# DNS

**民田 雅人** <minmin@jprs.co.jp>
**株式会社日本レジストリサービス**
2005年10月6日
JPNIC・JPCERT/CC Security Seminar 2005

# 本日の内容

- DNSのおさらい
- Pharming
- コンテンツサーバとキャッシュサーバ
- 虚偽の応答
- BIND
- djbdns

# DNSのおさらい

# DNS (Domain Name System)とは?

- **もともとはIPアドレスとホスト名の対応をとるための機構の一つ**
  - **ホスト名**　www.example.jp
  - IP**アドレス**　172.16.37.65

  - http://www.example.jp/ - **覚えやすい**
  - http://172.16.37.65/ - **覚えにくい**

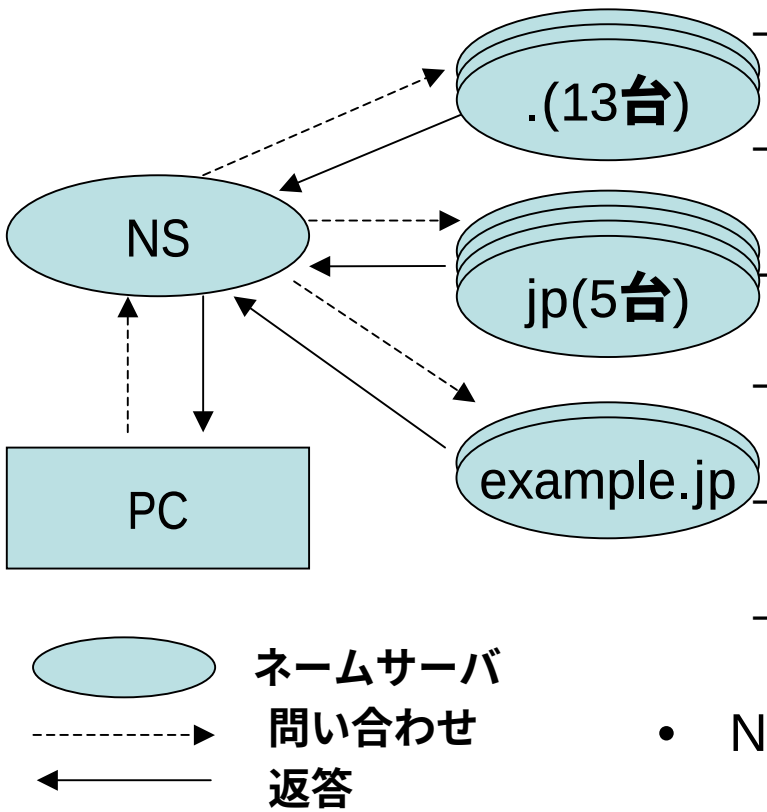
# サーバ･クライアントモデル

- DNS(Domain Name System)**は、サーバとクライアントから成り立つ**
- **サーバ**
  - **専用のサービスプログラム ⇒「ネームサーバ」**
- **クライアント**
  - **ライブラリ　⇒「リゾルバ」**
  - **サービスプログラム ⇒「ネームサーバ」**

# WEBブラウザで http://www.example.jp/を参照

- www.example.jpのIPアドレスを検索
  - WEBブラウザが稼動するPCには、ネームサーバのIPアドレスを、予め、手動あるいはDHCP等による自動で設定する
- ネームサーバへwww.example.jpのIPアドレス問い合わせると、答えが返る
- WEBブラウザがページを表示する

# www.example.jpの IPアドレスの検索

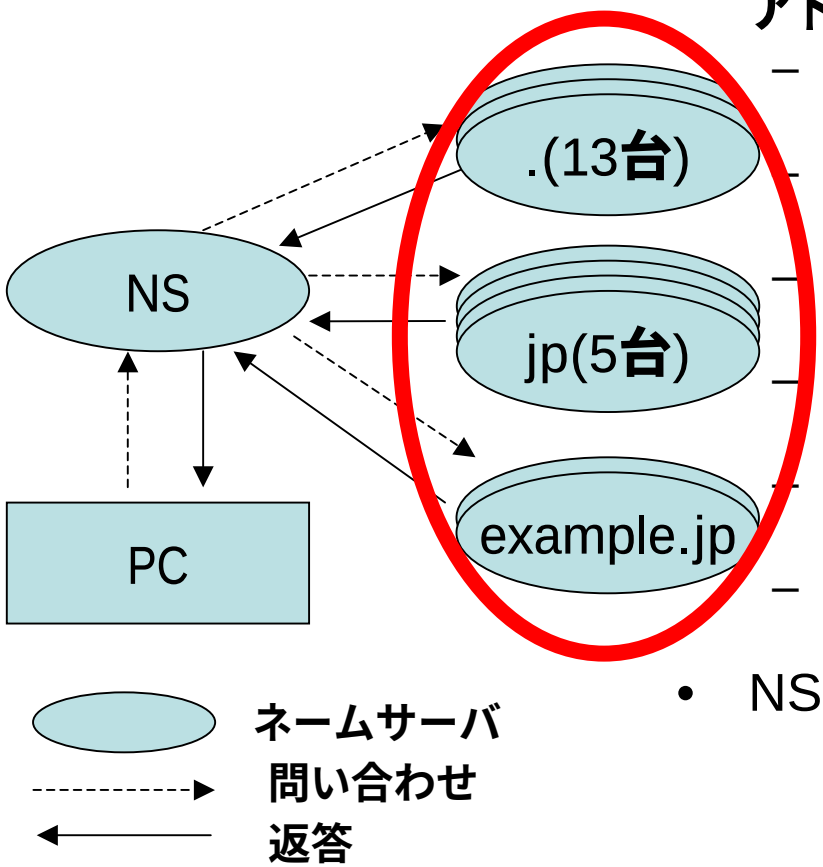



- PCからNSへwww.example.jpのIPアドレスの問い合わせ
  - NSが“.”(ルート)のネームサーバにIPアドレスを問い合わせ
  - “.”のネームサーバがjpのネームサーバを返答
  - NSがjpのネームサーバへIPアドレスを問い合わせ
  - jpのネームサーバがexample.jpのネームサーバを返答
  - NSがexample.jpのネームサーバへIPアドレスを問い合わせ
  - example.jpのネームサーバがwww.example.jpのIPアドレスを返答
- NSがPCへIPアドレスを返答

