Network Working Group A. Dekok
Internet-Draft Network RADIUS SARL
Updates: 5880 (if approved) M. Jethanandani
Intended status: Standards Track Kloud Services
Expires: 7 August 2024 S. Agarwal
Cisco Systems, Inc
A. Mishra
Aalyria Technologies
A. Saxena
Ciena Corporation
4 February 2024

Meticulous Keyed ISAAC for BFD Authentication
draft-ietf-bfd-secure-sequence-numbers-13

Abstract

This document describes a new BFD Authentication mechanism, Meticulous Keyed ISAAC. This mechanism can be used to authenticate BFD packets with less CPU time cost than using MD5 or SHA1, with the tradeoff of decreased security. This mechanism cannot be used to signal state changes, but it can be used as an authenticated signal to maintain a session in the the "Up" state.

This document updates RFC 5880.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

BFD [RFC5880] (Section 6.7.2) defines a number of authentication mechanisms, including Simple Password, and various other methods based on MD5 and SHA1 hashes. The benefit of using cryptographic hashes is that they are secure. The downside to cryptographic hashes is that they are expensive and time consuming on resource-constrained hardware.

When BFD packets are unauthenticated, it is possible for an attacker to forge, modify, and/or replay packets on a link. These attacks have a number of side effects. They can cause parties to believe that a link is down, or they can cause parties to believe that the link is up when it is, in fact, down. The goal of this specification is to use a simple method to prevent spoofing of the BFD session being "Up". We therefore define a fast Auth Type method which allows parties securely signal that they are still in the Up state.

This document proposes the use of an Authentication method which provides meticulous keying, but which has less impact on resource constrained systems. The algorithm chosen is a seeded pseudo-random number generator named ISAAC [ISAAC]. ISAAC has been subject to significant cryptanalysis in the past thirty years, and has not yet been broken. It requires only a few CPU operations per generated 32-bit number, can take a large secret key as a seed, and it has an extremely long period. These properties make it ideal for use in BFD.

2. Requirements Language

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. Updating RFC 5880

Some of the state variables in BFD [RFC5880] (Section 6.8.1), are related to the authentication type being used for a particular session. However, the definitions given in BFD [RFC5880] are specific to Keyed MD5 or SHA1 Authentication, which limit their utility for new authentication types. This specification updates the definition of some of the state variables as given below.

These updated definitions are entirely compatible with the definitions given in BFD [RFC5880] (Section 6.8.1), and require no changes to existing configurations or implementations. Instead, the updated definitions clarify that the state variables apply to the current authentication type, no matter what it is.

These updated definitions also mean that Authentication Sections SHOULD include a Sequence Number field. Where a Sequence Number is not used (as with Simple Password) the variables bfd.RcvAuthSeq and bfd.XmitAuthSeq MUST be set to zero.

bfd.AuthType:

The current authentication type in use for this session, as defined in BFD [RFC5880] (Section 4.1), or zero if no authentication is in use. Note that the session MAY change AuthType during a session. For example, where the session transitions from a more secure AuthType to a less secure one, or vice versa.

Packets which indicate a state transition SHOULD use a secure AuthType. Where the bfd.SessionState value is Up, packets MAY use a less secure AuthType, such as Meticulous Keyed ISAAC.

bfd.RcvAuthSeq:
A 32-bit unsigned integer containing the last sequence number for the current Authentication Section that was received. The initial value is unimportant.

bfd.XmitAuthSeq:
A 32-bit unsigned integer containing the next sequence number for for the Authentication Section which will be transmitted. This variable MUST be initialized to a random 32-bit value.

bfd.AuthSeqKnown:
Set to 1 if the next expected Authentication Section has a sequence number which is known, or 0 if it is not known. This variable MUST be initialized to zero.

This variable MUST be set to zero after no packets have been received on this session for at least twice the Detection Time. This ensures that the sequence number can be resynchronized if the remote system restarts.

4. Architecture of the Auth Type Method

When BFD uses authentication, methods using MD5 or SHA1 are CPU intensive, and can negatively impact systems with limited computational power.

However, once the session transitions into the Up state, there is no need to authenticate every packet. An optimized authentication mechanism as described in Optimizing BFD Authentication [I-D.ietf-bfd-optimizing-authentication], permits BFD to use a relaxed authentication, that satisfies the ability to provde a less expensive authentication, but strong enough that periodic reauthentication is not strictly required to prevent a person-in-the-middle attack.

