2004 年度 卒業論文

プライバシーを考慮した DNS システムの提案

提出日： 2005 年 2 月 2 日

指導：後藤滋樹教授

早稲田大学　理工学部情報学科
学籍番号： 1G01P062-1

舘直芳

目次

図一覧

表一覧

第 1 章

序論

1.1 研究の背景

現在インターネットは社会生活に欠かせないものとなっている。その中で、インターネットの扱うアプリケーションは様々な分野へと広がりを見せている。かつては e-mail や Web といった片方向のやり取りが中心であったが、ユーザのインターネットへの接続環境がブロードバンド化、常時接続化されるに従い、インスタントメッセンジャーや IP 電話、テレビ会議システムなどのリアルタイム双方向で通信を行うアプリケーションも増えてきている。こうした中で、通信相手を特定するアドレスや、番号はアプリケーションごとに様々であり、一元管理できる仕組みが求められるようになってきた。そこで考えだされたのが ENUM（tElephone NUmber Mapping）と呼ばれる、世界的に一意な電話番号を元に、インターネット上にアドレス帳をつくろうという仕組みである。具体的には、電話番号をキーとして DNS を引くと、その電話番号に登録された様々なアプリケーションの情報が取り出せるようにするものである。ところで DNS はインターネットユーザに対しては誰彼の区別なく、登録されている情報を返すオープンなデータベースである。現在 spam メールやワン切りが社会的な問題となっている。 DNS のレコードとして e-mail アドレスなどの個人情報を公開することは、意図しない悪用をされる恐れがあり、プライバシーの観点から問題となっている。

1.2 研究の目的

前述のとおり、 ENUM のように新しい仕組みの中で DNS を用いることで、これまでにない問題が浮上してきた。プライバシーの問題である。 DNS はその構成上、登録されたデータは世界に公開されることになる。 ENUM では e-mail アドレスや SIP アドレスといった個人情報になりうるものが登録されるが、それを隠す技術が存在しない。本論文ではこのようなプライバシーを守る手段として、 DNS に登録されるデータである資源レコード情報を DNS 管理者が自由に

隠せる仕組みを提案し、実装する。 ENUM では主に NAPTR 資源レコードが用いられるが、本研究では現在問題となっている NAPTR はもちろん、任意の資源レコードを隠すことを目的とする。任意の資源レコードを隠すことを可能としておけば、現在ある NAPTR の問題だけでなく、今後 DNS のデータを隠したい場面が出た場合にも有効である。

1.3 本論文の構成

本論文は以下の章により構成される。

第 1章 序論

本研究の概要について述べる。

第 2章 DNS と ENUM

DNS と ENUM の概要を述べる。

第 3章 プライバシーを考慮した DNS システム

システムの概要について述べる。

第 4章 実験

提案した方法に基づき実装し、実験を行う。

第 5章 まとめ

まとめ。

第 2 章

DNS と ENUM

2.1 DNS（Domain Name System）の概要

DNS はインターネットを支える重要なシステムである。 DNS は人間が覚えやすいホスト名と、コンピュータが扱いやすいインターネットアドレスとの対応付けを行う。そして、実際にはアドレスだけにとどまらず、インターネット上のホストに関する各種情報を公表し、アクセスするための標準的なメカニズムを提供する。 DNS は電子メールをはじめ、 Telnet、 FTP、 Web ブラウザなど、ほとんどすべてのインターネットソフトウェアで利用されている。また DNS はホスト情報をインターネット上のどこからでもアクセスできるようにしている。

2.2 ENUM（tElephone NUmber Mapping）の概要

ENUM は、電話番号から DNS を用いてインターネット上で利用できるアプリケーションのサービス情報を得る仕組みである。電話番号を名前空間として、アプリケーションのサービス情報を DNS サーバに登録する。そして電話番号をキーとして DNS サーバにアクセスすることにより、電話番号に対応付けられたサービス情報を得る。 ENUM では電話番号として E.164 番号を、またアプリケーションのサービス情報として URI を用いる。

2.3 資源レコード（Resource Record）

ネームサーバでは、 IP アドレスだけではなく様々な情報を資源レコードと呼ばれる形式のデータで管理している。資源レコードには、クラスとタイプという 2 つの分類がある。クラスはインターネットクラス（IN）を使用するのが一般的である。表 2.1と表 2.2に現在利用されているタイプを示す。

表 2.1: 現在使われている資源レコードタイプ (1)

資源レコードタイプ	型コード	説明
INVAID	0	Cookie
A	1	Host address
NS	2	Authoritative server
MD	3	Mail destination
MF	4	Mail forwarder
CNAME	5	Canonical name
SOA	6	Start of authority zone
MB	7	Mailbox domain name
MG	8	Mail group member
MR	9	Mail rename name
NULL	10	Null resource record
WKS	11	Well known service
PTR	12	Domain name pointer
HINFO	13	Host information
MINFO	14	Mailbox information
MX	15	Mail routing information
TXT	16	Text strings
RP	17	Responsible person
AFSDB	18	AFS cell database
X25	19	X25 calling address
ISDN	20	ISDN calling address
RT	21	Router
NSAP	22	NSAP address
NSAP-PTR	23	Reverse NSAP lookup (deprecated)
SIG	24	Security signature
KEY	25	Security key
PX	26	X.400 mail mapping
GPOS	27	Geographical position (withdrawn)
AAAA	28	Ip6 Address
LOC	29	Location Information
NXT	30	Next domain (security)

表 2.2: 現在使われている資源レコードタイプ (2)

資源レコードタイプ	型コード	説明
EID	31	Endpoint identifier
NIMLOC	32	Nimrod Locator
SRV	33	Server Selection
ATMA	34	ATM Address
NAPTR	35	Naming Authority PoinTeR
KX	36	Key Exchange
CERT	37	Certification record
A6	38	IPv6 address (deprecates AAAA)
DNAME	39	Non-terminal DNAME (for IPv6)
SINK	40	Kitchen sink (experimentatl)
OPT	41	EDNS0 option (meta-RR)
APL	42	Address prefix list (RFC 3123)
TKEY	249	Transaction key
TSIG	250	Transaction signature
IXFR	251	Incremental zone transfer
AXFR	252	Transfer zone of authority
MAILB	253	Transfer mailbox records
MAILA	254	Transfer mail agent records
ANY	255	Wildcard match
ZXFR	256	BIND-specific, nonstandard
MAX	65536	Max

2.4 NAPTR 資源レコード

ENUM で利用される NAPTR 資源レコードの構成を以下に示す。

```
IN  NAPTR  order  preference  flags  service  regexp  replacement
```

