東京大学大学院
情報理工学系研究科 電子情報学専攻
修士論文

機械学習による
リアルタイム悪性 DNS クエリ分類システム

Real-time malicious DNS query classification system by machine learning

須賀 灯希
Toki Suga

指導教員　江崎 浩　教授

2019 年 1 月

概要

サイバー攻撃から情報を守るために、多くのセキュリティ研究者が様々な攻撃検知・防御手法を提案してきた。未知の攻撃検知や、検知の自動化を目的として機械学習や深層学習を用いた自動判別が盛んに研究されている。理想的には、攻撃によるセキュリティインシデントを未然に防ぐために、ネットワーク側でのパケット分類による攻撃検知・防御が最も有用である。しかし、機械学習を用いた検知手法は従来のシグネチャを利用した攻撃検知と比べてより長い処理時間を要する傾向にある。ネットワーク上で処理時間を要する検知手法を動作させると通信に遅延が生じるため、機械学習を用いた検知手法はネットワーク側での検知に適していない。こうした状況から、通信に遅延を生じさせずに未知の攻撃によるインシデントを未然に防ぐために、ネットワーク上においてパケット転送遅延を抑えつつ機械学習を用いた検知手法を動作させる機構が必要であると考える。本研究では、機械学習を利用したパケット分類処理アーキテクチャの提案とプロトタイプシステムの実装を行い、予備実験によりアーキテクチャのパケット転送性能と分類処理時間確保の必要性を明らかにした。さらに実験結果に基づき、DNS の名前解決時間を活用することで、機械学習を用いて悪性 DNS クエリを検知し名前解決を未然に防ぐリアルタイム悪性 DNS クエリ分類システムを提案する。システムの実装・評価を行い、3 種類の機械学習ライブラリ・フレームワークにおいて処理遅延を抑えつつ悪性 DNS クエリの名前解決の未然防止が可能であることを示した。

Abstract

Numerous security researchers have proposed various attack detection methods against cyber-attacks. Machine learning (ML) or deep learning based detection methods are promising way to detect unknown attacks and automate detection process. Ideally, malicious packets must be intercepted on network paths before target hosts have been compromised by the attacks. However, ML-based detection methods tend to take more longer processing time to detect attacks than legacy detection methods based on attack signatures. Time-consuming detection methods on network paths lead to communication latency. For this reason, ML-based detection methods are not suitable for detection on network paths. To solve the problem, an architecture that can operate ML-based detection methods with low forwarding latency on network paths is needed to prevent security incidents and reduce communication latency. In this paper, we propose a packet forwarding and classification architecture with ML-based classification methods. We implemented the architecture as a prototype system with the DPDK which is a software-based packet processing framework and common ML libraries, and conducted preliminary experiments. The results reveal the system performance and the necessity of time for ML-based classification. According to the results, we then proposed the real-time malicious DNS query classification system which processes the machine learning classification during DNS name resolution not only a query forwarding process from a client to a DNS server. Our evaluation results show that the system can completely classify malicious queries before they are resolved in three types of ML algorithms.

目次

図目次

表目次

第 1 章

序論

1.1 背景と目的

個人や組織を狙ったサイバー攻撃はインターネットインフラと利用者への大きな脅威となっており、世界中で様々な攻撃が観測されている [1]。サイバー攻撃から情報を守るために、多くのセキュリティ研究者が様々な攻撃検知・防御手法を提案してきた。攻撃検知方法は検知方針によりシグネチャ型とアノマリ型の 2 種類に大別される。シグネチャ型は攻撃のパターンを予め定義しその定義に一致したものを攻撃とみなす方式である。一方、アノマリ型は正常な通信やデータのパターンを予め定義しその定義から逸脱したものを攻撃とみなす方式である。しかしながら、これらの検知から逃れるため攻撃の傾向や手法は日々変化しており、ゼロデイ攻撃 [2] など実際にパターンマッチングによる検知をすり抜ける事例も多く発生している。未知の攻撃検知と検知の自動化を目的として機械学習や深層学習を用いた検知手法が盛んに研究されている。

また攻撃検知地点の区分として、ネットワーク側での検知と端末側での検知がある。理想的には、攻撃によるセキュリティインシデントを未然に防ぐために、ネットワーク側での攻撃検知・防御が最も有用である。ネットワーク上で攻撃を検知・防御できれば、悪性パケットが攻撃対象ホストに到達する前に効果的に遮断し、悪性パケットが内部ネットワークや外部ネットワークに広まるのを防ぐ。しかし、ネットワーク側での攻撃検知は、セキュリティ機器によりネットワークに到着する全てのパケットをインラインでリアルタイムに検査しなければならない。実際、次世代ファイアウォールや Intrusion Detection System (IDS) のような現在のセキュリティ機器はシグネチャを用いて最大で 100 Giga bit per second (Gbps) の回線速度でネットワークトラフィックを検査できる [3]。一方、回線速度でのパケット検査に許容される 1 パケットの処理時間は非常に小さい。具体的には、10 Gbps のスループットを達成するためには、64 byte のパケットを受信から送信まで合計 50 ナノ秒以内に処理しなければならない。

セキュリティインシデントを未然に防ぐためにはネットワーク側での迅速な攻撃検知が肝要であるが、機械学習を用いた攻撃検知手法の多くはリアルタイム攻撃検知に適していない。なぜなら機械学習を用いた攻撃検知手法は、検知対象となる攻撃に対し新たな解析アルゴリズム、複数の特徴量を用いることで検知精度を向上させている。その一方で、検知に必要な処理

時間は考慮されておらず、処理時間がアルゴリズムの種別や特徴量の数に応じて長くなる傾向にある。特にパケットから直接取得できない外部情報を特徴量として利用すると処理時間が非常に長くなる。ネットワーク上における検査処理時間が長くなるとパケット転送に時間を要し、正規の通信に遅延が生じる。

こうした状況から、通信に遅延を生じさせずに未知の攻撃によるインシデントを未然に防ぐために、ネットワーク上においてパケット転送遅延を抑えつつ機械学習を用いた検知手法を動作させる機構が必要であると考える。本研究では、ネットワーク上のパケットに対して機械学習を用いた攻撃検知処理を高速に行い、パケットの分類を行う処理機構の提案を目的とする。攻撃検知をネットワーク側で行い、悪性通信を即座に遮断することで、インシデント発生前に攻撃を防ぐことが可能となる。しかし、この処理機構では機械学習による分類処理を通信に影響を与えない時間内で処理しなければならない。そのため本研究では個々の通信プロトコルの特徴や実際のネットワークを流れるトラフィックの特性に着目し、パケット処理以外の部分で機械学習による分類処理時間を得る方法について議論する。議論を踏まえ、DNS の名前解決に要する時間を活用することで、機械学習を用いて悪性 DNS クエリを検知し名前解決を未然に防ぐリアルタイム悪性 DNS クエリ分類システムを提案する。

1.2　本論文の構成と内容

本論文の構成と内容は以下の通りである。2 章では巧妙化するサイバー攻撃の例を挙げ、攻撃検知のための機械学習を用いた既存研究をまとめる。さらに既存研究のほとんどは検知に要する処理時間を考慮しておらず、リアルタイム検知には適用できないという問題点について議論する。3 章では機械学習を利用したリアルタイム悪性パケット分類を実現するためのアーキテクチャについて、要件定義、要件を満たすアーキテクチャの概要とその実装についての説明、予備実験によるアーキテクチャの性能評価を行う。4 章では 3 章での予備実験の結果を踏まえ、個々の通信プロトコルの特徴や実際のネットワークを流れるトラフィックの特性に着目し、パケット処理以外の部分で機械学習による分類処理時間を得る方法について議論する。さらに DNS の名前解決が完了するまでの時間を利用することで、機械学習を用いたリアルタイム攻撃検知を実現する DNS パケット処理システムを提案する。5 章では攻撃によるインシデントを未然に防止できるかについて、提案システムの評価を行う。6 章で評価結果の考察を議論し、最後に 7 章で本論文をまとめ、今後の課題を述べる。

第 2 章

サイバー攻撃への対処

本章では、サイバー攻撃の種類や対処について整理する。2.1 節で攻撃例を挙げ、2.2 節で攻撃を検知するための機械学習技術を用いた既存研究について述べる。次に既存研究のほとんどは検知に要する処理時間を考慮しておらず、リアルタイム検知には適用できないという問題点について議論する。

2.1 巧妙化するサイバー攻撃

個人や企業の情報の破壊・窃取・改ざんを試みるサイバー攻撃はインターネットインフラと利用者の大きな脅威となっている。現在までに多数の攻撃が観測されており、代表的なものに、機器のポートを走査してセキュリティ上問題のあるサービスを検出するポートスキャン、プログラムのバグを突いて機器のメモリに不正なデータを書き込むバッファオーバーフロー攻撃、機器の大量のパケットを送りつけ CPU やメモリ等の資源を過負荷状態にする DoS/DDoS 攻撃、Web サーバに不正な SQL 文を入力することでデータベース内の情報を取得する SQL インジェクション、偽の送金指示メールを送信して金銭を騙し取るフィッシング等がある。また DNS の名前解決を悪用した攻撃の例としては、感染したノードが動的にドメイン名を生成し、攻撃命令を行うサーバと通信する Domain Generation Algorithm (DGA) 型ボットネット、取得した情報を DNS の問い合わせドメイン名に乗せて外部に情報を送信する DNS トンネリング等がある。本節では、近年盛んに検知技術が研究されている DGA 型ボットネット、フィッシングを取り上げる。

2.1.1 Domain Generation Algorithm (DGA) 型ボットネット

ボットネットとは、悪意のある攻撃者によって乗っ取られた多数のコンピュータ群のことであり、攻撃者の指示を受けて DDoS 攻撃や個人情報の不正入手等様々な被害をもたらす。近年では電子機器技術の発達により多種多様なデバイスがインターネットに接続されるようになり Internet of Things (IoT) と呼ばれる。一方、IoT デバイスの脆弱性をついたマルウェアによる大規模ボットネットが問題となっており、 2016 年 10 月に DNS サービス事業者 Dyn

表 2.1. DGA が生成するドメイン名の例

DGA	ドメイン名
Zeus	krhafeobleyhiytrwuduzlbucutwt.org
	vsmfcabubenvibwolvgilhirvmz.com
Conficker	gfedo.info
	ydqtkptuwsa.org
Kraken	bjjdhgpby.dyndns.org
	scttfzou.mooo.com
Torpig	16ah4a9ax0apra.com
	3ah0a16ax0apra.com
Bedep	rbnfimetzgg9v.com
	akgsuqlnipxhwf.com

への DDoS 攻撃による大規模接続障害を引き起こした Mirai ボットネット [4] などの事例が報告されている。

ボットネットの制御方法には大きく分けて集中制御型や P2P 型、集中制御と P2P のハイブリッド型、 HTTP と P2P を組み合わせた HTTP2P 型があるが [5]、この内集中制御型においては、悪意ある攻撃者は単一の Command & Control (C&C) サーバから各ボットに命令を送り、DDoS 攻撃や機密情報の不正入手、スパムメールの送信などを行わせる。集中制御型の欠点として C&C サーバが単一障害点になり、C&C サーバの IP アドレスを特定し、そのアドレスからの通信を遮断すればボットネット全体の機能を停止できる。特にサーバが固定ドメイン名を持つ場合はブラックリスト方式による検知が容易となる。このブラックリスト方式による検知を回避するのが DGA 型ボットネットである。

DGA 型ボットネットは、 C&C サーバは例えば文献 [6] に示されるようなアルゴリズムを用いて多数のドメイン名を生成し、その一部を DNS サーバに登録して C&C サーバのドメイン名として利用する。各ボットも同じアルゴリズムを用いてドメイン名を次々と生成し、それらを DNS サーバに問い合わせる。問い合わせたドメイン名が C&C サーバのドメイン名と一致した場合は正しく名前解決が行われ、 C&C サーバの IP アドレスが返ってくるため、その IP アドレス宛てにパケットを送ることで C&C サーバと通信を行う。DGA では現在時刻や Twitter のトレンドワードなどランダムな値が入力値として用いられるため実行の度に生成されるドメイン名が異なる。C&C サーバのドメイン名の更新は頻繁に行われるため、ブラックリスト方式での検知から逃れられる。DGA 型ボットネットの例としては、Zeus [7]、Conficker [8]、Kraken [9]、Torpig [10]、Bedep [11] 等がある。各 DGA が生成するドメイン名の例を表 2.1 に示す。

2.1.2　フィッシング

フィッシング (phishing) は電子メール等を用いて利用者を偽の Web サイトに接続させ個人情報を詐取する攻撃である。代表的な攻撃の手口として、送信者を銀行や EC サイトに詐称

したメールを送信し、利用者にメール内のリンクをクリックさせ予め用意した偽の Web サイトに接続させる。利用者にクレジットカード番号やアカウント情報等の個人情報を入力させ、各種情報を入手する。フィッシングに用いられる要素技術としては、 Web ブラウザの脆弱性 [12]、Web ページの UI に細工を施してユーザの意図しない動作を行わせるクリックジャッキング [13]、Web ページに悪意のあるスクリプトを埋め込むクロスサイトスクリプティング [14]、Web ページに接続したユーザにマルウェア等の不正なソフトウェアを強制的にダウンロードさせるドライブバイダウンロード [15] 等がある。またこれらの技術を組み合わせることでフィッシングが可能になる事例も観測されている。例えば 2015 年にはクロスサイトスクリプティングの脆弱性とクリックジャッキングを組み合わせて LinkedIn の Web ページ内のリンクを乗っ取り、ユーザを外部の Web ページに誘導可能となる事例が報告された [16]。

2.2　従来の攻撃検知・防御技術とその問題点

サイバー攻撃から情報を守るため、様々な攻撃検知・防御手法が研究され、実際のネットワーク上で運用されている。代表的な防御手法には、予め設定されたルールに基づきパケットの中継・破棄を行うことで不要なパケットを遮断するファイアウォール、ネットワークやホストの状態を監視して侵入や攻撃を検知し管理者に通知する侵入検知システム (Intrusion Detection System：IDS)、使用中のネットワークから隔離された環境を用意し侵入したマルウェア等を解析するサンドボックス等がある。実際のネットワーク上で運用されている攻撃検知・防御手法は、予め設定した攻撃のパターンに一致したものを攻撃とみなすシグネチャ型がほとんどであり、過去に観測された攻撃を効率よく検知できる。一方でパターンにない攻撃は検知できず、未知の攻撃手法に対応できない。

2.3　機械学習を利用した攻撃検知技術とその問題点

未知の攻撃への検知精度を高めるために、近年コンピュータビジョンや自然言語処理等の分野で活用されている機械学習技術を用いた検知手法が盛んに研究されてきた。本節では 2.1 節で詳細に取り上げた DGA 型ボットネットとフィッシングの検知に機械学習を用いた研究を整理する。

