HTTPAUTH
Internet-Draft
Intended status: Standards Track
Expires: May 28, 2016

A. Melnikov
Isode Ltd

November 25, 2015

Salted Challenge Response (SCRAM) HTTP Authentication Mechanism
draft-ietf-httpauth-scram-auth-11.txt

Abstract

The secure authentication mechanism most widely deployed and used by Internet application protocols is the transmission of clear-text passwords over a channel protected by Transport Layer Security (TLS). There are some significant security concerns with that mechanism, which could be addressed by the use of a challenge response authentication mechanism protected by TLS. Unfortunately, the HTTP Digest challenge response mechanism presently on the standards track failed widespread deployment, and have had success only in limited use.

This specification describes a family of HTTP authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM), which addresses security concerns with HTTP Digest and meets the deployability requirements. When used in combination with TLS or an equivalent security layer, a mechanism from this family could improve the status-quo for HTTP authentication.

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 http://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 May 28, 2016.

Table of Contents

1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Formal syntax is defined by [RFC5234] including the core rules defined in Appendix B of [RFC5234].

Example lines prefaced by "C:" are sent by the client and ones prefaced by "S:" by the server. If a single "C:" or "S:" label applies to multiple lines, then the line breaks between those lines

are for editorial clarity only, and are not part of the actual protocol exchange.

1.1. Terminology

This document uses several terms defined in [RFC4949] ("Internet Security Glossary") including the following: authentication, authentication exchange, authentication information, brute force, challenge-response, cryptographic hash function, dictionary attack, eavesdropping, hash result, keyed hash, man-in-the-middle, nonce, one-way encryption function, password, replay attack and salt. Readers not familiar with these terms should use that glossary as a reference.

Some clarifications and additional definitions follow:

- Authentication information: Information used to verify an identity claimed by a SCRAM client. The authentication information for a SCRAM identity consists of salt, iteration count, the "StoredKey" and "ServerKey" (as defined in the algorithm overview) for each supported cryptographic hash function.

- Authentication database: The database used to look up the authentication information associated with a particular identity. For application protocols, LDAPv3 (see [RFC4510]) is frequently used as the authentication database. For network-level protocols such as PPP or 802.11x, the use of RADIUS [RFC2865] is more common.

- Base64: An encoding mechanism defined in Section 4 of [RFC4648] which converts an octet string input to a textual output string which can be easily displayed to a human. The use of base64 in SCRAM is restricted to the canonical form with no whitespace.

- Octet: An 8-bit byte.

- Octet string: A sequence of 8-bit bytes.

- Salt: A random octet string that is combined with a password before applying a one-way encryption function. This value is used to protect passwords that are stored in an authentication database.

1.2. Notation

The pseudocode description of the algorithm uses the following notations:

- ":=": The variable on the left hand side represents the octet string resulting from the expression on the right hand side.

- "+": Octet string concatenation.

- "[ ]": A portion of an expression enclosed in "[" and "]" may not be included in the result under some circumstances. See the associated text for a description of those circumstances.

- Normalize(str): Apply the Preparation and Enforcement steps according to the OpaqueString profile (see [RFC7613]) to a UTF-8 [RFC3629] encoded "str". The resulting string is also in UTF-8. Note that implementations MUST either implement OpaqueString profile operations from [RFC7613], or disallow use of non US-ASCII Unicode codepoints in "str". The latter is a particular case of compliance with [RFC7613].

- HMAC(key, str): Apply the HMAC keyed hash algorithm (defined in [RFC2104]) using the octet string represented by "key" as the key and the octet string "str" as the input string. The size of the result is the hash result size for the hash function in use. For example, it is 32 octets for SHA-256 and 20 octets for SHA-1 (see [RFC6234]).

- H(str): Apply the cryptographic hash function to the octet string "str", producing an octet string as a result. The size of the result depends on the hash result size for the hash function in use.

- XOR: Apply the exclusive-or operation to combine the octet string on the left of this operator with the octet string on the right of this operator. The length of the output and each of the two inputs will be the same for this use.

- Hi(str, salt, i):

```
U1   := HMAC(str, salt + INT(1))
U2   := HMAC(str, U1)
...
Ui-1 := HMAC(str, Ui-2)
Ui   := HMAC(str, Ui-1)

Hi := U1 XOR U2 XOR ... XOR Ui
```

  where "i" is the iteration count, "+" is the string concatenation operator and INT(g) is a four-octet encoding of the integer g,

most significant octet first.

