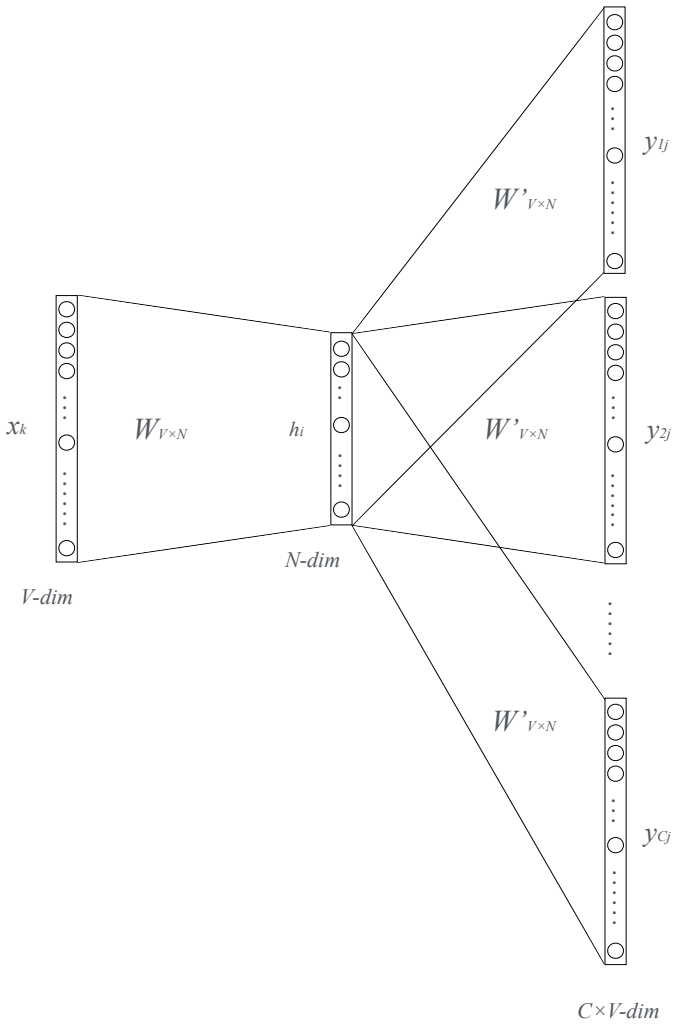


図 2.3: コンテクストサイズ C の Skip-gram Model のニューラルネットワーク

2.4.3.4 SCDV

SCDV とは word2vec で算出したベクトル空間をクラスタリングして，クラスタリング結果も考慮に入れてベクトル空間上に再表現する手法のことである．以下は公式ドキュメントである．まず，ある単語 wiwi のベクトル wviwvi を取得 (word2vec)．ある単語 wiwi の IDF 値 idf(wi)idf(wi) を計算する．GMM で全単語ベクトルについてクラスタリングを行い，ある単語が各クラスタ ckck に属する確率 P(ck—wi)P(ck—wi) を取得する．語彙空間における各単語 wiwi について 5 を繰り返す．各クラスタ ckck についてクラスタの数だけ (*) を繰り返す．クラスタの数だけ計算したら，(**) を実行する．(*）クラスタを考

慮した新たな単語ベクトル wcvikwcvik を wvi × P(ck—wi)wvi × P(ck—wi) として算出．(**) IDF を考慮した新たな単語ベクトル wtviwtvi を idf(wi) × Kk wcvikidf(wi) × kK wcvik で算出 (は concatenation)．各ドキュメント (n から Nn から N) について 9 の操作を繰り返す．ドキュメント DnDn についてベクトルを初期化し，DnDn に含まれる単語 wiinDnwiinDn についてベクトルを足し合わせていき平均する．最後に得られた文書ベクトル DnDn をスパースにして SCDVDnSCDVDn を得る．[17]

第3章　本研究における問題定義と仮説

本章では，II 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，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他のホストに攻撃を仕掛けられてしまう，所謂踏み台攻撃の踏み台にされる問題である．
以下の図 3.1 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–である．問題として，2 番目のフロ–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が考えられる．高対話型 Honeypot で使用した OS 自体の新たな脆弱性を突かれることに限った状況で，3 番目のフロ–の Honeypot への侵入者に 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問題として挙げられる．

本研究ではこの中でも，2 番目のフロ–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，Honeypot の username のデフォルトが決まっているため，username から Honeypot であることを検知されてしまう問題もある．これらの検知手法を用いて，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い問題がある．

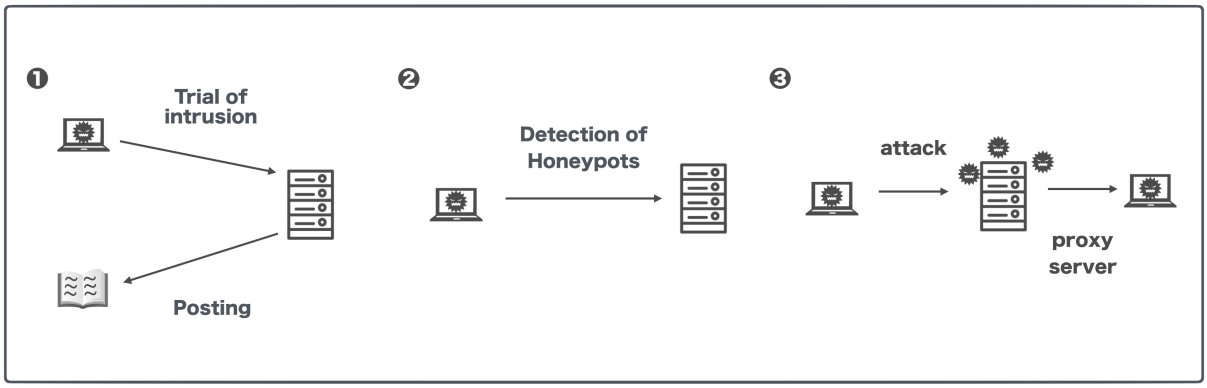


図 3.1: 不正な SSH 侵入者の想定行動フロー

3.1.1.1.1 Honeypot の Username の問題

SSH の低対話型 Honeypot の内、特に Cowrie は kippo を改良したものであるため，kippo のデフォルトの username である”Richard”が cowrie の username となっている．このため，username が”Richard”で，honeypot 特有の挙動をした場合に，侵入者に Honeypot であると検知されてしまう．

