Computer Security Symposium 2016
11 - 13 October 2016

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

山本　健太[†1]　　齊藤泰一[†2]

概要：ハニーポットは，脆弱に見せかることにより攻撃を誘うシステムであり，攻撃者の行動を観察したり攻撃手法を解析したりする目的で利用されている．それらの多くは,実際のサーバの機能を部分的にエミュレートすることによりハニーポットとしての機能を提供している．しかし，攻撃者が実行可能なアクションが限られるため，十分な解析ができなかったり、囮であると気づかれてしまう可能性がある．本研究では Linux コンテナ技術を利用することで限りなく実物のサーバーに近い環境を提供しつつ，解析が容易な SSH ハニーポットのシステムの提案・評価をする．

キーワード：Honeypot, SSH, Linux container, Docker

SSH honeypot based on Linux container technology

Kenta Yamamoto[†1]　Taiichi Saito[†2]

Abstract: Honeypot is a computer system that works as a decoy to lure attackers for observing their behavior and analyzing their attack techniques. Many of existing honeypots emulate partial functions of real servers such as HTTP servers, FTP servers and SSH servers. However, because the honeypots restrict attackers' actions, they are can be easily detected and cannot collect sufficient attackers' footprints. In this paper, we propose a new honeypot system that provides an environment close to real server by using Linux container technology.

Keywords: Honeypot, SSH, Linux container, Docker

1. はじめに

ハニーポットは OS やアプリケーションの脆弱性などにより攻撃者を誘惑し不正アクセスを促すシステムのことで，ハニーポットを利用して不正アクセスを行った攻撃者の行動や，攻撃手法に関する情報を収集することができる．低対話型ハニーポットは，囮にしたいシステムやプロトコルをエミュレートすることによって，攻撃者から様々な情報を得ることができる．エミュレートするシステムやプロコトルによって様々なハニーポット[1]が提供されている．

SSH ハニーポットは SSH プロトコルを解釈して攻撃者にサーバーへのログインを促し，攻撃情報を収集することができる．SSH ハニーポット実現する方法として既知の脆弱性を持った OpenSSH（sshd）をサーバーに配置する方法か，低対話型 SSH ハニーポットを配置する方法が考えられる．脆弱性を持った OpenSSH を配置する方法は，侵入に成功した攻撃者がサーバーの環境そのものを利用することができるためリスクが高い方法である．低対話型は SSH プロトコルやシェル環境，ファイルシステムなどをエミュレートするため，比較的安全に攻撃情報の収集が可能である．しかし，攻撃者が実行できるアクションが極端に少なくなるため，十分な攻撃情報を収集できなかったり，囮であると気づかれてしまう可能性がある．低対話型は自動化された攻撃には有効な手法だが，熟練した手動の攻撃には失敗する．

本稿では，Linux コンテナを利用した SSH ハニーポットを提案し，その評価を行う．Linux コンテナは，ファイルシステム，プロセス空間、ネットワーク設定をホストから隔離することができる OS レベル仮想化手法である．提案する Linux コンテナを利用した SSH ハニーポットは，攻撃者が実行できるアクションが実際の Linux システムとほとんど同等であるという，高対話型の側面をもつ．一方で OS レベル仮想化によるホストからのファイルシステムの隔離と，オーバーヘッドの少ないコンテナの生成・破棄が可能である低対話型のような側面をもつ．

2. SSH ハニーポットの現状

2.1 高対話型ハニーポット

ハニーポットを運用するには脆弱なサーバーやアプリケーションを用意する必要がある．物理サーバーに脆弱な OS やアプリケーションを動作させ，ハニーポットとして運用するスタイルを高対話型と呼ぶ．高対話型では，実物の OS やアプリケーションを使うことから，高度な攻撃手法を観察できることが期待できる．しかし，攻撃者は OS やアプリケーションの機能を全て使うことができるため，物理的なマシンの破壊，ファイルシステムの破壊，BOT として運用などのリスクがある．

物理サーバーへのダメージリスクを低減させる手法としてしばしば仮想化ハニーポットが利用される．仮想化ハニーポットは，仮想マシン上で OS を動作させ，それをハニーポットとして利用するものである．仮想マシンを複数起動することでイントラネットを再現することも容易である．

