情報化された組織のセキュリティマネジメント WG 成果報告書

第 3 部

「セキュリティ向上を主眼とする DNS 設定ガイド」

2012 年 5 月 11 日

サイエンティフィック・システム研究会

情報化された組織のセキュリティマネジメント WG

B グループ

WG メンバー

■全体

			氏名	機関/所属(2012 年 3 月 31 日現在)
会員	担当幹事		長谷川　明生	中京大学
	推進委員	(まとめ役)	山守　一徳	三重大学
			吉田　和幸	大分大学
			笠原　義晃	九州大学
			武蔵　泰雄	熊本大学
			湯浅　富久子	高エネルギー加速器研究機構
			鈴木　聡	高エネルギー加速器研究機構
			只木　進一	佐賀大学
			西村　浩二	広島大学
賛助会員(富士通)	推進委員	(まとめ役)	吉田　真和	富士通(株)
			山下　眞一郎	富士通(株)
			飯島　敏治	富士通(株)
			南場　進	富士通(株)
			山路　光昭	富士通(株)
			櫻井　秀志	富士通(株)
			田口　雅晴	(株)富士通九州システムズ
			須永　知之	(株)富士通ソーシアルサイエンスラボラトリ
	オブザーバ		山口　正雄	富士通(株)

■グループ別

グループ	テーマ	メンバー
A グループ	BCP 情報漏えい	只木 進一 [班長]、 湯浅 富久子、 西村 浩二、 山守 一徳、 山下 眞一郎、山路 光昭、櫻井 秀志、須永 知之
B グループ	DNS	鈴木 聡 [班長]、 吉田 和幸、 笠原 義晃、 武蔵 泰雄、 長谷川 明生、 吉田 真和、 飯島 敏治、 南場 進、 田口 雅晴

目次

第３部　セキュリティ向上を主眼とするDNS設定ガイド

■図表について

冊子版は白黒のため見づらい図表がありますが、SS研WebサイトのPDF版はカラーですので、詳細はそちらをご覧ください。

http://www.ssken.gr.jp/MAINSITE/ →「資料ダウンロード」→「WG関連資料」

■登録商標について

1.　はじめに

1.1.　DNS サーバの安全性について

昔から DNS の脆弱性に対する指摘はあり、それを利用してユーザを騙すことが可能なこともわかっていました。しかし、騙すための計算量・ネットワーク帯域などのコストが高いため、昔は「そこまでして悪さをする人はかなりの物好き」というような扱われ方でした。今ではインターネットが商取引で使われるようになり、詐欺の片棒を担ぐ手段の一つとなってきています。クロスサイトスクリプティングとの合わせ技でフィッシングサイトに誘導する手段の一つとして使われるので非常に危険です。

※参考:「インターネット 10 分講座:DNS キャッシュポイズニング」

http://www.nic.ad.jp/ja/newsletter/No40/0800.html

この報告書は主に DNS サーバを自分で運用している管理者の方に注意喚起を主眼として作成しました。前任者の過去の設定を引き継いで運用している方は、昨今のセキュリティ情勢の変化によって、従来と同じ設定では危険であることを認識していただきたいと思います。この報告書を作成している間にもセキュリティ上の問題が多数発生しました。問題の多くは身内用の DNS サーバが外からもアクセス可能になっている場合に重大な問題となります。自分の DNS サーバがそのようになっているかどうかは、御自宅から ISP のネットワークを通じて自組織のネームサーバに

dig -t a example.com. @「自組織のネームサーバ」

を発行して返事をするか確認したり、WWW 上のネームサーバ診断サービスなどを使うことができます。(たとえば http://www.e-ontap.com/internet/check など)。

DNS サーバの設定ミスは大量の苦情を発生させるため、前任者の設定を変更するのはかなり気が重い作業ですが、フィッシングに引っかかった場合はもっと大変なことになります。そもそも DNS はプロトコル上の脆弱性があるのですが、運用によってその穴を完全にではないにしても常に小さくとどめておくことが重要です。

典型的な例をもとに、どのような問題と対策があるかを以降で説明していきます。

1.2.　騙す手口

たとえばホスト名や IP アドレスを導出する際に、正規の DNS サーバからの返事が質問者に到着する前に偽装したパケットを質問者に送り返すことで騙すことが可能です。これが安易にできては危険なので、パケット内の 16 ビット乱数値(Transaction ID)とポート番号のランダム性によって簡単に偽装パケットを製造できないようにしています。BIND のバージョンによってはこの乱数に偏りがあったり、ポート番号が固定もしくは推測しやすい値になっていたりするため、次の質問パケットが使用するであろう乱数値とポート番号をある程度絞ることができます。偽装パケットは複数投げてもよいのでこの範囲がある程度狭ければ危険は現実のものとなります。これまで DNS の検証と言えば正引きと逆引きの検査をつきあわせて比較する程度でしたが、このような方法だけでは騙されていることはわかりません。また、対象ドメインの NS レコードを騙すことに成功すればそのあとしばらくの間は問い合わせが正規の DNS サーバではなく偽装サーバに送られることになるので、さらなる被害を招きます。

2012 年 2 月にはドメイン名が決済不能などの理由により失効してしまったときに、引けなくなるはずのそのドメイン名を横取りする攻撃手法が発表されるなど、ドメイン名を詐称する手口も日々進化しています。単に BIND のバージョンアップだけで問題が解決しないこともままあります。

1.3. どのような設定が求められるのか

DNS サーバには 2 つの役割があり、あるドメインが自分の情報を公開するための権威サーバと、ユーザからの質問に応じて外部の権威サーバを渡り歩いて検索するキャッシュサーバに大別できます。権威サーバとキャッシュサーバを歴史的経緯から同一ホストで動かしている例がかなり見受けられますが、「前任者がこういう設定をしていったから」式の設定では安全性を担保するのが難しく、安全面を考えれば分離した方が運用コストを低く抑えることができます。

BIND は最も広く使われている DNS サーバプログラムであり、両方の役割をこなすことができます。その一方でプログラムが肥大して複雑さが増しているため、セキュリティホールが発生しやすい傾向があります。分離して運用する場合、権威サーバ・キャッシュサーバのそれぞれに特化した DNS サーバプログラムがありますので、それらを利用することも運用コストを下げる手段として有効です。

1.4. 権威サーバとキャッシュサーバ

権威サーバ（Authoritative Server）

上位組織の権威サーバから指名されており、担当部分の内容について問い合わせに返答する役割を果たします。担当空間以外の内容については答える必要はなく、拒否できます。担当しているドメインのさらに下位ドメインについても直接返答する必要はなく、どの DNS サーバに聞けばいいか移譲先を答えるだけで十分です。

BIND での master と slave がこれにあたります。

キャッシュサーバ（Cache Server）

DNS サーバを動かしていないホストやプログラムからの問い合わせを一手に引き受けるための DNS サーバです。権威サーバからのたらいまわし情報を元に次々に権威サーバをあたって問い合わせに対応する返事を作成します。

フォワーダ （Forwarder）

BIND での「Forwarder」を設定した場合、用途としてはキャッシュサーバなのですが、自分で再帰問合せを行わず、他のキャッシュサーバに問合せを 1 回投げつけて返事をもらうだけで、たらいまわしの処理を自分で行いません。安易にキャッシュサーバを増やすことができますが、押しつけているキャッシュサーバの内容がおかしくなると巻き添えになります。ゾーンを限定してプライベートなドメインの DNS サーバを直接参照したいときなどにも使用されます。

権威サーバは外部からの問い合わせに答える必要がありますが、キャッシュサーバは外部からの問い合わせに答える必要はありません。

権威サーバの返事はホストの追加などがない限り定型的でメモリの消費量も変動しませんが、キャッシュサーバの場合はクライアントからの問い合わせによってはメモリを圧迫することになります。

1.5. DNS の名前解決の仕組み

DNS はデータ分散システムといえます。1 つの DNS サーバが、全体ではなく一部についてだけ知っていればよく、知らないことについて問い合わせを受けた場合他の DNS サーバに聞いて解決すればよいという戦略です。ただし、悪意のある DNS サーバからのでたらめ情報を真に受けるようでは困るので、DNS サーバ毎に担当する名前空間を決めることにします。担当範囲には上下関係があり、自分の担当範囲の一部に限って他の DNS サーバに権限を委譲することができます。上位の権威サーバから委譲を受けている DNS サーバが権威サーバです。

図 1 のように、jp 全体のネームサーバは jp の配下である co.jp や example.jp を直接自分で面倒を見ることはせず、それぞれの範囲を定めて権威サーバに名前解決を任せることにします。jp 全体の権威サーバに委譲してある空間に含まれる aaa.example.co.jp について問い合わせても答えを教えてはくれず、直接の移譲先を答えるだけです（図 2）。

この委譲情報を元にたらいまわしで解決します。最上位のドメインは「.」であり、これを担当している権威サーバはルートサーバと呼ばれます。ルートサーバは「com」「net」「org」「edu」「jp」など、最も右端に存在するドメインについて、誰に聞けばいいかだけを

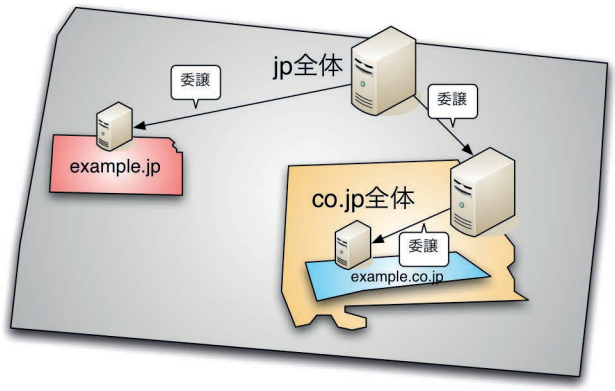


図 1

jp 全体の権威サーバはその下位ドメインである example.jp や co.jp を別のサーバに委譲しており、co.jp のサーバはさらに example.co.jp を別のサーバに委譲している。

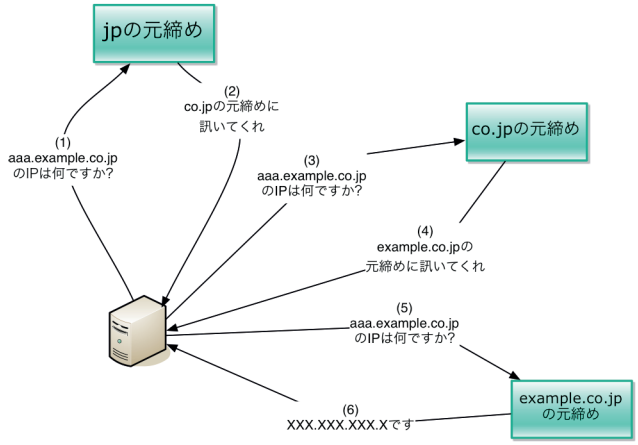


図 2

jp の元締めに aaa.example.co.jp について質問しても co.jp の元締め情報を教えて貰えるだけ。

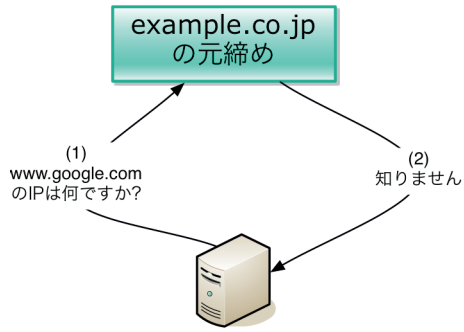


図 3

自分の担当範囲でなければ移譲先を答える必要もない。

答えます。例えが悪いですが、申請書類を提出しに出かけてみると最終的な提出先がわからず、次々に「ここではないのでよそへ行ってみよ」式に行き先を告げられるような状態です。委譲されていない DNS サーバには本来外部から問合せがくる筈はないので返答をする必要はなく、委譲されている権威サーバであっても委譲範囲外について返答する必要はありません(図 3)。

