mailmaint
Internet-Draft
Intended status: Informational
Expires: 5 March 2026

D. Eggert
M. Diephouse
Apple Inc

1 September 2025

Automatic Configuration of Email, Calendar, and Contact Server Settings
draft-eggert-mailmaint-uaautoconf-01

Abstract

This document specifies an automatic configuration mechanism for email, calendar, and contact user agent applications. Service providers publish standardized configuration information that user agent applications retrieve and use to simplify server setup procedures.

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 5 March 2026.

Table of Contents

1. Introduction

Manual configuration of email, calendar, and contact user agent applications requires users to correctly specify numerous technical parameters including server hostnames, port numbers, and authentication protocols. This manual process frequently results in configuration errors and setup failures, even among technically skilled users.

This document defines a mechanism that significantly simplifies this configuration process. Service providers can publish standardized configuration data that user agents can automatically retrieve and use. In most cases, users need only provide their email address and account password to complete the setup.

For service providers that support the OAuth Profile for Open Public Clients [OAuthPublic], this mechanism also enables automatic OAuth configuration. The user agent automatically determines all necessary details to set up OAuth authentication for the associated account.

2. Document Conventions

In protocol examples, this document uses a prefix of "C: " to denote lines sent by the user agent to the server, and "S: " for lines sent by the server to the user agent. Lines prefixed with "// " are comments explaining the previous protocol line. These prefixes and comments are not part of the protocol. Lines without any of these prefixes are continuations of the previous line, and no line break is present in the protocol unless specifically mentioned.

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.

3. Overview

The automatic configuration process begins when a user wants to set up their user agent application. The user agent requests the user’s email address and then uses the domain portion of that address to retrieve configuration data for the account.

Configuration retrieval uses the Well-Known URIs mechanism defined in [RFC8615]. The specific suffix for this configuration is user-agent-configuration. Section 5.2 provides detailed information about how to retrieve this configuration.

The configuration is provided as a JSON document (detailed in Section 4) that informs the user agent about:

* Human-readable information about the service provider
* Supported protocols for this domain
* Server endpoints for each protocol
* OAuth Profile for Open Public Clients support availability

The user agent uses this configuration to determine whether the provider supports protocols that the user agent can also use, helping decide whether to continue with the setup process. For example, an email user agent might check for support of JMAP, IMAP, POP3, and SMTP protocols.

Next, the user agent typically connects to the relevant services to verify which authentication methods they support and whether the user agent can use those methods. For instance, if the user agent needs both IMAP and SMTP, it will connect to both servers to check their capabilities and confirm compatibility with the user agent’s supported authentication mechanisms.

Based on the available options, the user agent then proceeds with one of two approaches:

1. Continue configuration using OAuth Profile for Open Public Clients, or
2. Request the user’s password for traditional authentication

Finally, the user agent uses the obtained credentials (OAuth tokens or password) to connect to the servers specified in the configuration document and validates that the credentials work correctly with the desired protocols.

4. JSON Configuration

The configuration document describes which services are available and their corresponding endpoints for a specific domain.

The configuration file MUST conform to the JSON schema specification provided in Appendix A.

The configuration uses the media type application/json. The HTTP server MUST set the corresponding Content-Type header as specified in [RFC9110].

4.1. Example

The following example demonstrates a typical JSON configuration:

```
{
    "protocols": {
        "jmap": {
            "url": "https://jmap.example.com/session"
        },
        "imap": {
            "host": "imap.example.com"
        },
        "pop3": {
            "host": "pop3.example.com"
        },
        "smtp": {
            "host": "smtp.example.com"
        },
        "caldav": {
            "url": "https://sync.example.com/calendar/"
        },
        "carddav": {
            "url": "https://sync.example.com/contacts/"
        }
    },
    "authentication": {
        "oauth-public": {
            "issuer": "auth.example.com"
        },
        "password": true
    },
    "info": {
        "provider": {
            "name": "Example Provider Name",
            "shortName": "Example",
            "logo": [
                {
                    "url": "https://www.example.net/logo.svg",
                    "content-type": "image/svg+xml"
                }
            ]
        },
        "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"]
        }
    }
}
```

4.2. Protocols

The protocols object specifies all protocols that the domain supports and identifies the server endpoint for each protocol.

For the following HTTP-based protocols:

* JMAP defined in [RFC8620],

* CalDAV defined in [RFC4791], and

* CardDAV defined in [RFC6352]

* WebDAV defined in [RFC4918]

each protocol entry MUST include a url value containing a URL with the https scheme. This URL MUST specify an endpoint that serves the corresponding protocol. The server MUST use the default port 443 for HTTPS, and the URL MUST NOT include an explicit port number.

