BIND 9

(Part 5 - Dynamic DNS Update Security, TSIG and Catalog-Zones)

Carsten Strotmann and the ISC Team
Welcome

Welcome to part five of our BIND 9 webinar series
In this Webinar

- Authenticating dynamic updates with TSIG
- Access Control Lists for dynamic DNS updates
- Catalog-Zones
- Hands-On dynamic zone management
Transaction Signatures (TSIG)
What is TSIG

- DNS Transaction Signatures (TSIG) are defined in RFC 8945 "Secret Key Transaction Authentication for DNS (TSIG).
- TSIGs secure the communication between two endpoints.
  - TSIGs use HMAC (i.e. symmetric encryption).
  - Trust is required between all systems (all endpoints).
  - Securely installing the key on all systems is external to DNS.
  - The endpoints must have reasonably accurate clocks.
- TSIG is independent of DNSSEC.
TSIG use cases

- TSIG use cases:
  - DNS dynamic updates (client / dhcp-server <-> server)
  - BIND server control messages (rndc <-> server)
  - Server communication (zone transfers, notifies, queries, …)
  - Queries (client <-> server): Impractical and rare.
TSIG implementation

TSIG Signed DNS Message
- Header
- Question
- Answer
- Authority
- Additional
- TSIG
TSIG implementation

• A TSIG is a dynamically generated pseudo-RR.
  ■ TSIG RRs are not found in zone files, but do have the standard format of RRs.
  ■ TSIGs are never cached (TTL=0).
  ■ A TSIG RR is sent in the additional section.
  ■ A TSIG assures the integrity and authenticity of the entire DNS message.
TSIG operation

Plain Text + PSK → HMAC-SHA256 → Hash → Plain Text

Plain Text + PSK → HMAC-SHA256 → Hash

transmit Plain Text & Hash

compare

Hash

All content © 2021 Internet Systems Consortium, Inc.
TSIG algorithms

- TSIG offers a choice of HMAC algorithms:
  - hmac-md5 (deprecated)
  - hmac-sha1 (deprecated)
  - hmac-sha224
  - hmac-sha256
  - hmac-sha384
  - hmac-sha512
TSIG algorithms

- All the algorithms take a random length input and create a fixed length fingerprint.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>fingerprint length</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>16 byte</td>
</tr>
<tr>
<td>SHA1</td>
<td>20 byte</td>
</tr>
<tr>
<td>SHA256</td>
<td>32 byte</td>
</tr>
<tr>
<td>SHA512</td>
<td>64 byte</td>
</tr>
</tbody>
</table>
Generating TSIG keys

- These BIND tools generate HMAC (TSIG) keys:
  - `rndc-confgen`: designed for keys for remote control.
  - `ddns-confgen`: designed for keys for DDNS/nsupdate.
  - `tsig-keygen`: a generic tool HMAC key creation. It was introduced in BIND 9.10.
Generating TSIG keys

- BIND 9.10 comes with the new command `tsig-keygen` to generate TSIG-keys.

```bash
% tsig-keygen my-key
key "my-key" {
    algorithm hmac-sha256;
    secret "EvfRifkexW+81OmqFJSc9Z07IBVZxvZbKVJBhF8BwHo=";
};
```
TSIG for Remote Control (RNDC)

- BIND's remote control tool, `rndc`, may use TSIG for authentication.
  - `rndc-confgen` creates a template configuration including keys.
TSIG to Secure Zone Transfer

- A TSIG can be used to secure zone transfers (e.g. primary to secondary).
  - The key must be configured on the server providing the zone and on the server transferring it in.

```plaintext
key prim-sec-example.com {
  algorithm hmac-sha256;
  secret "pDCQLRGpPN0h9ksqHBnGBra3U15QwlqfI5aPN05d5xE=";
};
zone "example.com" {
  type primary;
  file "example.com";
  allow-transfer { key prim-sec-example.com; }
};
```
TSIG to Secure Zone Transfer

- Note that both the key and the key-name must match on the sender and receiver.

```plaintext
key prim-sec-example.com {
    algorithm hmac-sha256;
    secret "pDCQLRGpPN0h9ksqHBnGBra3U15QwlQ5aPN05d5xE=";
};

# secondary zone
zone "example.com" {
    type secondary;
    file "example.com";
    masters { 192.0.2.53 key prim-sec-example.com; };
};
```
TSIGs for Securing All Communication Between DNS Servers

