FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Architecture for monitoring CDNSKEY records from multiple sites for automated DNSSEC management in FRED system
Student:	Bc. Marina Shchavleva
Supervisor:	Ing. Jiří Šádek
Study Programme:	Informatics
Study Branch:	Computer Security
Department:	Department of Information Security
Validity:	Until the end of summer semester 2020/21

Instructions

FRED is open-source software for running a domain and ENUM Registry, developed by CZ.NIC. Automated DNSSEC management is implementation of two RFCs (RFC 7344 and RFC 8078) in the FRED system. Main goal is to make initial trust setup for insecure domains more secure through monitoring records from multiple vantage points in the network and enhance system robustness against network or nameserver failures.
Become familiar with domain name system (DNS) and its security extension DNSSEC in general and also in relation to FRED system. Study IETF documents related to automated DNSSEC management from parent zone (RFC 7344, RFC 8078). Explore current implementation in FRED system. Redesign architecture to enable monitoring of CDNSKEY records from multiple sites (design and implement scheduler algorithm for multi-site scanning of CDNSKEY records, design and implement algorithm for result analysis with focus on recovery from record retrieval failures, implement diagnostic interfaces).

References

Will be provided by the supervisor.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague November 27, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Architecture for monitoring CDNSKEY records from multiple sites for automated DNSSEC management in FRED system

Bc. Marina Shchavleva

Department of Information Security
Supervisor: Ing. Jiří Šádek

May 28, 2020

Acknowledgements

I would like to express my gratitude to my supervisor Ing. Jiří Šádek for his valuable advises and help, and CZ.NIC z. s. p. o. for giving me the opportunity to work on this project. I would also like to thank my family for their support and understanding. Special thanks go to my partner Ladislav for his patience and care.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 28, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Shchavleva, Marina. *Architecture for monitoring CDNSKEY records from multiple sites for automated DNSSEC management in FRED system.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Systém doménových jmen (zkráceně DNS) je kritickou infrastrukturou, a její zabezpečení je důležitý úkol. Existuje hodně různých útoku na DNS, například podvrh paketů a cache poisoning. Zabránit těmto útokům má bezpečnostní rozšíření DNSSEC, které poskytují autentizaci původu dat a integritu dat. DNSSEC používá model řetězce důvěry a asymetrickou kryptografii pro podepsání záznamů v odpovědích od nameserveru. Problematické místo tohoto přístupu je složitý proces změny klíčů, protože majitel domény musí kontaktovat registr aby zaktualizoval kotvu důvěry na straně rodiče přes kanál mimo DNS. Byl vytvořen nový mechanizmus oznámení o změně klíčů na straně potomka. Pomocí tohoto mechanizmu majitel domény publikuje `CDS`/`CDNSKEY` záznamy na autoritativních serverech pro doménu, aby potom automatický monitorovací nástroj registru mohl zpracovat tyto záznamy a umístit výsledek do své zóny. Slabina tohoto systému je v tom, jak se mají zavádět do DNSSEC ještě nezabezpečené domény. Jedna z několika metod, které se k tomu používají, spočívá v monitorování `CDS`/`CDNSKEY` záznamu během nějaké doby z několika míst. Nástroj FRED-AKM vyvinutý CZ.NIC z. s. p. o. implementuje tuto strategii, používá akceptační dobu pro zavedení nezabezpečených domén. Cíl této práce je rozšíření funkcionality FRED-AKM tak, aby podporoval skenování z více lokalit. Přínosem této práce je lepší zabezpečení procesu zavádění DNSSEC pro nezabezpečené domény.

Klíčová slova DNS, DNSSEC, CDNSKEY, automatická správa DNS keysetů

Abstract

Domain Name System (DNS) is a critical internet infrastructure, and securing it is an important task. There are multiple attacks against DNS, such as packet forgery and cache poisoning. In order to mitigate those attacks, Domain Name System Security Extensions (DNSSEC) were added, that provide data origin authentication and data integrity. It uses chain of trust model and asymmetric cryptography to sign records in response messages from nameserver. The problem of this approach is complicated key rollover procedure, since domain owner should request registry to update domain's trust anchor at parent's site through some out of bound method. New mechanism of notifying registry about new keys through DNS was conceived. With that mechanism, domain owner could publish `CDS/CDNSKEY` records at domain's authoritative servers, so automatic monitoring tool at registry could process those records and place them into it's zone. Weak point of this automation is setting up DNSSEC for a domain that is not yet secured. One method of several that are used for that suggests continuous monitoring of `CDS/CDNSKEY` records for some time from multiple vantage points. FRED-AKM developed by CZ.NIC, z. s. p. o. implements strategy, with acceptance period for insecure domains. The aim of this work is to augment FRED-AKM so it will perform scans from multiple locations. This work will contribute to overall security of the process of setting up DNSSEC for insecure domains.

Keywords DNS, DNSSEC, CDNSKEY, automated DNS keyset management

Contents

List of Figures

List of Tables

Introduction

Domain Name System (DNS) is a hierarchical system for assigning names to physical entities. This thesis discusses DNS operations in more detail, describes concepts such as namespaces, nameservers and resolvers, zones and resource records, authority and delegation. It also describes administrative concepts, such as Registry-Registrar-Registrant model. DNS is a critical infrastructure and it is necessary to pay close attention to it's security. Different aspects of DNS security are discussed, and issues that DNS has. Thesis also points toward solutions to those issues.

One such solution is DNS Security Extensions, or DNSSEC for short. DNSSEC adds to DNS authentication of data origin and authenticated denial of existence through use of asymmetric cryptography. It specifically doesn't provide confidentiality. Conceptual model of chain of trust is described, as well as what DNSSEC adds to DNS from operational perspective, such as new resource records and need to implement new validating software. Thesis takes a closer look at what issues are solved by DNSSEC, what alternatives are considered and also what criticisms DNSSEC faces.

At the core DNSSEC is complex to deploy, as periodic key updates need to be done not only on a authoritative nameserver for a domain, but also at parent's nameserver. It means, that domain owner needs to contact owner of parental domain in order to update keys. Many domain owners do not finish procedure of key update, and so a lot of domains are not validated through a complete chain of trust.

In order to ease key update process, automated solution was developed. Now domain owner needs only to publish at their nameserver special record, that tells parental domain owner that child wishes to update keys. Parent now only needs to run periodic monitors of children's nameservers in order to know about those records. The biggest problem posed here is how to introduce keys for domains, that are not yet secured by DNSSEC. Multiple strategies exist, one of those is "Accept after Delay", that recommends multiple scans of those records from one or more locations in order to ensure, that records

are unchanged over time, and then it is possible to include them into parent's zone. Some software exists that does that, but this automation is not yet widely deployed by owners of Top Level Domains.

At the Top Level Domain `.CZ` update of keys at parental site is automated by a tool called FRED-AKM. This tool is closely coupled with FRED system for administering a Registry. It performs daily scan of children's nameservers in order to find new keys or update existing ones. This tool is also able to notify technical contact for domain that keys are discovered and domain is in the process of setting up DNSSEC or that keys are updated immediately if domain is already secured. The scan is performed from one location.

The main goal of this thesis is to augment FRED-AKM so it is able to scan from multiple locations, thus achieving greater security upon introducing new keys for insecure domains. Secondary goals are decoupling AKM from FRED, and from unrelated tasks such as notification. This project implements two types of interfaces: operational (related to scan itself) and diagnostic. It also redesigns scheduling, as current implementation is now scheduled through other means rather than from the inside of an application. I consider possibilities for recovery from scan result retrieval failures.

The thesis is organised in the following manner: Chapter 1 introduces reader to the Domain Name System, it's intricacies and security issues, following with description of DNSSEC, what it does and doesn't solve and what problems deployment of DNSSEC poses, how is DNSSEC could be automated and finally, software solutions that implement this automation, and focus my attention of FRED-AKM. Chapter 2 outlines design of a developed solution called AKM Multi Scanner, what overall architecture of a system is and what is role of each part of a system. Chapter 3 describes implementation of the AKM Multi Scanner, what technology was used and what challenges I encountered.

Chapter 1

The Domain Name System and security

This chapter will introduce reader to Domain Name System, it's general principles, and will shortly discuss administration of DNS at a larger scale. Security issues inherent to the DNS design will be presented, as well as attempts to mitigate them. One of those is relevant to this work: DNSSEC.bis, commonly known as just DNSSEC. I will examine details of DNSSEC, what does it solve, and what problems DNSSEC has. Major problem of DNSSEC is that it is notoriously hard to deploy, and what was done to lessen this complexity.

1.1 Main concepts behind Domain Name System

The Domain Name System was created to tackle the problem of assigning names to physical entities in the network to ease the access to them for humans. Overall aim of the system is to store, serve and administer attributes of a named resource. As Ron Aitchison pointed out, "the problem of converting names to physical addresses is as old as computer networking"[1], and RFCs, that define DNS, date back to 1987 and remain largely intact, but updated by newer RFCs.

The core ideas and concepts behind DNS are presented in RFC 1034, and RFC 1035 digs more into details of formats and implementation. This section relies heavily on those RFCs, but sometimes the language there could be quite confusing and complicated, so at times this text would rely rather at explanation of basics that could be found in [1].

RFC 1034 defines main components of DNS:

- *Domain Name Space* is thought of as a tree structured namespace, where each node is labeled and contains a set of information associated with it, which are called *Resource Records*

- *Nameservers* are units, that contain and serve information about domain name space; nameserver is authority for a part of a namespace, when it has complete information
- *Resolvers* are programs, which extract information from nameservers in response to client requests[2].

1.1.1 Namespace

DNS uses hierarchical structure of a tree, each node has a label. What we usually call "domain name" is concatenation of names from some node in the tree to the root divided by dots, for example `fit.cvut.cz.`. One could notice dot at the end of this domain name, it happens because of the root label which is an empty string; when domain name is written like this it is called "Fully Qualified Domain Name" or FQDN for short. Organization, that is responsible for management and operation of the node is called *authority* for this part of namespace. Authority of some part of a namespace could be delegated to some other authority, which allows finer management.

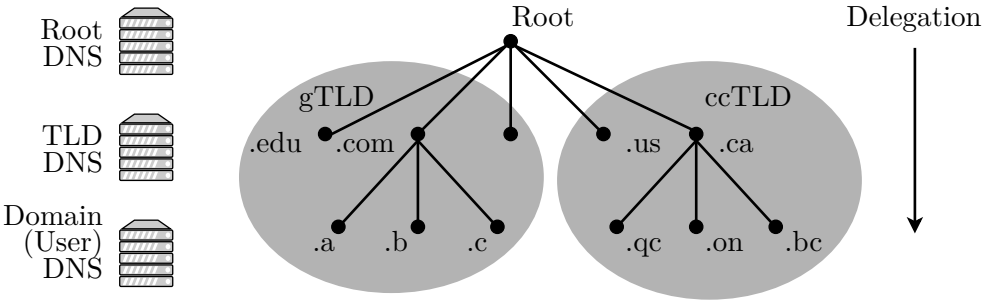

Figure 1.1: Hierarchical structure of DNS. It also shows direction of delegation of authority[1].

After root label there are another significant labels called Top Level Domains or TLDs. Figure 1.1 shows two major groups of TLDs: generic TLDs (gTLD) and country code TLDs (ccTLD). Authority for the root domains lies with ICANN, which in turn delegates ccTLDs to local authorities. For example, authority for `.CZ` is CZ.NIC, z. s. p. o (to which this text refers as just CZ.NIC).

1.1.2 Nameservers

DNS acts as a database of domain names, pieces of that database scattered conceptually in a tree and practically throughout nameservers. Those pieces are called "zones", and nameserver could be authoritative for multiple of them. Every zone also has multiple authoritative nameservers (RFC 1034 recommends a minimum of two[2]); such redundancy improves availability of

the whole system. One of those nameservers is primary (it holds definitive information about a zone), and other ones are secondary and zone data is distributed to those from primary through a mechanism of a zone transfer. Another type of nameserver is caching nameserver: it holds data about a zone for a defined period of time and is not authoritative to this zone. Caching nameservers exist to reduce load from authoritative nameservers, and generally they improve performance of a lookup. Nameserver's job is to respond to queries.

Zone is distinct part of domain namespace that was delegated to a single authority. Zone includes all leaf nodes of a domain, except for those that are delegated. Zone is described by a "zone file", which contains vital information about a zone, such as info about it's nameservers, mailservers, physical addresses and subzones. Zone file also defines the way it will be processed, such as "Time to Live" directives and expiration time for secondary nameservers.

Updating a zone manually – editing zone file and then restarting server – could damage availability if server contains large amount of zone files or change in a zone file is massive. The mechanism of "dynamic update" was introduced in RFC 2136, where remote admin could edit zone files and updates are propagated immediately[3].

1.1.3 Resource records

Resource record is a unit of information about a name, which defines a forward mapping of hosts (domain name to address). There is a multitude of different RRs, the most important are:

- `SOA` **Start of Authority** defines key features of a zone, like it's primary nameserver and secondary nameserver update parameters
- `NS` **Nameserver** records list authoritative nameservers for a zone
- `A/AAAA` **Address** is IPv4 (or IPv6 respectively) address of a host[2]
- DNSSEC relevant RRs will be listed at section that described DNSSEC.

When requester queries the nameserver, resource records is what is expected as an answer. DNS protocol and other details are described in RFC 1035. Both request and response use the same message format, described at table 1.1.

Resource records correspond to particular name in the zone. It is possible to omit the last dot in FQDN (which is done commonly in web browsers, for example), but in the context of DNS those names are considered local: in a zone `cvut.cz` there could be record for `fit.cvut.cz.` and `fit` – both relate to the same entity.

Header	Specifies which of remaining sections are present, for example if it is a query or a response, if response is authoritative, or if an error occurred.
Question	Carries information about what is requested.
Answer	RRs that answer the question.
Authority	RRs that point towards authority if nameserver cannot answer the question.
Additional	RRs that hold additional information, data that is relevant to original query.

