BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

DNS RESOLVER TESTING

Filip Široký*

Abstract
This paper describes automation of creating scenarios for Deckard, which is DNS resolver testing tool. The scenarios are based on real traffic between a web browser and a web page gathered when the web browser loads that page. Captured traffic is then supplemented with queries that resolvers might require with query minimization on and off. The outcome should provide repeatable conditions same as live environment changes. It is due to things like IP address rotation, different content on servers authoritative for the same zone, content modification and so on. Also, no query should remain unanswered. As for other usages, having a response from the traffic of the previous version of the resolver leads to being able to detect changes in behavior based on the same data between two versions, and discovering potential disability to resolve a web page caused by a bug. As well as creating scenarios by hand is a long and hard process and automation could increase the code coverage. It might take some time to achieve satisfying universality of the tool as there is a large number of different ways to store DNS content.

Keywords: DNS — Knot Resolver — test automation

Supplementary Material: *N/A*

*xsirok08@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Domain Name System (DNS) is one of the most important services on the Internet. If it stopped working, a lot of users would be affected as it is used by many other online services and protocols (FTP, HTTP, SIP).

DNS records are defined loosely and the infrastructure of name servers can be set up in different ways. This allows an inconsistency between servers expected to have the same content and other issues with resolution. Also IP rotation, ever-changing content, localization and cloud services contribute to the unpredictable environment. For testing, we need deterministic and repeatable conditions.

To create the desirable environment, I will use Deckard and its scenarios. It is a testing tool we use at CZ.NIC labs to test our implementation of a DNS resolver. Knot Resolver is still in development. The changes vary from implementation of new features to refactoring the old code. Every change of any size might introduce a new bug. That bug might change the way resolver responds in certain situations. Handcrafted tests cover very little and might not uncover this bug.

The goal of this thesis is to create a tool to automate creating scenarios and extend the set of scenarios we currently have. Unlike handcrafted scenarios, generated scenarios should compare different version of a DNS resolver implementation or even different implementations. Creating scenarios manually, which is not an easy task and takes hours, might be replaced by generating them if we have the DNS servers with the desired use case. Also by using a list of queries that are required to open a web page gives us an opportunity to include content commonly used by users on the internet. This could prevent issues that would affect many people and provide more code coverage.

When the user (or selenium script) enters a domain of a web page the browser asks the DNS resolver to resolve the IP for that domain. Then the browser loads the page content from the server whose IP he got and processes the content. At this point, the browser would finish opening the simple page. However, a lot of pages use content on different servers. Examples are styles, scripts, images, advertisements, etc. For example, a bootstrap style sheet would cause querying maxcdn.bootstrapcdn.com. After processing all web page dependencies, then the page loaded successfully. [1]

Styles and scripts should cause the same queries every time. Advertisements and other content, such as applications, cause that only 13 out of 50 most visited pages have a persistent list of queries for repeated capture (the list is taken from top-1m.csv in response_differences/data in [2]). 7 of them produce a different list for PC and mobile version. The average number of extra queries is 16. These numbers come from capturing the traffic of Firefox, Chrome, Chrome for IOS and Chrome for Android with Bind, Unbound and Knot Resolver. Each combination of resolver and browser was captured five times.

2. Used Tools

Automated testing uses several tools to achieve its goals. For creating scenarios, we use Docker and Selenium. The scenarios are used by Deckard.

Selenium

Selenium is a tool for automating browser testing. Selenium supports many browsers from which Chrome and Firefox are the most important as they cover over half of the market and are available for Linux. Chrome also allows mobile browser version simulation. If loading a page is not sufficient, it is easy to extend the script for going through the page to get more of its DNS traffic. [3]

Docker

Docker is software container platform useful for capturing DNS traffic with tcpdump inside. A separated network interface allows easy filtering the required traffic. Docker also comes with selenium images making Selenium easy to use. [4]

Deckard

Deckard is a tool made and used by CZ.NIC labs for testing a DNS software in a controlled and stable environment. Deckard is designed to ensure reproducible tests for DNS resolvers. At the time of writing this document, Deckard supports Knot Resolver, Unbound, and PowerDNS specifically by scripts testing them by Deckard.

