ANALIZA PROTOKOLOV ZA ELEKTRONSKO POŠTO IN ZASNOVA NOVEGA PROTOKOLA

RAČUNALNIŠTVO IN INFORMATIKA

RAZISKOVALNA NALOGA

Mitja Ševerkar
9. razred

Blaž Abram, prof. mat. in rač.

2021/2022

OSNOVNA ŠOLA VODMAT
Potrčeva ulica 1, 1000 Ljubljana

KAZALO

KAZALO i
KAZALO SLIK ii
KLJUČNA DOKUMENTACIJSKA INFORMACIJA iii
KEY WORDS DOCUMENTATION iv
UPORABLJENE KRATICE v
ZAHVALE vi
UVOD 1
INTERNETNI PROTOKOL 2
DELOVANJE ELEKTRONSKE POŠTE 3
SMTP 4
POP 7
IMAP 8
JMAP 9
Internet Mail 2000 10
IMPLEMENTACIJE E-POŠTNIH PROTOKOLOV IN E-POŠTNI KLIENTI 12
Exchange 12
Apache James 12
Thunderbird 12
Roundcube 12
MOJ PROTOKOL 13
PREDNOSTI IN SLABOSTI MOJEGA PROTOKOLA 15
MAIL SANDBOXING 17
POTEK POŠILJANJA SPOROČILA 18
VARNOST PROTOKOLA 23
ODGOVARJANJE NA SPOROČILA 25
POSREDOVANJE SPOROČIL 27
PRIPONKE IN VARNOST TEH 31
MOJA IMPLEMENTACIJA 33
ZAKLJUČEK 34
VIRI, LITERATURA IN REFERENCE vii

KAZALO SLIK

Slika 1 - Prenos elektronske pošte po internetu – Lastno delo; Narejeno 7.12.2021 s programom Dia 3
Slika 2 - Primer komunikacije v SMTP protokolu – Lastno delo; Narejeno 7.12.2021 5
Slika 3 - Primer komunikacije v POP3 protokolu - Lastno delo; Narejeno 26.12.2021 7
Slika 4 - Primer komunikacije v IMAP protokolu - Lastno delo; Narejeno 26.12.2021 8
Slika 5 - Poenostavljen diagram organizacije pri Internet Mail 2000 protokolu – Lastno delo; Narejeno 9.1.2022 s programom Dia..... 10
Slika 6 - Primer HTML-ja – Lastno delo; Narejeno 9.1.2022 14
Slika 7 - Primer Markdowna – Lastno delo; Narejeno 9.1.2022 14
Slika 8 - Primer prevedenega HTML-ja v Brave brskalniku – Lastno delo; Narejeno 9.1.202214
Slika 9 - Izolacija sporočil pri SMTP2 protokolu - Lastno delo; Narejeno 12.2.2022 s programom Dia 17
Slika 10 - Primer JWT ključa na spletni strani jwt.io; Lastno delo – Narejeno 16.1.2022 18
Slika 11 - Primer zahtevka za ustvarjanje osnutka; Lastno delo - Narejeno 16.1.2022..... 19
Slika 12 - Primer odziva za ustvarjanje osnutka; Lastno delo - Ustvarjeno v Brave (Chromium) orodjih za razvijalce (zavihek Omrežje) - Narejeno 14.3.2022 19
Slika 13 - Primer zahtevka za posodabljanje osnutka; Lastno delo - Narejeno 16.1.2022..... 19
Slika 14 - Primer zahtevka za pošiljanje sporočila na pošiljateljevi strani; Lastno delo - Narejeno 23.1.2022 20
Slika 15 – Zahtevek na prejemnikov strežnik ob pošiljanju sporočila – Lastno delo; Narejeno 15.1.2022 20
Slika 16 - Potek pošiljanja sporočila na SMTP2 protokolu - Lastno delo; Narejeno 24.1.2022 s programom Dia 22
Slika 17 - Zahtevek MVP protokola – Lastno delo; Narejeno 15.1.2022 23
Slika 18 - Potek SMTP2 MVP protokola – Lastno delo; Narejeno 15.1.2022 s programom Dia 24
Slika 19 - Poenostavljen prikaz odgovarjanja na sporočila - Lastno delo; Narejeno 13.2.2022 s programom Dia..... 26
Slika 20 - Posredovanje sporočila - Lastno delo; Narejeno 30.1.2022..... 27
Slika 21 - Posredovanje sporočil v SMTP2 protokolu - Lastno delo; Narejeno 31.1.2022 s programom Dia 28
Slika 22 - Priponke pri ustvarjanju posredovanega sporočila - Lastno delo; Narejeno 1.2.2022 s programom Dia..... 29
Slika 23 - Postopek prejema priponke pri SMTP2 protokolu - Lastno delo; Narejeno 1.2.2022 s programom Dia..... 29
Slika 24 - Potek prenosa priponk pri SMTP2 protokolu - Lastno delo; Narejeno 31.1.2022 s programom Dia 32

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

ŠD Osnovna šola Vodmat, 2021/2022
KG E-pošta, zasnova novega protokola za elektronsko pošto, varnost protokolov
AV Mitja Ševerkar, 9. razred
SA Blaž Abram – Osnovna šola Vodmat
KZ SI-1000 Ljubljana, Potrčeva ul. 1
ZA Osnovna šola Vodmat
LI 2022
IN ANALIZA PROTOKOLOV ZA ELEKTRONSKO POŠTO IN ZASNOVA NOVEGA PROTOKOLA
TD Raziskovalna naloga
OP 34 strani, 24 slik, 26 referenc
IJ sl
JI sl/en

IZVLEČEK:

Elektronska pošta (krajše e-pošta) predstavlja zelo pomemben del našega življenja. Dandanes delamo vse preko elektronske pošte, pa naj si bo to enostavno preusmerjanje sporočil oziroma priponk, načrtovanje srečanj ali pa pošiljanje sporočil raznim uradnim ustanovam. Protokoli, ki jih danes uporabljamo za pošiljanje elektronske pošte po internetu, so v uporabi že zelo dolgo časa, zato bi bilo morda vredno razmisliti o posodobitvi oz. nadgradnji le-teh. Obstoječi protokoli imajo tudi nekaj slabosti, izpostavil pa bi predvsem to, da poslanega sporočila ni mogoče izbrisati ali ga urediti, potem ko je enkrat že poslano. Te slabosti še nihče ni popravil, zato sem s to raziskovalno nalogo želel to raziskati in hkrati tudi popraviti s popolnoma novim protokolom, ki bi bil zasnovan za učinkovitost in preprostost.

KEY WORDS DOCUMENTATION

ND Osnovna šola Vodmat, 2021/2022
CX E-mail, designing new protocol for electronic mail, safety of e-mail protocols
AU Mitja Ševerkar, 9th grade
AA Blaž Abram – Osnovna šola Vodmat
PP SI-1000 Ljubljana, Potrčeva ul. 1
PB Osnovna šola Vodmat
PY 2022
TI ANALYSIS OF PROTOCOLS FOR ELECTRONIC MAIL AND DESIGNING NEW PROTOCOL
DT Research work
NO 34 pages, 24 pictures, 26 references
LA sl
AL sl/en

ABSTRACT:

Electronic mail (shorter e-mail) represents a very important aspect of our lives. We are doing everything through electronic mail, it's either forwarding messages and/or attachments, planning of meetings or sending messages to different official institutions. Protocols, which we use today for sending e-mails around the internet are in use for a very long time, thus it might be worth to think about upgrading or replacing them. Current e-mail protocols have some downsides, from which I'd like to expose that you cannot delete nor edit a message once it has been sent. These downsides weren't fixed by anyone, so with this research work I wanted to research it and at the same time fix it, with a whole new e-mail protocol, which would be designed for efficiency and simplicity.

UPORABLJENE KRATICE

- SMTP – Simple Mail Transfer Protocol
- POP – Post Office Protocol
- IMAP – Internet Message Access Protocol
- SMTP2 - Simple Mail Transfer Protocol verzija 2 – ime mojega protokola
- URL – Uniform Resource Locator
- URI – Uniform Resource Identifier
- IETF – Internet Engineering Task Force
- RFC – Request For Comments
- HTTP – HyperText Transfer Protocol
- HTTPS – HyperText Transfer Protocol Secure – kriptografsko varnejša različica HTTP-ja
- ARPANET - Advanced Research Projects Agency Network – predhodnik današnjega interneta
- IM2000 – Internet Mail 2000
- JSON – JavaScript Object Notation – tekstovni format
- ASCII – American Standard Code for Information Interchange
- ID – Identifier (slovensko Identifikator)
- MVP – Message Validation Protocol – protokol uporabljen v SMTP2 za validacijo
- DNS – Domain Name Server

ZAHVALE

Zahvalil bi se rad vsem, ki so posredno ali neposredno prispevali k tej raziskovalni nalogi.

Posebne zahvale gredo Blažu Abramu, mojemu mentorju, za konstantno podporo.

Najlepša hvala tudi Luciji Hočevar, profesorici slovenščine in angleščine, za lektoriranje te raziskovalne naloge.

Zahvale gredo tudi mojim staršem, ki so me med to raziskovalno nalogo podpirali.

Zahvalil bi se tudi raznim podjetjem in organizacijam, ki so neposredno prispevala k tej raziskovalni nalogi. To so:

- GitHub – storitev za gostenje kode mi je olajšala delo in organizacijo pri gradnji lastnega protokola
- JetBrains – njihova razvijalska orodja so izjemno dobra in so mi izjemno olajšala delo pri gradnji lastnega protokola.
- ARNES, GÉANT in pa GRNET za strežniško storitev
- Let's Encrypt – za SSL certifikate za domene
- Freenom – za brezplačno domeno

UVOD

Nekega dne sem razmišljal o tem, kako je internet zgrajen na stari infrastrukturi, kako to zamenjujemo s sodobnejšo tehnologijo, HTTP smo namreč zamenjali z HTTPS, ARPANET smo zamenjali z internetom, stare programske jezike smo zamenjali z novimi in lahko bi še našteval. Uvidel sem, da je ena stvar, ki je enaka že vse od ARPANET-a, in to je elektronska pošta.

Elektronska pošta je zelo uporabna stvar, a kaj, ko vse temelji na stari infrastrukturi. SMTP protokol ni dosti razširljiv, uporaben za današnje čase. Časi so drugačni od tistih izpred 30 let. Včasih si lahko brez problema čakal na prenos neke stvari čez noč po slabih internetnih povezavah, medtem ko danes vse poteka tako hitro, da postanemo zelo nestrpni, če ne dobimo informacij takoj. Zato se včasih zgodi tudi kakšna napaka na elektronskih sporočilih, a tega ne moremo popraviti, razen če še enkrat pošljemo sporočilo s popravljeno napako.

Zato sem se odločil odpraviti te napake z novim protokolom, ki bi gradil na prednostih in slabostih starih, preprostosti in sodobni infrastrukturi.

V prvem delu te raziskovalne naloge vas bom popeljal po današnjih protokolih za elektronsko pošto, predstavil njihove prednosti in slabosti, v drugem delu pa bom opisal svoj novi protokol, ki bi morda lahko nadomestil stare, ter vse njegove prednosti in tudi slabosti.

INTERNETNI PROTOKOL

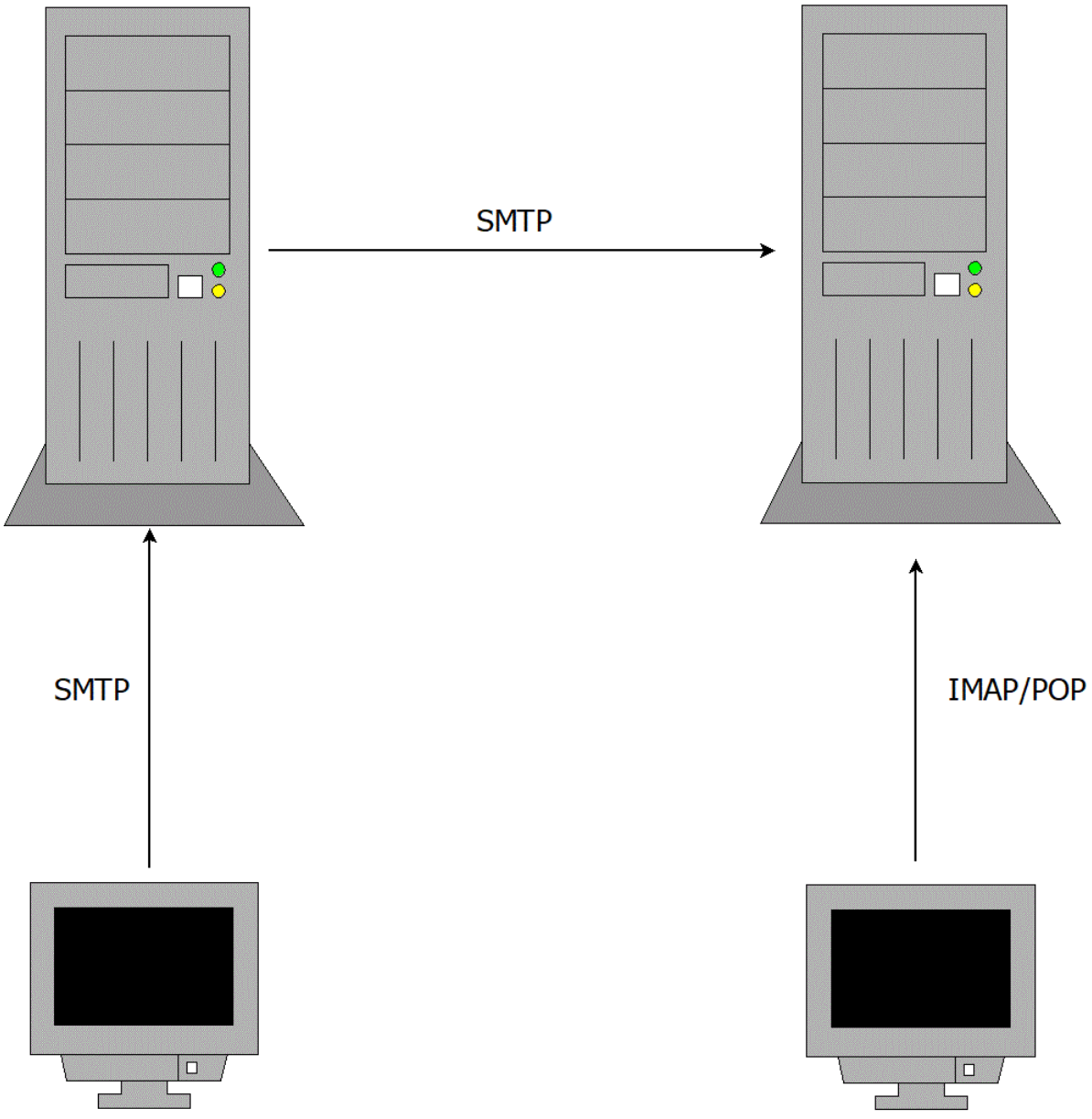
Po definiciji v Slovarju slovenskega knjižnega jezika 2 je protokol zbirka pravil za izmenjavo sporočil med elektronskimi napravami. Protokoli se uporabljajo v celotnem računalništvu – računalništvo je zgrajeno na protokolih. Protokoli so izjemno pomembni, da stvari lahko tečejo tako, kot bi morale. Protokoli so po navadi standardizirani, le tako so namreč sploh smiselni. Če ne bi bili standardizirani, bi imeli cel kup protokolov, nihče pa ne bi vedel, katerega uporabljati. Internet bi bil v tem primeru popolna zmeda.