Hostnames in URLs MUST support Internationalized Domain Names (IDNs) as defined in [RFC5890] and [RFC5891]. Service providers MAY use either Unicode labels (u-labels) or ASCII Compatible Encoding labels (a-labels) in their configuration. User agents MUST support both forms and handle the conversion between them as specified in [RFC5891].

Example:

```
"jmap": {
    "url": "https://jmap.example.com/session"
}
```

For the following text-based protocols:

* IMAP defined in [RFC9051],

* SMTP defined in [RFC5321], and

* POP3 defined in [RFC1939]

* ManageSieve defined in [RFC5804]

each protocol entry MUST include a host value specifying the hostname of the server that provides the corresponding protocol. The server MUST use the default port number for that protocol, with connections secured using TLS. Section 5.4.3 goes into details about TLS and which ports to use.

Hostnames MUST support Internationalized Domain Names (IDNs) as defined in [RFC5890] and [RFC5891]. Service providers MAY use either Unicode labels (u-labels) or ASCII Compatible Encoding labels (a-labels) in their configuration. User agents MUST support both forms and handle the conversion between them as specified in [RFC5891].

Example:

```
"imap": {
    "host": "imap.mail.example.com"
}
```

4.3. Authentication

The authentication object specifies which authentication mechanisms the provider supports.

The nested password value indicates whether the provider supports username and password authentication.

The nested oauth-public object contains a single issuer value that specifies the authorization server issuer identifier. User agents can use this identifier for authentication via [OAuthPublic]. The presence of the oauth-public object indicates that the provider supports OAuth.

4.4. Informational

The info object contains information that user agents can present to users. The only required field is info/provider/name, which SHOULD clearly identify the provider to the user. The optional info/provider/shortName provides a shorter version of name.

The info/provider/logo array can contain one or more entries pointing to the provider’s logo images. User agents can use these to display the provider’s logo to users.

The info/help/documentation field provides a URL that can be presented to users for additional information about the provider. The info/help/developer field contains a URL with information specifically for user agent developers. The info/help/contact array lists URLs that users can use to contact the provider, such as mailto URLs.

A minimal info example:

```
"info": {
    "provider": {
        "name": "Example Provider Name"
    }
}
```

4.4.1. Provider Information

provider object can have name, shortName, and logo properties.

The name and shortName values are intended for display to the user.

The name property specifies the provider's display name, such as the name used in the provider's marketing materials. It SHOULD be no longer than 30 characters, but MUST be no longer than 60 characters.

The shortName property is optional. It specifies a brief version of the provider's name, as commonly recognized by users. It SHOULD be no longer than 12 characters, and it MUST NOT be longer than 20 characters.

Both name and shortName MUST NOT include any Control-characters and SHOULD NOT include any excessive white space.

The logo property contains an array of provider logo variants. User agents select the variant that best matches their UI requirements and technical constraints.

Each object in the array contains the following properties:

* url - Location where the logo can be retrieved. User agents MAY download the logo file during configuration and store it locally.

* content-type - Media type of the logo image. This follows [RFC2045] media type format, specifying the main type and subtype without parameters. The main media type MUST be image.

* width - Image width in pixels. Optional. Omitted for SVG files.

* height - Image height in pixels. Optional. Omitted for SVG files.

The logos SHOULD at least include one of each

* image/svg+xml

* image/png with size 128 by 128

* image/png with size 512 by 512

If the server provides images in the SVG format, these images MUST use the SVG Tiny PS Profile specified in [I-D.svg-tiny-ps-abrotman].

4.4.2. Help

These fields are intended to allow a service provider to provide additional information about configuring user agents.

The documentation property should link to a document intended for users. It can provide additional information to users about how to configure their user agent. A user agent MAY choose to display a link to this URL. User agent SHOULD NOT display links with a URL scheme other than https.

The developer property should link to a document intended for user agent developers. The document this URL links to can provide additional information to the developer.

The contact property should provide a way for user agent developers to contact the service providers. It is _not_ intended as a way for users to contact the service provider. User agents MUST NOT display this link in their user interface. This URL would typically be a mailto URL or a URL linking to a contact form for developers to use.

5. Flow

Section 3 provides a high-level overview of the automatic configuration process. This section details the individual steps involved in this process.

5.1. Email Address

During the initial configuration process, the user agent requests the user’s email address. User agents SHOULD accept any valid mailbox format as specified in Section 3.4 of [RFC5322].

5.2. Configuration Source

After the user provides their email address, the client extracts the domain part of the user’ s email address according to Section 3.4.1 of [RFC5322]. For example

Text Entered by User	domain
jdoe@foo.example.com	foo.example.com
"J Doe" <jdoe@foo.example.com>	foo.example.com
<jdoe@foo.example.com>	foo.example.com

Table 1: Domain from email address

5.2.1. Well-Known Uniform Resource Identifier

The user agent retrieves configuration data over HTTP using the format described in Section 4.

