**平成２９年度国立情報学研究所クラウド利活用実証実験成果報告書**

Ⅰ．基本情報

（１）実験代表者

|  |  |
| --- | --- |
| 氏名 | 小出　洋 |
| 所属 | 九州大学情報基盤研究開発センター |
| 職名 | 教授 |

（２）実験参加者

|  |  |
| --- | --- |
| 氏名 | 野見山賢人 |
| 所属 | 株式会社CyberZ（実験当時　九州工業大学大学院情報工学府） |
| 職名 | エンジニア（実験当時　大学院博士前期課程） |

（３）実験課題名

|  |
| --- |
| ネットワークアプリケーションにおける攻撃者の攻撃傾向の検出実験 |

Ⅱ．実証実験の成果概要

本実証実験では，既存のWebアプリケーションを用いて攻撃者を誘導することにより，Webアプリケーションに対する攻撃手法の収集・分析・Webアプリケーションの保護を可能にすることを目的とする．その目的を達成するために，Webアプリケーションに疑似APIの機能を付加し，アプリケーションをハニーポット化することで攻撃検出を行う手法を提案する．近年，情報システムに対する攻撃手法が変化してきている．特に特定の組織を対象とした標的型攻撃が増加しており，攻撃の主流となっている．その一方でWebアプリケーション等の情報システムは一部クラウド環境等にも用いられるようになり，情報システムのシステム構成等がオンプレミス環境での単独利用に比べ複雑となっている．そのため，現在運用されている情報システムを把握することは難しく，セキュリティ機能を持たせることは困難である．以上のことから，運用されているWebアプリケーションを変更せず，密に連携したセキュリティ機能を持つことが必須である．

|  |
| --- |
| 「II. 実証実験の成果概要」の内容について、国立情報学研究所が実証実験の成果として外部公開することに同意頂ける場合は以下のチェックボックスをチェックしてください。  ■同意する。 |

Ⅲ．実証実験の成果詳細

1. はじめに

近年，情報システムに対する攻撃手法が変化してきている．特に特定の組織を対象とした標的型攻撃が増加しており，攻撃の主流となっている．標的型攻撃は，メールの添付ファイルの実行や改竄された Web サイトの訪問等を通じてマルウェアを感染させる手法を始め，対象の情報システムの内部に侵入するために巧妙な攻撃手法が用いられている．特に Webサイトの改竄においてはクロスサイトスクリプティングやSQLインジェクションなどを用いてWebページの改竄を行い，改竄されたWebページを利用者が訪問することでマルウェアを配布する悪性Webサイトに誘導し，マルウェアを利用者端末にダウンロードするドライブ・バイ・ダウンロード攻撃[10]に発展する場合もある．

その一方でWebアプリケーション等の情報システムは組織内部に設置されているオンプレミス環境以外にも，一部クラウド環境等にも用いられるようになっている．図1.1にクラウド環境を併用した場合のシステム構成を示す．一部をクラウド環境に移行することでオンプレミス環境のみと比較し，情報システムのシステム構成やネットワーク構成が，より大規模かつ複雑になっていることから，現在運用されている情報システムのすべてを把握することはオンプレミス環境のみの場合と比較した際，難しくなる．そのため，運用されているWebアプリケーションを変更せず，Webアプリケーションと密に連携したセキュリティ機能を持たせるようにすることが必須である[12]．Webアプリケーションを変更せず既存の情報システム以上にセキュリティ機能を満たすシステムとして，Web Application Firewall（WAF）[15]が挙げられる．

WAFはWebアプリケーションをクロスサイトスクリプティングやSQLインジェクションなどから保護するシステムである．WAFはPOSTメソッドやGETメソッドなどから得られるパラメータなどの情報から，攻撃の検出を可能にしており，攻撃を検出すると通信の遮断などの対応を行う．しかし，全ての攻撃からWebアプリケーションを保護することは難しい．例えば，他のユーザになりすます不正ログインを行う攻撃は，WAFでは正常な通信として処理されてしまうため，WebアプリケーションをWAFで保護することは難しい．

本実証実験では，既存のWebアプリケーションを用いて攻撃者を誘導することにより，Webアプリケーションに対する攻撃手法の収集・分析・保護を可能にすることを目的とする．その目的を達成するために，Webアプリケーションに疑似APIの機能を付加し，ハニーポット化することで攻撃検出を行うシステムである，Hoppin（Honeypot in web application framework）を提案する．ハニーポットとはアプリケーションの脆弱性により攻撃者を誘い出し不正アクセスを促すシステムである．ハニーポットを利用することにより攻撃者の攻撃行動の観察や分析を行うことが可能となる．Hoppin の設計・実装を行い，未使用時とのレスポンス時間の比較を行ったり，実際にクラウド環境上に設置して攻撃手法の収集を行ったりすることで，現状の問題点を確認した．

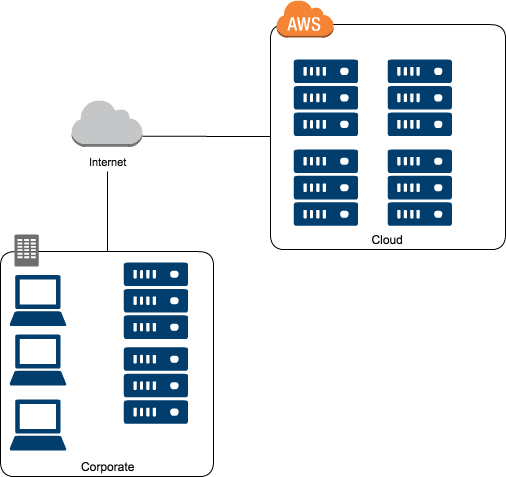


図1.1 クラウド環境併用時のシステム構成．

本報告書の構成は以下の通りである．第2章では，既存のWebアプリケーションを保護するシステムであるWAFについて機能の特徴，問題点について述べる．第3章では，攻撃行動の観察に用いられるハニーポットについての特徴について述べる．第4章では，本実証実験で提案している攻撃検出システムHoppinについて，設計とその実装について述べる．第5章では，実装したHoppinについてレスポンス時間の比較やWAFとの比較を行い，クラウド環境上に設置した際の有用性の評価を行った．その結果について考察を述べる．最後に第6章で，まとめと今後の課題について述べる．

1. Web Application Firewall

Webアプリケーションの脆弱性を利用した攻撃を防ぐためには「脆弱性を作り込まないこと」，「脆弱性が見つかったら修正すること」が重要である．しかし，修正・改修を行えないWebアプリケーションなど脆弱性の対策が困難な場合においてはWeb Application Firewall（WAF）[15]が有用である．WAFは，クロスサイトスクリプティングやSQLインジェクションなどといった，Webアプリケーションの脆弱性を利用した攻撃からWebアプリケーションを保護するシステムである．WAFを導入することでWebアプリケーションが所持している脆弱性を利用した攻撃を検出し，通信遮断及びWebアプリケーションを保護することが可能となる．図2.1にWAFの動作概要を示す．

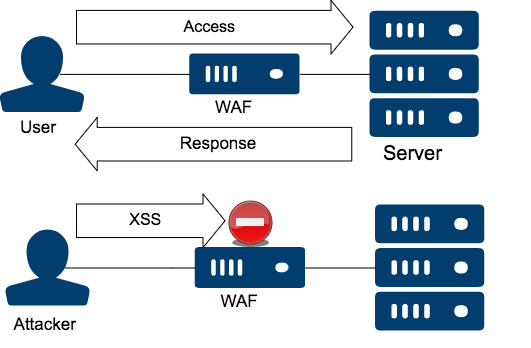


図2.1 WAFの動作概要.

WAFはHTTPリクエストやHTTPレスポンスから得られる情報を基に検査を行う検査機能，攻撃が検出された通信に対する処理を行う処理機能，WAFの動作の記録を行うログ機能の３つから成り立つ．

* 1. 検査機能

検査機能はHTTPリクエストやHTTPレスポンスを検出パターンに基づき検査を行う機能である．

HTTP通信における検査箇所は，HTTPリクエストではHTTPメソッド，URI，プロトコルのバージョンが示されているリクエストライン，リクエストヘッダの２箇所であり，POSTメソッドの場合は合わせてボディ部も行う．HTTPレスポンスでは，ステータスコード，レスポンスヘッダ，ボディ部の３箇所を行う．検査機能による攻撃の検出には，検出パターンに基づき通信のパラメータ値を比較するため，目視で判定する場合とWAFで判定する場合とでは異なった結果が生じる場合がある．そのため，悪意のある攻撃の検出が行われない場合や，正規のユーザが接続された際に誤検知され通信遮断される場合もある．そのような誤検知を防ぐためにWAFではブラックリストやホワイトリストを用いることで誤検知率を下げることが可能である．

ブラックリストは，通信のおける攻撃パターンを定義したリストである．ブラックリストを利用して検出を行うと，通信内容がリスト内のパターンと一致した場合に対象の通信を不正通信として判断する．最新の攻撃を検出する場合には随時リストの更新が必須となる．

