サーバアプリケーション用の免疫系を模擬したセキュリティモジュールの実装と性能評価

多羅尾　光宣†　　岡本　剛†

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回復力のあるサーバアプリケーションのためのセキュリティモジュールを開発することを目的とする．セキュリティモジュールは，免疫系を模擬した仕組みで動作し，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・既知のサイバー攻撃を検知する．獲得免疫機能は，自然免疫機能が検知した攻撃を学習し，2 回目以降の攻撃を検知する．これまでの研究で，シミュレーション評価により，機械学習が高い検出精度であることを確認している．本論文では，シミュレーション評価ではなく，実際に攻撃することにより，検出精度とセキュリティモジュールのオーバーヘッドを計測し，その結果を報告する．

キーワード：免疫系，機械学習，エクスプロイト，シェルコード，DoS 攻撃

An implementation and its evaluation of artificial immune security modules for a server application

Mitsunobu Tarao†　　Takeshi Okamoto†

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 artificial immune security modules for mission-critical servers against cyber attacks on the Internet. Similar to the human immune system, the security modules consist of innate and adaptive immune functions. The innate immune function detects cyber-attacks on a known or unknown vulnerability. The adaptive immune function learns the cyber-attack detected by the innate immune function and prevents a second cyber-attack before the innate immune function. Our previously described simulation results showed that a random forest classifier is suitable for the adaptive immune function. This paper reports the implementation and its evaluations of the artificial immune security modules.

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

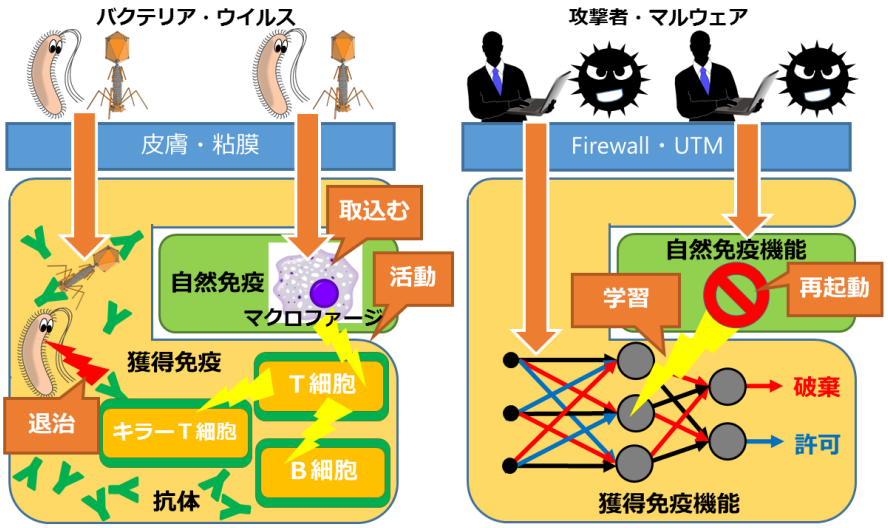
1. はじめに

IoT 時代の幕開けとともに，ミッションクリティカルサーバの需要が高まりつつある．ミッションクリティカル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，ミッションクリティカルサーバにおいて，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

† 神奈川工科大学
Kanagawa Institute of Technology



(a) 生物の免疫系　　(b) 免疫を模擬したセキュリティモジュール

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]

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．これまでの研究により，サーバアプリケーションが受信したリクエストデータのファジーハッシュ値と攻撃に利用されたリクエストデータのファジーハッシュ値の類似度を計算することにより，特定の攻撃であれば，適応的に攻撃を検知できることを確認した[4]．しかし，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た．また，シミュレーションにより機械学習が誤検知を減らせることを確認した[5]．そこで，本稿では，シミュレーションではなく，機械学習の機能をセキュリティモジュールに実装し，機械学習の学習過程を可視化するとともに機械学習のオーバーヘッドを定量評価して，免疫を模擬したセキュリティモジュールの有効性を示す．