Hi() is, essentially, PBKDF2 [RFC2898] with HMAC() as the PRF and with dkLen == output length of HMAC() == output length of H().

2. Introduction

This specification describes a family of authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM) which addresses the requirements necessary to deploy a challenge-response mechanism more widely than past attempts (see [RFC5802]). When used in combination with Transport Layer Security (TLS, see [RFC5246]) or an equivalent security layer, a mechanism from this family could improve the status-quo for HTTP authentication.

HTTP SCRAM is adoptation of [RFC5802] for use in HTTP. (SCRAM data exchanged is identical to what is defined in [RFC5802].) It also adds 1 round trip reauthentication mode.

HTTP SCRAM provides the following protocol features:

- The authentication information stored in the authentication database is not sufficient by itself (without a dictionary attack) to impersonate the client. The information is salted to prevent a pre-stored dictionary attack if the database is stolen.

- The server does not gain the ability to impersonate the client to other servers (with an exception for server-authorized proxies).

- The mechanism permits the use of a server-authorized proxy without requiring that proxy to have super-user rights with the back-end server.

- Mutual authentication is supported, but only the client is named (i.e., the server has no name).

3. SCRAM Algorithm Overview

The following is a description of a full HTTP SCRAM authentication exchange. Note that this section omits some details, such as client and server nonces. See Section 5 for more details.

To begin with, the SCRAM client is in possession of a username and password (*) (or a ClientKey/ServerKey, or SaltedPassword). It sends the username to the server, which retrieves the corresponding authentication information, i.e. a salt, StoredKey, ServerKey and the iteration count i. (Note that a server implementation may choose to use the same iteration count for all accounts.) The server sends the

salt and the iteration count to the client, which then computes the following values and sends a ClientProof to the server:

(*) - Note that both the username and the password MUST be encoded in UTF-8 [RFC3629].

Informative Note: Implementors are encouraged to create test cases that use both username passwords with non-ASCII codepoints. In particular, it's useful to test codepoints whose "Unicode Normalization Form C" and "Unicode Normalization Form KC" are different. Some examples of such codepoints include Vulgar Fraction One Half (U+00BD) and Acute Accent (U+00B4).

```
SaltedPassword  := Hi(Normalize(password), salt, i)
ClientKey       := HMAC(SaltedPassword, "Client Key")
StoredKey       := H(ClientKey)
AuthMessage     := client-first-message-bare + "," +
                   server-first-message + "," +
                   client-final-message-without-proof
ClientSignature := HMAC(StoredKey, AuthMessage)
ClientProof     := ClientKey XOR ClientSignature
ServerKey       := HMAC(SaltedPassword, "Server Key")
ServerSignature := HMAC(ServerKey, AuthMessage)
```

The server authenticates the client by computing the ClientSignature, exclusive-ORing that with the ClientProof to recover the ClientKey and verifying the correctness of the ClientKey by applying the hash function and comparing the result to the StoredKey. If the ClientKey is correct, this proves that the client has access to the user's password.

Similarly, the client authenticates the server by computing the ServerSignature and comparing it to the value sent by the server. If the two are equal, it proves that the server had access to the user's ServerKey.

For initial authentication the AuthMessage is computed by concatenating decoded "data" attribute values from the authentication exchange. The format of these messages is defined in [RFC5802].

4. SCRAM Mechanism Names

A SCRAM mechanism name (authentication scheme) is a string "SCRAM-" followed by the uppercased name of the underlying hash function taken from the IANA "Hash Function Textual Names" registry (see http://www.iana.org) .

For interoperability, all HTTP clients and servers supporting SCRAM MUST implement the SCRAM-SHA-256 authentication mechanism, i.e. an authentication mechanism from the SCRAM family that uses the SHA-256 hash function as defined in [RFC7677].