Configuration retrieval uses [RFC8615] Well-Known URIs. The user agent constructs the URI using the _domain_ from the user's email address and the _configuration name_ user-agent-configuration. The detailed URI construction process is specified in [RFC8615].

The domain name MUST support Internationalized Domain Names (IDNs) as defined in [RFC5890] and [RFC5891]. User agents MUST handle both Unicode labels (u-labels) and ASCII Compatible Encoding labels (a-labels) when processing domain names from email addresses, and convert them appropriately for URI construction as specified in [RFC5891].

The URI follows this template:

https://{domain}/.well-known/user-agent-configuration

If the user' s email address' domain is foo.example.com, the user agent constructs the following URI:

https://foo.example.com/.well-known/user-agent-configuration

This URI always uses the https scheme. User agents MUST retrieve the configuration file only via https (HTTP over TLS).

The user agent MUST validate that the connection is secured by TLS. See Section 7.4 for details about TLS with respect to retrieving the configuration.

The media type of the configuration is application/json, and the HTTP server MUST set a corresponding Content-Type type header. See [RFC9110]. The user agent MUST validate that the content-type of the returned resource is application/json.

The HTTP server that serves the JSON configuration MUST NOT require any form of HTTP authentication to return the configuration.

For example:

```
C: GET /.well-known/user-agent-configuration HTTP/2
C: Host: foo.example.com
C: user-agent: curl/8.4.0
C: accept: application/json
C:

S: HTTP/2 200
S: server: nginx/1.25.3
S: date: Mon, 23 Oct 2025 10:30:00 GMT
S: content-type: application/json
S: content-length: 1184
S:
S: {
S:     "protocols": {
S:         "jmap": {
S: ...
```

5.2.2. DNS Fallback

To construct the Well-Known URI described in Section 5.2.1, the user agent requires a hostname. If the URI generated according to that section does not allow the user agent to retrieve a valid configuration, the user agent MAY attempt to retrieve an alternative hostname using DNS as described in this section. The user agent then uses this alternative hostname to construct an alternative Well-Known URI.

The user agent MUST NOT use DNS to retrieve an alternative hostname if the URI described in Section 5.2.1 successfully provides a valid configuration.

When a user agent uses DNS for configuration discovery, it SHOULD validate all DNS responses using DNSSEC as described in [RFC9364] or a similar secure mechanism. See Section 7.3 for detailed security considerations.

5.2.2.1. DNS SRV

The user agent sends a DNS request for the SRV Resource Record Type with a QNAME (Query Name) using the template _ua-auto-config._https.{domain}.

For example, if the user entered the email address jdoe@foo.example.com, the user agent performs a SRV lookup of _ua-auto-config._https.foo.example.com..

The DNS response would then give the user agent the _name_ to use instead of the email address' domain to construct the Well-Known URI. For example:

```
;; QUESTION SECTION:
;_ua-auto-config._https.foo.example.com.        IN    SRV

;; ANSWER SECTION:
_ua-auto-config._https.foo.example.com. 7200 IN    SRV    10 60 443 qux.example.com.
```

The DNS response returns the name qux.example.com, and the user agent would then attempt to retrieve the configuration from the Well-Known URI

https://qux.example.com/.well-known/user-agent-configuration

The _target_ in the response is the domain name that will be used for the Well-Known URI. The _port_ in the response MUST be 443, and user agents MUST ignore any response where the _port_ is not 443.

[RFC2782] describes DNS SRV in detail.

5.2.2.2. DNS MX

If retrieving the configuration using both the well-known domain (as described in Section 5.2.1) and using DNS SRV (as described in Section 5.2.2.1) fail, the user agent MAY choose to retrieve an alternative hostname by retrieving an MX record as described in the following.

The user agent MUST NOT use DNS MX to retrieve an alternative hostname if the approaches described in Sections 5.2.1 or 5.2.2.1 allow the user agent to retrieve a valid configuration.

The user agent requests the MX records for the domain of the user's email address. It then picks the hosts with the lowest priority value.

For each of the lowest priority value hostnames, the user agent then performs a DNS SRV lookup as described in Section 5.2.2.1 to retrieve a hostname and in turn construct a Well-Known URI for the configuration.

For example, if the user entered jdoe@foo.example.com as their email address, the user agent would construct a DNS query MX lookup using foo.example.com as the QNAME (Query Name).

```
;; QUESTION SECTION:
;foo.example.com.        IN    MX

;; ANSWER SECTION:
foo.example.com.    3600    IN    MX    10 mail1.example.com.
foo.example.com.    3600    IN    MX    20 mail2.example.com.
```

The user agent would then perform a SRV lookup using _ua-auto-config._https.mail1.example.com as the QNAME:

```
;; QUESTION SECTION:
;_ua-auto-config._https.mail1.example.com.        IN    SRV

;; ANSWER SECTION:
_ua-auto-config._https.mail1.example.com. 7200 IN    SRV    10 60 443 mail.example.com
.
```

The user agent would then use the Well-Known URI

https://mail.example.com/.well-known/user-agent-configuration

to retrieve the configuration.