# 2種類のネームサーバ(1)

- ゾーンデータを保持し、問合せに答える
  - 「www.example.jpのIPアドレスは10.10.10.1である」等の情報を持つ
- 上位ドメインに登録するネームーサーバ
  - jpに登録するexample.jpのネームサーバ
- 「コンテンツサーバ」と呼ぶ
  - Authoritative Nameserver
  - 「権威サーバ」、「権威DNSサーバ」とも言われる

# コンテンツサーバ

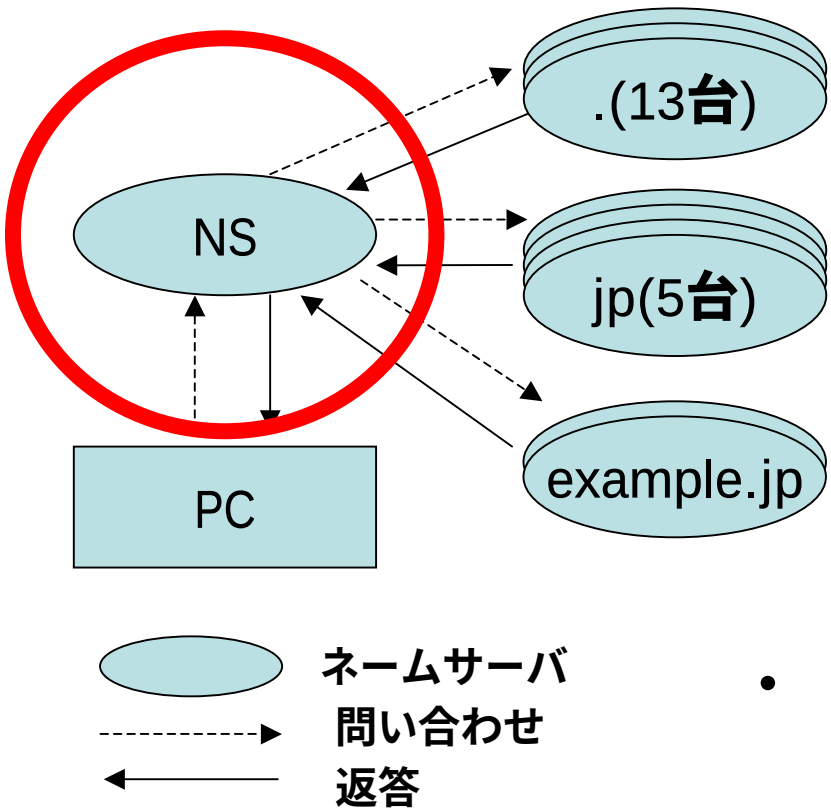



- PCからNSへwww.example.jpのIPアドレスの問い合わせ
  - NSが“.”(ルート)のネームサーバにIPアドレスを問い合わせ
  - “.”のネームサーバがjpのネームサーバを返答
  - NSがjpのネームサーバへIPアドレスを問い合わせ
  - jpのネームサーバがexample.jpのネームサーバを返答
  - NSがexample.jpのネームサーバへIPアドレスを問い合わせ
  - example.jpのネームサーバがwww.example.jpのIPアドレスを返答
- NSがPCへIPアドレスを返答

# 2種類のネームサーバ(2)

- クライアントの要求により検索を行う
  - www.example.jpのIPアドレスはいくつ?
  - IPアドレスが10.20.30.40のホスト名は?
- DHCP、PPP等でPCに配布するサーバ
  - あるいは各PCにマニュアルで設定
- 結果をキャッシュしてトラフィックを削減
  - 原則、自分自身はデータを持たない
- 「キャッシュサーバ」と呼ぶ
  - 「フルリゾルバ」とも呼ばれる

# キャッシュサーバ




- PCからNSへwww.example.jpのIPアドレスの問い合わせ
  - NSが“.”(ルート)のネームサーバにIPアドレスを問い合わせ
  - “.”のネームサーバがjpのネームサーバを返答
  - NSがjpのネームサーバへIPアドレスを問い合わせ
  - jpのネームサーバがexample.jpのネームサーバを返答
  - NSがexample.jpのネームサーバへIPアドレスを問い合わせ
  - example.jpのネームサーバがwww.example.jpのIPアドレスを返答
- NSがPCへIPアドレスを返答

# NS RR

- **ネームサーバの名前を示す**RR
  - RR - Resource Record

**親ゾーン**(example.jp**の場合、**JP**ゾーン**)**に記述**

```
example.jp.     IN NS       ns.example.jp.
                   A        10.10.10.10
```

**自ゾーン**(example.jp)**に記述**

```
@               IN SOA      .....
                   NS       ns.example.jp.
ns                 A        10.10.10.10
```