2.3.1　Domain Generation Algorithm (DGA) 型ボットネット

一般的なボットネットの検知手法にはウイルス対策ソフトやリバースエンジニアリング等がある。ウイルス対策ソフトは各コンピュータにインストールされ、コンピュータに侵入したボットをシグネチャ方式で検知し駆除する。しかし、ボットには新たなプログラムに置き換わりウイルス対策ソフトの検知をすり抜けるためのプログラムアップデート機能が備わっている。また攻撃者も既知のウイルス対策ソフトによる検知を逃れるために改良した大量の亜種を日々生成しているため、ウイルス対策ソフトでのボットへの完全な対処は困難である。リバー

スエンジニアリングではボットの検体を解析しプログラム内容を把握することで適切な拡散防止策を立てることができる。また特に DGA 型ボットネットの場合には、検体を解析して DGA を把握することで、同じ DGA で動作するボットや C&C サーバの特定が容易になる。一方で、ボットにはコマンドの暗号化等ソースコードの解析対策が備わっている場合がありリバースエンジニアリングにはかなりの時間を要する。

一般的なボットネットの検知手法に加え、DGA 型ボットネット特有の検知方法として DNS トラフィックの解析が挙げられる。DGA 型ボットネットは各ボットと C&C サーバが通信を行う際に必ず DNS サーバに対し名前解決を試みる。そのため、DNS サーバに送信される名前解決の問い合わせとその応答を解析し、DGA 型ボットネットが生成したとされる悪性ドメイン名を特定することでボットネットを検知できる。本節では DGA 型ボットネット検知手法について利用される特徴量の種類で分類する。

NXDOMAIN

DGA 型ボットネットは、短時間に大量のドメイン名を生成して DNS サーバに問い合わせる。しかし、C&C サーバが持つドメイン名と一致して正しく名前解決が行われ IP アドレスが応答されるドメイン名は一部であり、その他の問い合わせに対しては、ドメイン名が存在しないを意味する NXDOMAIN 応答が返答される。存在しないドメイン名は不在ドメイン名と呼ばれる。一般的な NXDOMAIN 応答はドメイン名のタイプミスやアプリケーションの設定ミス等の場合が多い。それらと比較して、DGA 型ボットネットからの問い合わせがあると、NXDOMAIN 応答の数が通常よりも多くなる。DNS トラフィック全体の中での NXDOMAIN 応答の割合は少ないため [17]、NXDOMAIN 応答を解析することで効率よく DGA 型ボットネットを検知できる。Antonakakis ら [18] は、同一 DGA アルゴリズムを用いるボットが生成したドメイン名は類似するという考えに基づき Pleiades を開発した。Pleiades ではドメイン名の類似度に基づいて NXDOMAIN がクラスタリングされる。また予め既知の DGA をモデル化し、生成されたクラスタが各モデルに割り当てられるか調べる。どのモデルにも割り当てられなかった場合、新たなモデルが生成される。Pleiades は大きく分けて DGA Discovery モジュールと DGA Classification and C&C Detection モジュールの 2 個のモジュールから成り立っている。Pleiades は Alternating decision tree (ADT) アルゴリズムを用いて学習を行い、15 ヶ月間の計測の結果 6 種類の既知の DGA を分類し、6 種類の新しい DGA を発見することに成功した。

言語的特徴

DGA は文字を機械的に組み合わせてドメイン名を生成するため、表 2.1 のように文字列が単語として意味をなさない場合がほとんどである。また同じアルゴリズムで生成されたドメイン名は文字列に含まれる数字・母音の割合等の言語的特徴が似ている可能性が高い。Schiavoni ら [19] は、文字列と IP アドレスに基づく特徴を用いて Phoenix を提案した。Phoenix はドメイン名が DGA によって生成されたものかを判定し、DGA によって生成されたドメイン群がどのボットネットを代表するかを識別するシステムである。Phoenix は

- DGA によって生成されたドメイン名を同定しモデル化する Discovery モジュール
- DNS トラフィックに付随するドメイン名を集め、Discovery モジュールが生成したモデルを用いてドメイン名が自動生成されたものであるか判定する Detection モジュール
- 上記 2 個のモジュールが出力した結果からボットネット検知に有用な情報を抽出する Intelligence and Insights モジュール

の 3 種類のモジュールから構成されている。 Intelligence and Insights モジュールが抽出した情報例として、C&C サーバアドレスの AS 間の移動が挙げられている。Phoenix は特徴量として、文字列内における英語辞書に登録された意味を持つ単語の割合、N-gram 法を用いた正規スコアを利用した。正規スコアは N が 1 から 3 の場合における文字列の N-gram モデルにおいて、それぞれの要素から始まる単語が英語辞書内に現れる頻度を足しあわせ、文字列長で割ったものである。Phoenix は与えられたドメイン名が DGA によって生成されたかどうかを 94.8 % の精度で判別可能にした。

ヒューリスティック

Zhang ら [20] は DGA が生成するドメイン名の言語的特徴の利用をさらに推し進め、既知の DGA 型ボットネットの解析により観測された事実をヒューリスティックとして特徴量に用いた手法として、BotDigger を提案した。BotDigger は DGA 型ボットネットの検知のために、quantity evidence、temporal evidence、linguistic evidence の 3 種類の evidence を組み合わせた chain of evidences という概念を導入している。例えば www.foo.example.com というドメイン名があった場合、最も右側にある com をトップレベルドメイン (TLD) 、右から 2 番目にある example、3 番目にある foo をそれぞれセカンドレベルドメイン (2LD) 、サードレベルドメイン (3LD) とする。また NXDOMAIN 内の 2LD を 2LDNX と定義する。quantity evidence はボットが問い合わせるドメイン名では、正規のホストよりも 2LDNX の数が多いということを意味する。temporal evidence は

1. ボットが C&C サーバのドメイン名を探し始めると、ボットから送信される DNS クエリ中のドメイン名の 2LDNX の数が突然増大する
2. ボットが目的の C&C サーバのドメイン名を突き止めると、2LDNX の数は減少する

の 2 つを意味する。linguistic evidence はあるボットから問い合わせられる不在ドメイン名と実在する C&C サーバのドメイン名は、同じアルゴリズムによって生成されるため文字列の長さ・エントロピー等の言語的特徴が類似することを意味する。BotDigger は代表的な DGA 型ボットネットである Kraken [9] が生成した全てのドメイン名と Conflicker [8] が生成した 99.8 % のドメイン名の検知に成功している。一方、例えばボットが偶然に数回の問い合わせで C&C サーバのドメイン名の名前解決に成功した場合など、evidence に反する挙動が起きた場合についてはシステムは正しく機能できないと述べている。

2.3.2　フィッシング

フィッシングによる被害を軽減するためには，利用者が接続する Web サイトの URL が正常か悪性かを判断し，悪性の場合は遮断や注意喚起をするなどの対策が必要である。本節ではフィッシング検知手法に関連する研究をその着目する特徴量の観点で分類し整理する。

URL 文字列

Zouina ら [21] は URL のみを用いてフィッシング Web サイトを検知するシステムを提案した。特徴量には URL の長さ、ハイフンの数、ドットの数、数字の数、URL に IP アドレスが含まれるかどうか、文字列の類似度の 6 種類が用いられ、分類アルゴリズムには Supprt Vector Machine (SVM) が用いられた。評価では正規の Web サイトとフィッシングサイトを合わせた 2,000 個のデータセットに対し 95.8 % の識別率を達成した。この手法の特徴として、動作が軽量であるためスマートフォンや組み込み機器に適しているという利点が挙げられている。

Sahingoz ら [22] は決定木、Adaboost、KStar、k 近傍法、Random Forest、逐次最小問題最適化法、Naive Bayes の 7 種類の異なる機械学習アルゴリズムを用いたフィッシング検知システムを実装した。検知システムは入力された URL の前処理を行う Data pre-processing module、URL の文字列を辞書的に意味のある単語群に分解する Word decomposer module、単語がランダムな文字列であるかを検知する Random word detection module、タイポスクワッティングの検知を目的とした Maliciousness analysis module から構成される。タイポスクワッティングは URL ハイジャッキングとも呼ばれ、多くのユーザがアクセスする URL に対し、その URL とよく似ておりユーザが打ち間違えやすい URL を意図的に所有しておくことである。また分類に用いる特徴量として、URL から抽出された単語の数、URL 内の数字の個数等計 40 種類の自然言語処理に基づく特徴量、word vectors、両者のハイブリッド型の 3 種類の異なる特徴量を用いた。自作したデータセットにおける評価の結果、機械学習アルゴリズムが Random Forest、特徴量が自然言語処理の場合に検知システムは最も高い 97.9 % の検知精度を達成した。

Web ページの視覚的情報

また URL の情報だけでなく、Web ページの視覚的情報を特徴量として用いる手法も盛んに研究されている。例えば Zhang ら [23] は Web ページに登場する単語内容である Textual content と Web ページを一枚の画像とした Visual content を特徴量として用いたフィッシング検知フレームワークを提案した。フレームワークでは Textual content はベイズ分類器、Visual content は earth mover's distance (EMD) を用いてそれぞれ分類が行われ、これら 2 つの分類結果を重み付けまたはベイズの定理を用いて統合し最終的な分類結果が得られる。

2.3.3 問題点

これまでに取り上げた既存研究は機械学習技術を活用することでいずれも高い検知精度を達成している。しかし、多くの手法は検知システムに入力するデータとして、ログ情報等の、機器が悪性パケット受信後に取得する情報を用いている。機器内で取得した情報から攻撃の検知を試みた場合、攻撃となるパケットは攻撃対象となる機器に到達している可能性が高く、攻撃によるインシデントを未然に防ぐことは難しい。また検知に要する処理時間について論文中で言及されていない場合は、検知の即時性を定量的に議論できない。一方検知に要する処理時間を評価した多くの研究においても、回線速度を維持しつつパケット分類を行うには不十分である。以下に研究例を示す。

例えば DGA 型ボットネットの検知においては、Luo ら [24] が開発した DGASensor はランダムフォレストを用いて DGA 型ボットネットを即座に検知できると論文内で述べられているが、1 秒間に処理可能な DNS クエリは 7,000 個 (1 パケットあたり 143 マイクロ秒) であり、1 クエリが 648 ビットであると仮定してスループットに換算すると 4.536 Mbps である。別の研究例では、[25] では実際の DNS トラフィックから自動的に特徴量を抽出するために Deep Neural Networks (DNNs) が採用されている。しかしながら、その Convolutional NN (CNN) 予測は 1 ドメイン名あたり 50 ミリ秒から 70 ミリ秒を要するため、ネットワークに遅延が生じる。

また悪性 URL に基づくフィッシングの検知においては、Jain ら [26] は従来の機械学習を利用した検知手法で特徴量として用いられていた Web ページのトラフィック情報や WHOIS レコード等の第三者サービスから取得可能な情報を用いず、Web ページに接続するクライアントが直接取得可能なページの URL とソースコードのみを特徴量として用いる検知手法を提案した。この手法は 10 秒から 13 秒の応答時間を要する既存手法より短い応答時間を達成したとしつつも、2 秒から 6 秒の応答時間を要する。従って Web ブラウザに検知手法が導入された場合、利用者は Web ページへのアクセスに 2 秒から 6 秒を要する。

2.4 既存技術のまとめ

未知の攻撃を検知するため、また検知を自動化するために機械学習を用いた攻撃検知手法が数多く研究されているが、多くの研究はネットワーク上での攻撃検知には適していないことを示した。こうした状況からネットワーク上において機械学習による未知の攻撃検知を迅速に行い、インシデントを未然に防ぐための機構が求められている。本研究では、ネットワーク上のパケットに対して機械学習を用いた攻撃検知処理を高速に行い、パケットの分類を行う処理機構を提案することを目的とする。

第 3 章

機械学習を利用したリアルタイム悪性パケット分類

本章では、機械学習を利用したリアルタイム悪性パケット分類を実現するためのアーキテクチャについて議論する。まずアーキテクチャが満たすべき要件を定義する。次に要件を満たすアーキテクチャの概要とその実装について説明し、最後に予備実験によるアーキテクチャの性能評価を行う。

3.1 要件

本研究において提案する、機械学習を利用したリアルタイム悪性パケット分類を実現するアーキテクチャが満たすべき要件を以下に定義する。

1. リアルタイム検知処理
   攻撃によるインシデントを未然に防ぐには、悪意のあるパケットが攻撃対象に到達する前にネットワーク上で遮断するのが理想である。悪性パケット分類によりこれを達成できるが、分類に要する処理時間がネットワーク利用者の遅延となる。従って、提案アーキテクチャはパケット受信から送信までの処理時間ができるだけ短くなければならない。具体的には、現在広く利用されている回線速度である 1 - 10 Gbps のスループットを維持しなければならない。
2. 分類機能のモジュール化とモジュール追加・削除の柔軟性
   分類処理を行う部分はモジュールとして独立化し、分類システムは分類モジュールを置換や追加/削除できる十分な柔軟性を持つ必要がある。なぜなら 1 つの攻撃検知器で全ての悪意ある攻撃を防ぐことはできないからである。従って、様々な攻撃を検知し防ぐために多数の検知器が分類モジュールとして必要であり、同時に複数の分類モジュールを動作させる必要がある。加えて、新たな検知器が絶え間なく開発されているため、古い検知器の置換や他の検知器の追加は容易でなければならない。

3.2 ソフトウェアによる高速パケット処理技術

ネットワーク側での機械学習を用いた攻撃検知を達成するためには、パケット処理性能を向上させる既存の手法を活用する必要がある。高速パケット処理技術はハードウェアによる実装とソフトウェアによる実装に大別される。FPGA を用いた実装 [27] のようなハードウェアによる実装はソフトウェアによる実装よりも高速に動作するが、論理回路の設計が必要になる等実装完了までの工程はソフトウェアよりも多い。本研究では提案アーキテクチャの実装にあたり、要件 2 を満たすためより実装コストが小さく分類モジュールへの柔軟性を利用できるソフトウェアによる実装を採用した。

汎用 Linux サーバ上でソフトウェアによるネットワーク処理を行う場合、Linux カーネルによるシステムコール呼び出しやハードウェア割り込み等によるオーバーヘッドが発生するため機器のネットワーク性能を最大限引き出すことができない。これに対処するために Data Plane Development Kit (DPDK) [28] や netmap [29] などの、Linux カーネルをバイパスして直接ハードウェアにアクセスしてパケットデータを送受信し、高速のパケット I/O 処理を実現するライブラリやフレームワークが提案されている。

さらにこうした高速パケット I/O 処理を活用して既存のハードウェアルーターやスイッチと同じ機能をソフトウェアで実現する技術も開発されている。例えば、Lagopus [30] は DPDK を利用した高性能の SDN ソフトウェアスイッチであり、64 byte の最小パケットを 10.1 Mpps (5.66 Gbps) の性能で処理できる。Lagopus では、高速なパケット処理を実現するために、各処理を並列に実行するパイプライン構造や、送受信処理を異なる CPU コアに割り当てるピンニングといった工夫がされている。

3.3 機械学習を利用したパケット分類処理アーキテクチャ

本節では、図 3.1 に示す機械学習を利用したパケット分類処理アーキテクチャを提案し概要について説明する。提案アーキテクチャは、パケット処理を担う受信、パーサ、特徴量抽出、分類、送信の 5 つのパートから構成される。さらに受信、送信パートを送受信モジュールとしてまとめ、パーサ、特徴量抽出、分類パートを追加・削除が容易な分類モジュールとしてまとめる。各パートはそれぞれ次の動作を行う。