1.6. クライアントの動作について

DNS クライアントとは、DNS を利用して名前解決したいプログラムやホストを指します。上に述べたように、本来は権威サーバを上位から手繰ってたらいまわしにあうことになりますが、あらゆるクライアントがそれぞれたらいまわしで右往左往するのは CPU やネットワーク帯域の無駄なので、いわばワンストップ窓口用プログラムを別に用意し、それが質問者の代わりに駆け巡ってきます。これがキャッシュサーバの役割です。せっかくなので、1 回検索した情報には寿命を持たせ、一定時間控えておくことにしましょう。この寿命(TTL)は権威サーバ側の情報で決定することになっています。

すべての DNS サーバがワンストップ窓口としての動作をしているわけではありません。DNS の問合せパケットには様々なフラグがありますが、クライアントが DNS サーバ側に「DNS サーバ側でたらいまわし解決して欲しい」と主張するための RecursiveDesire フラグがあります。ワンストップ窓口的動作を期待している場合 RecursiveDesire フラグを立てて問合せ(再帰問合せ)を行いますが、質問された DNS サーバはこのフラグが立っている問合せには答えないように設定されているかもしれません。

2. 典型的な攻撃方法

通常、DNS の応答は UDP で行われているため、本物の DNS サーバが返答するよりも早く嘘パケットを生成して返すことができれば嘘が採用されてしまいます。キャッシュサーバに嘘が格納されてしまうと TTL が切れるまでの間、嘘がばらまかれることになります。これを「キャッシュ汚染」と呼びます。本物の DNS サーバに対して DoS 攻撃を仕掛けるなどして負荷を上昇させ、応答性能を悪化させることで嘘が採用されやすくするなどの手口もあり得ます。

またキャッシュサーバはメモリ上に情報を記録しておくため、あまりにも問合せが殺到するとメモリ上に全部データをおいておくことができなくなり、一部のデータがキャッシュから消去されます。すると TTL によってまだメモリに残っているべきデータであっても再度問合せをしなくてはなりません。するとキャッシュ汚染攻撃の機会が増えることになります。

小型 NAT ルータなど、DNS キャッシュサーバ機能を持つアプライアンス機器の中には TTL が短いレコードに対しても長い TTL が指定されているかのように取り扱うものがあります。このような実装もキャッシュ汚染の被害に遭いやすいことに注意が必要です。

ファイヤーウォールのルールが増えることを嫌ってあえてキャッシュサーバから権威サーバへの問い合わせを UDP の port 53 から出るように設定している場合がありますが、これは汚染される可能性が非常に高まるのでやめるべきです。太古の昔はこのような DNS サーバは多数ありましたが、現状ではとても許されるものではありません。

既に述べたように、一応の対策として乱数値やポート番号による撹乱が行われていますが万全ではありません。一方、権威サーバについては master であれば自分の掌握している設定ファイルを参照しますし、slave であれば master から TCP によってデータを転送します。このゾーン転送については Transaction Signature (TSIG)によって暗号化や署名が可能なので、安全性は担保できると言えます。

DNS サーバを用意する時に安易にフォワーダを設置することが多いですが、投げつけ先のキャッシュサーバは攻撃を受けないのか今一度考えてみるべきです。

2.1. 何故分離した方が安全なのか?

DNS サーバに対する組織外からの問合せを受け付ける必要があるのは権威サーバだけです。権威サーバに嘘が混入してしまうと他組織に迷惑をかけることになります。攻撃は DNS サーバが他の DNS サーバに対してなにがしかの問合せを行っている時に偽装パケットを送信することで行うので、権威サーバ自らが外部に対する問合せを行わなければ危険は低くできます。権威サーバは担当範囲についてだけ返答するように設定すれば、権威サーバが外部に対して発行する問合せを抑制できます。

キャッシュサーバはその役割上どうしても外部への問合せを抑制することはできませんが、外部からの問合せに利用できるようにしておくと次に述べる DNS Amplifier DoS の片棒を担がされることになります。外から再帰問合せを無制限に受け付ける DNS サーバを設置することはやめましょう。

また、権威サーバとキャッシュサーバを兼任していると、上位からの委譲を失った後にもひきつづき権威サーバのつもりで古いデータを答えてしまうことがあるので問題になりやすいです。

なお、DNSSEC が導入され始めたことにともない、TCP でのクエリも発生するようになっています。ファイヤーウォールや DNS サーバ自体のフィルタ(iptables など)では TCP での問い合わせを許可する必要があります。TCP の問い合わせは UDP よりも DNS サーバのリソースを消費しますので、大量の問い合わせが発生する場合は注意が必要です。

2.2. DNS Amplifier DoS

気に入らないホストに大量のパケットを投げつける嫌がらせの 1 つで、おおむね以下の手順をとります。

(1) 脆弱な権威サーバを 1 つ乗っ取り、ゾーンファイル中に返事が大きくなるようなレコードを追加します。TXT レコードが使用されるケースが多いです。

(2) 外部に対して全開しているキャッシュサーバに問合せを送り、キャッシュサーバに内容を覚えさせます。

(3) 多数のキャッシュサーバに対して上記を繰り返します。

(4) 気に入らないホストがそれらの全開キャッシュサーバに対して問合せを行ったかのようなパケットを偽造し、一斉に投げつけます。

(5) キャッシュサーバは気に入らないホストに対して DNS 問合せの返事として大きなパケットを次々に投げるので、気に入らない人のネットワークがパンクしてしまいます。

被害者から見るとキャッシュサーバから DoS を投げてきているように見えるので、キャッシュサーバの所属するネットワークに対して苦情が寄せられる可能性があります。(1)の段階においては問合せパケットの送信元 IP アドレスをキャッシュサーバの内部ネットワークの物に偽造することで問合せ制限を迂回できますが、再帰問合せを外部に許可していなければ (4)の段階で大きなパケットが出て行くことはありません。

3. BIND 以外による運用

分離しない場合、運用は実績・多機能・対応 OS の多さから BIND が第一候補となりますが、分離して運用する場合は BIND 以外で、実績がある物として Unbound と NSD があります。

3.1. Unbound

Unbound は NLnet Labs によって開発されたキャッシュサーバ専用デーモンです。http://unbound.net/ から取得できます。応答速度が BIND より 10 倍と速いことや、メモリ量も少なく、キャッシュ汚染攻撃への耐性が高めてあるなどの利点があります。一方でファイルディスクリプタやポート番号は起動直後に予め確保してしまうため、これらのリソースの見た目の消費は BIND より大きくなります。/etc/hosts を静的データとして使用することも可能ですし、ゾーンを指定してフォワーダを指定する機能もあります。新しいサーバとして立てるときはいいのですが、BIND の完全上位互換ではないので、既存のサーバを入れ替える場合は注意が必要です。

まず、デフォルトでは再帰問い合わせ以外の問い合わせを拒否します。設定次第でそれ以外も受け付けることはできますが、返事に「これは権威サーバの返事です」というフラグ(AA)を立てることができないので NS レコードに記載しているホストで動かしてはいけません。BIND で動作している既存の兼任サーバを NS レコードから外して Unbound のキャッシュ専用サーバにする場合は、後述のクエリのログ取りを行って再帰問い合わせしかやってこないことを確認してからにするべきです。一端 unbound にすると問い合わせフラグは統計情報として取ることはできますが、個別のログとして確認することはできません。入れ替えた後に権威サーバからの返答を要求されていることがわかった場合は wireshark などで直接確認しなくてはなりません。

また、BIND の「view」に相当する機能がなくクライアントの IP アドレスに応じて返事を変えるということはできません。

Unbound によるキャッシュサーバの設定例

```
server:
        verbosity: 1

        # munin で統計情報を見たいので
        extended-statistics: yes
        statistics-cumulative: no
        statistics-interval: 0  # 5 分に 1 回 munin が見に来る

        port: 53

        # 権威サーバに質問するときに使用するアドレス
        outgoing-interface: 192.168.53.254
        # unbound への問合せを許す範囲
        access-control: 192.168.53.0/24 allow

remote-control:

        control-enable: yes
```

Unbound の設定は非常に簡単です。

ルートサーバの IP アドレスはプログラム中にヒントが記載されているので設定ファイルで指定する必要がありません。BIND の `rndc` に相当するプログラムは `unbound-control` です。`unbound-control-setup` を実行して unbound に `SIGHUP` を送れば `unbound-control` が使用できるようになります。OpenSSL が古いと `unbound-control-setup` がエラーになりますが、unbound-control-setup スクリプト中で HASH 変数を変更して OpenSSL コマンドに渡しているオプションの`-sha256` を`-sha1` に変更するなどの対処で回避できます。

`access-control` に `allow` ではなく `allow_snoop` を指定すると非再帰問い合わせも許可されるようになります。

3.2. NSD

NSD は Unbound と同じく NLnet Labs による権威サーバ専用デーモンで、キャッシュサーバよりも機能が少ないため、さらに速くなっています。http://www.nlnetlabs.nl/projects/nsd/ から取得できます。

※参考：「Benchmarking of authoritative DNS implementations」

https://www.dns-oarc.net/files/workshop-201005/MartinHaller-OARC.pdf

ルートサーバでの運用実績もあり、BIND のゾーンファイルを処理することができ、BIND を master サーバとした slave サーバの運用にも使用できます。短所は DNS ラウンドロビンができないこと、またゾーン転送受付が AXFR(全転送)のみで IXFR(差分転送)はできません。slave として運用する場合、master からの転送に IXFR を使うことはできます。ゾーンのアクセス制御が AXFR の可不可しかなく、問合せそのものを拒否することはできません。

NSD による master サーバの設定例

```
まったく設定を記述していない場合のデフォルト値を確認する
# nsd-checkconf -v /dev/null
# Read file /dev/null: 0 zones, 0 keys.
# Config settings.
server:
        debug-mode: no
        ip4-only: no
        ip6-only: no
        hide-version: no
        database: "/var/db/nsd/nsd.db"
        #identity:
        #nsid:
        #logfile:
        server_count: 1
        tcp_count: 10
        tcp_query_count: 0
        tcp_timeout: 120
        ipv4-edns-size: 4096
        ipv6-edns-size: 4096
        pidfile: "/var/run/nsd/nsd.pid"
        port: "53"
        statistics: 0
        #chroot:
        username: "bind"
        zonesdir: "/usr/local/etc/nsd"
        difffile: "/var/db/nsd/ixfr.db"
        xfrdfile: "/var/db/nsd/xfrd.state"
        xfrd_reload_timeout: 10
```

