BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
and Communication

MASTER'S THESIS

Brno, 2018

Bc. Karel Nekuža

BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

CLIENT SIDE DNSSEC DEPLOYMENT

NASAZENÍ DNSSEC NA KLIENTSKÉ STRANĚ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. Karel Nekuža

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Zdeněk Martinásek, Ph.D.

BRNO 2018

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Karel Nekuža ***ID:*** 155204

Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Nasazení DNSSEC na klientské straně

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s protokolem DNS (Domain Name System) a jeho rozšířením DNSSEC (DNS Security Extensions). Analyzujte problémy, které mohou vzniknout na straně klienta při přechodu mezi různými sítěmi, soustřeďte se na RFC 8027 - DNSSEC Roadblock Avoidance. Analyzujte a porovnejte současné řešení DNSSEC vhodné pro nasazení na klientské straně. Porovnejte existující DNS resolvery vhodné pro nasazení na straně klienta pro lokální DNSSEC validaci. Analyzujte dostupný software pro konfiguraci sítě, který je běžně využíván v linuxových distribucích a zaměřte se na možnosti integrace s jiným aplikacemi. Navrhněte řešení, které bude schopné reagovat na změny v síťové konfiguraci a nakonfiguruje lokálně běžící DNS resolver tak, aby byl schopen provádět DNSSEC validaci. Výslednou konfiguraci otestujte a výsledky přehledně zpracujte.

DOPORUČENÁ LITERATURA:

[1] BLACKA, David, et al. DNS security (DNSSEC) hashed authenticated denial of existence. 2008.

[2] CONRAD, David. Indicating resolver support of DNSSEC. 2001.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.

Konzultant: Ing. Tomáš Hozza

prof. Ing. Jiří Mišurec, CSc.

předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRACT

This thesis deals with a problem of end-user's access to the DNSSEC validation. It explores the possibilities to implement and configure a locally running resolver to address the security issue. It proposes a solution for Fedora Workstation operating system. The solution is implemented and compared to the current solution.

KEYWORDS

domain name system, DNSSEC, security extensions, Fedora, Linux, resolver, Unbound, NetworkManager

ABSTRAKT

Diplomová práce se zabývá problémem přístupu koncového uživatele k odpovědím ověřeným pomocí protokolu DNSSEC. Práce posuzuje možnosti nasazení a nastavování resolveru za účelem zlepšení bezpečnosti pro koncové uživatele. V práci je navrhnuto řešení problému pro operační systém Fedora Workstation. Navrhnuté řešení je realizováno a porovnáno s již existujícím řesením.

KLÍČOVÁ SLOVA

domain name system, DNSSEC, security extensions, Fedora, Linux, resolver, Unbound, NetworkManager

NEKUŽA, Karel. *Client side DNSSEC deployment*. Brno, 2017, 50 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications. Advised by Ing. Zdeněk Martinásek, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 2.63; `http://latex.feec.vutbr.cz`

DECLARATION

I declare that I have written the Master's Thesis titled "Client side DNSSEC deployment" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno . . . .

author's signature

ACKNOWLEDGEMENT

I would like to thank the thesis supervisor Ing. Zdeněk Martinásek, Ph.D for professional guidance, consultation, patience and valuable suggestions. I would like to thank the thesis consultant Ing. Tomáš Hozza for professional guidance, consultation, patience and valuable suggestions.

Brno . . . .

author's signature

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

ACKNOWLEDGEMENT

Research described in this Master's Thesis has been implemented in the laboratories supported by the SIX project; reg. no. CZ.1.05/2.1.00/03.0072, operational program Výzkum a vývoj pro inovace.

Brno ....

author's signature

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

CONTENTS

1 Theoretical part **13**

1.1 Domain name system 13

1.1.1 Resource record 14

1.1.2 Communication 14

1.1.3 Master file 17

1.1.4 Domain name server 18

1.1.5 Inverse queries 19

1.1.6 Resolver 19

1.2 Domain name system security extensions 21

1.2.1 Concepts 21

1.2.2 Authentication and integrity 23

1.2.3 States 23

1.2.4 Resolvers 24

1.2.5 EDNS0 24

1.2.6 Resolver tests 25

2 Practical part **26**

2.1 Environment 26

2.2 Solution proposal 26

2.3 Unbound 27

2.3.1 Configuration 27

2.3.2 Deployment 28

2.4 NetworkManager 28

2.4.1 Network change 29

2.5 Watcher 29

2.5.1 Main cycle 30

2.5.2 Received signal 30

2.5.3 Name servers 30

2.5.4 Devices 31

2.5.5 Unbound configuration 31

2.6 Unreliability of resolution 32

2.6.1 Description 32

2.6.2 Tried solutions 35

2.6.3 Untried solutions 35

2.7 Comparison with *DNSSEC-trigger* 36

2.7.1 Trigger script 36

2.7.2 Other OS 36

3 Conclusion **37**

Bibliography **38**

List of symbols, physical constants and abbreviations **39**

List of appendices **40**

A Full packet examples **41**

B Source code of watcher.py **46**

LIST OF FIGURES

1.1 Simplified DNS overview 13
1.2 DNS message header[3] 16
1.3 Example of a DNS query structure[3] 17
1.4 Authentication chain 21
1.5 DNSSEC header[8] 22
2.1 Solution design 27

LIST OF TABLES

1.1 TYPE field values examples 15
1.2 CLASS field values examples 16

LISTINGS

2.1 query for nic.cz 32
2.2 unsigned response for nic.cz 33
2.3 query for dnssec-deployment.org 34
2.4 signed response for dnssec-deployment.org 34
A.1 query for nic.cz 41
A.2 response for nic.cz 41
A.3 query for dnssec-deployment.org 42
A.4 response for dnssec-deployment.org 42
B.1 source code for solution using threads 46
B.2 source code for solution using the `resolve_async` function 48

INTRODUCTION

There are many services and protocols utilized on the Internet that do not take security into consideration. The DNS (Domain Name System) is one of them and it is a crucial and necessary part of the Internet as we know it today. There are three security aspects to consider. They are confidentiality, integrity and authentication. The first aspect does not have to be considered since DNS data that is being sent can be asked for by other stations on the network. The latter two should be a part of the DNS system, but it was not designed with security in mind. Security addition to the system was introduced in a form of DNSSEC (Domain name system security extensions). It provides much needed authenticity and integrity, although just to a certain point in a network. That point is the last validating resolver. To make full use of DNSSEC, additional settings need to be made. There are two ways to provide security for DNS data from the end-users' point of view. The first way is to secure the channel between the end-user and a resolver that is capable of DNSSEC validation. Other way is to run local validating resolver on the end-user's workstation.

This thesis focuses on running a local validating resolver to provide a DNSSEC authenticated services. Its objective is to find a solution for automatic deployment and configuration of the resolver without having an impact on normal user experience. To achieve this there need to be automatic setting of resolver during start of the system, detection and reaction when the network environment changes and testing of possibilities on any network. The outline of the solution is described in the practical part of the project. The theoretical part focuses on providing explanations of mechanisms, principles and concepts used in the practical part.

1 THEORETICAL PART

1.1 Domain name system

DNS (Domain Name System) is a protocol for naming resources, services and devices accessible through network. It provides more understandable designation and adds another layer of virtualization.

It retrieves information based on a query, sent from a client and processed by resolvers and name servers. The information is searched hierarchically from the root zone as shown in figure 1.1. All terms are explained further in this theoretical part. If there is no record of a name server address higher level zone is asked.

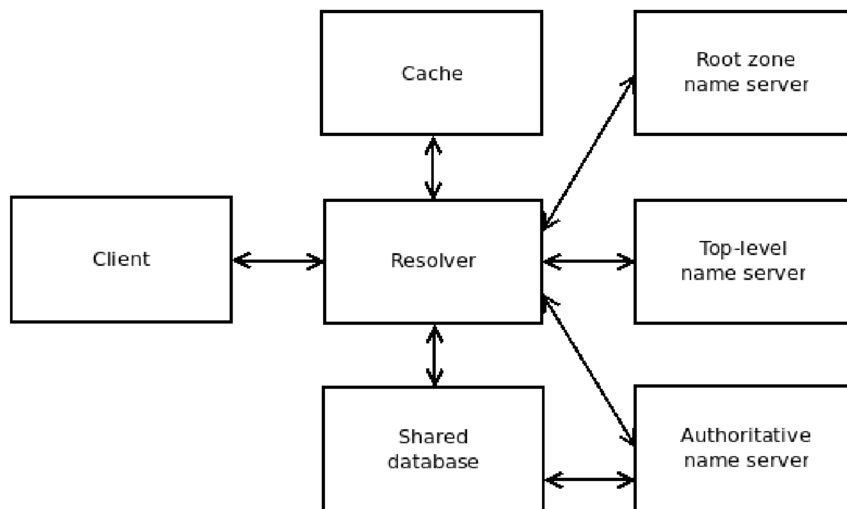

Fig. 1.1: Simplified DNS overview

Domain is a sphere of knowledge identified by a name. Typically, the knowledge is a collection of facts or a number of network points or addresses. In this case the term domain can be interchangeable with the term zone.

Zones are parts of a domain name space managed by a single entity. It contains a database for a whole subtree of domain space. The data needs to be periodically checked from a local or another name server whether it is up-to-date[3].

Root zone is labeled ".". It is the highest level domain available. The root zone is operated by root servers with specified addresses. Responsibility for it is placed with IANA (Internet Assigned Numbers Authority)[4].

Shared database can be located between the name server and the resolver. It usually contains authoritative data provided and periodically updated by the name server and cached data taken from previous resolver requests[3].

