卒業論文 2018 年度 (平成 30 年)

低対話型 Honeypot のコマンド拡張による
高対話型 Honeypot への近似

慶應義塾大学 総合政策学部
菅藤 佑太

卒業論文要旨 - 2018 年度 (平成 30 年度)

低対話型 Honeypot のコマンド拡張による 高対話型 Honeypot への近似

PC の普及や IoT デバイスのシステム高度化により, 高度な処理系を組むことが可能になり, デバイス上に Linux 系などの OS が動く機器が広く人々に使われるようになった. そうした中で SSH の不正な侵入によって自分の機器が踏み台にされ, 自身の機器から第三者のネットワーク機器に攻撃されていたり, ウイルスやバックドアが設置されて自身の機器が不正にウイルスが実行されるような環境を作られる問題が発生している. 侵入された際に侵入者がどのような挙動をしているのかを知る手段として,Honeypot というものがある. これは SHELL の擬似的な挙動をアプリケーション上で実現し, 敢えて SSH で侵入しやすいような環境を作ることで, 侵入者にログイン試行に成功したと検知させ, その際に実行したコマンドを観測する. 本研究では,Honeypot の機能を拡張することでより多くの, 高精度な攻撃ログを取得し, これを評価した.

キーワード:
1. SSH, 2. Honeypot, 3. 機械学習, 4. 自然言語処理

慶應義塾大学 総合政策学部
菅藤 佑太

Abstract of Bachelor's Thesis - Academic Year 2018

Approximation of high-interaction honeypots by Command Extension of low-interaction honeypot

I can make high processing system, and a moving apparatus is wide, and the OS's such as the Linux system came to be used for people on a device, and, meanwhile, one's apparatus is stepped over by an unjust invasion of SSH, and I am attacked to the network apparatus of the third party from an apparatus of oneself by the spread of PCs and the system advancement of the IoT device, and a problem made in the environment that a virus and a back door are installed, and is carried out a virus an apparatus of oneself illegally occurs. As a window, there was a thing called Honeypot what kind of ways an intruder behaved when it was invaded, and this realized pseudobehavior of SHELL on application and let an intruder detect it by making the environment that was easy to invade it daringly in SSH when I succeeded in a login trial and acquired more highly precise attack log by observing the command that I carried out on this occasion, and expanding the function of Honeypot in this study and evaluated this.

Keywords :
1. SSH, 2. Honeypot, 3. Machine Learning, 4. Natural Language Processing

Keio University, Faculty of Policy Management Studies
Yuta Sugafuji

目 次

図 目 次

表 目 次

第1章　序論

本章では本研究の背景，課題及び手法を提示し，本研究の概要を示す.

1.1　研究の背景

PC の普及や IoT デバイスのシステム高度化により, 高度な処理系を組むことが可能になり, デバイス上に Linux 系などの OS が動く機器が広く人々に使われるようになった. そうした中で SSH の攻撃によって自分の機器が踏み台にされ, 自身の機器から第三者のネットワーク機器に攻撃されていたり, ウイルスやバックドアが設置されて自身の機器が不正にウイルスが実行されるような環境を作られる問題が発生している.
侵入された際に侵入者がどのような挙動をしているのかを知る手段として,Honeypot というものがある. これは Shell の擬似的な挙動をアプリケーション上で実現し, 敢えて SSH で侵入しやすいような環境を作ることで, 侵入者にログイン試行に成功したと検知させ, その際に実行したコマンドのログを収集する.

SSH の Honeypot は大きく二種類に分けることができ, 一つは低対話型 Honeypot, もう一つは高対話型 Honeypot である. 低対話型 Honeypot は実際の Shell の挙動をエミュレートしたアプリケーションシステムである. 高対話型 Honeypot は他のホストに攻撃しないようにネットワークの設定を適切に行い, また root の権限が取られないように user 権限を適切に行ったりするなどの設定を施した実際の OS を用いた仕組みである. 高対話型 Honeypot は低対話型 Honeypot と比較すると, 本物の OS を用いている分高精度な攻撃ログを取得することができるが, 乗っ取られ他のホストに攻撃をしたりウイルスに犯されてしまうなどの危険を孕んでいるため設置コストが高く, 普及率も非常に低い. 一方で低対話型 Honeypot はアプリケーションシステムであるため,root 権限を取られるような危険が極めて少なく, アプリケーションシステム内での脆弱性だけに限った問題しか存在しない. そのため設置コストが低く, ドキュメントが多く存在し比較的誰でも安全に設置できるため, 普及率が高い. しかし, 実際の Shell とは異なる挙動や,Honeypot に特有な挙動をしてしまうため, 設置した Honeypot に侵入した悪意のあるユーザーに侵入先が Honeypot であると検知されてしまう可能性がある.

この低対話型 Honeypot の設置コストの低さで, かつ Honeypot に侵入してきた悪意のあるユーザーに侵入先が Honeypot であると検知されないように攻撃ログを取得するためには, 低対話型 Honeypot の挙動をできるだけ実際の Shell に近似すれば良いのではないかと考えられる.

本研究の予備実験では,SSH の低対話型 Honeypot に実装されていないコマンドで悪意

のある侵入者が使うようなコマンドを実装し, 何の追加実装も施していない SSH の低対話型 Honeypot で取れた侵入者の実行コマンドログと , 追加実装を施した SSH の低対話型 Honeypot の侵入者の実行コマンドログを比較することで, 追加実装を施した SSH の低対話型 Honeypot の方がコマンドパターンとして多く収集できることを示した.

本研究では低対話型 Honeypot を実際の Shell の挙動に近似するために, 実際の Shell に実装されているもので低対話型 Honeypot に実装されていないコマンドの実装や, 低対話型 Honeypot に特有の異常な挙動を修正を行いこの問題を解決できるのではないかと考えた.

実際の Shell の挙動に近似するように実装した低対話型 Honeypot で取れた攻撃ログが, 何の実装を施していない純正の低対話型 Honeypot で取れた攻撃ログと比較して, 実際の OS で用いた攻撃ログにどれほど近しいかを時系列データを教師データとした機械学習を用いて検証することで評価した.

