NETWORK WORKING GROUP
Internet-Draft
Intended status: Standards Track
Expires: March 14, 2010

A. Menon-Sen
Oryx Mail Systems GmbH
A. Melnikov
Isode Ltd
C. Newman
N. Williams
Sun Microsystems

September 10, 2009

Salted Challenge Response (SCRAM) SASL and GSS-API Mechanism
draft-ietf-sasl-scram-07.txt

Status of this Memo

Copyright Notice

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 challenge response mechanisms presently on the standards track all fail to meet requirements necessary for widespread deployment, and have had success only in limited use.

This specification describes a family of Simple Authentication and Security Layer (SASL, RFC 4422) authentication mechanisms called the Salted Challenge Response Authentication Mechanism (SCRAM), which addresses the security concerns 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 application protocol authentication and provide a suitable choice for a mandatory-to-implement mechanism for future application protocol standards.

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 is more common.

- Base64: An encoding mechanism defined in [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.

- 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 20 octets for SHA-1 (see [RFC3174]).

- 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):

```
U0   := HMAC(str, salt + INT(1))
U1   := HMAC(str, U0)
U2   := HMAC(str, U1)
...
Ui-1 := HMAC(str, Ui-2)
Ui   := HMAC(str, Ui-1)

Hi := U0 XOR 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.

- o This 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. 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 application protocol authentication and provide a suitable choice for a mandatory-to-implement mechanism for future application protocol standards.

For simplicity, this family of mechanisms does not presently include negotiation of a security layer [RFC4422]. It is intended to be used with an external security layer such as that provided by TLS or SSH, with optional channel binding [RFC5056] to the external security layer.

SCRAM is specified herein as a pure Simple Authentication and Security Layer (SASL) [RFC4422] mechanism, but it conforms to the new bridge between SASL and the Generic Security Services Application Programming Interface (GSS-API) called "GS2" [I-D.ietf-sasl-gs2]. This means that this document defines both, a SASL mechanism and a GSS-API mechanism.

SCRAM provides the following protocol features:

o The authentication information stored in the authentication database is not sufficient by itself to impersonate the client. The information is salted to prevent a pre-stored dictionary attack if the database is stolen.

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

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

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

o When used as a SASL mechanism, SCRAM is capable of transporting authorization identities (see [RFC4422], Section 2) from the client to the server.

A separate document defines a standard LDAPv3 [RFC4510] attribute that enables storage of the SCRAM authentication information in LDAP.

See [I-D.melnikov-sasl-scram-ldap].

For an in-depth discussion of why other challenge response mechanisms are not considered sufficient, see appendix A. For more information about the motivations behind the design of this mechanism, see appendix B.

Comments regarding this draft may be sent either to the sasl@ietf.org mailing list or to the authors.

3. SCRAM Algorithm Overview

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. 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:

```
SaltedPassword  := Hi(password, salt)
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.

The AuthMessage is computed by concatenating messages from the authentication exchange. The format of these messages is defined in Section 7.

4. SCRAM Mechanism Names

A SCRAM mechanism name 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), optionally followed by the suffix "-PLUS" (see below). Note that SASL mechanism names are limited to 20 characters, which means that only hash function names with lengths shorter or equal to 9 characters (20-length("SCRAM-")-length("-PLUS") can be used. For cases when the underlying hash function name is longer than 9 characters, an alternative 9 character (or shorter) name can be used to construct the corresponding SCRAM mechanism name, as long as this alternative name doesn't conflict with any other hash function name from the IANA "Hash Function Textual Names" registry.

For interoperability, all SCRAM clients and servers MUST implement the SCRAM-SHA-1 authentication mechanism, i.e. an authentication mechanism from the SCRAM family that uses the SHA-1 hash function as defined in [RFC3174].

The "-PLUS" suffix is used only when the server supports channel binding to the external channel. If the server supports channel binding, it will advertise both the "bare" and "plus" versions of whatever mechanisms it supports (e.g., if the server supports only SCRAM with SHA-1 then it will advertise support for both SCRAM-SHA-1 and SCRAM-SHA-1-PLUS); if the server does not support channel binding, then it will advertise only the "bare" version of the mechanism (e.g., only SCRAM-SHA-1). The "-PLUS" exists to allow negotiation of the use of channel binding. See Section 6.