Table 1.1: Top level description of DNS message format[4].

1.1.4 Resolvers

Resolver is a program, that performs domain name resolution through requesting nameservers. Resolution of a domain name is recursive: resolver asks root nameserver if it knows this domain name. Root nameserver most probably doesn't have info about asked domain name, and points to TLD authoritative nameserver. Resolver asks this nameserver, and it in turn receives pointer to authoritative nameserver for second-level domain name, and so on further down the tree until finally some nameserver is able to response with a definitive answer.

As this tree traversal is rather lengthy and a lot of users visit similar sites, it is useful to have some resolver in the network. Resolver would cache query results and then answer fast to known questions, which largely improves resolution speed and reduces network load in general. User now has to have lightweight resolver on their machine called "stub resolver" and list of recursive resolvers on the network.

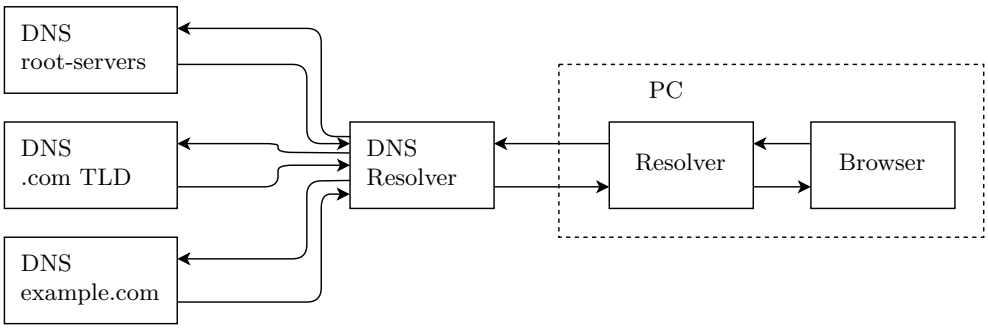

Figure 1.2: Resolution of a domain name.[1]

1.1.5 Domain registration process

There are multiple models how one could register domain name under some TLD, either generic or country code, for example through direct communication with authority. This authority is called *Registry*, and individual or organization, that wishes to register a domain is *Registrant*. Another way would be that in order to register a domain one doesn't directly interact with entity, that administers TLD's zone, but with entity, to which authority on registration process is delegated, which is called *Registrar*. Upon registering a domain name, Registrant enters into legal relationship with Registrar, under which both parties have rights and responsibilities. Registrant then manages domain settings through Registrar. Registrar transmits data about registered domains to Registry, which propagates those changes to zone file and, ultimately, to the Internet[5]. Strictly speaking, Registry acts as a database for domain names. This model is called Registry-Registrar-Registrant.

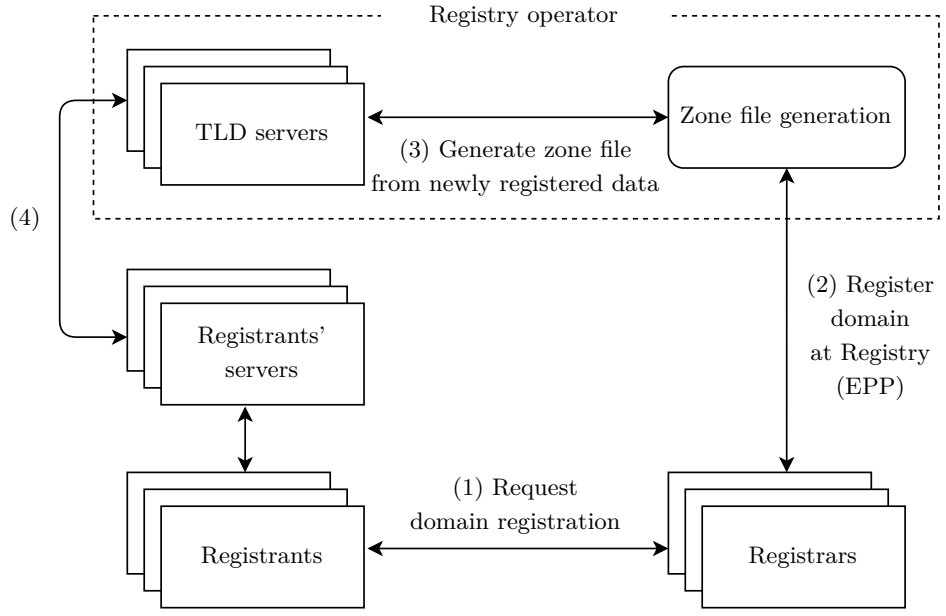

Figure 1.3: Registry-Registrar-Registrant relationship. Image based on [1, p. 14], with my extension of registrant.

There might be some other parties in this scheme, for example, domain managers. Domain owner (Registrant) outsources management of a domain name to an organization. It is important to note, that domain managers are not Registrars: in a situation, when domain owner needs to update it's registration information, it has to contact it's Registrar. Domain management

services provide organizations such as Cloudflare, DNSimple and myriad of others.

1.1.6 Common DNS server software

There is a number of DNS server implementations, and it might be hard to talk about DNS and about real life usage without mentioning software. Most notable server implementations include:

- BIND (Berkeley Internet Name Domain)[6] is an open source full featured DNS system and is one of the most widely used DNS software[1]
- Knot DNS[7] and Knot Resolver[8] are authoritative server and caching DNS resolver respectively, are developed by CZ.NIC; Knot DNS is used as authoritative DNS server for `.CZ` domain[9] and also for some of the root servers[10]
- others, such as NSD, tinydns, djbdns, PowerDNS and Unbound.

1.2 DNS security overview

DNS is crucial infrastructure for enterprises and for the Internet as a whole, it naturally follows that it might attract a lot of malicious attention. DNS might be a primary target for an attack as well as just a mean to an end.

Performing a DDoS attack on business' nameservers, or on DNS provider (for example [11]), or even on a root servers (the alleged plan "Operation Global Blackout"[12]) might make domains that are dependent on nameservers under attack seem unreachable, which could significantly hurt the company, or might be a simple show of force. In [13] reader can find a brief overview of history of security breaches involving DNS. DDoS attacks in the context of DNS do not end just there: DNS uses UDP and because there is no connection established, it is easy to forge source IP address of DNS query to IP address of a victim.

DNS packets could be used in message tunneling. Malware could communicate with Command and Control center through subdomain names. Attacker could register some domain, and communicate through subdomain names, which could be binary data encoded by Base32, or through `TXT` records. Some work has been done in an effort to detect such communication, reader could refer to paper[14], which lists solutions to this problem and proposes it's own.

1.2.1 Analysis of DNS structure in security context

For attacks on DNS itself, one should first consider the anatomy of DNS. DNS is a complicated system, and [1] classifies following levels from closest to the source of data to the furthest, where one must consider security of the system:

1. **Administrative security**: through error or malicious actions zone file might be corrupted; or any administration mischief not specific to DNS protocol such as misconfigured ACLs, weak or all too eager firewall, social engineering of registrar's technical support or even missed bill from a registrar[13]
2. **Dynamic updates**: it is possible to perform unauthorized updates through lack of security measures
3. **Zone transfers**: zone updates could be forged to invalidate records in the zone, and even to insert malicious data. Interception of a zone transfer could even serve as reconnaissance source to attacker. Default mechanism of zone transfer from Primary server to Secondary might be weaker from a security standpoint than other methods of zone file distribution, one might even consider running multiple Primary servers rather than one Primary and multiple Secondaries
4. **Zone integrity**: integrity of the zone on the way from authoritative nameserver to resolver and eventually, to the end user. For example, cached queries might get poisoned through data interception or even compromised nameserver, thus breaking the integrity.

1.2.2 Unauthenticated responses

One major problem at "Zone integrity" level is that there is no way to authenticate origin of query response other than source IP address. It means that virtually anyone could redirect end user to attacker controlled site and user might remain oblivious to the fact. Here I will describe techniques used by malicious actors that abuse this lack of authentication of server.

Response forgery

Requests are issued by resolvers. Request packet is identified by four things: 16 bit transaction ID, 16 bit source port, source IP address and the query itself. Response is paired to those parameters, when those do not match, resolver could reject it. Typically, when attacker is setting up the attack, they've already picked target domain and a victim (more on that later). This leaves only two variables: transaction ID and source port. In the early days, many resolvers utilized same source ports for queries, even today BIND still has a setting that allows hard-coded source port. This leaves only 16 bit of transaction ID, and according to birthday paradox, only around 700 packets with randomly picked transaction IDs is needed to match some request with 99% probability. This is called "blind forgery", and under some circumstances, attacker could just eavesdrop on DNS requests and reply with knowledge of all of those variables. There are efforts to increase size of transaction ID, but it still doesn't protect against eavesdropping[15].

Cache poisoning

Through response forgery and other means, it is possible to poison the cache of a resolver. Resolver saves already resolved records in cache for a limited amount of time. If someone manages to insert false data into cache without resolver actually resolving this name, and resolver then would server it's users false data.

In DNS terms, "in-bailiwick" data means "Data for which the server is either authoritative, or else authoritative for an ancestor of the owner name"[16]. Example from relevant RFC: "the server for the parent zone `example.com` might reply with glue records for `ns.child.example.com`. Because the `child.example.com` zone is a descendant of the `example.com` zone, the glue records are in-bailiwick"[16]. "Bailiwick rule" states, that resolver would not accept responses from authoritative nameserver for, for example, `example.com` that contain `google.com`. Without it, rogue server could insert in Additional section of a response for, say, `seznam.cz` address of `csas.cz`, which points at a attacker-controlled host.

Cache poisoning could happen in the following way:

1. Attacker issues a request for `www.example.com`.
2. Resolver, if it doesn't already have cached entry for `www.example.com`, sends requests to authoritative nameservers down the hierarchy.
3. Attacker at the same time issues responses for `www.example.com` with forged transaction ID.
4. Now it's a time race: which response will reach resolver first.

This severely limits possibility of forgery, since an attacker has one chance a day (a common Time to Live for a record).

Dan Kaminsky in 2008 exposed a bug (commonly known as "Kaminsky bug") which is capable to overcome the "bailiwick rule"[17]. The exploit works very similar to what was discussed earlier, but now attacker issues requests to some nonexistent sister domain like `sdfh34b.example.com`. Since this domain is almost certainly not in the cache of the resolver, it will initiate requests. Attacker requests more of those nonexistent domains thus elevating their chances of hitting the target.

Interesting cache poisoning technique is mentioned in RFC 3833, called "name chaining", where the answer includes domain name (`CNAME` and `NS` for example). Such name could lead to an attacker-controlled domain[18].

Another possible way to poison the cache of a victim, is targeting specific workstations. It could be done by malware so it can redirect victim to attacker-controlled sites, as well as a mean of protection, redirecting security related websites to localhost[13]. Or, if the attacker is on the same network as a victim, it could eavesdrop on the requests and serve malicious responses.

1.3 Securing DNS

With DNS being such a critical infrastructure, there is a lot of effort put to secure it. Every solution aims at some part of the system, each has it's strong points as well as weak ones. Figure 1.4 overviews how solutions apply to the parts of a system.

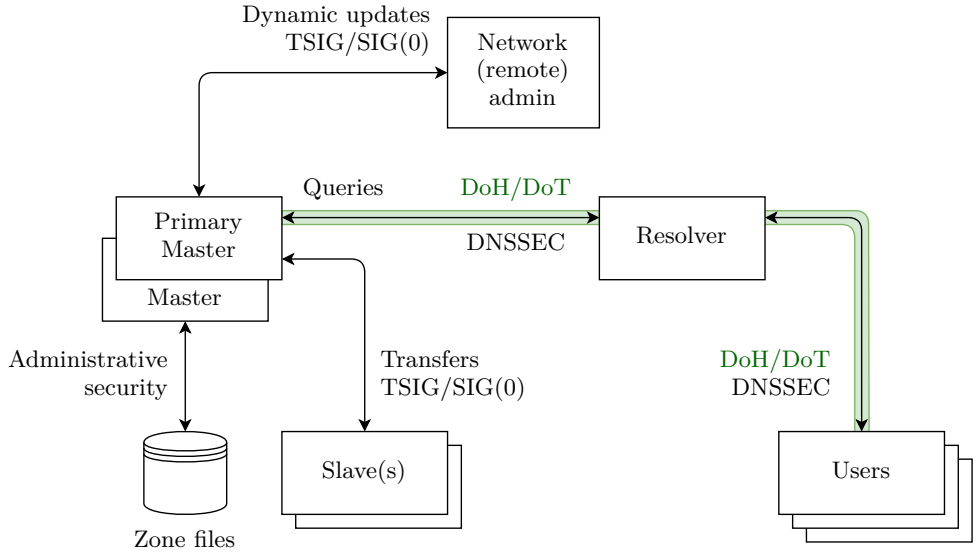

Figure 1.4: Overview of security solutions in DNS. Image based on [1, p. 60], with my extension of users and showing exactly where does each solution belong.

1.3.1 Administrative security

Going from the level closest to the source of zone data to the farthest, complexity grows. The easiest one to solve is administrative security: Ron Aitchison in [1] suggests proper software updates, detailed attention to ACL and other measures.

1.3.2 Zone transfers and dynamic updates

Defined in RFC 2845, mechanism of Transaction Signature (TSIG) allows authentication of messages between two DNS entities. Message Authentication code is derived from a shared secret, which in turn is set through out of band distribution, or it could be established automatically through `TKEY` RR and Diffie-Hellman key exchange. Similar technology is SIG(0), which uses asymmetric cryptography to sign their messages. SIG(0) is more scalable because it is easier to share public keys, but TSIG is considered to be less

expensive in the terms of server load[19]. TSIG is designed to be used for dynamic updates, and it's design makes it harder to use in the context of transactions between nameserver and resolvers (resolver has to be trusted). SIG(0) doesn't particularly solve this either, as resolver has to trust the public key of a nameserver through some other means than DNS.