We use ISAAC here as a way to generate an infinite stream of pseudo-random numbers, referred to here as "Auth-Key"s. With Meticulous Keyed ISAAC, these Auth Keys are used as a signal that the sending party is authentic. That is, only the sending party can generate the correct Auth-Keys. Therefore if the receiving party sees a correct Auth-Key, then only the sending party could have generated it. The sender is therefore authentic, even if the packet contents have potentially been modified in transit.

Note that with this Auth Type method, the full packet contents are not signed or authenticated. Therefore, the Meticulous Keyed ISAAC method MUST NOT be used to signal BFD state changes. For BFD state changes, and a more optimized way to authenticate packets, please refer to BFD Authentication [I-D.ietf-bfd-optimizing-authentication]. Instead, the packets containing Meticulous Keyed ISAAC are only a signal that the sending party is still alive, and that the sending party is authentic. That is, this Auth Type method must only be used when bfd.SessionState=Up, and the State (Sta) field equals 3 (Up).

5. Meticulous Keyed ISAAC Authentication Format

If the Authentication Present (A) bit is set in the header, and the State (Sta) field equals 3 (Up), and the Authentication Type field contains TBD1 (Meticulous Keyed ISAAC), the Authentication Section has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Auth Type   |   Auth Len    |  Auth Key ID  |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Seed                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Auth-Key                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Auth Type:
The Authentication Type, which in this case is TBD1 (Meticulous Keyed ISAAC). If the State (Sta) field value is not 3 (Up), then Meticulous Keyed ISAAC MUST NOT be used.

Auth Len:
The length of the Authentication Section, in bytes. For Meticulous Keyed ISAAC authentication, the length is 16.

Auth Key ID:
The authentication key ID in use for this packet. This allows multiple secret keys to be active simultaneously.

Reserved:
This field MUST be set to zero on transmit, and ignored on receipt.

Sequence Number:
The sequence number for this packet. For Meticulous Keyed ISAAC Authentication, this value is incremented once for each successive packet transmitted for a session. This provides protection against replay attacks.

Seed:
A 32-bit (4 octet) seed which is used in conjunction with the shared key in order to configure and initialize the ISAAC pseudo-random-number-generator (PRNG). It is used to identify and secure different "streams" of random numbers which are generated by ISAAC.

Auth-Key:
This field carries the 32-bit (4 octet) ISAAC output which is associated with the Sequence Number. The ISAAC PRNG MUST be configured and initialized as given in Section 10, below.

Note that the Auth-Key here does not include any summary or hash of the packet. The packet itself is completely unauthenticated.

When the receiving party receives a BFD packet with an expected sequence number and the correct corresponding ISAAC output in the Auth Key field, it knows that only the authentic sending party could have sent that message. The sending party is therefore "Up", and is the only one who could have sent the message.

6. Meticulous Keyed ISAAC Authentication

In this method of authentication, one or more secret keys (with corresponding key IDs) are configured in each system. One of the keys is used to seed the ISAAC PRNG. The output of ISAAC is used to signal that the sender is authentic. To help avoid replay attacks, a sequence number is also carried in each packet. For Meticulous Keyed ISAAC, the sequence number is incremented on every packet.

The receiving system accepts the packet if the key ID matches one of the configured Keys, and the Auth-Key derived from the selected Key, Seed, and Sequence Number matches the Auth-Key carried in the packet, and the sequence number is strictly greater than the last sequence number received (modulo wrap at 2^32)

Transmission Using Meticulous Keyed ISAAC Authentication

The Auth Type field MUST be set to TBD1 (Meticulous Keyed ISAAC). The Auth Len field MUST be set to 16. The Auth Key ID field MUST be set to the ID of the current authentication key. The Sequence Number field MUST be set to bfd.XmitAuthSeq.

The Seed field MUST be set to the value of the current seed used for this session.

The Auth-Key field MUST be set to the output of ISAAC, which depends on the secret Key, the current Seed, and the Sequence Number.

For Meticulous Keyed ISAAC, bfd.XmitAuthSeq MUST be incremented on each packet, in a circular fashion (when treated as an unsigned 32-bit value). The bfd.XmitAuthSeq MUST NOT be incremented by more than one for a packet.