# グルー(Glue) RR

- **ドメイン名の検索に必要不可欠な親ゾーンの**NS RR**に付随する**RR

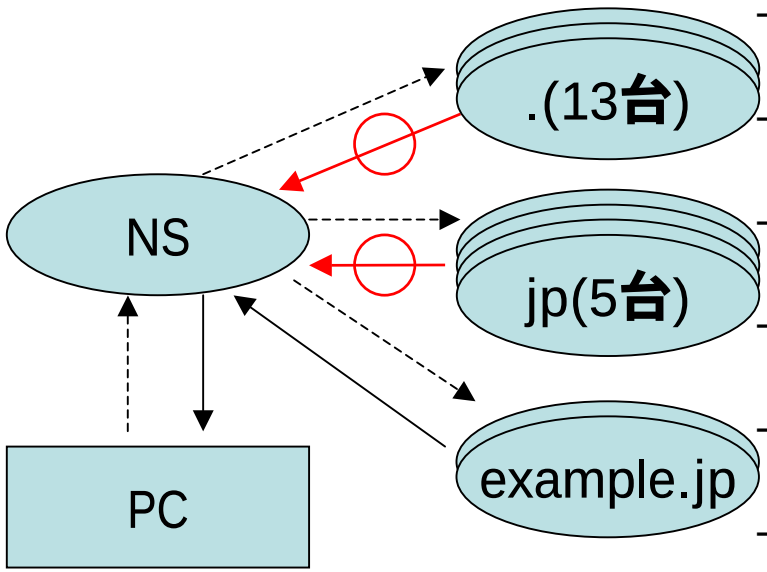
```
$ORIGIN jp.
example.jp.     IN NS       ns.example.jp.
                   A        10.10.10.10
```

  – A RR **または** AAAA RR

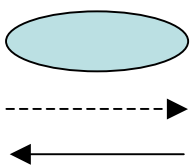
- **自ゾーンの場合グルーではなく正式な**RR

```
$ORIGIN example.jp.
example.jp.     IN NS       ns.example.jp.
                   A        10.10.10.10
```

# NSとグルー




- PCからNSへwww.example.jpのIPアドレスの問い合わせ
  - NSが“.”(ルート)のネームサーバにIPアドレスを問い合わせ
  - “.”のネームサーバがjpのネームサーバを返答
  - NSがjpのネームサーバへIPアドレスを問い合わせ
  - jpのネームサーバがexample.jpのネームサーバを返答
  - NSがexample.jpのネームサーバへIPアドレスを問い合わせ
  - example.jpのネームサーバがwww.example.jpのIPアドレスを返答
- NSがPCへIPアドレスを返答



# グルー無し

- NSが同じドメインでのグルー無し(**内部名**)

```
$ORIGIN jp.
example.jp.     IN NS       ns.example.jp.
```

  – 設定ミス ⇒ 当該ドメイン名の検索が**不能**になる

- NSが違うドメインでのグルー無し(**外部名**)

```
$ORIGIN jp.
example.jp.     IN NS       ns.example.net.
```

  – DNSの仕様 ⇒ グルーRRは**不要**

# 外部名のグルー無し

- ネームサーバが外部名で、グルーが得られない場合

```
$ORIGIN jp.
example.jp.     IN NS       ns.example.net.
```

キャッシュサーバは、NS RRのホスト(例ではns.example.net)のIPアドレスを検索するために、新たな検索を行う

  - 内部名の場合に比べて余計な時間がかかる

# セカンダリネームサーバ

- コンテンツネームサーバを複数設定する
  - 冗長性を増し、負荷分散を行う
  - プライマリネームサーバに対し
    セカンダリネームサーバと呼ぶ
- DNS的にはプライマリとセカンダリは同じ
  - プライマリ同じゾーンデータを保持する
  - 違いはゾーンデータの設定方法
    ゾーンデータをなんらかの方法でコピーする

# 複数のNS RR

- DNSクライアントは、複数のNS RRがある場合、そのうちのいずれかにアクセスを行う
  - どのサーバにアクセスするかは実装による
  - 失敗すると別のサーバへアクセス

例) NS RRにns0とns1がある場合

  - ある名前解決で、ns0にアクセスができて結果が獲られれば、ns1にはアクセスを行わない
  - ns1にアクセスするのは次の機会になる
  - 極端な話、ns0とns1の内容が違っても動作する

# 外部名のNS指定

- ネームサーバのうち1つが外部名

```
example.jp.     IN NS ns1.example.jp.
                   NS dns1.example.net.
```

- すべてのネームサーバが外部名

```
example.jp.     IN NS dns1.example.net.
                   NS dns2.example.net.
```

- よく見られる設定例
  ⇒ 技術的には正しい

# 外部名の設定が意味するもの

- 「example.jpは、example.netのゾーン全体を100%信用する」ということを意味する
  - example.net側のネームサーバがクラックされ、ゾーンを乗っ取られた場合、example.jpの名前空間を乗っ取られるリスクがある
  - example.netのネームサーバに外部名があればさらに可能性が広がる
- MX RR, CNAME RRにも同様のリスク
  - 値にホスト名が指定できるRRすべて

