リモート型シェルコードのエミュレーションによる攻撃成否判定手法

鐘本　楊[1,2,a)]　青木　一史[1]　岩村　誠[1]　三好　潤[1]　小谷　大祐[3]　高倉　弘喜[4]　岡部　寿男[3]

概要：外部に公開しているサーバは攻撃者からアクセスしやすく，攻撃が頻発しているのが現状である．サーバ管理者や SOC アナリストは IDS から攻撃を示すアラートを受け取り，侵害があったか否かを確認する．攻撃が頻発する現在では通知されるアラートも大量にあるため，迅速に侵害があったアラートを発見することが難しい．本稿では攻撃に利用されるシェルコードに着目し，シェルコードをエミュレーションした際に発生する通信の挙動を基に攻撃の成否を判定する手法を提案する．攻撃の成否を判定することで，侵害につながる重要なアラートを迅速に発見する．実験により，通信機能を有するシェルコードに対して 60.0%以上の精度で成否判定できること，およびセキュリティコンテストの通信に適用し，得られた効果を示す．

キーワード：アラート検証，シェルコード，IDS，エミュレーション

Detecting Successful Attacks Based On Emulation of Remote Shellcodes

Yo KANEMOTO[1,2,a)]　Kazufumi AOKI[1]　Makoto IWAMURA[1]　Jun MIYOSHI[1]
Daisuke KOTANI[3]　Hiroki TAKAKURA[4]　Yasuo OKABE[3]

Abstract: Public servers are easily accessible to attackers, and attacks occur frequently. Server administrators and SOC analysts receive alerts from IDS and check whether attacks are succeeded. However, due to the large number of alerts, it is difficult to handle them quickly. In this paper, we focus on the shellcodes used for attacks. We propose a method to determine the success or failure of an attack based on the behavior of communication of shellcodes. The proposed method discriminates important alerts that lead to compromise. Experiment shows that proposed method can deal with more than 60.0% of shellcodes and can handle practical attack cases.

Keywords: Alert Verification, Shellcode, IDS, Emulation

1. はじめに

外部公開しているサーバは一般の利用者だけでなく，攻撃者からアクセスされやすく攻撃にさらされやすい．攻撃者は攻撃用ツールを作成して，ボット化したマシンを活用して攻撃を行うため，対象となるサーバに対して継続的に脆弱性を悪用する攻撃を行うことが可能である [1]．こうした脆弱性を悪用する攻撃に対応するのが CSIRT (Computer Security Incident Response Team) や SOC (Security Operation Center) である．CSIRT や SOC の分析者は攻撃を検知する IDS (Intrusion Detection System) からアラート通知を受け取り，攻撃による侵害がないか，侵害がある場合どの程度の被害なのかを調査する．しかし，攻撃が大量に発生する現状では，アラート数も大量になってしまう

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[2] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
[3] 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University
[4] 国立情報学研究所アーキテクチャ科学研究系
Information Systems Architecture Science, National Institute of Informatics
[a)] yo.kanemoto.zx@hco.ntt.co.jp

ため，被害が発生していない攻撃に関するアラートかあるいは，攻撃が成功し，被害の程度を調査する必要があるアラートがかを判断することが課題である．そのため，アラートに優先度を持たせることにより，より効率的に対応する必要がある．

本研究ではIDSのアラートを引き起こす攻撃の一つの要因であるシェルコード [2,3] に着目する．ここでいうシェルコードとは攻撃者が攻撃対象サーバを制御するために使用する機械語の命令列である．攻撃者が挿入するシェルコードのエミュレーションを行い，エミュレーション時に観測する動作から攻撃の痕跡を抽出する．攻撃の痕跡に含まれる通信の挙動と実際の通信を比較することで攻撃の成否を判定する．分析者に攻撃の成否に関する参考情報を与えることでアラート分析の効率を向上させる．

既存研究では攻撃に対して詳細な事前知識を必要としたり [4–11]，サーバの改変が必要だったり [3,12–14]，リクエストに対するレスポンスに攻撃の痕跡が残る場合のみに対応可能だったり [15]，という制約が存在する．攻撃に対する詳細な事前知識を要する場合，IDSの運用者に要求するスキルレベルが高くなり運用が困難になる．またサーバの改変は商用環境等に適用する場合，再度，動作検証を要することになり，運用者に負担を要してしまう．本研究ではこれらの制約を受けないより実用性の高い手法を提案する．

本研究の貢献は以下の通りである．

- リクエストに対するレスポンスに攻撃の痕跡が現れる攻撃に対応した既存研究 [15] を拡張し，レスポンス以外の通信に攻撃の痕跡が現れる攻撃にも対応する成否判定手法およびその実装を提案した．
- 評価により，通信に特徴が現れる攻撃に対して60%以上を正しく判定できた．
- 実際に行われたセキュリティコンテストの通信に対して手法を適用し，攻撃の成否を自動的に判定できた事例が複数存在した．