受信

NIC に到着したパケットデータをシステム内に読み込む。

パーサ

パケットデータから受信元/送信元 IP アドレスや受信元/送信元ポート番号などのヘッダ情報、アプリケーション固有の値などのペイロード情報を抜き出す。抜き出される値は検知対象と分類に用いる特徴量に応じて決まる。

特徴量抽出

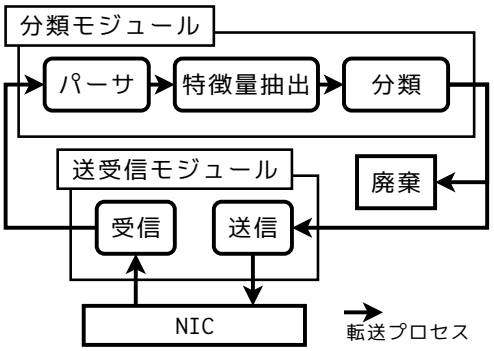


図 3.1. 機械学習を利用したパケット分類処理アーキテクチャ

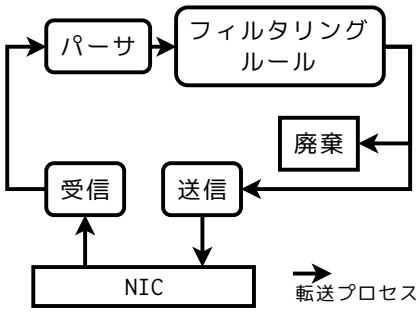


図 3.2. ファイアウォールのパケット処理アーキテクチャ例

パーサパートが抜き出した値を特徴量ベクトルに変換する。

分類

特徴量ベクトルを用いてパケットを良性または悪性のどちらかに分類する。もし受信したパケットが良性と分類された場合は、パケットを送信パートに転送する。しかし、悪性と分類された場合は、パケットをシステム内で破棄する。本論文では分類結果の正当性については問わないものとする。

送信

パケットデータを NIC に送信する。

提案アーキテクチャの比較対象として、ファイアウォールのパケット処理アーキテクチャ例を図 3.2 に示す。ファイアウォールでは、受信したパケットをパースし宛先 IP アドレスやポート番号等の情報を取得する。次に予め定義したフィルタリングルールと取得した情報を照らし合わせ、結果に応じてパケットの送信または破棄を行う。これに対し、提案アーキテクチャはパケット送信または破棄の判断をフィルタリングルールではなく学習済み分類器により行うため、フィルタリング処理が機械学習による分類処理に置換された形になる。さらに、機械学習分類の入力には特徴量ベクトルが必要であるため、パケットをパースして取得した情報を特徴量ベクトルに変換する特徴量抽出処理が追加される。

3.4　機械学習を利用したパケット分類処理アーキテクチャの実装

本節では提案アーキテクチャの実装について説明する。本研究では提案アーキテクチャのパケット転送性能を明らかにし、ボトルネックとなる処理を分析するためにパケット分類システムを実装した。具体的には、受信と送信パートにおいては、パケット転送時の Linux カーネルによるオーバーヘッドを回避するために DPDK [28] ライブラリを用いた。DPDK のバー

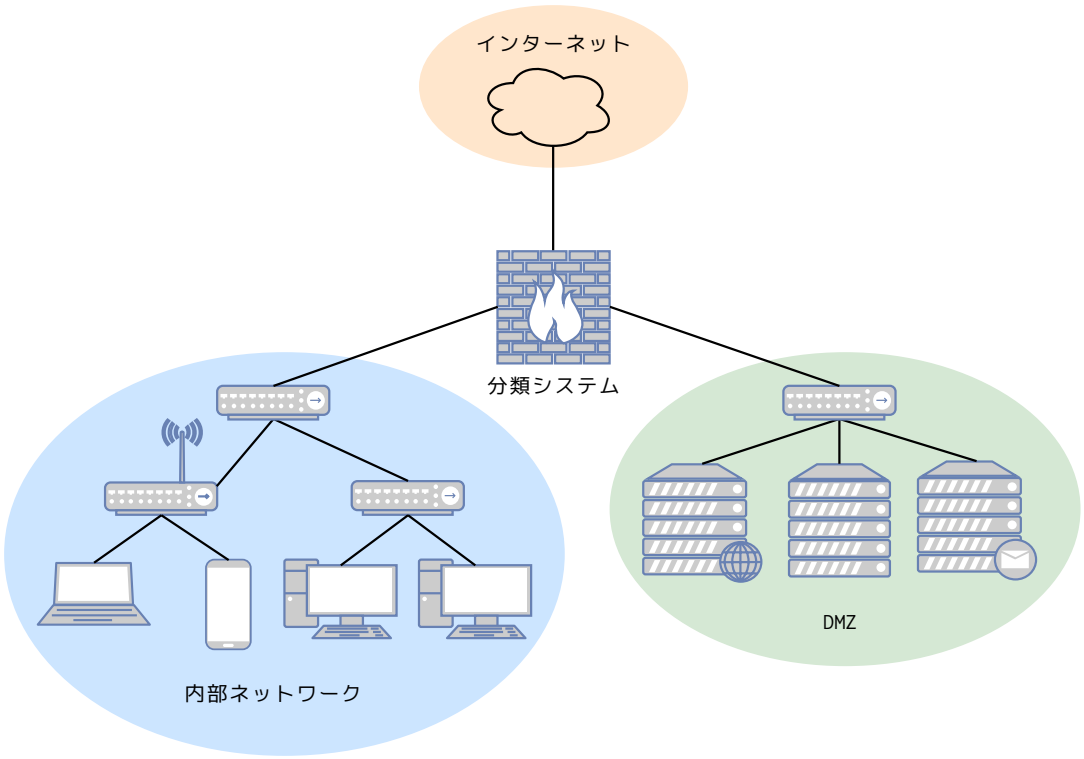


図 3.3. 分類システムの想定動作環境

ジョンは 17.11 である。また複数の異なる分類モジュールにおける性能差を評価するために、分類パートにおいては 3 種類の機械学習ライブラリまたはフレームワーク (OpenCV [31]、LIBSVM [32]、TensorFlow [33]) を使用した。ソフトウェアのバージョンはそれぞれ 4.0.0、3.23、1.8.0 である。パーサパートと特徴量抽出パートは自作した。使用したプログラミング言語は C 言語と C++ 言語である。使用したコンパイラは g++ (GCC) 8.2.1 である。

3.5 想定環境

本研究の目的であるリアルタイム悪性パケット分類システムはネットワークパス上に設置され、システムを通過するパケットを分類器に入力し良性または悪性の判定を行う。悪性と判定されたパケットは即座に破棄される。システムの設置場所として、図 3.3 に示す組織の内部ネットワークとインターネット、非武装地帯 (DeMilitarized Zone; DMZ) の間を想定する。DMZ にはインターネットに公開する Web サーバや DNS サーバ、メールサーバ等が設置される。インターネットから内部ネットワークや DMZ へと流れる悪性パケットには DoS/DDoS 攻撃、ポートスキャン等がある。また内部ネットワークや DMZ からインターネットへと流れる悪性パケットには DGA により作成された DNS クエリ、フィッシングサイトへの HTTP 接続等がある。

3.6　予備実験

3.3 節と 3.4 節で設計及び実装を行った提案アーキテクチャの基本性能を明らかにし、機械学習を利用したパケット分類のボトルネックを明らかにするため、予備実験として パケット転送性能評価と特徴量ベクトル次元数の依存関係の評価の 2 種類の性能計測を行った。本節では分類モジュールの詳細を含む評価手順の詳細と実験手順、結果について説明する。

3.6.1　分類モジュールの詳細

検知対象

分類システムが行う分類タスクとして、多数の DNS クエリの中から DGA により生成された悪性 DNS クエリを分類するタスクを設定した。予備実験で行う分類は一例であり本研究の検知対象は DGA に限定されない。システムは適切に学習された検出器を分類モジュールとして使用することで他の攻撃も検知可能である。

検知アルゴリズム

検知アルゴリズムとして、DGA 型ボットネットの検知を対象とした既存研究で使用されている Random Forest、Supprt Vector Machine (SVM) と Neural Network を用いた。Random Forest は決定木による複数の弱学習器を統合させて汎化能力を向上させるアンサンブル学習アルゴリズムである。本研究では学習パラメータとして木の最大の深さを 2,147,483,647、葉ノードの最小サンプル数を 2 とした。SVM は線形入力素子を利用して 2 クラスのパターン識別器を構成する手法であり、訓練サンプルから各データ点との距離が最大となるマージン最大化超平面を求めるという基準 (超平面分離定理) で線形入力素子のパラメータを学習する。本研究では SVM の種類と SVM のカーネル関数として、nu-SVC と線形関数をそれぞれ選択した。Neural Network はシナプスの結合によりネットワークを形成した人工ニューロン（ノード）が、学習によってシナプスの結合強度を変化させ、問題解決能力を持つようなモデルである。本研究ではパラメータとして活性化関数をソフトマックス関数、層の数を 1、損失関数を交差エントロピー誤差、学習方法を勾配降下法に設定した。このモデルは MNIST [34] のデータセットにおいて約 91 % の分類精度を達成した。

データセット

データセットとして、公開された利用可能なデータセット [35, 36] に基づき、良性または悪性のラベルが付与されたドメイン名のリストを作成した。具体的には、55,000 個の良性ドメイン名を Alexa Top 1 Million Sites [35] からランダムに抽出し、55,000 個の悪性ドメイン名を [36] からランダムに抽出した。合計 110,000 個のドメイン名の内、100,000 個のドメイン名を学習用、10,000 個を評価用として用いた。学習用データセットと評価用データセットにおける良性ドメイン名と悪性ドメイン名の割合はどちらも 1:1 である。

アルゴリズム 1 特徴量抽出のアルゴリズム

Input: $domain, dimension, features$

Output: $vector$

```
vector[dimension] ⇐ (0, 0, ..., 0)
for index1, character1 in enumerate(domain) do
  for index2, character2 in enumerate(features) do
    if character1 = character2 then
      newindex ⇐ index2 + 39 ∗ index1
      vector[newindex] ⇐ 1
    end if
  end for
end for
```

分類で用いた特徴量

分類モデルの学習のための特徴量抽出の方式として、ドメイン名の文字の順序を考慮した bag-of-words モデル [37] を用いた。具体的には、それぞれのドメイン名の文字列から 2,847 次元のベクトルが抽出され、ベクトルのそれぞれの要素は 0 または 1 が設定される。ドメイン名には 39 種類の文字 (a, b, c, ⋯, y, z, 0, 1, ⋯, 8, 9, ., -, _) が用いられる。また実験データセットにおけるドメイン名の最大長は 73 である。ドメイン名の N 番目の文字が、39 種類の文字の中で K 番目である X であった場合、生成される特徴量ベクトルの 39(N-1)+K 番目の要素に 1 が設定される。例えば、ドメイン名 *ask.com* が特徴量ベクトルに変換される場合、1 (a)、58 (s)、89 (k)、154 (.)、159 (c)、210 (o)、247 (m) 番目の 7 個の要素に 1 が設定され、残りの 2,840 個の要素には 0 が設定される。

特徴量抽出のアルゴリズムをアルゴリズム 1 に示す。特徴量抽出に要する処理時間を削減するため、検索に用いる文字列（アルゴリズム 1 内の変数 *features* ）における文字の並び順を Alexa Top 1 Million Sites [35] に登場する文字の頻度順 (oe.acmritsnludpgbhkfyvw-zxj12q04389576_) に設定した。特徴量ベクトルに変換する処理時間は並び順をアルファベット順 (abcdefghijklmnopqrstuvwxyz0123456789.-_) に設定した場合と比較して約 27.8% 削減された。

分類器の学習

パケット分類に用いられる分類器の学習は分類前に一度のみ行い、実験途中での再学習は行わない。これは学習処理に要する時間がパケット処理に要する時間よりも大きく、実験中に分類器を更新できないためである。

表 3.1. サーバ環境

	送信サーバ	転送サーバ	受信サーバ
CPU	Intel Core i7-4770K 3.50 GHz	Intel Core i7-6700K 4.00 GHz	Intel Core i7-4770K 3.50 GHz
Memory	32 GB (8 GB x4) DDR3 1333 MHz	32 GB (8 GB x4) DDR4 2133 MHz	32 GB (8 GB x4) DDR3 1600 MHz
OS	GNU/Linux Ubuntu 16.04.5 LTS 64-bit	GNU/Linux Arch Linux	FreeBSD
Kernel Version	4.4.0-137-generic	4.17.12-arch1-1-ARCH	11.1-STABLE
NIC	Intel Ethernet Connection I219-V	Intel 82571EB Gigabit Ethernet Controller	Intel Ethernet Connection I219-V

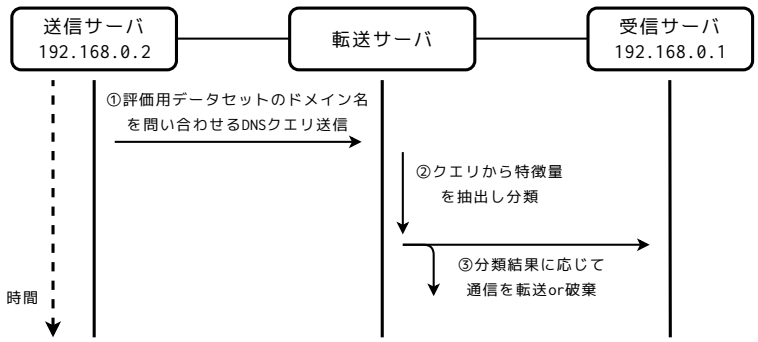


図 3.4. 予備実験の概要

3.6.2　パケット転送性能評価

実験では 3 台の物理サーバ（送信サーバ、転送サーバ、受信サーバ）を使用した。サーバの環境を表 3.1 に示す。送信サーバと受信サーバは転送サーバと接続しており、転送サーバは 3.6.1 節で示した 3 種類の分類モジュールを組み込んだ提案システムを実装している。実験の概要を図 3.4 に示す。送信サーバは評価用データセットのドメイン名から生成した DNS クエリを受信サーバに送信する。DNS クエリの送信にはパケット生成をサポートする Python のライブラリである Scapy [38] を用いた。使用した Scapy のバージョンは 2.4.0 である。また使用した Python のバージョンは 2.7.12 である。実験環境では回線速度は 1 Gbps に設定した。受信パートと送信パートのみで構成される DPDK のソースコード内のサンプル転送プログラムを用いてパケット転送性能を計測した結果、回線速度でパケット転送できることを確認した。従って送受信モジュールのパケット転送遅延への影響は小さい。実験では図 3.1 に示した受信・パーサ・特徴量抽出・分類・送信の各パートの処理時間を、プログラム内の各パートの前後に *clock_gettime* 関数を埋め込み計測した。

