Vol.2017-DPS-170 No.25
Vol.2017-CSEC-76 No.25
2017/3/3

サイバー攻撃に対するサーバアプリケーション向け免疫強化モジュールの改良

多羅尾　光宣[†1]　岡本　剛[†1]

神奈川工科大学
〒243-0292 神奈川県厚木市下荻野 1030
s1685011@cce.kanagawa-it.ac.jp　take4@nw.kanagawa-it.ac.jp

概要：本研究は，サイバー攻撃に対してレジリエンスを強化するサーバアプリケーション向けセキュリティモジュールの開発を目的としている．セキュリティモジュールは，免疫のように攻撃を適応的に学習することにより，攻撃の検出精度を改善することができる．これまでの研究では，攻撃の学習を担う機械学習を外部の Python スクリプトに依存していたため，攻撃の学習と分類におけるオーバーヘッドが大きかった．このオーバーヘッドを減らすため，機械学習をネイティブコードで実装した．さらに，サーバの内部リソースを消費させる DoS 攻撃も学習できるようにした．本稿では，改良したセキュリティモジュールの検出精度やオーバーヘッドを報告する．また，サーバの長期稼働に向けて，学習データの記憶領域を管理する方法と記憶領域の大きさについても報告する．

キーワード：免疫系，機械学習，エクスプロイト，DoS 攻撃，侵入検知

Improvement of an immunity-enhancing module for server applications against cyber attacks

Mitsunobu Tarao[†1]　Takeshi Okamoto[†1]

Kanagawa Institute of Technology.
1030, Shimo-ogino, Atsugi, Kanagawa 243-0292, JAPAN
s1685011@cce.kanagawa-it.ac.jp　take4@nw.kanagawa-it.ac.jp

Abstract: This study focuses on an artificial immune security module for high-availability servers against cyber attacks on the Internet. The security module improves its detection accuracy against cyber attacks by adaptively acquiring immunity against the attacks. Our previous paper showed that the overhead of the security module with the random forest classifier was approximately 54%, because the module depended on a Python script to classify and learn a data. To reduce the overhead, we implemented a new security module using ranger, which is a fast implementation of random forest classifiers. The module has a new function to learn some of DoS attacks. This paper reports the detection accuracy of the new security module and its overhead. In addition, we showed a suitable size of learning data stored in memory for long-term operation.

Keywords: artificial immune system, machine learning, exploit, shellcode, DoS attack, intrusion detection

1. はじめに

IoT 時代の幕開けとともに，高可用性サーバの需要が高まりつつある．高可用性サーバの重要な機能の 1 つに，フォールトトレランスがあるが，フォールトトレランスは，サーバアプリケーションの脆弱性を悪用したサイバー攻撃に対応していない．サーバアプリケーションに任意のコードを実行できる脆弱性があれば，サーバアプリケーションの機密性・完全性・可用性を失うおそれがある．これは，高可用性サーバにおいて，重大な致命的な問題である．

次世代型のセキュリティ対策ソフトとして位置づけられる Palo Alto Networks traps，CrowdStrike Falcon Host，FFRI yarai，Microsoft EMET などは，未知や既知のサイバー攻撃を検知・防止できる．また，製品の他に学術研究機関の成果として，ROPGuard[1], ROPecker[2], SecondDEP[3]などがある．しかし，これらの製品や手法は，サイバー攻撃の検知と防止を目的としているため，たとえサイバー攻撃を検知・防止できたとしても，サーバアプリケーションのプロセスは正常な制御を失っているため，サービスを提供できない．サーバアプリケーションを再起動すれば，サービスを再開できるが，同じサイバー攻撃を行われるとサービスが停止する．したがって，サービスの再起動だけでは可用性を確保できない．

サイバー攻撃に対するサーバの可用性を確保するために，これまで免疫を模擬したセキュリティモジュールを開発してきた．このモジュールは自然免疫と獲得免疫の機能で構

†1 神奈川工科大学
Kanagawa Institute of Technology

Vol.2017-DPS-170 No.25
Vol.2017-CSEC-76 No.25
2017/3/3

成され，自然免疫機能によりサイバー攻撃を検知し，獲得免疫機能により適応的にサイバー攻撃を検知・学習する．これまでの研究で，複数種の攻撃を学習すると，正常なリクエストを攻撃と誤検知することがわかっている[4]．また，シミュレーションによりランダムフォレスト分類器が誤検知を減らせることを確認した．しかし，Python の scikit-learn のランダムフォレスト分類器で実装したセキュリティモジュールは，scikit-learn パッケージの処理が原因で，約 54%のオーバーヘッドを観測した[5]．このオーバーヘッドを減らすために，C++言語で実装されたランダムフォレストの ranger により，セキュリティモジュールを実装した．さらに，サーバの内部リソースを消費させる DoS 攻撃も学習できるようにした．本稿では，改良したセキュリティモジュールの検出精度やオーバーヘッドなどを報告する．

2. 免疫を模擬したセキュリティモジュール

免疫を模擬したセキュリティモジュールは，自然免疫機能と獲得免疫機能から構成される．自然免疫機能は，未知・既知のサーバアプリケーションの脆弱性を悪用したサイバー攻撃を検知する．この機能は，免疫系のナチュラルキラー細胞とマクロファージのものと類似している．ナチュラルキラー細胞は，未知や既知にかかわらずウイルスに感染した細胞を認識し破壊する．また，マクロファージも同様に未知や既知にかかわらずバクテリアなどを貪食することにより無害化する．獲得免疫機能は，自然免疫機能が検知したサイバー攻撃を学習し，2 回目以降の攻撃を検知する．この機能は，免疫系の免疫記憶と類似している．免疫記憶は，2 回目の感染に対して，1 回目の感染よりも速やかにかつ強力に感染細胞を排除するメカニズムが働く．図 1 に生物の免疫系とその免疫系を模擬したセキュリティモジュールの類似性を示す．

