Vol.2018-DPS-174 No.3
Vol.2018-CSEC-80 No.3
2018/3/5

ISC BIND9 のため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[†1]　岡本　剛[†1]

概要：本研究は，これまで脆弱性へのサイバー攻撃に対するサーバアプリケーションの可用性を向上するため，適応的にサイバー攻撃を検知するセキュリティモジュールを提案してきた．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，エクスプロイトを検知する機能と DoS 攻撃検知機能を有する．獲得免疫機能は，自然免疫機能が検知した攻撃を機械学習により学習し，2 回目以降の類似の攻撃を未然に防ぐ．これまでの研究で，独自のウェブ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し，エクスプロイトと DoS 攻撃を適応的に検知できることを確認した．本稿では，セキュリティモジュールの有用性を評価するために，広く普及している BIND9 に適用可能なセキュリティモジュールを実装して，脆弱性（CVE-2015-5477 と CVE-2016-2776）に対して攻撃を行い，モジュールの有効性を評価した．その結果，96.87%の検出精度（True negative rate: 99.25%，True positive rate: 94.49%）が得られたが，125.66%のオーバーヘッドがあった．

キーワード：BIND9，免疫系，侵入検知，脆弱性，ランダムフォレスト，サーバの可用性

An implementation and its evaluation of an artificial immune-security module for ISC BIND9

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

Abstract: This study focuses on an immunity-enhancing security module for Internet servers against cyberattacks. The security module consists of innate and adaptive immune functions. The innate immune function detects known and unknown cyberattacks, whereas the adaptive immune function learns and detects the cyberattacks detected by the innate immune function. Our previous paper showed that the security module was applied to our original web server application, thereby adaptively detecting known and unknown cyberattacks. In this paper, we implemented the module for ISC BIND9, which is the most used DNS server application on the Internet, and we evaluated the detection accuracy of BIND9 with the module by exploiting two known vulnerabilities: CVE-2015-5477 and CVE-2016-2776. The results showed that the detection accuracy of the module was 96.87% (true negative rate: 99.25%, true positive rate: 94.49%), whereas the overhead of the module was 125.66%.

Keywords: BIND9, immune system, intrusion detection, vulnerability, random forest, server availability

1. はじめに

サーバの可用性を維持する重要な機能の 1 つに，フォールトトレランス設計があるが，広く普及しているフォールトトレランス設計は，同一構成の冗長化であり，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．サイバー攻撃に対応するために，多様なサーバアプリケーションによりサービスを冗長化するアプローチ[1][2][3][4]があるが，多様なサーバアプリケーションをホストするには多くの物理リソース（物理サーバマシン，物理プロセッサ，物理メモリなど）を要することや導入と管理の工数が多いため，ほとんど普及していない．

サイバー攻撃に対して，可用性を維持するために，ネットワークトラフィックを監視して，攻撃を検知・防止するシステム（IDS など）がある．これらの商用システムの多くは，シグネチャマッチングに基づくため，ゼロデイ攻撃など未知の攻撃を検知することが難しい．未知の攻撃にも対応するため，機械学習を活用する手法が数多く提案されている[5][6][7]．しかし，これらの手法が攻撃を検知するには，正常データや攻撃データを大量に学習させる必要がある．攻撃データに関しては，ゼロデイ攻撃など，事前に攻撃データを収集することが難しいこともある．正常データに関しては，大量のデータがすべて正常であることを保証することが難しいケースも想定される．

アプリケーションの挙動を監視して，未知の脆弱性に対する攻撃を検知・防止する手法がある．例えば，国内企業の FFRI や米国企業の Palo Alto Networks などが未知の脆弱性に対する攻撃を検知する技術を提供している．学術研究機関でも，kBouncer[8]，ROPecker[9]や SecondDEP[10]など，先進的な技術開発が盛んである．しかし，これらの技術は，攻撃を検知した時点で，アプリケーションは正常な実行の制御を失っているため，アプリケーションを強制的に終了せざるを得ない．アプリケーションを再起動させても，同様のサイバー攻撃により，アプリケーションは再び停止させられるため，サービスの可用性を維持できない．

そこで，本研究では，サイバー攻撃に対するサーバアプ

† 神奈川工科大学

Vol.2018-DPS-174 No.3
Vol.2018-CSEC-80 No.3
2018/3/5

リケーションの可用性を維持するために，以下の要件を満たすセキュリティモジュールを開発してきた．

- 物理リソースを追加しなくても動作する（ソフトウェアモジュールを追加するだけでよい）．
- 事前に攻撃データを必要としない．
- 事前に大量の正常データを必要としない．
- 稼働中に攻撃データを学習することにより，類似の攻撃であれば未然に検知・防止できる．