設定ファイルを記述したら nsd-checkconf を使用して確認します。nsd-checkconf の-v オプションを使用するとデフォルトの値はどのように指定されているかが表示されるので、実際にどこのファイルをアクセスしようとするのかがわかります。設定ファイルの nsd.db が存在しないと nsd は起動しません。まずテキストのゾーンファイルを用意してから「nsdc rebuild」を実行すると nsd.db が生成されるので、その後は起動できます。

```
zone:
  name: "example.co.jp"
  # テキストのゾーンファイルの置き場所
  zonefile: "/etc/nsd/example.co.jp.zone"
  #このゾーンに対して許可する slave サーバ
  provide-xfr: 192.168.53.254 NOKEY
```

NSD による slave サーバの設定例

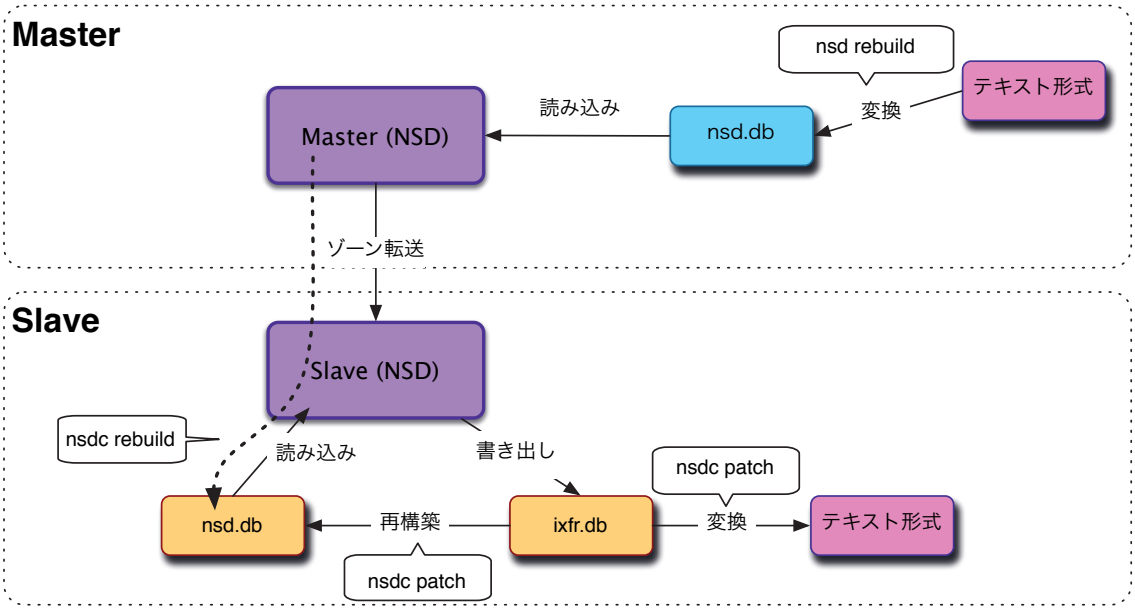
```
zone:
  name: "example.co.jp"
  # テキストのゾーンファイルの置き場所
  zonefile: "/var/lib/nsd/example.co.jp.zone"
  #このゾーンの master サーバ
  request-xfr: 192.168.53.254 NOKEY
```

slave では provide-xfr の代わりに request-xfr を指定します。slave の場合も nsd.db がないと起動できないので、「nsdc rebuild」を実行します。ゾーン転送を行ってその内容を nsd.db およびゾーンのテキストファイルに記録するので、その後は nsd が実行できるようになります。

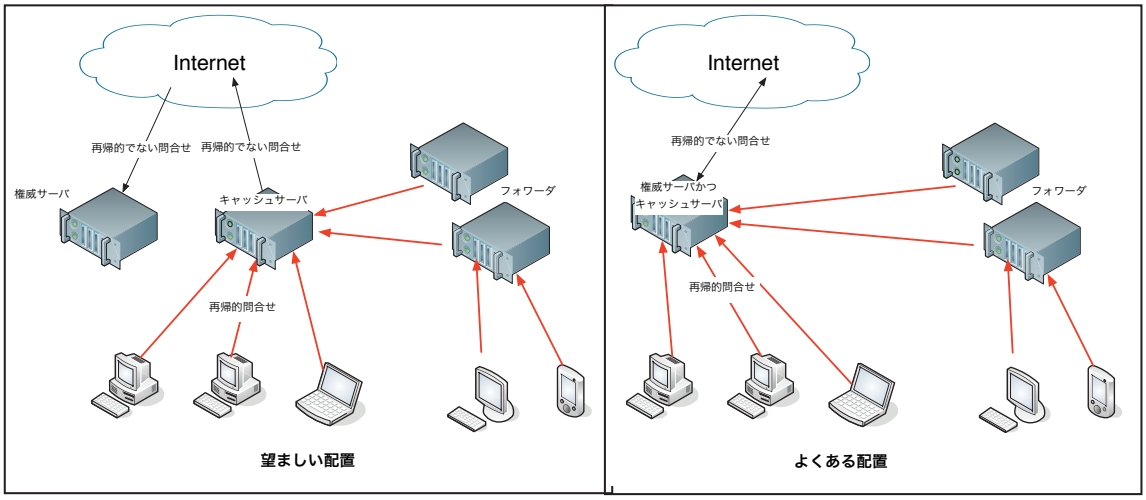
master でゾーンが更新された場合、slave の nsd が転送してきたデータはバイナリファイル(nsd-checkconf -v で表示される difffile が指すファイルでデフォルトでは ixfr.db)に格

納され、zonefile が指定しているテキストのファイルは更新されないことに注意が必要です。バイナリファイルからテキストファイルを復元するためには「nsdc patch」を実行します。

master から自動的に転送してきた情報は ixfr.db に順次追記されゾーン転送が行われるたびにファイルは際限なく膨張していきます。「nsdc patch」は ixfr.db を整理して nsd.db に反映する作業も行いますので定期的に「nsdc patch」を実行する必要があります。



4. BIND による運用



現在 BIND で権威サーバとキャッシュサーバを動作させている場合、とにかく分離するのが一番安全です。しかし、”分離するには予算の稟議が必要でとても無理”等といった場合、アクセスコントロールリスト(ACL)や BIND の「view」によって制限することが可能です(運用コストは高くなりますが)。権威サーバとキャッシュサーバを兼用しており、さらにそれに DNS サーバと称してフォワーダを多数ぶら下げているという事例(図:よくある配置)が多いことでしょう。とにかくキャッシュサーバに対する外部からのアクセスを禁止することが最優先です。キャッシュサーバの動作に影響する ACL には allow-query, allow-query-on, allow-recursion, allow-recursion-on, allow-query-cache, blackhole などがあります。

4.1. ACL による制御

4.1.1. allow-query

DNS 問合せを許可するネットワークアドレスを指定します。デフォルト値は any です(どこからの問い合わせでも受け付ける)。権威サーバも兼ねている場合、やたらにこれを厳しくすると外から引けないドメインになってしまうので注意が必要です。

4.1.2. allow-query-on

DNS サーバを動かしているホストが複数の IP アドレスを持つ場合、DNS 問合せを受け付ける IP アドレスを制限することができます。

4.1.3. allow-recursion

再帰問合せを許可するネットワークを列挙します。以下のように指定すると 192.168.0.0/16, 172.22.0.0/16 以外からの再帰問合せはできませんが、再帰でない問合せは可能です。列挙されていないネットワークからの問合せは再帰問合せフラグが立っていないかのような扱いになります。権威サーバしか行っていない場合、`allow-recursion` は空であるべきです。BIND 9.3 までは指定しなかった場合は無制限でしたが、BIND 9.4 から厳しくなっているのでアップデート時には注意が必要です。

4.1.4. allow-query-cache

```
acl mycompany {
   192.168.0.0/16;
   172.22.0.0/16;
};

allow-recursion { mycompany; };
```

BIND 9.4 以降の allow-recursion の値

recursion	allow-recursion	allow-query-cache	allow-query	allow-recursion として使用される値
off	設定なし	設定なし	設定なし	none
on	設定なし	設定なし	設定なし	localhost + localnets
なんでもよい	設定なし	設定なし	設定あり	allow-query の値
なんでもよい	設定無し	設定有り	なんでもよい	allow-query-cache の値
なんでもよい	設定有り	なんでもよい	なんでもよい	allow-recursion の値

BIND 9.4.1-P1 から登場しました。それ以前の BIND で記述するとエラーになります。問合せに対してキャッシュの内容を用いて返答してよいネットワークを列挙します。不許可と判断されたホストからの再帰問い合わせには空の返答が帰ります。デフォルトの値は以下のようになっているため、`recursion, allow-query, allow-recursion, allow-query-cache` のいずれも設定せずにキャッシュサーバを運用している場合、BIND 9.4.1-P1 以降にバージョンアップすると他のホストからの再帰問い合わせにまともに答えなくなります(これに陥りやすい)。試しにバージョンアップする場合は、`allow-recursion` か `allow-query` を設定してテストするのがよいでしょう。

BIND 9.4 以降の allow-query-cache の値

recursion	allow-query-cache	allow-recursion	allow-query	allow-query-cache として使用される値
off	設定なし	設定なし	設定なし	none
on	設定なし	設定なし	設定なし	localhost + localnets
なんでもよい	設定なし	設定なし	設定あり	allow-query の値
なんでもよい	設定なし	設定有り	なんでもよい	allow-recursion の値
なんでもよい	設定有り	なんでもよい	なんでもよい	allow-query-cache の値

4.1.5. blackhole

DNS 問合せは ACL によって拒否された場合、REFUSED フラグの立った返答が返されますが、DNS 問合せは UDP で行われているので、必ずしも送信 IP アドレスに返事が返せるとは限りません (たとえば社内の使っていないプライベートアドレスが送信元 IP であったなど)。その場合は拒否されていることを通知せずに無視した方がよいので、無視するためのネットワークを列挙します。blackhole は view 毎には指定できません。

4.2. view による分離

上に挙げた ACL による制御は機能毎にアドレスを列挙していく体裁をとりますが、「view」を用いることによって逆にまずアドレス(等)によって所属グループを決定し、そのグループに対して機能の可不可を列挙することが可能です。特に現在サービスがあらゆるアドレスに全開になっている場合、ACL を書いたことによって現在のユーザからの Query が拒否され、苦情に追い回されるという不安がある場合はまず view による制御を検討した方がよいでしょう。キャッシュサーバの役割しかない場合は、以下のような設定で内部と外部からのアクセスを分離できます。

```
options {
    recursion no; /* 基本的に再帰問合せは拒否する */
};

acl "intranet" {
    10.0.0.0/8;
    192.168.0.0/16;
    198.51.100.0/24;
};

view "internal" {
    match-clients { "intranet"; }; /* intranet からの問い合わせは許可 */
    recursion yes;
    allow-recursion { any; }; /* match-clients で制限しているのであえて厳しくしな
くても良い */
};

view "external" {
    match-clients { any; };
    recursion no;
};
```

キャッシュサーバの場合はゾーンの設定がないので view を増やしてもたいしたことはないのですが、権威サーバとして担当しているゾーンが多い場合、くどい設定になるのは仕方ありません。

4.2.1. 分離手順の例

権威サーバとキャッシュサーバを兼ねており、キャッシュサーバ専用の DNS サーバを割り当てることができない場合の対策として、view を使用する例を示します。

(1) 初期状態

```
options {
  directory "/var/named";
  recursion on;
};

zone "." {
  type hint;
  file "named.root";
};

zone "example.co.jp" {
  type master;
  file "example.zone";
};
```

