IP-Based Actions

IP-based actions are similar to BIND 9's RPZ-IP triggers and corresponding actions. They are similar to the existing local-zone or other tag-based actions, but defined for particular IP-netblocks. If the IP(v6/v4) address of an AAAA or A in the answer section matches one of the specified netblocks, the corresponding action will apply to the response. For example, it may change the address to a different one (redirect) or make the entire resolution result in NXDOMAIN.

This note describes details of proposed implementation of IP-based actions in Unbound.

Configuration Syntax

IP-based actions require a new (optional) Unbound module placed above iterator (see the implementation note below). So the `module-config` option must be specified explicitly:

```
module-config: "respip iterator"
```

If DNSSEC-validation is supposed to be enabled, it should be:

```
module-config: "respip validator iterator"
```

We also introduce three new configuration options: `response-ip`, `response-ip-data`, `response-ip-tag`. These options are part of the **`server:`** clause.

`response-ip:` `<IP-netblock> <action>`

If the IP address in an AAAA or A RR in the answer section of a response matches the specified IP-netblock, the specified action will apply. <action> has generally the same semantics as that for `access-control-tag-action` , but there are some exceptions (see below).

`response-ip-data:` `<IP-netblock> <"resource record string">`

This specifies the action data for `response-ip` with action being `redirect`. "resource record string" is similar to that of `access-control-tag-action` , but it must be of either AAAA, A, or CNAME. If the IP-netblock is an IPv6/IPv4 prefix, the record must be AAAA/A, respectively unless it's CNAME (which can be used for both versions of IP netblocks). If it's CNAME, there must not be more than one `response-ip-data` for the same IP-netblock. Also, CNAME and other types of records must not coexist for the same IP-netblock. The textual domain name for the CNAME does not have to be explicitly terminated with a dot (.); the root name is assumed to be the origin for the name.

`response-ip-tag:` `<IP-netblock> <"list of tags">`

Assign tags to response IP-netblocks. If the IP address in an AAAA or A RR in the answer section of a response matches the specified IP-netblock, the specified tags are assigned to the IP address. Then, if an `access-control-tag` is defined for the client and it includes one of the tags for the response IP, the corresponding `access-control-tag-action` will apply. Tag matching rule is the same as that for `access-control-tag` and `local-zones`. Unlike l`ocal-zone-tag`, `response-ip-tag` can be defined for an IP-netblock even if no `response-ip` is defined for that netblock.

`response-ip` and `response-ip-data` can also be specified for a view, just like `local-zone` and `local-data`.

If multiple `response-ip-tag` options are specified for the same IP-netblock (in different lines), all but the one that appears first will be ignored (following the similar case for `access-control-tag`). Even if it's not rejected as a configuration error, it's quite unlikely that the result is the intended behavior and would be meaningless in practice.

The `response-ip-xxx` options require the `respip` module; if any of these options are specified but the `respip` module isn't enabled, unbound will refuse to (re)load the configuration. In fact, some part of these options (i.e., modifying cached answers) would work without the `respip` module, but such an incomplete behavior would be rather confusing or even harmful. It should be better to explicitly reject such configurations as invalid.

Exceptions on response-ip Actions

Actions specified for `response-ip` are different from those for local-zone in that in case of the former there is no point of such condition as "the query matches it but there is no local data". Because of this difference, the semantics of `response-ip` actions are modified or simplified as follows:

- `static`, `refuse`, `transparent`, `typetransparent`, and `nodefault` are invalid for `response-ip`. If any of these is specified the configuration will be rejected.
- `deny` is non-conditional, i.e., it always results in dropping the corresponding query. The resolution result before applying the deny action is still cached and can be used for other queries.

On the other hand, actions specified in an `access-control-tag-action` that has a matching tag with `response-ip-tag` can be those that are "invalid" for response-ip listed above, since `access-control-tag-action` s can be shared with local zones. For these actions, if they behave differently depending on whether local data exists or not in case of local zones, the behavior for response-ip-tags is largely compatible with that. However, the case of no corresponding `response-ip-data` will generally result in NOERROR/NODATA instead of NXDOMAIN, since the response-ip data are inherently type specific, and non-existence of data doesn't indicate anything about the existence or non-existence of the qname itself. For example, if the matching tag action is static but there is no data for the corresponding response-ip

configuration, then the result will be NOERROR/NODATA. The only case where NXDOMAIN is returned is when an `always_nxdomain` action applies.