このモジュールは，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 SecondDEP などにより，未知や既知の脆弱性に対する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データを学習することにより， 2 回目以降の攻撃を未然に防ぐ．これまでの研究では，2 つの脆弱性がある独自のテスト用ウェブ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して，その有効性を評価した結果，TPR（True Positive Rate）が 78.95%，TNR（True Negative Rate）が 95.70%であり，モジュールのオーバーヘッドは，9.98%であった[11]．そこで，本稿では，モジュールの有用性を評価するため，広く普及している ISC BIND9 に適用可能なセキュリティモジュールを実装し，その有用性を評価した結果を報告する．本稿で明らかにする内容は以下の通りである．

- テスト用ウェブサーバアプリケーションと同等以上の検出精度 96.87%（TNR：99.25%，TPR：94.49%）が得られた．
- モジュールのオーバーヘッドが 125.66%であり，テスト用ウェブサーバアプリケーションより 115.68 ポイント大きくなった．
- BIND9 の内部関数（ユーザ定義関数）をフックする必要があったため，ソースコードの解析とバイナリの静的解析が必要であった．

2. 免疫系を模擬したセキュリティモジュール

免疫系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機能に相当する．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，免疫系の免疫記憶に相当する．獲得免疫系は，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対応関係を示す．

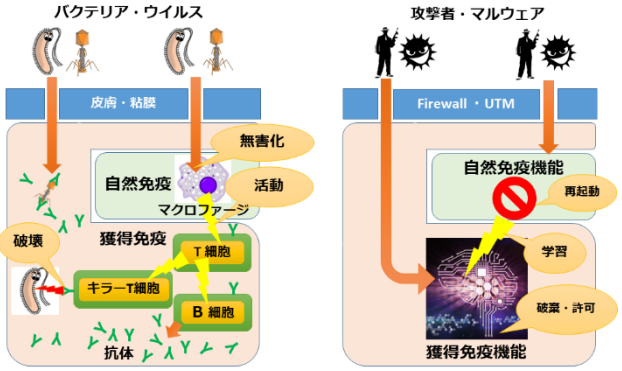

図 1 生物の免疫系とセキュリティモジュールの対応関係

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，獲得免疫機能に攻撃を検知したことを伝え，新たにサーバアプリケーションのプロセスを生成し，攻撃を受けたプロセスを終了させる．

本研究が対象にするサイバー攻撃は，実装上の脆弱性に対する攻撃とする．具体的には，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) バッファー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．

この実行手順に基づいて，検知手法には，(1) に対して，脆弱性攻撃の検知，(2) に対して，OS のセキュリティ機能回避の検知，(3) に対して，シェルコードの検知の 3 つがある．脆弱性攻撃の検知手法には，Snort などのシグネチャマッチングやカナリアなど脆弱性攻撃固有の検知・防止手法がある．セキュリティ機能回避の検知手法には，ROPGuard[8]や ROPecker[9]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[10]がある．エクスプロイトの検知には，これらのすべての検知機能かいくつかの検知機能を利用する．

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．

タイプⅠ 脆弱性を悪用してサーバアプリケーションを停止させる．
タイプⅡ SYN フラッド攻撃や HTTP POST DoS 攻撃など，サーバアプリケーションのリソースを枯渇させる．
タイプⅢ DNS や NTP などを悪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枯渇させる．

タイプⅠ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ある．

タイプⅡ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．

タイプⅢ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 攻撃対策サービスや DOTS（DDoS Open Threat Signaling）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攻撃を学習する学習機能と，サーバアプリケーションが受信したリクエストを「正常」または「攻撃」に分類する分類機能がある．これらの機能により，サーバアプリケーションは適応的にサイバー攻撃を検知する．

2.2.1 学習機能

獲得免疫機能は，正常なリクエスト（正常データ）と攻撃に利用されたリクエスト（攻撃データ）を機械学習により学習する．機械学習は，教師あり学習と教師なし学習に分けられる．教師あり学習には，決定木や SVM などがあり，教師なし学習には，K 平均法や混合ガウスモデルなどがある．本研究のデータは「正常」と「攻撃」の 2 種類であり，自然免疫機能から教師ありデータが得られるため，教師ありの機械学習を利用する．

獲得免疫機能が攻撃データを学習するタイミングは，自然免疫機能が攻撃を検知したときである．一方，正常データを学習するタイミングは，サーバアプリケーションが応答を送信する直前である．サーバアプリケーションが応答を送信できるなら，受信したリクエストは正常であったと判断するからである．

2.2.2 分類機能

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，そのリクエストを「正常」または「攻撃」に分類する．

獲得免疫機能がリクエストを「攻撃」に分類したら，獲得免疫機能は，そのリクエストを破棄または無害化する．この処理により，攻撃データが脆弱性に到達する前に破棄（無害化）されるので，サーバアプリケーションを再起動させることなくサービスを継続できる．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．一方，獲得免疫機能がリクエストを「正常」に分類したら，サーバアプリケーションにリクエストの処理を継続させる．