# Pharming
# (**ファーミング**)

# Pharming

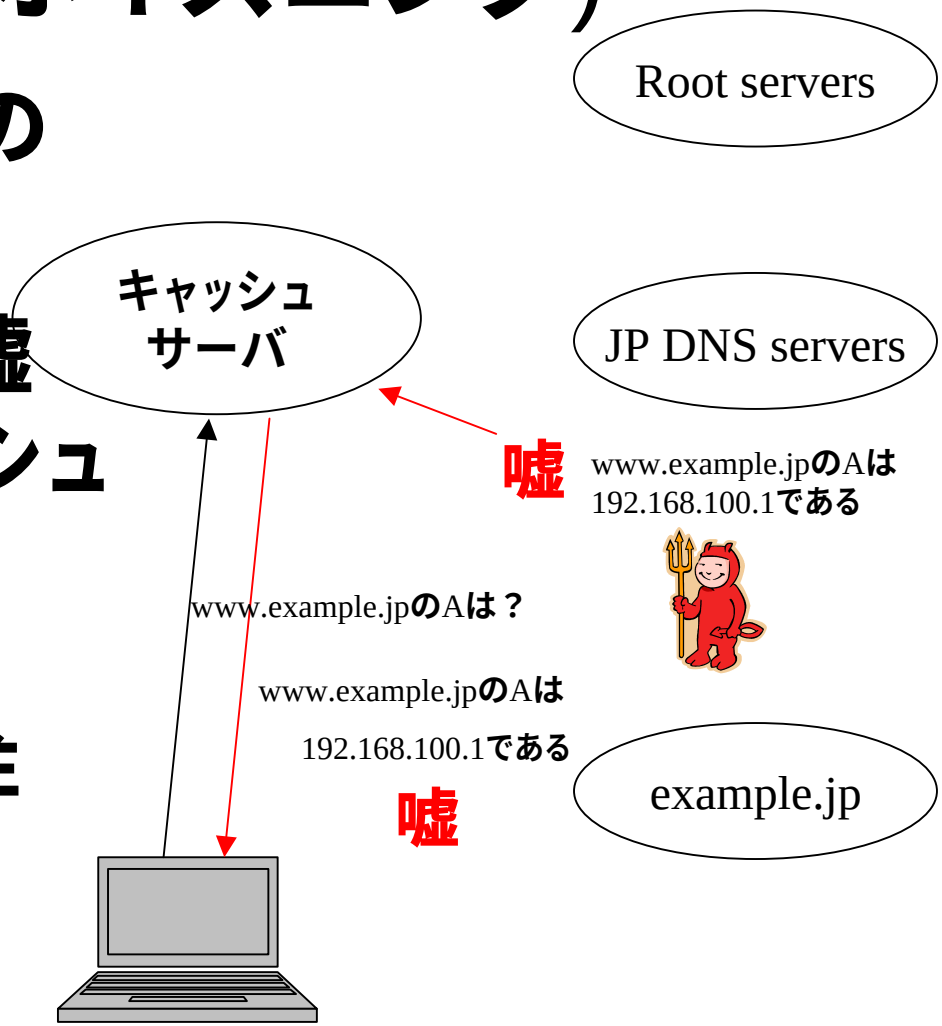
- Phishing (**フィッシング**)
  - fishing(**釣り**)**をもじった造語**
  - **メール等で偽のサイトへ誘導 ⇒ 餌を撒いて釣る**
- Pharming (**ファーミング**)
  - farming(**農業**)**をもじった造語**
  - **(予め)仕込んだサイトへ誘導されるのを待つ ⇒種を撒いて実が生るのを待つ**
  - **言葉が新しいだけで、以前から知られたもの**
  - Phishing**の一種という分類をする場合もある**

# 代表的なPharming

- DNSになんらかの細工を施すことで、別のサイトに誘導する(後述)
- ウィルスなどに感染させてhostsファイルを書き換え、別のサイトに誘導する
  - 多くのシステムではDNSよりhostsファイルを優先
- ブラウザなどで正規のサイトにアクセスしているように見えても、偽のサイトにアクセス
  - IDとパスワードや、クレジットカード情報等を盗む

# Cache Poisoning (キャッシュ・ポイズニング)

- DNSでのPharmingの代表的な手法
- キャッシュサーバに嘘の情報を予めキャッシュさせるテクニック
  - 嘘のグルーを設定し、キャッシュサーバに注入する



# Cache Poisoningの例

- example.gr.jpでexample.jpを攻撃

```
$ORIGN example.gr.jp.
www.example.gr.jp     IN NS     ns.example.jp
ns.example.jp.           A      1.2.3.4
```

- dig @<攻撃対象のキャッシュ> www.example.gr.jp

```
;; AUTHORITY SECTION:
www.example.gr.jp.   1D IN NS     ns.example.jp.
;; ADDITIONAL SECTION:
ns.example.jp.       1D IN A      1.2.3.4
```