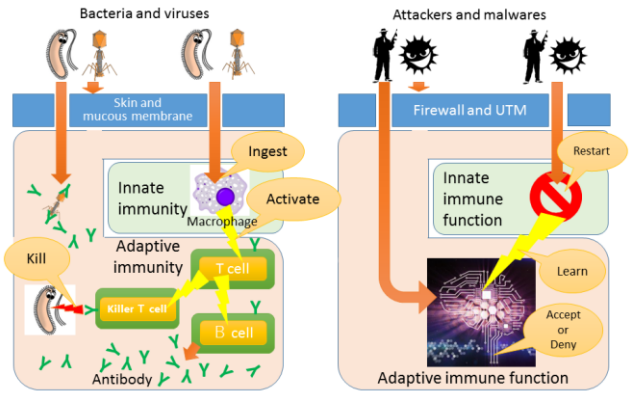

図 1 生物の免疫系とセキュリティモジュールの類似性[4]

2.1 自然免疫機能

自然免疫機能は，未知・既知のサーバアプリケーションの脆弱性を悪用したサイバー攻撃を検知する．もし，自然免疫機能がサイバー攻撃を検知したら，新たにサーバアプリケーションのプロセスを生成し，サイバー攻撃を受けたプロセスを終了させる．また，同時に，自然免疫機能は，獲得免疫機能にサイバー攻撃を検知したことを伝える．

本研究が対象にするサイバー攻撃は，任意のコードを実行するエクスプロイトといくつかの DoS 攻撃である．

2.1.1 エクスプロイトの検知

攻撃者は，任意のコードを実行するために，以下に示す 3 つの手順を実行する．

（1） バッファオーバーフローなどの脆弱性を悪用して，プロセスの実行制御を奪う
（2） DEP などの OS のセキュリティ機能を回避する
（3） シェルコードを実行する

したがって，検知手法は，脆弱性攻撃の検知，OS のセキュリティ機能回避の検知，シェルコードの検知の 3 つに分類できる．脆弱性攻撃の検知手法には，Snort などのパターンマッチングがある．セキュリティ機能回避の検知手法には，ROPGuard[1]や ROPecker[2]がある．シェルコードの検知には，Microsoft EMET のエクスポートアドレスフィルタリングや SecondDEP[3]などがある．エクスプロイトの検知は，これらのすべて検知機能かいくつかの検知機能を有する．

2.1.2 DoS 攻撃の検知

DoS 攻撃は，以下に示す 3 種類に分類できる．

- セキュリティホールを悪用してサーバアプリケーションを停止させる攻撃[6][7][8]
- SYN フラッド攻撃や HTTP POST DoS 攻撃など，サーバのリソースを枯渇させる攻撃
- DNS や NTP を利用したリフレクション攻撃などにより，大量のリクエストをサーバに送信し，ネットワーク帯域を逼迫させる攻撃

本研究は，1 つ目と 2 つ目の攻撃を対象にする．3 つ目の攻撃は，サーバの上流でトラフィックを抑えなければならないため，本研究の対象としない．この種の攻撃には，インターネットサービスプロバイダーやコンテンツデリバリネットワークが提供している DoS 攻撃対策サービスを使うこととする．

1 つ目の攻撃を検知するには，サーバアプリケーションの異常終了やリクエストの応答時間を監視する方法がある．2 つ目の攻撃を検知するには，サーバのリソースを監視する方法や，SYN クッキーなど OS から提供されているセキュリティ機能を利用する方法がある．

2.1.3 獲得免疫機能

獲得免疫機能は，自然免疫機能が検知したサイバー攻撃を機械学習により学習する．自然免疫機能がサイバー攻撃を検知するごとに，獲得免疫機能は，より多くのサイバー攻撃を検知できるようになると考えられる．

サーバアプリケーションが，リクエストを受信したら，獲得免疫機能は，そのリクエストが，過去に学習した攻撃

リクエストと類似しているかを確認する．もし，類似していたら，獲得免疫機能が，そのリクエストを破棄する．

獲得免疫機能によりサイバー攻撃を検知したら，サーバアプリケーションは再起動せずにサービスを提供できる．つまり，獲得免疫機能によりサーバアプリケーションの可用性を維持できる．

機械学習は，教師あり学習と教師なし学習に分けられる．教師あり学習の分類器には，決定木や SVM などがあり，教師なし学習の分類器には，K 平均法や混合ガウスモデルなどがある．本研究では，分類は「正常」と「攻撃」の 2 つであり，自然免疫機能により正しい分類結果が得られるため，教師ありの機械学習を利用する．

3. プロトタイプシステムの構成と実装

免疫を模擬したセキュリティモジュールの有効性を評価するために，プロトタイプシステムと脆弱性があるウェブサーバアプリケーションを実装した．

プロトタイプシステムは，3 つのモジュールから構成される．自然免疫機能と獲得免疫機能は IMMUNITY.DLL によって動作する．IMMUNITY.DLL はウェブサーバアプリケーションの内部で動作できるように MONITORING.EXE によってウェブサーバアプリケーションに注入される．また，MONITORING.EXE は，自然免疫機能による強制終了を検知したら，サーバアプリケーションを再起動させる．図 2 にプロトタイプシステムの構成を示す．