3. プロトタイプの実装

免疫系を模擬したセキュリティモジュールの有用性を評価するために，広く普及している DNS サーバの実装である ISC BIND9 に適用可能なモジュールのプロトタイプを実装した．

プロトタイプは，2 つモジュール（IMMUNITY.DLL と MONITORING.EXE）から構成される．自然免疫機能と獲得免疫機能は IMMUNITY.DLL によって提供される．IMMUNITY.DLL は，BIND9 の内部で動作するように MONITORING.EXE によって注入される．また，MONITORING.EXE は，BIND9 のプロセスが強制終了したら，BIND9 を再起動させる．図 2 にプロトタイプの構成を示す．なお，プロトタイプは，スタブリゾルバとフルサービスリゾルバ間の UDP プロトコルを対象とする．TCP プロトコルやフルサービスリゾルバと権威サーバ間の通信は，今後の課題とする．

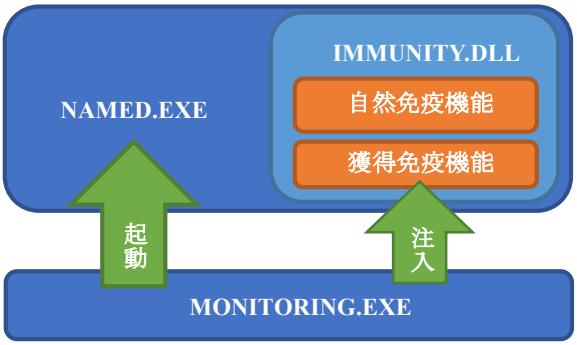

図 2　プロトタイプの構成

3.1 クエリとレスポンスの取得

セキュリティモジュールは，攻撃を学習して検知するため，クエリとそのレスポンスを取得する必要がある．Windows の BIND9 は I/O 完了ポートの仕組みを利用して，非同期にクエリの受信とレスポンスの送信を行っている．クエリの受信には，Windows API の WSARecvFrom を使用するが，I/O 完了ポートでは，WSARecvFrom はネットワークデバイスへ I/O リクエストを発行するだけで，受信を完了する前にその処理を終了する．つまり，WSARecvFrom をフックしても，フック関数においてクエリを取得できない．受信の完了は，Windows API の GetQueuedCompletionStatus により知ることができる．しかし，GetQueuedCompletionStatus は，受信のみならず，送信などあらゆる I/O リクエストの完了を待ち受けるため，GetQueuedCompletionStatus からクエリを取り出すには，数多くの処理を必要とする．

そこで，プロトタイプでは，Windows が標準で提供している API からクエリを取り出すのではなく，BIND9 が定義した内部関数(ユーザ定義関数)から取り出すことにした．BIND9 のソースコードによれば，GetQueuedCompletionStatus による I/O リクエストの待ち受け後に，受信したクエリは最終的に BIND9 の内部関数 send_recvdone_event に引き渡されるので，プロトタイプは send_recvdone_event をフックし，フック関数でクエリを取得する．send_recvdone_event はエクスポート関数ではないため，BIND9 の PE ファイルを IDA Pro で静的解析して，se

nd_recvdone_event のアドレスを特定して，そのアドレスをハードコーディングした上で，フックすることにした．ただし，send_recvdone_event のアドレスは，セキュリティの修正パッチやバージョンアップなどにより BIND 9 の PE ファイルが更新されたとき，そのアドレスが変化する可能性が高いため，別の方法を検討する必要がある．

クエリを取得したら，自然免疫機能と獲得免疫機能がクエリを特定できるように，クエリのトランザクション ID をキーとするハッシュテーブルにクエリデータを登録する．なお，異なるクエリであるにもかかわらず，トランザクション ID が同一のこともあるので，ハッシュテーブルのキーの管理に改良が必要である．

一方，レスポンスの送信には Windows API の WSASendTo を使用する．WSASendTo の引数にはレスポンスデータが含まれるので，WSASendTo をフックすれば，そのフック関数によりレスポンスを取得できる．

セキュリティモジュールは，2.2.1 項で述べたように，レスポンスを送信できるようになった時点で，そのレスポンスに対応したクエリを正常データとして学習する．つまり，レスポンスからそれに対応したクエリを特定する必要がある．プロトタイプでは，DNS のクエリとレスポンスのトランザクション ID が同じである仕様に基づいて，レスポンスのトランザクション ID からクエリを特定することにした．

3.2 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．これらの機能が攻撃を検知・防止したら，攻撃に利用されたクエリデータを獲得免疫機能に渡す．

3.2.1 エクスプロイトの検知

エクスプロイトの検知には，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，従来のシグネチャマッチングによる検知が一般的であるが，この手法は未知の攻撃を検知することが困難である．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．そこで，プロトタイプには，シェルコード検知の手法を利用する．ただし，シェルコード検知の手法は，シェルコードを利用しない攻撃を検知できない．