ホワイトリストは，通信における正規パターンを定義したリストである．ホワイトリストを利用して検出を行うと，通信内容がリスト内のバターンに一致しない場合に対象の通信を不正通信として判断し，検出する．ホワイトリストの作成には，Webアプリケーションにおけるパラメータの設計に基づき作成し依存する．そのため，Webアプリケーション毎にリストの作成を行う必要がある．図2.2にブラックリスト・ホワイトリストを用いた動作の概要を示す．

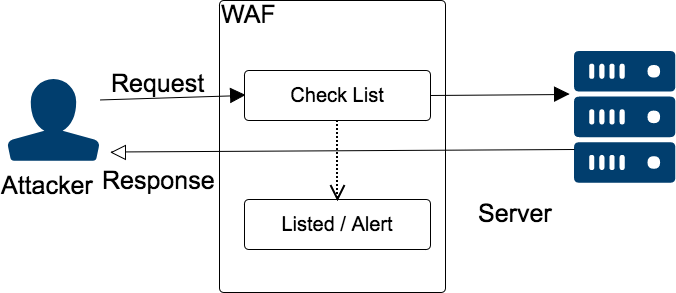


図2.2 ブラックリスト・ホワイトリストを用いた動作の概要．

* 1. 処理機能

処理機能は，検査機能により検出された通信に対して次のような処理を行い，Webアプリケーションや利用者端末にHTTPリクエストやHTTPレスポンス等の情報を送信している機能である．WAFが行える処理として，“通過処理”，“エラー処理”，“遮断処理”，“書き換え処理”の４種類が利用可能である．

“通過処理”は検出された通信をWAFによる処理を行わず，そのまま送信する処理である．一般的にWAFの導入時の検証や検証の記録に用いられる．図2.3に通過処理におけるWAFの動作概要を示す．

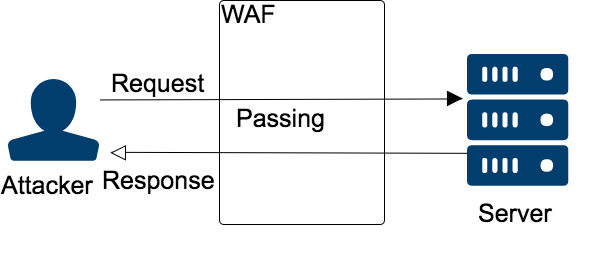


図2.3 通過処理の動作概要．

“エラー処理”は検出された通信を送信せず，WAF側でHTTPステータスコード403などのエラー応答を含めた通信を生成し送信を行う処理である．エラー応答のページ内容はWAF側で設定を行うことが可能であり，任意の内容のページを生成し送信することが可能である．図2.4にエラー処理におけるWAFの動作概要を示す．

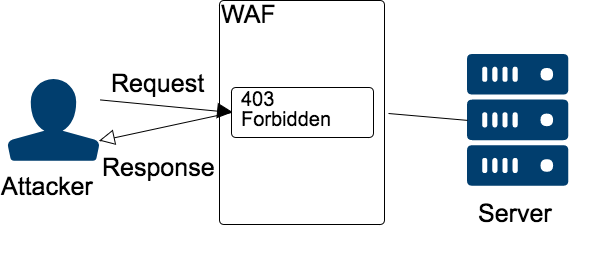


図2.4 エラー処理の動作概要．

“遮断処理”は検出された通信を意図的に宛先に送信を行わない処理である．遮断処理を行う際には通信切断の応答を送信する方法や応答しない方法が用いられる．図2.5に遮断処理におけるWAFの動作概要を示す．

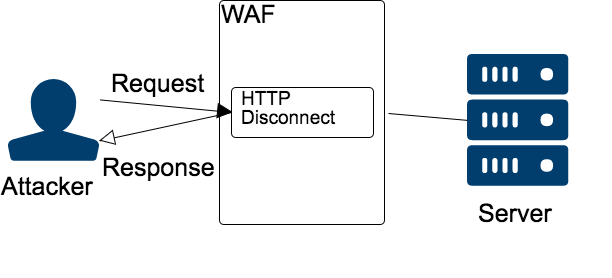


図2.5 遮断処理の動作概要．

“書き換え処理”は検出された通信の一部を書き換え送信する処理である．クロスサイトスクリプティングやSQLインジェクションなどが通信上に存在している通信を継続する場合に用いられる．また，HTTPレスポンスにおいては，不必要な情報を含む通信を継続する場合に用いられることもある．図2.6に書き換え処理におけるWAFの動作概要を示す．

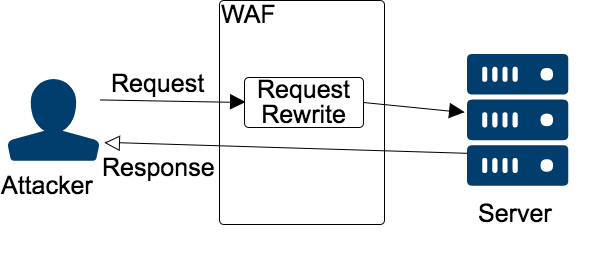


図2.6 書き換え処理の動作概要．

* 1. ログ機能

ログ機能は，検査機能や処理機能で行ったWAFの処理を記録している機能である．ログは“監査ログ”と“動作ログ”の２種類があり，それらはファイルやデータベース上に格納される．“監査ログ”は，検査機能から検知された通信とその処理について記録されるログである．記録される内容は次の情報などが挙げられる．

* 日時
* 行った処理
* 接続元IPアドレス
* 接続先パス
* 検知箇所
* 検知パターン

“動作ログ”は，WAFが行った動作の情報が記録されるログである．動作情報の中にはエラー情報も記録される．記録される内容は，日時，WAFの動作情報，エラー情報などが挙げられる．動作ログはシステムの担当者がWAFの動作を確認するために用いられる．

* 1. 拡張機能

拡張機能は，WAFが搭載している基本機能の他に独自に実装を行っている機能である．独自に実装を行っているため，WAF毎に所有している機能は異なる．代表的な機能として，通信の正当性を確認する通信確認機能と管理の利便性を高める管理機能がある．

* + 1. 通信確認機能

通信確認機能は，セッション上のHTTP通信におけるパラメータやリクエスト情報における正当性を確認する機能である．パラメータにおける正当性確認は，HTTPレスポンスの特定パラメータをWAF側に持たせることで，次のHTTPリクエストが送信された際にパラメータと送信されたパラメータの値が一致するかの確認を行っている．これにより，パラメータによる不正を防ぐことが可能となる．リクエスト情報における正当性確認は，送信したリクエスト情報が正規のリクエストであるかを確認する機能である．確認には，HTTPレスポンスを送信する際にWAF側で利用者に予測されない情報を付与し，次のHTTPリクエストが送信された際に付与された情報が送信されているかの確認を行っている．これにより，クロスサイトリクエストフォージェリ(CSRF)攻撃により攻撃者が用意したサーバに誘導することを防ぐことが可能となる．また，HTTPヘッダに含まれるリファラヘッダがWAF側で定義したパターンに一致するか確認することにより，画面遷移型の攻撃にも対応が可能となる．

2.4.2　管理機能

管理機能は，システム担当者が管理を行う際の利便性を高めるために用いる機能であり，システム担当者が“監査ログ”を解析し，不正な通信の件数の確認や，検査機能における検出パターンの更新の手間を省くことが可能である．管理機能において提供されている機能は以下の機能である．

* レポート機能
* 通知機能
* リストの自動生成・更新機能

レポート機能はログ機能において記録されているログ情報を解析し、その情報をレポートとして出力する機能である．出力されるレポートは一定期間毎に生成され，不正通信と判断された回数，IPアドレス，接続先URL，検知理由等の情報が記述されている．通知機能は検査機能において不正通信を検出した際，システム担当者にメールなどの手段により，通知を行う機能である．　　図2.7に通知機能の動作概要を示す．

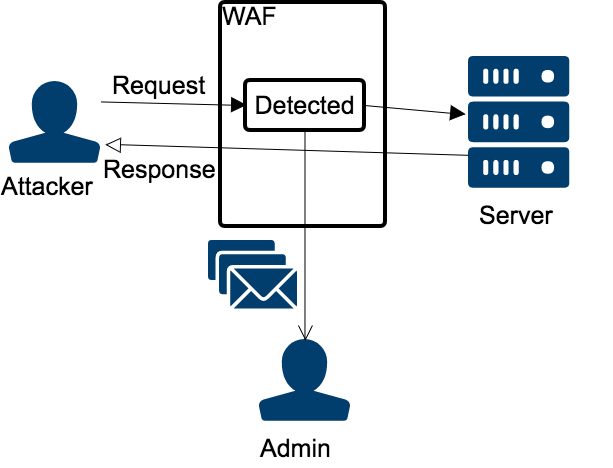


図2.7 通知機能の動作概要．

リストの自動生成・更新機能はWAFを経由する通信や新種の攻撃手法が発見された場合にホワイトリストやブラックリストを更新する機能である．自動生成・更新機能を用いることにより，Webアプリケーション毎にリストの作成を行う必要があるホワイトリストの短所や，最新の攻撃を検出する場合には随時リストの更新が必須となるブラックリストの短所を補うことが可能となる．

