

mailmaint  
Internet-Draft  
Intended status: Informational  
Expires: 22 January 2026

B. Bucksch  
Beonex  
21 July 2025

PACC - Automatic configuration for mail servers, calendar and contacts  
sync  
draft-ietf-mailmaint-pacc-00

## Abstract

Set up a mail account with only email address and password. This uses the DNS SRV mechanism to find the configuration. It is meant for new mail servers that adapt their configuration to current best practices.

## About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at  
<https://benbucksch.github.io/pacc/draft-ietf-mailmaint-pacc.html>.  
Status information for this document may be found at  
<https://datatracker.ietf.org/doc/draft-ietf-mailmaint-pacc/>.

Discussion of this document takes place on the mailmaint Working Group mailing list (<mailto:mailmaint@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/mailmaint/>. Subscribe at <https://www.ietf.org/mailman/listinfo/mailmaint/>.

Source for this draft and an issue tracker can be found at  
<https://github.com/benbucksch/pacc>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 22 January 2026.

## Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Data format . . . . .	4
2.1. JSON config file . . . . .	4
2.2. servers . . . . .	5
2.2.1. Protocol . . . . .	6
2.2.2. URL . . . . .	8
2.2.3. authentication . . . . .	8
2.2.4. Username . . . . .	10
2.2.5. Multiple server protocols . . . . .	11
2.2.6. TLS validation . . . . .	11
2.3. provider . . . . .	11
2.3.1. name and shortName . . . . .	12
2.3.2. logo . . . . .	12
2.4. help . . . . .	12
2.4.1. documentation . . . . .	12
2.4.2. developer . . . . .	13
2.4.3. contact . . . . .	13
2.5. version . . . . .	13
2.6. JSON validation . . . . .	13
3. Config retrieval by mail clients . . . . .	14
3.1. Email address domain . . . . .	14
3.1.1. DNS SRV query . . . . .	14
3.1.2. DNS SRV response . . . . .	14
3.1.3. URL construction . . . . .	15
3.1.4. Config file retrieval . . . . .	15
3.2. MX domain . . . . .	15
3.2.1. DNS MX query . . . . .	16
3.2.2. DNS MX response . . . . .	16
3.2.3. DNS SRV query . . . . .	16
3.2.4. DNS SRV response . . . . .	16

3.3.	Example DNS SRV records for an email domain . . . . .	17
3.4.	Example DNS SRV records for an email hoster . . . . .	17
3.5.	No authentication for config . . . . .	17
4.	Config validation . . . . .	17
4.1.	User approval . . . . .	18
4.2.	Login test . . . . .	18
4.3.	OAuth2 windows . . . . .	18
4.4.	OAuth2 requirements . . . . .	18
5.	Alternatives considered . . . . .	19
5.1.	CAPABILITIES . . . . .	19
6.	Security Considerations . . . . .	20
6.1.	Risk . . . . .	20
6.2.	DNS . . . . .	20
6.3.	Config updates . . . . .	21
6.4.	Server security . . . . .	21
7.	IANA Considerations . . . . .	21
7.1.	Registration . . . . .	21
7.2.	Contents . . . . .	22
7.3.	Initial registration . . . . .	22
7.4.	Addition of DNS SRV Service Names pacc and pacc_mx . . . . .	23
7.4.1.	New records to be registered . . . . .	23
7.4.2.	Registration of SRV Service Name pacc . . . . .	23
7.4.3.	Registration of SRV Service Name paccmx . . . . .	23
7.4.4.	Use of _https as protocol . . . . .	24
8.	Conventions and Definitions . . . . .	24
9.	Normative References . . . . .	24
	Author's Address . . . . .	25

## 1. Introduction

This protocol allows users to set up their existing email account in a new mail client application, by entering only their name, email address, and password. The mail application, by means of PACC, will determine the parameters that are required, including the URL, authentication method, and OAuth2 server. Contact sync and calendar can also be set up automatically.

