

Internet Engineering Task Force
Internet-Draft
Updates: [5931](#) (if approved)
Intended status: Informational
Expires: April 22, 2017

D. Harkins
HP Enterprise
October 19, 2016

Adding Support for Salted Password Databases to EAP-pwd
draft-harkins-salted-eap-pwd-07

Abstract

EAP-pwd is an EAP method that uses a shared password for authentication using a technique that is resistant to dictionary attack. It included support for raw keys and [\[RFC2759\]](#)-style double hashing of a password but did not include support for salted passwords. There are many existing databases of salted passwords and it is desirable to allow their use with EAP-pwd.

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 April 22, 2017.

Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4.e](#) of

Internet-Draft

Abbreviated Title

October 2016

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
1.1.	Background	2
1.2.	Keyword Definition	3
2.	Salted Passwords in EAP-pwd	3
2.1.	Password Pre-Processing	3
2.2.	The Salting of a Password	4
2.3.	Using UNIX crypt	5
2.4.	Using scrypt	5
2.5.	Using PBKDF2	6
2.6.	Protocol Modifications	6
2.7.	Payload Modifications	7
3.	Acknowledgements	8
4.	IANA Considerations	8
5.	Security Considerations	8
6.	References	9
6.1.	Normative References	9
6.2.	Informative References	9
	Author's Address	10

[1.](#) Introduction

[1.1.](#) Background

Databases of stored passwords present an attractive target for attack--get access to the database, learn the passwords. To confound such attacks a random "salt" was hashed with the password and the resulting digest stored, along with the salt, instead of the raw password. This has the effect of randomizing the password so if two distinct users have chosen the same password the stored, and salted, password will be different. It also requires an adversary who has compromised the security of the stored database to launch a dictionary attack per entry to recover passwords.

Dictionary attacks, especially using custom hardware, represent real-world attacks and merely salting a password is insufficient to protect a password database. To address these attacks an sequential memory hard function such as described in [[RFC7914](#)] is used.

While salting a password database is not sufficient to deal with many real-world attacks the historic popularity of password salting means there are a large number of such databases deployed and EAP-pwd needs to be able to support them. In addition, EAP-pwd needs to be able to

support databases using more modern sequential memory hard functions for protection.

EAP-pwd imposes an additional security requirement on a database of salted passwords that otherwise would not exist, see [Section 5](#).

[1.2](#). Keyword Definition

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](#). Salted Passwords in EAP-pwd

[2.1](#). Password Pre-Processing

EAP-pwd is based on the "dragonfly" password-authenticated key exchange (PAKE)--see [[RFC7664](#)]. This is a balanced PAKE and requires that each party to the protocol obtain an identical representation of a processed password (see [Section 5](#)). Salting of a password is therefore treated as an additional password pre-processing technique of EAP-pwd. The salt and digest to use is conveyed to the peer by the server and the password is processed prior to fixing the password element (see [Section 2.8.3 of \[RFC5931\]](#)).

This memo defines eight (8) new password pre-processing techniques for EAP-pwd:

- o TBD1: a random salt with SHA-1 ([[SHS](#)])
- o TBD2: a random salt with SHA-256 ([[SHS](#)])
- o TBD3: a random salt with SHA-512 ([[SHS](#)])
- o TBD4: UNIX crypt() ([[CRY](#)])

- o TBD5: scrypt ([\[RFC7914\]](#))
- o TBD6: PBKDF2 with SHA-256 ([\[RFC2898\]](#))
- o TBD7: PBKDF2 with SHA-512 ([\[RFC2898\]](#))
- o TBD8: SASLprep then a random salt with SHA-1 ([\[SHS\]](#))
- o TBD9: SASLprep then a random salt with SHA-256 ([\[SHS\]](#))
- o TBD10: SASLprep then a random salt with SHA-512 ([\[SHS\]](#))

- o TBD11: SASLprep then UNIX crypt() ([\[CRY\]](#))
- o TBD12: OpaqueString then scrypt ([\[RFC7914\]](#))
- o TBD13: OpaqueString then PBKDF2 with SHA-256 ([\[RFC2898\]](#))
- o TBD14: OpaqueString then PBKDF2 with SHA-512 ([\[RFC2898\]](#))

When passing salt, the size of the salt SHOULD be at least as long as the message digest of the hash algorithm used. There is no guarantee that deployed salted databases have followed this rule, and in the interest of interoperability, an EAP peer SHOULD NOT abort an EAP-pwd exchange if the length of the salt conveyed during the exchange is less than the message digest of the indicated hash algorithm.