Deckard tests the resolver by playing a scenario (More about scenarios in subsection 2.3.1). To achieve deterministic behavior, Deckard requires all responses, the resolver might ask, to be in the scenario. If anything is missing, or the resolvers answer doesn't match the expected answer, the test will fail.

Deckard provides a way to simulate changing record - ranges. Deckard also supports generic answers to multiple queries by using different match values and adjusting answers from the server to fit the ID, question, etc. This comes in handy, especially when we need a delegation of a domain with many subdomains. So, instead of having multiple entries, each having the same delegation, you get only one generic.

Scenario

Scenarios comprise two parts (steps and ranges). Steps are a sequence of pairs. The first query of the pair is a question Deckard asks the resolver. The second query is a response Deckard expects. Each range represents one or more servers and contains responses to the questions the server can answer.

STEP ID QUERY and STEP ID CHECK_ANSWER define how to use the entries of following DNS queries. Each step has also ID which defines the order of the steps. Queries have adjusted format inside the entry block. Flags are in their text representation after REPLY keyword. RRSETs are placed after a line marking which section they belong to (for example SECTION ANSWER).

```
; send query to the resolver
STEP 1 QUERY
ENTRY_BEGIN ; query to be send
REPLY RD
SECTION QUESTION
www.google.com.  IN A
ENTRY_END

; compare the answer with this entry
STEP 2 CHECK_ANSWER
ENTRY_BEGIN
MATCH all ; all fields must match
REPLY QR RD RA NOERROR
SECTION QUESTION
www.google.com.  IN A
SECTION ANSWER
www.google.com.  300 IN A 1.2.3.4
SECTION AUTHORITY
SECTION ADDITIONAL
ENTRY_END
```

MATCH field defines which parts of the query have to be the same in the response resolver returned. The entry may be required to match in multiple elements. The elements can be parts of the query or, for example, sub-domain (instead of one query name, subdomains to that query are accepted as well), all (the whole query must match), etc.

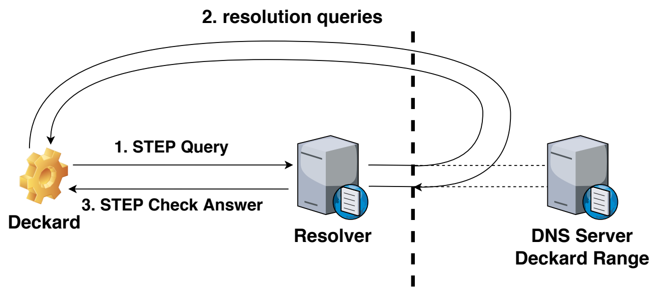

Figure 1. One Deckard step comunication

Range defines a range of ID's for which its content is visible. In the example below, it would be for IDs up to 1000. Also, IPs that should respond with the content inside the range.

```
RANGE_BEGIN 0 1000
ADDRESS 216.239.34.10
ADDRESS 2001:4860:4802:32::a

ENTRY_BEGIN
MATCH qname qtype
ADJUST copy_id
REPLY QR AA RD NOERROR
SECTION QUESTION
www.google.com.  IN A
SECTION ANSWER
www.google.com.  300 IN A 1.2.3.4
ENTRY_END

RANGE_END
```

Entries in ranges contain an extra field - ADJUST - which defines fields that should be replaced by those in the question query. It can be one of two options. One is a copy_id which copies query id and query domain name. The other one is a copy_query and copies the whole question section.

The figure 1 depicts communication between Deckard and the tested resolver which is described below.

First, Deckard sends the first query defined in steps from the scenario to the tested resolver. When the resolver attempts to communicate with any server, Deckard intercepts the communication and looks for a valid response in scenarios ranges. If we find a matching answer, the query is returned. If no answer matches Deckard doesn't respond.

When resolver comes up with an answer, Deckard compares it to the next step. If the response doesn't match the step, the test failed, else Deckard proceeds with the next pair of steps. If all steps are successful and Deckard answered all queries, the test passed.

[5]

3. Creating scenarios

The first step of creating scenarios from DNS traffic of a web browser is to capture the traffic. Selenium provides docker images with browsers and their dependencies. Using Selenium's images has several benefits. There are no queries to the resolver from different applications. For every capture, we build a new container from the image to avoid having the browser or resolver cache from the previous capture. To capture the traffic, we need to add programs and packages of our own such as the resolver, and tcpdump. Once we have built our custom image, we can capture the traffic. The process is described below and in a figure 2.

