Computer Security Symposium 2019
21 - 24 October 2019

IoTマルウェア駆除のためのキルコマンド等の自動抽出

三須 剛史[1,a)]　岩本 一樹[1]　高田 一樹[1]　吉岡 克成[2]

概要：IoT 機器を狙ったマルウェアの感染が深刻化している．これらのマルウェアは IoT マルウェアと呼ばれ，その多くは C&C サーバからコマンドを受信し動作するという特徴がある．我々はこれまでに，IoT マルウェアに対してプロセスの終了を促す駆除情報（キルコマンド等）を擬似 C&C サーバから送信する手法を提案した．これにより，駆除情報を受信した IoT マルウェアは自身のプロセスを IoT 機器内から終了させる．駆除情報はマルウェア毎に異なるため，定期的な駆除情報の更新が必要になる．これまでの研究において，駆除情報の抽出は手動で行っているが，大量の IoT マルウェアを手動で静的解析するには多くの時間と専門的な知識が求められるため，自動で駆除情報を抽出する必要がある．本稿では，IoT マルウェアの一つである Bashlite を対象として駆除情報の自動抽出手法を検討した．本手法では，Bashlite の駆除情報であるキルコマンドがソースコード上において exit 関数の近傍に存在することに着目し抽出を試みる．実験では，検体の種類を把握するために動的解析を行い，検体から C&C サーバへ送信する特徴的な通信をシグネチャとして分類を行った．次に，関数名の特定が可能な検体から，キルコマンドを抽出するために必要な関数の命令パターンを保持し，その命令パターンを用いてキルコマンドの抽出を行った．最後に，分類を行った結果から Bashlite と思われる検体に対して，本手法で抽出したキルコマンドを送信する実験を行った結果，いくつかの検体において駆除が可能であることを明らかにした．

キーワード：IoT，マルウェア，駆除，Bashlite

Automated extraction of kill commands for disabling IoT malware

Takeshi Misu[1,a)]　Kazuki Iwamoto[1]　Kazuki Takada[1]　Katsunari Yoshioka[2]

Abstract: Malware Infection of IoT devices are becoming serious problem. We have proposed a method to disable IoT malware remotely by sending kill commands from a dummy C&C server. A drawback of the method is that we need to manually extract the kill commands from malware binaries. In this paper, we consider automatic extraction of kill commands from Bashlite, one of the IoT malware. First, we identify Bashlite binaries from our IoT malware dataset by detecting its characteristic C&C communication using dynamic analysis. Then, we statically extract kill commands from the binaries by focusing on the fact that kill commands appear near exit function in the binaries. Finally, we evaluate the extraction method and show its effectiveness by actually sending the commands to the malware executed in sandbox.

Keywords: IoT, Malware, Disablement, Bashlite

1. はじめに

IoT 機器を狙ったマルウェアの感染が深刻化している．特に活発な活動が確認されている IoT マルウェアである Mirai[1], Bashlite[2], Tsunami[3] 等は，いずれも C&C サーバからコマンドを受信し動作するという特徴がある．我々はこれまでに，マルウェアに感染した IoT 機器に対して，

[1] 株式会社セキュアブレイン
Securebrain Corporation
[2] 横浜国立大学大学院環境情報研究院/先端科学高等研究院
Graduate School of Environment and Information Sciences,Yokohama National University / Institute of Advanced Sciences
[a)] takeshi_misu@securebrain.co.jp

プロセスの終了を促す駆除情報（キルコマンド等）を擬似C&Cサーバから送信することで，プロセスを終了する（駆除）手法を提案した [4]．加えて，擬似C&Cサーバのシステム化と実用性の検討を行った [5]．擬似C&CサーバはIoTマルウェアがコマンドを受け付ける状態になるまでに行う初期通信の特徴を元にマルウェアの種類を判別する機能と，判別したIoTマルウェアの種類に適した駆除情報を送信する機能を持つ．なお，駆除情報や初期通信の内容はマルウェア毎に異なるため，常時これらの情報をマルウェア本体から抽出するなどの方法で把握し，C&Cサーバに反映させる必要がある．