Domain name may be written in many ways possible, although in order to function properly there are recommendations for preferred syntax when communicating outside of administered zone.

Case of the domain name is not taken in account but in order to avoid multiple entries single record with different case characters should not be added. Case differences may be discarded when structured database is formed[3].

1.1.1 Resource record

Every domain name system consists of RRs (Recourse Record). RR contains retrievable information, which is the reason to run DNS. It consists of six parts mentioned below. Recourse record fields are:

- NAME – variable length field representing designation of the node connected to the record
- TYPE – 2 byte field containing number specifying RR type
- CLASS – 2 byte field defining class of a record
- TTL – 4 byte field defining TTL (Time To Live) in seconds until the record is discarded
- RDLENGTH – 2 byte field specifying the number of bytes used for RDATA field
- RDATA – field carrying needed information.

List of TYPE value examples can be found in tab. 1.1 [3]. Classes used in DNS can be associated with different network types. Possible classes are mentioned in tab. 1.2.

1.1.2 Communication

It is preferred for DNS to communicate with a client using UDP (User Datagram Protocol). For database update, shown in figure 1.1, it is preferred to use TCP (Transmission Control Protocol) for its reliability. Both types of transmissions are made on port 53, if not set otherwise. There is default limitation in size for UDP messages of 512 bites without headers. Longer messages are truncated. Communication inside DNS is carried out by using messages with constant format. There are five sections named: Header, Question, Answer, Authority and Additional[3].

Tab. 1.1: TYPE field values examples

TYPE	Numerical value	Record
A	1	IPv4 host address
NS	2	Authoritative name server
CNAME	5	Canonical name for an alias
SOA	6	Start of a zone of authority
MB	7	Mailbox domain name
MG	8	Mail group number
MR	9	Mail rename domain name
NULL	10	Null
PTR	12	Domain name pointer
MX	15	Mail exchange
TXT	16	Text string
RP	17	Responsible person
SIG	24	Signature
KEY	25	Key record
AAAA	28	IPv6 host address
NAPTR	35	Naming authority pointer
KX	36	Key exchanger record
CERT	37	Certificate record
DS	43	Delegation signer
SSHFP	44	SSH public key fingerprint
IPSECKEY	45	IPsec key
RRSIG	46	DNSSEC signature
NSEC	47	Next secure record
DNSKEY	48	DNS key record
DHCID	49	DHCP identifier
NSEC3	50	Next secure record version 3
NSEC3PARAM	51	NSEC3 parameters
TLSA	52	TLSA certificate association
CDS	59	Child copy of DS record
CDNSKEY	60	Child copy of DNSKEY record
OPENPGPKEY	61	OpenPGP public key
TSIG	250	Transaction signature
CAA	257	Certification authority authorization
TA	32768	DNSSEC trust authorities
DLV	32769	DNSSEC lookaside validation record

Tab. 1.2: CLASS field values examples

CLASS	Numerical value	Description
IN	1	the Internet
CS	2	CSNET (Computer Science Network)
CH	3	Chaosnet
HS	4	Hesiod

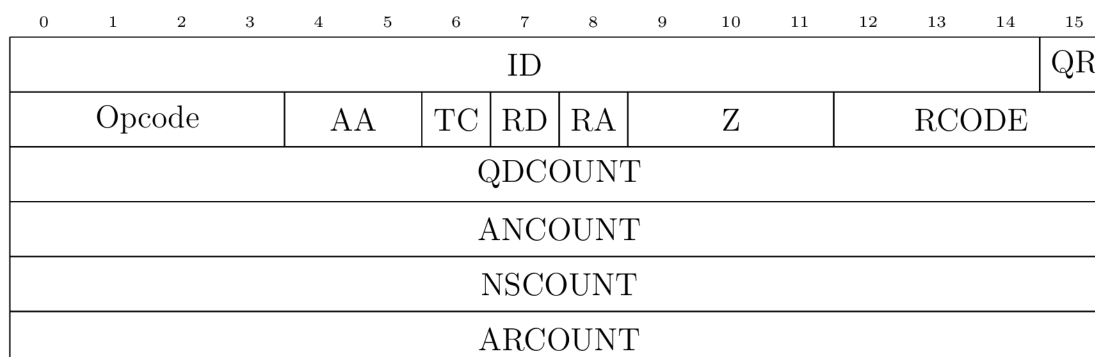

Fig. 1.2: DNS message header[3]

Unused sections are excluded. Present sections and their number can be viewed in last 4 fields in header called QDCOUNT, ANCOUNT, NSCOUNT, ARCOUNT[3]. Visual representation is in figure 1.2. With introduction of other RFC (Request For Comments) documents, DNS header was improved by dedicating some of the unused bits. This is further explained in chapter 1.2.

Other parameters in header are:

- ID – identifier serves to match question to the reply
- QR – differentiates between query, marked 0, and response, marked 1
- OPCODE – defines query type
- AA – authoritative answer
- TC – TrunCation is used when message needs to be truncated because of its size
- RD – recursion desired
- RA – recursion available
- Z – unused bits
- RCODE – error messages used in responses.

Question section is used in most queries. It contains three fields named QNAME, QTYPE and QCLASS representing domain name, query type and query class.

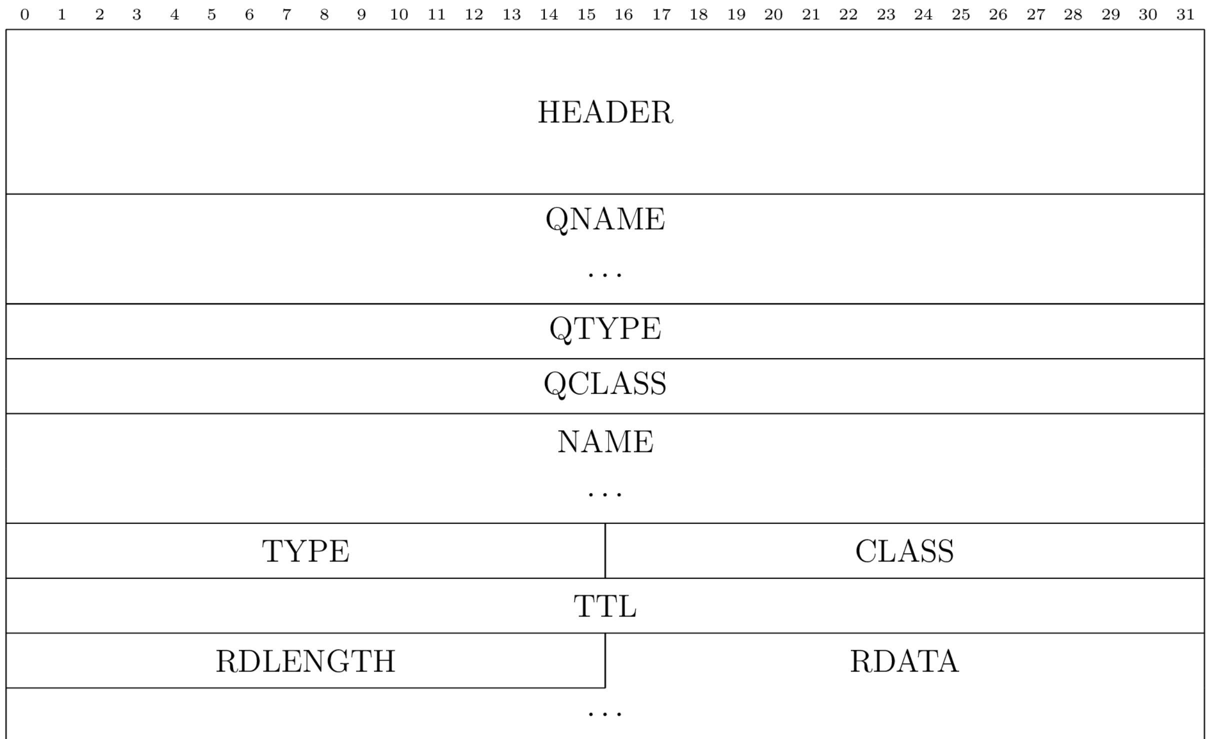

Fig. 1.3: Example of a DNS query structure[3]

Answer, authority, and additional section have identical format, whose parameters can be seen in section 1.1.1. There is a method for size reduction using pointer to the previous recourse record in the same message. The pointer has size of 2 bytes and it is signalized by the first 2 bits, which are set to 1[3].

1.1.3 Master file

The master file is a text file containing RRs of certain zone. Items in file are separated with spaces or tabs. Comment can be written using semicolon. There are five types of entries:

- blank entry
- entry containing RR
- entry containing RR and the domain name
- control entry with $ORIGIN keyword and the domain name
- control entry with $INCLUDE keyword and the file name.

The keyword $ORIGIN defines the start of the zone file, if it is detected later, it rewrites domain names to stated domain names after the keyword. The keyword $INCLUDE can insert named files to file it is used in. It does not reset the $ORIGIN

keyword if contained. RR entries are assigned to the domain name mentioned before the RR or to the domain name mentioned before if it is not stated. Domain names can be stated as relative or absolute. The absolute domain name ends with the root zone marked as a dot. From there it specifies the top-level domain and then it gradually progresses. Relative domain name does not include root zone, but extends the domain name in the $ORIGIN control entry. Errors in a master file are not acceptable since even a single error can corrupt the whole file. In order to avoid the corruption, loading of the master file should be stopped after encountering an error. Four recommended checks, which can be used to check the master file[3].