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食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．

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，三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，プログラムの実行の制御を奪う (2) DEP などの OS のセキュリティ機能を回避する (3) シェルコードと呼ばれる小さなプログラムコードを実行し，目的の処理を行う．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．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，3 種類に分類できる．1 つ目は，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である (例えば BIND 9 の CVE-2012-1667, CVE-2015-8704, CVE-2015-8705 などがある)．2 つ目は，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である．3 つ目は，DNS や NTP を利用したリフレクション攻撃で，莫大なリクエスト数をサーバに送信し，ネットワーク帯域を大量に消費させる攻撃である．

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．

2.2 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．

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．自然免疫機能は IMMUNITY.DLL と MONITORING.EXE 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．獲得免疫機能は Python で実装した classifier.py モジュールによって動作する．図 2 にプロトタイプシステムの構成を示す．水色の矢印は通信によるデータ交換を表す．オレンジ色の矢印はプロセスの操作を表す．Python モジュールと他のモジュールは UDP によりデータを共有する．

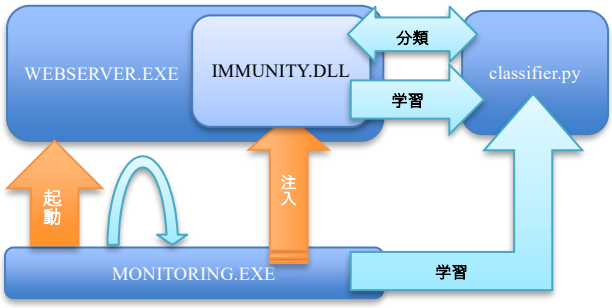


図 2　プロトタイプシステムの構成

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 と MONITORING.EXE の 2 つのモジュールに含まれる．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，攻撃に使用されたリクエストデータを UDP で獲得免疫機能に通知する送信機能もある．図 3 に自然免疫機能の構成を示す．オレンジ色の楕円はプロセス内部の機能を表す．赤い矢印はプロセス内部のデータ交換を表す．

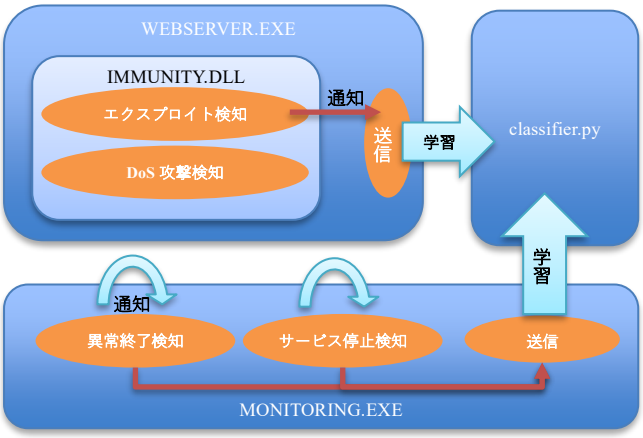


図 3　自然免疫機能の構成

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伝達し，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，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，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．異常終了による応答不能は，サーバアプリケーションの親プロセス MONTORING.EXE が WaitForSingleObject 関数でサーバアプリケーションのプロセスを監視することにより検知される．ただし，エラー報告アプリケーション（WerFault.exe）によるサーバアプリケーションの中断を防止するため，サーバアプリケーションの起動時に，IMMUNITY.DLL が SetErrorMode 関数により，エラー報告アプリケーションを起動しないようにする（通常，アプリケーションに異常が発生したら，Windows はエラー報告アプリケーションを起動し，アプリケーションの実行を中断する）．もし，サーバアプリケーションが終了したら，MONITORIING.EXE は，サーバアプリケーションを起動する．無限ループによる応答不能は，MONITORING.EXE がサーバアプリケーションに 1 秒ごとにリクエストを送信してレスポンスを受信できることを確認することにより検知する．