5.3. Configuration Validation

User agents MUST validate that the configuration contains valid JSON syntax according to RFC 8259. Additionally, user agents MUST validate the retrieved configuration against the JSON schema specification provided in Appendix A. If the JSON syntax is invalid, user agents MUST ignore the configuration entirely and not use any portion of it.

User agents MUST process only the properties that they support and MUST ignore properties not specified in the schema. This requirement enables future extensions of the format without breaking existing user agent implementations.

5.4. Selecting Protocols and Authentication

After retrieving the configuration JSON, the user agent needs to determine:

* Which protocols to use

* Which authentication mechanism to employ

Both user agents and servers support multiple protocols. The user agent needs to decide which protocols to use and determine whether the protocols specified in the configuration are sufficient to proceed. This document does not provide recommendations regarding protocol preferences or minimum protocol requirements, as these decisions depend on the specific needs and implementation details of individual user agents.

The automatic configuration flow covers two distinct authentication types:

* Username and password-based authentication

* OAuth-based authentication

User agents can support one or both authentication methods. Similarly, servers for each protocol can support one or both methods. Additionally, the specific implementation details of authentication can vary between protocols.

User agents SHOULD probe each protocol and server of interest to determine compatible authentication methods. This probing process could influence the user agent’s protocol selection. This probing process also allows the user agent to verify server availability.

This document does not provide recommendations regarding protocol preferences. The user agent MUST make these decisions based on its needs, and any such decision is very specific to the implementation at hand.

If the user agent determines that it can use either password based authentication or OAuth authentication, the user agent SHOULD prefer using OAuth based authentication.

5.4.1. OAuth

The JSON configuration includes an oauth-public entry when the provider supports OAuth. According to Section 2.2 of [OAuthPublic], the OAuth Authorization Server Metadata is served at a URL constructed using the issuer value with the template https://{issuer}/.well-known/oauth-authorization-server.

For example, if the JSON configuration contains this entry:

```
"oauth-public" : {
    "issuer" : "auth.example.com"
},
```

The OAuth Authorization Server Metadata would be at the URL https://auth.example.com/.well-known/oauth-authorization-server

Before the user agent chooses to use OAuth for any protocols, it SHOULD make sure that it can retrieve the OAuth Authorization Server Metadata from the corresponding URI.

The user agent SHOULD connect to each protocol that it might be interested in to detect which authentication mechanism it supports. The user agent MUST NOT perform any authentication at this point, but it SHOULD record which mechanism(s) the server for each protocol of interest supports.

The user agent SHOULD still probe each server as described in the next sections 5.4.2 and 5.4.3 to check that the servers it is interested in have OAuth support.

5.4.2. HTTP Authentication Mechanisms

To check which HTTP authentication schemes a particular HTTP server supports, the user agent SHOULD send a request to the server’ s endpoint without any ‘Authorization‘ header. For example

```
GET /jmap/session HTTP/1.1
host: api.example.com
accept: application/json
```

As detailed in [RFC9110] the server SHOULD respond to this with a 401 Unauthorized status code and a www-authenticate response header indicating which authentication scheme(s) are supported.

The user agent SHOULD include an accept header with the content type it would expect for the given protocol. For JMAP, for example, the user agent SHOULD include accept: application/json as a header in this request.

If the server supports OAuth, the www-authenticate response header would include Bearer. [RFC6749] describes this in more detail. [RFC6750] and [RFC7616] describe username + password authentication using the so-called 'Basic' HTTP authentication scheme and the HTTP digest access authentication respectively. Other password based authentication mechanisms exist for HTTP, but their discussion is outside the scope of this document.

For example, a response header

HTTP/1.1 401 Unauthorized
www-authenticate: Bearer realm="api.example.com"

would indicate that the server supports OAuth.

Alternatively, a response such as

HTTP/1.1 401 Unauthorized
www-authenticate: Digest realm="api.example.com", qop="auth", algorithm=SHA-256, nonce
="dcd98b7102dd2f0e8b11d0f600bfb0c093", opaque="5ccc069c403ebaf9f0171e9517f40e41"

would indicate that the server supports digest access authentication.

Details of HTTP authentication are described in [RFC9110] but it is worth noting that a server supporting multiple authentication methods can return either multiple www-authenticate header fields in its response or a single www-authenticate header field with multiple authentication methods.

5.4.3. Other Protocols Authentication Mechanisms

For text based protocols (IMAP, POP3, and SMTP) the client SHOULD connect to the server to check that

* the client can reach the server,

* the server implements the expected protocol, and

* which authentication mechanisms the server supports.