我々の研究 [4], [5] において，駆除情報の抽出は全て手動で行っていた．しかし大量のIoTマルウェアを手動で解析し，駆除情報を抽出するには多くの時間と専門的な知識が必要であるため，自動的に抽出する必要がある．そこで本稿ではIoTマルウェアの一つであるBashliteを対象として，駆除情報の自動抽出手法を検討した．本手法では，Bashliteの駆除情報であるキルコマンドがexit関数の近傍に存在することに着目し抽出を試みる．実験では，検体の種類を把握するために動的解析を行い，検体からC&Cサーバへ送信する初期通信の特徴を用いて分類を行った．次に，関数名の特定が可能な検体から，キルコマンドを抽出するために必要な関数の命令パターンを保持し，その命令パターンを用いてキルコマンドの抽出を行った．最後に，抽出したキルコマンドを用いてBashliteの駆除実験を行った．これらの結果から，提案手法を用いて有効なキルコマンドを一部のBashliteから抽出可能であることを明らかにした．

本稿の構成を以下に示す．2章では関連研究について述べる．3章では擬似C&Cサーバを用いたIoTマルウェア駆除手法について述べる．4章では提案手法について述べる．5章では実験内容と実験結果について述べる．6章では実験結果に対する考察について述べる．7章ではまとめと今後の課題について述べる．

2. 関連研究

本章では関連研究について述べる．

篠宮らは [6]，インターネット上に存在するIoTマルウェアのC&Cサーバを効率的に発見するシステムを提案している．具体的には，動的解析によって，C&Cサーバとの通信時に送信するペイロードとレスポンスを取得し，これらの情報を用いてシグネチャを作成し，作成したシグネチャを基にしてC&Cサーバの探索を行っている．本稿では，C&Cサーバとの通信時に発生するペイロードの特徴をIoTマルウェアの分類に用いている．また，本稿では，疑似C&CサーバによるIoTマルウェアの駆除を最終目的としており，研究の目的が異なる．

星澤らは [7]，BOT型のマルウェアのコマンドを自動的に抽出する手法を提案している．具体的にはマルウェアが用いる標準ライブラリなどで提供される文字列処理関数に対して戻り値の書き換えなどの処理を追加し，動的解析によってコマンドの抽出を行う．本稿においては，動的解析ではなく，静的解析によってマルウェアのキルコマンドを抽出するという点が異なる．

岡田らは [8]，BOT型のマルウェアを動作させるために必要な各情報（パスワードやコマンド等）を抽出する手法を提案している．具体的には，取得したい情報の上下に存在する変化しない固有のバイト列を事前に特定している．そして，対象検体を実行したメモリダンプの結果から特定した固有のバイト列を用いて必要な情報を抽出している．岡田らはデータ領域に着目し情報の抽出を行っているのに対して，本稿では実行可能な命令に注目して情報の抽出を行う点が異なる．

羽田らは [9]，実行可能ファイルから関数を検索するBinGrepを提案している．BinGrepは，制御フローグラフの編集距離と逆アセンブルリストの一致率を基に関数を検索する．本稿のキルコマンド抽出手法は，既知の関数から抽出した特徴を比較し関数を特定する．BinGrepの手法を関数の特定に応用すれば，より高い精度で関数を特定できる可能性がある．

与那嶺らは [10]，IoTマルウェアを対象にシステムコールに注目した解析と，シンボリック実行 [11], [12] を利用するためのツールangr[13]を用いた解析について報告している．この報告によると，シンボリック実行では探索する対象の実行パスが膨大なことから，有益な結果は得られなかった．コマンドの受信からプロセスの終了に至るパスをシンボリック実行で探索することで，本稿の目的である駆除情報の抽出を行える可能性はある．しかし，何らかの方法で探索する対象を絞り込まなければ，与那嶺ら [10] の報告と同様に有益な結果は得られないと考えられる．

3. 擬似C&Cサーバを用いたIoTマルウェア駆除手法

本章では，我々の研究 [4] で提案したIoTマルウェア駆除手法について述べる．

IoTマルウェア駆除手法は，BOT型のIoTマルウェアがC&Cサーバと通信を行い特定の機能を実行することに着目した駆除手法である．なお，本稿におけるIoTマルウェアとはIoT機器に感染するBOT型のマルウェアを指す．