図 3.5 に各パートの処理時間を対数目盛で示す。また、表 3.2 に各パートの平均処理時間

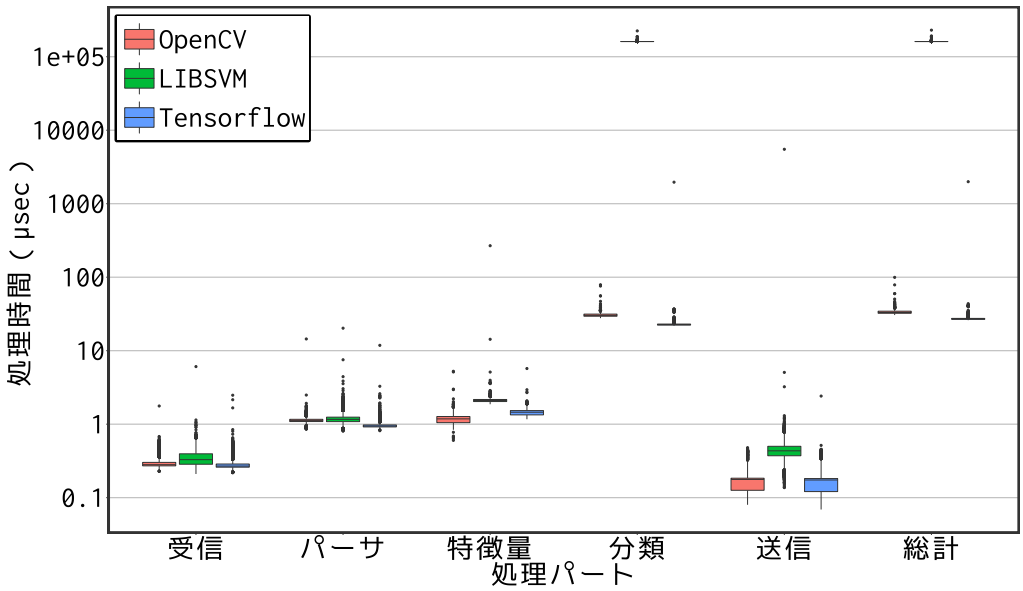


図 3.5. 各パートの処理時間

表 3.2. 各パートの平均処理時間（マイクロ秒）

実装（アルゴリズム）	受信	パーサ	特徴量抽出	分類	送信	総計
OpenCV (Random Forest)	0.296	1.131	1.174	30.526	0.164	33.458
LIBSVM (SVM)	0.347	1.185	2.137	161061.989	0.996	161068.748
Tensorflow (Neural Network)	0.282	0.962	1.447	22.966	0.162	27.408

を示す。結果は全ての分類モジュールにおいて、分類パートが最も時間を要するパートであることを示している。具体的には、全体処理時間に占める分類処理時間の割合が OpenCV、LIBSVM、Tensorflow の場合においてそれぞれ 91.237 %、99.997 %、83.793 % である。またパケット転送性能のスループットの計算量を把握するために、全体処理時間の平均値をスループット概算値に変換した。その値は OpenCV、LIBSVM、Tensorflow でそれぞれ 21.39 Mbps (29.89 Kpps)、4.44 Kbps (6.21 pps)、27.72 Mbps (38.73 Kpps) となる。カッコ内の値は単位時間あたりに処理されたパケット数 (pps: packet per second) である。スループット概算値への変換は、評価用データセットの DNS クエリパケットの平均長が 715.6736 bit であるため式 3.1 に基づき行った。

$$（スループット概算値）= \frac{715.6736}{（処理時間の平均値）} \tag{3.1}$$

3.6.3 特徴量ベクトル次元数の依存関係

本節では分類パートに入力する特徴量ベクトルの大きさの処理時間への影響を計測した。本実験では、特徴量ベクトルの次元数を 10 から 2,847 まで変化させ各条件で分類パートの処理

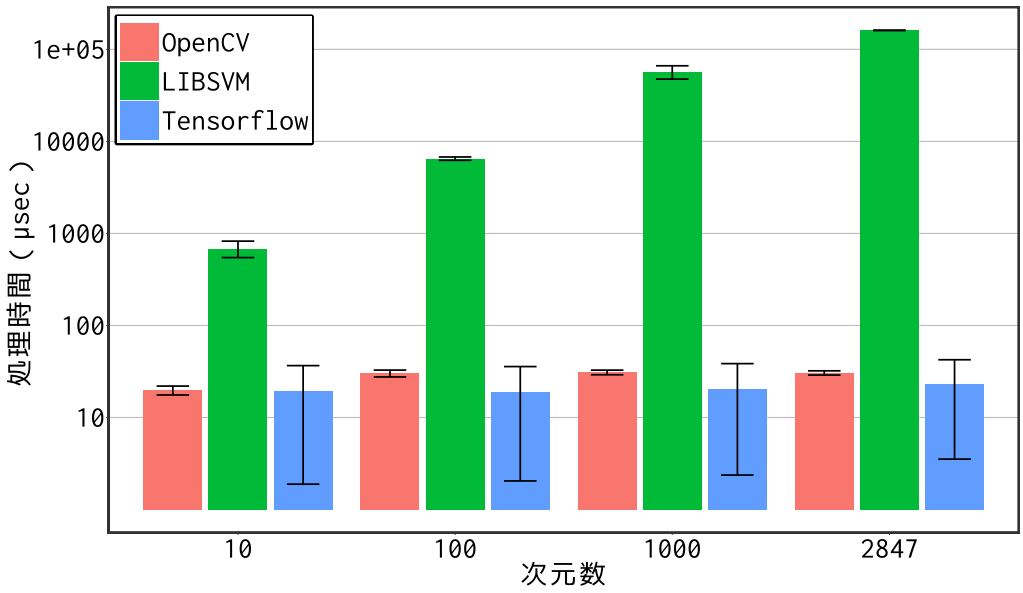


図 3.6. 各次元数における分類パートの処理時間

表 3.3. 各次元数における分類パートの平均処理時間（マイクロ秒）

	次元数			
実装（アルゴリズム）	**10**	**100**	**1000**	**2847**
OpenCV (Random Forest)	19.754	30.154	30.917	30.526
LIBSVM (SVM)	685.689	6507.751	56948.418	161061.989
Tensorflow (Neural Network)	19.246	18.896	20.426	22.966

時間を比較した。特徴量ベクトルの次元数以外の条件は全て 3.6.2 節の性能評価実験と同じである。

図 3.6 に各次元数における分類パートの処理時間を対数目盛で示す。また表 3.3 に各次元数における分類パートの平均処理時間を示す。LIBSVM では次元数の増加にともない線形に平均処理時間が増加している。一方で、OpenCV と Tensorflow における平均処理時間の増加率は次元数 10 次元と 2,847 次元で比較すると 54.5 %、19.3 % である。また平均処理時間はいずれの分類モジュール・次元数においても 10 マイクロ秒のオーダーを要する。この結果は 3 種類の分類モジュールが特徴量ベクトルの次元数を 10 まで削減させても 10 マイクロ秒以上の処理時間を要することを示している。

表 3.4 から表 3.7 に特徴量ベクトルの各次元数における分類タスクの全体正解率、適合率、検出率、偽陽性率をそれぞれ示す。ここで全体正解率は全ドメイン名の中で正しく分類されたドメイン名の割合であり、適合率は悪性であると分類したドメイン名の中で実際に悪性であったものの割合である。また検出率は実際の悪性ドメインの中で悪性と分類されたドメイン名の割合であり、偽陽性率は実際の良性ドメインの中で悪性と分類されたドメイン名の割合である。全体正解率、適合率、検出率は数値が大きいほど、偽陽性率は数値が小さいほど分類モ

表 3.4. 各次元数における全体正解率 (%)

	次元数			
	10	**100**	**1000**	**2847**
OpenCV	59.62	62.29	76.73	70.24
LIBSVM	42.88	79.45	91.46	91.42
Tensorflow	59.97	75.01	87.21	87.22

表 3.5. 各次元数における適合率 (%)

	次元数			
	10	**100**	**1000**	**2847**
OpenCV	56.67	57.50	98.55	99.61
LIBSVM	40.28	81.16	96.25	96.29
Tensorflow	57.29	77.45	91.50	91.61

表 3.6. 各次元数における検出率 (%)

	次元数			
	10	**100**	**1000**	**2847**
OpenCV	81.74	94.24	54.26	40.64
LIBSVM	29.50	76.70	86.28	86.16
Tensorflow	78.34	70.56	82.04	81.94

表 3.7. 各次元数における偽陽性率 (%)

	次元数			
	10	**100**	**1000**	**2847**
OpenCV	62.50	69.66	0.80	0.16
LIBSVM	43.74	17.80	3.36	3.32
Tensorflow	58.40	20.54	7.62	7.50

ジュールの検知精度が高いと判断できる。結果より特徴量ベクトルの次元数を大きくすることでいずれの分類モジュールにおいても検知精度が向上する傾向にあることを確認した。実験で得られた検知精度は一例であり、実運用に用いる場合は特徴量の選び方・分類アルゴリズムをさらに検討し検知精度を高める必要がある。

第 4 章

機械学習によるリアルタイム悪性 DNS クエリ分類システム

本章では、3.6 節で示した性能評価の結果を踏まえつつ、機械学習を利用したリアルタイム攻撃検知を達成するためにパケット処理以外の部分で処理時間を稼ぐ方法について議論する。さらにそのアイデアを元に、提案アーキテクチャの改良手法として、DNS の名前解決に要する時間を活用した悪性 DNS クエリ検知システムを提案する。

4.1 分類処理時間の必要性

3.6 節での性能評価で示した通り、プロトタイプシステムのパケット転送性能は Tensorflow の場合において 10 Mbps の計算量であり、要求される転送性能である 1 - 10 Gbps の水準に大きく劣る。また処理全体の中で分類パートが最も時間を要しており、特徴量ベクトル次元数の依存関係の計測では、入力する特徴量ベクトルの次元数削減による分類処理時間の削減には限度があることが示された。一方で、より多くの特徴量と長い分類処理時間が機械学習の分類精度向上に寄与することを確認した。従って、高い分類精度を維持しながら要件を満たすには、システム全体の処理時間を削減すると同時に、より多くの分類パートの処理時間を要する必要がある。本研究ではその目的を達成するため、個々の通信プロトコルの特徴と実際のネットワークを流れるトラフィックの特性に着目する。

4.2 DNS の名前解決時間の活用

DNS は 数字で表現された IP アドレスを人間が把握しやすいドメイン名に対応付ける仕組みであり、ドメイン名から IP アドレスへの変換処理は名前解決と呼ばれる。DNS サーバにはドメイン名の情報を保持する権威サーバと名前解決の問い合わせを行うキャッシュサーバがある。ドメイン名は階層構造で管理され、名前解決ではキャッシュサーバにより上位の DNS サーバから下位の DNS サーバへと再帰的に問い合わせが行なわれる。また同じ問い合わせを繰り返し行うことを避けるため、キャッシュサーバは問い合わせ結果を一定時間記憶（キャッ

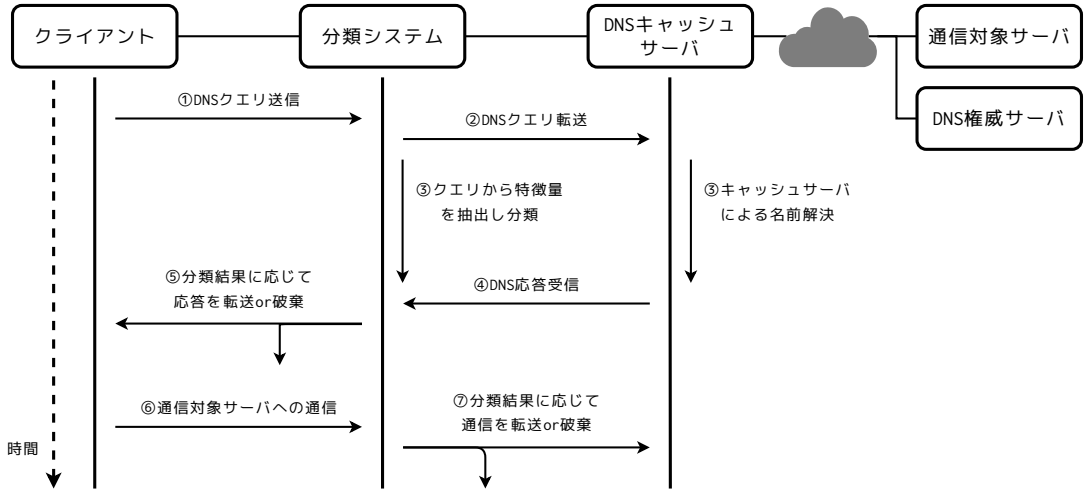


図 4.1. DNS の名前解決に要する時間を活用するモデル

シュ）する。IP アドレスが不明な機器との通信前には必ず名前解決が行なわれるため、悪性ホストドメイン名の DNS 問い合わせを検知しクエリまたは応答を遮断すれば悪性ホストへの通信を防止できる。

DNS において、キャッシュサーバがクライアントから問合わせを受信し名前解決を行い、その結果をクライアントに送信するまでには一定の時間を要する。具体的には、ネットワーク環境にも依るが、問合わせがキャッシュにヒットした場合は 1 ミリ秒から 10 ミリ秒、ヒットせず再帰的な問合わせを行う場合は 10 ミリ秒から 100 ミリ秒の時間を要する。本研究ではこの名前解決に要する時間を追加の分類処理時間として活用する。アイデアを図式したものを図 4.1 に示す。分類システムは DNS クエリを受信後すぐに転送し、転送後分類モジュールによる分類を行う。転送したクエリに対する応答を受信する前に分類処理が完了しクエリが悪性と判定された場合、応答を破棄することでクライアントによる悪性ホストドメイン名の名前解決を防止できる。本研究では、図 3.1 に示す提案アーキテクチャを拡張し、機械学習による悪性 DNS クエリ検知をリアルタイムに実現するパケット分類システムを提案する。

また予備実験で使用したトラフィックデータにデータの重複はないが、実際のネットワークを流れるトラフィックは定期的に行われるハートビート通信等の周期的なパターンを持つ傾向にある。従って分類結果をホワイトリストやブラックリストのようなリストに記憶させ、重複するトラフィックデータを予めリストとのパターンマッチングにより取り除き、分類処理に入力するパケット数を削減することで、システム全体の処理時間を削減できる。

4.3 システム概要

本研究では 4.2 節で示した 2 つのアイデアを図 4.2 に示すシステムに統合した。統合の要点は、DNS の名前解決の間に時間を要する機械学習分類を完了させ、その分類結果を単純な