Protokole sprejema neprofitna organizacija po imenu IETF (daljše tudi *Internet Engineering Task Force* (1)). Sprejemajo jih v obliki tako imenovanih RFC-jev (angl. *Request for Comments*). Ko nekdo odda pobudo za nov protokol, se tej pobudi, če in ko postane RFC, dodeli specifična številka. Niso pa vsi RFC-ji internetni standardi, namreč RFC-ji se lahko tudi zavrnejo iz več razlogov, kot piše v RFC1796 (2). Ko postane RFC standard, se mu pripne predpona STD (kar stoji za *standard*).

Ob omembi kakršnegakoli protokola v nadaljevanju raziskovalne naloge bom opisu dodal tudi številko protokola in povezavo do specifikacije protokola.

DELOVANJE ELEKTRONSKE POŠTE

Elektronska pošta je sestavljena iz protokola, poimenovanega SMTP, kar pomeni *Simple Mail Transfer Protocol*, in protokolov, kot sta POP (*Post Office Protocol*) in IMAP (*Internet Message Access Protocol*). SMTP je standardiziran, njegova vloga pri elektronski pošti pa je, da pošilja sporočila, torej komunicira med strežnikoma, POP in IMAP pa sprejemata elektronsko pošto s strežnika, kjer gre torej za komunikacijo med strežnikom in osebnim računalnikom oziroma e-poštnim odjemalcem na njem.



Slika 1 - Prenos elektronske pošte po internetu – Lastno delo; Narejeno 7.12.2021 s programom Dia

SMTP

SMTP (*Simple Mail Transfer Protocol*) je protokol za pošiljanje elektronske pošte. Prva izdaja je bila specificirana v RFC-ju 788 (3). Na začetku svoje poti v novembru 1981 je SMTP imel nekaj omejitev, vendar je bila marsikatera do danes odpravljena. Primer tega je popolna ASCII implementacija. Na začetku svoje poti je SMTP prišel s specifikacijo, da uporablja samo ASCII tekst. ASCII pomeni *American Standard Code for Information Interchange* in vsebuje samo črke angleške abecede in nekaj simbolov. To je bilo zelo omejujoče za uporabnike iz drugih držav, saj niso mogli uporabljati svojih črk. Prav tako ni bilo možno pripeti kakršnih koli ne-tekstovnih/binarnih datotek.

Prišlo je veliko novih RFC-jev, ki izboljšujejo SMTP. Ta SMTP, ki ga uporabljamo danes, se imenuje "*Modern SMTP*" oziroma po slovensko Moderni SMTP.

V moderni SMTP so prišle naslednje specifikacije, ki jih je vredno omeniti:

- Mehanizem za odkrivanje razširitev (angl. *Extension discovery mechanism*) je specificiran v RFC-ju 1869 (4). Ta razširitev omogoča, da klient na začetku povezave namesto HELO pošlje EHLO. Na HELO pošlje strežnik nazaj samo pozdrav, medtem ko na EHLO pošlje vse dodatne ukaze in razširitve.
- Binarni prenos podatkov (angl. *Binary data transfer*) je specificiran v RFC-ju 1652 (5). Omogoča to, da namesto 7-bitnega ASCII črkovnega sistema (v katerem so samo črke angleške abecede, številke in nekaj karakterjev) uporabljamo osem-bitni UTF-8, ki je veliko bolj širok, saj namreč podpira tudi črke iz ostalih abeced, več simbolov in emotikone.

```
S: <strežnik čaka na vratih 25 na odjemalca>
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<mitja@example.org>
S: 250 Ok
C: RCPT TO:<blaz@example.com>
S: 250 Ok
C: RCPT TO:<marko@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Mitja" <mitja@example.org>
C: To: "Blaž" <blaz@example.com>
C: Cc: marko@example.com
C: Date: Tue, 07 Dec 2021 18:04:16 +0100
C: Subject: Testno sporočilo
C:
C: Pozdravljeni!
C: To je testno sporočilo, poslano s SMTP protokolom.
C: Lep pozdrav,
C: Mitja
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
S: <strežnik zapre povezavo>
```

Slika 2 - Primer komunikacije v SMTP protokolu – Lastno delo; Narejeno 7.12.2021

Sporočilo, preneseno preko SMTP protokola, mora vsebovati naslednja polja, ki so specificirana tukaj (6):

1. Od (angl. *From*); kdo je pošiljatelj sporočila (po izbiri avtorja sporočila, čeprav se pri odjemalcih (klientih) navadno tega ne da spreminjati).
2. Za (angl. *To*); primarni prejemniki sporočila. Teh je lahko več.
3. Kp (stoji za Kopija; angl. *Carbon Copy - Cc*) – neobvezno polje; drugi prejemniki tega sporočila.
4. Skp (stoji za Skrita kopija; angl. *Blind Carbon Copy - Bcc*) – neobvezno polje; skriti prejemniki sporočila. Prejemnik v tem primeru ne vidi, komu (iz Bcc vrstice) je bilo to sporočilo še poslano.
5. Zadeva (angl. *Subject*) – polje je lahko prazno; zadeva sporočila, kot nek kratek naslov tega sporočila.
6. Datum (angl. *Date*) – ni obvezno polje; Čas ko je bilo sporočilo poslano od pošiljateljevega klienta. To polje v glavi sporočila nastavlja klient (odjemalec) in ne strežnik. Čas je lokalen (torej ni UTC), je pa napisana časovna cona. Zapisano je v formatu EEE, dd MMM yyyy HH:mm:ss K (7).
   Primer je Tue, 07 Dec 2021 18:04:16 +0100, torej poslano je bilo v torek, 7. decembra 2021, ob 18:04 in 16 sekundah, in sicer z zamikom 1 ure od UTC-ja.

7. Identifikacija sporočila (angl. *Message-ID*) – edinstvena identifikacijska številka sporočila. Mora biti edinstvena po celem svetu. Navadno je v formatu <identifikacijska številka>@<domena prejemnika>. Primer je *B27397-0100000@arnes.si*.
8. Tip vsebine sporočila (angl. *Content-Type*) – je standardno HTTP polje (8). Polje pove strežniku oziroma klientu, kateri tip sporočila bo prejel. Lahko prejme JSON, lahko prejme navadno besedilo (sicer najbolj uporabljeno), lahko pa prejme neko avdio datoteko. V to polje v glavi sporočila se napiše eden izmed t. i. Medijskih tipov (angl. *Media-Type*). Te tipe standardizira IANA (*Internet Assigned Numbers Authority*). IANA na splošno skrbi za vse samoumevne lastnosti interneta, kot so domene in pa DNS (*Domain Name Server*) strežniki. Medijske tipe lahko pogledamo na njihovi uradni spletni strani (9), zapisujejo pa se v formatu <zvrst tipa>/<natančen tip>. Primer medijskega tipa za JSON je *application/json*.
9. Odnaroči (angl. *List-Unsubscribe*) – polje, ki se pošlje, ko se elektronska pošta pošilja masovno, navadno v sklopu novic ali naročnine. V tem polju se pošlje URL do spletne strani za odjavo od naročnine. To pošljejo razni servisi, kot so Mailchimp ali pa Mailerlite.
10. Vsebina sporočila (angl. *Body*) – ni posebno polje, ampak je del sporočila od zadnjega polja v glavi sporočila pa vse do pike na koncu vsebine.

Celoten prenos podatkov je bil včasih v ASCII (7-bitnem) znakovnem sistemu, danes pa je (zapisan) v 8-bitnem (UTF-8) znakovnem sistemu, zahvaljujoč binarnemu prenosu podatkov.

SMTP-ju so dodali tudi kriptografijo, zato naj bi bil varen podobno kot HTTPS, čeprav so pred nekaj leti odkrili nekaj varnostnih lukenj (10).

Sporočila se preverjajo s protokoli, kot so DKIM (*DomainKeys Identified Mail*), SPF (*Sender Policy Framework*) in pa DMARC (*Domain-based Message Authentication, Reporting and Conformance*).

POP

POP oziroma Post Office Protocol je protokol za pridobivanje sporočil iz strežnika. Je sicer veliko manj uporabljen kot IMAP. Zadnja verzija tega protokola je POP3, kljub temu da so poskusili narediti tudi POP4, vendar jim to ni uspelo. POP za razliko od IMAP-a izbriše sporočilo iz strežnika takoj, ko ga klient prenese na svoj računalnik in od takrat ga je možno videti le na tej napravi, kar je izjemno neuporabno v današnjih časih.

Vsi ukazi v POP-u so dolgi natanko 4 znake. To ga naredi izjemno učinkovitega, kar se tiče prejemanja in razčlenjevanja sporočil, a malo manj berljivega (primer: »Retrieve« (slovensko *prejmi*) je RETR). Danes imamo seveda napredno tehnologijo razdiranja besedila na manjše koščke na določenih znakih. Temu rečemo prepolavljanje (angl. *splitting*).

```
S: <čaka na povezave na TCP vratih 110>
C: <odpre povezavo>
S:    +OK POP3 server ready <1896.697170952@ponudnik.com>
C:    APOP mitja c4c9334bac560ecc979e58001b3e22fb
S:    +OK mitja's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <the POP3 server sends message 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <the POP3 server sends message 2>
S:    .
C:    DELE 2
S:    +OK message 2 deleted
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C: <zapre povezavo>
S: <čaka na naslednjo povezavo>
```

Slika 3 - Primer komunikacije v POP3 protokolu - Lastno delo; Narejeno 26.12.2021

IMAP

IMAP je malo novejši protokol za pridobivanje elektronske pošte. Narejen je bil, da bi konkuriral POP protokolu, katerega je tudi nadvladal. POP je bolj ali manj uporabljen samo za t. i. »legacy« podporo (se ne dodaja več funkcij, se samo vzdržuje). IMAP pusti sporočilo na strežniku, torej lahko več naprav prenese to sporočilo s strežnika, pa se ne bo izbrisalo, dokler uporabnik izrecno ne izbriše tega sporočila.

V IMAP-u ukazi nimajo fiksne dolžine.

```
C: <odpre povezavo>
S:   * OK IMAP4rev1 Service Ready
C:   a001 login mitja geslo
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * 2 RECENT
S:   * OK [UNSEEN 17] Message 17 is the first unseen message
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   a002 OK [READ-WRITE] SELECT completed
C:   a003 fetch 12 full
S:   * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -
0700"
      RFC822.SIZE 4286 ENVELOPE ("Sun, 26 Dec 2021 14:37:43 -0700
(CET)"
      "IMAP test"
      (("Mitja Severkar" NIL "mitja.severkar" "guest.arnes.si"))
      (("Mitja Severkar" NIL "mitja.severkar" "guest.arnes.si"))
      (("Mitja Severkar" NIL "mitja.severkar" "guest.arnes.si"))
      ((NIL NIL "mitja.severkar" "osvodmat.si"))
      "<B27397-0100000@arnes.si>")
      BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028
      92))
S:   a003 OK FETCH completed
C:   a004 fetch 12 body[header]
S:   * 12 FETCH (BODY[HEADER] {342}
S:   Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:   From: Mitja Severkar <mitja.severkar@guest.arnes.si>
S:   Subject: IMAP test
S:   To: mitja.severkar@osvodmat.si
S:   Message-Id: <B27397-0100000@arnes.si>
S:   MIME-Version: 1.0
S:   Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S:   )
S:   a004 OK FETCH completed
C    a005 store 12 +flags \deleted
S:   * 12 FETCH (FLAGS (\Seen \Deleted))
S:   a005 OK +FLAGS completed
C:   a006 logout
S:   * BYE IMAP4rev1 server terminating connection
S:   a006 OK LOGOUT completed
```

Slika 4 - Primer komunikacije v IMAP protokolu - Lastno delo; Narejeno 26.12.2021