上のような設定だったとします。

(2) 現在の設定を引き継ぐ view を作る

```
options {
  directory "/var/named";
  recursion on;
};

view "default" {
  match-clients { any; };
  allow-recursion { any; };
  zone "." {
    type hint;
    file "named.root";
  };
  zone "example.co.jp" {
    type master;
    file "example.zone";
  };
};
```

view を 1 つでも定義したとき、view の外に書いたゾーン定義は無視されるので、全てのゾーン定義を view の中に繰り込む必要があります。view 中には options が指定できますが、logging は指定できません。

```
acl team1 { 198.51.100.0/24; };
acl team2 { 203.0.113.0/24; };
acl mygroup { team1; team2; };
acl external { ! mygroup; }; /* !で否定ができます */
```

(3) ひとまずアドレスによっておおざっぱなグループに分ける

チーム内・社内・社外程度、もしくは社内・社外程度でよいでしょう。それぞれに対応する ACL を作成します。あらかじめ定義されている ACL として、any(何にでもマッチする), none(何にもマッチしない), localhost(named が動作しているホストの IP アドレス全てにマッチする), localnets(named が動作しているホストが接続されているサブネット全てにマッチする)があります。localhost は 127.0.0.1「以外」にもマッチすることに注意が必要です。

(4)　各グループに対応する view を作る

```
/* option 等についての記述は略 */
acl branch1 { 198.51.100.0/24; };
acl branch2 { 203.0.113.0/24; };
acl mycompany { branch1; branch2; };
acl external { ! mycompany; };

view "company" {
  match-clients { mycompany; };
  recursion on;
  allow-recursion { any; };
  zone "example.co.jp" {
    type master;
    file "example.zone";
  };
};
view "default" {
  match-clients { any; };
  recursion on;
  allow-recursion { any; };
  zone "example.co.jp" {
    type master;
    file "example.zone";
  };
};
```

「recursion on」を指定するだけではなく「allow-recursion」も指定します。

view のマッチ検査は上から行われ、最初にマッチしたものが採用されます。ゾーン定義は公開したい対象の view に全て記載する必要があり、記載していない view に対してそのドメインの問い合わせが来ると「そのようなドメインは存在しない」エラーが帰ります。社外に公開したくないドメインがある場合はこれで隠すことができます。また、master の場合、ゾーンのファイルは全て同じでかまいません。slave の場合はゾーンの名前が同じでも view 毎に master からの転送が行われるので、ファイルは view 毎に異なるファイル名を指定する必要があります。

(5)　グループ毎に recursive の動作を変える

まず、最後の view(社外)からの recursive は停めるべきなので、recursion off;にします。もしも社外にもサービスを提供するべき相手がいるのであれば、社外から recursive を送ってくる相手をあらかじめログとして取るべきです。

```
options {
  directory "/var/named";
  recursion no;
};

/* option 等についての記述は略 */
acl branch1 { 198.51.100.0/24; };
acl branch2 { 203.0.113.0/24; };
acl mycompany { branch1; branch2; };
acl external { ! mycompany; };

view "company" {
  match-clients { mycompany; };
  recursion on;
  allow-recursion { any; };
  zone "example.co.jp" {
    type master;
    file "example.zone";
  };
};
view "default" {
  match-clients { any; };
  recursion no;  /* 外からの再帰問い合わせは禁止 */
  zone "example.co.jp" {
    type master;
    file "example.zone";
  };
};
```

4.3. 再帰問合せ数の制限

無法な数の再帰問合せを行うことでキャッシュを溢れさせる攻撃を防止するために、再帰問合せの数を ACL の `recursive-clients` を指定して制限することができます。ただし、この ACL は view 毎には指定できないので、注意が必要です。攻撃が間欠的・瞬間的に発生しているならキャッシュ汚染攻撃の前段階としての攻撃を抑制できますが、本来のクライアントに対してもとばっちりが発生するので DoS のような効果は発生してしまいます。

5. DNSSEC について

UDP ベースで DNS の問合せが行われていることに本質的な脆弱性があることは明白なので、乱数によって攻撃を避けようというのではなく、公開鍵暗号と電子署名によって内容の正当性を保証しようという拡張仕様「DNS Security Extensions(DNSSEC)」があります。キャッシュサーバに対して偽造した応答パケットを送ろうとしても秘密鍵を知らない限り不可能なので、キャッシュ汚染攻撃を防止できます。

上位ドメインが下位ドメインの DNS サーバを指名するときに公開鍵のハッシュ値を羅列することでキャッシュサーバはたらいまわしされた後で正しく指名されていることを検証できます。ルートサーバからこれを手繰っていくことで正規の答えを得ることができますが、ルートサーバの公開鍵だけは明示的に DNS サーバに教えてやる必要があります。

権威サーバが DNSSEC に対応することは自分の出す内容が正しいことの証明になります。しかし署名のための鍵を取得したり、ゾーンデータに署名したりする作業が必要で、人間への負荷がそれなりに大きいです。キャッシュサーバは単によその DNS サーバがつけてきた署名を検証するだけなので対応は容易です。

(参考文献 「DNSSEC 解説」情報処理 Vol52, No. 9, Sep 2011)

5.1. BIND での DNSSEC

DNSSEC ジャパンの以下サイトに手順が掲載されています。

「DNSSEC を利用するリゾルバーのためのトラストアンカーの設定方法について 第 2 版」

http://dnssec.jp/wp-content/uploads/2011/02/20110124-techwg-dnssec-trustanchor-install-howto-2.pdf

5.2. Unbound での DNSSEC

unbound-1.4.7 以降では unbound-anchor コマンドが用意されているので、ルートゾーンのトラストアンカーを取得してそのファイル名を unbound.conf に指定するだけです。

取得

```
# unbound-anchor –a /var/unbound/root.key
# chown unbound:unbound /var/unbound/root.key
```

unbound.conf に記述

```
server:
    auto-trust-anchor-file: "root.key"
```

unbound を再起動するとこのファイルが自動的に更新されます。

6. ログ取り

外部からアクセスされるサーバには ACL の設定に誤りがないか、アクセスが急に増えたりしていないかを確認するべきで、セキュリティ事象を未然に防ぐためにはログが重要です。特にキャッシュ汚染攻撃を行っている攻撃元は、攻撃対象のキャッシュサーバに詐称したいドメインに近い名前を解決しようとする問合せを大量に送ってきます。ログにこれが残っていればどのドメインに対して汚染を行いたいのかを推定することができますし、また普段から自分の DNS サーバがどのような攻撃を受けているのかを知っておくことが大切です。

6.1. 問合せ自体のログ採取

```
logging {
        channel qlog { /* qlog でなくても勝手な名前でよい */
                syslog local1;
        };
        category queries { qlog; };
};
```

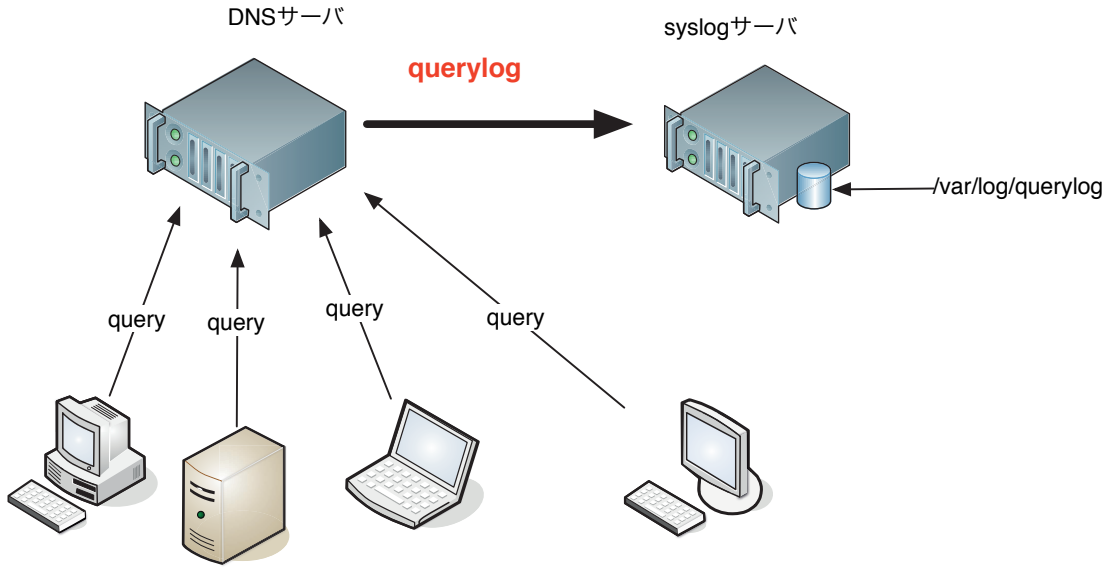
BIND には受信したクエリパケットのログを syslog 経由で出力でき、攻撃の検知に利用できます。その他の DNS サーバの場合はクエリの内容を選んで出力することができないので、tcpdump などでパケットのログを取るか、全体のデバッグレベルを上げて全てのログをとるしかなく、あまり長期間の運用には向いていません。BIND でクエリ毎のログを採取するには named.conf の logging オプションで指定します。

```
local1.*      /var/log/qlog/querylog
```

local1 の facility 属性で syslog が大量に発生するので DNS サーバが直接ファイルに記録するとディスク IO によって反応が悪くなってしまいます。syslog を記録するサーバを別途用意した上で、syslog.conf に

```
local1.*      @syslog サーバのホスト名
```

などと記載してディスク IO はそのサーバに任せた方がよいでしょう。このとき、syslog サーバ側では以下のように設定して適当なファイルに出力します。



かなりの勢いでファイルが成長するので、logrotate や newsyslog の設定を忘れると大惨事となります。BIND-9.2 以前ではログには問合せが recursive かどうかのフラグは表示されませんし、どの view が処理したかも表示されません。組織外からの再帰問合せをすぐに禁止できない場合は BIND を更新するべきです。

```
実際に出力されるログの例
Oct 12 08:38:24 kun named[533]: client 172.101.xxx.yyy#39815: view external: query: 101.1.168.192.in-addr.arpa IN PTR -E
Oct 12 08:38:25 kun named[533]: client 192.168.xxx.yyy#39825: view internal: query: hoge.com IN MX +
Oct 12 08:38:43 kun named[533]: client 192.168.xxx.yyy#40010: view internal: query: mail.hoge.comt IN A +
```

再帰的問合せかどうかは、ログ中の「+」か「-」かで識別できます。BIND のバージョンが上がるとログの文字列が増える傾向があるので、必ずしも末尾とは限りませんが、ここを確認することで外部からキャッシュサーバ扱いで取りに来ているかどうかがわかります。自分が行ったクエリや、どのような返事をクライアントに返したかは記録されないので、ある時間帯にキャッシュが汚染されていたかどうかはログ情報からだけではわかりません。

Unbound は 1.4.11 以降で問い合わせのログがとれるようになっています。

```
server:
        verbosity: 1
        log-queries: yes      # この行を追加すると問い合わせのログが出力される
```

ただし、フラグは記録されませんし、BIND のように syslog の facility や priority を指定することはできません。他のログと全部くっついて出てしまいます。