1.2 本論文の構成

本論文の構成を以下に示す. 第 1 章では, 本研究の背景と目的について述べた. 第 2 章では, 本研究の要素技術となる Shell と Honeypot と時系列データの扱いについて整理する. 第 3 章では本研究にあたっての事前実験の概要と結果を述べ, 第 4 章では, 本研究の問題の定義をする. 第 5 章では, 本提案手法のシステムについて解説する. 第 6 章では, 実装方法や実装例について述べ, 第 7 章で提案システムの評価手法について説明する. 第 8 章では先行研究や事例について紹介する. 第 9 章では本研究のまとめと展望について言及する.

第2章　本研究の要素技術

本章では, 本研究の要素技術となる Shell と Honeypot と時系列データの扱いについて各々整理する.

2.1　Honeypot

使われているデバイスで不正な SSH によって侵入された際に侵入者がどのような挙動をしているかのログを収集する手段としての Honeypot について, その技術について解説する.SSH の Honeypot[3] は低対話型 Honeypot と高対話型 Honeypot の大きく二種類に分けることができる. 以下に侵入者が SSH で不正に機器に侵入してから踏み台にして他の機器に攻撃を仕掛けるまでの一般的なフローを図 2 に示す.

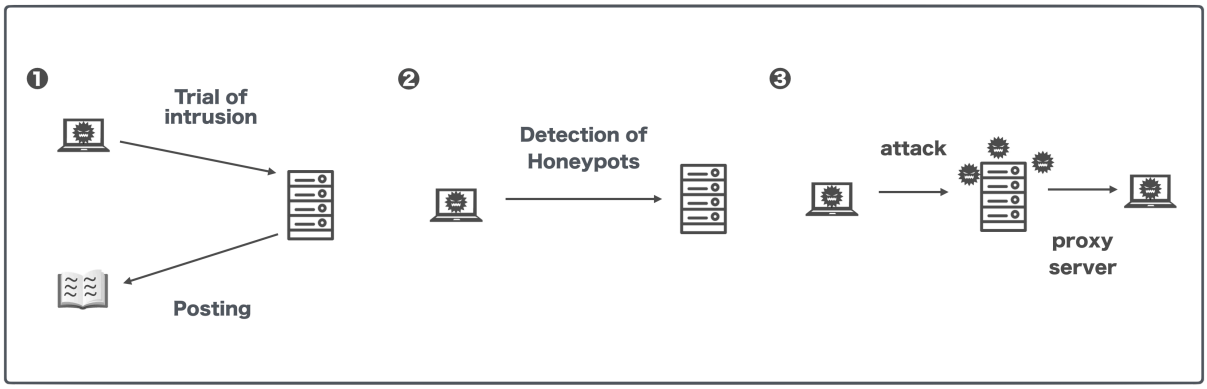

図 2.1: 不正な SSH 侵入者の想定行動フロー

2.1.1　低対話型 Honeypot

SSH の低対話型 Honeypot は実際の Shell の挙動をエミュレートしたアプリケーションシステムである. エミュレートされたものであるのでコマンドやその挙動についての機能が限定されており, 実際の Shell の機能として不足があり, 侵入者に侵入先が Honeypot であると検知され, 本来取れるはずの攻撃ログが収集できなくなってしまう可能性を含んでいるため, 収集ログの精度に問題がある. しかし, 実際の Shell の挙動をエミュレートしただけ

のアプリケーションシステムなので, 脆弱性がアプリケーションシステム内に限られ,root 権限を侵入者に許してしまい, 踏み台にされてしまうなどの危険が極めて少ない.

2.1.1.1 Kippo

Kippo は, 悪意のある SSH のログイン試行者や侵入者の挙動やログを記録するために使用される Python で実装された SSH の低対話型 Honeypot であり [?],Kippo は前身の Kojoney[4] に大きく影響を受けている. 実装は Python で行われており, ネットワークは Twisted[5] というフレームワークで組まれている.Kippo のプロジェクトは低対話型 Honeypot として 2009 年に登場し,Raspberry Pi[6] などを筆頭としたシングルボードコンピュータ [7] の普及により 2010 年頃から勃興した IoT デバイスの高度化 [8] と相まって広く設置された. Kippo の機能の特徴としては収集したコマンドログ を時系列データとして保存されており,"playlog" という Kippo 内にあるプログラムを実行することで, 過去のコマンドログ をそのまま再生することができる. また, 侵入者によってダウンロードされたファイルも実行ができないように保存しておくことができる.Kippo は 2014 年頃を最後に現在はプロジェクトが進んでいない.[9] Kippo は IoT デバイスの高度化広く設置された SSH の低対話型 Honeypot のうちの一つであったが, 実装されているコマンドも 17[10] と少なく, また Kippo 特有の異常な挙動があったりと多くの問題があった.

2.1.1.2 Cowrie

Cowrie は Kippo 同様, 悪意のある SSH のログイン試行者や侵入者の挙動やログを記録するために使用される Python で実装された SSH の低対話型 Honeypot であり, 実装は Kippo のコマンドや運用における機能の拡張をしたものとなっている. Kippo 特有の異常な挙動を改善しているものの, 実装コマンド数は 38[11] と Kippo より少し多くなっているものの [12],Cowrie 特有の異常な挙動もまだまだ多い.

2.1.2 製造責任と知的財産権に関する法制度

本項では既存の製造責任と知的財産権に関する法制度を概説し，それらのパーソナルファブリケーションの中での問題について整理する.

2.1.3 高対話型 Honeypot

2.1.3.1 Honeynet Project

2.1.4 SSH の Honeypot の比較

以上をまとめた SSH の低対話型 Honeypot と SSH の高対話型 Honeypot の比較を行った表を図 2 に示す.