NAPTR 資源レコードの各フィールドについて説明する。

- `IN NAPTR`

それぞれ資源レコードのデータのクラスとタイプを表している。

- `order, preference`

  複数の NAPTR 資源レコードが返された場合に、処理順序を示す 16bit 符号無し整数。小さい値が優先される。まず `order` が比較され、同値の場合は `prefernce` が評価される。

- `flags`

  正規表現処理やフィールドの解釈を制御する。 ENUM の NAPTR 資源レコードでは、置換処理後の文字列が URI であることを示す “u” のみが定義されている。

- `service`

  ENUM で使うプロトコルである E2U (ENUM to URI) とサービス名を “+” で結びつけたもの。例を表 2.3に示す。

- `regexp, replacement`

  `regexp` には正規表現を書く。正規表現処理に関しては次節で説明する。 `replacemnet` は `regexp` がある場合はドット “ . ” を書く。

表 2.3: `service` フィールドと URI の例

サービス	service フィールド	URI の例
SIP	E2U+SIP	sip:tati@sip.goto.info.waseda.ac.jp
H.323	E2U+H323	h323:tati@h323.goto.info.waseda.ac.jp
電話	E2U+tel	tel:+81352863182
ファックス	E2U+fax:tel	tel:+81352863182
e-mail	E2U+email:mailto	mailto:tati@goto.info.waseda.ac.jp
Web	E2U+web:http	http://www.goto.info.waseda.ac.jp/
FTP	E2U+ft:ftp	ftp://ftp.goto.info.waseda.ac.jp

URI とは、インターネット上の情報資源を特定するための統一された構文を持つ一般的な形式である。このように登録される URI が個人情報になりうるわけである。

2.5 DNS メッセージフォーマット

2.5.1 全体の構成

DNS メッセージフォーマットは様々な情報を扱えるようになっている。その構成は次のようになっている。ヘッダ部により、該当データがどのような内容を含んでいるかがわかり、それにより続くデータの内容が決まってくる。

表 2.4: DNS メッセージのフォーマット

ヘッダ部	常に存在
問い合わせ部	ネームサーバへの問い合わせ
回答部	ネームサーバの回答
権威部	権威情報
付加情報部	付加的な情報

2.5.2 ヘッダ部

ヘッダ部は DNS メッセージの先頭に必ず存在するデータである。この内容に応じたデータがヘッダ部以降に続くことになる。フォーマットは次のとおりである。

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td colspan="16">ID</td></tr>
<tr><td>QR</td><td colspan="4">OPCODE</td><td>AA</td><td>TC</td><td>RD</td><td>RA</td><td colspan="3">Z</td><td colspan="4">RCODE</td></tr>
<tr><td colspan="16">QDCOUNT</td></tr>
<tr><td colspan="16">ANCOUNT</td></tr>
<tr><td colspan="16">NSCOUNT</td></tr>
<tr><td colspan="16">ARCOUNT</td></tr>
</table>

図 2.1: DNS メッセージ：ヘッダ部

ヘッダ部の項目について説明する。

- `ID`

  識別用の数値。問い合わせと応答のマッチングを行う。

- QR

  0：問い合わせ、1：応答

- OPCODE

  0：標準的な問い合わせ（QUERY）、1：逆問い合わせ（IQUERY）、2：サーバー状態（STATUS）、3 〜 15：将来のための予約

- AA

  権威付の応答かどうか

- TC

  転送チャネルの最大長より長いため切り詰めたかどうか

- RD

  再帰的な問い合わせの指示

- RA

  サーバが再起可能かどうか

- Z

  将来のための予約。必ず 0 にする。

- RCODE

  0：エラーなし、1：フォーマットエラー、2：サーバ障害、3：指定ドメインが存在しない、4：問い合わせにサーバが非対応、5：拒否

- QDCOUNT

  問い合わせ部のエントリ数。

- ANCOUNT

  回答部の資源レコード数。

- NSCOUNT

  権威部のネームサーバ資源レコード数。

- ARCOUNT

  付加情報部の資源レコード数。

2.5.3 問い合わせ部

ヘッダ部の QR が問い合わせを意味する 0 の場合に必要であり、応答にも含まれる。フォーマットは次のとおりである。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
QNAME															
QTYPE															
QCLASS															

図 2.2: DNS メッセージ：問い合わせ部

問い合わせ部の項目について説明する。

- QNAME

  長さ（1 バイト）＋ドメイン名の繰り返し。ヘッダ部の QDCOUNT で指定されている数入る。可変長である。

- QTYPE

  問い合わせの型を示す。

- QCLASS

  問い合わせのクラスを示す。インターネットクラスであれば IN になる。

2.5.4 回答部、権威部、付加情報部

DNS サーバからの応答に含まれる。ヘッダ部の QR が応答を意味する 1 の場合に含まれる。フォーマットは次のとおりである。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NAME															
TYPE															
CLASS															
TTL															
RDLENGTH															
RDATA															

図 2.3: DNS メッセージ：回答部、権威部、付加情報部

回答部、権威部、付加情報部の項目について説明する。

- NAME

  長さ（1 バイト）＋ドメイン名の繰り返し。ヘッダ部の QDCOUNT で指定されている数入る。可変長である。

- TYPE

  資源レコードの型。 RDATA のデータの意味。

- CLASS

  RDATA のデータのクラスを示す。

- TTL

  資源レコードのキャッシュ時間。 0 の場合はキャッシュしてはならない。

- RDLENGTH

  RDATA フィールドの長さ（バイト数）。

- RDATA

  資源を記述した可変長のデータ。 TYPE と CLASS によってフォーマットが変わる。

第 3 章

プライバシーを考慮した DNS システム

3.1 基本方針

新しい資源レコードを定義し、その情報を元に特定の資源レコードを隠す方法を述べる。具体的に、すべきことは以下の 3 つである。

- HIDE 資源レコードの定義
- DNSCLIENT の作成
- DNSPROXY の作成

まず DNS に隠したい資源レコード情報を新しい資源レコードである HIDE 形式で記述する。この HIDE 資源レコードには隠したい資源レコードタイプと、それに対するキーワード情報が入っている。 DNSPROXY を DNS サーバ上で動かし、 DNSPROXY は HIDE 情報を元に、問い合わせに対して、応答の可否を決定する。 DNSCLIENT はキーワードを付加した DNS クエリを作成し、 DNS サーバに送信する。なお、今回は DNS 自体のプログラムには改変を行わない方法をとる。この動作を図 3.1 に示す。

