Postmail Server Standard

Protocol and Expected Behaivour

1 Ports and interaction protocol.

Postmail uses TCP and a standard HTTP server as interface for communication between parties. The expected port on which the HTTP service should run is port *27059*. Since HTTP is an unencrypted protocol and Postmail was designed to run exposed to a wide network of possibly malevolent parties, port `27059` should never be exposed publicly, but rather be forwarded through a reverse-proxy[1] that encrypts communications as described by the HTTPS standard to port `27050`.

An SSL certificate may be obtained from a certificate authority or by self-generation, the latter being used by parties not in possesion of a FQDN and possibly prompting a security message in Postmail clients for increased awareness.

Note 1. The standard exposed port through which HTTPS communications will be performed with the Postmail Server is **`27050`**.

2 Behaivoural arguments, request paths, queries.

For communcations, Postmail uses HTTP requests at specific addresses to determine action, and uses query strings[2] to pass specific parameters that give useful context to the action. Parameters can be required or optional, and requests lacking required parameters are considered null and ignored. As such, Postmail requests can be performed using any HTTP compliant program, such as `curl` or any web browser.

Only the bare-minimum server to server protocols are specified in the standard for guaranteed compatibility, client to server communication may be decided independently for each implementation to suit every use-case scenario, though some recommendations will be provided later in this paper.

1. In computer networks, a reverse-proxy program is an application that forwards requests and packets from a hidden backend port to a exposed frontend port, sometimes with modifications or intentional limitations. NGINX and APACHE can be used to act as reverse proxies if configured properly.

2. Query Strings are a uniform resource locator that assignes a specified value to a specified parameter using the format `parameter=value`.

The purpose of Postmail is to retain the basic ammount of protocol to ensure basic self-compatibility yet still be extensible and tailored to each network and scenario.

3 Basic concepts, communication examples

Postmail functions on 2 core concepts, **notices** and **index** files. Files and messages sent are abstracted to their `(SHA256,hex)` checksums for most of this, until the entire file is needed such as for final distribution.

Postmail also functions on the priciple of own-storage, files and messages sent by foo to bar will be stored on foo's server, whereas bar's server will only recieve a notification of the mail being sent out. After bar logs into their own server, they will be able to see the notification containing receival instructions and bar will fetch the message and files directly from foo's server. During this, bar did not need to host any files on their server, since the nature of the received message did not neccesarily concern him. This also makes it possible to receive postmail without a server or an "account", you only need such infrastructure to send messages.

Postmail, as will be seen later[3], does not need arbitrary sender verification as does the email protocol. Since files don't come arbitrarly from the void to the receiver server but rather sit on the sender's server until retrieval, requests can be done automatically to the sender's server when a `notice` is received to confirm sender authenticity (a request is made to the presumed sender asking if they have sent file x to receiver bar, where the presumed sender can either confirm or deny and the notice is automatically ignored and classified as fraudulent). This is part of the protocol to make sure sender verification is wide-spread and will be detailed later.

Own-storage communication also allows for post-transaction message deletion. A message or specific part can be deleted all the way up to retrieval by the receiver, the receiver will be notified that a mail has been sent to them by you but, if delted, will not be able to fetch contents or even title.

At the same time, notices and indexes contain checksums of everything included, as such message tampering cannot happen, as an invalid checksums will lead to instant message dismissal. A message can be deleted, but it **cannot** be tampered with or modified in any way.

3.1 Basic communication example

3. During the /sendnotice verification routine

Let `foo` be the sender and `bar` the receiver. Let the file being transmitted be `x` and `SHA256(x) = y`. Let `token` be a secret sent by foo to bar for the purpose of authentification of retrieval. Furthermore, let both foo and bar have a personal client as well as a server used for this communication.

```
FOO connects to FOO's server.

FOO uploads x to FOO's server and generates token 'token'.

FOO sends a notice to BAR's server containing: x's checksum (from now referenced as x's
ID), sender (FOO@FOO_SERVER), user (BAR) and token.

BAR's server receives notice, contacts FOO's server providing sender, id and token to
verify message legitimacy

FOO's server returns a positive response, confirming legitimacy

BAR's server stores notice

BAR connects to BAR's server, receives notice

BAR makes request to FOO's server providing id, sender and token

FOO's server serves file X to BAR

BAR confirms file validity using notice checksum

---Communication is complete---

FOO may choose to delete file X as no more tokens are associated with it.
```

This is the core of Postmail communications and is the basis of the protocol. The core standardization of this procedure is done by emitting notices. For postmail standard compliance, notices should contain at LEAST:

- Data IDENTIFIER
- Authentification Token
- Sender string under the format (user:server) or (user@server)
- Destinatation ('user' if notice is a server request, full string if other form)

3.2 Index files, metadata delivery, complete mail format

The above protocol is enough for describing the communication of basic binary data or singular files, which is not how we typically think of mail. Useful mail contains several metadata contexts as well as multiple payloads, specifically Title, Message, Attachements, Reply Tag, etcetera. The solution I propose that would be compliant to the own-storage philosophy are Postmail Index files (*.pmail). These files would contain a data structure (default: JSON) that would contain these following keys and their values:

- Title: UTF-8 string containing Mail Title
- Message: `sha256,hex` identifier to message file
- Attachements: Array of `key:value` pairs where key is the filename of the attachement and value is the `sha256,hex` identifier.
- (Optional) Reply: `sha256,hex` identifier of the replied-to index file

Any other platform-specific or pseudo-standardized keys may be added in the future and the format is not limited to these keys, but these keys are part of the standard.