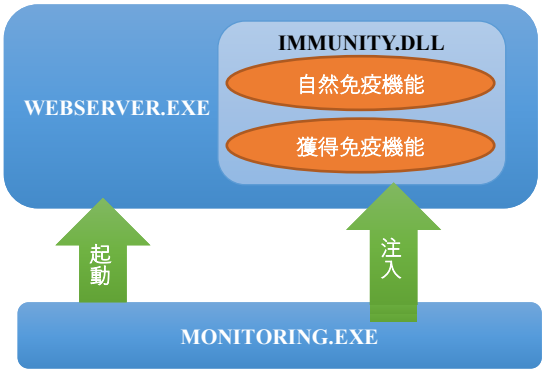

図 2　プロトタイプシステムの構成

3.1 自然免疫機能

自然免疫機能には，エクスプロイト検知機能と DoS 攻撃検知機能がある．これらの機能は IMMUNITY.DLL に含まれる．これらの機能が攻撃を検知すると同時に攻撃を防止して，検知した情報を獲得免疫機能に送信する．

3.1.1 エクスプロイトの検知

エクスプロイトの検知は，2.1.1 項で述べた 3 つの手法がある．脆弱性の悪用を検知する手法は，ソースファイルからビルドするときに，Visual Studio の `/GS` コンパイラーオプションなど脆弱性攻撃を検知するセキュリティ機能を有効にする必要があるため，アプリケーションのビルドに依存する．また，攻撃の方法が脆弱性に強く依存するため，従来のパターンマッチングによる検知が有効である．セキュリティ機能の回避を検知する手法は，CVE-2014-0515 などの DEP 回避を必要としない攻撃を検知できない．シェルコード検知の手法は，シェルコード特有のコードのパターンや動作の仕組みに基づいて検知するため，新種のコードを検知できない可能性がある．しかし，シェルコード検知は，上述の 2 つの手法と比べて，検知するための制約が少なく，カバーできるシェルコードの種類が豊富であるため，プロトタイプシステムはエクスプロイトの検知にシェルコード検知を採用する．

シェルコードの検知には，いくつかの手法[1][2]があるが，カバーできるシェルコードの範囲が広いと考えられる SecondDEP を利用する[3]．SecondDEP は，通常の API とシェルコードの API の呼び出し元アドレスの属性の違いに基づいて，シェルコードによる API 呼び出しを検知する．通常のアプリケーションの場合，Windows API の呼び出し元アドレスは，コード領域にあるのに対して，シェルコードの場合，Windows API の呼び出し元アドレスはデータ領域にある．SecondDEP がシェルコードの実行を検知したら，サイバー攻撃に利用された攻撃リクエストを獲得免疫機能に伝える．その後，SecondDEP は，実行の制御を奪われたサーバアプリケーションを終了させる．

3.1.2 DoS 攻撃の検知

DoS 攻撃の検知は，2.1.2 項で分類した 2 種類の攻撃を対象とする．次に述べる方法により，これらの攻撃を検知したら，リクエストデータを獲得免疫機能に送信し，サーバアプリケーションを終了させる．ただし，プロトタイプシステムは，後述の通り，一部の DoS 攻撃に対してリクエストデータではなく，リソースの使用状況に基づいて検知するため，攻撃に利用されたリクエストデータを獲得免疫機能に伝達する機能はない．

1 つ目の DoS 攻撃は，アクセス違反によるサーバアプリケーションの異常終了または無限ループ（シェルコードによる無限ループや実装の不備による無限ループなど）を引き起こす攻撃である．アクセス違反による異常終了は，サーバアプリケーションで例外が発生する．その後，例外が発生したアプリケーションは強制終了される．この例外を取得するために，IMMUNITY.DLL は `AddVectoredExceptionHandler` により例外ハンドラを登録する．もし，アクセス違反による例外が発生したら，その例外ハンドラは獲得免疫機能に攻撃データに使用されたリクエストデータを送信する．無限ループによる応答不能は，サーバアプリケーションがリクエストを受信してから，レスポンスが送信されるまでの時間に基づいて判断する．プロトタイプシステムでは，その時間を 1 秒とし，1 秒以内にレスポンスが送信されないとき，攻撃として検知する．ここから実装方法について述べる．プロトタイプシステムを適用するウ

ェブサーバアプリケーションは，リクエストを recv 関数で受信するため，IMMUNITY.DLL は，recv 関数をフックする．recv のフック関数は，SetThreadpoolTimer 関数によりスレッドにタイマーを設定し，CreateThreadpoolTimer 関数により 1 秒後に実行する関数を登録する．なお，タイマーは，1 秒に設定した．タイマーが作動したら，無限ループによる攻撃と判断し，攻撃に使用されたリクエストデータを獲得免疫機能に送信する．また，1 秒以内にレスポンスが送信されたとき，タイマーを止めるために，レスポンス送信時に呼び出される send 関数をフックして，設定したタイマーを無効にする．