The protocol works by first determining the domain from the email address, and the querying a DNS SRV record at the email provider, which returns a URL to a JSON document, which then contains the configuration parameters.

While this PACC protocol fulfills the same purpose as the earlier Autoconfig protocol, it uses a different mechanism - DNS SRV instead of well-known URLs -, and requires that the server employs certain best practices, e.g. to use the full email address as username. PACC mandates Direct TLS for all servers. For OAuth2, it uses Open ID Connect and Dynamic Client Registration.

## 2. Data format

The configuration file MUST have the following data format and qualities.

The MIME type is text/json.

### 2.1. JSON config file

The following example shows the syntax of the JSON config file.

```
{
  servers: {
    jmap: {
      url: "https://jmap.example.net/session",
      authentication: [
        "OAuth2",
        "http-basic",
      ]
    },
    imap: {
      url: "imaps://imap.example.net",
      authentication: [
        "OAuth2",
        "sasl-plain",
      ]
    },
    pop3: {
      url: "imaps://pop3.example.net",
      authentication: [
        "OAuth2",
        "sasl-SCRAM-SHA-256-PLUS",
        "sasl-plain",
        "native"
      ]
    },
    smtp: {
      url: "imaps://pop3.example.net",
      authentication: [
        "OAuth2",
        "sasl-SCRAM-SHA-256-PLUS",
        "sasl-plain"
      ]
    },
    caldav: {
      url: "https://sync.example.net/calendar/",
      authentication: [
        "OAuth2",

```

```
        "http-basic",
    ]
},
carddav: {
    url: "https://sync.example.net/contacts/",
    authentication: [
        "OAuth2",
        "http-basic",
    ]
},
},
oAuth2: {
    issuer: "auth.example.net"
},
provider: {
    name: "ACME BestService WorkPlaceMail",
    shortName: "ACME",
    logo: [
        {
            url: "https://www.example.net/logo.svg",
            mimetype: "image/svg"
        },
        {
            url: "https://www.example.net/logo.png",
            mimetype: "image/png",
            width: "128",
            height: "128",
        },
        {
            url: "https://www.example.net/logo-512.png",
            mimetype: "image/png",
            width: "512",
            height: "512",
        }
    ]
},
},
help: {
    documentation: "https://help.example.net/howto/set-up-your-mail-app.html",
    developer: "https://developer.example.net/client-apps/",
    contact: "mailto:it@team.example.net"
},
version: "1.0"
}
```

## 2.2. servers

E.g.

```
servers: {  
  jmap: {  
    url: "https://jmap.example.net/session",  
    authentication: [ ... ]  
  },  
  imap: { ... }  
}
```

servers is a top-level property containing an object map. The key is a protocol name, defined in section "Protocol". The value is an object which describes the server and is defined in this section.

#### 2.2.1. Protocol

The protocol is the key of the object map inside the servers property.

It specifies which wire protocol to use for this server.

Protocol	URL scheme	Port	Name	Specification
jmap	https	443	JMAP	RFC 8620, RFC 8621, RFC 8887, RFC 9610 et al
imap	imaps	993	IMAP	RFC 9051 or RFC 3501, et al
pop3	pop3s	995	POP3	RFC 1939, RFC 5034
smtp	submissions	465	SMTP	RFC 5321, RFC 2822
caldav	https	443	CalDAV	RFC 4791
carddav	https	443	CardDav	RFC 6352
webdav	https	443	WebDAV	RFC 4918
managesieve	sieves	443	ManageSieve	RFC 5804, RFC 5228
ews	https	443	Exchange Web Services	
activeSync	https	443	ActiveSync	
graph	https	443	Microsoft Graph	

Table 1

Other protocol names can be added using an IANA registry. Their respective registrations need to define:

- \* Protocol: The protocol name, as appearing in the JSON
- \* URL scheme: Which URL scheme or schemes are to used in the URL
- \* Port: The default port number, if none is given in the URL
- \* Name: The commonly used name of the protocol

- \* Specification: Which RFCs or document specifies the protocol, and
- \* Additional Properties: (Optional) Protocol-specific JSON properties and their meaning.