	設置コスト (リスク)	Honeypotであることの検知されにくさ
低対話型Honeypot	設置コストが低い	検知されやすい
高対話型Honeypot	設置コストが高い	検知されにくい

図 2.2: SSH の低対話型 Honeypot と SSH の高対話型 Honeypot の比較

2.1.5 Shell

Shell は OS のユーザーのためにインタフェースでカーネルのサービスへのアクセスを提供するソフトウェアである. 本研究での”Shell”はコマンドラインシェルのことについてのことを指す.

2.1.5.1 Secure Shell

Secure Shell（セキュアシェル、SSH）は、暗号や認証の技術を利用して、安全にリモートコンピュータと通信するためのプロトコル。パスワードなどの認証部分を含むすべてのネットワーク上の通信が暗号化される.[13]SSH における問題としては, 通信する上での認証方法が鍵認証ではなくパスワード認証になっていてパスワードの総当たり攻撃を受けたり, パスワードが標準のままの設定になっていることで不正なログイン試行によって侵入を許してしまうというものである.

2.1.5.2 BusyBox

BusyBox は標準 UNIX コマンドで重要な多数のプログラムを単一のバイナリファイルに含むプログラムである.BusyBox に含まれる, 多数の標準 UNIX コマンドで必要とするプログラムの実行ファイルは”Linux 上で最小の実行ファイル”となるよう設計されている. 一般にインストールされる実行ファイルは一部だけを実装できるように選択することができ, 一般的には BusyBox の機能は 200 以上も用意されている.[14](今回使用したものに含

まれるコマンドの数は 219) この BusyBox をインストールして実際にこれを実行するためには, 例えば「/hoge/busybox/ xxx」(”xxx”は標準 UNIX コマンドなどが入る) とすれば良い.

2.1.6 自然言語処理

過去のデータの入力に対して未知のデータをどのようにして出力するのかについては様々な手法がある. 本研究において自然言語処理は意味解析についてこれを使用した.

2.1.6.1 統計的意味解析

過去のデータの入力に対して未知のデータを統計的に出力する.

2.1.6.1.1 マルコフ連鎖

2.1.6.1.2 コーパス

2.1.6.1.3 シソーラス

シソーラス [15]

2.1.6.2 ベクトル空間表現

2.1.6.2.1 word2vec

第3章　本研究における問題定義と仮説

本章では，１章で述べた背景より，本章では, 現状の Honeypot の問題点を整理し, この問題をどのように解決すれば良いのかを定義する.

3.1　本研究における問題定義

現状の Honeypot の問題点を列挙していき, 整理する.

3.1.1　SSH Honeypot の現状の問題

Honeypot には運用する上で大きな問題が 2 つある. 一つは設置した Honeypot に侵入した悪意のある侵入者が侵入先を Honeypot であると検知してしまう問題である. もう一つは Honeypot に侵入を許した侵入者に Honeypot を設置した機器から攻撃が仕掛けられてしまう危険がある問題である.
以下の図 2 は, 悪意のある侵入者が不正に機器に侵入してから踏み台にして他の機器に攻撃を仕掛けるまでの一般的なフローであるが,2 番目のフローの悪意のある侵入者が侵入した先が Honeypot であると検知してしまうことや,3 番目のフロの Honeypot に侵入を許した侵入者に Honeypot を設置した機器から攻撃が仕掛けられてしまう危険があることが今回の問題である.

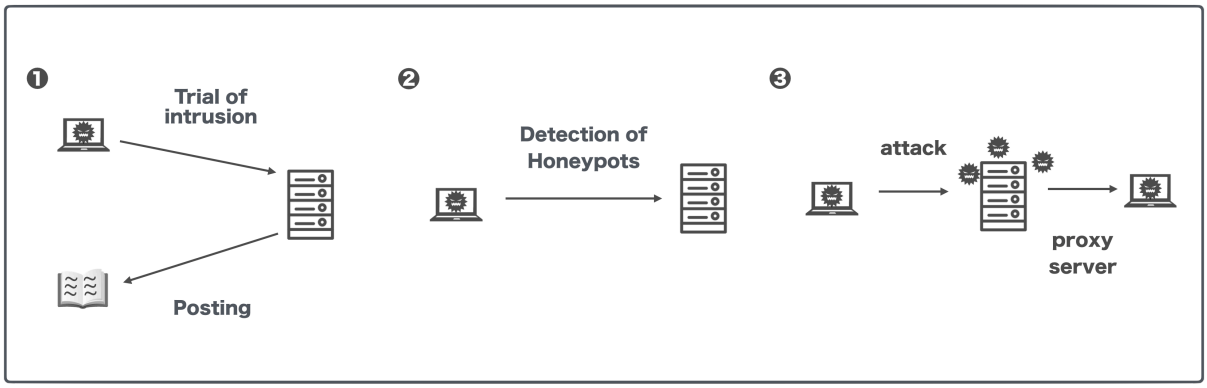

図 3.1: 不正な SSH 侵入者の想定行動フロー

本研究ではこの中でも,2 番目のフローの SSH の低対話型 Honeypot が設置した Honeypot に悪意のある侵入者が侵入先を Honeypot であると検知してしまう問題に着目した.

3.1.1.1 SSH の低対話型 Honeypot における問題

SSH の低対話型 Honeypot は実際の Shell の挙動をエミュレートしたものであるのでコマンドやその挙動についての機能が限定されており, 実際の Shell の機能として不足がある. また SSH の低対話型 Honeypot 特有の以上な挙動も存在する. さらに,SSH で Honeypot にセッションを確立する際に, そのレイテンシを計測し, そのレイテンシが Honeypot の場合に通常とは明らかに異なることにより,Honeypot であると検知されてしまう問題がある. また,Honeypot の username が”Richard”がデフォルトのため, これによって Honeypot であることを検知されてしまう問題もある. そのため侵入者に侵入先が Honeypot であると検知され, 本来取れるはずの攻撃ログが収集できなくなってしまう可能性を含んでいるため, 収集ログの精度に問題がある.