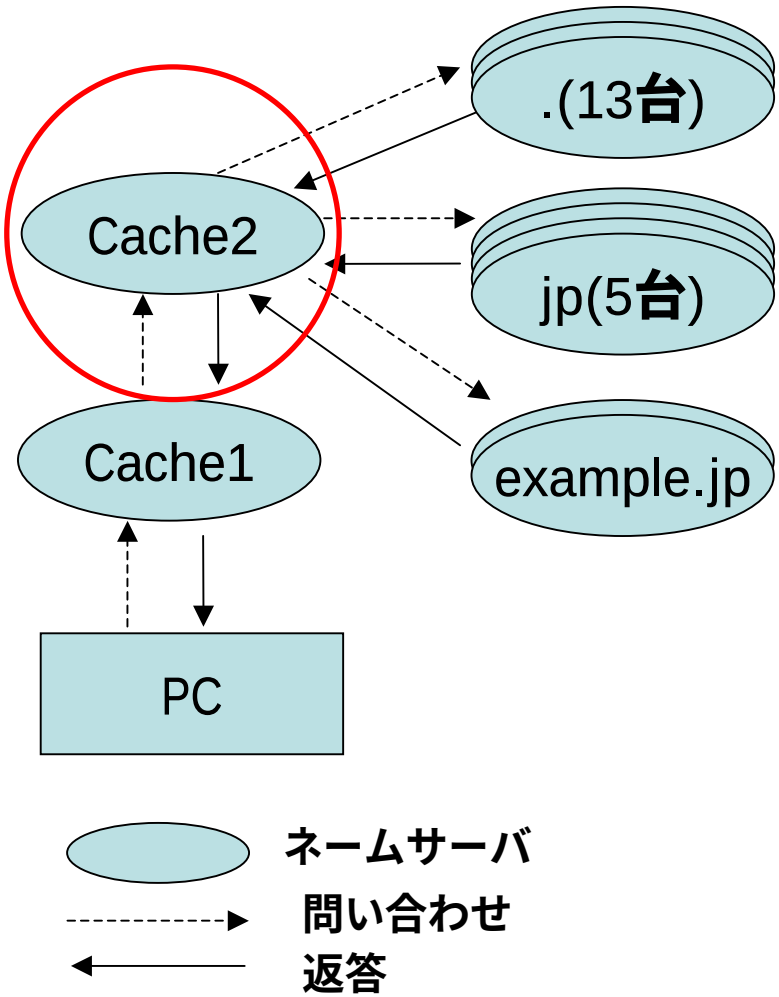
- 結果を鵜呑みにするキャッシュサーバ

# Cache Poisoning対策

- 古くから知られた問題で基本的に解決済み
  - コンテンツサーバはゾーン外の情報を持たない
  - キャッシュサーバはゾーン外の情報を信用しない
- 特定の条件下で発生することがある
  - BIND 8 or 4をフォワーダに使う場合等
- 一部の実装には問題がある
  - Windows 2000のSP2までのデフォルト

    http://support.microsoft.com/default.aspx?scid=kb;ja;jp241352

# フォワーダ(forwarder)




- Cache1が名前解決をCache2に頼る場合、Cache2がフォワーダ
- Cache2がBIND 8 or 4で、特にCache1がWindows DNS サービスだとリスクがある
  - BIND 9では問題無い

# ドメイン名の登録とDNSサーバの設定に関する注意喚起

- IPAの緊急対策情報 2005年6月27日
  http://www.ipa.go.jp/security/vuln/20050627_dns.html
- Pharmingの可能性に関する注意喚起

# IPAの注意喚起 (1)

- ネームサーバの1つ以上(or 全部)が外部名
  - 例)

```
example.jp. IN NS ns1.example.jp.
               NS dns1.example.net.
```

- example.jp側が気づかないうちにexample.netドメインの登録組織が消滅
  - あるいは突然サービス停止等
- 時間を経てexample.netドメインも消滅
  - dns1.example.netが動作しなくなる

# IPAの注意喚起 (2)

- DNSの冗長性のため、ns1.example.jp だけで example.jpドメインは正常稼働
  - 見かけ上まったく問題無い
- ある日、第三者が example.jpのネームサーバのドメイン名であるexample.netが空いていることに気づく
- 第三者はexample.netドメインを登録し、dns1.example.netというサーバを運用する

# IPAの注意喚起 (3)

- **第三者に善意がある場合 ⇒ 稀なケース**
  - **現状の**example.jp**のゾーンデータを調べ、同様の内容を登録する**
- **第三者に悪意がある場合 ⇒ 多くのケース**
  - dns1.example.net**に偽の**example.jp**のゾーン情報を設定する**
  - **ユーザが**www.example.jp**をアクセスすると偽のサイトへ誘導される**
  - Pharming**が成立する(この場合確立**50%)

# IPAの注意喚起 (4)
# 自ドメイン内が確かでも…

- example.netは正常に運用されている

```
example.jp.   IN NS ns.example.jp.
                    NS dns1.example.net.
```

- example.netのネームサーバに外部名

```
example.net. IN NS dns1.example.net.
                   NS ns2.provider.dom.
```

- provider.domをなんらかの手段で制御できれば25%の確立でexample.jp.を乗っ取れる

# IPAの注意喚起 (5)
# バリエーション

- ネームサーバの名前を間違える(入力ミス等)
  - nsexample.jp　　ピリオドが無い
  - ns.exampke.jp　　隣のキーのlとkを間違えた
- ネームサーバのIPアドレスを間違える
  10.10.10.10 のはずが 10.10.19.10
  - 10.10.19.10で偽のネームサーバを運用する

**いずれも気づきにくい！**

# DNS設定のチェックサイト

- **客観的なチェックに頼るのが効果的**
- Squish DNS Checker
  - http://www.squish.net/dnscheck/
- DNS Report
  - http://www.dnsreport.com/
- Zone Check
  - http://www.zonecheck.fr/

# コンテンツサーバと
# キャッシュサーバ

# コンテンツサーバと キャッシュサーバの区別

- BINDには明示的な区別が無い
  - namedの1プロセスで両方を兼用できる
  - WindowsのDNSサービスも同様
- 別プログラムとして実装している例もある
  - djbdnsは別プログラムで実装し、兼用不能
    tinydns　　コンテンツサーバ
    dnscache　キャッシュサーバ
  - NSDはコンテンツサーバ専用の実装

# コンテンツ・キャッシュ兼用ネームサーバ

- 極めて多い運用例
  - 過去には、それぞれの区別があまり意識されていなかった
- 特にBINDの場合…
  - 簡単に設定できる(デフォルトで兼用になる)
  - 設定例も、区別を意識していないものが多い

# 兼用ネームサーバの問題点

- キャッシュ情報管理の問題
  - 検索結果のキャッシュによるメモリ肥大
  - キャッシュした情報がゾーンに混ざる可能性
  - コンテンツサーバに影響が出る可能性
- BIND 8 (4を含む)の実装はすべて該当
  - BIND 9はコンテンツデータと
    キャッシュデータを分離して管理

# コンテンツサーバと キャッシュサーバは分離する

- 近年指摘されている運用TIPS
  - 過去にはあまり言われていなかった
- キャッシュサーバにトラブルがあっても コンテンツサーバに影響ないようにする
- 逆も同じ
  - コンテンツサーバにトラブルがあっても キャッシュサーバに影響ないようにする
- BINDであっても設定で分離できる

# 第三者のキャッシュサーバの利用

- 通常の使用であれば問題は少ない
- 不正にドメインの検索を大量に行われると…
  - 負荷の増大
  - キャッシュメモリの肥大化(BIND 8)
  - プログラムの不具合を突く可能性もありうる
- いずれもサービス不能(DoS)攻撃につながる
  - 兼用の場合、コンテンツサーバへ悪影響も