* 1. WAFの問題点

WAFの基本機能である，検査機能・処理機能による検出を行っているだけでは対応することが難しいものもある．例えば，特定の利用者のみが利用可能な機能が全ての利用者に対して使用可能な状態になっている場合や，辞書攻撃やパスワードリスト攻撃といったブルートフォース攻撃を用いて，他人になりすまし不正にログインするような攻撃に対してはWAFでは正常な通信として判断される．また，ブラックリストによる検知機能を使用した場合は未知の攻撃への対処に対して対応することは不可能である．このため，すべての攻撃をWAFを用いて検出，保護することは難しい．

1. ハニーポット

ハニーポットは未対策のアプリケーションやOSに存在する脆弱性をシステム上に用意し，攻撃者をシステムへ誘導することで，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ことで，攻撃者の行動や攻撃手法といった情報を収集することが可能である．ハニーポットには，攻撃者からの接続により初めてハニーポットとしての役割を持つ対話型ハニポットがある．

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．図3.1にハニーポットの動作概要を示す．

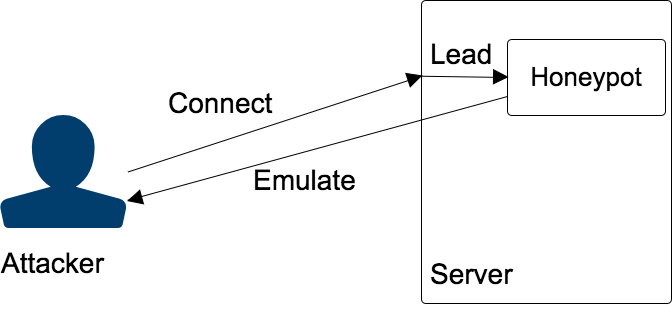


図3.1 ハニーポットの動作概要．

* 1. 低対話型ハニーポット

低対話型ハニーポットは特定のプロトコルや脆弱性を持ったアプリケーションを疑似的に模倣することで，攻撃者をエミュレーション環境に誘導し攻撃活動を促すハニーポットである[14]．低対話型ハニーポットの特徴として，攻撃者が接続したシステム上で行っている動作やコマンドはシステム側でエミュレートされているものであるということが挙げられる．攻撃者はシステムに用意された疑似コマンドを利用するため，攻撃者側の画面上では攻撃が成功しているように見えているが，実際には実行されていないコマンドのためホストマシンへの影響はない．代表例として SSHハニーポットのcowrie[2]やHTTPハニーポットのglastopf[9]などが挙げられ，それらを統合したハニーポットとしてT-Pot[3]が存在する．

3.1.1 cowrie

cowrie[2]はMichel Oosterhof氏が開発を行っている，Pythonで記述されたオープンソースソフトウェアのSSHハニーポットである．cowrieは，LinuxディストリビューションであるDebian5.0のファイルシステムを真似ており，初期状態ではインストール完了直後の状態で動作している．このファイルシステムはエミュレート対象のファイルを追加・削除を行うことで，より実環境に近づけさせることが可能となる．また，SSHに接続する際に用いるパスワードはユーザDBとして用意されており，cowrieの設置者が自由に追加が可能となっている．ユーザ名とパスワードは1対多対応することで多くの攻撃者の接続に対応している．図3.2にユーザDBの情報を示す．

|  |
| --- |
| root:x:!root  root:x:!123456  root:x:\*  richard:x:\*  richard:x:fout |

図3.2 ユーザDBの情報．

3.1.2 glastopf

glastopf[9]はMushMush 財団が開発を行った，Pythonで記述されているオープンソースソフトウェアのHTTPハニーポットである．glastopfは，攻撃タイプ別にWebアプリケーションをエミュレートしており，include処理の脆弱性を利用し，リモートファイルを実行するようなリモート・ファイル・インクルード攻撃[13]やクロスサイトスクリプティング，SQLインジェクション等といった攻撃を行える疑似環境が用意されている．この機能が存在するため，攻撃者の幅広い攻撃手法を取得することが可能となる．

3.1.3 T-Pot

T-Pot[3]はドイツテレコム社が開発を行った，複数のハニーポットを用いて様々なサービスから攻撃者の行動を記録し，可視化を行うプラットフォームである．T-Pot上で運用されているハニーポットはすべて Dockerにより管理されており，構築に用いるDockerfileはT-Pot用にカスタマイズされた物を用いている．図3.3にT-Potの動作概要を示す．

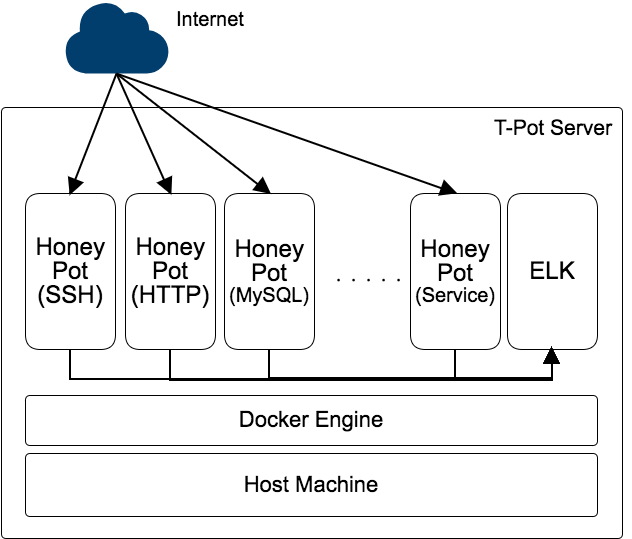


図3.3 T-Potの動作概要．

各ハニーポットのコンテナで収集された情報はELKコンテナに格納され可視化される．ELKは収集されたデータを元に全文検索を可能にするエンジンであるElasticSearch[5]，データの収集を行うLogstash[7]，可視化エンジンのkibana[6]で構成される．kibanaでは収集されたデータをタグクラウド形式やグラフ等を用いて収集されたデータを確認することが可能である．図3.4にSSHハニーポットで収集されたユーザ情報の可視化結果を示す．これにより，T-Potの運用者は各ハニーポットのログを確認することなく，容易に収集された情報を確認することが可能となる．



図3.4 攻撃者が用いたユーザ情報の可視化結果．

3.2 高対話型ハニーポット

高対話型ハニーポットは実際の物理マシンや仮想マシン上に脆弱性を持ったアプリケーションやOSを配置し動作させることにより，攻撃者をシステムに接続させ攻撃活動を促すハニーポットである[14]．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．そのため，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もある．このことから高対話型ハニーポットを設置する上で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周りを考慮することが必須となるため，現実的には運用が難しい．

1. 攻撃検出システムHoppin

4.1 概要

本実証実験では，攻撃検出機能を搭載することで，攻撃からWebアプリケーションを保護し，攻撃手法の収集を行うシステムHoppin(Honeypot in Web Application Framework)を提案する．HoppinはHTTPリクエストを受け取った際にユーザ属性の状態に応じてコンテンツ側のAPIを正規用と疑似用に切り替えることで，Webアプリケーションを攻撃から保護することが可能である．また，攻撃手法の収集も行っており，攻撃傾向の分析が可能となる．

Hoppin の各システム及びコンテンツはLinux コンテナ技術である Docker[4]を用いて構築している．DockerはDocker社が開発したコンテナ型の仮想化環境を提供するオープンソースソフトウェアである．コンテナ型はホストOS上に仮想化されたハードウェア上で動作するハイパーバイザ型やホスト型とは異なり，プロセス環境上においてファイルシステムやネットワークインターフェース，名前空間を独立するようにさせて動作させるものである．ホストOSの1プロセスとして動作するため，軽量であり，特定の環境に依存することなく動作させることが可能である．

Dockerを用いることで，各コンテナ内で実行されているプロセスはコンテナ内部で完結されており，他のコンテナに対して影響を与えない．そのため，本実証実験で提案するHoppinは高対話型のハニーポットの機能を有しているが，低対話型のハニーポットと同等の安全性を担保している．したがって，従来では運用が困難であった高対話型ハニーポットとしての運用を可能とした．

4.2 システム構成

Hoppinでは，攻撃検知を行うリバースプロキシ機能を搭載した攻撃検出システムと収集した情報の閲覧・分析を行う管理システムの２種類から成り立っている．管理システムと攻撃検出システムは分離して動作させることも可能であるため，攻撃検出システムのみを搭載した端末とそれらの情報を集約する管理システムに分けて運用することも可能である．攻撃検出システムで収集した情報は管理システムのデータベース上で記録されており，管理システム上で確認することが可能である．管理システムのデータベースはセッション情報とリクエスト情報を格納するRDBと接続情報のキャッシュとPOSTメソッドの場合のパラメータを格納するNoSQL DBで構成されている．なお，管理システムに関しては，接続可能なIPアドレスを限定する設計を行うことにより，攻撃者からの攻撃を受けないようにする．

