平成 29 年度 修士論文

セキュリティインシデント対応に向けたトラフィック自動分類システム

Automatic Traffic Classification System for Efficient Security Incident Responses

指導教員　森 達哉 准教授

早稲田大学　基幹理工学研究科　情報理工・情報通信専攻

学籍番号 5116F083-7

水野 翔

平成 30 年 1 月 30 日

概要

我々が普段利用しているネットワークには，様々なトラフィックが流れている．ここ数年では，IoT 端末の急速な普及・発展によってトラフィック量が増加の一途を辿っており，それに伴い攻撃者による悪性通信も多く観測されるようになった．このことから，ネットワーク管理者は組織内のトラフィック情報を常時収集・分析し，いかなる事態にも対応できるように万全の体制を整えておくことが重要である．

このような背景から，本研究ではトラフィック分類に関する 2 つの手法を提案する．1 つ目は，正常通信と悪性通信を弁別する手法である．約 40,000 種類のマルウェア検体による悪性通信データとキャンパスネットワークを流れる様々な正常通信データを収集し，機械学習による自動分類システムの実装を行った．その結果，誤検知率を 1% 未満に抑えつつ，約 97% の精度で，正常通信と悪性通信の弁別が可能であることを示した．

2 つ目の手法として，パケットからそのホストの OS を推定する手法を提案する．約 400 種類の ClientHello パケットデータを収集し，TCP/IP 及び SSL/TLS フィンガープリントを用いた各 OS の多クラス分類実験を行った．この結果，約 89% の精度で各 OS の分類が可能であり，SSL/TLS フィンガープリントが OS 分類において非常に有意な特徴であることを示した．

これら 2 つの手法を組み合わせることで，セキュリティインシデントの効率的な対処が可能になることが期待できる．

目次

図目次

表目次

第 1 章　はじめに

Cisco [1] は 2016 年から 2021 年までのネットワークトラフィックの分析及び予測を行っている．2016 年に全体の 46% を占めていたスマートデバイスの割合が，2021 年にはほぼ倍となる 82% を占めるという．また，同様に 2016 年には約 7 エクサバイトだったネットワークトラフィック量が，2021 年にはその 7 倍である約 49 エクサバイトまで増加するという．これは Internet-of-Things (IoT) 端末を代表としたスマートデバイスが今後ますます普及し，トラフィックの多様化が進行することを如実に表している．これに伴い，マルウェアの攻撃対象も多様化し，オペレーティングシステムや端末が新種のマルウェアに感染する可能性が生じてくる．例えば，これまでのマルウェアの多くは Windows を攻撃対象としていたものの，MaCafee の発表 [2] では，mac OS を対象としたマルウェアの検体数が 2016 年に約 460,000 種類にまで増加したという．ESET は 2016 年に IoT 端末への攻撃が増加すると予測した [3]．実際に，世界の 105 カ国に設置された約 25,000 台のセキュリティカメラがボットに感染し，大規模な DDoS 攻撃を行うという事案も発生した [4].

これらの問題は，今日，あらゆる端末が新種のマルウェアに感染する可能性があることを示唆しており，ネットワーク管理者にとっての脅威と言える．さらに，アンチウィルスソフトを端末にインストールして利用するだけでは，必ずしも安全とは言えない．特定の OS に起因する脆弱性を突いた攻撃が行われる場合も多く，その場合はアンチウィルスソフトでも完全に防ぐことは不可能である．また，近年マルウェアによるアンチウィルスソフトの検知回避技術も進歩しており [5]，感染前に検出することは困難になりつつある．ここで重要になるのは，いかに感染前に OS の脆弱性に気付けるか，そして感染を許してしまった場合でもいかに素早く感染端末を特定できるかである．これらの問題の解決策として，本研究では 2 つの手法の提案を行う．

1 つ目の解決策として，ネットワークを流れるトラフィックから悪性通信を弁別する手法を提案する．マルウェアに感染した端末は外部の Command and Control (C&C) サーバと通信を開始するため，我々はインターネットトラフィックの監視はマルウェア感染端末を見つける上で有効なアプローチであるという仮説を立てた．トラフィック観測による手法の利点としては，多数のエンドホストを監視できる点である．もしエンドホストが悪質，あるいは疑わしい通信を行っていることが確認できた場合，そのホストはマルウェアに感染した可能性が高い．この仮説を検証するため，我々は *BotDetector* と呼ぶシステムを構築し，以下のアプローチを取る．

まず，外部の C&C サーバと通信するための手段として，多数のマルウェアが HTTP プロトコルを使用する [6] ため，端末から送信された HTTP パケットを収集し利用する．次に，機械学習を利用し，収集したトラフィックに対して悪性か良性かの二値分類を行う．ここで重要となる技術的な課題として，提案手法をスケーラブルにすることである．HTTP ヘッダに記録できる値は非常に自由度が高く，確認しただけでも約 10K の異なる特徴ベクトルを持つ．そこで，高い分類精度を達成しつつ保持する情報量を削減するため，我々は類似した特徴を統計的手法で集約するテンプレート自動生成技術を開発した．これは有用な特徴を選択する際のドメイン知識に依存しないため，マルウェアのトラフィックに対してシステムをロバストに保つことが可能である．大規模なトラフィックデータを利用した広範囲な実験を通して，*BotDetector* がマルウェアに感染した端末をスケーラブルかつ正確に検知できることを示す．

2 つ目の解決策として，我々は HTTPS パケットを利用した通信元 OS 推定手法の提案を行う．これにより，特定の OS に脆弱性が確認された場合に迅速に対処することが可能となる．また，一部の IDS/IPS やファイアウォールでは，特定の攻撃を検知する際にターゲットとされているホストの OS 情報が必要となる場合があり，常にネットワーク内の OS 情報を収集することは非常に重要である．そこで我々は，*HelloScanner* と呼ぶシステムを構築した．悪性通信検知の際は HTTP パケットを利用したが，ここでは HTTPS パケットを収集し利用する．理由として，HTTP パケットを利用する場合は User-Agent を確認することで OS の識別が可能であるが，マルウェア感染端末の通信はそれを書き換える場合があり，正確な情報が得られない．また，HTTPS 対応の Web サイトの割合が増加傾向 [7] にあり，これに対応する必要があることも挙げられる．技術的な課題として，HTTPS は暗号化されているためにエンドユーザ以外はヘッダ情報を確認することができない．そこで我々は，暗号化前に行われる SSL/TLS ハンドシェイクに着目し，その中の ClientHello パケットを用いて分類実験を行った．既存の TCP/IP フィンガープリントに加えて SSL ヘッダに記載される CipherSuite リストを特徴量に加え，機械学習による分類を行った．その結果，非常に高い精度での分類が可能であり，CipherSuite リストの貢献が大きいことがわかった．

本論文は以下のように構成される．第 2 章では *BotDetector* の概要を示す．第 3 章では *HelloScanner* に関する概要を示す．第 4 章は実際に使用したデータセットの詳細を述べる．第 5 章では本研究の実験結果を示す．第 6 章では本研究の提案手法及び制約についての議論を行う．第 7 章では関連研究と本研究との違いを示し，第 8 章では我々の提案手法の結論をまとめる．

第 2 章　BotDetector フレームワーク

このセクションでは，我々の提案するシステムである *BotDetector* に関する全体的な説明を行う．図 2.1 は *BotDetector* の概要図である．本システムの目的は，正常通信と悪性通信を弁別することでマルウェア感染端末を見つけ出すことである．*BotDetector* は以下の 3 つのステップから構成される．**Step1**: HTTP ヘッダフィールド抽出 (2.1 節)，**Step2**: テンプレート自動生成 (2.2 節)，そして **Step3**: トラフィック分類 (2.3 節) である．以下に，これらのステップに関する詳細を示す．

2.1　Step1: HTTP ヘッダフィールド抽出

先に述べたように，マルウェアの多くはインストール時や実行中に C&C サーバとの通信のために HTTP プロトコルを利用する [6]．そこで *BotDetector* は，HTTP ヘッダ内に記録されている情報を利用する．図 2.2 は HTTP ヘッダフィールドを抽出する際の過程を示す．まず初めに，HTTP ヘッダフィールドから Key とそれに対応する Value を抽出する．URI 情報は厳密には HTTP ヘッダフィールドではないが，path と query から構成される有用な情報の 1 つであり，これも同様に抽出する．URI に関しては Key を “URI” とする．

また，分析を公正に行うため，データ収集環境に依存する可能性のあるいくつかの Value は意図的に削除してから抽出を行う．ここでは日付情報や言語情報等が挙げられる．具体的には，以下の Key に対応する Value を削除する．

- Accept-Language
- Date
- Expires
- If-Modified-Since
- Last-Modified

また，全ての訓練用データから 10 回未満しか抽出できなかった HTTP ヘッダフィールドや，プロキシサーバ等のミドルボックスを通過した際に追加されるフィールドに関しても同様に削除を行う．全てのフィールドを利用しようとした場合，特徴量の数が膨大になってしまう点が考慮される．そこで，ある程度フィールドを削減した状態で精度を確認した結果，大きな影響

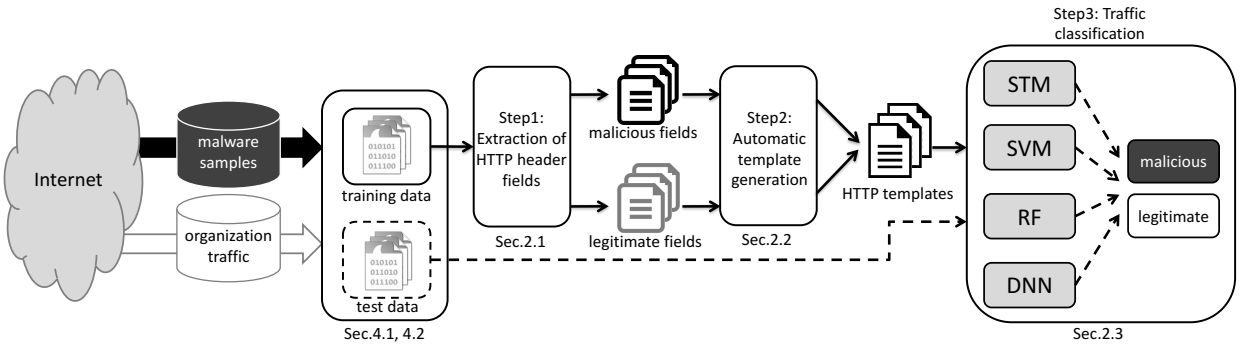

図 2.1　BotDetector の概要図.