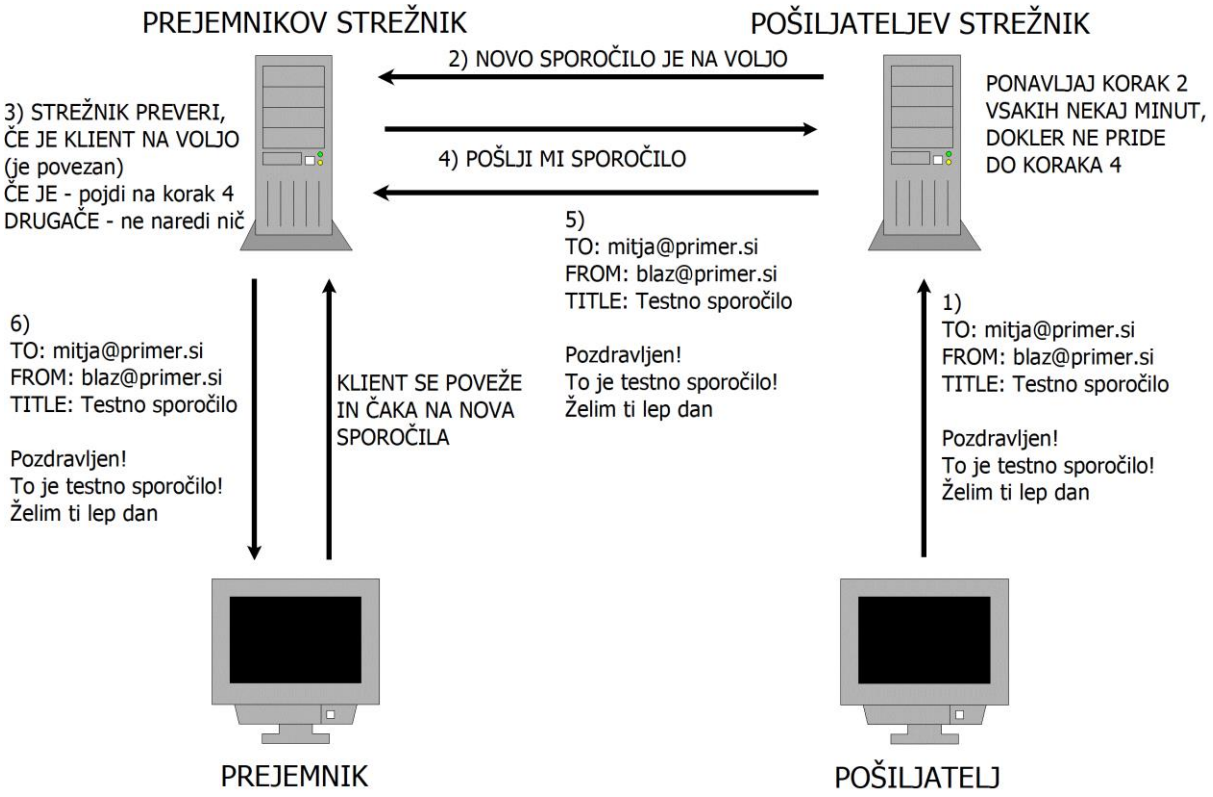
JMAP

IMAP je zelo zastarel, počasen in ni preprost. To ga ne naredi idealnega za današnji svet. Razvijalci so to opazili in so ustvarili protokol, ki naj bi konkuriral oz. zamenjal IMAP. Vsebuje funkcije, ki so izjemno priročne, je hkrati zelo hiter in zelo »lahek«. Implementirali so ga z JSON-om (kar je krajše za *JavaScript Object Notation*), ki je dandanes najbolj uporabljen format za komuniciranje med aplikacijami. Hkrati uporabljajo tudi HTTPS, ki je najbolj uporabljen protokol za prenos podatkov preko interneta. HTTPS je tudi izjemno varen sam po sebi in zato je tudi ta protokol zelo varen pred t. i. »man in middle« napadom. »Man in middle« napad je, ko napadalec prestreže povezavo med klientom in strežnikom in tako dobi dostop do občutljivih podatkov, ki jih klient pošilja strežniku.

Internet Mail 2000

Daniel J. Bernstein je že leta 2000 predlagal nov protokol, ki naj bi zamenjal SMTP. Poimenoval ga je Internet Mail 2000, na kratko IM2000. Ta protokol je osnoval na drugačen način, kot je bilo tipično za SMTP, in sicer tako, da bi bil izjemno fleksibilen. To lahko dosežemo le tako, da gostimo sporočila na pošiljateljevem strežniku in ne na prejemnikovem, kot je to takrat delal SMTP. To bi mu omogočalo veliko fleksibilnost, tudi kasneje, ko bi delal na novejši različici. Žal pa ta protokol nikoli ni bil na široko uporabljen.

Internet Mail 2000 je temeljil na principu, da pošiljateljev strežnik ob poslanem sporočilu na prejemnikov strežnik na vsakih nekaj minut pošlje obvestilo, da je na voljo novo sporočilo. Ta postopek se ponavlja, dokler prejemnik tudi potrjeno ne vidi sporočila (glej sliko 5).



Slika 5 - Poenostavljen diagram organizacije pri Internet Mail 2000 protokolu – Lastno delo; Narejeno 9.1.2022 s programom Dia

To je sicer tudi za današnje računalnike kar velik zalogaj, še posebej, če imamo več 1000 sporočil, ki se pošiljajo preko interneta in tudi več neaktivnih uporabnikov oziroma zapuščenih računov, ki jih uporabnik ne preverja več redno. Hkrati pa ta način pomeni, da noben strežnik teoretično ne potrebuje zbirke podatkov, ampak bi jo vseeno uporabil, da bi se shranila poslana sporočila. Ker mora pošiljateljev strežnik vseeno vedeti, kaj je uporabnik poslal, da bi znal sporočiti, kaj je bila vsebina sporočila, takrat, ko bi prejemnikov strežnik to zahteval.

Daniel J. Bernstein si je zastavil kar nekaj vprašanj glede svojega protokola (prirejeno po (11) - Razdelek »Some questions«):

1. Kako naj bodo prejemniki identificirani? Kako bo pošiljateljev ISP (*Internet Service Provider* – slovensko *Ponudnik internetnih storitev*) našel prejemnikovega ISP-ja?
2. Kako naj bodo pošiljatelji identificirani? Kako bo prejemnik našel pošiljateljevega ISP-ja? Prejemniki bodo sčasoma dodajali pošiljatelje na »varno listo«, kar bi omogočalo večje zaupanje v določen strežnik.
3. Kako naj bodo sporočila identificirana in prenesena iz pošiljateljevega strežnika? Sporočila bi se lahko prenašala preko http-ja, a NFS/FSP-stiliran protokol, zgrajen na UDP-ju, bi bil veliko bolj odporen na napad zavračanja storitve (DDoS – *Denial Of Service Attack*, slovensko *Napad zavračanja storitve*).
4. Kako naj bodo notifikacije, sporočila in potrdila zaščitena pred vohunstvom in sabotažo? Diffie-Hellman avtentikacija se zdi bolj primerna kot javen in zasebni ključ za tako ali tako zasebna sporočila; je prav tako veliko hitrejša in prav tako priročna.
5. Kako naj pošiljatelj ustvari novo sporočilo?
6. Kako naj prejemnik prenese notifikacije o sporočilih?
7. Kakšen format naj sporočila uporabljajo?

Ta vprašanja so zelo dobro zastavljena, zato bom tudi odgovoril nanje glede na moj protokol in ne na njegov IM2000.

1. Prejemniki in pošiljatelji bodo identificirani s standardnim elektronskim naslovom, to je ime@domena.si
   Identifikacija domen je standardna procedura v HTTP/HTTPS protokolu in poteka preko DNS strežnikov.
2. Odgovor je isti kot pri odgovoru na vprašanje 1.
   Glede varne liste pa bi rekel, da to spet ni najbolj varno, ker se lahko gesla določenega uporabniškega računa razkrijejo »hekerjem« in ta zlorabi zaupanje na varni listi.
3. Sporočila bodo identificirana z enoličnimi identifikatorji, ki bodo različni tako na prejemnikovemu kot tudi pošiljateljevem strežniku.
   Prenesena bodo preko HTTP/HTTPS protokola.
4. Če bo strežnik imel HTTPS, bo avtomatično protokol uporabil HTTPS namesto HTTP. HTTPS sam po sebi pride s kriptografsko varnostjo, in sicer z avtentikacijo z javnim in zasebnim ključem.
5. Sporočila bodo shranjena na pošiljateljevem strežniku. Ustvarjena bodo preko HTTP zahtevka.
6. Ko bo uporabnik poslal sporočilo, bo prejemnikov strežnik poslal podatke o sporočilu prejemnikovemu strežniku, ki jih bo shranil v svojo podatkovno bazo, da bodo ves čas na voljo prejemniku. Seveda te podatki ne vključujejo telesa sporočila niti priponk. Za dodatne informacije si lahko ogledamo tudi poglavje o poteku pošiljanja sporočila (POTEK POŠILJANJA SPOROČILA).
7. Sporočila bodo uporabljala standarden JSON format.

IMPLEMENTACIJE E-POŠTNIH PROTOKOLOV IN E-POŠTNI KLIENTI

Obstaja več implementacij e-poštnih protokolov, ker pač moramo nekako prejemati sporočila. Implementacije delimo predvsem na dva dela, in sicer na strežniške in klientske.

Exchange

Microsoft Exchange je plačljiva implementacija za strežnik. To pomeni, da se poganja na centralnem strežniku, ki sprejema in pošilja sporočila drugim strežnikom in klientom. V trenutku pisanja je zadnja verzija Microsoft Exchange 2019. Exchange je namenjen predvsem poslovnim uporabnikom, ker je zelo stabilen in vsebuje dodatne funkcije.

Apache James

Apache James je odprtokodna, brezplačna implementacija za strežnik. Med drugim podpira tudi JMAP protokol, ki ga Exchange ne. Licencirana je pod Apache License v2. Apache James je relativno stabilen in varen, ni pa tako hiter, ker je zgrajen na Javi, ki je sama po sebi interpretiran jezik, za razliko od C# (ki je gradljiv jezik (angl. *compiled language*)), ki ga uporablja Exchange.

Thunderbird

Mozilla Thunderbird je odprtokodna, brezplačna implementacija za klienta. To pomeni, da se namesti kot program na uporabnikov računalnik oz. napravo in tam preko protokolov, kot so IMAP, POP in pa JMAP, odjema elektronsko pošto. Ima veliko različnih funkcij, napisana je v raznih jezikih, licencirana pod Mozilla Public License v2.

Roundcube

Za razliko od Thunderbirda je Roundcube mišljen kot nekaj vmes med implementacijo za strežnik in implementacijo za klienta. S tem mislim, da ga lastniki strežnika, na katerem poteka tudi implementacija za strežnik, namestijo, konfigurirajo in končni uporabnik se lahko prijavi preko spletne strani, preko katere se lahko navadno prijavi le preko iste domene v elektronskem naslovu, ki jo ima spletna stran. To pomeni, da ima končni uporabnik veliko manj dela z raznimi programi, ki bi jih moral drugače namestiti. Roundcube nima (uradne) mobilne oz. namizne verzije, saj bi tako v resnici postal Thunderbird.

MOJ PROTOKOL

Celotna infrastruktura je izjemno stara in včasih tudi kar težko berljiva. Na primer, SMTP protokol je zelo lepo berljiv, čeprav bi lahko rekli, da je HTTP protokol še malo bolj berljiv. POP protokol je kar solidno berljiv, medtem ko je IMAP po mojem mnenju zelo težko berljiv – ima veliko sintaktičnega sladkorja in vse je preveč kompleksno.

V današnjem svetu, v katerem vsi preveč hitimo, se seveda zgodi kakšna napaka, pa naj bo to drobna tipkarska napaka, napačen zapis datuma ali ure srečanja, kdaj pa celo velike napake, kot je pošiljanje sporočila napačnemu prejemniku.

Moj cilj je bil, da naredim izjemno preprost nov protokol, ki bi temeljil na spoznanjih, pridobljenih iz prednosti in slabosti starih protokolov.

Zato sem se odločil, da bom uporabljal preproste HTTP/HTTPS zahtevke (angl. *request*). Moj protokol se bo, podobno kot današnji protokoli, najprej poskušal povezati na varni priključek/vrata, v tem primeru so to TLS vrata 443, če pa bo to spodletelo, bo uporabil nevarovana/nekriptirana vrata, v tem primeru vrata 80 (standardni HTTP priključek).

Ker se nam, kot sem že omenil, pri pošiljanju elektronske pošte pripetijo razne napake, sem se odločil dodati še urejanje teksta že poslanega sporočila in pa brisanje poslanega sporočila. To je izjemno težko izvedljivo z pristopom SMTP-ja, zato sem uporabil pristop, ki bi ga izbral Internet Mail 2000, torej, da so poslana sporočila skrb pošiljatelja in ne prejemnika ter se vštejejo v kvoto pošiljatelja in ne v prejemnikovo kvoto.

Ker pa zaradi omejitev (limitacij) pri Internet Mail 2000 protokolu, ki sem jih omenil že prej (torej threadanje pri pošiljanju notifikacij), ne bi bilo ravno praktično uvesti čisto enakega protokola, sem se odločil za malo bolj prefinjen način, in sicer da uporabljam zbirko podatkov tudi za prejeta sporočila. V tej zbirki podatkov sem se odločil zbirati čim manj podatkov, in sicer URL do sporočila na pošiljateljevem strežniku, od koga je sporočilo, za koga je namenjeno in pa tudi naslov sporočila, ki bi se lahko kasneje na pošiljateljevem strežniku tudi spremenil.

URL bi moral ostati skrivnost, namreč tako bi lahko kdorkoli, ki bi imel dostop do prejemnikovega računalnika, dobil trajen dostop do e-maila, tudi brez piškotka, ki bi postal neveljaven po enem dnevu od prijave.

Protokol sem poimenoval SMTP2 (*Simple Mail Transfer Protocol version 2*), ker je bil narejen kot zamenjava za obstoječega.

V telesu sporočila bi uporabljal »Markdown«. »Markdown« je preprost format za oblikovanje teksta, podobno kot HTML, samo da je veliko bolj preprost.

Poglejmo si, kako bi razlika med zapisom v HTML-ju in zapisom v Markdownu izgledala v praksi. Na sliki 6 lahko vidimo primer zapisa v HTML obliki, na sliki 7 pa je prikazan zapis v Markdownu.

```
<h1>Pozdravljen!</h1>
Kako smo kaj danes?
<p>Lep pozdrav</p>
<p>Mitja</p>
```

Slika 6 - Primer HTML-ja – Lastno delo; Narejeno 9.1.2022

```
# Pozdravljen!
Kako smo kaj danes?

Lep pozdrav

Mitja
```

Slika 7 - Primer Markdowna – Lastno delo; Narejeno 9.1.2022

Rezultat ene ali druge možnosti bi bil enak temu, kar lahko vidimo na sliki 8.

Pozdravljen!

Kako smo kaj danes?

Lep pozdrav

Mitja

Slika 8 - Primer prevedenega HTML-ja v Brave brskalniku – Lastno delo; Narejeno 9.1.2022

PREDNOSTI IN SLABOSTI MOJEGA PROTOKOLA

Vsaka stvar ima prednosti in slabosti. Podobno velja tudi za moj protokol, zato bom v nadaljevanju omenil nekaj pozitivnih in nekaj negativnih lastnosti le-tega.

Ena izmed prednosti je, da moj protokol omogoča zelo enostavno urejanje in brisanje sporočil. Ker je pošiljatelj dejanski lastnik sporočila, lahko brez težav popravlja vse v sporočilu – briše ali dodaja priponke, ureja sporočilo in ga tudi izbriše, v kolikor ga prejemnik pred tem še ni prebral.

Za lažji prikaz si oglejmo naslednji primer.

Učitelj želi drugim učiteljem istega predmeta preko elektronske pošte poslati test. Ker pa je zelo zaposlen in konstantno dela nekaj drugega, izbere kot prejemnika tudi učenca. Učitelj pošlje sporočilo in se takoj zave svoje napake. Na trenutnem protokolu (SMTP) popravek ne bi bil mogoč, pri SMTP2 protokolu pa lahko nemudoma izbriše sporočilo, učenec pa bo pri sebi lahko videl novo prejeto sporočilo, vendar pa ne bo mogel dostopati do njegove vsebine.