The client MUST attempt to connect to the server on the default ports

* port 993 for IMAP

* port 995 for POP3

* port 465 for SMTP

using TLS. See [RFC2595] and [RFC3207] for details about using TLS with these protocols.

The client SHOULD NOT attempt to connect to an IMAP, POP3, or SMTP server on their cleartext ports.

The client MUST NOT allow for configuration with a cleartext protocol that is not protected by TLS.

The client MUST validate that the connection is secured by TLS, and that the TLS certificate is valid and matches the expected domain name. See Section 7.4 for details on TLS validation.

If the server announces either the OAUTHBEARER SASL authentication method, the client can assume that this server supports using [OAuthPublic]. For IMAP for example, if the server's CAPABILITY contains AUTH=OAUTHBEARER this would indicate this support.

Servers SHOULD NOT announce support for the OAUTHBEARER SASL mechanism if they do not support [OAuthPublic]. Servers MUST announce support for the OAUTHBEARER SASL mechanism if they support OAuth Profile for Open Public Clients. Servers SHOULD NOT use the XOAUTH2 SASL mechanism.

There are various SASL authentication mechanisms for password based authentication, and additionally IMAP and POP3 also support password based authentication through LOGIN and APOP respectively. The client needs to probe each server to determine which authentication methods are supported by both the client and the server.

5.4.3.1. IMAP

The IMAP protocol provides various other authentication mechanisms. There are various SASL mechanisms for username + password authentication. [RFC4959] describes how to use SASL with IMAP.

Unless the server sends the LOGINDISABLED capability, clients know that they can use the LOGIN command for username + password authentication.

For IMAP, a sample session might look like this:

```
S: * OK [CAPABILITY IMAP4 IMAP4rev1 SASL-IR AUTH=PLAIN AUTH=OAUTHBEARER] IMAP Server
C: A1 LOGOUT
S: * BYE
S: A1 OK
```

In this case the server returned its IMAP capabilities as part of the so-called greeting messages. The support for the OAUTHBEARER SASL authentication mechanism indicates to the client that the server supports OAuth. Similarly the PLAIN SASL authentication mechanism indicates that the server supports username + password authentication. The absence of the LOGINDISABLED also indicates support for username and password authentication.

If the server doesn't include its capabilities in the server greeting, the client SHOULD send a CAPABILITY command as outlined in Section 6.1.1 of [RFC9051]. The client SHOULD send a LOGOUT command as outlined in Section 6.1.3 of [RFC9051] and check that the server sends an untagged BYE response.

5.4.3.2. SMTP

SMTP similarly announces which authentication mechanisms is supports. [RFC4954] describes how to use SASL with SMTP.

For SMTP, the client would use the EHLO command to retrieve the SMTP server's supported authentication mechanisms. For example:

```
S: 220 smtp.example.com ESMTP service ready
C: EHLO client.example.com
S: 250-smtp.example.com
S: 250-PIPELINING
S: 250-SIZE 10240000
S: 250-AUTH PLAIN LOGIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 8BITMIME
C: QUIT
S: 221 Bye
```

Section 3.2 of [RFC5321] describe this client initiation of the SMTP protocol. [RFC4954] describes SMTP authentication in detail. In the above example PLAIN and LOGIN indicate that the server supports username + password based authentication, and OAUTHBEARER indicates that the server supports OAuth.

The client SHOULD send a QUIT command and check for the server's 221 Bye response.

5.4.3.3. POP3

For POP3 [RFC5034] describes how to use SASL. POP3 additionally supports password based authentication using its USER and PASS commands, or through the optional APOP command. The client can check which SASL authentication methods the server supports using the CAPA command.

For example:

```
S: +OK pop.example.com POP3 server ready
C: CAPA
S: +OK List of capabilities follows
S: SASL PLAIN OAUTHBEARER
S: IMPLEMENTATION BlurdyBlurp POP3 server
S: .
C: QUIT
S: +OK POP3 server signing off (maildrop empty)
```

The client SHOULD send a QUIT command and check for the server's +OK response.

5.5. Confirming Server Names

The client SHOULD display to the user the hostnames of all servers that it intends to authenticate with. This list of server hostnames gives the user a chance to validate if they want to move ahead.

The client MAY choose to limit the hostnames to their second-level domain names when displaying this list. Instead of displaying mail.example.com the client would display example.com if it chooses to do so. Limiting the hostnames to their second-level domain names helps users identify if this is the domain they intend to connect to or not.

The client MUST NOT cut off parts of long second-level domains, to avoid spoofing. At least 63 characters of the second-level domain names MUST be displayed.

If the user mistyped their email address in Section 5.1, letting the user confirm the server hostnames gives the user a chance to notice this.

Clients can enhance the user experience by using the info section of the configuration (described in Section 4.4) to display the provider's name and logo during this configuration step.

Section 7.1 list security considerations related to the user confirming these hostnames.