コンテンツは最低1つでも動作するが，複数個設置することで攻撃者に1つのコンテンツがハニーポットと判断された場合において，再接続の際に，他のコンテンツに接続させて引き続き攻撃活動を誘導することが可能となる．これは，接続状態のキャッシュが無効になった場合に接続を行うと初回接続と同様とみなして通信を行うためである．また，コンテンツに外部からの接続を行うことは不可能であり，攻撃検出システムを介してからの接続でのみ接続が可能となる．図4.1にHoppinのシステム構成を示す．

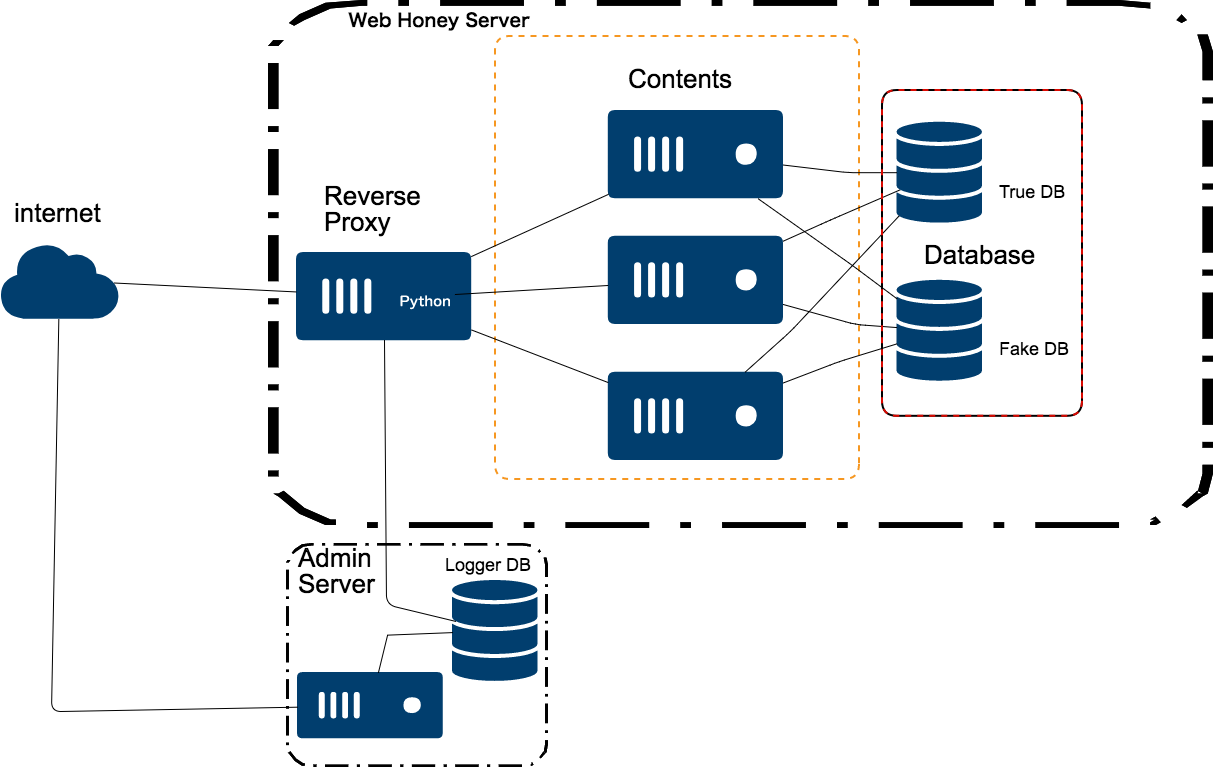


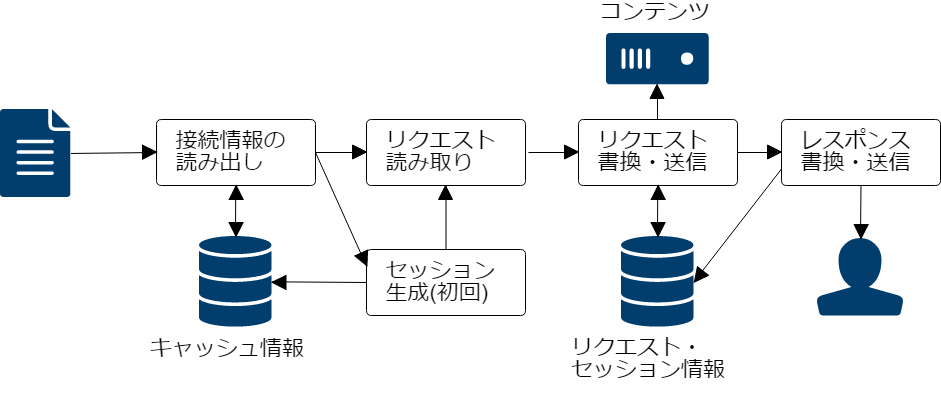
図4.1 システム構成．

4.2.1攻撃検出とその処理

ユーザがHoppinに接続を行う場合，攻撃検出システムを介して各コンテンツに接続を行う．その際に行う処理の流れは次の通りである．

1. 接続先情報の読み出し
2. コンテンツと接続元IPアドレスの紐付け(初回接続時のみ)
3. HTTPリクエストの読み取り
4. セッション情報の記録
5. HTTPリクエスト情報の記録・書き換え
6. コンテンツにHTTPリクエストを送信
7. HTTPレスポンスを書き換え
8. ユーザにHTTPレスポンスを送信

図4.2にHoppinの処理の流れを示す．

図4.2 Hoppinの処理の流れ．

4.2.2 接続先情報の読み出し

ユーザが初めてHoppinに接続を行った場合，攻撃検出システムは接続元IPアドレスをキーとしてランダムに決定されたコンテンツを紐付けし，セッション情報とともに JSON でキャッシュを行う．これによりシステムが指定したコンテンツに一定時間接続することが可能となる．2回目以降の接続はキャッシュされた情報を読み出し，その情報を用いて接続を行う．セッション情報を識別する値はUUID Version4 によりランダムに生成され，コンテンツ情報はシステムに登録されたものからランダムに決定をしてそのポート番号を格納する．キャッシュされる情報を図4.3に示す．

|  |
| --- |
| "Session": "da275c8d4ffd4d6a9eebfc1900e911ee",  "Port": 8000 |

図4.3 紐付けたコンテンツのキャッシュ情報．

4.2.3 情報の読み取り・取得

接続先情報が読み出された後，HTTPリクエストの読み取りを行い，セッション情報を管理データベースに記録を行う．HTTPリクエストの読み取り後，ユーザ属性の状態に応じて情報の記録・HTTPリクエストの書き換えを行う．ユーザ属性の状態は通常モードと記録モードの2種類ある．通常モードは，一般ユーザがシステムに対し接続を行う際の用いるモードであり，一般的には通常モードが使用される．記録モードは，ユーザがシステムに対して攻撃を続けた際に，通常モードから切り替えを行い攻撃情報の収集を行うモードである．ユーザモードの切り替えはモードスコアの値により判断を行う．モードスコアの値はHTTPレスポンスを確認し，認証が必要なページに認証なしで接続する場合に返される403や，ファイルが存在しない場合に返される404などといった場合に増加する．これは，通常利用では起こり得ないHTTPステータスコードが生じることで，攻撃の疑いがあると推測を行っているためである．また，切り替えの基準となるモードスコアの値は自由に変更することが可能であり，誤検知が多くなった場合など柔軟な運用が可能である．攻撃者が各コンテンツに用意した認証ページに対してブルートフォース攻撃により複数回認証に失敗する場合も同様にモードスコアの値を増加させる．これにより，該当するセッションに対して通信情報の記録を行うことが可能となる．

4.2.4 記録モードとHTTPリクエストの書き換え

記録モードにおける通信情報は，ユーザが接続する度にHTTPリクエストから情報の取得を行っており，その情報を管理データベースに記録している．HTTPリクエストのヘッダには最低，接続メソッド，接続パス，ホスト名，ユーザが接続する際に用いるハードウェアやソフトウェアの情報が記載されているユーザエージェント，どのようなデータを受け取りたいかを記すAccept情報が記載されている．図4.4にHTTPリクエストのヘッダを示す．

|  |
| --- |
| GET /api/home HTTP/1.1  Host: localhost:8000  User-Agent: curl/7.52.1  Accept: \*/\* |