6.2. 組織外からの再帰問い合わせ具合を調べる

```
/* option, logging, ACL 等はこれより上に記述 */

view “internal” { /* 社内からの問い合わせはわざわざログに取らない */
  matchclients { mycompany; };
  recursion yes; /* 再帰問い合わせ許可 */
  querylog no;
};

view “externalclients” {
  match-clients { any; }; /* 社内からの問い合わせは上にマッチしているのでここにこない
はず */
  match-recursive-only yes; /* 社内からでなく再帰問い合わせの場合のみマッチ */
  recursion yes; /* とりあえず再帰問い合わせ許可 */
  querylog yes;
};

view “external” { /* 再帰問い合わせは externalclients にマッチするのでここにはこないは
ず */
  match-clients { any; };
  recursion no; /* そもそも再帰問い合わせはこの view にならないはずだが、一応拒否 */
  querylog no; /* 再帰問合せではないのでまあログも取らなくても良いかもね */
};
```

`logging` ブロックで `category queries` を指定すると、再帰問い合わせかどうかに関わらず全ての問い合わせがログとして出てきてしまうので「(外部からの再帰問い合わせを拒否したいけど)本来予定していないところからの再帰問い合わせがどのくらいあるのか」だけを調べるには不向きです。

「組織外からで」かつ「再帰問い合わせのみ」にマッチする view を作成し、その view の中で `querylog` オプションを指定するのがよいでしょう。view 中に `querylog` を記述しない場合、`logging` ブロック中の `queries` があれば出力されてしまいます。問い合わせのログを出したくない view には明示的に `querylog no;` を書いた方が安全です。短期的にログを取ってみて権利のなさそうなクライアントしかこない場合は、view「`externalclients`」の `recursion` を `no` に設定し、時々様子を見て view「`mycompany`」の ACL にクライアントを追加していくという対処がよいでしょう。

6.3. 統計情報の採取

BIND や Unbound の場合、クエリの量や種類を内部的にカウンタとして記録しており、その増え具合から付加情報や不審な動きを検知できる場合があります。特に Unbound の場合は応答時間の分布についても記録しており、パフォーマンスに影響するような統計は BIND よりも豊富に取得できます。BIND も 9.5 からは大幅に統計情報が増え、さらに 9.5.2-P1 からは応答時間毎のヒストグラムが採取できるようになっています。

統計情報を可視化するにはさまざまな手段がありますが、いずれの場合もテンプレートが同梱されていない場合は結構な手間がかかります。

本文書では Munin によって Unbound の統計情報をモニタする例を挙げてみます。

Munin は DNS サーバが生成するログを可視化するので非常に簡便ですが、DNS サーバ自体と独立にパケットをモニタして DNS 統計情報を可視化する DSC (http://dns.measurement-factory.com/tools/dsc/) も強力です。

6.3.1. Munin について

Munin は統計情報を可視化するための WWW サーバとモニタされるノード(ホストのことです)で成り立っています。情報を供給するのはノードの役割で、供給された情報を可視化するのがサーバの役割となります。Munin の場合、ノードでは「自分がサーバに対してどのような情報を公開できるのか」をノード自らが伝えることになっており、MRTG や Cacti 等に比べるとモニタ項目の追加が非常に容易に行えます。パッケージは Debian や Fedora であれば OS のレポジトリにありますし、Red Hat Enterprise Linux(以下 RHEL)であれば Extras から、CentOS などであれば DAG レポジトリからインストールできますが、RHEL6 およびその互換ディストリビューション用には提供されていません。RHEL4 および RHEL5 用の古めの SRPM を取得して rpmbuild するか、Fedora 用の SRPM を取得して rpmbuild することになります。FreeBSD であれば ports もしくは packages からインストールすることができます。

パッケージに頼らずソースからコンパイルする場合、munin 本体はさしたる問題はないのですが、rrdtool が多数のパッケージに依存しているので注意が必要です。特に FreeBSD の場合 cairo や pango といった大物を必要とするので X11 抜きでインストールした状態から ports でコンパイルして入れるとかなりの時間がかかります。Munin 配布元の tar にも SPEC ファイルが同梱されていますが、RedHat 用ではなく SUSE 用であり、ファイル名などにも間違いがあるので Fedora の SRPM を流用するのが簡便です。

RHEL4 ベースのディストリビューションでも、たとえば Fedora16 用の munin-1.4.6-4.fc16.src.rpm を取得してビルドすることで munin-1.4.6 を使用できます。その際必要な perl-Module-Build, perl-Log-Log4perl, perl-Net-SSLeay が OS 本体のレポジトリにないのですが、DAG レポジトリから YUM でインストールすることができます。

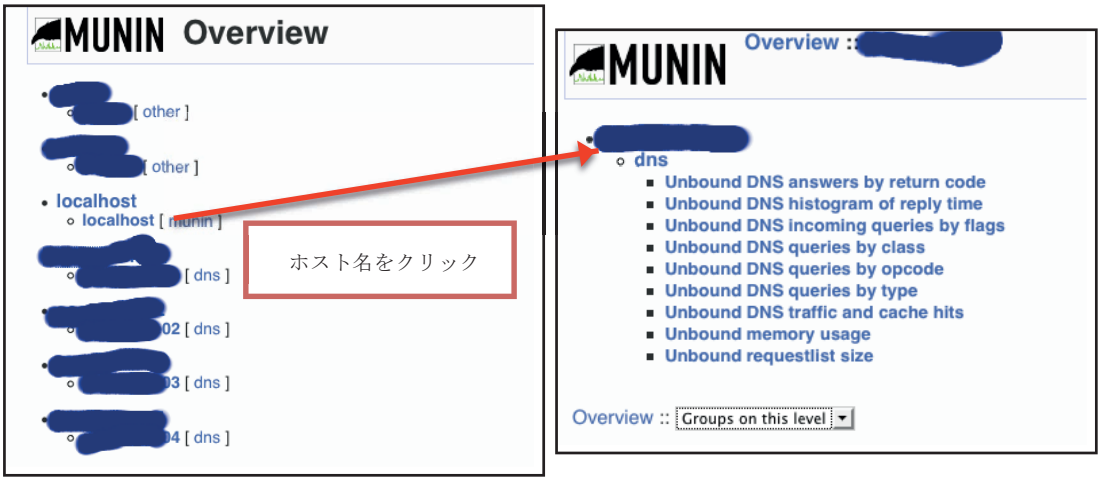
なお、ルータなどのトラフィックモニタとしては MRTG や Cacti の方が優秀です。

サーバの設定

可視化側のサーバではパッケージ名「munin」(RedHat, Debian 等)もしくは 「munin-master」(FreeBSD ports)をインストールします。Munin サーバでは Unbound を動かしているホストからデータを取得するように設定ファイルに 3 行追加するだけです。

```
[unbound-server1]
address 192.168.2.254
use_node_name yes
```

パッケージでインストールした場合、デフォルトでは「http://ホスト名/munin」の URL でアクセスできるようなディレクトリ(RedHat 系ならば/var/www/munin, FreeBSD 系ならば /usr/local/www/munin)に配置されるので、この URL でよければ httpd.conf の設定をいじ



る必要もありません。インストール直後は index.html はありませんが、cron によって統計が採取された後は index.html が生成されているはずです。

また、インストール直後は cron ですべてのグラフを生成し、CGI を設定する必要はないような設定になっています。グラフが大量になってしまう場合、5 分に一度グラフを生成していると重たいので CGI で必要なときだけグラフを生成するような設定も可能ですが、ここでは割愛します。

ノード側(Unbound を動かしているホスト)の設定

(1) munin-node パッケージのインストール

munin-node パッケージには Unbound 用のスクリプトが同梱されていませんので、更に Unbound のソース中の contrib/munin_unbound_ を munin の plugin ディレクトリにコピーします。Red Hat Enterprise Linux と同系列であれば /etc/munin/plugins/munin_unbound としてコピーします。

(2) munin_unbound_のシンボリックリンク作成

unbound_munin_ という名前のファイルを unbound_munin_by_class, unbound_munin_by_flags, ... という名前でシンボリックリンクします。

```
/etc/munin/plugins% ls -l *unbound*
-rwxr-xr-x 1 root root 17416 Apr 16  2010 unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_by_class -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_by_flags -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_by_opcode -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_by_rcode -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_by_type -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_histogram -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_memory -> unbound_munin_
lrwxrwxrwx 1 root root    14 Apr 17  2010 unbound_munin_queue -> unbound_munin_
```

(3) plugin で使用される環境変数の設定

プラグインの環境ファイルを用意します(Red Hat Enterprise 系統であれば /etc/munin/plugin-conf.d/unbound, FreeBSD ならば /usr/local/etc/munin/plugin-conf.d/unbound)。env.unbound_conf や env.unbound_control はインストール状態に合わせて書く必要があります。

```
[unbound*]
user root
env.statefile /var/lib/munin/plugin-state/unbound-state
env.unbound_conf /var/unbound/unbound.conf
env.unbound_control /usr/sbin/unbound-control
env.spoof_warn 1000
env.spoof_crit 100000
```

(4) munin-node の再起動

unbound がリストされているか確認します。munin-node は 4949 番で待っていますので、接続して「list」と入力することでモニタ項目一覧が取得できます。

```
% telnet localhost 4949
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
# munin node at localhost.localdomain
list
open_inodes irqstats if_err_eth3 unbound_munin_memory df swap
unbound_munin_by_rcode load unbound_munin_queue cpu df_inode
unbound_munin_histogram forks open_files iostat memory
unbound_munin_by_type vmstat unbound_munin_ entropy processes
unbound_munin_by_opcode if_err_eth2 postfix_mailqueue if_eth3
netstat interrupts unbound_munin_by_class if_eth2
unbound_munin_by_flags
quit
Connection closed by foreign host.
```

(5) munin-node.conf の設定

munin-node をインストールした各ホストについて、munin-node.conf を編集し、munin サーバの IP アドレスを追記します。デフォルトでは 127.0.0.1 しか許されていないため、munin サーバ本体以外には必ず必要です。

(6) munin-node の再起動

5 分待った後、サーバ側のモニタ項目に Unbound 関連 が増えていれば OK です。

BIND の場合

BIND の情報も Unbound ほどではありませんが、munin で可視化できます。Unbound の場合は `unbound-control` を実行してその出力をスクリプトが拾って報告しているのですが、BIND

```
options {
  statistics-file "/var/run/named.stats";
};
```

の場合は `named` に統計ファイルを出力させ、そのファイルを解釈するスクリプトを実行します。統計ファイルの場所を `named` と `munin-node` のプラグインに伝える必要がありますが、プラグインが `/var/run/named.stats` を仮定して作られているので、named にそのパス名を指定する方が簡便です。

また、この統計ファイルのフォーマットが変遷しているので BIND のバージョンに注意が必要です。BIND9 用のプラグインは munin に同梱されているのですが、`munin-node-configure` は自動設定してくれないので手動で行います。BIND9 のプラグインは `bind9` と `bind9_rndc` の 2 つがあり、`bind9` は `querylog` の出力を検査して A や MX など何のレコードをクエリされた

か、bind9_rndc は rndc stat で取得した統計情報からクエリが成功/失敗した数などを取得します。ただし、bind9 は view を使用している場合の出力に対応しておらず、querylog のフォーマットが変わるとすぐ追従できなくなります。

