Computer Security Symposium 2019
21 - 24 October 2019

Webアプリケーションフレームワークに組み込むハニーポットの提案

久保田 康平[1,a)]　小出 洋[2,b)]

概要：本研究は Web フレームワークに高対話型ハニーポットを設置し，攻撃の誘導と記録を行うことで Web アプリケーションのセキュリティを向上させることを目的にしている．特定の企業を標的とし，様々な方法を用いてそのサーバを攻撃する標的型攻撃や，脆弱性が発見された際，セキュリティパッチが当てられる前にその脆弱性に対して攻撃を行う，ゼロデイ攻撃が存在し，それらを防御することは難しくなっている．また，マルウェアを一部変更した亜種を用いたり，全く新しい攻撃を開発したりすることで，防御をしづらくする技術的な攻撃の多様化も問題である．これらの攻撃の変化に対して，攻撃を受けないようにする入口対策だけでは難しい．そのため攻撃を完全に遮断するのではなく，機密情報の存在するところから別の場所に誘導する研究が行われており，その研究の一つがハニーポットである．これは脆弱に見える Web 上にサーバを設置し，攻撃させることで，被害にあうことなく検出と記録を行い，攻撃手法を収集する仕組みである．本論文では Web アプリケーションフレームワーク内に高対話型ハニーポットを設置した Web アプリケーションフレームワークを提案する．

キーワード：高対話型ハニーポット，Web アプリケーション，Web アプリケーションフレームワーク，Web アプリケーションファイアウォール

A Proposal of Web Application Framework Which Has an embedded Honeypot

Kohei Kubota[1,a)]　Hiroshi Koide[2,b)]

Abstract: The purpose of this study is that Web application's cyber security improves by installing a high interaction honeypot in a Web application framework, guiding and recording attacks. As a background, it is difficult to protect Web applications from APT(Advanced Persistent Threat) attacks or zero-day attacks. In addition, technical attack method's diversity, namely, subspecies of attacks and new attack methods developing, causes difficulty detecting attacks. These attack methods changing, it is difficult to block any attacks to Web applications. Hence, measures about leading attacks to information of no importance exist. One of the measures is a honeypot. It is a Web application decoy systems having a few purposes: detecting attacks, leading them to information of no importance, and collecting valuable information about attack methods. We suggest a Web application framework which installs a high interaction honeypot.

Keywords: Honeypot, Web application, Web application framework, Web appliaction firewall

[1] 九州大学大学院システム情報科学府
Kyushu University Graduate School and Facility of Information Science and Electrical Engineering
[2] 九州大学 情報基盤研究開発センター
Kyushu University, Fukuoka, 819–0385, Japan
[a)] kouhei.kubota@icloud.com
[b)] koide@cc.kyushu-u.ac.jp

1. はじめに

本研究の目的はアプリケーション内に搭載した高対話型ハニーポットを用いて攻撃を検出，記録， そして攻撃から Web アプリケーションの保護を可能にすることである．こ

の目的を達成するために一般的な Web アプリケーションフレームワークの機能に加えて，HTTP 通信における特定のリクエストを攻撃と推測する攻撃検出機能を用いて，攻撃者であるクライアントの IP アドレスとリクエストパス，リクエストのパラメータ，リクエストを送信した時間を記録し，その攻撃者と推測される IP アドレスからのリクエストに対して，API を変更することで Web アプリケーションを保護する Web アプリケーションフレームワークを提案する．

近年，情報システムへの攻撃手法が変化している．その一つが特定の企業等を標的とし，様々な方法を用いてそのサーバを攻撃する標的型攻撃 [8] である．この攻撃は具体的な攻撃手法が存在するのではなく，目的を達成するまで執拗に様々な攻撃を繰り返すものであり巧妙化・多様化している．その中には新しい攻撃手法の開発や既存の攻撃手法を一部変更した亜種によって，既存の対策では対策しきれないようにするといった方法が存在している．