1.3.3 Confidentiality

Plain DNS protocol transfers data in the plain text, so anyone could eavesdrop on the traffic. This raises some privacy concerns, as stated in RFC 7626. There are several proposals, that solve this problem.

Two such proposals are "DNS over TLS" defined in RFC 7858 and "DNS over HTTPS" from RFC 8484. Those are similar in a sense, that both use TLS as a carrier protocol, but differ in the details. DNS over TLS (called "DoT" for short) is essentially original DNS, which uses TLS connections over TCP. It proposes usage of a separate port for communication[20]. DNS over HTTPS (DoH) uses HTTP servers and clients for communication, which results in using same port as HTTPS and "allowing web applications to access DNS information via existing browser APIs in a safe way consistent with Cross Origin Resource Sharing (CORS)"[21]. Some major companies are providing public DNS resolvers that support one or both those solutions[22][23].

Another noteworthy attempt at improving confidentiality (and availability with integrity, according to the project's page[24]) is DNSCurve. To put it simply, DNSCurve is TLS for DNS, that uses elliptic curve based cryptography and tunnels encrypted data over `TXT` records. Key are stored in NS records using base32 encoding, and session key is derived using elliptic curve Diffie-Hellman key agreement scheme. Similar protocol, DNSCrypt, is essentially DNSCurve, with minor differences, including parties, which communicate through those protocols: DNSCurve for communication between authoritative servers and resolvers, DNSCrypt is between resolvers and end users[25].

1.3.4 Zone integrity

This work's focus is on the last – "Zone integrity" – level of security issues involving DNS. I will only mention, that security of dynamic updates and zone transfers, also administrative security, is covered by [1] and [13] in more length.

Major effort in securing the nameserver-resolver communication is DNSSEC. Next section is paying more attention to it, reader will also find there comparison of DNSSEC and other technologies, that are partially aiming in the same direction.

1.4 DNSSEC

According to RFC 4033, aim of DNSSEC is to add data origin authentication and data integrity to the DNS, including mechanisms for authenticated denial of existence[26]. It does not aim at confidentiality of the data, which means that DNS queries and responses are still sent in clear.

1.4.1 Main principles

DNSSEC uses asymmetric cryptography to authenticate data origin. DNSSEC establishes chain of trust in DNS tree, and, in a way, is similar to Public Key Infrastructure. The difference is that in PKI signature, created by Certificate Authority is part of a certificate, and in DNSSEC infrastructure, signature of a zone's public key is located at the site of the "authority" (in the context of DNS, at the parent zone). As I see it, this is done for consistency sake, as public keys are part of DNS infrastructure itself, thus they have to be distributed through DNS protocol. Every zone signs it's own data, and having a RR that is signed by some other entity breaks this consistency. As of 2010, root zone is signed[27].

Zone is considered signed when it contains signed RRsets. RRset is a set of RRs with the same type. RRset is signed by private key of a zone, and at request, is sent along with a signature and a public key. In the iterative nature of DNS requests, resolver already has information from parent about this public key and can validate it against those data. With public key of a zone that is verified to be legit, resolver could proceed to verify the RRset against signature.

1.4.2 Keys

Authenticity of a key is ensured by the data about this key, stored at the parental zone's site. Through query of parental nameserver (which should be done anyway) resolver could pair this data with a key in the child zone and by this authenticate the child. Original RFCs about DNSSEC do not define a way how data about a key is transferred to parent. It means it should be done out of band, which makes key rollover quite complex procedure (more on this later in section1.5).

To partially address this problem, the protocol introduces two set of keys, though they could be the same:

- **Zone Signing Key (ZSK)** is used to sign RRsets
- **Key Signing Key (KSK)** is key that signs other keys, in this context it signs ZSK.

Using this division, only KSK has to be mentioned at the parent, and ZSKs are still secure to use. With ZSK signed by KSK administrator of a zone could

rollover ZSKs more often, increasing the security of signing process. Typically ZSK and KSK have different operational parameters such as validity time. As KSK is another RR in the zone, ZSK also signs KSK.

1.4.3 Resource Records

DNSSEC introduces new RRs, which are described in RFC 4034:

- `DNSKEY`: public key, which is used as KSK or ZSK; table 1.2 shows structure of `DNSKEY` RR.
- `RRSIG` **RR signature**: digital signature of RRset; it must have the same class as RRset that it signs, it also has validity interval
- `NSEC` **Next Secure**: points to the next record in the zone; ensures authenticated denial of existence
- `DS` **Delegation Signer**: hash of DNSKEY RR; unlike other RRs, resides at the parental zone; table 1.3 shows structure of `DS` RR[28]

Field name	Description
Flags	16 bit field that defines how the key is used, does this key correspond to `DS` record at parent's site, is this ZSK or any other type of key.
Protocol	Holds value 3, otherwise key is invalid.
Algorithm	Identifies cryptographic algorithm of a public key, determines the format of Public Key field
Public Key	The public key material.

Table 1.2: Description of `DNSKEY` RR[28].

Field name	Description
Key Tag	Special value computed from `DNSKEY` it refers to.
Algorithm	Refers to the algorithm of `DNSKEY` it refers to.
Digest Type	Type of digest that is used to compute hash value of `DNSKEY`
Digest	Digest computed from `DNSKEY` owner name and `DNSKEY` data itself

Table 1.3: Description of `DS` RR[28].

The chain of trust is alternating sequence of `DNSKEY` and `DS` records. With `DS` record signed by delegating site, signature vouches for authenticity of `DNSKEY` record at delegated site, allowing this `DNSKEY` to be used in signing.

Lower links in the chain are then trusted through this cumulative process. Figure 1.5 shows relationship between `DS` and `DNSKEY` records and how the chain of trust is established through this. Useful tool for visualisation, that uses real data about domains is DNSViz, available at `https://dnsviz.net`.

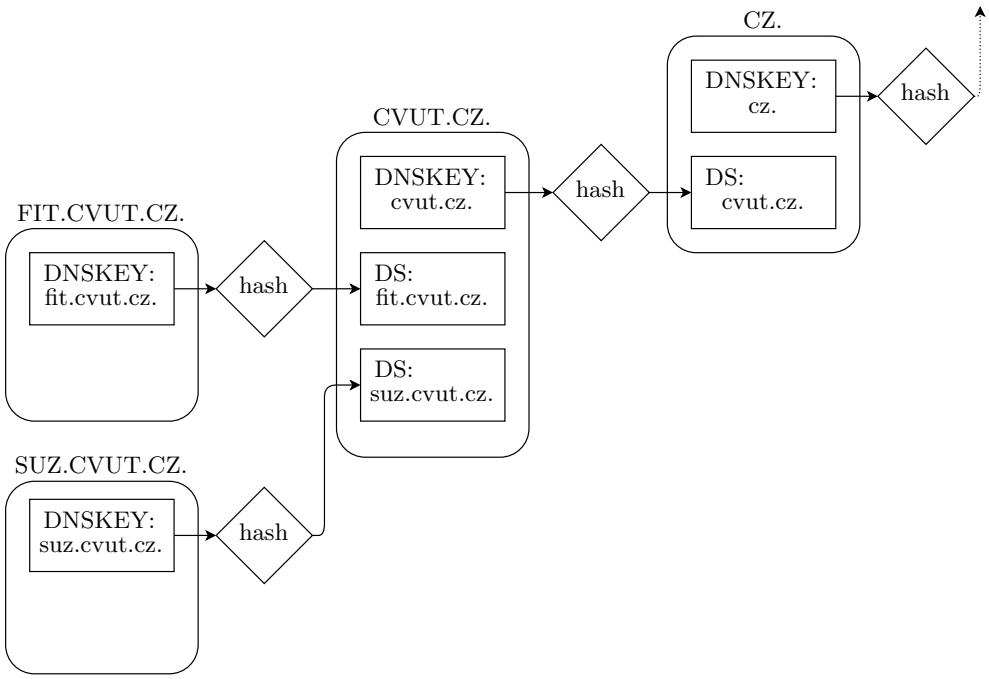

Figure 1.5: Example of a tree structure of DNS in the context of DNSSEC and relationships between DS and DNSKEY records.

In classic DNSSEC, communication about new `DNSKEY` should be done out of band. Most of the time, it is done by manually editing some settings through domain name administration interface at Registrar's site.

1.4.4 DNSSEC evaluation

RFC 3833 takes a work to list issues, that are solved, at least partially, by DNSSEC. Discussed security problems, such as name chaining, packet forgery and transaction ID guessing, are solved by signing the records. But this document also raises concern about some possible weaknesses of DNSSEC[18].

Security issues of DNS Security Extensions

DNSSEC bloats response size, thus DNSSEC-aware servers are a prime target for amplification. This is not problem of DNSSEC, but an inherited one from DNS itself.

DNS is essentially public service, but there are necessary distinctions to be made: first, requester has to know exactly what name it needs to resolve (i.e. nameserver doesn't publish a list of available names), second, even though domain name is public, the fact that somebody visits it shouldn't be (privacy)[29]. Second concern is explicitly out of scope of DNSSEC.

With `NSEC` it is possible to enumerate entire zone. There are solutions (`NSEC3` and `NSEC5`) that through hashing try to eliminate this leakage.

Issue that is relevant to DNSSEC is Shambles attack: in short, it is a chosen-prefix collision on SHA-1 hash function. SHA-1 is long known to be an insecure hash function and is deprecated, but is still used in DNSSEC. Article at APNIC describes hypothetical attack scenario, that utilizes the fact that some records could still be signed with algorithm that uses SHA-1[30].

Criticism

DNSSEC initially faced a lot of criticism, mainly because of a complexity it adds. Offline key rollover (manual key generation and zone update) is complex and it is an obstacle on a way to adopting DNSSEC. Now, one could configure BIND[31] and Knot DNS[32] to automate keyset rollovers. Key rollover at root is even harder: KSK was introduced into root in 2010 and first rollover happened in 2018, and it was a complex procedure[33].

Another criticism that DNSSEC faces is that it increases traffic. NIST mentions, that practical size for `DNSKEY` is 1500 bytes or less[34, p. 51]. Previous subsection mentioned increased response size due to inclusion of signatures, which makes DNSSEC-aware server a perfect target for amplification. Even for legitimate uses, to fully validate a domain, two additional queries has to be done for each initial query: `DS` and `DNSKEY` records. This problem is addressed by RFC 6605, which introduces Elliptic Curve Digital Signature Algorithm to DNSSEC[35]. According to NIST, 256 bit key of ECC is comparable in complexity to 3072 key of, for example, RSA[36, p. 54-55], which allows to drastically reduce size of responses and keep the security.

Previously I described domain name resolution process, which is done by two parties: lightweight stub resolver, that requests resolution, and recursive resolver, that performs majority of the work. It might happen, that recursive resolver is not DNSSEC-aware, which means that stub resolver has to fallback to performing recursive resolution itself in order to validate the domain. This might be problematic, since stub resolver could reside at weak hardware, which in turn could incapacitate the operation of this hardware[37].

DNSSEC is evolving: it started in 1993[18] and since then more than 50 RFCs relate to it, updating and obsoleting older ones. Some issues are already addressed, some are yet to be solved, and work is still in progress [1].

[1] visit `https://datatracker.ietf.org/doc/stats/newrevisiondocevent?name=dnssec&sort=&rfcs=on&activedrafts=on&by=group&group=` to see that work on DNSSEC is pretty much consistent effort from community

1.4.5 Comparison of DNSSEC and other solutions

Obvious question, that arises when reader encounter DNSSEC is: if it's that hard to deploy, why don't we use something else? Some of those solutions discussed earlier, provide confidentiality – does it substitute for DNSSEC? Here I will compare it to other solutions that I noticed are quite often compared to DNSSEC, namely DNS over TLS, DNS over HTTPS and DNSCurve. There is no need to mention one obvious difference between those solutions and DNSSEC (confidentiality thus better privacy), so I do not have to mention it multiple times.

DNS over TLS/HTTPS

For the purpose of this comparison, DNS over TLS and DNS over HTTPS are very similar, so I will omit DoH and mention only DoT, but reader could apply the same reasoning to DoH.

Mainly, DNSSEC is transitive, which means that any user could query for public keys and signatures for RRsets and verify the correctness of name resolution. Contrary to that, DoH provides only end-to-end protection: user might query the resolver over TLS and be sure, that data between them are protected both from confidentiality and integrity standpoint, but user can not be certain, that resolver and authoritative nameservers communicate safely. Analogous issue is if resolver betrays the user: without DNSSEC user cannot really verify authenticity of the data[38].

Another significant difference is that DNS uses primarily UDP (it also can use TCP), and TLS uses TCP. In a context of DNS, establishing a connection, especially secure one is quite an overhead.

DNS over HTTPS issues

Interesting aspects, where difference between DoT and DoH comes into play (namely, using the same port as HTTPS and channeling data over it) is the fact, that DoH is much more criticized as a security hazard. Unlike DoT, DoH passes data at the same port as HTTPS traffic, which might make impression that DNS traffic is hidden in HTTPS traffic. Author at APNIC argues, that much of metadata, including Server Name Indication, is leaked through TLS, which renders DNS encryption pointless[39].

Another side of this hiding is that it is now harder to blocklist pages which might seem like a good thing for privacy, but it in fact impairs the ability of system administrator to block malicious (or otherwise unwanted) traffic[40]. DoH might also act as a covert channel[41].

DNSCurve

One crucial difference between DNSCurve and DNSSEC is that DNSCurve does not introduce any new Resource Records, as it uses `NS` for public key posting and `TXT` for encrypted communication. Another one is that DNSCurve uses symmetric cryptography and connection is established through key agreement.

DNSCurve is criticized[42][43], mainly because author positions it as a direct competitor to DNSSEC[44]. Especially now some claims are obsolete, but even at the time of creation some claims were dated (for example, that DNSSEC requires offline signing[42].

