Status of this Memo

This document is intended to be, after appropriate review and revision, submitted to the RFC Editor as a Experimental document. Distribution of this memo is unlimited. Technical discussion of this document will take place on the IETF SASL WG mailing list <ietf-sasl@imc.org>. Please send editorial comments directly to the author <Kurt.Zeilenga@Isode.COM>.

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

Abstract

This document describes a password authentication mechanism, called Zeilenga
YAP-SHA-256, for use in protocols which support Simple Authentication and Security Layer (SASL) framework. The mechanism relies on security services provided by a lower layer, such as Transport Layer Security (TLS), to protect the authentication exchange, and subsequent application data exchange, from common attacks. The YAP-SHA-256 mechanism may be viewed as an alternative to other password-based SASL mechanism, such as PLAIN, CRAM-MD5, and DIGEST-MD5.

1. Introduction

There exist multiple password-based mechanisms for use in the Simple Authentication and Security Layer (SASL) [RFC4422] framework. These include the PLAIN [RFC4616], CRAM-MD5 [RFC2195], and DIGEST-MD5 [RFC2831]. None of these mechanisms, themselves, provide integrity and confidential protection over the entirety of the authentication exchange. Only DIGEST-MD5 offers a security layer and, even so, the specification and its implementations suffer from multiple problems. And while these mechanisms may be used in conjunction with lower-level security services, these mechanism do not offer any facility to bind the channels [RFC5056].

This situation has lead to multiple efforts to design "better" SASL password-based mechanism. This document not only specifies yet another password mechanism, YAP-SHA-256, but defines a family of related password mechanisms, YAP-*.

YAP-* is a family of simple password SASL mechanisms based upon the Keyed-Hash Message Authentication Code (HMAC) [RFC2104] algorithm. The YAP-SHA-256 uses the SHA-256 [FIPS180-2] cryptographic hash function in conjunction with the HMAC algorithm.

YAP is specified as a family of SASL mechanisms to provide hash agility.

YAP mechanisms rely on services provided at a lower level, such as Transport Layer Security (TLS) [RFC5246], to secure the authentication exchange and subsequent application data exchange and, hence, YAP-SHA-256 does not offer a SASL security layer. YAP mechanisms require the lower-level security layer to be bound [RFC5056] in the authentication.

1.1 Experimental

This specification is part of a research and development effort exploring alternatives to current password-based authentication mechanisms. The authors make no claim that the alternative specified
here is suitable for any particular use (or general use).

1.2 Terminology

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 RFC 2119 [RFC2119].

2. The YAP-* Family of Mechanisms

Each mechanism in this family differs only by the choice of hash algorithm. Each mechanism has a name of the form YAP-HASH where HASH is a string chosen to reflect the hash algorithm used. To define a new mechanism within this family, the mechanism specification need only identify that it is a YAP mechanism, identify the hash algorithm used, and identify the name of the mechanism. The name of the mechanism is to be registered as discussed in [RFC4422].

The mechanism involves a single message from the client to the server.

```
message = authzid separator [ authcid ] separator data
separator = %x00
```

where:
- <authzid>, when present, is the authorization identity in the form specified by the application protocol specification, represented in UTF-8 [RFC3629], and
- <authcid> is authentication identity, a simple user name [RFC4013], prepared using the SASLprep [RFC4013] and represented in UTF-8 [RFC3629],
- <data> is a Keyed-Hash Message Authentication Code (HMAC) [RFC2104] produced as described below.

Implementors should note that the data portion of the message may contain a zero-valued octet and hence should parse the message front-to-back.