シェルコードの検知には，いくつかの手法[8][9]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[10]．SecondDEP は，Windows API の呼び出し元アドレスのメモリ保護属性が，正常なコードとシェルコードでは異なることに基づいて，シェルコードを検知する．

SecondDEP がシェルコードの実行を検知したら，攻撃に利用された攻撃クエリを獲得免疫機能に渡す．その後，SecondDEP は，新しく BIND9 のプロセスを起動してから，実行の制御を奪われた BIND9 を終了させる．

3.2.2 DoS 攻撃の検知

タイプⅠの DoS 攻撃（2.1.2 項参照）は，実装上の脆弱性を悪用してサービスを停止させる．この攻撃は，BIND9 がクエリを受信してからレスポンスを送信するまでの時間に基づいて検知される．プロトタイプでは，10 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とにした．

この処理を実現するために，クエリの受信が完了したら，Windows API の SetThreadpoolTimer により 10 秒後に作動するタイマースレッドをスレッドプールに登録する．タイマースレッドが攻撃に使用されたクエリを特定できるように，タイマースレッドには受信したクエリのトランザクション ID を引き渡す．10 秒後にタイマースレッドが作動したら，ハッシュテーブル（3.1 節参照）からトランザクション ID に対応したクエリデータを取り出して，そのクエリデータを攻撃データとして獲得免疫機能に渡す．次に，新しく BIND9 のプロセスを起動して，攻撃を受けた BIND 9 のプロセスを終了させる．

10 秒以内に WSASendTo が呼び出されたとき（受信したクエリの処理が正常に完了してレスポンスの送信の準備が完了したとき），WSASendTo が送信しようとしたレスポンスに対応したクエリは正常であったと判断する．このとき，ハッシュテーブルからレスポンスのトランザクション ID に対応したクエリデータを取り出して，そのクエリデータを正常データとして獲得免疫機能に渡す．さらに，そのクエリに対応したタイマースレッドを削除にする．

しかし，BIND9 の CVE-2015-5477 や CVE-2016-2776 など多くの脆弱性が攻撃されたとき，アサーションにより Windows のエラー報告（Windows Error Reporting：WER）を起動し，BIND9 のプロセスの制御はこの WER に奪われる．そのため，これらの脆弱性が攻撃されたとき，上述のスレッドプールに登録したタイマースレッドが起動しない．つまり，攻撃の原因となったクエリを学習する機会を失う．この問題を解決するために，WER が起動する前に呼び出される Windows API の SetUnhandledExceptionFilter をフックし，WER の呼び出し元スレッドを停止させて，WER が起動しないようにした．これにより，タイマースレッドが 10 秒後に作動し，攻撃を検知して，攻撃クエリを学習できるようになる．

タイプⅡのリソースを消費させる DoS 攻撃は，多数のリクエストの送信などにより，サーバアプリケーションのプロセス数またはスレッド数など内部リソースを大量に消費させ，サーバアプリケーションのサービスを停止させる．Windows における BIND9 の実装は，ソケットの入出力を行うワーカースレッド（BIND9 の内部関数 SocketIoThread）の個数には上限が設けられているため，スレッド数を枯渇させる攻撃に耐性がある．この他に，BIND9 のプロセスで使用する全てのスレッドを無限ループに陥れること

により，サービスを停止させる方法が考えられるが，この攻撃は 1 つ目の DoS 攻撃の検知機能により検知できる．

3.3 獲得免疫機能

獲得免疫機能は，分類器としてランダムフォレストを使用する．ランダムフォレストの実装は，現在，最も計算速度が速いとされている ranger である[12].

分類器に入力するデータはクエリのファジーハッシュ値である．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[13]. ファジーハッシュ値は，コロンで区切られた 3 つの部分から構成される．先頭は，入力データのサイズから導き出された整数である．残りの 2 つは，最長 64 文字の BASE64 符号化文字列と最長 32 文字の BASE64 符号化文字列で構成される．最長文字数に満たない場合，0 でパディングする．

3.3.1 学習機能

獲得免疫機能は正常なクエリ（正常データ）と攻撃に利用されたクエリ（攻撃データ）を学習する．獲得免疫機能は，`WSASendTo` が呼び出されたとき，すなわち，BIND9 がクエリに対応したレスポンスを返せるようになったとき，そのクエリを正常データとして学習する．ただし，獲得免疫機能が攻撃を検知した場合は，3.3.2 項で述べるように，獲得免疫機能はフォーマットエラーのレスポンスを返す仕様であるので，このときは，レスポンスを返せるようになったとしてもクエリを学習させない. 一方, 攻撃データは，自然免疫機能が攻撃を検知したとき，そのクエリを学習させる．なお，学習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, 正常データと攻撃データの両方をまとめて学習させる．