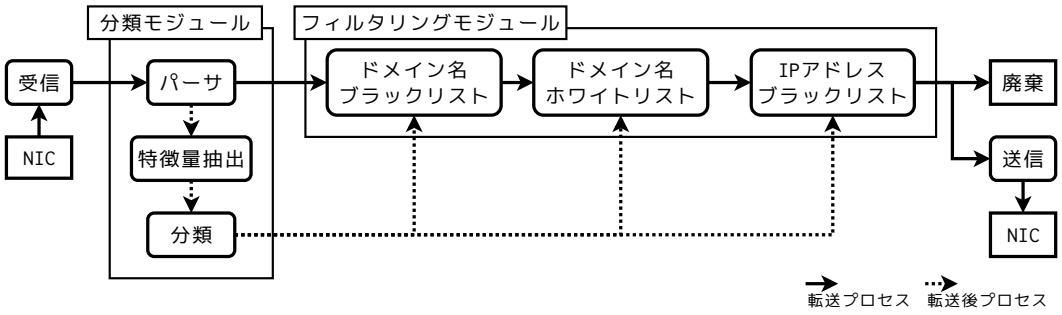


図 4.2. 提案システムの概要

フィルタリング手法に適用させることである。新たな処理要素としてドメイン名ホワイトリスト、ドメイン名ブラックリスト、IP アドレスブラックリストの 3 種類のフィルタリング手法を追加した。これらはフィルタリングモジュールとしてまとめられる。また 5 種類の新たな処理パートとして、ドメイン名ホワイトリストによるフィルタリング、ドメイン名ブラックリストによるフィルタリング、IP アドレスブラックリストによるフィルタリング、ドメイン名ブラックリストの更新、IP アドレスブラックリストの更新が追加される。各パートはそれぞれ次の動作を行う。

ドメイン名ホワイトリストによるフィルタリング
: 入力されたドメイン名がホワイトリスト内のドメイン名と完全一致するか検索する。一致した場合はそのドメイン名は良性と判定される。

ドメイン名ブラックリストによるフィルタリング
: 入力されたドメイン名がブラックリスト内のドメイン名と完全一致するか検索する。一致した場合はそのドメイン名は悪性と判定される。

IP アドレスブラックリストによるフィルタリング
: 入力された IP アドレスがブラックリスト内の IP アドレスと完全一致するか検索する。一致した場合はそのドメイン名は悪性と判定される。

ドメイン名ブラックリストの更新
: 入力されたドメイン名をドメイン名ブラックリストに追加する。

IP アドレスブラックリストの更新
: 入力された IP アドレスを IP アドレスブラックリストに追加する。

提案システムでは受信したパケットをパーサパートの結果により DNS クエリ、DNS 応答、それ以外のパケットの 3 つに分類し、それぞれのパケットに応じた処理を次の 3 種類のステージに区別する。

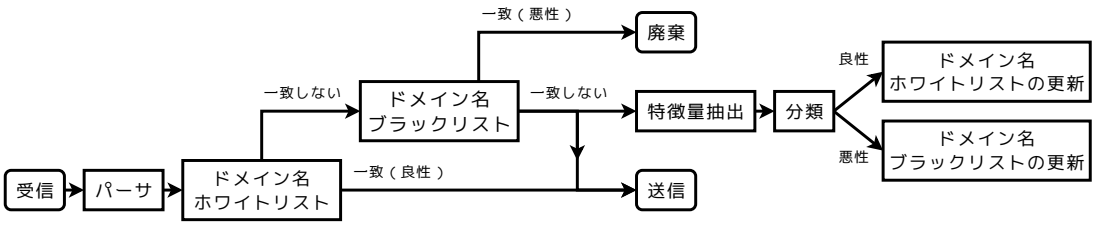


図 4.3. ステージ 1：クライアントから DNS サーバ（DNS クエリの処理）

4.3.1 ステージ 1：クライアントから DNS サーバ（DNS クエリの処理）

クライアントから送信された DNS クエリに対してはドメイン名ホワイトリストによるフィルタリングが行なわれる。問合わせドメイン名がホワイトリストに登録された既知のドメイン名と合致し良性と分類された場合、DNS クエリパケットは送信パートに転送される。一方、問合わせドメイン名がホワイトリストに一致しなかった場合は、ドメイン名ブラックリストによるフィルタリングが行われる。問合わせドメイン名が悪性と分類された場合、パケットは転送されず破棄される。一方、問合わせドメイン名がブラックリストに一致しなかった場合、ドメイン名はシステムの動作後初めて観測されたドメイン名であり、分類モジュールにより良性か悪性か分類する必要がある。分類は名前解決処理の間に行うため、パケットは送信パートに転送される。次に、パケット送信後、特徴量抽出パートが問合わせドメイン名を特徴量ベクトルに変換し、分類パートが分類する。分類結果が良性であった場合、問合わせドメイン名はドメイン名ホワイトリストに追加される。反対に分類結果が悪性であった場合、ドメイン名はドメイン名ブラックリストに追加される。目的は同じ問合わせドメイン名を持つクエリを機械学習による分類を行うことなく早期に遮断するためである。

4.3.2 ステージ 2：DNS サーバからクライアント（DNS 応答の処理）

DNS サーバからの DNS 応答に対しては最初にドメイン名ホワイトリストによるフィルタリングを適用する。問合わせドメイン名が良性と分類された場合、DNS 応答パケットは送信パートに転送される。一方、問合わせドメイン名がホワイトリストに一致しなかった場合、パケットはドメイン名ブラックリストによるフィルタリングが行われる。問合わせドメイン名が悪性と分類された場合、パケットは破棄され応答メッセージ内の RCODE が調べられる。RCODE には NoError（正常）、FormErr（フォーマットエラー）、ServFail（サーバエラー）、NXDOMAIN（問い合わせされた名前は存在しない）、NotImp（問い合わせは実装していない）、Refused（何らかの理由による拒否）等がある。RCODE が NoError であった場合は、ドメイン名に対応する IP アドレスが存在するため、当該 IP アドレスを抽出し、IP アドレスブラックリストに追加する。目的はステージ 3 で IP アドレスブラックリストでのフィルタリングを行い、当該 IP アドレスへのパケットを遮断するためである。一方問合わせドメイン名

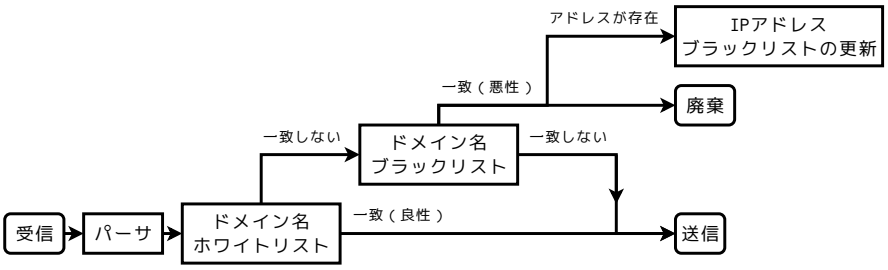


図 4.4. ステージ 2：DNS サーバからクライアント（DNS 応答の処理）

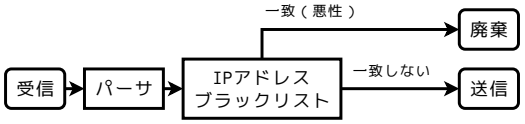


図 4.5. ステージ 3：クライアントからサーバ（名前解決された IP アドレスへの最初の TCP パケットまたは UDP パケットの処理）

が良性と分類された場合、受信したパケットは送信パートに転送される。

4.3.3　ステージ 3：クライアントからサーバ（名前解決された IP アドレスへの最初の TCP パケットまたは UDP パケットの処理）

DNS クエリと DNS 応答以外のパケットに対しては宛先 IP アドレスに対して IP アドレスブラックリストによるフィルタリングが行なわれる。宛先 IP アドレスが IP アドレスブラックリスト内の IP アドレスと一致した場合、パケットは転送されずシステム内で破棄される。一方、宛先 IP アドレスが IP アドレスブラックリストに一致しなければ、パケットは送信パートに転送される。ステージ 3 は DNS クエリと DNS 応答以外の全てのパケットに対して行われるが、その目的は名前解決された悪性 IP アドレスへの最初の TCP パケットまたは UDP パケットを検知することである。ステージ 1 での機械学習による悪性判定がステージ 2 までに完了せずステージ 3 までに完了した場合、クライアントがその悪性 IP アドレスへの通信を行うことはできなくなり、インシデントを未然に防止できる。

第 5 章

評価

本システムの目的は悪性ホストの名前解決を防ぐことでインシデントを未然に防止することである。従って評価では以下の 2 つの評価項目を設定した。

- 悪性ホストへの通信を通信の到達前にどれだけ遮断できるか
- 実際にネットワークを流れたトラフィック（実トラフィック）でも同様の動作をするか

本章ではそれぞれの評価項目に対応した 2 種類の実験により提案システムを評価する。

5.1 提案システムの実装

本研究では図 4.2 に示すシステムを、3 章で議論したアーキテクチャを拡張する形で実装した。ドメイン名ブラックリスト、ドメイン名ホワイトリスト、IP アドレスブラックリストの 3 種類のフィルタリングモジュールをハッシュテーブルを用いて実装した。ハッシュテーブルの実装には GLib [39] を用いた。使用した GLib のバージョンは 2.58.1 である。一般的に Alexa Top 1 Million Sites [35] のようなブラックリストやホワイトリストの初期値として用いられ得るドメイン名の公開リストはサブドメインを含まないものが多い。一方、ホワイトリストの検索においてはサブドメインを除く部分が一致すれば良性ドメインと判断できる。従って、サブドメインを持つドメイン名とドメイン名ホワイトリスト内のドメイン名の一致率を向上させるため、セカンドレベルドメイン (SLD) のリスト [40] を用いて、抽出したドメイン名を 1 つのラベルとセカンドレベルドメインとトップレベルドメイン (TLD) で構成された部分文字列に変換する処理をパースパートに追加した。変換処理を高速化するため、SLD リストのデータ構造には木構造を用いた。ドメイン名が www.u-tokyo.ac.jp の場合の変換処理を（例）に示す。

（例）$\mathrm{www.u-tokyo.ac.jp} \rightarrow \underbrace{\mathrm{u-tokyo}}_{\text{ラベル}} . \underbrace{\mathrm{ac}}_{\mathrm{SLD}} . \underbrace{\mathrm{jp}}_{\mathrm{TLD}}$

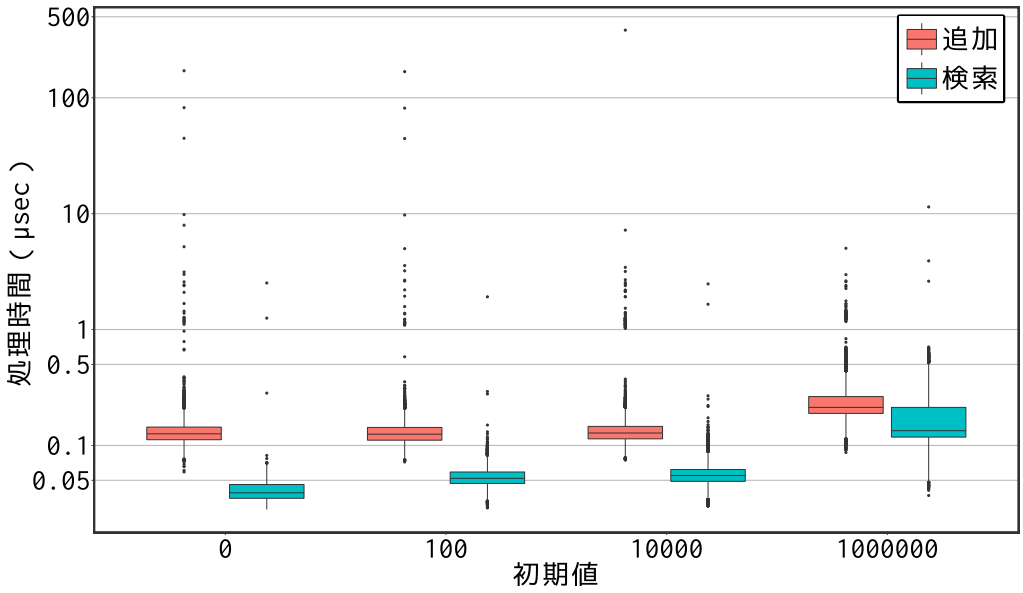


図 5.1. 各初期値におけるドメイン名ホワイトリストの処理時間

表 5.1. 各初期値におけるドメイン名ホワイトリストの平均処理時間（マイクロ秒）

	初期値			
処理	**1**	**100**	**10,000**	**1,000,000**
追加	0.165	0.163	0.179	0.244
検索	0.040	0.054	0.057	0.180

5.2　フィルタリングモジュールの処理時間評価

本節では提案手法でプロトタイプシステムに新たに追加されたフィルタリングモジュールの処理時間評価を行う。3 種類のフィルタリングモジュールは全て同じ実装であるため、評価実験はドメイン名ホワイトリストに対してのみ行った。フィルタリングモジュールに対し行なわれる操作はドメイン名または IP アドレスの追加または検索である。従ってハッシュテーブルへの追加・検索操作に対し 3.6.2 節での予備実験と同じく平均処理時間を計測する。実験では予め一定数のドメイン名が初期値として追加されたドメイン名ホワイトリストに対し、予備実験で用いた評価用データセット 10,000 件のドメイン名の追加または検索を行う。ホワイトリストの初期値は Alexa Top 1 Million Sites を用いた。

図 5.1 に各初期値におけるドメイン名ホワイトリストの処理時間を対数目盛で示し、表 5.1 に平均処理時間を示す。ハッシュテーブルにおいては追加または検索処理の計算量のオーダーは 1 であるが、初期値の増加に伴いドメイン名の追加または検索に要する時間は増加する傾向にある。また表 3.2 における各パートの平均処理時間との比較において、フィルタリングモジュールの処理時間はパケットの送受信処理と同じ 0.1 マイクロ秒の計算量となった。

5.3 提案システムのパケット転送性能評価

本節では提案システムのパケット転送性能を評価する。図 5.2 に性能計測の実験概要を示す。実験では 3 台の物理サーバ（クライアント、転送サーバ、DNS キャッシュサーバ）を使用した。クライアントは 3.6 節の予備実験における送信サーバ、DNS キャッシュサーバは受信サーバに相当する。提案システムの動作には、DNS の問い合わせ送信から DNS サーバでの名前解決、応答の送信、名前解決された IP アドレスへの通信までの一連の通信が必要である。一連の通信を再現するため、実験では、クライアントはデータセットで与えられたドメイン名で指定された通信対象サーバに対し Telnet 接続を行う。クライアントが Telnet において宛先にドメイン名を指定した場合、パケットは DNS サーバに転送され、名前解決後、クライアントは対応する IP アドレスを受信し、再度その IP アドレスへ Telnet 接続を行う。クライアントが問い合わせるドメイン名のリストは予備実験で用いた評価用データセットと同一である。Telnet 接続には Python の telnetlib モジュールを用いた。DNS キャッシュサーバは問合わせを受けたドメイン名の名前解決を行う。転送サーバには提案システムが組み込まれている。本実験を行うにあたっての前提条件として、実験環境において悪性ドメイン名はまだ観測されていないと仮定した。また提案システムではドメイン名ホワイトリストで良性と分類された場合機械学習による分類処理は行われないため、実際の運用では既知の良性ドメイン名を予めドメイン名ホワイトリストの初期値に設定すると想定される。このため、本実験での 3 種類のフィルタリング手法の初期値はそれぞれドメイン名ホワイトリスト：10,000、ドメイン名ブラックリスト：0、IP アドレスブラックリスト：0 とした。ドメイン名ホワイトリストの初期値として、OpenDNS Random Sample List [41] を用いた。このリストはクライアントが問い合わせるドメイン名のリストと 9 件の重複を持つ。実験では評価用データセット 10,000 件の数だけクライアントが Telnet 接続を行い、提案システムでの各ステージで分類されたドメイン名の数、各ステージにおけるパケット処理時間、各 DNS クエリの名前解決に要した時間を計測した。