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．ただし，本研究の設計では，接続禁止リストによる接続禁止機能は学習機能に含まれるので，獲得免疫機能にあるべきであるが，プロトタイプシステムでは，実装を簡単にするため，自然免疫機能のモジュールに含まれる．

3.1.3 送信機能

攻撃のリクエストを学習するために，自然免疫機能は，これらの攻撃に利用されたリクエストのデータを獲得免疫機能に送信する（リソースを消費させる攻撃を除く）．送信するデータ形式は，先頭 1 バイトにデータの種別を表す値（0 が正常，1 が攻撃，255 が未分類のリクエストを表す）があり，その後にリクエストデータが続く．

リクエストデータは，受信したリクエストのリクエスト行からメッセージボディまでのデータを含む．ただし，プロトタイプシステムは，この送信に要する時間を短くするために，リクエストデータのファジーハッシュ値を獲得免疫機能に送信する．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．

これまでの研究により，ファジーハッシュ値の機械学習により高い検出精度が得られることがわかっているが，ファジーハッシュ値以外の入力データを試行できるようにするために，送信機能はファジーハッシュ値ではなくリクエストデータそのものを獲得免疫機能に送る設計を検討する必要がある．

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」に分類する機能と自然免疫機能から受信したリクエストデータを学習する機能がある．

3.2.1 分類機能

分類機能は，機械学習の分類器とリクエストデータを分類器に送信して，分類結果に応じて，リクエストを破棄する検査機能から構成される．図 4 に分類機能と分類器のやりとりを示す．

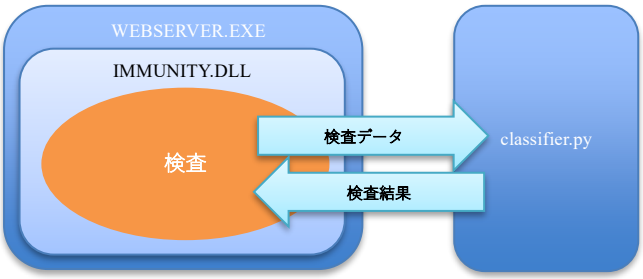


図 4　分類機能と分類器のやりとり

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送信されるので，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（深部で攻撃データが処理される前）に獲得免疫機能により攻撃を検知するためである．

分類器は，これまでの研究結果[5]から，ランダムフォレ

ストを採用した. この分類器の実装は, Python の scikit-learn パッケージに含まれる RandomForestClassifier を使用する. 分類器は 3.1.3 項で述べたデータ形式にしたがってリクエストデータを UDP で受信し, 分類結果を送信する. データの種別は分類前のため, 未分類を表す「255」を設定する.

検査機能は, 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, 検査機能は, データ種別を「255」に設定してリクエストデータのファジーハッシュ値を分類器に UDP で送信する. 次に, 分類器から分類結果を受信し, 「1」として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.

3.2.2 学習機能

学習機能は, 自然免疫機能の送信機能から送信されたデータを UDP で受信し, データ種別に応じて 96 文字からなる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.

ランダムフォレストは, 逐次学習に対応していない. そのため, これまで受信したすべてのリクエストデータを蓄積し, 学習データが追加されるたびに, すべての学習データを学習しなければならない. scikit-learn パッケージには逐次学習に対応した分類器が複数あるが, これまでの研究[5]により, いずれも検出精度が著しく低かったため, 速度より検出精度を優先して, ランダムフォレストを選択した.

3.2.3 送信機能

正常なリクエストも学習させるために, サーバが応答を返せるようなった時点, つまり, send 関数などが呼び出されたとき, リクエストが正常に処理され, 攻撃データではないと判断し, このリクエストデータを学習させる. そのリクエストデータは, send 関数をフックして, send 関数がレスポンスを送信したら, そのリクエストデータを正常なリクエストとして獲得免疫機能に送信する. ただし, 上述の分類機能により攻撃を検知した場合でも, send 関数が呼び出されることもある. そのため, 獲得免疫機能で攻撃を検知した場合は, send 関数が呼び出されても, 学習用のリクエストデータを送信しない. データの送信形式は, 3.1.3 項と同様である.