図4.4 HTTPリクエストヘッダ．

取得する情報としては，接続パス，接続メソッドの2種類である．接続メソッドがPOSTメソッドの場合に関してはパラメータの情報も合わせて取得を行う．情報の取得を行う際には正規表現を用いており，マッチしたグループの情報を参照することで情報の取得を可能にしている．またPOSTメソッドの場合のパラメータはHTTPリクエストを受け取った際ボディ部分に存在するため，ヘッダとボディを分けている改行コードの位置で文字列を分割して管理データベース上に記録を行っている．しかし，受け取るパラメータが接続するパス毎に異なるため，UUID Version4で一意にキーを生成してNoSQLのデータベースに格納している．これにより，異なるパラメータへの対応が容易となる．正規表現におけるマッチングの条件を図4.5に示す．図4.5に示すグループは，前者が接続メソッド，後者が接続パスである．

|  |
| --- |
| r'([A-Z]{3,6}) (.\*) HTTP/1.1' |

図4.5 正規表現のマッチング条件．

HTTPリクエストの書き換えは通常モードではポート番号のみ書き換えているが，記録モードではポート番号に加えて接続パスの情報を書き換えている．接続パスの書き換えはコンテンツ側の疑似APIに接続を行わせるために書き換えを行う．書き換えは正規APIのパスの一部を正規表現でマッチングを行い，マッチしたものに対して疑似APIに書き換えを行う．これにより攻撃者には正規APIで接続を行っていると錯覚させることが可能となり，より高度な攻撃を誘導することが可能になる．

4.2.5 HTTPレスポンスの修正

コンテンツより受け取ったHTTPレスポンスはボディの情報をユーザが利用可能なようにポート番号の修正を行う必要がある．例えば，ページ遷移時のリンクの情報がコンテンツから受け取った情報をそのままユーザに送信するとコンテンツのポートに対して接続を行い，攻撃検出システムを介さないため，ユーザのクライアントからコンテンツに接続を行うことが不可能になる．ポート番号の修正に関しては正規表現を用いてコンテンツポート番号とマッチしたものを攻撃検出システムが用いているポート番号に書き換えて送信を行う．これにより，攻撃検出システムを介して接続を行うことでユーザがコンテンツに接続することが可能となる．

4.3 コンテンツと疑似API

Hoppinで使用するコンテンツは管理者のみ接続が可能なエリアとAPIを用意する．コンテンツに接続されるデータベースは，正規運用中の場合に用いるデータベースと疑似運用中の場合に用いるデータベースの2種類が接続されており，正規APIは正規運用のデータベース，疑似APIは疑似運用のデータベースに接続している．2つのデータベースのテーブル構造はどちらも同一の構造が格納されている．これは攻撃者がSQLインジェクションを行ってシステム側が記録モードに移行した際にテーブル構造の変化によりハニーポットだと判断されることを防止する役割がある．

コンテンツAPIの作成時に攻撃検出システムと連動を行うために正規用と疑似用の2種類のAPIの作成を行う必要がある．その際に設定するルーティング情報は攻撃検出システム側でパス情報が変更された状態でコンテンツに送信されるためパスの統一を行う必要がある．以下はhomeというAPIを実行する場合の一例である．

正規API

/api/home

疑似API

/fake/api/home

正規APIと疑似APIは同一動作を行うため，同じデータ構造を用意する必要がある．疑似API作成にはオブジェクト指向の手法を用いて，正規APIを継承することにより，容易に作成することが可能である．データベースを用いる場合に関してはデータベースへの接続を行うメソッドを作成した後にオーバーライドする形で実装することで正規運用のデータベースと疑似運用のデータベースに接続することが可能である．

図4.6に正規のHomeAPIを，図4.7に疑似のHomeAPIを示す． これらのHome API を実際に実行を行った際のレスポンスとして，図4.8に正規運用時を，図4.9に疑似運用時を示す．

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  class APIHomeController extends Controller  {  protected function ret\_msg （）{  return 'true API for Home';  }  public function index （ Request $req ）{  return response （）->  json （['msg'=>$this->ret\_msg （）],  200 ）;  }  } |

図4.6 正規のHome API．

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  class FakeAPIHomeController extends APIHomeController  {  protected function ret\_msg （）{  return 'Fake API Home';  }  } |

図4.7 疑似のHome API．

|  |
| --- |
| {  "msg": "true API for Home"  } |

図4.8 正規運用のHome APIの実行例．

|  |
| --- |
| {  "msg": "Fake API Home"  } |

 図4.9 疑似運用のHome APIの実行例．

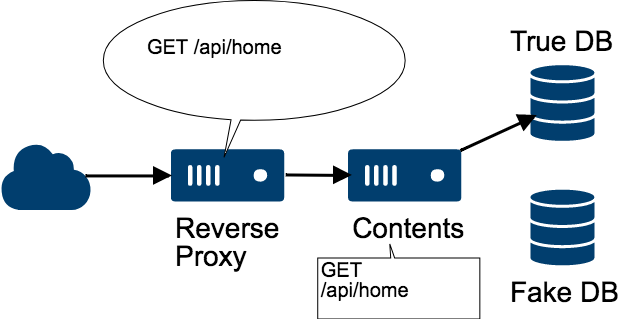


図4.10 正規運用時の接続．

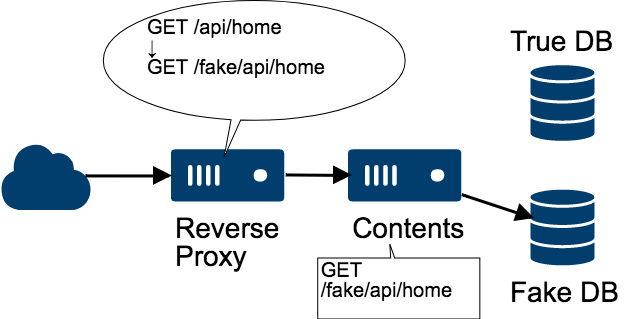


図4.11 疑似運用時の接続．

図4.8，図4.9のように正規運用時と疑似運用時で同一のレスポンス構造にすることにより，APIに直接接続する以外にフロントエンドからのAjaxによる間接的な接続に対応することが可能である．また，フロントエンド側で呼び出した際の処理は正規運用も疑似運用も共通で利用可能であるためユーザインターフェースを崩さずに運用を続けることが可能となる．図4.10に正規運用時の接続の流れを，図4.11に疑似運用時の接続の流れを示す．図4.10，図4.11の例はHomeAPIを実行した場合である．

正規運用時に攻撃検出システムは通常モードのため，正常通信としてシステムが認識しており，受信したHTTPリクエストをコンテンツにそのまま送信している．正規APIに接続しているため，読み出しを行うデータベースは正規運用のデータベースが読み出される．一方，疑似運用時に攻撃検出システムは記録モードに移行しているため，全ての通信情報を記録している．またシステムは正規APIを疑似APIに書き換えを行いHTTPリクエストをコンテンツに送信している．疑似APIに接続しているため，読み出しを行うデータベースは疑似運用のデータベースが読み出される．これにより疑似運用時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の情報の安全性を保つことが可能となる．

1. 評価実験

5.1 レスポンス時間

5.1.1 Hoppin未使用時との比較

コンテンツAPIのレスポンス時間を本実証実験で提案している初期実装HoppinとHoppinを未使用の場合について比較を行った．応答時間の計測には様々なプロトコルを用いてデータの通信を行うcURLを用いてエンドノードのAPIに対して，200回接続しその平均値の値を取った．計測に使用したスクリプトを図5.1に示す．

|  |
| --- |
| for i in {1..200};  do  curl -s -o /dev/null $TARGET -w '%{time\_total}\n' ;  done \  | awk '{sum += ($1) ; count +=1; } END {print "----"; print sum/count }'  fi |

図5.1. レスポンス時間計測スクリプト．

また，計測に使用したAPIは計算時間の短い，文字列を表示するAPIを使用した．計測した結果，図5.2で示す結果となり，初期実装Hoppinは2.437[s]，未使用の場合が0.485[s]となった．2つの場合を比較した際， 初期実装Hoppinの計測値が未使用の場合より約5倍の遅延が発生していることが分かった．図5.3に初期実装Hoppinの動作を示す．