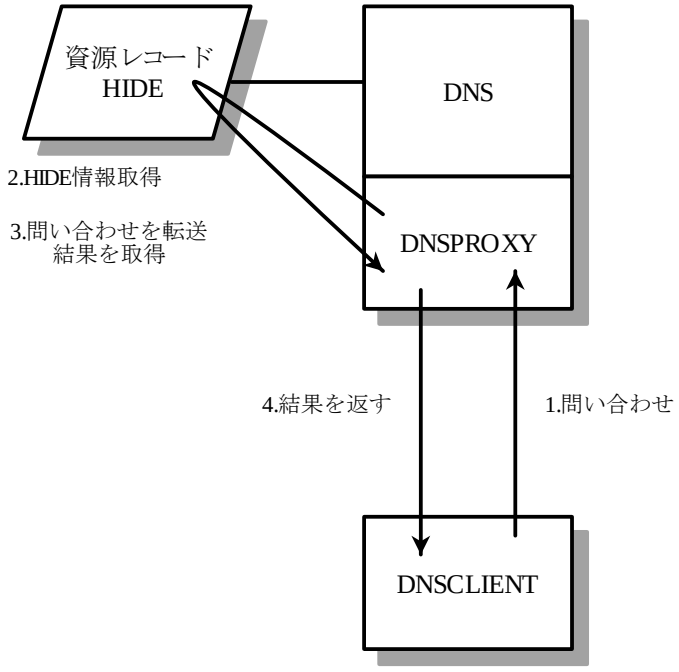


図 3.1: プライバシーを考慮した DNS システムの概略

3.2 実装準備

新しい資源レコードの定義、 DNS への記述方法、 DNSPROXY、 DNSCLIENT の動作を説明する。

3.2.1 HIDE 資源レコードの定義

新しい資源レコードとして HIDE を以下のように定義する。

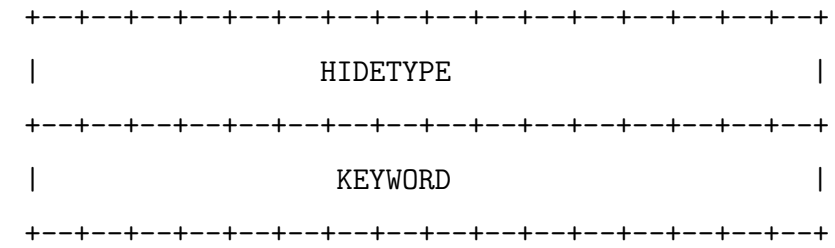
テキスト表現

owner[1] ttl[2] class HIDE hidetype keyword

例： 2.4 IN HIDE NAPTR password

バイナリ表現

HIDE 型のコード： 99



説明

HIDETYPE　　隠したい資源レコードタイプ（1 オクテット）

KEYWORD　　資源レコードに対するキーワード（8 オクテット）

HIDE 資源レコードは HIDETYPE と KEYWORD の 2 つのフィールドから成る資源レコードである。 HIDETYPE に 255 を指定した場合は、その名前に対してすべての資源レコードを隠す。

3.2.2 DNS への記述方法

HIDE レコードをそのままの形で DNS へ記述すると、不正なタイプの資源レコードとしてエラーを返す。つまり、 HIDE をそのままの形で実装するためには、 HIDE 情報の記述された DNS サーバだけでなく、それを利用するクライアントのネームサーバソフトウェアに対する変更が必

[1]owner とは、資源レコードの定義対象のドメイン名を指す（RFC1035 参照）。

[2]ttl は省略可能である。例では ttl を省略している。

要になる。さらにはそれを含んでいるゾーンの全スレーブサーバと、ある場合にはクライアントが使用するキャッシュネームサーバとフォワーダの変更も必要である。そこで今回は現在稼動している DNS サーバに変更を加えることなく実装するため、 RFC3597 に基づき、 DNS へ記述することとする。 RFC3597 は未知のタイプの資源レコードの扱いについて記述している。具体的には「タイプ」フィールドは、未知の資源レコードタイプを単語「TYPE」と空白なしで直後に続く 10 進数の資源レコードタイプ番号で表現する。また「クラス」フィールドでは、未知のクラスを、単語「CLASS」と直後に続く 10 進数のクラス番号で表現する。資源レコード部分では特別なトークン\#、オクテット単位で資源データの長さを指定する符号なしの 10 進整数、偶数個の 16 進数を含む実際の資源データフィールドをコード化するゼロ以上の 16 進データをそれぞれ記述する。

```
owner ttl class HIDE hidetype keyword
↓
owner ttl IN TYPE99 \#9 hidetype keyword
```

このように class は通常通り IN、 HIDE を TYPE99 と表現し、資源レコードの長さは 9 オクテットであるので、\#9 と記述する。そのあとに続く hidetype と keyword は 16 進で記述する。このように記述することで現在稼動中の DNS サーバ上でも不正な資源レコードではなく、未知の資源レコードとして扱われるため問題なく動作するようになる。

```
2.4 IN TYPE99 \#9 23 70617373776f7264
```

これは NAPTR にキーワードを ”password” として登録する場合の例である。 23 は NAPTR の型コード 35 を 16 進数に、 70617373776f7264 は password を ASCII 文字コードに基づき 16 進数に変換したものである。

3.2.3 DNSCLINET

DNSCLIENT はキーワードを付加したクエリを作成し、 DNS サーバに送信するプログラムである。キーワードは付加情報部に TXT として付加し、 TTL を 0 とする。 DNSCLIENT が作成するクエリは以下のようになる。