- same class on all RRs
- exactly one SOA RR
- containing delegation and glue information if needed
- non-authoritative RRs should be the glue information

The glue record or the glue information is a RR that contains IP address for a name server in its delegating domain. It is needed when the name server is a host for its own domain[1].

1.1.4 Domain name server

There are several rules and recommendations to be considered when deploying and maintaining a domain name server.

Blocking UDP requests is unacceptable. There are several reasons where this could happen, for example server can be preoccupied with a data refresh.

Parallel processing of requests is necessary in order to address all requests. Various ways to serialize the processing can be used, but they should not cause a substantial delay.

Three part database is recommended but not necessary. It consists of catalog of pointers to the zones, master files for each zone and cached data. Changing of the zone file should be performed by changing the pointer. Authoritative data should be preferred over cached data.

Time should be stored in two ways, as an absolute value and as a time left to the next data refresh[3].

1.1.5 Inverse queries

Inverse queries are used in order to a obtain domain name based on a resource. They are not necessarily supported but are required to return a response to indicate if this feature is not supported. It is received in a message header with the RCODE section set to value 4. Inverse queries are used mainly for debugging and system management. Queries of this type are send with an empty question field and answer field filled in the standard query structure[2].

1.1.6 Resolver

Domain names are translated with assistance of a local agent called the resolver. It is responsible for hiding the distribution of data among name servers from the user. It is usual for the resolver to address several name servers in order to finish the translation. The resolver has no need to know the addresses of all the the name servers, it can be obtained from other name servers. It is also responsible for dealing with failures by reaching to redundant name servers. The communication is managed by a query, that is associated with a specific type of information needed to be retrieved.

We differentiate between authoritative data stored permanently, and cached data, which is stored in the resolver and updated according to authoritative data. Cached data does not need to contain whole copy of a database. Storing the most used parts of the database can greatly improve the system performance. After a set amount of time passes, cached data is discarded. Shared database is used for a zone check[3].

State block contains request that has not been processed. A time stamp can be added to determine whether some RRs are suitable. Timers or other limiters can also be added to prevent long request processing due to for example an error. SLIST is added to keep track of a state of the request in relation to foreign name servers.

SLIST is a structure that connects name servers and zone, that are being queried. It serves as a guide to obtain desired information according to data previously gained by resolver.

Sending queries is usually carried out by NS records. Resolver should consider the probability of receiving answer when contacting authoritative name server. The time and number of transmissions should also be taken in account. There is a timer that indicates if another action is needed. If there is no NS record to be found

the resolver contacts all name servers known to it in parallel. The resolver will keep track of response time for each name server address and consider this in its selection.

Responses are to be parsed and each part evaluated. Upon receiving response, correct formatting, TTL and a header are checked. Then it is matched to a request. This is usually done by matching an ID field and checking if appropriate field was filled in. There are three major bugs that have to be accounted for. Firstly, the response will be received from a different address than the request was sent to. Secondly, if retransmission occurs, there need to be separate times for each request but the response needs to be associated with both transmissions. Lastly, when a name server does not have a zone file, it needs to be removed from the current SLIST and a different name server should be addressed.

Cache can improve performance of a resolver, therefore a general rule is to use it as much as possible although there are exceptions for data that should not be cached[3].

- Only a whole set of RRs available for a particular owner name should be cached and not its parts.
- Authoritative data have preference over cached data, so there is no reason to keep it in cache.
- Inverse queries should not to be cached.
- Queries containing “*” in their QNAME field. This symbol is used for wildcards.
- Unsolicited responses and other unreliable data should not be cached.

1.2 Domain name system security extensions

DNSSEC (Domain name system security extensions) is a way to provide the authentication and integrity assurance to the DNS. It can also provide public key distribution. In order to provide this 2 bits had to be added to the message header and dedicated RR types that can be seen in tab. 1.1. Header bits are listed below and can be seen in figure 1.5 [8].

- CD – Checking Disabled
- AD – Authenticated Data.

DNSSEC employs an asymmetric cryptography, more specifically a digital signature. The signature is verifiable through the authentication chain, also called the chain of trust, to the trust anchor[8]. The types of queries carrying a signature can be seen in figure 1.4.

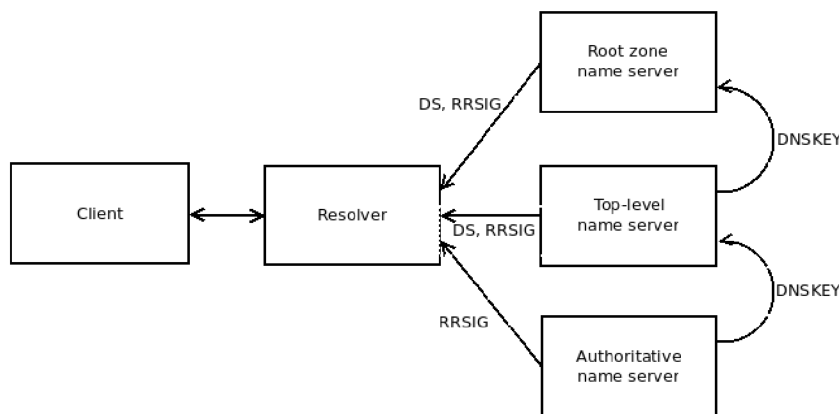

Fig. 1.4: Authentication chain

On figure 1.4 we can see a delivery of RRs necessary for proper function of the DNSSEC. After a query is sent to the resolver it searches the answer gradually from the highest zone to the zone with an exact record. Delegation, or recommendation, to a zone with better records, between zones is done by the DS type RR. This type of record contains a hash of the DNSKEY type RR. During all steps the RRSIG type RR is sent with a response to ensure validation.

An algorithm used for the digital signature is a combination of RSA (Rivest Shamir Adleman) and SHA (Secure Hash Algorithm)[10]. DNSKEY type RR contains public key to verify signatures by name servers[7].

1.2.1 Concepts

RR sets are multiple RR that are signed together rather than signing each RR separately. They are signed by zone-signing key.

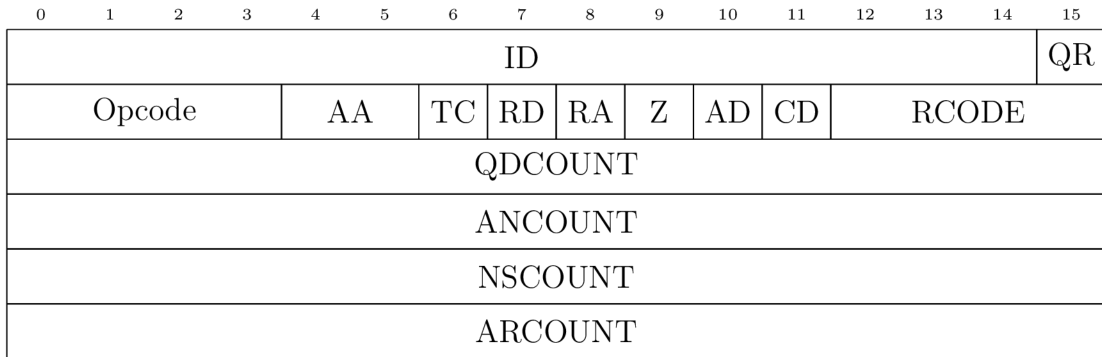

Fig. 1.5: DNSSEC header[8]

Zone-signing key is a private key used for a specific zone. Without a unique zone-signing key no signature would be valid. A public key derived from the particular zone-signing key and in combination with the RRset can be used to validate the record. It is stored in RRSIG type RR and sent with the set.

Key-signing key creates RRSIG type RR for DNSKEY type RR that contains a key. The reason to separate these keys is to provide security high enough for an authentication chain without putting too much strain on the domain name server.

Delegation signer RR serves to transfer trust from parent to child zone. Key-signing key is hashed and given to the parent zone to be published as a DS type RR. To check a child zone, hash of the key-signing key needs to be compared. It is prone to malfunction when changing a DS type RRs since it requires several steps to perform. Solution is to separate zone-signing keys and key-signing keys.

Authentication chain uses the key-signing keys and DS type RR to establish trust through several levels of zones, one referring to the next. The root zone has no parent zone. Authenticity of this zone is established in a “root signing ceremony”, where several subjects vouches for specific domain name servers and provide them with private and public keys.

Island of security is a signed zone without the parent containing a proper DNSKEY type RR, therefore not a part of the authentication parents chain. Secure key delivery can be done by other means.

Trust anchor is the starting point for an authentication chain. It is a DNSKEY type RR or DS type RR hash that was not obtained from a name server[8]. Usually

it a public key for the the root zone distributed in RPM (Red Hat Package Manager) package containing DNS server.

1.2.2 Authentication and integrity

The domain name server stores digital signatures in the RRSIG type RRs. These signatures are compared to the data generated on a resolver side. A specific signature is associated with a particular zone. Delivery of a public key must be provided. This can be done via another domain name server where th keys are stored in DNSKEY type RR. In this case we speak of an authentication chain[8].

NSEC type RR was designed to solve a problem of authenticating a negative response. Instead of sending error response this type of RR is sent and authenticated same way as any other[8].

NSEC3 type RR is a more secure option to the NSEC type RR. Its main purpose is to mitigate "zone walking". It is an attempt to gain a full list of RRs managed by the name server by sending queries for nonexistent DNS records. There is a simple solution to hash the domain name. This puts an additional load on the server. There can be added salt to the hash. This changes the hash value after the time when it is resigned. This and other parameters, hash algorithm, number of iterations and flags, are retrievable by the NSEC3PARAM type RR[9]. Depending on available resources the zone administrator can use NSEC instead of NSEC3[11].