5. SCRAM Authentication Exchange

HTTP SCRAM is a HTTP Authentication mechanism whose client response (<credentials-scram>) and server challenge (<challenge-scram>) messages are text-based messages containing one or more attribute-value pairs separated by commas. The messages and their attributes are described below and defined in Section 7.

```
challenge-scram   = scram-name [1*SP 1#auth-param]
      ; Complies with <challenge> ABNF from RFC 7235.
      ; Included in the WWW-Authenticate header field.

credentials-scram = scram-name [1*SP 1#auth-param]
      ; Complies with <credentials> from RFC 7235.
      ; Included in the Authorization header field.

scram-name = "SCRAM-SHA-256" / "SCRAM-SHA-1" / other-scram-name
      ; SCRAM-SHA-256 and SCRAM-SHA-1 are registered by this RFC.
      ;
      ; SCRAM-SHA-1 is registered for database compatibility
      ; with implementations of RFC 5802 (such as IMAP or XMPP
      ; servers), but it is not recommended for new deployments.

other-scram-name = "SCRAM-" hash-name
      ; hash-name is a capitalized form of names from IANA
      ; "Hash Function Textual Names" registry.
      ; Additional SCRAM names must be registered in both
      ; the IANA "SASL mechanisms" registry
      ; and the IANA "authentication scheme" registry.
```

This is a simple example of a SCRAM-SHA-256 authentication exchange (no support for channel bindings, as this feature is not currently supported by HTTP). Username 'user' and password 'pencil' are used. Note that long lines are folded for readability.

```
C: GET /resource HTTP/1.1
C: Host: server.example.com
C: [...]

S: HTTP/1.1 401 Unauthorized
S: WWW-Authenticate: Digest realm="realm1@example.com",
       Digest realm="realm2@example.com",
       Digest realm="realm3@example.com",
       SCRAM-SHA-256 realm="realm3@example.com",
       SCRAM-SHA-256 realm="testrealm@example.com"
S: [...]

C: GET /resource HTTP/1.1
C: Host: server.example.com
C: Authorization: SCRAM-SHA-256 realm="testrealm@example.com",
       data=biwsbj11c2VyLHI9ck9wck5HZndFYmVSV2diTkVrcU8K
C: [...]

S: HTTP/1.1 401 Unauthorized
S: WWW-Authenticate: SCRAM-SHA-256
        sid=AAAABBBBCCCCDDDD,
        data=cj1yT3ByTkdmd0ViZVJXZ2JORWtxTyVodllEcFdVYTJSYVRDQWZ1eEZJbGo
          paE5sRixzPVcyMlphSjBTTlk3c29Fc1VFamI2Z1E9PSxpPTQwOTYK
S: [...]

C: GET /resource HTTP/1.1
C: Host: server.example.com
C: Authorization: SCRAM-SHA-256 sid=AAAABBBBCCCCDDDD,
       data=Yz1iaXdzLHI9ck9wck5HZndFYmVSV2diTkVrcU8laHZZRHBXVWEyUmFUQ0FmdXhG
            SWxqKWhObEYscD1kSHpiWmFwV0lrNGpVaE4rVXRlOXl0YWc5empmTUhnc3FtbWl6
            N0FuZFZRPQo=
C: [...]

S: HTTP/1.1 200 Ok
S: Authentication-Info: sid=AAAABBBBCCCCDDDD,
       data=dj02cnJpVFJCaTIzV3BSUi93dHVwK21NaFVaVW4vZEI1bkxUSlJzamw5NUc0PQo=
S: [...Other header fields and resource body...]
```

In the above example the first client request contains data attribute which base64 decodes as follows: "n,,n=user,r=rOprNGfwEbeRWgbNEkqO" (with no quotes). Server then responds with data attribute which base64 decodes as follows: "r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFIlj)hNlF,s=W22ZaJ0SNY7soEsUEjb6gQ==,i=4096". The next client request contains data attribute which base64 decodes as follows: "c=biws,r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFIlj)hNlF,p=dHzbZapWIk4jUhN+Ute9ytag9zjfMHgsqmmiz7AndVQ=". The final server response contains a data attribute which base64 decodes as follows:

"v=6rriTRBi23WpRR/wtup+mMhUZUn/dB5nLTJRsjl95G4=".

Note that in the example above the client can also initiate SCRAM authentication without first being prompted by the server.

Initial "SCRAM-SHA-256" authentication starts with sending the "Authorization" request header field defined by HTTP/1.1, Part 7 [RFC7235] containing "SCRAM-SHA-256" authentication scheme and the following attributes:

- A "realm" attribute MAY be included to indicate the scope of protection in the manner described in HTTP/1.1, Part 7 [RFC7235]. As specified in [RFC7235], the "realm" attribute MUST NOT appear more than once. The realm attribute only appears in the first SCRAM message to the server and in the first SCRAM response from the server.