5.6. Username

For password-based authentication, the full addr-spec part of the email address MUST be used as the username. For example:

Text Entered by User	username
jdoe@example.com	jdoe@example.com
"J Doe" <jdoe@example.com>	jdoe@example.com
<jdoe@example.com>	jdoe@example.com

Table 2: Username from email address

For OAuth Profile for Open Public Clients, the client MUST send this username as the login_hint in the authorization request URL.

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 will 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.

5.7. Configuring OAuth

If the client and the provider both support OAuth Profile for Open Public Clients, and if the servers that the client wants to use support OAuth, the client can configure OAuth.

The authentication section in the JSON configuration described in Section 4.3 will let the client create a URL to retrieve the OAuth Authorization Server Metadata as described in section Section 5.4.1. With this metadata, the client can then use [OAuthPublic] to configure OAuth.

Part of this will require opening the authorization request URL in an external user-agent, which is typically the default browser. The client SHOULD make it very apparent to the user which hostname is being used for the authorization request URL. The client SHOULD make the user confirm least the second-level domain name(s) of the hostname of this URL. This is to minimize the risk of the user exposing their credentials to a third party.

As described in Section 5.5, the client SHOULD also confirm the server hostnames with the user. For OAuth authentication, the client SHOULD include the hostname of the authorization request URL in this list of hostnames.

5.8. Configuring Password-Based Authentication

If the client and provider both support password-based authentication, and if the servers that the client wants to use support password-based authentication, the client can configure itself to use these servers.

The client SHOULD at this point ask the user for the password for their account with the provider.

Once the user has entered their password, the client SHOULD validate that it is able to authenticate using the entered password with all servers that it is interested in. Only then would it save the new configuration.

6. Configuration Validation

6.1. User Approval

Regardless of the mechanism used to obtain the configuration, clients SHOULD display the configuration details to users and request explicit confirmation before use. During this confirmation process:

* At least the second-level domain names of all involved hostnames MUST be displayed clearly and prominently.

* Clients MUST NOT truncate long second-level domain names to prevent spoofing attacks. At least 63 characters MUST be displayed.

6.2. OAuth2 requirements

If OAuth2 is used, the OAuth2 server MUST adhere to [OAuthPublic].

Notably, the Dynamic Client Registration MUST be implemented and return a working Client ID in response HTTP calls defined by the specification.

The OAuth2 scopes defined in [OAuthPublic] MUST be included and MUST give access to the servers returned in the JSON configuration described in Section 4.

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

7. Security Considerations

While the mechanism described in this document makes it easier for users to correctly configure their user agent, there's an associated risk with making it easier for users to expose their credentials to a third party.

User agents using the mechanism described in this document need to design their user interface and user experience such that this risk is minimized. Actual affordances depend on implementation details of the user interface and are outside the scope of this document.

7.1. Mistyped Domain Names

As part of the first step of the mechanism described in this document, the user enters their email address. If the user mistyped the domain part of their email address, and if the mistyped domain exists and supports the mechanism described in this document, the user could expose their credentials to the owner of that domain. An attacker could use this in an attempt to gain knowledge of the user's credentials.

As a result, user agents need to carefully design their user interface and user experience as to let the user know which domain is being used. It would make sense to display this very clearly to the user, before they enter any credentials.

The user agent would want to display a list of all second-level domains (SLDs) for all the servers that it intends to use. The user can then confirm these.

When the mechanism described in this document uses OAuth, it would make sense to ask the user to confirm the domains of the servers that will be used and additionally confirm the domains of

* the URL to retrieve the OAuth Authorization Server Metadata and/or

* the authorization request URL.

When asking the user to confirm these domains, it would make sense to only display the second-level domain (SLD) of those domains. This would make it more difficult for an attacker to do URL obfuscation and use subdomain phishing.

When using a browser for OAuth, the user agent would want to display the second-level domain (SLD) that the browser is currently displaying, and update this when redirects happen or new pages are loaded.

7.2. Attacker Controlled JSON Configuration

If an attacker can direct the user agent to use an attacker-controlled JSON configuration, the attacker would be able to direct the user to servers of the attacker’s choosing.

In Section 7.1 some mechanisms to limit this attack vector are described.

However, the user agent needs to combine this with a conservative trust policy for its TLS when retrieving the JSON configuration.

7.3. DNS

If the user agent uses the DNS mechanisms described in Section 5.2.2, care needs to be taken to make sure that an attacker hasn’t altered the DNS response. Using DNSSEC is one method of improving the security aspects of this approach. Alternatively, user agents MAY choose not use the DNS based mechanisms described in Section 5.2.2 at all.

7.4. TLS Validation

Whenever TLS is used, clients MUST validate TLS certificates and ensure that certificates are valid for the hostnames specified in the configuration. If certificate validation fails or the TLS certificate is otherwise invalid, the client MUST disconnect and MUST NOT use any configuration retrieved from that URI.