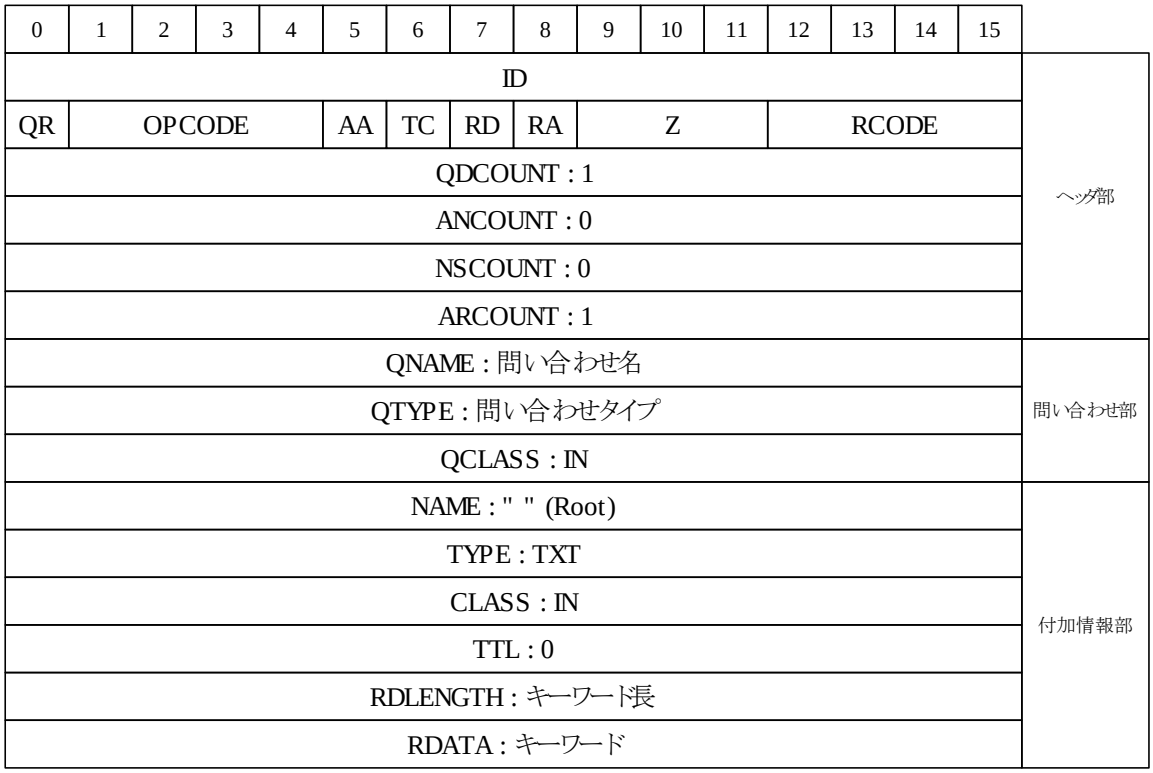


図 3.2: DNSCLIENT が作成するクエリ

実際に問い合わせの名前を 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp、タイプを 35 （NAPTR）、キーワードを ”testpass” として作成したクエリを以下に示す。

```
./dnsclient 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 testpass

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10282
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      2.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN
.                        0S IN TXT       "testpass"
```

3.2.4 DNSPROXY

DNSPROXY は DNS メッセージの解析を行うプログラムである。動作の流れは下のようになっている。

1. クライアントからの DNS メッセージを受け取る。

↓

2. 問い合わせの名前、問い合わせタイプ、キーワードを取り出す。

↓

3. 問い合わせの名前、タイプをもとに、 DNS サーバに HIDE 情報があるかどうか調べる。

↓

4-a.HIDE 情報が存在しない場合、問い合わせを許可する。 DNS サーバに問い合わせた結果をクライアントに返す。

4-b.HIDE 情報が存在する場合、キーワードが一致するかどうかをチェックする。

↓

5.keyword が一致した場合、問い合わせを許可する。 DNS サーバに問い合わせた応答をクライアントに返す。

↓

6. 一致しなかった場合、クライアントからの DNS メッセージをクライアントに返す。

上には大まかな流れを説明した。以下にそれぞれ補足を加える。

- 問い合わせの名前、問い合わせタイプ、キーワードを取り出す

  受け取ったクエリの問い合わせ部より、 QNAME、 QTYPE を、付加情報部よりキーワードを取り出す。付加情報部がない場合は、 keyword を空白とする。

- 問い合わせの名前、タイプをもとに、 DNS サーバに HIDE 情報があるかどうか調べる

  名前、タイプをもとに独自にクエリを作り、 DNS サーバに問い合わせを行う。 HIDE 情報がある場合は、タイプとキーワードを取り出す。

- キーワードが一致するかどうかをチェックする

  クライアントより取得のキーワードと HIDE 情報のキーワードを文字列比較する。

- DNS サーバに問い合わせた結果を CLIENT に返す

  クライアントのクエリを DNS サーバに転送し、その結果をクライアントに返す。

この一連の動作をまとめたものが図 3.3である。

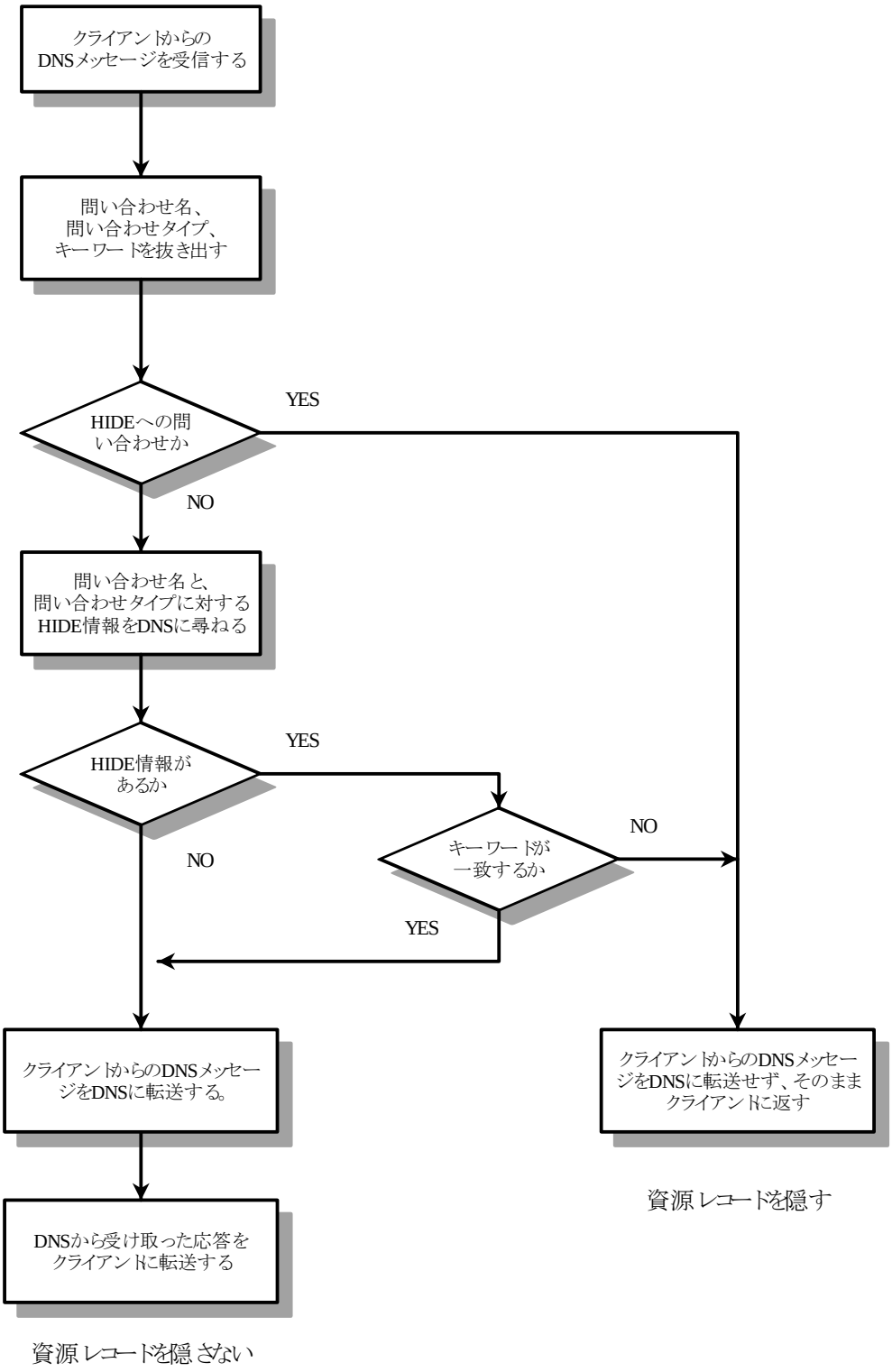


図 3.3: DNSPROXY の動作

第 4 章

実験の概要

4.1 実験の目的

第 3 章で提案した方法で、資源レコードを隠すあるいは隠さないことが可能であることを、プログラムを作成することで証明する。ここで用いたプログラムのソースコードを付録 A に載せた。