Notes on Detailed Behavior

No AA Bit

If the original answer is tweaked due to a response-ip action, the final response sent to the client will never have the AA (authoritative answer) bit on in the header section. This is different from some cases of local-zone and access-control-tag actions (faked NXDOMAIN and local/redirect data), where the final response is considered “authoritative” and has the AA bit on. This behavior is consistent with BIND 9 RPZ (and in that case it’s not specific to RPZ-IP triggers).

Answer Records Only

IP-based actions apply only to records in the answer section. Even if the IP address of an AAAA or A record in the additional (or, unlikely in practice, authority) section matches an IP-netblock of some `response-ip` options, these records (or the response itself) won’t be modified.

Multi-RR Case

If an AAAA or A RRset in the answer section of a query contains multiple RRs and one or more of the addresses are subject to response-ip processing, the action for the first matching address in the answer message will be exclusively used. For example, if an answer AAAA RRset contains the following IPv6 address in this order:

```
2001:db8::1
2001:db8::2
2001:db8::3
```

and the following IP-based actions are configured:

```
response-ip: 2001:db8::1/128 always_nxdomain
response-ip: 2001:db8::3/128 redirect
response-ip-data: 2001:db8::3/128 AAAA 2001:db8::bad
```

then the first matching action (always_nxdomain) will apply and the response will be converted to NXDOMAIN (with no answer records). Likewise, if there are multiple matching redirect actions for different IP-netblocks of the same answer RRset, only the first matching redirect data will be used, and the resulting response will contain only one address (which is the matching redirect data). Since it appears that Unbound caches an RRset in the order it receives from the upstream server, the same action will be applied to subsequent responses to the same query when they are directly answered from the cache.

This behavior largely follows the behavior of BIND 9’s RPZ-IP triggers. Note, however, that the order of the answer RRs from remote servers is not always predictable, so it is also

unpredictable which `response-ip` action is used when there are multiple candidates. This is different from BIND 9 RPZ-IP, where the triggers are applied to sorted RRs by default and the matching rule is generally predictable for the same RRset.

CNAME Chasing

If a CNAME is specified as the action data for an IP-based action, Unbound will automatically chase the CNAME target until it gets the final answer (whether positive or negative, or some other error), and return the complete CNAME chain to the end client. For example, for the above original AAAA answer RRset, if we define a redirect response-ip action as follows:

```
response-ip: 2001:db8::1/128 redirect
response-ip-data: 2001:db8::1/128 CNAME target.example.
```

then Unbound will resolve AAAA for target.example. (or retrieve it from the cache), and include the result in the final answer after the specified CNAME. So an example final answer is:

```
<original qname> CNAME target.example.
target.example. AAAA 2001:db8::ffff
```

Unlike “CNAME-based redirect” for local-zone actions, this CNAME chasing will take place even if the redirect data is used for an access-control-tag-action that is not redirect but can use local data (e.g. static). For example, in addition to the above configuration, if there’s a following tag setup:

```
access-control-tag-action: 192.0.2.1/32 "mytag" static
response-ip-tag: 2001:db8::1/128 "mytag"
```

then, if the client at 192.0.2.1 sends the same query, this static tag will refer to the redirect data for the corresponding response-ip-data, and its CNAME target will be chased (in retrospect, this would have been more intuitive for the local zone case, too).

Even if the CNAME target RRset has a valid RRSIG, the RRSIG won’t be included in the final response to the client, nor the AD bit will be set in the response, regardless of whether or not the original query sets the DO or CD bit (see also “DNSSEC implications” below). This behavior is consistent with BIND 9.

Type-ANY query suppresses this chasing. For instance, in the above example configuration if the query type is ANY, the answer will only contain the CNAME RR. This behavior is consistent with BIND 9.

Note: Chasing CNAME targets for an IP-based action may be especially expensive in terms of performance (see implementation notes below). It’s probably advisable to avoid this configuration whenever possible.