3.1.1.1.2 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． 2.2.2 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，92 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 3.1 とプログラム 3.2 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 root@localhost-neco:~$ echo -n hello
2 -n hello
3 root@localhost-neco:~$
```

プログラム 3．1 が通常の挙動でプログラム 3．2 が SSH の低対話型 Honeypot の挙動である．echo コマンドの-n オプションは改行出力末尾にしないようにするものである．しかし，実際の Shell の出力は改行がされない一方，Honeypot の挙動ではオプション部分

も出力されてしまい，末尾も改行されてしまう．これは SSH の低対話型 Honeypot 特有の挙動であるため，この挙動を観測することによって侵入者に侵入先が 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

3.3.1 と 3.3.2 で示す問題解決のための要点を踏まえると，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と考えた．

3.3.1 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とで，Honeypot への侵入者が実行できるコマンドの少なさによる，Honeypot であることの検知を回避することができる．

3.3.2 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とで，Honeypot について認知している侵入者が，侵入先を Honeypot であると検知することを回避することができる．

第4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，本研究の手法について概説する．

4.1　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つの要件を満たすために，本研究では低対話型Honeypotに実装されていないコマンドを実装し，さらに低対話型Honeypotの既実装コマンドで，低対話型Honeypotに特有の異常な挙動をするコマンドの修正を行う．略図を図4.1に示す．

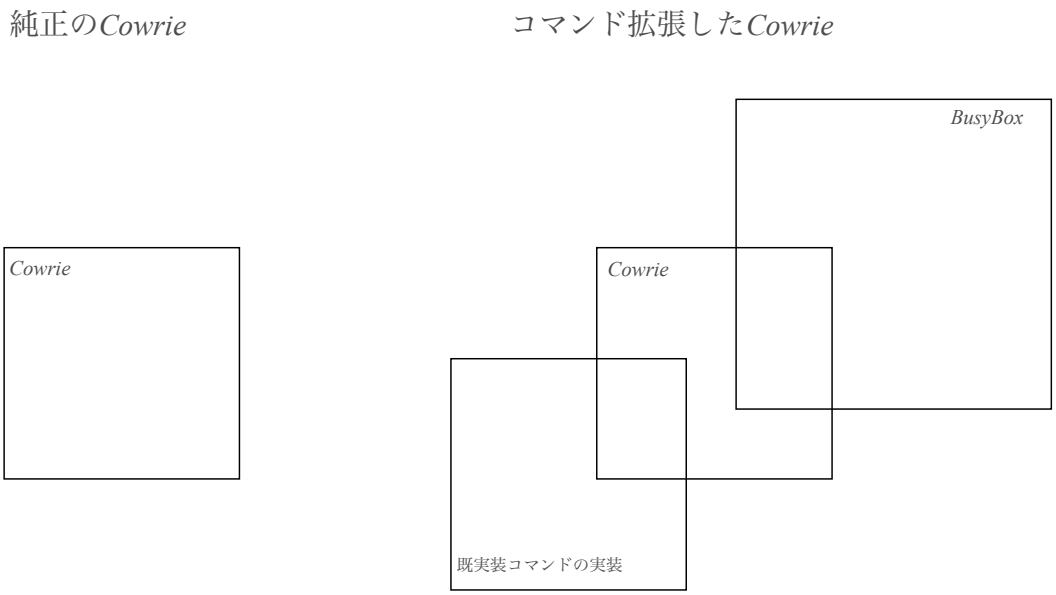


図 4.1: 低対話型 Honeypot の拡張

第5章　実装

本章では，低対話型 Honeypot と高対話型 Honeypot の設置環境についても示し，4.1 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．

5.1　実装環境

本研究で実装するシステムを構成するためのハードウェアおよびソフトウェアについて説明する．表 5.1 に詳細なバージョンを示す．

表 5.1: 実装環境

ハードウェア/ソフトウェア	実装環境	Version(date)
純正の Cowrie	CentOS5	1．4．0
修正済みの Cowrie	CentOS5	1．4．0（self made）
Honeywall	CentOS5	1．4

5.1.1　純正の Honeypot で未実装のコマンド種類

本研究において純正の Honeypot は Cowrie[18] 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[12] に含まれるコマンドの実装を行なった．2.2.2 や **??**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 92 しか存在しない．この差分を Python で実装する．
また BusyBox に含まれるコマンドと Cowrie の実装コマンド，Cowrie に未実装のコマンドの一覧は付録 A の表 A.1 に示しておく．

5.1.2 純正の Honeypot で未実装のコマンドの実装

5.1.1 で示した Cowrie に未実装のコマンドについての一部の実装を示す．他の実装は ?? の表に示す．Cowrie に実装されているコマンドは cowrie/src/cowrie/commands/に格納されており，ここに追加コマンドを実装する．

以下の 5.1 に cowrie/src/cowrie/commands/base.py に dmidecode コマンドを追加実装したものを示す．[19]

プログラム 5.1: dmidecode

```
class command_dmidecode(HoneyPotCommand):
    """
␣␣␣␣"""
    def call(self):
        """
␣␣␣␣␣␣␣␣"""
        self.write(b"""#␣dmidecode␣3.1
Getting␣SMBIOS␣data␣from␣sysfs.
SMBIOS␣3.0.0␣present.
Table␣at␣0xDDBA4000.

(snip)

Handle␣0x0055,␣DMI␣type␣13,␣22␣bytes
BIOS␣Language␣Information
␣␣␣␣Language␣Description␣Format:␣Long
␣␣␣␣Installable␣Languages:␣1
␣␣␣␣␣␣␣␣en|US|iso8859-1
␣␣␣␣Currently␣Installed␣Language:␣en|US|iso8859-1

Handle␣0x0056,␣DMI␣type␣127,␣4␣bytes
End␣Of␣Table\n""")
commands['dmidecode'] = command_dmidecode
```

追加実装以外ではファイルを追加することで，コマンドを追加したものを以下の 5.2 に示す．[20]

プログラム 5.2: diff コマンド

```
# coding: utf-8
import sys

class Diff:
  def __init__(self, a, b):
```

```
6       self.a = a
7       self.b = b
8
9     def solve(self):
10      self.result = None
11      self.createTable()
12      self.selectMatches()
13      self.connectPath()
14      self.genResult()
15      return self.result
16
17    def createTable(self):
18      self.table = [[x == y for y in self.b] for x in self.a]
19
20    def selectMatches(self):
21      self.matches = [ (i, j)
22        for i in range(len(self.table))
23        for j in range(len(self.table[i]))
24        if self.table[i][j]]
25
26      self.max_path = []
27      self.path = []
28
29      self.search((-1, -1)) # 左上から探索
30      # print self.max_path
31
32    def search(self, pos):
33      def isReachable(pos, match):
34        return match[0] > pos[0] and match[1] > pos[1]
35
36      if pos != (-1, -1): self.path.append(pos)
37      is_term = True
38      for match in self.matches:
39        if isReachable(pos, match):
40          is_term = False
41          next_pos = match
42          self.search(next_pos)
43
44      if is_term: # 終端の場合
45        if len(self.path) > len(self.max_path):
46          self.max_path = list(self.path) # 最良経路の更新
47      if pos != (-1, -1): self.path.pop()
48
49    def connectPath(self):
50      # self.path = [(0, 0)]
51      self.path = []
52      prev = (0, 0)
```