5. SCRAM Authentication Exchange

SCRAM is a SASL mechanism whose client response and server challenge messages are text-based messages containing one or more attribute-value pairs separated by commas. Each attribute has a one-letter name. The messages and their attributes are described in Section 5.1, and defined in Section 7.

SCRAM is a client-first SASL mechanism (See [RFC4422], Section 5, item 2a), and returns additional data together with a server's indication of a successful outcome.

This is a simple example of a SCRAM-SHA-1 authentication exchange when the client doesn't support channel bindings:

```
   C: n,,n=Chris Newman,r=ClientNonce
   S: r=ClientNonceServerNonce,s=PxR/wv+epq,i=128
   C: c=biwsCg==,r=ClientNonceServerNonce,p=WxPv/siO5l+qxN4
   S: v=WxPv/siO5l+qxN4
```

[[anchor5: Note that the all hashes above are fake and will be fixed during AUTH48.]]

With channel-binding data sent by the client this might look like this (see [tls-server-end-point] for the definition of tls-server-end-point TLS channel binding):

```
   C: p=tls-server-end-point,,n=Chris Newman,r=ClientNonce
   S: r=ClientNonceServerNonce,s=PxR/wv+epq,i=128
   C: c=cD10bHMtc2VydmVyLWVuZC1wb2ludCwsy1hFtXOnZ+ySrQM6srFp
      l/77uqvtxrg7nBY1BetEr/g=,r=ClientNonceServerNonce,p=Wx
      Pv/siO5l+qxN4
   S: v=WxPv/siO5l+qxN4
```

[[anchor6: Note that all hashes above are fake and will be fixed during AUTH48.]]

First, the client sends the "client-first-message" containing:

- o a GS2 header consisting of a flag indicating whether channel binding is supported-but-not-used, not supported, or used, and an optional SASL authorization identity;

- o SCRAM username and a random, unique nonce attributes.

Note that the client's first message will always start with "n", "y" or "p", otherwise the message is invalid and authentication MUST fail. This is important, as it allows for GS2 extensibility (e.g., to add support for security layers).

In response, the server sends a "server-first-message" containing the user's iteration count i, the user's salt, and appends its own nonce to the client-specified one.

The client then responds by sending "client-final-message" with the same nonce and a ClientProof computed using the selected hash function as explained earlier.

The server verifies the nonce and the proof, verifies that the authorization identity (if supplied by the client in the first message) is authorized to act as the authentication identity, and, finally, it responds with a "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. SCRAM Attributes

This section describes the permissible attributes, their use, and the format of their values. All attribute names are single US-ASCII letters and are case-sensitive.

Note that the order of attributes in client or server messages is fixed, with the exception of extension attributes (described by the "extensions" ABNF production), which can appear in any order in the designated positions. See the ABNF section for authoritative reference.

o a: This is an optional attribute, and is part of the GS2 [I-D.ietf-sasl-gs2] bridge between the GSS-API and SASL. This attribute specifies an authorization identity. A client may include it in its first message to the server if it wants to authenticate as one user, but subsequently act as a different user. This is typically used by an administrator to perform some management task on behalf of another user, or by a proxy in some situations.

   Upon the receipt of this value the server verifies its correctness according to the used SASL protocol profile. Failed verification results in failed authentication exchange.

If this attribute is omitted (as it normally would be), the authorization identity is assumed to be derived from the username specified with the (required) "n" attribute.

The server always authenticates the user specified by the "n" attribute. If the "a" attribute specifies a different user, the server associates that identity with the connection after successful authentication and authorization checks.

The syntax of this field is the same as that of the "n" field with respect to quoting of '=' and ','.

- n: This attribute specifies the name of the user whose password is used for authentication (a.k.a. "authentication identity" [RFC4422]). A client MUST include it in its first message to the server. If the "a" attribute is not specified (which would normally be the case), this username is also the identity which will be associated with the connection subsequent to authentication and authorization.

  Before sending the username to the server, the client SHOULD prepare the username using the "SASLPrep" profile [RFC4013] of the "stringprep" algorithm [RFC3454] treating it as a query string (i.e., unassigned Unicode code points are allowed). If the preparation of the username fails or results in an empty string, the client SHOULD abort the authentication exchange (*).

  (*) An interactive client can request a repeated entry of the username value.

  Upon receipt of the username by the server, the server SHOULD prepare it using the "SASLPrep" profile [RFC4013] of the "stringprep" algorithm [RFC3454] treating it as a query string (i.e., unassigned Unicode code points are allowed). If the preparation of the username fails or results in an empty string, the server SHOULD abort the authentication exchange. Whether or not the server prepares the username using "SASLPrep", it MUST use it as received in hash calculations.