IoTマルウェアは，感染時に感染した機器のIPアドレスや固有の文字列などをC&Cサーバーへ送信する．これは，C&Cサーバが感染した端末を識別し，IoTマルウェアへコマンドを送信するために行われる．この感染時の通信を本稿では初期通信と呼ぶ．

初期通信にはIoTマルウェア毎に特徴的なパターンが存在するため，これを用いて種類を判別し駆除情報の送信を

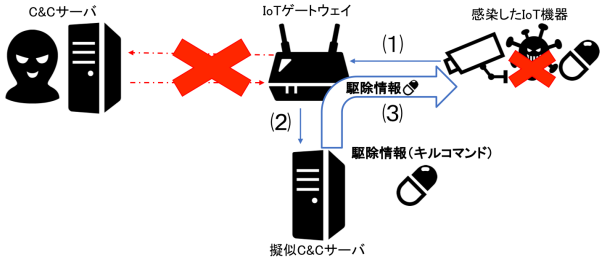

図 1　IoT マルウェア駆除手法概要

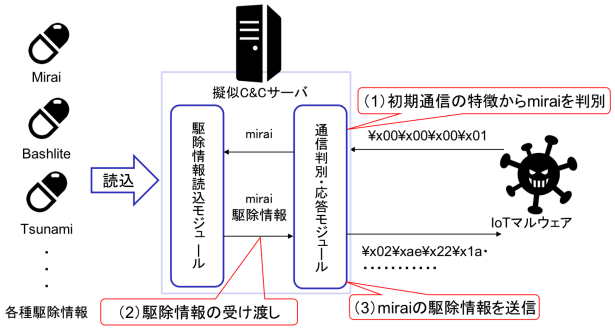

図 2　擬似 C&C サーバ構造

行う．駆除情報とは，IoT マルウェアの持つ機能を利用しプロセスの終了を促すコマンド等の情報を指す．本稿で述べる Bashlite のキルコマンドも駆除情報にあたる．なお，マルウェアの通信を擬似 C&C サーバに転送する際は，既知の C&C サーバの IP アドレスやドメイン名（アドレスリスト）を用いて転送を行う事を想定している．駆除情報の送信手順を以下に示す（**図 1**）．

(1) IoT マルウェアに感染した機器が C&C サーバへ通信する
(2) 機器からの通信先が C&C サーバのアドレスリストに一致する場合に擬似 C&C サーバへ通信を転送する
(3) IoT マルウェアに感染した機器に対して駆除情報を送信する

図 1 内の擬似 C&C サーバの構造を**図 2** に示す．擬似 C&C サーバは通信判別・応答モジュールと，駆除情報読み込みモジュールの 2 つからなる．

擬似 C&C サーバが IoT マルウェアから初期通信を受信した際の動作について説明する．まず初めに，通信判別・応答モジュールは，疑似 C&C サーバへ転送された初期通信の特徴からマルウェア種別の判別を行う（図 2(1)）．次に，判別したマルウェア種別を用いて駆除情報読込モジュールに問い合わせを行い，対応する駆除情報を取得する（図 2 (2)）．最後に，取得した駆除情報をマルウェアに送信する（図 2 (3)）．なお，1 つのマルウェア種別に対して駆除情報が複数ある場合，マルウェアと擬似 C&C サーバとのセッションが切断されるまで，リスト形式で保持している駆除情報を先頭から順次送信する．この際，リスト内の駆除情報を全て送信しても駆除が行えなかった場合，その後の駆除情報の送付は行わない．

駆除情報は，Mirai，Bashlite，Tsunami の一部に対応しているが，新種や亜種に対応するために定期的な更新が必要になる．定期的な更新を行うためには，大量のマルウェアの駆除情報を抽出する必要があり，抽出の自動化は必須である．本稿では，Bashlite を対象として自動で駆除情報を抽出する手法を提案した．Bashlite の駆除情報であるキルコマンドの抽出手法については 4 章で述べる．