1.2.3 States

Based on the result of validating a RR we differentiate four states.

Secure state is the desired result. The resolver has a complete authentication chain with trusted anchor and all signatures validated.

Insecure state is reported when a signed record of non-existence of a DS type RR is received. This proves that referenced zone cannot be validated. It is possible to mark a part of the domain as secure and a part as insecure.

Bogus is caused by a validation error. There could be various causes, the most usual ones are expired signature, unsupported algorithm, missing signature, missing data or an attempt for an attack.

Indeterminate state is caused by missing the trust anchor, so there is no way to determine which part is secure.

1.2.4 Resolvers

Security-aware resolver or a name server is the resolver or the name server capable of using the DNSSEC functionality.

Stub resolver is a downsized resolver that does not perform all the functionality and makes extensive use of recursive name servers. It depends on the security of the above mentioned name servers as well as a security of a communication channel for proper DNSSEC functionality. Whether the message was authenticated or not is indicated in the AD bit of the message header. It answers only to the recursive resolver[8].

Non-validating stub resolver is a stub resolver that delegates some of its functions to security-aware recursive name servers.

Security-aware recursive name server combines a security-aware name server and the security-aware resolver functions[8].

Security-aware resolver should be configured with at least one anchor. It must be able to perform cryptographic operations necessary to validate a digital signature and recognize and properly process important DNSSEC RRs such as DNSKEY, DS and RRSIG. It is recommended that the time of validation is taken into an account when determining a TTL[8].

1.2.5 EDNS0

EDNS0 (Extension Mechanism for DNS) provides backward compatible mechanism for extending the DNS enlarge messages. The effective use of DNSSEC requires this extension to the protocol for the same reason it was introduced. The EDNS0 provides more space for data transfer in a single query by attaching an OPT type pseudo-RR[5]. DNSSEC uses available space by adding a control bit called "DNSSEC OK" or a DO bit. The DO bit indicates whether the sender is capable of accepting signed responses. With the DO bit set to zero value the response must not contain DNSSEC attributes. Value in query is sent back in a response[6].

1.2.6 Resolver tests

Several test should be performed before using a upstream resolver to avoid resolution and validation problems due to an obstruction in a network infrastructure. When any of the test fails measures need to employed to avoid the blocking or different resolver need to be chosen[12]. Recommended tests are listed below:

- Test the UDP answers
- Test the TCP answers
- Test the EDNS0
- Test the the DO bit
- Test the the AD Bit using DNSKEY or DS type RR for algorithms 5 (RSASHA1) and 8 (RSASHA256)
- Test the return of RRSIG type RR
- Test the query for DNSKEY type RR
- Test the query for DS type RR
- Test the query for negative answers with NSEC3 type RR
- Test the query for DNAME type RR
- Test the TCP over port 53
- Test the UDP over port 53
- Test the UDP fragmentation
- Test unknown RRs
- Test the DNSKEY and DS type RR combination[12].

2 PRACTICAL PART

2.1 Environment

Working environment is Linux OS (Operating System) Fedora Workstation 27. All source code is written in python 3.6. Used utilities are:

- NetworkManager – a network interface control tool
- Unbound – a local validating resolver
- D-Bus – a basis of APIs used for data transfer
- libdns – a library used to display queries
- subprocess – a library for usage of shell commands within python
- watcher.py – a contributed python application made specifically for unbound configuration and network connection state gathering
- ipdb – a python library for debugging
- wireshark – a network protocol analyzer.

During the development the environment was changed in order to find a solution for an error described later. The original environment consisted of the features mentioned above. The difference was that it was run in a virtual environment under OS RHEL (Red Head Enterprise Linux) 7.3. More information is provided in section 2.6.

2.2 Solution proposal

The goal is to provide a validation on a client device. This will be achieved this by installing a local validation resolver. The adjective "local" suggests it runs on the same device as the original sender of the query, therefore the communication with unsigned queries can be carried out through the localhost interface.

Unbound was chosen as a resolver since it was theoretically capable all required features. Feature are validation, simple configuration during the run, sending test queries and python accessible API. More information follows is in section 2.3.

Resolver configuration needs to adapt to active connections, especially with a change of DHCP received name servers. There also needs to be an arbitrator if more connections of name servers are available or if there is a problem with the current one. If there are no suitable name servers, the resolver will perform a full recursion. These functions are executed in watcher.py.

The watcher.py will receive data from a network utility, in this case NetworkManager, and changes resolver settings accordingly. If there is no sufficient information received, it can use a different utility to obtain more information about name server addresses. This is done by sending queries for known addresses. Changes to the

resolver configuration file can be done by copying a template file with a fewer parts filled with data from a network utility or by a command if the change is small, such as a change of a name server address. A diagram of the concept is represented in picture 2.1.

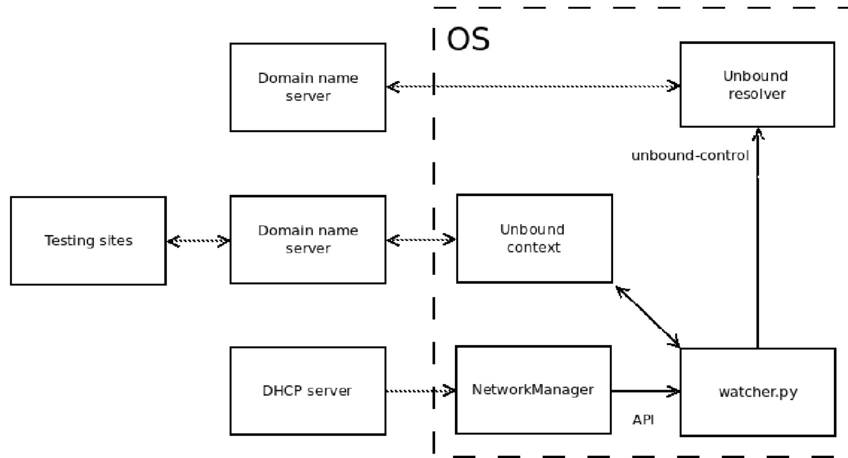

Fig. 2.1: Solution design

2.3 Unbound

Unbound is a validating, recursive and caching DNS resolver. The source code is published under BSD (Berkeley Software Distribution) license. It will be used as a local validating resolver. It was chosen for this implementation for its ability to validate signed queries and proven functionality under similar conditions in utility *DNSSEC-trigger*.

2.3.1 Configuration

Unbound is configured in "unbound.conf" file located in the directory /etc/unbound/.A restart is required to load a configuration from the file. This file is left in a default configuration and further changes are made using an unbound-control utility. The configuration file is divided into several clauses.

Server clause is the most extensive. An interface keyword is used to specify the interface that the resolver will be listening to for queries and to which it will send a response. In this case it should be set to localhost, 127.0.0.1 and ::1, since it will be set in the "resolv.conf" file in section 2.4. To set a port on the interface, there is a port keyword. The default port is 53. Do-tcp and do-udp keywords are recommended to be set to the value "yes". UDP datagrams are mainly used, although

under various conditions a name server can choose to use TCP. An access-control keyword allows access for networks or hosts. By default only localhosts' queries are processed. The auto-trust-anchor-file keyword should be set up with a path to the "root.key" file, which contains a trust anchor. The keyword pidfile specifies a path to a file containing current PID (Process ID) of the resolver. The unwanted-reply-threshold keyword is a defensive measure that will trigger an action if the number of unwanted replies reaches the number that has been set. A zero value turns the feature off. To use a specific module, the keyword module-config needs to be used. In this case I used the module "validator iterator" to enable DNSSEC usage.

Python clause can be used to add modules written in this language.

Remote-control clause contains means for configuration change without accessing the "unbound.conf" file directly.

Stub-zone clause serves as a storage of authoritative data accessible only from private zones.

Forward-zone clause contains addresses to recursive resolvers, where queries can be sent if they can not be resolved by this resolver. The name keyword specifies domain names to be forwarded. To forward all queries the keyword must be set with the root domain. Addresses to forward to are defined by the forward-addr keyword.

Include is a keyword usable anywhere in the configuration file. It copies the content of a file specified in the keyword into the configuration file.

2.3.2 Deployment

Several adjustments were made to the default unbound configuration. These are checked during a system start.

- validator – this module need to be included to configure the unbound for the DNSSEC validation
- tcp-upstream – its use will ensure that queries exceeding the limit of 512 B will be delivered

2.4 NetworkManager

NetworkManager is a default open-source network connection configuration tool used in Fedora Linux distribution. It searches for network connections and estab-

lishes them. It is possible to configure The NetworkManager using API (Application Programming Interface) written in the python named python-networkmanager. This API utilizes D-Bus (Desktop Bus). The NetworkManager rewrites the file "resolv.conf" during every system start. The file "resolv.conf" is located in the directory /etc/ and contains address of domain name servers that are used by the OS. The default configuration of the NetworkManager is set to rewrite the file with DHCP (Dynamic Host Configuration Protocol) obtained DNS addresses, this needed to be changed since the implementation requires a local validating resolver. File is rewritten during each change by the NetworkManager. To compensate, it is going to be rewritten again by the watcher.py using subprocess and echo utilities. To obtain the data, active connections must be accessed. That is be done by the use of the "ActiveConnections" class, when each connection can receive different DNS addresses. To gain received settings we can use the "Dhcp4Config" for IPv4 addresses.