3.1.1.1.1 SSH でのセッション確立におけるレイテンシの問題

3.1.1.1.2 Honeypot の Username の問題

3.1.1.1.3 Honeypot のコマンドの実装の問題

SSH の低対話型 Honeypot は実際の Shell の挙動をエミュレートしたものであるのでコマンドやその挙動についての機能が限定されており, 実際の Shell の機能として不足がある. 2.1.5.2 で述べたように,”Linux 上で最小の実行ファイル”となるよう設計されている BusyBox に含まれるコマンドの数が 200 以上あるのに対し, 現状で広く使われている SSH の低対話型 Honeypot である Cowrie に実装されているコマンドは 2.1.1.2 でも述べた通り,38 しか存在しない. また,SSH の低対話型 Honeypot 特有の挙動が存在し, 以下にその 1 例であるプログラム 1 とプログラム 2 を示す.

プログラム 3.1: 正しい Shell の挙動

```
1 nadechin@cpu:~$ echo -n test
2 testnadechin@cpu:~$
```

プログラム 3.2: Kippo 特有の異常な挙動の例

```
1 s15445ys@s15445ys-neco:~$ echo -n hello
2 -n hello
3 s15445ys@s15445ys-neco:~$
```

上のプログラムが通常の挙動で下のプログラムが SSH の低対話型 Honeypot の挙動である.echo コマンドの-n オプションは改行をしないようにするというものであるが, 実際の Shell の挙動が改行がされることなく正しく出力されているのに対して,Honeypot の挙動

ではオプション部分も出力されてしまっているという問題がある. これは SSH の低対話型 Honeypot 特有の挙動であるため, これによって Honeypot であると検知されてしまう可能性がある.

3.1.2 本研究の問題

3.1.1.1 で列挙した SSH の低対話型 Honeypot の問題の中で, 実際の Shell に実装されているコマンドの不足がある. また SSH の低対話型 Honeypot に特有の異常な挙動も存在するため, 設置した Honeypot が悪意のある侵入者に侵入先を Honeypot であると検知されてしまい, 実際の OS に悪意のある侵入者が侵入した時の侵入ログとの違いが大きく出てしまう問題に着目した.

3.2 問題解決のための要点

3.1.2 で着目した問題を解決するためには, 以下 2 つの手法を取る必要がある.

コマンドの追加実装: 実際の Shell に実装されているコマンドで,SSH の低対話型 Honeypot に実装されていないコマンドを実装する

既実装コマンドの修正: SSH の低対話型 Honeypot に特有の異常な挙動をする既実装コマンドを修正する

3.3 仮説

3.2 で示した 2 つの手法を用いれば,SSH の低対話型 Honeypot に侵入した悪意のある侵入者に侵入先を Honeypot であると検知させず,SSH の低対話型 Honeypot に悪意のある侵入者が侵入した時の侵入ログを, 実際の OS に悪意のある侵入者が侵入した時の侵入ログに近似できるのでないか.

第4章　事前実験

本章では， 3.1.1.1 で述べた手法を実現するための事前実験を概説する.

4.1　概要

SSH の低対話型 Honeypot である Cowrie はコマンドの実装数が少なく,Cowrie 特有の異常な挙動が多く, 本来実際の OS への攻撃であれば取れるはずであった侵入ログが取れない問題がある. また, 収集ログを分析する際に, これまで用いられてきた”危険なコマンド”としてインデックスを作り, それらを危険なコマンドとしてパターンマッチングする手法では, 今後出現してくる様々なコマンドパターンなどに対応できない.
予備実験では, 実装を施していない純正の Cowrie と Cowrie に BusyBox に含まれるコマンドを実装した修正済みの Cowrie の両方でコマンドログの収集を行うことで, 実装を施していない純正の Cowrie で収集した侵入ログと Cowrie に BusyBox に含まれるコマンドを実装した修正済みの Cowrie で収集した侵入ログとでは, 収集ログのパターンに変化があるのではないかと考えた. 評価として収集した二つのログを Skip-gram モデルを用いてスコアリングし, どちらがより多くのコマンドログのパターンを収集できているのかを検証した. その結果, より多くのコマンドパターンを取れたのが Cowrie に BusyBox に含まれるコマンドを実装した修正済みの Cowrie であるという結果を出した.

4.2　要素技術

予備実験の要素技術に関しては第 2 章の要素技術で全て説明している.

4.3　問題定義

侵入者に侵入先が SSH の低対話型 Honeypot であると検知されてしまい, 本来取れるはずの収集ログが収集できないため, 本来実際の OS への攻撃であれば取得できたはずの侵入ログが収集できない.

4.4 予備実験の手法

実装を施していない純正の Cowrie に対して, これには実装されていないが Shell には実装されているコマンドを実装した.

4.5 実装

純正の Cowrie に BusyBox に含まれるコマンドを実装し, また Honeypot 特有の異常な挙動を修正した. 予備実験の実装に関しては第 6 章の実装で全て説明している.

4.6 評価

実装を施していない純正の Cowrie と Cowrie に BusyBox に含まれるコマンドを実装した修正済みの Cowrie の両方で侵入ログの収集を行い,Word2vec の Skip-Gram Model により次のコマンドの予測, スコアリングを行い評価をした. スコアリングでは, あるコマンドが実行された時に次のコマンドの出やすさを予測したため, 次に実行されるコマンドがスコアとして高い数値を出せばそのコマンドパターンがパターンとして存在しやすいものであるというものである. 予備実験の評価に関しては第 7 章の評価で一部説明している. 本研究の評価と違う評価手法としては, モデル化を純正の Honeypot に BusyBox に含まれるコマンドを実装したものしか行っていないため, 実際の OS に近いログが取れたことが証明できておらず, 比較する対象が少なかった.

4.7 結果

SSH の低対話型 Honeypot の稼働期間は 12/10$\tilde{2}$/1(54 日間) で, 収集できたものとしてコネクション数, パターン数, コマンド数を以下の図 3 に記す.

	純正のHoneypot	修正済みのHoneypot
コネクション数	19829	27914
パターン数	53	91
コマンド数	470	841

図 4.1: 収集した SSH の低対話型 Honeypot のデータ

また, モデル化を行い純正の Cowrie と Cowrie に BusyBox に含まれるコマンドを実装した修正済みの Cowrie のスコアリングを行なった結果を以下の図 4 に記す.