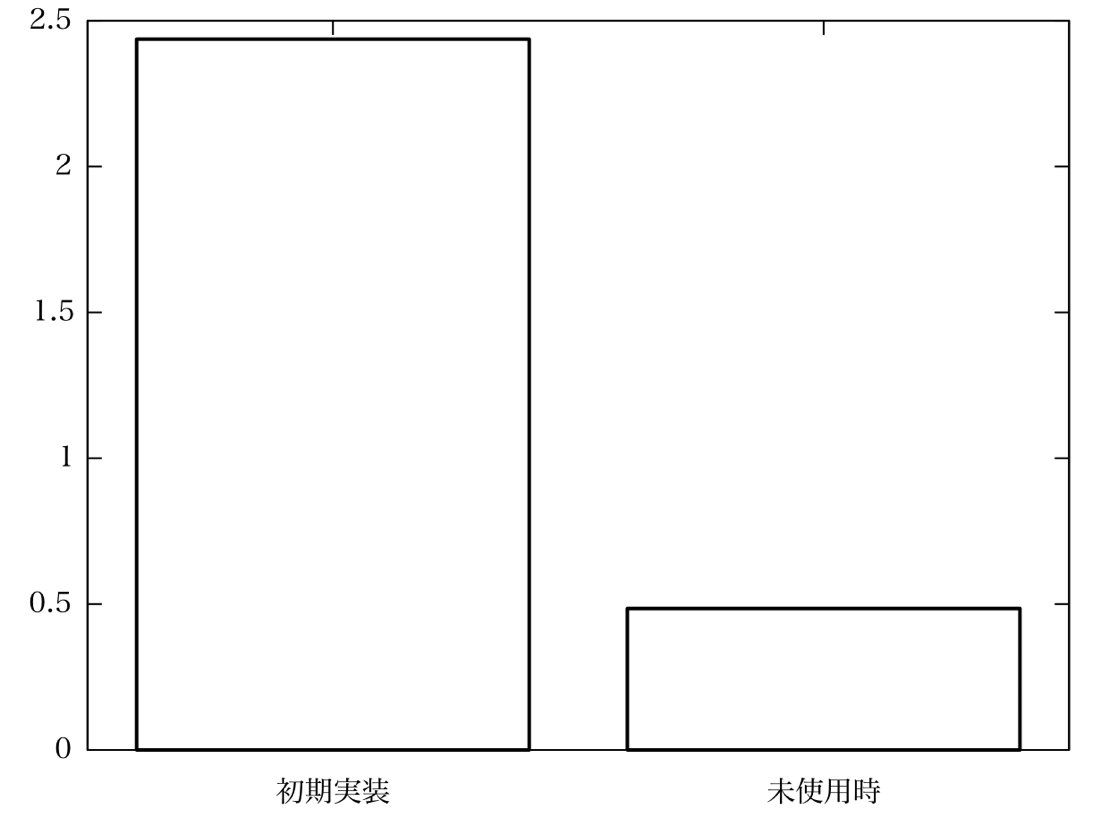


図5.2 初期実装と未使用時におけるレスポンス時間の計測結果．

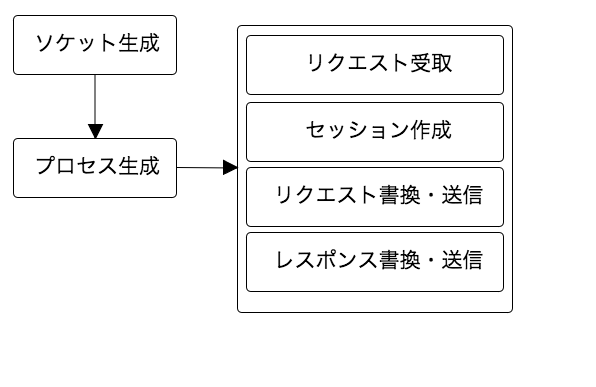


図5.3 初期実装Hoppinの動作．

初期実装Hoppinに遅延が生じた主な要因としてはHTTPリクエスト及びレスポンスの書き換えを逐次的に処理していることが考えられる．HoppinではHTTPリクエスト及びレスポンスの全文すべてを正規表現によるマッチングで書き換え位置を決定している．書き換えを行っている箇所はポート番号と接続パスの2種類であり，その書き換えを終えた後ユーザに送信しているため遅延が生じていることが考えられる．初期実装Hoppinでは，この処理を終了しない限り次の通信の受付を行わないため，I/O待ちを含む逐次処理となり，遅延が生じていたと考えられる．

5.1.2 Hoppinの最適化

遅延が生じた点を踏まえ，攻撃検出システムをマルチプロセス化する改良実装Hoppinを作成した．初期実装Hoppinではシステムがソケットの生成からHTTPレスポンスの書き換え・送信をすべて逐次実行していたため，次の通信の処理の受付が行えなかった．そこで，改良実装ではソケット生成部とHTTPリクエスト及びレスポンスの処理部を切り分けて実装している．機能を切り分けて実装することにより，HTTPリクエストの処理中の間に他のプロセスで通信の待ち受けが可能となり，処理の終了を待たずに次の処理を行うことが可能となる．図5.4に改良実装Hoppinの動作の概略を示す．

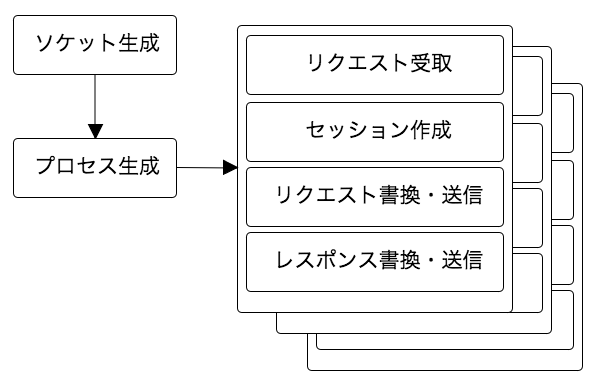


図5.4 改良実装Hoppinの動作．

改良実装Hoppinを再度同一条件で計測を行ったところ，計測結果は，0.500[s]となった．図5.5に示す計測結果より，改良実装Hoppinを初期実装Hoppinと比較すると，約5倍の遅延は解消されており，未使用の場合と比較すると同等のレスポンス時間となった．このことから，マルチプロセス化することで，初期実装Hoppinが所持していたボトルネックの問題の解決が可能となった．

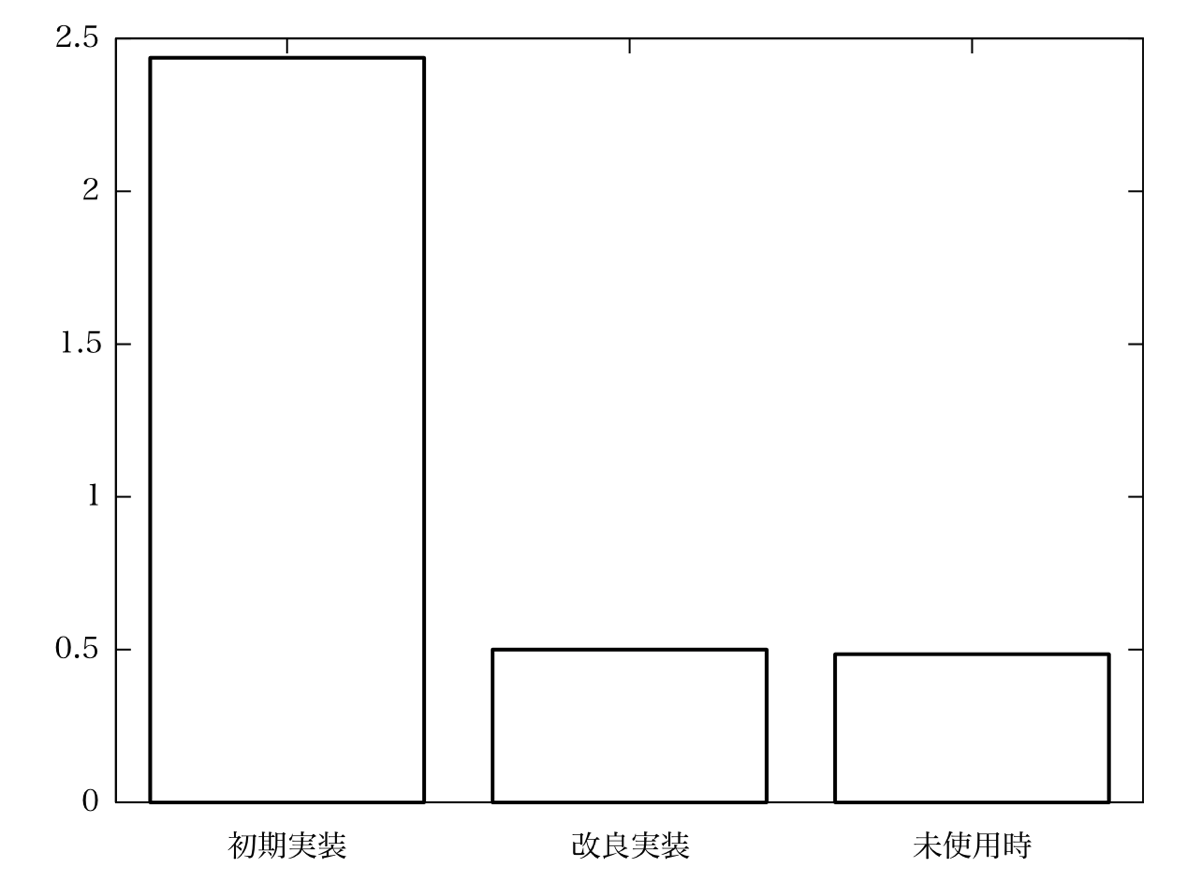


図5.5 初期実装，改良実装と未使用時におけるレスポンス時間の計測結果．

5.2 有用性実験

5.2.1 概要

本実証実験で提案している　Hoppin の有用性実験を行った．検証環境はAmazon Web Service（AWS）[1]の東京リージョンとMicrosoft Azure(Azure)[8]の東日本リージョンにある，クラウド環境上に設置を行った．Azureには仮想インスタンスの3台それぞれにHoppinの機能を設置し，AWSにはEC2インスタンスの3台に攻撃検出システム，1台に管理システムとして設置し運用を行った．攻撃者を含むユーザの接続は各インスタンスに設置されたHoppinに対して行われる．AWS内に設置している管理システムへの接続は一部のネットワークからの接続のみ可能とした．データの収集は2017年11月13日から2018年1月20日の79日間で行いシステムの機能・攻撃手法の収集及び継続性について検証を行った．表5.1にAWSのインスタンス性能， 表5.2にAzureインスタンスの性能，図5.6に本実験の検証環境を示す．