4.2 実験の環境

プログラムの作成、実行環境は表 4.1のようになっている。 DNSPROXY は DNS と同じマシン上の 5353 ポートで動作させた。

表 4.1: プログラム開発と実行環境

OS	FreeBSD 5.2.1-RELEASE
コンパイラ	gcc version 3.3.3
DNS サーバ	bind 9.3.0beta4

4.3 実験の内容

DNS に記述する HIDE 情報、クライアントに付加するキーワードをそれぞれ変更して、動作を見る。クライアントの実行コマンドは

```
./dnsclient 問い合わせ名 問い合わせタイプ キーワード
```

のようになっている。

4.3.1 実験 1

まずは HIDE 情報がない場合を行う。この場合、すべてのクエリが許可される。

実験のために登録した資源レコード

```
@ORIGIN 2.9.3.0.1.5.0.0.1.8.e164.jp

1.4 IN NAPTR 100 200 "u" "E2U+msg"
                    "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .
    IN NAPTR 100 300 "u" "E2U+sip"
                    "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .
```

このように NAPTR 資源レコードを 2 つ登録した。

DNSCLIENT の実行例：キーワードは任意

```
./dnsclient 1.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 key
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47748
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0
;;      1.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN


1.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 300 "u" "E2U+sip"
                    "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .

1.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 200 "u" "E2U+msg"
                    "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .


2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 1.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 35 cli_key: key                 // タイプ、キーワード取得
hide_type: 0 dns_key:                     //HIDE 情報が存在しない
To DNS 133.9.68.162 ok!                   //DNS へ問い合わせ
From DNS 133.9.68.162 ok!                 // 問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q             // クライアントへ送信 ：終了
```

このように、 HIDE が存在しない場合は、クエリのキーワードが存在する場合それが何であっても通常の DNS と同じ応答を返す。キーワードが存在しない場合も同様である。

4.3.2 実験 2

特定の 1 つの資源レコードに HIDE を情報を付加する場合を考える。ここでは NAPTR に HIDE 情報を持たせた。この場合、 NAPTR 以外のクエリはすべて許可される。また NAPTR を表示させるためには、キーワードが必要となる。

実験のために登録した資源レコード

```
@ORIGIN 2.9.3.0.1.5.0.0.1.8.e164.jp

2.4 IN NAPTR 100 200 "u" "E2U+msg"
                     "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .
    IN NAPTR 100 300 "u" "E2U+sip"
                     "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .
    IN TXT "Test TXT RR"
    IN TYPE99 \# 9 23 7465737470617373
```

NAPTR 資源レコードを 2 つ、 TXT 資源レコードを 1 つ、 HIDE 資源レコードを 1 つ登録した。この HIDE 情報は NAPTR に対し、キーワード ”testpass” という意味である。

DNSCLIENT の実行例： NAPTR キーワードが一致

```
./dnsclient 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 testpass
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26716
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0
;;      1.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN


2.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 300 "u" "E2U+sip"
                     "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .

2.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 200 "u" "E2U+msg"
                     "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .


2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 1.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 35 cli_key: testpass            // タイプ、キーワード取得
hide_type: 35 dns_key: testpass           // キーワードが一致
To DNS 133.9.68.162 ok!                   //DNS へ問い合わせ
From DNS 133.9.68.162 ok!                 // 問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q             // クライアントへ送信 ：終了
```

このように、正しいキーワードの場合、 NAPTR 資源レコードの情報を得ることができる。

DNSCLIENT の実行例:NAPTR キーワードが不一致

```
./dnsclient 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 password
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54895
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      2.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN

.                       0S IN TXT       "password"
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 35 cli_key: password            // タイプ、キーワード取得
hide_type: 35 dns_key: testpass
KEY ERR :q                                // キーワードが不一致 : 終了
```

このように、キーワードが不一致の場合、クライアントは自ら送信したクエリを得るのみである。

DNSCLIENT の実行例:TXT キーワード任意

```
./dnsclient 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp 16 key
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59137
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;;      2.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = TXT, class = IN

2.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN TXT  "Test TXT RR"

2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 16 cli_key: key                 // 問い合わせタイプ、キーワード取得
hide_type: 0 dns_key:                     //HIDE 情報が存在しない
To DNS 133.9.68.162 ok!                   //DNS へ問い合わせ
From DNS 133.9.68.162 ok!                 // 問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q             // クライアントへ送信  : 終了
```

このように、 HIDE 情報は指定したタイプ以外には及ばない。よって TXT に対する問い合わせは正しく返される。

DNSCLIENT の実行例： HIDE 情報に対するクエリ

```
./dnsclient 2.4.2.9.3.0.1.5.0.0.1.8.e164.jp 99 key
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56456
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      2.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = 99, class = IN
.                        0S IN TXT       "key"
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!          // クエリ受信
QUERY TYPE ERR :q                     // クエリタイプが HIDE 情報だったので終了
```

このように、 HIDE 情報へのクエリは許可されない。よって HIDE 情報のキーワードを得ることはできない。

4.3.3 実験 3

複数の資源レコードに HIDE 情報を付加する場合を考える。 NAPTR と TXT にそれぞれ HIDE 情報を持たせた場合。この場合、 NAPTR、 TXT 以外のクエリはすべて許可される。 NAPTR、 TXT を表示させるためには、それぞれキーワードが必要となる。

実験のために登録した資源レコード

```
@ORIGIN 2.9.3.0.1.5.0.0.1.8.e164.jp

3.4 IN NAPTR 100 200 "u" "E2U+msg"
                    "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .
    IN NAPTR 100 300 "u" "E2U+sip"
                    "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .
    IN TXT "Test TXT RR"
    IN TYPE99 \# 9 23 6162636465666768
    IN TYPE99 \# 9 10 696a6b6c6d6e6f70
```