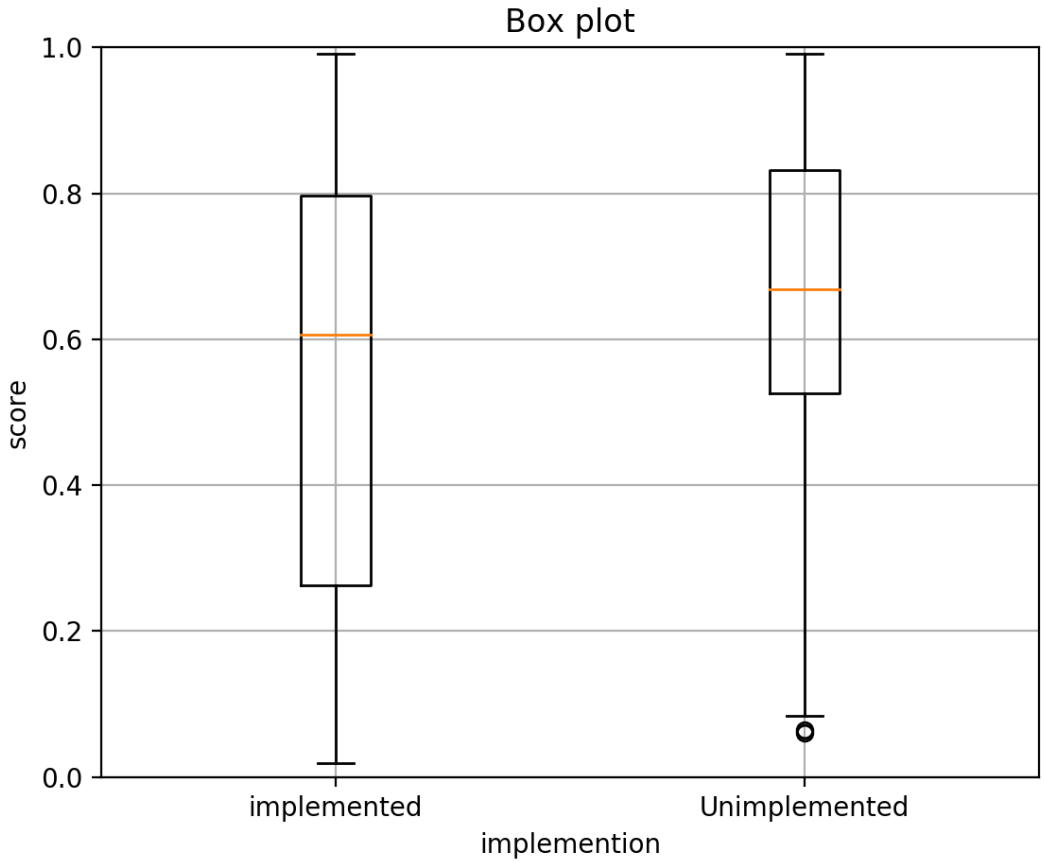

図 4.2: 純正の Cowrie と修正済みの Cowrie のスコアリングによる比較

本研究の予備実験では,Cowrie に実装されていないコマンドで悪意のある侵入者が使うようなコマンドを実装し, 何の追加実装も施していない Cowrie で取れた侵入者の実行コマンドログと , 追加実装を施した Cowrie の侵入者の実行コマンドログを比較することで, 追加実装を施した SSH の Cowrie の方がコマンドパターンとして多く収集できることを示した.

第5章　本研究の手法

本章では，3.3 節で述べた仮説を検証するために, 本研究で行なった手法について概説する.

5.1　問題解決の為のアプローチ

3.2 で述べた問題解決のための 2 つの要件を, 本研究の手法として提案する.

5.1.1　コマンドの追加実装

実際の Shell に実装されているコマンドで,SSH の低対話型 Honeypot に実装されていないコマンドを実装する. これによってコマンドの追加実装を行なった低対話型 Honeypot に侵入した侵入者は, 実際の Shell と同じような挙動をする低対話型 Honeypot を Honeypot であると検知できなくなる.

5.1.2　既実装コマンドの修正

SSH の低対話型 Honeypot に特有の異常な挙動をする既実装コマンドを修正する. これによって既実装コマンドの修正を行なった低対話型 Honeypot に侵入した侵入者は, 実際の Shell と同じような挙動をする低対話型 Honeypot を Honeypot であると検知できなくなる.

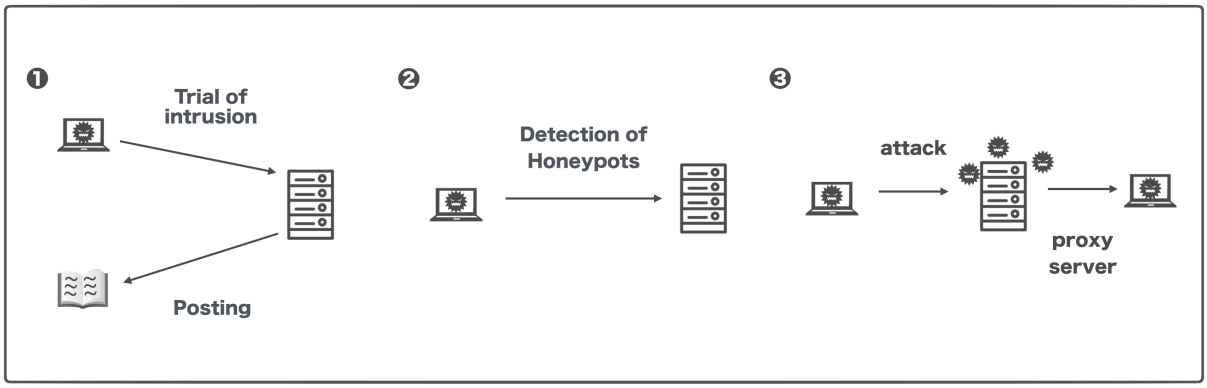

図 5.1: 不正な SSH 侵入者の想定行動フロー

第6章　実装

本章では，5.1 節で述べた手法を用いて純正の Honeypot にどのようなコマンドを実装し,Honeypot 特有の異常な挙動を修正したのかを説明する.