  The characters ',' or '=' in usernames are sent as '=2C' and '=3D' respectively. If the server receives a username which contains '=' not followed by either '2C' or '3D', then the server MUST fail the authentication.

- m: This attribute is reserved for future extensibility. In this version of SCRAM, its presence in a client or a server message MUST cause authentication failure when the attribute is parsed by

the other end.

- r: This attribute specifies a sequence of random printable ASCII characters excluding ',' which forms the nonce used as input to the hash function. No quoting is applied to this string. As described earlier, the client supplies an initial value in its first message, and the server augments that value with its own nonce in its first response. It is important that this value be different for each authentication. The client MUST verify that the initial part of the nonce used in subsequent messages is the same as the nonce it initially specified. The server MUST verify that the nonce sent by the client in the second message is the same as the one sent by the server in its first message.

- c: This REQUIRED attribute specifies the base64-encoded GS2 header and channel-binding data. It is sent by the client in its second authentication message. The attribute data consist of:

  * the GS2 header from the client's first message (recall: a channel binding flag and an optional authzid). This header is going to include channel binding type prefix (see [RFC5056]), if and only if the client is using channel binding;

  * followed by the external channel's channel binding data, if and only if the client is using channel binding.

- s: This attribute specifies the base64-encoded salt used by the server for this user. It is sent by the server in its first message to the client.

- i: This attribute specifies an iteration count for the selected hash function and user, and MUST be sent by the server along with the user's salt.

  For SCRAM-SHA-1/SCRAM-SHA-1-PLUS SASL mechanism servers SHOULD announce a hash iteration-count of at least 4096. Note that a client implementation MAY cache SaltedPassword/ClientKey for later reauthentication to the same service, as it is likely that the server is going to advertise the same salt value upon reauthentication. This might be useful for mobile clients where CPU usage is a concern.

- p: This attribute specifies a base64-encoded ClientProof. The client computes this value as described in the overview and sends it to the server.

- v: This attribute specifies a base64-encoded ServerSignature. It is sent by the server in its final message, and is used by the

client to verify that the server has access to the user's authentication information. This value is computed as explained in the overview.

5.2. Compliance with SASL mechanism requirements

This section describes compliance with SASL mechanism requirements specified in Section 5 of [RFC4422].

1) "SCRAM-SHA-1" and "SCRAM-SHA-1-PLUS".

2a) SCRAM is a client-first mechanism.

2b) SCRAM sends additional data with success.

3) SCRAM is capable of transferring authorization identities from the client to the server.

4) SCRAM does not offer any security layers (SCRAM offers channel binding instead).

5) SCRAM has a hash protecting the authorization identity.

6. Channel Binding

SCRAM supports channel binding to external secure channels, such as TLS. Clients and servers may or may not support channel binding, therefore the use of channel binding is negotiable. SCRAM does not provide security layers, however, therefore it is imperative that SCRAM provide integrity protection for the negotiation of channel binding.

Use of channel binding is negotiated as follows:

- Servers SHOULD advertise both non-PLUS (SCRAM-<hash-function>) and the PLUS-variant (SCRAM-<hash-function>-PLUS) SASL mechanism names. If the server cannot support channel binding, it MAY advertise only the non-PLUS variant. If the server would never succeed authentication of the non-PLUS variant due to policy reasons, it MAY advertise only the PLUS-variant.

- If the client negotiates mechanisms then the client MUST select SCRAM-<hash-function>-PLUS if offered by the server and the client wants to select SCRAM with the given hash function. Otherwise (the client does not negotiate mechanisms), if the client has no prior knowledge about mechanisms supported by the server and wasn't explicitly configured to use a particular variant of the SCRAM mechanism, then it MUST select only SCRAM-<hash-function> (not suffixed with "-PLUS").

- If the client supports channel binding and the server appears to support it (i.e., the client sees SCRAM-<hash-function>-PLUS), or if the client wishes to use channel binding but the client does not negotiate mechanisms, then the client MUST set the GS2 channel binding flag to "p" in order to indicate the channel binding type it is using and it MUST include the channel binding data for the external channel in the computation of the "c=" attribute (see Section 5.1).

