CcspXDNS

Introduction

CcspXDNS is a CCSP component which provides RDK-B the ability to cater to customise the DNS related requests. In essence this component enables the XDNS feature on RDK-B.This component can be used to set the xDNS information for a specific client MAC, group of clients or even a whole gateway. By default this feature is disabled. The related Tr-181 parameter is:

`Device.DeviceInfo.X_RDKCENTRAL-COM_EnableXDNS`

This feature can be enabled by setting the above mentioned parameter as true. When enabled all DNS requests are interrupted regardless of the client DNS configuration.

Xfinity DNS (XDNS) is a feature that classifies DNS requests from connected user devices. The XDNS server will apply policy such as URL filtering based on the request classification.

XDNS contains two functional components:

1. XDNS Resolver: A XDNS Resolver is the DNS Stub Resolver implemented in RDKB that appends a Tag to DNS request based on the provisioned metadata.
2. XDNS Server: A XDNS Server is the DNS recursive/caching server that would read the Tag and execute the policy associated to the tag.

The XDNS can be used for various use cases. For example: XDNS can be used for DNS based parental control function and DNS based redirect function. This specification uses the parental use case as the primary function. But the same mechanism will work on any use case that uses the tagging in the DNS request. Note that DNS server is a critical infrastructure that is built to be highly scalable with minimum overhead. Any update must be thoroughly tested and does not introduce heavy load to the server. Given this limitation, the design in the specification is designed to alleviate necessary changes in DNS server.

Features

XDNS Parental Control Feature

XDNS Parental Control is a feature that enables users to apply URL content filtering. For example: we can setup three network wided URL filter profiles (e. g., G, PG and R). Users can choose to enable content filtering for the entire home, VLAN and/or a device. The filters are hierarchical (e.g, PG is a super-set of G) and managed by the XDNS infrastructure. They are system-wide and can’t be customized.

Since this is domain-level (i.e., FQDN) filtering, XDNS doesn't support URL based categorization. As such, when a domain is allow (or disallowed), the entire domain will be allowed (or disallowed). XDNS can't filter sub-domain level URL. For example: XDNS can't be configured to allow "www.foo.com" but disallows "www.foo.com/gambling".

DNS Blacklist and Whitelist

In some occasions user may want to create Blacklist and/or Whitelist on top of the standard profile. For example: A user may set a tablelet to use PG profile but disallows "Social Network" such as Twitter, Google+ and Facebook. A user can enter individual FQDN into the account to allow or disallow a set of URLs. The Blacklist and Whitelist must be implemented locally in the RDKB. Each device can have up to 50 entries combined for Blacklist and Whitelist. The XDNS function could be implemented as a new column in the existing Parental Table or a new Table. The primary key will be the user device mac-address. Here is pseudo code to implement a new XDNS profile:

Pseudo XDNS Profiles illustrated in Golang

```
type profile_t {
    Name, Category string
 }

type xdns_t struct {
    Profile profile_t
    Blist, Wlist map[string]int
}

// This is the XDNS profile list that contains all profiles indexed by Device MAC-Address
xdns_profiles := make(map[string]*xdns_t)

// Create a XDNS profile for a new Connected Device "00:11:22:33:44:55"
var xdns xdns_t
xdns_profiles["00:11:22:33:44:55"] = &xdns

// Insert "www.example.com" to Whitelist
xdns_profiles["00:11:22:33:44:55"].Wlist["www.example.com"] = 1

// Insert "www.foo.com" to Blacklist
xdns_profiles["00:11:22:33:44:55"].Blist["www.foo.com"] = 1
```

XDNS Requirements for RDKB

Each profile has a corresponding "XDNS_PROFILE_ID". The XDNS_PROFILE_ID is a local OPT code (i.e. 65001-65534) that is carried in the EDNS (RFC 6891). XDNS_PROFILE_ID must contain both the XDNS profile and Client's mac-address separated by ";". For example: A user device 00:11:22:33:44:55 was provisioned to use "G" XDNS profile. When the user device made a DNS query, the DNS query must contain XDNS_PROFILE_ID *G;00:11:22:33:44:55* in the EDNS OPT RR. Note that the OPT is yet to defined. DNS server only needs to read the first string separated by ";" to classify the filter (e.g, G). The second part of the string (i.e., mac-address) is irrelevant to DNS server.