2 つ目のリソースを消費させる DoS 攻撃は，サーバアプリケーションの消費メモリやプロセス数またはスレッド数などリソースの使用量を監視することにより検知する．プロトタイプシステムでは IP アドレス毎のスレッド数を数える．プロトタイプシステムの評価に利用するウェブサーバアプリケーションは，クライアントから接続を要求されたら，accept 関数を呼び出す．IMMUNITY.DLL は，accept 関数をフックし，接続元 IP アドレスが接続禁止リストに含まれるかをチェックする．もし，接続禁止リストに含まれるなら，接続を拒否し，そうでないなら，接続元 IP アドレス毎に，スレッド数を 1 つ加える．また，コネクション切断時に呼び出される closesocket 関数をフックして，サーバアプリケーションが，クライアントの接続を切断したら，スレッド数を 1 つ減らす．スレッド数が閾値を超えたら，接続元 IP アドレスを接続禁止リストへ追加し，これ以降の接続を禁止する．なお，閾値は，性能評価用に 100 に設定した．同時に，攻撃により消費されたスレッドを回復させるためにサーバアプリケーションを終了させる．

3.2 獲得免疫機能

獲得免疫機能は，サーバアプリケーションが受信したリクエストデータを分類器により「正常」または「攻撃」に分類する機能と「正常」と「攻撃」データを学習する機能がある．

3.2.1 分類機能

分類器には，サーバアプリケーションが受信したすべてのリクエストデータが入力されるので，すべてのリクエストデータを分類することになる．これは，自然免疫機能が攻撃を検知する前に，獲得免疫機能により攻撃を検知するためである．

サーバアプリケーションが受信したリクエストデータを取得するために，recv 関数をフックする．recv 関数がリクエストを受信したら，リクエストデータのファジーハッシュ値を分類器に送信する．

ファジーハッシュ値は，入力データが類似していたらファジーハッシュ値も類似する特徴を持つ暗号学的なハッシュ値である．ファッジーハッシュ値の計算には，SSDEEP ライブラリに含まれている fuzzy_hash_buf 関数を用いた．SSDEEP 値は，コロンで区切られた 3 つの部分で構成させる．先頭は，入力データのブロックサイズである．残りの 2 つは，最大 64 文字と 32 文字の合計 96 文字のハッシュ値で構成される．最大文字数に満たない場合，0 でパディングする．

分類器は，これまでの研究結果から，ランダムフォレストを採用した[4]．この分類器の実装は，現在，最も計算速度が速いとされる ranger のランダムフォレストを使う[9]．分類器による分類結果が，「攻撃」に分類されていれば，リクエストデータを「POST /b.htm HTTP/1.1¥r¥n」に書き換えて，リクエストデータを無害化する（b.htm ファイルは存在しないファイルと仮定する）．ウェブサーバには，存在しないファイルをリクエストされたことになるので，ウェブサーバは 404 のレスポンスメッセージを返す処理を行うことになる．

DoS 攻撃を検知するために，サーバアプリケーションはクライアントの接続時に，接続元 IP アドレスが接続禁止リストに含まれているかを確認する．もし，接続禁止リストに含まれるなら，その接続を拒否する．

3.2.2 学習機能

学習機能は，正常データと攻撃データのファジーハッシュ値を分類器に入力して分類器を学習させる．

正常なリクエストを学習させるために，サーバが応答を返せるようになった時点，つまり，send 関数などが呼び出されたとき，正常データであると判断し，そのリクエストデータを学習させる．ただし，獲得免疫機能が攻撃を検知した場合は，404 のレスポンスメッセージを返すため，このリクエストは学習させない．攻撃データは，自然免疫機能が検知したリクエストデータを学習させる．

学習の処理を実施するタイミングは，学習の負荷を軽減するために，自然免疫機能により攻撃を検知したときに，すべてのデータをまとめて学習させる．

この他に，自然免疫機能により送信元 IP アドレスが通知されたら，接続禁止リストに送信元アドレスを登録する機能もある．

3.3 脆弱性のあるサーバアプリケーション

プロトタイプシステムの有効性を調べるために，32 ビットのプロセスで動作する脆弱なウェブサーバアプリケーションを実装した．

ウェブサーバアプリケーションには，2 つの脆弱性がある．1 つはリクエスト行に含まれるリクエスト URI の解釈にバッファオーバーフローの脆弱性があり，任意のコードの実行が可能である．図 3 にその脆弱性を攻撃するリクエストの例を示す．正常なリクエストであれば，リクエスト行とヘッダーには空白が含まれるが，この攻撃リクエスト

には POST メソッドの直後の空白からメッセージボディまで空白がないため，バッファオーバーフローを起こす．緑色の部分にシェルコードのアドレスが配置され，黄色の部分にはシェルコードが配置されている．なお，このシェルコードは正常なクエリ文字列に偽装するため，ASCII コードにエンコードされている．

```
POST /test.htmlHTTP/1.1
Accept:text/html,application/xhtml+xml,*/*
Accept-Language:ja-JP
User-Agent:Mozilla/5.0(WindowsNT6.1; *以下，省略*
Accept-Encoding:gzip,deflate
Host:localhost
DNT:1
Connection:Keep-Alive

q=R%e....äÙÀÙsôXPYIIIICCCCCCQZVTX30VX4AP0A3HH0A00A
BAABTAAQ2AB2BB0BBXP8ACJJIKLM8MRS05PC050MYM5FQYP3TL
KF0VPLK0R4LLKPR24LKBRWXTONWQZGV6QKONLGLE13LS2VL7PI
QHOTMUQYWZBL2QB1GLK1BDPLKPJ7LLKPLTQSHKS0HEQN10QLKQ
I7PS1XSLKW9DXZCWJ79LK04LK5QYF6QKONLYQXODM319WP8KPC
EL6S33MZX7K3MQ4T5KT0XLKV86DS1N3E6LKTLPKLK0X5L5QXSL
K34LKS1XPLIW4VDGTQKQKE169PZV1KOKPQO1O0ZLK22ZKLMQMR
JS1LMK582S0UPUPPP
```