2.4.1 Network change

With each change in the network, scripts in the directory "/etc/NetworkManager/dispatcher.d/" are executed. To enable a configuration change with a network change the bash script was added to send the signal `SIGUSR1` to a PID of the watcher.py. The content of directory is executed in an alphabetical order. Since the script is very simple and the connection is a crucial feature I have put it in the second place with the name "02-network-change". To follow the standard of other scripts, two digits were placed before the name. Content can be seen below.

```bash
#!/bin/bash
PID=$(cat /home/knekuza/devel/Client_side_DNSSEC/pid)
kill -USR1 $PID
```

2.5 Watcher

Watcher is a program written in python 3.6 with the purpose to configure the unbound in accordance with the data received from the NetworkManager and gained from its own testing. It receives signals from the script "02-network-change" whenever the network environment changes. It is presumed that during the start of this program the unbound resolver will be running, which is achieved by enabling the unbound service in the systemd.

2.5.1 Main cycle

The start sequence and methods executed when receiving a signal are almost identical since it is presumed that all connections can be lost during both events. There is also a defined function for receiving signal.

The start of a watcher.py with the "resolve.conf" file is overwritten, because the NetworkManager writes there DHCP received name servers. Then an instance of a class `unboundConf` is created. It populates its variables with name servers during the constructor execution. With the running resolver there is no guarantee it is capable of required tasks. The `moduleCheck` method is called to check and set the resolver if needed. Then the previously obtained name servers are checked by the `checkNameServers` method. The forwarding is periodically set according to the results.

2.5.2 Received signal

The library "signal" and a method with the same name was used for inter-process communication. Only two arguments are needed. The first is the type of the signal that can be used the same library, and second is a handler. A handler is a function called when signal is received. It is named "receive_signal" in this case. As discussed in previous paragraph it creates the `unboundConf` instance, retrieves addresses, checks them and configures them accordingly. The handler is called with two arguments, the signal identification and a pointer to the current stack frame. Neither of those are useful for the intended usage.

2.5.3 Name servers

For each name server there is an instance of a class named `nameServer`. The `nameServer` has attributes that represent the type of queries it supports. The values of these attributes are determined by a method `sendQuery`. A constructor sets attributes and the unbound context by calling a method `setContext`. In order for this class to be comparable, methods `__hash__` and `__eq__` were modified to compare according to the IP address. This will be useful in a class `device` which could contain several instances of this class.

The method `sendQuery` utilizes function of the unbound API named `resolve`. It sends queries to known sites via the tested name server. The response then indicates the name server's capability to resolve various query types. It is set under an unbound context, which is a separate setting for query resolution. The context is represented by a class `ub_ctx` and each instance sets its own context using the method `setContext`. The `resolve_async` method was also tested to replace the

`resolve`. It uses a callback function that is called whenever a response is received from sent queries. This proved less stable then the `resolve` function so I chose to use the faster option. In order to shorten the time for determining these attributes for every use of the `resolve` function, there is a separate thread.

The method `setContext` contains configuration for the instance of the resolver. All setting are methods of the unbound context class. They determine how the queries will be resolved. The resolver has to be set to provide validation by the `set_option("module-config","validator iterator")` and has to have an access to the trust anchor, `add_ta_file("/var/lib/unbound/root.key")`. Above mentioned settings will be the same for all `nameServer`, unlike the forwarding settings. It is set to the IP address of the `nameServer` by method `set_fwd(str(self.address))`.

2.5.4 Devices

Class devices divides name servers to groups according to a device that can also be called an interface, on which it was received. This is done to divide name servers according to reliability and security of the connection.

2.5.5 Unbound configuration

The `unboundConf` class serves to configure the unbound recursive resolver. It is set in methods `confForward` and `moduleCheck`. The latter configures basic setting to ensure a proper function of the resolver. Name servers are stored in variable `forward` which contains a list of instances of the `device` class. They are gained through the `retrieveNameServers` method.

The `moduleCheck` method is guarantees that the resolver is capable of tasks, which will be required. It is using the unbound-control commands that are executed in the subprocess. This enables easy retrieval of output to determine the running settings of unbound. Two parameters are checked. Module configuration, that has to be set with "validator" keyword and trust anchor, that should point to the default location of trust anchor file.

The `confForward` method checks properties of a name server, specifically the ability to send signed A, AAAA and NSEC type queries. The functional name servers are then set for forwarding using subprocess and the unbound-control. After each setting of a forward zone, the cache is flushed, so there are no record from the previous queries.

The `retrieveNameServers` method communicates with the NetworkManager to gather name servers received through DHCP. Addresses are stored according to the interface they were received on. Localhost addresses and bind interfaces are

excluded. This method utilizes the NetworkManager python API available through the *python3-networkmanager* package.

The `checkNameServers` method uses the `sendQueries` method from each instance of `nameServer` to determine capabilities of a selected name server. It is also responsible for dealing with threads. For threading it uses the "threading" library. A site that is taken as a reference is chosen randomly from a list of verified sites called the `checkSites`, which is an attribute of the `unboundConf` class.

2.6 Unreliability of resolution

The success of this program dependents on a the capability to recognize a name server with functional DNSSEC properties. This was not achieved since the reliability of a resolution was not satisfactory. Some responses were either sent without the RRSIG type RR or were not received at all. I made several alterations that could fix the problem, but I did not manage to completely ensure reliable result. I tested all solution in different networks to make sure that there are no network settings affecting it. Specifically it was tried on guest and employee networks in company *Red Hat* with enterprise level of connection and managed name servers, and on VDSL (Very High Speed Digital Subscriber Line) connection. All connections had similar results.

2.6.1 Description

The resolution is done by the `resolve` method. This sends queries to the selected name server. The responses should contain the original query, requested data and a signature that can be validated. There are four listings from the wireshark as an example of the problem. Queries received without the RRSIG type RR were sent with the NAME field containing "nic.cz", queries without the signature were sent for the "dnssec-deployment.org". Both sites were proven to have functioning signed RRs available during testing outside of the watcher.py.

The first two are listings 2.1 and 2.2. Their response was sent without a signature. Listings 2.3 and 2.4 gained valid a RRSIG type RR. A symptom could be the CD bit, in this case marked "Non-authenticated data", set to a positive value but it is better to receive an unsigned response then to do not receive any since it reduces the resources consumption and gives more information about the probed name server. It is also worth pointing out that a DO bit in the EDNS0 OPT type pseudo-RR is set to "1" in both examples.

Listing 2.1: query for nic.cz

```
Domain Name System (query)
```

```
    Transaction ID: 0xd55a
    Flags: 0x0110 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...1 .... = Non-authenticated data: Acceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        nic.cz: type AAAA, class IN
            Name: nic.cz
            [Name Length: 6]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Response In: 40]
```

Listing 2.2: unsigned response for nic.cz

```
Domain Name System (response)
    Transaction ID: 0xd55a
    Flags: 0x8190 Standard query response, No error
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .0.. .... .... = Authoritative: Server is not an authority for domain
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... 1... .... = Recursion available: Server can do recursive queries
        .... .... .0.. .... = Z: reserved (0)
        .... .... ..0. .... = Answer authenticated: Answer/authority portion was not
            authenticated by the server
        .... .... ...1 .... = Non-authenticated data: Acceptable
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
    Queries
        nic.cz: type AAAA, class IN
            Name: nic.cz
            [Name Length: 6]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Answers
        nic.cz: type AAAA, class IN, addr 2001:1488:0:3::2
            Name: nic.cz
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 116
            Data length: 16
            AAAA Address: 2001:1488:0:3::2
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Request In: 39]
```

```
    [Time: 0.001757417 seconds]
```

Listing 2.3: query for dnssec-deployment.org

```
    Transaction ID: 0xdfe6
    Flags: 0x0010 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...0 .... .... = Recursion desired: Don't do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...1 .... = Non-authenticated data: Acceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        dnssec-deployment.org: type AAAA, class IN
            Name: dnssec-deployment.org
            [Name Length: 21]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Response In: 4790]
```

Listing 2.4: signed response for dnssec-deployment.org

```
Domain Name System (response)
    Length: 763
    Transaction ID: 0xdfe6
    Flags: 0x8410 Standard query response, No error
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .1.. .... .... = Authoritative: Server is an authority for domain
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...0 .... .... = Recursion desired: Don't do query recursively
        .... .... 0... .... = Recursion available: Server can't do recursive queries
        .... .... .0.. .... = Z: reserved (0)
        .... .... ..0. .... = Answer authenticated: Answer/authority portion was not
            authenticated by the server
        .... .... ...1 .... = Non-authenticated data: Acceptable
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 2
    Authority RRs: 6
    Additional RRs: 10
    Queries
        dnssec-deployment.org: type AAAA, class IN
            Name: dnssec-deployment.org
            [Name Length: 21]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Answers
        dnssec-deployment.org: type AAAA, class IN, addr 2001:41c8:20::4001
            Name: dnssec-deployment.org
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 300
            Data length: 16
            AAAA Address: 2001:41c8:20::4001
        dnssec-deployment.org: type RRSIG, class IN
            Name: dnssec-deployment.org
            Type: RRSIG (46)
            Class: IN (0x0001)
            Time to live: 300
            Data length: 169
```

```
Type Covered: AAAA (IPv6 Address) (28)
Algorithm: RSA/SHA1 (5)
Labels: 2
Original TTL: 300 (5 minutes)
Signature Expiration: May 29, 2018 22:40:03.000000000 CEST
Signature Inception: May 15, 2018 22:40:03.000000000 CEST
Key Tag: 60423
Signer's name: dnssec-deployment.org
Signature: 67007b75ad7508a0e77aefcdd4900d9491ecd85685fdc33e...
```

2.6.2 Tried solutions

Firstly, I tried using different method for resolution. The first is the `resolve_async` method in python unbound API. This allows a resolution of queries without waiting for a response. This is achieved by using the callback function. The callback function receives pointer to the structure in an argument, where it stores the data. The callback method is also a part of a the `nameServer` class. It determines the result and the type of a query to adjust class attributes. When this approach proved unsuccessful I used the `resolve` method. The waiting was evaded using different threads for each query. It lowered the number of false reposes, but not sufficiently.