When a device makes a DNS query, RDKB must:

- Intercept the query and forward to the local DNS proxy
- Identify the DNS requester's profile based on the Connected Device's mac-address
- Add the EDNS OPT RR that contains the XDNS_PROFILE_ID . Here is the pseudo logic of the XDNS stub resolver (e.g., dnsmasq):

Stub Resolver Pseudo Logic

```
// User device abc makes a dns request
if xdns_profiles[abc.Dev_MAC].Wlist[dnsReq.Fqdn] == 1 {
    Leave the dns_query intact
} else if xdns_profiles[abc.Dev_MAC].Blist[dnsReq.Fqdn] == 1 {
    Create edn0_rr with "BLOCK;abc.Dev_MAC"
    Add to the dns_query
else {
    Create edn0_rr with xdns_profiles[abc.Dev_MAC].Profile.Category
    Add to the dns_query
}

Proxy the dns_query to dns_anycast_address
```

RDKB Implementation Notes

- Must avoid IP fragmentation because DNS servers will drop fragmented packets. Recommend to set the packet size at least 1280-byte
- Must avoid switching to TCP unless the query is larger than IP payload size. TCP setup overhead is large. It will have significant impact to the latency. It also increases the DNS server load.
- May consider enabling DNS caching in the proxy. The TTL must be equal to the TTL of the RR in the response. This may decrease hitting the DNS server for the same query in a given TTL interval.

XDNS Profile Provisioning

When a new device associated to the RDKB, the RDKB must query XPC for the XDNS profile to this device. We may leverage the Parental object and add a new column to store the Category. The Key provisioning may also leverage the Parental control feature that sets Internet Access to a device.

Back to our example, XDNS would create three categories called "G, PG and R". When a user device connects to the RDKB, RDKB must signal XPC that a new device is just attached (e.g., hosts table notification). XPC will post the corresponding profile of the device to associated CPE, VLAN and device.

XDNS Profile Management

XDNS profile can be set on CPE, VLAN and device. The XPC API is hierarchical. As such, CPE is the root of VLAN and VLAN is the root of Device. Therefore, it is possible that the profile set on CPE is different from the profile set on a device. The rule here is:

1. Device XDNS profile overwrites VLAN profile
2. VLAN profile overwrites CPE profile

For example: CPE's XDNS profile is set to G and VLAN A profile is set to PG. A device in VLAN A will use PG rather than G. RDKB must use the PG's token in EDNS0 when it sends the query to the XDNS infrastructure.

RDKB doesn't implement this hierarchical structure for connected devices. This requires XPC to flatten the structure internally before passing the correct token associated to the connected device.

XDNS Report Requirements

XDNS Reporting is a key feature that shows the XDNS value to the user. For this phase, XDNS will show a non-realtime offline report for blocked queries. For each blocked query, the report will contains:

1. Time-of-Day of the Block
2. Profile Name
3. Site FQDN
4. Blocked Category
5. Client mac-address

To create a token in the DNS response to identify this is a response of XDNS, we will create a CNAME RR for each blocked FQDN. For example: For each FQDN that is categorized for ", we will create a CNAME "redirect.xdns.xfinity.net." Hostname "redirect" represents the "action". Sub-domain "xdns.xfinity.net " is the sub-string that classifies the query is blocked. Here is the design overview:

Code Flow

XDNS Server Requirements

Thread Intel Server must provision a CNAME for each blocked FQDN. The CNAME RR is just a simple CNAME RR. DNS server will process it just like any CNAME. It doesn't understand the context.

When DNS server receives a query that is being blocked, it must return the CNAME RR and the A RR (or AAAA RR) in the response.

Example:

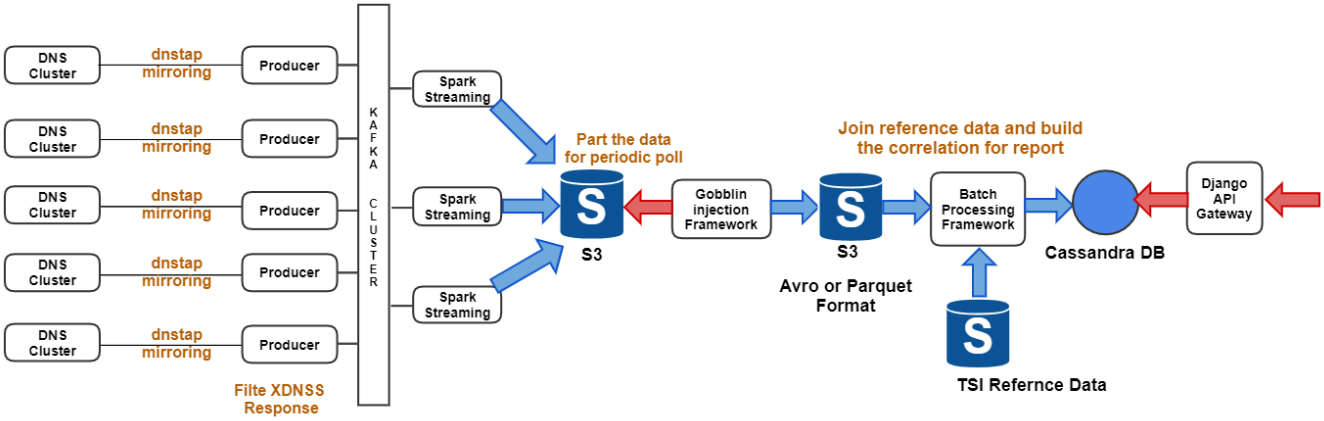
```
tomato:~ xdns_client$ dig @a.b.c.d www.poker1234.com

;; QUESTION SECTION:
;www.poker1234.com.                IN    A

;; ANSWER SECTION:
www.poker1234.com.          5       IN   CNAME redirect.xdns.xfinity.net.
redirect.xdns.xfinity.net. 7182     IN   A     76.33.44.55
redirect.xdns.xfinity.net. 8        IN   AAAA  2001:559:0:d::1234:a001
```