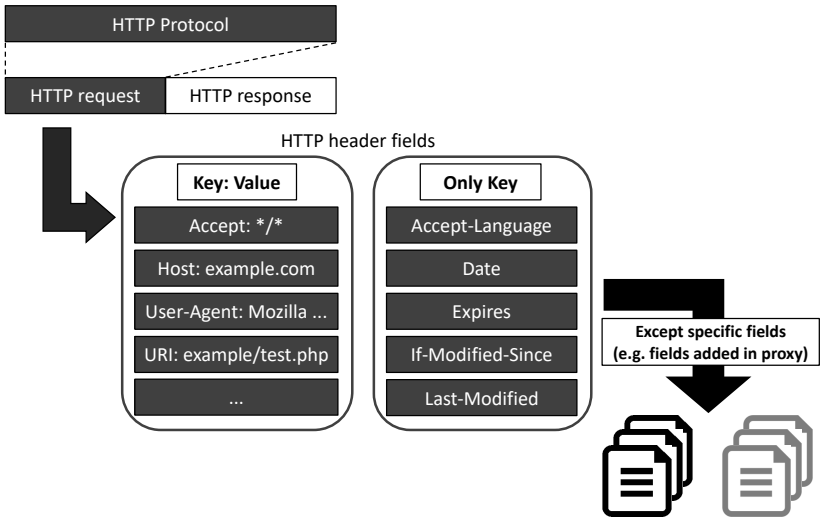

図 2.2　HTTP ヘッダフィールド抽出の概要図.

は確認できなかったため，経験的に 10 回という回数を指定し削減を行った．ミドルボックス通過時に付加される特殊なフィールドに関しても，特定の環境に強く依存してしまうため，これらのフィールドを包含するのは有用ではないと判断した.

2.2　Step2: テンプレート自動生成

データセットから抽出することができたユニークな HTTP ヘッダフィールドの個数は約 10K にも及ぶ．しかしながら，不必要な特徴量が多い場合，機械学習を用いる際に過学習を招く恐れがある．そこで，我々は HTTP ヘッダフィールドを構成する単語の可変性に着目し，その情報を圧縮するため，DBSCAN [8] に基づいたテンプレート自動生成手法 [9] を応用した.

2.2.1 スコアリング

まず初めに，スコアリングについて説明する．訓練データから抽出した全ての HTTP ヘッダフィールドを利用し，各フィールドについて，フィールド内の単語にスコアを算出する．各フィールドは，空白，“/”，“=”，そして “,” のような区切り文字によって単語に分割されていることに留意する．そこで，各単語の条件付き確率をスコアとして計算する．与えられたフィールド F 内の単語 w に対して，$S(w;F)$ という単語のスコアは次の条件付き確率の計算式によって算出される．

$$S(w;F) = P(w \mid \mathrm{pos}(w,F), \mathrm{len}(F))$$

$$= \frac{n(w, \mathrm{pos}(w,F), \mathrm{len}(F))}{n(\mathrm{pos}(w,F), \mathrm{len}(F))}$$

ここで，$\mathrm{pos}(w,F)$ は与えられたフィールド F 内の単語の位置を示し，$\mathrm{len}(F)$ はフィールド内の単語の個数を示す．もし $F = \{\mathrm{foo}, \mathrm{bar}, \mathrm{baz}, \mathrm{qux}\}$ と $w = \mathrm{bar}$ が与えられた場合，$\mathrm{pos}(w,F) = 2$，$\mathrm{len}(F) = 4$ となる．$n(X)$ はフィールド全体の変数 X の出現回数を示す．

2.2.2 DBSCAN

DBSCAN [8] は，事前にクラスタ数を指定する必要がないクラスタリングアルゴリズムの 1 つであり，任意の形状でクラスタを抽出することができる．あらかじめ最小距離と対象数の閾値をそれぞれ ϵ，m として設定する．あるデータ D と 2 つの要素 p と q が与えられたとき，以下の式に従って $N_\epsilon(p)$ を定義する．

$$N_\epsilon(p) = \{q \in D \mid d(p,q) \leq \epsilon\},$$

$d(x,y)$ は x と y の間のユークリッド距離である．$N_\epsilon(q)$ は半径 ϵ 内に m 以上のサンプルを有する．もし p と q が以下の条件を満たす場合，それらは同じクラスタに分類される．

$$p \in N_\epsilon(q),$$
$$|N_\epsilon(q)| \geq m.$$

2.2.3 クラスタリング

最後に，スコアリングと DBSCAN を組み合わせたテンプレート自動生成のアルゴリズムについて説明をする．表 2.1 と図 2.3 はテンプレート自動生成の概要である．あらかじめ閾値 δ ($\delta \geq 0$) と β ($0 < \beta < 1$) を設定する．これらの閾値の値は後に経験的に決定される．δ はクラスタ間の最小距離を決定する閾値である．クラスタリングは以下のステップを経て行われる．

表 2.1　HTTP ヘッダフィールドから生成したテンプレートの例.

original HTTP header fields	automatically generated templates
Accept: text json	Accept: text *
Accept-Encoding: gzip deflate	Accept-Encoding: gzip *
Connection: Keep-Alive	Connection: *
User-Agent: Mozilla 4.0 (MSIE 6.0; Windows 5.1)	User-Agent: Mozilla * (MSIE * Windows *

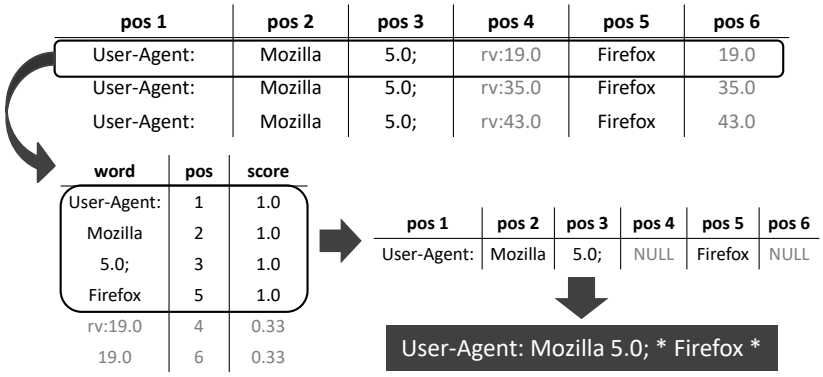

図 2.3　テンプレート生成の概要図．この例では，3 つの User-Agent フィールドが与えられており，閾値 $\delta = 10^{-1}$ と $\beta = 0.5$ が設定されている．テンプレートは各フィールドから生成されるが，重複は除かれる．

単語のソート:

先に算出したスコアに基づいて，フィールド内の単語を降順にソートする．

クラスタ生成:

DBSCAN アルゴリズムを利用し，単語をクラスタリングする．スコアの順にクラスタを形成する．クラスタ内の平均スコアと次の単語のスコアの差が δ 未満である場合，現在のクラスタへ単語を追加する．そうでない場合は次の単語を現在のクラスタとして設定する．フィールド内の全ての単語がクラスタに含まれるまでこれを繰り返す.

