Internet Engineering Task Force (IETF)
Request for Comments: 8146
Updates: 5931
Category: Informational
ISSN: 2070-1721

D. Harkins
HP Enterprise

April 2017

Adding Support for Salted Password Databases to EAP-pwd

Abstract

EAP-pwd is an Extensible Authentication Protocol (EAP) method that utilizes a shared password for authentication using a technique that is resistant to dictionary attacks. It includes support for raw keys and double hashing of a password in the style of Microsoft Challenge Handshake Authentication Protocol version 2 (MSCHAPv2), but it does 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 document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8146.

Table of Contents

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; even 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, a 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 4.

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 [RFC2119].

2. Salted Passwords in EAP-pwd

2.1. Password Preprocessing

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 4). Therefore, salting of a password is treated as an additional password preprocessing technique of EAP-pwd. The salt and digest to use are 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 preprocessing techniques for EAP-pwd:

- 0x03: a random salt with SHA-1
- 0x04: a random salt with SHA-256
- 0x05: a random salt with SHA-512
- 0x06: UNIX crypt()
- 0x07: scrypt
- 0x08: PBKDF2 with SHA-256
- 0x09: PBKDF2 with SHA-512
- 0x0A: SASLprep then a random salt with SHA-1
- 0x0B: SASLprep then a random salt with SHA-256
- 0x0C: SASLprep then a random salt with SHA-512
- 0x0D: SASLprep then UNIX crypt()
- 0x0E: OpaqueString then scrypt
- 0x0F: OpaqueString then PBKDF2 with SHA-256
- 0x10: OpaqueString then PBKDF2 with SHA-512

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() ([CRY]), scrypt ([RFC7914]), and PBKDF2 ([RFC8018]) impose additional formatting requirements on the passed salt. See below.

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

SASLprep has been deprecated, but databases treated with SASLprep exist; it is necessary to provide code points for them. When using

SASLprep, a password SHALL be considered a "stored string" per [RFC3454]; therefore, unassigned code points are prohibited. The output of SASLprep SHALL be the binary representation of the processed UTF-8 character string. Prohibited output and unassigned code points encountered in SASLprep preprocessing SHALL cause a failure of preprocessing, and the output SHALL NOT be used with EAP-pwd.

When performing one of the preprocessing techniques (0x0E-0x10), the password SHALL be a UTF-8 string and SHALL be preprocessed 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 salting and hashing in the style of [RFC7616], 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)
```

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:

- N, the cost parameter
- r, the block size
- p, the parallelization parameter
- 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 length of the digest produced by the underlying hash function, but the capability is provided to do so anyway.

2.5. Using PBKDF2

The PBKDF2 function requires two parameters:

- c, the iteration count
- 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 a dkLen greater than the length of the digest produced by 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, and the initial identity supplied by the client is not useful for authentication purposes. Because of this, the server is required to indicate its intentions, including the password preprocessing it wishes to use, before it knows the true identity of the client. This prevents the server from supporting 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 preprocessing technique. The client indicates its acceptance of the password preprocessing technique and identifies itself in the EAP-pwd-ID/Response. If the client does not accept any of the offered preprocessing 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 EAP-pwd-Commit/Request before it begins fixing the password element.

2.7. Payload Modifications

When a salted password preprocessing technique is agreed upon during the EAP-pwd-ID exchange, the EAP-pwd-Commit payload is modified to include the salt and salt length (see Figure 1). The server passes the salt and salt length in the EAP-pwd-Commit/Request; the client's EAP-pwd-Commit/Response is unchanged, and it MUST NOT echo the salt length and salt in its EAP-pwd-Commit/Response.

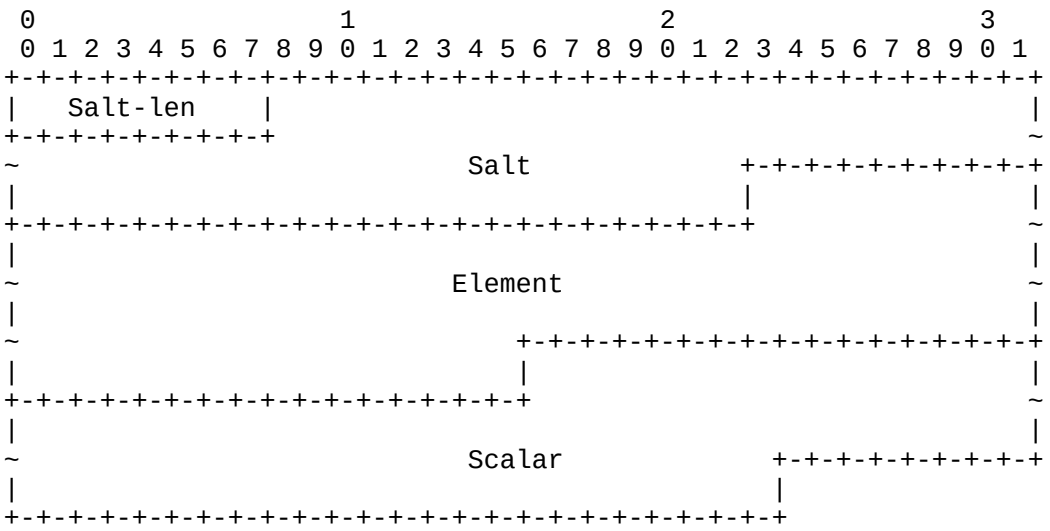

Figure 1: Salted EAP-pwd-Commit/Request

The "salt-len" SHALL be non-zero, and it indicates the length, in octets, of the salt that follows. The "Salt" SHALL be a binary string. The "Element" and "Scalar" are encoded according to Section 3.3 of [RFC5931].

Note: when a non-salted password preprocessing method is used, for example, any of the methods from [RFC5931], the EAP-pwd-Commit payload MUST NOT be modified to include the salt and salt length.

3. IANA Considerations

IANA has allocated fourteen (14) values from the "password preprocessing method registry" established by [RFC5931].

4. 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].

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.

5. References

5.1. Normative References

[CRY] Linux Programmer's Manual, "CRYPT(3)", August 2015, <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, 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>.

[RFC5931] Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, DOI 10.17487/RFC5931, August 2010, <http://www.rfc-editor.org/info/rfc5931>.

[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>.

[RFC8018] Moriarty, K., Ed., Kaliski, B., and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1", RFC 8018, DOI 10.17487/RFC8018, January 2017, <http://www.rfc-editor.org/info/rfc8018>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

5.2. Informative References

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

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

Acknowledgements

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.

Author's Address

Dan Harkins
HP Enterprise
3333 Scott Boulevard
Santa Clara, CA 95054
United States of America

Email: dharkins@arubanetworks.com