NAPTR 資源レコードを 2 つ、 TXT 資源レコードを 1 つ、 HIDE 資源レコードを 2 つ登録した。この HIDE 情報はそれぞれ、 NAPTR に対しキーワード ”abcdefgh”、 TXT に対しキーワード ”ijklmnop” という意味である。

DNSCLIENT の実行例:NAPTR キーワードが一致

```
./dnsclient 3.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 abcdefgh
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29900
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0
;;      3.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN


3.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 300 "u"
                    "E2U+sip" "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .
3.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 200 "u"
                    "E2U+msg" "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .


2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 3.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 35 cli_key: abcdefgh            // タイプ、キーワード取得
hide_type: 35 dns_key: abcdefgh           // キーワードが一致
To DNS 133.9.68.162 ok!                   //DNS へ問い合わせ
From DNS 133.9.68.162 ok!                 // 問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q             // クライアントへ送信 ：終了
```

キーワードが一致したため、正しい応答を返す。

DNSCLIENT の実行例:NAPTR キーワードが不一致

```
./dnsclient 3.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 ijklmnop
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8899
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      3.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN
.                       0S IN TXT       "ijklmnop"
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
domain: 3.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 35 cli_key: ijklmnop            // タイプ、キーワード取得
hide_type: 35 dns_key: abcdefgh           // キーワードが不一致
KEY ERR :q                                // 終了
```

当然ながら TXT に設定されたキーワードで NAPTR を見ることはできない。このように HIDE 情報はそれぞれの資源レコードに対し独立に設定することができる。

TXT に対しても表示結果に違いがあるのみで動作は NAPTR と変わらないので割愛する。

4.3.4 実験 4

すべての資源レコードに HIDE 情報を持たせる場合を考える。 HIDETYPE に 255 を指定した場合。この場合すべてのクエリについてキーワードが必要となる。

実験のために登録した資源レコード

```
@ORIGIN 2.9.3.0.1.5.0.0.1.8.e164.jp

4.4 IN NAPTR 100 200 "u" "E2U+msg"
                    "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .
    IN NAPTR 100 300 "u" "E2U+sip"
                    "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .
    IN TXT "Test TXT RR"
    IN TYPE99 \# 9 ff 7465737470617373
```

NAPTR 資源レコードを 2 つ、 TXT 資源レコードを 1 つ、 HIDE 資源レコードを 1 つ登録した。この HIDE 情報はすべての資源レコードに対しキーワード ”testpass” という意味である。

DNSCLIENT の実行例:NAPTR キーワードが一致

```
 ./dnsclient 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 testpass
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15785
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 0
;;      4.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN


4.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 200 "u" "E2U+msg"
                    "!^.*$!mailto:tati@goto.info.waseda.ac.jp!" .
4.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NAPTR  100 300 "u" "E2U+sip"
                    "!^.*$!sip:tati@sip.goto.info.waseda.ac.jp!" .


2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!              // クエリ受信
```

```
domain: 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp //問い合わせ名取得
cli_type: 35 cli_key: testpass          //タイプ、キーワード取得
hide_type: -1 dns_key: testpass         //キーワードが一致
To DNS 133.9.68.162 ok!                 //DNS へ問い合わせ
From DNS 133.9.68.162 ok!               //問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q           //クライアントへ送信 ：終了
```

キーワードが一致したため、正しい応答を返す。

DNSCLIENT の実行例:NAPTR キーワードが不一致

```
./dnsclient 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp 35 password
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42226
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      4.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = NAPTR, class = IN
.                       0S IN TXT       "password"
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!            //クエリ受信
domain: 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp //問い合わせ名取得
cli_type: 35 cli_key: password          //タイプ、キーワード取得
hide_type: -1 dns_key: testpass         //キーワードが不一致
KEY ERR :q                              //終了
```

キーワードが不一致のため、正しい応答は返らない。

DNSCLIENT の実行例:TXT キーワードが一致

```
./dnsclient 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp 16 testpass
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25377
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 0
;;      4.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = TXT, class = IN

4.4.2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN TXT  "Test TXT RR"

2.9.3.0.1.5.0.0.1.8.e164.jp.  5M IN NS  dnssec-co.goto.info.waseda.ac.jp.
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!            //クエリ受信
```

```
domain: 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 16 cli_key: testpass          // タイプ、キーワード取得
hide_type: -1 dns_key: testpass         // キーワードが一致
To DNS 133.9.68.162 ok!                 //DNS へ問い合わせ
From DNS 133.9.68.162 ok!               // 問い合わせ応答取得
To CLIENT 133.9.68.163 ok! :q           // クライアントへ送信 ：終了
```

TXT に対しても同じキーワードチェックを行う。

DNSCLIENT の実行例:TXT キーワードが不一致

```
./dnsclient 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp 16 password
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29218
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;;      4.4.2.9.3.0.1.5.0.0.1.8.e164.jp, type = TXT, class = IN
.                       0S IN TXT       "password"
```

DNSPROXY の処理

```
From CLIENT 133.9.68.163 ok!            // クエリ受信
domain: 4.4.2.9.3.0.1.5.0.0.1.8.e164.jp // 問い合わせ名取得
cli_type: 16 cli_key: password          // タイプ、キーワード取得
hide_type: -1 dns_key: testpass         // キーワードが不一致
KEY ERR :q                              // 終了
```

キーワードが不一致のため、正しい応答は返らない。

このように HIDE 情報に 255 （ANY）と記述すると、すべての資源レコードに対しキーワードを要求する。

第 5 章

まとめ

5.1 結論

本研究では新しい資源レコードを定義することにより、任意の資源レコードを隠す方法を考案した。新しい資源レコードとして HIDE を利用している。さらに考案した方法が正しいことをプログラムを作成することで示した。従って第一の目的であった ENUM を利用する際の NAPTR 情報の保護という点は実現できたと言える。現在個人情報として隠したい資源レコードは NAPTR 以外には特に存在していない。さらに、今回 NAPTR 以外の任意の資源レコードを隠すことを可能としたことで、将来 DNS に新しい機能が追加された場合にも対応できる。また、既に存在し、公開されている資源レコードについても、公開の必要性が見直されることがあるかもしれない。そういう場合に柔軟に対応できる今回のこのシステムは非常に有用である。

5.2 今後の課題

今回提案したシステムは少数の限られた空間の DNS への実装しか行っていない。実際に DNS に今回の HIDE を実装するとした場合には、現在稼動している無数の DNS に与える影響について考慮しなければならない。また、今回キーワードが一致しない場合の処理として、送られてきたメッセージをそのまま送り返しているが、その場合の処理をどうするかという問題も残る。どのような場面で利用されるかを十分に検討し、 DNSPROXY にあたる処理を改善する必要がある。

謝辞

本学士論文の作成にあたり日頃より御指導を頂いた早稲田大学理工学部の後藤滋樹教授に深く感謝致します。そして研究を進めるにあたり貴重なアドバイスを頂いた NTT コミュニケーションズの中崎雄祐氏、ならびに後藤研究室の杉田隆俊氏、日頃より助言を頂いた後藤研究室の諸氏に感謝いたします。

参考文献

[1] Paul Albitz, Cricket Liu 著, 高田広章, 小島育夫 監訳, 小舘光正 訳『DNS & BIND 第 4 版』, オライリー・ジャパン, 2002.

