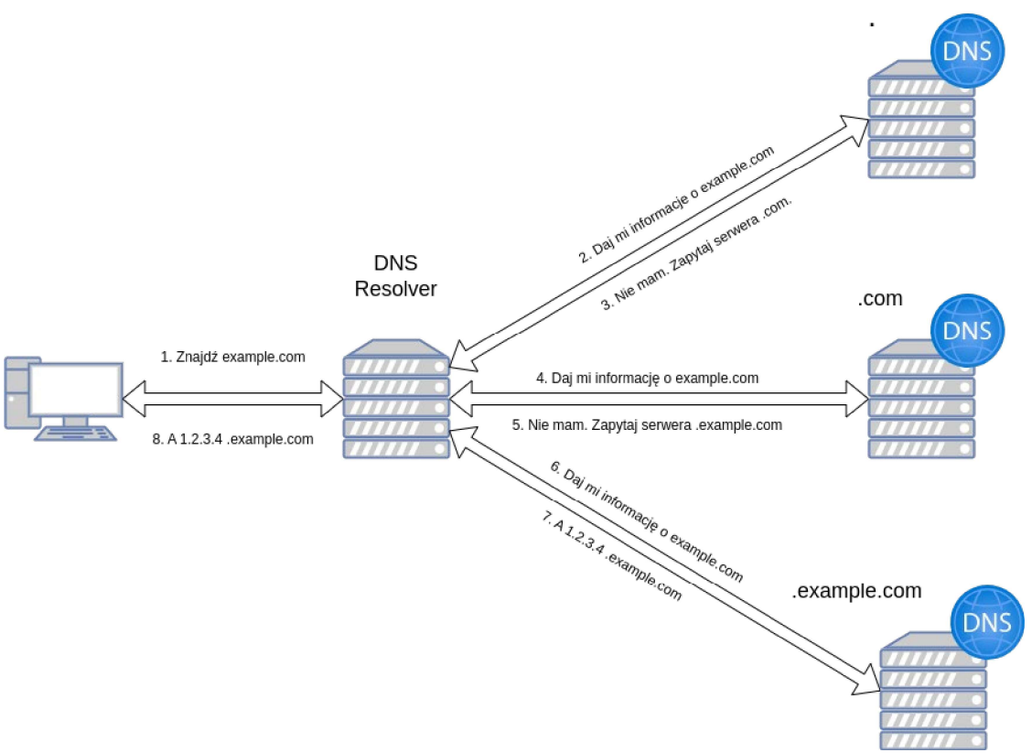
Następnie resolver wysyła zapytanie do serwera korzeniowego DNS. Adresy tych serwerów są z góry znane [13]. Resolver wysyła zapytanie o całą domenę, a serwer zwraca informacje, które ma w swojej bazie. W tym konkretnym przypadku serwer korzeniowy zwróci informację o nazwach serwerów DNS zawierających strefę .com oraz ich adresy.

Kolejno resolver odpytuje serwer DNS odpowiedzialny za strefę .com o adres domeny example.com, ale ten serwer również nie zna odpowiedzi na to pytanie. Wie natomiast, który serwer ma informację o strefie, i wysyła jego adres i nazwę hosta.

W ostatnim kroku resolver pyta ostatni serwer DNS odpowiedzialny za strefę example.com, który zna adres IP danej domeny. Serwer ten nazywany jest serwerem autorytatywnym. Warto zauważyć różnicę w odpowiedziach DNS – resolver zwraca odpowiedzi jako nieautorytatywne, podczas gdy serwery nazw (autorytatywne) przekazują wiarygodne, ostateczne odpowiedzi.

Klient DNS i cache

Ostatnia rzecz, o której warto pamiętać w architekturze DNS, to *cache* po stronie klienta (klient może być częścią aplikacji lub częścią systemu operacyjnego; oba warianty są możliwe). Na co dzień nie musimy się nim przejmować, ani pamiętać o jego istnieniu, jednak czasami potrzebuje on ręcznego czyszczenia. Proces ten różni się w zależności od użytego oprogramowania [14].



Rysunek 2. Przykład zapytania DNS z użyciem resolvera

REKORDY STREF

Dane o strefie składają się z tzw. rekordów, które mają określony format:

Listing 3. Przykładowy rekord w strefie example.com:

```
NAME          TYPE    DATA                TTL
example.com   A       93.184.215.14       1406
```

Odpowiednio są to nazwa domeny, typ wpisu (w tym przypadku A oznacza adres IP), dane (adres IP) oraz parametr TTL (ang. *Time To Live*), który określa, jak długo *cache* DNS powinien przechowywać rekord zanim stanie się on nieaktualny.

W Listingu 3 widzimy więc wpis, który informuje, że adres IP dla domeny example.com to 93.184.215.14, a *cache* DNS może przechowywać tę informację przez 1406 sekund.

Ważne jest, by zrozumieć podstawowe typy rekordów, ponieważ będą one często używane podczas konfiguracji nowych domen [15].