```
53      for pos in self.max_path:
54        p = prev
55        while p[0] < pos[0]:
56          p = (p[0] + 1, p[1])
57          self.path.append(p)
58        while p[1] < pos[1]:
59          p = (p[0], p[1] + 1)
60          self.path.append(p)
61        p = (p[0] + 1, p[1] + 1)
62        self.path.append(p)
63        prev = p
64      p = prev
65      while p[0] < len(self.table):
66        p = (p[0] + 1, p[1])
67        self.path.append(p)
68      while p[1] < len(self.table[0]):
69        p = (p[0], p[1] + 1)
70        self.path.append(p)
71      # print self.path
72
73    def genResult(self):
74      prev = (0, 0)
75      i_a = 0
76      i_b = 0
77      self.result = []
78      for pos in self.path:
79        if pos == (0, 0): continue
80        if pos[1] == prev[1]: # 縦への移動 : a の -
81          self.result.append((self.a[i_a], '-'))
82          i_a += 1
83        elif pos[0] == prev[0]: # 横への移動 : b の +
84          self.result.append((self.b[i_b], '+'))
85          i_b += 1
86        else: # 斜めへの移動
87          self.result.append((self.a[i_a], ' '))
88          i_a += 1
89          i_b += 1
90        prev = pos
91        # print self.result
92
93    def printResult(self):
94      for line, sign in self.result:
95        print sign, line,
96
97  def diff(filename_a, filename_b):
98    a = []
99    with open(filename_a) as f:
```

```
100       for line in f:
101         a.append(line)
102
103     b = []
104     with open(filename_b) as f:
105       for line in f:
106         b.append(line)
107
108     # a = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
109     # b = ['a', 'b', 'x', 'c', 'y', 'e', 'g']
110     solver = Diff(a, b)
111     result = solver.solve()
112     solver.printResult()
113
114 if __name__ == "__main__":
115     if len(sys.argv) != 3:
116       print "Usage: python %s fileA fileB" % sys.argv[0]
117       quit()
118     diff(sys.argv[1], sys.argv[2])
```

さらに，コマンド実行時に出力結果を print 関数で出力するのみのコマンドを追加したものの例を以下の 5.3 示す．本例では mount コマンドを以下のように実装した．本研究の実装はこちらがほとんどとなっている．

プログラム 5.3: mount コマンド

```
1  # coding: utf-8
2
3  string = '''sysfs on /sys type sysfs (rw,nosuid,nodev,
       noexec,relatime)
4  proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
5  udev on /dev type devtmpfs (rw,nosuid,relatime,size
       =16459820k,nr_inodes=4114955,mode=755)
6  devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,
       gid=5,mode=620,ptmxmode=000)
7  tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size
       =3294084k,mode=755)
8  /dev/sda1 on / type ext4 (rw,relatime,discard,data=ordered)
9  securityfs on /sys/kernel/security type securityfs (rw,
       nosuid,nodev,noexec,relatime)
10 tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
11 tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,
       relatime,size=5120k)
12 tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,
       mode=755)
13 cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,
```

```
    nodev,noexec,relatime,xattr,release_agent=/lib/systemd/
    systemd-cgroups-agent,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,
    noexec,relatime)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,
    nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,
    nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,
    noexec,relatime,pids)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,
    nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid
    ,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,
    nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,
    nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,
    nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,
    noexec,relatime,rdma)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev
    ,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,
    nodev,noexec,relatime,perf_event)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,
    relatime,fd=26,pgrp=1,timeout=0,minproto=5,maxproto=5,
    direct,pipe_ino=11662)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,
    pagesize=2M)
configfs on /sys/kernel/config type configfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,
    relatime)
/dev/sdb1 on /mnt type ext4 (rw,relatime,data=ordered)
lxcfs on /var/lib/lxcfs type fuse.lxcfs (rw,nosuid,nodev,
    relatime,user_id=0,group_id=0,allow_other)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,
    relatime,size=3294084k,mode=700,uid=1000,gid=1000)'''

print(string)
```

print 関数で出力するのみのコマンドを追加の手法として python のライブラリである subprocess を用いて標準入出力を行う．以下の本例 5.4 では ar コマンドを示す．

プログラム 5.4: ar