テンプレート出力:

クラスタ内の総単語数が $\beta \times \text{len}(F)$ 以上になるまで，クラスタを順番に結合していく．閾値を超えなかったクラスタの残りの部分に関しては，図 2.3 に示すように単語の位置に '*' を挿入する．

2.3 Step3: トラフィック分類

本ステップでは，最適な機械学習による分類器を利用して悪性通信の弁別を行う．特徴量として，前ステップで生成した HTTP テンプレートを利用した．分類器として Simple Template Matching (STM)，Support Vector Machine (SVM)，Random Forest (RF)，そして Deep Neural Network (DNN) を用意し，それぞれの性能の比較を行った．これらの分類器を選択した理由として，一般的に様々な論文で良い結果を出していることが知られているためである．また，STM は機械学習ではないが，分類精度の基準として追加することとした．STM は非常にシンプルなアルゴリズムであり，訓練用データ内の悪性通信でのみ利用されているテンプレートが評価用データの HTTP リクエストパケット内で一度でも使用された場合，そのパケットは悪性通信であると判定する．SVM，RF，DNN を実装する際にそれぞれ libsvm [10]，scikit-learn [11]，そして TensorFlow [12] を利用した．SVM と RF に関しては，最適なパラメータを設定するために，訓練用データ内で 5-fold cross-validation による grid search を行った．DNN は入力層，隠れ層 $\times 2$，出力層の合計 4 層から構築されている．また，勾配法は ADAM を使用し，誤差関数及び活性化関数はそれぞれクロスエントロピーと ReLU 関数に基づいている．学習率とバッチサイズはそれぞれ 0.0001，200 に設定し，エポック数は 9,000，各層のノード数は，入力層から順に X，500，50，2 とした．X は採用した特徴量の数に依存し，テンプレートを生成する際のパラメータによって値が異なる．より詳しくは Section 5 にて述べる．過学習を回避するため，Dropout [13] を利用した．Dropout は指定した割合でランダムにノードを無効にし，擬似的なアンサンブル学習を行うことができる．今回は最適な値であると言われている 50% [14] に設定した．

第 3 章　HelloScanner フレームワーク

本セクションでは OS フィンガープリントを特徴量とした OS 推定システムである *HelloScanner* に関する概要を示す．OS フィンガープリンティングとは，対象の通信の特徴を分析することで OS を推定する技術であり，Active 型と Passive 型の 2 種類がある．前者は対象に向けてパケットを送信し，その応答パケットを解析することで OS 推定を行う手法である．nmap [15] や Xprobe [16] が該当する．Passive 型は対象の端末が行う通信を観測し，分析をすることでそのパケットのホストの OS を推定する手法である．代表的なツールとして，SYN パケットを用いて解析を行う p0f [17] が挙げられる．*HelloScanner* も同様に Passive 型のシステムとして実装を行った．図 3.1 は本提案手法の概要図である．我々は HTTPS 通信の際に行われる SSL/TLS ハンドシェイクに着目し，その中の ClientHello パケットを利用することで OS 分類を行った．図 3.2 は SSL/TLS ハンドシェイクを示す．ClientHello パケットはクライアントがサポートしている暗号化方式等をサーバに送信するパケットである．暗号化方式は OS やブラウザによって異なり，OS フィンガープリンティングにおいて有意な情報となることが考えられる．本システムは *BotDetector* と同様に，データセットを独自に収集した．そこから特徴量を選択・抽出

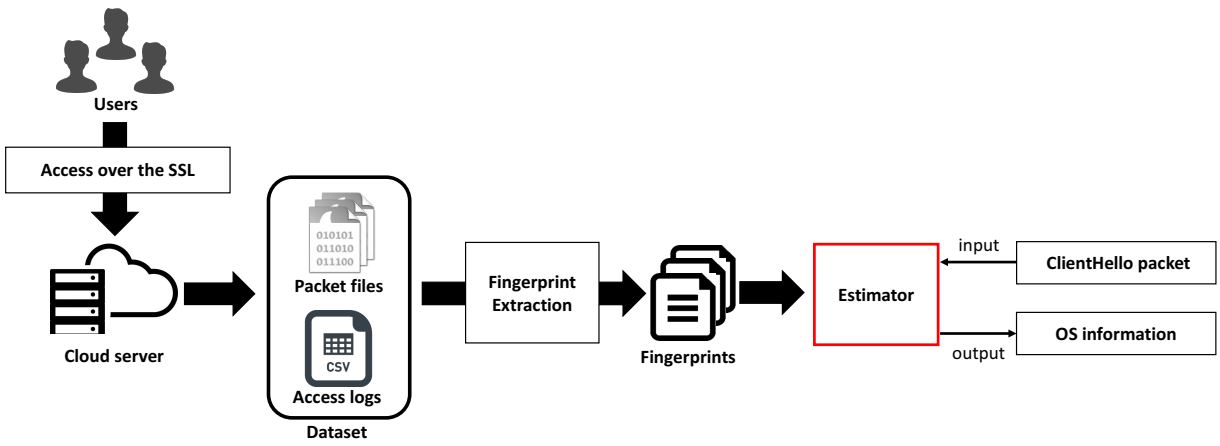

図 3.1　HelloScanner の概要図.

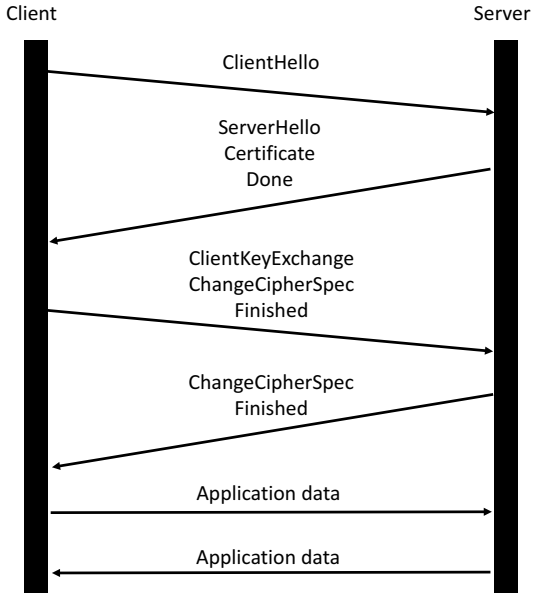

図 3.2　SSL/TLS ハンドシェイクの概要図.

し，機械学習による分類実験を行った．データの詳細に関しては Section 4 で述べる．ここでは抽出したフィンガープリントと機械学習による分類に関する説明を行う．

3.1　OS フィンガープリント抽出

ここでは，使用する OS フィンガープリントに関する説明を行う．

3.1.1　TCP/IP フィンガープリント

p0f でシグネチャとして利用されている TCP/IP フィンガープリントを本研究でも採用し，抽出を行った．ただし，本研究で用いるデータは SYN パケットではなく ClientHello パケットであるため，これらのフィンガープリントの中から TTL，Don't Fragment ビット，window size scaling factor，TCP オプションに限り利用する．Don't Fragment ビット及び TCP オプションに関しては，利用されていれば 1，されていなければ 0 として特徴量に設定する．また，TTL と window size scaling factor に関しては, 値を 0 から 1 までに正規化することで利用する．

3.1.2　SSL/TLS フィンガープリント

SSL/TLS フィンガープリントとして，CipherSuite リストを特徴量として抽出した．CipherSuite はクライアントが対応している暗号化方式であり，OS・ブラウザによって実装が異なることが知られている [18]．今回抽出した CipherSuite は全部で 101 種類であり，これらが利

表 3.1　CipherSuites リストの一部.

CipherSuites
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA

用されていれば 1，されていなければ 0 とするバイナリ表現で特徴量に採用する．表 3.1 に CipherSuites リストの一部を示す.

3.2　多クラス分類

前節で述べた TCP/IP 及び SSL/TLS フィンガープリントを特徴量として，機械学習による OS 多クラス分類を行った．機械学習のアルゴリズムとして，Support Vector Machine (SVM) 及び Random Forest (RF) を採用し，精度の比較を行った．今回はデータセットの都合上，10-fold cross-validation による精度の評価のみを行った．また，事前に grid search を行い，最適なパラメータを設定している.

第 4 章　データセット

データセットとして，*BotDetector* 用パケットと *HelloScanner* 用の SSL 通信パケットを用意した．*BotDetector* 用のデータセットでは，訓練用データ（表 4.3）と評価用データ（表 4.4）を用意した． 4.1 節と 4.2 節ではそれぞれ訓練用データと評価用データに関する詳細な内容を説明する．我々はロバストなアプローチを実現するため，訓練用データと評価用データをそれぞれ独立させている．つまり，マルウェア検体をそれぞれ異なるソースから，重複がないように収集した．表 4.1 は我々が収集した検体に関する詳細を示す．ファミリ名は比較的検知率が高かった Kaspersky [19] のものを参考にしている．表 4.2 は収集した検体の内，HTTP を利用する検体のみの詳細を示す．ここで，全検体に占める訓練用の HTTP 検体と評価用の HTTP 検体の割合を比較すると，後者が極端に小さいことがわかる．これは，我々が評価用データの分析の際に，動的解析ソフトウェアとして Cuckoo Sandbox [20] を利用しているからであることが考えられる．Cuckoo Sandbox はオープンソースのソフトウェアであり，それはつまり攻撃者も動的解析への対策を行っていることが予想される．動的解析中にマルウェアが Sandbox を検知し，その実行を停止してしまうと，パケットデータをキャプチャすることは不可能となる． 4.3 節では *HelloScanner* 用データセットの内容を示す．データセットの都合により 10-fold cross-validation による実験評価を行うため，今回は訓練用と評価用の区別はしていない.

表 4.1　収集したマルウェアの概要.

	# of collected samples	Malware type (Kaspersky)
Training data	24,260	not-a-virus:AdWare (39%), not-a-virus:HEUR:AdWare (29%), Undetectable (8.3%), not-a-virus:Downloader (7.9%), Trojan (5.2%), Trojan-Downloader (3.6%), HEUR:Trojan (2.3%), not-a-virus:WebToolbar (1.3%), not-a-virus:HEUR:Downloader (1.0%), Others (2.4%)
Test data	15,000	Undetectable (24%), not-a-virus:HEUR:WebToolbar (15%), HEUR:Trojan (12%), Trojan (8.0%), not-a-virus:AdWare (4.7%), P2P-Worm (4.0%), not-a-virus:Downloader (3.7%), Worm (3.0%), Trojan-Ransom (3.0%), Others (23%)

表 4.2　収集したマルウェアの内，HTTP 通信を行う検体の概要.