物理マシンを利用した高対話型より容易に仮想マシンの生成と破棄が可能である点が大きな違いとなる．

仮想化ハニーポットは物理マシン相当で OS を動作させる関係上，物理サーバーの要求スペックが高くなる傾向がある．また，仮想マシン生成や破棄のオーバーヘッドが大きいため，実装によっては連続するリクエストやセッションを捌くことが難しい場合がある．

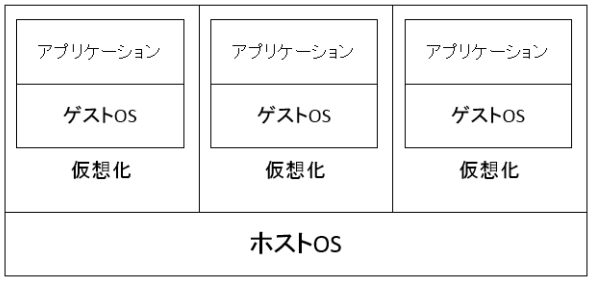

図 1．仮想マシン実行の構成イメージ図

物理サーバーへのダメージのリスクや，コンピューターリソースの消費を抑えるため，低対話型ハニーポットが利用される．

2.2 低対話型ハニーポット

低対話型ハニーポットは，あるプロトコルやアプリケーションをエミュレートすることで攻撃者をおびき寄せるハニーポットである．低対話型 SSH ハニーポットを提供するアプリケーションとして“ Kippo”や“Cowrie”がある．Kippo や Cowrie は SSH でログインを試みたセッションや，shell や exec コマンドの履歴を記録することができる．SSH ハニーポットにログインした攻撃者は偽の Linux ファイルシステム上で不正アクセスのための活動を行う．実際には利用できるコマンドは全てフェイクであり，あくまでそのコマンドが動作しているかのように振舞うだけである．攻撃者にスクリプトやバイナリの実行権限はなく，攻撃者がログアウトするとファイルシステムはデフォルト状態に戻り，次の攻撃者のために待機する．低対話型は攻撃者のアクションを制限することで，ホストサーバーへ影響が出ないように作られていることが特徴である．

また、ハニーポット上で実行可能なコマンドは制限されている．Kippo では頻繁に使われるコマンドはエミュレートしているが，コマンドの出力が不完全，いくつかのコマンドラインオプションがサポートされていない，リダイレクト，パイプなど使用不可など問題がある．例えば echo コマンドの-e, –n オプションが Kippo ではサポートされない．攻撃者が echo コマンドの出力によってその環境がハニーポットであるか判断をすることが可能である．

Cowrie は Kippo のフォークバージョンとして開発されている．Kippo で対応が不十分だったコマンドを修正し，エミュレートするコマンドも増えている．しかし，実際のコマンドがアップデートなどで，オプションや出力のフォーマットが変化した場合，Cowrie でそのコマンドの再開発が必要になってくる．また，Cowrie や Kippo には静的な出力を返すコマンドも存在し，複数回コマンドを実行することでハニーポットによるコマンドエミュレートであると判断できる可能性がある．例えばメモリの消費量を表示する“free”コマンドは Kippo や Cowrie は必ず決まった出力をするよう設計されている．また，RHEL7 や Ubuntu 16.04 では free コマンドの出力フォーマットが変化しており，Cowrie でも対応が必要になる課題がある．

3. Linux コンテナ

3.1 Linux コンテナ

Linux コンテナとは Linux カーネルが提供する OS レベル仮想化手法である．Xen や KVM のような仮想化システムと異なり, Linux コンテナは OS レベル仮想化システムである．隔離された Linux ファイルシステムを実行することができる．Linux コンテナは，ファイルシステムをホストからの隔離だけでなく，プロセステーブルやネットワーク設定の隔離や，メモリ容量，CPU パワーの使用制限を可能とする．Linux コンテナはファイルシステムやメモリなどのリソースの隔離を行うだけで，エミュレーションなどは行わない．そのため Linux コンテナにはマシン起動時などに発生する仮想化特有のオーバーヘッドが存在しない．また，ホストマシンの Linux カーネルを共有して利用するため，使用メモリ量を抑えることができる．