ランダムフォレストはオンライン学習できないため，新しいデータを学習するには，これまで学習したデータも必要である．そのため，稼働時間とともに，消費メモリと学習時間が増加する．特に，学習時間の増加は，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加させるため，検出精度を下げる要因となる．また，保持するデータの個数を制限すれば，検出精度が下がる．そこで，保持するデータの個数と検出精度の関係に基づいて，適切な個数を設定することとする．

保持するデータ群の構造はキューである．プロトタイプのキューには，正常データを保持する正常キューと攻撃データを保持する攻撃キューがある．キューを 2 つに分ける理由は，攻撃者がすべてのキューを攻撃データで埋め尽くしたとき，分類器はあらゆるデータを「攻撃」として誤って分類するためである．なお，これらのキューの長さは等しく，4.3 節でこれらの長さを分析する．

3.3.2 分類機能

BIND9 が，クエリを受信したら，獲得免疫機能は，そのクエリを「正常」または「攻撃」に分類する．分類結果が，「攻撃」であれば，そのクエリをゼロクリアし，DNS ヘッダの RCODE フラグを 1（フォーマットエラー）に変更して，クエリを無害化する．分類結果が，「正常」であれば，BIND9 にクエリの処理を継続させる．

4. 性能評価

プロトタイプの性能を評価するために，獲得免疫機能の検出精度とモジュールのオーバーヘッドを評価した. また, 3.3.1 項で述べた学習データの管理を分析して, 適切なキューの長さを示す．これらの評価では，次のシナリオを想定し，3 番目から性能評価の計測を開始する．

1. 脆弱性が見つかっていない状態で, BIND9 がキャッシュサーバとして運用され，これまで 80 個の正常なクエリを受信した（受信したクエリは，4.1 項で述べる 8 種類のタイプが各 10 個含まれる）.
2. BIND9 が, CVE-2016-2776 の脆弱性に対する攻撃クエリを受信するが, 自然免疫機能により攻撃が検知され, この攻撃クエリを学習した.
3. 継続的に正常クエリと攻撃クエリ（CVE-2015-5477 と CVE-2016-2776 に対する攻撃）を受信する．

評価で使用した PC の環境は次の通りである．

- DNS サーバ
  - CPU : Intel Core i5-2540M vPro 2.6GHz
  - Memory : DDR-1066 8GB
  - Disk : CSSD-S6T240NMG1Q 240GB
  - OS : Windows 8.1 Enterprise 64bit
  - DNS : ISC BIND 9.7.0
- DNS クライアント
  - DNS サーバと同一の PC

4.1 実験データ

実験データには，正常データと攻撃データの 2 種類がある．正常データは，以下の 8 種類の質問セクションのいずれかを含む DNS クエリである．PTR タイプで設定する IP アドレスは，IPv4 の範囲のランダムな数字である．その他のタイプで設定するドメイン名は，Alexa が公開している上位 5,000 件のドメイン名からランダムに選択した．

- A タイプ
- AAAA タイプ
- TXT タイプ
- CNAME タイプ
- PTR タイプ
- NS タイプ
- SOA タイプ
- MX タイプ

攻撃データは，CVE-2015-5477 及び CVE-2016-2776 の脆弱性を攻撃するクエリである．CVE-2016-2776 は，BIND9 の TSIG（Transaction Signature）クエリのパーサにある脆弱

性であり，CVE-2015-5477 は，BIND9 の TKEY（Transaction Key）クエリのエラー処理にある脆弱性である．いずれも攻撃されたら，BIND9 が強制終了する．

図 3 と図 4 に CVE-2015-5477 と CVE-2016-2776 の脆弱性を攻撃するクエリを示す．赤色で示した部分が攻撃特有のデータである．これらのクエリは，Metasploit Framework のモジュール（bind_tkey.rb と bind_tsig.rb）を参考した．

- DNS ヘッダ
  - トランザクション ID：16 ビットの乱数
  - フラグ：0x0000
  - 質問セクション：1
  - 回答セクション：0
  - 権限セクション：0
  - 追加セクション：1
- 質問セクション
  - 名前：**最長 42 文字のランダムな文字列**
  - タイプ：TKEY
  - クラス：IN
- 追加セクション
  - 名前：**質問セクションの名前と同じ値**
  - タイプ：TXT
  - クラス：IN
    ＊一部，省略＊
  - テキスト長：テキストのバイト数
  - テキスト：**最大 42 文字のランダムな文字列**