Rekord	Znaczenie
A/AAAA	Definiuje adres IP dla podanej nazwy domenowej. Typ A odpowiada adresowi IPv4, natomiast AAAA adresowi IPv6
CNAME	Definiuje alias nazwy domenowej. W takim rekordzie pole NAME jest aliasem, a DATA to nazwa, do której ten alias się odnosi. Na przykład rekord `www.mojblog.pl CNAME mojblog.pl. 86400` oznacza, że zarówno nazwa www.mojblog.pl oraz mojblog.pl odnoszą się do tego samego hosta
NS	Definiuje nazwę serwera DNS odpowiedzialnego za domenę. Rekordy tego typu najczęściej będziemy obserwować w odpowiedziach od serwerów korzeniowych oraz od serwerów TLD
MX	Definiuje nazwę serwera, który obsługuje pocztę mailową dla danej domeny. Jeśli chcemy, aby nasza domena miała własną pocztę e-mail, będziemy musieli skonfigurować ten rekord
PTR	Wpis odwrotny do rekordów A/AAAA. Umożliwia wykonywanie tzw. reverse lookup, czyli zapytań DNS, które tłumaczą adres IP na nazwę domenową
TXT	Jest to właściwie dowolny tekst, który możemy dodać do zarządzanej domeny. Często wykorzystywany przez serwisy w celu upewnienia się, że jesteśmy właścicielami danej domeny. Na przykład kiedy chcemy, aby dostawca poczty mailowej (np. Google) obsługiwał naszą domenę, będziemy musieli ustawić rekordy TXT na konkretne wartości, które będą następnie weryfikowane przez serwery google

Tabela 1. Typy rekordów i ich znaczenie

NARZĘDZIA DO ANALIZY DNS

Mając teoretyczne podstawy, przejdźmy do praktyki – wykonajmy kilka zapytań DNS za pomocą konkretnych narzędzi i przeanalizujmy ich odpowiedzi. Do eksperymentów będziemy wykorzystywali popularne narzędzia, takie jak dig czy nslookup. Następnie ruch sieciowy wygenerowany przez te narzędzia przeanalizujemy w aplikacji Wireshark.

Zacznijmy od próby uzyskania adresu dla domeny google.com.

Listing 4. Proste użycie aplikacji dig i jej wyjście

```
$ dig google.com

; <<>> DiG 9.20.1 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id:
41572
;; flags: qr rd ra; QUERY: 1,
;; ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.  282  IN  A  142.250.203.142

;; Query time: 43 msec
;; SERVER: 10.11.0.1#53(10.11.0.1) (UDP)
;; WHEN: Mon Oct 14 19:13:09 CEST 2024
;; MSG SIZE  rcvd: 55
```

Jak widać, wyjście programu dig na pierwszy rzut oka wydaje się dość skomplikowane, zwłaszcza jeśli porównamy je ze znacznie bardziej czytelnym wyjściem programu nslookup:

Listing 5. Proste użycie aplikacji nslookup

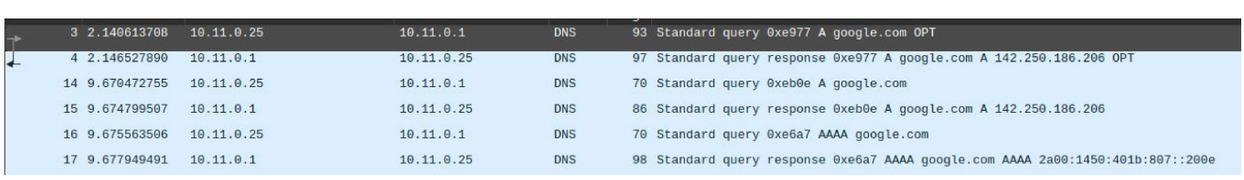
```
$ nslookup google.com

Server:         10.11.0.1
Address:        10.11.0.1#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.186.206
Name:   google.com
Address: 2a00:1450:401b:807::200e
```

Najważniejszą sekcją wyjścia programu dig jest sam rekord: `google.com.  282  IN  A  142.250.203.142`, który może się zgubić w gąszczu przedstawionych informacji.

W skrócie, rekord ten informuje, że host dla nazwy domenowej google.com znajduje się pod adresem IP 142.250.203.142. Jest to typ rekordu `A`, tak jak omawialiśmy to w Tabeli 1. Rekord instruuje również *cache* co do czasu przetrzymywania kopii rekordu (w tym przypadku są to 282 sekundy). Klasa rekordu – `IN` – mówi nam, że jest to klasa Internet. W praktyce nie będziemy spotykać się z innymi klasami.

Jeśli spojrzymy na wyjście programu nslookup, zobaczymy tam również adres IPv6. Program nslookup w rzeczywistości wykonuje 2 zapytania DNS: jedno z nich o rekordy typu `A`, a drugie rekordy typu `AAAA`. Możemy to zaobserwować, analizując ruch w aplikacji Wireshark.

Na Rysunku 3 widzimy trzy zapytania typu `Standard query` oraz 3 odpowiedzi typu `Standard query response`. Warto zwrócić uwagę



Rysunek 3. Zapytania DNS aplikacji dig oraz nslookup

na unikalną wartość przy każdym zapytaniu i odpowiedzi zaczynających się od `0x`. Jak można się domyślić, jest to identyfikator zapytania. Odpowiedź na zapytanie będzie zawierać dokładnie taki sam identyfikator, co zapytanie. W ten sposób wiemy, do jakiego zapytania odnosi się odpowiedź.

W podsumowaniu widzimy „treść" zapytania. W pierwszym przypadku jest to `A google.com OPT`, w drugim widzimy to samo zapytanie bez `OPT`. Nie będziemy szczegółowo omawiać części `OPT`, która służy do umieszczania dodatkowych informacji w zapytaniach i odpowiedziach DNS. Więcej informacji na ten temat można znaleźć w Bibliografii [16].

Na Rysunku 3 następnie przedstawiono zapytanie aplikacji nslookup, gdzie dokładnie widać 2 zapytania – jedno z nich pyta o rekordy `A`, a drugie o `AAAA`. W aplikacji możemy jawnie powiedzieć, jakiego typu rekordu szukamy. Co więcej, możemy również zdefiniować, jakiego serwera DNS należy użyć do rozwiązania zapytania.