2. 背景

2.1 攻撃成否判定

サーバに対する攻撃の結果は，成功と失敗に大別される．攻撃の成功とは，攻撃者の意図通りに攻撃が行われている状態のことである．例えば2014年に話題となったShellshock[*1]の攻撃コードが下記のようだったとする．

```
() { :;}; wget http://x.x.x.x/exploit.sh
```

攻撃が成功すると，攻撃者が挿入したコマンド `wget http://x.x.x.x/exploit.sh` が実行され，攻撃対象のサーバがファイル `exploit.sh` を取得する．外部へのアクセスログに攻撃コードが示す接続先 `x.x.x.x` との通信

[*1] CVE-2014-6271
http://www.nca.gr.jp/2014/shellshock/

```
1  push    0x6603a8c0        ; ipaddr = 192.168.3.102
2  push    0xefb0            ; port = 45295
3  push    2
4  mov     ecx, esp
5  push    0x10
6  push    ecx
7  push    eax
8  mov     ecx, esp
9  mov     esi, eax
10 push    3                 ; set connect
11 pop     ebx
12 push    0x66              ; set socketcall
13 pop     eax
14 int     0x80              ; syscall
```

図 1: シェルコードの逆アセンブル結果の例
Fig. 1 Example of disassembled result of shellcode.

が以下のように記録されていた場合，攻撃は成功していると判断できる．

```
192.168.1.1 TCP_MISS/200 424
GET x.x.x.x/malware.sh - DIRECT/x.x.x.x
```

一方，攻撃が失敗している場合には挿入されたOSコマンドが実行されないため，攻撃コードが示す接続先への通信は発生しない．したがって，アクセスログに攻撃コードに含まれる接続先とのログが含まれていなければ攻撃は失敗したと判断できる．本研究では攻撃成否判定の結果をIDS等のアラートに付与し，一様だったアラートに対して優先度付けを行う．これにより，侵害等が起きた際に迅速に対応可能にすることを目的としている．

2.2 シェルコード

本稿では、シェルコードを攻撃者が攻撃対象サーバを制御するために使用する機械語の命令列と定義する．つまり，可読文字列からなるスクリプトコードやOSコマンドにより侵害ホストの制御を奪うコードは本研究の対象外とする．図1に本研究で対象とするシェルコードを逆アセンブルした例を挙げる．このシェルコードでは1,2行目で接続先のIPアドレスおよびポート番号を指定し，14行目のシステムコール発行命令により，通信を行う．シェルコードはローカル型，リモート型の2種類に大別される [16]．ローカル型は攻撃者が攻撃対象のサーバのシェル・ターミナルをすでに制御しており，権限昇格等を狙う際に使用される．リモート型は攻撃者がネットワークを介してサーバを攻撃する際，TCP/UDP通信を攻撃者が用意したサーバに行わせることにより攻撃対象サーバを制御する際に使用される．リモート型のシェルコードはその通信方法により更にconnect-back, bindshell, socket-reuseの3種類に分かれる．connect-backの場合は攻撃対象サーバから攻撃者のサーバへ新たに接続を行う場合，bindshellは攻撃者のサーバから攻撃対象サーバへ新たに接続を行う場合を意味して

いる．socket-reuse は攻撃する際の通信をそのまま制御に利用するので新たな接続を行わない場合を意味している．socket-reuse の場合は攻撃時に現在の通信が切断されないようにする必要があるため実際に利用されることは限定的である．