```
1 # coding: utf-8
2
3 import subprocess
4 import sys
5
6 cmd = "ar␣t"
7 opt = sys.argv[1]
8 cmdall = cmd + "␣" + opt
9 subprocess.call(cmdall.split())
```

第6章　評価と考察

本章では，予備実験と5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．

6.1　予備実験

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い．そのため，侵入者にHoneypotであると検知されることで，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，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いため，コマンドの追加実装を行い，Cowrie特有の異常な挙動を修正した．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のCowrieの両方でコマンドログの収集を行い，比較することで，収集ログのパターンの変化を観測でき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のCowrieであるということが明らかとなった．

6.1.1　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ものを設置し，ログを収集した．

6.1.2　侵入ログの収集環境

6.1.3　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．ここでの実装を紹介する．

6.1.4 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比較を行うことで，差異を評価した．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ことを示す．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．本研究の評価と評価手法の違いは，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．

6.1.5 結果

SSH の低対話型 Honeypot の稼働期間は 12/10 から 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表 6.1 に記す．

表 6.1: 予備実験の結果

検知したもの	純正の Honeypot	修正済の Honeypot
コネクション数	19829	27914
パターン数	53	91
コマンド数	470	841

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の Cowrie のスコアリングを行なった結果を以下の図 6.1 に記す．横軸はコマンド拡張を行なった Honeypot か素の Honeypot であるかを示している．縦軸は予備実験の評価手法によって算出されたコマンドの表れやすさを数値化したものであり，数値が高くなれば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ことを示している．予備実験ではコマンドの拡張実装をしたものの方がコマンドパターンとして存在しにくいものを観測できたという結果になった．

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．

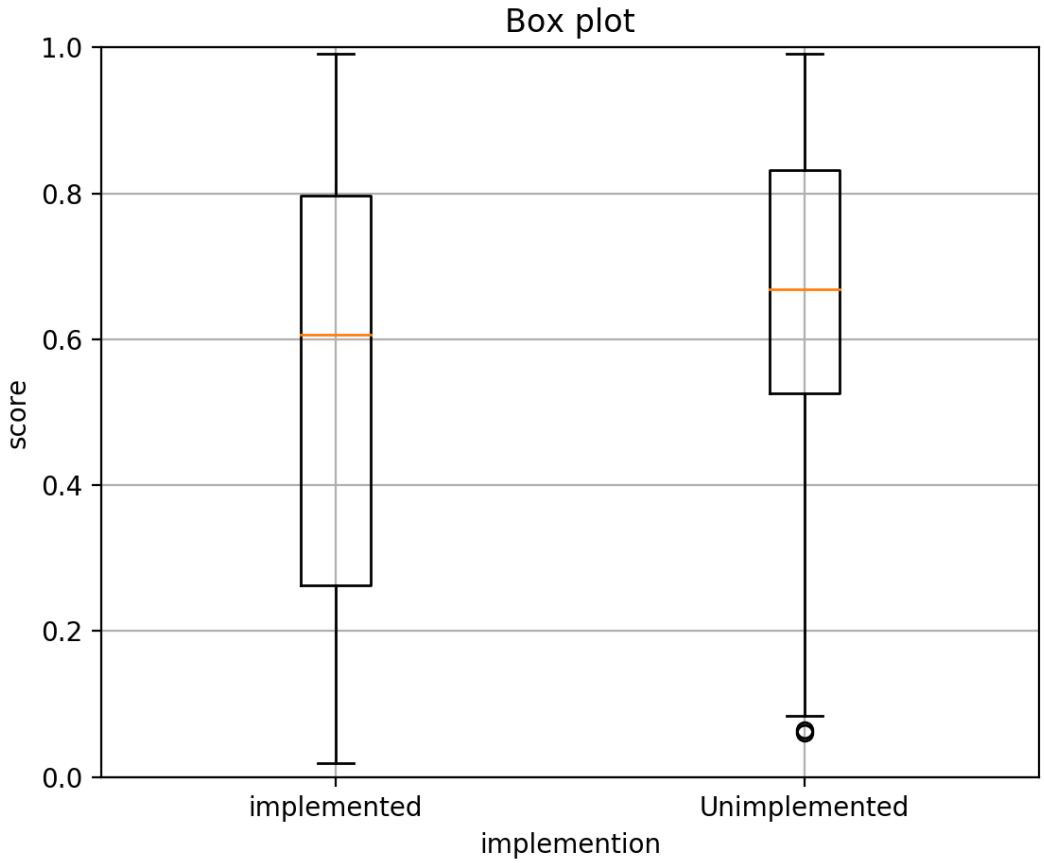


図 6.1: 純正の Cowrie と修正済の Cowrie のスコアリングによる比較

6.2 評価手法

本研究の仮説の検証手法としての評価として，3.2 節で述べた要件に対して評価を行う．予備実験では，素の低対話型 Honeypot よりも，コマンドを拡張した Honeypot の方がコマンドパターンが多く収集できることを示した．本研究では拡張した Honeypot で収集したコマンドログが，どれほど一般的な UNIX ユーザーの実行するコマンド [21] から離れたのかを評価した．
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．

また予備実験では，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 6.2 に示す．

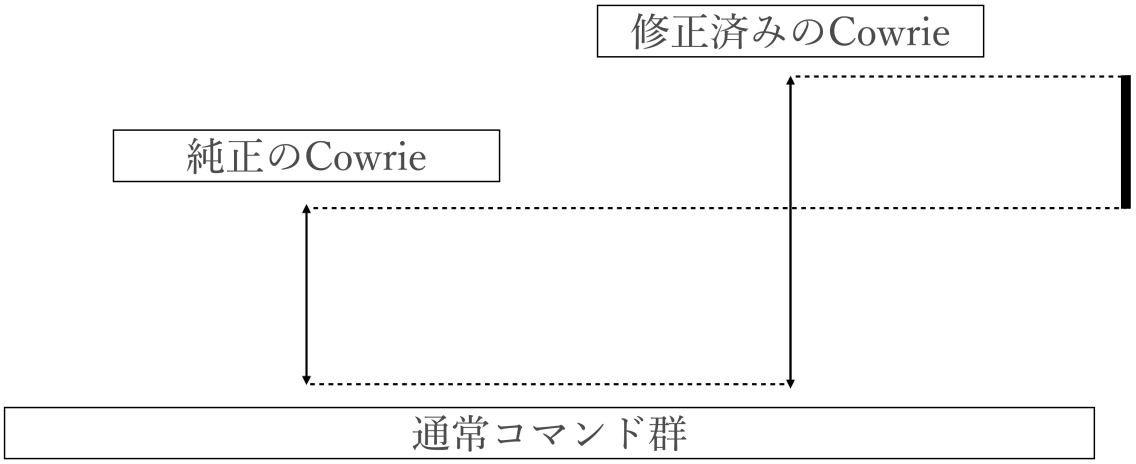


図 6.2: 予備実験の評価の概念図

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の Honeypot の侵入者の実行コマンドログと高対話型 Honeypot の侵入者の実行コマンドログを比較することで，修正済の Honeypot の侵入者の実行コマンドログが一般的な UNIX ユーザーの実行するコマンドから離れたのかを評価した．

6.2.1 コマンドログのスコアリング手法の実装の提案

コマンドログの比較を行う手法は多く存在する．例えば評価基準として，あるコマンドが実行された時に，そのコマンドは危険であるとしたブラックリストを作成するパターンマッチングの手法がある．また，攻撃であるとされたコマンドを ??で説明したマルコフモデルで学習させることで，攻撃性を表現する手法がある．しかしパターンマッチングであれば静的解析であるので未知の攻撃に対応ができず，マルコフモデルであれば現在の状態だけに依存して次の状態への推移確率が決まるので，未知の特徴量を無視してしまうので，いずれも未知の攻撃に対応できない．しかし，2.4 で説明した意味解析をコマンドログに導入することで，コマンド名が別でも同じような内容のコマンドを実行しようとし

た時に，それが同じような内容であることを検知できる自然言語処理における意味解析のニューラルネットワークのモデルを評価基準とすることで未知の攻撃にも対応できる．そのため，本研究では自然言語処理における意味解析のニューラルネットワークのモデルを評価基準とした．

6.2.2 機械学習を用いたコマンドログのスコアリング

本研究では，評価基準となる，自然言語処理における意味解析のニューラルネットワークのモデルとして Word2vec の skip-gram モデルを採用した．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 6.1 に示す．

プログラム 6.1: 3 つの実行コマンドの例