- The client also includes the data attribute that contains base64 encoded "client-first-message" [RFC5802] containing:

  * a header consisting of a flag indicating whether channel binding is supported-but-not-used, not supported, or used . Note that this version of SCRAM doesn't support HTTP channel bindings, so this header always starts with "n"; otherwise the message is invalid and authentication MUST fail.

  * SCRAM username and a random, unique nonce attributes.

In HTTP response, the server sends WWW-Authenticate header field containing: a unique session identifier (the "sid" attribute) plus the "data" attribute containing base64-encoded "server-first-message" [RFC5802]. The "server-first-message" contains the user's iteration count i, the user's salt, and the nonce with a concatenation of the client-specified one with a server nonce.

The client then responds with another HTTP request with the Authorization header field, which includes the "sid" attribute received in the previous server response, together with the "data" attribute containing base64-encoded "client-final-message" data. The latter has the same nonce and a ClientProof computed using the selected hash function (e.g. SHA-256) as explained earlier.

The server verifies the nonce and the proof, and, finally, it responds with a 200 HTTP response with the Authentication-Info header field [RFC7615] containing the "sid" attribute (as received from the client) and the "data" attribute containing base64-encoded "server-final-message", concluding the authentication exchange.

The client then authenticates the server by computing the ServerSignature and comparing it to the value sent by the server. If the two are different, the client MUST consider the authentication exchange to be unsuccessful and it might have to drop the connection.

5.1. One round trip reauthentication

If the server supports SCRAM reauthentication, the server sends in its initial HTTP response a WWW-Authenticate header field containing: the "realm" attribute (as defined earlier), the "sr" attribute that contains the server part of the "r" attribute (see s-nonce in [RFC5802]) and optional "ttl" attribute (which contains the "sr" value validity in seconds).

If the client has authenticated to the same realm before (i.e. it remembers "i" and "s" attributes for the user from earlier authentication exchanges with the server), it can respond to that with "client-final-message". When constructing the "client-final-message" the client constructs the c-nonce part of the "r" attribute as on initial authentication and the s-nonce part as follows: s-nonce is a concatenation of nonce-count and the "sr" attribute (in that order). The nonce-count is a positive integer that that is equal to the user's "i" attribute on first reauthentication and is incremented by 1 on each successful re-authentication.

> The purpose of the nonce-count is to allow the server to detect request replays by maintaining its own copy of this count - if the same nonce-count value is seen twice, then the request is a replay.

If the server considers the s-nonce part of the nonce attribute (the "r" attribute) to be still valid (i.e. the nonce-count part is as expected (see above) and the "sr" part is still fresh), it will provide access to the requested resource (assuming the client hash verifies correctly, of course). However if the server considers that the server part of the nonce is stale (for example if the "sr" value is used after the "ttl" seconds), the server returns "401 Unauthorized" containing the SCRAM mechanism name with the following attributes: a new "sr", "stale=true" and an optional "ttl". The "stale" attribute signals to the client that there is no need to ask user for the password.

> Formally, the "stale" attribute is defined as follows: A flag, indicating that the previous request from the client was rejected because the nonce value was stale. If stale is TRUE (case-insensitive), the client may wish to simply retry the request with a new encrypted response, without reprompting the user for a new username and password. The server should only set stale to TRUE

if it receives a request for which the nonce is invalid but with a valid digest for that nonce (indicating that the client knows the correct username/password). If stale is FALSE, or anything other than TRUE, or the stale directive is not present, the username and/or password are invalid, and new values must be obtained.

When constructing AuthMessage Section 3 to be used for calculating client and server proofs, "client-first-message-bare" and "server-first-message" are reconstructed from data known to the client and the server.