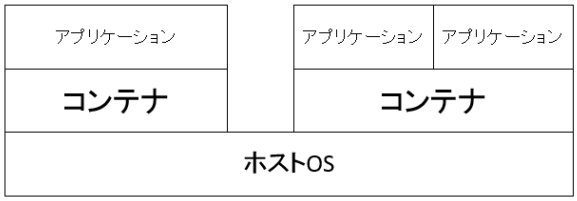

図 2．コンテナ実行の構成イメージ図

3.2 Docker

Docker とは Linux コンテナを取り扱うためにフロントエンドとして開発されているソフトウェアである．docker コマンドを利用することにより容易に Linux コンテナと仮想ネットワークを取り扱うことができる．Docker はベースとなるファイルシステム“イメージ”を起点としコンテナを実行する．Docker コンテナで採用されているブロックデバイスは，起点となるイメージからの差分のみが保持される設計のため，ディスク消費量を抑えることができる．

Docker で生成した Linux コンテナ内ではホスト環境と同等の Linux システムを獲得できるが，セキュリティ上コンテナ内で禁止されているアクションがある．mount コマンド，raw ソケットへのアクセス，新しいデバイスの作成，ファイルの属性フラグの変更，カーネルモジュールの読み込み等である．

3.3 T-Pot

T-Pot は Docker を利用した低対話型ハニーポット統合管理 OS の名称である. 既存のハニーポット製品や IDS を Docker で生成したコンテナ上で動作させて，統合管理できる点が特徴である．Suricata，Honeytrap，Cowrie，Glastopf，Dionaea などのハニーポットや IDS が利用できる. Kibana, Logstash ，ElasticSearch を組み合わせて，各ハニーポットのログを統合管理できる仕組みを持つ．Docker を利用しているものの，運用するには T-Pot 提供元から配布される OS ディスクイメージを利用しなければならない.

3.4 dockerpot

xinetd と Docker を組み合わせたハニーポット運用法である. スーパーサーバーとして動作する xinetd がホストマシンにおいて特定ポートで待ち受ける．特定のポートにアクセスがあった場合, 対応するサーバを搭載した Docker コンテナを起動して，iptable によりホスト側ポートとコンテナとの間でポートフォワーディングを設定する．接続元のリモートホストに対して iptables がパケットを転送させるため，1 つの接続元リモートホストに対して，1 つの Docker コンテナが起動される．1 コンテナに割り当てるメモリ量を制限することで DoS に強くなるとされている. ただしサーバーにアクセスしてきたリモートホストの数だけコンテナを起動するため，複数のリモートホストからアクセスがあった場合，ホストマシンの動作に影響を及ぼす懸念がある.

4. 提案手法

Linux コンテナ上で動作する高対話型の持つ自由度の高さと低対話型のファイルシステム隔離の容易性を併せ持つ SSH ハニーポットを提案する．xinetd を用いた手法のようなホスト側でパケットを待ち受けるのではなく，攻撃者とのパケットのやり取りはコンテナだけで完結できるようにし，ホスト環境はコンテナの生成，解析などのコンテナ制御の機能をもたせるものとする．できるだけホスト環境に影響の出ないような構造を提案する.

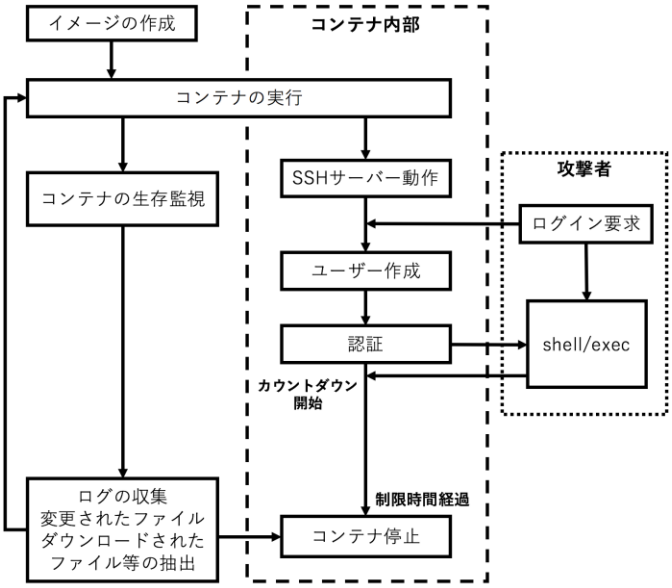

図 3. 提案 SSH ハニーポットのライフサイクル

4.1 構成

提案する SSH ハニーポットを運用するために制作したスクリプトは，ホスト側でコンテナの生成と解析を行う python スクリプト **`manager.py`**，コンテナ内部で SSH サーバーとして動作する python スクリプト **`sshserver.py`**，ユーザー名とパスワードを引数に取るユーザー作成シェルスクリプト **`adduser.sh`** の 3 種類である.