[2] Nicolai Langfeldt 著, でびあんぐる 監訳, 竹内里佳 訳『DNS & BIND 入門』, オーム社, 2001.

[3] W・リチャード・スティーヴンス 著, 井上尚司 監訳, 橘康雄 訳『詳解 TCP/IP Vol.1 プロトコル』, ピアソン・エデュケーション, 2000.

[4] 小俣光之『C 言語による TCP/IP セキュリティプログラミング』, ピアソン・エデュケーション, 2002.

[5] 村山公保『基礎からわかる TCP/IP ネットワーク実験プログラミング』, オーム社,2001.

[6] 林晴比古『改定新 C 言語入門シニア編』, ソフトバンクパブリッシング, 1991.

[7] ENUM 研究グループ, ENUM 研究グループ報告書, 社団法人日本ネットワークインフォメーションセンター, 2003.

[8] P. Mockapetris, RFC1034, DOMAIN NAMES - CONCEPTS AND FACILITIES, 1987.

[9] P. Mockapetris, RFC1035, DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION, 1987.

[10] P. Faltstrom, RFC3761, The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM), 2004.

[11] A. Gustafsson, RFC3597, Handling of Unknown DNS Resource Record (RR) Types, 2003.

[12] 中崎雄祐『プライバシーを考慮した ENUM システムの提案と実装』, 早稲田大学理工学部 2003 年度修士論文, 2004.

[13] 杉田隆俊『ENUM を応用した三者間の通信法』, 早稲田大学理工学部 2003 年度卒業論文, 2004.

[14] 横澤一岐『ENUM における個人情報の保護システム』, 早稲田大学理工学部 2004 年度卒業論文, 2005.

付録 A

作成したプログラム

A.1 dnsproxy.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <arpa/nameser.h>
#include <resolv.h>

#define PORTNUM 5353    //port
#define BUFSIZE 65507    //buffer size


void gethide(char domain[],int q_type, int cq_type);    //prototype

char dns_key[256];    //type99 key
int q_type; // 問い合わせ query_type
int h_type; //hide query_type
char hide_type[2];
```

```
//main
main(){

    struct sockaddr_in proxy;
    struct sockaddr_in server;
    struct sockaddr_in client;
    unsigned long svr_ip;
    int s_cp, s_ps;    //ディスクリプタ
    int ser_len, cli_len;
    int que_len, res_len;

    int port = PORTNUM;
    char que_buf[BUFSIZE];
    char res_buf[BUFSIZE];

    ns_msg handle;
    ns_rr rr;
    u_char *cp;

    char domain[256];
    char cli_key[256];

    int i,j;



    while(1){

        //UDP でソケットをオープン
        if ((s_cp = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
            perror("socket");
            exit(-1);
        }
        //アドレスを設定
        memset((char *)&proxy, 0, sizeof(proxy));
```

```
proxy.sin_family = AF_INET;
proxy.sin_addr.s_addr = htonl(INADDR_ANY);
proxy.sin_port = htons(port);
if (bind(s_cp,(struct sockaddr *) &proxy, sizeof(proxy)) < 0){
    perror("bind");
    exit(-1);
}
// クライアントからクエリを受信
cli_len = sizeof(client);
while ((que_len = recvfrom(s_cp, que_buf, BUFSIZE-1, 0, (struct sockaddr *)
    printf("From CLIENT %s ok!\n",inet_ntoa(client.sin_addr));

// 初期化
dns_key[0]='\0'; cli_key[0]='\0';hide_type[0]='\0';
h_type=0;

// クライアントからのクエリの処理
ns_initparse( que_buf, que_len, &handle);// ネームサーバライブラリルーチン
を呼び出す
ns_parserr(&handle, ns_s_qd, 0, &rr);// 問い合わせ部を rr に展開

q_type=ns_rr_type(rr);
if (q_type != 99){//hide 情報へのクエリでないなら

// 問い合わせドメインを取得
cp = (u_char *)ns_rr_name(rr);
for(i=0;i<=strlen(cp);i++){
    domain[i] = cp[i];
}
    domain[i-1]='\0';

//add section があれば keyword を取り出す
if(ns_parserr(&handle, ns_s_ar, 0, &rr)==0){
    cp = (u_char *)ns_rr_rdata(rr);
```

```
            for(i=1; i<=cp[0]; i++){
                cli_key[i-1] = cp[i];
            }
            cli_key[i-1]='\0';
        }

        printf("domain: %s \n", domain);// 問い合わせ名表示
        printf("cli_type: %d cli_key: %s \n", q_type,cli_key);// クライアントキー
ワード表示

        //domain の type99 情報取得
        gethide(domain,99,q_type);

        printf("hide_type: %d hide_key: %s \n", h_type,dns_key);//DNS キーワード
表示

        if((q_type==h_type&&strcmp(cli_key,dns_key)!=0)||(h_type==-1&&strcmp(cli_ke

        //send client Error
        printf("KEY ERR :q\n");

        if (sendto(s_cp, que_buf, que_len, 0, (struct sockaddr *) &client, sizeof(c
            break;
        }

        }else{

        //UDP でソケットをオープン
        if ((s_ps = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
            perror("socket");
            exit(-1);
        }
        // アドレスを設定
        proxy.sin_port = htons(0);
```

```
if (bind(s_ps,(struct sockaddr *) &proxy, sizeof(proxy)) < 0){
    perror("bind");
    exit(-1);
}
//DNS にクエリを送信する
memset((char *)&server, 0, sizeof(server));
svr_ip = inet_addr("162.68.9.133");
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(svr_ip);
server.sin_port = htons(53);
if (sendto(s_ps, que_buf, que_len, 0, (struct sockaddr *) &server, sizeof(s
    break;
}
printf("To DNS %s ok!\n",inet_ntoa(server.sin_addr));

//DNS からクエリを受信
ser_len = sizeof(server);
res_len = recvfrom(s_ps, res_buf, BUFSIZE-1, 0, (struct sockaddr *) &server
if ( res_len < 0){
    break;
}

printf("From DNS %s ok!\n",inet_ntoa(server.sin_addr));



//send client
if (sendto(s_cp, res_buf, res_len, 0, (struct sockaddr *) &client, sizeof(c
    break;
}
printf("To CLIENT %s ok! :q\n",inet_ntoa(client.sin_addr));