3.3 脆弱性のあるウェブ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 5 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
JS1LMK582S0UPUPPP3XVQLKBOK7KO8UOKZPH592PVE8OVJ5OMM
MKO8UGLEV3LEZMPKKM0CEDEOKQW5CD2BO3ZC0F3KO8UU351BLB
CFNBECHCU5PAA
```

図 5　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. プロトタイプシステムの評価

プロトタイプシステム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て学習した（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, このリクエストデータを攻撃データとして学習した
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/Linux Jessie（Kernel 3.16.0）
- ウェブサーバ（VMware のゲスト）
  - 仮想 CPU: 1CPU × 2 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
  - Python: 2.7.6（scikit-learn 0.17.1）
- クライアント（ウェブサーバと同一の仮想マシン）

4.1 学習過程の分析

学習の過程を調べるために，200 個の正常なリクエストと 200 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 2 つの攻撃をそれぞれ 100 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 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

これまでの研究では，上述の攻撃タイプ以外に，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引き起こす DoS 攻撃（タイプ B）と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（タイプ D），およびスロー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．プロトタイプシステムは，タイプ B とタイプ D に対して，リクエストデータの情報を手掛かりに攻撃を判別するのではなく，攻撃元 IP アドレスの照合により判別している．つまり，これらの攻撃はすべて機械学習に関係しないため，学習過程の評価では，これらの攻撃を除外した．

図 6 に 10 回の試行によって得られた平均真陽性率（True positive rate: TPR）と平均真陰性率（True negative rate: TNR），および正解率（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 TPR と TNR，及び正解率に関する平均値と分散と最小値，及び最大値を示す．

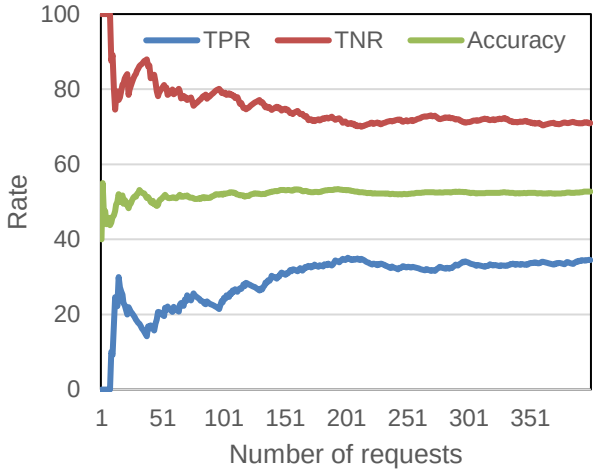


図 6 TPR と TNR，及び正解率

表 1 TPR と TNR の統計値

	平均	分散	最小値	最大値
TPR（%）	34.45	64.13611	29.5	56.5
TNR（%）	70.95	64.13611	66.0	93.0

シミュレーション評価では，各試行の平均値は，TPR が 60.7%，TNR が 96.7%であったが，実機での計測では著しく低い値となった．この原因は，2 つほどの要因が考えられる．1 つは，シミュレーション評価と本稿の評価に利用したデータが異なることである．シミュレーション評価では，4 つのタイプの攻撃データをシャッフルし，正常なリクエストを 400 個と各攻撃タイプに 100 個のリクエストの合計 800 個を使用した．つまり，学習可能なデータの種類と量の両方が 2 倍多い．ただし，データの量に関しては，図 6 に示した通り，300 回目のリクエスト付近で各指標が頭打ちになっているので，量による影響は小さいかもしれない．もう一つは，各試行の TPR と TNR の分散が大きいことである．ばらつきが大きくなる要因は，テスト前にランダムに生成した初期の学習データの影響が考えられる．今後は，初期の学習データの影響を詳細に分析する必要がある．

4.2 ベンチマークテスト