4.1.1 ホスト側スクリプト manager.py

`manager.py` は，主にハニーポットに使用する Docker イメージを作成する機能，コンテナを作成する機能，停止したコンテナからデータの抽出を行う機能をもつ．このスクリプトで提供する一部の機能は Docker の API を取り扱うライブラリ docker-py を通じて行う．停止した Docker コンテナからデータを抽出する手法は，docker diff コマンドと docker cp コマンドを利用する．docker diff コマンドは，コンテナが生成されてから，追加したファイル(A)，編集したファイル(C)，削除したファイル(D) を一覧表示するコマンドである．docker cp コマンドを使ってコンテナ内のファイルをホスト環境にコピーすることができる．停止したコンテナが破棄（削除）されない限り，いつでも実行することができる.

4.1.2 コンテナ内部のスクリプト sshserver.py

`sshserver.py` の機能は，SSH サーバーとして待ち受けする機能と, コンテナ生存のためのカウントダウン機能，攻撃者が SSH でログインするためのユーザー名とパスワードのバリデーションを行う機能がある.

このスクリプトによる SSH サーバーは SSH2 プロトコルを解釈し，攻撃者に exec，shell などの機能を与える．攻撃者はパスワード認証によるログインが可能である.

この SSH サーバーはハニーポットとして, 侵入の容易さ

が開発要件となる．攻撃者は任意のユーザー名とパスワードの組み合わせでログインできるような作りでなくてはならない．任意のユーザー名とパスワードでログイン可能なシステムを実現するために，攻撃者によるログイン要求の直後に，攻撃者の入力したユーザー名とパスワードの組み合わせでコンテナ内にユーザーを作成する．

コンテナ内でスクリプトが動作していることが攻撃者に悟られる可能性を極力無くすため，動作時のプロセス名は実在のサービスを模したプロセス名（sshd）として動作する．コンテナ内ではスクリプトは root 権限で動作するため，一般ユーザーでログインした攻撃者は SSH サーバーを kill することができない．

4.1.3 ユーザー作成スクリプト adduser.sh

adduser.sh はコンテナ内部で任意のユーザーを作成するためのスクリプトである．**adduser.sh** は Linux ユーザー作成のほかホームディレクトリの作成と.bash_history ファイルの定義を行う．**adduser.sh** は，攻撃者によるユーザー名とパスワードの認証要求を **sshserver.py** が受け付けた時に，**sshserver.py** によって呼び出される．予めユーザーを作成するのではなく，要求時に作成することによって任意のユーザー名とパスワードによるログインを実現している．ユーザー名およびパスワードが Linux システム上で有効な文字列であることを **sshserver.py** でバリデーションした上で **adduser.sh** を呼び出す構造である．

```
#!/bin/bash
if [ $(id -u) -eq 0 ]; then

username=$1
password=$2
egrep "^$username" /etc/passwd >/dev/null
if [ $? -eq 0 ]; then
   echo "User exists"
   exit 1
else
   pass=$(perl -e 'print crypt($ARGV[0], "password")' $password)
   useradd -s "/bin/bash" -m -p $pass $username
   su - $username -c "bash -c 'exit'"
   echo "export PATH=$PATH:/usr/sbin:/sbin:/usr/local/sbin" >>
/home/$username/.bash_profile
   echo 'shopt -s histappend' >> /home/$username/.bash_profile
   echo 'PROMPT_COMMAND="history -a;
$PROMPT_COMMAND"' >> /home/$username/.bash_profile
   echo 'set -o history' >> /home/$username/.bash_profile
   echo 'HISTFILE=/home/$username/.bash_history' >>
/home/$username/.bash_profile
   su - $username -c "bash -c 'exit'"
   [ $? -eq 0 ] && echo "created user" || echo "failed"
fi
else
echo "you are not root"
exit 2
fi
```