# キャッシュサーバのアクセス制限

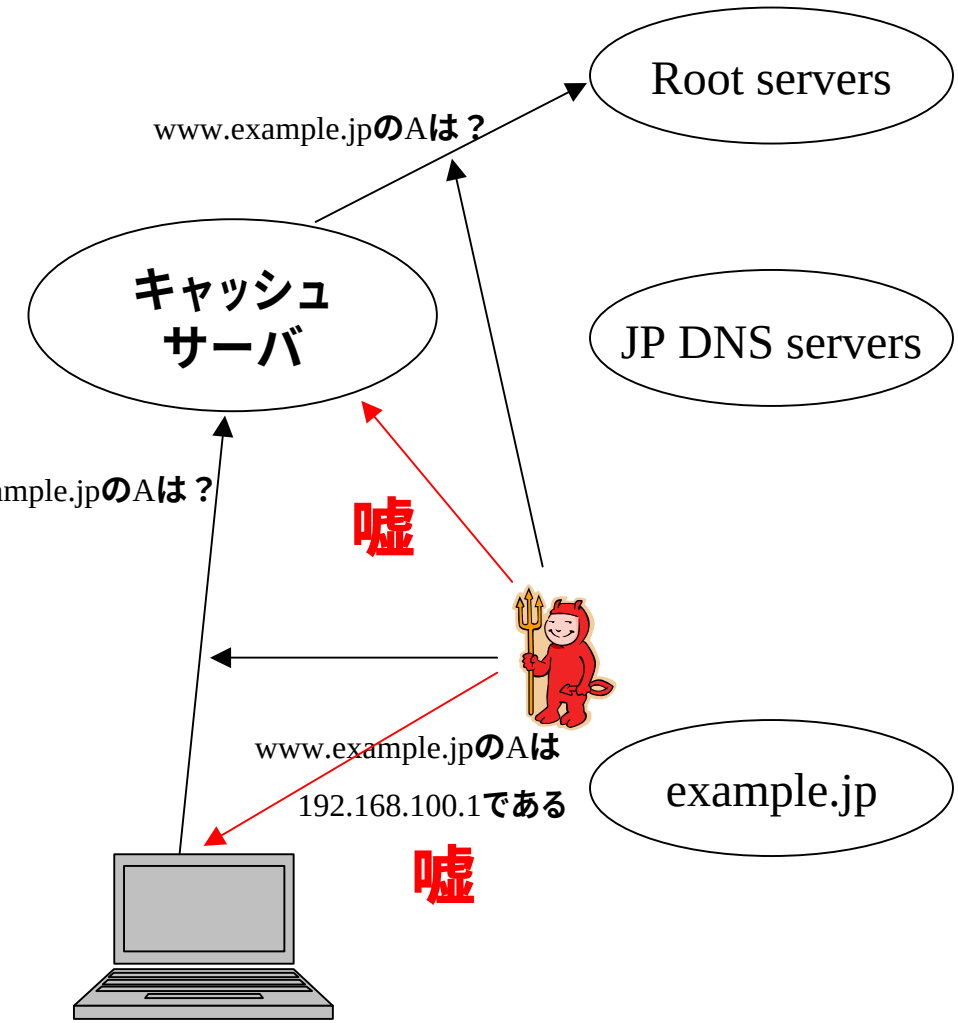
- キャッシュサーバを不特定多数に使用させないよう、適正なアクセス制限をかけ、自ネットワーク内からのみアクセスできる設定を行う
  - Cache Poisoningが起こりにくくなる
  - メモリの肥大化も起きにくくなる
  - 極端な負荷の増加を招きにくくなる
  - 潜在的な不具合にあたりにくくなる

  (身内は善人という前提)

# 虚偽の応答

# 盗聴と虚偽の応答

- **共有イーサネットや無線LANでパケットを盗聴し嘘の応答を先に返す**
  - Man in the Middle Attack
- Pharmingにつながる



# パケットの盗聴

- 共有イーサネットや無線LANなら簡単
- ネームサーバーと同一LANセグメントで
  パケットを覗けば、横取りはたやすい。
  - ネットワーク的にも近いため、
    正しい回答より先に嘘を返せる確立が高まる

# DNSパケットの横取り対策

- 「スイッチングハブならパケットは覗けない」と安心するのは大きな誤り
- ARP Poisoning
  - 3ポート(A,B,C)のスイッチングハブ
    通常、A⇔B間のパケットは、Cに流れない
  - これを盗聴可能にする技術
  - ARP Spoofingとも呼ばれる

# ARP Poisoning (1/4)

- **スイッチングハブに3つの機器が接続**
  - **ホスト**A(**キャッシュサーバー**)
  - **ホスト**B(**管理の甘いサーバ**)
  - GW(**ルータ**)

