修士論文

トンネリング抑止を目的とした
分散ハッシュテーブルを利用したDNSに関する研究

高須賀 昌烈

2020年1月28日

奈良先端科学技術大学院大学
先端科学技術研究科

本論文は奈良先端科学技術大学院大学先端科学技術研究科に
修士 (工学) 授与の要件として提出した修士論文である。

高須賀 昌烈

審査委員：

門林 雄基 教授　（主指導教員）
笠原 正治 教授　（副指導教員）
林 優一 教授　（副指導教員）
妙中 雄三 准教授　（副指導教員）

トンネリング抑止を目的とした
分散ハッシュテーブルを利用したDNSに関する研究*

高須賀 昌烈

内容梗概

キーワード

ドメインネームシステム，DNSトンネリング，分散ハッシュテーブル，P2Pネットワーク

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 2020年1月28日.

A Study of Domain Name System using Distributed Hash Table for Tunneling Deterrence*

Shoretsu Takasuka

Abstract

Keywords:

Domain Name System(DNS), DNS Tunneling, Distributed Hash Table, P2P Network

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, January 28, 2020.

目 次

図 目 次

表 目 次

アルゴリズム目次

1. 序論

1.1 背景

増加し続けているサイバー攻撃に対して，現在多くの組織は，SIEM[*1]のようなネットワークトラフィックを監視するシステムから発せられるアラートを処理することで脅威に対処している．他方で，機密情報の奪取や諜報活動を行う攻撃者は，そのような監視システムを迂回するために，秘匿通信手法[*2]を用いることが知られている [1]．このような攻撃に対して，検知の閾値や悪性モデルを調整するアプローチが考えられるが，誤検知とのトレードオフの関係にある．通信プロトコルは，このような秘匿通信として利用されないように，セキュア指向で設計されているが重要である．

秘匿通信として代表的な手法が，DNS トンネリングである．本来，DNS(Domain Name System) は，IP アドレスをはじめとしたドメイン名に関連づけられたリソースレコードを解決するシステムである．DNS によるこの仕組みは名前解決と呼ばれ，ユーザはこの仕組みを利用することで，識別子づらい IP アドレスを直接利用することなくサーバのリソースにアクセスすることができる．インターネットの利用において，名前解決はメールの送受信やウェブ検索など全ての通信に先立って行われるため，一般に DNS のトラフィックをフィルタリングすることはインターネットの利活用に支障が生じる．DNS トンネリングは，DNS がフィルタリングされにくいというこの特性を利用することで，容易に秘匿通信として機能する．また，DNS の名前解決の仕組みは，クライアントからの問い合わせがサーバで処理され，サーバは問い合わせの結果をクライアントに応答することによって成り立っている．DNS トンネリングでは，この名前解決においてデータが双方向に転送される仕組みを悪用することで，任意データの転送を実現させている．このため，DNS トンネリングは，ターゲット組織から取得したデータを外部に流出させる際の手段としてだけでなく，ターゲットネットワークに潜伏しているマル

[*1] SIEM(Security Information and Event Management)：複数のソフトウェアや機器からトラフィックなどのログ情報を一元的に管理し，異常や脅威が発生した時に管理者に通知する仕組み
[*2] 秘匿通信：情報を不正・秘密裏に転送するために使用される回避通信手法

ウェアに対する C2 サーバ[*3]からの命令を送る手段として，サイバー攻撃で広く利用されている [2–9]．このような課題があるにも関わらず，DNS の名前解決の仕組みは，設計当初のまま変更されることなく使用されている現状にある．DNS トンネリングに対して，同一ドメインに対する時間あたりのトラフィック頻度や問い合わせられるドメイン名のサブドメインにおける文字列の分布や長さといった特徴に基づいた検知アプローチがこれまでに多数提案されている [10–15]．それら検知手法における評価では，DNS トンネリングの擬似通信として，Github[*4]より入手可能なトンネリング実装プログラムが用いられている．多くの先行研究で検知対象となっている Iodine [16] や DNSCat2 [17] などのトンネリング実装は，DNS を用いたインタラクティブシェルを目的としており，そのため時間あたりに高いトラフィック量となる特性がある．このような顕著な特性を示すトンネリング実装に対して，パケットごとのインターバルを 1ヶ月間などトラフィック頻度を調整したり，正規の FQDN の平均の長さまで注入するデータ量を下げるなどによる秘匿性を高めたアプローチがある [13]．これら秘匿性を高めた秘匿手法を利用した場合，トンネリング実装のような顕著な特徴量が現れないため，検知することが極めて困難である．

これまでに多数の次世代名前解決システムは提案されてきているが，DNS トンネリング抑止を目的としたシステムは筆者が知りうる限り提案されていない．そこで本研究では，秘匿通信手法である DNS トンネリングの発生を抑止する次世代の名前解決システムを提案する．

1.2 目的

本研究は，DNS の名前解決の仕組みを悪用した秘匿通信手法の DNS トンネリングを抑止しながら，従来通りドメイン名に関連づけられたリソースレコード情報を解決できる機能を両立する名前解決システムを開発することを目的とする．

既存の DNS による名前解決システムは，現在のインターネットの根幹技術であるため，その名前解決エコシステムに大幅な変更を加えることは，高い導入コ

[*3] C2 サーバ：Command & Control サーバ
[*4] Github：ソースコード共有プラットフォーム

ストが要求されるため望まれない．例えば，期待されないシステムとしては，以下のようなものが予想される．

- DNS トンネリングの抑止の機能は実現できるが，リソースレコード情報の解決の機能がない
- DNS トンネリングの抑止の機能は実現できるが，既存システムからの大幅な変更が必要になり，未対応のコンピュータのインターネット接続に支障をきたす

以上のことを踏まえて，既存システムのエコシステムとの互換性を保ちながら，目的を達成することが重要である．

1.3 貢献

提案システムは，既存システムにおけるクライアント・サーバアーキテクチャを流用しながら，既存の再帰問い合わせの仕組みのみに改変処理を施すことで，DNS トンネリングを抑止する名前解決システムの実現を目指す．すなわち，提案システムは，スタブリゾルバとフルサービスリゾルバ間における処理はそのままに，フルサービスリゾルバと権威サーバ間における処理にのみ変更を加えることで，DNS のエコシステムとの互換性を保ちながら，DNS トンネリングを抑止することを実現する．提案システムでは，既存クライアントに変更を加えずに秘匿通信としての機能を抑止することができるため，セキュアな名前解決システムとして広く一般に利用されることが期待される．

1.4 本論構成

本稿の構成は次の通りである．第 2 章では，本論で対象とする脅威モデルである DNS トンネリングを説明し，これまでの検知に基づくアプローチの限界を示す．第 3 章では，これまでに提案されてきた名前解決システムについて説明し，DNS トンネリングに対する課題を示す．第 4 章では，提案手法を説明する．第 5 章では，提案手法の DNS トンネリングに対する評価を行い，併せて提案手法の

特定についても説明する．第 6 章では，提案手法の課題について議論する．最後に，第 7 章にて結論を述べる．

2. 脅威モデル

本章では，はじめに DNS の概要について述べる．次に，本研究における脅威モデルである DNS トンネリングについて説明する．

2.1 DNS の概要

DNS は，ドメイン名に関連づけられたリソースレコードを解決するシステムである [18, 19]．DNS がユーザから問い合わせられたドメイン名の IP アドレスを解決することで，ユーザは識別しづらい IP アドレス (IPv4:“93.184.216.34”, IPv6:“2606:2800:220:1:248:1893:25c8:1946”) を直接入力することなく，サーバにアクセスすることができる．このような利便性を実現する DNS による名前解決の機能は，ユーザがインターネットを利活用する上で極めて重要である．

2.1.1 名前空間

DNS において，各種リソースレコードが関連づけられるドメイン名は，ルートを頂点とする逆ツリー構造で構成されている．ドメインの序列を表記する際には，上位ドメインを右に，下位ドメインを左にする．

“example.com.”を例にとると，ドットで表記されるルートが最も右に位置し，ルートの一つ下層に位置する TLD(Top Level Domain) である “com”が後続する．ドメインの区切りには，ドットが使用され，TLD の一つ下層に位置づく SLD(Second Level Domain) として “example”が連結していることを “example.com.”は表している．図 1 に示す通り，全てのドメインには，ゾーンと呼ばれる名前空間があり，このゾーンはそれぞれのドメインに配置される権威サーバによって管理される．DNS には，委譲という仕組みを利用することでゾーンを分割できる特徴がある．例えば，“example”と “google”というドメインが連結している “com.”について考える．委譲の仕組みを使用しなかった場合には，“com.”が “www.example.com.”や “mail.google.com.”といった “example.com.”と “google.com.”に連結した全てのドメイン名を管理する．しかし，“com.”がそれぞれのドメインにゾーンを委譲した

場合，先の “www.example.com” は “example.com.” が管理し，“mail.google.com.” は “google.com.” が管理することになる．DNS では，この委譲の仕組みを利用することで，ゾーンの肥大を抑止させ，サーバの負荷分散を実現する設計になっている．