Conclusion on comparison

All those solutions do not really provide authentication of data source. As TLS, for example, is essentially out-of-band solution for security, it might so happen that attacker obtained rogue certificate (legitimate certificate issued by trusted certificate authority, but that has been compromised). Moreover, it is possible to issue certificates with invalid Common Name[45], specifically, with locally scoped one, which allows to impersonate organization, for which certificate was issued in the first place. Here complexity of inserting `DS` record at parent might be actually a benefit. This doesn't mean that lack of confidentiality in DNS is not a valid concern, and all of those solutions are complementary to DNSSEC. At this moment, there is no viable alternative to DNSSEC.

1.4.6 DNS-based Authentication of Named Entities

To address some issues stemming from PKI, RFC 6698 proposed solution "DNS-based Authentication of Named Entities", DANE for short[46]. It utilizes the fact, that DNSSEC signs RRsets, thus proving, that DNS operator is responsible for the nameserver. Majority of connections start with DNS lookup of an IP associated with a name, so the idea of DANE is to among with signed IP address of a domain, it is possible to pass signed information about server's certificate. The reason is that it is logical to assume, that if DNS operator is authorized to give information about a zone, they might as well make authoritative binding between a domain name and certificate that host at that domain uses. Essentially, the job of PKI is placed at DNS chain of trust from root to leaves, eliminating large net of Certificate Authorities.

Even though the proposal might be promising, for example in securing mail services[47], it faces a lot of obstacles. One such obstacle is that, according to [48], many TLDs use 1024 bit RSA as ZSK, possibly degrading security in comparison to current certificate practices.

1.5 Automating key update at Parent's site

As mentioned earlier, DNSSEC key rollover and associated with it manual update is a problematic part of a protocol. This makes for a weak point of DNSSEC: any manual interaction is a potential point of failure, or simply an annoyance so Operator will not go through steps needed to propagate key to Parent. DNS is often though of as "set-and-forget" type of infrastructure[13, p. 25] thus any regular (and key rollover *should* be regular) manual activity associated with it could lead to a lot of pain and through a lack of experienced personnel (which happens with "set-and-forget" technology) – to errors.

To address this, RFC 7344 introduces mechanism, how parent could automatically update `DS` records through DNS itself[49]. The main idea is, that parent could update `DS` record on the basis of what is posted at child site.

There are three reasons, why child might want to change `DS` record at parent:

- Child wants to roll over already present keys
- Child wants to enroll into DNSSEC as it doesn't have yet any keys: it wants to introduce a new `DS` RR
- Child wants to turn off DNSSEC validation: it wants to delete `DS` record

1.5.1 Resource Records

For the purpose of indication that the child wishes to update DS record at parental zone, new RRs were introduced:

- **`CDNSKEY` Child `DNSKEY`**
- **`CDS` Child `DS`**

Both are essentially the same as their original counterparts, the only difference is names. It's parent's responsibility to decide, which one of those records will be monitored, parent might as well check both of those.

Difference between `DNSKEY` and `CDNSKEY` is that RFC 8087 defines special algorithm value (in a sense of `DNSKEY` RR field), that is used to delete `DS` record at parent's site[50]. This algorithm value (zero) is still reserved for `DNSKEY`. For the purpose of deletion it is possible to use `CDS` too.

1.5.2 Detection of new keys by parent

Child posted new keys as `CDS`/`CDNSKEY` records. The RFC 7344 proposes two methods of key detection at parent's site:

- **Push**: Child through mechanisms of domain administration at some third party (say, Registrar) notifies parent that new keys are present. Parent then fetches those keys and updates it's zone.
- **Poll**: Parent runs periodic checks child if there are new `CDS/CDNSKEY`.

"Push" strategy minimizes human error by fetching data directly though DNS, yet it doesn't utilize full automation protection that this mechanism offers. The intent of updating keys is already expressed by posting `CDS/CDNSKEY` in the first place. On the other hand, "Poll" strategy is fully automated, but it creates more DNS traffic as periodic checks has to be run.

1.5.3 Enable DNSSEC through `CDS/CDNSKEY`

One major problem of "Poll" approach is that when DNSSEC is not enabled yet by the domain, it might be difficult to check, that `CDS/CDNSKEY` are authentic. RFC 8078 suggests multiple ways, how child could enable DNSSEC through `CDS/CDNSKEY` RRs, which are described in the following text[50]. One must note that monitoring of `CDS/CDNSKEY` records should happen at *all* nameservers of a child, it means that those records should be consistent across nameservers. Parent should make clear what method of acceptance it is using, as well as which RR is relevant for it.

Accept Policy via Authenticated Channel

Essentially, this is a "Push" strategy, where child notifies parent by some authenticated out of band channel, that `CDS/CDNSKEY` exist.

Accept with Extra Checks

This policy is an expansion of the previous one, which addition of extra checks performed by parent. RFC 8078 suggests checks such as if there were no other changes in registration in the last few days, confirmation mails, etc.

Accept after Delay

This strategy could be a variation on "Extra checks" policy, as well as a fully automated procedure. Parent monitors `CDS/CDNSKEY` RRsets at child's nameservers over some period of time to make sure nothing changes. Monitoring could possibly happen from multiple vantage points in the network to minimize threat of interception. After this period expires and `CDS/CDNSKEY` RRsets were consistent through it, parent could update it's `DS` record.

Setting of this acceptance period and frequency of requests is up to the parent. From security standpoint, the longer domain is in the check process, then parent could be more confident about child's intent, but not only this results in child's discontent with the length of the process, it also stops child's

nameservers from securely serving it's data. Another aspect is that network is never faultless, and longer process might result in more false negatives due to network failures, or some other accidents. Frequency is important too, as too frequent requests could overload nameservers even though they are usually designed to endure heavy loads.

Accept with Challenge

In this scenario, parent requests child to insert in it's zone some specific record, that will prove the ability of a child to control the nameservers. This one could not be automated fully, as parent has to communicate securely what record should it be. It add another record which parent has to monitor, and parent can apply "Accept after delay" policy to this record as well.

Accept from Inception

When parent adds new child domain, which contains `CDS/CDNSKEY`, it could add relevant `DS` record along with `NS`. This has the benefit that child domain never enters insecure state in the first place. This, obviously, could not be applied to already existing domains.

1.5.4 Existing solutions

To my knowledge, this approach is not widely adopted by TLDs, perhaps because those RFCs are relatively new. Besides `.CZ` approach, which will be discussed later in section 1.6, the only other ccTLDs that utilize `CDS/CDNSKEY` I found were a TLD for `.CH` and `.LI` (Switzerland and Liechtenstein respectively), which are operated by SWITCH. As described in their guidelines, SWITCH monitors `CDS` record at child's nameservers for the period of three days[51]. They also provide method of checking at which stage of monitoring domain name currently is[52]. Their solution is proprietary, so I cannot discuss here any implementational details.

Several DNS managers offer `CDS/CDNSKEY` support, like DNSimple[53] and Cloudflare[54]. It is important to remind reader, that DNS managers are not registrars and do not have any direct contact with Registry. This means, that unless TLD, under which domain is registered, supports `CDS/CDNSKEY`, those records do not serve any particular purpose, only waiting when will the time come.

Monitors

It might still be useful to automatically manage subdomains, even if TLD doesn't support `CDS/CDNSKEY`. For example, domain contains a lot of subdomains, which in turn manage a number of subdomains themselves (e.g. `cvut.cz`).

I came into contact with only a couple of CDS/CDNSKEY monitors. One of those is "CDNSKEY AutoDNS", which works with another domain management system AutoDNS (open source at https://github.com/InterNetX/cdnskey-autodns). It is paired to AutoDNS account, and updates DS record through it's API. This tool checks only CDNSKEY and computes DS itself.

Second tool I encountered is "CDS/CDNSKEY Monitoring Prototype" made by former CZ.NIC employee as an entry to a hackathon (open source at https://github.com/fcelda/cds-monitor). Currently it supports CDS record. It fetches CDS, checks if parent zone contains same key, and if not, updates it. This tool is connected directly to server software, and update happens through Dynamic DNS mechanism.

1.6 CZ.NIC's FRED-AKM

TLD .CZ implements CDNSKEY monitoring of domains in it's zone, which is run every day. If domain is not yet secured by DNSSEC, "Accept after Delay" strategy is applied, and if monitor had found consistent and CDNSKEY records across all of the domain's nameservers, domain becomes candidate for DNSSEC and technical contact for this domain is informed. Now monitor for seven consecutive days checks whether those keys are in exactly the same state every time monitor scans the domain [2]. If so, DS record at the .CZ zone is updated and notification of technical contacts is issued. If during this period key changes it's state, then period is considered broken and starts all over again, and technical contact is notified.

Tool, that performs it is called "FRED-AKM" and is actually part of a bigger system called FRED. Following text is based on FRED documentation which could be found at https://fred.nic.cz/documentation/html/index.html.

1.6.1 FRED

FRED (Free Registry for ENUM and Domains) the domain name registry software developed by CZ.NIC as an open-source solution. It is structured around the Registry-Registrar-Registrant model, allows automation of zone file generation, notifications of contacts and a lot of other features. It is quite complicated piece of software with multiple components.

[2] I use words "monitoring" and "scan" interchangeably with the meaning "querying for CDS/CDNSKEY records in order to analyze them for the possible update". Relevant RFCs prefer word "monitoring", "scan" is an internal term for FRED-AKM.

In the top-level view of the system, there are four classes of acting components:

- **User agents** communicate with a system on behalf of the user and are not part of FRED
- **Clients** perform actual work initiated by user through user agent, or started as registry jobs
- **Servers** manipulate with database on the instructions from clients
- **Database** which holds information related to every feature.

Classes, that are part of FRED, contain components corresponding to each feature, such as registrar interfaces, generation of a zone and so on [3]. Basically, every functionality is represented by group of programs: one is "Server" part, other is "Client" part.

1.6.2 FRED-AKM

FRED-AKM stands for "FRED Automated Keyset Management" and it is a program, that is designed to monitor `CDNSKEY` RR in the domains, registered under TLD. It is divided into `fred-akmd`, CORBA server daemon, that communicates with database, `fred-akm`, command-line tool that initiates scans and evaluates scan results and `cdnskey-scanner`, utility that performs scan and evaluates state of a key (it's immediate state: if the key is present, or was scanner even able to connect to nameserver and so on). FRED-AKM serves multiple functions:

- Performs a scan of given domains on all of their nameservers
- Evaluates results of a scan: was `CDNSKEY` RR present and if so, for how long? Is this record signed and if not, was this presence consistent throughout the acceptance period?
- Updates `DS` record at parent zone
- Notifies contacts about the state of a scan: when was acceptance period initiated, was it disrupted by some error, and about `DS` record update.

In the following text I will describe `fred-akm` and `cdnskey-scanner` in more length. I will not talk about `fred-akmd` daemon, as it is not as relevant to this work.

[3] Detailed description of FRED is out of scope of this work, and for more information on it visit `https://fred.nic.cz/documentation/html/`

cdnskey-scanner

Through standard tools, such as `dig` it is possible to get `CDNSKEY` record of a domain. The problem is, that for the large number of domains and nameservers such as `.CZ` TLD, this solution is not effective enough. Motivation behind `cdnskey-scanner` is more efficient scan of a large number of domains.

This tool as an input takes list of nameservers and corresponding domains, and outputs status of `CDNSKEY` record. To prevent flooding of DNS infrastructure, `cdnskey-scanner` implements simple scheduler, that, given a time limit for running, spreads requests universally across this period. It is important, as many domain owners delegate management of a domain to a third party, so small number of nameservers hold authoritative data for a lot of domains.

`cdnskey-scanner` accepts two explicitly defined types of domains: secure (that already have `DS` record at parent's site) and insecure. Output has following types of results:

- **insecure**: at particular nameserver for insecure domain there was found `CDNSKEY` included
- **insecure-empty**: at particular nameserver for insecure domain `CDNSKEY` record was not found
- **secure**: for secured domain exists signed `CDNSKEY` included
- **secure-empty**: for secured domain no `CDNSKEY` was found
- **untrustworthy**: for secured domain it was not possible to verify existence or non-existence of `CDNSKEY` record
- **unknown**: tool was unable to acquire any information about secured domain.
- **unresolved**: tool was unable to acquire any information about insecure domain from particular nameserver
- **unresolved-ip**: tool was unable to get IP for nameserver.

fred-akm

`fred-akm` is a program, that performs most of the work in automated keyset management. It is a CLI-tool, that runs a command and then exits. It has it's own operational database, where it stores intermediate scan results throughout acceptance period (in the context of `.CZ` zone, it is seven days).

There are five different commands:

- **load** loads list of domains, that need to be scanned, from database. On practice, it loads all domains in the zone.
- **scan** initiates scan through running `cdnskey-scanner` as a subprocess, giving it all domains inserted by "load" operation either in it's entirety, or in smaller batches
- **notify** takes last scan as current state of a domain, and, depending on the latest notification regarding this domain, issues email to technical contact
- **update** performs necessary evaluations about status of the key, and updates main FRED database through remote procedure call at server `fred-akmd`
- **clean** cleans scan results that are older than acceptance period.

Sequence of those commands is run periodically once a day.

This tool implements "Accept after Delay" strategy, but performs scanning only from one location. The problem of this approach, is that someone in the network could subvert responses from insecure domains. The aim of this project is to correct this flaw.

Summary

DNS is a system, that translates domain names to respective IP addresses, it also able to attach to those names other information, such as mail server. DNS is hierarchical, with root servers serving addresses of nameservers of TLDs, and TLDs serve names to second level domains and so on. One common way to manage a TLD is Registry-Registrar-Registrant model, where Registry is responsible for administering TLD, setting policies for this TLD and so on, Registrar is an organization, to which Registry delegated the authority of domain registration, and Registrant, owner of a domain name. It is complicated system, with lots of caveats, so there are many attack vectors against DNS.