```
1  $ uname
2  $ free
3  $ ps x
4  $
```

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 6.3 にモデル化のフローを示す．

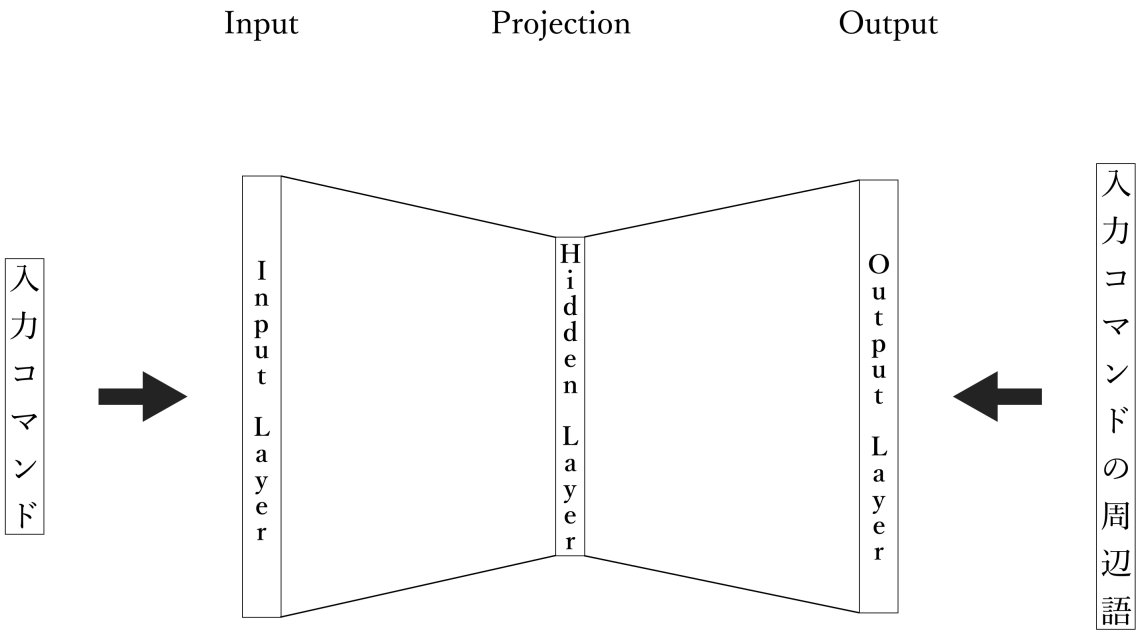


図 6.3: 評価のフロー [1] [2]

また，このようにして Honeypot の収集ログに対して各々のモデルを構築する．モデルは *HiddenLayer* に単語ベクトルのインデックスとして構築される．ここで SCDV [2 4 3 4] を使い，word2vec で取得した単語ベクトルを使い，idf 値を計算する． [2 4 3 1 1] 次に単語ベクトルごとにクラスタリングすることで，ある単語があるクラスタに所属する確率を算出する．クラスタを考慮した新たな単語ベクトルを再構築し，idf 値を考慮した新たな単語ベクトルを再構築する．1 セッションごとに含まれる単語ベクトルを平均化し，文章ベクトルを得る．

6.2.2.1 コマンド群データのベクトル表現の比較

本研究の評価手法によって任意の 2 つのコマンドログを比較した時の，各々のログの違いを文章分類の手法で算出した結果は以下の図 [6 2]，図 [6 3]，図 [6 4]，のようになった．

表 6.2: 修正前と後の honeypot のコマンドログの比較

	正解率	適合率	再現率	F1 値	session 数
素の低対話型 Honeypot	0.807909	0.782520	0.872437	0.825036	3951
修正済の低対話型 Honeypot	0.807909	0.842795	0.738251	0.787067	3660

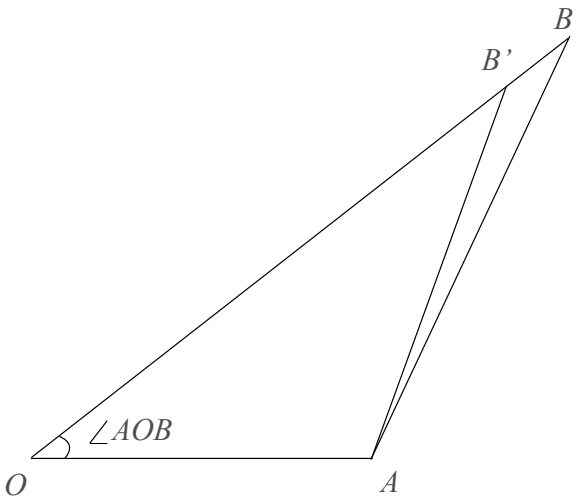
表 6.3: 修正前の honeypot と一般 UNIX ユーザーのコマンドログの比較

	正解率	適合率	再現率	F1 値	session 数
素の低対話型 Honeypot	0.778404	0.788169	0.765108	0.776467	3657
一般の UNIX ユーザー	0.778404	0.769086	0.791863	0.780308	3613

表 6.4: 修正後の honeypot と一般 UNIX ユーザーのコマンドログの比較

	正解率	適合率	再現率	F1 値	session 数
素の低対話型 Honeypot	0.822134	0.793420	0.892497	0.840047	3972
修正済の低対話型 Honeypot	0.822134	0.863229	0.744887	0.799703	3618

上記の表から，素の Honeypot と一般ユーザーのコマンドログが一番近く，修正済の Honeypot と一般ユーザーのコマンドログが一番遠いことが分かった．すなわち，ベクトル空間上において一般ユーザーのコマンドログの文章ベクトルを原点 O とし，素の Honeypot コマンドログの文章ベクトルと A，修正済の Honeypot コマンドログの文章ベクトルと B とした時に，$|\vec{OA}| < |\vec{AB}| < |\vec{OB}|$ であり，なす角 $\angle AOB$ は鋭角である．以下に図を示す．



したがって，$0 < n$ を満たす n と任意のベクトル $\vec{\alpha}$ を使うと，$\vec{OB}$ は $\vec{OB} = (1+n)\vec{OA}+\vec{\alpha}$ と表すことができる．したがって，修正済の Honeypot のコマンドログは一般ユーザーのコマンドログを始点とすると，素の Honeypot のコマンドログのベクトル方向よりも正の向きに遠くに位置することが分かった．
また，SCDV によって獲得した各々の Honeypot の攻撃ログにおける文章ベクトルを，t-SNE で次元削減することで可視化を行なった結果を図 6.4，図 6.5，図 6.6 に示す．

図 6.4: 修正前と後の honeypot のコマンドログの文章ベクトルの可視化

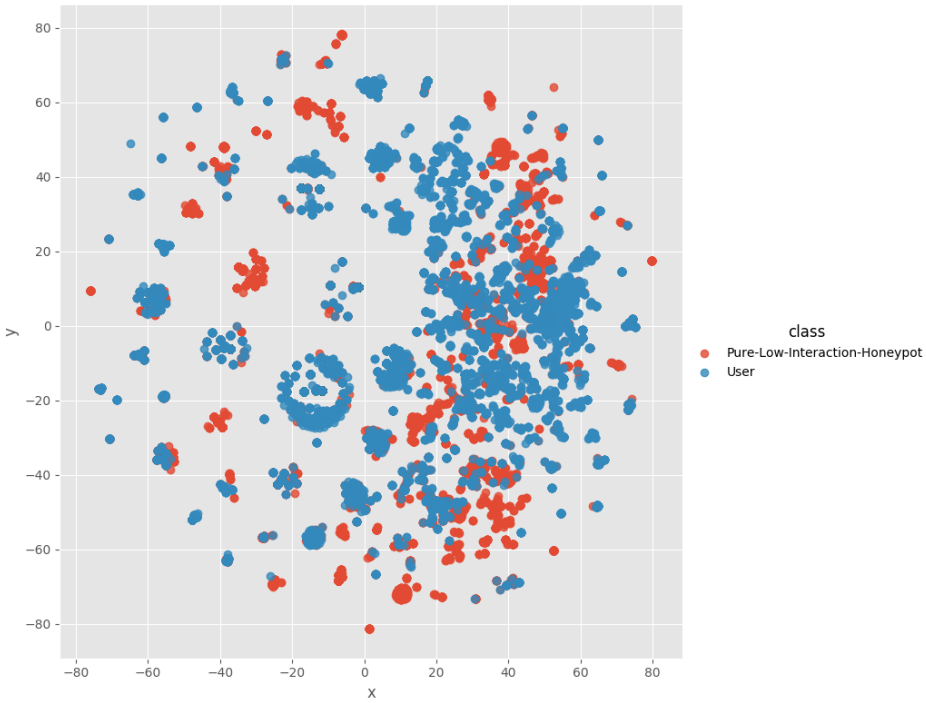


図 6.5: 修正前の honeypot と一般 UNIX ユーザーのコマンドログの文章ベクトルの可視化

図 6.6: 修正後の honeypot と一般 UNIX ユーザーのコマンドログの文章ベクトルの可視化

上記の図において，Fixed-Low-Interaction-Honeypot（修正済の Honeypot）は Pure-Low-Interaction-Honeypot よりも広く分布していることが分かり，他のログでは取れていないログを取得できたことが分かる．

6.3 考察

文章ベクトルの意味解析において高い精度を持つ [17]SCDV を用いた素の低対話型 Honeypot と修正済の Honeypot，高対話型 Honeypot の攻撃ログを文章ベクトル空間上で比較した結果，修正済の Honeypot のコマンドログは一般ユーザーのコマンドログを始点とすると，素の Honeypot のコマンドログのベクトル方向よりも正の向きに遠くに位置することが分かり，素の Honeypot のコマンドログと比較して，修正済の Honeypot は一般の Unix ユーザーのコマンドログにとっての異常なコマンドログを収集できることを示した．また，高対話型 Honeypot の攻撃ログの取得数が極端に少なく，その原因は，その Honeypot のサポートが最新版が 2009 年であり，対策を施されてしまった結果ではないかと考えられる．

第7章　関連研究

7.1　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．

7.1.1　AccessTracer

日本電気株式会社中央研究所の開発した異常行動検出エンジン AccessTracer[22] において，ユーザーの UNIX コマンドの履歴をオンライン忘却型学習アルゴリズムによる行動モデルの学習によってユーザーの普段と違うコマンド履歴を検知することができる．本研究のモデル構築におけるモデルの作成方法が異なる．

7.1.2　自然言語処理における意味解析

2.4 で述べたように，自然言語処理とは人間が日常的に使っている自然言語をコンピュータに処理させる一連の技術である．現在，意味解析において大きくシソーラス解析とベクトル空間分析の 2 つが手法として多く，本研究ではこのうちベクトル空間分析を使用した．そのため関連研究ではベクトル空間解析について述べる．

7.1.2.1　ベクトル空間解析

2.4.3 で述べたように，自然言語処理の意味解析の手法の一つにベクトル空間解析というものがある．本研究で取得した Honeypot のデータを用いて，ベクトル空間解析の方法でも評価を試みた．word2vec だけによる比較は以下の表 7.1 の通り．