The index file is to be kept at a sane file size and not bloated unnecessarily. Clients may choose to discard index files with abnormal file sizes.

Example Postmail Index file in a JSON format:

```
{
  "title": "Group Assignment!",
  "message": "98ea6e4f216f2fb4b69fff9b3a44842c38686ca685f3f55dc48c5d3fb1107be4",
  "attachements": {
    "assignment.pdf": "24c80ab6d36b7ef9579aa40478d468c61bc2a120bf776b15afe0f2c6b2911488",
    "presentation.pptx": "29a1a3260ac643ceef6b792af40db6ea5eeecc910641deed2e62c1266009f26c"
  }
}
```

4 Recommandation: The purgatory system

Since Postmail does not require any non-protocolar spam filters or sender licensors to function, as opposed to traditional email, I propose the following system for dealing and limiting the impact of spam mail:

By default, every user that has been neither allowlisted or banned is limited to only 5 notices. Once that quota is reached, no more notices will be accepted ever, pending either ban or allowlisting. The receiver may then investigate the mail, having the following options:

- Allowlist if sender is trusted
- Ban if sender is presumed to be malevolent
- Increase notice quota to allow for more notices to be received without explicit allowlisting

The client will be shown mail in two pannels, allowed and purgatory. The allowed section containing allowlisted users can resemble a traditional mail pannel with every message in a single list, ordered cronologically, while the purgatory section can group notices by user (DNS resolving or any other method can be used to put duplicates under the same quota, for example a single server using two domains) where the client can choose to see all mail sent by the selected user and accordingly allowlist or ban them. For whitelisted users the index file of the mail can be downloaded and cached instantly as the notice is sent, since benevolence is presumed and we can assume the file is not harmful in any way (such as artificially big file size, exploits, binary data containing unwanted files, etcetera).

5 Documented Commands

Documented commands, their parameters and their expected behaivour, as well as recommandations. The following commands will be transmitted as file paths on HTTP request and the parameters as queries.

`/postmailping`
`https://server:27050/postmailping`

Returns string `postmailpong`. Is used to check if server hosts a Post Mail server.

`/get`
`args (user, id, token)`
`https://server:27050/get?user=foo&id=1&token=secret`

Fetches file with the `sha256,hex` id provided in the args, from user provided in the args. If file exists or token is invalid, returns string `"no"`. If file exists AND token is valid, serves file. Token then may be removed from list of valid tokens.

`/checkrepo`
`args (user, id, token)`
`https://server:27050/checkrepo?user=foo&id=1&token=secret`

Does the same as `/get` but returns string `"yes!"` instead of serving the file if file exists and token is valid. This command should not nullify token validity or perform any changes to the server state. Is called upon by `/sendnotice` from another presumed receiver server.

`/getinfo`
`args (user, file)`
`https://server:27050/getinfo?user=foo&file=bio`

Fetches Public Information file for specified user, this can be anything, but the standard recommends the following:

- `"bio"` - User's Bio
- `"name"` - User's Complete Name (with spaces or special characters)
- `"pgp"` - User's public (armored) PGP key
- `"pfp"` - User's Profile Picture (.png format)
- `"at"` - Alternate communication handles (phone, email, social medias, etc)
- `"status"` - User's current status
- `"site"` - User's website
- `"resume"` - User's Resume

Any other files may be added by the user, these are simply recommandations. If requested file is absent, a 404 may be given.

`/sendnotice`
`args (user, id, token, sender)`
`https://server:27050/sendnotice?id=1&token=secret&user=foo&sender=bar:barserver`

Notifies system that `bar:barserver` sent a mail with the given id, token, to the mentioned user on the server that received this request. Notices can also generally be exported as "Noticefiles" `(.notice)` which contains those args in JSON and can be used to give acces to a file through other means such as alternative communication channels or physical media. When a notice is sent, considering the id is valid on the sender's server, this represents the actual moment of "sending" a mail.

These requests are neccesary for postmail standard compliance in any server software. Data may be stored on the server and user client whoever one decides. A recommandation protocol for standard server-client communications may get release later on, but by far the simplest way to do this is reflect stored data on the server as a filesystem, and allow users acces it through `SFTP`, where then tools or shell scripts may be used to easily configure and perform operations.

6 Final Word, Future Installments, Licensing

Along with this paper, you should receive a digital signiture file, signed by PGP key with ID C2462D5103FA6059E1E3279C9E9299CD96C65EC6, any other installments relating to this paper signed on a date following the day this paper is signed may override any aspect of the protocol or paper, excepting the license under which this paper is released

The current state of email is represented by a oligopoly of email giants which force unrealistic standards of email security, which can only really be achived by the enterprise world, thus halting the federalisation of email. This has to stop and safety should be baked right into the protocol.

Thank you for reading this paper.

6.1 LICENSE