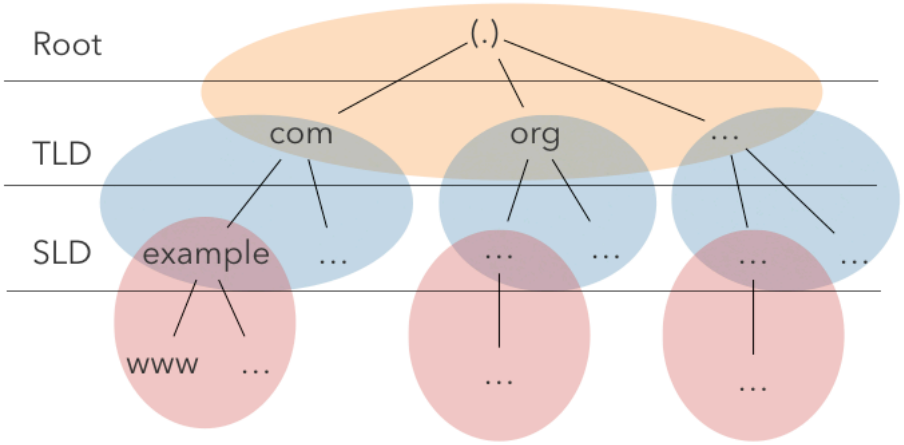

図 1 ゾーンごとに名前空間を色で区分した様子．Root 権威が管理するオレンジで表現されたゾーンには，com や org をはじめとしたドメインが含まれる．Root の直下に位置する TLD の一つである com ドメインは，青色で囲まれた example ドメインを含むゾーンを管理する．com に紐づけられる example ドメインは，www などを管理する．

アーキテクチャ

DNS は，クライアント・サーバアーキテクチャで構成され，機能に基づき 3 つのサービスに分類することができる．

- スタブリゾルバ
- フルサービスリゾルバ
- 権威サーバ

スタブリゾルバは，名前解決の問い合わせを行うクライアントノードである．フルサービスリゾルバ (キャッシュサーバ・リカーシブサーバとも呼称される) は，スタブリゾルバに代わって，リソースレコードを保持する権威サーバに問い合わ

せるクライアントノードである．名前解決をする際には，ルートから順に TLD，SLD という具合に権威サーバに再帰的に問い合わせることで，最終的に目的のドメイン名に関するリソースレコード情報を取得する．この時，はじめのルート権威サーバのアドレスは “root.hints” と呼ばれるファイルに基づいて問い合わせるが，それより下位のドメインについては，上位の権威サーバが次の権威サーバのアドレスを応答することで名前解決のチェーンを繋げている．すなわち，ルート権威サーバが TLD の権威サーバのアドレスを応答し，TLD の権威サーバが SLD の権威サーバーのアドレスを応答していく具合である．権威サーバは，リソースレコードを保持するサーバノードであり，フルサービスリゾルバからの問い合わせ依頼に応答する．

2.1.2 ドメイン名とリソースレコード

ルートからホストまでの階層構造の繋がりは，“label3.label2.label1.” のように右から左方向に連結することで表現される．この表記は，FQDN(Fully Qualified Domain Name) と呼ばれ，右端のドットがルート，“label1” が TLD，“label2” が SLD を表現し，ドメインごとの階層にはドットが使用される．ドメインの階層構造において，全てはルートを頂点としているため，ルートを意味するドットを略記した “label3.label2.label1” が，一般にはドメイン名として解釈される．次にドメイン名に関する長さおよび使用できる文字列について説明する．

$$(LabelD).(LabelC).(LabelB).(LabelA). \quad (1)$$

$$(Length) + (LabelD) + ... + (length) + (LabelB) + (length) + (LabelA) + 0 \quad (2)$$

$$1 + (Max63) + ... + 1 + (Max63) + 1 + (Max63) + 1 = (Max255) \quad (3)$$

(1) は，複数のラベルで構成されたドメイン名の例である．(2) は，Question ヘッダーに注入される際のそのドメイン名を表すデータである．Question セクションでは，ドメイン名を表す際にドットは省略され，ラベルの長さとラベル名，そしてドメイン名の終わりを意味する “0” で表現される．(3) は，ラベルの長さとラベル名のサイズを表す．ラベルの長さは，1 バイトのサイズで表現され，ラベル自体の最大長は 63 バイトである．Question セクションの最大長 255 バイトは，ラ

ベルの長さとラベル，そしてドメイン終了を表す “0” を含めた長さである．このため，最初のラベル長を表す 1 バイトとドメイン名の終了を意味する “0” を表すための 1 バイトを差し引いた 253 バイトが，実際のドメイン長の最大長である．

ラベルには，数字とアルファベットおよびハイフン (“-”) を使用することができ，ラベル中に大文字・小文字の区別はない．他方で，アルファベットなどの ASCII 以外にも，国際化ドメイン名 (IDN: Internationalized Domain Name) を使用すると日本語やアラビア語なども使用することができる．IDN は，Punnycode[*5] などのエンコーディング手法に基づき，DNS クエリする際には ASCII コードに変換される [20].

表 1 主要リソースレコード一覧

タイプ	目的
A	ホストの IPv4 アドレス
NS	権威サーバ
CNAME	別名
SOA	権威ゾーンの開始
NULL	NULL(実験用)
PTR	ドメイン名のポインター (逆引き)
MX	メール交換
TXT	任意文字列
AAAA	ホストの IPv6 アドレス
SRV	ドメイン名に対するサービスの場所
RRSIG	リソースレコード署名
NSEC	DNSSEC のための不在署名
DNSKEY	DNSSEC のための公開鍵
NSEC3	NSEC のバージョン 3
CAA	証明書を発行する認証局の指定

[*5] Punnycode: Unicode 文字列を一意かつ可逆的に ASCII 文字列に変換する符号化方式

ドメイン名に関連づけられる情報はリソースレコードと呼ばれ，目的に応じて複数のタイプが定義されている．例えば，ドメイン名に IPv4 アドレスを関連づけることを考える．そのドメインの権威サーバは，関連づけたい IPv4 アドレスをレコードタイプ A として，ゾーンファイルに記述する．クライアントは，ドメイン名とレコードタイプとして A を指定しサーバに問い合わせることで，権威サーバが事前に登録した A レコードに記述されたアドレスを取得することができる．DNS では，IPv4 アドレス以外にも様々な情報をドメイン名に関連づけることができる．代表的なソースレコードのタイプを表 1 で示す．

次に，DNS のパケットフォーマットについて説明する．図 2 で示すように，DNS のパケットは 5 つのセクションに分けられる．クライアントが DNS の名前解決

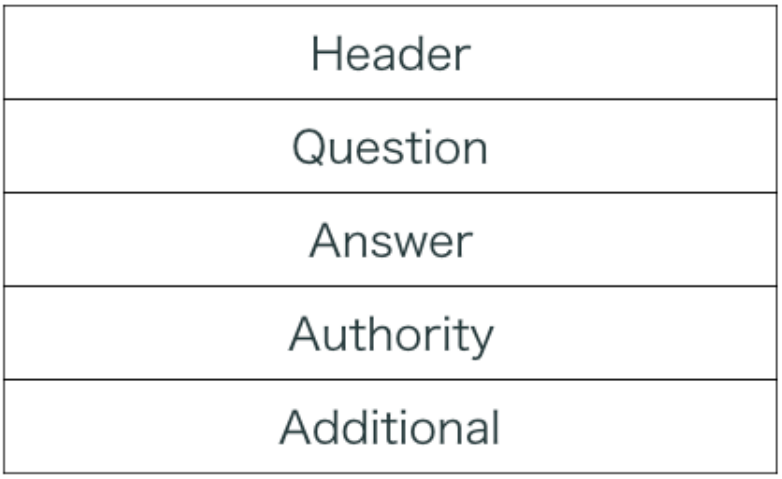

図 2 DNS のパケットフォーマット

をする際，解決したいレコードタイプとそのドメイン名は図 3 における Qname フィールに含まれる．各フィールドはサイズが決まっており，Qname フィールドが最大長 255 バイト，リソースレコードのタイプを表す Qtype フィールドとクエリクラスを表す Qclass フィールドがそれぞれ 2 バイトとなっている．

図 3 DNS のパケットの Answer セクション (bytes)

2.1.3 名前解決の仕組み

参考として，クライアントが “www.example.com” の IPv4 アドレスを解決する場合を考える．

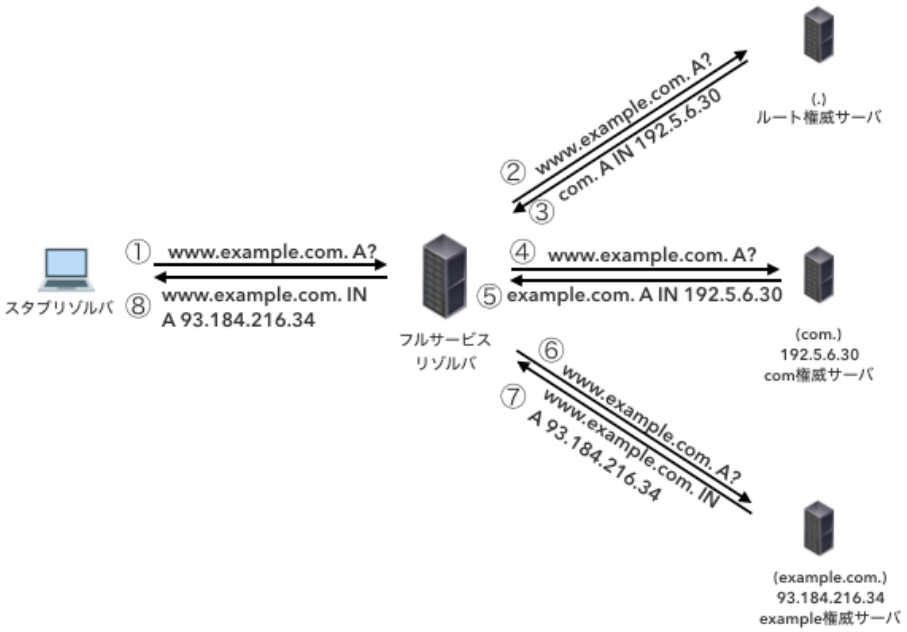

図 4 DNS による名前解決

はじめに，クライアントとなるスタブリゾルバは，スタブリゾルバと同一セグメント内のフルサービスリゾルバもしくは，ネットワークセグメントに依らないどこからでもアクセスできるフルサービスリゾルバ (オープンリゾルバ，パブリッククリゾルバとも呼称される) に問い合わせる．フルサービスリゾルバは，その名前解決クエリが過去に解決したものでないかキャッシュデータを確認する．キャッシュにヒットした場合にはキャッシュの情報をクライアントに応答され，ヒットし