Once everything is ready, we can open the browser using Selenium and open the page we want the scenario to be based on. Selenium has a significant advantage as the test waits for the opened page to load before the script continues. When Selenium script ends, we have to stop tcpdump.

Now we have PCAP with queries between a browser and a resolver, and the resolver and other DNS servers. I experimented with two approaches to create scenarios. Both use the browser-resolver traffic to generate scenario steps. However, we can distinguish them by the process of including all necessary answers.

The first method lies in parsing the traffic between the resolver and other DNS servers and creating all required answers from it. Therefore, we need none additional requests and internet connection. However, we might experience issues with this method using DNSSEC as we cannot fake signatures. Also, the data in the scenario are more artificial than in the second method. The benefit would be in the resolvers responses being valid as IP rotation won't affect the content.

The second method doesn't use the traffic between the resolver and other DNS servers at all. We could use it to check if all queries from PCAP are present in the scenario, but that is not essential to the process. In this method, we resolve each domain (even those that are in the answer) name label-by-label (using query minimization) until we get the result. By keeping an eye on zone cuts, we can also optimize the delegations. When the response has a delegation to a zone, we only need a subdomain to match, not full query name. With this method, testing DNSSEC poses no problem as well (for every record we can include the signature if it exists). Also, the scenario will be closer to the real environment than the one from the first method. However, here we depend on the name servers. If the servers are not accessible, we cannot create the scenario. Also, IP rotation can cause different IP's

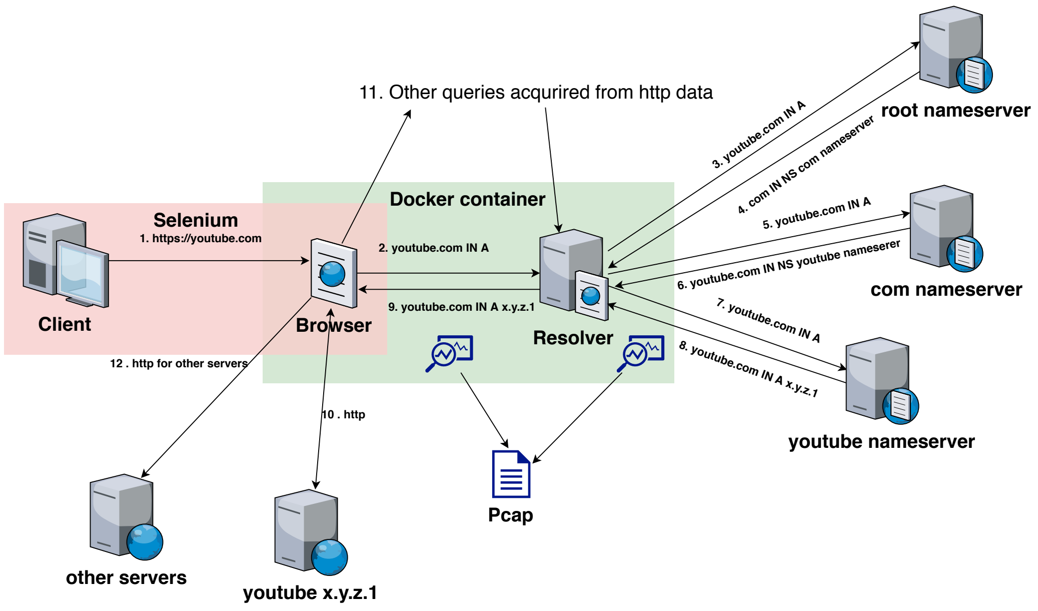

Figure 2. Traffic capture diagram for a web page

being in the scenarios ranges than in the steps. I solved this issue by replacing the record in the step by records in ranges.

To test DNSSEC, I implemented the second method. Although I have even the first method to some extent implemented, not that universal, and it is harder to create passing scenarios. The current implementation of the second method creates scenarios that pass with query minimization both on and off. Also, scenarios pass for every resolver implementation supported by Deckard. However, not every web page results in a scenario. There are still problematic records I have to solve. About 40 out of 50 scenarios were created successfully from top 50 domains from top-1m.csv in response_differences/data in [2].