- If the client supports channel binding but the server does not appear to (i.e., the client did not see SCRAM-<hash-function>-PLUS) then the client MUST either fail authentication or it MUST choose the non-PLUS mechanism and set the GS2 channel binding flag to "y" and MUST NOT include channel binding data for the external channel in the computation of the "c=" attribute (see Section 5.1).

- If the client does not support channel binding then the client MUST set the GS2 channel binding flag to "n" and MUST NOT include channel binding data for the external channel in the computation of the "c=" attribute (see Section 5.1).

- o Upon receipt of the client first message the server checks the GS2 channel binding flag (gs2-cb-flag).

  * If the flag is set to "y" and the server supports channel binding the server MUST fail authentication. This is because if the client sets the GS2 channel binding flag set to "y" then the client must have believed that the server did not support channel binding -- if the server did in fact support channel binding then this is an indication that there has been a downgrade attack (e.g., an attacker changed the server's mechanism list to exclude the -PLUS suffixed SCRAM mechanism name(s)).

  * If the channel binding flag was "p" and the server does not support the indicated channel binding type then the server MUST fail authentication.

The server MUST always validate the client's "c=" field. The server does this by constructing the value of the "c=" attribute and then checking that it matches the client's c= attribute value.

For more discussions of channel bindings, and the syntax of the channel binding data for various security protocols, see [RFC5056].

6.1. Default Channel Binding

A default channel binding type agreement process for all SASL application protocols that do not provide their own channel binding type agreement is provided as follows.

'tls-unique' is the default channel binding type for any application that doesn't specify one.

Servers MUST implement the "tls-unique" [tls-unique] [I-D.altman-tls-channel-bindings] channel binding type, if they implement any channel binding. Clients SHOULD implement the "tls-unique" [tls-unique] [I-D.altman-tls-channel-bindings] channel binding type, if they implement any channel binding. Clients and servers SHOULD choose the highest- layer/innermost end-to-end TLS channel as the channel to bind to.

Servers MUST choose the channel binding type indicated by the client, or fail authentication if they don't support it.

7. Formal Syntax

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

```
ALPHA = <as defined in RFC 5234 appendix B.1>
DIGIT = <as defined in RFC 5234 appendix B.1>
UTF8-2 = <as defined in RFC 3629 (STD 63)>
UTF8-3 = <as defined in RFC 3629 (STD 63)>
UTF8-4 = <as defined in RFC 3629 (STD 63)>

attr-val        = ALPHA "=" value
                  ;; Generic syntax of any attribute sent
                  ;; by server or client

value           = 1*value-char

value-safe-char = %x01-2B / %x2D-3C / %x3E-7F /
                  UTF8-2 / UTF8-3 / UTF8-4
                  ;; UTF8-char except NUL, "=", and ",".

value-char      = value-safe-char / "="

printable       = %x21-2B / %x2D-7E
                  ;; Printable ASCII except ",".
                  ;; Note that any "printable" is also
                  ;; a valid "value".

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

base64-4        = 4base64-char

base64-3        = 3base64-char "="

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

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

posit-number = %x31-39 *DIGIT
                  ;; A positive number

saslname        = 1*(value-safe-char / "=2C" / "=3D")
                  ;; Conforms to <value>

authzid         = "a=" saslname
                  ;; Protocol specific.
```

```
   cb-name         = 1*(ALPHA / DIGIT / "." / "-")
                      ;; See RFC 5056 section 7.
                      ;; E.g. "tls-server-end-point" or
                      ;; "tls-unique"

   gs2-cbind-flag  = "p=" cb-name / "n" / "y"
                     ;; "n" -> client doesn't support channel binding
                     ;; "y" -> client does support channel binding
                     ;;        but thinks the server does not.
                     ;; "p" -> client requires channel binding.
                     ;; The selected channel binding follows "p=".

   gs2-header      = gs2-cbind-flag "," [ authzid ] ","
                     ;; GS2 header for SCRAM
                     ;; (the actual GS2 header includes an optional
                     ;; flag to indicate that the GSS mechanism is not
                     ;; "standard" but since SCRAM is "standard" we
                     ;; don't include that flag).

   username        = "n=" saslname
                     ;; Usernames are prepared using SASLPrep.

   reserved-mext  = "m=" 1*(value-char)
                     ;; Reserved for signalling mandatory extensions.
                     ;; The exact syntax will be defined in
                     ;; the future.

   channel-binding = "c=" base64
                     ;; base64 encoding of cbind-input

   proof           = "p=" base64

   nonce           = "r=" c-nonce [s-nonce]
                     ;; Second part provided by server.

   c-nonce         = printable

   s-nonce         = printable

   salt            = "s=" base64

   verifier        = "v=" base64
                     ;; base-64 encoded ServerSignature.

   iteration-count = "i=" posit-number
                     ;; A positive number

   client-first-message-bare =
```

```
                       [reserved-mext ","]
                       username "," nonce ["," extensions]

     client-first-message =
                       gs2-header client-first-message-bare

     server-first-message =
                       [reserved-mext ","] nonce "," salt ","
                       iteration-count ["," extensions]

     client-final-message-without-proof =
                       channel-binding "," nonce [","
                       extensions]

     client-final-message =
                       client-final-message-without-proof "," proof

     gss-server-error = "e=" value
     server-final-message = gss-server-error /
                       verifier ["," extensions]
                       ;; The error message is only for the GSS-API
                       ;; form of SCRAM, and it is OPTIONAL to
                       ;; implement it.

     extensions = attr-val *("," attr-val)
                       ;; All extensions are optional,
                       ;; i.e. unrecognized attributes
                       ;; not defined in this document
                       ;; MUST be ignored.

     cbind-data    = 1*OCTET

     cbind-input   = gs2-header [ cbind-data ]
                       ;; cbind-data MUST be present for
                       ;; gs2-cbind-flag of "p" and MUST be absent
                       ;; for "y" or "n".
```