Listing 6. Przykład użycia konkretnego serwera DNS do rozwiązywania nazwy domenowej google.com

```
> dig @1.1.1.1 google.com aaaa

; <<>> DiG 9.20.1 <<>> @1.1.1.1 google.com aaaa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2341
;; flags: qr rd ra; QUERY: 1,
;; ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;google.com.                    IN      AAAA

;; ANSWER SECTION:
google.com.  64  IN  AAAA  2a00:1450:401b:80d::200e

;; Query time: 13 msec
;; SERVER: 1.1.1.1#53(1.1.1.1) (UDP)
;; WHEN: Mon Oct 14 20:49:49 CEST 2024
;; MSG SIZE  rcvd: 67
```

Parametr `@1.1.1.1` informuje aplikację dig, że zapytanie DNS ma być wysłane pod adres `1.1.1.1`, który jest adresem serwera DNS firmy Cloudflare.

Aby lepiej zrozumieć, dlaczego wyjście z aplikacji dig jest tak rozbudowane, możemy przyjrzeć się szczegółom komunikacji DNS w aplikacji Wireshark.

To, co widzimy na początku, to nagłówek. Najważniejsze jego elementy to:

- identyfikator,
- flagi, w tym przypadku informujące nas o tym, że jest to standardowe zapytanie,
- ilość pytań w zapytaniu (w jednym zapytaniu możemy prosić o rozwiązanie wielu nazw domenowych).

Po sekcji nagłówka widzimy sekcję pytań (Queries). W tym przypadku składa się ona z jednego zapytania o domenę google.com. Zapytanie składa się jedynie z nazwy domeny, typu rekordów, o które pytamy (w tym przypadku `aaaa`), oraz klasy (zawsze `IN`).

Na takie zapytanie otrzymaliśmy odpowiedź załączoną na Rysunku 5.

Widzimy, że w odpowiedzi mamy powtórzone zapytanie (sekcja Queries). Widzimy również „Answer RRs" (ang. RR – Resource Record), które ma wartość `1`, co oznacza, że odpowiedź z serwera zawiera jedną odpowiedź na nasze zapytanie, która ma jeden rekord.

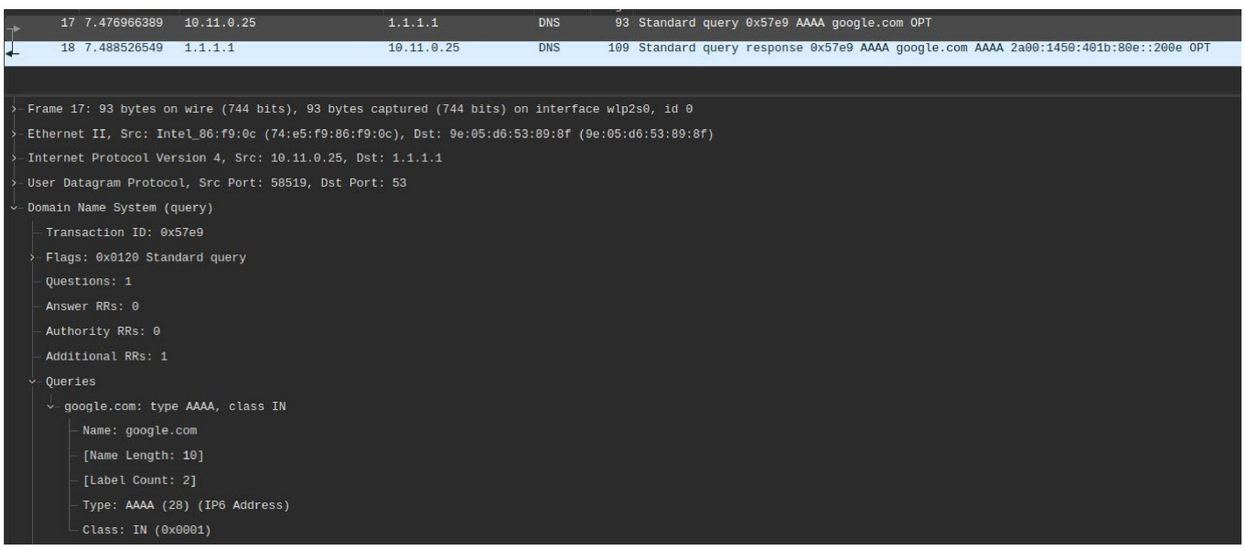
W wyjściu z aplikacji dig możemy zauważyć wszystkie te pola, które właśnie opisaliśmy, a więc:

Listing 7. Nagłówek odpowiedzi DNS przedstawiony w programie dig

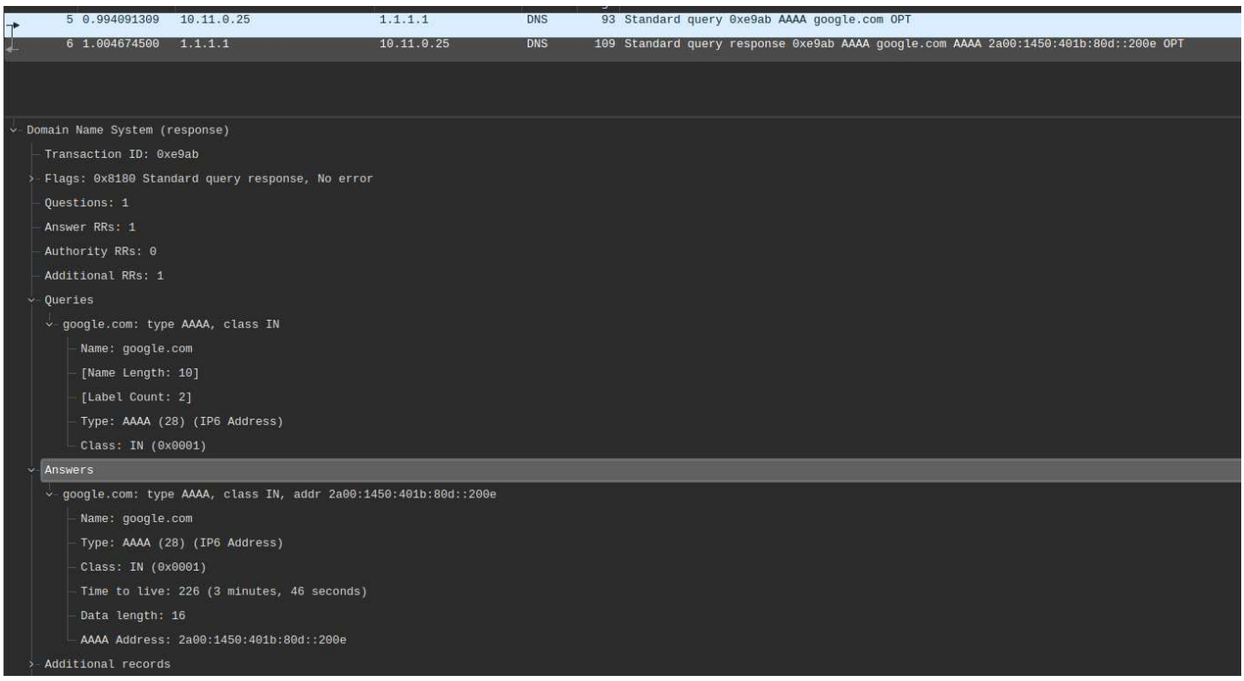
```
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2341
;; flags: qr rd ra; QUERY: 1,
;; ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
```

Got answer (otrzymaliśmy odpowiedź):

- *opcode: Query* – informuje nas, że jest to standardowa odpowiedź – w przeciwieństwie do odwrotnej (ang. *inverse* – nie mylić z reverse), której nie będziemy omawiać,



Rysunek 4. Zapytanie DNS widziane w aplikacji Wireshark



Rysunek 5. Odpowiedź DNS widziana w aplikacji Wireshark

- *status: noerror* – wskazuje na brak błędu (możliwe są błędy zapytania czy chociażby błędy serwera) – *id: 2341*,
- id transakcji ,
- *flags: qr rd ra* – ustawione flagi:
  - - qr – oznacza odpowiedź (w przeciwieństwie do flagi zapytania),
  - - rd – zapytanie było rekursywne,
  - - ra – odpowiedź jest rekursywna (o czym więcej w sekcji „Zapytania iteracyjne").
- *Query: 1* – jedno zapytanie,
- *Answer: 1* – jedna odpowiedź,
- *Authority: 0* – brak odpowiedzi typu authority,
- *Additional: 1* – dostępna jedna dodatkowa informacja.

Po nagłówku widzimy dwie sekcje przedstawione w Listingu 8.

Listing 8. Sekcja pytań i odpowiedzi w odpowiedzi na zapytanie DNS

```
;; QUESTION SECTION:
;google.com.                    IN      AAAA
;; ANSWER SECTION:
google.com.  64  IN  AAAA  2a00:1450:401b:80d::200e
```

Odpowiadają one jednemu zapytaniu i jednej odpowiedzi z sekcji nagłówkowej. Następnie widzimy szczegóły dotyczące samego zapytania – jego czas trwania, serwer, do którego wysłane zostało zapytanie, oraz rozmiar odpowiedzi.

Dokładny opis struktury zapytań i odpowiedzi można znaleźć w bibliografii [17].

Zapytania odwrotne

Zapytania odwrotne (DNS reverse lookup) to zapytania, w których do adresu IP próbujemy dopasować nazwę domeny. Warto wiedzieć, że standard DNS definiuje również tzw. inverse queries, które są czym innym, których tutaj nie omawiamy i które są rzadko używane.

Zapytania odwrotne są tak naprawdę zwykłymi zapytaniami. To, co je wyróżnia, to dość nietypowa nazwa domenowa oraz typ rekordu w zapytaniu – PTR.

Dla przykładu, chcąc znaleźć nazwę domeny dla adresu 1.2.3.4, należy wywołać komendę `dig 4.3.2.1.in-addr.arpa ptr`. Widzimy więc, że nazwa domeny, której szukamy, to odwrócony adres IP z „rozszerzeniem" in-addr.arpa.

Program dig obsługuje flagę -x, która ułatwia użycie – nie trzeba pamiętać specyficznej nazwy in-addr.arpa ani odwracać adresu IP. Wywołanie takiego programu wyglądałoby następująco: `dig -x 1.2.3.4`.