Many RFCs defined security solutions for DNS, aiming at different situations and needs. One such solution is DNSSEC, which introduces data origin authentication and authenticated denial of existence. It solves security problems such as packet interception and cache poisoning. More than 90% of TLDs are signed and have trust anchors at the root zone(`DS` records)[55]. Yet, adoption of DNSSEC is still low and slow: major role in this play registrars, even ones that support DNSSEC may not be able to update `DS` records at parent's site[56].

To the rescue comes new proposal, that will allow update of records at parent site in semi- or fully-automated way. `CDS/CDNSKEY` records signal to the parent, that child wishes to update trust anchor at parent. It is quite new, and is not yet widely adopted, only a handful of TLDs perform such zone scans in order to retrieve those records. There are a couple of software projects, that implement this feature, but most of them are not applicable in the TLD scale.

FRED-AKM is a tool, that provides such functionality as a part of bigger registry administration system, that follows "Accept after Delay" strategy from RFC 8078. FRED-AKM is coupled closely with that system, so in order to benefit from it, registry should use it in it's entirety. It lacks some recommended features, such as multiple scan points, and the point of this project is to add those features.

Chapter 2

Design

This chapter outlines major ideas and motivations behind this project. It also describes design and architecture of the program.

The project is aiming to provide a tool for `CDNSKEY` scan of a zone from multiple locations; thus the name of a project AKM Multi Scanner. (from the original, and, in many cases, parent project FRED-AKM by CZ.NIC). The main goal is securing domains that are not yet DNSSEC-enabled by scanning it's `CDNSKEY` RRs from multiple locations, thus minimizing a chance for spoofed packets. One of the major ideas of this project is to have independent system for scanning and evaluation process. "Independent" in a sense that it would not be a part of any particular registry administration system, also flexible in a way user can locate scanning vantage points.

As mentioned, FRED-AKM is a project from which this work has risen. Some of code and ideas were taken from it, as it's not necessary to invent a bike when already working solutions might be taken and improved on. Those improvements include:

- Addition of multiple vantage points for more secure enrollment process
- Decoupling from registry administration system FRED
- Decluttering from unrelated tasks such as contact notification in a spirit of UNIX philosophy
- Shift from "run and get result" to continuous daemon/server
- Migrate from SQLite to a better database solution

2.1 Requirements

The system designed should comply with the following functional requirements:

- Import list of domains for a scan
- Scan nameservers corresponding to those domains for `CDNSKEY` RRs from multiple vantage points
- Evaluate results of a scan
- Export results of a scan
- Provide system health checks.

System should also meet the following non-functional requirements:

- Be configurable to use any number of vantage points
- Use more suitable database solution for the amount of data scanner has to manipulate
- Transport scan requests securely to avoid situation similar to zone-walking, as domains that require scan from multiple points are the most vulnerable ones.

2.2 Architecture

System uses Master-Worker pattern: Master provides interface to the system, administers Workers and performs evaluations; Worker is the one who actually scans domains, but does nothing else. The system is distributed; that is, conceptually Master and Workers could be located in the different networks.

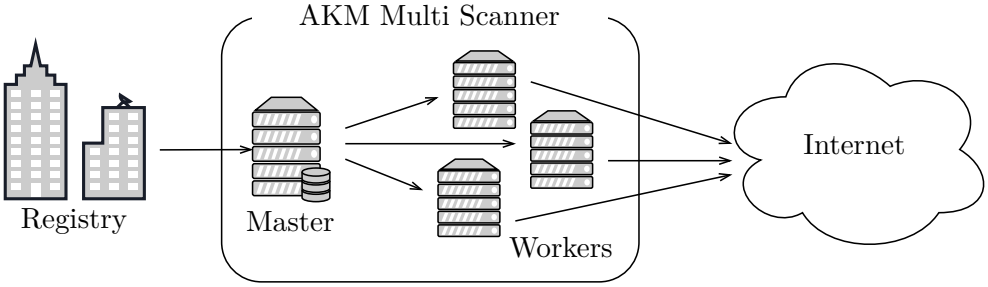

Figure 2.1: General architecture of AKM Multi Scanner.

The following text uses terms related to the scan operation listed and explained below:

- **Scan job** is atomic structure that contains information needed to perform scan of one domain at one nameserver
- **Scan queue** scan jobs that are prepared for a scan iteration
- **Scan task** is collection of scan jobs, organised either by domain or by nameserver
  - **Domain** scan task reflects the fact that domains usually have more than one nameserver;
  - **Nameserver** scan task refers to the way domain information is stored; nameserver might be responsible for containing more than one domain.
- **Scan request** is one or more nameserver scan tasks, sent to Workers
- **Scan result** is paired to scan job and describes status of a scan: if `CDNSKEY` is present or not, was scanner even able to connect and so on
- **Domain status** is evaluation of latest scan results regarding particular domain across nameservers, time and workers for insecure domains, and simply latest scan result for secure domains.

2.3 Master's operation

2.3.1 Interface

Procedures, that are formally provided by Master, are:

- import of domains for a scan
- export of scan results
- diagnostics: request health of database or health of Workers.

One might notice the lack of a scan itself. While Master initiates scan process, it doesn't scan itself and outsources this work to Workers. The scanner is meant to run as a daemon, making a scan everyday at fixed time. Job of a registry is to provide domains that are planned for a scan for each planned iteration.

Import of domains for a scan

At import scan queue is created, it is assumed that each scan queue is prepared for one scan iteration, for example if scan should happen once a day then each queue corresponds to one day. Scan queue is then stored in a form of collection of scan jobs.

The reason for creating new scan queue each time is that domain list changes everyday: new domains are created, some are removed and others might be re-registered (thus making it new domain with the same name). Then interface for import is a collection of domain scan tasks.

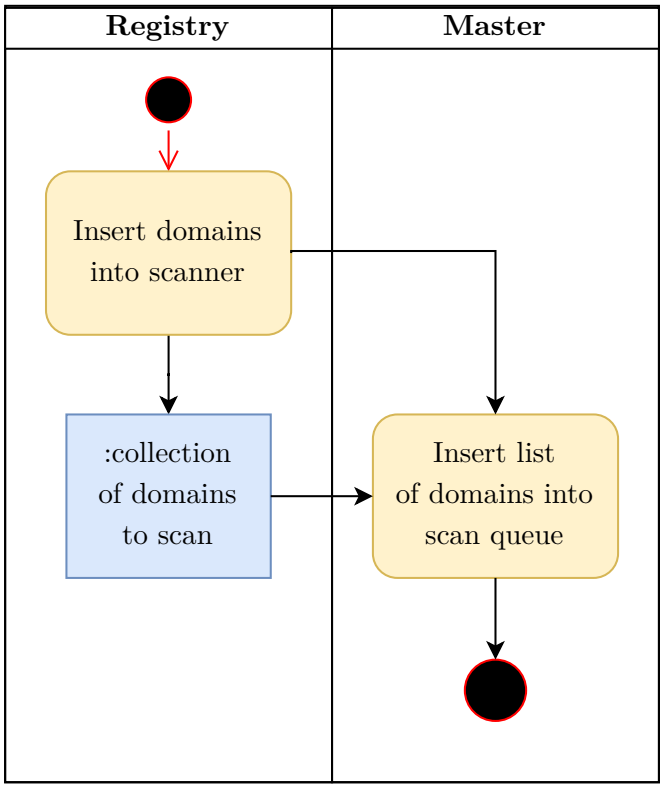

Figure 2.2: Process of importing domains for a scan.

Export scan results

It is unnecessary for a Master to perform extra job by evaluating results without request for them, so evaluation would happen at the moment Registry requests results. It means, that older (irrelevant) scan results would be wiped no matter what, and Master will respond only with current results.

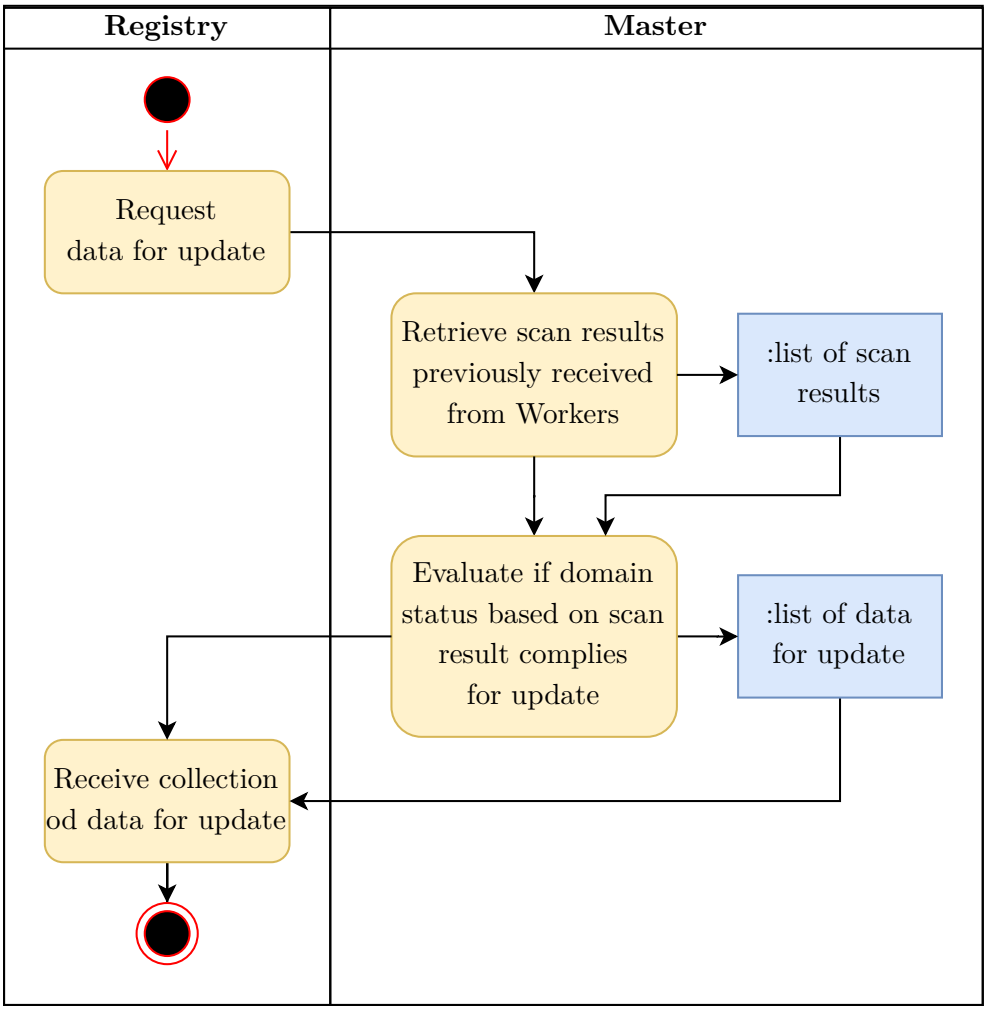

Figure 2.3: Process of exporting scan results from Master.

Diagnostics

There are two types of diagnostic commands:

- Worker health check: are all of the Workers still up and running?
- Database health check: is connection to the database still up?

Diagnostic commands are straightforward: upon receiving a diagnostic request from Registry, Master performs necessary checks and returns result. For database check it just checks if connection to database is still possible. For Worker heath check Master issues "ping" messages to Workers and after receiving "pong" (or not receiving anything for some time) Master replies to Registry with results.

2.3.2 Evaluation

For a secure domain, it's enough to resolve query through a normal means of DNS, i.e. request resolution from trusted DNSSEC-aware recursive resolver, because `CDNSKEY` records would be signed just as any other record in this zone. So evaluation process for those domains is just looking at the result of a request if it contains correctly signed `CDNSKEY` record.

For insecure domains it is much more complicated. First, every nameserver for this domain needs to be monitored in order to avoid introducing new key from attacker-controlled nameserver. If the attacker took advantage of a nameserver, it is not probable that they took advantage of all of them. Second, it might so happen that attacker is capable of issuing forged responses directly to the scanner (perhaps being in the same network). But it is hard to keep it up for a long time without being noticed, and for this reason acceptance period is implemented. Both of those measures are implemented in FRED-AKM. To harden security even more, for the situations where an attacker was able to forge responses for sufficient amount of time, was introduced scan from multiple locations.

Essentially, for insecure domain all responses from every nameserver, every Worker for enough amount of time should be the same so new keys could be introduced into the parent zone. It is possible to reduce rigidness of this system by utilising the idea of "quorum". With quorum – minimal number of valid responses from Workers – system could be more flexible in the context of network errors. Consider a situation when one Worker was out because of server maintenance, and for this reason without quorum every scan result in iteration for this day is now invalid, breaking the acceptance period.

2.3.3 Scanning process

Scanning process starts with retrieving previously stored scan queue and transforming it into two collections of nameserver scan tasks, one for domains that are not yet secured by DNSSEC, and the other for domains that are secured. This is done, because those groups need to be processed differently: domains that are secured do not need to be scanned from multiple locations multiple times. `CDNSKEY` records are already secured by a present key. This decreases complexity and load on nameservers. Insecure domains are those that need scanning from multiple locations.

Master sends those tasks, and upon receiving results from Workers, it saves them for future evaluation.