サイバー攻撃から Web アプリケーションを防御するシステムの一つに Web Application Firewall[7](WAF) があげられる．WAF は，クライアントと Web アプリケーション間であるネットワーク上，もしくはアプリケーションがある Web サーバに設置される．WAF はクライアントから送られたリクエストから，パラメータ中のシングルクォーテーションのような，特定パターンの検出機能に基づいて Web サイトとその利用者の通信の内容を機械的に検査する．攻撃と認められる特定のパターンを見つけると自動的に通信を遮断する．それにより脆弱性攻撃から Web サーバや Web アプリケーションを保護することができるというシステムである．WAF は直接管理できない Web アプリケーションに対策を実施したり，Web アプリケーションの修正が困難な場合であったり，Web アプリケーションへの攻撃をすぐに防御する必要があったりする際に，有効である．一方で WAF が有効でない場合がある. その一つが WAF には検出できない攻撃が存在することである. 辞書式攻撃やパスワードリスト攻撃のようなログイン試行を繰り返すことによって権限を持っているユーザしか閲覧できないページに侵入するブルートフォース攻撃は，HTTP 通信においては正常な通信であるため，一般的に WAF では検出することができない．

また，WAF は認証や権限が必要なページへの遷移が正常かどうかを判定することが難しい．WAF において，そのような遷移が妥当であるかを調べるためのリクエストの情報として，HTTP リファラ [12] という属性が存在する．リファラは HTTP でのリクエストを発生させる要因となった URL のことである．ほとんどの Web サーバには全トラフィックログのログが存在し，ブラウザがリクエストごとに送信した HTTP リファラを記録している．リファラを用いることでクロスサイトリクエストフォージェリの対策が可能である．しかし，リファラ情報はプライバシーに関わることがあるため，プロキシやファイアウォールで除去することがある．この機能で除去した際，WAF がこの除去を攻撃と判断することがあり，リクエストが遮断されることによって可用性を損なう欠点がある．

また，Web アプリケーション等のシステムはオンプレミス環境だけではなく，一部をクラウド化した環境も見られるようになっている．一部クラウド化することによって，全てオンプレミス環境だった時よりもシステム全体が大規模かつ複雑化し，把握することが難しくなる．したがって，より Web アプリケーションと密に連携したセキュリティ機能が必要である．

これらの問題解決のための手法が，目的に示した高対話型ハニーポットを搭載した Web アプリケーションの開発である．まず，WAF で検出しづらい攻撃を検出することが可能になる．WAF で検出しづらい攻撃とは HTTP 通信上では正常であるがアプリケーションに対しては攻撃である通信である．これに対して，アプリケーションから見て正常かどうかアプリケーションフレームワークで検出することができるため，より幅広い攻撃の検出・記録が可能になる. また，フレームワークにハニーポット機能を搭載することで，Web アプリケーション単位でセキュリティ機能を持つことが可能になるため，クラウド化によって大規模・複雑化する情報システムに対しても，システムを把握しやすい. したがって Web アプリケーションフレームワークにハニーポットを組み込むことで，WAF では検出しづらい攻撃の検出を可能とし，Web アプリケーションに密接なセキュリティ機能を持たせることが可能になる．

2. 関連研究・技術

2.1 高対話型ハニーポット

ハニーポット [9] とは, ネットワーク上で攻撃者やマルウェアを誘導するために，あえて脆弱性や脆弱に見せかけた仕組みをアプリケーションや OS，もしくはそれを模倣したシステム上に用意し，攻撃者を誘導することで，攻撃活動や不正アクセスを促すシステムである．ハニーポットを用いる事で，攻撃者の行動や攻撃手法といった情報を収集する事が可能である．ハニーポットには，実現方法によって分類されその 1 つが対話型ハニーポットがある．対話型ハニーポットは，低対話型と高対話型の 2 種類に分類される．そのうち，高対話型ハニーポットは低対話型ハニーポットとは異なり, 実際のマシン上でアプリケーションや OS の動作を行う．したがって，より高度な攻撃手法の取得が可能になる一方，OS コマンドやアプリケーションの全機能を利用できることから，ファイルシステムの破壊やハニーポット自体が踏み台とされるリスクがある．この事から高対話型ハニーポットを設置する際，攻撃者が他のシステムに影響を与えないようにネットワークを考慮するこ