2.3 関連研究

著者らが既存の研究をを調べた限りでは攻撃に対して成否を判定する手法は，大きく 3 つのアプローチに分けられる [17]．

- **SIDS (Stateful IDS)：** SIDS では攻撃が成功した際の状態を定義し，攻撃があった際，予め定義された状態に至ったかどうかで攻撃の成否を判定する [18]．例えば，Vigna らの手法 [4] では攻撃の状態遷移に基づいて攻撃の成否を判定する．Robin らの手法 [5] および Zhou らの手法 [6] は既存の IDS のシグネチャを活用して，シグネチャマッチをネットワーク通信に対して複数箇所で行うことで，攻撃の成否を判定する．また脆弱性自体の処理を捉える研究 [7,8] も存在する．問題点として，攻撃状態をルールで定義したり，アプリケーションの脆弱な箇所の動作をルールで定義したりする必要があり，ルール定義に要求される知識レベルが高い点が挙げられる．そのため多様な攻撃に対応することが難しい．ルールを自動的に生成する手法 [19,20] も提案されているが，様々なアプリケーションに対する攻撃の成否ラベルを付与した通信の学習データを用意することも必要である．
- **CIDS (Correlation-based IDS)：** CIDS では IDS のアラートと脆弱性スキャナの情報を関連づけることで攻撃対象のサービス・アプリケーションが脆弱性を持っているかどうかで攻撃の成否を判定する [9–11]．例えば，Kruegel らの手法 [9] では IDS のアラートに含まれる CVE 番号と脆弱性スキャナで発見した脆弱性の CVE 番号が同一かどうかで関連付けを行う．この手法では Web アプリケーションや Web サーバに脆弱性が存在するバージョンを利用していないか確認し，その脆弱性に対する攻撃を検知した場合は有効な攻撃と判定する．近年の SIEM (Security Information and Event Management) 製品でも同様の判定アルゴリズムを実装しているものも存在する．SIDS と同様に脆弱性スキャナのルールの更新，IDS シグネチャとの関連付けのための情報の付与が必要になり，多様な攻撃に対応することが難しい．
- **EIDS (Emulation-based IDS)：** EIDS は攻撃があった際に，その攻撃コードの挙動をエミュレートし，実行する機械語命令列を抽出する手法である [3,12,13]．ホスト内でも同じ機械語命令列の実行を観測した場合，攻撃が実行されたとしてアラートを出力することで攻撃が成功したときのみを判定することが可能となる．これらの手法は x86 アセンブリの攻撃コードのエミュレーションに限られるが，Web アプリケーションに対しては，コードインジェクションの攻撃に特化した WeXpose [14] が存在する．ただ，この手法ではエミュレーション後の挙動を実システム内で観測する必要があるため，導入先の改変が必要である．また，著者らがすでに提案した AVT Lite [15] はシステム改変は必要ないが，攻撃リクエストに対するレスポンスに痕跡がある場合にしか対応できない．

表 1: 関連研究との比較
Table 1 Comparison with related works.

	SIDS	CIDS	EIDS	AVT Lite	本研究
制約 1	No	No	Yes	No	**No**
制約 2	Yes	Yes	No	No	**No**
制約 3	No	No	No	Yes	**No**

関連研究と本研究の差異をシステム改変が必要 (**制約 1**) 攻撃に対する詳細な事前知識が必要 (**制約 2**) リクエストに対するレスポンスに痕跡が現れることが必要 (**制約 3**) という 3 つの観点で比較した結果を表 1 に示す．

本研究では EIDS と同様にシェルコードをエミュレーションすることで攻撃の痕跡を抽出するため，SIDS や CIDS のような攻撃の仕組みに関する情報を必要 (**制約 2**) としないため，より実用的である．既存の EIDS との違いはサーバを改変 (**制約 1**) せず，ネットワーク通信のみから攻撃の成否を判定する所にある．また，著者らがすでに提案した AVT Lite [15] は攻撃リクエストに対するレスポンスに痕跡がある場合のみに対応可能 (**制約 3**) だったが，本研究は AVT Lite では対応できていないレスポンス以外に攻撃の痕跡がある場合に対応しており，既存研究と異なる．

3. 提案手法および実装

本研究ではリモート型のシェルコードによる攻撃の成否を判別する手法を提案する．図 2 に提案手法の各処理の概要を示す．

提案手法には 4 つのステップがある．最初のステップである **Code Extraction** では TCP/UDP ストリームからシェルコードである可能性が高いバイト列を抽出する．次に，抽出したシェルコードをエミュレータを用いて実行する (**Emulation**)．エミュレーションの際，シェルコードの挙動としてシステムコールを観測する．次に観測したシステムコールから攻撃の痕跡として利用できるものを選出する (**Indicator Extraction**)．最後に，選出された痕跡を活用し，実際の環境で観測したトラフィックと比較することで攻撃の成否を判定する (**Verification**)．

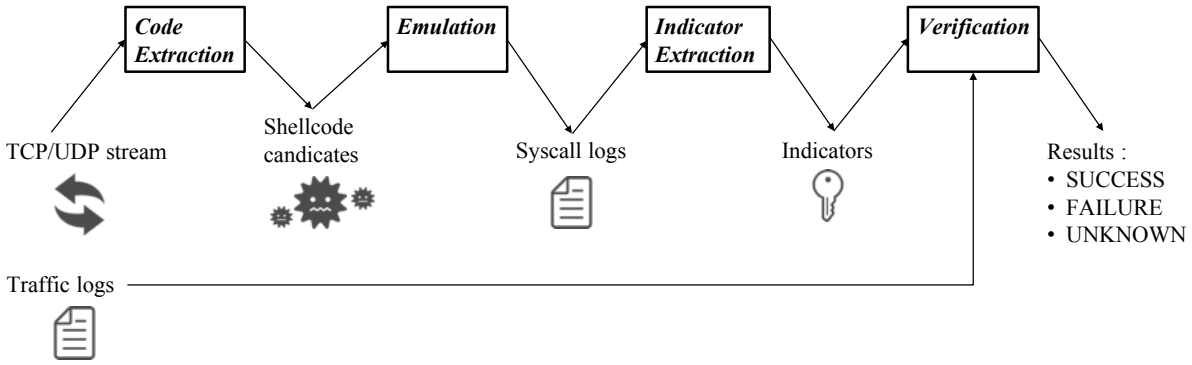

図 2: 提案手法の概要
Fig. 2 Overview of proposed method.