8. SCRAM as a GSS-API Mechanism

This section and its sub-sections and all normative references of it not referenced elsewhere in this document are INFORMATIONAL for SASL implementors, but they are NORMATIVE for GSS-API implementors.

SCRAM is actually also GSS-API mechanism. The messages are the same, but a) the GS2 header on the client's first message and channel binding data is excluded when SCRAM is used as a GSS-API mechanism, and b) the RFC2743 section 3.1 initial context token header is prefixed to the client's first authentication message (context token).

The GSS-API mechanism OID for SCRAM is <TBD> (see Section 10).

8.1. GSS-API Principal Name Types for SCRAM

SCRAM does not name acceptors. Therefore only GSS_C_NO_NAME and names of type GSS_C_NT_ANONYMOUS shall be allowed as the target name input of GSS_Init_sec_context() when using a SCRAM mechanism.

SCRAM supports only a single name type for initiators: GSS_C_NT_USER_NAME. GSS_C_NT_USER_NAME is the default name type for SCRAM.

There is no name canonicalization procedure for SCRAM beyond applying SASLprep as described in Section 5.1.

The query, display and exported name syntax for SCRAM principal names is the same: there is no syntax -- SCRAM principal names are free-form. (The exported name token does, of course, conform to [RFC2743] section 3.2, but the "NAME" part of the token is just a SCRAM user name.)

8.2. GSS-API Per-Message Tokens for SCRAM

The per-message tokens for SCRAM as a GSS-API mechanism SHALL be the same as those for the Kerberos V GSS-API mechanism [RFC4121] (see Section 4.2 and sub-sections), using the Kerberos V "aes128-cts-hmac-sha1-96" enctype [RFC3962].

The 128-bit session "protocol key" SHALL be derived by using the least significant (right-most) 128 bits of HMAC(StoredKey, "GSS-API session key" || ClientKey || AuthMessage). "Specific keys" are then derived as usual as described in Section 2 of [RFC4121], [RFC3961] and [RFC3962].

The terms "protocol key" and "specific key" are Kerberos V5 terms

[RFC3961].

SCRAM does support PROT_READY, and is PROT_READY on the initiator side first upon receipt of the server's reply to the initial security context token.

8.3. GSS_Pseudo_random() for SCRAM

The GSS_Pseudo_random() [RFC4401] for SCRAM SHALL be the same as for the Kerberos V GSS-API mechanism [RFC4402]. There is no acceptor-asserted sub-session key for SCRAM, thus GSS_C_PRF_KEY_FULL and GSS_C_PRF_KEY_PARTIAL are equivalent for SCRAM's GSS_Pseudo_random(). The protocol key to be used for the GSS_Pseudo_random() SHALL be the same as the key defined in Section 8.2.

9. Security Considerations

If the authentication exchange is performed without a strong security layer, 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. An external security layer with strong encryption will prevent this attack.