No Recursive Application

This proposed implementation does not try to apply IP-based actions “recursively”; that is, it does not apply an IP-based action to the IP address specified as a result of data of a redirect

response-ip action. For example, assume we apply a redirect action for `2001:db8::3/128` used in the previous example and rewrite it to `2001:db8::bad`. Then, even if there is another IP-based action which `2001:db8::bad` matches, that action won't apply. (This is the same as BIND 9's RPZ-IP.)

The same restriction applies to the target address of a CNAME when the CNAME is the redirect data of an IP-based action. For example, in the example of the CNAME chain shown above:

```
<original qname> CNAME target.example.
target.example. AAAA 2001:db8::ffff
```

even if there is another IP-based action which `2001:db8::ffff` matches, that action won't apply. In this case, only the CNAME will be returned to the client in order to avoid including an IP address in the answer (in this example, `2001:db8::ffff`) that would otherwise be subject to an IP-based action. It will also help avoid having a weird corner cases like a CNAME loop (e.g., consider the case where `2001:db8::ffff` is redirect to `<original qname>` again). Note that a sophisticated client (such as a local caching server using this Unbound as a "forwarder") could re-query for the CNAME target when it gets the incomplete CNAME chain. In this case the intended IP-based action will apply and that client will get an answer that the administrator of this action would probably envision. For traditional stub resolvers such an incomplete CNAME chain effectively means resolution failure. This is not ideal, but it would be acceptable in practice as in most cases the primary intent of such an action would be to avoid returning a specific IP address.

In any case, such a situation is considered a kind of configuration error, and Unbound leaves an informational level of log message when it detects the situation:

```
CNAME target of redirect response-ip action would be subject to
response-ip action, too; stripped
```

This is also consistent with BIND 9 RPZ, which doesn't apply RPZ rules to CNAME targets if the CNAME comes from an RPZ.

Similar to the above cases, even if there is a local-zone action to which `target.example` would be subject, it won't apply. In this case the above AAAA RR will be included in the answer to the client (it's no different from how a redirected CNAME target in local-zone or tag-based actions works in general).

This is different from BIND 9 RPZ, which never applies RPZ rules more than once. In this implementation we defer from it to keep the implementation simpler, but we may want to revisit it (this is also different from what's written in draft-vixie-dns-rpz).

No Override for Other Local Data

Similar to the previous subsection, IP-based actions will not apply to the result of other local-data or tag-based actions. For example, if we have the following configuration:

```
local-zone: example.com. redirect
local-data: "example.com. IN A 192.0.2.1"
response-ip: 192.0.2.0/24 always_nxdomain
```

then a query for example.com/A will be answered from the local-zone with the A RDATA of 192.0.2.1. Even if it would match the IP-netblock of `response-ip`, its action won't apply to this answer. This is consistent with BIND 9 RPZ.

Authority and Additional Sections

If an AAAA or A RR of the answer section of a response is modified due to an IP-based action, the authority and additional sections of the resulting final response to the client will be cleared (an EDNS OPT RR may still be added to the additional section). This is the case regardless of the action type, even if it's `redirect` or `nxdomain` variants. This is different from BIND 9 RPZ: it adds an SOA of the corresponding RPZ to the authority section for positive answers and to the additional section for NXDOMAIN.

DNSSEC Implications

If an original response is modified due to an IP-based action, the resulting final response will never have the AD bit on even if the original response was DNSSEC-validated. Any RRSIG RRsets for the modified RRset will be removed from the answer section; however, if the original response is a CNAME chain and some of the CNAMEs have RRSIGs, these RRSIG will be kept in the final response (note that CNAMEs can never be subject to an IP-based action). This behavior is compatible with BIND 9 RPZ.

Multi-Level Netblock Matching

We will (eventually) try to implement multi-level matching: if the best (longest) matching IP-prefix does not have a matching tag for a client but a less-specific matching IP-netblock has a matching tag, the action for the less-specific IP-netblock should apply. For example, if we have the following two response-ip-tag configurations:

```
response-ip-tag: 192.0.2.128/28 "tag1 tag2"
response-ip-tag: 192.0.2.255/32 "tag3"
```

and if an A RR in the answer section has address `192.0.2.255` but tags for the client include "tag1" but not "tag3", then the action for `192.0.2.128/28` should apply even if the best matching netblock is `192.0.2.255/32`.