UNIX crypt(), scrypt, and PBKDF2 impose additional formatting requirements on the passed salt. See below.

SASLprep has been deprecated but databases treated with SASLprep exist and it is necessary to provide code points for them. When using SASLprep a password SHALL be considered a "stored string" per [\[RFC3454\]](#) and unassigned code points are therefore prohibited. The output of SASLprep SHALL be the binary representation of the processed UTF-8 character string. Prohibited output and unassigned codepoints encountered in SASLprep pre-processing SHALL cause a failure of pre-processing, and the output SHALL NOT be used with EAP-pwd.

When performing one of TBD12-TBD14 the password SHALL be a UTF-8

string and SHALL be pre-processed by applying the Preparation and Enforcement steps of the OpaqueString profile in [RFC7613] to the password. The output of OpaqueString, also a UTF-8 string, becomes the EAP-pwd password and SHALL be hashed with the indicated algorithm.

There is a large number of deployed password databases that use [RFC7616]-style salting and hashing but these deployments require a nonce contribution by the client (as well as the server) and EAP-pwd does not have the capability to provide that information.

[2.2.](#) The Salting of a Password

For both parties to derive the same salted password there needs to be a canonical method of salting a password. When using EAP-pwd, a password SHALL be salted by hashing the password followed by the salt using the designated hash function:

```
salted-password = Hash(password | salt)
```

Harkins

Expires April 22, 2017

[Page 4]

Internet-Draft

Abbreviated Title

October 2016

The server stores the salted-password, and the salt, in its database and the client derives the salted-password on-the-fly.

[2.3.](#) Using UNIX crypt

Different algorithms are supported with the UNIX crypt() function. The particular algorithm used is indicated by prepending an encoding of "setting" to the passed salt. The specific algorithm used is opaque to EAP-pwd as the entire salt, including the encoded "setting", is passed as an opaque string for interpretation by crypt(). The salted password used for EAP-pwd SHALL be the output of crypt():

```
salted-password = crypt(password, salt)
```

The server stores the salted-password, and the encoded algorithm plus salt, in its database and the client derives the salted-password on-the-fly.

If the server indicates a crypt() algorithm that is unsupported by the client, the exchange fails and the client MUST terminate the connection.

[2.4.](#) Using scrypt

The scrypt function takes several parameters:

- o N, the cost parameter
- o r, the block size
- o p, the parallelization parameter
- o dkLen, the length of the output

These parameters are encoded into the "salt" field of the modified EAP-pwd message. Parameters r and dkLen SHALL be 16-bit integers in network order. The other parameters SHALL each be 32-bit integers in network order. The "salt" field that gets transmitted in EAP-pwd SHALL therefore be:

N || r || p || dkLen || salt

where || represents concatenation.

The value of N represents the exponent taken to the power of two in order to determine the CPU/Memory cost of scrypt-- i.e. the value is 2^N . Per [[RFC7914](#)] the resulting CPU/Memory cost value SHALL be less

than $2^{(128 * r / 8)}$ and the value p SHALL be less than or equal to $((2^{32} - 1) * 32) / (128 * r)$.

Note: EAP-pwd uses the salted password directly as the authentication credential and will hash it with a counter in order to obtain a secret element in a finite field. Therefore it makes little sense to use dkLen greater than the digest of the underlying hash function but the capability is provided to do so anyway.

[2.5.](#) Using PBKDF2

The PBKDF2 function requires two parameters:

- o c, the iteration count

- o dkLen, the length of the output

These parameters are encoded into the "salt" field of the modified EAP-pwd message. The parameters SHALL be 16-bit integers in network order. The "salt" field that gets transmitted in EAP-pwd SHALL therefore be:

```
c || dkLen || salt
```

where || represents concatenation.

Note: EAP-pwd uses the salted password directly as the authentication credential and will hash it with a counter in order to obtain a secret element in a finite field. Therefore it makes little sense to use dkLen greater than the digest of the underlying hash function but the capability is provided to do so anyway.

[2.6.](#) Protocol Modifications

Like all EAP methods, EAP-pwd is server initiated. The server is required to indicate its intentions, including the password pre-processing it wishes to use, before it knows the identity of the client. This limits the ability of the server to support multiple salt digests simultaneously in a single password database. To support multiple salt digests simultaneously, it is necessary to maintain multiple password databases and use the routable portion of the client identity to select one when initiating EAP-pwd.