If the external security layer used to protect the SCRAM exchange uses an anonymous key exchange, then the SCRAM channel binding mechanism can be used to detect a man-in-the-middle attack on the security layer and cause the authentication to fail as a result. However, the man-in-the-middle attacker will have gained sufficient information to mount an offline dictionary or brute-force attack. For this reason, SCRAM includes the ability to increase the iteration count over time.

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. The WG selected not to use EKE like mechanisms as basis for SCRAM.

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 SASL 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).

Note that to protect the SASL mechanism negotiation applications normally must list the server mechs twice: once before and once after

authentication, the latter using security layers. Since SCRAM does not provide security layers the only ways to protect the mechanism negotiation are: a) use channel binding to an external channel, or b) use an external channel that authenticates a user-provided server name.

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.

10. IANA Considerations

IANA is requested to add the following family of SASL mechanisms to the SASL Mechanism registry established by [RFC4422]:

```
To: iana@iana.org
Subject: Registration of a new SASL family SCRAM

SASL mechanism name (or prefix for the family): SCRAM-*
Security considerations: Section 7 of [RFCXXXX]
Published specification (optional, recommended): [RFCXXXX]
Person & email address to contact for further information:
 IETF SASL WG <sasl@ietf.org>
Intended usage: COMMON
Owner/Change controller: IESG <iesg@ietf.org>
Note: Members of this family must be explicitly registered
using the "IETF Review" [RFC5226] registration procedure.
Reviews must be requested on the SASL WG mailing list.
```

"IETF Review" [RFC5226] registration procedure MUST be used for registering new mechanisms in this family. The SASL mailing list <sasl@ietf.org> (or a successor designated by the responsible Security AD) MUST be used for soliciting reviews on such registrations.

Note to future SCRAM- mechanism designers: each new SCRAM- SASL 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 entries to the SASL Mechanism registry established by [RFC4422]:

```
To: iana@iana.org
Subject: Registration of a new SASL mechanism SCRAM-SHA-1

SASL mechanism name (or prefix for the family): SCRAM-SHA-1
Security considerations: Section 7 of [RFCXXXX]
Published specification (optional, recommended): [RFCXXXX]
Person & email address to contact for further information:
 IETF SASL WG <sasl@ietf.org>
Intended usage: COMMON
Owner/Change controller: IESG <iesg@ietf.org>
Note:
```

```
To: iana@iana.org
Subject: Registration of a new SASL mechanism SCRAM-SHA-1-PLUS

SASL mechanism name (or prefix for the family): SCRAM-SHA-1-PLUS
Security considerations: Section 7 of [RFCXXXX]
Published specification (optional, recommended): [RFCXXXX]
Person & email address to contact for further information:
 IETF SASL WG <sasl@ietf.org>
Intended usage: COMMON
Owner/Change controller: IESG <iesg@ietf.org>
Note:
```

This document also requests IANA to assign a GSS-API mechanism OID for SCRAM from the iso.org.dod.internet.security.mechanisms prefix (see "SMI Security for Mechanism Codes" registry).

11. Acknowledgements

This document benefited from discussions on the SASL WG mailing list. The authors would like to specially thank Dave Cridland, Simon Josefsson, Jeffrey Hutzelman, Kurt Zeilenga, Pasi Eronen and Peter Saint-Andrefor their contributions to this document.

Appendix A. Other Authentication Mechanisms

The DIGEST-MD5 [I-D.ietf-sasl-digest-to-historic] mechanism has proved to be too complex to implement and test, and thus has poor interoperability. The security layer is often not implemented, and almost never used; everyone uses TLS instead. For a more complete list of problems with DIGEST-MD5 which lead to the creation of SCRAM see [I-D.ietf-sasl-digest-to-historic].

The CRAM-MD5 SASL mechanism, while widely deployed has also some problems, in particular it is missing some modern SASL features such as support for internationalized usernames and passwords, support for passing of authorization identity, support for channel bindings. It also doesn't support server authentication. For a more complete list of problems with CRAM-MD5 see [I-D.ietf-sasl-crammd5-to-historic].

The PLAIN [RFC4616] SASL mechanism allows a malicious server or eavesdropper to impersonate the authenticating user to any other server for which the user has the same password. It also sends the password in the clear over the network, unless TLS is used. Server authentication is not supported.

Appendix B. Design Motivations

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

- o The SASL mechanism has all modern SASL features: support for internationalized usernames and passwords, support for passing of authorization identity, support for channel bindings.