Each DNS cluster must enable a mirror port in the cluster to mirror all requests and reports to an Offline Report Engine. The Report Engine must scale to capture a large amount of DNS messages and must go beyond to support all DNS data.

XDNS Report Engine Requirements

A propose mechanism is to use Kafka cluster and Spark Streaming to filter and pair a request and response only for XDNS. In future, we could potentially scale the Report Engine infrastructure to include all data for DNS for analytic. The architecture should work for XDNS as well as general DNS data injection.

This will leverage the Crystalball Batch Process architecture. The following is the high-level architecture diagram:



DNS Cluster will mirror or packet capture all the DNS request and responses to the Kafka Cluster. For XDNS, the Producer process will filter DNS response that contains CNAME with "xdns.xfinity.net". This will help to reduce the number of messages to the Crystalball infrastructure.

Spark consumer will store all the messages in S3. Gobblin Injection Framework will wake up periodically and pull the data from S3. Gobblin will group and transfer it from ASCII to Parquet format, then part the data in S3 in hierarchical order by date. More Gobblin architecture can be found here.

XDNS Report Engine (XDNS-RE) will be running in the "Batch Processing Framework". The XDNS-RE will correlate the information from the reference data. The reference data is provided by the "Thread Intel Service (TIS)". The reference data must be made available to the XDNS-RE and updated in a predefined interval (e.g., /12hrs or /24hrs). The reference data will be stored in the following format:

```
[{
    "fqdn" : "www.poker1234.com",
    "action" : "redirect",
    "level" : "pg",
    “category” : "gambling",
    "create_timestamp" : 1469815213,
    "update_timestamp" : 1469815213
}]
```

XDNS-RE will join the TIS reference data to the XDNS responses and store the information in the Cassandra database in the following structure:

```
[{
    "fqdn" : "www.poker1234.com",
    "cname" : "redirect.xdns.xfinity.net.",
    "timestamp" : 1473127506,
    "dev-mac-address": "00:11:22:33:44:55",
    "action" : "redirect",
    "level" : "pg",
    “category” : "gambling"
}]
```

Safe Search Design

Google, Yahoo and Bing allow DNS service provider to enforce "Safe Search". The common method is to create a CNAME Record in DNS server pointing to the "safesearch fqdn" (e.g., `forcesafesearch.google.com`). What is the best way to signal this to DNS server? It is possible to create a new parameter to enable and disable Safe Search for individual user device. However this will increase the DNS server load to examine individual query.

A better approach is to enable Safe Search for all restricted policies (e.g., P, PG and PG13) and only disable Safe Search in less restricted policies (e.g., R). Users won't have an option to enable or disable Safe Search as an independent function.

Limitations of DNS-based Filtering Solutions

- Malware can bypass DNS-based filtering solutions by just hard-coding an IP address thereby avoiding a DNS lookup altogether. This is one of the biggest known weakness of DNS based Anti-Malware Solution. For DNS-based parental solution, this isn't as much concern as Anti-Malware.
- Users can enter other DNS servers, thereby bypassing our DNS. In XDNS implementation, all DNS packets will be intercepted and forced to the XDNS server.

- These solutions act at the domain level rather than the URL level so are not granular by default and offer limited scope in terms of categorization. This is another known weakness for DNS-based filtering solution. DNS is domain based, it can't filter multiple-level URL.

- HTTPS is gaining more popularity. Many conventional websites start using https as default protocol even for non-secure content. When XDNS returns a RR that contains the "XDNS Redirect Server"'s IP and the client initiates a https connection the the XDNS Redirect Server, this will break TLS client handshake. Modern web client application including all well-known web browsers will stop the TLS handshake and pop-up a security warning. The client will interpret this as a man-in-the-middle intercept and won't start the https transaction (i.e., won't communicate to the XDNS Redirect Server). This will give a bad experience to users and leave the client in limbo for user interaction (accept the unsafe cert and refuse the connection).