Authorisation and Notification of Changes in the RIPE Database

Daniel Karrenberg
RIPE NCC

Marten Terpstra
RIPE NCC

Document ID: ripe-120
Obsoletes: ripe-096

ABSTRACT

Two new attributes are defined for all objects in the RIPE database in order to implement a generalised method for authorising changes and to notify interested parties of any changes made to a specific database object. Further the maintainer object is defined to represent entities maintaining database objects.

History

Updates to the RIPE database [ripe-050] are currently done almost exclusively by electronic mail which is processed automatically. The number of updates processed is substantial [ripe-106]. Historically there was no authorisation and anyone could change any object in the database. From the start extensive audit trails of changes have been kept in order to identify problems with automatic processing or to trace unintended changes. For more than 4 years of experience with this model we have not encountered any instance of a malicious change to an object until very recently. The amount of accidental unwanted changes is also surprisingly low.

In order to help detect unwanted changes the **notify** attribute was introduced [ripe-096]. It will be described below.

The RIPE community feels a need to introduce authorisation and authentication mechanisms for updates to the RIPE database. The specific need arose when the database started to be used as a routing registry [ripe-081].

As a simple straightforward measure *guarded objects* were introduced. All **aut-num** and **community** objects were guarded. Any update to such an object had to be manually authorised by RIPE NCC staff. While this was easy to implement it obviously does not scale well.

The data representation schema used for the routing registry also combined allocation and routing registry information in a single object, the **inetnum** object. Routing registry information is not necessarily maintained and controlled by those controlling the corresponding allocation information. In order to solve this the special mechanism of guarded attributes was introduced [ripe-108]. This has turned out to be unwieldy since it uses a special mechanism for updates to some attributes of an object and is not well integrated into the database update model.

For this and other reasons the representation of routing information in the RIPE database has been changed to clearly separate routing from allocation information by storing them in different objects. Consequently the authorisation of changes can again be done at the object level.

The Notify Attribute

Each RIPE database object has an optional attribute called **notify.** The value of the **notify** attribute is one valid RFC822 e-mail address. There can be multiple **notify** attributes. Whenever the object concerned is changed in the database a notification message will be sent to each e-mail addresses appearing in a **notify** attribute.

This makes it straightforward to keep track of changes to specific objects and prevent changes from going unnoticed. Multiple **notify** attributes make it possible to notify a number of interested parties. This could be used to alert all contact persons for an object or the local contact persons as well as the relevant service provider. Although it may be tempting to put many **notify** attributes on database objects in order to notify everyone even remotely interested, this is not recommended. A very few key addresses should be sufficient. Prior to entering any mail address here, the explicit or implicit consent of the person responsible for that particular mailbox needs to be obtained.

Obviously the **notify** attributes used for notification are those stored in the database *before* the update. This also guarantees proper notification about deletes.

Note that there is another, often more easily maintainable, way to effect notification of changes to the maintainer of an object. See the section about the **mntner** object for details.

Authorisation Model

The new model for authorisation of changes to the database is fully integrated into the database model and applies to any object.

Optionally each database object can be associated with one or more maintainers who are authorised to make changes. Only those maintainers and the RIPE NCC are then authorised to change or delete the object.

Each maintainer is also represented in the database by its own **mntner** object which stores information such as contact persons, authorisation and notification details.

Whenever a change to an object is attempted the **mnt-by** attribute of the current database object is examined.

If there is no **mnt-by** attribute, the update always proceeds causing any notifications specified in **notify** attributes of the object. This ensures backward compatibility. It is also a perfectly legitimate authorisation model for those objects that do not need sophisticated authorisation. Also we would like to stress that using stronger authorisation requires timely processing of update requests. An unresponsive maintainer preventing others from making updates frequently is a worse solution than no authorisation.

If the update is originated by a maintainer referenced in a **mnt-by** attribute, the update also proceeds causing the necessary notifications. There are various methods to authenticate the origin of an update request. These can be configured in the **mntner** object described below.

If a new object with a **mnt-by** attribute is added to the database or a **mnt-by** attribute is added to an object that previously had no such attribute, the authorisation step is performed on the maintainer *to be added*.