とが必要であるため，運用が難しい．

2.2　リバースプロキシ

リバースプロキシ [10] とは，特定のサーバへの要求を必ず経由するように設置されたプロキシサーバ (代理サーバ) のことである．このリバースプロキシを設置することで，管理者には幾つかの利点がある．まず一つ目に，リバースプロキシを前置することで，直接アプリケーションサーバに対して要求することがなくなるので，サーバのセキュリティ機能を向上させることができる．二つ目に，要求を異なるアプリケーションサーバへルーティングすることで，負荷を複数のサーバに分散できる負荷分散という利点がある．三つ目に，複数のサービスを一つのサーバで実行しているよう，クライアントに見せることができるため，仮想的なサーバの統合をすることができるようになる．

2.3　リバースプロキシを用いた実装

先行研究 [1] の一つにリバースプロキシを用いた高対話型ハニーポットの搭載がある．リバースプロキシをアプリケーションサーバに前置し，クライアントが要求を行うとその要求から，クライアントの IP アドレスやリクエストヘッダなどを読みこむ．検出パターンに基づいて攻撃と判定された場合，攻撃とみなされた要求を管理データベースに保存し，アプリケーションのためのデータベースを正規のものから擬似のものに変更することで，攻撃の影響を低減するものであった．以下にそのシステムの簡易図を示す．

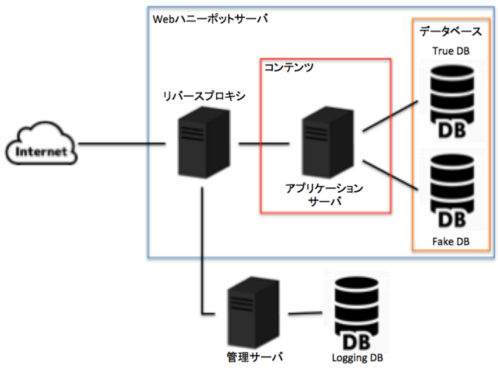

図 1　リバースプロキシを用いたシステム構成の簡易図．
Fig. 1　Previous system diagram using reverse proxy.

この研究において，利点は複数のアプリケーションをリバースプロキシに接続することで，複数のアプリケーションに攻撃検出機能を追加可能な点である．また，リバースプロキシとしての利点もある．そのうち，セキュリティ面では手前にプロキシサーバを前置することで機能を向上させることが可能という点である．しかし，データベースを変更する時にリクエストパスを変更する必要があり，その際にアプリケーションを理解しなければ十分に機能させることができないことがある．

3.　Bottle を用いた高対話型ハニーポットを搭載した Web アプリケーションの提案

リバースプロキシを用いた実装は，サービスが増えた時，そのサービスをリバースプロキシに接続すれば，攻撃検出を行えることが利点であるが，アプリケーション単位でのセキュリティ機能の向上という点においては，十分ではないことがある．したがってこの改善案として，アプリケーションフレームワークにハニーポット機能を備えることを提案する．

この機能は上記の先行研究と比較して，アプリケーションの情報を攻撃検出に利用できるという利点がある．例えば，ページ遷移情報を取得することで，不正なページへの遷移を攻撃として検知することが可能になる．一方で先行研究と異なる点としては，アプリケーション 1 つずつに設定が必要な点である．例えば，不正なページ遷移に対して攻撃を検出する場合，正しいページ遷移の情報がアプリケーションごとに必要である．

3.1　Bottle フレームワーク

Bottle[11] は Python における Web アプリケーションフレームワークの一つである．このフレームワークの特徴は動作が軽く軽量な点である．また，このフレームワーク自体が 1 ファイルによって構成されており，Python の標準ライブラリ以外の依存がないため扱いやすい．ルーティングは route() デコレータを用いて行い，URL パスとコールバック関数を結びつけてくれる．

3.2　hook() デコレータ

Bottle フレームワークには，hook() デコレータが存在する．hook('before_request') デコレータを用いることで，クライアントがリクエストを送信すると，hook() デコレータで定義した関数を毎回実行する．また，hook('after_request') デコレータを用いると，ルーティング後に毎回実行する関数を呼び出す．