表 5.2 に各ステージで分類されたドメイン名の数を示す。「分類器へ転送」行は分類パートを通過したドメイン名の数と予測結果を示す。「全ステージを通過」行はステージ 1 で悪性と分類されたが最初の Telnet パケットが通信対象サーバに転送される前に分類が完了しなかった Telnet 通信の数を示す。「取得漏れ」行は問い合わせに対する応答を受信する前に次の問い合わせを受信したために計測結果が取得できなかった通信の数を示す。「防止率」行は提案システムが悪性と分類したドメイン名の中で、最初の Telnet パケットが通信対象サーバに転送される前に分類が完了し名前解決を防止できた割合であり、式 5.1 で定義する。

$$（防止率）= 1 - \frac{（全ステージを通過）+（取得漏れ）}{（分類器で悪性と分類）} \tag{5.1}$$

実験結果は分類パートが OpenCV と Tensorflow の場合では 100 %、LIBSVM の場合においては 99.98 % の悪性クエリが最初の Telnet パケットが通信対象サーバに到達する前に

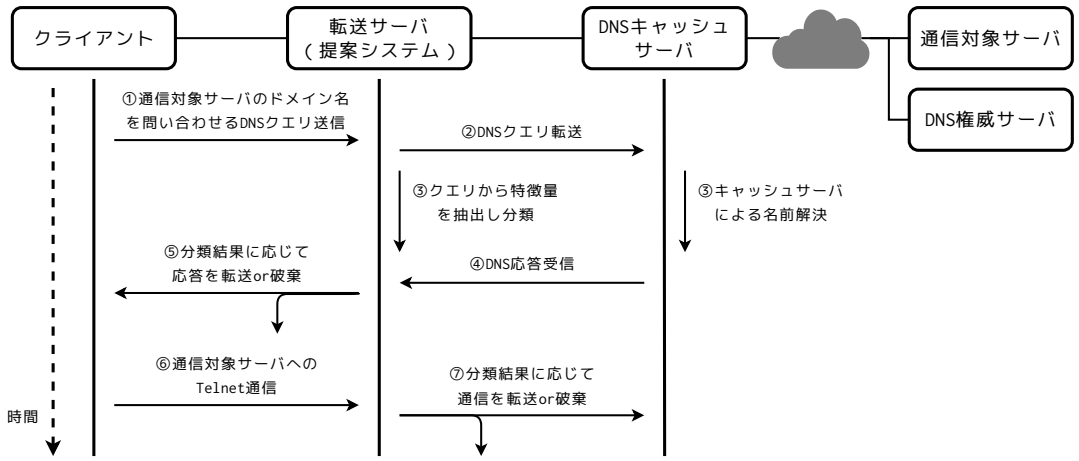


図 5.2. 性能計測の実験概要

表 5.2. 各ステージで分類されたドメイン名の数（評価用データセット）（ローカルサーバ）

ステージ	判定		OpenCV	LIBSVM	Tensorflow
	良性		9	9	9
1	分類器へ転送	良性	9,991 7,951	9,991 5,517	9,991 5,519
		悪性	2,040	4,474	4,472
	悪性		0	0	0
2	良性		7,960	5,526	5,528
	悪性		2,040	4,473	4,472
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	1	0
	取得漏れ		0	0	0
	防止率		1.0000	0.9998	1.0000

フィルタリングモジュールにより遮断されたことを示している。

図 5.3 から図 5.5 に各ステージにおける受信パートから送信パートまでのパケット転送時間を対数目盛で示す。図 5.3 は分類パートが OpenCV (Random Forest) 、図 5.4 と図 5.5 は分類パートがそれぞれ LIBSVM (SVM) と Tensorflow (Neural Network) の場合である。また表 5.3 と表 5.4 に各ステージにおける平均パケット転送時間を示す。ステージ 1 では、3.6 節での予備実験で計測したパケット転送時間と比較して、OpenCV の場合で約 17.5 マイクロ秒、LIBSVM の場合で約 0.16 秒、Tensorflow の場合で約 30.3 マイクロ秒の転送時間が削減されている。一方、ステージ 2 とステージ 3 では、分類モジュールが Tensorflow の場合において、予備実験と比較してそれぞれ約 16.5 マイクロ秒、約 0.4 マイクロ秒の処理時間が追加されている。また OpenCV と LIBSVM の場合においても同様にパケット転送時間が増加している。これは提案システムではフィルタリングモジュールによる処理が、それぞれステージ

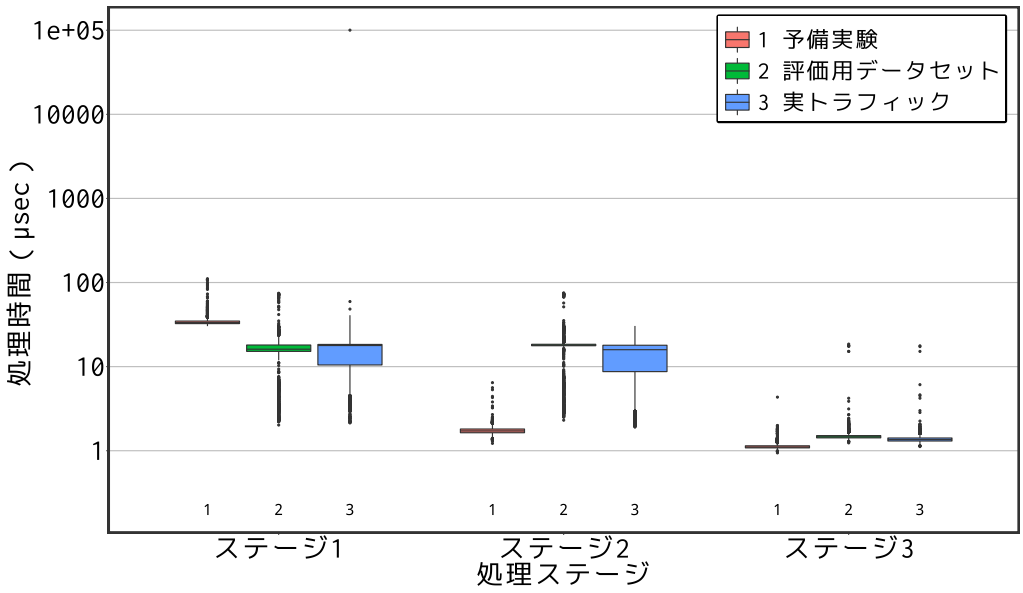


図 5.3. 各ステージにおけるパケット転送時間 (OpenCV)

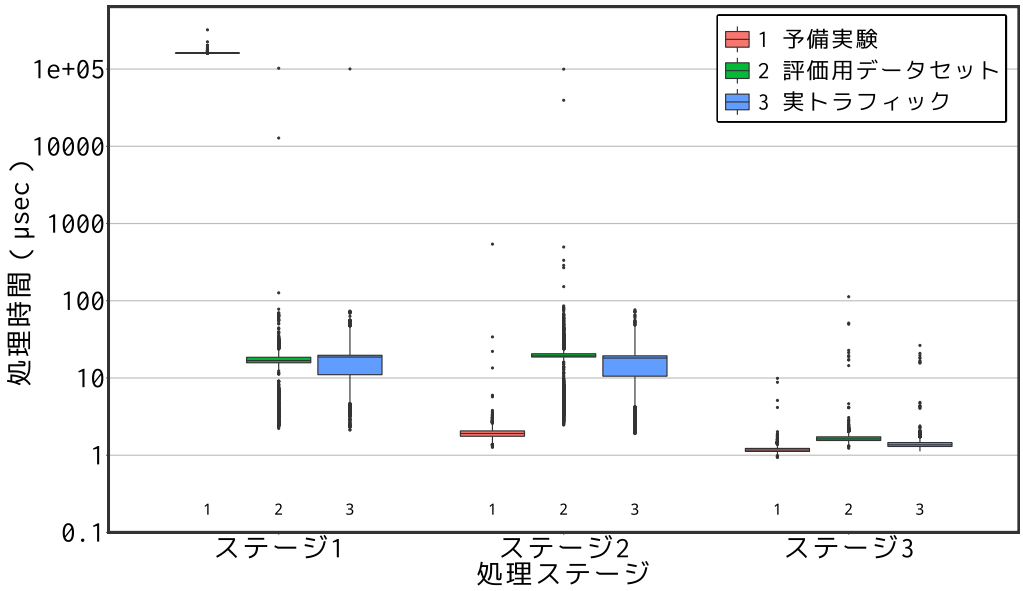


図 5.4. 各ステージにおけるパケット転送時間 (LIBSVM)

2 とステージ 3 に相当する DNS 応答パケットと最初の TCP パケットに対して追加されたためである。

図 5.6 の「評価用データ（ローカル）」の線に各 DNS クエリの名前解決に要した時間の累積分布を示す。線が縦に大きく増加する範囲が名前解決時間が集中する範囲である。全名前解決の 75% 以上が 0.01 秒から 1.0 秒の範囲に含まれている。計測結果より、提案システムが活用できる機械学習分類のための追加の処理時間が 0.01 秒から 1.0 秒であることを確認した。

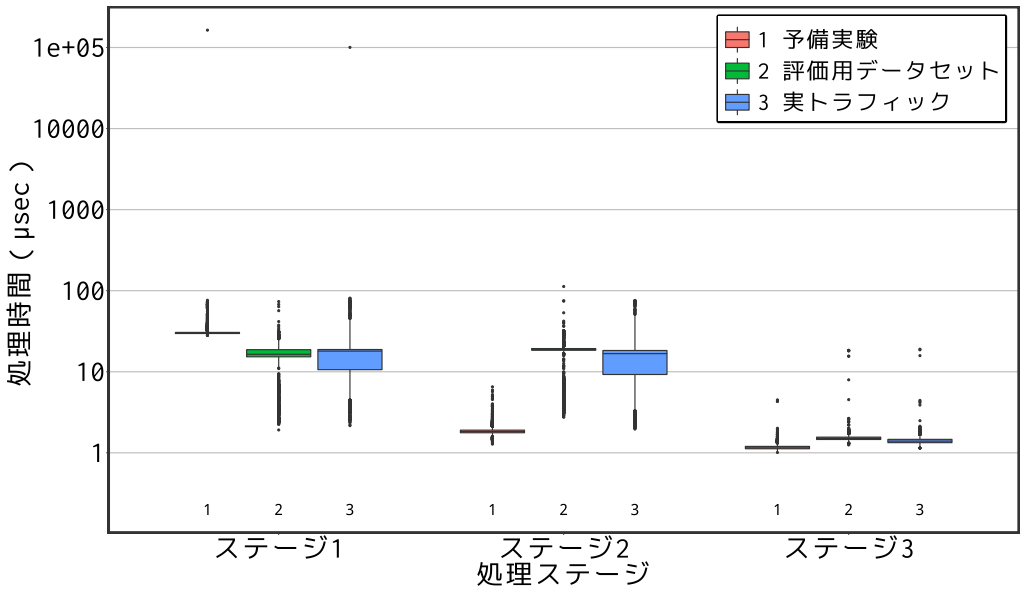


図 5.5. 各ステージにおけるパケット転送時間 (Tensorflow)

表 5.3. 各ステージにおける平均パケット転送時間 (OpenCV, LIBSVM) （マイクロ秒）

	OpenCV (Random Forest)			**LIBSVM (SVM)**		
	ステージ **1**	ステージ **2**	ステージ **3**	ステージ **1**	ステージ **2**	ステージ **3**
予備実験	33.888	1.731	1.133	161207.009	2.005	1.189
評価用データセット	16.361	17.731	1.505	28.489	33.616	1.718
実トラフィック	24.763	13.909	1.377	26.978	15.795	1.406

表 5.4. 各ステージにおける平均パケット転送時間 (Tensorflow) （マイクロ秒）

	Tensorflow（Neural Network）		
	ステージ **1**	ステージ **2**	ステージ **3**
予備実験	46.976	1.860	1.180
評価用データセット	16.643	18.384	1.545
実トラフィック	25.693	14.584	1.414

5.4　実トラフィックデータでのパケット転送性能評価

本節では、クライアントが Telnet 接続を行う通信対象サーバのリストとして、実際のネットワークで収集された DNS クエリを用いたパケット転送性能計測を行った。データセットとして日本のある学術ネットワークで集められた 24 時間分の DNS クエリを利用した。データの詳細を表 5.5 に示す。本実験を行うにあたっての前提条件として、5.3 節と同様に実験環境において悪性ドメイン名は観測されていないと仮定した。このために 3 種類のフィルタリング手法の初期値はそれぞれドメイン名ホワイトリスト：1,000,000、ドメイン名ブラックリス

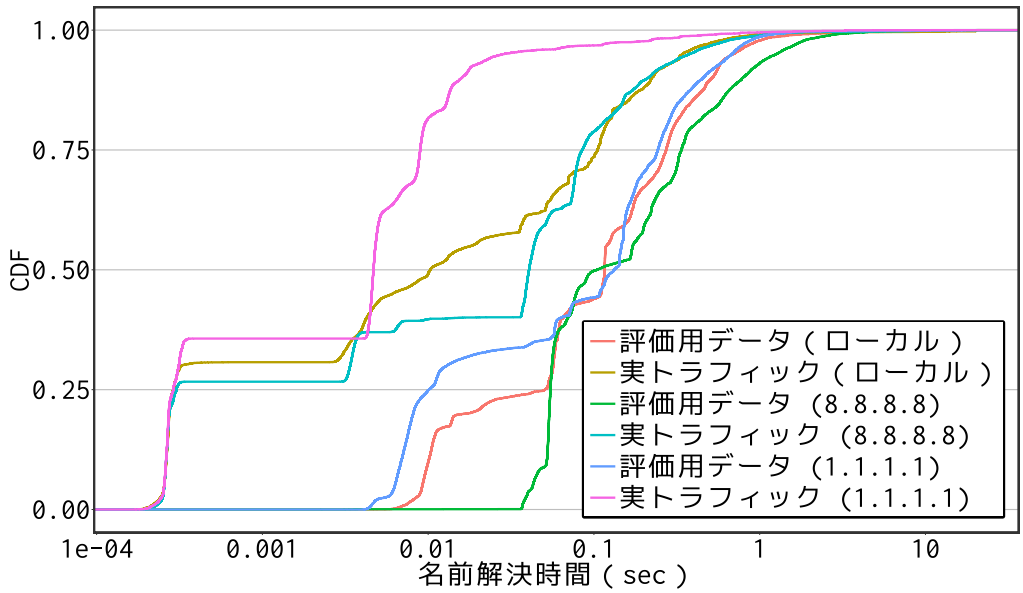


図 5.6. 各 DNS クエリデータの名前解決に要した時間

表 5.5. 実トラフィック DNS クエリデータの詳細

期間	2018-1108-06:28:31 - 2018-1109-06:28:22
総計	28165972
ドメイン名の種類	786611

ト：0、IP アドレスブラックリスト：0 とした。本実験ではドメイン名ホワイトリストの初期値を Alexa Top 1 Million Sites を用いて作成した。実験ではデータセットから 10,000 件のドメイン名を無作為に抽出し、クライアントが Telnet 接続を行う通信対象サーバのリストとした。このリストはドメイン名ホワイトリストの初期値と 5,891 件の重複を持つ。計測した項目は 5.3 節における実験と同じく提案システムでの各ステージで分類されたドメイン名の数、各ステージにおけるパケット処理時間、各 DNS クエリの名前解決に要した時間である。