なかった場合には，root.hints ファイルを参照しルート権威サーバにリクエストパケットを転送する．クエリ (問い合わせ) を受け取ったルート権威サーバは，“com” ドメインを委譲した権威サーバのアドレスを応答する．次に，フルサービスリゾルバは，“com” の権威サーバに対し同じクエリを転送する．“com” の権威サーバは，“example.com” ドメインを委譲した権威サーバのアドレスを応答する．フルリゾルバは，“example.com” の権威サーバに同じクエリを転送する．“example.com” の権威サーバは，保持するゾーンファイルからクエリされたドメインのリソースレコードについて探索し，探索の結果としてレコード情報をフルサービスリゾルバに応答する．フルサービスリゾルバは，権威サーバから応答された情報をスタブリゾルバに転送することで，問い合わせられた名前は解決される．DNS による名前解決の一連の動作を図 4 で示す．

2.2 DNS トンネリング

現在の情報流出を目的としたマルウェアのほとんどは，秘匿通信手法を利用していると言われている [13] DNS トンネリングは，そのような秘匿通信の代表的な手法である．DNS トンネリングは，DNS をデータ転送のメディアとした秘匿通信手法の総称であり，転送元と転送先の方向によって二つに分類することができる．スタブリゾルバから権威サーバへの通信の DNS Exfiltration と，権威サーバからスタブリゾルバへの通信の DNS Infiltration である．DNS トンネリングは，以下に示す DNS の特性に基づいる．

- 通常のインターネットの利活用において名前解決は必要な機能であるため，一般に DNS のサービスポートが閉ざされることがない
- 名前解決のトラフィックはほとんどのサービスに先立って発生するため，クエリログが肥大化しやすく長期のログ保存が困難である
- パケットフォーマットの構造において，任意の文字列を注入できるフィールドを保持する

DNS トンネリングがデータ転送のキャリアとするフィールドは，クエリの Question セクションの Qname と，Answer セクションの Rdata である．Question セクションの Qname フィールドを利用することで，スタブリゾルバから権威サーバ方向にデータを転送できる．この方向の通信は，ビーコン通信やターゲットから取得した情報を外部に漏えいさせるといった攻撃の最終目的を達成するのに使われる．また，Answer セクションの Rdata フィールドを利用することで，データを転送することができる．この通信は，ターゲットネットワーク内のホストに潜伏したマルウェアなどへの命令コードを送信するのに使われる．さらに，この二つのキャリアを利用することが双方向の通信路を確保できるため，C2 通信を実施することも可能である．

DNS トンネリング手法が初めて一般に公開されたのは，1998 年に，ポートスキャンで知られる Nmap のメーリングリストだとされている [21, 22]．2004 年には，Dan Kaminsky が OzymanDNS [23] と呼ばれる DNS トンネリングの実装を公開した [24] ことをきっかけに広く知られるようになった．それ以降，数多くの

DNS トンネリングの実装 [16, 17, 25–37] が公開され，実際のサイバー攻撃に悪用されるようになっている．

2.2.1 DNS Exfiltration

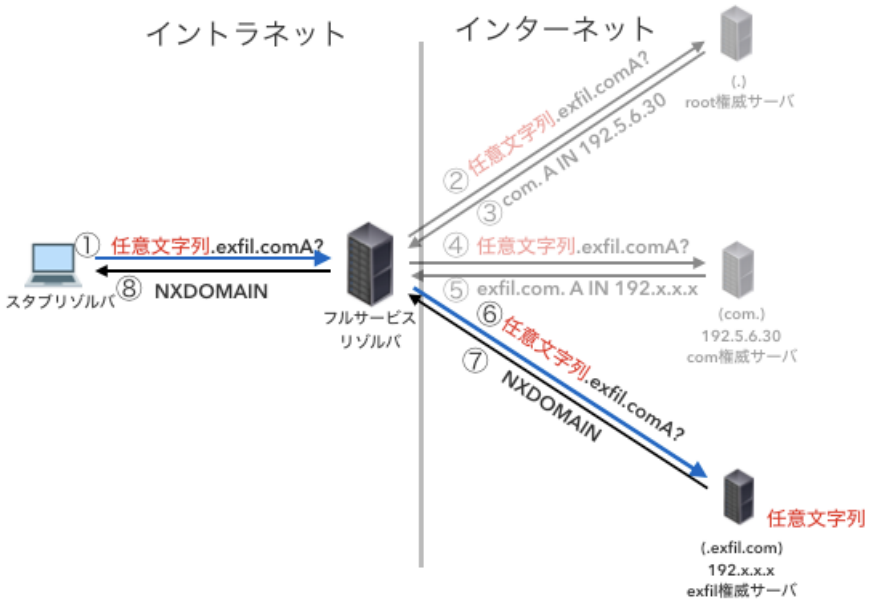

図 5　DNS Exfiltration メソッドに基づいて，ドメイン名のラベル部に任意文字列がサブドメインとして注入された DNS クエリが，イントラネット内のスタブリゾルバからインターネット上の権威サーバ (“exfil.com”) に転送される様子

本項では，スタブリゾルバから権威サーバ方向にデータを転送する手法である DNS Exfiltration の詳細について説明する．DNS Exfiltration は，名前解決として問い合わせられるドメイン名が，そのドメインのゾーンを管理する権威サーバに転送される仕組みを利用した手法である．DNS では，ドメイン名に関連づけられるリソースレコードの情報は，そのドメインをゾーンとする権威サーバが保持しており，ルートから再帰的に問い合わせていくことでその権威サーバからの応答を受け取る．このため，問い合わせられたドメイン名が実在しない場合でも，再帰問い合わせの仕組みに従って，そのドメイン名の最後の権威サーバまで転送されることになる．権威サーバでは通常，クエリされたドメイン名の実在有無に

寄らず，問い合わせられたクエリ情報をログとして管理する．このような特性に踏まえて DNS を利用すると，DNS クエリのドメイン名のラベルに組織外ネットワークに転送したい文字列を注入することで，組織外ネットワーク上に設置された権威サーバにそのデータを転送することができる．これが DNS Exfiltration の動作原理である．

このような仕組みの DNS Exfiltration を動作させるには，宛先となる権威サーバを用意する必要があり，グローバルなドメインを取得することを前提としている．第 2.1.3 項で述べるように，ドメイン名の最大長は 253 バイトであり，その内ラベルの最大長は 63 バイトまでという制約がある．そのため，DNS Exfiltration 手法を用いてデータを転送する際には，TLD のラベルと宛先権威サーバのラベルもしくは SLD ラベルと権威サーバのラベルを差し引いたサイズが実際に転送できる最大長となる．また，任意の文字列を DNS Exfiltration メソッドを用いて外部に転送するにあたり，転送キャリアであるドメイン名における文字列制約を満たすように転送したいデータに前処理を施す必要がある．ドメイン名に使用できる文字列は，第 2.1.3 項で述べるように，“a” から “z” までのアルファベットと “0” から “9” までの数字と先頭以外のハイフン “-” 記号である．この文字列制約については，転送したいデータをバイナリデータに変換し，そのバイナリデータをラベルとして印字可能な ASCII コードに変換することでその制約を満たすことができる．この前処理について，既存の DNS トンネリング実装の多くが Base Encoding [38] を用いている．この処理によって，転送データがバイナリデータである際にも転送効率上げたり，ラベルの文字列制約を満たさないデータも転送することができる．また，エンコーディングのラベルは，自然言語とは異なるため，メッセージの意味抽出を困難にすることにも機能する．

ここで，DNS Exfiltration を用いて，あるイントラネット内のホストからイントラネット外のホストにデータを転送することを考える．転送される宛先となるイントラネット外のホストには，“exfil.com” より下位の全ての名前空間をゾーンとする権威サーバ (“exfil.com”) を指定する．転送したい文字列にエンコーディング前処理を施した後，“用意した文字列.exfil.com” という具合に文字列をラベルとして含めることで，ドメイン名が用意できる．適当なリソースレコードタイプを

指定し，DNS クエリとして転送すると，その権威サーバにはログとして，文字列を含んだドメイン名を取得する．最後に，受け取ったサーバサイドは，前処理と逆のデコード処理を施すことで，オリジナルのデータを取得できる．以上のように再帰問い合わせとラベルという転送キャリア，エンコーディング処理を組み合わせることで，イントラネット内のホストから外部ネットワークに任意の情報を転送することができる．これが，DNS Exfiltration の動作メカニズムである．図 5 に，DNS Exfiltration のメカニズムを図解した様子である．