プロトタイプシステム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．プロトタイプシステムは，リクエストを受信するたびに，分類と学習を行う仕組みであり，ベンチマークテストでは，すべてのリクエストが正常であるため，受信するたびに学習することになる．

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，100 回の計測で得られた値である．表 2 からプロトタイプシステムに適用により，RTT が 1.85 倍程度遅くなることがわかった．遅くなる原因は，エクスプロイト検知に利用した SecondDEP と分類器による分類と学習の処理が考えられる．SecondDEP のオーバーヘッドは PCMark のベンチマークによれば 1%程度であった[3]がマイクロ秒レベルで評価すると無視できないオーバーヘッドになっているかもしれない．また，シミュレーション評価で，ランダムフォレストは，学習に約 0.04 秒も要していたことから[5]，学習時間が大きな原因であると考えられる．学習時間による遅延を軽減するために，分類処理と学習処理を並行させ，サーバの負荷が大きくないときに学習させ，負荷が大きいときは，学習を遅延させる回避策が考えられる．

表 2　RTT の計測時間（秒）

プロトタイプシステム未適用		プロトタイプシステム適用	
平均	分散	平均	分散
0.008975	6.24×10^{-6}	0.016628	4.68×10^{-5}

5. 関連研究

免疫系に関連したセキュリティ技術として，Kephart ら[6]の人工免疫システムがある．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したが，提案手法はサーバが対象であり人手を必要としない．

Forrest ら[7]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．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．

この他に，免疫系を参考にした侵入検知システム（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欠点があった．この欠点を克服するために，Danforth[8]は，攻撃を検知するたびに，IDS 自身の定義ファイルを更新する．これにより，新しい攻撃に対して適応的に検知率が向上する．しかし，この手法は定義ファイルに基づいた検知であるため，定義ファイルに類似していない攻撃は検知が困難である．これらの手法に対して，提案手法は，定義ファイルではなく，シェルコードの挙動に基づいて検知するため，定義ファイルを必要としない．

Glickman ら[9]や Peng ら[10]は，免疫の適応能力を参考にし，機械学習を取り入れた侵入検知システムを提案した．Peng ら[10]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侵入検知システムを提案した．

6. 考察と今後の課題

プロトタイプシステムの実装や評価を終えて見えてきた今後の課題について考察する．

6.1 機械学習に対する攻撃とその対策

人工知能が予期せぬ学習や予測が問題になることがある．Interlop Tokyo 2016 の基調講演「人工知能の敵」でも人工知能のセキュリティが話題になった．機械学習をセキュリティ技術に応用するときも，ユーザや管理者の意図しない学習や予測が考えられる．したがって，誤った学習や予測を誘引する攻撃に対する耐性を評価する必要がある．この攻撃は 2 種類考えられる．

1 つは，攻撃データに類似した正常なリクエストを少しずつ，攻撃データとの類似性が高いリクエストに書き換えながら繰返し送ることにより，攻撃データを正常なリクエストと誤分類させる．しかし，提案手法には，自然免疫機能があるため，たとえ，獲得免疫機能をすり抜けても，最終的には自然免疫機能で攻撃を防げる．さらに，この時点でこの攻撃リクエストを「攻撃」として学習するので，獲得免疫機能は，2 度目以降の同じ攻撃に対して，検知できる可能性がある．

もう 1 つは，正常なリクエストと類似性の高い攻撃データを少しずつ書き換えながら繰り返し送ることによって，正常なリクエストを攻撃データと誤分類させて，サービスを不能にする攻撃が考えられる．このような攻撃は，多くの場合，攻撃データをランダムに生成する手口が予想される．仮に，正常なリクエストを誤分類させることができても，実際には，サービスを提供していないリクエストの可能性が高い．例えば，ウェブサーバであれば，存在しないファイルに対するリクエストなどが予想される．しかし，実在するファイルやサービスに対して正常なリクエストを誤分類する可能性は否定できない．そこで，分類器によって攻撃と分類されたら，サンドボックスのような安全性が確保された仮想環境で，分類結果の信頼性を確認する仕組みが考えられる．

6.2 高負荷時のベンチマーク