表 5.6 に各ステージで分類されたドメイン名の数を示す。分類モジュールが OpenCV の場合においては、分類モジュールで悪性と分類されたクエリが 0 件であったため、防止率は空欄とした。実験結果は分類パートが Tensorflow の場合では 96.32 %、LIBSVM の場合においては 97.59 % の悪性クエリが最初の Telnet パケットが通信対象サーバに到達する前にフィルタリングモジュールにより遮断されたことを示している。

図 5.3 から図 5.5 に各ステージにおける受信から送信までのパケット処理時間、表 5.3 と表 5.4 に各ステージにおける平均パケット転送時間を示す。評価用データセットと実トラフィックとで大きな処理の時間の差がないことを示している。

図 5.3 から図 5.5 に各ステージにおける受信パートから送信パートまでのパケット転送時間を対数目盛で示す。図 5.3 は分類パートが OpenCV (Random Forest) 、図 5.4 と図 5.5 は分類パートがそれぞれ LIBSVM (SVM) と Tensorflow (Neural Network) の場合である。また

表 5.6. 各ステージで分類されたドメイン名の数（実トラフィック）（ローカルサーバ）

ステージ	判定		OpenCV	LIBSVM	Tensorflow
1	良性		7,047	7,040	7,034
	分類器へ転送	良性	2,952	2,560	2,283
		悪性	0	163	290
	悪性		0	236	392
2	良性		9,999	9,600	9,317
	悪性		0	393	675
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	0	0
	取得漏れ		0	6	7
	防止率		-	0.9632	0.9759

ステージ 1 では、3.6 節での予備実験で計測した元の処理時間と比較して Tensorflow の場合で約 40 マイクロ秒の合計時間が削減されている。一方、ステージ 2 とステージ 3 では、予備実験と比較してそれぞれ約 16.5 マイクロ秒、約 0.4 マイクロ秒の処理時間が追加されている。これは提案システムではフィルタリングモジュールによる処理が DNS 応答パケットと最初の TCP パケットに対して追加されたためである。

図 5.6 の「実トラフィック（ローカル）」の線に各 DNS クエリに対する名前解決時間を累積分布で示す。ネットワークを流れる実際のトラフィックでは、多くの人がアクセスする Web ページのドメイン名等、同じドメイン名の問い合わせが観測される。重複する問合わせは DNS キャッシュサーバのキャッシュにヒットするため、再帰的な問合わせを行う場合と比較して名前解決に要する時間が短くなる。このために、約 30 % の名前解決が 0.001 秒以内に完了している。また本システムが機械学習を用いた分類に利用可能な時間は評価用データセットの場合よりも平均で 53.7 % 減少している。

5.5　異なる DNS キャッシュサーバを用いたパケット転送性能評価

5.3 節と 5.4 節で計測した、DNS クエリの名前解決に要した時間は DNS キャッシュサーバのアドレスにより変化する。また Google が提供する Google Public DNS [42] 等のパブリック DNS サーバに対し C&C サーバのアドレスの名前解決を試みるマルウェアも報告されている [43]。DNS キャッシュサーバアドレスの違いによる提案システム動作の変化を評価するため、再帰的な問い合わせを行う DNS キャッシュサーバを、評価のために用意したクライアントと同じセグメントの DNS キャッシュサーバから、パブリック DNS サーバに変更して 5.3 節の実験と同様の計測を行った。DNS サーバの変更により提案システムの処理は変化しないため、計測項目は提案システムでの各ステージで分類されたドメイン名の数、各 DNS

クエリの名前解決に要した時間とした。パブリック DNS サーバは Google Public DNS と 1.1.1.1 [44] の 2 種類を用いた。2019 年 1 月現在 Google Public DNS の IP アドレスには 8.8.8.8 と 8.8.4.4 の 2 種類があり、実験では 8.8.8.8 を指定した。

表 5.7 から表 5.10 に各ステージで分類されたドメイン名の数を示す。表 5.7 が送信データが評価用データセットかつ DNS キャッシュサーバが Google Public DNS の場合、表 5.8 が送信データが実トラフィックかつ DNS キャッシュサーバが Google Public DNS の場合、表 5.9 が送信データが評価用データセットかつ DNS キャッシュサーバが 1.1.1.1 の場合、表 5.10 が送信データが実トラフィックかつ DNS キャッシュサーバが 1.1.1.1 の場合である。送信データが実トラフィックかつ分類モジュールが OpenCV の場合においては、分類モジュールで悪性と分類されたクエリが 0 件であったため、防止率は空欄とした。防止率が 1 未満となったのは送信データが評価用データセットかつ分類モジュールが LIBSVM の場合のみであり、その他の場合においては 100 % の悪性クエリが最初の Telnet パケットが通信対象サーバに到達する前にフィルタリングモジュールにより遮断されたことを示している。

図 5.6 の「評価用データ (8.8.8.8)」「実トラフィック (8.8.8.8)」「評価用データ (1.1.1.1)」「実トラフィック (1.1.1.1)」の線に各 DNS クエリの名前解決に要した時間の累積分布を示す。「評価用データ (8.8.8.8)」は送信データが評価用データかつ DNS キャッシュサーバが Google Public DNS の場合、「実トラフィック (8.8.8.8)」は送信データが実トラフィックかつ DNS キャッシュサーバが Google Public DNS の場合、「評価用データ (1.1.1.1)」は送信データが評価用データかつ DNS キャッシュサーバが 1.1.1.1 の場合、「実トラフィック (1.1.1.1)」は送信データが実トラフィックかつ DNS キャッシュサーバが 1.1.1.1 の場合である。計測結果は、DNS キャッシュサーバが Google Public DNS と 1.1.1.1 のどちらの場合においても、送信データの違いによる名前解決時間の比較ではローカルの DNS キャッシュサーバと同じく、実トラフィックの方が処理時間が短い傾向であることを示した。また DNS キャッシュサーバの違いによる名前解決時間の比較では、Google Public DNS が最も小さく、ローカルの DNS キャッシュサーバ、1.1.1.1 の順に大きくなる傾向が示された。

5.6　フィルタリングモジュールの時間推移

本節では提案システムの実ネットワーク下での運用性能を評価する。送信データとして 5.4 節で用いた実トラフィック DNS クエリのログデータからタイムスタンプを再現したパケットデータを生成する。実験ではクライアントが tcpreplay コマンドを用いてパケットデータを転送サーバに送信する。使用した tcpreplay コマンドのバージョンは 3.4.4 である。転送サーバでは 10 秒ごとに各フィルタリングモジュールのデータ数・分類モジュールに入力されたクエリ数を記録する。

図 5.7 (a) に累積クエリ数推移、図 5.7 (b) に分類モジュールに入力されたクエリ数推移、図 5.7 (c) にドメイン名ホワイトリストのデータ数推移、図 5.7 (d) に 2 種類のブラックリストのデータ数推移を示す。各図の横軸は 11 月 8 日 06:28:31 から 11 月 9 日 06:28:22 までの時刻である。累積クエリ数は 14 時まで緩やかに増加後、17 時から 23 時にかけて急激に増加

表 5.7. 各ステージで分類されたドメイン名の数（評価用データセット） (8.8.8.8)

ステージ	判定		OpenCV	LIBSVM	Tensorflow
1	良性		9	9	9
	分類器へ転送	良性	7,951	5,517	5,519
		悪性	2,040	4,474	4,472
			9,991	9,991	9,991
	悪性		0	0	0
2	良性		7,960	5,526	5,528
	悪性		2,040	4,472	4,472
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	1	0
	取得漏れ		0	1	0
	防止率		1.0000	0.9996	1.0000

表 5.8. 各ステージで分類されたドメイン名の数（実トラフィック） (8.8.8.8)

ステージ	判定		OpenCV	LIBSVM	Tensorflow
1	良性		7,047	7,041	7,034
	分類器へ転送	良性	2,952	2,559	2,283
		悪性	0	163	290
			2,952	2,722	2,573
	悪性		0	236	392
2	良性		9,999	9,600	9,317
	悪性		0	399	682
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	0	0
	取得漏れ		0	0	0
	防止率		-	1.0000	1.0000

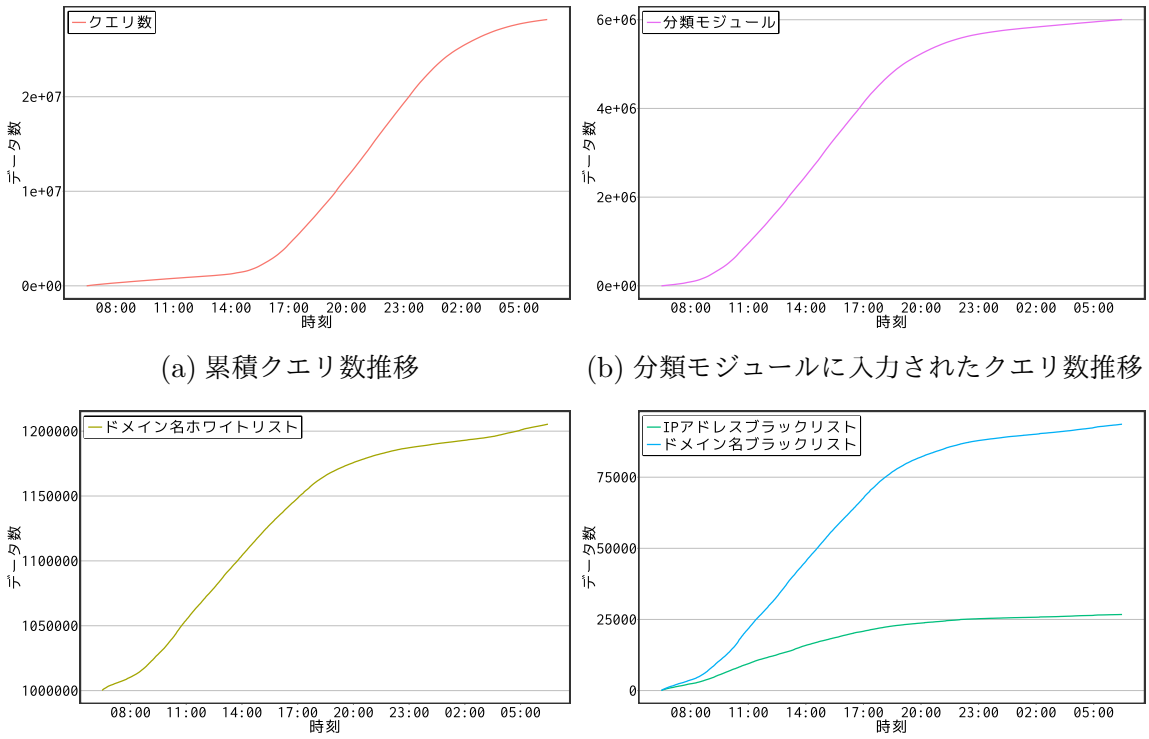
している。一方分類モジュールに入力されたクエリ数、各フィルタリングモジュールのデータ数は 11 時から 17 時にかけて最も急激に増加している。

表 5.9. 各ステージで分類されたドメイン名の数（評価用データセット） (1.1.1.1)

ステージ	判定		**OpenCV**	**LIBSVM**	**Tensorflow**
1	良性		9	9	9
	分類器へ転送	良性	7,951	5,517	5,519
		悪性	2,040	4,474	4,472
			9,991	9,991	9,991
	悪性		0	0	0
2	良性		7,960	5,526	5,528
	悪性		2,040	4,472	4,472
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	2	0
	取得漏れ		0	0	0
	防止率		1.0000	0.9996	1.0000

表 5.10. 各ステージで分類されたドメイン名の数（実トラフィック） (1.1.1.1)

ステージ	判定		**OpenCV**	**LIBSVM**	**Tensorflow**
1	良性		7,047	7,041	7,034
	分類器へ転送	良性	2,952	2,559	2,283
		悪性	0	163	290
			2,952	2,722	2,573
	悪性		0	0	0
2	良性		9,999	9,600	9,317
	悪性		0	399	682
3	良性		0	0	0
	悪性		0	0	0
	全ステージを通過		0	0	0
	取得漏れ		0	0	0
	防止率		-	1.0000	1.0000



(a) 累積クエリ数推移　　(b) 分類モジュールに入力されたクエリ数推移

(c) ホワイトリストのデータ数推移　　(d) ブラックリストのデータ数推移

図 5.7. フィルタリングモジュールのデータ数推移

第 6 章

議論

本章では 5 章での評価結果を踏まえ提案システムについて議論する。3.1 節で定義した 2 種類の要件について、6.1 節と 6.2 節でそれぞれ議論する。

6.1 リアルタイム動作性

6.1.1 攻撃の未然防止

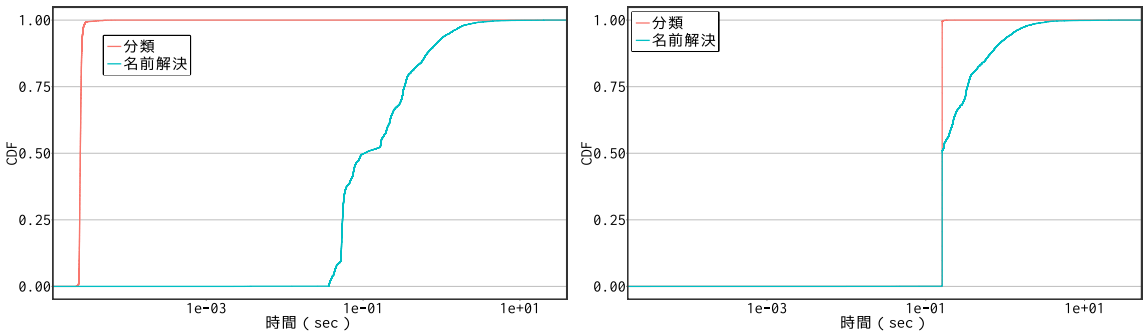
5 章での性能評価実験における防止率を取り出し表 6.1 に示す。提案システムは最も防止率が低い場合においても、悪性と判断したドメイン名の約 96 % を名前解決の完了前に検知した。原因を考察するため、図 6.1 に DNS キャッシュサーバが Google Public DNS、送信データが評価用データセットの場合における分類処理時間と名前解決時間の比較を累積分布で示す。図 6.1 (a) は分類モジュールが Tensorflow の場合、図 6.1 (b) は LIBSVM の場合である。それぞれの防止率は 1.0000、0.9996 である。Tensorflow においては分類処理時間の曲線は名前解決時間の曲線から離れ時間が小さい方向にあるため、全ての分類処理が名前解決の間に完了したことがわかる。一方 LIBSVM においては分類処理時間の曲線が名前解決時間の曲線と一部重なっているため、名前解決の間に完了しない分類処理が存在し得ることが示される。