表 7.1: word2vec による各々の honeypot のコマンドログの比較

	正解率	適合率	再現率	F1 値	session 数
素の低対話型 Honeypot	0.648172	0.638172	0.697262	0.719872	3629
修正済の低対話型 Honeypot	0.793772	0.828741	0.768549	0.847428	3939
高対話型 Honeypot	0.619941	0.628721	0.559174	0.608271	44
一般の UNIX ユーザー	0.691876	0.753382	0.691302	0.700291	3671

word2vec は，単語ベクトルのクラスタリングを行なった SCDV による評価と比較して，全ての評価基準においてやや精度の低い結果となった．

7.1.2.1.1 ベクトル空間モデル

2.4.3.1 で述べたように，ベクトル空間解析にはベクトル空間モデルというものが存在する． 2.4.3.1.1 で述べた TF-IDF を用いて，同一文章内での単語の出現頻度と，様々な文章におけるある単語の逆文書頻度の 2 つを重みとし，文章を多次元マトリクスで表現する．文章同士の距離をなす角 θ の，$0° \leqq \theta \leqq 90°$ における $\cos\theta$ の値の大きさによって文章の類似度を算出する．多次元マトリクスにおける 2 つの文章のベクトルの方向は文章の特徴であるので，$0° \leqq \theta \leqq 90°$ において θ の値が小さくなればなるほど，つまり $\cos\theta$ の値が大きくなればなるほど文章同士の類似度が高いということになる． ref で示したように，TF-IDF で重み付けされたベクトル空間モデルの $\cos\theta$ の値は，m 個の単語が使用されている文章 d における各単語の重要度を $w_{d1}, w_{d2}, w_{d3}, \ldots, w_{dm}$ とし、同様に n 個の単語が使用されている文章 e における各単語の重要度を $w_{e1}, w_{e2}, w_{e3}, \ldots, w_{en}$ とすると，

$$\cos\theta = \frac{\sum_{i=1}^{m}((tf(t_i,d)\cdot idf(t_i))(tf(t_i,e)\cdot idf(t_i))}{\sum_{i=1}^{m}\sqrt{(tf(t_i,d)\cdot idf(t_i))^2}\sum_{i=1}^{n}\sqrt{(tf(t_i,e)\cdot idf(t_i))^2}}$$

である．

第8章　結論

本章では，本研究のまとめと今後の課題を示す．

8.1　本研究のまとめ

本研究では，侵入者から Honeypot であることの検知を回避するために，低対話型 Honeypot の実行コマンドの挙動を本物の Shell の実行コマンドの挙動に近づけることを提案した．侵入者から Honeypot であることの検知を回避するためには，本物の Shell に実装されているコマンドの実装と，低対話型 Honeypot 特有の異常な挙動をするコマンドの修正を行う必要がある．そこで，本物の Shell に実装されているコマンドを低対話型 Honeypot に全て実装し，低対話型 Honeypot 特有の異常な挙動をするコマンドの修正を行った．低対話型 Honeypot の実行コマンドの挙動を本物の Shell の実行コマンドも挙動に近づけたかを検証するために，追加実装を施していない低対話型 Honeypot とコマンド拡張を行なった低対話型 Honeypot，高対話型 Honeypot を設置してコマンドログを収集し，一般の UNIX ユーザーの実行コマンドログと比較した．コマンドログの自然言語処理による意味解析を行ない，個々のコマンドの意味を多次元のベクトル空間上で表現することで，コマンド拡張を行なった低対話型 Honeypot のコマンドログが素の低対話型 Honeypot と比較して，一般的な UNIX ユーザーの実行するコマンドログから離れたことを明らかにした．このことから，素の Honeypot にコマンド拡張を行い，低対話型 Honeypot の実行コマンドの挙動を本物の Shell の実行コマンドの挙動に近づけることで，素の低対話型 Honeypot と修正済の Honeypot，高対話型 Honeypot の攻撃ログを，SCDV を用いた文章ベクトル空間上で比較した結果，修正済の Honeypot のコマンドログは一般ユーザーのコマンドログを始点とすると，素の Honeypot のコマンドログのベクトル方向よりも正の向きに遠くに位置することが分かった．また，素の Honeypot のコマンドログと比較して，修正済の Honeypot は一般の Unix ユーザーのコマンドログにとっての異常なコマンドログを収集でき，修正済の低対話型 Honeypot で取れたコマンドログが様々な攻撃パターンと異常な挙動を収集できることを示した．

8.2　本研究の課題と展望

本節では，提案手法の課題とその展望を述べる．SSH の低対話型 Honeypot に実装するコマンドについて，ディストリビューションごとに実装コマンドが異なるが，今回は BusyBox をそのまま Python で実装した．したがって，低対話型 Honeypot がエミュレー

ションしているディストリビューションで実装されているコマンドは必要条件しか満たしていない．また，3.1.1.1 で述べた通り，低対話型 Honeypot には本研究と着目した問題以外にも，SSH でのセッション確立におけるレイテンシの問題や，Honeypot の Username の問題も存在する．しかし本研究ではこれらを扱わず，本物の Shell の挙動を完全にエミュレートできていないため，侵入者から Honeypot であると検知されてしまう余地がある．

8.2.1 文章ベクトルの評価指標

6.2.2.1 節において，コマンドログを SCDV で文章ベクトル化することで評価したが，その評価指標として，Accuracy，precision，recall，f1-score を算出した．また，この評価指標の内，Accuracy によって文章の近さを測定したが，これが一番評価指標として適切か否かについての議論が本研究で行わなかった為，評価の再検討をする余地がある．

8.2.2 高対話型 Honeypot のログ収集の不足

本研究において honeywall を用いた高対話型 Honeypot のコマンドログの収集を行なったが，収集ログ数が極端に少なく，またその原因をつかむことができなかった．しかし，収集ログの数が現行の低対話型 Honeypot の Cowrie が圧倒的に収集することができたことを示すことができた．高対話型 Honeypot の設置と多数の収集ログ数の取得によって，本物の OS と差のないコマンドログを収集することができる為，コマンド実行における ”攻撃性 ” をベクトル空間上で示すことができる．これによって低対話型 Honeypot における攻撃性のあるコマンドログが収集できたかを評価することができる．