If authentication fails the update request is forwarded to the mailbox listed in the **upd-to** attribute of the maintainer(s) of the object for processing and the originator of the request is notified. No other notifications are performed in this case.

If an update is not authorised and no appropriate maintainer can be identified the request will be forwarded to the RIPE NCC for action.

See the separate section below for details on authentication methods and related matters.

The Maintained-By Attribute

Each RIPE database object has an optional attribute called **mnt-by** (maintained by). The value of the **mnt-by** attribute is a reference to a **mntner** object in the database which describes those authorised to make changes to the object. See below for details.

Multiple **mntner** objects can be referenced by separating them with blanks, which is preferred, or by using multiple **mnt-by** attributes per object.

In this case all maintainers referenced are equally authorised to make changes to the object. For instance this is applicable to person objects maintained by both a toplevel domain registry and a local address space registry. Because close coordination is required if an object is to be maintained by multiple maintainers, this is expected to be the exception rather than the rule.

When responding to queries, the RIPE whois server will **not** automatically return the associated **mntner** object with any matching object as is done with contact persons.

The Maintainer Object

The **mntner** object represents an entity maintaining objects in the RIPE database. The maintainer is identified and referred to by a unique maintainer name. The **mntner** object is used every time a database object with a **mnt-by** attribute is added, updated or deleted to determine whether the originator of the update request is authorised to make the update.

In addition the **mntner** object provides the same kind of notification ability that is also available with the **notify** attribute. When using the **notify** attribute, the user must specify who to notify in every object that he is responsible for. If he wants to change who is notified, every object must be changed. By putting this notification information in the **mntner** object, that information is centralized and can be managed more easily.

Adding a new **mntner** object has to be authorised manually by RIPE NCC staff. Updates to **mntner** objects follow the normal authorisation rules but receive special scrutiny by NCC staff too.

```
mntner:          [mandatory]          [single]
descr:           [mandatory]          [multiple]
admin-c:         [mandatory]          [multiple]
tech-c:          [optional]           [multiple]
upd-to:          [mandatory]          [multiple]
mnt-nfy:         [optional]           [multiple]
auth:            [mandatory]          [multiple]
remarks:         [optional]           [multiple]
notify:          [optional]           [multiple]
mnt-by:          [optional]           [multiple]
changed:         [mandatory]          [multiple]
source:          [mandatory]          [single]
```

Each attribute has the following syntax:

mntner:

Maintainer name.

Format:

An upper case text string consisting of alphanumeric characters and "-" (dash) which is not the same as any maintainer name already defined. The name has to be unique with regard to other maintainer names only. It can be the same as handles, autonomous system or community names.

Examples:

```
mntner: FOO-NOC

mntner: NN-DOMREG
```

Status: mandatory, only one line allowed

descr:

A short description of the maintainer entity.

Format:

```
free text
```

Examples:

```
descr: FOO Networking Inc. NOC
descr: Serving all customers of FOO Networking Inc.

descr: Domain Registrar for the NN toplevel domain.
```

Status: mandatory, multiple lines allowed

admin-c:

Full name or uniquely assigned NIC-handle of an administrative contact person. This is the one with whom coordination should be done.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Example:

```
admin-c: Joe T Bloggs
admin-c: JTB1
```

The handle form is preferred because it is not ambiguous.

Status: mandatory, multiple lines allowed

tech-c:

Full name or uniquely assigned NIC-handle of a technical contact person. If defined this is someone to be contacted for technical problems such as bounced e-mail etc.

Format:

`<firstname> <initials> <lastname>` or `<nic-handle>`

Examples:

```
tech-c: John E Doe
tech-c: JED31
```

The handle form is preferred because it is not ambiguous.

Status: optional, multiple lines allowed

upd-to:

Updates to. Any unauthorised update request of an object maintained by this maintainer will be forwarded to this address.

Format:
RFC-822 address

Example:

```
upd-to: noc@foobar.net

upd-to: domreg@nn-nic.nn
```

Status: mandatory, multiple lines allowed

mnt-nfy:

Maintainer notification. This e-mail address will receive notification messages if any object maintained by this maintainer is added, changed or deleted. The functionality is exactly the same as if a **notify** attribute had been defined in the object. Specifying it here has the advantage that any changes of the address(es) affect only one object. For more information see the section about the **notify** attribute.