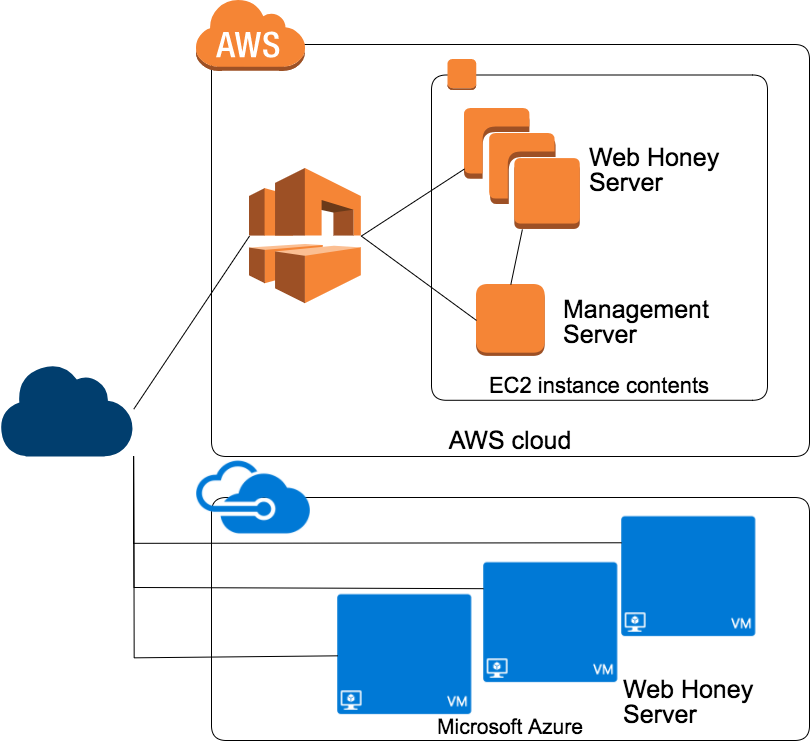


図5.6 検証環境．

表5.1 AWSインスタンスの性能．

|  |  |
| --- | --- |
| インスタンス | m4.large |
| CPU | 2.3 GHz Intel Xeon E5-2686 v4 または  2.4 GHz Intel Xeon E5-2676 v3 |
| メモリ[GB] | 8 |
| ストレージ | SSD 20GB |

表5.2 Azureインスタンスの性能．

|  |  |
| --- | --- |
| インスタンス | D2 v2 |
| CPU | 2.4 GHz Intel Xeon E5-2673 v3 |
| メモリ[GB] | 7 |
| ストレージ | SSD 100GB |

5.2.2 実験結果

攻撃者の接続状況

5.2.1の環境での実験の結果，ユーザが検証環境の各インスタンスへ接続している件数は総計で1556件あり，内8%の119件が記録モードに移行した．各インスタンスに接続している接続ユーザを国別に分類すると，上位10カ国は多い順に，アメリカ，中国，日本，ロシア，フランス，ニュージーランド，ドイツ，香港，韓国，イギリスであった．図5.7に通常モードにおける接続ユーザの結果を示す．

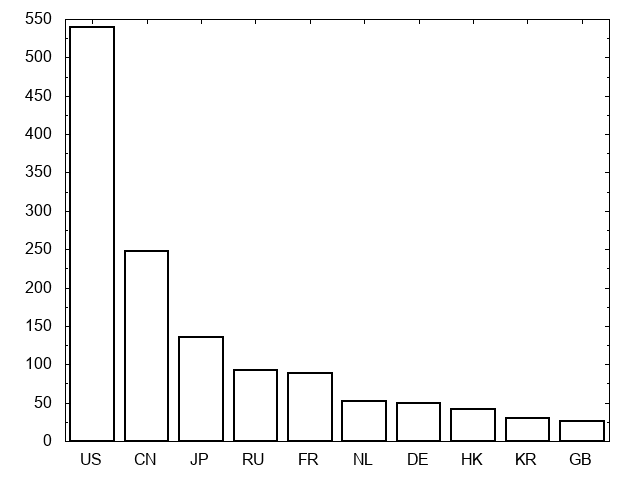


図5.7 通常モードにおける接続状況．

一方，記録モードに移行した接続ユーザを国別に分類すると上位10カ国は，ドイツ，フランス，アメリカ，ニュージーランド，日本，ロシア，韓国，台湾，オーストラリア，スイスであった．

図5.8に記録モードにおける接続ユーザの結果を示す．

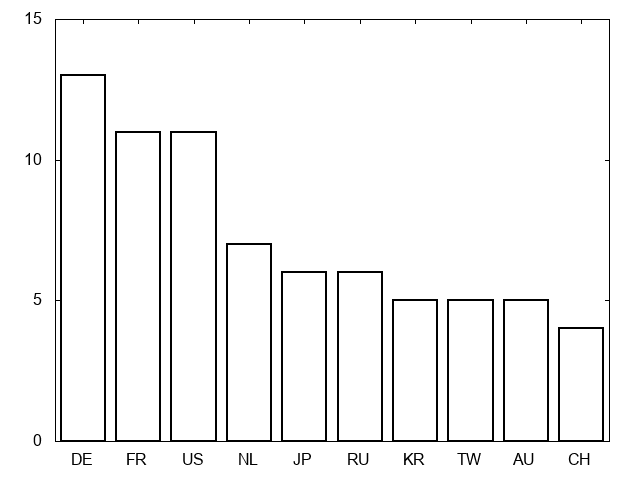


図5.8 記録モードにおける接続状況．

また，取得したHTTPリクエスト情報はHEADメソッド及びGETメソッドで取得されており，HEADメソッドが96%であり，GETメソッドは4%であった．表5.3にメソッド毎のリクエスト数とその割合を示す．

表5.3 メソッド毎のリクエスト数の割合．

|  |  |  |
| --- | --- | --- |
|  | リクエスト数 | 割合 |
| HEAD | 9329 | 96 % |
| GET | 419 | 4 % |

収集されたリクエスト情報から，データベース管理ツールであるphpMyAdmin[11]のディレクトリ及び setup.php に対しての接続が全体の90%であった．接続した時刻に関しても１秒間に複数リクエストを送信しており，機械的にクローリングを行ってシステムのディレクトリの存在を確認していたと考えられる．以上のことから，攻撃情報の収集を行うことが可能であることを判断した．

Hoppinの機能性

Hoppinでは，攻撃者から収集している情報として，IPアドレス，接続パス，接続メソッドの情報を記録している．また，内部でセッション情報の管理を行っているため，内部用のセッションIDを付与することでセッション中に接続したユーザの情報を収集している．これらの情報について，本実験で攻撃者の攻撃手法を予測することが可能であったことから，ハニーポットとしての機能を満たしていると判断した．

攻撃の継続性

攻撃の継続性は，初回接続時に攻撃者が攻撃対象のディレクトリが存在するかをクローリングを行うことで判断し，次回接続時に対象サーバに対して攻撃を仕掛けた場合に関して継続性があると考えている．図5.9に攻撃の継続性を概要を示す．

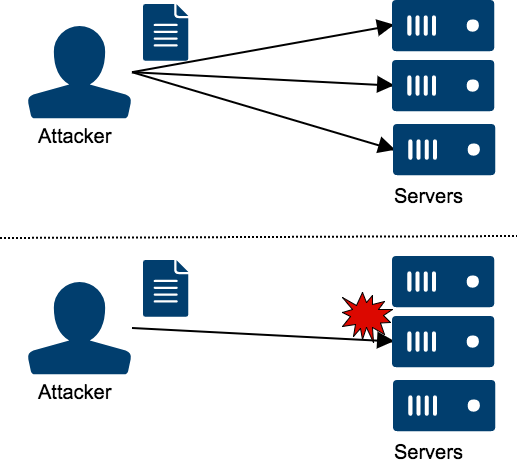


図5.9 攻撃の継続性の概要．

攻撃の継続性に関しては，実験中に再度訪問したユーザは２件存在した．1件目は2017年12月5日に接続を行い，その後2018年1月8日に再度接続を行った．その際の接続はGETメソッドにより行っており，phpMyAdminのsetup.phpに対して接続を試みていた．2件目は2017年11月15日に接続し，その後2017年12月6日に再度接続を行った．その際の接続はHEADメソッドにより行っており，phpMyAdminのディレクトリに対して接続を行っていた．これは，ディレクトリの存在を確認することにより次の攻撃を行うクローリングを行っていたのではないかと考えている．