Przeanalizujmy przykład dla adresu IP: 172.217.16.14 (adres należący do google.com):

Listing 9. Zapytanie odwrotnie dla adresu 172.217.16.14

```
$ dig -x 172.217.16.14

#...
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;14.16.217.172.in-addr.arpa.    IN      PTR

;; ANSWER SECTION:
14.16.217.172.in-addr.arpa. 77610 IN
    PTR     waw02s13-in-f14.1e100.net.
14.16.217.172.in-addr.arpa. 77610 IN
    PTR     mil02s06-in-f14.1e100.net.
#...
```

Jak widać, w sekcji `QUESTION` pytamy o rekord `PTR` dla wspomnianej specyficznej domeny. W odpowiedzi uzyskujemy dwie nazwy domenowe, jednak żadna z nich nie jest domeną, z której oryginalnie pozyskaliśmy adres IP.

Jest to możliwe, a co więcej, czasami możemy w ogóle nie otrzymać informacji w sekcji ANSWER SECTION.

Wszystko zależy od konfiguracji. Jeden adres IP może należeć do wielu domen i nie dla wszystkich domen możemy chcieć posiadać odpowiadający im wpis PTR.

Zapytania iteracyjne

Zapytania DNS dzielą się na rekursywne i iteracyjne. Rekursywne zapytanie DNS polega na tym, że klient wysyła takie zapytanie do resolvera, który następnie wykonuje iteracyjne zapytania do kolejnych serwerów DNS w imieniu klienta.

Zapytanie iteracyjne to właśnie odpytywanie po kolei różnych serwerów DNS w celu rozwiązania nazwy [18].

Niektóre serwery nazw np. domeny root czy TLD nie obsługują zapytań rekursywnych. Możemy to łatwo sprawdzić, a przy okazji zapoznamy się z dość istotną funkcjonalnością, a mianowicie interaktywnym trybem pracy narzędzia nslookup. W kolejnych listingach przedstawimy sekwencję komend wykonywanych w tym programie.

Zacznijmy od zapytania o serwery root DNS (jeśli zapomnimy ich nazwy domenowe, możemy po prostu o nie zapytać naszego resolvera).

Listing 10. Wyszukiwanie nazw serwerów root w aplikacji nslookup

```
$ nslookup
> set type=NS
> .

Server:         10.11.0.1
Address:        10.11.0.1#53

Non-authoritative answer:

.  nameserver = a.root-servers.net.
.  nameserver = b.root-servers.net.
.  nameserver = c.root-servers.net.
.  nameserver = d.root-servers.net.
.  nameserver = e.root-servers.net.
//...
```

Jak widać, po wywołaniu komendy nslookup bez argumentów zaczęliśmy interaktywny sposób pracy z tym narzędziem.

Widzimy w pierwszym wpisanym poleceniu, że możemy ustawiać parametry zapytań DNS. W tym przypadku ustawiamy typ rekordu, o który chcemy pytać. Jak już wspomnieliśmy, rekordy tego typu wskazują na nazwę domenową serwera.

. to nazwa domeny (zobacz sekcję „FQDN"). Po wpisaniu nazwy domeny program wysyła zapytanie i wyświetla odpowiedź, w której otrzymujemy nazwy domenowe root serwerów.

Domyślnie zapytania DNS wysyłane zarówno przez aplikację dig, jak i nslookup zawiera ustawioną flagę recurse, co oznacza, że chcielibyśmy, żeby serwer, jeśli jest taka potrzeba, zachował się również jako resolver. Niemniej jednak dla pominięcia wszelkich wątpliwości w Listingu 11 ustawiliśmy ją jawnie.

Listing 11. zapytanie rekursywne do serwera root DNS

```
> server a.root-servers.net
> set recurse
> set type=A
> set debug
> google.com
//...
------------
  QUESTIONS:
    google.com, type = A, class = IN
  ANSWERS:
  AUTHORITY RECORDS:
    ->  com nameserver = l.gtld-servers.net.ttl = 172800
    ->  com nameserver = j.gtld-servers.net.ttl = 172800
//...
  ADDITIONAL RECORDS:
    ->  l.gtld-servers.net
        internet address = 192.41.162.30
        ttl = 172800
    ->  l.gtld-servers.net
        has AAAA address 2001:500:d937::30
        ttl = 172800
    ->  j.gtld-servers.net
        internet address = 192.48.79.30
        ttl = 172800
    ->  j.gtld-servers.net
        has AAAA address 2001:502:7094::30
        ttl = 17280
*** Can't find google.com: No answer
```

Widzimy w odpowiedzi, że serwer nie może znaleźć serwera google.com (prawda jest taka, że nawet nie próbował go znaleźć). Wysłał nam więc pustą sekcję odpowiedzi ANSWERS, za to dodał rekordy w sekcji AUTHORITY RECORDS, które wskazują na serwery nazw dla TLD .com. W ADDITIONAL RECORDS znajdujemy coś, o co nikt nie prosił, a każdy potrzebował, czyli adresy IP serwerów przekazanych w sekcji AUTHORITY_RECORDS. Dzięki temu zabiegowi, przy rozwiązywaniu nazw, nie będziemy musieli wysyłać dodatkowego requestu DNS z prośbą o rozwiązanie nazwy dla serwera TLD.

W kolejnym kroku zapytamy serwer TLD o nazwę google.com.

Listing 12. Zapytanie o domenę google.com serwera TLD

```
> server a.gtld-servers.net
> google.com

//...
------------
    QUESTIONS:
        google.com, type = A, class = IN
    ANSWERS:
    AUTHORITY RECORDS:
    ->  google.com
        nameserver = ns2.google.com.
        ttl = 172800
    ->  google.com
        nameserver = ns1.google.com.
        ttl = 172800
    ADDITIONAL RECORDS:
//...
*** Can't find google.com: No answer
```

Podobnie jak w poprzednim przypadku, serwer odpowiada, że nie wie, gdzie jest google.com (podobnie jak poprzednio nie próbował jej nawet znaleźć), wie za to, gdzie szukać serwera autorytatywnego, podając nam jego nazwę i adres (w pominiętej sekcji ADDITIONAL RECORDS).