2.2.2 DNS Infiltration

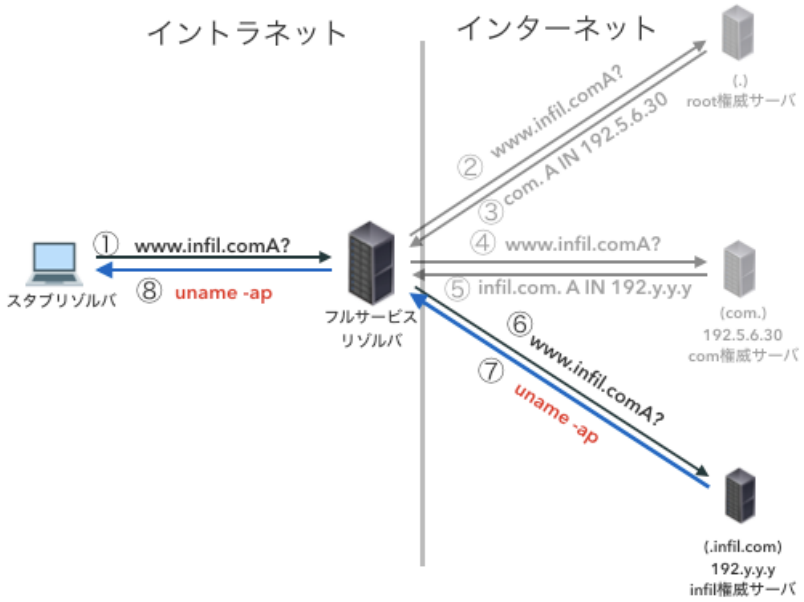

図 6 TXT レコードに登録された情報について，DNS クエリで問い合わせることで権威サーバから命令情報を取得している様子

本項では，権威サーバからスタブリゾルバ方向にデータを転送する手法である DNS Infiltration の詳細について説明する．

DNS Infiltration は，DNS における幾つかのリソースレコードが任意の文字列を記述できる設計を利用したデータ転送手法である．ドメイン名に関連づけられた情報を管理･提供する権威サーバは，ゾーンファイルに関連づけたい情報を記述

する．リソースレコードには，レコード情報を検証する機構が備わっていないため，任意の文字列を登録することができる．特に，記法が決まっていない TXT タイプや NULL タイプなどもあり，DNS Infiltration ではこのようなレコードタイプに転送したいデータを登録しておく．このようにして登録されたレコード情報について，名前解決問い合わせすることによって，インターネット (権威サーバ) からイントラネット (スタブリゾルバ) にデータを転送することができる．DNS Infiltration として利用され得るレコードタイプについて，これまでのトンネリング実装で使用されたものに基づいてまとめたのが，表 2 である．

表 2 DNS Infiltration として使用することができるレコードタイプの一覧

タイプ	最大サイズ (byte)	説明	実装
A	4	ホストの IPv4 アドレス	
NS	4	権威サーバ	[17]
CNAME	253	別名	[16], [17], [31], [37]
NULL*	255	NULL(実験用)	[16]
PTR	4	ドメイン名のポインター (逆引き)	
MX	253	メール交換	[16], [17]
TXT*	255	任意文字列	[16], [17], [28], [29], [31], [32], [34]
AAAA	32	ホストの IPv6 アドレス	
SRV	180	ドメイン名に対するサービスの場所	[16]
DNSKEY	40	DNSSEC のための公開鍵	[36]

NS・CNAME・MX レコードでは，DNS Exfiltration と同じ要領でドメイン名のラベルに転送したい文字列を注入できる．また，NULL・TXT[*6]・SRV・DNSKEY

[*6]EDNS0 を使用する場合，65535bytes が最大長となる

を用いる場合には，レコード構文に指定がないため任意の文字列をそのまま注入できる．最後に，A・AAAA・PTR レコードを用いる場合には，転送したい文字列を数字に変換させた後に，ドット (.) 区切りもしくはコロン (:) 区切りで注入できる．

いま，TXT レコードタイプを用いて DNS Infiltration することを考える．(4) は，TXT レコードを用いてホストで実行させる Linux コマンドを転送する例である．転送される uname プログラムは，システム情報を取得するプログラムである．実行結果を DNS Exfiltration 手法を用いて転送することで，ホストマシンにおけるシステムのカーネルバージョンやプロセッサの種類などの情報を取得することができる．

$$www.exfil.com \qquad IN \quad TXT \quad uname \ -ap \tag{4}$$

次に，スタブリゾルバは，“www.exfil.com” の “TXT” レコードタイプを通常通り問い合わせる．再帰問い合わせの仕組みに基づいて，その DNS クエリは “exfil.com” まで転送され，ゾーンファイルの TXT レコードタイプの値がフルサービスリゾルバを経由したのち，スタブリゾルバまで応答される．DNS Infiltration の流入通信を図解した様子が，図 6 である．このようにして，DNS Infiltration では，正規の名前解決の方法を用いて，インターネットから組織内へとデータを取得することができる．

2.3 DNS トンネリングへの既存対策

本節では，DNS トンネリングに対するこれまでの対策手法を紹介し，検知手法として用いられるアプローチについて説明する．最後に，それら検知アプローチを迂回する脅威モデルを示し，検知に基づくアプローチに限界があることを明らかにする．

DNS トンネリングに対して，森下らは提案されている DNS トンネリングへの対策アプローチを以下のようにまとめている [39]．

1. DNS クエリログの取得と保存・内容の調査
2. エンタープライズネットワークにおける OP53B[*7]の適用
3. DNS ファイヤーウォールの導入

クエリログの取得は，リアルタイムではなく後に解析・調査するにためには必要不可欠である．OP53B の適用は，組織における DNS 通信をイントラネットのフルサービスリゾルバに集約するのに効果が期待される．DNS ファイヤーウォールの導入は，ベンダーが開発したトンネリング通信のモデルや閾値に基づき検知またはブロックするのに効果が期待される．

2.3.1 特徴量

従来，DNS トンネリング通信の検知には，以下に示す特徴量について統計分析を用いた閾値の算出や機械学習を用いた悪性モデルが用いられてきた．トンネリング実装などによって発生する一般的な DNS トンネリング通信の検知にあたり，以下のような特徴を利用した手法がこれまでに多数提案されている．

ドメイン名の長さとクエリパケットのサイズ

クライアントからサーバ方向にデータを転送させる DNS Exfiltration 手法におい

[*7] OP53B：組織外に設置されたオープンリゾルバのようなフルサービスリゾルバや権威サーバとの通信を抑止するために，組織外ネットワークを宛先とする 53 番ポートの通信をブロックする仕組み

て，転送キャリアとなるドメイン名が注入されるデータ量に応じて長くなる [40]．例えば，DNS Exfiltration において，一回あたりのデータ転送量を増加させる場合，Qname フィールドのドメイン長もそれに比例して長くなり，結果としてパケットサイズも増加するという具合である．DNS Infiltration においても同様で，一回あたりのデータ転送量を増加させる場合，Rdata フィールド内のデータ量も大きくなり，応答パケットのサイズが増加する．

表 3　正規 DNS クエリと DNS トンネリングにおけるドメイン名の違い

種類	ドメイン名
正規	www.example.com
トンネリング	arbitrary-long-text-utill-it-reaches-253byte.example.com

同一ドメインあたりのトラフィック頻度

DNS トンネリングでは，一度に転送できるデータ量に限界があるため，目的のデータを全て転送するには分割する必要がある．トンネリング実装のように対話的にシェルコマンドを実行する通信の場合，トラフィック頻度は極めて高頻度になる．また，サイズの大きいデータを DNS Exfiltration を用いて転送する場合も同様に，複数のパケットに分割されたデータを転送するにあたって，多数のトラフィックが発生することになる．

リソースレコードのタイプ

理論的に全てのリソースレコードを用いてデータを転送することは可能であるが，第 2.2.2 項で示すように，使用するレコードタイプによって転送できるデータ量は大きく異なる．表 2 で示すように，実際のトンネリング実装における DNS Infiltaration を目的とする通信では，A や AAAA などは使われず，CNAME や TXT が主に使用される．A や AAAA などのレコードタイプが使用されない背景には，数字のみの文字列制約が厳しさと最大のデータサイズに小さいことが考えられる．TXT の最大サイズが 255bytes であるのに対して，A が 4bytes で AAAA が 32bytes なのは明らかに小さいことが確認できる．他方で，表 4 で示すように，通常のインターネットの利活用において使用されるレコードタイプに極端な分布の偏りがあることが知られている．Herrymann ら [41] は，2010 年 1 月 1 日から

表 4 レコードタイプの分布

Type	パケット数	割合
A	236,210,050	54.778
AAAA	149,322,427	34.629
PTR	43,060,608	9.986
SRV	1,497,622	0.347
MX	474,827	0.110
ANY	281,023	0.0657
SOA	226,975	0.053
TXT	115,300	0.027
NS	12,028	0.003
TKEY	4518	0.001
NAPTR	4281	0.001
SPF	512	0.000
CNAME	196	0.000
AXFR	2	0.000
NULL	2	0.000
合計	431,210,371	100.000

6 月 30 日までの期間において，大学構内に設置されたフルサービスリゾルバによる DNS ログデータを収集した．結果が示すように，ドメイン名に対するアドレス解決の通信が全体の 89.407%を占めていることが確認できる．以上のことから，TXT や NULL といった任意の文字列を注入できるレコードタイプは効率的なデータ転送を実現できる反面，レコードタイプとして使用頻度低いという特性から，データ転送と秘匿性がトレードオフの関係にあることがわかる．

