Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: October 26, 2015

N. McCallum
Red Hat, Inc.

April 24, 2015

SPAKE Pre-Authentication
draft-mccallum-kitten-krb-spake-preauth-00

Abstract

This document defines a new password authenticated key exchange based pre-authentication mechanism for performing Kerberos authentication. This mechanism has three goals. First, it makes Kerberos pre-authentication more resilient against time synchronization errors by removing the need to transfer an encrypted timestamp. Second, it increases the security of the Kerberos pre-authentication exchange by making offline brute-force attacks impossible. Third, it enables the use of secure second factor authentication without FAST by utilizing the existing trust relationship established by the shared first factor.

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 October 26, 2015.

Copyright Notice

Table of Contents

1. Introduction

Existing Kerberos pre-authentication methods generally involve encrypting a well-known value, often a timestamp, in the client principal's long-term secret. This approach has two signficant drawbacks. First, where a timestamp is used, time synchronization between the client and KDC becomes a point of fragility. Second, a passive attacker can perform offline brute-force attack against the transferred packet.

1.1. Properties of PAKE

Diffie-Hellman key exchange (DHKE) is a mechanism by which two parties can derive a shared session key over an insecure network which cannot be derived by a passive attacker. DHKE could replace the above approaches without their drawback. Unfortunately, DHKE can be easily compromised by an active attacker by using a man-in-the-middle attack.

Password authenticated key exchange (PAKE) is a technique which extends DHKE by using a shared secret - in the case of Kerberos, the client principal's long-term secret - in order to protect the underlying DHKE from an active attacker. This property of PAKE makes it ideal for use as a Kerberos pre-authentication mechanism.

1.2. PAKE and 2FA

Using PAKE in a pre-authentication mechanism also has another benefit when coupled with two-factor authentication (2FA). 2FA methods often require the secure transfer of plaintext material to the KDC for verification. This includes one-time passwords, challenge/response signatures and biometric data. Attempting to encrypt this data using the long-term password results in packets that are vulnerable to offline brute-force attack; a problem we are already trying to solve.

In the past, this problem has been mitigated by FAST which uses a secondary trust relationship to create a secure encryption channel within which pre-authentication data can be sent. However, the requirement for a secondary trust relationship has proven to be cumbersome to deploy and often introduces third parties into the trust chain (such as certificate authorities). By using PAKE, the calculated session key can be leveraged into an encryption key for 2FA data without the need for a secondary trust relationship.

1.3. Which PAKE?

The most common technique in PAKE is to modify a DHKE by encrypting one or more of the public keys exchanged using the shared secret. This ensures that only the party which knows the shared secret can decrypt the public key and complete the DHKE. The earliest widely deployed PAKE, which also used this model, was the Diffie-Hellman Encrypted Key Exchange (DH-EKE). Unfortunately, DH-EKE depends on the property that the public key is indistinguishable from random. This unfortunately means that DH-EKE cannot be used with elliptic curve Diffie-Hellman (ECDH).

SPAKE [I-D.irtf-cfrg-spake2] is a variant of the technique used by DH-EKE which ensures that all public key encryption and decryption

operations result in a member of the underlying group. This property allows SPAKE to be used with ECDH, permitting the pre-authentication packets to use markedly smaller key sizes with equivalent security. Additionally, unlike some other PAKE methods, SPAKE can complete the key exchange in just a single round-trip. These properties make SPAKE an ideal PAKE to use for Kerberos pre-authentication.

2. Document Conventions

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].

3. Prerequisites

3.1. PA-ETYPE-INFO2

KDCs which implement SPAKE pre-authentication MUST return a PA-ETYPE-INFO2. This PA-ETYPE-INFO2 MUST contain only one ETYPE-INFO2-ENTRY. This ETYPE-INFO2-ENTRY will be the etype used for all cryptographic operations and to select the long-term key for the SPAKE operation.

3.2. Cookie Support

KDCs which implement SPAKE pre-authentication MUST have some secure mechanism for retaining state between AS-REQs. This method will most commonly be an encrypted PA-FX-COOKIE. If PA-FX-COOKIE is used, then clients MUST also support PA-FX-COOKIE.

3.3. More Pre-Authentication Data Required

Both KDCs and clients which implement SPAKE pre-authentication MUST support the use of KDC_ERR_MORE_PREAUTH_DATA_REQUIRED.

4. SPAKE Pre-Authentication Message Protocol

This section will describe the flow of messages when performing SPAKE pre-authentication. We will begin by explaining the most verbose version of the protocol which all implementations MUST support. Then we will describe several optional optimizations to reduce round-trips.