	# of samples using HTTP	Malware type (Kaspersky)
Training data	18,164	not-a-virus:HEUR:AdWare (37%), not-a-virus:AdWare (33%), Undetectable (10%), not-a-virus:Downloader (9.5%), Trojan-Downloader (3.9%), HEUR:Trojan (2.3%), not-a-virus:WebToolbar (1.7%), Others (2.6%)
Test data	3,593	not-a-virus:HEUR:WebToolbar (34%), Undetectable (22%), HEUR:Trojan (11%), not-a-virus:AdWare (5.4%), Trojan-Ransom (4.0%), not-a-virus:HEUR:AdWare (3.6%), Trojan (3.1%), Others (17%)

4.1 BotDetector 訓練用データ

4.1.1 悪性通信データ

訓練用の悪性通信データとして，24,260 種類のマルウェアとその HTTP トラフィックデータを使用した．各マルウェア検体は TrendMicro [21] と Kaspersky [19] によって悪性と判定されたものであり，検体間に重複は存在しない．表 4.2 に，我々が収集した検体の中で HTTP トラフィックを使用する検体のみの詳細を示す．ファミリに関して詳細に見ると，*Adware*，*Downloader*，そして *Trojan* が多く占めていることがわかる．これらの検体は，2014 年 12 月から 2015 年 9 月にかけて，サーバ型のハニーポット [22] とクライアント型のハニーポット [23] を利用して収集したものであり，商用の Sandbox を使用してインターネットに繋がった状態で解析を行った．Sandbox は Windows XP 及び 7 が搭載されており，各マルウェア検体を最大 5 分間実行させる．また，実行中のマルウェアによる他のシステムや端末への攻撃を防ぐため，あらかじめそのような通信を外部に届く前に妨ぐように設定している．ここで観測されたパケットデータから，我々は HTTP ヘッダの情報を抽出した．

4.1.2 正常通信データ

正常通信データとして，我々は正常な通信が大半を占めると考えられるキャンパスネットワークデータ（/16）を利用した．これらのデータは pcap 形式になっており，tcpdump によって 2016 年 8 月にゲートウェイで収集した．今回観測したユニークな IP アドレス数は全部で 1,110 だった．また，MalwareDomainBlocklist [24] に掲載されている URL にアクセスしているパケットは悪性が疑われるため，実験で使用する前にあらかじめ削除してから HTTP ヘッダ情報の抽出を行った．しかしながら，これでもキャンパスネットワーク内の通信の全てが正常な通信であると言い切ることはできないが，悪性通信が占める割合は限りなく低いことが想定されるため，今回我々はこれらのデータを正常通信として扱った．

4.2 BotDetector 評価用データ

4.2.1 悪性通信データ

悪性通信の評価用データとして，15,000 種類のマルウェア検体とその通信データを利用した．データ間の偏りを防ぐため，2016 年 6 月に検体配布サイトである Malwr [25]，MalShare [26]，そして VirusShare [27] から 5,000 検体ずつ Web クローラを作成し収集を行った．また，これらの検体は収集日から 1 年以内に各 Web サイトに登録されたものからランダムに収集したものであり，VirusTotal [28] によって少なくとも 1 つ以上のアンチウィルスベンダによって悪性であると判定された検体のみに限定している．表 4.2 を見ると，*Undectable* の割合が大きい．

表 4.3　BotDetector 訓練用データの詳細.

	# of samples	# of HTTP requests	collection periods
malicious	18,164	117,407	Dec 2014 - Sep 2015
benign	-	130,619	Aug 2016

表 4.4　BotDetector 評価用データの詳細.

	# of samples	# of HTTP requests	collection periods
malicious	3,593	77,110	Jun 2016
benign	-	79,751	Sep 2016

表 4.5　HelloScanner 用データの詳細.

	Windows	Mac OS	iOS	Android
# of ClientHello packets	316	45	58	54

Undectable は Kaspersky では検知できなかった検体のことを示すが，これは我々が特定のアンチウィルスソフトウェアに依存せずに検体を収集したためである．また，訓練用データと評価用データのマルウェア検体に重複は存在しない．マルウェアの通信データは，オープンソースの Sandbox である Cuckoo Sandbox [20] による動的解析によって収集した．動的解析の環境は，インターネットに繋いだ状態で Windows XP を利用し，Sandbox 内でマルウェアを最大 90 秒間実行した．

4.2.2　正常通信データ

評価用データの正常通信として，2016 年 9 月に訓練用データと同じ地点で再び収集したデータを用いる．これに関しても同様に，あらかじめ URL のブラックリストを用いてパケットの破棄を行い，HTTP 情報を抽出した．

4.3　HelloScanner 用データ

ここでは *HelloScanner* 実装のために収集したデータセットに関して述べる．ClientHello パケットを収集するためには，SSL サーバを設置する必要がある．そこで，我々は Amazon EC2 [29] を利用して SSL 対応 Web サーバを構築し，クラウドソーシングサービスの 1 つであるランサーズ [30] を利用してユーザにサーバへのアクセスを依頼した．この他，研究室のメンバーへ Web サーバアクセスの依頼も行った．サーバでは tcpdump がバックグラウンドで実行されており，443 ポート宛の通信のみ保存するよう設定されている．また，ClientHello パケットのみでは OS 情報との紐付けが不可能であるため，ユーザがサーバへアクセスする際に，自

身の利用している OS・ブラウザ情報を入力するようにフォーム画面を用意している．表 4.5 に各 OS 毎に収集した ClientHello パケットの数を示す．合計 473 パケットを用意した．

第 5 章　結果

本セクションでは，*BotDetector* と *HelloScanner* の実験結果及び有用な特徴量についてを述べる．

5.1　BotDetector の結果

5.1.1　テンプレート自動生成

まず初めに，テンプレート自動生成において閾値 β を我々がどのように定めたかに関して述べる．図 5.1 は β を変化させた際のテンプレート数の変化を表したグラフである．$\delta = 10^0$ の場合はテンプレートが生成されなかったことを意味する．δ と β の値が小さくなればなるほど，HTTP ヘッダフィールドの削減効果が大きくなることがわかる．また，図 5.2 では，ワイルドカードを含むテンプレートの割合も同様の傾向で増加することが示されており，最大で約 35% を占めている．これは，我々の自動テンプレート生成が非常に有効であることを示す．どちらの場合でも，テンプレート数は $\beta = 0.5$ の場合に顕著な差が現れていることがわかる．図 5.3 は $\mathrm{len}(F)$ の累積分布関数である．これを見ると，約 50% のヘッダフィールドが $\mathrm{len}(F) = 2$ であることがわかる．この結果は，β が 0.5 未満のときに，これらのフィールドのテンプレートが作成されることを示している．以上のことを考慮して，最適な閾値として $\beta = 0.5$ を設定した．

5.1.2　トラフィック分類結果

我々は評価用データに対する STM，SVM，RF，そして DNN のそれぞれの結果の比較を行った．閾値 δ の様々な値における ACC（Accuracy）s，FPR（False Positive Rate）s，FNR（False Negative Rate）s，そして予測にかかる時間を表 5.1 に示す．それぞれの結果は 5 回実験を行った際の平均値を採用しており，Ubuntu 13.10 (CPU: Intel Xeon CPU E5-2620 v2 @ 2.10 GHz, Memory: 64 GB) という環境で予測時間の計測を行った．ACC，FPR，そして FNR は以下の計算式によって求められる．

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

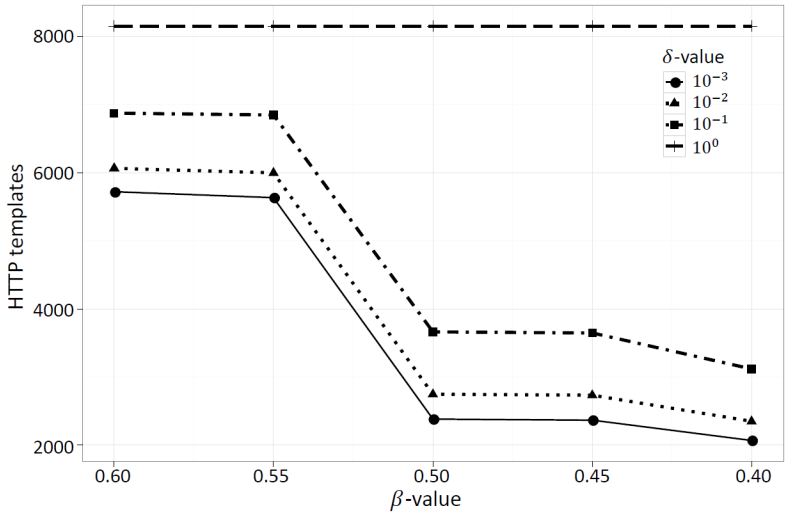

図 5.1　異なる閾値によるテンプレート生成数の違い.