Hkrati je protokol zgrajen na sodobni infrastrukturi, in sicer na JSON-u in HTTP-ju. Implementacija tega protokola (12) je napisana v Go-ju in je zato izjemno lahko berljiva, hitra ter zanesljiva. Go je namreč striktno tipiran jezik (to je jezik, kateremu pri spremenljivkah vnaprej določiš tip vrednosti (npr. številka, znak ali pa binarna vrednost), kar omogoča enostavno predvidevanje vseh možnih napak. Velja pa omeniti, da je programski jezik odvisen tudi od implementacije.

Nevtralna lastnost je to, da SMTP2 poskrbi tudi za t. i. »mail sandboxing«, ki ga bom podrobneje predstavil v naslednjem poglavju

SMTP2 uporablja princip, da ko hoče prejemnik pogledati sporočilo, mora prejemnikov strežnik zahtevati na pošiljateljevega, da mu pošlje podatke sporočila, to pa se da uporabiti, da se sledi elektronski pošti.

Povzeto po Wiredu (13), se na dan pošlje okoli 269 milijard sporočil. To je okoli 35 sporočil na vsako osebo na svetu. Okoli štirideset odstotkom sporočilom naj bi sledili. Torej potemtakem sledijo okoli 108 milijardam sporočil. To je izjemno veliko. To je sicer bonus za nekatera podjetja in vlade, tako namreč lahko vedo, kdaj natanko je uslužbenec odprl sporočilo, hkrati pa je tudi velik poseg v zasebnost. Vemo, da se danes sledi elektronski pošti, poslani preko SMTP protokola tako, da pred pošiljanjem sporočila vstavijo majhno (nevidno) sliko (ki ni pripeta kot priponka, ampak je povezava do strežnika), navadno veliko en piksel. Ko prejemnik odpre sporočilo, bo odjemalec avtomatično naložil vse slike, vključno s to sliko. Sicer nekateri klienti blokirajo prenos slik in pred prenosom vprašajo uporabnika, če naložijo slike. Lep primer tega je Mozilla Thunderbird. Moj protokol pa temu ponuja rešitev na dlani. To je kar problematično, ampak tudi relativno uporabno. Odvisno je od lastne etike in morale, ali boste sledili sporočilu. Jaz vam tega ne bom preprečeval, ampak vseeno pomislite na druge, ko ugotovijo, da sledite vsemu.

Slabost je, da SMTP2 lahko poskrbi za brisanje sporočil ali urejanje. To je sicer zelo uporabno, ampak prinaša tudi razne negativne opcije.

Predstavljajte si, da cela država uporablja SMTP2, vključno z direktorji. In kar naenkrat se zgodi škandal. Javno podjetje se je mimo postopka javnega naročanja dogovorilo za naročilo, zato hočejo specialci pregledati elektronsko pošto direktorja, ampak ne najdejo nič, ne na prejemnikovi strani ne na pošiljateljevi strani, ker je pošiljatelj vse izbrisal. Ne morejo vložiti obtožnice, ker imajo premalo dokazov.

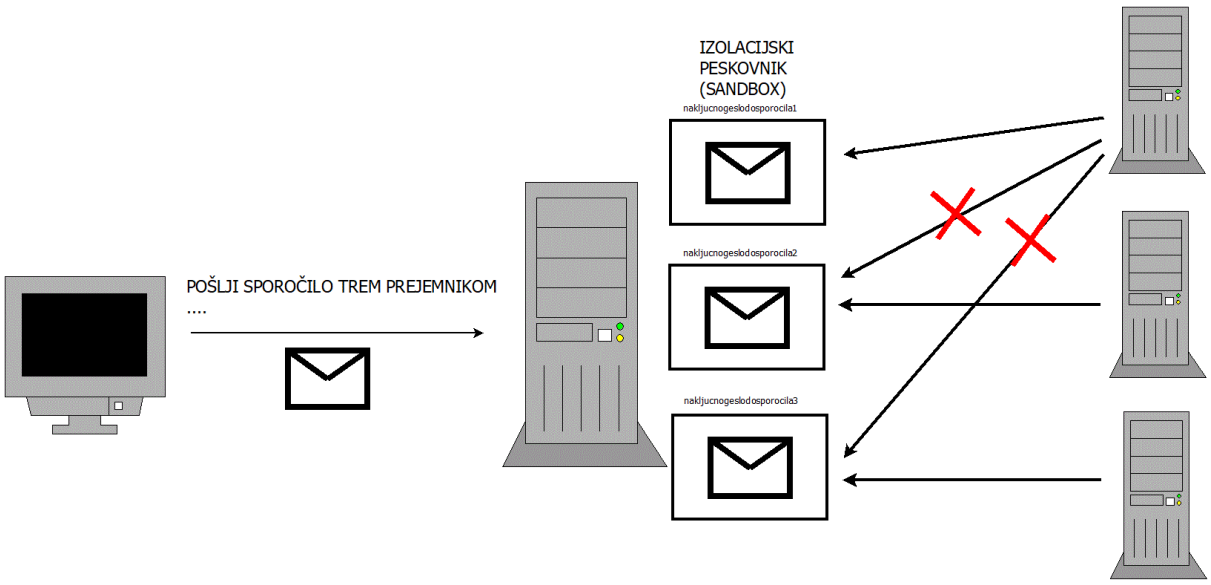
Zelo pomembno je vedeti tudi, da je SMTP2 odvisen od pošiljateljevih strežnikov. Če kakšen izmed korporacijskih strežnikov samo nekaj minut ne bi deloval, bi lahko privedlo do vsesplošne panike. Še posebej, ko vidimo, da smo odvisni od velikih korporacij, kot so Google in Microsoft, ki ponujajo največje storitve elektronske pošte, kot so Gmail in Outlook. Če bi se zgodilo, da se med posodabljanjem česarkoli zgodi kakršnakoli napaka, bi bila vsaj polovico elektronske pošte nedosegljiva za ta čas. Nihče ni popoln, niti korporacije, ki imajo nadzor nad temi strežniki. Videli smo že Googlov štiriminutni kolaps. Ko Google cele štiri minute ni deloval, se je globalni internetni promet zmanjšal kar za 40% (14)

MAIL SANDBOXING

Mail sandboxing (slovensko *Izolacija sporočil*) je zmožnost omejevanja sporočila na določenega prejemnika. Povedano z drugimi besedami, če izberemo več prejemnikov, bo SMTP2 avtomatično ustvaril več sporočil, in sicer za vsakega prejemnika svoje. Ker ima vsak prejemnik svoj ključ za dostop do sporočila, je mogoče sporočilo omejiti na vsakega prejemnika posebej. V praksi to pomeni, da lahko sporočilo, ki smo ga nehote poslali nekemu tretjemu prejemniku, izbrišemo ali uredimo, pri tem pa ne vplivamo na ostala sporočila, ki smo jih poslali ostalim prejemnikom. Ta funkcionalnost se lahko izkaže kot dobra rešitev v primeru, če nekdo ukrade geslo enemu od prejemnikov sporočila, saj lahko to sporočilo izoliramo in izbrišemo ter tako poskrbimo, da bodo do sporočila res dostopali le tisti, ki jim je vsebina namenjena.

Za uporabo te funkcijo sem se odločil, ker bi morebitna uporaba enega sporočila za vse prejemnike vplivala na kompleksnost protokola, moj cilj pri sami izdelavi SMTP2 protokola pa je bil njegova enostavnost.

Kljub pozitivnim vplivom uporabe izolacije sporočil pa je tukaj potrebno omeniti tudi negativno lastnost oziroma težavo, ki bi se lahko pojavila ob uporabi funkcije. V kolikor bi želeli urediti sporoča vseh prejemnikov, bi to za porabili ogromno časa. Težavo bi lahko rešili z različnimi implementacijami, ki bi vsa sporočila povezovale v eno samo sporočilo.



Slika 9 - Izolacija sporočil pri SMTP2 protokolu - Lastno delo; Narejeno 12.2.2022 s programom Dia

POTEK POŠILJANJA SPOROČILA

V nadaljevanju bom predstavil, kako poteka pošiljanje sporočila preko mojega protokola.

Vse skrivnosti (ključi - tokeni), ki sem jih napisal v tej raziskovalni nalogi, so zdavnaj potekli in niso več veljavni.

Večina začetnih postopkov je specifična zgolj moji implementaciji. Pri vsakem koraku bom zapisal, ali gre za del, ki je specifičen zgolj za mojo implementacijo, ali ne.

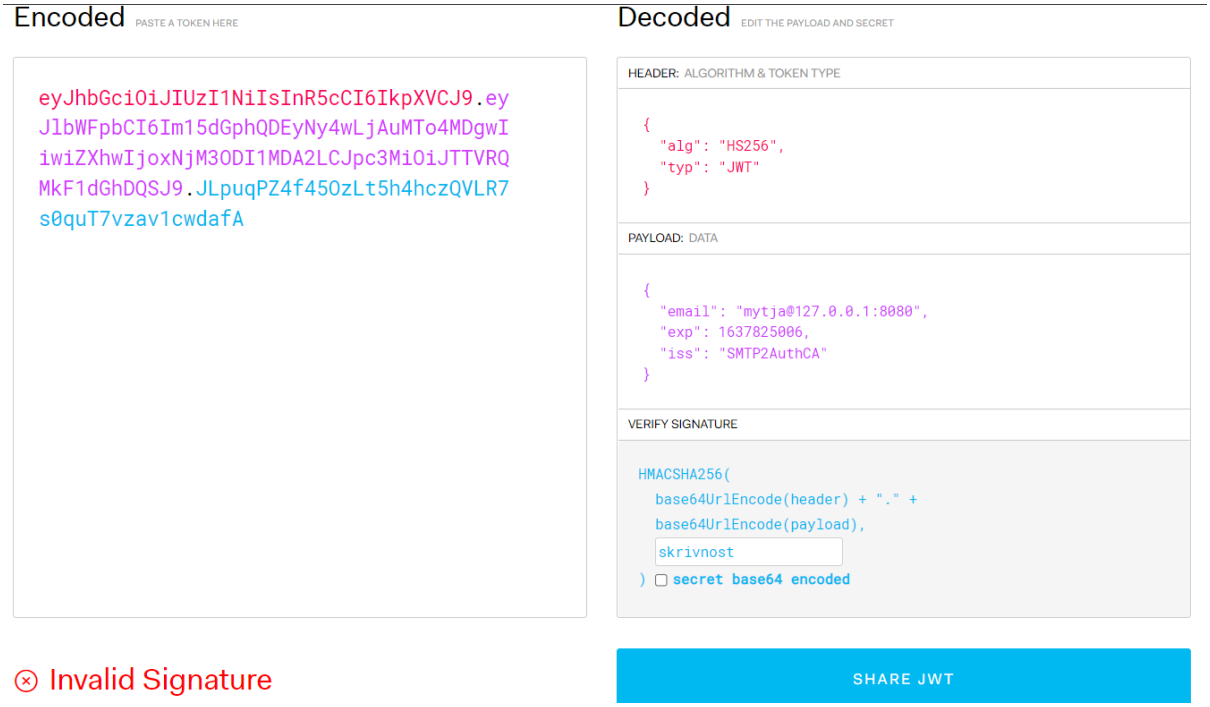
Postopek pošiljanja sporočila:

1. Uporabnik se registrira in prijavi.

S prijavo pridobi t. i. JWT (*JSON Web Token*), ki služi overjanju klienta na strežniku. Ta postopek je sicer specifičen samo za mojo implementacijo protokola (12).

JWT-ji tvorijo neko besedilo, ki se sprva zdi naključna, vendar se izkaže, da temu ni tako. V resnici gre za ključ JSON, ki je kodiran s podatki.

Pozitivna lastnost JWT-jev je to, da so kriptografsko podpisani. To pomeni, da jih brez podpisa lahko dekodiramo in pridobimo njihovo vsebino, vendar ni možno ustvariti novih JWT-jev, vsaj brez podpisa ne. Strežnik primerja podpis JWT-ja s podpisom na strežniku in v primeru, da zazna ujemanje, to pomeni, da je strežnik zagotovo izdal ta ključ, zato lahko strežnik nadaljuje.



Slika 10 - Primer JWT ključa na spletni strani jwt.io; Lastno delo – Narejeno 16.1.2022

2. Klient ustvari osnutek

Ta postopek je specifičen za mojo implementacijo.

POST /smtp2/draft/new

X-Login-Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im15dGphQDEyNy4wLjAuMTo4MDgwIiwiZXhwIjoxNjM3ODI1MDA2LCJpc3MiOiJTTVRQMkF1dGhDQSJ9.JLpuqPZ4f45OzLt5h4hczQVLR7s0quT7vzav1cwdafA
ReplyTo: <prazen string>
UseMessage: <prazen string>

Slika 11 - Primer zahtevka za ustvarjanje osnutka; Lastno delo - Narejeno 16.1.2022

Strežnik klientu vrne ID osnutka in ostale podatke, kot je na primer telo sporočila, v katerem sta podpis in posredovano sporočilo v primeru posredovanja. V primeru odgovarjanja na sporočilo se pošlje tudi podatke o pošiljatelju ter prejemniku (»From« in »To« polji) ter naslov (Title).


```
▼{success: true,…}
  ▼data: {ID: 86, To: "", Title: "", Body: "↲\↲↲---↲↲Mitja Ševerkar, 9.b - OŠ Vodmat↲", From: "",…}
      Attachments: []
      Body: "\n\\\n\n---\n\nMitja Ševerkar, 9.b - OŠ Vodmat\n"
      From: ""
      ID: 86
      Title: ""
      To: ""
    error: ""
    success: true
```


Slika 12 - Primer odziva za ustvarjanje osnutka; Lastno delo - Ustvarjeno v Brave (Chromium) orodjih za razvijalce (zavihek Omrežje) - Narejeno 14.3.2022

V telo sporočila se avtomatično vstavi podpis, ki ga je uporabnik prej nastavil. Podpis podpira Markdown.

3. Uporabnik posodobi osnutek

Ta postopek je specifičen za mojo implementacijo.