#### 2.2.2. URL

E.g.

```
url: "https://jmap.example.net/session"  
url: "imaps://imap.example.net:1993"
```

The url property specifies the where and how to contact the server.

##### 2.2.2.1. URL scheme

- \* For protocols based on HTTPS or WebDAV, the URL scheme is https.
- \* For protocols based on WebSockets, the URL scheme is wss.
- \* TCP-based protocols might use other URL schemes, like imaps, as defined in section "Protocols".

##### 2.2.2.2. Hostname

The hostname in the URL MUST be the fully qualified domain name of the server which the client is supposed to contact for that protocol.

##### 2.2.2.3. Port

If no port is in the URL, the Port as defined in the "Protocols" or in the IANA registry is used as the default port. If the server port matches that default port, it SHOULD NOT be specified in the URL. If the URL contains an explicit port, that part MUST be used by the client.

##### 2.2.2.4. Path

For some protocols like IMAP, POP3 and SMTP, the path is empty.

#### 2.2.3. authentication

E.g.

```
authentication: [ "http-basic" ]  
authentication: [ "sasl-SCRAM-SHA-256-1" ]
```



The authentication property is an array that defines which authentication methods are available to use. Each of them MUST work.

#### 2.2.3.1. Authentication methods

- \* For HTTPS or WebSocket-based protocols, these can be authentication methods used in WWW-authenticate the HTTP header. The value is http- plus the HTTP authentication scheme name, case-insensitive. See RFC 7617 and 7616. E.g. http-basic for WWW-Authenticate: Basic, or http-NTLM.
- \* For TCP-based protocols, these can be SASL schemes. The value is sasl- plus the SASL scheme name in uppercase. E.g. sasl-SCRAM-SHA-256-1 or sasl-PLAIN.
- \* For OAuth2, the value is OAuth2.
  - With TCP, SASL OAUTHBEARER (current) or XOAUTH2 (deprecated) or successors may be used.
  - With HTTP, WWW-Authenticate: Bearer or a successor is used. See RFC 6750 Section 3.
  - The provider MUST adhere to the requirements defined in section "OAuth2 requirements" in this specification.

#### 2.2.3.2. Multiple authentication alternatives

The authentication array may contain multiple mechanisms for a single server. Each of them MUST work.

The client can choose which of those to use, based on implementation support, available authentication data, and client policy.

If none of the authentication methods are supported by the client, the client MUST ignore that server section and use the remaining server sections.

#### 2.2.3.3. Authentication verification and fallback

The client SHOULD test the configuration during setup, with an actual authentication attempt.

If the authentication fails, the client decides based on the authentication error code how to proceed. E.g. if the authentication method itself failed, or the error code indicates a systemic failure, the client MAY use another authentication method from the list.

If the authentication method is working, but the error code indicated that the username or password was wrong, then the client MAY ask the user to correct the password.

For that reason, the server SHOULD be specific in the error codes and allow the client to distinguish between

- \* an unsupported or non-working authentication method or other systemic failures
- \* the client being rejected by the server
- \* the user being blocked from login
- \* the user authentication failing due to wrong password or username
- \* other reasons

If the server were to return the same error code for all these cases, the client might tell the user that the password is wrong, and the user starts attempting other passwords, potentially revealing passwords to unrelated higher-value assets, which is highly dangerous.

If the authentication succeeded, the client SHOULD take note of the working configuration and use that for all subsequent connections, until an explicit reconfiguration occurs. During normal everyday operation, the client SHOULD NOT fallback nor attempt multiple different authentication methods.

#### 2.2.4. Username

For all protocols, the email address that the user entered during setup will be used by the client as username on the target protocol level.

The provider MUST ensure that any valid email address that the user might enter during setup is a valid username for all servers given in this configuration. This may require a mapping on the server level from email address to internal username. This mapping happens internally in the server and the client is not involved in this mapping.