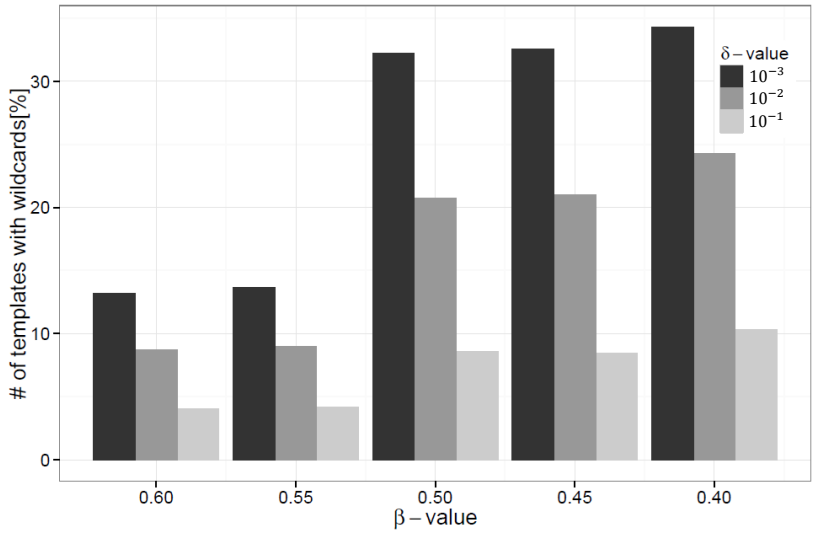

図 5.2　全体に占めるワイルドカード入りのテンプレートの割合.

$$FPR = \frac{FP}{FP + TN}$$

$$FNR = \frac{FN}{FN + TP}$$

TP と FP はそれぞれ true positive と false positive を示し，同様に TN と FN は true negative, false negative を示す．どの学習モデルでも，δ を 10^{-1} に設定した場合が最も ACC が高い結果となった．最も高い精度を達成したのは DNN による 97.1% であり，それに次いで SVM が高かった．DNN と SVM は精度自体にそこまで大差はないが，予測時間では SVM は DNN の約 1.5 倍長くかかることがわかった．予測時間では STM が最も優れていることがわかるが，ACC

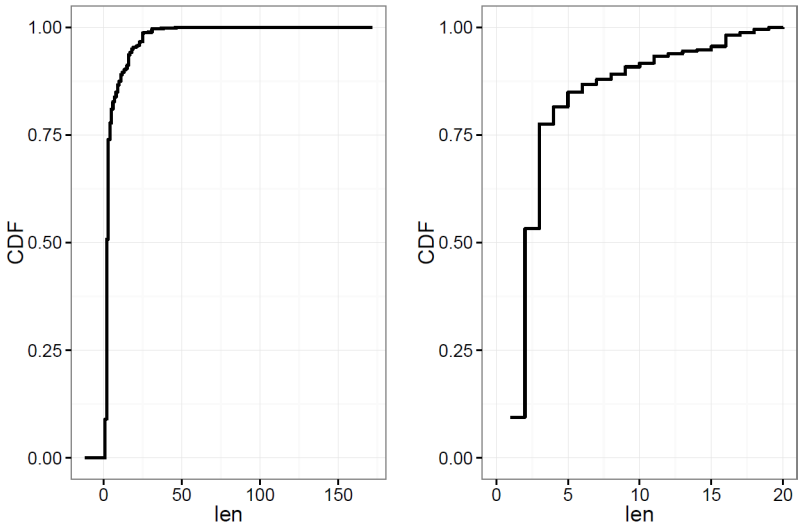

図 5.3　フィールド内の単語数（len(F)）の累積分布関数．左は原寸図を示し，右は拡大図を示す．

が他のモデルよりも大きく劣っている．RF も予測時間では比較的優れていたが，やはり ACC が DNN や SVM と比較して劣ることがわかった．これらの結果を考慮して，我々は DNN が最も優れた機械学習モデルであると判断した．

表 5.1　各機械学習モデルによる悪性通信の弁別精度.

δ	# of features	model	ACC (mean/std)	FPR (mean/std)	FNR (mean/std)	prediction time (mean/std) [s]
10^{0}	2,405	STM	0.856/0.000	0.006/0.000	0.361/0.000	0.234/0.005
10^{0}	8,155	SVM	0.923/0.000	0.004/0.000	0.152/0.000	310.7/12.67
10^{0}	8,155	RF	0.772/0.006	0.004/0.000	0.460/0.012	120.7/2.853
10^{0}	8,155	DNN	0.930/0.002	0.005/0.001	0.137/0.006	196.0/0.215
10^{-1}	616	STM	0.842/0.000	0.006/0.000	0.314/0.000	0.210/0.013
10^{-1}	3,662	SVM	0.962/0.000	0.007/0.000	0.070/0.000	134.4/3.591
10^{-1}	3,662	RF	0.824/0.095	0.007/0.000	0.350/0.193	47.54/1.527
10^{-1}	**3,662**	**DNN**	**0.971/0.000**	**0.007/0.000**	**0.052/0.000**	**88.38/0.182**
10^{-2}	413	STM	0.605/0.000	0.006/0.000	0.797/0.000	0.208/0.007
10^{-2}	2,745	SVM	0.960/0.000	0.008/0.000	0.074/0.000	108.5/3.192
10^{-2}	2,745	RF	0.706/0.013	0.008/0.002	0.589/0.026	39.12/1.589
10^{-2}	2,745	DNN	0.876/0.080	0.011/0.003	0.240/0.165	65.12/0.093
10^{-3}	366	STM	0.605/0.000	0.009/0.000	0.794/0.000	0.210/0.000
10^{-3}	2,382	SVM	0.959/0.000	0.008/0.000	0.074/0.000	99.26/2.470
10^{-3}	2,382	RF	0.711/0.012	0.007/0.000	0.581/0.025	34.77/1.545
10^{-3}	2,382	DNN	0.894/0.060	0.012/0.003	0.204/0.124	56.52/0.179

表 5.2　相互情報量上位 10 種のテンプレート．長いテンプレートは一部省略している．

HTTP templates	MI
User-Agent: Mozilla 4.0 (compatible; MSIE * Windows NT 5.1; Trident 4.0; ...	0.1523
User-Agent: Mozilla 4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident 4.0; ...	0.1371
Cache-Control: * 0	0.1298
Cache-Control: max-stale 0	0.1177
Accept-Encoding: gzip deflate *	0.1067
Accept-Encoding: gzip deflate sdch	0.1055
Pragma: *	0.0995
Pragma: no-cache	0.0993
Connection: *	0.0843
Content-length: *	0.0786

5.1.3　テンプレート自動生成の有効性

ここでは，テンプレート自動生成が実験結果にどれほど貢献しているかに関して述べる．閾値を $\delta = 10^{-1}$，$\beta = 0.5$ に設定した際に生成されたテンプレートに関して，我々は相互情報量 (MI) を算出した．MI は特定の特徴量の貢献率を調べるのに適した手法の 1 つであり，以下の計算式によって求められる．

$$MI(X;Y) = \sum_{i=1}^{m}\sum_{j=1}^{n} p(x_i, y_j) \log_2 \frac{p(x_i, y_j)}{p(x_i)p(y_j)},$$

$p(x_i)$ は確率変数 $X = \{x_1, x_2, ..., x_m\}$ 内の x_i の生起確率を示し，$p(y_j)$ は確率変数 $Y = \{y_1, y_2, ..., y_n\}$ 内の y_j の生起確率を示す．また，$p(x_i, y_j)$ は x_i と y_j の同時確率を示す．

表 5.2 は MI が高かった上位 10 種類のテンプレートを示す．ワイルドカードが挿入されたテンプレートが MI の上位にあることがわかる．この結果は，トラフィック分類においてテンプレート自動生成が特徴量の数を減らすだけでなく，効果的な特徴量を抽出することにも貢献していることを意味している．

図 5.4 は特徴量に多く含まれていた Key の数を示す．左図はテンプレート自動生成を使用しなかった場合の上位 10Key であり，右図はテンプレート自動生成を適応した後の Key である．この結果から，テンプレート自動生成によって Host フィールドと Content-Length フィールドが削減されたことがわかる．これは特徴ベクトルの大幅な削減に非常に貢献していることを意味する．一方で，Referer フィールドや Cookie フィールド，URI フィールドや User-Agent フィールドはあまり削減されていないことがわかる．Referer フィールドと Cookie フィールドは，クライアントが Web ブラウザを介して Web サーバにアクセスする際に用いられるフィー

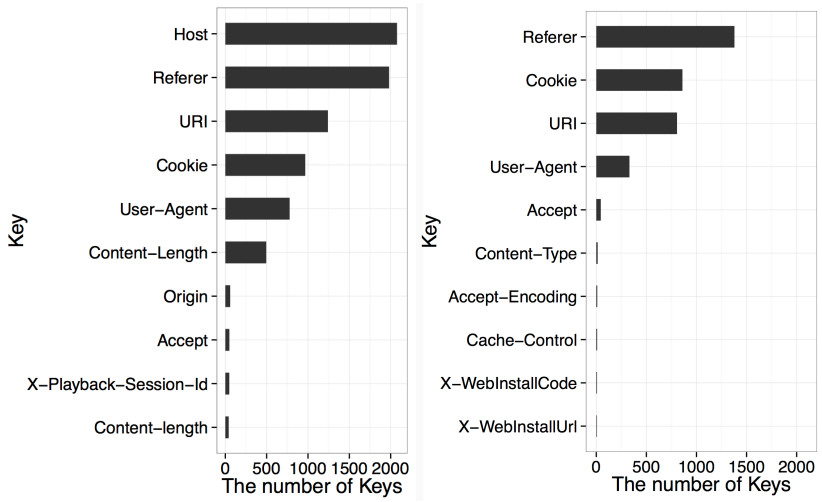

図 5.4　テンプレート生成の効果．左は β=0.5 と δ=1.0 のとき（テンプレート未生成）．右は β=0.5 と δ=0.1 のとき．各上位 10Key のみ抜粋．