8.2.3 ネットワークセグメント毎のログ収集環境の違い

2018 年度の慶應義塾大学が主催する ORF において，ORF-NOC として従事し，その過程で rg のネットワーク内に低対話型 Honeypot を設置した．その結果，本研究における一日あたりのコマンドログの収集数よりも約 1．5 倍もの多くのコマンドログの収集数を取得することができた．したがって，同一の Honeypot でも配置する環境によって大きな差が出ることがわかり，設置する環境について考慮する余地がある．

8.2.4 ディストリビューションごとの実装コマンド

ディストリビューションごとの実装コマンドについてはバージョンの問題にも依存する．しかし，OS のリリースには LTS(Long Term Support) があり，広く使われている OS での実装コマンドの検証は十分に可能である．

8.2.5 様々な Honeypot のコマンドログの精度評価への応用

現在様々な種類の SSH の低対話型 Honeypot が普及しているおり，それらの Honeypot ごとに実装コマンドが異なる．そのため，Honeypot の種類ごとに攻撃ログを収集し，本研究の評価手法を用いることで，自然言語処理による意味解析において使用した Honeypot で取れたコマンドログが，高対話型 Honeypot のコマンドログにどれほど空間的距離が近いのかを検証することができる．

8.2.6 コマンド系ごとの評価への応用

SSH の低対話型 Honeypot を設置し，自然言語処理による意味解析を行なったコマンドログにおいて，個々のコマンドごとに持つ意味が異なる，あるコマンドが使用できないような実装を施すと高対話型 Honeypot のコマンドログにどれほど空間的距離が遠くなるかを検証できる．

付 録A　付録

A.1　実装コマンド

A.1.1　純正の Honeypot で未実装のコマンド種類

実装コマンド一覧

BusyBox に含まれるコマンドと Cowrie の実装コマンドの違い

表 A.1: 実装コマンド一覧

Busybox に含まれるコマンド	Cowrie に実装されているか
acpid	○
adjtimex	○
ar	○
arp	○
arping	○
ash	○
awk	×
basename	○
blockdev	○
brctl	×
bunzip2	×
bzcat	×
bzip2	×
cal	×
cat	○
chgrp	×
chmod	×
chown	○
chpasswd	○
chroot	×
chvt	×

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
clear	×
cmp	×
cp	○
cpio	○
crond	○
crontab	×
cttyhack	×
cut	×
date	○
dc	×
dd	×
deallocvt	×
depmod	○
devmem	○
df	×
diff	×
dirname	×
dmesg	○
dnsdomainname	○
dos2unix	×
dpkg	×
dpkg-deb	○
du	×
dumpkmap	○
dumpleases	×
echo	×
ed	×
egrep	×
env	×
expand	○
expr	×
FALSE	○
fdisk	○
fgrep	×
find	○
fold	×

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
free	×
freeramdisk	×
fstrim	×
ftpget	○
ftpput	×
getopt	×
getty	×
grep	×
groups	○
gunzip	○
gzip	○
halt	×
head	○
hexdump	×
hostid	○
hostname	×
httpd	○
hwclock	×
id	×
ifconfig	×
ifdown	×
ifup	○
init	○
insmod	×
ionice	○
ip	×
ipcalc	○
kill	○
killall	○
klogd	×
last	×
less	×
ln	×
loadfont	×
loadkmap	○
logger	○

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
login	○
logname	○
logread	○
losetup	○
ls	○
lsmod	○
lzcat	○
lzma	○
lzop	○
lzopcat	○
md5sum	○
mdev	○
microcom	○
mkdir	○
mkfifo	○
mknod	○
mkswap	○
mktemp	○
modinfo	○
modprobe	○
more	○
mount	○
mt	○
mv	○
nameif	○
nc	○
netstat	○
nslookup	○
od	○
openvt	○
passwd	○
patch	○
pidof	○
ping	○
ping6	○
pivot_root	○

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
poweroff	○
printf	○
ps	○
pwd	○
rdate	○
readlink	○
realpath	○
reboot	○
renice	○
reset	○
rev	○
rm	○
rmdir	○
rmmod	○
route	○
rpm	○
rpm2cpio	○
run-parts	○
sed	○
seq	○
setkeycodes	○
setsid	○
sh	○
sha1sum	○
sha256sum	○
sha512sum	○
sleep	○
sort	○
start-stop-daemon	○
stat	○
static-sh	○
strings	○
stty	○
su	○
sulogin	○
swapoff	○

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
swapon	○
switch_root	○
sync	○
sysctl	○
syslogd	○
tac	○
tail	○
tar	○
taskset	○
tee	○
telnet	○
telnetd	○
test	○
tftp	○
time	○
timeout	○
top	○
touch	○
tr	○
traceroute	○
traceroute6	○
TRUE	○
tty	○
tunctl	○
udhcpc	○
udhcpd	○
umount	○
uname	○
uncompress	○
unexpand	○
uniq	○
unix2dos	○
unlzma	○
unlzop	○
unxz	○
unzip	○

次ページに続く

前ページからの続き

Busybox に含まれるコマンド	Cowrie に実装されているか
uptime	○
usleep	○
uudecode	○