- o The protocol supports mutual authentication.

- o The authentication information stored in the authentication database is not sufficient by itself to impersonate the client.

- o 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.

- o The mechanism is extensible, but [hopefully] not overengineered in this respect.

- o Easier to implement than DIGEST-MD5 in both clients and servers.

Appendix C. Internet-Draft Change History

(RFC Editor: Please delete everything after this point)

Changes since -10

- Converted the source for this I-D to XML.
- Added text to make SCRAM compliant with the new GS2 design.
- Added text on channel binding negotiation.
- Added text on channel binding, including a reference to RFC5056.
- Added text on SCRAM as a GSS-API mechanism. This noted as not relevant to SASL-only implementors -- the normative references for SCRAM as a GSS-API mechanism are segregated as well.

Changes since -07

- Updated References.
- Clarified purpose of the m= attribute.
- Fixed a problem with authentication/authorization identity's ABNF not allowing for some characters.
- Updated ABNF for nonce to show client-generated and server-generated parts.
- Only register SCRAM-SHA-1 with IANA and require explicit registrations of all other SCRAM- mechanisms.

Changes since -06

- Removed hash negotiation from SCRAM and turned it into a family of SASL mechanisms.
- Start using "Hash Function Textual Names" IANA registry for SCRAM mechanism naming.
- Fixed definition of Hi(str, salt) to be consistent with [RFC2898].
- Clarified extensibility of SCRAM: added m= attribute (for future mandatory extensions) and specified that all unrecognized attributes must be ignored.

Changes since -05

- Changed the mandatory to implement hash algorithm to SHA-1 (as per WG consensus).

- Added text about use of SASLPrep for username canonicalization/ validation.

- Clarified that authorization identity is canonicalized/verified according to SASL protocol profile.

- Clarified that iteration count is per-user.

- Clarified how clients select the authentication function.

- Added IANA registration for the new mechanism.

- Added missing normative references (UTF-8, SASLPrep).

- Various editorial changes based on comments from Hallvard B Furuseth, Nico William and Simon Josefsson.

Changes since -04

- Update Base64 and Security Glossary references.

- Add Formal Syntax section.

- Don't bother with "v=".

- Make MD5 mandatory to implement. Suggest i=128.

Changes since -03

- Seven years have passed, in which it became clear that DIGEST-MD5 suffered from unacceptably bad interoperability, so SCRAM-MD5 is now back from the dead.

- Be hash agnostic, so MD5 can be replaced more easily.

- General simplification.

12. References

12.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

12.2. Normative References for GSS-API implementors

[I-D.ietf-sasl-gs2]
Josefsson, S. and N. Williams, "Using GSS-API Mechanisms in SASL: The GS2 Mechanism Family", draft-ietf-sasl-gs2-12 (work in progress), April 2009.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.

[RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.

[RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 4402, February 2006.

[tls-unique]
Zhu, L., "Registration of TLS unique channel binding (generic)", IANA http://www.iana.org/assignments/channel-binding-types/tls-unique, July 2008.

12.3. Informative References

[I-D.altman-tls-channel-bindings]
Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", draft-altman-tls-channel-bindings-06 (work in progress), August 2009.

[I-D.ietf-sasl-crammd5-to-historic]
Zeilenga, K., "CRAM-MD5 to Historic", draft-ietf-sasl-crammd5-to-historic-00 (work in progress), November 2008.

[I-D.ietf-sasl-digest-to-historic]
Melnikov, A., "Moving DIGEST-MD5 to Historic", draft-ietf-sasl-digest-to-historic-00 (work in progress), July 2008.

[I-D.melnikov-sasl-scram-ldap]
Melnikov, A., "LDAP schema for storing SCRAM secrets", draft-melnikov-sasl-scram-ldap-02 (work in progress), July 2009.

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4616] Zeilenga, K., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, August 2006.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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.

Authors' Addresses

Abhijit Menon-Sen
Oryx Mail Systems GmbH

Email: ams@oryx.com

Alexey Melnikov
Isode Ltd

Email: Alexey.Melnikov@isode.com

Chris Newman
Sun Microsystems
1050 Lakes Drive
West Covina, CA 91790
USA

Email: chris.newman@sun.com

Nicolas Williams
Sun Microsystems
5300 Riata Trace Ct
Austin, TX 78727
USA

Email: Nicolas.Williams@sun.com