パケットの応答ステータス

DNS のヘッダーは，図 7 で示すようなフィールドを持っており，問い合わせに対して，表 5 のようなステータス情報を応答する．第 2.3.2 項で示すような検知迂

回手法を使う場合を除いて，通常の DNS Exfiltration では，権威サーバが未知のデータがクラアントから転送される．そのため，クライアントからの問い合わせには，コンテンツ不在を意味する “NXDomain” が応答される．応答パケットのステータスが “NXDomain” であるとき，DNS Exfiltration の可能性がある．

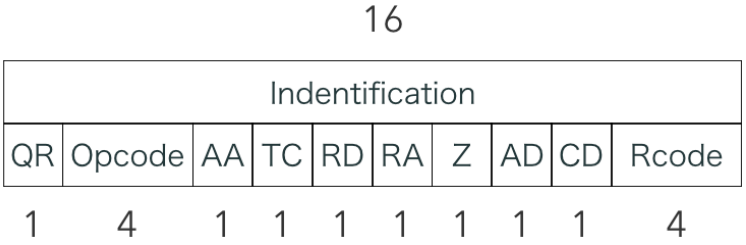

図 7 DNS のヘッダー (bytes)

表 5 代表的な Rcode 一覧

値	名前	意味
0	NoError	正常
1	FormErr	フォーマットエラー
2	ServFail	サーバエラー
3	NXDomain	存在しないドメイン
4	NotImp	未実装
5	Refused	問い合わせ拒否

ドメイン名に含まれる文字列の出現頻度

Born ら [10] は，ドメイン名に使用されている文字列の分布について，流布しているトンネリング実装と正規の DNS 通信について調査した．その結果，正規のドメイン名が英語における文字列の出現分布と相関があるのに対して，トンネリング実装によって生成されるドメイン名における文字列の出現頻度では相関がみられず，文字列の出現頻度はランダムとなる傾向にある．

2.3.2 検知迂回の脅威モデル

第 2.3.1 項で述べる特徴量に基づき通信を監視することで，Iodine に代表されるトンネリング実装の通信を検知するのは，既存の検知手法で十分対処することは可能であると考えられる．しかし，秘匿性を高めた DNS トンネリング通信に対しては，既存の検知手法では十分でない場合が考えられる．例えば，DNS トンネリングの秘匿性を高める手法には，一回の問い合わせで転送するデータサイズを小さくする方法が考えられる．従来の検知手法では，通常の DNS 通信の統計的な分布から外れる異常通信として検知する．データサイズを小さくするということは，一般の DNS クエリに使用されるようなドメイン長や応答パケットサイズに調整するということである．この場合，従来のトンネリング実装などと比べて転送効率は下がる代わりに，パケットは一般のトラフィックに極めて類似することができる．

このように異常通信の特徴になる要素を削減することで，通常のインターネットの利用時に見られる入力ミスによって生じる通信に分類される通信クラスに模倣するというものである．さらに，トラフィックの頻度の抑える手法を組み合わせることが考えられる．第 2.3.1 項で述べるようにトラフィックの頻度に基づいて異常を検知するアプローチは広く利用されているが，DNS の通信においてはトラフィック量が肥大化しやすく長期間ログが取得されることは稀であることが予想される．APT に代表される攻撃では，攻撃手法の解析などを回避するために，極めて高いモチベーションでこのような秘匿技術が用いられることが予想される．現在の検知に基づく対策では，このような秘匿通信に対処するには課題がある．Asaf らは，システムのメモリやディスクを追加の特徴量とすることによって，そのような秘匿性を高めたトンネリング通信を検知する手法を提案している [13]．しかし，Asaf らの提案手法では，リアルタイム性が考慮されていない．攻撃者にとって，秘匿手法によって小分けされたデータが情報という意味のあるものになったタイミングで情報流出という目的は達成される．すなわち，情報流出の本質的な対策には，クラスタリングではなく，未然に抑止することが求められる．

3. 提案システム

本章では，DNS トンネリングの発生を抑止を目的とした名前解決システム DNS-TD(DNS for Tunneling Deterrence) について説明する．

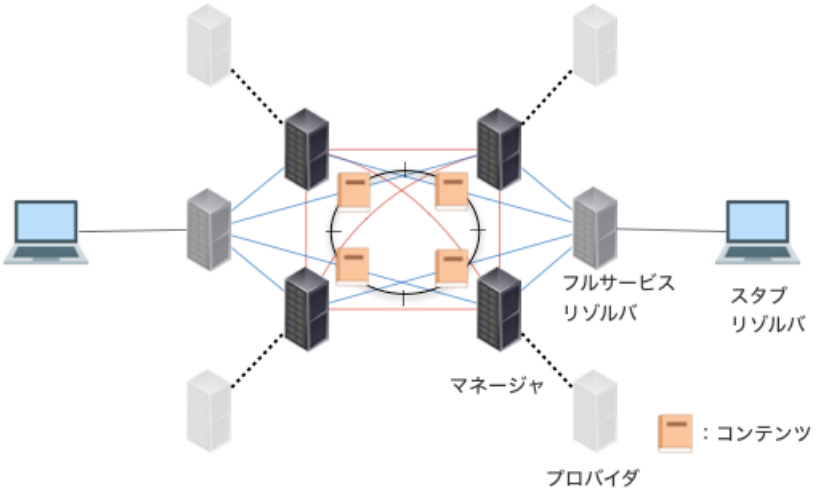

図 8　提案システムの概略図

3.1 概要

現在の DNS の名前解決の仕組みには，データ転送のキャリアとして機能する設計上の課題がある．過去に多数の検知に基づく対策手法が提案されているが，第 2.3.2 節で述べるように，迂回手法を用いる場合には既存検知手法では対処するのに不十分である．この課題に対して，そこで，フラットな名前空間と分散ハッシュテーブルの仕組みを利用することによるトンネリング通信の発生を抑止する名前解決システムを提案する．提案システムの DNS-TD の主な主要な要素は，以下の通りである．

- ドメイン名ではなく識別子に基づいた名前解決
- 224bit のフラットな名前空間
- ソートされたハッシュ空間に基づいたゾーン分割

- レコード情報の操作機能と管理機能の分離
- レコード情報に対する認証機能

コンテンツ情報を変更する機能とクライアントにコンテンツを提供する機能を分離することによって，DNS トンネリングにおけるクライアントとコンテンツを操作するサーバとの通信の発生を抑止する．また，DNS-TD では，ゾーンファイルによって権威サーバによってドメインに紐付けるレコード情報を一元的に管理するのではなく，リソースレコードのタイプごとにコンテンツの識別子を付与し，識別子に基づき担当のサーバ，提案システムではマネージャによって分散的に管理される．識別子は，224bit の名前空間をもつハッシュ関数によって算出されるメッセージダイジェストを利用する．レコード情報を操作するのは，実際にレコード情報を保持するマネージャではなく，マネージャに階層的に連結したプロバイダと呼ばれるサービスノードである．マネージャは，プロバイダからのレコード情報に対する操作リクエストに基づいて，担当のマネージャに転送する．DNS-TD の名前空間は，ソートされたハッシュ値の範囲に基づき分割され，既存システムの TLD によって分割された範囲をゾーンとして分割的に管理される．DNS-TD におけるクライアントは，シンボルを算出することによって，レコード情報を保持するノードを一意に特定することができる．また，シンボルからレコード情報を保持するノードを一意に特定の名前解決プロセスでは，レコード情報を作成したエンティティは介在されない．すなわち，レコード情報を作成する機能とレコード情報を管理する機能を異なるサービスノードに分離させることによって，DNS-TD では DNS Exfiltration が発生することを抑止する．

DNS-TD では，DNS Infiltration に対して，認証基盤の導入と使用できるリソースレコードを制限するメソッドを採用する．認証基盤では，ドメインに関連づけるレコード情報との関連性を評価することで不審なレコード情報がハッシュテーブル上にストアされることを防止する．Qname にシンボルとして 224bit のメッセージダイジェストを使用することによって，副次的効果として，偽装 DNS 応答パケットの作成が困難にできるという性質が期待される．また，既存システムにおける DNSSEC によって実現されてきた送信元のトレーサビリティについて，DNS-TD では認証基盤とシンボルに基づく DNS 応答パケットの偽装困難性によっ

て実現され，DNSSEC の必要性はなくなる．さらに，任意の文字列を注入することができるレコードタイプ “NULL” と “TXT” について，実験目的の NULL タイプの使用制限と TXT タイプは機能をシンタックスの限定している SPF に回帰させることで対処する．DNS-TD における用語については，表 6 で示す．

表 6 DNS-TD における用語

表記	意味または機能
コンテンツ	・識別子に関連づけられたレコード情報の実体
コンテンツ ID	・識別子
ドメイン ID	・識別子 (コンテンツ ID が重複した際に使用)
レコード情報	・リソースレコードの具体的な値 (例 IP アドレス)
リソースレコードタイプ	・オブジェクトに関連づけるリソースレコードの型 (例 A, AAAA, MX)
オブジェクト	・問い合わせる対象 (ドメイン名もしくは IP アドレス)
スタブリゾルバ	・名前解決クライアント
フルサービスリゾルバ	・スタブリゾルバからのクエリハンドリング ・識別子の作成
マネージャ	・フルサービスリゾルバからのクエリハンドリング ・ゾーンの管理 ・コンテンツの保持
プロバイダ	・コンテンツの作成・更新・削除操作