コード 1．ユーザー作成スクリプト

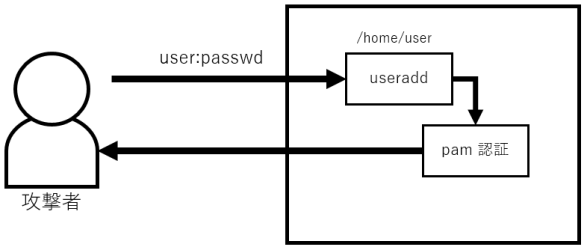

図 4．SSH ハニーポットのログインフロー

4.2 コンテナの展開

ファイルシステムは Ubuntu 14.04(Trusty)の Docker イメージを利用する．Ubuntu イメージでは Ubuntu リポジトリと，パッケージ管理システム apt が利用できる．apt を利用して SSH サーバーの動作に必要なライブラリを導入できる．適切なライブラリを用意すれば CentOS なども利用可能である．Docker の Ubuntu イメージは軽量化のため，製品版では導入されているような一部の頻出コマンドが導入されていない．SSH ハニーポットの Docker イメージを作成する際に導入する必要がある．

Docker イメージを作成するには“Dockerfile”と呼ばれるイメージの構成を定義したファイルを利用する．

Dockerfile に記述された命令をコンテナ内で実行しイメージを作成する．

FROM 命令はベースとなるイメージを指定する．今回は ubuntu(trusty)のイメージを利用している．

RUN 命令はコンテナで実行するコマンドを指定する．ベースが Ubuntu のイメージであるため，パッケージを導入するコマンドに apt を利用している．apt を利用して SSH サーバーの動作に必要な python パッケージを導入する．その他，攻撃者がコンテナ内で活動するとき利用するコマンドも apt を利用して導入する．今回は nano，wget，curl コマンドを導入する．

ADD 命令はホスト側のファイルをコンテナに転送する命令である．SSH サーバーとなるスクリプトとユーザー追加スクリプトをコンテナ内に転送し，実行権限を与える．

コンテナ内に転送したスクリプトは隠しファイルとしてリネームする．

```
FROM ubuntu:trusty
RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y python python-dev ¥
  build-essential python-pip nano wget curl libssl-dev libffi-dev
RUN apt-get clean
RUN pip install twisted pam cryptography setproctitle
ADD src/pot/sshserver.py /root/.s.py
ADD src/pot/adduser.sh /root/.a.sh
RUN chmod 500 /root/.s.py && chmod 500 /root/.a.sh
```

コード 2．SSH サーバーコンテナ作成用 Dockerfile

Dockerfile に記述された命令をすべて実行し終えると，コンテナが停止しその状態をイメージ化する

docker run コマンドを利用して，Docker イメージからコンテナを立ち上げて SSH サーバーを動作させる．動作中はホストサーバーの標準出力にコンテナ内で動作している SSH サーバーのログが出力される．

```
docker run -t -h privateServer -p 22:22 e0a95c7d9d52 /root/.s.py
```

コード 3．SSH サーバーコンテナを作成

イメージ ID“e0a95c7d9d52”起点として docker run コマンドによってコンテナを起動する．docker run コマンドの-p オプションでは，コンテナ上で待ち受けるポートをホストの NIC で待ち受けるポートに割り当てることができる．ホスト上でパケット待ち受ける仕組みは不要で，該当ポートに来たパケットは直接コンテナに転送される．/root/.s.py に配置した sshserver スクリプトが終了するとコンテナも自動的に停止する．

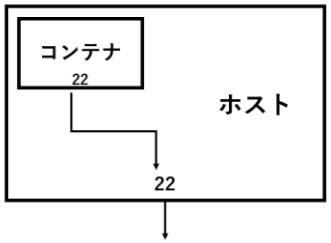

図 5．コンテナ側ポート 22 番をホスト側 22 番に割当

4.3 コンテナ内セキュリティ

コンテナ内での攻撃者の動作がホスト環境および外部ネットワークに影響しないように，コンテナ内での活動においていくつかの制限を設ける．

4.3.1 root ログインの制限

root ログインを禁止する．攻撃者は一般ユーザーのみ権限でのみ活動できる．また，root 権限実行を防ぐために，攻撃者のログインするユーザーは sudo コマンドを利用できないようにする．

4.3.2 ネットワークの制限

Bot や C2 サーバーとして運用されることを防ぐために，コンテナに生存の制限時間を設ける．ユーザーログイン後，一定時間が経過するとコンテナを停止し，環境がリセットされた新規のコンテナを起動する．同様に Bot 運用や C2 サーバーとの通信を防ぐために，ネットワークトラフィックに送受信バイト数の閾値を超えることができない制限を設ける．また，送信トラフィックに対して帯域幅を制限する．