All messages will use the following pre-authentication type:

PA-SPAKE TBD

4.1. First Pass

The SPAKE pre-authentication exchange begins when the client sends an initial authentication service request (AS-REQ) without pre-authentication data. Upon receipt of this AS-REQ, a KDC which requires pre-authentication and supports SPAKE SHOULD reply with KDC_ERR_PREAUTH_REQUIRED containing an empty PA-SPAKE pre-authentication data. This indicates to the client that the KDC supports SPAKE pre-authentication.

4.2. Second Pass

Once the client knows that the KDC supports SPAKE pre-authentication and the client desires to use it, the client will generate a new AS-REQ which includes a SPAKESupport message. This message indicates to the KDC which groups the client prefers for the SPAKE operation. The groups sequence is ordered from the most preferred group to the least preferred group.

```
SPAKESupport ::= SEQUENCE {
    groups   SEQUENCE (SIZE(1..MAX)) OF OBJECT IDENTIFIER,
    ...
}
```

Upon receipt of the SPAKESupport message, the KDC will select a group. The KDC SHOULD choose a group from the groups provided by the SPAKESupport message. However, if the SPAKESupport message does not contain any group that is supported by the KDC, the KDC MAY select another group in hopes that the client might support it.

Once the KDC has selected a group, the KDC will reply to the client with KDC_ERR_MORE_PREAUTH_DATA_REQUIRED containing a SPAKEChallenge message.

```
SPAKEChallenge ::= SEQUENCE {
    group   OBJECT IDENTIFIER,
    pubkey  OCTET STRING,
    factors SEQUENCE (SIZE(1..MAX)) OF SPAKESecondFactor,
    ...
}
```

The group field indicates the KDC-selected group used for all SPAKE calculations.

The pubkey field indicates the KDC's public key generated using the M constant in the SPAKE algorithm. The format of this field's contents and the value of the M constant will depend upon the group chosen.

The factors field contains an unordered list of second factors which can be used to complete the authentication. Each second factor is represented by a SPAKESecondFactor.

```
SPAKESecondFactor ::= SEQUENCE {
    type Int32,
    data OCTET STRING OPTIONAL
}
```

The type field is a unique integer which identifies the second factor type. The factors field of SPAKEChallenge MUST NOT contain more than one SPAKESecondFactor with the same type value.

The data field contains optional challenge data. The contents in this field will depend upon the second factor type chosen.

4.3. Third Pass

Upon receipt of the SPAKEChallenge message, the client will complete its part of of the SPAKE process resulting in a public key and a SPAKE key. Then, the client derives the session key from the SPAKE key.

Next, the client chooses one of the second factor types listed in the challenge field of the SPAKEChallenge message and gathers whatever data is required for this second factor type; possibly using the challenge data for this second factor type. Finally, the client sends an AS-REQ with a SPAKEResponse message.

```
SPAKEResponse ::= SEQUENCE {
    pubkey OCTET STRING,
    factor EncryptedData, -- SPAKESecondFactor
    ...
}
```

The pubkey field indicates the client's public key generated using the N constant in the SPAKE algorithm. The format of this field's contents and the value of the N constant will depend upon the group chosen by the KDC.

The factor field indicates the client's chosen second factor data. This data is encrypted using a derivation of the PAKE key. The plain text inside the EncryptedData is an encoding of SPAKESecondFactor. Once decoded, the SPAKESecondFactor contains the type of the second factor and any optional data used. The contents of the data field will depend on the second factor type chosen. The client MUST NOT send a response containing a second factor type which was not listed in the factors field of the SPAKEChallenge message.

When the KDC receives the SPAKEResponse message from the client, it will use the pubkey to complete the SPAKE process resulting in the SPAKE key. The KDC will then derive the session key from the SPAKE key. If the SPAKE process is successful, the client and the KDC will have the same SPAKE key and session key.

The KDC will then use the same derivation of the PAKE key as the client to decrypt the factors field. If decryption is successful, the first factor is successfully validated. Upon decoding the SPAKESecondFactor, the KDC then validates the second factor. If both factors successfully validate, the KDC responds by issuing a TGT encrypted in the session key. If either factor fails to validate, an appropriate KRB-ERROR is returned. If validation of the second factor requires further round-trips, the KDC MUST reply to the client with KDC_ERR_MORE_PREAUTH_DATA_REQUIRED and include an encoded EncryptedData message containing an encoded, encrypted SPAKESecondFactor message. As before, the type field of this message will contain the second factor type and the data field will optionally contain second factor type specific data.

4.4. Subsequent Passes

Any number of EncryptedData message roundtrips may occur. The precise details of this are second factor type specific. The only constraint is that if a client receives EncryptedData from the KDC it MUST reply with a subsequent AS-REQ with EncryptedData. These exchanges conclude when the KDC indicates that either authentication has failed (via an appropriate KRB-ERROR) or a TGT has been issued (via an AS-REP).

4.5. Optimizations

The full protocol has two possible optimizations.