図 3　攻撃リクエストの例

もう 1 つの脆弱性は，バッファオーバーフローのようなバイナリーコードレベルの脆弱性ではなく，開発段階の機能に脆弱性が含まれることを想定して，無限ループを引き起こす架空の DOS メソッドを持つ．リクエスト行に DOS メソッドが指定されたら，無限ループに陥る．

4. プロトタイプモジュールの評価

プロトタイプモジュールの性能を評価するために，獲得免疫機能の学習過程を分析した．また，獲得免疫機能のオーバーヘッドを定量的に評価した．これらの評価では，次のシナリオを想定し，3 番目から計測を開始する．

1. 脆弱性が見つかっていない状態で，サービスが正常に運用され，これまで 200 個の正常なリクエストを受信した（そのリクエストには，64 文字から 256 文字からなるランダムに生成された文字列が含まれる）
2. 攻撃者に脆弱性を攻撃され，任意のコード実行を行うシェルコードが実行されるが，自然免疫機能により，攻撃が検知されて，この攻撃データを学習した
3. 継続的に正常なリクエストと攻撃データを受信するようになる

なお，評価に使用した PC の環境は次の通りである．

- VMware のホスト
  - CPU: AMD Opteron 6234 2.4 GHz ×2 CPUs
  - メモリ: DDR3-PC12800 32GB
  - OS: Debian GNU/Kali Linux（Kernel 4.5.5）
- ウェブサーバ（VMware のゲスト）
  - 仮想 CPU: 1CPU × 4 Cores
  - 仮想メモリ: 4GB
  - 仮想 NIC：仮想 NAT
  - 仮想 OS: Windows 8.1 Education 64bit
  - サーバアプリケーション：オリジナル
- クライアント（ウェブサーバと同一の仮想マシン）

4.1 学習の分析

学習の過程を調べるために，800 個の正常なリクエストと 800 個の攻撃リクエストをランダムに並び替えて検知精度の変化を調べた．攻撃リクエストは，次の 4 つの攻撃をそれぞれ 200 個含む．

- タイプ A
  バッファオーバーフローの脆弱性を悪用してシェルコードによりコマンドを実行するエクスプロイト
- タイプ B
  バッファオーバーフローの脆弱性を悪用してシェルコードにより無限ループを発生させる DoS 攻撃
- タイプ C
  バッファオーバーフローの脆弱性を悪用してジャンクコードによりサーバプロセスを落とす DoS 攻撃
- タイプ D
  開発段階の機能の脆弱性を悪用して無限ループを引き起こす DoS 攻撃

これまでの研究では，上述の攻撃タイプ以外に，スロー HTTP POST DoS 攻撃などリソースを消費させる攻撃（タイプ E）があった．プロトタイプモジュールは，タイプ E の攻撃を，リクエストデータの内容ではなく，送信元 IP アドレスにより判別する．つまり，この攻撃は機械学習の対象ではないため，本章の評価では，この攻撃を除外した．

図 4 に 10 回の試行によって True negative rate（以下，TNR）と True positive rate（以下，TPR），および Accuracy の変化を示す．横軸は，リクエストの送信回数である．なお，正常リクエストと攻撃リクエストの送信順は試行ごとにランダムに変更している．

表 1 に各試行における TNR と TPR，及び Accuracy に関する平均値と分散と最小値，及び最大値を示す．

図 4 から，自然免疫機能が攻撃を検知するたびに，獲得免疫機能の検知率が向上していることがわかる．つまり，モジュールをウェブサーバアプリケーションに適用することにより，攻撃に対するサーバの可用性を改善している．

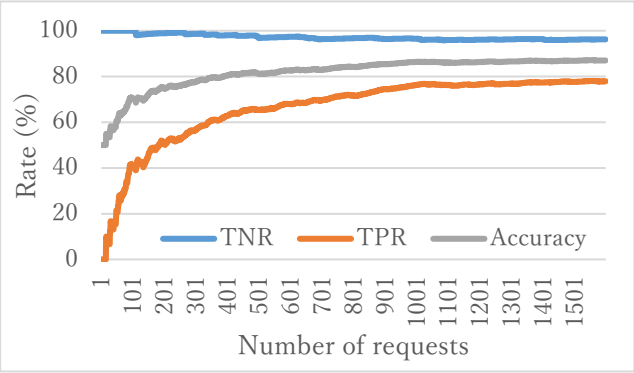

図 4　TNR と TPR，及び Accuracy

Vol.2017-DPS-170 No.25
Vol.2017-CSEC-76 No.25
2017/3/3

表 1　TNR と TPR，及び Accuracy の統計データ

	平均	分散	最大値	最小値
TNR (%)	95.70	1.46	97.50	93.13
TPR (%)	78,95	1,44	81.38	77.38
Accuracy (%)	87.32	0.71	88.88	86.31

4.2 ベンチマークテスト

プロトタイプモジュールのオーバーヘッドを評価するために，ウェブサーバアプリケーションの RTT（Round-Trip Time）を計測した．この計測では，RTT は，存在するファイルに対するリクエストの送信から受信までに要する時間である．