図 3　CVE-2015-5477 の攻撃クエリ

- DNS ヘッダ
  - トランザクション ID：16 ビットの乱数
  - フラグ：0x0000
  - 質問セクション：1
  - 回答セクション：0
  - 権限セクション：0
  - 追加セクション：1
- 質問セクション
  - 名前：**10 文字のランダムな文字列**
  - タイプ：A
  - クラス：IN
- 追加セクション
  - 名前：**250 文字のランダムな文字列**
  - タイプ：TSIG
    ＊一部，省略＊
  - データ長：252
  - アルゴリズム名：**215 文字のランダムな文字列**
    ＊以下，省略＊

図 4　CVE-2016-2776 の攻撃クエリ

4.2　検出精度

プロトタイプの性能を評価するために，獲得免疫機能の検出性能を評価した．BIND9 に送信する実験データは，800 個の正常データと 800 個の攻撃データの合計 1,600 個のクエリである．正常データには，4.1 節で述べた 8 種類のタイプから生成したクエリが各 100 個含まれる．攻撃データには，CVE-2015-5477 と CVE-2016-2776 から生成したクエリが各 400 個含まれる．図 5 に，10 回の試行によって得られた TNR（True negative rate）と TPR（True positive rate），および Accuracy の平均値の変化を示す．横軸は，DNS クエリの送信回数である．なお，1,600 個のクエリの送信順は，試行ごとにランダムに変更した．

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

図 5 から，自然免疫機能が攻撃を検知するたびに，獲得免疫機能の検出精度が向上していることがわかる．つまり，モジュールを BIND9 に適用することにより，攻撃に対するサーバの可用性が向上している．

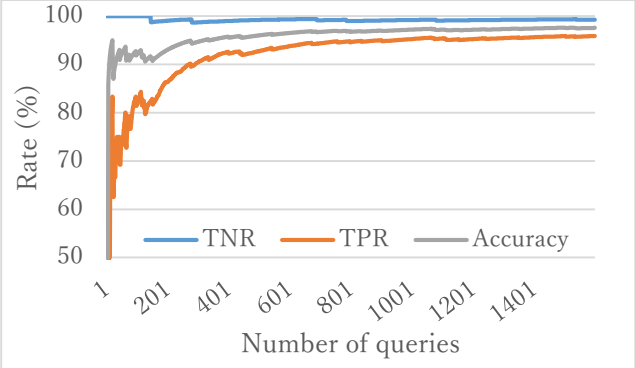

図 5 TNR と TPR，及び Accuracy

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 (%)	99.25	0.22	99.75	98.38
TPR (%)	94.49	0.51	95.88	93.38
Accuracy (%)	96.87	0.12	97.56	96.50

4.3　学習データのキューと検出精度

獲得免疫機能は，3.3.1 項で述べたように，学習データのキューを管理する．このキューの長さが検出精度や学習時間に影響することを述べた．そこで，適切なキューの長さを明らかにするために，キューの長さと検出精度の関係を分析した．この実験で使用するデータは，4.2 節の実験データと同じである．図 6 に，10 回の試行によって得られた TNR と TPR および Accuracy の平均値を示す．横軸は，正常キューと攻撃キューの長さを合計した長さである．なお，実験データの送信順は，試行ごとにランダムに変化する．

図 6 から，TNR は，キューが長くなるにつれて，緩やかに増加し，キューが 900 ぐらいになると，収束し始める．一方，TPR は，緩やかに減少して，キューが 900 ぐらいになると，収束し始める．キューの長さが 900 のとき，キューに含まれる正常データは平均で 450 個，攻撃データは平均で 43 個であった．つまり，攻撃データは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，0.074 秒であった．この学習時間は，BIND9 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，キューの長さが 900 の検出精度は，キューの長さに制限がないときの検出精度（4.2 節の結果）とほぼ一致している．以上から，キューの長さは 900（正常キューが 450，攻撃キューが 450）が適している．

Vol.2018-DPS-174 No.3
Vol.2018-CSEC-80 No.3
2018/3/5

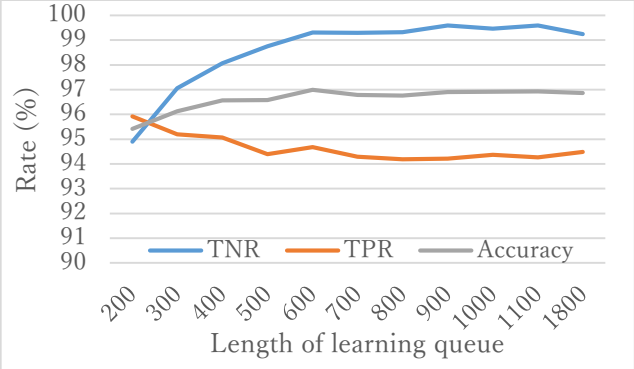

図 6 TNR と TPR，及び Accuracy

4.4 モジュールのオーバーヘッド

モジュールのオーバーヘッドを評価するために，BIND9 の RTT（Round-Trip Time）を計測した．RTT は，クエリの送信から受信までに要する時間である．計測に使用するクエリは，予め BIND9 に登録したドメイン名の問い合わせである．なお，計測には，DNS の性能評価ツールである dnsperf[14]を用いた．そのツールの実行オプションは，以下の通りである．

```
dnsperf -s 127.0.0.1 -d queries.txt -c 100
```