}

}else{// クライアントからのクエリタイプが 99
```

```
        printf("QUERY TYPE ERR :q\n");
        if (sendto(s_cp, que_buf, que_len, 0, (struct sockaddr *) &client, sizeof(c
            break;
        }
        }

        }
      close(s_ps);close(s_cp);
    }
}


//type99 取得
void gethide(char domain[],int hq_type,int cq_type){

    struct sockaddr_in dnsserver; //DNS サーバのソケットアドレス構造体
    struct sockaddr_in dnsclient; // クライアントのソケットアドレス構造体
    unsigned long dnsserver_ip; // サーバーの IP アドレス
    int dnsport = 53;
    int seradd_len;
    int dnsque_len, dnsres_len;
    int soc; // ソケットディスクリプタ
    u_char dnsquery[NS_PACKETSZ];
    u_char response[BUFSIZE];
    ns_msg dnshandle; // 応答メッセージ用ハンドル
    ns_rr dnsrr; //RR (リソースレコード)
    u_char *dnscp;


    //socket
    if ((soc = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
        perror("socket");
        exit(-1);
    }
```

```
//bind
memset((char *)&dnsclient, 0, sizeof(dnsclient)); // 初期化
dnsclient.sin_family = AF_INET;
dnsclient.sin_addr.s_addr = htonl(INADDR_ANY);
dnsclient.sin_port = htons(0);
if (bind(soc,(struct sockaddr *) &dnsclient, sizeof(dnsclient)) < 0){
    perror("bind");
    exit(-1);
}

//DNS パケット作成
dnsque_len = res_mkquery(
        ns_o_query,
        domain,
        ns_c_in,
        hq_type,
        (u_char *)NULL,
        0,
        (u_char *)NULL,
        (u_char *)&dnsquery,
        sizeof(dnsquery) );

//sendto
memset((char *)&dnsserver, 0, sizeof(dnsserver)); // 初期化
dnsserver_ip = inet_addr("162.68.9.133");
dnsserver.sin_family = AF_INET;
dnsserver.sin_addr.s_addr = htonl(dnsserver_ip);
dnsserver.sin_port = htons(dnsport);
if (sendto(soc, dnsquery, dnsque_len, 0, (struct sockaddr *) &dnsserver, sizeof
    printf("send err !");
}

//recv
```

```
seradd_len = sizeof(dnsserver);
dnsres_len = recvfrom(soc, response, BUFSIZE-1, 0, (struct sockaddr *) &dnsserv
if ( dnsres_len < 0){
    exit;
}


//パケット解析

int k,p;
int answer_num;
int tmp_type;

ns_initparse( response, dnsres_len, &dnshandle);

    answer_num=ns_msg_count( dnshandle, ns_s_an);

    if(answer_num!=0){
    for(p=0;p<=answer_num;p++){
    ns_parserr(&dnshandle, ns_s_an, p, &dnsrr);
        dnscp = (u_char *)ns_rr_rdata(dnsrr)-1;
        //hide RR のクエリタイプ取得
        hide_type[0]=ns_get16(dnscp);
        tmp_type=hide_type[0];
        dnscp++;

        if(tmp_type==cq_type||tmp_type==-1){
        //hide RR のキーワード取得
        for(k=0; k<8; k++){
        dns_key[k]=ns_get16(dnscp);
        dnscp++;
        }
        dns_key[k]='\0';
        h_type=tmp_type;
```

```
            }
        }
        }

}
```

A.2 dnsclient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <arpa/nameser.h>
#include <resolv.h>


#define BUFSIZE 65507    //buffer size

int main(int argc, char *argv[]){

    struct sockaddr_in server;
    struct sockaddr_in client;
    int soc; //ディスクリプタ
    int seradd_len;
    int que_len, add_len, res_len;

    u_char query[NS_PACKETSZ];
    u_char add[NS_PACKETSZ];
    u_char response[BUFSIZE];
    ns_msg handle;
```

```
ns_rr rr;
u_char *cp;

unsigned long server_ip; //問い合わせ先 IP アドレス
int port = 5353;

char domain[32];//問い合わせ内容
char *kw;//付加するキーワード
int q_type;//クエリタイプ

//コマンド引数取得
if(argc!=4){
    printf("useage:naptrpluskw domain querytype keyword\n");
    exit(EXIT_FAILURE);
}

strcpy(domain,argv[1]);//ドメイン
q_type=atoi(argv[2]);//クエリタイプ
kw = argv[3];  //キーワード

//UDP でソケットをオープン
if ((soc = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
    perror("socket");
    exit(-1);
}

//アドレスを設定
memset((char *)&client, 0, sizeof(client)); //初期化
client.sin_family = AF_INET;
client.sin_addr.s_addr = htonl(INADDR_ANY);
client.sin_port = htons(0);
if (bind(soc,(struct sockaddr *) &client, sizeof(client)) < 0){
    perror("bind");
    exit(-1);
```

```
}

// クエリ作成
que_len = res_mkquery(
        ns_o_query,
        domain,
        ns_c_in,
        q_type,
        (u_char *)NULL,
        0,
        (u_char *)NULL,
        (u_char *)&query,
        sizeof(query) );

// 追加部分のクエリ
add_len = res_mkquery(
        ns_o_query,
        "",  // ドメイン名部分は空白（ルート）
        ns_c_in,
        ns_t_txt,
        (u_char *)NULL,
        0,
        (u_char *)NULL,
        (u_char *)&add,
        sizeof(add) );

// キーワード付加パケットを作成
query[11] = query[11] + 1;  // 追加情報
memcpy( &query[que_len],&add[12], add_len-12);  // ドメイン名、タイプ、クラス
ns_put32( 0, &query[que_len+add_len-12]);  //ttl
ns_put16( strlen(kw)+1, &query[que_len+add_len-12+4]);  //RR データ長
query[que_len+add_len-12+4+2] = strlen(kw);  // 文字列の頭の長さを表す 1bit
memcpy( &query[que_len+add_len-12+4+2+1], kw, strlen(kw)); // キーワード
```

```
    // サーバーに送信
    memset((char *)&server, 0, sizeof(server));
    server_ip = inet_addr("162.68.9.133");
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(server_ip);
    server.sin_port = htons(port);
    if (sendto(soc, query, que_len+add_len-12+4+2+1+strlen(kw), 0, (struct sockaddr
        printf("send err !");
    }

    // サーバーから受信
    seradd_len = sizeof(server);
    res_len = recvfrom(soc, response, BUFSIZE-1, 0, (struct sockaddr *) &server, &s
    if ( res_len < 0){
        exit;
    }

    // 受け取った DNS メッセージの中身を可読形式で表示する
    fp_nquery(response,res_len,stderr);

}
```