Certificate validation failures represent significant security risks, as they may indicate attempts to redirect users’ credentials to an attacker. Clients SHOULD NOT allow users to override TLS certificate validation checks.

Clients MAY implement more restrictive TLS policies for configuration retrieval than those typically required for web browsing, in order to provide stronger security guarantees. Clients MAY want to limit the allowed protocol version(s) to recent versions, and MAY similarly want to restrict the allowed cipher suites.

7.5. Updating the Configuration

The mechanism described in this document can be used to upgrade a user's configuration. The user agent could check for configurations even when it has a working configuration for an account. If the user agent finds a configuration that is better (in some way) than the already existing configuration, it could then upgrade the existing configuration.

But in doing so, the user agent would increase the attack window that a potential attacker has. Instead of only giving an attacker the opportunity when the user configures their user agent for the first time, the attacker would now have an opportunity each time the user agent checks for a better configuration. That could be undesirable.

7.6. Provider Information

Image parsers are a common attack vector, and clients MUST NOT display the logo images described in Section 4.4.1 unless they can do so in a way that doesn't expose the client to such attacks.

8. IANA Considerations

8.1. The User-Agent Auto-Configuration Protocol Registry

This document establishes the user-agent auto-configuration protocol registry.

User-agent auto-configuration protocols are registered on the advice of one or more Designated Experts (appointed by the IESG or their delegate), with a Specification Required (using terminology from [RFC8126]). However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests are sent to the ____@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for well-known URI: example").

Before a period of 14 days has passed, the Designated Expert(s) will either approve or deny the registration request, communicating this decision both to the review list and to IANA. Denials should include

an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@iesg.org mailing list) for resolution.

Registry Name:
User-Agent Auto-Configuration Protocols

Registration Procedure:
Specification Required (per [RFC8126])

8.1.1. Registration Template

Name
The commonly used name of the protocol

Protocol Key:
The JSON key using in the protocols object in the JSON configuration described in Section 4

URL Scheme:
(Optional) The URL scheme to be used for this protocol. HTTP based protocols specify https. If left empty, the protocols entry in the JSON configuration will only specify a hostname using host. If a URL scheme is specified, the JSON configuration entry will instead use url with a URL using this URL scheme.

Specification
Which RFC(s) or document(s) specify the protocol

Additional Properties:
Which additional property values are present in the protocol's JSON object.

8.1.2. Initial Registrations

Name	Protocol Key	URL scheme	Specification	Additional Properties
JMAP	jmap	https	RFC 8620, RFC 8621, RFC 8887, RFC 9610	
IMAP	imap		RFC 9051	
POP3	pop3		RFC 1939, RFC 5034	

SMTP	smtp		RFC 5321, RFC 2822	
CalDAV	caldav	https	RFC 4791	
CardDAV	carddav	https	RFC 6352	
WebDAV	webdav	https	RFC 4918	
ManageSieve	managesieve		RFC 5804, RFC 5228	

Table 3: Initial registrations to the User-Agent Auto-Configuration Protocol Registry

The _Additional Properties_ field is empty in all of the initial values.

8.2. Registration

8.2.1. New record to be registered

This registers the user-agent-configuration name from Section 5.2.1 according to [RFC8615].

URI suffix: user-agent-configuration
Change controller: IETF
Specification document(s): This document
Related information: /

8.2.2. Registration of SRV Service Name uaac

This registers the name ua-auto-config from Section 5.2.2 according to [RFC6335].

Service Name: ua-auto-config
Port Number: /
Transport Protocol: tcp
Description: User-Agent Automatic Configuration
Reference: This document

9. Normative References

[RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, DOI 10.17487/RFC1939, May 1996, <https://www.rfc-editor.org/info/rfc1939>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[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/info/rfc2119>.

[RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <https://www.rfc-editor.org/info/rfc2595>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, DOI 10.17487/RFC3207, February 2002, <https://www.rfc-editor.org/info/rfc3207>.

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <https://www.rfc-editor.org/info/rfc4791>.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <https://www.rfc-editor.org/info/rfc4918>.

[RFC4954] Siemborski, R., Ed. and A. Melnikov, Ed., "SMTP Service Extension for Authentication", RFC 4954, DOI 10.17487/RFC4954, July 2007, <https://www.rfc-editor.org/info/rfc4954>.

[RFC4959] Siemborski, R. and A. Gulbrandsen, "IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response", RFC 4959, DOI 10.17487/RFC4959, September 2007, <https://www.rfc-editor.org/info/rfc4959>.