表 2 に，プロトタイプシステムを適用していないウェブサーバアプリケーションとプロトタイプシステムを適用したウェブサーバアプリケーションの RTT について，平均値と分散を示す．なお，これらは，1,000 回の計測で得られた値である．表 2 から，プロトタイプモジュールのオーバーヘッドは約 10%であった．これまでの研究の結果と本稿の結果を比べて，RTT が 2%悪化した．遅くなった原因は，DoS 攻撃機能の追加によりモジュールの処理が増えたためと考えられる．

表 2　RTT の計測時間（秒）

プロトタイプモジュール未適用		プロトタイプモジュール適用	
平均	分散	平均	分散
0.0039887	0.0000002	0.0043869	0.00000004

4.3 学習データの管理

ランダムフォレストはオンライン学習できないため，これまで学習したデータを蓄積する必要がある．これまでに学習したデータとこれから学習するデータをすべて記憶することは，消費メモリと学習時間を増大させる．特に，学習時間の増大は，新規の学習データが分類器に反映されるまでの時間を増大させるため，検出精度を下げる要因となる．蓄積する一方，学習可能なデータの容量を制限しても，検出精度を下げる要因となる．そこで，学習可能なデータの容量と検出精度の関係を調べる．これには，学習データの容量を超えたとき，新規のデータを捨てるか，過去のデータを捨てるかを決める必要がある．前者は，新規に攻撃を学習できないことを意味する．攻撃者が容量を超えたことを何らかの方法で確認できたとき，攻撃者は同一の攻撃を繰り返し行うことにより，サーバアプリケーションの再起動を繰り返させてサービス不能に陥れることができる．一方，後者は，モジュールが捨てたデータを攻撃者が特定することが容易ではないため，前者のような問題は起こりにくい．このことから，学習データの破棄には後者を採用した．学習データの容量が超えた場合は，正常データを古いものから削除する．つまり，学習データのデータ構造はキューとなる．

訓練データとテストデータは，4.1 節で使用したデータと同じ構成（合計 1801 個）である．図 5 に 10 回の施行によって得られた TNR と TPR，および Accuracy の変化を示す．横軸は，キューの長さである．なお，正常リクエストと攻撃リクエストの送信順は試行ごとにランダムに変化する．図 5 から，TNR は，キューが長くなるにつれて，急速に増加し，キューが 901 ぐらいから，飽和した．一方，TPR は，緩やかに減少して，キューが 901 ぐらいから，減少が終わった．キュー長が 901 のとき，キューに含まれる正常データは平均で 729 個，攻撃データは平均で 172 個であった．つまり，データがキューから溢れなかったことがわかる．このとき，学習に要した最大時間は，0.125 秒であった．この学習時間は，サーバアプリケーションの性能に悪影響を及ぼす可能性は小さいと考えられる．また，キュー長が 901 のときの検出精度は，キュー長に制限がないときの検出精度（4.1 節の結果）とほぼ一致している．以上から，保存数の上限は 901 個が適している．

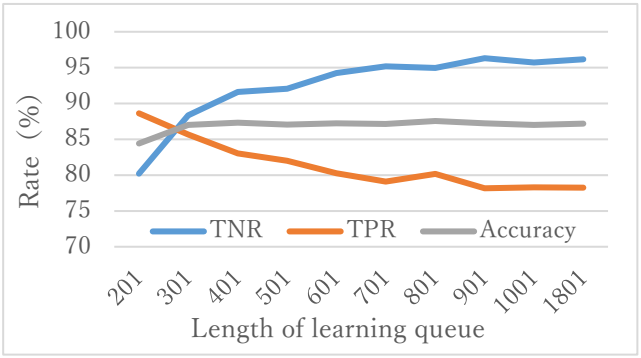

図 5　TNR と TPR，及び Accuracy

学習データの容量が増えるにつれて，TPR が増加すると予測したが，計測結果は，予測と対照的に TPR は減少したため，この原因について考察する．これまでの研究で，攻撃データがタイプ A とタイプ C の 2 種類の場合（キュー長は無制限の場合），最終的な TPR は 85.2%であったため[10]，タイプ B とタイプ D の攻撃が TPR を減少させた原因であると考えられる．そこで，タイプ B とタイプ D の攻撃データと正常データの類似度を SSDEEP で調べた．その結果，タイプ D の攻撃データと正常データの類似度は 0 で，タイプ B の攻撃データと正常データの類似度は 62.14 であった．これらの結果から，タイプ B の攻撃が，正常データとして誤分類され，TPR が減少したと考えられる．

5. 考察

Huang らは，機械学習に対する攻撃を，攻撃による影響，セキュリティ侵害，特異性の属性に基づいて分類した[11]．セキュリティ侵害には，完全性，可用性，プライバシーの侵害がある．

完全性の侵害とは，攻撃データを正常と誤分類させて検

知を回避する侵害行為である．この侵害を行うには，攻撃データを正常に分類した偽装データを分類器に入力して学習させる方法がある．本研究の手法では，分類器はサーバにあり，リモートの攻撃者が分類器を操作できないため，偽装データを学習させることはできない．この他に侵害を行うには，正常データの特徴を模倣して攻撃データを作成する方法がある．本研究の手法は，この侵害方法により，獲得免疫機能から検知を回避される可能性がある．しかし，仮に，回避されても，最終的に自然免疫機能が攻撃を検知する．その結果，獲得免疫機能がその攻撃を学習するので，同じ攻撃であれば，獲得免疫機能により検知される．