Receipt using Meticulous Keyed ISAAC Authentication

If the received BFD Control packet does not contain an Authentication Section, or the Auth Type is not correct (TBD1 for Meticulous Keyed ISAAC), then the received packet MUST be discarded.

If the Auth Key ID field does not match the ID of a configured authentication key, the received packet MUST be discarded.

If the Auth Len field is not equal to 16, the packet MUST be discarded.

If bfd.AuthSeqKnown is 1, examine the Sequence Number field. For Meticulous keyed ISAAC, if the sequence number lies outside of the range of bfd.RcvAuthSeq+1 to bfd.RcvAuthSeq+(3*Detect Mult) inclusive (when treated as an unsigned 32-bit circular number space) the received packet MUST be discarded.

If bfd.MetKeyIsaacRcvKeyKnown is "true" and the Seed field does not match the current Seed value, bfd.MetKeyIsaacRcvAuthSeed, the packet MUST be discarded.

Calculate the current expected output of ISAAC, which depends on the secret Key, the current Seed, and the Sequence Number. If the value does not matches the Auth-Key field, then the packet MUST be discarded.

If bfd.MetKeyIsaacRcvKeyKnown is "false", the ISAAC related variables are initialized as per Section 10.2 using the contents of the packet.

Note that in some cases, calculating the expected output of ISAAC will result in the creation of a new "page" of 256 numbers. This process will irreversible, and will destroy the current "page". As a result, if the generation of a new output will create a new "page", the receiving party MUST save a copy of the entire ISAAC state before proceeding with this calculation. If the outputs match, then the saved copy can be discarded, and the new ISAAC state is used. If the outputs do not match, then the saved copy MUST be restored, and the modified copy discarded, or cached for later use.

7. New State variables for Meticulous Keyed ISAAC

This document defines a few new state variables for use with Meticulous Keyed ISAAC.

bfd.MetKeyIsaacRcvKeyKnown:
A boolean value which indicates whether or not the system knows the receive key for the Meticulous Keyed ISAAC Auth Type method. The initial value is "false". This value is changed to "true" when a party verifies that the other party has started to use the Meticulous Keyed ISAAC Auth Type method, with an authenticated Auth Key.

bfd.MetKeyIsaacRcvAuthBase:
A 32-bit unsigned integer containing a copy of the bfd.RcvAuthSeq number which is associated with the current ISAAC "page" for authenticating received packets.

bfd.MetKeyIsaacRcvAuthIndex:
An 8-bit number used to index within a particular "page" of pseudo-random numbers.

bfd.MetKeyIsaacRcvAuthSeed:
A 32-bit unsigned integer containing a copy of the Seed associated with received packets.

bfd.MetKeyIsaacRcvAuthData:
A data structure which contains the ISAAC data for the received Auth Type method.

bfd.MetKeyIsaacXmitKeyKnown:
A boolean value which indicates whether or not the system knows the xmit key for the Meticulous Keyed ISAAC Auth Type method. The initial value is "false". This value is changed to "true" when a party starts to transmit using the Meticulous Keyed ISAAC Auth Type method.

bfd.MetKeyIsaacXmitAuthBase:
A 32-bit unsigned integer containing a copy of the bfd.XmitAuthSeq number which is associated with the current ISAAC "page" for authenticating sent packets.

bfd.MetKeyIsaacXmitAuthIndex:
An 8-bit number used to index within a particular "page" of pseudo-random numbers.

bfd.MetKeyIsaacXmitAuthSeed:
A 32-bit unsigned integer containing a copy of the Seed associated with sent packets.

bfd.MetKeyIsaacXmitAuthData:
A data structure which contains the ISAAC data for the sending Auth Type method.

8. Secret Key

The security of this Auth Type depends on the Secret Key. The Secret Key is mixed with a per-session Seed as discussed below. The result is used to initialize a stream of pseudo-random numbers using the ISAAC random number generator

A particular Secret Key is identified via the Auth Key ID field. This Auth Key ID is either placed in the packet by the sender, or verified by the receiver. The Meticulous Keyed ISAAC authentication method permits systems to have multiple Secret Keys configured, but we do not discuss how those keys are managed or used. We do, however, require that a session MUST NOT change the Auth Key ID for Meticulous Keyed ISAAC, during a session. There is no defined way to re-sync or re-initialize an ongoing session with a different Auth Key ID and correspondingly different Secret Key

If this Auth Type method was defined as being initialized without a per-session Seed, then an attacker could pre-compute the ISAAC states for many keys, and perform an off-line dictionary attack. The use of the Seed makes these attacks infeasable.