4.3.3 コンピューターリソースの制限

攻撃者がコンテナ内で高い負荷のかかる活動を行いホスト環境に影響が出ることを防ぐために，コンピューターリソースの制限を設ける．docker run コマンドによるコンテナ起動時に以下のオプションを加える．

-m 128m --cpuset-cpus="0-1"

-m オプションによる 128MB のメモリ制限を設けた．

--cpuset-cpus オプションは CPU コアを割り当てるオプションである．0 及び 1 コアをコンテナに割り当てた．

ディスク容量の制限は，Docker ではデフォルトでコンテナのディスク領域を 10GB に制限しているため，変更は行わないものとする．Docker のストレージオプションを変更することで，コンテナで使用できるディスクサイズを変更することができる．

その他 docker run コマンドでは，--cpu-period，--cpu-quota，-memory-swap，-device-read-bps，-device-write-bps 等のオプションによる CPU 使用率，メモリスワップ容量指定，ブロックデバイスのアクセス速度制限が利用可能であるが，今回は使用しない．いずれもコンピューターリソースの制限には有効な手段だと考える

5. 解析

5.1 基本データの収集

停止させたコンテナ及びサーバーが出力したログから攻撃者の情報を収集することができる．ユーザー名，パスワードの組み合わせと送信元 IP アドレスがサーバーのログから抽出できる．ログは docker logs コマンドで出力可能である．ログそのものは sshserver.py が出力したものである．docker はコンテナ起動の際に指定したコマンドの出力をログとして保存する．

以下に示すログはすべてダミーであり，Source IP アドレスは RFC5737 で定められた Documentation Address Block である．

```
……..
2016-08-04 00:00:00+0000 [-] USERNAME: ubuntu
2016-08-04 00:00:00+0000 [-] PASSWORD: 12345
2016-08-04 00:00:00+0000 [-] Start counting down
2016-08-04 00:00:00+0000 [SSHService ssh-userauth on
    SSHServerTransport,0,203.0.113.11] admin authenticated with
    password
2016-08-04 00:00:00+0000 [SSHService ssh-userauth on
    SSHServerTransport,0,203.0.113.11] starting service ssh-connection
2016-08-04 00:00:01+0000 [SSHService ssh-connection on
    SSHServerTransport, 0, 203.0.113.11] got channel session request
2016-08-04 00:00:01+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,0,203.0.113.11] channel
    open
2016-08-04 00:00:01+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,0,203.0.113.11] pty request:
    xterm (40, 130, 1040, 720)
2016-08-04 00:00:01+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,0,203.0.113.11] getting shell
2016-08-04 00:00:33+0000 [-] exitCode: 0
2016-08-04 00:00:33+0000 [-] sending request exit-status
2016-08-04 00:00:33+0000 [-] sending close 0
……..
```

コード 4．docker logs コマンドの出力例①
(IP アドレスは Documentation Address Block を使用)

コード 4．のログにおいて，攻撃者は IP アドレス 203.0.113.11 から, ユーザー名“ubuntu”, パスワード“12345”でログイン試行をしたと判断できる．仮想端末をリクエストして，攻撃者側からコネクションを切っている事がわかる．

また，shell 経由ではなく exec 経由でのコマンド実行も dcoker logs コマンドで収集できる．executing command に続くダブルクォーテーションで囲まれた文字列をコンテナ内で実行する．

```
……..
2016-07-28 08:01:08+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20] channel
    open
2016-07-28 08:01:08+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20]
    executing command "cat /proc/version"
……..
```

コード 5．docker logs コマンドの出力例②

(IP アドレスは Documentation Address Block を使用)

コード 5. のログにおいて, SSH サーバーが攻撃者から“cat /proc/version”コマンドを受信していることがわかる．

また，docker diff コマンドと docker cp コマンドを使うことにより，攻撃者がコンテナ内で活動した際，ダウンロードされたファイルや変更されたファイルの閲覧と抽出が可能になる．

```
# docker diff e0a95c7d9d52

C /home
A /home/admin
A /home/admin/.bash_history
A /home/admin/.bash_logout
A /home/admin/.bash_profile
A /home/admin/.bashrc
A /home/admin/.profile
A /home/admin/script.sh
D /home/admin/dummy.txt
C /var
C /var/log
C /var/log/faillog
C /var/log/lastlog
```