6.1　実装環境

TBD

6.1.1　純正の Honeypot で未実装のコマンドの実装

本研究において純正の Honeypot は Cowrie[?] を使用し, 実際の Shell には実装されているが, 純正の Honeypot で未実装のコマンドについては BusyBox[14] に含まれるコマンドの実装を行なった.2.1.1.2 や 2.2.2 で紹介した通り,BusyBox に含まれるコマンドの種類が 219 ある中で,Cowrie の実装コマンド数は 38 しか存在しない. この差分を Python で実装する.
実装例は以下の通り.(付録にて例で紹介できなかったコマンドを掲載する)

第7章　評価および考察

本章では，6 章で実装した本研究での提案手法の評価とその考察を述べる．

7.1　評価手法

本実験システムの評価として， 3.2 節で述べた要件に対して評価を行う．

本研究では, 以下の三種類の Honeypot を設置する.
1. 広く利用されている SSH の低対話型 Honeypot
2.　実際の Shell には実装されているが,1. の Honeypot で未実装のコマンドを実装した Honeypot
3. 広く利用されている高対話型 Honeypot

これ以降,1. の広く利用されている SSH の低対話型 Honeypot のことを ”純正の低対話型 Honeypot ” , 2. の実際の Shell には実装されているが,1. の Honeypot で未実装のコマンドを実装した Honeypot のことを”修正済みの低対話型 Honeypot” , 3. の広く利用されている高対話型 Honeypot のことを”高対話型 Honeypot”と呼ぶこととする.

以上 3 つの純正の Honeypot, 修正済みの Honeypot, 高対話型 Honneypot のそれぞれで侵入ログを収集する.

また第 3 章の予備実験では, 純正の Honeypot に実装されていないコマンドで悪意のある侵入者が使うようなコマンドを実装し, 純正の Honeypot で取れた侵入者の実行コマンドログと, 修正済みの Honeypot の侵入者の実行コマンドログを比較することで, 修正済みの Honeypot の方がコマンドパターンとして多く収集できることを示した.
予備実験における収集ログの比較の概念図を図 6 に示す.

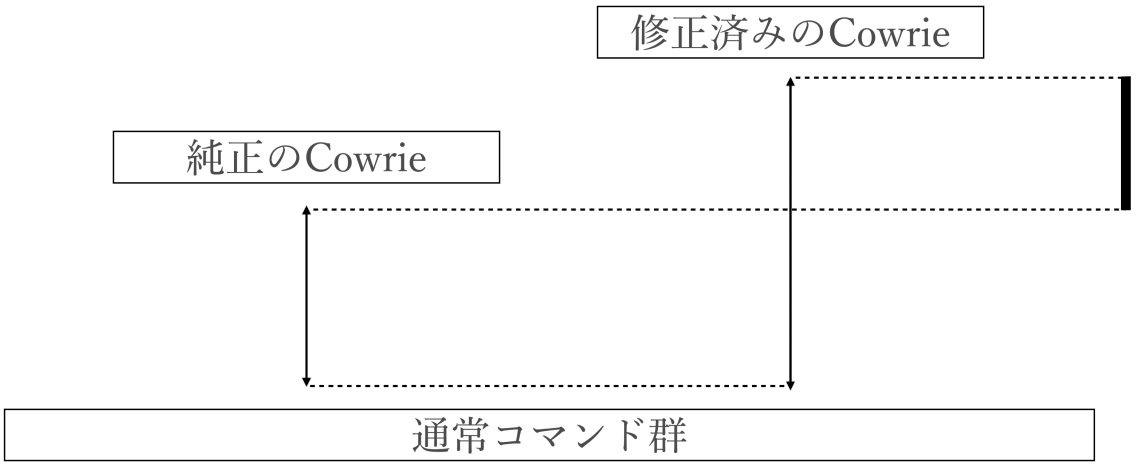

図 7.1: 予備実験の評価の概念図

この予備実験では評価として何の追加実装も施していない SSH の低対話型 Honeypot で取れた侵入者の実行コマンドログと追加実装を施した SSH の低対話型 Honeypot の侵入者の実行コマンドログとを比較したのに対して, 本件研究の評価手法では, 純正の Honeypot で取れた侵入者の実行コマンドログと修正済みの Honeypot の侵入者の実行コマンドログをと高対話型 Honeypot の侵入者の実行コマンドログを比較することで, 修正済みの Honeypot の侵入者の実行コマンドログが実際の Shell の挙動にどれほど近似したのかを評価した. 予備実験における収集ログの比較の概念図を図 7 に示す.