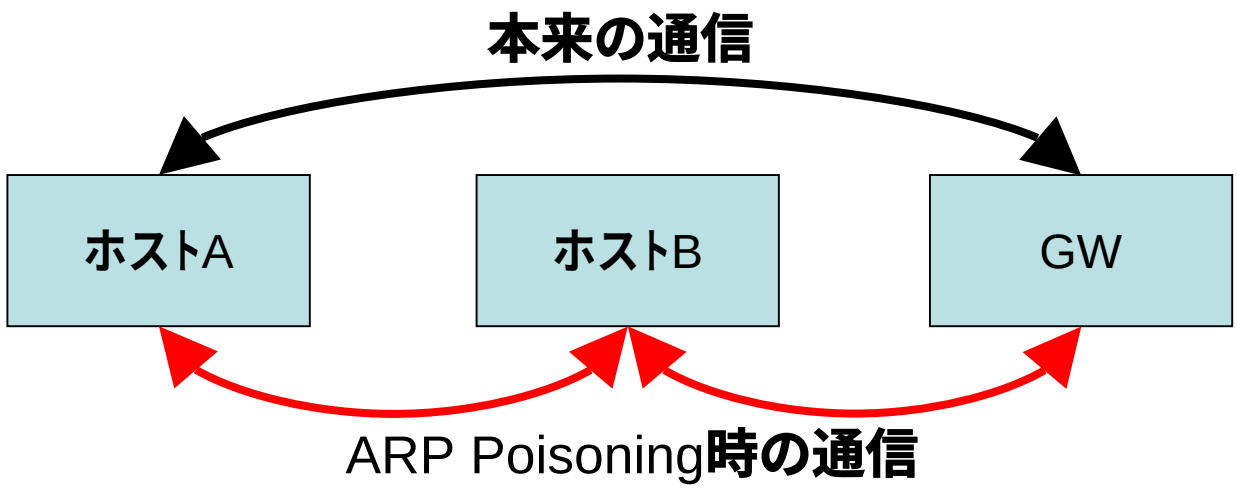
| | IPアドレス | MACアドレス |
|---|---|---|
| GW | 10.10.10.1 | 0:1:1:1:1:1 |
| ホストA | 10.10.10.2 | 0:2:2:2:2:2 |
| ホストB | 10.10.10.3 | 0:3:3:3:3:3 |

# ARP Poisoning (2/4)

- **アタッカーはホストBに侵入しroot権限を入手**
- **ホストBから偽のARP応答を送る**
  - **ホストAへ**
    10.10.10.1 **の**MACaddr**は** 0:3:3:3:3:3
  - GW**へ**
    10.10.10.2 **の**MACaddr**は** 0:3:3:3:3:3
- **ホストAとGWのARPテーブルが書き換わる**
  - **すべての通信はホストBを経由するようになる**

# ARP Poisoning (3/4)

- **ホストBでは入ってくるパケットを覗き、そのまま本来のIPアドレスへ転送する**
  - Layer2**ではホストB宛なので、ネットワークインターフェースをプロミスキャスにする必要も無い**



# ARP Poisoning (3/4)

- ARP Poisoning**されても通常の通信は問題なく行えるため気づきにくい**
  - OS**によっては**ARP**テーブルが変化すると**syslog**に残る**

```
arp: 10.10.10.1 moved from 00:01:01:01:01:01 to 00:03:03:03:03:03 on em0
arp: 10.10.10.1 moved from 00:03:03:03:03:03 to 00:01:01:01:01:01 on em0
arp: 10.10.10.1 moved from 00:01:01:01:01:01 to 00:03:03:03:03:03 on em0
arp: 10.10.10.1 moved from 00:03:03:03:03:03 to 00:01:01:01:01:01 on em0
```

# ARP Poisoning対策

- 同一セグメントに繋がっているホストをすべて正しく管理
  - セキュリティホールを残さないこと
  - パスワード管理も正しく行う
- ARPテーブルをスタティックに登録する
  - 手間はかかるが、管理したマシンしか接続できなくなるため、セキュリティ的には強固になる。

# DNSパケットの内容(簡略版)

- 問合せパケット
  - ドメイン名とRRとID (Query ID)
  - Query IDは16bit
- 応答パケット
  - ドメイン名とRRとIDと回答
- トランスポートは通常UDP
  - ポート番号はDNSなので53
  - ソースアドレスの偽造は易しい

# ドメイン名とIDの予想 (1)

- Cache Poisoningのテクニックの一つ
  - Pharmingにつながる
- **キャッシュサーバが検索するものを予想し、嘘の回答を大量に送り込む**
- **単純に推測しても当たる可能性は低いのでキャッシュサーバに対して、大量の問い合わせを行う**
  - **キャッシュサーバが問い合わせを行うドメイン名がある程度予想できる**

# ドメイン名とIDの予想 (2)

- ドメイン名とRRの予想は容易
- クエリIDは16bitのランダム
- BINDのnamedは送信元ポート番号が固定
  - 調べるのは簡単
- 送信先のポート番号はDNSなので53
- クエリIDが一致すれば嘘の情報を送り返し Cache Poisoningが成立する
  - $1/2^{16}$の確立

# ドメイン名とIDの予想 (3)

- dnscache(djbdnsのキャッシュサーバ)
  - 送信元ポートを毎回ランダムに変更する
  - このため確立は $1/2^{32}$ まで低くなる
- 確立は低くなるが完全とは言えない

**DNSプロトコルの欠陥**

# DNS虚偽応答への対応

- **従来は、嘘をつかれても検出できなかった**
  - **アプリケーションで対策する手法**
    - SSH
    - SSL/TLS
- RFC3833 Threat Analysis of the Domain Name System (DNS)
  - **ドメインネームシステムの脅威の分析**
- **根本対策は**DNSSEC**の利用**

# DNSSECとは

- DNS Security Extension
  - RFC4043,4044,4045で定義
- DNSゾーンに権限を持つ管理者が、公開鍵暗号技術を用いて、自らのゾーン情報に秘密鍵で署名を行うDNSの運用方式
  - ゾーン情報の第三者による改ざん・騙りを検証することが可能となる
  - 万一にも騙りを許したくないDNSレコードを守ることが可能となる

# BINDを設定する

# BIND設定のポイント

- コンテンツサーバか?キャッシュサーバか?
- アクセス制限は適正か
- 万が一、BINDのプロセス経由で侵入された場合に影響はないか?