The HMAC is produced using the mechanism-specific hash algorithm, such as SHA-256 [FIPS180-2], as the cryptographic hash function, H. The secret key, K, is the unique channel binding [RFC5056] for the lower-level security protocol, padded with zero octets to the block size of the hash function. Where the unique channel binding is longer than the block size of the hash function, K is hash of the unique channel binding. The text is the concatenation of the authcid, the
authzid, and the hash of the user’s password, a simple password [RFC4013], prepared using SASLprep [RFC4013] and represented in UTF-8 [RFC3629]. That is, the <data> is computed as illustrated by the following pseudo code.

```
HMAC(
    Pad( Length(ChannelBinding)>HashBlockSize
         ? H(ChannelBinding) : ChannelBinding, 0, HashBlockSize),
    Concat( authzid, authcid, H(UTF8( SASLprep( password )))))
```

Note, in this pseudo code, the first argument of the HMAC function is the secret key and the second is the text. The cryptographic hash function used in the HMAC is implicitly H. The Pad function pads the first argument to the length specified in the third argument with the octet value provided in the second argument. The variable HashBlockSize is the block size of hash function, H. The Length function returns the length of its argument. The Concat function returns an octet which is the concatenation of its arguments. The UTF8 function returns the UTF-8 encoding of its argument. The SASLprep function prepares it argument according to the SASLprep algorithm. The H function returns the hash of its argument.

The hash of the user’s password is a password equivalent. Servers may choose to store this hash instead of the user’s password. In either case, the stored value must be adequately protected.

Implementations SHOULD NOT advertise availability of any mechanism in this family unless a lower-level security service providing both data integrity and data confidentiality protection is in place. Client implementations SHOULD NOT utilize any mechanism in this family without first verifying the identity of the server within the lower-level security service. Client implementors should consult the application protocol specification, in conjunction with the specification of the lower-level security service, for details on how to implement this verification.

Implementations MUST support Transport Security Layer (TLS) [RFC5246] channel bindings, as describe in [CBIND-TLS].

3. The YAP-SHA-256 Mechanism

The YAP-SHA-256 mechanism, named "YAP-SHA-256", is a YAP SASL mechanism. The YAP-SHA-256 mechanism, as the name suggests, uses the SHA-256 [FIPS180-2][RFC4634] hash algorithm.

4. YAP-SHA-256 Examples
This section will provide examples of an YAP-SHA-256 authentication exchanges...

5. Security Considerations

Security is discussed throughout this document.

This family of mechanisms was specifically designed to rely on security services offered at lower-levels to secure the authentication exchange and subsequent data exchanges. To ensure lower-level security services are provided end-to-end, the mechanisms utilize channel bindings [RFC5056].

To avoid disclosing the identity information to a rogue server, the client verifies the server’s identity using the lower-layer security service before utilizing any mechanism in this family.

Hash agility is provided in the family of mechanisms through the specification of additional mechanisms.

To avoid requiring server implementations maintain access to the user’s password, a password equivalent is used. The password equivalent is a simple hash of the password.

While it is likely that those choosing to store the password equivalent instead of the password would prefer the equivalent be designed to hinder dictionary attack with precomputed direction entries, a simple hash was chosen to avoid adding a server challenge. Use of the authcid as a salt was considered but rejected as it would tie the password equivalent to a particular authcid. It is desirable for the password equivalent to be usable with multiple authcid values (kurt and KURT) representing the same entity. It was also realized that it likely that implementors would (continue to) choose to store the password instead of a mechanism-specific password equivalent. Storing the password avoids significant implementation complexity and facilitates mechanism agility.

6. IANA Considerations

It is requested that IANA process the following request(s) upon approval of this document for publication as an RFC.

Subject: Registration of SASL YAP family of mechanisms
SASL family name (or prefix for the family): YAP-*
Security considerations: see RFC XXXX
Published specification (recommended): RFC XXXX
7. Acknowledgments

TBD.

8. Author’s Address

Kurt D. Zeilenga
Isode Limited

Email: Kurt.Zeilenga@Isode.COM

9. References

[[Note to the RFC Editor: please replace the citation tags used in referencing Internet-Drafts with tags of the form RFCnnnn where possible.]]

9.1. Normative References


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


9.2. Informative References


Full Copyright

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of
publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.