s オプションは，DNS サーバのアドレス，d オプションは，問い合わせるクエリ，c オプションは，クエリを送信するクライアント数である．

表 2 に，プロトタイプを適用していない BIND9 とプロトタイプを適用した BIND9 の RTT を示す．なお，これらは，10,000 回の計測で得られた値である．表 2 から，プロトタイプのオーバーヘッドは，125.66%であった．なお，オーバーヘッドの内訳は，自然免疫機能が 25.71%，獲得免疫機能が，74.29%であった．このことから，3.3 節で述べた分類機能や学習機能の処理に時間を要している．

表 2 RTT の計測時間（秒）

プロトタイプ未適用		プロトタイプ適用	
平均	標準偏差	平均	標準偏差
5.16×10^{-3}	4.775×10^{-3}	1.164×10^{-2}	4.11×10^{-3}

5. 考察

5.1 機械学習に対する攻撃

Huang らは，機械学習特有のセキュリティ侵害の可能性を指摘している[15]．セキュリティの侵害には，可用性，完全性，機密性（プライバシー）の侵害がある．本節では，可用性および完全性の侵害行為について考察を行う．なお，セキュリティモジュールは可用性の強化を目的としているため，機密性（プライバシー）の侵害については，考察しない．

5.1.1 可用性の侵害

可用性の侵害とは，正常データを攻撃に誤分類させて，サービスを妨害する侵害である．例えば，正常データに類似した様々な攻撃データを大量に送り続ける方法が考えられる．本研究のセキュリティモジュールでは，獲得免疫機能が正常データに類似する攻撃を学習すると，いくつかの正常データは攻撃に誤分類される可能性がある．そこで，セキュリティモジュールが可用性の侵害に対して耐性を有するのかをシミュレーションにより評価を行った．

シミュレーション評価では，次のシナリオを想定し，3 番目から計測を開始する．

1. BIND9 がキャッシュサーバとして正常に運用され，これまで 1,000 個のクエリを受信し，獲得免疫機能が正常なクエリを学習した．
2. 可用性を侵害するために，攻撃者が正常データに類似した 1,000 個の攻撃クエリを BIND9 に送信して，獲得免疫機能がこれらの攻撃を学習した．
3. 継続的に正常クエリ（手順 1 と同様のクエリ 1,000 個）と攻撃クエリ（手順 2 と同様のクエリ 1,000 個）を受信する．

手順 1 の正常データには，4.1 項で述べた 8 種類のタイプに AXFR と IXFR の 2 種類を追加した合計 10 種類のタイプのクエリがそれぞれ 100 個ずつ含まれる（合計 1,000 個）．なお，AXFR，IXFR 及び SOA のクエリには，TSIG の追加セクションが含まれる．

手順 2 の攻撃データには，CVE-2016-2776 を攻撃するクエリが含まれる．攻撃クエリを正常なデータと類似させるために，攻撃クエリの質問セクションは，手順 1 の正常なクエリと同じである．つまり，TSIG の追加セクションの名前とアルゴリズム名以外はすべて正常なクエリと同じである．

表 3 に，10 回の試行によって得られた TNR と TPR および Accuracy を示す．なお，クエリの送信順は，試行ごとにランダムに変更した．表 3 から，TNR が 99.83%であったことから，可用性はほとんど侵害されていなかったことがわかる．ただし，シナリオの手順 2 のクエリをさらに巧妙に細工する手口や，手順 3 において攻撃者が攻撃クエリと類似した正常なクエリを混ぜる手口も考えられるので，引き続き機械学習に対する攻撃を評価する必要がある．

表 3 TNR と TPR，及び Accuracy

TNR (%)	TPR (%)	Accuracy (%)
99.83	100.00	99.92

5.1.2 完全性の侵害

完全性の侵害とは，攻撃データを正常に誤分類させて検知を回避する侵害である．例えば，攻撃データに類似した正常データを大量に送信してから，攻撃を行う方法が考えられる．本研究のセキュリティモジュールでは，獲得免疫機能が最初の攻撃を検知できない可能性があるが，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，2 回目以降は同じ攻撃であれば，獲得免疫機能により検知できる．