3.1 Code Extraction

攻撃と検知された TCP/UDP ストリームを入力として，そのストリームからシェルコードの抽出を行う．抽出方法は以下の 3 つである．

(1) 既知シグネチャの検知による抽出
(2) NOP スレッドの有無を用いた抽出
(3) GetPC コードの有無を用いた抽出

1 番目の抽出方法では IDS の検知シグネチャを公開しているサイトである Emerging Threats [*2]等から収集した．2,3 番目の抽出方法は既存研究 [21–23] で報告されているシェルコード部分の特徴を利用した．

NOP スレッドとは何も処理をしない命令列のことである．バッファオーバーフローなどの攻撃の場合，オーバーフローした際に実行される命令が参照できないと，セグメンテーション違反が起き，攻撃者が用意したシェルコードは実行されない．そのため，攻撃者はセグメンテーション違反等を回避するためにメモリアドレス空間を NOP 命令で埋めることにより，用意したシェルコードを実行させる可能性を高める．NOP スレッドはバイト値 `0x90` 等が連続するといった特徴的があるため，本手法はその特徴を基に NOP スレッドの検出を行う．NOP スレッドが検出された場合，NOP スレッドの終点を起点として，その後に続くバイト列をシェルコードとして抽出する．抽出するシェルコード終点は TCP/UDP ストリームの終点である．

シェルコードによっては自身の一部を難読化しておいて，実行時のみに復号して実行するものも存在する．復号には暗号化した部分のメモリアドレスの絶対アドレスが必要になるが，このアドレスを算出するために GetPC コードと呼ばれる命令列を実行することが多い．本手法はその特徴に着目し，GetPC コードよく利用される FSTENV 等の命令列のバイト列を検出し，そのバイト列を含む機械語命令として有効な可能な一連のバイト列を抽出する．抽出するシェルコードの起点は GetPC コードの開始位置より以前を逆順に走査して GetPC コードまで機械語命令・メモリ違反なく命令列を実行できる最長の位置がが起点となる．機械語命令・メモリ違反の判定には CPU エミュレータである Unicorn [24] を用いた．抽出するシェルコード終点は NOP の場合と同様に TCP/UDP ストリームの終点である．

シェルコードを抽出できない場合は判定不可の結果を出力し，処理を終える．

3.2 Emulation

前処理で抽出したシェルコードのエミュレーションを行う．アーキテクチャの差異によってエミュレーションできず，攻撃による痕跡を観測できない場合を減らすため，x86，x86-64， ARM など様々な環境を用意する．エミュレーションではシェルコードのバイト列をメモリに読み込み，読み込んだバイト列のメモリアドレスから実行を開始するプログラムを用意して，そのプログラムの実行する．シェルコードによって発生する挙動を観測するため，シェルコードを実行しているプロセスが発行するシステムコールを監視する．プロセスが終了した場合はシステムコール監視を停止する．接続待ちや攻撃者による意図的な遅延によってエミュレーションが終了しない事態を避けるため，エミュレーションにはタイムアウトを設ける．

3.3 Indicator Extraction

前処理によって観測したシステムコールの情報を基に攻撃の成否判定に利用できる痕跡を抽出する．提案手法ではネットワーク通信から観測可能な以下の 4 つのシステムコールの発行を攻撃の痕跡として記録する．

- `connect`, `sendto` システムコール
  外部に対して接続を行う挙動を表す．接続先 IP アドレスおよびポート番号を痕跡として記録する．
- `bind`, `recvfrom` システムコール
  外部からの接続を受け付ける挙動を表す．攻撃対象サーバの IP アドレスおよびポート番号を痕跡として記録する．

[*2] `https://rules.emergingthreats.net`

表 2: 抽出した攻撃痕跡の例
Table 2 Example of indicators.

動作	IP アドレス	ポート番号
`connect`, `sendto`	192.168.1.2	4444
`bind`, `recvfrom`	192.168.1.1	4444

TCP 通信であれば，socket システムコール後に connect/bind システムコールを発行する必要があり，UDP 通信も，一般的には connect/bind システムコールを利用するが，sendto/recvfrom システムコールでもデータの送受信が可能なため，この 4 つのシステムコールを監視対象とした．リクエストに対するレスポンスに攻撃の痕跡が現れる場合はすでに既存研究 [15] で取り扱われているため，本稿では対象外とする．IP アドレスおよびポート番号に加えて通信の内容も痕跡として，同じ内容を送受信しているかを判定することでより確実な成否判定が可能となる．例えば，TCP セッションの確立後，シェルコードが `ABCD` の文字列を送信することがわかっていれば，この `ABCD` という文字列の存在の有無によってより確度高く成否判定できる．しかし，攻撃者は容易に通信の内容を暗号化することが可能なため，検出を回避される恐れがある．そのため，提案手法では動作，IP アドレスおよびポート番号のみを攻撃痕跡の情報として利用した．抽出する痕跡の例を表 2 に示す．