PATCH /smtp2/message/update

X-Login-Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im15dGphQDEyNy4wLjAuMTo4MDgwIiwiZXhwIjoxNjM3ODI1MDA2LCJpc3MiOiJTTVRQMkF1dGhDQSJ9.JLpuqPZ4f45OzLt5h4hczQVLR7s0quT7vzav1cwdafA

{“ID”: 86, “Body”: “To je posodobljen body brez podpisa”, “To”: “mitjas@mail1.smtp2.tk”, “Title”: “Naslov sporočila”}

Slika 13 - Primer zahtevka za posodabljanje osnutka; Lastno delo - Narejeno 16.1.2022

V primeru, da strežnik uspešno shrani nove podatke v osnutek, s sporočilom OK potrdi posodobitev osnutka, sicer pa v primeru, da osnutka ni uspel shraniti, javi napako.

4. Uporabnik pošlje sporočila

Ta postopek je specifičen za mojo implementacijo.

POST mail1.smtp2.tk/smtp2/message/new

{“DraftID”: 86}

Slika 14 - Primer zahtevka za pošiljanje sporočila na pošiljateljevi strani; Lastno delo - Narejeno 23.1.2022

Od tu naprej je vse standardizirano in strežnik prevzame vso breme pošiljanja sporočila. Zato bomo od zdaj naprej gledali, kaj se dogaja na strani strežnika in ne več, kaj se dogaja na pošiljateljevi strani.

5. Pošiljateljev strežnik pošlje ključne podatke o sporočilu prejemnikovemu strežniku

POST mail2.smtp2.tk/smtp2/message/receive

Title: Naslov sporočila
To: mitjas@mail1.smtp2.tk
From: posiljatelj@mail1.smtp2.tk
ServerPass: <naključno ustvarjeno geslo za dostop do sporočila>
ReplyPass: <naključno ustvarjeno geslo>
ReplyID: <naključno ustvarjeno geslo>
OriginalID: -1
ServerID: <ID tega sporočila>
MVPPass: <naključno ustvarjen ključ za verifikacijo sporočila>
URI: <URL do sporočila na pošiljateljevem strežniku>

Slika 15 – Zahtevek na prejemnikov strežnik ob pošiljanju sporočila – Lastno delo; Narejeno 15.1.2022

Tukaj pa stvar postane zanimiva. To je, kot sem že omenil, standardiziran postopek in se ne sme razlikovati od implementacije do implementacije.

Kar strežnik naredi je, da pošlje prejemniku vsa pomembna polja za dostop do sporočila.

Za vsa polja bom razložil, zakaj so pomembna in kako približno izgleda vsebina sporočila.

Title (slovensko *Naslov*) je naslov sporočila. Pošlje se naslov, ki ga je pošiljatelj izbral ob času pošiljanja. Naslov se seveda sčasoma lahko na pošiljateljevem strežniku spreminja, zaradi česar lahko pride do tega, da je naslov v poštnem predalu (angl. *inbox*) drugačen kot potem, ko uporabnik odpre sporočilo. Naslov sem vključil zaradi praktičnih razlogov, saj ob pregledu poštnega predala najprej pogledamo naslove. V kolikor bi od pošiljatelja vedno na novo zahteval naslove že na poštnem predalu, bi bilo zamudno in bi tudi odprlo nove možnosti, predvsem za prevarante, ki iščejo delujoče elektronske predale (elektronske naslove), ker bi tako lahko vsakič, ko bi zahtevali naslov, videli, da ta elektronski predal obstaja.

To (slovensko *Za*) polje, opisuje, komu je namenjeno sporočilo, t. i. prejemnik. Prejemnike in pošiljatelje se označuje po standardnem načinu, tj. ime-postnega-naslova@domena.si.
Moja implementacija podpira UTF-8 simbole, zato so lahko tako naslovi kot tudi elektronski naslovi z UTF-8 simboli, torej tudi s posebnimi črkami slovenske abecede (šumniki), čeprav se tega generalno ne priporoča.
Primer: prejemnik@mail1.smtp2.tk

From (slovensko *Od*) polje opisuje, kdo je poslal to sporočilo, tj. pošiljatelj.
Primer: posiljatelj@mail1.smtp2.tk

ServerPass (slovensko (malo na široko) *Geslo do sporočila na strežniku*) polje vsebuje naključno ustvarjeno geslo, ki bo uporabilo prejemnikov strežnik za dostop do sporočila na pošiljateljevem strežniku.

ReplyPass (slovensko *Geslo za odgovarjanje*) in pa ReplyID (slovensko *Identifikator za odgovarjanje*) polji vsebujeta naključno ustvarjeni gesli, ki se povezujeta z ostalimi sporočili, da lahko identificiramo odgovore na sporočila.

OriginalID (slovensko *Originalni identifikator*) sporoči prejemnikovemu strežniku če je to čisto prvo sporočilo v celotnem pogovoru (odgovarjanju). Če je čisto prvo, strežnik pošlje »-1«, sicer pa pošlje ID prvega sporočila na pošiljateljevem strežniku (glej *ServerID*).

ServerID (slovensko *Strežniški Identifikator*) ni edinstveni identifikator strežnika, ampak identifikator sporočila na pošiljateljevem strežniku. To se pošlje zato, ker ne shranita oba strežnika sporočila pod enakim ID-jem, saj bi se lahko zgodilo, da bi jih več poslalo ID 1, kar bi privedlo do velike zmede, saj bi prejemnikov strežnik shranil več sporočil pod 1, česar pa SQL (*Structured Query Language* – jezik ki se uporablja za podatkovne baze) ob uporabi ključa *UNIQUE* (slovensko *Edinstven*) ne dopušča.
Primer: 86

MVPPass (slovensko *Geslo za validacijski protokol za sporočila*) je naključno ustvarjeno geslo za preverjanje istovetnosti pošiljateljev sporočila. Odločil sem, da ne bom uporabljal gesla za dostop do sporočila, saj bi tako lahko prišlo do shranjevanj gesel v hekerske podatkovne baze. Celoten validacijski protokol sem razložil v poglavju o varnosti protokola.

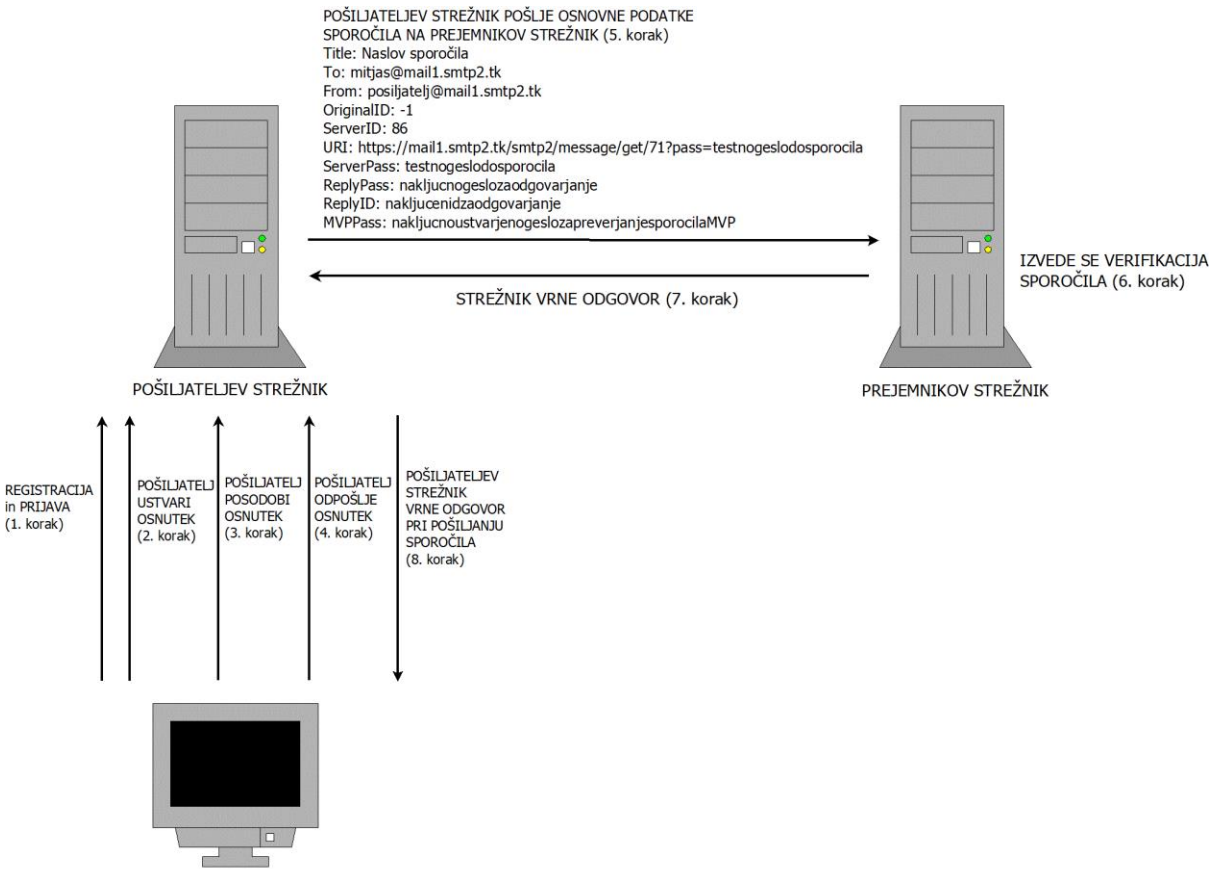
URI (krajše za *Uniform Resource Identifier*) je URL (krajše za *Uniform Resource Locator*) za dostop do sporočila na pošiljateljevem strežniku. Zato mora pošiljateljev strežnik nujno vedeti, ali uporablja HTTPS (moj protokol ima veliko raje HTTPS, kot standarden HTTP) in pa na kateri domeni servira. To mu mora uporabnik podati ročno.
Primer: https://mail1.smtp2.tk/smtp2/message/get/1?pass=nakljucnogeslodosporocila

6. Sledi tako imenovani MVP, katerega potek si lahko ogledate v naslednjem poglavju o varnosti protokola.
7. Prejemnikov strežnik lahko ob prejemanju sporočila vrne naslednje odgovore:

- 201 – Created (slovensko *Ustvarjeno*) se vrne, kadar je uspešno prestalo vse in se je shranilo sporočilo v podatkovno bazo.
- 400 – Bad request (slovensko *Neveljaven zahtevek*) se zgodi, kadar se pošlje kaka vrednost premalo ali pa kadar ID ni veljavna številka.
- 403 – Forbidden (slovensko *Prepovedano*) se zgodi, kadar MVP (*Message Validation Protocol*) ne more preveriti pošiljatelja sporočila.
- 406 – Not acceptable (slovensko *Zahtevka ne morem sprejeti*) se zgodi, kadar hoče nekdo, ki ni pošiljatelj oziroma prejemnik tega sporočila, odgovarjati na sporočilo, ali pa kadar ne more najti prejemnika na tem strežniku (kar lahko različne implementacije tudi ne sporočijo).
- 500 – Internal Server Error (slovensko *Notranja napaka v strežniku*) se zgodi, kadar ne mora iz podatkovne baze pridobiti sporočil ali pa se pojavi druga nepričakovana napaka.

8. Če je vse v redu, si pošiljatelj shrani podatke sporočila, prejemnik pa osnovne podatke do sporočila. Hkrati klientu vrne, kar je pošiljateljev strežnik prejel od prejemnikovega strežnika.

Potek protokola najlažje ponazorimo z diagramom (slika spodaj).



Slika 16 - Potek pošiljanja sporočila na SMTP2 protokolu - Lastno delo; Narejeno 24.1.2022 s programom Dia

VARNOST PROTOKOLA

Da bi protokol zavarovali pred vsiljeno pošto, moramo uporabljati nek verifikator elektronske pošte, ki preveri, če je pošiljatelj elektronske pošte verodostojen. V ta namen sem oblikoval protokol, ki sem ga poimenoval SMTP2 MVP (*Message Validation Protocol*).

Potek protokola SMTP2 MVP:

1. Pošiljateljev strežnik pošlje sporočilo prejemnikovemu strežniku (Slika 15)
2. V prejetem sporočilu sta dve polji v glavi, in sicer »From« (elektronski naslov od koga je to sporočilo) in pa »URL« (povezava do sporočila).
3. Ko prejemnikov strežnik prejme sporočilo, bo iz URL do sporočila in iz elektronskega naslova pošiljatelja pridobil domeno
4. Prejemnikov strežnik bo preveril, ali ti domeni uporabljata HTTP ali HTTPS z uporabo HEAD zahtevka (angl. *request*).
5. Prejemnikov strežnik bo naredil HTTP(S) zahtevek na strežnik za vsako domeno.

   GET /smtp2/message/verify?id=<id sporočila>&pass=<MVP verifikacijsko geslo>

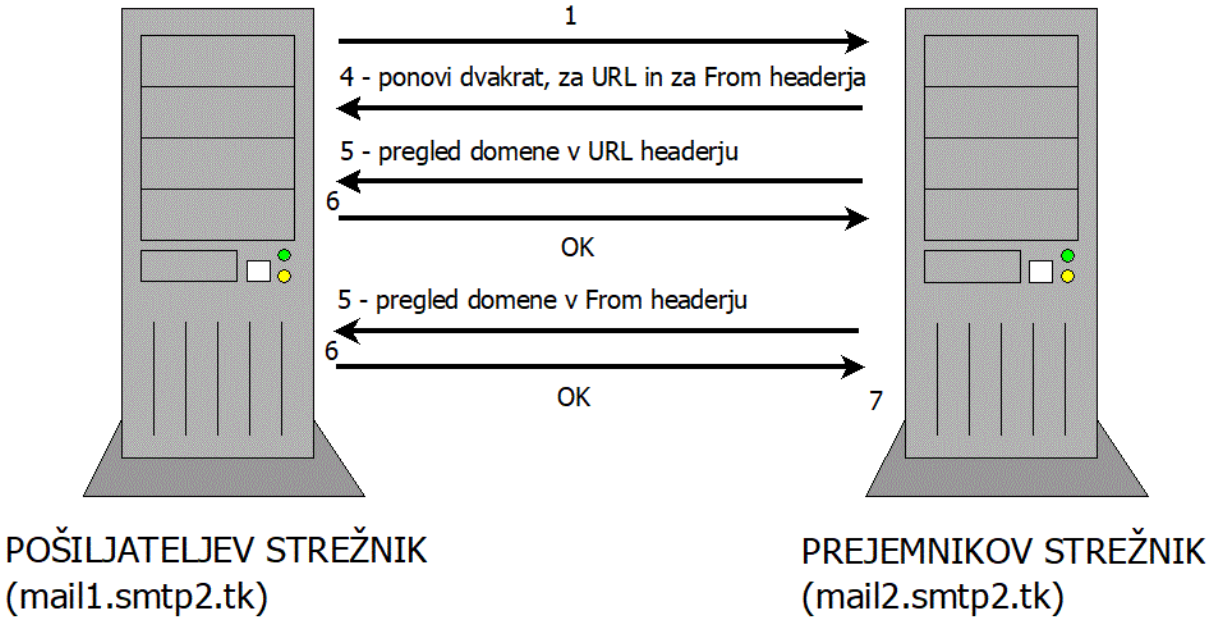
   Slika 17 - Zahtevek MVP protokola – Lastno delo; Narejeno 15.1.2022

   <id sporočila> se zamenja z ID-jem sporočila na pošiljateljevem strežniku. Tega pošiljatelj pošlje v »ServerID« polju v glavi sporočila.
   <MVP verifikacijsko geslo> je naključno ustvarjeno geslo (v moji implementaciji dolgo 80 znakov), ki je drugačno od gesla za dostop do sporočila (ker bi to lahko povzročilo resno varnostno težavo (15)). To geslo je prisotno med podatki, ki jih pošiljateljev strežnik pošlje prejemnikovemu, ko se sporočilo pošlje – natančneje v »MVPPass« polju v glavi.
6. Strežnik na drugi strani preveri, ali je dejansko poslal to sporočilo. Preveri, če se:
   1. Na njegovem strežniku nahaja sporočilo z tem ID-jem,
   2. MVP geslo tega sporočila ujema z poslanim.