Another possible reason for the problem could be the usage of UDP. The simple solution is to use TCP. There are two possible setting that can achieve this. They are the *tcp-upstream* and the *do-udp* commands. These will influence the configuration of the unbound context and will be therefore applicable only for one set of queries. The resolver needs to communicate with a workstation so turning off the resolution through the UDP could be harmful. Testing with the *tcp-upstream* setting proved to be ineffective.

Two workspaces were tried. The original workspace consisted of a virtual machine running Fedora 27 OS in Red Head Enterprise Linux 7.3 OS. There were two connections, the virtual network 192.168.122.0/24 and a bind interface. Through the bind interface it was able to receive the address of a local DNS resolver. There was a possibility that the *dnsmasq* running on the RHEL interferes with the resolution, since it was compiled without the DNSSEC validation. I tried the second workspace running Fedora 27 on bare-metal. Neither this solution was successful.

2.6.3 Untried solutions

There were other solutions to consider, but those were not included because of the lack of time necessary for implementation.

Probably the best option would be to choose a different programming language. The C language would be a much better choice since it can provide more optimized application and it accesses better API, than the python API. This would be useful to do even without having to resolve discussed problem.

Another solution could be to ask specifically for the RRSIG type RR. It would have to be distinguished which responses arrived without a signature and sent queries for their RRSIG type RRs.

2.7 Comparison with *DNSSEC-trigger*

The *DNSSEC-trigger* is the only solution I was able to find for the DNSSEC deployment on a client device that uses the locally running resolver.

The main difference, which was the reason for not only updating the *DNSSEC-trigger* with new options, is in the trigger scrip. It is located in directory the "dispatcher.d" mentioned in chapter 2.4.

2.7.1 Trigger script

The *DNSSEC-trigger* is using the part to retrieve and select information from the NetworkManager to configure the unbound resolver and forward zones. During the run time of these procedures, it is necessary to wait for the completion before other scrips in the directory can be executed. It is put in the second place in order of execution within the directory since it is a crucial feature, but that makes it even more dependent on a swift execution for other services. Using the `SIGUSR1` signal mitigates the need for waiting since it is done by a different program.

2.7.2 **Other** OS

There is downside to the use a signal in a different OS. The library "signal" is not made to receive the `SIGUSR1` type. This would require to choose a different type or change the basic execution to that of a *DNSSEC-trigger* where no signals are needed.

3 CONCLUSION

The diploma thesis deals with possibilities of DNSSEC query usage on the client side utilizing a local validating resolver. The proposal of this thesis is directly compared to the existing solution named *DNSSEC-trigger*. The purpose is to create a proof of concept for an improvement for this project.

In the first chapter of the theoretical part there can be found terms and principles explanations for the basic DNS functionality, most importantly there is the structure of queries, headers within them, and their types. The second chapter builds on the first and explains an extension to the system, the DNSSEC.

The first chapter of the practical part describes the working environment. The environment was changed during the development. Only common features are described in this chapter, other information is in the following paragraphs. The second chapter outlines the overall design, ideas and connections. It specifies the tools described in more detail in the next two chapters. These are the recursive resolver, the unbound, and the networking tool, the NetworkManager, which are used in both projects. The fifth chapter describes the program watcher.py. This is a software specifically designed for this solution. The sixth chapter describes a problem with realization of the intended functionality and goes through possible solutions. The last chapter compares all of the above with a solution that is provided by *DNSSEC-trigger* in key points.

The solution proposed in this thesis did not function as intended, therefore, I cannot compare the results, although the features can be compared. The basic difference is in the usage of a signal instead of executing it from a shell script in the "the dispatcher.d" directory. The downside to the signal is that there is no information about the change and all interfaces need to be tested. This causes only a minor setback since the same information is later retrieved from the NetworkManager.

The watcher.py is functional since it uses a fallback mechanism of a full recursion if there are no available name servers. That is the same situation when tests fail to prove that the found name servers are capable of validating. This and other properties make it slower than the *DNSSEC-trigger*. I advise using the *DNSSEC-trigger* for its swiftness and reliability of the resolution. I also recommend testing the use of a signal apart from a direct execution in the *DNSSEC-trigger*.

BIBLIOGRAPHY

[1] Lottor, M. *Domain administrators operations guide*, STD 13, RFC 1033, November 1987.

[2] Mockapetris, P. *Domain names - concepts and facilities*, STD 13, RFC 1034, November 1987.

[3] Mockapetris, P. *Domain names - implementation and specification*, STD 13, RFC 1035, November 1987.

[4] Manning, B., Vixie, P. *Operational Criteria for Root Name Servers*, RFC 2010, October 1996.

[5] Vixie, P. *Extension Mechanisms for DNS (EDNS0)*, RFC 2671, August 1999.

[6] Conrad, D. *Indicating Resolver Support of DNSSEC*, RFC 3225, December 2001.

[7] Kolkman, O., Schlyter, J., Lewis, E. *Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag*, RFC 3757, April 2004.

[8] Arends, R., Austein, R., Larson, M., Massey, D., and S.Rose. *DNS Security Introduction and Requirements*, RFC 4033, Colorado State University, March 2005.

[9] Laurie, B., Sisson B., Arends, R. and Blacka, B. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, RFC 5155, March 2008.

[10] Jansen, J. *Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*, RFC 5702, October 2009.

[11] Kolkman, O., Mekking, W. and Gieben, R. *DNSSEC Operational Practices, Version 2*, RFC 6781, December 2012.

[12] Hardaker, W., Gudmundsson, O., Krishnaswamy, S.. *DNSSEC Roadblock Avoidance*, RFC 8027, November 2016.

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

DNS	Domain Name System
RR	Recourse Record
TTL	Time To Live
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
DNSSEC	Domain name system security extensions
BSD	Berkeley Software Distribution
DNF	Dandified yum
API	Application Programming Interface
D-Bus	Desktop Bus
OS	Operating System
DHCP	Dynamic Host Configuration Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
PID	Process ID
IP	Internet Protocol
RHEL	Red Head Enterprise Linux
VDSL	Very High Speed Digital Subscriber Line
RFC	Request For Comments
IANA	Internet Assigned Numbers Authority
RSA	Rivest Shamir Adleman
SHA	Secure Hash Algorithm
EDNS0	Extension Mechanism for DNS
RPM	Red Hat Package Manager

LIST OF APPENDICES

A Full packet examples **41**

B Source code of watcher.py **46**

A FULL PACKET EXAMPLES

Listing A.1: query for nic.cz

```
No.     Time           Source                  Destination           Protocol Length Info
     39 2.426188709    2a00:1028:83a0:5e16:f171:49ac:d480:ba98 2a00:1028:83a0:5e16::1 DNS
         97     Standard query 0xd55a AAAA nic.cz OPT

Frame 39: 97 bytes on wire (776 bits), 97 bytes captured (776 bits) on interface 0
Ethernet II, Src: IntelCor_77:96:29 (00:28:f8:77:96:29), Dst: AsustekC_ba:59:10 (38:d5:47:ba
    :59:10)
Internet Protocol Version 6, Src: 2a00:1028:83a0:5e16:f171:49ac:d480:ba98, Dst: 2a00:1028:83a0:5
    e16::1
User Datagram Protocol, Src Port: 53145, Dst Port: 53
Domain Name System (query)
    Transaction ID: 0xd55a
    Flags: 0x0110 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...1 .... = Non-authenticated data: Acceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        nic.cz: type AAAA, class IN
            Name: nic.cz
            [Name Length: 6]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Response In: 40]
```

Listing A.2: response for nic.cz

```
No.     Time           Source                  Destination           Protocol Length Info
     40 2.427946126    2a00:1028:83a0:5e16::1 2a00:1028:83a0:5e16:f171:49ac:d480:ba98 DNS
         125    Standard query response 0xd55a AAAA nic.cz AAAA 2001:1488:0:3::2 OPT

Frame 40: 125 bytes on wire (1000 bits), 125 bytes captured (1000 bits) on interface 0
Ethernet II, Src: AsustekC_ba:59:10 (38:d5:47:ba:59:10), Dst: IntelCor_77:96:29 (00:28:f8
    :77:96:29)
Internet Protocol Version 6, Src: 2a00:1028:83a0:5e16::1, Dst: 2a00:1028:83a0:5e16:f171:49ac:
    d480:ba98
User Datagram Protocol, Src Port: 53, Dst Port: 53145
Domain Name System (response)
    Transaction ID: 0xd55a
    Flags: 0x8190 Standard query response, No error
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .0.. .... .... = Authoritative: Server is not an authority for domain
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... 1... .... = Recursion available: Server can do recursive queries
        .... .... .0.. .... = Z: reserved (0)
        .... .... ..0. .... = Answer authenticated: Answer/authority portion was not
            authenticated by the server
        .... .... ...1 .... = Non-authenticated data: Acceptable
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 1
```

```
    Queries
        nic.cz: type AAAA, class IN
            Name: nic.cz
            [Name Length: 6]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Answers
        nic.cz: type AAAA, class IN, addr 2001:1488:0:3::2
            Name: nic.cz
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 116
            Data length: 16
            AAAA Address: 2001:1488:0:3::2
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Request In: 39]
    [Time: 0.001757417 seconds]
```