### 2.2.5. Multiple server protocols

While PACC supports only a single server per protocol, it MAY give the client the choice of different protocols. Not all clients might implement all protocols. That's why the PACC file SHOULD contain all protocols that the provider offers to its users.

The client chooses which protocol to use, based on

- \* which protocols the client implements,
- \* the configuration returned, e.g. the config specifies only an OAuth2 authentication and the client either doesn't implement OAuth2, or there is a problem in the OAuth2 flow with this provider,
- \* client policy, e.g. the client preferring JMAP over IMAP.

Server protocols that the client does not support MUST be ignored and the client MUST continue to parse the other server sections, which may contain configs that the client understands and supports. The client ignores the file only if there is no supported and working config found.

### 2.2.6. TLS validation

In all cases where TLS is used, the client MUST validate the TLS certificate and ensure that the certificate is valid for the hostname given in this config. If not, or if the TLS certificate is otherwise invalid, the client MUST either disconnect or MAY warn the user of the dangers and ask for user confirmation. Such fallback with warning and confirmation is allowed only at original configuration and MUST NOT be allowed during normal everyday connection.

If the server had a valid TLS certificate during original configuration, and the TLS certificate is later invalid during normal connection, the client MUST disconnect.

As an exception, if the problem is that the TLS certificate expired recently, the client MAY choose to not consider that a failure during normal connection and MAY use other recovery mechanisms.

## 2.3. provider

provider property is a top-level property and contains name, shortName, and logo.

### 2.3.1. name and shortName

E.g.

```
name: "ACME BestService WorkPlaceMail"
shortName: "ACME"
```

The name property contains the name of the provider, e.g. as preferred by the marketing of the provider itself. It SHOULD be no longer than 30 characters, but MUST be no longer than 60 characters.

The shortName property contains the name of the provider, as typically used by end users. It SHOULD be no longer than 12 characters, and it MUST NOT be longer than 20 characters.

### 2.3.2. logo

The logo property contains an array of alternative logos of the provider. The client chooses the variant that best fits its UI and technical requirements.

Each object in the array contains the following properties: \* url - Where the logo can be downloaded. The client MAY download the logo file during configuration and save it locally. \* mimetype - The media type of the logo. It MUST start with image/. At least one of the logos SHOULD be of type image/svg or image/png. \* width - The width in pixels of the image in the file. Optional. Not given for SVG files. \* height - The height in pixels of the image in the file. Optional. Not given for SVG files.

It is RECOMMENDED to include either an SVG logo, or a PNG files of sizes 128x128 and 512x512, or all of these. Additional sizes and file formats MAY be included.

### 2.4. help

This is purely informational and not required for the automatic setup. All of the information is optional. A contact SHOULD be included.

#### 2.4.1. documentation

E.g. documentation: "https://help.example.net/howto/set-up-your-mail-app.html"

Records the user help webpage at the provider that describes the mail server settings.

The config in the PACC file does not necessarily have to match the config proposed on this webpage.

#### 2.4.2. developer

E.g. developer: "https://developer.example.net/client-apps/"

Webpage with information for mail application developers.

#### 2.4.3. contact

E.g. contact: "mailto:it@team.example.net"

Allows a direct contact from mail application developers to mail server administrators. This is useful to resolve issues with this configuration, e.g. when the configuration in the PACC file is outdated, or with the mail servers, e.g. when the mail server is misconfigured or otherwise has a compatibility or security problem.

This MUST NOT be a generic hotline for end users, nor the general company switchboard. It SHOULD be a way to reach the technical team of the provider, to resolve technical issues on the side of the provider.

Contains an URL with an email address (mailto: URL), phone number (tel: URL), or webpage (https URL) which contains this contact info. Note that problems might appear only after many years, so please ensure longevity of the contact.

#### 2.5. version

version property is a top-level property.