3.2 ハイブリットなアーキテクチャ

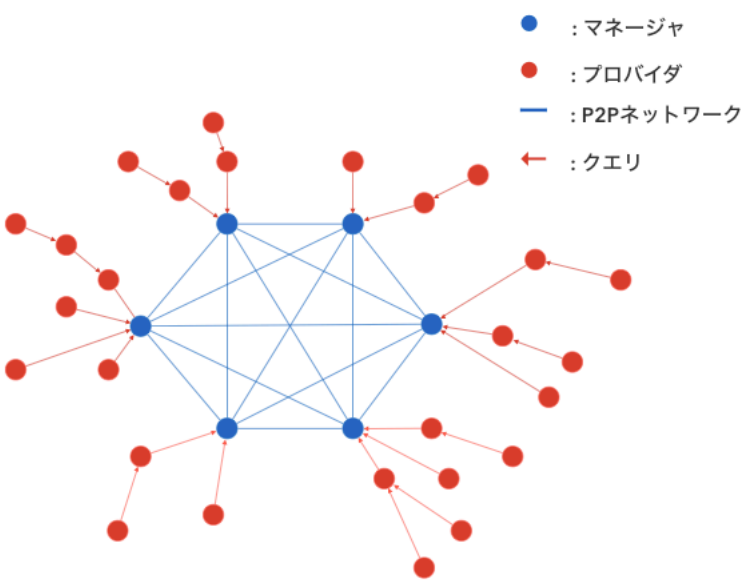

図 9　マネージャとプロバイダの関係図

DNS-TD では，全体のアーキテクチャを従来のクライアントサーバアーキテクチャを踏襲している．クライアントサーバアーキテクチャを採用することによって，クライアントは既存の DNS サービスに変更を加えることなく使用することができる．他方で，サーバ同士はフルメッシュなネットワークで構成される．

3.3 サービスノード

DNS-TD を構成するサービスノードは，以下の 4 つである．

- スタブリゾルバ
- フルサービスリゾルバ
- マネージャ
- プロバイダ

スタブリゾルバは，既存の DNS と同様，ドメイン名に関連づけられたリソースレコード情報を問い合わせるクライアントである．DNS-TD が，既存のシステムと大きく異なるのは，問い合わせられたドメイン名とそのレコードタイプからコンテンツ ID を識別子とすることで，レコード情報を解決する点である．これを実現するために，既存システムのフルサービスリゾルバと権威サーバに変更が加えられ，機能が分割している．

フルサービスリゾルバ

DNS-TD におけるフルサービスリゾルバは，従来システムと同様のキャッシュ機能を担いながら，コンテンツ ID の算出し，コンテンツを保持するマネージャに問い合わせるサービスノードとして機能する．フルサービスリゾルバは，スタブリゾルバからの従来のフォーマットの DNS クエリについて，ドメイン名とレコードタイプからハッシュ値の算出を行う機能を担う．

DNS-TD では，権威サーバが機能に基づき二つサービスノードに分割される．DNS-TD では，既存の DNS における権威サーバの機能を二つのサービスノードで動作させることによって，DNS トンネリングの抑止を図る設計になっている．権威サーバの機能を列挙すると以下のようになる．

- 上位ドメインから委任されたゾーンを管理する機能
- レコード情報を保持する機能
- スタブリゾルバからの名前解決問い合わせに応答する機能
- レコード情報を作成・修正・削除する機能

DNS トンネリングは，これら機能と性質を組み合わせることで発生する手法である．例えば，スタブリゾルバから権威サーバ方向のデータ転送手法である DNS Exfiltration は，“スタブリゾルバからの名前解決問い合わせに応答する機能”と“”を組み合わせることで機能する．また，権威サーバからスタブリゾルバ方向のデータ転送手法である DNS Infiltration は，“スタブリゾルバからの名前解決問い合わせに応答する機能”と “ゾーン内のレコード情報を作成・修正・削除する機能”が組み合わさることで機能するとそれぞれ捉えられる．そこで，DNS-TD では，

アルゴリズム 1. フルサービスリゾルバにおける問い合わせ転送処理

```
クエリハンドリング
handler(data):
    content_id, domain_id ← calculate_id(data)
    manager_addr ← find_manager(start, end, content_id)
    answer ← query_manager(manager_addr, content_id, domain_id)
    response_client(client_address, qname, answer.rcode, answer.rdata)

コンテンツ ID とドメイン ID の算出
calculate_id(qname, rtype):
    content_id ← hash.sha3_224(qname + qtype)
    domain_id ← hash.md5(qname) / 2
    return content_id, domain_id

コンテンツ ID が含まれるゾーンを保持するマネージャアドレスの解決
find_manager(start, end, content_id):
    for i, j in map_start, map_end :
        if i ≤ content_id ≤ j :
            p ← map_start.index(i)
            manager_addr ← map.addr[p]
            return manager_addr
```

権威サーバの機能を二つのサービスノード (マネージャとプロバイダ) に分けることで，既存の名前解決機能と DNS トンネリング抑止機能を実現する．

マネージャ

マネージャは，ドメイン名に対応するレコード情報を保持し，フルサービスリゾルバからの問い合わせに応答するサービスノードである．マネージャは，既存の DNS における TLD に相当する権威サーバが担当する．

アルゴリズム 2. マネージャにおける名前解決問い合わせ処理

```
フルサービスリゾルバからのクエリハンドリング
handler(data):
    content_id, qtype ← parser(data)
    record_value ← db_accesser(content_id)
    if value :
        payload ← benigh_response(content_id, qtype, ttl, record_value)
    else:
        payload ← error_response(content_id, qtype)
    payload ← payload.pack()
    connection.sendto(payload, client_address)
```

プロバイダ

プロバイダは，レコード情報を作成･更新および消去といった操作を担当するサービスノードである．プロバイダは，既存の DNS における SLD 以降の権威サーバに相当し，ドメインの階層構造を従いマネージャに接続される．プロバイダは，マネージャを介在することで，レコード情報を操作することができる．例えば，example.com プロバイダが “www” の IP アドレス情報を作成することを考える．example.com プロバイダは，“www.example.com” とレコードタイプ “A” およびその値 “93.184.216.34” を含むデータを接続先の com マネージャにリクエストする．com マネージャは，リクエストされたドメイン名とそれに関連づけるレコードタイプから識別子を算出し，担当のマネージャにストアリクエストを転送するという具合で動作する．

3.4 名前空間

本項では，レコード情報にアクセスするために用いる識別子，コンテンツ ID とその名前空間について説明する．既存の DNS の名前解決システムでは，正引

きをする際，ドメイン名とレコードタイプの二つの情報をキーとして，サーバは保持するゾーンファイルから該当するレコード情報が求まる．他方，DNS-TD では，ドメイン名とレコードタイプに基づき算出されるコンテンツ ID を識別子として指定することでレコード情報にアクセスする．

におけるレコード情報へのアクセス方法は，識別子をキーとすることでデータを取得することができる．コンテンツ ID と呼ぶ，この識別子は，ドメイン名とレコードタイプの順番でその文字列の和を引数とするメッセージダイジェストで表現することができる．例えば，ドメイン名を “www.example.com”，レコードタイプを “A” とした場合，引数となるのは “www.example.comA” という具合である．

3.4.1 ハッシュアルゴリズム

続いて，DNS-TD が採用するハッシュアルゴリズムについて説明する．DNS-TD では，全てのコンテンツ ID がフラットな名前空間上にマップされる．従来では，異なるリソースレコードタイプのレコード情報について一つのゾーンファイルに記述することで，管理することができた．他方で，DNS-TD では，ドメイン名とリソースレコードタイプの組をハッシュ値の引数とするため，コンテンツ ID は，レコード情報の数に比例して増加する特性がある．また，識別子の引数の一つにドメイン名が含まれていることから，識別子から元のメッセージが導き出くことが困難な性質を備えていなくてはならない．この性質を満たすことで，なんらかの方法で識別子を悪意の第三者が取得した際に DNS Exfiltration として機能してしまうことを抑止することができる．最後に，DNS-TD では，既存の DNS のプロトコルフォーマットを流用する設計になっているため，識別子が格納される DNS の Question Section の Qname のサイズ制約を満たさなくてはならない．Qname は，ラベルの最大長が 63byte とする任意長の領域である．以上のことを踏まえて，本システムにおけるハッシュアルゴリズムの要件は，以下の通りである．

1. 名前空間は不足を無視できる程度に大きくなくてはならない
2. アルゴリズムは一方向性の性質を備えなくてはならない
3. ラベル長は最大 63byte，ドメイン長は最大 253byte である

上記の内容から，DNS-TD では，56byte の名前空間をもつ sha3 のアルゴリズムを採用する．

続いて，分離連鎖法と 2 重ハッシュ法によるコリジョン対策方法について説明する．DNS-TD では，コンテンツのストアリングフェーズで ID にコリジョンが発生した場合，分離連鎖法に基づきストアされるハッシュテーブルに連結リストという形式でコンテンツがストアされる．リスト構造で延長するコンテンツの識別には，ドメイン ID を識別子として利用する．ドメイン ID は，コンテンツ ID と同様のハッシュアルゴリズムを用いて算出されるメッセージダイジェストの先頭 32bit で表現される，ドメイン名を引数として生成される識別子である．例えば，ドメイン名を “www.example.com” とする場合，そのメッセージダイジェストが “86ff20100c058b857bae9785bf0267e6c6afb740c18b8e9a87258485” であるとすると，“86ff” がドメイン ID となる具合である．このように算出されたドメイン ID は，DNS の Question Section のうち，それぞれ 16bit 分の領域を持つタイプとクラスの領域に埋め込まれる．上記の仕組みによって，コリジョンが発生した際には，ドメイン ID をキーとしてコンテンツを識別する．