3.3　実装について

このフレームワークは，攻撃かどうかの判定として規定回数以上レスポンスコードが 400 番台通信を行ったクライアントを攻撃者とみなすものとした．判定が上記のものであるのは，攻撃者が，管理者以外は閲覧できないようなページに対して不正に閲覧を試みる際，管理ページであると推測されるパスを繰り返し要求することで，管理ページを閲覧する攻撃が存在すること，そしてこの攻撃は HTTP 通信においては正常な通信とみなされるからである．リクエストを受け取った直後とその後レスポンスが生成された直後の二箇所が記録・検出部分であり，以下のような要素

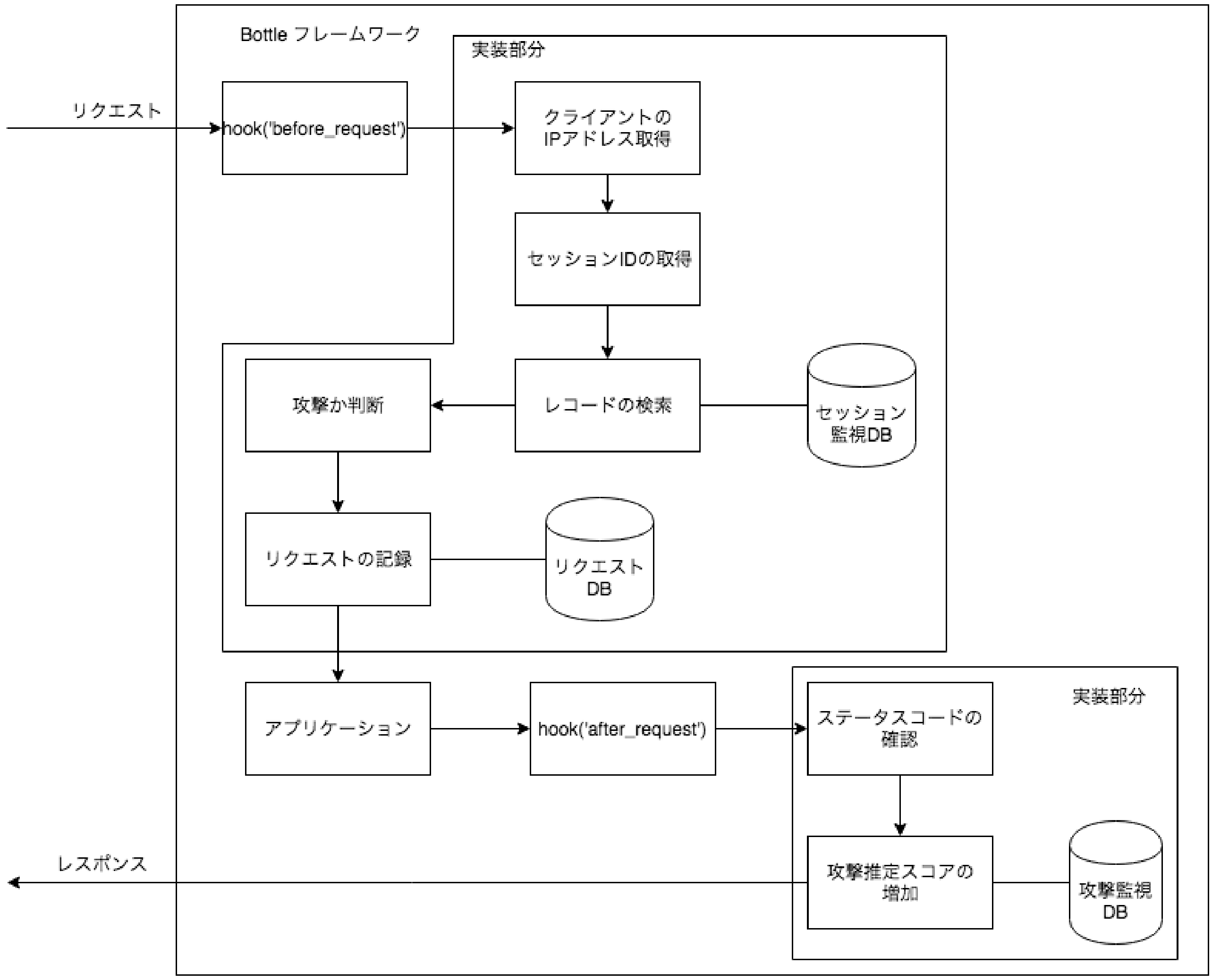

図 2　本システムの動作概要図.