BIND 9.0

rndc で統計ファイルが出力できないので、bind9_rndc は使用できません。

BIND 9.1 ～ 9.4.3

bind9, bind9_rndc ともに使用可能です。

BIND 9.4 ～

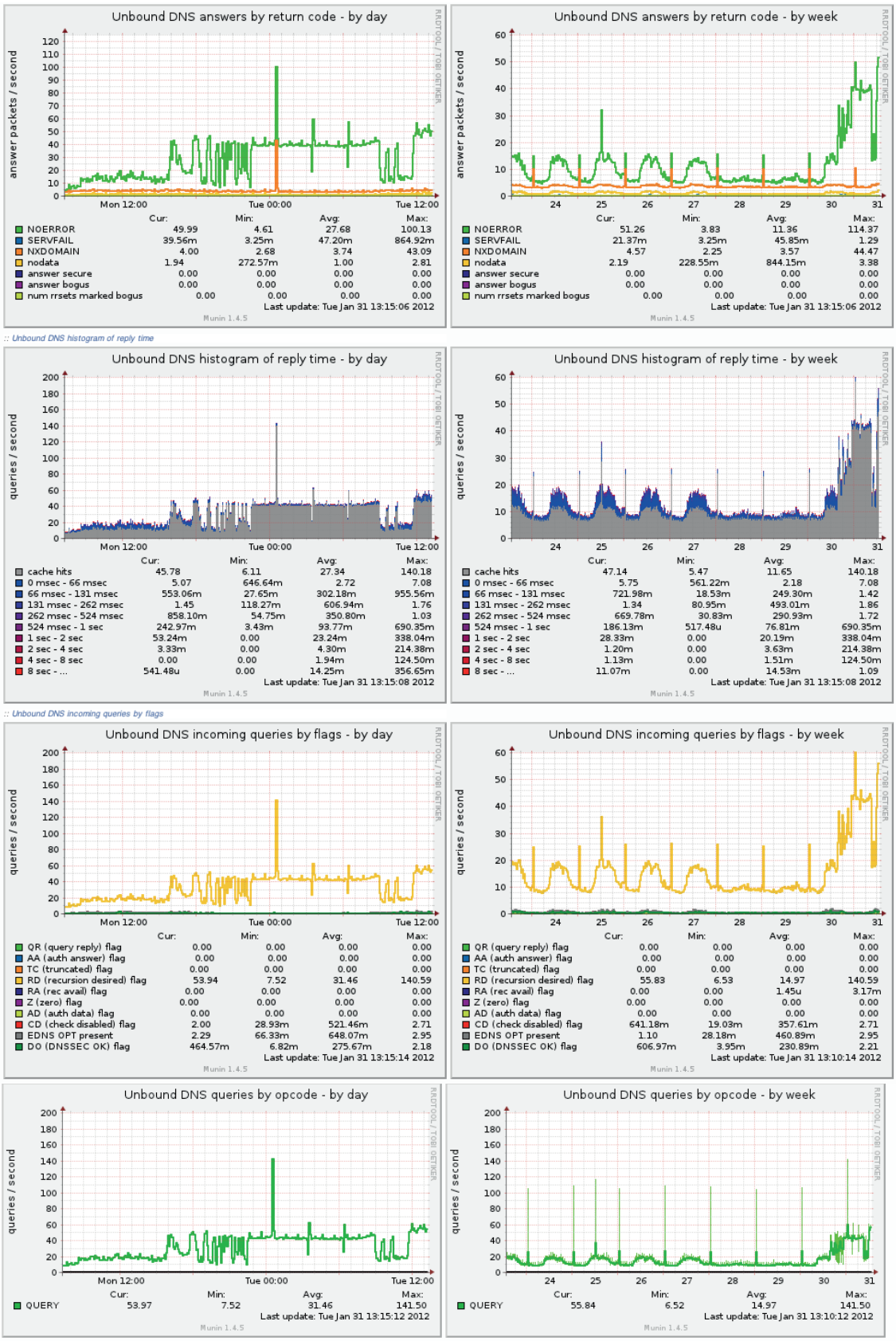
named.stats のフォーマットが変わっているので、munin-1.4 以降が必要です。何らかの問題で munin-node のパッケージ自体を更新できない場合は、bind9_rndc プラグインだけを持ち込めば十分です。

BIND 9.6 ～

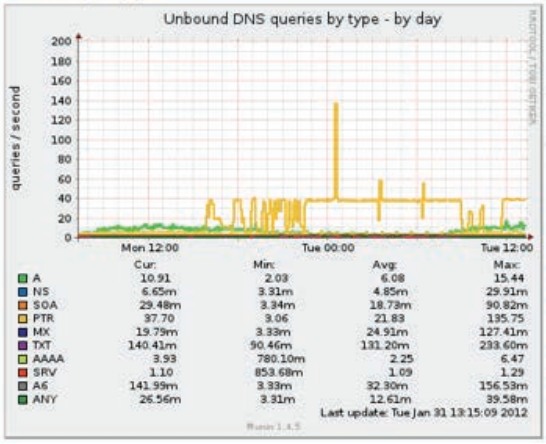
統計ファイルにクエリの種類と数、さらに Unbound のような「クエリがどのくらいの所要時間で処理できたか」も出力されるようになりましたが、残念ながらまだ munin のプラグイン側がそれに対応していないようです。

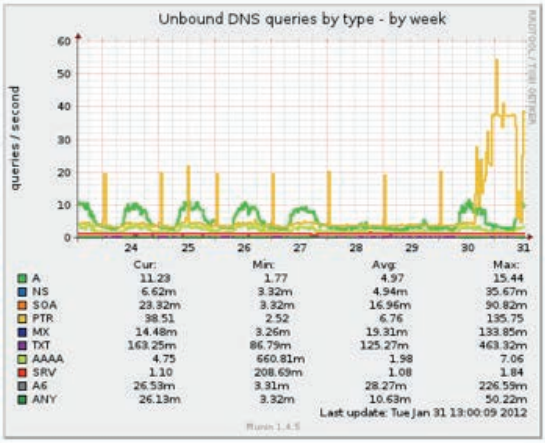
Unbound での実例

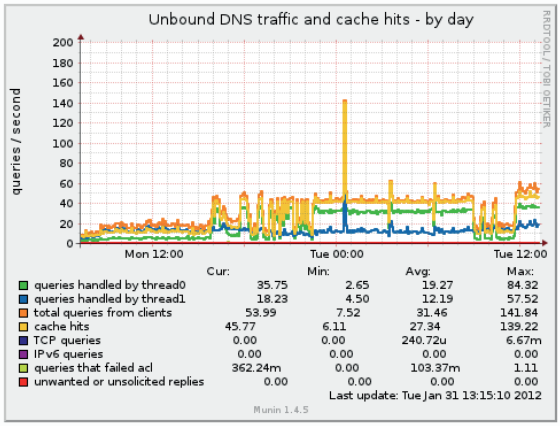
大量のグラフが出ます。不要であればモニタされる側のホスト上で plugins 中のシンボリックリンクを消すことでグラフ化されないようにできます。(冊子版は白黒ですが、SS 研 Web サイトの PDF 版はカラーです)