The version is 1.0 for the version defined in this specification. Higher versions are for future specifications. The client MUST NOT reject a config file solely based on the version number.

#### 2.6. JSON validation

The client SHOULD validate that the config file is valid JSON as per RFC 8259, and if the JSON syntax is invalid, the client SHOULD ignore the entire file. In contrast, if there are merely unknown JSON properties, the client MUST NOT ignore the file.

The client SHOULD read only the properties that are supported by the client, and MUST ignore the others that are unknown to the client.

The client may optionally want to validate the XML before parsing it. This is not required. If the client choses to validate, the validation MUST ignore unknown properties and MUST NOT drop or ignore a configuration that contains unknown properties. This is required to allow future extensions of the format without breaking existing clients.

### 3. Config retrieval by mail clients

The mail client application, when it needs the configuration for a given email address, will perform several steps to retrieve the configuration from the email domain or from the email hoster.

The client MAY perform both queries in parallel, but MUST give precedence to the results from the direct email domain, even if they return slower.

In the URLs below, %EMAILDOMAIN% shall be replaced with the email domain extracted from the email address that the user entered and wishes to use. For example, for "fred@example.net", the email domain is "example.net", and for "fred@test.cs.example.net", the email domain is "test.cs.example.net".

#### 3.1. Email address domain

First step is to directly ask the mail provider and allow it to return the configuration. This step ensures that the protocol is decentralized and the mail provider is in control of the configuration issued to mail clients.

##### 3.1.1. DNS SRV query

The client makes a DNS SRV lookup for \_pacc.\_https on the domain of the user's email address:

DNS SRV \_pacc.\_https.%EMAILDOMAIN%.

e.g.

DNS SRV \_pacc.\_https.example.com.

##### 3.1.2. DNS SRV response

The DNS server returns a DNS SRV record with the hostname of the HTTPS server where the PACC config file can be found:

\_pacc.\_https.%EMAILDOMAIN%. SRV 0 0 443 %HOSTNAME%.

e.g.

```
_pacc._https.example.com. SRV 0 0 443 pacc.example.com.
```

The port **MUST** be 443. The priority and weight may be disregarded. Only the hostname is extracted from the response.

### 3.1.3. URL construction

The client takes the hostname, and constructs the PACC retrieval URL from it, by using https, that hostname, standard port 443, and path /.well-known/pacc.json:

```
https://%HOSTNAME%/.well-known/pacc.json
```

e.g.

```
https://pacc.example.com/.well-known/pacc.json
```

### 3.1.4. Config file retrieval

The client retrieves the https URL constructed in the previous step.

The HTTP Accept request header **MUST** allow text/json, e.g. Accept: text/json or text/\* or \*/\*.

The returned file **MUST** be a PACC file following the specifications in section "Data format".

## 3.2. MX domain

Many companies do not maintain their own mail server, but let their email be hosted by a hosting company, which is then responsible for the email of dozens or thousands of domains. For these hosters, it may be difficult to set up the configuration server with valid TLS certificate for each of their customers, and to convince their customers to modify their root DNS specifically for PACC. To handle such domains, the protocol first needs to find the domain of the party hosting the email.

If the query on the email domain as described above yields no result, the client **SHOULD** perform a DNS MX lookup on the email domain, and retrieve the MX server hostname for that domain and look for a PACC file for the MX hostname, using the following mechanism.

### 3.2.1. DNS MX query

The client makes a DNS MX lookup on the domain of the user's email address:

DNS MX %EMAILDOMAIN%.

e.g.

DNS MX example.com.

### 3.2.2. DNS MX response

The DNS server returns the DNS MX records with the hostnames of the SMTP servers that accept email for the user's email address:

example.com MX %PRIORITY% %MXSERVER%.

e.g.

example.com MX 10 beetruche1.mx.example.net.

example.com MX 10 beetruche2.mx.example.net.

example.com MX 30 beetruche3.mx.example.net.