[RFC5034] Siemborski, R. and A. Menon-Sen, "The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism", RFC 5034, DOI 10.17487/RFC5034, July 2007, <https://www.rfc-editor.org/info/rfc5034>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC5804] Melnikov, A., Ed. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, DOI 10.17487/RFC5804, July 2010, <https://www.rfc-editor.org/info/rfc5804>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, DOI 10.17487/RFC5891, August 2010, <https://www.rfc-editor.org/info/rfc5891>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6352] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <https://www.rfc-editor.org/info/rfc6352>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <https://www.rfc-editor.org/info/rfc7616>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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/info/rfc8174>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8620] Jenkins, N. and C. Newman, "The JSON Meta Application Protocol (JMAP)", RFC 8620, DOI 10.17487/RFC8620, July 2019, <https://www.rfc-editor.org/info/rfc8620>.

[RFC9051] Melnikov, A., Ed. and B. Leiba, Ed., "Internet Message Access Protocol (IMAP) - Version 4rev2", RFC 9051, DOI 10.17487/RFC9051, August 2021, <https://www.rfc-editor.org/info/rfc9051>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9364] Hoffman, P., "DNS Security Extensions (DNSSEC)", BCP 237, RFC 9364, DOI 10.17487/RFC9364, February 2023, <https://www.rfc-editor.org/info/rfc9364>.

[I-D.svg-tiny-ps-abrotman] Brotman, A. and J. T. Adams, "SVG Tiny Portable/Secure", Work in Progress, Internet-Draft, draft-svg-tiny-ps-abrotman-10, 1 May 2025, <https://datatracker.ietf.org/doc/html/draft-svg-tiny-ps-abrotman-10>.

[OAuthPublic] Jenkins, N. and B. Bucksch, "OAuth Profile for Open Public Clients", Work in Progress, Internet-Draft, draft-ietf-mailmaint-oauth-public, <https://datatracker.ietf.org/doc/html/draft-ietf-mailmaint-oauth-public>.

Appendix A. Configuration JSON Schema

The following JSON schema defines the format of the JSON configuration in Section 4

```
{
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "$id": "ua-auto-conf",
    "title": "User-Agent Automatic Configuration",
    "description": "Automatic Configuration of Email, Calendar, and Contact Server Set
tings",
    "type": "object",
    "properties": {
        "protocols": {
            "type": "object",
            "properties": {
                "caldav": { "$ref": "#/$defs/http-server" },
                "carddav": { "$ref": "#/$defs/http-server" },
                "imap": { "$ref": "#/$defs/text-server" },
                "jmap": { "$ref": "#/$defs/http-server" },
                "managesieve": { "$ref": "#/$defs/text-server" },
                "pop3": { "$ref": "#/$defs/text-server" },
                "smtp": { "$ref": "#/$defs/text-server" },
                "webdav": { "$ref": "#/$defs/http-server" }
            }
        },
        "authentication": {
            "properties": {
                "oauth-public": {
                    "properties": {
                        "issuer": {
                            "format": "hostname",
                            "type": "string"
                        }
                    },
                    "required": [
                        "issuer"
                    ],
                    "type": "object"
                },
                "password": {
                    "type": "boolean"
                }
            },
            "required": [ "password" ]
        },
        "info": {
            "type": "object",
            "properties": {
```

```
                    "provider": {
                        "type": "object",
                        "properties": {
                            "name": { "type": "string", "minLength": 1 },
                            "shortName": { "type": "string", "minLength": 1 },
                            "logo": {
                                "type": "array",
                                "items": {
                                    "type": "object",
                                    "properties": {
                                        "url": { "type": "string", "format": "uri" },
                                        "content-type": { "type": "string", "minLength": 1
},
                                        "width": { "type": "integer", "minimum": 1 },
                                        "height": { "type": "integer", "minimum": 1 }
                                    },
                                    "required": [ "url", "content-type" ]
                                }
                            }
                        },
                        "required": [ "name" ]
                    },
                    "help": {
                        "type": "object",
                        "properties": {
                            "documentation": { "type": "string", "format": "uri" },
                            "developer": { "type": "string", "format": "uri" },
                            "contact": {
                                "type": "array",
                                "items": { "type": "string", "minLength": 1 }
                            }
                        }
                    }
                },
                "required": [ "provider" ]
            }
        },
        "required": [ "protocols", "info" ],
        "$defs": {
            "http-server": {
                "type": "object",
                "properties": {
                    "url": { "type": "string", "format": "uri" }
                },
                "required": ["url"]
            },
            "text-server": {
                "type": "object",
                "properties": {
```

```
                    "host": { "type": "string", "format": "hostname" }
                },
                "required": ["host"]
            }
        }
    }
```

Acknowledgements

This document is based on the work of Ben Bucksch.

Authors' Addresses

Daniel Eggert
Apple Inc
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: deggert@apple.com
URI: https://www.apple.com

Matt Diephouse
Apple Inc
One Apple Park Way
Cupertino, CA 95014
United States of America
Email: diephouse@apple.com
URI: https://www.apple.com