# BINDでのコンテンツサーバ

- named.confには自組織関連のゾーンを記述
- recursion no;
- fetch-glue no;
  - BIND9では常にno
- hint情報不要(zone “.”)
- セカンダリの場合、zoneの記述部分で、マスターから転送するように設定する

```
options {
    ...
    recursion  no;
    fetch-glue no;
    ...
} ;
zone "example.jp" {
    type master ;
    file "example.jp.zone" ;
} ;
```

# BINDでのキャッシュサーバ

- recursion yes;
- fetch-glue no;
- hint情報が必要
- allow-queryでアクセス制限して第三者に不正に利用させない
- 127.0.0.1 (localhost) の情報も加える
  - ::1もお忘れなく

```
options {
    ...
    recursion yes;
    fetch-glue  no;
    allow-query { 10.0.0.0/8 ;
                  127.0.0.1 ; };
    ...
};
zone "." {
    type hint;
    file "named.root";
};
zone "0.0.127.IN-ADDR.ARPA" {
    type master;
    file "localhost.rev";
};
```

# 1台でキャッシュサーバとコンテンツサーバを運用(1/3)

- namedプロセスを2つ起動する
- コンテンツサーバー用/etc/named.conf

```
options {
    ...
    recursion  no;
    fetch-glue no;
    listen-on { 10.10.10.1 ; } ;
    ...
} ;
```

- listen-onでサーバーのIPアドレスのみ

# 1台でキャッシュサーバとコンテンツサーバを運用(2/3)

- **キャッシュサーバー用**/etc/cache.conf
  - named –c /etc/cache.conf **で起動**

```
options {
    ...
    pid-file     "/var/run/cache-named.pid" ;
    listen-on { 127.0.0.1 ; } ;
    ...
};
```

  - /etc/resolv.conf**では**“nameserver  127.0.0.1”
  - 127.0.0.1**だけなのでアクセス制限は不要**

# 1台でキャッシュサーバとコンテンツサーバを運用(3/3)

- listen-onで利用するインターフェースを分離
- controlsの指定もキャッシュとコンテンツで別のものになるよう設定する
- dump-fileやstatistics-fileにも注意
  - 2つのnamedプロセスで上書きの可能性があるため一方を名前を変更する

# コンテンツサーバ ゾーン転送の制限

- BIND 8にとってゾーン転送は極めて重い処理
  - DoSの可能性がある
  - ゾーンの内容を隠蔽する効果もあるが…
- 適正なアクセス制限を施すべき
- BIND 9はBIND 8に比べるとゾーン転送は軽い

```
options {
    ....
    allow-transfer {
        10.10.10.10 ;
    };
    ...
} ;
zone "example.jp" {
    type master ;
    file "example.jp.zone" ;
    allow-transfer { ... } ;
} ;
```

# キャッシュサーバ
# キャッシュメモリの制限

- BIND 9**でのみ可能**
  - BIND 8**にはキャッシュメモリのサイズを制限する機能は無い**
- **サーバの実メモリ量を考慮して決める**

```
options {
    ....
    max-cache-size 100M;
    ...
} ;
```

# 設定の確認

- dig @10.10.10.1 example.jp ns
  - ;; flags: qr aa rd **ra**; **なら** recursion yes ; **のまま**
- dig @10.10.10.1 <**適当なドメイン名**>
  - **管理ドメイン以外は検索できないのを確認**
- dig @127.0.0.1 <**適当なドメイン名**>
  - **正常に検索できるか**
- **できる限り、自ネットワーク外からチェック**
  - **アクセス制限の効果を確認する**

# 万が一named経由で侵入されたときへの備え (1)

- rootとは別の、named専用ユーザーを用意しその権限で稼動するようにする
  - named –u <user> …
    - /var/run/named.pidなどのオーナーに注意

# 万が一named経由で侵入されたときへの備え (2)

- namedをchroot環境で稼動させて、アクセスできるファイルを制限する
  - named –t <chrootディレクトリ> …
- BIND9での設定例

  http://www.unixwiz.net/techtips/bind9-chroot.html
- djbdnsは元々chroot環境下で動作する

# BINDのバージョン

- 大きくわけて4系、8系、9系がある
- 8系まではキャッシュとして使うには不具合が多い
  - セキュリティホールが無くても潜在的な可能性がある
- いずれにしても最新版を使うようこころがける

○: 推奨できる　×: 推奨できない

△ 新たに使うメリットがない

| | 4系 | 8系 | 9系 |
|---|---|---|---|
| キャッシュ | × | × | ○ |
| コンテンツ | △ | ○ | ○ |

# djbdns

# djbdns を BIND と比べると (1)

- コンテンツ、キャッシュサーバを分離した実
  - 分離したものとしては初の実装
  - tinydns　　コンテンツサーバ
  - dnscache　　キャッシュサーバ
  - コンテンツとキャッシュではキャッシュサーバのほうが複雑な実装となる
- デフォルトでdnscacheはアクセス制限する
  - 明示的にアクセスするネットワークを指定する必要がある

# djbdns を BIND と比べると (2)

- デフォルトでchroot環境下で動作
  - セキュリティ的に安心できる
- セキュリティホールが無いことを保障している
  - セキュリティに関して設計段階から注意深く実装されている
  - 不具合が無いことを保障しているわけではない

# dnscacheのアクセス制限

- root/ip にファイルを用意する
  - "/"を使ったネットマスクの制限はできない
  - root/ip/10　　10.0.0.0/8
  - root/ip/10.20　　10.20.0.0/16
  - root/ip/10.20.30　　10.20.30.0/24
- 10.20.30.32/29の場合8個のファイルを用意
  - root/ip/10.20.30.32 〜 root/ip/10.20.30.39
  - 機械的に作成できる

# djbdns

- BINDに比べ、デフォルトのセキュリティに対する対策はかなりすぐれている
  - 処理速度など別の問題はある

# Q & A