The client takes only the highest priority result, i.e. the one with the lowest priority number (in this example 10), given that the priority numbers are in reverse order. If there are multiple responses with the same lowest priority number, the client takes only the first one. The client takes the hostname of this MX server as result %MXSERVER%.

### 3.2.3. DNS SRV query

The client makes a DNS SRV lookup for \_paccmx.\_https on the hostname of the MX server retrieved in the last step:

DNS SRV \_paccmx.\_https.%EMAILDOMAIN%.

e.g.

DNS SRV \_paccmx.\_https.beetruche1.mx.example.net.

Please note that the service part in this case is \_paccmx, not \_pacc.

### 3.2.4. DNS SRV response

The DNS server returns a DNS record with the hostname of the HTTPS server where the PACC config file can be found:



```
_paccmx._https.%MXSERVER%. SRV 0 0 443 %HOSTNAME%.
```

e.g.

```
_paccmx._https.beetruchel.mx.example.net. SRV 0 0 443 pacc.example.net.
```

The port MUST be 443. The priority and weight may be disregarded. Only the hostname is extracted from the response.

From here, the client continues as described above under sections "URL construction" and "Config file retrieval".

### 3.3. Example DNS SRV records for an email domain

In this example, example.com hosts its email and PACC config file itself:

```
$ORIGIN example.com.  
_pacc._https SRV 0 0 443 pacc.example.com.  
pacc A 192.0.2.9
```

### 3.4. Example DNS SRV records for an email hoster

In this example, example.com hosts its email with hoster example.net:

```
$ORIGIN example.com.  
@ MX 10 beetruchel.mx.example.net  
  
$ORIGIN example.net.  
_paccmx._https.beetruchel.mx SRV 0 0 443 pacc.example.net.  
pacc A 192.0.1.9
```

### 3.5. No authentication for config

Any of the above URLs for retrieving the config file MUST NOT require authentication, but MUST be public.

This is because the configuration information in the PACC file includes the authentication method. Without the PACC file, the client does not know which authentication method to use. Given that this information is required for authentication, the PACC config file itself cannot require authentication.

## 4. Config validation

#### 4.1. User approval

Independent of the mechanisms used to find the configuration, before using that configuration, you SHOULD display that configuration to the end user and let him confirm it. While doing so:

- \* At least the second-level domain name(s) of the hostnames involved MUST be shown clearly and with high prominence.
- \* To avoid spoofing, the client MUST NOT cut off parts of long second-level domains. At least 63 characters MUST be displayed.

#### 4.2. Login test

After the user confirmed the configuration, you SHOULD test the configuration, by attempting a login to each server configured. Only if the login succeeded, and the server is working, should the configuration be saved and retrieving and sending mail be started.

#### 4.3. OAuth2 windows

If the configuration contains OAuth2 authentication, or any other kind of authentication that uses a web browser with URL redirects, you MUST show the full URL or the second-level domain of the current page to the end user, at all times, including after page changes, URL changes, or redirects. The authentication start URL may be the email hoster, but it may redirect to a corporate server for login, and then back to the hoster. This allows for setups where the hoster is not allowed to see the plaintext passwords.

Showing the URL or hostname allows the end user to verify that he is logging in at the expected page, e.g. the login server of their company, instead of the email hoster's page. It is important that the user verifies that he enters the passwords on the right domain.

#### 4.4. OAuth2 requirements

If OAuth2 is used, the OAuth2 server MUST adhere to the OAuth Profile for Open Public Clients (<https://datatracker.ietf.org/doc/draft-ietf-mailmaint-oauth-public/>).

Particularly, the Dynamic Client Registration MUST be implemented and give a working Client ID in response HTTP calls defined by the specification. Alternatively, the Client ID open MUST be accepted without client secret. Failure to do so implies a cartell or monopolistic behavior to lock out competing email applications from fulfilling their purpose on behalf of end users, which may be contrary to laws in multiple countries.

The specifications also contain requirements for expiry times and the login page, which are needed for mail client applications to work.