Ostatnim krokiem będzie odpytanie serwera autorytatywnego o domenę google.com.

Listing 13. Zapytanie o domenę google.com autorytatywnego serwera

```
> server ns2.google.com
> google.com

//...
------------
```

```
    QUESTIONS:
        google.com, type = A, class = IN
    ANSWERS:
    ->  google.com
        internet address = 142.250.203.142
        ttl = 300
    AUTHORITY RECORDS:
    ADDITIONAL RECORDS:
------------

Name:   google.com
Address: 142.250.203.142
```

Udało się! To, co zrobiliśmy w Listingach 11-13. to iteracyjne zapytanie DNS. Jest to coś, co zazwyczaj wykonuje nasz resolver DNS, kiedy prosimy go o rozwiązanie nazwy.

Takie podejście pozwala nam dokładnie przeanalizować każdy krok rozwiązywania nazw DNS. Niestety wiąże się to z wklepywaniem wielu komend. Na szczęście program dig ma parametr `+trace`, który automatycznie wykona całą powyższą sekwencję requestów DNS. Analizę outputu komendy `dig google.com +trace` zostawiam jako „zadanie domowe".

Odpowiedzi autorytatywne

Przyglądając się poprzednim przykładom i opisom z artykułu, trafiliśmy na określenie „autorytatywny". Spójrzmy szybko na przykład:

Listing 14. Przykład nieautorytatywnej odpowiedzi DNS

```
$ nslookup google.com
Server:         10.11.0.1
Address:        10.11.0.1#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.186.206
Name:   google.com
Address: 2a00:1450:401b:80e::200e
```

Widzimy w wyjściu programu, że odpowiedź jest nieautorytatywna (Non-authoritive answer). Najprościej rzecz ujmując, odpowiedź nie pochodzi od serwera DNS, do którego wysłaliśmy zapytanie, a od innego serwera DNS, który został odpytany przez ten serwer.

Taka odpowiedź w rzadkich przypadkach będzie niepoprawna ze względu na istnienie cache'a.

W Listingu 12 poznaliśmy autorytatywny serwer dla domeny google.com. Jeśli więc wyślemy zapytanie bezpośrednio do tego serwera o domenę, która do niej należy, dostaniemy odpowiedź autorytatywną.

Listing 15. Przykład autorytatywnej odpowiedzi DNS

```
$ nslookup google.com ns2.google.com
Server:         ns2.google.com
Address:        216.239.34.10#53

Name:   google.com
Address: 142.250.203.142
Name:   google.com
Address: 2a00:1450:401b:80e::200e
```

Dla wyjaśnienia, serwer autorytatywny dla domeny google.com sam istnieje w domenie google.com. I oczywiście, żeby aplikacja mogła wysłać zapytanie do serwera ns2.google.com, musi najpierw rozwiązać tę nazwę do adresu IP. Pod spodem więc aplikacja rozwiązuje nazwę ns2.google.com, co widać chociażby w aplikacji Wireshark.

Warto zwrócić uwagę na adresy IP, do których wysyłane są zapytania.

8	0.067111937	10.11.0.25	10.11.0.1	DNS	74	Standard query 0xde0f A ns2.google.com
9	0.070016267	10.11.0.1	10.11.0.25	DNS	90	Standard query response 0xde0f A ns2.google.com A 216.239.34.10
10	0.070574890	10.11.0.25	216.239.34.10	DNS	70	Standard query 0x3cc5 A google.com
12	0.130797248	216.239.34.10	10.11.0.25	DNS	86	Standard query response 0x3cc5 A google.com A 142.250.203.142

Rysunek 6. Zapytanie o adres dla nazwy ns2.google.com przed zapytaniem o nazwę google.com

Pierwsze zapytanie o rozwiązanie nazwy ns2.google.com jest wysyłane do serwera DNS w sieci lokalnej (w tym przypadku router pełni również rolę resolvera DNS). Po rozwiązaniu nazwy do adresu IP (216.239.34.16) pod ten właśnie adres jest wysyłane zapytanie o domenę google.com zgodnie z naszym oczekiwaniem przy wywoływaniu komendy `nslookup google.com ns2.google.com`.

Ostatnia kwestia, o której wspomnę, to gdzie znaleźć informację o autorytatywności odpowiedzi w przypadku aplikacji dig:

Listing 16. Przykład autorytatywnej odpowiedzi DNS w programie dig

```
$ dig @ns2.google.com google.com

; <<>> DiG 9.20.1 <<>> @ns2.google.com google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id:
27367
;; flags: qr aa rd; QUERY: 1,
;; ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
//...
```

Za informację o autorytatywności odpowiada flaga `aa` umieszczona w nagłówku odpowiedzi. Jej obecność potwierdza autorytatywność odpowiedzi, a jej brak świadczy o braku autorytatywności (każdy serwis DNS może ustawić tę flagę wedle uznania; nie znaczy to więc, że odpowiedź autorytatywna jest w pełni godna zaufania).

Zanim skończymy podróż po świecie DNS, krótko jeszcze omówię kilka, moim zdaniem, przydatnych informacji o rozwiązaniach w serwisach DNS, które poprawią nasze bezpieczeństwo i prywatność.