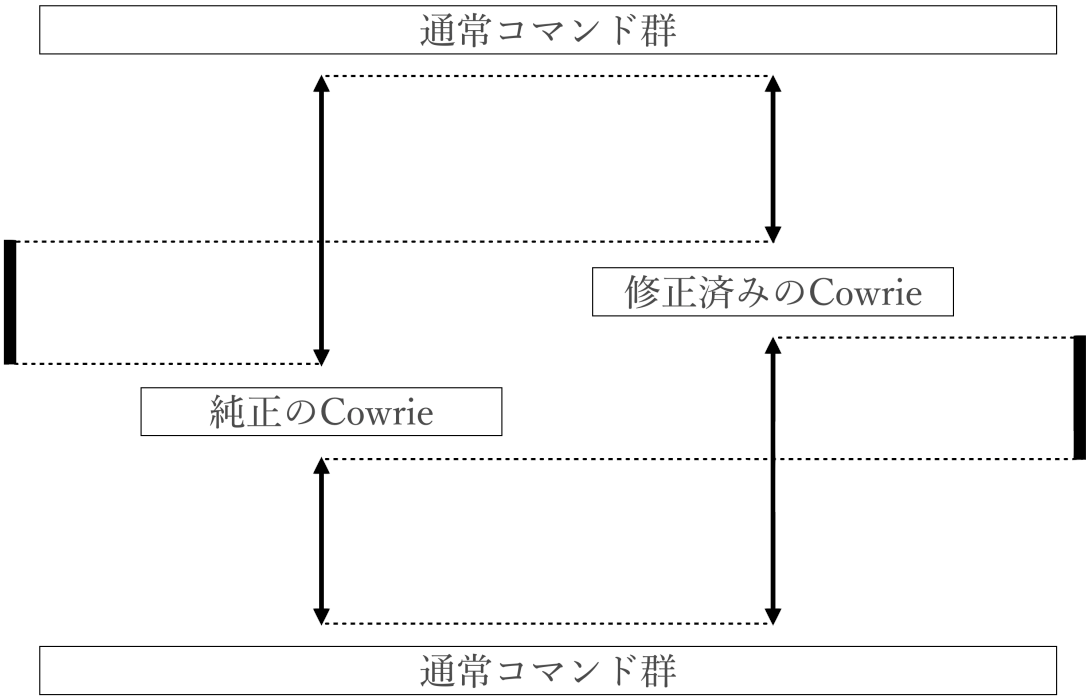

図 7.2: 本研究の評価の概念図

図6, と図7で示したようにして, 取れた収集ログを比較することでいかに高対話型 Honeypot に近似できたのか検証する.

7.1.1 評価手法の実装

純正の低対話型 Honeypot で収集した侵入ログで skip-gram モデルの隠れ層の重みを学習させ (これをモデル 1 とする), 同様にして高対話型 Honeypot で収集した侵入ログも skip-gram モデルの隠れ層の重みを学習させる (これをモデル 2 とする). 次に修正済みの Honeypot で収集したログをセッション開始からセッション終了までに打たれたコマンドごとに (以降これを 1 セッションごとと呼ぶ) モデル 1 とモデル 2 のそれぞれに入力していき, 出力された数値 a を活性化関数としてソフトマックス関数をかけることで,0≦ a≦ 1 の範囲を取るようにし確率的な数値として出力することでスコアリングを行う. このため入力に対して多数存在する出力を全てを合計すると 1 になる. 純正の低対話型 Honeypot や高対話型 Honeypot の収集ログをモデル化する際、入力層として収集ログのコマンドの入力に対してそのコマンドの周辺のコマンドを出力として与えることでこれを学習させる. 例えば 3 つのコマンドが打たれたとしたとしたものを以下のプログラム 3 に示す.

プログラム 7.1: 3 つの実行コマンドの例

```
1  $ uname
2  $ free
3  $ ps x
```

モデルを構築する際には”free”コマンドを入力にした時に, 出力として”uname”コマンド”ps”コマンドを用意しておくことで,free が入力として与えられた時に他 2 つの出力される周辺のコマンドが出力する確率が高くなるようにする. また, 実装としては周辺語をどこまで広げるのかはパラメータとして window size で与えることができ, 上記の例の周辺語は”1”であり,window size を”2”にすればモデル化する際に出力層に与えられる数は 4 つとなる.

以下の図 8 にモデル化のフローを示す.

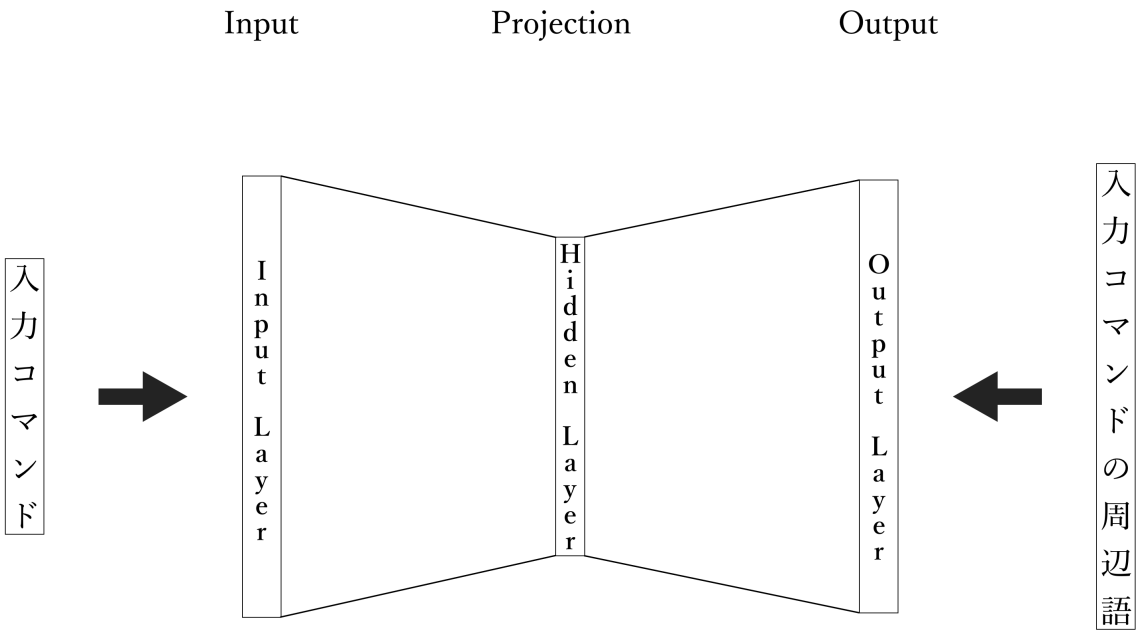

図 7.3: 評価のフロー [1][2]

また, このようにして純正の低対話型 Honeypot の収集ログと高対話型 Honeypot の収集ログに対して各々のモデルを構築する．次にこのモデルに対して, 修正済みの Honeypot で収集したログを入力して, 確率的にスコアリングしていくことで数値を出力する．