3.4.2 ゾーン分割

本項では，ゾーンの分割方法およびマネージャノードのアドレスとそのゾーンの範囲に関する対応表について説明する．はじめに比較のために，従来のシステムの場合について説明する．従来のシステムでは，ドメインの階層構造に従い，ドメインの管理ノードを下位のドメイン管理ノードに委譲することでゾーンが分割される．この仕組みでは，ゾーン内の全てのレコード情報はゾーンファイルに画一的にまとめられ，そのゾーンを管理する権威サーバがレコード情報の保持機能とクライアントから応答するという二つの機能を担う．このゾーン分割メソッドでは，レコード情報の帰属が明確であり，ドメインの管理ノードがトラストアンカーとしての役割を同時に担うことができるメリットがある．一方の DNS-TD では，識別子を算出する際に使用するハッシュ関数によって構成される名前空間に基づき，ソートされたハッシュの名前空間の連続した範囲で分割する．この分割された連続した範囲に基づきゾーンがマネージャに割り当てられることで，既

存システム同様にレコード情報全体を分散的に管理する．

上記で説明するように，マネージャが管理するゾーンは，ハッシュの名前空間の連続した一部の範囲である．従って，レコード情報は，ハッシュの名前空間上で識別子をソートした際に，帰属する範囲を管理するマネージャによって保持される．マネージャのアドレスを解決する方法には，ゾーンとしてハッシュ値の範囲とそのマネージャおよびマネージャのアドレスに関する対応表 7 によって解決される．DNS-TD では，全てのサービスノードがこの対応表を保持できることを想定しており，ノードは識別子に基づきどのマネージャがコンテンツを保持しているのかを一意に特定する．

表 7　6 つのマネージャによって管理されるハッシュテーブルにおいて，マネージャの情報とそのマネージャが管理するゾーンが記載された対応表の例

ゾーン	マネージャアドレス	ドメイン
(00…00, 4z…zz)	192.35.51.30	com.
(50…00, az…zz)	192.5.6.30	net.
(b0…00, gz…zz)	199.249.112.1	org.
(h0…00, mz…zz)	213.248.216.1	uk.
(n0…00, sz…zz)	199.254.31.1	info.
(t0…00, zz…zz)	194.0.0.53	de.

3.5 リソースレコード

3.5.1 認証基盤

本項では，レコード情報の信頼性担保のための認証基盤について説明する．

DNS-TD では，全てのコンテンツについて，信頼される第三者からストアしても良いと認可されていることを前提としている．すなわち，ハッシュテーブル上のコンテンツへの操作，またはコンテンツをハッシュテーブル上にストアするなどの操作処理をする際，プロバイダは，信頼される第三者からのレコード情報に操作

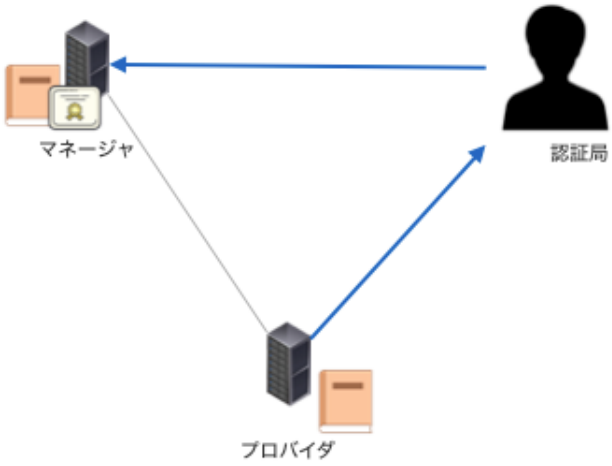

図 10　レコード情報操作におけるプロセスの概略図

することを認可してもらう必要がある．認可の証明書を発行する認証局は，プロバイダの基本情報とレコード情報に基づき証明書の発行を決定する．いま，ドメイン名 “www.example.com” のリソースレコードタイプ A として “93.184.216.34” というレコード情報を関連づけるとする．プロバイダは，認証局に対して証明書発行リクエストを転送する．認証局は，リクエストされたレコード情報について IP アドレスの到達性と不審な文字列が含まれていないこと，利用目的について評価を施す．認可された場合には，その証にディジタル証明書を発行し，マネージャにストアリクエストを転送する．マネージャは，ディジタル証明書に付与された署名に基づきコンテンツの完全性を評価し，認証された場合コンテンツ ID を計算し，担当マネージャにストアリクエストを転送する．上記の手続きを経たコンテンツがハッシュテーブル上にストアされる．

3.5.2　レコードタイプ

本項では，DNS-TD で使用するリソースレコードのタイプについて説明する．

はじめに，DNS-TD における DNSSEC の位置づけについて述べる．DNSSEC [42] は，権威サーバからの応答パケットの偽装を検知することを目的として，データの作成元の確認とデータの完全性および，不在情報応答情報の証明する DNS の拡張仕様である．これは，主として DNS の応答パケットを偽装できる程度のパ

ラメータであることに起因する．他方で，DNS-TD では，応答パケットに 224bit のメッセージダイジェストが含まれるため，悪意のある応答パケットをフルサービスリゾルバに意図的にキャッシュすることは極めて困難である．以上の理由から，DNS-TD では DNSSEC の目的にそぐわないため，リソースレコードとして使用されない．

次に，DNSSEC 以外のリソースレコードについて説明する．第 2.2.2 項で示すように，既存の名前解決システムでドメインに関連づけることができるリソースレコードのいくつかのタイプは，DNS Infiltration として機能することができる．DNS Infiltration を抑止するリソースレコードであることの必要条件は，ドメインに関連のない任意の文字列がレコード情報に含められないことである．既存の DNS のリソースレコードのタイプのうち，任意の文字列を含めることができるのタイプは以下の通りである．

表 の DNS Infiltration として機能する可能性のあるリソースレコードのタイプのうち，IP アドレスを偽装して情報を転送するものについては，第 3.5.1 項で述べた認証基盤によってレコード情報の正当性評価でスクリーニングすることができる．他方で，任意の文字列を注入できるのが，NULL・TXT・CNAME タイプである．

NULL について考える．NULL タイプの目的は，実験用と定義されている [19]．TXT について考える．CNAME について考える．ホスト名に対する別名で関連づけることができる CNAME は，一つのサーバにおいてサービスごとにサーバの名前を変更させるために使用される．

3.5.3 コンテンツのデータフォーマット

本項では，マネージャにて管理されるコンテンツのフォーマットについて説明する．

図 11　コンテンツのデータフォーマット

4. 評価

本章では，提案システム上でランダム長のドメイン名をトンネリング通信と仮定して動作させることによって，フラットな名前空間と権威サーバの機能分割という提案システムに使われている提案手法がトンネリング抑止として機能することを明らかにする．また，既存システムとの比較に基づいたシミュレーションテストにより，名前解決速度・トラフィック量などの特性を明らかにする．

トンネリング抑止の機能評価テストでは，フルサービスリゾルバとマネージャおよびプロバイダを Python3 を用いて実装し，それらサービスを Docker のコンテナとして動作させることによって，提案システムにおける名前解決環境を再現したプロトタイプを用いた．また，DNS トンネリングの通信には，ランダムなドメイン名を dig コマンドを用いることによって再現した．プロトタイプ上における dig コマンドによる擬似トンネリング通信について，プロバイダにデータが転送されないことを確認する．特性評価では，ランダムな名前解決クエリを用意し，既存の DNS と提案システムの両方で問い合わせた際のトラフィックと名前解決の速度について，統計的に評価を実施する．

4.1 DNS トンネリング

4.1.1 環境

検証環境には，Docker を使用し，コンテナ同士の接続によって DNS のチェーンを実現する．その概略図は，以下の通りである．

4.1.2 DNS Exfiltration

本項では，提案システムの DNS Exfiltration への機能性について評価した結果を述べる．評価には，実装したシステムを

4.2 要素ごとの特性

本節では，DNS との比較評価に基づいた提案手法の特性について説明する．提案システムと既存システムの名前解決における差分には，表 8 のようなものが予想される．

表 8 DNS と DNS-TD の特性比較

	DNS	**DNS-TD**
ドメイン長	変長 (最大 253byte)	固定長 (72byte) (コンテンツ ID(56) & ドメイン ID(16))
トランザクション 試行回数	ラベル数回	1
RTT	全ての権威サーバ との RTT 総和	マネージャとの RTT のみ
ゾーンファイル	ファイル	インメモリデータベース
その他		・ハッシュ計算の発生 ・マネージャ探索

上記の特性に関する評価にあたり，従来の DNS トラフィックのデータセットと提案システムにおける名前解決時のトラフィックデータセットを用意した．これらデータセットについて，統計アプローチに基づいて，評価を行った結果を示す．

dns-td では，スタブリゾルバからの名前解決クエリのうち，そのヘッダー情報に基づき識別子を算出することで，この識別子に関連づけられたレコード情報を操作する仕組みに基づき名前解決機能が動作する．DNS では，ゾーンをドメインごとに分割し，フルサービスリゾルバはルート権威サーバから目的の権威サーバに向かって再帰的に問い合わせ，権威サーバのアドレスを解決しながら，最終的にコンテンツを保持する権威サーバからコンテンツを取得する．既存の DNS と dns-td の特性を表 8 に示す．