Listing A.3: query for dnssec-deployment.org

```
Domain Name System (query)
    Length: 50
    Transaction ID: 0xdfe6
    Flags: 0x0010 Standard query
        0... .... .... .... = Response: Message is a query
        .000 0... .... .... = Opcode: Standard query (0)
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...0 .... .... = Recursion desired: Don't do query recursively
        .... .... .0.. .... = Z: reserved (0)
        .... .... ...1 .... = Non-authenticated data: Acceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
    Queries
        dnssec-deployment.org: type AAAA, class IN
            Name: dnssec-deployment.org
            [Name Length: 21]
            [Label Count: 2]
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
    Additional records
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
                1... .... .... .... = DO bit: Accepts DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Response In: 4790]
```

Listing A.4: response for dnssec-deployment.org

```
Domain Name System (response)
    Length: 763
    Transaction ID: 0xdfe6
    Flags: 0x8410 Standard query response, No error
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .1.. .... .... = Authoritative: Server is an authority for domain
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...0 .... .... = Recursion desired: Don't do query recursively
        .... .... 0... .... = Recursion available: Server can't do recursive queries
        .... .... .0.. .... = Z: reserved (0)
```

```
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was not
        authenticated by the server
    .... .... ...1 .... = Non-authenticated data: Acceptable
    .... .... .... 0000 = Reply code: No error (0)
Questions: 1
Answer RRs: 2
Authority RRs: 6
Additional RRs: 10
Queries
    dnssec-deployment.org: type AAAA, class IN
        Name: dnssec-deployment.org
        [Name Length: 21]
        [Label Count: 2]
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
Answers
    dnssec-deployment.org: type AAAA, class IN, addr 2001:41c8:20::4001
        Name: dnssec-deployment.org
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 16
        AAAA Address: 2001:41c8:20::4001
    dnssec-deployment.org: type RRSIG, class IN
        Name: dnssec-deployment.org
        Type: RRSIG (46)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 169
        Type Covered: AAAA (IPv6 Address) (28)
        Algorithm: RSA/SHA1 (5)
        Labels: 2
        Original TTL: 300 (5 minutes)
        Signature Expiration: May 29, 2018 22:40:03.000000000 CEST
        Signature Inception: May 15, 2018 22:40:03.000000000 CEST
        Key Tag: 60423
        Signer's name: dnssec-deployment.org
        Signature: 67007b75ad7508a0e77aefcdd4900d9491ecd85685fdc33e...
Authoritative nameservers
    dnssec-deployment.org: type NS, class IN, ns ns1.sea1.afilias-nst.info
        Name: dnssec-deployment.org
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 27
        Name Server: ns1.sea1.afilias-nst.info
    dnssec-deployment.org: type NS, class IN, ns ns1.mia1.afilias-nst.info
        Name: dnssec-deployment.org
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 11
        Name Server: ns1.mia1.afilias-nst.info
    dnssec-deployment.org: type NS, class IN, ns ns1.ams1.afilias-nst.info
        Name: dnssec-deployment.org
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 11
        Name Server: ns1.ams1.afilias-nst.info
    dnssec-deployment.org: type NS, class IN, ns ns1.yyz1.afilias-nst.info
        Name: dnssec-deployment.org
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 11
        Name Server: ns1.yyz1.afilias-nst.info
    dnssec-deployment.org: type NS, class IN, ns ns1.hkg1.afilias-nst.info
        Name: dnssec-deployment.org
        Type: NS (authoritative Name Server) (2)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 11
        Name Server: ns1.hkg1.afilias-nst.info
    dnssec-deployment.org: type RRSIG, class IN
        Name: dnssec-deployment.org
        Type: RRSIG (46)
        Class: IN (0x0001)
        Time to live: 300
        Data length: 169
```

```
            Type Covered: NS (authoritative Name Server) (2)
            Algorithm: RSA/SHA1 (5)
            Labels: 2
            Original TTL: 300 (5 minutes)
            Signature Expiration: May 29, 2018 22:40:03.000000000 CEST
            Signature Inception: May 15, 2018 22:40:03.000000000 CEST
            Key Tag: 60423
            Signer's name: dnssec-deployment.org
            Signature: 67003f397006abd9d30c058a0823f07b312a346ee347f6b2...
    Additional records
        ns1.ams1.afilias-nst.info: type A, class IN, addr 65.22.6.79
            Name: ns1.ams1.afilias-nst.info
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 4
            Address: 65.22.6.79
        ns1.hkg1.afilias-nst.info: type A, class IN, addr 65.22.6.1
            Name: ns1.hkg1.afilias-nst.info
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 4
            Address: 65.22.6.1
        ns1.mia1.afilias-nst.info: type A, class IN, addr 65.22.7.1
            Name: ns1.mia1.afilias-nst.info
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 4
            Address: 65.22.7.1
        ns1.sea1.afilias-nst.info: type A, class IN, addr 65.22.8.1
            Name: ns1.sea1.afilias-nst.info
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 4
            Address: 65.22.8.1
        ns1.yyz1.afilias-nst.info: type A, class IN, addr 65.22.9.1
            Name: ns1.yyz1.afilias-nst.info
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 4
            Address: 65.22.9.1
        ns1.hkg1.afilias-nst.info: type AAAA, class IN, addr 2a01:8840:6::1
            Name: ns1.hkg1.afilias-nst.info
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 16
            AAAA Address: 2a01:8840:6::1
        ns1.mia1.afilias-nst.info: type AAAA, class IN, addr 2a01:8840:7::1
            Name: ns1.mia1.afilias-nst.info
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 16
            AAAA Address: 2a01:8840:7::1
        ns1.sea1.afilias-nst.info: type AAAA, class IN, addr 2a01:8840:8::1
            Name: ns1.sea1.afilias-nst.info
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 16
            AAAA Address: 2a01:8840:8::1
        ns1.yyz1.afilias-nst.info: type AAAA, class IN, addr 2a01:8840:9::1
            Name: ns1.yyz1.afilias-nst.info
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 3600
            Data length: 16
            AAAA Address: 2a01:8840:9::1
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x8000
```

```
            1... .... .... .... = DO bit: Accepts DNSSEC security RRs
            .000 0000 0000 0000 = Reserved: 0x0000
        Data length: 0
[Request In: 4659]
[Time: 0.176171254 seconds]
```

B SOURCE CODE OF WATCHER.PY

Listing B.1: source code for solution using threads

```
#!/usr/bin/python3
import os, time, signal, subprocess, random
import NetworkManager, ipaddress
from unbound import ub_ctx,RR_TYPE_A,RR_CLASS_IN, RR_TYPE_A
from unobund import RR_TYPE_AAAA, RR_TYPE_RRSIG, RR_TYPE_NSEC, RR_TYPE_NSEC3
import ipdb
import threading

class nameServer:

    def sendQuery(self, site, number):
        if(number == 0):
            self.status[0], result = self.ctx.resolve(site, RR_TYPE_A, RR_CLASS_IN)
            if(result.secure and result.havedata):
                self.a = True
            print(number)
        elif(number == 1):
            self.status[1], result = self.ctx.resolve(site, RR_TYPE_AAAA, RR_CLASS_IN)
            if(result.secure and result.havedata):
                self.aaaa = True
            print(number)
        elif(number == 2):
            self.status[2], result = self.ctx.resolve("nefunguje."+site, RR_TYPE_A, RR_CLASS_IN)
            if(not result.havedata):
                self.nsec = True
            print(number)

    def setContext(self):
        self.ctx.add_ta_file("/var/lib/unbound/root.key")
        self.ctx.set_fwd(str(self.address))
        self.ctx.set_option("tcp-upstream", "yes")
        self.ctx.set_option("module-config","validator iterator")
        self.ctx.set_option("key-cache-size", "32m")
        self.ctx.set_option("max-udp-size", "8192")
        self.ctx.set_option("prefetch-key", "yes")
        self.ctx.set_option("do-tcp", "yes")
        self.ctx.set_option("do-udp", "yes")

    def __eq__(self, other):
        if isinstance(other, nameServer):
            return(self.address == other.address)
        else:
            return False

    def __hash__(self):
        return hash(self.address)

    def __init__(self, address):
        self.address = ipaddress.ip_address(address)
        self.a = False
        self.aaaa = False
        self.nsec = False
        self.status = [1,1,1]
        self.ctx = ub_ctx()
        self.setContext()


class device:

    nameservers = set()

    def addNameserver(self, nameserver):
        self.nameservers.add(nameServer(nameserver))

    def __init__(self, name):
        self.name = name

class unboundConf:

    forward = []
    checkSites = ["isc.org",
                  "nic.cz",
                  "dnssec-deployment.org",
                  "internetsociety.org",
```

```
                "dnssec-tools.org"]

    def confForward(self):
        servers = ""

        for f in self.forward:
            for n in f.nameservers:
                if(n.a and n.aaaa and n.nsec):
                    servers = servers + " " + str(n.address)
        if servers:
            subprocess.check_output("unbound-control forward " + servers, shell=True)
            subprocess.check_output("unbound-control flush_zone .", shell=True)

    def checkNameServers(self):
        resolveThread = list()
        for f in self.forward:
            for n in f.nameservers:
                for i in range(3):
                    resolveThread.append(threading.Thread(target=n.sendQuery,
                                         args=(self.checkSites[random.randint(0,len(self.
                                             checkSites)-1)],
                                         i)))
                    resolveThread[-1].start()

    def moduleCheck(self):
        modules = str(subprocess.check_output("unbound-control get_option module-config", shell=
            True))
        if("interator" not in modules or "validator" not in modules):
            subprocess.check_output("unbound-control set_option module-config: validator
                iterator", shell=True)
        tcpupstream = str(subprocess.check_output("unbound-control get_option tcp-upstream",
            shell=True))
        if("yes" not in tcpupstream):
            subprocess.check_output("unbound-control set_option tcp-upstream: yes", shell=True)
        anchorfile = str(subprocess.check_output("unbound-control get_option auto-trust-anchor-
            file ", shell=True))
        if("/var/lib/unbound/root.key" not in anchorfile):
            subprocess.check_output("unbound-control set_option auto-trust-anchor-file: \"/var/
                lib/unbound/root.key\"",
                                    shell=True)
        subprocess.check_output("unbound-control set_option aggressive-nsec: no", shell=True)

    def retrieveNameServers(self):
        for conn in NetworkManager.NetworkManager.ActiveConnections:
            settings = conn.Connection.GetSettings()
            for dev in conn.Devices:
                if(dev.DeviceType != 13):
                    self.forward.append(device(dev.Interface))
                    if('Nameservers' in dir(dev.Ip4Config)):
                        for ns in dev.Ip4Config.Nameservers:
                            if(ns != "127.0.0.1" and ns != "::1"):
                                self.forward[-1].addNameserver(ns)
                    if('Nameservers' in dir(dev.Ip6Config)):
                        for ns in dev.Ip6Config.Nameservers:
                            if(ns != "127.0.0.1" and ns != "::1"):
                                self.forward[-1].addNameserver(ns)


    def frw(self):
        for f in self.forward:
            print(f.name)
            for n in f.nameservers:
                print(n.address)
                print(n.a)
                print(n.aaaa)
                print(n.nsec)
            print()

    def __init__(self):
        self.retrieveNameServers()

def receive_signal(signum, stasudock):
    subprocess.check_output("echo -e \"#Generated by DNSSEC watcher\nnameserver 127.0.0.1\
        nnameserver ::1\" >
                            /etc/resolv.conf",
                            shell=True)
    u = unboundConf()
    u.checkNameServers()
    u.confForward()
    u.frw()
```

```
def main():
    subprocess.check_output("echo -e \"#Generated by DNSSEC watcher\nnameserver 127.0.0.1\
        nnameserver ::1\" >
                              /etc/resolv.conf",
                              shell=True)
    signal.signal(signal.SIGUSR1, receive_signal)
    u = unboundConf()

    with open('/home/knekuza/devel/Client_side_DNSSEC/pid', 'w') as f:
        f.write(str(os.getpid()))

    u.moduleCheck()
    u.checkNameServers()

    while True:
        u.confForward()
        u.frw()
        time.sleep(5)


if __name__ == '__main__':
    main()
```