5.2 モジュールの実装の課題

BIND9 に適用可能なセキュリティモジュールを実装して，明らかになった実装上の課題について考察する．3.1 節では，受信クエリを取得するために，BIND9 の内部関数（send_recvdone_event）をフックした．内部関数のアドレスは，ソースコードが改変されれば，変化することがあるため，このプロトタイプはソースコードのバージョンに強く依存する．バージョン依存の問題を解決するには，BIND9 の内部関数を使わずに，Windows API でクエリを取得する必要がある．しかし，BIND9 に限らず，非同期の送受信処理に関する実装は，OS やサーバアプリケーションの実装に強く依存する．そのため，仮に Windows API でクエリを取得できたとしても，BIND9 以外のサーバアプリケーションに BIND9 用のモジュールを適用することは難しく，OS とサーバアプリケーションの組み合わせごとに新規にモジュールを実装する必要があると考えられる．

もう 1 つの課題は，同一のトランザクション ID を複数受信したときに発生する．クエリとレスポンスの対応関係はトランザクション ID に基づくため，最初のクエリを処理している最中に，トランザクション ID が最初のクエリと同一である「新しいクエリ」を受信したとき，ハッシュテーブルに登録されたクエリデータが後から受信した「新しいクエリ」のデータに上書きされる．この問題を解決するため，ハッシュテーブルのキーを，トランザクション ID ではなく，クエリデータのハッシュ値（16 ビット）とする．クエリが異なればハッシュ値も異なるので，クエリデータが失われることはない．また，レスポンスからクエリを特定できるように，クエリを受信したら，クエリのトランザクション ID をハッシュ値に置き換える必要がある．BIND9 がクエリの処理を完了してレスポンスを送信するとき，レスポンスのトランザクション ID を元に戻す．この方法により，同一トランザクション ID の問題を解決できると考えられる．

6. おわりに

本研究は，これまで脆弱性へのサイバー攻撃に対するサーバアプリケーションの可用性を向上するため，適応的にサイバー攻撃を検知するセキュリティモジュールを提案してきた．本稿では，セキュリティモジュールの有用性を評価するために，広く普及している BIND9 に適用可能なセキュリティモジュールを実装して，2 つの脆弱性（CVE-2015-5477 と CVE-2016-2776）に対して攻撃を行い，モジュールの有効性を評価した．その結果，自然免疫機能が正しく攻撃を検知でき，獲得免疫機能による適応的な学習により，最終的な検出精度は，96.87%に達した．また，最適な検出精度を得るために必要な学習データのキューの長さを示した．しかし，獲得免疫機能の影響によりモジュールのオーバーヘッドは，125.66%もあった．

今後は，モジュールの検出精度を維持し，オーバーヘッドを小さくすることが課題である．Windows API によるクエリの取得や機械学習に対する攻撃の詳細な分析も課題である．

参考文献

[1] Saidane, A. et al., The design of a generic intrusion-tolerant architecture for web servers, IEEE Transactions on Dependable and Secure Computing, 2009, vol. 6, no. 1, p. 45–58.
[2] Sousa, P. et al., Highly available intrusion-tolerant services with proactive-reactive recovery, IEEE Transactions on Parallel and Distributed Systems, 2010, vol. 21, no. 4, p. 452–465.
[3] Seondong, H. et al., Designing and implementing a diversity policy for intrusion-tolerant systems, IEICE Transactions on Information and Systems, 2017, vol. E100-D, no. 1, p. 118-129.
[4] Okamoto, T. et al., Implementation and Evaluation of an Intrusion-Resilient System for a DNS Service, International Journal of Innovative Computing, Information and Control, 2017, vol. 13, no. 5, p. 1735-1750.
[5] Lin W. C. et al., CANN: An intrusion detection system based on combining cluster centers and nearest neighbors. Knowledge-based systems, 2017, 78, p. 13-21.
[6] Ashfaq R. A. R. et al., Fuzziness based semi-supervised learning approach for intrusion detection system. Information Sciences, 2017, 378, p. 484-497.
[7] Mohammadi M. et al., A fast anomaly detection system using probabilistic artificial immune algorithm capable of learning new attacks. Evolutionary Intelligence, 2014, 6(3), p. 135-156.
[8] Pappas, V. et al., Transparent ROP Exploit Mitigation Using Indirect Branch Tracing, USENIX Security, 2013.
[9] Cheng, Y. et al., ROPecker: A generic and practical approach for defending against ROP Attack, 2014.
[10] Okamoto, T., SecondDEP: Resilient Computing that Prevents Shellcode Execution in Cyber-Attacks. Procedia Computer Science 2015, 60:691-669.
[11] Tarao, M. and Okamoto, T., Performance evaluation of an immunity-enhancing module for server applications. Procedia Computer Science 2017, 112:2165-2174.
[12] Wright, M. N. and 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.
[13] Jesse, K., Identifying almost identical files using context triggered piecewise hashing. Digital investigation 3:91-97
[14] nominum, Measurement Tools, https://www.nominum.com/measurement-tools/, (参照 2018-02-03)
[15] Huang, L. et al., Adversarial machine learning. In: Proc. of the 4th ACM workshop on Security and artificial intelligence, ACM, 2011, p.43-58.