痕跡が抽出できない場合，攻撃の成否を判断することができないことから，判定不可の結果を出力し，一連の処理を終える．

3.4 Verification

前処理によって得られた痕跡を基に実際の通信が攻撃の痕跡と同じであるかか否かを判定する．痕跡の動作が `connect`, `sendto` の場合，抽出した IP アドレスおよびポート番号に接続を行う通信，つまり TCP の SYN パケットあるいは UDP のパケットが送信されたか否かによって判定を行う．送信された場合，攻撃は成功，送信されていない場合に攻撃は失敗と判定する．

動作が `bind`, `recvfrom` の場合，攻撃が行われたサーバの IP アドレスおよび痕跡として抽出したポート番号への接続が確立したか否かによって判定を行う．TCP を利用した通信の場合，攻撃者のクライアントから SYN パケットが抽出した IP アドレスおよびポート番号に送信された場合，SYN-ACK を返送したか否かによって判定する．UDP を利用した通信の場合，ICMP Port unreachable が攻撃者に返送されたか否かによって判定を行う．ICMP Port unreachable を返送した場合，ポートは利用できる状況でないため，攻撃は失敗したと判定する．返送しない場合攻撃は成功したと判定する．

通信の観測には一定時間 T が設けられており，時間 T 以内の通信記録から判定する．T はユーザが指定する閾値である．閾値 T を設けた理由は，通信先が通常のアクセスでも行う IP アドレスとポート番号の場合，誤判定になってしまうため，時間制約により判定の正確性を維持するためである．

時間 T 以内に接続を行う通信を観測できなければ，シェルコードは実行されていないため，攻撃は失敗したと判定する．

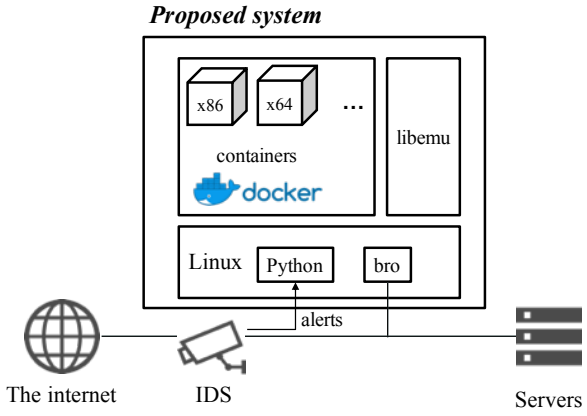

図 3: 提案システムの概要
Fig. 3 Overview of proposed system.

3.5 実装

提案手法を実装したシステムの全体像を図 3 示す．TCP/UDP ストリームの取得には OSS のネットワーク監視ツールである Bro[*3]を利用した．攻撃成否の判定の部分は Python で実装した．攻撃コードをエミュレートする環境には Docker[*4]および既存の CPU エミュレータである libemu[*5]を採用した．libemu は x86 のシェルコードのみに対応しているため，x86 のシェルコードの場合のみ追加で libemu でもエミュレーションを行う．CPU エミュレータはより詳細にシェルコードを分析できる利点があるが，シェルコードを動作させるためには正しくメモリ空間をする必要がある．また，システムコールや API をエミュレートする訳ではないため，利用者で適切に定義する必要があるという難点がある．Docker を用いたエミュレータの場合，Docker のみではシェルコードを詳細に分析できないが実態により則した環境を用意できることが利点である．そのため，提案システムでは Docker と libemu を組み合わせて利用する．提案システムでは攻撃が発生する度，CPU エミュレータである libemu によるエミュレーションおよび，エミュレータとなる Docker Container を作成し，その環境内でシェルコードを実行する．なお，Docker によるエミュレーション時は外部への攻撃を防ぐため，ネットワー

[*3] https://www.bro.org/
[*4] https://www.docker.com/
[*5] https://github.com/buffer/libemu

表 3: D_{msf} 生成に用いた MSFvenom のオプション
Table 3 MSFvenom options.