2.4 Worker's operation

Worker is designed to perform a single operation, that is running a scan, from any place. In a sense, Worker acts as a RPC server: upon receiving request

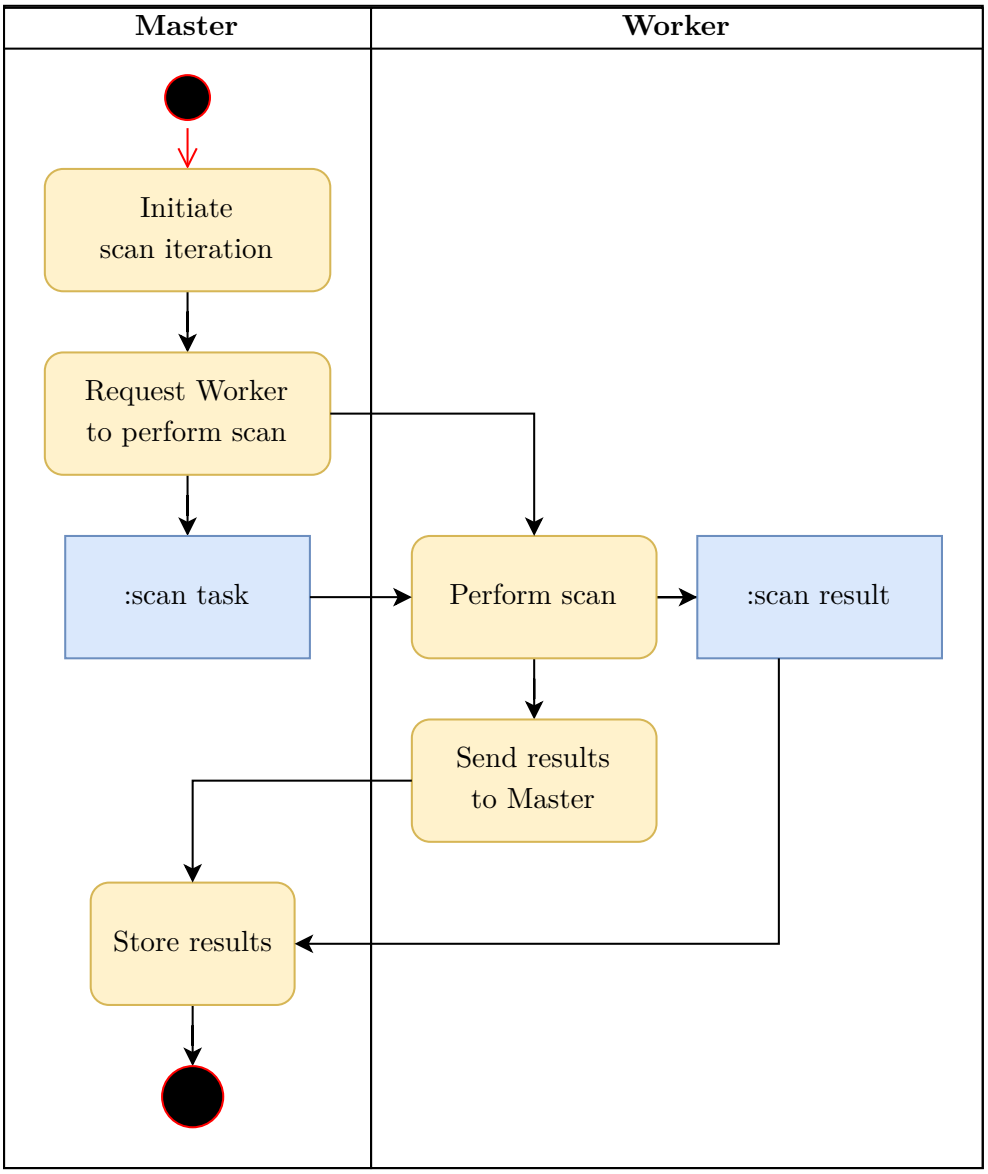

Figure 2.4: Process of delegation of domains to Worker and receiving scan results.

for a scan, worker runs scanner tool and returns results to Master as soon as they are ready. System allows for a health check on Workers through Master.

Scanner tool is run as a subprocess, allowing user to choose scanning tool of their liking. AKM Multi Scanner uses scanning tool `cdnskey-scanner`, which is available at `https://github.com/CZ-NIC/fred-cdnskey-scanner`. Worker uses it's syntax for input and output parsing, so in order to use some other tool, wrapper is needed that will mock the I/O of the `cdnskey-scanner`.

`cdnskey-scanner` also schedules scans of domains; generally scanner needs to space queries in time in a way that will not trigger nameserver's reaction to sudden load spike. Strictly speaking, scheduling is not a mandatory functionality of a scanner for the whole system to work, but without it it might so happen that domain would be longer in enrollment or key change process than necessary because nameserver would be overloaded and not be able to provide responses to queries.

2.4.1 Recovery from record retrieval failures

Record retrieval failures could be analyzed by Master or by Worker. In order to actually recover from those failures, either new query has to be issued and Master is not conceptually capable of doing so thus it has to be done by Worker, or if failure occurred in a comparatively small group of Workers, those faulty results might be just discarded (as in quorum model described earlier). The advantage of analysis at Master is that Worker gets to keep it's simplicity, but at the cost that Master does much more work and also in the case of new requests, it has to distribute those requests again. The analysis is not really that complex, so decision was to keep record retrieval failure analysis on the Worker.

If nameserver for domain is unreachable because of the network failure, or perhaps due to server maintenance, Worker wouldn't send results to Master and it might try querying it again multiple times later until it receives different result, or until it exhausts all available attempts. It then send whichever result to Master.

Summary

Responding to a need to perform `CDNSKEY` monitoring from multiple sites, newly developed solution AKM Multi Scanner is a distributed application, that follows Master-Workers pattern. It is based on original solution FRED-AKM by CZ.NIC, yet it is decoupled from FRED itself. Master acts as an interface for Registry: it imports domains for a scan, exports scan results, and performs diagnostics. It is also a command center for Workers, it distributes tasks for scans and evaluates scan results. Worker is expected to be located in different networks, and it's single task is to perform scans on demand of a Master. Evaluation of scan results happens on demand issued by Registry.

Chapter 3

Implementation

This chapter is going to detail implementation of this project, mainly technology that was used. The system is written in C++, thus technological solutions to problems stated by this project should be compatible at least with C. I discuss current state of the implementation, as well as possibilities for future improvements.

3.1 Communication

Communication is a core problem that must be solved by a distributed application.

The system has two separate communication processes: between Registry and Master (called in text "external"), and between Master and Workers ("internal"). There is no direct communication between Registry and Workers during operation. The communication processes, while being quite similar (in a sense that both Master for Registry and Worker for Master are akin to RPC servers), are different in the timing of operations: Master is fast to store tasks and evaluate results already present, but Worker is much slower as it has to actually communicate with nameservers, accommodate for network failures and so on.

On the early stages of design it was important to choose suitable communication framework for both of those processes. Due to a clear line in the sand between internal and external communications, it is possible to use different frameworks. This might have some advantages as well as drawbacks. Using just one framework might be better, because programmer and administrator of a system doesn't have to care about all the different ways those technologies function and how they have to be maintained. It also might improve consistency of the system. The problem arises when one has to bend one framework over the case, where some different framework might work better.

Due to slight difference between internal and external communications discussed earlier, the choice was made to use different frameworks. For ex-

ternal communication gRPC was chosen, for internal AMQP with RabbitMQ message broker was picked as the most suitable. The reasoning behind these choices is discussed in more length in the next two subsections, subsection 3.1.2 elaborates further on the decision to use different frameworks.

3.1.1 Registry-Master

Communication between Registry and Master is designed to follow RPC pattern. RPC (or Remote Procedure Call) is model of distributed programming, where program calls subroutine, which in fact is executed on the remote machine, just as if it was local subroutine. Here Master provides remote subroutine for Registry: import of domains and export of results.

There is a number of frameworks available that provide such functionality, and gRPC was chosen for a number of reasons. It is fairly easy to use and setup. Other systems, such as CORBA, might provide richer set of features at the expense of increased complexity. Since RPC is all that is needed, gRPC is a perfect choice. Another reason is that gRPC is becoming standard at CZ.NIC, and using some new technology creates more confusion and inconsistencies in the projects. gRPC also natively supports C++.

The RPC server is binary on it's own, running independently of scanner itself. This design decision was made to increase reliability. Failure of RPC server doesn't impact scanner, and failure of a scanner is then detectable by RPC server through diagnostics. It also provides clearer division between functionalities.

Control interface

In current version control interface contains only two procedures: `ImportTasks` and `GetScanResults`.

Interface definition for `ImportTasks`:

- **Input data**: List of Domain Scan Tasks. Each Domain Scan Task contains following data:
  - Domain information: UID, FQDN, and scan type: either `secure` or `insecure`. Scan type is provided by Registry, as additional queries to TLS's nameservers would complicate the process and overload nameservers even more.
  - List of relevant nameservers names
- **Output data**: Id of a queue that was created from those tasks.

Interface definition for `GetScanResults`:

- **Input data**: doesn't take any input.

- **Output data**: List Scan Results. Each Domain Scan Task contains following data:
  - Domain's UID (as said earlier, to prevent newly registered domain of the same name to enter acceptance period in the middle)
  - `CDNSKEY` value, that was extracted. Contains every field for a `CDNSKEY` as seen in 1.2.

Diagnostic interface

Diagnostic interface implements two simple features: check health of a worker and health of a database (is it possible to connect). Both do not require any input. `WorkersHealth` returns a list of messages, that contain Worker's id and boolean status `ok`. `DatabaseHealth` returns a message that contain just boolean status `ok`.

3.1.2 Master-Worker

Communication between Master and Worker is spread in time. Master might send entire scan queue to Workers in one message, or send it in batches. Worker might as well process tasks in batches, especially in the context of scheduling DNS queries across nameservers. This style of communication is rather uncomfortable to implement using RPC, even though Worker is literally performing subroutine for Master, mainly due to time required for response.

If I insisted on using gRPC, I would have to use asynchronous calls, which are clumsy in C++ and require waiting on futures anyway, eliminating the entire advantage of using them in the first place. To dodge asynchronous calls architectural changes should be made to utilize either push or pull strategy, both of which create unnecessary traffic. Push strategy in this context would be making Worker RPC server, and after sending scan tasks, query Worker for results in some time intervals. Pull strategy makes Master the server (which, above all, is counter-intuitive in a design perspective), and while sending results is not a problem, task distribution is – Worker now needs to query Master for work, which makes Worker the one who needs to schedule scans, breaking the logic of Master-Worker dynamic. It is still possible to keep architecture and use gRPC though: make both Master and Worker servers, where Worker serves scan, and Master serves result saving. Such technique would massively overcomplicate the program and at this point it is obvious, that both Master and Worker should play equal roles in message sending.

It is possible to achieve what is needed with message-oriented middleware. Message-oriented middleware (MOM) acts as a broker between system's entities that need to pass messages, so entities actually communicate with broker instead of each other. Then, broker is server and other entities are clients. This model of communication is asynchronous. Broker solves for those parties

tasks such as unicast, anycast, multicast and broadcast. Every client in the context of communication with broker could publish a message, and subscribe to receiving those messages.

AMQP (Advanced Message Queuing Protocol) is a messaging protocol that enables communication with a message broker middleware. One middleware that supports AMQP is RabbitMQ, open source message broker. One could directly install RabbitMQ on their machine, or use a Docker image. There are also "RabbitMQ as a Service" such as CloudAMQP. There is no official client for C++, but there is a couple of decent implementations. I use the "AMQP-CPP" library by Copernica Email Marketing Software, available at `https://github.com/CopernicaMarketingSoftware/AMQP-CPP`. RabbitMQ provides possibility of secure communication over TLS; AMQP protocol over TLS is called AMQPS.

Some parts of gRPC are utilized here too, namely protobuf IDL. Though both gRPC and AMQP are not dependent on any particular IDL, protobuf seems like an optimal option as it is gRPC native and easy to serialize to use in any transfer.

3.1.3 Internal layout of RabbitMQ server

Following text relies on RabbitMQ documentation, which explains basics of AMQP model and RabbitMQ specialities. The documentation could be found at `https://www.rabbitmq.com/documentation.html`.

AMQP has two basic types of resources: *exchanges*, which route messages to queues, and *queues*, from which messages are then consumed. Both of those entities are created independent on each other, and are identified by their names (a simple string of characters). When message is published, it is published to specific exchange. In order to consume message, queue has to exist and be bound to exchange. There are multiple types of exchanges:

- **Direct** exchange routes message to specific queue based on message's routing key. Routing key is also a property of binding between exchange and queue.
- **Fanout** exchange broadcasts message to all connected queues.
- **Topic** exchange routes one message to many queues, based on pattern of routing which was used in binding.
- **Headers** exchange routes message to queues based on multiple attributes, that are more easily put in message header, rather that the routing key.

Queue is a "First-in-First-out" type of sequence, which could be bound on one ore more exchanges in order for messages to be consumed. It is possible

to limit queue's length by the number of messages, as well as by size in bytes. If multiple consumers are connected to queue, then messages are distributed between them. Queues have their own set of properties, such as:

- **Durable** queue is able to survive restart of a broker
- **Exclusive** queue could be used by only one connection, and will be deleted after the connection is closed
- **Auto-deletable** is queue, that could serve more consumers, and when the last one is gone, queue is deleted
- other properties, that might be used by broker and are implementation-specific to a particular broker.

In order to logically separate applications that use the same RabbitMQ server, there are *Virtual Hosts* or "vhosts", which represent a logical group of entities. RabbitMQ server allows setting up *users*, that could be given different permissions. Permissions are granted to user regarding particular vhost, include things like permission to write to particular exchange, to read from queue, or to create/remove any of the resources.

RabbitMQ sports a wide range of configuration options with which user could better adjust functionality of resources to suit their needs such as high availability or throughput.

3.1.4 Layout of a vhost for AKM Multi Scanner

Layout preparation for this application should be done in configuration.

AKM Multi Scanner should use one vhost `/akm-multi-scanner/`, inside of which other resources are defined. The main reason, why the distributed application is needed in the first place, is to secure introduction of a new keys for domains, that are not secured by DNSSEC yet. There are four exchanges:

- `e-insecure` is is a fanout exchange, that broadcasts scan tasks for insecure domains scanning from Master. To this exchange are connected multiple queues, each corresponding to a Worker.
- `e-secure` is a direct exchange, that sends scan tasks for secure domains from Master to one or more Workers through one queue. Current implementation connects to this queue only one Worker, an it is assumed, that this Worker is on the same machine/network as Master.
- `e-diag` is a fanout exchange, that sends a health check from Master to Workers
- `e-response` is exchange, which Workers serves as a collective exchange for every scan result, Master then through message metadata decides, from which Worker result was sent.