DNS A BEZPIECZEŃSTWO I PRYWATNOŚĆ

Blokowanie treści szkodliwych

Dzięki dostawcom serwerów DNS takich jak chociażby Cloudflare (akurat sam używam tego dostawcy; niektóre inne serwisy również mają podobne funkcjonalności) możemy blokować dostęp do stron z pornografią oraz z drastycznymi treściami oraz hosty znane z tego, że serwują malware.

W przypadku Cloudflare domyślny serwer DNS to `1.1.1.1` i jako taki nie zawiera żadnych filtrów. Poza tym adresem jednak Cloudflare ma jeszcze:

- `1.1.1.2` – adres DNS, który dodatkowo filtruje domeny znane z serwowania malware'u.
- `1.1.1.3` – adres DNS, który filtruje to, co adres `1.1.1.2`, a także strony nieodpowiednie dla nieletnich.

Ustawienie serwera DNS używanego przez dane urządzenie czy router (jeśli chcemy wprowadzić ograniczenie na całą sieć) zależy od konkretnego urządzenia.

Jak takie filtrowanie działa? Z naszą wiedzą możemy to po prostu sprawdzić! Przykładowy adres do testów to https://nudity.testcategory.com/ (witryna stworzona przez Cloudflare w celu testowania ustawień DNS). Sprawdźmy. jak działa resolvowanie tego adresu.

Listing 17. Porównanie odpowiedzi DNS z resolvera filtrującego i niefiltrującego na zapytanie o adres strony z treściami nieodpowiednimi dla nieletnich

```
$ nslookup
> server 1.1.1.1
> nudity.testcategory.com
//...
Non-authoritative answer:
Name:   nudity.testcategory.com
Address: 104.18.5.35
//...

> server 1.1.1.3
> nudity.testcategory.com
//...
Non-authoritative answer:
Name:   nudity.testcategory.com
Address: 0.0.0.0
//...
```

Jak widzimy, serwer `1.1.1.1` rozwiązał nazwę poprawnie, podczas, gdy `1.1.1.3` zwrócił nieistniejący adres IP `0.0.0.0` (o adresie tym wspomnieliśmy dokładniej w poprzednim artykule. Takie IP nie należy do żadnego serwisu internetowego; w przypadku systemów Linux jest to odpowiednik localhost). Resolvery więc działają na prostej zasadzie blacklistowania znanych, szkodliwych domen.

Ale to nie wszystko!

Wyszukiwarki internetowe takie jak google.com czy duckduckgo.com również korzystają z dobrodziejstw takich resolverów. I tak, w przypadku, gdy korzystamy z takiego bezpiecznego resolvera, nie będziemy mogli wyłączyć filtra „safe search" w wymienionych wyszukiwarkach. Nietrudno się domyślić – to też wynika z innych rezultatów podanych przez resolver DNS:

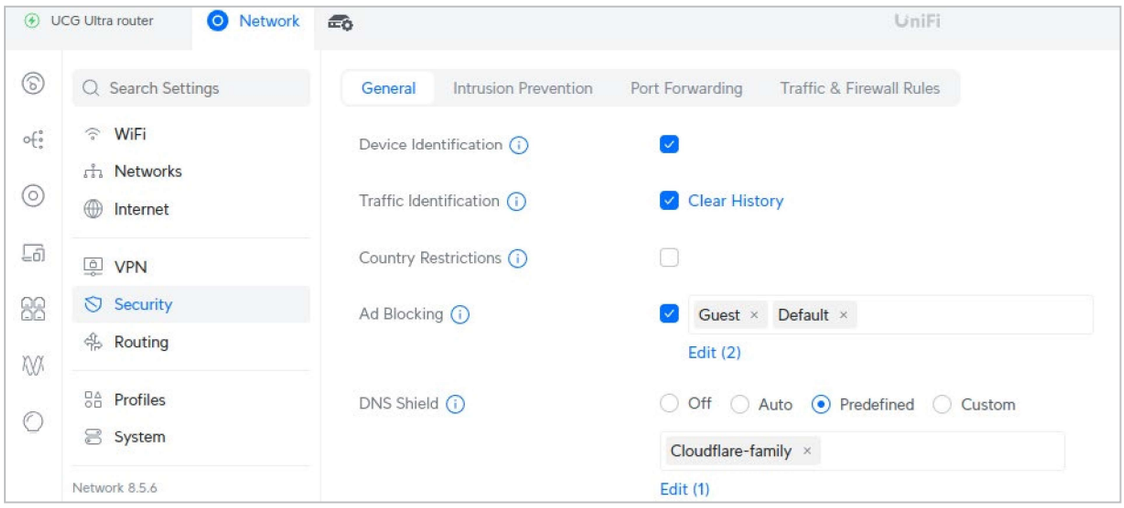
Listing 18. Porównanie odpowiedzi DNS z resolvera filtrującego i niefiltrującego na zapytanie o duckduckgo

```
$ nslookup
> server 1.1.1.1
> duckduckgo.com

//...
non-authoritative answer:
Name:   duckduckgo.com
Address: 40.114.177.156
//...

> server 1.1.1.3
> duckduckgo.com

//...
Non-authoritative answer:
duckduckgo.com  canonical name = safe.duckduckgo.com.
Name:   safe.duckduckgo.com
Address: 40.114.177.246
//...
```

Jak widzimy, w przypadku korzystania z serwera `1.1.1.3` ten udaje, że istnieje jedynie rekord `CNAME` dla domeny duckduckgo.com, który jest aliasem na safe.duckduckgo.com. Ostatecznie oba są rozwiązywane do różnych adresów IP, w ten sposób trafiamy na inne „wersje" wyszukiwarek, w których zawsze filtry są włączone.