Format:
RFC-822 address

Example:

```
mnt-nfy: noc@foobar.net

mnt-nfy: domreg@nn-nic.nn
```

Status: optional, multiple lines allowed

auth:

specifies which scheme will be used identify and authenticate update requests from this maintainer.

Format:

```
<scheme-id> <auth-info>
```

The scheme-ids currently defined are: NONE, MAIL-FROM and CRYPT-PW. The auth-info is additional information required by a particular scheme. When more than **auth** attribute is specified any of them can be used alternatively for authentication of updates, i.e. specifying NONE and others does not make much sense. The auth attribute is not protected information in any way; it is returned normally like any attribute by the whois server and available in stored copies of the database. The strength of an authentication scheme thus has to lie in the scheme itself and cannot be based on the obscurity of the **auth** attribute. See the section about authentication schemes for more details.

Example:

```
auth: NONE

auth: CRYPT-PW dhjsdfhruewf

auth: MAIL-FROM .*@ripe.net
```

Status: mandatory, multiple lines allowed

remarks:

Remarks/comments, to be used only for clarification.

Format:

```
free text
```

Example:

```
remarks: This is a test/example object.
```

Status: optional, multiple lines allowed

notify:

The notify attribute contains an email address to which notifications of changes to this object should be send. See also the section "The Notify Attribute" near the end of this document.

Format:

```
<email-address>
```

The `<email-address>` should be in RFC822 domain syntax wherever possible.

Example:

```
notify: Marten.Terpstra@ripe.net
```

Status: optional, multiple lines allowed

mnt-by:

This attribute specifies who maintains this object in the RIPE database. See also the section "The Maintained-By Attribute".

Format:

```
<maintainer name>
```

Example:

```
maintainer: FOO-NOC
```

Status: optional, multiple lines allowed

changed:

Who changed this object last, and when was this change made.

Format:

```
<email-address> YYMMDD
```

`<email-address>` should be the address of the person who made the last change. `YYMMDD` denotes the date this change was made.

Example:

```
changed: johndoe@terabit-labs.nn 900401
```

Status: mandatory, multiple lines allowed

source:

Source of the information.

This is used to separate information from different sources kept by the same database software. For RIPE database entries the value is fixed to `RIPE`.

Format:

```
RIPE
```

Status: mandatory, only one line allowed

Authentication Schemes

The authorisation model supports multiple authentication schemes. Currently only relatively weak authentication is defined. It is entirely possible to use stronger authentication schemes based privacy enhanced mail (PEM, PGP). It is expected that such schemes will be defined as the need arises.

Please note again that the authentication scheme and the additional `<auth-info>` is public information available in the database. The strength of an authentication scheme must be inherent in that scheme and not be based on keeping this information secret. The reason for this is that it is very difficult to keep the information confidential and thus the RIPE NCC cannot take this responsibility.

NONE

This is the simplest authentication scheme which is entirely backwards compatible with the one currently used. No authentication is provided, i.e. it is assumed that all update requests originate from an authorised maintainer or are at least coordinated with one. Anyone in doubt whether it is OK to issue update requests should check with the maintainer concerned first, preferably at the mailbox specified in the **upd-to** attribute. When making any changes the **mnt-by** attribute should not be changed without explicit consent from the current maintainer.

This scheme is for those who want to give everyone the possibility to make changes while at the same time using the **mnt-by** attribute for its notification and documentation features. A good reason to use this scheme is that the maintainer cannot guarantee timely processing of updates himself.

MAIL-FROM

This authentication method checks the content of the RFC822 From: header of an update request against the regular expression specified as `<auth-info>`. If the regular expression matches the content of the From: header the update request is authenticated successfully. The regular expressions supported are described in

POSIX 1003.2 section 2.8. As it is expected that most regular expressions will either be literals or of a form similar to `.*@some\.domain\.or\.other` an extensive description of the possibilities will not be given. Note that the matching is applied to the whole content of the From: header including comments if present. No attempt is made to isolate the mailbox part.