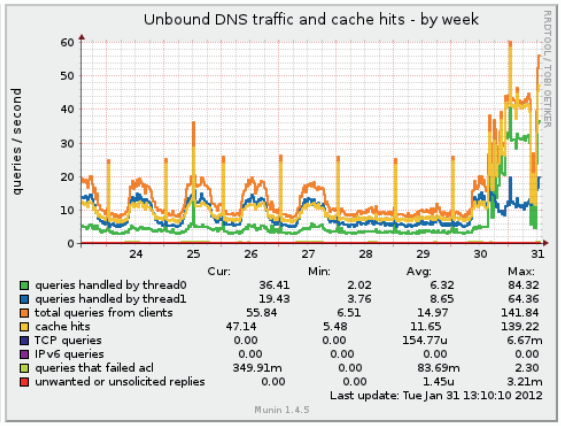


:: Unbound DNS queries by type

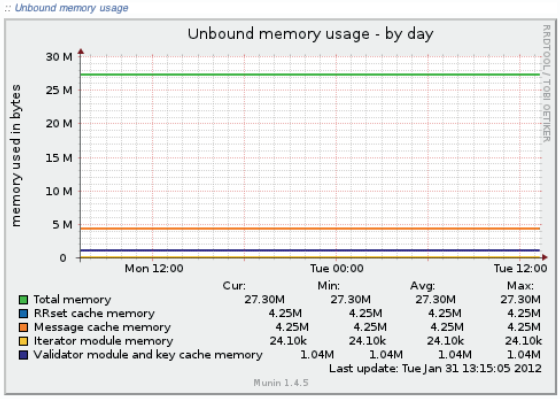


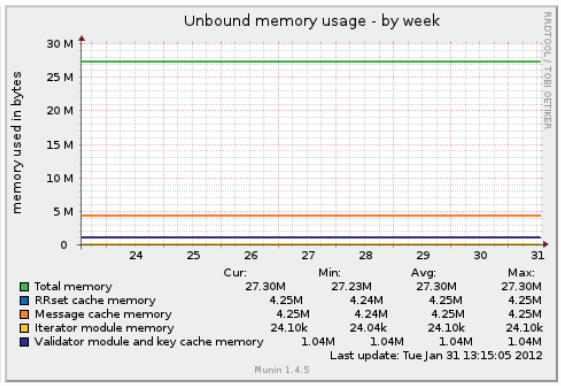




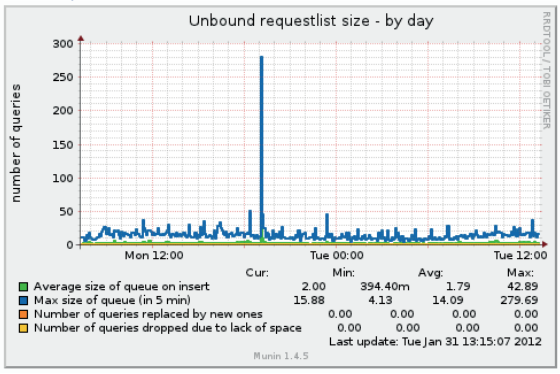


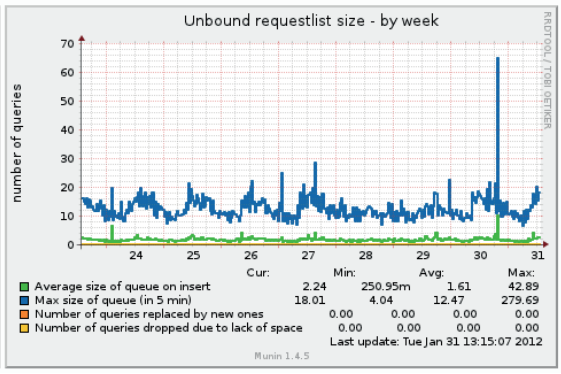
:: Unbound memory usage





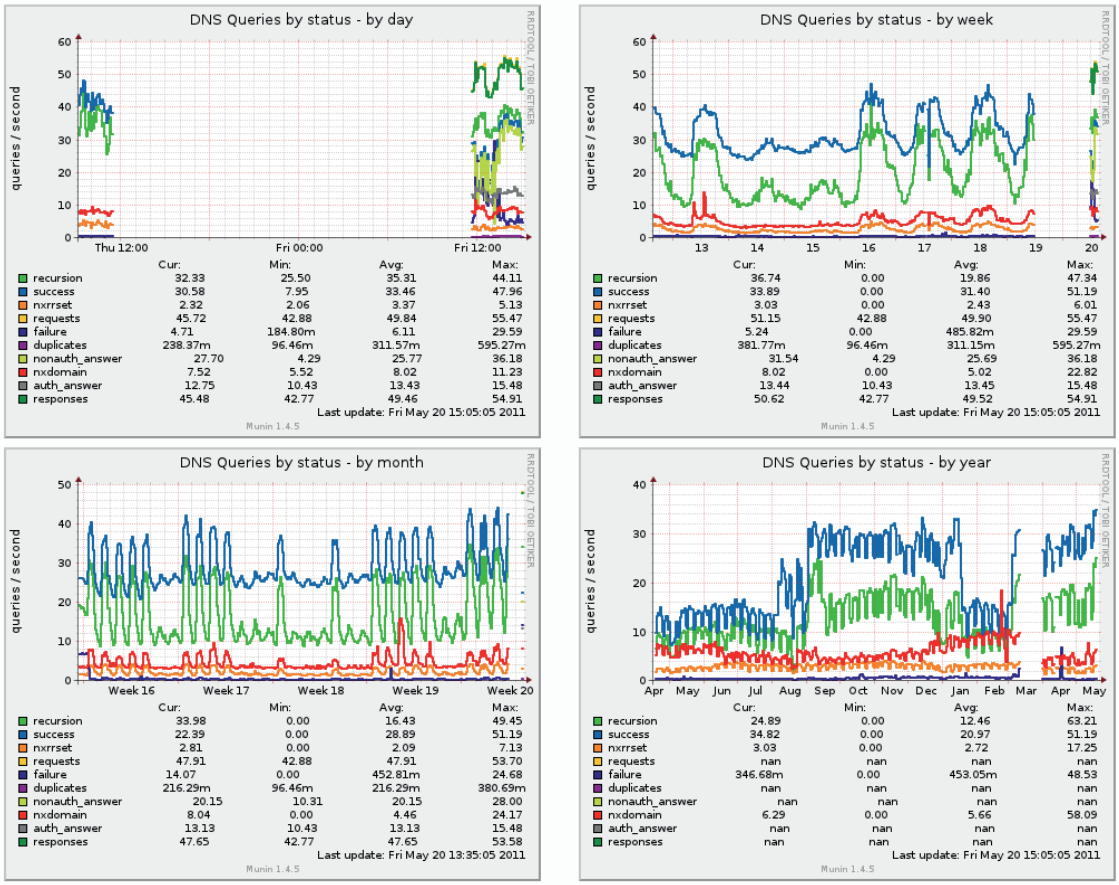
:: Unbound requestlist size





BIND での実例

下の例では途中で BIND をバージョンアップ(9.2→9.7)したので、再開後はグラフの線が増えています。



BIND でのキャッシュの可視化

キャッシュサーバに特化した Unbound はキャッシュの使用量を可視化するためのプラグインがソースに添付されています。BIND はプラグインこそ添付されていませんが、rndc の出力する統計情報を用いて同様なものを作成することができます。

rndc のキャッシュ使用量情報は view 毎に分割されており、以下のようになっています。また、

```
 ++ Cache DB RRsets ++
 [View: internal (Cache: internal)]
                 1083 A
                  471 NS
                    2 CNAME
                    7 MX
                  206 AAAA
                    2 !A
                    1 !MX
                   59 !AAAA
                    2 NXDOMAIN
 [View: external (Cache: external)]
                 2485 A
                 1058 NS
(以下略)
```

定義しないのに_bind という名前の view も出力されますが、この view は CHAOS クラスで BIND のバージョン番号などを問い合わせたクエリについて記録されます。view が一つも定義さ

れていない場合、「default」という view が使用されますので、「default」と「_bind」という 2 つの view があるかのような出力になります。rndc stats で情報をはき出させると statistics-file の末尾に追加されるので、ファイルの末尾から見るために、GNU fileutils の tac コマンドを使用しています。

プラグインのファイルを bind9_rndc_cache としてセーブし、先頭の rndc のパスおよび named.stats のパスを自分の環境に合わせて編集します。プラグイン自体が正しく動くか単独で実行してみます。

```
% ./bind9_rndc_cache
type_A.value 1068
type_NS.value 457
type_CNAME.value 2
type_SOA.value 0
type_PTR.value 0
type_MX.value 6
type_TXT.value 0
type_AAAA.value 207
```

引数なしで実行

```
% ./bind9_rndc_cache config
graph_title DNS Cache occupancy
graph_args -l 0
graph_vlabel Cache Entries
graph_category BIND
type_A.label A
type_A.draw AREASTACK
type_A.min 0
type_NS.label NS
```

引数 config つきで実行

```
% telnet localhost 4949
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
# munin node at localhost
config bind9_rndc_cache
graph_title DNS Cache occupancy
graph_args -l 0
graph_vlabel Cache Entries
graph_category BIND
type_A.label A
type_A.draw AREASTACK
```

動作したらプラグインスクリプト本体を munin のプラグインディレクトリ(/etc/muin/plugin.d/ や /usr/local/etc/munin/plugin.d/など)に bind_rndc_cache として設置します。munin-daemon を再起動し、telnet で munin-daemon に接続してプラグインが動作しているか確認します。

プラグインに渡す設定は次のようになります。plugin-conf.d/bind9_rndc_cache などの単独のファイルを新たに作成するか、すべてのプラグインの設定を記述しているファイル plugin-conf.d/plugins.conf に記述します。

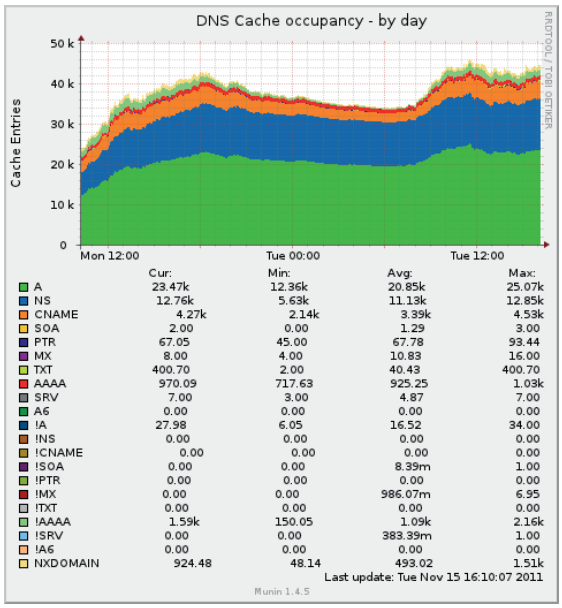
プラグインスクリプト本体 を次ページに掲載します。SS 研 Web サイトからも取得できます。

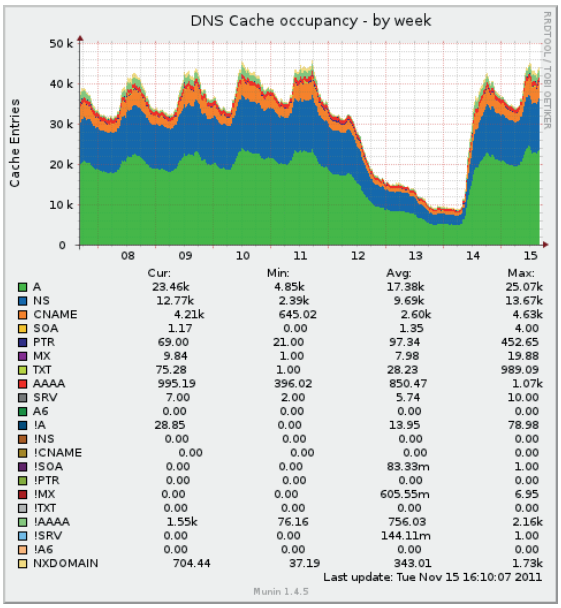
```
#!/usr/bin/perl -w
use strict;
my $rndc = $ENV{rndc} || '/usr/sbin/rndc';
my $querystats = $ENV{querystats} || '/var/run/named.stats';
my %IN;
my $version96 = 0;

# check to see if we're running bind 9.6
if ( open VERSION, "$rndc 2>&1 |" ) {
    while ( my $line = <VERSION> ) {
        if ( $line =~ m/^Version:\s+9\.(\d+)\D/o ) {
            $version96 = 1 if $1 >= 6;
        }
    }
}
exit 0 unless $version96;
my $lines = [];
my %sections;
open(my $tac, '/usr/bin/tac ' . $querystats . ' |');  # FreeBSD では/usr/local/bin/gtac
while (<$tac>) {
        chomp;
        push (@{$lines}, $_);
        last if /^\+\+\+ Statistics Dump \+\+\+/;
        if (/^\++\s+(.*\S)\s+\++$/) {
                $sections{$1} = $lines;
                $lines = [];
        }
}
close($tac);
my $cache;
my $cache_view;
foreach (reverse(@{$sections{'Cache DB RRsets'}})) {
        if (/^\[View: \S+ \(Cache: (\S+)\)\]$/) {
                $cache_view = $1;
        } elsif (/^\[View: (\S+)\]$/) {
                $cache_view = $1;
        } elsif (/^\s+(\d+) ([!A-Z]+)$/) {
                $cache->{$cache_view}->{$2} = $1;
        }
}
my @types = qw/A NS CNAME SOA PTR MX TXT AAAA SRV A6 !A !NS !CNAME !SOA !PTR  !MX !TXT !AAAA !SRV !A6
NXDOMAIN/;

foreach my $t (@types) {
        my $tt = $t;
        $tt =~ s/!/not_/;
        $IN{$tt} = $cache->{default}->{$t} || 0;
# default を適当な view の名前に変更するとその情報が出力される
}
if (defined($ARGV[0]) and ($ARGV[0] eq 'config')) {
    print "graph_title DNS Cache occupancy\n";
    print "graph_args -l 0\n";
    print "graph_vlabel Cache Entries\n";
    print "graph_category BIND\n";
    for my $key (@types) {
        my $label = $key;
        $key =~ s/!/not_/;
        print "type_$key.label $label\n";
        print "type_$key.draw AREASTACK\n";
        print "type_$key.min 0\n";
    }
} else {
    foreach my $key (@types) {
        $key =~ s/!/not_/;
        print "type_$key.value $IN{$key}\n";
    }
}
```