   Če se vse ujema, odgovori z *OK*, sicer pa s *FAIL*.

7. Ko prejemnikov strežnik dobi odgovore na vse preglede, preveri, če sta oba strežnika odgovorila z OK, sicer avtomatično zavrne sporočilo in to sporoči pošiljateljevem strežniku v odzivu.
   Če je vse v redu, prejemnikov strežnik shrani podatke tega sporočila in pošiljateljevemu strežniku sporoči, da je sporočilo shranjeno.
8. Pošiljateljev strežnik preveri odgovor in ga sporoči nazaj pošiljateljevemu klientu (računalniku).

Potek protokola SMTP2 MVP najlažje/lahko ponazorimo s sliko spodaj (slika 18).



Slika 18 - Potek SMTP2 MVP protokola – Lastno delo; Narejeno 15.1.2022 s programom Dia

Zgoraj opisani protokol zadošča v primeru, ko imamo lažen elektronski naslov pošiljatelja. Kot primer vzemimo nek uraden elektronski naslov, recimo @gov.si.

V »From« polje v glavi sporočila vpišemo ponarejen elektronski naslov, na primer test@gov.si, »URL« polje v glavi pa vpišemo veljaven URL (ki ne bi bil gov.si), da lahko prejemnik sploh vidi, kaj je v sporočilu. Recimo, da je URL naslednji:

http://ni-gov.si/smtp2/message/get/10?pass=nakljucnogeslodosporočila

V tem primeru bi MVP iz »From« polja v glavi dobili »gov.si« in iz URL-ja »ni-gov.si«.

Kot odgovor s strani »ni-gov.si« prejme »OK«, saj gre za naslov, s katerega je bilo sporočilo poslano, medtem ko je odgovor »gov.si« »FAIL«, saj sporočilo s tega naslova ni bilo poslano.

Ker je ena domena odgovorila »FAIL« se sporočilo avtomatično zavrne na prejemnikovemu strežniku in se tudi izbriše na pošiljateljevem strežniku.

Ker pa je sumljivo že to, da imamo namesto ene same kar dve domeni, kljub uspešno prestani MVP verifikaciji shranimo opozorilo poleg tega sporočila. Če klient/prejemnik želi prejeti podatke sporočila, mu te podatke pošljemo skupaj z opozorilom. Klient/prejemnik se potem sam odloči, kako bo opozorilo interpretiral, tj. kot grozljivo nevarnost ali pa kot nizko nevarnost.

Lahko se sicer zgodi, da domene ponaredijo potrdila o poslanem sporočilu (»OK«). Torej, kljub temu da sporočila niso poslale, izpišejo OK na validaciji. Tega sicer legitimne spletne strani, kot so javni ponudniki elektronske pošte (Arnes Mail, Google Mail idr.), vladne spletne strani (kot je gov.si) ali pa druge legitimne spletne strani ne bodo počele. Vseeno pa je tukaj potrebno poudariti to, da nam v primeru, če naletimo na takšen primer, ni pomoči.

ODGOVARJANJE NA SPOROČILA

Sedaj, ko poznamo proces pošiljanja sporočil in temeljne lastnosti SMTP2 protokola, bom predstavil, kako je zastavljeno odgovarjanje na sporočila.

Čeprav odgovarjanje na prvi pogled izgleda malce kompleksno, pa je v resnici zastavljeno na relativno enostaven način.

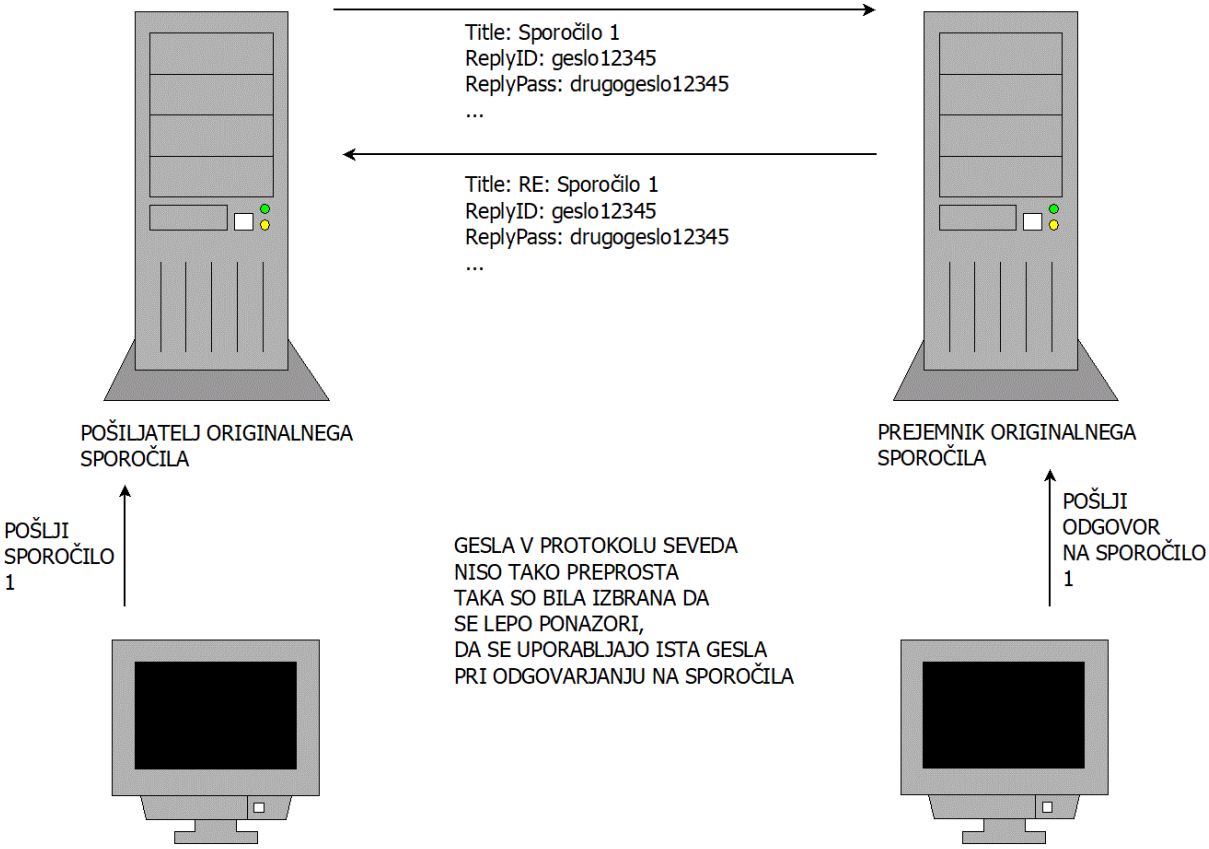
Vsako sporočilo prejme 2 polja v glavi sporočila, in sicer »ReplyPass« in »ReplyID«. To sta naključni gesli, dolgi približno 80 znakov. To omogoča identifikacijo verige sporočil. Ker imamo drugačne ID-je sporočil na vsakem strežniku, bi bilo nemogoče identificirati verige sporočil z ID-ji sporočil oziroma vsaj zelo težko implementirati. Obe polji se z vsakim odgovorom na sporočilo pošiljata naokoli, tj. vsakič ko se na sporočilo odgovori, se priključita isti »ReplyPass« in »ReplyID« kot na sporočilu, na katerega se odgovarja. To omogoča, da ko klient zahteva sporočilo, vmes še identificira vse odgovore na sporočila.

Ker imamo 2 gesli, ki imata 80 znakov, je (za zdaj) nemogoče ugotoviti obe gesli s požrešnim napadom (angl. *brute-force attack*), zato bo morda namesto napada prišlo do naključnih, nenamernih napak. V primeru, če bo pošiljatelj legitimna oseba, bo prejemnikov strežnik sporočilo zagotovo zavrnil, saj pošiljatelj oziroma prejemnik te verige sporočil nista enaka pošiljatelju oziroma prejemniku druge verige sporočil. To nas mora vsaj malo skrbeti, saj tako pride do nejevolje uporabnikov tega protokola, ko je bilo njihovo sporočilo zavrnjeno brez razloga. To lahko popravimo tako, da povečamo število znakov v geslih.

ID-ji se, vsaj v moji implementaciji, shranjujejo po zaporedju. To pomeni, da v primeru, če izbrišemo neko sporočilo, potem ID-ja, ki je bil dodeljen temu sporočilu, ni več mogoče zasesti. Zaradi tega lahko lažje ugotovimo, katera sporočila smo prejeli pred ali po prejemu določenega sporočila.

Pomemben del odgovarjanja je tudi »OriginalID« polje v glavi sporočila, ki prejemnikovemu strežniku sporoči, ali gre za originalno sporočilo, tj. prvo sporočilo v verigi sporočil. Če gre za prvo sporočilo, se prejemnikovemu strežniku pošlje »-1«, sicer pa ID sporočila na pošiljateljevem strežniku (»ServerID«).

Za lažje razumevanje zapisanega si oglejmo sliko 19.

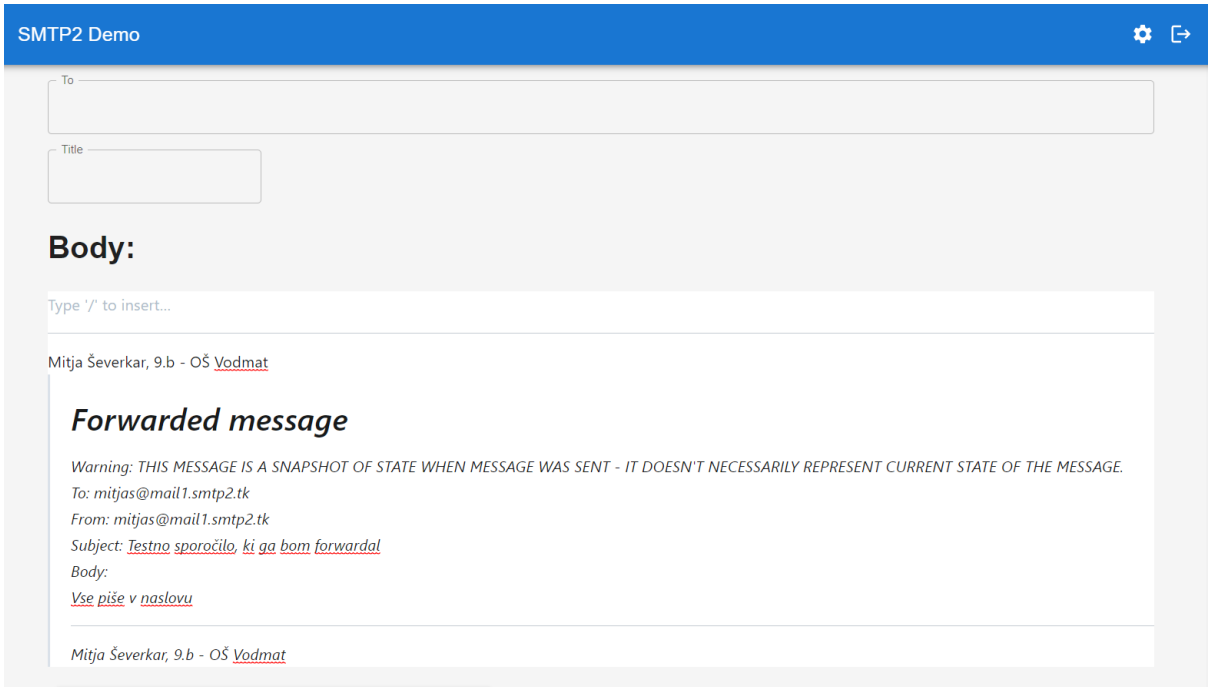


Slika 19 - Poenostavljen prikaz odgovarjanja na sporočila - Lastno delo; Narejeno 13.2.2022 s programom Dia

POSREDOVANJE SPOROČIL

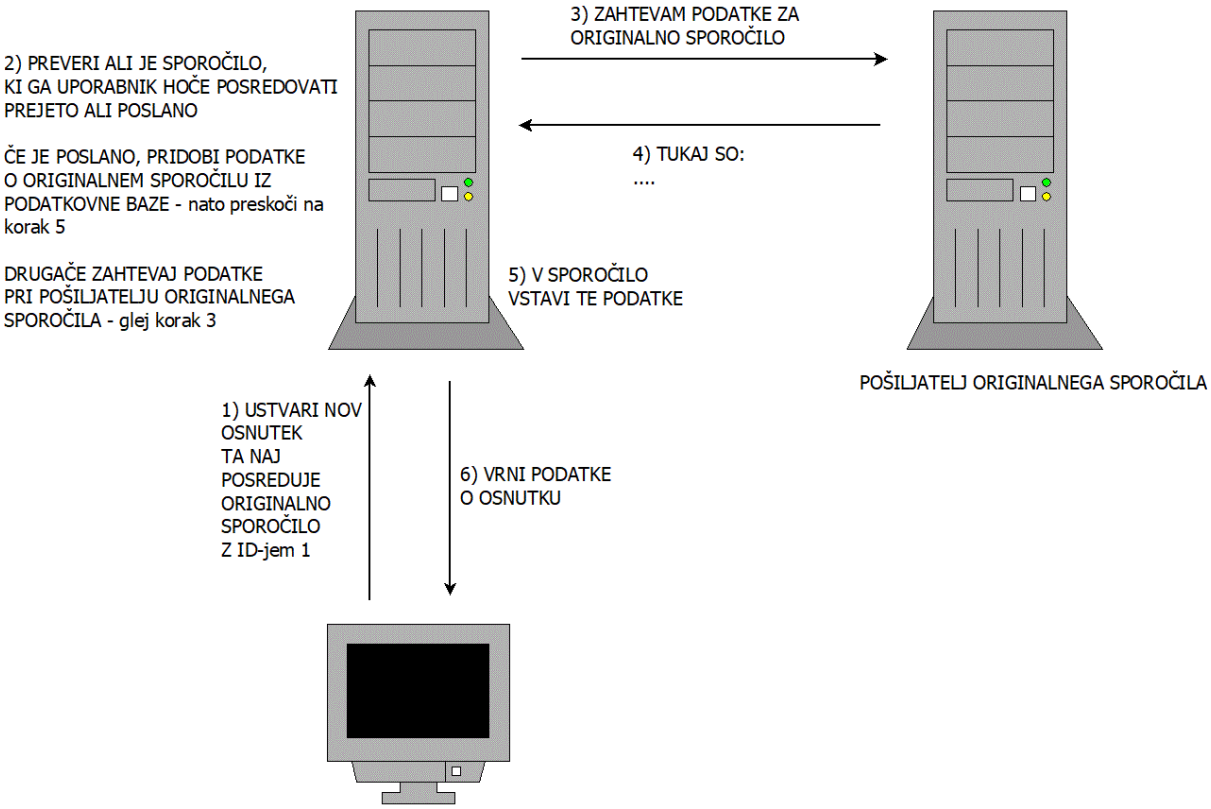
Posredovanje sporočil je relativno preprosto.