可用性の侵害とは，正常データを攻撃と誤分類させてサービスを妨害する侵害行為である．この侵害を行う方法は 3 つある．1 つ目は，正常データを攻撃に分類した偽装データを分類器に入力して学習させる方法であるが，上述の通り，本研究の手法では，攻撃者がこれを行う術がない．2 つ目は，攻撃データの特徴を模倣して正常データを作成する方法である．本研究の手法は，獲得免疫機能が攻撃に分類したら，正常データが遮断される．もし，この細工された正常データが，実際に正当なリクエストと類似していた場合，そのリクエストが遮断される．この問題を解決するために，サーバアプリケーションと同一の実装が動作する仮想の実行環境（サンドボックス）を利用する方法が考えられる．獲得免疫機能がリクエストを攻撃に分類したら，そのリクエストをサンドボックスにも送信し，攻撃でないことを確認できたら，獲得免疫機能にそのリクエストを正常データとして学習させる．3 つ目は，様々な攻撃データを繰り返し送り続けて，学習データを溢れさせることにより，正常データを破棄させ，すべてを攻撃データにする方法である．学習データに正常データが含まれないとき分類器はあらゆるリクエストを攻撃に分類する．この問題を解決するため，学習データに記憶できる攻撃データの個数に上限を設定する方法が考えられる．

プライバシーの侵害とは，分類器の分類結果から，機密情報を推測する侵害行為である．この侵害を行う方法とその対策は今後の課題とする．

6. 関連研究

6.1 免疫系を参考にした研究

免疫系に関連したセキュリティ技術として，Kephart らの人工免疫システムがある[12]．このシステムは，IBM で開発されていたウイルス対策ソフトを基盤にしたシステムである．エンドポイントのウイルス対策ソフトで検知された感染の疑いのあるファイルから，ウイルス検知のための定義ファイルをエンドポイントに配布するまでの一連のシステムが免疫のようであることから，人工免疫システムと呼ばれていた．人工免疫システムは，PC が対象であり，定義ファイルの作成は専門家を必要とする．

Forrest らは，免疫システムのクローン選択説を模擬して，ウイルスやプログラムの異常を検知する手法を提案した[13]．しかし，クローン選択説のように検知するためのデータをランダムに生成するため，人間が悪意を持って作成したウイルスや攻撃を検知できる可能性は低く，まだ実用に至っていない．

Biancaniello らは，免疫システムの自己と非自己を区別する仕組みを参考にし，脆弱性に対する攻撃を検知する手法を提案した[14]．この手法は，正常なアプリケーションから発生されるイベントは自己として，攻撃によりアプリケーションから発生するイベント（マルウェアのダウンロードのようなネットワークイベントなど）は非自己とする．この自己と非自己の違いに基づいて，攻撃を検知する．しかし，この手法は，攻撃を学習する機能がないため，サーバアプリケーションは 2 回目以降の攻撃を未然に防げない．

6.2 適応的な学習による攻撃検知に関する研究

攻撃に対して適応的に学習する侵入検知システム（以降，IDS）がある．従来の IDS は，定義ファイルが更新されるまで，新たな攻撃を検知できない．この問題を解決するために，Danforth は，攻撃を検知するたびに，定義ファイルを更新する[15]．これにより，新しい攻撃に対して適応的に検知できるようになる．Farid らは，ブースティングと単純ベイズ分類器の学習アルゴリズムを改良した IDS を提案した[16]．このアルゴリズムは，訓練データが誤分類されたら，分類器の重みを増やし，訓練データが正しく分類されたら分類器の重みを減らす．この調整は，すべての訓練データが分類器により正しく分類されるまで行うことで，IDS の検出精度が向上する．また，Haque らは，複数のネットワークアナライザーにより収集した攻撃データを複数の分類器に入力し，分類器の検出精度が最も高くなるように分類器のパラメータを調整する IDS を提案した[17]．しかし，これらの手法（[15-17]）は，定義ファイルに基づいているため，定義ファイルに類似していない攻撃を検知することは困難である．

Glickman らや Peng らは，免疫の適応能力を参考にし，機械学習を取り入れた IDS を提案した（[18-19]）．Peng らは，免疫系のネガティブ・セレクションを参考にし，大容量ネットワーク下で，リアルタイム処理可能な IDS を提案した[19]．これらの手法は，定義ファイルを自動的に更新する仕組みがないため，長期の連続稼働には課題がある．

6.3 サーバのレジリエンス強化に関する研究

本研究は，サーバアプリケーションにセキュリティモジュールを適用することにより，サーバアプリケーションのレジリエンスを強化する方法を提案した．本研究の他に，サーバの構成（OS とサーバアプリケーションの実装）を多

様化することによって，サーバアプリケーションのレジリエンスを強化する方法がある．例えば，ウェブサービスのレジリエンスを強化するなら，Windows Server 上で動作する Apache2，Linux 上で動作する NginX などのように，OS とサーバアプリケーションの多様な実装を組み合わせる．多様なマシンの中からいずれかのマシンがサービスを提供し，その他は，待機する．ここで，攻撃を受けサービスが提供できなくなったとき，待機中のマシンに切り替える．これは，1 つの脆弱性がすべての OS やサーバアプリケーションに含まれたことはないという観測に基づく.ただし，共有ライブラリの脆弱性や仕様上の脆弱性は対象外である.