Reauthentication can look like this:

```
C: GET /resource HTTP/1.1
C: Host: server.example.com
C: [...]

S: HTTP/1.1 401 Unauthorized
S: WWW-Authenticate: Digest realm="realm1@example.com",
       Digest realm="realm2@example.com",
       Digest realm="realm3@example.com",
       SCRAM-SHA-256 realm="realm3@example.com",
       SCRAM-SHA-256 realm="testrealm@example.com",
sr=%hvYDpWUa2RaTCAfuxFIlj)hNlF
       SCRAM-SHA-256 realm="testrealm2@example.com", sr=AAABBBCCCDDD,
ttl=120
S: [...]

[Client authenticates as usual to realm "testrealm@example.com"]

[Some time later client decides to reauthenticate.
 It will use the cached "i" (4096) and "s" (W22ZaJ0SNY7soEsUEjb6gQ==)
 from earlier exchanges. It will use the nonce-value of 4096 together
 with the server advertised "sr" value as the server part of the "r".]

C: GET /resource HTTP/1.1
C: Host: server.example.com
C: Authorization: SCRAM-SHA-256 realm="testrealm@example.com",
       data=Yz1iaXdzLHI9ck9wck5HZndFYmVSV2diTkVrcU80MDk2JWh2WURwV1VhMlJhVENB
            ZnV4RklsailoTmxGLHA9ZEh6YlphcFdJazRqVWhOK1V0ZTl5dGFnOXpqZk1IZ3Nx
            bW1pejdBbmRWUT0K

C: [...]

S: HTTP/1.1 200 Ok
S: Authentication-Info: sid=AAAABBBBCCCCDDDD,
       data=dj02cnJpVFJCaTIzV3BSUi93dHVwK21NaFVaVW4vZEI1bkxUSlJzamw5NUc0PQo=
S: [...Other header fields and resource body...]
```

6. Use of Authentication-Info header field with SCRAM

When used with SCRAM, the Authentication-Info header field is allowed in the trailer of an HTTP message transferred via chunked transfer-coding.

7. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC5234]. The "UTF8-2", "UTF8-3" and "UTF8-4" non-terminals are defined in [RFC3629].

```
ALPHA = <as defined in RFC 5234 appendix B.1>
DIGIT = <as defined in RFC 5234 appendix B.1>

base64-char     = ALPHA / DIGIT / "/" / "+"

base64-4        = 4base64-char

base64-3        = 3base64-char "="

base64-2        = 2base64-char "=="

base64          = *base64-4 [base64-3 / base64-2]

sr              = "sr=" s-nonce
                  ;; s-nonce is defined in RFC 5802.

data            = "data=" base64
                  ;; The data attribute value is base64 encoded
                  ;; SCRAM challenge or response defined in
                  ;; RFC 5802.

ttl             = "ttl" = 1*DIGIT
                  ;; "sr" value validity in seconds.
                  ;; No leading 0s.

reauth-s-nonce  = nonce-count s-nonce

nonce-count     = posit-number
                  ;; posit-number is defined in RFC 5802.
                  ;; The initial value is taken from the "i"
                  ;; attribute for the user and is incremented
                  ;; by 1 on each successful re-authentication.

sid             = "sid=" token
                  ;; See token definition in RFC 7235.

stale           = "stale=" ( "true" / "false" )

realm           = "realm=" <as defined in RFC 7235>
```

8. Security Considerations

If the authentication exchange is performed without a strong security layer (such as TLS with data confidentiality), then a passive eavesdropper can gain sufficient information to mount an offline dictionary or brute-force attack which can be used to recover the user's password. The amount of time necessary for this attack

depends on the cryptographic hash function selected, the strength of the password and the iteration count supplied by the server. SCRAM allows the server/server administrator to increase the iteration count over time in order to slow down the above attacks. (Note that a server that is only in posession of "StoredKey" and "ServerKey" can't automatic increase the iteration count upon successful authentication. Such increase would require resetting user's password.) An external security layer with strong encryption will prevent these attack.

If the authentication information is stolen from the authentication database, then an offline dictionary or brute-force attack can be used to recover the user's password. The use of salt mitigates this attack somewhat by requiring a separate attack on each password. Authentication mechanisms which protect against this attack are available (e.g., the EKE class of mechanisms). RFC 2945 [RFC2945] is an example of such technology.

If an attacker obtains the authentication information from the authentication repository and either eavesdrops on one authentication exchange or impersonates a server, the attacker gains the ability to impersonate that user to all servers providing SCRAM access using the same hash function, password, iteration count and salt. For this reason, it is important to use randomly-generated salt values.