- TSIGs can be used to secure all communication between servers (queries, notifies, zone-transfers ...):

```plaintext
key server1-server2 {
    algorithm hmac-sha256;
    secret "pDCQLRGpPN0h9ksqHBnGBra3U15QwlpQI5aPN05d5xE=";
};
server 192.0.2.53 { # <-- IP Address of the remote DNS server
    keys { server1-server2; };
};
```
dynamic update security with ACLs and TSIG keys

- On a primary authoritative server, `allow-update {};` enables DDNS.
  - It also limits the updates to specifics keys or addresses.
  - In part 1 of this webinar on dynamic DNS zone management, we've used IP addresses to authenticate dynamic update sources.
  - Using IP addresses for authentication is weak security, as the source address of UDP based DNS update messages can be spoofed.
  - Attackers might be able to make unauthorized changes to the DNS zone content.
dynamic update security with ACLs and TSIG keys

- Using a symmetric key (TSIG) is more secure and recommended practice:

```plaintext
key dynamic-update-example.com {
    algorithm hmac-sha256;
    secret "pDCQLRgPN0h9ksqHBnGBra3U15QwlpQI5aPNO5d5xE=";
};

zone "example.com" {
    type primary;
    file "example.com";
    allow-update { key dynamic-update-example.com; }
};
```
Dynamic Update Policies
Fine Grained control for dynamic updates

- With allow-update, any change to the primary zone is secured by a key
  - The key can update the whole zone
  - Often more fine grained control is required
- The option update-policy can be used to provide a more flexible access control for dynamic updates
  - update-policy is mutually exclusive with allow-update
Update-Policy configuration syntax

• With `update-policy` we can grant or deny the resource record (RR) `identity` making changes
  - the `identity` is the name of a TSIG key

```
update-policy { grant identity matchtype tname [rr]; };  
deny identity matchtype tname [rr]; 
```
Update-Policy matchtype

- The *matchtype* controls which domain names can be updated with a TSIG key

```plaintext
update-policy { grant identity matchtype tname [rr]; };  
update-policy { deny identity matchtype tname [rr]; };  
```

<table>
<thead>
<tr>
<th>Matchtype (literal)</th>
<th>Modification available</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Target name (tname) only</td>
</tr>
<tr>
<td>subdomain</td>
<td>Subdomains of tname</td>
</tr>
<tr>
<td>zonesub</td>
<td>Subdomains of zone in which the update-policy statement appears</td>
</tr>
<tr>
<td>self</td>
<td>Own name (TSIG key name) only</td>
</tr>
<tr>
<td>selfsub</td>
<td>Updates own name and sub-domains of TSIG key name</td>
</tr>
<tr>
<td>selfwild</td>
<td>Only subdomains of self can be updated</td>
</tr>
<tr>
<td>wildcard</td>
<td>Wildcard expansion</td>
</tr>
</tbody>
</table>
**Update-Policy matchtype (Microsoft and Kerberos)**

- The *matchtype* can be used to create rules for updates from Microsoft AD or Kerberos signed updates.

```
update-policy { grant identity matchtype tname [rr];
                   deny identity matchtype tname [rr];
                       }
```

<table>
<thead>
<tr>
<th>Matchtype (literal)</th>
<th>Modification available</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms-self</td>
<td>allows a Microsoft client to update its own hostname</td>
</tr>
<tr>
<td>ms-selfsub</td>
<td>allows a Microsoft client to update its own hostname and subdomains</td>
</tr>
<tr>
<td>ms-subdomain</td>
<td>allows a Microsoft client to update any records inside its domain</td>
</tr>
<tr>
<td>krb5-self</td>
<td>allows a Kerberos client to update its own hostname</td>
</tr>
<tr>
<td>krb5-selfsub</td>
<td>allows a Kerberos client to update its own hostname and subdomains</td>
</tr>
<tr>
<td>krb5-subdomain</td>
<td>allows a Kerberos client to update any records inside its domain</td>
</tr>
</tbody>
</table>
Update-Policy matchtype (Microsoft and Kerberos)

- The matchtype can also be used to create rules for updates matching IP-addresses via TCP or 6to4 prefix names. The decision about update permission can also be delegated to an external process:

```
update-policy { grant identity matchtype tname [rr]; }; 
update-policy { deny identity matchtype tname [rr]; }; 
```