For interoperability, the management interface by which the key is configured MUST accept ASCII strings, and SHOULD also allow for the configuration of any arbitrary binary string in hexadecimal form. Other configuration methods MAY be supported.

The Secret Key MUST be at least eight (8) octets in length, and SHOULD NOT be more than 128 octets in length.

There are no known issues with using the same secret Key for multiple Auth Type methods. However, it is RECOMMENDED that adminsitrators different Secret Keys for each Auth Type.

9. Transition to using ISAAC

Once a session transitions to the Up state, the packets MAY contain AuthType of Meticulous Keyed ISAAC. A system receiving such a packet will initialize the ISAAC PRNG state using the Seed from the packet. A system originating such a packet will generate a Seed, and place it into the packet which is then sent. Further discussion of initialization is below in Section 10.1 and Section 10.2.

There is no negotiation when using this Auth Type method. A sending system simply starts sending packets which contain Auth Type of Meticulous Keyed ISAAC.

Similarly, a receiving system sees that it has received a packet contains AuthType of Meticulous Keyed ISAAC when bfd.MetKeyIsaacRcvKeyKnown variable is "false". The receiving system then initializes its variables, and authenticates the received packet, by comparing the Auth Key in the packet with the key it generated itself.

Note that switch to a different AuthType method does not affect the values of the bfd.RcvAuthSeq or bfd.XmitAuthSeq variables. The variables MUST continue using their previous values which were using the previous AuthType method.

However, the operation of those variables MUST now satisfy the requirements of the new AuthType method. For example, the AuthType could change from Keyed SHA1 (where the variables were not updated on every packet) to Meticulous Keyed ISAAC (where the variables are updated on every packet).

That is, when changing AuthTypes in a session, the current value of the bfd.RcvAuthSeq and bfd.XmitAuthSeq variables is used as the initial value(s) for the new AuthType.

When there is a transition to using ISAAC the first time, the initial state has to be seeded. The next section describes this seeding process.

10. Seeding ISAAC

The Seed field is used to is used to identify and secure different "streams" of random numbers which are generated by ISAAC. Each session uses a different Seed, which is used along with the "Your Discriminator" field, and the Secret Key, to initialize ISAAC.

The value of the Seed field MUST be derived from a secure source. Exactly how this can be done is outside of the scope of this document.

A new Seed value MUST be created every time a BFD session transitions into the "Up" state. In order to prevent continuous rekeying, once the session is in the "Up" state, the Seed for a session MUST NOT be changed until another state transition occurs.

The data used to initialize the ISAAC PRNG is taken from the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Seed                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Your Discriminator                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Secret Key ...                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where the "Your Discriminator" field is taken from the BFD packet defined in RFC5880 Section 4.1 [RFC5880]. This field is taken from the respective values used by a sending system. For receiving systems, the field are taken from the received packet. The length of the Secret Key MUST be 1016 octets or less.

The data is padded to 1024 octets using zeroes, and then is processed throught the "randinit()" function of ISAAC. Pseudo-random numbers are then produced by calling the "isaac()" function.

For the sender, this calculation can be done outside of the BFD "fast path" as soon as the "Your Discriminator" value is known. For the receiver, this calculation can only be done when the Seed is received from the sender, and therefore needs to be done in the BFD "fast path".

The following figure give Seed and Your-Discriminator as 32-bit hex values, and the Secret Key as an eleven-character string. The subsequent figure shows the first eight Sequence numbers and corresponding Auth Key values which were generated using the above initial values.

```
Seed    0x0bfd5eed
Y-Disc  0x4002d15c
Key     RFC5880June
```

```
Sequence Auth Key
00000000 739ba88a
00000001 901e5075
00000002 8e84991c
00000003 93e534cd
00000004 fc213b4b
00000005 f78fc6e6
00000006 3a44db86
00000007 7dda6e6a
```

Note that this construct requires that the "Your Discriminator" field not change during a session. However, it does allow the "My Discriminator" field to change as permitted by RFC5880 Section 6.3 [RFC5880]

This construct provides for 64 bits of entropy, of which 32 bits is controlled by each party in a BFD session. For security, each implemention SHOULD randomize their discrimator fields at the start of a session, as discussed in RFC5880 Section 10 [RFC5880].