オプション	値
エンコード方法	無，`add_sub`，`alpha_mixed`，`bloxor`，`shikata_ga_nai`，等 (22 種類)
エンコード回数	1, 2
回避文字列	無, `\x00\xff`
NOP スレッド	無, 100

ク通信は全て遮断した．

4. 評価

精度および性能の 2 つの観点から評価を行った．

4.1 精度評価

精度評価では複数のシェルコードを用意し，攻撃が成功した際，提案手法が正しく攻撃成功判定ができている割合 (再現率)，提案手法によって成否判定が可能な際に正しく攻撃成功が判定できている割合 (適合率) および，仮に IDS が誤検知した際に正常の通信をシェルコードによる攻撃として判定してしまう割合 (誤判定率) を調べた．評価ではバッファオーバーフローの脆弱性があるサーバプログラムを用意し，シェルコードを挿入することで攻撃を再現した．なお，攻撃を再現する際に NOP スレッドを用いているため，シェルコードが正しく抽出できている状態である．

再現率および適合率を評価するデータセットは 2 種類用意した．1 種類目のデータセット D_{edb} は様々なシェルコードに対応できるかという観点で，シェルコードのデータベースである Exploit-DB[*6]から収集したシェルコード 898 件を用いて作成した．今回の評価では Linux OS を対象とした x86 および x86-64 アーキテクチャに対するシェルコード及び攻撃の痕跡が通信に残るシェルコード 115 件のみを対象とした．2 種類目のデータセット D_{msf} はもとは通信の接続を行う 1 つの x86 用シェルコードであるが，様々な変換・難読化を施すことにより生成した複数のシェルコードを用いた．変換・難読化には Metasploit[*7]の機能である MSFvenom を利用し，変換・難読化に成功したもののみを用いた．表 3 にその際に利用したオプションを示す．

誤判定率の評価に用いるデータセットには，Wireshark のサイト[*8]から収集した様々なプロトコルの通信が含まれる 779 個の PCAP ファイルから作成した．収集した PCAP ファイルから 87,774 件の TCP/UDP ストリームを抽出し，

[*6] 2018 年 7 月 25 日収集
`https://exploit-db.com/`
[*7] `https://www.metasploit.com/`
[*8] 2018 年 8 月 1 日収集
`https://wiki.wireshark.org/SampleCaptures`

表 4: 精度評価結果
Table 4 Evaluation result of accuracy.

項目	データセット	割合
再現率	D_{edb}	60.0% (69/115)
	D_{msf}	64.8% (70/108)
適合率	D_{edb}	100% (69/69)
	D_{msf}	100% (70/70)
誤判定率		0.01% (5/87,774)

各ストリームを提案手法で攻撃として判定してしまうか否かを調べた．表 4 に評価結果を示す．D_{edb}, D_{msf} どちらともに 60%以上のシェルコードから痕跡を抽出でき，正しく攻撃が成功したと判定できていた．シェルコードから痕跡を抽出できなかった理由はシェルコードの実行に失敗してたためである．この原因は大きく 2 つであった．1 つ目はシェルコードが別の OS コマンド，例えば `wget` や `nc` を呼び出して外部通信を行う仕組みになっていたことである．エミュレータではそれらコマンドインストールしておらず，差異が発生し攻撃成功を正しく判定できなかった．2 つ目はエミュレーション時，シェルコードが正しい番号のシステムコールの番号を計算できなかったことにある．x86 命令ではどのシステムコールを発行するか `eax` レジスタの値によって決まる．シェルコードから痕跡を抽出できなった際はこの `eax` レジスタの値が存在しないシステムコールの値になっていたため，痕跡を観測できなった．

誤判定率は 0.01%程度であった．誤判定した原因は通常の通信で偶然にもシェルコードの特徴である NOP スレッドや GetPC コードと同じバイト列が発生したためであった．

4.2 性能評価

D_{edb} および D_{msf} のシェルコードに対してエミュレーションに要した時間を計測した結果を図 4 に示す．約 18%のシェルコードのエミュレーションはタイムアウト (5 秒) を引き起こしていたため，図 4 はタイムアウトを引き起こさない場合の処理時間である．シェルコードの全体約 82%のシェルコードは 1 秒以内に判定が終了することから実環境でも多くの攻撃を瞬時に処理することが可能な性能である．タイムアウトの原因は接続を待ち受ける動作をするシェルコードによって接続待ちになってしまうことおよび，4.1 節で述べたでエミュレーション環境で正しい番号のシステムコールの発行が行われず，ループ処理に陥ってしまうからである．

4.3 事例分析

セキュリティコンテストである MACCDC (Mid-Atlantic Collegiate Cyber Defense Competition) のネットワーク通信に対し，提案手法を適用した際の結果を示す．MACCDC

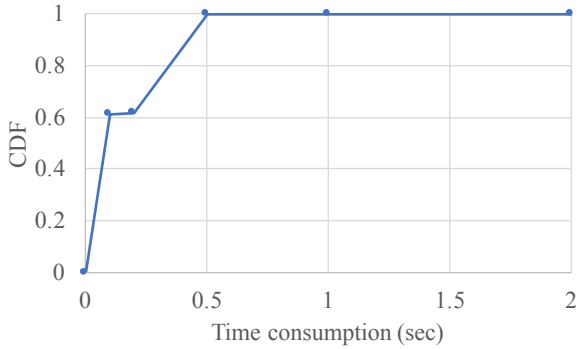

図 4: 処理時間の累積分布 (タイムアウトした場合を含まない)
Fig. 4 CDF of time consumption (not include timeout case).