<table>
<thead>
<tr>
<th>Matchtype (literal)</th>
<th>Modification available</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp-self</td>
<td>Allows updates via TCP that match the domain name of the sender's IP address reverse name resolution</td>
</tr>
<tr>
<td>6to4-self</td>
<td>Allows the name matching a 6to4 IPv6 prefix to be updated via TCP from the 6to4 network or from the corresponding IPv4 address</td>
</tr>
<tr>
<td>external</td>
<td>Delegates the decision of whether to allow a given update to an external daemon</td>
</tr>
</tbody>
</table>

⚠ It is theoretically possible to spoof TCP sessions.
Update-Policy target name

update-policy { grant identity matchtype tname [rr]; };  
update-policy { deny identity matchtype tname [rr]; };  

- The target name tname defines the domain name or start of domain name-space that can be updated by this TSIG key
  - The target name can be a wild card
Update Policy Resource Record list

```
update-policy { grant identity matchtype tname [rr]; };
update-policy { deny identity matchtype tname [rr]; };
```

- **rr**: an optional space-delimited list of the resource record types on which updates are allowed (or denied)
- Keyword ANY matches any resource record type except NSEC and NSEC3
- If no record type is specified, it matches all record types except RRSIG, NS, SOA, NSEC, and NSEC3.
Example: Changing only record for a specific domain name

- holder of update-key can change A and AAAA records of www.example.com (and nothing below)

update-policy { grant update-key name www.example.com A AAAA; };
Example: Match TSIG key name to record

- The dynamic DNS update configuration below allows a system with a TSIG key with its own name to update its own IPv6 AAAA record
  - if the TSIG key is named www.example.com, that key can change the IPv6 address of the domain name www.example.com

```plaintext
update-policy { grant * self * AAAA; };
```
Example: A TSIG key to change all records below a sub-domain

- The holder of the TSIG key with the name superkey can change anything at or below example.org:

  ```
  update-policy { grant superkey subdomain example.org ANY; };
  ```
automatic DNS provisioning with Catalog-Zones
Provisioning New Zones

- Adding or deleting new zones can be a challenge.
- In addition to updating the primary, every secondary needs to be modified.
- It is intensive work for installations with many secondaries, or with frequent zone additions & deletions.
Provisioning New Zones - Solutions

- Many organizations have written scripts (or use tools like Ansible or SaltStack) for automatically modifying secondary DNS servers.
- A catalog zone provisions normal zones using standard DNS content and communication.
  - They are an ISC creation, new in BIND 9.11 (2016), and are being standardized by the IETF.
  - Internet draft (RFC "work in progress"): DNS Catalog Zones / February 2021
  - in addition to BIND 9, KnotDNS already has a fully functional implementation since version 3.0.0 (September 2020)
  - PowerDNS has a proof of concept external program called PowerCATZ ([https://github.com/PowerDNS/powercatz/](https://github.com/PowerDNS/powercatz/)), that can process DNS Catalog Zones
Catalog Zone

- A catalog zone works like a normal DNS zone.
- A catalog zone is maintained on the primary server.
- It contains zone names and configuration metadata that should exist on secondaries.
- Zones added to the catalog zone are automatically provisioned on secondaries.
  - Zones in a catalog zone are member zones.
Prerequisites for Catalog Zones

• A primary DNS server hosting a catalog zone, does not need to be updated to BIND 9.11 or later.
  ▪ This is because catalog zones use standard DNS content and communication.
  ▪ Secondaries need to be updated so they will use a catalog zone's content as provision information.

• A secondary will have a catalog zone for each primary.
  ▪ Assuming the primary is configured with a catalog zone.
Catalog Zone: named.conf:Primary

- In a primary's named.conf, a catalog zone is a completely normal zone.
- It can be a static (managed via text editor) or dynamic zone.
- There are no special requirements for the configuration, nor for the name of the zone. However, the name of the catalog zone should not collide with any domain name used in the network (Internet and private DNS).

```plaintext
group {    // DNS server group information
    hosts {            // Names of servers to forward queries to.
        host {192.0.2.100 {forward}};    // Forward query for 192.0.2.100
        host {192.0.2.200 {forward}};    // Forward query for 192.0.2.200
    }
    type hints;  // This is the type of file.
    file "catalog.example.com";
    };

zone "catalog.example.com" {
    type primary;
    file "catalog.example.com";
};
```
Catalog Zone: Zonefile:Primary