6.1.2 パケット転送処理時間

3.3 節で提案したアーキテクチャは、パケット受信から送信までの間に、処理時間を要する機械学習分類が行われるために転送全体の処理時間は大きく増大する。一方提案システムでは、全てのパケットは受信から送信までの間に、機械学習分類と比較して処理時間の短いフィルタリングモジュールのみを通過する。従って DNS クエリ転送時におけるパケット転送時間の削減が可能である。具体的には、パケット転送性能評価実験において、表 5.3 と表 5.4 に示すステージ 1 での平均パケット転送時間を予備実験と評価用データセットとで比較すると、分類モジュールが OpenCV、LIBSVM、Tensorflow の場合においてそれぞれ 51.72 %、99.98 %、64.57 % の転送時間が削減された。削減割合は多くの処理時間を要する分類モジュールほど大

表 6.1. 性能評価実験における提案システムの防止率

DNS キャッシュサーバ	送信データ	OpenCV	LIBSVM	Tensorflow
ローカル	評価用データセット	1.0000	0.9998	1.0000
	実トラフィック	-	0.9632	0.9759
8.8.8.8	評価用データセット	1.0000	0.9996	1.0000
	実トラフィック	-	1.0000	1.0000
1.1.1.1	評価用データセット	1.0000	0.9996	1.0000
	実トラフィック	-	1.0000	1.0000



(a)Tensorflow　(b)LIBSVM

図 6.1. 分類処理時間と名前解決時間の比較

きくなることがわかる。またステージ 1 でのパケット転送時間を、式 3.1 を用いてスループット概算値に変換すると、分類モジュールが OpenCV、LIBSVM、Tensorflow の場合においてそれぞれ 43.74 Mbps、25.12 Mbps、43.00 Mbps となる。

一方、提案システムにおいてステージ 2 とステージ 3 で処理されるパケットは、図 3.1 に示すアーキテクチャでの処理は想定されておらず、提案システムで新たに追加される処理である。従ってステージ 2 とステージ 3 で増加するパケット転送処理時間は提案システムの欠点となる。具体的には、パケット転送性能評価実験において、表 5.3 と表 5.4 に示すステージ 2 での平均パケット転送時間を予備実験と評価用データセットとで比較すると、分類モジュールが OpenCV、LIBSVM、Tensorflow の場合においてそれぞれ 924.32 %、1576.61 %、888.39 % だけ転送時間が増加した。またステージ 3 ではそれぞれ 32.83 %、44.49 %、30.93 % だけ転送時間が増加した。特にステージ 2 においてパケット転送時間が大きく増加した原因を考察する。ステージ 2 でパケット受信から送信までに行なわれる追加処理はフィルタリングモジュールの検索とパーサパート内での 5.1 節に示すドメイン名の変換処理である。5.2 節におけるフィルタリングモジュールの処理時間評価の結果、フィルタリングモジュールの検索に要する時間は 0.1 マイクロ秒の計算量であるため、ドメイン名の変換処理がパケット転送時間増加に大きく寄与していると考えられる。ドメイン名の変換処理高速化の方法として、SLD リストデータ構造のハッシュテーブル等検索速度のより速いものへの変更等が考えられる、

6.2　分類機能のモジュール化とモジュール追加・削除の柔軟性

本システムはソフトウェアでの実装を行い、機械学習による分類処理をモジュール化しパケット送受信処理から独立させることで、3 種類の異なる機械学習ライブラリ・フレームワークを用いたパケット分類を可能にした。また分類モジュールを入れ替えてのシステムのビルドは数分で完了するため、分類モジュールの追加・削除は容易であると言える。一方、本システムが同時に動作可能な分類モジュールは 1 種類のみである。しかしながら、3.1 節で述べた通り、ネットワーク上の様々な攻撃を検知するためには、同時に多数の分類モジュールを動作させる必要がある。多数の分類モジュールを同時に動作させるためには並列化等の手法が考えられる。

第 7 章

結論

本章では, 本論文のまとめと今後の課題を示す.

7.1 まとめ

本研究では、サイバー攻撃によるセキュリティインシデントを未然に防ぐ 1 つの方法として、機械学習を用いたパケット分類処理アーキテクチャの提案とシステムの実装を行い、予備実験によってアーキテクチャのパケット転送性能と分類処理時間確保の必要性を明らかにした。実験結果に基づき、DNS の名前解決時間を活用することで、機械学習を用いて悪性 DNS クエリを検知し名前解決を未然に防ぐリアルタイム悪性 DNS クエリ分類システムを提案した。提案システムのパケット転送性能評価を行い、 3 種類の機械学習ライブラリ・フレームワークにおいて、処理遅延を抑えつつ DGA により生成された悪性 DNS クエリの名前解決の未然防止が可能であることを示した。

7.2 今後の課題

本節では本研究における今後の課題を 3 つ述べる。

1 つ目はパケット外データの活用である。本研究では機械学習分類に用いる特徴量をパケット内データのみから抽出したが、機械学習を利用した既存の攻撃検知技術ではパケット到着間隔の周期性、問い合わせドメイン名が登録された日時等、パケットからは取得できない外部データを特徴量として用いる場合が多い。従って既存の攻撃検知技術を提案システムに組み込むために、外部データ取得に要する処理時間を抑えつつ外部データを取得する方法を検討する必要がある。

2 つ目は DGA 型ボットネット以外の攻撃検知における提案システムの動作検証である。本研究の評価では検知対象として DGA により生成された悪性 DNS クエリのみを設定したが、提案システムは適切に学習された検出器を分類モジュールとして使用することで、DNS トンネリング等 DNS を悪用した他の攻撃も検知可能である。従って異なる検知対象においても提案システムが本論文の実験結果と同様のパケット転送を示すかを検証する必要がある。

最後に、提案システムのパケット転送性能の向上である。提案システムの DNS クエリ転送性能は 10 Mbps の計算量であり、1 - 10 Gbps の回線速度が一般的である現状を考慮すると転送性能のさらなる向上が求められる。転送性能の向上には各処理を並列に実行するパイプライン処理や送受信処理を異なる CPU コアに割り当てるピンニング等既存のソフトウェアでの高速パケット処理技術が活用可能であると考えられる。

発表文献

(1) 須賀灯希, 岡田和也, 江崎浩, “機械学習による悪性トラフィック遮断のための高速パケット分類機構の検討”, ネットワークソフトウェア (NWS) 研究会, 2018 年 6 月.

(2) Toki Suga, Kazuya Okada, Hiroshi Esaki, “Toward Real-time Packet Classification for Preventing Malicious Traffic by Machine Learning”, AIMLEM 2019, 1st International Workshop on Artificial Intelligence and Machine Learning Techniques for Enhanced Network Management, 2019 年 2 月予定.

参考文献

[1] Cisco 2018 annual cybersecurity report. https://www.cisco.com/c/dam/m/digital/elq-cmcglobal/witb/acr2018/acr2018final.pdf. (Accessed on 01/23/2019).

[2] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 833–844. ACM, 2012.

[3] Next-generation firewall - palo alto networks. https://www.paloaltonetworks.com/products/secure-the-network/next-generation-firewall.html. (Accessed on 01/30/2019).

[4] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, Vol. 50, No. 7, pp. 80–84, 2017.

[5] Ihsan Ullah, Naveed Khan, and Hatim A Aboalsamh. Survey on botnet: Its architecture, detection, prevention and mitigation. In *Networking, Sensing and Control (ICNSC), 2013 10th IEEE International Conference on*, pp. 660–665. IEEE, 2013.

[6] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 635–647. ACM, 2009.

[7] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. In *Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on*, pp. 116–123. IEEE, 2013.

[8] Seungwon Shin, Guofei Gu, Narasimha Reddy, and Christopher P Lee. A large-scale empirical study of conficker. *IEEE Transactions on Information Forensics and Security*, Vol. 7, No. 2, pp. 676–690, 2012.

[9] Paul Royal. Analysis of the kraken botnet. *Damballa, Apr*, Vol. 9, , 2008.

[10] Brett Stone-Gross, Marco Cova, Bob Gilbert, Richard Kemmerer, Christopher

Kruegel, and Giovanni Vigna. Analysis of a botnet takeover. *IEEE Security & Privacy*, Vol. 9, No. 1, pp. 64–72, 2011.

[11] D Schwarz. Bedep' s dga: Trading foreign exchange for malware domains, 2016.

[12] Microsoft ie does not properly display some urls - securitytracker. `https://securitytracker.com/id/1008425`. (Accessed on 01/06/2019).

[13] Sectheory - internet security. `http://www.sectheory.com/clickjacking.htm`. (Accessed on 01/06/2019).

[14] Gunter Ollmann. The phishing guide–understanding & preventing phishing attacks. *NGS Software Insight Security Research*, 2004.

[15] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, pp. 281–290. ACM, 2010.

[16] Style sheet vulnerability allowed attacker to hijack linkedin pages. `https://www.scmagazineuk.com/style-sheet-vulnerability-allowed-attacker-hijack-linkedin-pages/article/1479529`. (Accessed on 01/06/2019).

[17] Luca Deri, Lorenzo Luconi Trombacchi, Maurizio Martinelli, and Daniele Vannozzi. A distributed dns traffic monitoring system. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pp. 30–35. IEEE, 2012.

[18] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *USENIX security symposium*, Vol. 12, 2012.

[19] Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. Phoenix: Dga-based botnet tracking and intelligence. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 192–211. Springer, 2014.

[20] Han Zhang, Manaf Gharaibeh, Spiros Thanasoulas, and Christos Papadopoulos. Botdigger: Detecting dga bots in a single network. In *Proceedings of the IEEE International Workshop on Traffic Monitoring and Analaysis*, 2016.

[21] Mouad Zouina and Benaceur Outtaj. A novel lightweight url phishing detection system using svm and similarity index. *Human-centric Computing and Information Sciences*, Vol. 7, No. 1, p. 17, 2017.

[22] Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, and Banu Diri. Machine learning based phishing detection from urls. *Expert Systems with Applications*, Vol. 117, pp. 345–357, 2019.

[23] Haijun Zhang, Gang Liu, Tommy WS Chow, and Wenyin Liu. Textual and visual content-based anti-phishing: a bayesian approach. *IEEE Transactions on Neural*

Networks, Vol. 22, No. 10, pp. 1532–1546, 2011.

[24] Xi Luo, Liming Wang, Zhen Xu, Jing Yang, Mo Sun, and Jing Wang. DGASensor: Fast Detection for DGA-Based Malwares. In *Proceedings of the 5th International Conference on Communications and Broadband Networking*, pp. 47–53. ACM, 2017.

[25] Bin Yu, Daniel L Gray, Jie Pan, Martine De Cock, and Anderson CA Nascimento. Inline dga detection with deep networks. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, pp. 683–692. IEEE, 2017.

[26] Ankit Kumar Jain and B Brij Gupta. Towards detection of phishing websites on client-side using machine learning based approach. *Telecommunication Systems*, Vol. 68, No. 4, pp. 687–700, 2018.

[27] Khaled Alrawashdeh and Carla Purdy. Reducing calculation requirements in fpga implementation of deep learning algorithms for online anomaly intrusion detection. In *Aerospace and Electronics Conference (NAECON), 2017 IEEE National*, pp. 57–62. IEEE, 2017.

[28] Intel Data Plane Development Kit. `https://www.dpdk.org/`, 2018.

[29] Luigi Rizzo. Netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pp. 101–112, 2012.

[30] Yoshihiro Nakajima, Tomoya Hibi, Hirokazu Takahashi, Hitoshi Masutani, Katsuhiro Shimano, and Masaki Fukui. Scalable high-performance elastic software openflow switch in userspace for wide-area network. *Proc. Open Networking Summit (ONS 2014), Santa Clara, CA*, 2014.

[31] Opencv library. `https://opencv.org/`, 2018.

[32] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, Vol. 2, No. 3, p. 27, 2011.

[33] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, Vol. 16, pp. 265–283, 2016.

[34] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[35] Keyword Research Competitive Analysis, & Website Ranking — Alexa. `https://www.alexa.com/`, 2018.

[36] DGA Netlab OpenData Project. `https://data.netlab.360.com/dga/`, 2018.

[37] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1245–1254. ACM, 2009.

[38] Scapy. `https://scapy.net/`. (Accessed on 01/13/2019).

[39] Glib reference manual - gnome 開発センター. https://developer.gnome.org/glib/. (Accessed on 01/09/2019).

[40] List of second-level domains - domains index. https://domains-index.com/downloads/list-of-second-level-domains/, 2018.

[41] opendns/public-domain-lists: Opendns public domain lists of domain names for training/testing classifiers. https://github.com/opendns/public-domain-lists, 2018.

[42] Public dns — google developers. https://developers.google.com/speed/public-dns/. (Accessed on 01/09/2019).

[43] 注目の脅威：goznym. https://gblogs.cisco.com/jp/2016/10/goznym-html/. (Accessed on 01/09/2019).

[44] 1.1.1.1 — the internet's fastest, privacy-first dns resolver. https://1.1.1.1/. (Accessed on 01/09/2019).

謝辞

研究室に配属されて以来、学士と修士の両課程での研究活動にあたり、大変多くの方にご指導、ご協力をいただきました。ここに心より感謝を申し上げます.

指導教員である東京大学情報理工学系研究科教授 江崎浩博士には、研究を進めるにあたって幅広い知識と深い経験に基づいて本質的なことを見抜き、的確なご指導を頂きました。ここに深く感謝いたします。修士 2 年間の研究活動全般にわたり数多くの助言や大変親身なご指導を頂きました東京大学情報基盤センター情報メディア教育研究部門助教 岡田和也博士に深く感謝いたします。進捗報告の場等で幅広い知識から研究姿勢や内容について的確なご指導とご助言を頂きました情報理工学系研究科准教授 落合秀也博士、情報理工学系研究科特任助教 塚田学博士、生産技術研究所電子計算機室助教 山本成一博士に深く感謝いたします。研究を進める上でさまざまな助言をいただいたり、多方面から研究室生活をサポートしてくださった研究室の先輩や OB の皆様に感謝いたします。様々な境遇を共に乗り越えてきた研究室同期の今井元太氏、大友一樹氏、北沢昌大氏、幸田大智氏、野村祐太朗氏に感謝いたします。研究室生活を共に過ごした後輩の皆様に感謝いたします。諸事務を通じて研究室の生活を支えてくださった岩井愛映子秘書、高橋富美秘書に感謝いたします。最後に、研究生活や学生生活においてさまざまなご指導とご支援をくださったすべての皆様に感謝いたします。

2019 年 1 月 31 日

須賀 灯希