以下にこのモデルを使用した時の入力から出力のフローを図 9 を示す．

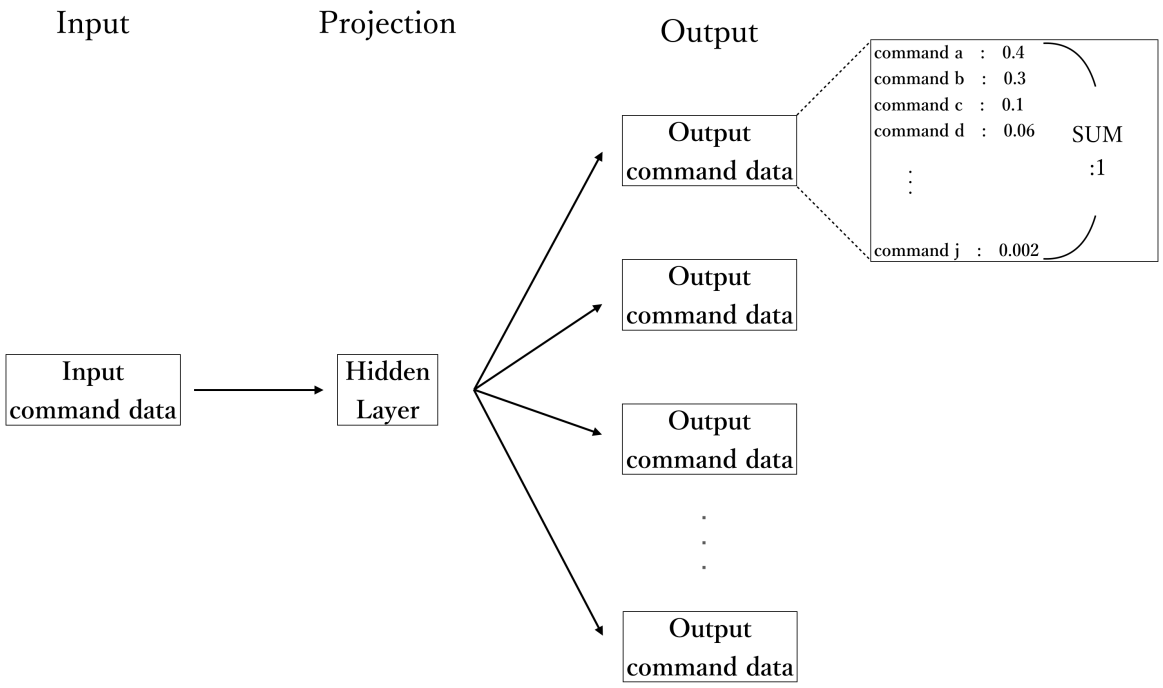

図 7.4: 評価のフロー [1][2]

このようにして出力された数値を 1 セッションごとに平均化し, また全てのセッションにおいてもセッションごとに平均化し, 全てのセッションの平均化を行うことで, 純正の低対話型 Honeypot の収集ログと高対話型 Honeypot の収集ログとの各々で構築したモデルごとに平均値を算出する.

7.1.1.1 コマンド群データのベクトル表現

7.1.1.2 SSH の低対話型 Honeypot の攻撃ログの比較

第8章　第8章

8.1　関連研究

本章では,SSH の Honeypot と時系列データの処理に関連する先行研究について紹介する.

8.1.1　SSH の Honeypot

8.1.1.1　低対話型 Honeypot

8.1.1.1.1　kiipo

8.1.1.1.2　cowrie

8.1.1.2　高対話型 Honeypot

8.1.1.2.1　honeynet

8.1.2　時系列データの処理

8.1.2.1　確率分布モデル

8.1.2.1.1　マルコフモデル

8.1.2.1.2　隠れマルコフモデル

8.1.2.2　ニューラルネット

8.1.2.2.1　畳み込みニューラルネットワーク

8.1.2.2.2　リカレントニューラルネットワーク

第9章　結論

本章では，本研究のまとめと今後の課題を示す.

9.1　本研究のまとめ

うんちっちブリブリ

9.2　本研究の課題と展望

SSH の低対話型 Honeypot に実装するコマンドの選定について,OS ごとに異なるはずであるが, 今回は BusyBox をそっくりそのまま移植しただけであったので, 厳密に実際の Shell や OS の挙動を模して本研究を再検証したい.

9.2.1　うんちっち～

ブリリリリリリリリ

9.2.2　うんちへの応用

リリリリリリリリ

謝辞

参考文献

[1] Greg Corrado Jeffrey Dean Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space. *ACM Transactions on Graphics (TOG)*, 32(4):138, 2013.

[2] Xin Rong. Word2vec parameter learning explained. *ACM Transactions on Graphics (TOG)*, 32(4):138, 2013.

[3] Kippo. Kippo. `https://github.com/desaster/kippo`, 2014.

[4] Satoshi Nakamoto. kojoney: Kojoney. `http://kojoney.sourceforge.net/`, 2008.

[5] Vitalik Buterin. Twisted. `https://twistedmatrix.com/trac/`, 2014.

[6] Raspberry pi. `http://www.idc.com/getdoc.jsp?containerId=prUS40960716`, 2016.

[7] Single board computer. `http://fablabjapan.org/`.

[8] Catarina Mota. Iot デバイスの普及. *Proceedings of the 8th ACM conference on Creativity and cognition*, pages 279–288, 2011.

[9] Kippo のプロジェクトの現在. `http://www.thingiverse.com/`.

[10] Number of kippo'commands. `https://github.com/desaster/kippo/tree/master/txtcmds`.

[11] Number of cowrie'commands. `https://github.com/cowrie/cowrie/tree/master/src/cowrie/commands`.

[12] 消費者庁. Kippo と cowrie の実装コマンドの違い. `http://www.caa.go.jp/seikatsu/shingikai2/kako/spc13/houkoku_g/spc13-houkoku_g-4-1.html`, 1992.

[13] Karl DD Willis and Andrew D Wilson. Secure shell. *ACM Transactions on Graphics (TOG)*, 32(4):138, 2013.

[14] Busybox. `https://proofofexistence.com/`.

[15] 松下 栄一 末岡 隆史 国分 芳宏, 梅北 浩二. シソーラスを組み込んだ意味解析システム. *ACM Transactions on Graphics (TOG)*, 32(4):138, 2013.