ルドである．よってそれらの Value はあまり重要ではく，フィールドが使われていることが重要であることを意味する．URI と User-Agent は既存の研究 [31]， [32]， [33]， [34] でも有効な特徴として利用されてきている．これらの結果から，テンプレート自動生成アルゴリズムは重要な特徴を維持しながら，意味のない特徴を減らすことが可能であるとわかる．

5.2　HelloScanner の結果

5.2.1　OS 推定結果

ここでは，*HelloScanner* の実験結果について述べる．本実験では，機械学習のアルゴリズムとして SVM と RF を選択し，それぞれの精度の比較を行った．それぞれ事前に grid search を行い，最適なパラメータを設定している．さらに，各フィンガープリント毎に特徴量として使用した場合としなかった場合の比較も行った．また，データセットの都合上，今回は評価用データは用意せず，10-fold cross-validation による評価のみを行った．表 5.3 に各フィンガープリント毎に特徴量として使用した場合の精度の平均値，中央値，最大値，最小値，そして標準偏差をそれぞれ示す．SVM，RF 共に TCP/IP 及び SSL/TLS フィンガープリントの両方を特徴量に利用した場合に精度が最も高い結果となった．特に，SVM では TCP/IP のみと比べて約 13% 向上することがわかる．一方で，標準偏差は両方のフィンガープリントを利用する場合がどちらの学習モデルでも最も高い結果となった．また，両方のフィンガープリントを利用した場合の結果では，どれも僅かな差ではあるが，中央値以外の全ての値で RF が SVM を上回った．

次に，システムがどの OS 同士を誤推定したのかを調べるため，RF で両方のフィンガープリントを特徴量として利用した際に，最も精度が低かったときの混同行列を作成した．表 5.4 に

表 5.3　各フィンガープリントの組み合わせ毎の HelloScanner の結果.

TCP/IP	SSL/TLS	model	ACC(Ave)	ACC(Med)	ACC(Max)	ACC(Min)	std
✓	✓	SVM	0.886	0.906	0.956	0.778	0.058
✓	✓	**RF**	**0.893**	**0.895**	**0.957**	**0.816**	**0.044**
✓		SVM	0.761	0.765	0.809	0.711	0.034
✓		RF	0.867	0.860	0.933	0.826	0.035
	✓	SVM	0.819	0.817	0.889	0.776	0.038
	✓	RF	0.802	0.800	0.854	0.755	0.032

表 5.4　精度が最も低かった際の混同行列.

		predection value			
		Windows	Mac	iOS	Android
actual value	Windows	29	0	0	3
	Mac	0	2	2	1
	iOS	1	0	5	0
	Android	2	0	0	4

その結果を示す．Windows と iOS の識別精度は高い一方で，Android と Mac の誤推定が多いことがわかる．特に Mac に関しては精度が 50% を切っており，これによって全体の精度を下げてしまっている．

5.2.2　特徴量の重要度

ここでは，どの特徴量が OS 推定において貢献したかを述べる．特徴量の貢献度はジニ係数の減少量を利用することで算出することができ，値が大きいほど重要な特徴であることが言える．ジニ係数とは，不純度さを表す指標であり，0 から 1 までの値をとる．今回は 0 に近いほど特徴量の変数の分割結果がまとまっていることを示す．これにより，RF を形成する全ての決定木におけるジニ係数の減少量の平均が，特徴量の重要度に結びつく．以下にジニ係数を算出するための計算式を示す．

$$G(f) = 1 - \sum_{i=1}^{m} p(c_i)^2$$

ここで m は分類クラスの数を示し，$p(c_i)$ はデータ集合 f において i 番目のクラス c_i のデータに含まれる確率を示す．表 5.5 に，ジニ係数の減少量が大きい上位 10 個の特徴とその値を示す．RF において最も精度が高かった際のものを採用した．上位 3 つの特徴は Don't Fragment ビット以外の TCP/IP フィンガープリントであった．また，表には載っていないが Don't Fragment ビットのジニ係数の減少量は 0.0065 であった．一般的に，TTL 初期値は Windows が 128,

表 5.5　ジニ係数の減少量上位 10 個の特徴とその値.

features	gini decreases
TTL	0.2378
TCP options	0.1935
Window size scaling factor	0.1177
TLS_RSA_WITH_3DES_EDE_CBC_SHA	0.0750
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	0.0198
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	0.0173
TLS_RSA_WITH_AES_128_CBC_SHA256	0.0150
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	0.0135
TLS_RSA_WITH_AES_256_CBC_SHA256	0.0126
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	0.0124

Mac，iOS，Android では 64 が利用されている．そのためにジニ係数の減少量が高く算出されたことが考えられる．また，*TLS_RSA_WITH_3DES_EDE_CBC_SHA* は iOS 以外の OS のほとんどのパケットで利用されており，同様の理由でジニ係数の減少量が高くなったことがわかる．

第 6 章　議論

本セクションでは，提案手法に関する様々な議論を行う．また，システムを運用する上での制約に関しても述べる．

6.1　各機械学習モデルの結果

BotDetector の精度の結果は，表 5.1 に示されている．テンプレート生成によって，STM のみ精度が悪化することがわかる．ワイルドカードが挿入されたテンプレートは様々なフィールドを内包する特徴とみなすことができ，悪意のあるフィールドのみに依存するこのモデルの精度は低下すると考えて差し支えない．また，RF の精度が低いことは我々にとって意外な事実であった．RF は多数の決定木を構築することによって分類を行う，アンサンブル学習の 1 つである．一般的に高い分類精度を発揮することで知られているが，今回使用したデータセットには適していなかった．SVM は最も有名な機械学習アルゴリズムの 1 つである．その精度は DNN に次ぐ結果となった．DNN は我々が試行したアルゴリズムの中で最も精度が高かったが，欠点として識別器の学習に時間がかかる点が挙げられる．

HelloScanner の精度の結果は，表 5.3 に示されている．SVM と RF 共に，TCP/IP 及び SSL/TLS フィンガープリントを組み合わせた場合に最も精度の平均値が高い結果となった．SVM で精度の平均値が次に高かったのは，SSL/TLS フィンガープリントのみを利用する場合であった．一方で，RF の場合は TCP/IP フィンガープリントのみを使う場合に高い精度を発揮した．また，学習モデルに限らず，両方のフィンガープリントを利用する場合に標準偏差が僅かながら高くなることがわかる．このことから，特徴量の数を増やすことで精度にばらつきが生じやすくなることが推測される．今後の課題として，多種多様な CipherSuites リストの中で特徴として利用するものとしないものを区別することが挙げられる．その他，データセットを拡充することが標準偏差を縮める上で有効であると考えられる．

6.2　誤検知の原因

BotDetector は高い精度を達成したが，誤検知が 0 というわけではない．実際に，β=0.5 かつ $\delta = 10^{-1}$ のときの DNN の FPR と FNR はそれぞれ 0.7% と 5.2% であった．誤検知の多

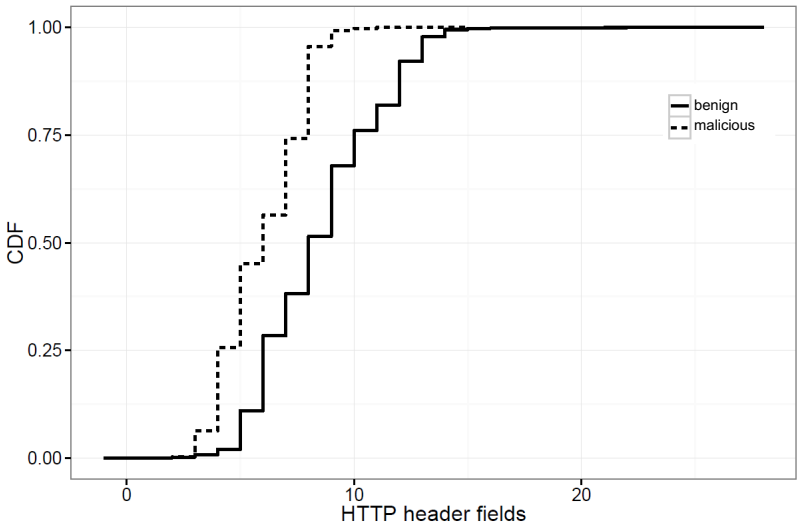

図 6.1　HTTP リクエストパケット内のフィールド数の累積分布関数.