以上より，再訪問に関しては接続があったものの，攻撃の継続性を行っていることに関して，攻撃手法が変化していないことから，引き続き実験を行い攻撃の継続性の評価を行う必要がある．

5.3 Web Application Firewallとの併用

Hoppin と Web Application Firewall（WAF） はそれぞれで対策できる攻撃が異なる．表5.4にHoppinとWAFとの対策可能な攻撃の比較を示す．

表5.4 WAFとの比較．

|  |  |  |
| --- | --- | --- |
|  | Hoppin | WAF |
| クロスサイトスクリプティング | Possible | Yes |
| SQL インジェクション | Possible | Yes |
| なりすまし攻撃 | Yes | No |

表5.4よりWAFではクロスサイトスクリプティング及びSQLインジェクションなどのフォームやパラメータに対しての攻撃に有効ではあるが，HoppinはWAFが設置されるべきシステムの境界におけるネットワークトラフィックを得ることが可能であるため，WAFの機能を持つことが可能である．しかし，現状ではその機能を実装していない．一方Hoppin では，不正ログインを行うことで他人になりすまし各コンテンツを利用する，なりすまし攻撃の検出が可能である．また，既存のWAFとHoppinは併用して使用することが可能であり，併用することでより多くの攻撃の検出を行うことが可能である．

1. まとめと今後の課題

本実証実験では既存のWebアプリケーションに疑似APIの機能を付加することにより，ハニーポット化するシステムHoppin を提案した．Hoppin を用いることにより Web Application Firewall で検出されないなりすまし攻撃の検出を可能とした．また疑似APIを用いることにより，コンテンツに対して影響を与えることなく攻撃活動を攻撃者に実施させることが可能となった．

Hoppin は実際のコンテンツ等を設置しているため，高対話型のハニーポットであるが，Dockerを用いることにより，攻撃活動により稼働が難しい場合，コンテナの破棄，再起動が容易に可能である．またコンテナ内のプロセスは独立しており，他のコンテナに対して影響を与えないため，低対話型ハニーポットと同等の安定性を担保することが可能となった．レスポンス時間は，初期実装時のHoppinではI/O処理の部分でボトルネックとなり、遅延が生じていたが，リクエストの処理部をマルチプロセス化した改良実装Hoppinの場合では，未使用時と同等のレスポンス時間を実現した．

今後の課題として，次の2点が挙げられる．

6.1ブラックリスト機能の実装

Hoppinは，記録モードに移行した後でも存在しないページに関してはHTTPステータスコード404を返している．これは，Hoppinが記録処理をリクエスト処理の際に行っているため，レスポンス結果のポート番号を変更する以外はそのままのデータをユーザに送信しているために生じている．図6.1に現状のHoppinの処理の概要を示す．

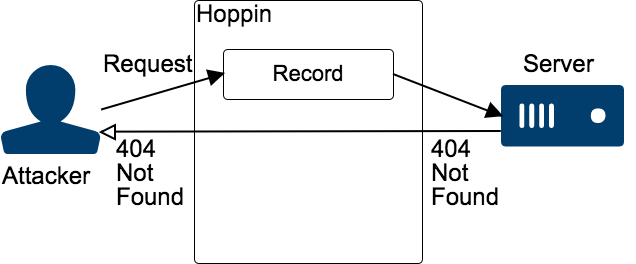


図6.1 現状のHoppinの処理概要．

しかし，実験より攻撃者はクローリングを用い複数のディレクトリに対して接続を行い，攻撃対象のディレクトリを確認していることが判明した．そこで，ブラックリストの実装を行い，リストに登録されたディレクトリに対しては記録モードに移行している間はHTTPステータスコード200を返すように実装を行う．行う処理としては，リクエスト処理の際に接続パスの情報とブラックリストに一致したディレクトリに対しレスポンスの際にヘッダ情報を書き換え，ユーザに送信することを行う．これにより，攻撃者はディレクトリが存在しているものだと認識し引き続き攻撃を行うことが期待される．図6.2にブラックリスト機能実装後のHoppinの処理の概要を示す．

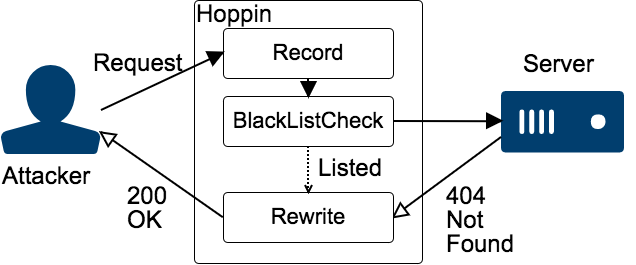


図6.2 ブラックリスト機能実装後のHoppinの処理概要．

6.2 Hoppinの有用性評価の継続

Hoppinは現在Amazon Web Service（AWS）[1]と Microsoft\\Azure(Azure)[8]のクラウド環境上に設置されている．機能面及び攻撃情報の収集に関しては現状行えているが，攻撃の継続性に関しては再訪問しているユーザが存在してはいるが，攻撃手法が発展していないため，再訪問しているとは言えず，継続性は行えていない．そこで，ブラックリスト機能の実装などHoppinの改良を行いながら継続性の確認等，引き続き有用性の評価を行う．

参考文献

1. Amazon Web Services, Inc.: Amazon Web Services (AWS) – Cloud Computing Services, <https://aws.amazon.com/>.
2. cowrie: cowrie, <https://github.com/cowrie/cowrie>.
3. Deutsche Telekom AG Honeypot Project: T-Pot: A Multi-Honeypot Platform, <http://dtag-dev-sec.github.io/mediator/feature/2015/03/17/concept.html>.
4. Docker Inc: Docker - Build, Ship, and Run Any App, Anywhere, <https://www.docker.com/>.
5. Elastic: Elasticsearch: RESTful, Distributed Search & Analytics Elastic, <https://www.elastic.co/products/elasticsearch>.
6. Elastic: Kibana: Explore, Visualize, Discover Data | Elastic, <https://www.elastic.co/products/kibana>.
7. Elastic: Logstash: Collect, Parse, Transform Logs | Elastic, <https://www.elastic.co/products/logstash>.
8. Microsoft: Microsoft Azure Cloud Computing Platform & Services, <https://azure.microsoft.com/>.
9. mushorg: glastopf, <https://github.com/mushorg/glastopf>.
10. N.Provos, D.McNamee, P.Mavrommatis, K.Wang and N.Modadugu: The ghost in the browser analysis of web-based malware, HotBots '07 (2007).
11. phpMyAdmin contributors: phpMyAdmin, <https://www.phpmyadmin.net/>.
12. T.Ishikawa and K.Sakurai: Parameter Manipulation Attack Pre-vention and Detection by Using Web Application Deception Proxy, Proc. IMCOM ’17, pp. 74:1{74:9 (2017).
13. WASC Threat Classi\_cation: The Web Application Security Consortium Remote File Inclusion, <http://projects.webappsec.org/w/page/13246955/Remote\%20File\%20Inclusion>.
14. 山本健太, 齊藤泰一: Linux コンテナ技術を利用したSSH ハニーポットの提案と評価．, コンピュータセキュリティシンポジウム2016 論文集, Vol. 2016, No. 2, pp. 770{776 (2016).
15. 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． (2011). https://www.ipa.go.jp/files/000017312.pdf.

Ⅳ．発表成果

（１）論文発表（会議予稿集・報告書を含む）

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と防御，コンピュータセキュリティシンポジウム2017論文集，Vol.2017, No.2 (2017).

（２）その他口頭発表・ポスター発表等

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).

Wai Kyi Kyi Oo, Hiroshi Koide, Danilo Vasconcellos Vargas, Sakurai Kouichi : An Implementation of Moving Target Defense Technology on Web System, ISIP2018 (2018).

Ⅴ．今後の展望

大学等の組織におけるネットワークや情報システムは今や必要不可欠なインフラであり，専門の部門により厳重に管理されることが当たり前のものとなっている．そのような状況では，大学等の組織におけるネットワークや情報システムを利用して，ネットワークそのもの，情報システムそのもの，それらに関連するサイバーセキュリティに関する教育や研究が困難となってきている．しかし，クラウドを利活用することにより，教育や研究に最適化された充分な規模の情報システムを一時的に構築することが可能であり，組織に影響を与えずに教育や研究に資することが可能である．今後もネットワーク，情報システム，サイバーセキュリティに関する教育や研究を実践的に行うには，クラウドを利活用することが必須であり，利用する予定である．

Ⅵ．ご意見

本実証実験に参加し，クラウドを利活用することにより，はじめて可能となる実験を行うことができ，結果を得ることができたことに感謝します．今後も多くの研究者がクラウドを利活用することにより，良い研究成果をあげさまざまな科学的知見を得られると良いと考えます．身近でも，教育や研究においてクラウドを利用することが多くなってきています．