This ideal behavior is consistent with how tags for local-zones match. But this may need non-trivial extensions to existing Unbound utilities, while such a multi-level setup is supposed to be quite rare. So we may skip this behavior and always consider the best matching netblock (if there is no matching tag, treat it as there is no matching netblock) in the initial implementation.

This is a **TODO** item as of this writing (as of February 10, 2017, this “ideal behavior” is not implemented).

Type ANY Query

The primarily intended usage of IP-based actions is to apply them to type AAAA or A queries. But it should also work for type ANY queries if the answer contains an AAAA or A record. This is consistent with BIND 9 RPZ. In this case, all RRs of the original answer will be removed except the one that triggered the IP-based action, for which it may be replaced (in case of a `redirect` action) or it may also be removed (in all other types of actions including `nxdomain` variants). This behavior is also consistent with BIND 9 RPZ.

Inform Log

If an `inform` or `inform_deny` IP-based action applies, a message will be logged with information about the client that triggers the action. This is similar to log messages for `inform` and `inform_deny` actions of local zones and their tag actions, but contains the matched netblock instead of the domain name. An example log message is as follows:

```
info: 192.0.2.0/24 inform 2001:db8::a@5005 inf.example.com. A IN
```

Overview of Implementation Design

Unlike other local-zone and tag-based actions, whether to apply an IP-based action cannot be determined completely locally since it depends on the result of normal resolution. Also, since the “CNAME chasing” behavior will require additional sub-queries, we cannot just tweak the answer immediately before sending it to the client.

So we chose to implement the main functionality as a separate Unbound module named `respip` (meaning “response IP”). It’s intended to be placed on top of the module stack (usually immediately above the iterator module) and works as a filter for resolution results. Its `operate()` function first passes the state to the lower module to resolve the original query. When it’s done, it checks the answer section for AAAA or A records that may require IP-based action processing. If some action needs to be applied, the module modifies the answer accordingly (changing the IP address to a different one, converting the whole answer to NXDOMAIN, etc.) and usually completes the module. The exception is the case where a redirect to CNAME, in which case a sub-query is triggered to resolve the CNAME target, and the original query state is held until it completes. On completion of the sub-query, the module’s `inform_super()` function is called. It appends the resolved target records to the (already tweaked) answer section of the original answer. Finally, the control comes back to the `operate()` function, which simply completes the module processing.

The `respip` module will need to get access to part of `acl_addr` of the client that triggers a particular query so that it can apply tag-based or per-view actions. So we’ll need to extend the existing `mesh_new_client()` function so it takes the sub part of `acl_addr` and stores it in

the `module_qstate`. A new structure named `respip_client_info` is introduced for this purpose (we didn't like to expose `acl_addr` outside of unbound daemon, as it was deemed to be module-boundary violation; otherwise we could simply use `acl_addr` itself instead of the new structure). The mesh state comparison function will also have to be updated so that it takes into account `respip_client_info`. Note that this can lead to more external queries for the same qname and qtype if many ACL entries are defined for many different clients.

We also need to update the code that answers queries directly from cache (mainly in `daemon/worker.c`) in case the answer to the original query is already cached and it has to be modified by an IP-based action. In terms of observable behavior this could be avoided if we always call the module stack including the `respip` module; however, it's quite likely to be unacceptable in terms of performance since it would require additional function calls and many more data copies. (We won't be able to use the wire-format cached data directly as the resolution needs to go to the generic `iterator` module.)

We implementation updates the `answer_from_cache()` function for this purpose. Before encoding the answer RRset, it now checks if an IP-based action should apply, and if so, tweaks the answer accordingly. Again, the tricky case is a redirect to CNAME. In this case, it first needs to see if the CNAME target is cached, but to do so it will first make a local copy of the original RRsets, tweaks the answer, and releases acquired locks (it needs to release the lock for the additional cache lookup, and so it needs to make a copy of the cache data as the reference to the cache is not protected by the cache). Also, if the CNAME target is not cached, it will behave as if the original cache lookup failed to trigger the usual resolution (and processing by the `respip` module). These are complicated and inefficient, but seem to be unavoidable costs.