Fig. 2　Schematic views of the system

によって記録・検出を試みた.

3.3.1　before_request 部分

リクエストを受け取った直後にこのシステムは以下のように動作する.

1. クライアントの IP アドレス取得
2. (初回)IP アドレスに基づくセッション ID の作成
3. IP アドレスとセッション ID からレコードを検索
4. レコードから攻撃者かどうか確認
5. (初回，3 でレコードがない時) レコードの追加
6. リクエストの記録・書き換え

クライアントが初めてアプリケーションサーバに接続すると，クライアントの情報を取得したのち，16 進 32 桁のランダムな値を識別子をその IP アドレスと紐付けする (この ID の有効期限は設定ファイルから変更することができる)．その後，IP アドレスと生成した ID を元にセッション情報を管理データベースに保存する．保存するのは，クライアントの IP アドレス，紐付けられた ID，攻撃者の判定 (初回は 0，上記の攻撃判定により攻撃者とみなされると 1 になる)，タイムスタンプである．その後，攻撃者かどうかを判定し攻撃者であれば，HTTP リクエストから情報を取得する．具体的には，HTTP リクエストのヘッダから接続メソッドとパス，POST メソッドの場合そのパラメータを取得する．取得したのち，リクエストを変更することで，アプリケーションに接続しているデータベースを変更する．この時，データベースのテーブルは，変更前と変更後で同一の動作をするため，同じ構造になっている．しかし，攻撃者と判定した場合データベースには正規運用のデータベースを読み出さないため，データベース内にある情報の安全性を確保することが可能である．

3.3.2　after_request 部分

リクエストの解決後は以下のように動作する.

1. レスポンスコードの確認
2. (400 番台であれば)400 番台リクエストの回数を 1 増加させる

リクエストが解決すると，レスポンスが送信される．その一つがレスポンスコードである．レスポンスコードはレ

スポンスの状態を示す 3 桁の整数であり，1 桁目が大まかな状態，2 桁目と 3 桁目が詳しい状態を示す数値である．この情報を元に攻撃かどうか判定を行い，攻撃の疑いがあると推測されれば 400 番台リクエストの回数というデータベースで記録している値を増加させる．その後，クライアントにレスポンスが送信される．この実装はまとめるは上ページの図 2 のようになる．

4. 結果と今後の課題

4.1 結果

Bottle フレームワークを用いた本システムについて以下のことが可能であることがわかった．

まず 1 つ目にデータベースと接続し，データベースに情報を保存，またその情報を取り出したり上書きしたりすることが可能ということがわかった．

2 つ目にクライアントの IP アドレスやリクエストヘッダ，パラメータを取得できることがわかった．Bottle フレームワークは，これらの情報を request.environ クラスに辞書形式で格納するため，そのクラスから必要な情報を取得することが可能である．レスポンスについてもリクエストをアプリケーションで解決したのちに取得可能であり，レスポンスコードは response.status から，取得可能であるとわかった．

3 つ目に，リクエストの変換．これも request.environ クラスを利用する．request.environ['PATH_INFO'] にクライアントが要求したパスが格納され，それを元にルーティングを行うことがわかったため，攻撃と判定された時，このパス情報を書き換えることで，リクエストを変換することができた．

ローカル環境で実際に 400 番台の通信を規定回数以上行った結果，行ったクライアント (今回は自身) の IP アドレス等の情報とリクエストの情報を取得することが可能であることがわかった．また，攻撃者と推測した後，リクエストパスを変換することも可能であった．