The server uses the EAP-pwd-ID/Request to indicate the password pre-processing technique. The client indicates its acceptance of the password pre-processing technique and identifies itself in the EAP-pwd-ID/Response. If the client does not accept any of the offered pre-processing techniques it SHALL terminate the exchange. Upon

receipt of the EAP-pwd-ID/Response, the server knows the identity of the client and can look up the client's salted password and the salt from the database. The server adds the length of the salt and the salt itself to the EAP-pwd-Commit/Request message (see [Section 2.7](#)).

The server can fix the password element ([Section 2.8.3 of \[RFC5931\]](#)) as soon as the salted password has been looked up in the database. The client, though, is required to wait until receipt of the server's

Thanks to Stefan Winter and the eduroam project for its continued interest in using EAP-pwd. Thanks to Simon Josefsson for his advice on support for scrypt and PBKDF2.

4. IANA Considerations

IANA is instructed to allocate fourteen (14) values from the "password preprocessing method registry" established by [[RFC5931](#)] and replace TBD1, TBD2, TBD3, TBD4, TBD5, TBD6, TBD7, TBD8, TBD9, TBD10, TBD11, TBD12, TBD13, and TBD14 above with the values assigned.

5. Security Considerations

EAP-pwd requires each side to produce an identical representation of the (processed) password before the password element can be fixed. This symmetry undercuts one of the benefits to salting a password database because the salted password from a compromised database can be used directly to impersonate the EAP-pwd client--since the plaintext password need not be recovered, no dictionary attack is needed. While the immediate effect of such a compromise would be compromise of the server, the per-user salt would still prevent the adversary from recovering the password, barring a successful dictionary attack, to use for other purposes.

Salted password databases used with EAP-pwd MUST be afforded the same level of protection as databases of plaintext passwords.

Hashing a password with a salt increases the work factor for an attacker to obtain the cleartext password but dedicated hardware makes this increased work factor increasingly negligible in real-world scenarios. Cleartext password databases SHOULD be protected with a scheme that uses a sequential memory hard function such as [[RFC7914](#)].

Plain salting techniques are included for support of existing databases. scrypt and PBKDF2 techniques are RECOMMENDED for new password database deployments.

EAP-pwd sends the salt in the clear. If EAP-pwd is not tunneled in another, encrypting, EAP method, an adversary that can observe traffic from server to authenticator or from authenticator to client would learn the salt used for a particular user. While knowledge of a salt by an adversary may be of a somewhat dubious nature (pre-image resistance of the hash function used will protect the client's password and, as noted above, the database of salted passwords must

still be protected from disclosure), it does represent potential additional meta-data in the hands of a untrusted third party.

6. References

6.1. Normative References

- [CRY] "crypt(3) man page",
<<http://man7.org/linux/man-pages/man3/crypt.3.html>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [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>>.
- [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>>.
- [RFC5931] Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", [RFC 5931](#), August 2010.
- [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>>.
- [RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", [RFC 7914](#), DOI 10.17487/RFC7914, August 2016, <<http://www.rfc-editor.org/info/rfc7914>>.
- [SHS] National Institute of Standards and Technology, , "Federal Information Processing Standard Publication 180-4: Secure Hash Standard (SHS)", March 2012,
<<http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>>.

6.2. Informative References

- [RFC2759] Zorn, G., "Microsoft PPP CHAP Extensions, Version 2", [RFC](#)

[2759](http://www.rfc-editor.org/info/rfc2759), DOI 10.17487/RFC2759, January 2000,
<<http://www.rfc-editor.org/info/rfc2759>>.

Harkins

Expires April 22, 2017

[Page 9]

Internet-Draft

Abbreviated Title

October 2016

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

[RFC7664] Harkins, D., Ed., "Dragonfly Key Exchange", [RFC 7664](http://www.rfc-editor.org/info/rfc7664), DOI 10.17487/RFC7664, November 2015,
<<http://www.rfc-editor.org/info/rfc7664>>.

Author's Address

Dan Harkins
HP Enterprise
1322 Crossman Avenue
Sunnyvale, CA 94089-1113
United States of America

Email: dharkins@arubanetworks.com

Harkins

Expires April 22, 2017

[Page 10]