から識別子に基づき，コンテンツを保持するサーバに名前解決リクエストを投

げる．dns-td は，フルサービスリゾルバにてスタブリゾルバから問い合わせられた名前解決命令に基づきレコード情報の識別子を算出する．この識別子に基づき，フルサービスリゾルバからは，1 ホップでコンテンツを保持するサーバに問い合わせる仕組みで動作する．

4.2.1 RTT(Round Trip Time)

本項では，dns-td が識別子から一意にレコード情報にアクセス可能である点について，DNS との比較評価を行う．dns-td の名前解決では，従来の DNS では，ラベルごとにゾーンが移譲されている場合，レコード情報を保持するノードまでのホップ数はラベル数 n に比例する．それに対して，dns-td では，識別子から一意にレコード情報を保持するマネージャノードを特定できるので，常にホップ数は 2 である．このため，既存の名前解決システムより速度の向上が期待される．

4.2.2 パケットサイズ

dns-td では，56byte を固定長とするコンテンツ ID をシンボルとすることによって，レコード情報にアクセスする．この仕組みの影響で，dns-td のパケットは従来のパケットと比較して肥大する特性がある．このため，送信元を目的ホストと偽装することで目的ホストの計算リソースを圧迫する DDoS 攻撃に対して，脅威を高める可能性が予想される．

4.2.3 トラフィック量

本項では，提案手法におけるトラフィックについて評価する．評価において，dns-td では，シンボル志向の名前解決メソッドによって，既存の再帰問い合わせによるメソッドよりも少ないトラフィックに抑えることが期待される．dns-td では，クエリ数とトラフィック量は比例関係にある．他方で，新たにマネージャ間通信という従来にはないトラフィックが発生する．本項では，これトラフィックがネットワーク全体にどの程度影響を及ぼすのかについて評価する．

5. 考察

本章では，第 4 で行なった評価を踏まえて，提案手法の有用性について考察する.

5.1 マネージャノードの最適な数

5.2 コンテンツ ID を計算する最適ノード

6. 結論

謝辞

ご指導ご鞭撻賜りありがとうございました.

参考文献

[1] M. Att&ck, "Custom command and control protocol." https://attack.mitre.org/techniques/T1094/. (accessed at 2020-1-5).

[2] KrebsonSecurity, "Deconstructing the 2014 sally beauty breach." https://krebsonsecurity.com/2015/05/deconstructing-the-2014-sally-beauty-breach/, May 2015. (accessed at 2020-1-5).

[3] IronNet, "Chirp of the poisonfrog." https://ironnet.com/blog/chirp-of-the-poisonfrog/, 2019. (accessed at 2020-1-5).

[4] N. Hoffman, "Bernhardpos." https://securitykitten.github.io/2015/07/14/bernhardpos.html, 2015. (accessed at 2020-1-5).

[5] FireEye, "Multigrain – point of sale attackers make an unhealthy addition to the pantry." https://www.fieeye.com/blog/threat-research/2016/04/multigrain_pointo.html, 2016. (accessed at 2019-11-30).

[6] P. alto Networks, "New wekby attacks use dns requests as command and control mechanism." https://unit42.paloaltonetworks.com/unit42-new-wekby-attacks-use-dns-requests-as-command-and-control-mechanism/, May 2016. (accessed at 2019-11-30).

[7] Kaspersky, "Use of dns tunneling for c&c communications." https://securelist.com/use-of-dns-tunneling-for-cc-communications/78203/, 2017. (accessed at 2019-11-30).

[8] C. Talos, "Spoofed sec emails distribute evolved dnsmessenger." https://blog.talosintelligence.com/2017/10/dnsmessenger-sec-campaign.html, 2017. (accessed at 2019-11-30).

[9] Cylance, "Threat spotlight: Inside udpos malware." https://threatvector.cylance.com/en_us/home/threat-spotlight-inside-udpos-malware.html, 2018. (accessed at 2019-11-30).

[10] K. Born and D. Gustafson, "Ngviz: Detecting dns tunnels through n-gram visualization and quantitative analysis," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, (New York, NY, USA), p. 4, Association for Computing Machinery, 2010.

[11] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu, "A bigram based real time dns tunnel detection approach," *Procedia Computer Science*, vol. 17, pp. 852 – 860, 2013. First International Conference on Information Technology and Quantitative Management.

[12] J. Liu, S. Li, Y. Zhang, J. Xiao, P. Chang, and C. Peng, "Detecting dns tunnel through binary-classification based on behavior features," in *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 339–346, Aug 2017.

[13] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," 2017.

[14] J. Steadman and S. Scott-Hayward, "Dnsxd: Detecting data exfiltration over dns," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–6, Nov 2018.

[15] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring enterprise dns queries for detecting data exfiltration from internal hosts," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2019.

[16] E. Ekman, "iodine." http://code.kryo.se/iodine/. (accessed at 2020-1-5).

[17] R. Bowes, "dnscat2." https://github.com/iagox86/dnscat2. (accessed at 2020-1-5).

[18] P. Mockapetris, "Domain names - concepts and facilities." https://www.rfc-editor.org/info/std13, November 1987. RFC1034.

[19] P. Mockapetris, "Domain names - implementation and specification." https://www.rfc-editor.org/info/std13, November 1987. RFC1035.

[20] J. Klensin, "Internationalized domain names for applications (idna): Definitions and document framework." https://www.rfc-editor.org/info/rfc5890, August 2010. RFC5890.

[21] O. Pearson, "Bugtraq mailing list archives dns tunnel - through bastion hosts." https://seclists.org/bugtraq/1998/Apr/79. (accessed at 2019-1-4).

[22] M. V. Horenbeeck, "Dns tunneling." http://web.archive.org/web/20060709044338/www.daemon.be/maarten/dnstunnel.html. (accessed at 2020-1-5).

[23] D. Kaminsky, "Ozymandns." https://dankaminsky.com/2004/07/29/51/. (accessed at 2020-1-5).

[24] D. Kaminsky, "Reverse dns tunneling staged loading shellcode." https://www.blackhat.com/presentations/bh-usa-08/Miller/BH_US_08_Ty_Miller_Reverse_DNS_Tunneling_Shellcode.pdf. (accessed at 2020-1-5).

[25] A. Revelli and N. Leidecker, "Heyoka." http://heyoka.sourceforge.net/. (accessed at 2020-1-5).

[26] Tim, "Tcp-over-dns." http://analogbit.com/software/tcp-over-dns/. (accessed at 2020-1-5).

[27] R. Bowes, "dnscat." https://wiki.skullsecurity.org/Dnscat. (accessed at 2020-1-5).

[28] M. Dornseif, "Denise." https://github.com/mdornseif/DeNiSe. (accessed at 2020-1-5).

[29] SensePost, “Dns-shell.” https://github.com/sensepost/DNS-Shell. (accessed at 2020-1-5).

[30] Sturt, “Dnsbotnet.” https://github.com/magisterquis/dnsbotnet. (accessed at 2020-1-5).

[31] F. Ceratto, “Dnscapy.” https://github.com/FedericoCeratto/dnscapy. (accessed at 2020-1-5).

[32] iceman, “dohtunnel.” https://github.com/jansect/dohtunnel/blob/master/doh/doh.go. (accessed at 2020-1-5).

[33] SensePost, “godoh.” https://github.com/sensepost/goDoH. (accessed at 2020-1-5).

[34] SpiderLabs, “Dohc2.” https://github.com/SpiderLabs/DoHC2. (accessed at 2020-1-5).

[35] MagicTunnel, “magictunnelandroid.” https://github.com/MagicTunnelM/agicTunnelAndroid. (accessed at 2020-1-5).

[36] “dns2tcp.” https://github.com/alex-sector/dns2tcp. (accessed at 2020-1-5).

[37] L. Nussbaum, “Tuns.” https://github.com/lnussbaum/tuns. (accessed at 2020-1-5).

[38] S. Josefsson, “The base16, base32, and base64 data encodings.” https://www.rfc-editor.org/info/rfc4648, October 2006. RFC4648.

[39] ☒. 泰宏 and 尾崎勝義, “Dns 運用の「見抜く」を探る〜インシデント事例の紹介と必要な要素・項目〜ランチのおともに dns.” https://jprs.jp/tech/material/iw2016-lunch-L3-01.pdf. (accessed at 2020-1-13).

[40] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver, "Practical comprehensive bounds on surreptitious communication over DNS," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, (Washington, D.C.), pp. 17–32, USENIX, 2013.

[41] D. Herrmann, C. Banse, and H. Federrath, "Behavior-based tracking: Exploiting characteristic patterns in dns traffic," *Comput. Secur.*, vol. 39, p. 17–33, Nov. 2013.

[42] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Dns security introduction and requirements." https://www.rfc-editor.org/info/rfc4033, March 2005. RFC4033.

付録

A. 発表リスト (国内研究会)

1. 高須賀 昌烈, 妙中 雄三, 門林 雄基, “非実在ドメインに対するネガティブキャッシュの拡張と再帰問い合わせハッシュ化の提案”, 電子情報通信学会情報ネットワーク研究会, 2019-10-ICTSSL-IN, 2019 年 10 月.