表 5: MACCDC データセットに適用した結果
Table 5 Verification result of MACCDC dataset.

年	2010	2011	2012
TCP セッション数	4,862,720	2,575,472	715,430
攻撃 TCP セッション数	11,215	16,233	25,640
攻撃成功判定数	2	0	0
攻撃失敗判定数	4,204	5	137
アラート削減割合	37.5%	0.03%	0.53%

はアメリカで開催されるコンテストであり，コンテスト運営者が攻撃者となり，大学生等から構成される参加チームのサーバに対して攻撃を行い，攻撃に対して参加チームがいかに迅速に対処するかを競うコンテストである．ただし，このデータセットには指標ラベル等が付与されておらず，ネットワーク環境を明記した資料もないため，精度等の評価にはできず，適用した際の結果のみを示す．2010 年から 2012 年までのコンテストの通信は PCAP ファイルとして公開されている[*9]ためこれらの通信に対して，提案手法を適用し，判定結果を表 5 に示す．攻撃 TCP セッション数はシェルコードを含む攻撃の数を表している．アラート削減割合は仮に全ての攻撃 TCP セッションを IDS が検知した際に，提案手法がアラートに対して失敗判定を下した割合である．2010 年において 2 件攻撃が成功したと判定した．いずれも指定された宛先 IP アドレス 157.99.66.56 のポート番号 5555 への接続を行う ELF[*10]ファイルを Web サーバからダウンロードした後，接続を行なっていた．分析の結果この 2 件の事象はユーザが故意にプログラムをダウンロードし実行したの可能性もあり，どちらも明確に攻撃を示唆するものではなかった．

2010 年および 2012 年に関しては攻撃を失敗として判定できていた．特に 2010 年においてはシェルコードを利用する攻撃全体に対して 36.8%も提案手法で自動的に攻撃が失敗と判定したことから実環境においてもアラートの削減に一定の効果をもたらす可能性を示した．

[*9] https://www.netresec.com/?page=MACCDC
[*10] Linux の実行ファイルの形式

表 6: MACCDC データセットから抽出した攻撃の痕跡
Table 6 Indicators extracted from MACCDC dataset.

年	動作	IP アドレス	ポート番号
2010	connect	217.22.112.14	4721, 8925, 12517, 34112
	connect	10.0.2.15	2244
	connect	157.99.66.59	4454
	connect	217.22.112.53	29797
	connect	157.99.66.56	5555
	bind	150.204.83.15	4186, 22990, 28369
2011	connect	192.168.202.172	1465, 1543, 3035, 3171, 3559
2012	connect	192.168.202.102	43248, 47191, 48351, 49124, 49305, 50049, 50413, 53022

提案手法が抽出した痕跡を表 6 に示す．痕跡を抽出できた攻撃のうち，2010 年の攻撃は IMAP サーバに対する攻撃，2012 年の攻撃は FTP サーバを対象とした攻撃であり，送信元 IP アドレスが 192.168.202.102 で，送信先 IP アドレスが 192.168.24.101 であった．2012 年のコンテストの開催資料[*11]から，192.168.21.0/24 から 192.168.28.0/24 の各/24 セグメントは 8 つの参加チームに与えられるサーバのネットワークであるため，参加チームが攻撃者からの攻撃を受けていたことが推測できる．提案手法はこれらの攻撃による侵害はないと自動的に判定することを可能にした．

5. 制約

提案手法は攻撃に利用されるシェルコード を抽出し，そのシェルコード をエミュレーションすることで攻撃が成功した際の痕跡を実際の通信と比較して成否を判定するものである．その性質上以下のような場合には対応できない．

- 攻撃の痕跡が通信に現れない場合
  クライアントからのリクエストに対するレスポンスに攻撃の痕跡が残る場合は，既存手法で対処可能であるため本稿では対象としないが，サーバ内をファイル等を改変する攻撃においてはエミュレーションすることは可能だが，通信の観測ではサーバ内の状況の変化を把握できないため成否判定ができない．4 章で用いたデータセット D_{edb} から推測すると，攻撃の痕跡が通信に現れないシェルコードは 783 (898-115) 件あり，シェルコード全体の 87.1%も存在する．
- シェルコードが存在しない場合
  バックドアがソースコードに仕込まれていて，特殊なリクエストを送信することでバックドアを起動し接続を行う場合，バックドアの通信を観測することはでき

[*11] https://www.nationalcyberwatch.org/ncw-content/uploads/2016/03/NCC_Press_How_To_Prepare_For_the_CCDC-1.pdf

るが，シェルコード を観測することができないため，エミュレーションができず痕跡を把握できないため，成否判定を行うことはできない．また，権限昇格・認証バイパス等の認可・認証に関わる攻撃は通信から状況の変化を区別できないため対応できない．