コード 6．docker diff コマンドの出力例

各行頭のシグネチャはコンテナ実行時から“A : 追加されたファイル”，“C : 変更されたファイル”，“D : 削除されたファイル”であることを示す．

```
# docker cp e0a95c7d9d52:/home/admin/script.sh script.sh
# ls
script.sh
```

コード 7．docker cp コマンドの使用例

Shell（pty）を要求した攻撃のコマンド履歴は bash history から得られる．また，.bashrc や.bash_profile 内で script コマンドを用いることでもコマンドログを取ることができる．bash history では，コマンドを実行した Unix 時間と実行コマンドを記録することができるが, script コマンドでは実行コマンドとその出力も記録することができる．

以下のコード 8．では，侵入した環境がハニーポットであるかを判断する“echo -n test”による検査を実施している．“echo -n test”を実行し，その出力を見ることで Kippo ハニーポット環境であるかどうかが判断できる．Kippo では echo コマンドのエミュレートが不完全であり，“echo -n test”を実行すると，オプション値を含んだ“-n test”が出力される．しかし，本稿で提案した SSH サーバーは echo をはじめ，コンテナ内で提供される各種コマンドは実際のバイナリであるため，この検査方法をすり抜けることができる．

```
……..
2016-07-28 08:01:07+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20]
    executing command "echo -n test"
2016-07-28 08:01:07+0000 [-] sending eof
2016-07-28 08:01:07+0000 [-] exitCode: 0
2016-07-28 08:01:07+0000 [-] sending request exit-status
2016-07-28 08:01:07+0000 [-] sending close 0
2016-07-28 08:01:08+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20] remote
    close
2016-07-28 08:01:08+0000 [SSHChannel session (0) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20] shell
    closed
2016-07-28 08:01:08+0000 [SSHService ssh-connection on
    SSHServerTransport,10,203.0.113.20] got channel session request
2016-07-28 08:01:08+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20] channel
    open
2016-07-28 08:01:08+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,10,203.0.113.20]
    executing command "cat /proc/version"
……..
2016-07-28 08:01:13+0000 [SSHService ssh-connection on
    SSHServerTransport,11,203.0.113.20] got channel session request
2016-07-28 08:01:13+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,11,203.0.113.20] channel
    open
2016-07-28 08:01:14+0000 [SSHChannel session (1) on SSHService
    ssh-connection on SSHServerTransport,11,203.0.113.20]
    executing command "cat >/tmp/.xs/daemon.armv4l.mod"
……..
```

コード 8．docker logs コマンドの出力例③

(IP アドレスは Documentation Address Block を使用)

6. おわりに

本稿では，Linux コンテナを利用した SSH ハニーポットの提案をした．Linux コンテナは OS レベル仮想化の技術で，ファイルシステムの隔離やコンピューターリソースの制限を行うことができる．Docker を使うことで予め用意した Linux ファイルシステムを，コンテナとして起動，破棄が容易に行える．Docker を用いたコンテナ技術は，攻撃者にシステム内への侵入を許すような“やられ”サーバーを運用するには最適である．Linux コンテナは通常の Linux が提供するシステムと同等の機能を提供できる．Linux コン

テナを利用した SSH ハニーポットは，従来の Kippo や Cowrie のような低対話型 SSH ハニーポットよりも更に高度な攻撃情報や，自動化された攻撃だけでなく，手動による攻撃の情報収集にも有効な手段になり得ることが期待できる．

制作した SSH ハニーポットは IP アドレスの収集，ユーザー名とパスワードの収集，コマンドの履歴，ファイルの抽出など SSH ハニーポットとして基本的な機能を有する．さらに，制作した SSH ハニーポットには Kippo や Cowrie のような低対話型 SSH ハニーポットとは異なり，実際の Linux システムで提供されているコマンドを扱うことができるという特徴を持つ．攻撃者が侵入したシステムがハニーポットであるかという判断方法に Kippo の echo コマンドを利用した検査があるが，本稿提案の Linux コンテナを利用した SSH ハニーポットは echo を利用した検査をすり抜けることができるということが収集できるログからわかった．

7. 参考文献

1. Kippo,
   https://github.com/desaster/kippo
   Cowrie,
   https://github.com/micheloosterhof/cowrie
   Honeyd,
   http://www.honeyd.org
   Dionaea,
   https://github.com/rep/dionaea
2. Hwan-Seok Yang, (2015), A study on attack information collection using virtualization technology
3. mrschyte/dockerpot: A docker based honeypot.,
   https://github.com/mrschyte/dockerpot