Če želimo posredovati sporočilo, se ustvari osnutek, strežnik (oziroma implementacija) pa v telo osnutka vdela sporočilo in ga vrne (glej sliko 12).



Slika 20 - Posredovanje sporočila - Lastno delo; Narejeno 30.1.2022

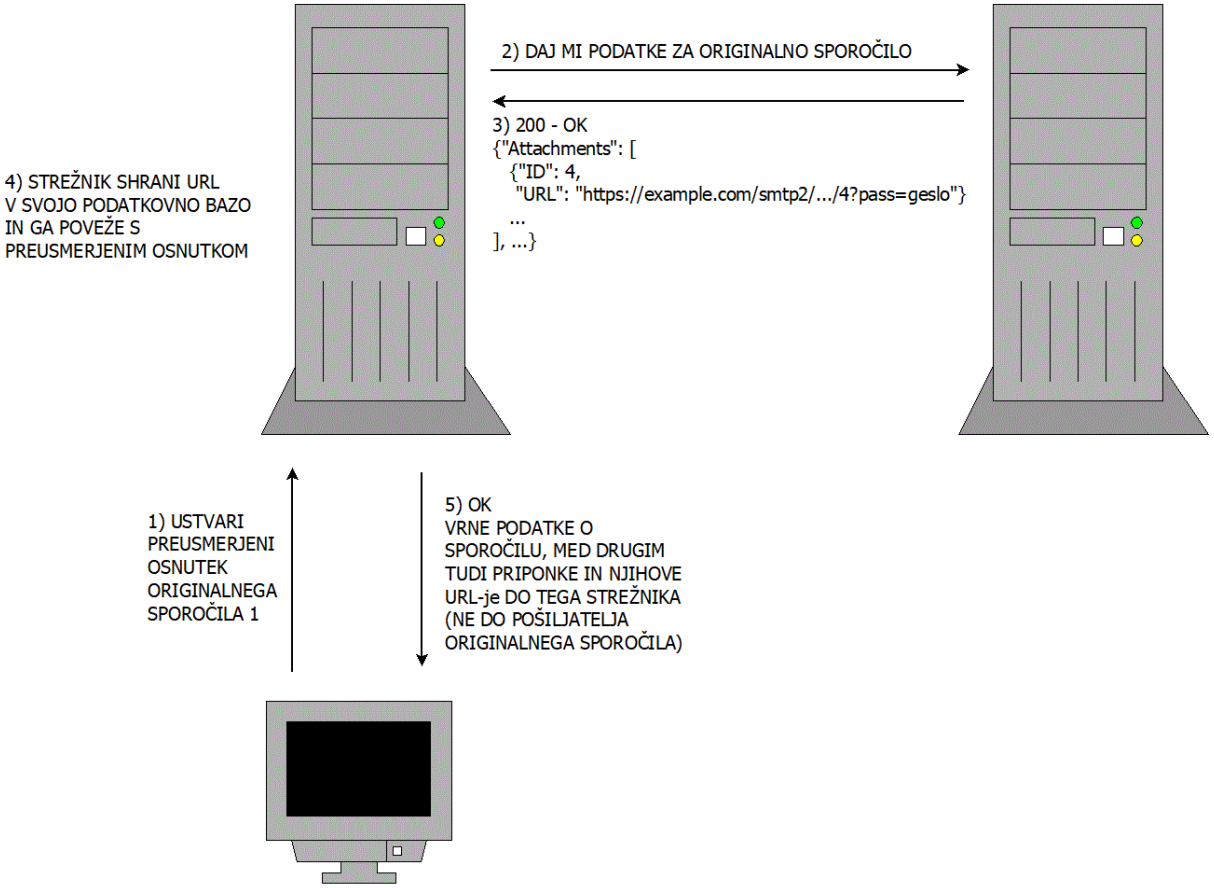
Na tem mestu moramo poudariti, da protokol ne zagotavlja posodabljanja posredovanih sporočil. Ko se ustvari osnutek s posredovanim sporočilom, implementacija v osnutek vstavi podatke sporočila v tistem času, ko je osnutek ustvarjen. Če bo pošiljatelj originalnega sporočila kaj spreminjal, se to ne bo poznalo na posredovanem sporočilu. Posodabljanje posredovanih sporočil bi bilo vsekakor možno dodati, vendar bi to pripomoglo h kompleksnosti protokola, kar pa je v nasprotju z mojo glavno idejo o preprostosti protokola.



Slika 21 - Posredovanje sporočil v SMTP2 protokolu - Lastno delo; Narejeno 31.1.2022 s programom Dia

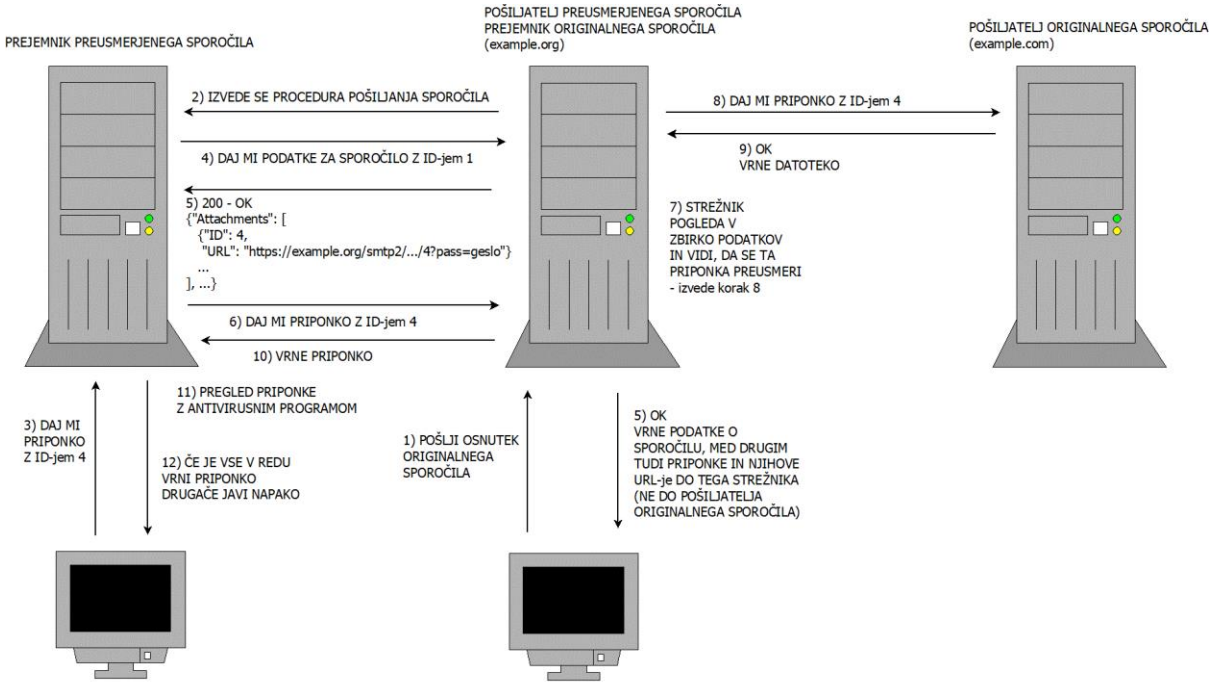
Problem se pojavi tudi pri shranjevanju priponk pri posredovanih sporočilih, saj se nekateri uporabniki odločijo za posredovanje sporočila zgolj z namenom posredovanja priponk. Zamislil sem si sistem, v katerem se, ko se osnutek ustvari, priponke na originalnem sporočilu prekopirajo, tako da se shranijo glavni podatki v podatkovno bazo. Namesto da prejemnikov strežnik zahteva priponko od originalnega sporočila, shrani URL do priponke na originalnem sporočilu.

Poskusimo to ponazoriti z diagramom (slika 22).



Slika 22 - Priponke pri ustvarjanju posredovanega sporočila - Lastno delo; Narejeno 1.2.2022 s programom Dia

Na sliki 23 si lahko ogledamo, kako bi izgledalo odpiranje priponke s strani prejemnika.



Slika 23 - Postopek prejema priponke pri SMTP2 protokolu - Lastno delo; Narejeno 1.2.2022 s programom Dia

V nadaljevanju si bomo ogledali nekaj prednosti in slabosti postopka prejema priponke.

Najprej bom opisal slabosti, nato pa prednosti.

Med drugim to zelo obremeni sistem, zasede namreč veliko RAM-a, CPU-ja in omrežne kapacitete.

Slabost je tudi tako imenovana rekurzija (angl. *recursion*). To je, ko funkcija kliče samo sebe. To lahko navadno privede do rekurzijske napake oziroma preobremenitev. Preobremenitev je, ko funkcija naredi veliko kroženj (ponavljanj), ampak nikoli ne pride do rezultata. To bomo najlažje razumeli ob naslednjem primeru. Predstavljajmo si, da sporočilo posredujemo nekomu, ki to isto sporočilo posreduje naprej, nato pa se posredovanje nadaljuje še kar nekaj časa. Recimo, da se to ponovi 10000-krat, ob tem pa imejmo v mislih, da imamo na voljo zgolj 2 strežnika. Zadnji prejemnik želi odpreti priponko in zato se mora protokol prebiti čez vseh 10000 sporočil na obeh strežnikih. Zgodi se, da en strežnik zahteva drug strežnik, ki zopet zahteva prvi strežnik in tako naprej, kar vodi do prekinitve povezave zaradi preobremenitve, hkrati pa tudi HTTP protokol sam prekine povezavo.

Med drugim lahko pride do tega, da v primeru, ko originalni pošiljatelj odstrani dostop do priponk, le-teh ne more videti nihče, niti tisti, kateremu je bilo sporočilo originalno poslano.

Prednost je, t. i. izolacija. Če hoče nekdo odstraniti dostop do priponk, lahko to naredi. Tako se bo izničil dostop do priponk prejemniku in vsem nadaljnjim posredovanjem tega sporočila.

Prednost je tudi, da gre vse preko strežnika, kateremu je bilo originalno sporočilo poslano. Tako lahko pošiljatelj misli, da to prenaša prejemnik tega sporočila, čeprav lahko prenaša tudi nekdo na čisto drugem koncu sveta. To deluje nekako po principu »proxyja« (preusmerjanje zahtevkov iz drugega IP naslova).

PRIPONKE IN VARNOST TEH

Priponke so zelo pomemben del elektronskih sporočil. Pri »klasičnem« pošiljanju sporočil se celotne priponke pošljejo prejemniku v sporočilu, ob prejemu pa teh priponk ni več mogoče izbrisati.

S svojim protokolom želim to popraviti, zato bodo priponke last pošiljatelja, prejemnikov strežnik jih pa bo na zahtevo prejemnikovega klienta prekopiral v prejemnikov strežnik.

Pomembno je vedeti, da priponk prejemnikov strežnik ne bo shranjeval (niti na zahtevo prejemnika), ampak jih bo samo prekopiral v RAM (začasni spomin računalnika), iz katerega se bo vse izbrisalo, ko bo zahtevka konec (tj. princip kolektorja smeti (angl. *garbage collector*)).

Tukaj se nam poraja vprašanje, kako bomo poskrbeli za varnost uporabnika, saj mu ne moremo kar direktno poslati URL-ja do priponke na pošiljateljevem strežniku, podobno, kot mu ne pošljemo URL-ja do sporočila na pošiljateljevem strežniku. O razlogih, zakaj prejemniku ne smemo poslati URL-ja, sem razpravljal v poglavju o mojem protokolu.