uuencode	○
vconfig	○
vi	○
watch	○
watchdog	○
wc	○
wget	○
which	○
who	○
whoami	○
xargs	○
xz	○
xzcat	○
yes	○
zcat	○

以上

謝辞

本論文の執筆にあたり，ご指導頂いた慶應義塾大学環境情報学部村井純博士，同学部教授中村修博士，同学部教授楠本博之博士，同学部准教授高汐一紀博士，同学部教授三次仁博士，同学部准教授植原啓介博士，同学部准教授中澤仁博士，同学部準教授 Rodney D. Van Meter III 博士，同学部教授武田圭史博士，同大学政策・メディア研究科特任准教授鈴木茂哉博士，同大学政策・メディア研究科特任准教授佐藤 雅明博士，同大学 SFC 研究所上席所員斉藤賢爾博士に感謝致します．

特に斉藤氏には重ねて感謝致します．研究活動の中で，様々な見地からのご助言をいただき，研究という枠組みの中で伸び伸びとした考えで研究生活を送ることができました．博士の指導なしには，卒業論文を執筆することはできませんでした．

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・中澤・武田合同研究プロジェクトに所属している学部生，大学院生，卒業生の皆様に感謝致します．なかでも慶應義塾大学政策メディア・研究科 阿部涼介氏に重ねて感謝いたします．入学当初から指導していただき，卒業論文作成に際してもつきっきりのご指導を頂きました．楽しい研究生活を送ることができたのは他ならぬ阿部氏によるもので，私生活においても多大な良い影響を受け，人生において欠かせない時間を過ごさせていただきました．また，卒業生の黒米

また NECO 研究グループとして様々な意見や発想を与えてくださった，塚越広彬氏，田中公人氏，渡部貴博氏，瀬川雅弘氏，島津翔太氏，宮本眺氏，風間宏治氏，松本光生氏，飯田悠斗氏，木内啓介氏，田中蓮氏，梶原留衣女史，倉重健氏，渡辺聡紀氏，高橋辰太朗氏に感謝致します．皆様にはグループのリーダとして至らない私を，何度も助けて頂きました．

研究室生活の中で多くを共にした，城一統氏，豊田安信氏，小西遼氏，安井瑛男氏，小林怜央氏，幅野莞佑氏，矢内洋祐氏，土田悠輝氏，石川達敬氏，熊谷啓孝氏，Korry Luke 氏，鈴木恒平氏に感謝致します．氏らと楽しい研究生活を共にしたことで、あらゆる苦難を乗り越えることができました，

黒米祐馬氏に感謝します．最初に Honeypot の着想を得られたのは氏の協力のおかげです．卒業後も研究に関するアドバイスを頂き，黒米氏なしでは本研究のテーマを決めることができませんでした．

同じクラスの同期である，波多光氏，國枝勇佑氏，田口義也氏に感謝します．氏らには全く違う学問分野からの意見を頂いたり，大学生活において様々な意見を交換させて頂きました．

芳田耕平氏に感謝します．学外で一番研究相談に乗って頂きました．

疋田りか女史に感謝します．私生活で全面的に支えて頂き，研究面においても発表練習に付き合って頂き，適切なアドバイスや客観的な意見を頂きました．

ここには書ききれない，自分の人生の中で出会った全ての人に感謝致します．様々な出会いがあり，意見を交換してきたからこと得られたものがたくさんあります．自分一人では何もできませんでした．

最後に，これまで育てて頂き，大学でこのような研究の機会を与えてくださった，父 雅哉，母 康子，妹 紀香に感謝します．

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] honeynet. High-interaction-honeypot. https://projects.honeynet.org/honeywall/, 2014.

[4] Kippo. https://github.com/desaster/kippo/.

[5] Single board computer. http://fablabjapan.org/.

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

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

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

[9] 消費者庁. Kippo と cowrie の実装コマンドの違い. https://etherscan.io/stat/miner, 1992.

[10] honeynet. honeywall. https://projects.honeynet.org/honeywall/, 2014.

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

[12] Busybox. https://proofofexistence.com/.

[13] Geoffrey Hinton Laurens van der Maaten. Visualizing data using t-sne. 2008.

[14] Wordnet. https://wordnet.princeton.edu/.

[15] Wordnet wikipedia. https://ja.wikipedia.org/wiki/WordNet/.

[16] 出口 利憲. ベクトル空間法による文書の類似度の算出. http://www.gifu-nct.ac.jp/elec/deguchi/sotsuron/hayashi/node20.html/, 2010.

[17] Bhargavi Paranjape Harish Karnick Dheeraj Mekala, Vivek Gupta. Scdv : Sparse composite document vectors using soft clustering over distributional representations. https://arxiv.org/pdf/1612.06778.pdf, 2017.

[18] Cowrie. https://github.com/cowrie/cowrie/.

[19] soji256. dmidecodecommand. https://soji256.hatenablog.jp/entry/2018/09/06/075700/, 2018.

[20] smurakami. diff command with python. https://github.com/smurakami/diff, 2014.

[21] UCI. Unix user data data set. http://archive.ics.uci.edu/ml/datasets/unix+user+data, 2012.

[22] 丸山祐子 山西健司, 竹内純一. データマイニングに基づくセキュリティ・インテリジェンス技術の研究開発. 先端技術大賞応募論文, 19:14, 2004.