Amazon's cloud service used, for instance, by Mozilla's `services.mozilla.com.` is a good example of records problematic for the first approach.

```
QUESTION SECTION:
localization.services.mozilla.com.  IN A
AUTHORITY SECTION:
services.mozilla.com.  IN NS ns-679.awsdns-20.net.
services.mozilla.com.  IN NS ns-258.awsdns-32.com.
services.mozilla.com.  IN NS ns-1655.awsdns-14.co.uk.
services.mozilla.com.  IN NS ns-1471.awsdns-55.org.
```

The resolver is delegated to four name servers with domain names in a different top-level domain. As a result, this creates several routes for a resolver to take and might cause nondeterministic behavior. However, we cannot bypass this issue as using DNSSEC requires signatures for validating the domain names of the name servers.

Servers in a delegation need to be reduced to one, for the scenario to be deterministic. We can achieve it by sorting the servers into groups and keeping only one server per group. This would be straightforward if the RRSET was consistent and IPs unique for each name server domain name. However, that is not the case with many name servers such as a cloud DNS and a larger DNS server,... For instance, we might get name servers with a domain in a different zone, different RRSETS for the same sub-domain, a different name server having the same IP address, etc.

Look at `digicert.com.` and its authoritative name servers for instance. The delegation contains two groups of servers under a different domain and responding with different RRSET.

Response from name server with `.com` zone:

```
QUESTION SECTION:
digicert.com.  IN NS
AUTHORITY SECTION:
digicert.com.  IN NS ns1.p03.dynect.net.
digicert.com.  IN NS ns2.p03.dynect.net.
digicert.com.  IN NS ns3.p03.dynect.net.
digicert.com.  IN NS ns4.p03.dynect.net.
digicert.com.  IN NS ns11.dnsmadeeasy.com.
digicert.com.  IN NS ns12.dnsmadeeasy.com.
digicert.com.  IN NS ns13.dnsmadeeasy.com.
```

Response from `ns11.dnsmadeeasy.com.`

```
QUESTION SECTION:
digicert.com.  IN NS
AUTHORITY SECTION:
digicert.com.  IN NS ns10.dnsmadeeasy.com.
digicert.com.  IN NS ns11.dnsmadeeasy.com.
digicert.com.  IN NS ns12.dnsmadeeasy.com.
digicert.com.  IN NS ns13.dnsmadeeasy.com.
digicert.com.  IN NS ns14.dnsmadeeasy.com.
digicert.com.  IN NS ns15.dnsmadeeasy.com.
```

I only used one approach to reducing servers to one. Current implementation splits name servers into several groups based on the RRSET content intersection. First, one server in the group responds to queries needed in the scenario. Then the content is merged.

I expect I will encounter more problematic records.

4. Conclusions

Generated scenarios allow for testing resolver implementations supported by Deckard even with query minimization and DNSSEC on real data. With some exceptions, they are deterministic and thus valid for testing.

Automation of DNS resolution is not simple. It requires adjustments based on finding records that cause issues. Some of them weren't solved yet and some solutions might need revisiting. Therefore, we cannot turn every domain into a scenario yet. However, even limited usability is valuable.

Acknowledgements

I would like to thank my colleague Petr Špaček from CZ.NIC for his help with Deckard and DNS and to my supervisor Matěj Grégr for his help with the written part.

References

[1] Bootstrap. *Bootstrap toolkit for developing HTML, CSS and JS*. `https://getbootstrap.com`, 2017. [Online; visited 15.01.2018].

[2] CZ.NIC. *Resolver benchmarking*. `https://gitlab.labs.nic.cz/knot/resolver-benchmarking/tree/master/`, 2017. [Online; visited 22.01.2018].

[3] *Selenium*. `https://www.seleniumhq.org/`, 2018. [Online; visited 01.04.2018].

[4] Docker Inc. *Docker*. `https://www.docker.com/`, 2018. [Online; visited 01.04.2018].

[5] CZ.NIC. *Deckard*. `https://gitlab.labs.nic.cz/knot/deckard`, 2017. [Online; visited 23.01.2018].