くは，パケット内の HTTP ヘッダフィールドの数に関連していることがわかった．図 6.1 は，各パケットの HTTP ヘッダフィールド数の累積分布を示す．悪性通信は少数のフィールドで構成されている傾向がある．一方で，正常通信の場合は多くのオプションを利用する可能性のあるブラウザ通信が多いという結果を解釈できる．この性質のため，正常通信に少数の HTTP ヘッダフィールドしか含まれていない場合，悪性であると誤って検出される可能性が考えられる．この問題を解決する 1 つの方法は，十分な情報がないときには決定を回避することである．しかしながら，本研究は決してブラウザと非ブラウザの通信を弁別するものではない．正常通信データにはブラウザ通信だけでなく，他のソフトウェアやスマートフォンアプリによる通信も少なからず含まれており，同様に悪性通信データにはソフトウェアによる通信のみならず，表 5.2 に示されるような Internet Explorer の User-Agent を模した通信が多く含まれている．それにもかかわらず，我々の提案手法では悪性通信と正常通信の高い弁別精度を達成している．

HelloScanner も同様に高い精度を達成しているが，これに関しても誤推定は無視できない値である．特に Mac の誤推定が顕著である．表 5.4 では Mac の誤推定率は 50% を超えてしまっている．これは，Mac を一意に識別するための有意な特徴が少なかったことが考えられる．例えば Windows では，TTL が他の OS とくらべて唯一 128 に設定されており，有意な特徴であることがわかる．また，iOS 及び Android に関しても，スマートフォンに実装されている傾向が強い暗号化方式がいくつかあり，これに関しても有意な特徴であると言える．Mac の精度を向上させるためには，このような特徴を新しく見つける必要があり，今後の課題の 1 つとして挙げられる．

6.3 制約

BotDetector は HTTP ヘッダフィールドを利用するため，HTTPS のように通信が暗号化された場合は適用することができない．それに加えて，HTTP プロトコルはマルウェアが利用するメジャーなプロトコルの 1 つではあるが，必ずしも利用するわけではない．UDP ベースのプロトコルを利用するマルウェアの場合は，検知が困難となる．その他の *BotDetector* の制約として，マルウェアがブラウザ通信を完全に模倣した場合も，検知は困難になることが想定される．これは現時点ではメジャーなケースではないが，将来的には標準的な脅威になる可能性がある．よって，その時点で特徴抽出と分類モデルを再度考慮する必要がある．我々が本研究で使用した検体は，Windows マルウェアに限定している．今後の課題として，他のプラットフォームのマルウェア検体を使用した提案手法の検証を行うことである．これらの制限を踏まえても，この研究の背後にある “有用な機能を自動的に見つける” という基本的な考え方は，悪性通信検知に向けた有効な特徴を発見するために有益であると考えている．

HelloScanner の制約は *BotDetector* とは逆であり，HTTPS，特に ClientHello パケットにしか適用できない点である．また， 6.2 節でも述べたように，Mac を推定する際の精度が不十分であり，この点も制約の 1 つと言える．一方で，HTTPS のパケットを利用した OS フィンガープリンティング技術はまだあまり浸透しておらず，希少性が高いと言える．よって，本システムのみを運用させて OS 推定を行うのではなく，既存の SYN パケットを利用した OS フィンガープリンティング等と組み合わせて利用することで，より目標の実現に近付くと我々は確信している．

6.4 実ネットワークでの実装

BotDetector と *HelloScanner* の機能を考慮すると，このシステムをセキュリティアプライアンスシステムの一部として実装し，悪性通信からユーザを保護する機能として提供することを提案する．これらのシステムを設置するのに最も適しているのは，バックボーンネットワーク内で多くのエンドユーザの通信が集約される場所である．我々の提案手法の高いスケーラビリティを考慮すると，膨大な量の HTTP・HTTPS リクエストパケットを処理することができる．別の実装としては，多くのエンドユーザのヘッダ情報を監視できる Web プロキシサーバーに本システムをインストールすることである．今後の課題として，上記のような環境での提案手法の実装が挙げられる．

第 7 章　関連研究

本セクションでは，HTTP 情報を用いた悪性検体・通信を検知する研究 [31]， [32]， [33]， [34]， [35]， [36]， [37] 及び OS フィンガープリントに関連する研究 [38]， [39]， [40]， [18]， [41] を示す．また，既存研究と我々の提案手法との違いについても述べる．

7.1　悪性検体の検知に関する研究

Zhang ら [31] は正規表現による検知システムを回避する User-Agent の存在を明らかにし，Web ブラウザが利用する User-Agent の属性を細かく分析している．また，この情報と OS フィンガープリンティングを活用することで，偽造された User-Agent を特定することが可能であるという．Grill ら [32] も User-Agent を利用して悪性通信を検知する研究を行っている．彼らは User-Agent を 5 つのカテゴリである *Legitimate user's browser*，*Empty*，*Specific*，*Spoofed*，そして *Discrepant* に分類した．*Spoofed* の場合はユーザ個人が使用しているブラウザの User-Agent をそのまま利用するため，それ単体の情報のみでは検知することは困難であるという．彼らの研究では悪性通信を検知する際に User-Agent しか使用していない．我々の提案手法では User-Agent はあくまで特徴量の 1 つとして設定しており，その他のヘッダ情報もテンプレートを応用することで特徴量として採用している．

Nelms ら [33] は *ExecScent* と呼ばれるシステムを提案している．*ExecScent* は HTTP ヘッダフィールドの情報を利用して bot を検知することに焦点をあてている．URL パスやクエリ，User-Agent 情報等と正規表現を利用してテンプレートを手動で作成し，シグネチャマッチングによる検知を行っている．Chiba ら [34] は *BotProfiler* と呼ばれるシステムを構築した．*BotProfiler* は *ExecScent* の精度を改善することに着目しており，URL パス，URL クエリ，User-Agent の 3 つに絞ってテンプレート生成を行い，シグネチャマッチングを行う．彼らの提案手法は，どちらも手動でテンプレートを作成する必要がある．一方で我々の手法では，自動的に効率的なテンプレートを生成することができるため，手動で作成する必要がない．

Xie ら [35] は *AutoRE* と呼ばれるシステムを実装している．*AutoRE* は URL 構造から正規表現を利用してシグネチャを作成し，ボットネットをベースにしたスパムメールを検知するためのシステムである．我々の *BotDetector* はマルウェア感染端末を発見することであり，スパムメールを検知することとは目的が異なる．Zallas ら [36] もまた，自動的にテンプレートを生成

する技術を示している．彼らのテンプレート生成手法は非常にシンプルなものであるが，我々の自動テンプレート生成は確率論に基づいた手法であり，それと同時にクラスタリングアルゴリズムを用いることで効率的なテンプレートを生成することが可能である．また，我々は様々な機械学習モデルを用いて精度の比較を行っている．Perdisci ら [37] は HTTP リクエスト・レスポンスパケットに着目し，HTTP ベースのマルウェアを検知する研究を行っている．彼らはマルウェアを数段階のクラスタに分類し，シグネチャを生成している．彼らの研究では HTTP リクエスト・レスポンスの両方のパケットを解析する必要があるが，我々の研究は HTTP リクエストのみを解析対象としている．これにより，もし C&C サーバの IP アドレスが変更され，レスポンスパケットが解析できない状況になった場合でも，我々の提案手法ではマルウェア感染端末の特定が可能である．

これらの既存研究はすべて，マルウェアを検出するための有用な特徴を抽出する際にドメイン知識に大きく依存していたが，我々はテンプレート生成アルゴリズムを利用して特徴抽出プロセスを自動化することを目指している．また，DNN はニューラルネットワークによって自動的に有効な特徴を抽出することが可能である．我々の提案手法は，マルウェアの通信機能の変更に対してロバストであるため，過去の研究で使用されていたアプローチを上回る利点があると考えている．

7.2 OS フィンガープリンティングに関する研究

Al ら [38] は p0f の改良を提案している．これまでの SYN パケットだけでなく，FIN+ACK パケットもシグネチャとして新たに追加し，分類をパターンマッチングから機械学習に変更することで，従来の p0f の精度を上回ることを実証した．Chen ら [39] は Windows，Mac，iOS の分類を行っている．従来の p0f で用いられてきた特徴に加え，端末のクロック周波数を特徴量に加えることで，高い精度で各 OS の分類が可能であることを示した．また，これによってテザリング回線の識別も可能であるという．これらの研究はどちらも SYN パケットを代表とした TCP パケットに焦点を当てている．我々は HTTPS パケットの中でも ClientHello パケットに着目しており，SSL/TLS フィンガープリントの有意性を示している．

Matsunaka ら [40] は Android 端末に関する OS フィンガープリンティング手法を提案している．彼らは Android 端末の通信にのみ現れる固有のドメインとその周期性を利用し，ネットワーク内の Android 端末台数の推定を行った．しかし，この提案手法は Android 端末にしか適用することができず，他の OS に対応することができない．

Husak ら [18] は ClientHello パケットのヘッダに記録されている CipherSuites リストを利用し，HTTPS クライアントの分類を行っている．彼らの提案手法では利用している端末がデスクトップ端末かモバイル端末かまでしか判別することができなかったが，我々は TCP/IP フィンガープリントを組み合わせることで，高い精度で OS の分類まで行うことを可能としている．Anderson ら [41] は Windows，Mac，iOS の OS 情報及びバージョン情報の分類を行っている．

彼らは 1 時間以内にユーザが送信した TCP，HTTP，HTTPS パケットを特徴量として利用することで，高い精度で分類が可能であることを実証した．しかしながら，それらは IP アドレスによって紐付けが行われるため，NAT を通過する場合に対応することができない．

これまでの既存研究では TCP・SYN パケットや DNS パケットを使うものが多いが，我々は ClientHello パケットに含まれる TCP/IP と SSL/TLS フィンガープリント利用した分類に着目している．また，我々の手法では ClientHello パケット単体で OS の推定を行うため，複数の異なるプロトコルのパケットを IP アドレスで紐付ける必要がない．したがって，NAT 配下の端末でも OS 推定を行うことが可能である．