There is no way to signal or negotiate Seed changes. The receiving party MUST remember the current Seed value, and then detect if the Seed changes. Note that the Seed value MUST NOT change unless sending party has signalled a BFD state change with a a packet that is authenticated using a more secure Auth Type method.

10.1. Sender Variable Initialization

A system which sends packets initializes ISAAC as described above. The ISAAC related variables are initialized as follows:

bfd.MetKeyIsaacXmitKeyKnown:
This variable transitions from "false" to "true" when the sender decides to start using ISAAC. The sender also initializes the other variables at the same time.

bfd.MetKeyIsaacXmitAuthBase:
The sender copies the bfd.XmitAuthSeq number from the current packet to be sent into this variable.

bfd.MetKeyIsaacXmitAuthIndex:
The sender sets this variable to zero.

bfd.MetKeyIsaacXmitAuthSeed:
The sender copies the current Seed value into this variable. This variable is then copied into the "Seed" field of each Auth Type packet.

bfd.MetKeyIsaacXmitAuthData:
The ISAAC state for sending is encapsulated in this variable.

10.2. Receiver Variable Initialization

When a system receives packets with the Meticulous Keyed ISAAC authentication type and is able to authenticate such a packet the first time, the ISAAC related variables are initialized as follows:

bfd.MetKeyIsaacRcvKeyKnown:
This variable transitions from "false" to "true" when the receiver sees that the sender has started using Meticulous Keyed ISAAC authentication. The receiver also initializes the other variables at the same time.

bfd.MetKeyIsaacRcvAuthBase:
The sender the bfd.RcvAuthSeq number from the current packet to be sent into this variable.

bfd.MetKeyIsaacRcvAuthIndex:
The receiver sets this value to zero

bfd.MetKeyIsaacRcvAuthSeed:
The receiver copies the Seed value from the received packet into this variable. Note that this copy only occurs when the bfd.MetKeyIsaacXmitKeyKnown variable transitions from "false" to true"

bfd.MetKeyIsaacRcvAuthData:
The ISAAC state for receiving is encapsulated in this variable.

As there may be packet loss, the reciever has to take special care to initialize the bfd.MetKeyIsaacRcvAuthBase variable. If there has been no packet loss, the bfd.MetKeyIsaacRcvAuthBase is taken directly from the bfd.RcvAuthSeq variable, and the bfd.MetKeyIsaacRcvAuthIndex is set to zero

If, however, the packet's Sequence Number differs from the expected value, then the the difference "N" indicates how many packets were lost. The receive then has to search through the first "N" Auth Keys derived from its calculated ISAAC state in order to find one which matches. If no key matches the Auth Key in the packets, the packet is deemed to be inauthentic, and is discarded.

If a calculated key at index "I" does match the Auth Key in the packet, then the bfd.MetKeyIsaacRcvAuthIndex field is initialized to this value. The bfd.MetKeyIsaacRcvAuthBase field is then initialized to contain the value of bfd.RcvAuthSeq, minus the value of bfd.MetKeyIsaacRcvAuthIndex. This process allows the pseudo-random stream to be re-synchronized in the event of lost packets.

That is, the value for bfd.MetKeyIsaacRcvAuthBase is the Sequence Number for first Auth Key used in this session. This value may be from a lost packet, but can never the less be calculated by the receiver from a later packet.

This document does not make provisions for dealing with the case of losing more than 256 packets. Implementors should limit the value of "Detect Multi" to a small number in order to keep the number of lost packets within an acceptable limit.

11. Operation

Once the variables have been initialized, ISAAC will be able to produce 256 random numbers to use as Auth Keys, at near-zero cost. The "AuthIndex" field is incremented by one for every new Auth Key generated. Each new value of the Sequence Number field (sent or received) is then calculated by adding the relevant "AuthBase" and "AuthIndex" fields.

When all 256 numbers are consumed the "AuthIndex" field will wrap to zero. The ISAAC mixing function is then run, which then results in another set of 256 random numbers. The "AuthBase" variable is then incremented by 256, to indicate that 256 Auth Keys have been consumed. This process then continues until a BFD state change.

ISAAC can be thought of here as producing an infinite stream of numbers, based on a secret key, where the numbers are produced in "pages" of 256 32-bit values. This property of ISAAC allows for essentially zero-cost "seeking" within a page. The expensive operation of mixing is performed only once per 256 packets, which means that most BFD packet exchanges can be fast and efficient.