4.2 今後の課題

4.2.1 Web 上への設置

現在，アプリケーション自体は実装済みだ，実際に Web 上に設置していないため，攻撃の収集にいたれていない．開発したアプリケーションを Web 上に設置して攻撃検出実験を行う必要がある．そして，その実験から攻撃検出方法が有効かどうか記録を収集できるかを確かめる必要がある．

4.2.2 攻撃検出手法

今回の攻撃検出方法をレスポンスコードが 400 番台であるような通信を規定回数以上行うこととしたが，これ以外にも WAF では検出しづらくアプリケーション単位でみれば検出できる攻撃検出手法が存在すると考えられる．例えば不正なページ遷移に対する検出はその一つであると考えられる．アプリケーションが取得可能な情報である，リクエストパスとそれに対するレスポンスヘッダを用いてユーザの直前のページを記録し，アプリケーションのページ遷移をあらかじめ作成しておき照合することで攻撃を検出可能ではないかと考えている．その他にも様々な攻撃検出手法を実装することで有効な検出手法を確かめることが課題である．

5. おわりに

本論文では，Web アプリケーションに高対話型ハニーポットを搭載することで，攻撃の検出と記録，そして攻撃の影響低減を行うことを提案をした．Bottle フレームワークを用いてローカル環境で実装を行ったところ，データベースへの接続，クライアントの情報を取得，リクエストの変換が可能だということがわかった．Web 上で監視することで，攻撃検出を行いこのシステムの有効性を評価すること，そして攻撃検出手法の改善をすることが今後の課題である．

謝辞　本研究は，国立研究開発法人科学技術振興機構（JST） 戦略的国際共同研究プログラム（SICORP）および JSPS 科研費 JP18K11295 の助成を受けたものである．

参考文献

[1] 野見山賢人，小出　洋: Web アプリケーションのための攻撃検出・防御システム Hoppin の設計・実装，第 117 回プログラミング研究会 (2018).

[2] Trend Micro Inc.: 標的型サイバー攻撃 | トレンドマイクロ - Trend Micro https://www.trendmicro.com/ja_jp/security-intelligence/research-reports/threat-solution/apt.html

[3] 澁谷芳洋，小池英樹，高田哲司，安村通晃，石井威望: 高対話型おとりシステムの運用経験に関する考察．情報処理学会論文誌，Vol. 45 No.8. (2004). https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_action_common_download&item_id=10839&item_no=1&attribute_id=1&file_no=1

[4] T ishikawa and K sakurai: Web Application Deception Proxy, Proc. IMCOM'17, pp. 74:1-74:9 (2017).

[5] 佐藤 聡, 佐藤 聖, 中井 央, 新城 靖: TLS/SSL プロトコルを対象にした汎用ハニーポットシステムの実装と HTTPS による収集結果. 情報処理学会研究報告, Vol.2015-CSEC-69 No.18, pp. 4-5 (2015)

[6] N Provos, D McNamee, P Macrommatis, K Wang and N Modadugu: The ghost in the browser analysis of web-based malware, HotBots '07 (2007).

[7] 独立行政法人情報処理推進機構: Web Application Firewall（WAF）読本改訂第 2 版第 3 刷．pp. 7:3-14:12 (2011). https://www.ipa.go.jp/files/000017312.pdf.

[8] 独立行政法人情報処理推進機構: テクニカルウォッチ：『新しいタイプの攻撃 (APT)』に関するレポート (2010) https://www.ipa.go.jp/files/000009366.pdf.

[9] Deniz Akkaya, Fabien Thalgott: Honeypot in Network Security. pp. 3:3-3:14 (2010) http://www.diva-portal.org/smash/get/diva2:327476/fulltext01.pdf.

[10] Nginx https://nginx.org/en/.
[11] Marcel Hellkamp: Bottle: Python Web Framework(2019). http://bottlepy.org/docs/dev/.
[12] The Internet Society: 「The Referer[sic] request-header field allows the client to specify [⋯] the address (URI) of the resource from which the Request-URI was obtained [⋯]」 RFC2616 14.36 (1999) https://tools.ietf.org/html/rfc2616