The OAuth2 scopes defined in the specification **MUST** be included and **MUST** give access to the servers published in PACC.

A single token **MUST** work for all servers returned in PACC, so that a single user login is sufficient for all services. For that purpose, the client will include all relevant scopes in the authentication requests.

## 5. Alternatives considered

### 5.1. CAPABILITIES

Deployments in the wild from actual ISPs show that protocol-specific commands to find available authentication methods, e.g. IMAP CAPABILITIES or POP3 CAPA, are not reliable. Many email servers advertize authentication methods that do not work.

Some IMAP servers are default configured to list all SASL authentication methods that have corresponding libraries installed on the system, independent on whether they are actually configured to work. The client receives a long list of authentication methods, and many of them do not work. Additionally, the server response may be only "authentication failed" and may not indicate whether the method failed due to lack of configuration, or because the password was wrong. Because some authentication servers lock the account after 3 failed login attempts, and it may also fail due to unrelated reasons (e.g. username form, wrong password, etc.), the client cannot blindly issue countless login attempts. Locking the account must be avoided. So, simply attempting all methods and seeing which one works is not an option for the email client.

Additionally, some email servers advertize Kerberos / GSSAPI, but when trying to use it, the method fails, and also runs into a long 2 minute timeout in some cases. End users consider that to be a broken app.

Additionally, such commands are protocol specific and have to be implemented in multiple different ways.

Finally, some non-mail protocols may not support capabilities commands that include authentication methods.

## 6. Security Considerations

### 6.1. Risk

If an attacker can provide a forged configuration, the provided mail hostname and authentication server can be controlled by the attacker, and the attacker can get access to the plain text password of the user. The attack can be implemented as man-in-the-middle, so the end user might receive mail as expected and never notice the attack.

An attacker gaining the plain text password of a real user is a very significant threat for the organization, not only because mail itself can contain sensitive information and can be used to issue orders within the organization that have wide-ranging impact, but given single-sign-on solutions, the same username and password may give access to other resources at the organization, including other computers or, in the case of admin users, even administrative access to the core of the entire organization.

Multi-factor authentication might not defend against such attacks, because the user may believe to be logging into his email and therefore comply with any multi-factor authentication steps required.

### 6.2. DNS

This protocol relies on DNS SRV and DNS MX lookups to find the PACC file. DNS requests are not signed and can therefore be intercepted, spoofed and manipulated. This would allow the attacker to change the PACC file URL and return email and authentication servers under the attacker's control, stealing passwords.

One possible mitigation is to check whether the domain of the PACC file URL matches the user's email address domain. However, that will not be the case for the majority of domains, which are served by email hosters.

Another possible mitigation is to use multiple different DNS servers and verify that the results match, e.g. to use the native DNS resolver of the operating system, and additionally also query a hardcoded DoH (DNS over HTTPS) server.

Nonetheless, the result should be used with care. If such configs are used, the end user MUST explicitly confirm the config, particularly the resulting second-level domains. See section "User approval".

### 6.3. Config updates

Part of the security properties of this protocol assume that the timeframe of possible attack is limited to the moment when the user manually sets up a new mail client. This moment is triggered by the end user, and a rare action - e.g. maybe once per year or even less, for typical setups -, so an attacker has limited chances to run an attack. While not a complete protection on its own, this reduces the risk significantly.

However, if the mail client does regular configuration updates using this protocol, this security property is no longer given. For regular configuration updates, the mail client **MUST** use only mechanisms that are secure and cannot be tampered with by an active attacker. Furthermore, the user **SHOULD** still approve config changes.

But even with all these protections implemented, the mail client vendor should make a security assessment for the risks of making such regular updates. The mail client vendor should consider that servers can be hacked, and most users simply approve changes proposed by the app, so these give only a limited protection.

### 6.4. Server security

Given that mail clients will trust the configuration, the server delivering the PACC file needs to be secure. A static web server offers better security. The server software **SHOULD** be updated regularly and hosted on a dedicated secure server with all unnecessary services and server features turned off.