ベンチマークに利用したウェブサーバアプリケーションは，エクスプロイト攻撃を受けられることを目的に実装したため，可用性の高い設計になっていない．そのため，本稿では高負荷時のベンチマークは行わなかったが，ミッションクリティカルなサーバは可用性が極めて重要である．今後は，プロトタイプシステムが高負荷時でも可用性を維持できることを示す必要がある．

高負荷時のベンチマークでは, 4.2 節で述べたように学習処理がボトルネックになることが予想される．プロトタイプシステムは Python を利用したが, Python のグローバルインタプリタロックが原因で，シングルプロセスでマルチコアを活用できない欠点がある．今後は，classifier.py の機能を C/C++言語で実装することが考えられる．

6.3 機械学習による検知対象の拡大

プロトタイプシステムの DoS 攻撃検知は，リソースの使用状況に基づくため，攻撃を検知したときに，攻撃の引き金となったリクエストを特定できなかった．今後は，リソースの使用状況だけでなく，攻撃に利用されたリクエストが特定できるように，リクエストに対応したレスポンスが送信されるまでに攻撃を検知する仕組みを導入する必要がある．例えば，文献[5]のタイプ B とタイプ D は，リクエストの受信からレスポンスの送信までの時間を監視する方法が考えられる（受信から送信までに 5 秒以上要するなら，攻撃として検知する）．このような方法により，いくつかの DoS 攻撃について，その攻撃に利用されたリクエストを特定でき，獲得免疫機能で学習・分類させることができる．

7. おわりに

本稿は，サーバアプリケーションの可用性を確保するために，免疫系を模擬したセキュリティモジュールを実装しその性能評価を行った．プロトタイプシステムの実装は，自然免疫機能と獲得免疫機能から構成される．自然免疫機能では，エクスプロイトの検知に SecondDEP を利用し，DoS 攻撃の検知にリソースの監視を行う．獲得免疫機能では，Python の scikit-learn パッケージに含まれる分類器 RandomForestClassifier を使って，正常なリクエストと自然免疫機能で検知した攻撃リクエストを学習する．

プロトタイプシステムの評価では，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が，シミュレーション評価よりも検出精度が低かった．また，ベンチマークテストでは，提案手法のオーバーヘッドが約 54%もあった．このことから，サーバの負荷が大きくないときに学習させ，負荷が大きいときは，学習を遅延させるなどの回避策を行う必要がある．

今後は，シミュレーション評価と同等以上かそれ以上の検出精度に改善することや DoS 攻撃の学習機能の実装，さらには機械学習に対する攻撃の対策などが考えられる．

参考文献

[1] Microsoft, Enhanced Mitigation Experience Toolkit 5.5 User Guide. https://www.microsoft.com/en-us/download/details.aspx?id=50766, (参照 2016-08-14).

[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, *In: Proc. of the 21st International Symposium on Artificial Life and Robotics;* p. 36-39, 2016.

[5] 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.

[6] 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.

[7] Forrest, S. et al.. A Sense of Self for Unix Processes, *In: Proc. of IEEE Symposium on Security and Privacy*; p. 120-128, 1996.

[8] Melissa, D. 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.

[9] Matthew, G. et al.. A Machine Learning Evaluation of an Artificial Immune System. Evolutionary Computation; 2005.

[10] Lingxi, P. Dynamically Real-Time Anomaly Detection Algorithm with Immune Negative Selection, Applied Mathematics & Information Sciences 7.2; 2013.