It should be stressed that this authentication scheme is very weak. Forging RFC822 headers does not take much effort or ingenuity. The reason for the scheme's existence is that it easily prevents accidental updates rather than allowing them first and fixing them later when notified.

This scheme is for those who want to prevent accidental updates by others and are able to process update requests in a timely manner.

CRYPT-PW

This scheme uses the Unix crypt(3) routine, which is also used for login passwords under Unix. This routine provides a so called "trap door" function, the inverse of which is somewhat hard to calculate. The password provided by the user is encrypted with this function and stored in its encrypted form only. When the user later provides the password again for authentication, the encryption is repeated and the results are compared. Since the original (cleartext) password cannot easily be computed from the encrypted version the encrypted password does not have to be kept secret.

The `<auth-info>` is the encrypted password. This can either be obtained locally with the appropriate Unix tools or on e-mail request from `<ripe-dbm@ripe.net>`. When sending in update requests the cleartext password has to be provided in the message body by specifying

password: cleartext-password

at the beginning of a line and preceding any update requests to be thus authenticated. The password will remain valid for all requests following it in the same e-mail message or until another password is specified.

This scheme is slightly stronger than the MAIL-FROM scheme. It is by no means meant to keep out a determined malicious attacker. The crypt function is vulnerable to exhaustive search by (lots of) fast machines and programs to do the searching are widely available. For this reason it is *strongly discouraged* to use encrypted passwords also used for other purposes such as Unix login accounts in this scheme. As you are publishing the encrypted password in the database it is open to attack. The usual caveats about crypt passwords apply, so is not very wise to use words or combinations of words found in any dictionary of any language. See [R. Morris, K. Thompson: Password Security: A Case History] for more details about the scheme and its vulnerabilities.

Multiple authentication methods can be specified in the same **mntner** object. These will be used alternatively, i.e. any one of the authenticators is sufficient to authenticate the update request from the maintainer. If multiple maintainers maintain an object this feature should *not* be used. Multiple maintainers should be represented by multiple **mntner** objects referenced in the **mnt-by** attribute. There is no way in the current model to require a combination of multiple authenticators to authenticate a request.

Special Rules in the Routing Registry

Because routes are originated by autonomous systems the autonomous system concerned should be the only one maintaining **route** objects. The maintainer of a route object is thus expected to be the same as the one of the **aut-num** object referenced in its **origin** attribute. We consciously decided not to enforce this rule until experience shows this to be necessary. We deem the added flexibility gained by not doing so to be useful in many cases. If necessary the values of the **mnt-by** attributes of the new **route** object and the referenced **mntner** objects should be required to be equal at creation time of the **route.**

In order to support the necessary routing coordination, special notification rules apply to the **route** object: Whenever a **route** object is created or deleted or the **comm-list** attribute changes, the guardians of all communities and ASes referenced by the old and new objects will be notified in addition to the normal notifications. This rule ensures that community guardians have retroactive control over community membership and that ASes get notified if someone else adds a route originated by them.

Note that this rule changes the level of membership control exercised by community and AS guardians have with respect to the "guardian file" method. Control is now by notification after the fact.

The second special notification rule concerns creation and deletion of other **route** objects for the same route but with different originators: Whenever a **route** object is created or deleted, the registry is searched for other **route** objects covering exactly the same address space as well as the smallest less specific routes. The guardians of all such **route** objects will be notified of the change as well as the existence of all **route** objects concerned. This also includes notification of the guardians of the **route** object just created. This rule tries to ensure that multiple insertions of the same route into the routing mesh as well as proxy aggregation are coordinated at least post factum. Due to the technical effort involved, implementation of this rule may be effected somewhat later than the rest of the authorisation package. Whether this notification should also include more specific and additional less specific routes is for experience to determine.