Figure 3.1 show entire architecture of AMQP resources. Diamonds depict exchanges, long rectangles depict queues.

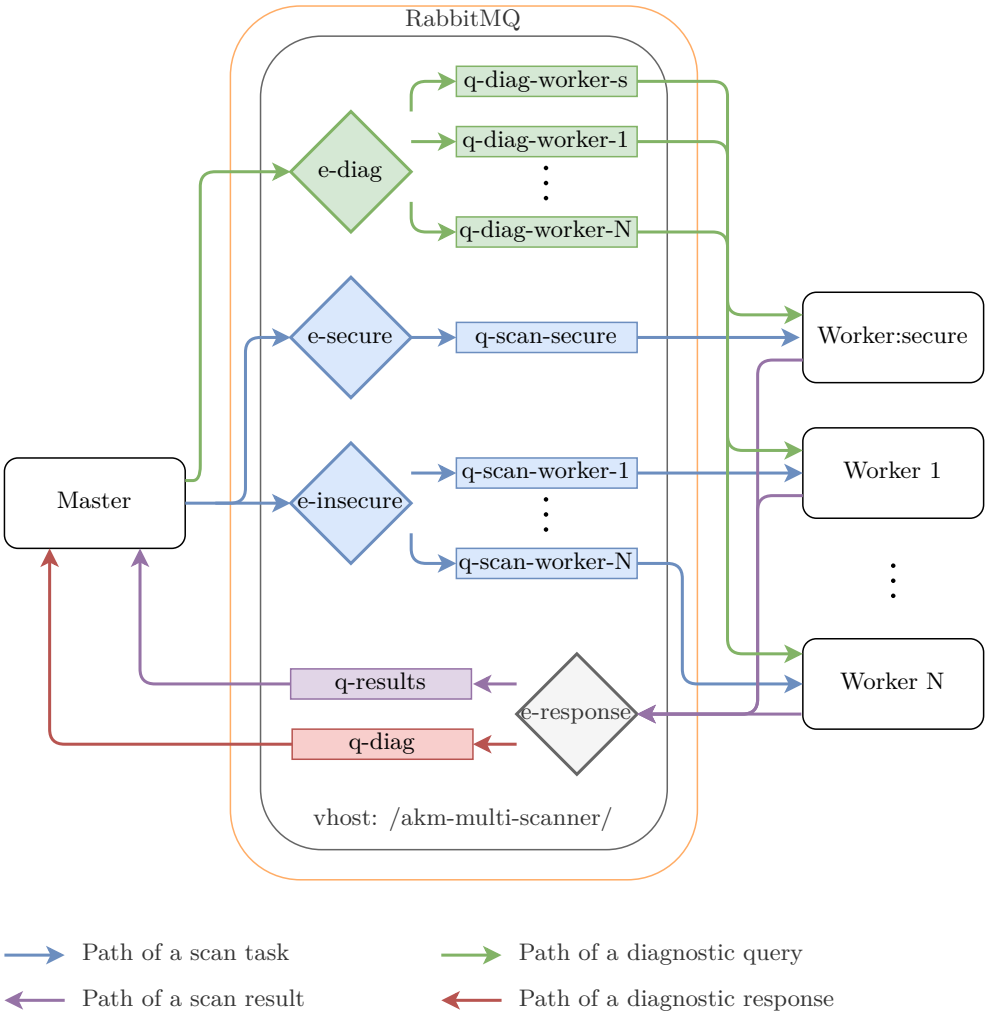

Figure 3.1: Complete picture of exchanges and queues in RabbitMQ server for AKM Multi Scanner

As mentioned in description of exchange `e-secure`, it is technically possible that more Workers could connect to the queue, that distributes scan tasks for domains that are secured. It might benefit that those domains could be scanned in parallel, which theoretically could improve efficiency. It is up for evaluation, if this actually would be the case. Figure 3.2 shows how portion of vhost for scanning secure domains would change.

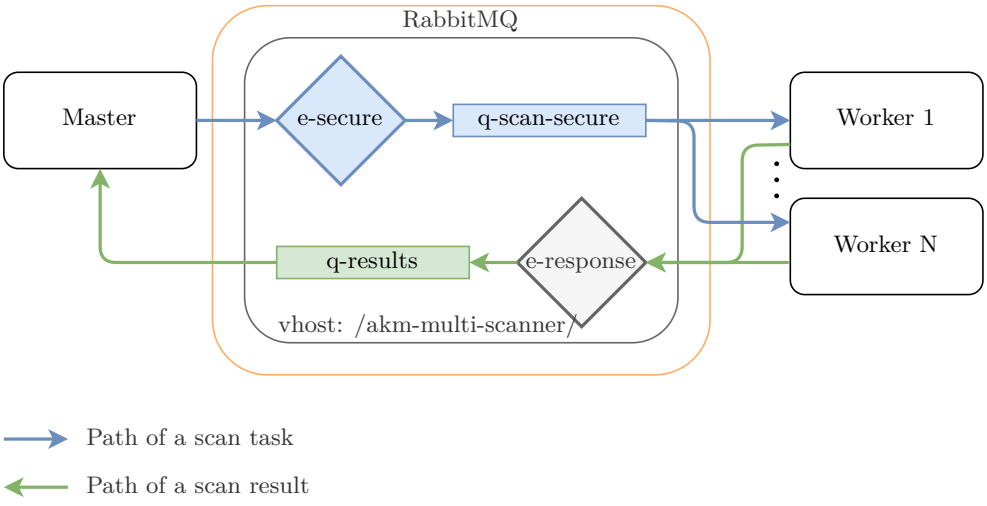

Figure 3.2: Possible alternative to the way, how secured domains are handled now: with a direct exchange with one queue, Workers could connect to this queue and work would be distributed evenly across them (with the assumption of a correct queue configuration).

Overview of a communication infrastructure

With the knowledge of technologies used here is presented the overall communication infrastructure. Master is in fact two entities: a scanning daemon, and a RPC server interface between Registry and the system. Both of those entities communicate with the database. Worker is a simple daemon that listens to the requests from Master. Both Master and Worker communicate through message broker, and are clients in the context of this relationship. Graphical representation of this relationship could be found in the figure 3.3.

3.2 Database

For Worker, it is enough to keep scan tasks in memory, since it's only job is to perform a scan and nothing more. Master, on the other hand, needs to keep results for some amount of time (for example, for `.CZ` domain it has to keep for seven days). Original FRED-AKM used SQLite, and practice shows, that for the amount of data that it saves, it is not a suitable solution, so a change must be made. For this project was chosen PostgreSQL. PostgreSQL is open source relational database, and is also used in FRED. To communicate with database in code I use C++ library `libpg`, a thin C++ wrapper around a C-language PostgreSQL library `libpq` that's written at CZ.NIC.

I use a very simple schema with only four main tables (and two enum tables, that define state of a domain and type of result of a scan). Figure 3.4

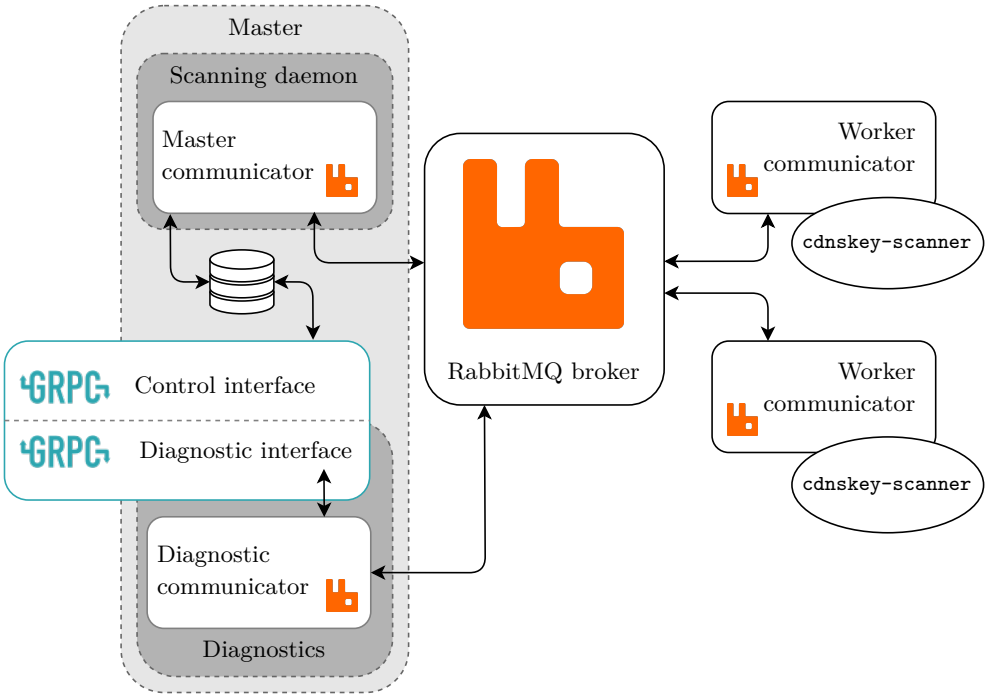

Figure 3.3: Complete structure of communication between elements of a system. Images for gRPC and RabbitMQ are taken from `https://grpc.github.io/` and `https://www.rabbitmq.com/` respectively.

shows relationships between tables.

3.3 Scanning process

After Worker received Nameserver Scan Tasks from Master, it translates this data to text format used by `cdnskey-scanner` and run it as a subprocess. Current implementation doesn't allow any other tool, but it is planned to change this to some configurable option. At the current stage Worker should leave scheduling between queries onto `cdnskey-scanner`, so Worker passes entirety of request from Master to it. `cdnskey-scanner` takes input from `stdin` and puts output on `stdout` in text format. Worker than translates this result from text to internal representation.

In the context of Worker, job is a sequence of scanning and sending results. Worker never really analyses the results, but it checks for `unknown`, `unresolved` and `unresolved-ip` results, that collectively tell that scan was unsuccessful perhaps because of some network failure or unavailability of any node in the chain of resolver and nameservers. Worker collects results that ended with any other status and sends those to Master. Jobs with faulty results are performed again multiple times (currently it is three, but in the

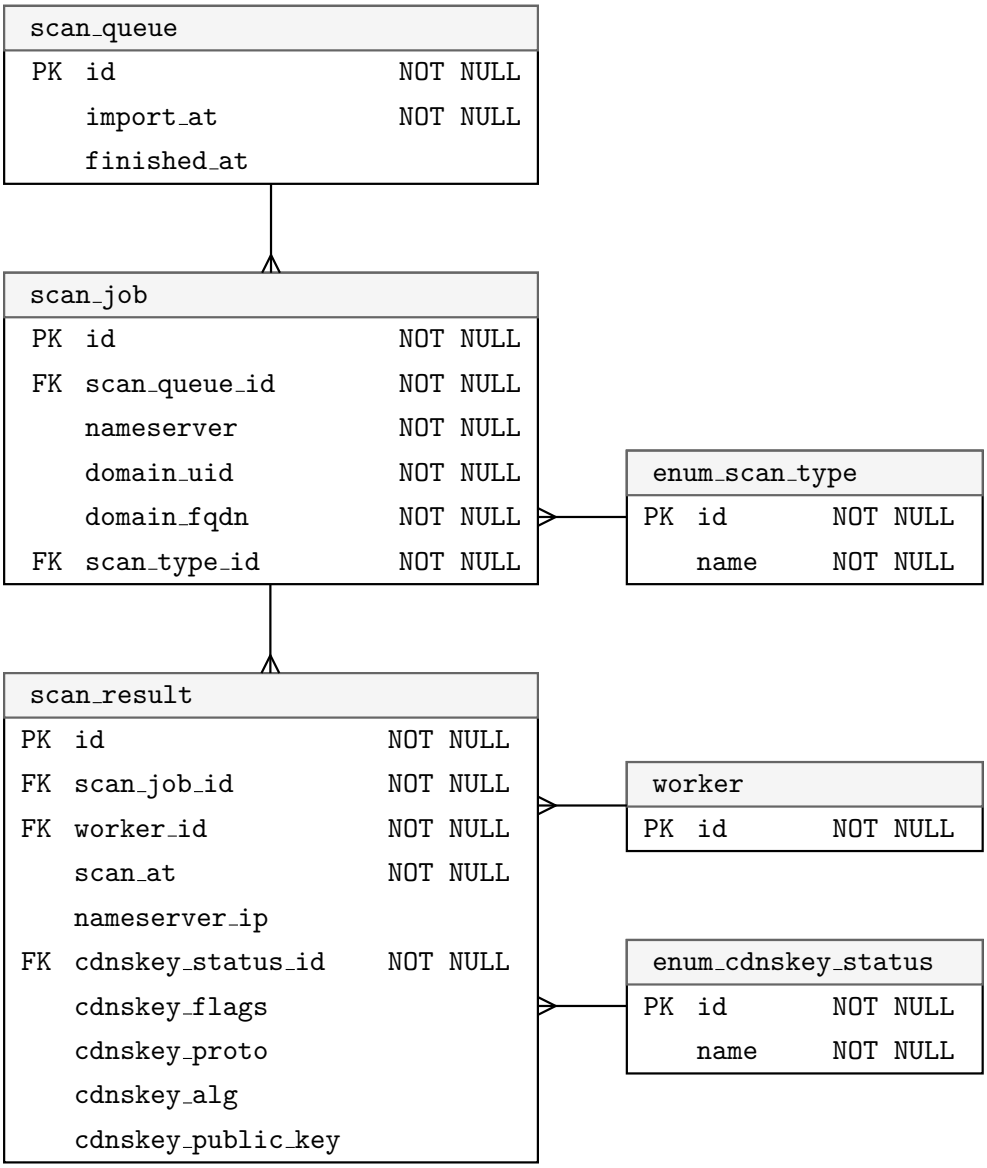

Figure 3.4: Database schema for Master.

future this amount would be configurable), each time analysing if faulty results are present again and sending results that resolved with any other valid status. If after multiple retries faulty result stays, it is sent as it is to the Master.