The receiving party can then look at the Sequence Number to determine which particular PRNG value is being used in the packet. By subtracting the bfd.MetKeyIsaacAuthBase from the Sequence Number (with possible wrapping), an expected "Index" can be derived, and a corresponding Auth Key found. This process thus permits the two parties to synchronize if/when a packet or packets are lost.

Incrementing the Sequence Number for every packet also prevents the re-use of any individual pseudo-random number which was derived from ISAAC.

The Sequence Number can increment without bounds, though it can wrap once it reaches the limit of the 32-bit counter field. ISAAC has a cycle length of 2^8287, so there is no issue with using more than 2^32 values from it.

The result of the above operation is an infinite series of numbers which are unguessable, and which can be used to authenticate the sending party.

Each system sending BFD packets chooses its own seed, and generates its own sequence of pseudo-random numbers using ISAAC, and place those values into the Auth Key field. Each system receiving BFD packets runs a separate pseudo-random number generator, and verifies that the received packets contain the expected Auth Key.

11.1. Page Flipping

Once all 256 Auth Keys from the current page have been used, the "next" page is calculated by calling the isaac() function. This function processes the current "page" to create the "next" page, and is inherently destructive. In order to prevent issues, care should be taken to perform this process correctly.

It is RECOMMENDED that implementations keep both a "current" page, and a "next" page associated with the ISAAC state. The "next" can be calculated by making a copy of the "current" page, and then calling the isaac() function.c. Both pages should be maintained at all times.

This process has a number of benefits. First, the "next" can be calculated asynchronously, and does not have to be done in the BFD "fast path". At 60 packets per second, the system has approximately four (4) seconds to calculate the "next" page.

Second, having the "next" page always available means that an attacker cannot spoof BFD packets, and force the received to spend significant resources calculating a "next" page on the BFD "fast path". Instead, the receiver can simply check the "next" page at near-zero cost, and discard the spoofed packet.

When the receiver determines that it needs to move to the "next" page, it can simply swap the "current" and "next" pages (updating the BFD variables as appropriate), and then notify an asynchronous system to calculate the "next" page. Such asynchronous calculations are preferable to calculating the next page in the BFD fast path.

12. Transition away from using ISAAC

There are two ways to transition away from using ISAAC. One way is via state changes: the link either goes down due to an fault, or one party signals a state change via a packet signed with a strong Auth Type. The second situation is where one party wishes to temporarily signal that it is still Up, using a strong Auth Type.

Since the Meticulous Keyed ISAAC authentication method does not provide for full packet integrity checks, it may be desirable for a party to periodically use a strong Auth Type. The switch to a different Auth Type can be done at any time during a session. The different Auth Type can signal that the session is still in the Up state.

It is RECOMMENDED that implementations periodically use a strong Auth Type for packets which maintain the session in an Up state. See BFD Authentication [I-D.ietf-bfd-optimizing-authentication] for appropriate procedures.

The nature of the Meticulous Keyed ISAAC method means that there is no issue with this switch, so long as it is for a small number of packets. From the point of view of the Meticulous Keyed ISAAC state machine, this switch can be handled similarly to a lost packet. The state machine simply notices that instead of Sequence Number value being one more than the last value used for ISAAC, it is larger by two. The ISAAC state machine then calculates the index into the current "page", and uses the found number to validate (or send) the Auth Key.

If the non-ISAAC Auth Type instead runs for extended periods of time, then the ISAAC process must continue "in the background" in order to maintain synchronization. This process is needed because this method does not provide for a way to reinitialize the ISAAC method with new Seed value.

13. IANA Considerations

For IANA Consideration, please refer to the IANA Considerations section of Optimizing BFD Authentication [I-D.ietf-bfd-optimizing-authentication].

Note to RFC Editor: this section may be removed on publication as an RFC.

14. Security Considerations

The security of this proposal depends strongly on the length of the Secret Key, and on its entropy. It is RECOMMENDED that the key be 16 octets in length or more.

The dependency on the Secret Key for security is mitigated through the use of two 32-bit random numbers, with one generated by each party to the BFD session. An attacker cannot simply perform an off-line brute-force dictionary attack to discover the key. Instead, any analysis has to include the particular 64 bits of entropy used for a particular session. As a result, dictionary attacks are more difficult than they would be if the PRNG generator depended on nothing more than the Secret Key.