Listing B.2: source code for solution using the `resolve_async` function

```
#!/usr/bin/python3
import os, time, signal, subprocess, dnslib, random
import NetworkManager, ipaddress
from unbound import ub_ctx,RR_TYPE_A,RR_CLASS_IN, RR_TYPE_A
from unobund import RR_TYPE_AAAA, RR_TYPE_RRSIG, RR_TYPE_NSEC, RR_TYPE_NSEC3
import ipdb

class nameServer:

    def sendQueries(self, site):
        for md in self.my_data:
            md['site'] = site
        self.status[0], async_id1 = self.ctx.resolve_async(self.my_data[0]['site'],
                                                           self.my_data[0],
                                                           self.call_back,
                                                           RR_TYPE_A,
                                                           RR_CLASS_IN)
        self.status[1], async_id2 = self.ctx.resolve_async(self.my_data[1]['site'],
                                                           self.my_data[1],
                                                           self.call_back,
                                                           RR_TYPE_AAAA,
                                                           RR_CLASS_IN)
        self.status[2], async_id3 = self.ctx.resolve_async("nefunguje."+self.my_data[2]['site'],
                                                           self.my_data[2],
                                                           self.call_back,
                                                           RR_TYPE_A,
                                                           RR_CLASS_IN)

    def call_back(self,my_data,status,result):
        if status == 0 and my_data['done_flag'] == False:
            if(result.secure):
                if(str(my_data['server']) == str(self.address)):
                    my_data['done_flag'] = True
                    if(result.qname == my_data['site'] and result.havedata):
                        #ipdb.set_trace()
                        if(result.qtype == 1): #A record
                            self.a = True
                        if(result.qtype == 28): #AAAA record
                            self.aaaa = True
                    if(result.qname == "nefunguje."+my_data['site']):
                        self.nsec = True

    def process(self):
        return self.ctx.process()

    def setContext(self):
        self.ctx.add_ta_file('/var/lib/unbound/root.key')
        self.ctx.set_fwd(str(self.address))
        self.ctx.set_option("tcp-upstream", "yes")
        self.ctx.set_option("module-config","validator iterator")
        self.ctx.set_option("key-cache-size", "32m")
        self.ctx.set_option("max-udp-size", "8192")
        self.ctx.set_option("do-tcp", "yes")
```

```
        self.ctx.set_option("do-udp", "yes")
        self.ctx.set_async(True)

    def __eq__(self, other):
        if isinstance(other, nameServer):
            return(self.address == other.address)
        else:
            return False

    def __hash__(self):
        return hash(self.address)

    def __init__(self, address):
        self.address = ipaddress.ip_address(address)
        self.a = False
        self.aaaa = False
        self.nsec = False
        self.status = [1,1,1] #chekc if query was sent, 0 for sent
        self.ctx = ub_ctx()
        self.setContext()
        self.my_data = list()

        for i in range(3):
            self.my_data.append({'done_flag':False,'arbitrary':"object", 'server' : address, '
                site': None})

class device:

    nameservers = set()

    def addNameserver(self, nameserver):
        self.nameservers.add(nameServer(nameserver))

    def __init__(self, name):
        self.name = name

class unboundConf:

    forward = []
    checkSites = ["isc.org",
                  "nic.cz",
                  "dnssec-deployment.org",
                  "internetsociety.org",
                  "dnssec-tools.org"]

    def confForward(self):
        subprocess.check_output("echo -e \"#Generated by DNSSEC watcher\nnameserver 127.0.0.1\
            nnameserver ::1\" >
                                /etc/resolv.conf",
                                shell=True)
        servers = ""

        for f in self.forward:
            for n in f.nameservers:
                if(n.a and n.aaaa and n.nsec):
                    servers = servers + " " + str(n.address)
        if servers:
            subprocess.check_output("unbound-control forward " + servers, shell=True)
            subprocess.check_output("unbound-control flush_zone .", shell=True)

    def checkNameServers(self):
        siteIndex = random.randint(0,len(self.checkSites)-1)
        for f in self.forward:
            for n in f.nameservers:
                n.sendQueries(self.checkSites[siteIndex])
                time.sleep(0.5)

    def moduleCheck(self):
        modules = str(subprocess.check_output("unbound-control get_option module-config", shell=
            True))
        if("interator" not in modules or "validator" not in modules):
            subprocess.check_output("unbound-control set_option module-config: validator
                iterator", shell=True)
        tcpupstream = str(subprocess.check_output("unbound-control get_option tcp-upstream",
            shell=True))
        if("yes" not in tcpupstream):
            subprocess.check_output("unbound-control set_option tcp-upstream: yes", shell=True)

    def retrieveNameServers(self):
        for conn in NetworkManager.NetworkManager.ActiveConnections:
```

```
            settings = conn.Connection.GetSettings()
            for dev in conn.Devices:
                if(dev.DeviceType != 13): #filterout bridge interface
                    self.forward.append(device(dev.Interface))
                    if('Nameservers' in dir(dev.Ip4Config)):
                        for ns in dev.Ip4Config.Nameservers:
                            if(ns != "127.0.0.1" and ns != "::1"):
                                self.forward[-1].addNameserver(ns)
                    if('Nameservers' in dir(dev.Ip6Config)):
                        for ns in dev.Ip6Config.Nameservers:
                            if(ns != "127.0.0.1" and ns != "::1"):
                                self.forward[-1].addNameserver(ns)

    def frw(self):
        for f in self.forward:
            print(f.name)
            for n in f.nameservers:
                print(n.address)
                print(n.a)
                print(n.aaaa)
                print(n.nsec)
            print()

    def __init__(self):
        self.retrieveNameServers()

def receive_signal(signum, stasudock):

    u = unboundConf()
    u.checkNameServers()
    waitResponse(u)
    u.confForward()
    u.frw()

def waitResponse(c):

    counter = 0
    persist = True
    while(persist and (counter <= 10)):
        for f in c.forward:
            for ns in f.nameservers:
                for i in range(3):
                    if ns.status[i] == 0 and ns.my_data[i]['done_flag'] == False:
                        ns.status[i] = ns.process()
                for md in ns.my_data:
                    if not md['done_flag']:
                        persist = False
        counter=counter+1
        time.sleep(0.1)

def main():
    subprocess.check_output("echo -e \"#Generated by DNSSEC watcher\nnameserver 127.0.0.1\
        nnameserver ::1\" >
                             /etc/resolv.conf",
                             shell=True)
    signal.signal(signal.SIGUSR1, receive_signal)
    u = unboundConf()

    with open('/home/testuser/Documents/NetworkManager/pid', 'w') as f:
        f.write(str(os.getpid()))

    u.moduleCheck()
    u.checkNameServers()
    waitResponse(u)
    u.confForward()
    u.frw()

    while True:
        time.sleep(3)


if __name__ == '__main__':
    main()
```