First, the KDC MAY reply to the initial AS-REQ containing no pre-authentication data with KDC_ERR_PREAUTH_REQUIRED and a SPAKEChallenge message. In this case the KDC optimistically selects a group which the client may not support. If the group chosen by the SPAKEChallenge message is supported by the client, the client MUST skip to the Third Pass by issuing an AS-REQ with a SPAKEResponse message. If the KDC's chosen group is not supported by the client, the client MUST continue to the Second Pass as if it had received an empty PA-SPAKE. Clients MUST support this optimization.

Second, clients which are somehow aware that they should use SPAKE pre-authentication with a KDC MAY skip the First Pass entirely. KDCs MUST support this optimization.

5. Key Derivation

5.1. Session Key Derivation

Implementations MUST NOT use the SPAKE key (denoted by K in Section 2 of SPAKE [I-D.irtf-cfrg-spake2]) directly for any cryptographic operation. Instead, the SPAKE key MUST be used to derive a session key as defined in this section. This method differs slightly from the method used to generate K' in Section 3 of SPAKE [I-D.irtf-cfrg-spake2] in order to provide message integrity.

Whenever a SPAKESupport or SPAKEChallenge message is sent or received, its encoded representation MUST be checksummed according to the mandatory checksum for the negotiated etype. Similarly, whenever a SPAKEResponse message is sent or received, its pubkey attribute MUST be encoded separately and checksummed. Each checksum MUST be concatenated to the end of a buffer containing the checksums of all previous messages in the order they were sent or received. This buffer is called the transcript hash.

In order to generate the session key, we concatenate the following values and pass the resulting buffer, along with the client principal's key, as input to the pseudo-random function defined for the etype: the uncanonicalized client principal, the uncanonicalized server principal, the SPAKE key and the transcript hash. During concatenation, each value must be prepended by the value's length as defined in Section 2 of SPAKE [I-D.irtf-cfrg-spake2]. The output from the pseudo-random function is the session key.

5.2. Encryption Key Derivation

Each EncryptedData message sent MUST use a distinct key, including the factor field in the SPAKEResponse message. This key is derived by using a method identical to the Session Key Derivation except for an additional input value appended to the end of the pseudo-random function input: an eight-byte big-endian count of the number of EncryptedData messages sent or received.

6. Second Factor Types

This document defines one second factor type:

SF-NONE 1

This second factor type indicates that no second factor is used. Whenever a SPAKESecondFactor is used with SF-NONE, the data field MUST be omitted. The SF-NONE second factor always successfully validates.

7. Recommended Groups

In the interest of broad compatibility, the following groups are recommended for implementation:

- P256
- P384
- P521

All of these groups encode their public keys in SEC1 compressed format.

8. Security Considerations

All of the security considerations from SPAKE [I-D.irtf-cfrg-spake2] apply here as well.

Message integrity is provided by the transcript hash being integrated into the final session key and encryption keys. This prevents downgrade attacks on SPAKESupport and SPAKEChallenge. However, please note that message integrity confirmation does not occur until after SPAKEResponse is sent. This means that data in SPAKEChallenge should be treated as untrusted before this point. The most obvious implication is that SPAKEChallenge should not contain information to prompt the users with as it may be modified to lie or try to do other nasty things.

However, since the response is encrypted, it is not possible to exploit the untrustworthiness of SPAKEChallenge into an encryption or signing oracle. Any attempt to do this will result in data that cannot be decrypted.

Given that each EncryptedData will begin a new encryption context, a failure to properly derive the encryption keys will result in a situation where the risk of compromise is non-negligible.

Weak checksums present a risk to the transcript hash. Any etype with a checksum based on one of the following algorithms MUST NOT be used:

- CRC32
- MD4
- MD4
- MAC

If more than one round-trip is required to authenticate a second factor, this reveals to an online brute-force attacker that his first factor guess was correct. This should be avoided if at all possible.

Both the size of the EncryptedData and the number of EncryptedData messages may reveal information about the second factor used in an authentication. Care should be taken to keep second factor messages as small and as few as possible.

9. IANA Considerations

One registry for numeric values has been created: Kerberos Second Factor Type Numbers. These are signed values ranging from -2147483648 to 2147483647. Positive values should be assigned only for algorithms specified in accordance with this specification for use with Kerberos or related protocols. Negative values are for private use; local and experimental algorithms should use these values. Zero is reserved and may not be assigned.

10. Normative References

[I-D.irtf-cfrg-spake2] Ladd, W., "SPAKE2, a PAKE", draft-irtf-cfrg-spake2-01 (work in progress), February 2015.

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

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011.

Appendix A. Acknowledgements

Simo Sorce (Red Hat)
Nico Williams (Oracle)
Greg Hudson (MIT)
Tom Yu (MIT)

Author's Address

Nathaniel McCallum
Red Hat, Inc.
100 East Davie Street
Raleigh, NC 27601
USA

EMail: npmccallum@redhat.com