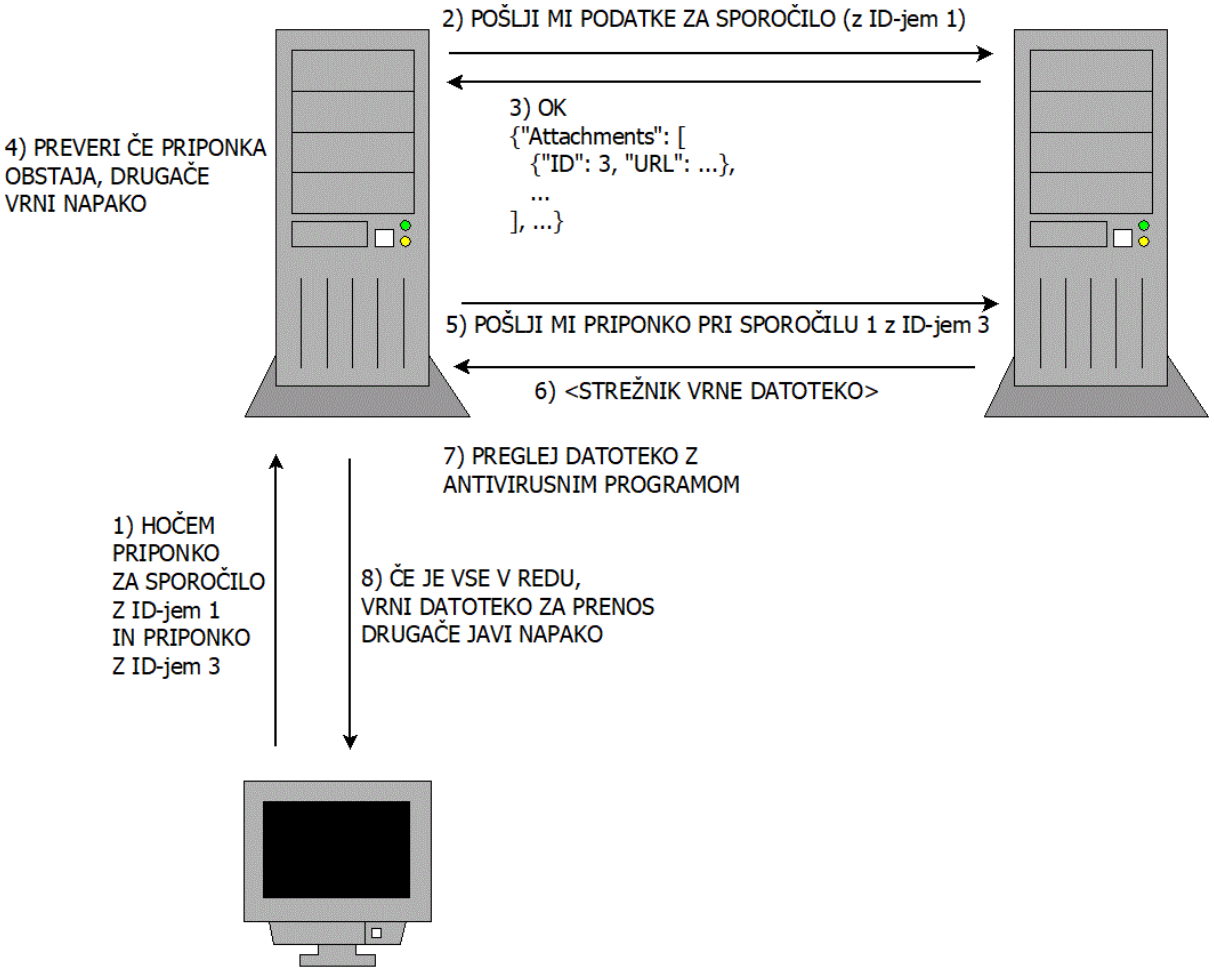
Zadevo bomo rešili tako, da bomo poslali zahtevek na prejemnikov strežnik, ki bo poslal zahtevek na pošiljateljev strežnik, ki bo vrnil priponko.

Hkrati je treba poskrbeti še za eno stvar. Če bi samo vrnili priponko, bi se lahko na prejemnikovem računalniku znašla zlonamerna programska oprema, kar je lahko zelo nevarno.

Zato bi medtem na prejemnikovem strežniku, ko bi prejemnikov klient hotel prenesti priponko, pognali antivirusni program, ki bi poskeniral datoteko. Če bi našel viruse, bi nemudoma zavrnil zahtevek prejemnika in mu sporočil, da je v datoteki virus.

Za antivirusni program sem v svoji implementaciji izbral ClamAV (16), ker deluje tako na Linuxu kot na Windowsu, lahko se poganja v Dockerju, ima zelo dobro zaznavo in je redno posodobljen. Hkrati sam posodablja svojo zbirko podatkov za viruse in zato ni potrebnega veliko vzdrževanja.

Na naslednji sliki (slika 24) lahko vidite potek prejema priponk.



Slika 24 - Potek prenosa priponk pri SMTP2 protokolu - Lastno delo; Narejeno 31.1.2022 s programom Dia

MOJA IMPLEMENTACIJA

Zdaj ko poznamo podrobnosti mojega protokola, je čas, da vam razkrijem še nekaj malega o moji implementaciji.

Poimenoval sem jo po protokolu, torej SMTP2.

Ustvaril sem tako spletno stran (angl. *frontend*) kot tudi strežnik, na katerem se dogaja vse procesiranje (angl. *backend*).

Strežniško kodo sem napisal v Go-ju. Go je statično in striktno tipiran (angl. typed) jezik, kar omogoča veliko varnost te kode. Hkrati izvira iz C-ja in je komplimiran jezik (angl. *compiled language*), kar mu omogoča izjemno veliko hitrost in razširljivost. Za poganjanje kode ni potrebno nič, ko zgradimo kodo, razen poganljive datoteke (angl. *executable*). Go ima sintakso, ki je zelo podobna pythonu in C-ju.

Spletna stran (angl. *frontend*), je napisana v javaScriptu. Uporabil sem react, za dizajnerski jezik pa Googlov Material UI.

Antivirusni program je, kot sem že prej omenil, ClamAV.

Koda se poganja z Dockerjem oziroma docker-composom (dodatna datoteka, s katero se upravlja več kontejnerjev z Dockerjem). Za strežnik bi bilo potrebne veliko konfiguracije, a z Dockerjem lahko napišemo svojo različico operacijskega sistema (v mojem primeru je to večino časa Alpine), tako da mu napišemo ukaze, kaj natanko, kako in v kakem zaporedju naj naredi, tj. zgradi strežniško kodo ter jo požene. Hkrati so te različice operacijskega sistema izjemno majhne, ker Docker ne poganja teh operacijskih sistemov (oziroma kontejnerjev) kot virtualno napravo, ampak si v resnici izposodi osrednji del operacijskega sistema, na katerem teče, tako imenovano jedro (angl. *kernel*), in ga uporablja v svojih kontejnerjih. Hkrati pa je ta kontejner izoliran od matičnega sistema in drugih kontejnerjev, razen če se mu eksplicitno ne naroči, naj obstaja kaka povezava med sistemi.

To pomeni, da potrebujete izjemno malo konfiguracije, da bi pognali strežnik in se vključili v mrežo SMTP2 protokola.

Poleg vseh izjemnih lastnosti sem v kontejnerje vključil generacijo SSL certifikatov. To v resnici povzroči, da se namesto HTTP protokola uporablja HTTPS. Certifikate nam generira Let's Encrypt, neprofitna organizacija, posvečena temu, da bi naredili internet varnejši. Velike zahvale gredo njim.

Koda je dostopna na naslednjih povezavah:

- Strežniška koda (*backend*) (12) - pod GPL-3 licenco
- Spletna stran (*frontend*) (17) - pod LGPL-2.1 licenco
- Docker konfiguracija (18) - pod Unlicense licenco

Strežniki, ki poganjajo kodo SMTP2 protokola, so nameščeni na naslednjih domenah in so prosto dostopni za vse za uporabo v namene te raziskovalne naloge:

- https://mail1.smtp2.tk
- https://mail2.smtp2.tk

ZAKLJUČEK

Po končani raziskovalni nalogi lahko ugotovimo kako veliko načrtovanja je potrebnega, da se nov protokol začne uporabljati, zakaj in kakšne prednosti in slabosti imajo protokoli, katere danes uporabljamo za prenos elektronske pošte po internetu, kako deluje večina današnjih protokolov za elektronsko pošto, kako pomembna je varnost protokolov in njihovih implementacij, kako lahko zagotovimo to varnost v novih protokolih ter kako narediti učinkovit in minimalen protokol.

S to raziskovalno nalogo sem želel ustvariti nek popolnoma nov protokol za elektronsko pošto. Želel sem, da bi bil ta protokol minimalen in učinkovit, da bi gradil na sodobni tehnologiji. To mi je tudi uspelo, čeprav je bilo potrebnega veliko načrtovanja. Varnost podatkov je dandanes zelo pomembna, stari protokoli niso bili nikoli zamišljeni z varnostjo. Internet ni več tak, kakršen je bil pred 20 leti. Danes je ogromno hekerjev in ljudi, ki hočejo zlorabljati internet, zato je kriptografija in zasebnost pomembna danes bolj kot kadarkoli. To zasebnost in varnost nam ponuja HTTPS protokol.

Elektronska pošta je pomemben del našega življenja, a hkrati je zelo stara in nujno potrebujemo zamenjavo pri protokolih. Videli smo, kako so predlagali več protokolov, ki naj bi zamenjali obstoječi SMTP, a jim to ni uspelo. Živimo v 21. stoletju. Protokoli so zelo pomemben del našega življenja. Brez protokolov ne bi bilo življenja. Mislim, da je absurdno, da ne moremo izbrisati ali urediti sporočila v 21. stoletju. Dandanes vsi hitimo, zato so hitri in dobri protokoli še posebej zaželeni. Ne živimo več v 20. stoletju, računalniki so zelo razširjeni, prav tako uporaba elektronske pošte. Ne moremo si privoščiti več napak pri pisanju elektronske pošte, namreč hitro se lahko zgodi kaj slabega. Ne živimo več v eri, ko so vse delali z UDP in TCP protokoloma, dandanes se uporablja HTTP/HTTPS kot dodatek okoli UDP/TCP protokolov, ki je veliko bolj uporaben in preprostejši. Zato je žalostno (če ne že sramotno) videti, da še vedno uporabljamo protokole, ki delujejo s staro tehnologijo, niso poznali kriptografije in še danes ne poznajo urejanja in brisanja sporočil. To je bil cilj moje raziskovalne naloge – odpraviti to.

VIRI, LITERATURA IN REFERENCE

1. **IETF.** Internet Engineering Task Force. [Elektronski] [Navedeno: 12. februar 2022.] https://www.ietf.org/.

2. —. RFC1796. *IETF RFCs.* [Elektronski] april 1995. [Navedeno: 12. februar 2022.] https://datatracker.ietf.org/doc/html/rfc1796.

3. **Postel, Jonathan B.** RFC788. *IETF RFCs.* [Elektronski] november 1981. [Navedeno: 12. februar 2022.] https://datatracker.ietf.org/doc/html/rfc788.

4. **Contributors, IETF.** RFC1869. *IETF RFCs.* [Elektronski] november 1995. [Navedeno: 12. februar 2022.] https://datatracker.ietf.org/doc/html/rfc1869.

5. —. RFC1652. *IETF RFCs.* [Elektronski] julij 1994. [Navedeno: 12. februar 2022.] https://datatracker.ietf.org/doc/html/rfc1652.

6. —. RFC4021. *IETF RFCs.* [Elektronski] marec 2005. [Navedeno: 12. februar 2022.] https://datatracker.ietf.org/doc/html/rfc4021 .

7. **Contributors, Microsoft Documentation.** Custom date and time format strings. *Microsoft Docs for .NET.* [Elektronski] Microsoft, 20. november 2021. [Navedeno: 13. februar 2022.] https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings.

8. **Contributors, Mozilla.** Content Type - HTTP Headers. *Mozilla MDN.* [Elektronski] [Navedeno: 12. februar 2022.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type.

9. **IANA.** Media Types. *IANA.* [Elektronski] 01. februar 2022. [Navedeno: 12. februar 2022.] https://www.iana.org/assignments/media-types/media-types.xhtml.

10. **Monitor.** ŠIFRIRANJE E-POŠTE S PGP IN S/MIME NI VARNO. *Monitor.* [Elektronski] 14. maj 2018. [Navedeno: 13. februar 2022.] https://www.monitor.si/novica/sifriranje-e-poste-s-pgp-in-s-mime-ni-varno/185499/.

11. **Bernstein, Daniel J.** Internet Mail 2000. *cr.yp.to.* [Elektronski] 2000. [Navedeno: 12. februar 2022.]

12. **Contributors, SMTP2.** SMTP2 implementacija. *GitHub.* [Elektronski] [Navedeno: 12. februar 2022.] https://github.com/mytja/SMTP2.

13. **Merchant, Brian.** How Email Open Tracking Quietly Took Over the Web. *Wired.* [Elektronski] 11. december 2017. [Navedeno: 12. februar 2022.] https://www.wired.com/story/how-email-open-tracking-quietly-took-over-the-web/.

14. **M&M Global.** Web Traffic Down 40% In Google Blackout. *MediaPost.* [Elektronski] 19. avgust 2013. [Navedeno: 12. februar 2022.] https://www.mediapost.com/publications/article/207076/web-traffic-down-40-in-google-blackout.html.

15. **Ševerkar, Mitja.** SMTP2: SSV naj zaradi večje varnosti ustvari začasen ključ za verifikacijo sporočil. *GitHub.* [Elektronski] 11. oktober 2021. [Navedeno: 12. februar 2022.] https://github.com/mytja/SMTP2/issues/6.

16. **Contributors, ClamAV.** ClamAV. [Elektronski] [Navedeno: 12. februar 2022.] https://www.clamav.net/.

17. **Contributors, SMTP2.** smtp2-frontend - Frontend za implementacijo SMTP2 protokola. *GitHub.* [Elektronski] [Navedeno: 12. februar 2022.] https://github.com/mytja/smtp2-frontend.

18. —. SMTP2-docker - Docker konfiguracija za SMTP2 backend, frontend in pa antivirusen engine. *GitHub.* [Elektronski] [Navedeno: 12. februar 2022.] https://github.com/mytja/SMTP2-docker.

19. **Ocepek, Aleš.** UPORABNIKI E-POŠTE. *Mladi raziskovalci.* [Elektronski] 2011. [Navedeno: 12. februar 2022.] https://mladiraziskovalci.scv.si/ogled?id=1151.

20. **Contributors, Wikipedia.** Internet Mail 2000. *Wikipedia.* [Elektronski] [Navedeno: 12. februar 2022.] https://en.wikipedia.org/wiki/Internet_Mail_2000.

21. —. Internet Message Access Protocol. *Wikipedia.* [Elektronski] [Navedeno: 12. februar 2022.] https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol.

22. —. Post Office Protocol. *Wikipedia.* [Elektronski] [Navedeno: 12. februar 2022.] https://en.wikipedia.org/wiki/Post_Office_Protocol.

23. —. JSON Web Token. *Wikipedia.* [Elektronski] [Navedeno: 12. februar 2022.] https://en.wikipedia.org/wiki/JSON_Web_Token.

24. —. Simple Mail Transfer Protocol. *Wikipedia.* [Elektronski] [Navedeno: 12. februar 2022.] https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol.

25. *Za boljšo dostavljivost elektronske pošte.* **Zupančič, Matic.** Letnik 30, Številka 11, s.l. : Monitor, 2020, Izv. XI. ISSN 1318-1017.

26. **Ševerkar, Mitja.** Remove e-mail passwords from /message/get endpoint. *GitHub.* [Elektronski] 25. oktober 2021. [Navedeno: 12. februar 2022.] https://github.com/mytja/SMTP2/issues/13.