Prywatność

Domyślnie skonfigurowany DNS niezbyt sprzyja naszej prywatności. Każda strona internetowa, którą odwiedzamy w przeglądarce, potencjalnie jest poprzedzona zapytaniem DNS. Zapytania te zazwyczaj nie są szyfrowane (o czym przekonaliśmy się, analizując pakiety w aplikacji Wireshark), więc każdy router, który będzie przetwarzać zapytanie, jest w stanie przeczytać treść zapytania DNS. Ponadto nasze serwisy DNS zazwyczaj implementują dostawcy Internetu, którzy znają nas z imienia i nazwiska.



Rysunek 7. Włączenie DoH w UniFi OS

Innymi słowy – dostawca Internetu jest w stanie w dość prosty sposób sprawdzić, jakie strony oglądaliśmy w danym dniu. Aby się przed tym zabezpieczyć, należy nie tylko użyć innego dostawcy DNS (takiego, który nie zna naszych personalnych danych), ale także zaszyfrować zapytania i odpowiedzi DNS (aby routery, przez które przechodzą zapytania, w tym te naszego dostawcy Internetu, nie były w stanie odczytać treści wiadomości).

Spotkamy tutaj rozwiązania DoT oraz DoH – odpowiednio DNS over TLS oraz DNS over HTTPS [19]. Obie te technologie pozwalają na przesyłanie zaszyfrowanych zapytań i odpowiedzi DNS między klientem a serwerem.

Zazwyczaj konfigurację DoT oraz DoH wprowadzamy w naszym routerze, który pełni rolę naszego resolvera DNS. Mając to na uwadze, mogę jedynie przedstawić sposób konfiguracji DoH w routerze Ubiquity w UniFi OS. Odpowiednie ustawienia znajdują się w górnej zakładce *Network* i bocznej zakładce *Security*. Ustawienie jest na tyle przyjemne, że po prostu wybieramy dostawcę DNS z predefiniowanej listy w ustawieniu o nazwie „DNS Shield" jak na Rysunku 7.

W przypadku telefonów z Androidem odpowiednie ustawienia znajdziemy w Ustawienia → Połączenia → Więcej ustawień połączenia → Prywatny DNS. Możemy tam chociażby ustawić nazwę one.one.one.one (dla serwerów Cloudflare).

Warto dodać, że mimo szyfrowania, komunikacji i używania zewnętrznego dostawcy serwisu DNS, nasz dostawca Internetu nadal może z dość dużą precyzją zgadywać, jakie strony odwiedzamy, biorąc pod uwagę adresy docelowe IP pakietów, które wysyłamy przez router dostawcy.

PODSUMOWANIE

Przeszliśmy przez podstawy protokołu DNS. Poznaliśmy architekturę systemu oraz protokół komunikacji wraz z narzędziami do wykonywania zapytań oraz inwestygowania problemów z protokołem. Zapoznaliśmy się również z rozwiązaniami dostępnymi na rynku, które mogą poprawić nasze bezpieczeństwo w sieci, prywatność danych czy odpowiednie filtrowanie treści w sieci.

Jako dalszą edukację w temacie DNS polecałbym utworzenie własnej domeny i zabawę z nią. Możemy to zrobić np. w portalu home.pl, ale jeśli poszukamy nieco dłużej, to prawdopodobnie uda nam się również stworzyć domenę za darmo.

W kolejnym artykule tej serii poznamy protokół HTTP, którego znajomość pozwoli nam na implementację własnych serwisów i stron internetowych. Do zobaczenia!

Bibliografia

1. https://en.wikipedia.org/wiki/Domain_Name_System
2. https://en.wikipedia.org/wiki/List_of_network_protocols_(OSI_model)
3. https://en.wikipedia.org/wiki/Hosts_(file)
4. https://www.geeksforgeeks.org/why-does-dns-use-udp-and-not-tcp/
5. https://networkencyclopedia.com/netbeui/
6. https://allthings.how/how-to-edit-hosts-file-in-windows-11/
7. https://www.lifewire.com/what-does-fqdn-mean-2625883
8. https://data.iana.org/TLD/tlds-alpha-by-domain.txt
9. https://icannwiki.org/Top-Level_Domain
10. https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority
11. https://www.hostinger.com/tutorials/what-is-a-domain-name
12. https://www.cloudflare.com/learning/dns/glossary/dns-zone/
13. https://www.iana.org/domains/root/servers
14. https://www.geeksforgeeks.org/how-to-flush-dns-cache/
15. https://phoenixnap.com/kb/dns-record-types
16. EDNS, OPT – https://en.wikipedia.org/wiki/Extension_Mechanisms_for_DNS
17. DNS RFC – https://www.rfcreader.com/#rfc1035_line1128
18. https://www.cloudflare.com/learning/dns/what-is-recursive-dns/
19. https://www.cloudflare.com/learning/dns/dns-over-tls/
20. https://www.iana.org/domains/root/servers
21. https://www.cloudflare.com/learning/dns/glossary/dynamic-dns/

DAWID PILARSKI

dawid.pilarski@panicsoftware.com

Z wykształcenia automatyk i robotyk, a z zawodu i pasji programista. Obecnie Software Engineer, Security Champion i Tech Lead w TomTom. Wolny czas przeznacza na zgłębianie wiedzy o bezpieczeństwie i sieciach. Można się z nim skontaktować poprzez adres e-mail: dawid.pilarski@panicosftware.com (w celach zawodowych) oraz dawid.pilarski@pm.me (w celach prywatnych).