これまで，複数の物理マシンを利用するシステム（[20-21]など）や，ハイパバイザーを利用して複数の仮想マシンを利用するシステム（[22-24]など）が提案されてきた．前者は，導入や運用のコストが大きい．後者は，ハイパバイザーで一元管理できるため，前者と比べて，大幅に導入と運用のコストを削減できる．しかし，それでも，ハイスペックなハイパバイザーを必要とすることと，システムが利用するすべての OS とサーバアプリケーションの設定を行う必要があることから，本研究の手法と比べて，導入のコストが大きいため，導入の大きな障壁となる可能性が予想される.

7. おわりに

本稿は，サイバー攻撃に対してレジリエンスを強化するため，免疫系を模擬したセキュリティモジュールを改良しその性能評価を行った．改良した機能は，DoS 検知機能であり，この改良により，シェルコードによる DoS 攻撃（タイプ B）と意図しない開発段階の機能を悪用した DoS 攻撃（タイプ D）を検知できるようにした．さらに，ランダムフォレストの実装を scikit-learn から， C++言語で実装された ranger に変更することにより，学習と分類の高速化を行った.

プロトタイプシステムの評価では，これまでの評価結果と同様に，リクエストを受信するたびに，学習が進行していることを確認した．ただし，タイプ D の攻撃が原因で TPR が 6.25%減少した．ベンチマークテストでは，改良後でもモジュールのオーバーヘッドがほとんどなかった．また，長期にわたるサーバ運用を想定して，学習データの管理方法を提案して，その有効性を示した.

今後は，Apache2，ISC BIND 9，および Microsoft IIS などの有名なアプリケーションにセキュリティモジュールを適用する.

参考文献

[1] Microsoft Enhanced Mitigation Experience Toolkit 5.52 User Guide. https://www.microsoft.com/en-us/download/details.aspx?id=54265, (参照 2017-02-10).
[2] Cheng, Y. et al. Deng H, ROPecker: A Generic and Practical Approach for Defending against ROP Attack. 2014
[3] Okamoto, T. SecondDEP: Resilient Computing that Prevents Shellcode Execution in Cyber-Attacks. Procedia Computer Science 60:691-669; 2015.
[4] Tarao, M. and Okamoto, T. Toward an Artificial Immune Server against Cyber Attacks : Enhancement of Protection against DoS attacks, KES-2016. 20th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems.
[5] Tarao, M. and Okamoto, T. An implementation and its evaluation of artificial immune security modules for a server application. Computer Security Symposium 2016
[6] CVE-2012-1667，http://www.cvedetails.com/cve/CVE-2012-1667/
[7] CVE-2015-8704，http://www.cvedetails.com/cve/CVE-2015-8704/
[8] CVE-2015-8705，http://www.cvedetails.com/cve/CVE-2015-8705
[9] Wright, M. N, Ziegler, A. ranger: A fast implementation of random forests for high dimensional data in C++ R, Journal of Statistical Software, in press. https://arxiv.org/abs/1508.04409
[10] Tarao, M and Okamoto, T. An Artificial Immunity-Enhancing Module for Inter Servers against Cyber Attacks, AROB2017, 22nd International Symposium on Artificial Life and Robotics.
[11] Huang, L. et al. Adversarial machine learning. In: Proc. of the 4th ACM workshop on Security and artificial intelligence, ACM, pp. 43-58, 2011.
[12] Kephart, JO. A Biologically Inspired Immune System for Computers. In: Proc. of the 4th International Workshop on the synthesis and simulation of living systems, Artificial Life IV. p. 130-139, 1994.
[13] Forrest, S. et al. A Sense of Self for Unix Processes, In: Proc. of IEEE Symposium on Security and Privacy; p. 120-128, 1996.
[14] Biancaniello, P. et al. AIS: A Framework for Adaptive Immune Response for Cyber Defence, 2011.
[15] Danforth, M. WCIS: A Prototype for Detecting Zero-Day Attacks in Web Server Requests, Proceeding of the 25th international conference on Large Installation System Administration. p. 21-14, 2011.
[16] Farid, D. M., Rahman, M. Z., and Rahman, C. M. Adaptive intrusion detection based on boosting and naive bayesian classifier. International Journal of Computer Application, 24.3, 12-19, 2011,
[17] Haque, M. E., Alkharobi, T. M, Adaptive Hybrid Model for Network Intrusion Detection and Comparision among Machine Learning Algorithms, International Journal of Machine Learning and Computing 5.1, 2015.
[18] Glickman, M. et al. A Machine Learning Evaluation of an Artificial Immune System. Evolutionary Computation; 2005.
[19] Peng, L. et al. Dynamically Real-Time Anomaly Detection Algorithm with Immune Negative Selection, Applied Mathematics & Information Sciences 7.2; 2013.
[20] Reynolds, J. et al. The design and implementation of an intrusion tolerant system, In: Proc. of International Conference on Dependable Systems and Networks, pp. 258-290, 2002.
[21] Arsenault, D. et al. Secure, resilient computing clusters: self-cleansing intrusion tolerance with hardware enforced security (SCIT/HES), In: Proc. of the 2nd International Conference on Availability, Reliability and Security, pp. 343-350, 2007.
[22] Bangalore, A. K. et al. Securing Web servers using self-cleansing intrusion tolerance (SCIT), In: Proc. of the 2nd International Conference on Dependability, pp. 60-65, 2009.
[23] Heo, S. et al. A novel intrusion tolerant system based on adaptive recovery scheme (ARS), IT Convergence and Security 2012, Springer, pp. 71-78, 2013.
[24] Sano, F. et al. A Cyber Attack-Resilient Server Using Hybrid Virtualization, Procedia Computer Science, Vol. 96, pp. 1627-1636, 2016.