SCRAM does not negotiate a hash function to use. Hash function negotiation is left to the HTTP authentication mechanism negotiation. It is important that clients be able to sort a locally available list of mechanisms by preference so that the client may pick the most preferred of a server's advertised mechanism list. This preference order is not specified here as it is a local matter. The preference order should include objective and subjective notions of mechanism cryptographic strength (e.g., SCRAM with a successor to SHA-1 may be preferred over SCRAM with SHA-1).

SCRAM does not protect against downgrade attacks of channel binding types. The complexities of negotiation a channel binding type, and handling down-grade attacks in that negotiation, was intentionally left out of scope for this document.

A hostile server can perform a computational denial-of-service attack on clients by sending a big iteration count value.

See [RFC4086] for more information about generating randomness.

9. IANA Considerations

New mechanisms in the SCRAM- family are registered according to the IANA procedure specified in [RFC5802].

Note to future SCRAM- mechanism designers: each new SCRAM- HTTP authentication mechanism MUST be explicitly registered with IANA and MUST comply with SCRAM- mechanism naming convention defined in Section 4 of this document.

IANA is requested to add the following entry to the Authentication Scheme Registry defined in HTTP/1.1, Part 7 [RFC7235]:

Authentication Scheme Name: SCRAM-SHA-256
Pointer to specification text: [[this document]]
Notes (optional): (none)

Authentication Scheme Name: SCRAM-SHA-1
Pointer to specification text: [[this document]]
Notes (optional): (none)

10. Acknowledgements

This document benefited from discussions on the HTTPAuth, SASL and Kitten WG mailing lists. The authors would like to specially thank co-authors of [RFC5802] from which lots of text was copied.

Thank you to Martin Thomson for the idea of adding "ttl" attribute.

Thank you to Julian F. Reschke for corrections regarding use of Authentication-Info header field.

Special thank you to Tony Hansen for doing an early implementation and providing extensive comments on the draft.

11. Design Motivations

The following design goals shaped this document. Note that some of the goals have changed since the initial version of the document.

- The HTTP authentication mechanism has all modern features: support for internationalized usernames and passwords, support for channel bindings.

- The protocol supports mutual authentication.
- The authentication information stored in the authentication database is not sufficient by itself to impersonate the client.
- The server does not gain the ability to impersonate the client to other servers (with an exception for server-authorized proxies), unless such other servers allow SCRAM authentication and use the same salt and iteration count for the user.
- The mechanism is extensible, but [hopefully] not overengineered in this respect.
- Easier to implement than HTTP Digest in both clients and servers.

12. References

12.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, DOI 10.17487/RFC3454, December 2002, <http://www.rfc-editor.org/info/rfc3454>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, DOI 10.17487/RFC4013, February 2005, <http://www.rfc-editor.org/info/rfc4013>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <http://www.rfc-editor.org/info/rfc5802>.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <http://www.rfc-editor.org/info/rfc5929>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7613] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 7613, DOI 10.17487/RFC7613, August 2015, <http://www.rfc-editor.org/info/rfc7613>.

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <http://www.rfc-editor.org/info/rfc7615>.

[RFC7677] Hansen, T., "SCRAM-SHA-256 and SCRAM-SHA-256-PLUS Simple Authentication and Security Layer (SASL) Mechanisms", RFC 7677, DOI 10.17487/RFC7677, November 2015, <http://www.rfc-editor.org/info/rfc7677>.

12.2. Informative References

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <http://www.rfc-editor.org/info/rfc2865>.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, DOI 10.17487/RFC2898, September 2000, <http://www.rfc-editor.org/info/rfc2898>.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, DOI 10.17487/RFC2945, September 2000, <http://www.rfc-editor.org/info/rfc2945>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4510] Zeilenga, K., Ed., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, DOI 10.17487/RFC4510, June 2006, <http://www.rfc-editor.org/info/rfc4510>.

[RFC4616] Zeilenga, K., Ed., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, DOI 10.17487/RFC4616, August 2006, <http://www.rfc-editor.org/info/rfc4616>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <http://www.rfc-editor.org/info/rfc4949>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[tls-server-end-point]
Zhu, L., , "Registration of TLS server end-point channel bindings", IANA http://www.iana.org/assignments/channel-binding-types/tls-server-end-point, July 2008.

Author's Address

Alexey Melnikov
Isode Ltd

Email: Alexey.Melnikov@isode.com