Examples

Use of the authorisation and notification scheme described in this document is totally optional. So the current object below remains fully valid:

```
inetnum:   193.0.0.0
netname:   RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Amsterdam, Netherlands
country:   NL
admin-c:   Daniel Karrenberg
tech-c:    Marten Terpstra
tech-c:    Tony Bates
rev-srv:   ns.ripe.net
rev-srv:   ns.eu.net
notify:    ops@ripe.net
changed:   tony@ripe.net 940708
source:    RIPE
```

But even if notification is the only feature desired, adding a maintainer object can be useful. It documents who the maintainer is and it puts the e-mail addresses for notification in one place only:

```
inetnum:   193.0.0.0
netname:   RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Amsterdam, Netherlands
country:   NL
admin-c:   Daniel Karrenberg
tech-c:    Marten Terpstra
tech-c:    Tony Bates
rev-srv:   ns.ripe.net
rev-srv:   ns.eu.net
mnt-by:    RIPE-NCC
changed:   tony@ripe.net 940708
source:    RIPE

mntner:    RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Maintains all objects for NCC resources.
admin-c:   DK58
tech-c:    MT2
tech-c:    TB230
upd-to:    ops@ripe.net
mnt-nfy:   ops@ripe.net
auth:      NONE
mnt-by:    RIPE-NCC
changed:   dfk@ripe.net 940910
source:    RIPE
```

Note that the **mntner** object itself is maintained by `RIPE-NCC` too, so notification will also happen if the object itself is changed.

Changing the addresses can then be done by changing just the **mntner** object and not all objects referring to the address. It is also easy to switch on authentication for all objects at once if needed:

```
mntner:    RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Maintains all objects for NCC resources.
admin-c:   DK58
tech-c:    MT2
tech-c:    TB230
upd-to:    ops@ripe.net
mnt-nfy:   notifications@ripe.net
auth:      MAIL-FROM .*@.ripe.net
notify:    mntner-change@ripe.net
mnt-by:    RIPE-NCC
changed:   dfk@ripe.net 940910
source:    RIPE
```

Note that `mntner-change@ripe.net` will be notified in addition to `notifications@ripe.net` if the **mntner** object itself changes.

If stronger authorisation is needed it can be switched on with a single update to the **mntner** object again.

```
mntner:    RIPE-NCC
descr:     RIPE Network Coordination Centre
descr:     Maintains all objects for NCC resources.
admin-c:   DK58
tech-c:    MT2
tech-c:    TB230
upd-to:    ops@ripe.net
mnt-nfy:   notifications@ripe.net
mnt-nfy:   daniel.karrenberg@ripe.net
auth:      CRYPT-PW 949WK1mIRby6c
notify:    mntner-change@ripe.net
mnt-by:    RIPE-NCC
changed:   dfk@ripe.net 940910
source:    RIPE
```

Note that any update of this object must now be preceded by a line of the form

```
password: NCC-PASS
```

to be properly authenticated.

Specifying alternative authentication methods is allowed. So if f.i. either of two passwords should be used to authenticate update requests this can be represented like this:

```
mntner:     RIPE-NCC
descr:      RIPE Network Coordination Centre
descr:      Maintains all objects for NCC resources.
admin-c:    DK58
tech-c:     MT2
tech-c:     TB230
upd-to:     ops@ripe.net
mnt-nfy:    notifications@ripe.net
mnt-nfy:    daniel.karrenberg@ripe.net
auth:       CRYPT-PW 949WK1mIRby6c
auth:       CRYPT-PW 95sF/QAyIMtgg
notify:     mntner-change@ripe.net
mnt-by:     RIPE-NCC
changed:    dfk@ripe.net 940910
source:     RIPE
```

If on the other hand one object is maintained by multiple maintainers this should be expressed by using multiple **mntner** objects. These will be equivalent in authentication checking. It speaks for itself that good coordination between the multiple maintainers of an object is an absolute necessity:

```
person:     Lena F. Karrenberg
address:    c/o RIPE NCC
address:    Kruislaan 409
address:    NL-1098 SJ Amsterdam
phone:      +31 20 5925065
e-mail:     lena.karrenberg@ripe.net
remarks:    Born August 12th 1994
remarks:    Assistant to the NCC manager for reality checks.
changed:    karrenberg@ripe.net 940812
mnt-by:     DANIEL BEATE
source:     RIPE
```

References

All references of the form "ripe-nnn" refer to RIPE documents obtainable from `ftp://ftp.ripe.net/ripe/docs`.

Acknowledgments

The authors acknowledge the very useful input from the RIPE database working group as well as the following individuals: Dale Johnson of MERIT.