- ROP 形式のシェルコードの場合
  バッファオーバーフロー等によるシェルコードを防ぐため，OS にはデータ実行防止 (DEP) 機構が備わっている．DEP を回避し，シェルコードを実行することを目的とした攻撃手法が ROP (Return-Oriented Programming) である．ROP の場合攻撃リクエストにはメモリアドレスが現れるため，3.1 節で述べた抽出手法ではシェルコードと認識できず，成否判定できない．

6. おわりに

大量にあるアラートに対して効率的に処理するという課題に対して，本研究では攻撃の成否判定を行うことによってアラートの優先度を分別することで課題の解決に貢献した．提案手法では攻撃に利用されるシェルコードをエミュレーションし，エミュレーションの結果得られる攻撃痕跡と実際の通信の挙動を照合することで攻撃の成否を判定した．評価よりリモート型シェルコードの約 60.0%を正しく判定可能であった．

今後の課題はシェルコード抽出精度の評価，エミュレーションの精度の向上および，実ネットワーク環境を利用した手法の評価がある．

参考文献

[1] Davide Canali and Davide Balzarotti. Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web. In *NDSS*, 2013.

[2] Ramkumar Chinchani and Eric van den Berg. A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows. In *RAID*, 2005.

[3] Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. Network Level Polymorphic Shellcode Detection Using Emulation. In *DIMVA*, 2006.

[4] Giovanni Vigna, William Robertson, Vishal Kher, and Richard A. Kemmerer. A Stateful Intrusion Detection System for World-Wide Web Servers. In *ACSAC*, 2003.

[5] Robin Sommer and Vern Paxson. Enhancing Byte-level Network Intrusion Detection Signatures with Context. In *CCS*, 2003.

[6] Jingmin Zhou, Adam J. Carlson, and Matt Bishop. Verify Results of Network Intrusion Alerts Using Lightweight Protocol Analysis. In *ACSAC*, 2005.

[7] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: Vulnerability-driven Network Filters for Preventing Known Vulnerability Exploits. In *SIGCOMM*, 2004.

[8] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards Automatic Generation of Vulnerability-Based Signatures. In *IEEE S&P*, 2006.

[9] Christopher Kruegel and William Robertson. Alert Verification - Determining the Success of Intrusion Attempts. In *DIMVA*, 2004.

[10] Frederic Massicotte and Mathieu Couture. Context-Based Intrusion Detection Using Snort, Nessus and Bugtraq Databases. In *PST*, 2005.

[11] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE DSC*, 2004.

[12] Ali Abbasi, Jos Wetzels, Wouter Bokslag, Emmanuele Zambon, and Sandro Etalle. On Emulation-Based Network Intrusion Detection Systems. In *RAID*, 2014.

[13] 嶋村誠, 河野健二. Yataglass: 攻撃の擬似実行による攻撃メッセージの振舞いの解析. 情報処理学会論文誌, 2009.

[14] Jennifer Bellizzi and Mark Vell. Wexpose: Towards online dynamic analysis of web attack payloads using just-in-time binary modification. In *SECRYPT*, 2015.

[15] 鐘揚, 青木一史, 三好潤, 嶋田創, 高倉弘喜. AVT Lite: 攻撃コードのエミュレーションに基づく Web 攻撃の成否判定手法. コンピュータセキュリティシンポジウム (CSS) 論文集, 2017.

[16] J.C. Foster. Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals. *Elsevier Science*, 2005.

[17] Neminath Hubballi and Vinoth Suryanarayanan. False Alarm Minimization Techniques in Signature-based Intrusion Detection Systems: A Survey. *Computer Communications*, 2014.

[18] Frédéric Cuppens and Alexandre Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *IEEE S&P*, 2002.

[19] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand, and Mathieu Couture. Automatic Evaluation of Intrusion Detection Systems. In *ACSAC*, 2006.

[20] Frederic Massicotte, Yvan Labiche, and Lionel C Briand. Toward Automatic Generation of Intrusion Detection Verification Rules. In *ACSAC*, 2008.

[21] 藤井孝好, 吉岡克成, 四方順司, 松本勉. エミュレーションに基づくシェルコード検知手法の改善. マルウェア対策研究人材育成ワークショップ (MWS), 2010.

[22] 神保千晶, 吉岡克成, 四方順司, 松本勉, 衛藤将史, 井上大介, 中尾康二. CPU エミュレータと Dynamic Binary Instrumentation の併用によるシェルコード動的分析手法の提案. 電子情報通信学会技術研究報告 (ICSS), 2010.

[23] 田中恭之, 後藤厚宏. 攻撃コードの特徴からみた対策の検討. 暗号と情報セキュリティシンポジウム (SCIS), 2014.

[24] NGUYEN Anh Quynh and DANG Hoang Vu. Unicorn: Next Generation CPU Emulator Framework. In *Black-Hat USA*, 2015.