第 8 章　結論

我々はセキュリティインシデント対応に向けて，2 つの手法を提案した．1 つ目は *BotDetector*，2 つ目は *HelloScanner* である．*BotDetector* は悪性通信を検知するためのシステムであり，マルウェアに感染した端末の検知を行った．このシステムの主なアイデアは，各 HTTP ヘッダフィールドの情報を収集し，検知に向けた機械学習を使用するための “テンプレート” を自動的に作成することである．大規模なデータセットを使用した実験の結果，我々は *BotDetector* が 97.1 %の精度で悪性通信を検知し，誤検知率を 1.0% 未満に抑えることが可能であることを実証した．特筆すべきは，保存する情報量を減らすだけでなく，有用な特徴を抽出するテンプレート自動生成アルゴリズムを導入したことである．

HelloScanner は OS を分類するためのシステムである．主要なアイデアは，HTTPS によって暗号化される前の通信である ClientHello パケットから TCP/IP 及び SSL/TLS フィンガープリントを抽出し，これらを組み合わせて特徴量とすることで，機械学習による OS の推定を行うことである．データセットを用いた実験の結果，*HelloScanner* が 89.3% の精度で OS 分類を可能とし，SSL/TLS フィンガープリントが有用な特徴となり得ることを示した．

本研究で使用された主要なアイデアやアプローチは，多数の機能を備えたマルウェアによる悪性挙動の分類や，様々な OS 及びそのバージョン情報を分類する他の研究に役立つと我々は確信している．

研究業績

学部・専攻在籍中の研究業績を下記に示す.

ジャーナル論文誌

1. S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Detecting Malware-infected Devices Using the HTTP Header Patterns", IEICE Transactions on Information Systems, Vol. XX, No. XX, pp. XX-XX, 2018 (条件付採録)

国際会議 (査読付き)

1. S. Mizuno, M. Hatada, T. Mori, and S. Goto, "BotDetector: A robust and scalable approach toward detecting malware-infected devices", Proceedings of the IEEE International Conference on Communications (ICC 2017), May 2017

国際会議 (招待講演)

1. S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Detecting malware-infected hosts using HTTP fingerprints”, The 42nd APAN Meeting, Network Security Workshop Aug 2016

国際ワークショップ

1. S. Mizuno, M. Hatada, T. Mori, and S. Goto, "Detecting malware-infected hosts based on the variability of HTTP header fields", Hanyang-Waseda IT Workshop 2016, Nov 2016

国内研究会

1. 水野 翔，畑田 充弘，森 達哉，後藤 滋樹，"HTTP ヘッダフィールドの可変性に基づくマルウェア感染端末の特定"，コンピュータセキュリティシンポジウム 2016 論文集，vol. 2016，No. 2，pp. 632-639，2016 年 10 月
2. 水野 翔，畑田 充弘，森 達哉，後藤 滋樹，"マルウェアに感染したホストによる通信の弁別方法"，信学技報，vol. 115，no. 488，ICSS2015-66，pp. 117-122，2016 年 3 月

謝辞

本研究を進めるにあたり，多くの議論と指導をしてくださった森達哉准教授に感謝いたします．畑田充弘氏にはデータセットを提供していただくと共に，数多くのご助言をいただきました．ここに感謝の意を表します．後藤滋樹教授には論文のご助言をいただくと共に，本論文の細部にわたりご指導をいただきました．ここに同氏に対して感謝の意を表します．このほか，ゼミをはじめ日頃より議論に応じてくださった森研究室の皆様に感謝いたします．本研究の一部は JSPS 科研費 16H02832 の助成を受けたものです．

参考文献

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[2] McAfee Labs Threats Report: April 2017. https://www.mcafee.com/us/security-awareness/articles/mcafee-labs-threats-report-mar-2017.aspx.

[3] ESET predictions and trends for cybercrime in 2016. http://www.welivesecurity.com/2015/12/23/eset-predictions-for-cybercrime-trends-in-2016/.

[4] Large CCTV Botnet Leveraged in DDoS Attacks. https://blog.sucuri.net/2016/06/large-cctv-botnet-leveraged-ddos-attacks.html.

[5] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, USENIX Security '14, pp. 287–301, Aug. 2014.

[6] John P John, Alexander Moshchuk, Steven D Gribble, and Arvind Krishnamurthy. Studying spamming botnets using botlab. In *NSDI*, Vol. 9, pp. 291–306, 2009.

[7] HTTPS encryption on the web. https://transparencyreport.google.com/https/overview.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, Vol. 96, pp. 226–231, 1996.

[9] Tatsuaki Kimura, Keisuke Ishibashi, Tatsuya Mori, Hiroshi Sawada, Tsuyoshi Toyono, Ken Nishimatsu, Akio Watanabe, Akihiro Shimoda, and Kohei Shiomoto. Spatio-temporal factorization of log data for understanding network events. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 610–618. IEEE, 2014.

[10] LIBSVM – A Library for Support Vector Machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[11] scikit-learn: machine learning in Python. `http://scikit-learn.org/stable/`.

[12] TensorFlow - an Open Source Software Library for Machine intelligence. `https://www.tensorflow.org/`.

[13] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhut-

dinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.

[14] Pierre Baldi and Peter J Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pp. 2814–2822. Curran Associates, Inc., 2013.

[15] Nmap: the Network Mapper - Free Security Scanner. https://nmap.org/.

[16] Ofir Arkin, Fyodor Yarochkin, and Meder Kydyraliev. The present and future of xprobe2: The next generation of active operating system fingerprinting. sys-security group. 2003.

[17] p0f v3 - Lcamtuf - coredump.cx. http://lcamtuf.coredump.cx/p0f3/.

[18] Martin Husák, Milan Cermák, Tomá Jirsík, and Pavel Celeda. Network-based https client identification using ssl/tls fingerprinting. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*, pp. 389–396. IEEE, 2015.

[19] Kaspersky Lab |Antivirus Protection & Internet Security Software. `http://www.kaspersky.com/`.

[20] Cuckoo Sandbox: Automated Malware Analysis. `https://www.cuckoosandbox.org/`.

[21] Content security software - Internet Security & Cloud - Trend Micro USA. `http://www.trendmicro.com`.

[22] 青木一史, 川古谷裕平, 秋山満昭, 岩村誠, 針生剛男, 伊藤光恭ほか. 能動的攻撃と受動的攻撃に関する調査および考察. 情報処理学会論文誌, Vol. 50, No. 9, pp. 2147–2162, 2009.

[23] Mitsuaki Akiyama, Makoto Iwamura, Yuhei Kawakoya, Kazufumi Aoki, and Mitsutaka Itoh. Design and implementation of high interaction client honeypot for drive-by-download attacks. *IEICE transactions on communications*, Vol. 93, No. 5, pp. 1131–1139, 2010.

[24] DNS-BH - Malware Domain Blocklist. `http://www.malwaredomains.com/`.

[25] Malwr - Malware Analysis by Cuckoo Sandbox. `https://malwr.com/`.

[26] MalShare. `http://malshare.com/`.

[27] VirusShare.com. `https://virusshare.com/`.

[28] VirusTotal - Free Online Virus, Malware and URL Scanner. `https://www.virustotal.com/`.

[29] Amazon Web Services. https://aws.amazon.com/jp/.

[30] クラウドソーシングなら日本最大級の「ランサーズ」. https://www.lancers.jp/.

[31] Yang Zhang, Hesham Mekky, Zhi-Li Zhang, Ruben Torres, Sung-Ju Lee, Alok Tongaonkar, and Marco Mellia. Detecting malicious activities with user-agent-based profiles. *International Journal of Network Management*, Vol. 25, No. 5, pp. 306–319, 2015.

[32] Martin Grill and Martin Rehak. Malware detection using http user-agent discrepancy identification. In *Parallel Computing Technologies (PARCOMPTECH), 2015 National Conference*

on, pp. 221–226. IEEE, 2015.

[33] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pp. 589–604, 2013.

[34] Daiki Chiba, Takeshi Yagi, Mitsuaki Akiyama, Kazufumi Aoki, Takeo Hariu, and Shigeki Goto. Botprofiler: Profiling variability of substrings in http requests to detect malware-infected hosts. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, Vol. 1, pp. 758–765. IEEE, 2015.

[35] Yinglian Xie, Fang Yu, Kannan Achan, Rina Panigrahy, Geoff Hulten, and Ivan Osipkov. Spamming botnets: signatures and characteristics. *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 4, pp. 171–182, 2008.

[36] Apostolis Zarras, Antonis Papadogiannakis, Robert Gawlik, and Thorsten Holz. Automated generation of models for fast and precise detection of http-based malware. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pp. 249–256. IEEE, 2014.

[37] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, Vol. 10, p. 14, 2010.

[38] Taher Al-Shehari and Farrukh Shahzad. Improving operating system fingerprinting using machine learning techniques. *International Journal of Computer Theory and Engineering*, Vol. 6, No. 1, p. 57, 2014.

[39] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. Os fingerprinting and tethering detection in mobile networks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 173–180. ACM, 2014.

[40] Takashi Matsunaka, Akira Yamada, and Ayumu Kubota. Passive os fingerprinting by dns traffic analysis. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference On*, pp. 243–250. IEEE, 2013.

[41] Blake Anderson and David McGrew. Os fingerprinting: New techniques and a study of information gain and obfuscation. *arXiv preprint arXiv:1706.08003*, 2017.