The security of this proposal depends strongly on ISAAC. This generator has been analyzed for almost three decades, and has not been broken. Research shows that there are few other CSRNGs which are as simple and as fast as ISAAC. For example, many other generators are based on AES, which is infeasibe for resource constrained systems.

In a keyed algorithm, the key is shared between the two systems. Distribution of this key to all the systems at the same time can be quite a cumbersome task. BFD sessions running a fast rate may require these keys to be refreshed often, which poses a further challenge. Therefore, it is difficult to change the keys during the operation of a BFD session without affecting the stability of the BFD session. Therefore, it is recommended to administratively disable the BFD session before changing the keys.

That is, while the Auth Key ID field provides for the use of multiple keys simultaneously, there is no way for each party to signal which Key IDs are supported.

The Auth Type method defined here allows the BFD end-points to detect a malicious packet, as the calculated hash value will not match the value found in the packet. The behavior of the session, when such a packet is detected, is based on the implementation. A flood of such malicious packets may cause a BFD session to be operationally down.

14.1. Spoofing

When Meticulous Keyed ISAAC is used, it is possible for an attacker who can see the packets to observe a particular Auth Key value, and then copy it to a different packet as a "man-in-the-middle" attack. However, the usefulness of such an attack is limited by the requirements that these packets must not signal state changes in the BFD session, and that the Auth Key changes on every packet.

Performing such an attack would require an attacker to have the following information and capabilities:

This is man-in-the-middle active attack.

The attacker has the contents of a stable packet

The attacker has managed to deduce the ISAAC key and knows which per-packet key is being used.

The attack is therefore limited to keeping the BFD session up when it would otherwise drop.

However, the usual actual attack which we are protecting BFD from is availability. That is, the attacker is trying to shut down then connection when the attacked parties are trying to keep it up. As a result, the attacks here seem to be irrelevant in practice.

14.2. Re-Use of keys

The strength of the Auth-Type methods is significantly different between the strong one like SHA-1 and ISAAC. While ISAAC has had cryptanalysis, and has not been shown to be broken, that analysis is limited. The question then is whether or not it is safe to use the same key for both Auth Type methods (SHA1 and ISAAC), or should we require different keys for each method?

If we recommend different keys, then it is possible for the two keys to be configured differently on each side of a BFD link. For example, a correctly configured key could allow to the BFD state machine to advance to Up. Then when the session switches to using to weaker Auth Type with a different key, that key may not match, and the session would immediatly drop. Requiring instead that the keys be identical means that no such misconfiguration is possible.

We believe that the use of the same key is acceptable, as the Auth Type defined for ISAAC depends on 64 bits of random data. The use of this randomness increases the difficulty of breaking the key, and makes off-line dictionary attacks infeasible.

15. Acknowledgements

The authors would like to thank Jeff Haas and Reshad Rahman for their reviews of and suggestions for the document.

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5880] Katz, D. and D. Ward, "Bidirectional Forwarding Detection (BFD)", RFC 5880, DOI 10.17487/RFC5880, June 2010, <https://www.rfc-editor.org/info/rfc5880>.

16.2. Informative References

[I-D.ietf-bfd-optimizing-authentication] Jethanandani, M., Mishra, A., Saxena, A., and M. Bhatia, "Optimizing BFD Authentication", Work in Progress, Internet-Draft, draft-ietf-bfd-optimizing-authentication-13, 1 August 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-bfd-optimizing-authentication-13>.

[ISAAC] Jenkins, R. J., "ISAAC", http://www.burtleburtle.net/bob/rand/isaac.html, 1996.

Authors' Addresses

Alan DeKok
Network RADIUS SARL
100 Centrepointe Drive #200
Ottawa ON K2G 6B1
Canada
Email: aland@freeradius.org

Mahesh Jethanandani
Kloud Services
Email: mjethanandani@gmail.com

Sonal Agarwal
Cisco Systems, Inc
170 W. Tasman Drive
San Jose, CA 95070
United States of America
Email: agarwaso@cisco.com
URI: www.cisco.com

Ashesh Mishra
Aalyria Technologies
Email: ashesh@aalyria.com

Ankur Saxena
Ciena Corporation
3939 North First Street
San Jose, CA 95134
United States of America
Email: ankurpsaxena@gmail.com