## 7. IANA Considerations

### 7.1. Registration

IANA will create the following registry in a new registry group called "PACC":

Registry Name: "PACC Protocol Type Names"

Registration Procedure: Specification Required, per RFC 8126, Section 4

Designated Expert: Ben Bucksch, author of this document.

## 7.2. Contents

Table, with fields Protocol (alphanumeric), URL scheme, Port (default port number, if not specified in the URL), Name, Specification, and Additional Properties

The registrations need to define \* Protocol: The protocol name, as appearing in the JSON \* URL scheme: Which URL scheme or schemes are to used in the URL \* Port: The default port number, if none is given in the URL \* Name: The commonly used name of the protocol \* Specification: Which RFCs or document specifies the protocol, and \* Additional Properties: (Optional) Protocol-specific JSON properties and their meaning.

## 7.3. Initial registration

Protocol	URL scheme	Port	Name	Specification	Additional Properties
jmap	https	443	JMAP	RFC 8620, RFC 8621, RFC 8887, RFC 9610 et al	
imap	imaps	993	IMAP	RFC 9051 or RFC 3501, et al	
pop3	pop3s	995	POP3	RFC 1939, RFC 5034	
smtp	submissions	465	SMTP	RFC 5321, RFC 2822	
caldav	https	443	CalDAV	RFC 4791	
carddav	https	443	CardDav	RFC 6352	
webdav	https	443	WebDAV	RFC 4918	
managesieve	sieves	443	ManageSieve	RFC 5804, RFC 5228	

ews	https	443	Exchange Web	
			Services	
+-----+	+-----+	+-----+	+-----+	+-----+
-----+				
activeSync	https	443	ActiveSync	
+-----+	+-----+	+-----+	+-----+	+-----+
-----+				
graph	https	443	Microsoft	
			Graph	
+-----+	+-----+	+-----+	+-----+	+-----+
-----+				

Table 2

The Additional Properties field is empty in all of the initial values.

#### 7.4. Addition of DNS SRV Service Names pacc and pacc\_mx

##### 7.4.1. New records to be registered

Service Name	Transport Protocol	References
pacc	https	[PACC] This document
paccmx	https	[PACC] This document

Table 3

##### 7.4.2. Registration of SRV Service Name pacc

In the "Service Name and Transport Protocol Port Number" registry:

- \* Service Name: pacc
- \* Transport Protocol(s): https
- \* Assignee: Ben Bucksch
- \* Contact: ben.bucksch@beonex.com
- \* Description: PACC - Automatic configuration of mail servers
- \* Reference: [PACC] This document
- \* Port Number: -
- \* Service Code: -
- \* Known Unauthorized Uses: -
- \* Assignment Notes: -

##### 7.4.3. Registration of SRV Service Name paccmx

In the "Service Name and Transport Protocol Port Number" registry:

- \* Service Name: paccmx



- \* Transport Protocol(s): https
- \* Assignee: Ben Bucksch
- \* Contact: ben.bucksch@beonex.com
- \* Description: PACC - Automatic configuration of mail servers via MX
- \* Reference: [PACC] This document
- \* Port Number: -
- \* Service Code: -
- \* Known Unauthorized Uses: -
- \* Assignment Notes: -

#### 7.4.4. Use of \_https as protocol

While tcp is typically used for proto, https is also valid and is more precise in this case.

https is already defined as service, and therefore may also be used as Transport Protocol, per the definition of RFC 2782: "Proto ... any name defined by Assigned Numbers or locally may be used (as for Service)".

In case https cannot be used as transport protocol, tcp will be registered instead.

References: \* RFC 2782 Section "The format of the SRV RR" \* RFC 6335 Section 5.2 \* <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbelrs.xhtml> (<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbelrs.xhtml>)

## 8. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 9. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

Author's Address

Ben Bucksch  
Beonex  
Email: [ben.bucksch@beonex.com](mailto:ben.bucksch@beonex.com)