- A catalog zone has a PTR RR for each member zone. In the catalog zone shown, no member zones are provisioned yet. It is empty. (It has no PTR RRs).
- It must also contain a TXT record with the version number of the catalog zone protocol implementation
  - Version 1: the catalog zone protocol as it has been implemented in BIND 9.11
  - Version 2: the catalog zone protocol as described in the Internet Draft and implemented in BIND 9.16
  - DNS server software will ignore catalog zones with a version number it does not support
Catalog Zone: Zonefile:Primary

```
$TTL 60
catalog.example.com. SOA primary.example.com. hostmaster ( 1001 2h 20m 41d 1h )
  NS primary.example.com.
  NS secondary01.example.com.
  NS secondary02.example.com.
version.catalog.example.com. TXT "1"
```

- A TXT record defines the version of the catalog zone feature supported.
- Secondary servers should have NS RRs so that they receive notifies when the catalog zone updates. (I.E. Standard procedure.)
- Names should be resolvable for notify to work.
Catalog Zone: named.conf:Secondary

- Configuration for catalog zones is found on secondaries.
Provisioned Member Zone

- The zone file for a member zone is created as normal on the primary, just as is done with any zone (not shown). It is added to `named.conf` just like any other zone.

```bash
zone "catalog.example.com" {
    type primary;
    file "catalog.example.com";
};

zone "newzone.example.com" {
    type primary;
    file "newzone.example.com";
};
```

The configuration added earlier for the catalog zone.

The new zone that will be automatically provisioned.
registering a new zone in the catalog

- the new zone needs to be registered in the catalog zone
- registration happens with a PTR (Pointer) record where the data part of that record is the zone name (newzone.example.com in this example)
  - the domain name of the record must be a unique name (for example the SHA1 hash of the new zone's name), the label zones and the domain name of the catalog zone (new-zone20210615.zones.catalog.example.com. in the example)
registering a new zone in the catalog

```
# cat catalog.example.com
$TTL 60
catalog.example.com. SOA authoritative.example.com. primary (1002 2h 20m 41d 1h )
  NS authoritative.example.com.
  NS secondary01.example.com.
  NS secondary02.example.com.

version.catalog.example.com. TXT "1"

new-zone2021061501.zones PTR newzone.example.com.
```

- The previously "empty" catalog zone, with one provisioned zone added & the SOA serial increased.
- Mapping the symbolic name to the name of the zone.
- Unique domain name identifying the new zone.
Provisioning Success

- The secondary automatically serves the new member zone.
- Updates to the member zone will automatically be transferred to the secondary.
  - This is with normal methods (NOTIFY, IXFR, etc).
- Additional zones added to the catalog zone will also be automatically provisioned on the secondaries.
- A zone removed from the catalog zone will be removed by the secondaries.
  - A zone backup file on a secondary will be deleted.
additional zone block configuration

- Catalog zones can contain configuration options for the new zone block (like Access Control Lists, list of primaries)
- Options can be specified global for the whole catalog zone, or specific for each zone listed in the catalog zone, etc.
- Details can be found in the BIND Administrator Reference Manual:
additional zone block configuration

- **primaries**: this option sets one or more primary DNS servers for the secondary zone. Primaries can be defined as IPv4 A or IPv6 AAAA records.
- **allow-query**: this option defines the allow-query ACL. The ACLs are defined with the help of the APL Resource record (see RFC 3123 "A DNS RR Type for Lists of Address Prefixes (APL RR)")
- **allow-transfer**: this option defines the allow-transfer ACL. It also uses the APL record.
additional zone block configuration

```
$TTL 60
catalog.example.com. SOA authoritative.example.com. primary ( 1004 2h 20m 41d 1h )
    NS authoritative.example.com.
    NS secondary01.example.com.
    NS secondary02.example.com.

version.catalog.example.com. TXT "1"
primaries AAAA 2001:db8::53
new-zone2021061501.zones PTR newzone.example.com.
primaries.new-zone2021061501.zones A 192.0.2.176
allow-transfer.new-zone2021061501.zones APL 1:192.0.2.196/32 1:192.0.2.199/32
```

definition of the default primary server(s)
definition of the zone’s primary server(s) (for zone “newline”)
access control list for zone-transfer (for zone “newzone”)}
Resources

- **TSIG:**

- **Dynamic Update Policies:**

- **BIND 9 Catalog Zones:**
Questions and Answers
Hands-On

- We have prepared a VM machine for every participant
- This time the sessions does not build upon each other and do not need to be done in order
- find the instructions at https://webinar.defaultroutes.de/webinar/05-ddns-workshop.html