3.4 Evaluation

Evaluation is performed directly through SQL query. For secure domains it is the latest result from latest closed queue, for insecure domains it will compare all of the results from every worker, every nameserver from seven latest queues. At this point, this evaluation strategy is very weak: it assumes that every reply from every Worker was received (that is, if some Worker didn't returned result of a scan task, this failure doesn't count as failure of evaluation). More reliable approach would be to check if every scan job has sufficient amount of results, otherwise scan job is considered failed and existing results for it could be wiped.

Summary

The system uses complex tools for communication. To communicate with Registry, it implements RPC interface through gRPC, to pass scan requests and results between Master and Workers it uses AMQP with RabbitMQ broker. As history is needed for evaluation of scan results for insecure domains, PostgreSQL database is used. While top-level view of the system is pretty simple and straightforward, there a lot of intricacies of algorithm and technology that needs to be considered. The result of this thesis is more of a "proof of concept" than finished product. With that being said, work on this project will be continued with the hopes of eventually reaching easy to setup and robust state of this software.

Conclusion

This thesis described operational and administrative details of DNS, it's security issues and how those issues could be mitigated. The main focus was DNSSEC, how complicated it could be to roll over keys at parent's site and how to ease this complexity by automation. In order to automate key rollovers new resource records should be introduced: `CDS` and `CDNSKEY`. There are not much done in this field, and analysis of software solutions for `CDS`/`CDNSKEY` scanning and zone updates is presented. Most of those solutions are not suitable for TLD scale. One such solution – FRED-AKM – is specifically designed for the scale of TLD, but lacks scanning from multiple vantage points.

New scanning tool was designed based on FRED-AKM: AKM Multi Scanner, which allows scanning from multiple locations and could also be used in any system through API. It uses Master-Workers pattern, where Master manages workers, responds to API calls and evaluated results of a scan, and Worker is a simple scanner with a network interface. Scheduling was moved from outer means such as `cron` to the inside of a system, so it could be run as a daemon. This work attempted at reducing false negatives by introducing redundant scans. It also implements simple diagnostic tool, which allows health checks of Workers and Master's database. This solution uses multiple different technologies for communication, for Registry-Master communication gRPC was chosen, and for Master-Workers it was more suitable to switch to AMQP.

Even though the main goals are met, this work is far from done, as time to work on this fairly big project was very limited. I had to get familiar with technologies that were new to me, and learn to use them to the extent necessary for this project. First of all, it is yet to be tested, and testing of such application is out of my competence and out of scope of this thesis. This means that a lot of edge cases are waiting to be discovered and covered. Second, there is a lot of space for improvement, from the logical stand point, as well as technical one. Configuration of Master and Workers as RabbitMQ clients received very little coverage, text merely pointed out possibility of detailed permission settings

for them. The whole area of administering such a system was not properly discussed, such as management of certificates used in RabbitMQ communication, or a necessity for configuration distribution (for example, should some configurations of a Worker be distributed by some out of band means, or could it be done through message broker). As already mentioned, work distribution is up for improvement too: as number of secured domains grows[4], grows time needed to scan those as well, so work might be distributed among Workers and not be delegated to just one selected Worker. Scheduling of scans in the context of a single iteration could also change, currently this solution relies solely on scheduling mechanism present at `cdnskey-scanner`. Even though it is out of the scope of this thesis, from a practical point of view, there needs to be agent, that connects AKM Multi Scanner with the Registry.

All in all, work on this project is not ending with this thesis, so development will continue and improvements are coming.

[4]Which is the reason why this project exists in the first place

Bibliography

[1] Aitchison, R. *Pro DNS and BIND 10*. Apress, second edition, 2011, ISBN 1430230487,9781430230489.

[2] Mockapetris, P. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. Available from: https://tools.ietf.org/html/rfc1034

[3] Vixie, P.; Thomson, S.; et al. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136, RFC Editor, April 1997. Available from: https://tools.ietf.org/html/rfc2136

[4] Mockapetris, P. Domain names - implementation and specification. STD 13, RFC Editor, November 1987. Available from: https://tools.ietf.org/html/rfc1035

[5] ICANN. Information for Domain Name Registrants. Available from: https://www.icann.org/registrants

[6] Internet Systems Consortium, I. BIND. 2020. Available from: https://www.isc.org/bind/

[7] CZ.NIC. Knot DNS. 2020. Available from: https://www.knot-dns.cz/

[8] CZ.NIC. Knot Resolver. 2020. Available from: https://www.knot-resolver.cz/

[9] CZ.NIC. Projekty pro odbornou veřejnost. 2020. Available from: https://www.nic.cz/page/2049/projekty-pro-odbornou-verejnost/

[10] ICANN. ICANN Managed Root Server. 2020. Available from: https://www.dns.icann.org/imrs/

[11] Hilton, S. Dyn Analysis Summary Of Friday October 21 Attack. 2016. Available from: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[12] Danchev, D. Anonymous launches 'Operation Global Blackout', aims to DDoS the Root Internet servers. 2012. Available from: https://www.zdnet.com/article/anonymous-launches-operation-global-blackout-aims-to-ddos-the-root-internet-servers/

[13] Allan Liska, G. S. *DNS Security. Defending the Domain Name System.* Syngress, first edition, 2016, ISBN 0128033061,978-0-12-803306-7,9780128033395,0128033398.

[14] Cejka, T.; Rosa, Z.; et al. Stream-wise detection of surreptitious traffic over DNS. In *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2014, pp. 300–304.

[15] Son, S.; Shmatikov, V. The Hitchhiker's Guide to DNS Cache Poisoning. In *Security and Privacy in Communication Networks*, edited by S. Jajodia; J. Zhou, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-16161-2, pp. 466–483.

[16] Hoffman, P.; Sullivan, A.; et al. DNS Terminology. RFC 7719, RFC Editor, December 2015.

[17] Wright, C. Understanding Kaminsky's DNS Bug. 2008. Available from: https://www.linuxjournal.com/content/understanding-kaminskys-dns-bug

[18] Atkins, D.; Austein, R. Threat Analysis of the Domain Name System (DNS). RFC 3833, RFC Editor, August 2004. Available from: https://tools.ietf.org/html/rfc3833

[19] Wellington, B. Secure Domain Name System (DNS) Dynamic Update. RFC 3007, RFC Editor, November 2000.

[20] Hu, Z.; Zhu, L.; et al. Specification for DNS over Transport Layer Security (TLS). RFC 7858, RFC Editor, May 2016.

[21] Hoffman, P.; McManus, P. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018.

[22] Vale, M. Google Public DNS now supports DNS-over-TLS. 2019. Available from: https://security.googleblog.com/2019/01/google-public-dns-now-supports-dns-over.html

[23] Cloudflare. DNS over TLS. Available from: https://developers.cloudflare.com/1.1.1.1/dns-over-tls/

[24] DNSCurve: Usable security for DNS. Available from: https://dnscurve.org/

[25] CZ.NIC. DNSCurve FAQ. 2020. Available from: https://dnscurve.io/faq/

[26] Arends, R.; Austein, R.; et al. DNS Security Introduction and Requirements. RFC 4033, RFC Editor, March 2005. Available from: https://tools.ietf.org/html/rfc4033

[27] Names, I. C. F. A.; Numbers. Root DNSSEC Status Update, 2010-07-14. Available from: https://www.root-dnssec.org/2010/07/16/status-update-2010-07-16/

[28] Arends, R.; Austein, R.; et al. Resource Records for the DNS Security Extensions. RFC 4034, RFC Editor, March 2005.

[29] Bortzmeyer, S. DNS Privacy Considerations. RFC 7626, RFC Editor, August 2015.

[30] Finch, T. SHA-1 chosen prefix collisions and DNSSEC. 2020. Available from: https://blog.apnic.net/2020/01/17/sha-1-chosen-prefix-collisions-and-dnssec/

[31] Consortium, I. S. Automatic DNSSEC Zone Signing Key rollover explained. 2018. Available from: https://kb.isc.org/docs/aa-00822

[32] DNS, K. Knot DNS 2.6.9 documentation. 2018. Available from: https://www.knot-dns.cz/docs/2.6/html/operation.html#dnssec-key-rollovers

[33] of the CTO, I. O. Review of the 2018 DNSSEC KSK Rollover. 2019. Available from: https://www.icann.org/en/system/files/files/review-2018-dnssec-ksk-rollover-04mar19-en.pdf

[34] Barker, E. B.; Dang, Q. H. Recommendation for Key Management Part 3:. Technical report, National Institute of Standards and Technology, Jan. 2015, doi:10.6028/nist.sp.800-57pt3r1. Available from: https://doi.org/10.6028/nist.sp.800-57pt3r1

[35] Hoffman, P.; Wijngaards, W. Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC. RFC 6605, RFC Editor, April 2012.

[36] Barker, E. Recommendation for key management Part 1:. Technical report, National Institute of Standards and Technology, May 2020, doi: 10.6028/nist.sp.800-57pt1r5. Available from: https://doi.org/10.6028/nist.sp.800-57pt1r5

[37] Overeinder, B. Bringing DNS security and privacy to the end user. 2018. Available from: https://blog.apnic.net/2018/02/07/bringing-dns-security-privacy-end-user/

[38] Huston, G. DNSSEC 'and' DNS over TLS. 2018. Available from: https://blog.apnic.net/2018/08/20/dnssec-and-dns-over-tls/

[39] Hubert, B. Opinion: Centralized DoH is bad for privacy, in 2019 and beyond. 2019. Available from: https://blog.apnic.net/2019/10/03/opinion-centralized-doh-is-bad-for-privacy-in-2019-and-beyond/

[40] Martin, A. J. Google's Chrome browser plans 'risk undermining fight against online child abuse', govt warned. 2019. Available from: https://news.sky.com/story/googles-chrome-browser-plans-risk-undermining-fight-against-online-child-abuse-govt-warned-11734166

[41] Jacobs, L. Waiting for goDoH, or DNS exfiltration over DNS over HTTPS (DoH) with godoh. 2018. Available from: https://sensepost.com/blog/2018/waiting-for-godoh/

[42] Kaminsky, D. DNSSEC Interlude 2: DJB@CCC. 2011. Available from: https://dankaminsky.com/2011/01/05/djb-ccc/

[43] Surý, O. DNSCurve – alternativní návrh k DNSSECu. 2008. Available from: https://blog.nic.cz/2008/09/01/dnscurve-alternativni-navrh-k-dnssecu/

[44] Differences between DNSCurve and DNSSEC. Available from: https://dnscurve.io/faq/differences-between-dnscurve-and-dnssec.html

[45] Durumeric, Z.; Kasten, J.; et al. Analysis of the HTTPS Certificate Ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, New York, NY, USA: Association for Computing Machinery, 2013, ISBN 9781450319539, p. 291–304, doi: 10.1145/2504730.2504755. Available from: https://doi.org/10.1145/2504730.2504755

[46] Hoffman, P.; Schlyter, J. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, RFC Editor, August 2012.

[47] Dukhovni, V.; Hardaker, W. SMTP Security via Opportunistic DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS). RFC 7672, RFC Editor, October 2015.

[48] Huque, S. Whither DANE? 2019. Available from: https://blog.apnic.net/2019/07/05/whither-dane/

[49] Kumari, W.; Gudmundsson, O.; et al. Automating DNSSEC Delegation Trust Maintenance. RFC 7344, RFC Editor, September 2014. Available from: https://tools.ietf.org/html/rfc7344

[50] Gudmundsson, O.; Wouters, P. Managing DS Records from the Parent via CDS/CDNSKEY. RFC 8078, RFC Editor, March 2017. Available from: https://tools.ietf.org/html/rfc8078

[51] SWITCH. Automated DNSSEC Provisioning. 2018. Available from: https://www.nic.ch/export/shared/.content/files/SWITCH_CDS_Manual_en.pdf

[52] SWITCH. CDS Status Check. 2018. Available from: https://www.nic.ch/security/cds/index.html

[53] Eden, A. Announcing CDS/CDNSKEY Support. 2019. Available from: https://blog.dnsimple.com/2019/02/cds_cdnskey/

[54] Isasi, S.; Shrestha, V. Expanding DNSSEC Adoption. 2018. Available from: https://blog.cloudflare.com/automatically-provision-and-maintain-dnssec/

[55] ICANN. TLD DNSSEC Report (2020-05-21 00:02:49). 2020. Available from: http://stats.research.icann.org/dns/tld_report/

[56] Chung, T.; van Rijswijk-Deij, R.; et al. Understanding the Role of Registrars in DNSSEC Deployment. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, New York, NY, USA: Association for Computing Machinery, 2017, ISBN 9781450351188, p. 369–383, doi: 10.1145/3131365.3131373. Available from: https://doi.org/10.1145/3131365.3131373

Appendix A

Acronyms

API Application Programming Interface

CDNSKEY Child DNSKEY

CDS Child DS RR

CLI Command Line

DDoS Distributed Denial of Service

DNS Domain Name System

DNSKEY DNS KEY RR

DNSSEC DNS Security Extensions

DS Delegation Signer RR

gRPC google RPC

FQDN Fully Qualified Domain Name

IDL Interface Definition Language

KSK Key Signing Key

RFC Requet For Comments

RPC Remote Procedure Call

RR Resource Record

SQL Structured Query Language

UID Unique Identifier

ZSK Zone Signing Key

Appendix B

Contents of enclosed CD

- `DP_Shchavleva_Marina_2020`
  - `readme.txt` ..... the file with CD contents description
  - `akm-multi-scanner` ..... AKM Multi Scanner sources
    - `protos` ..... definitions of protobuf messages and gRPC interfaces
    - `schema` ..... database schema scripts
    - `src` ..... implementation sources in C++
    - `Makefile` ..... compilation script
  - `thesis` ..... the directory of LaTeX source codes of the thesis
    - `sources` ..... LaTeX .tex files
      - `Makefile` ..... compilation script
    - `thesis.pdf` ..... the thesis text in PDF format