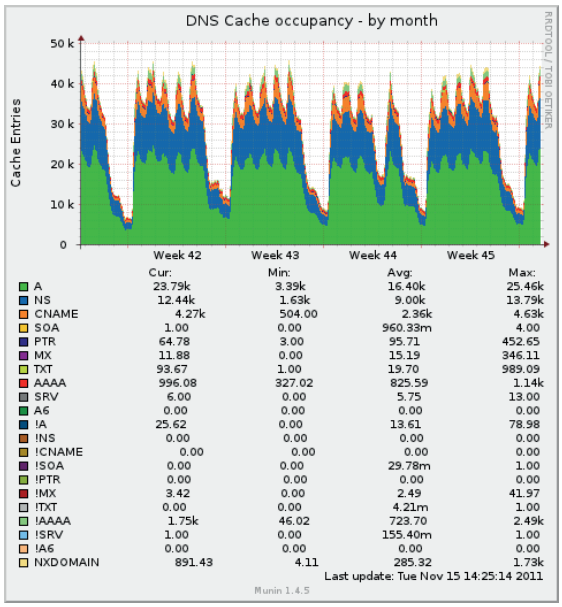
プラグイン用環境設定

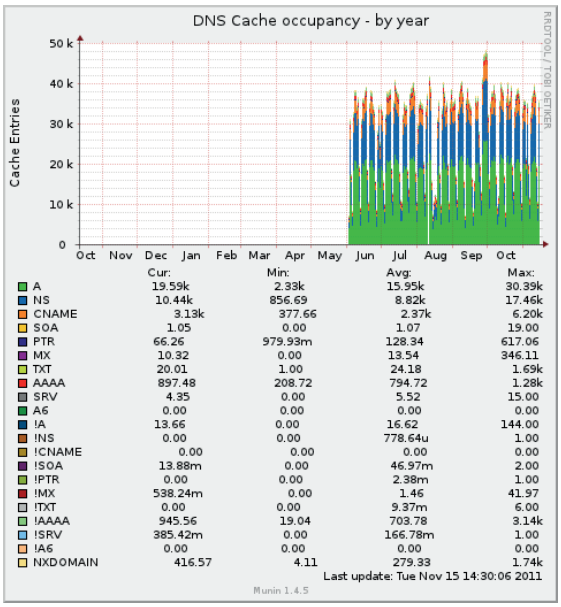
```
[bind9_rndc_cache]
# rndc のパスを指定
env.rndc /usr/local/sbin/rndc
# named.confで指定しているstatistics-fileのパスを指定
# chrootしている場合はnamed.confのパスそのままではいけないことに注意
env.querystats /var/named/chroot/var/stats/named.stats
```

結果









Field	Internal name	Type	Warn	Crit	Info
A	type_A	gauge			
NS	type_NS	gauge			
CNAME	type_CNAME	gauge			
SOA	type_SOA	gauge			
PTR	type_PTR	gauge			
MX	type_MX	gauge			

BIND でのクエリタイプの可視化

同様にクエリタイプも rndc の出力に含まれるので、これを可視化してみます。今度は view 毎には分割されていないのでもう少し手短なスクリプトで作成することができます。

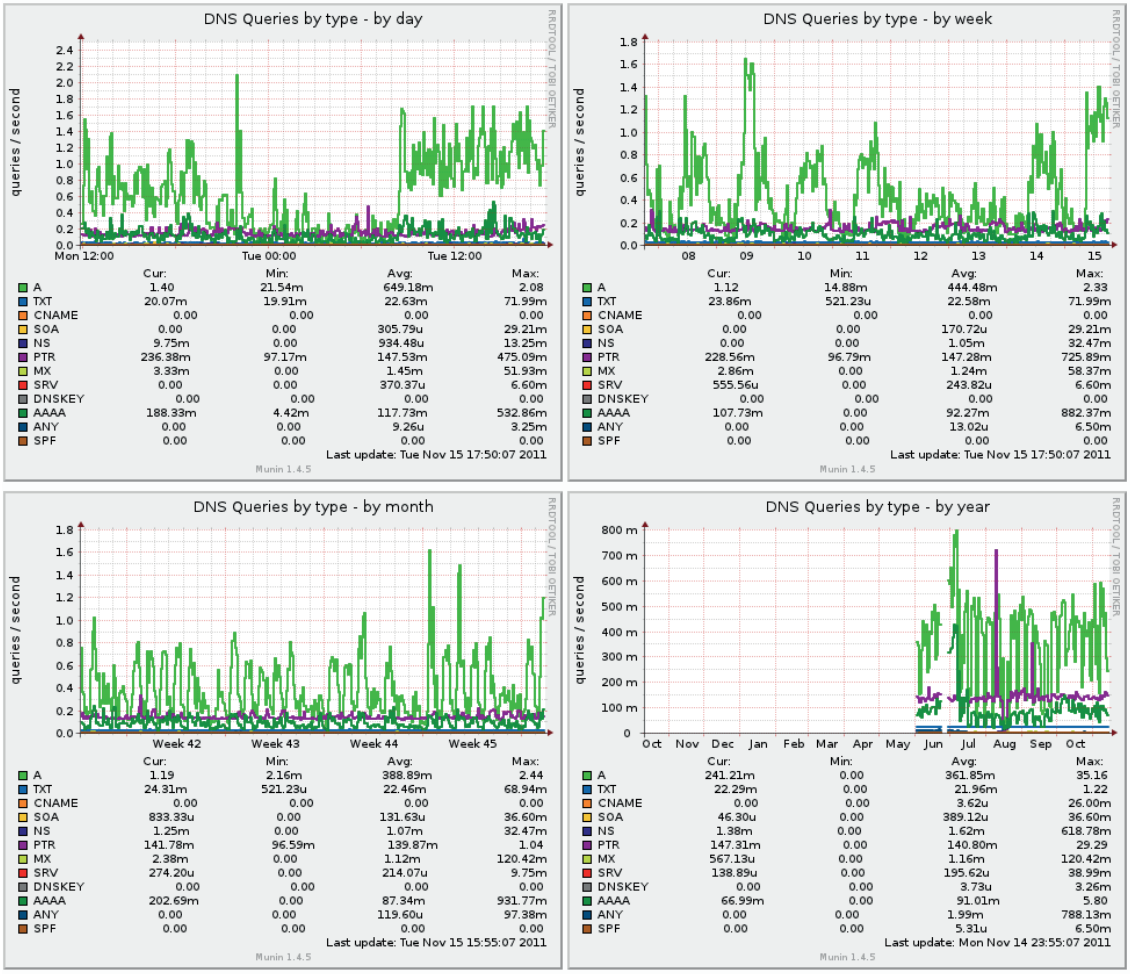
```
#!/usr/bin/perl -w
use strict;

my $rndc = $ENV{rndc} || '/usr/sbin/rndc';
my $querystats = $ENV{querystats} || '/var/run/named.stats';
my %IN;
my $version96 = 0;

# check to see if we're running bind 9.6
if ( open VERSION, "$rndc 2>&1 |" ) {
    while ( my $line = <VERSION> ) {
        if ( $line =~ m/^Version:\s+9\.(\d+)\D/o ) {
            $version96 = 1 if $1 >= 6;
        }
    }
}

exit 0 unless $version96;

my $lines = [];
my %sections;

open(my $tac, 'tac ' . $querystats . ' |');
while (<$tac>) {
        chomp;
        push (@{$lines}, $_);
        last if /^\+\+\+ Statistics Dump \+\+\+/;

        if (/^\++\s+(.*\S)\s+\++$/) {
                $sections{$1} = $lines;
                $lines = [];
        }
}
close($tac);

foreach (reverse(@{$sections{'Incoming Queries'}})) {
        if (/^\s+(\d+) ([A-Z]+)$/) {
                $IN{$2} = $1;
        }
}

if (defined($ARGV[0]) and ($ARGV[0] eq 'config')) {
    print "graph_title DNS Queries by type\n";
    print "graph_vlabel queries / \${graph_period}\n";
    print "graph_category BIND\n";

    for my $key (keys %IN) {
        print "type_$key.label $key\n";
        print "type_$key.type DERIVE\n";
        print "type_$key.min 0\n";
    }
} else {
    print "type_$_.value $IN{$_}\n" for keys %IN;
}
```

同様に実行して動作を確認する。

```
% ./bind9_rndc_type
type_HINFO.value 55
type_AFSDB.value 510
type_TXT.value 462809
type_NS.value 179959
type_LOC.value 7
type_AXFR.value 454
type_SRV.value 125294
type_TKEY.value 21282
type_AAAA.value 16938271
type_ANY.value 558223
```

```
% ./bind9_rndc_type config
graph_title DNS Queries by type
graph_vlabel queries / ${graph_period}
graph_category BIND
type_HINFO.label HINFO
type_HINFO.type DERIVE
type_HINFO.min 0
type_AFSDB.label AFSDB
type_AFSDB.type DERIVE
type_AFSDB.min 0
type_TXT.label TXT
type_TXT.type DERIVE
```

結果



Field	Internal name	Type	Warn	Crit	Info
A	type_A	derive			
TXT	type_TXT	derive			
CNAME	type_CNAME	derive			
SOA	type_SOA	derive			

7.　付録

7.1.　glue レコードとは

上位ドメインが下位ドメインを委譲する際、下位ドメインを担当する権威サーバとして NS レコードに「名前」を指定します。下位ドメインを検索するためにはこの権威サーバの IP アドレスがわからないとそもそも問合せができません。このため、上位ドメインの権威サーバのデータに下位ドメインの権威サーバの IP アドレスを付与しておきます。これが glue レコードです。example1.com について.com の元締めに訊いた際、移譲先が ns.example1.com や ns.example2.com のように.com の配下だった場合は.com の元締めが権威サーバの名前と共に IP アドレスも答えてきます。

example.com の権威サーバが ns.example.net や ns.example.org といった上位ドメインが異なる(権威サーバ自体は.com の配下の名前を持っていない)権威サーバが含まれることがありますが、上位ドメインの権威サーバが「そこは自分の担当している空間ではないので教えない」という場合は通知されるかもしれませんし、されないかもしれません。

```
; <<>> DiG 9.6.0-APPLE-P2 <<>> +norecur -t ns google.co.jp. @z.dns.jp
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37105
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 0

;; QUESTION SECTION:
;google.co.jp.                  IN      NS

;; AUTHORITY SECTION:
google.co.jp.           86400   IN      NS      ns1.google.com.
google.co.jp.           86400   IN      NS      ns3.google.com.
google.co.jp.           86400   IN      NS      ns2.google.com.
google.co.jp.           86400   IN      NS      ns4.google.com.

;; Query time: 11 msec
;; SERVER: 203.119.1.10#53(203.119.1.10)
;; WHEN: Tue May 17 10:19:09 2011
;; MSG SIZE  rcvd: 112
```

たとえば google.co.jp についての権威サーバ情報を co.jp の権威サーバである z.dns.jp などに問い合わせると権威サーバが ns1.google.com であることは伝えてきますが、それらの IP アドレスについては答えてきません。

7.2.　additional レコードとは

NS レコードではなく、IP アドレスを取得しようとして A を検索したら CNAME だったという場合や、メールを投げたいので MX を引いたがついでに A レコードも欲しいなど、最終的には A レコードがついでに欲しい場面は多々あります。そこで DNS サーバが「別にクエリされてはいないが、せっかくだから A レコードもつけて返してあげる」こともできます。この場合は別に A レコードがついてこなくても自分で再度問い合わせることは可能なので、glue ではありません。このようなレコードを additional レコードと呼びます。glue は必須ですが、additional は必須ではありません。

BIND では権威サーバが additional レコードを返す動作について制御ができます。

- `additional-from-auth`

自分が他のゾーンの権威サーバであった場合に、そのデータを利用して additional レコードを生成するかどうか

- `additional-from-cache`

  キャッシュに蓄積されたデータを利用して additional レコードを生成するかどうか

の 2 つのオプションで制御できます。どちらも BIND のデフォルトでは yes になっています。