4. 提案手法

本章では，IoT マルウェア駆除に必要な駆除情報の自動抽出手法について述べる．Bashlite の駆除方法は 2 種類存在する．1 つ目は重複感染防止コマンドを用いた駆除手法である．2 つ目はキルコマンドを用いた駆除手法である．どちらの手法も，擬似 C&C サーバから特定の文字列を受信した際に自身のプロセスを終了する（駆除）機能であるが，実装されている理由が異なると考えられる．重複感染防止コマンドは，同一の端末に複数の Bashlite が感染した場合に自動的に C&C サーバから送信されるコマンドであり，同一端末に複数感染した際に発生しうる誤動作を防止する目的で存在すると想定される．キルコマンドは，攻撃者が任意のタイミングで C&C サーバから送信するコマンドであり，Bashlite を制御するために，必要に応じてプロセスを終了する目的で存在すると想定される．本章ではこの 2 つのうち，キルコマンドの抽出手法について述べる．

我々は研究 [5] で，キルコマンド候補の自動抽出の調査として公開されている Bashlite のソースコードを静的解析した．**図 3** は，Bashlite のソースコードの一部である．図 3 に示す通り，キルコマンド文字列である “LOLNOGTFO” と exit 関数が近傍に存在していることを確認している．Bashlite のバイナリを逆アセンブルし，図 3 と一致する箇所を抽出した結果が**図 4** である．図 4 では一部の nop 命令は削除してある．また strcmp は strcoll の別名のため，strcmp と strcoll は同じ関数として扱う．図 4 から，文字列の参照，strcoll 関数呼び出し，exit 関数呼び出しが順番に並んでいるコードを探すことでキルコマンドを特定できる．しかし strip された（シンボルが削除された）検体では名前から strcoll 関数及び exit 関数が特定できない．そこで，あらかじめ strcoll 関数及び exit 関数の命令パターンを抽出する．strip されていない検体に対しては関数名から，strip された検体に対しては抽出した命令パターンから関数を特定した後，キルコマンドの抽出を試みる．

4.1 関数の命令パターン抽出

strcoll 関数及び exit 関数は C 言語のランタイムライブラリの関数である．通常，これらの関数のコードがコンパイル時に生成されることはなく，OS やコンパイラなどの配布者によって予め生成されたコードがリンクされる．その

```
if(!strcmp(argv[0], "LOLNOGTFO"))
{
        exit(0);
}
```

図 3　Bashlite ソースコード（一部抜粋）

```
lw    $v0, 0xC8+arg_4($fp)
lw    $v0, 0($v0)
move  $a0, $v0
la    $v0, loc_410000
addiu $a1, $v0, (aLolnogtfo - 0x410000)  # "LOLNOGTFO"
la    $t9, strcoll
jalr  $t9 ; strcoll
nop
lw    $gp, 0xC8+var_A8($fp)
bnez  $v0, loc_4098A4
nop
move  $a0, $zero
la    $t9, exit
jalr  $t9 ; exit
```

図 4　Bashlite 逆アセンブルリスト（一部抜粋）

ため，これらの関数のコードの種類は限られており，コードに基づく命令パターンによる照合で関数を特定できると考えられる．

具体的には，strip されていない実行可能ファイルの strcoll 関数及び exit 関数を逆アセンブルする．逆アセンブルリストから，オペランドを取り除きニーモニックだけを抽出し，ニーモニックの列を関数の命令パターンとする．例えば図 4 の命令パターンは $lw, lw, move, la, addiu, \cdots, la, jalr$ である．なお，命令パターンは strcoll 関数と exit 関数の両方が存在する実行可能ファイルから抽出したものを有効とする．

4.2　キルコマンド抽出

検体が strip されているときには，4.1 節の命令パターンと逆アセンブルリストを照合する．一致するニーモニックの列があるときには，strcoll 関数または exit 関数とみなす．検体が strip されていないときには，検体のシンボルテーブルから strcoll 関数及び exit 関数を特定する．

strcoll 関数と exit 関数の両方が存在するときには，キルコマンドの抽出を試みる．文字列の参照，strcoll 関数呼び出し，exit 関数呼び出しが順番に並んでいるコードがあるときには，参照している文字列をキルコマンドとして抽出する．コードの途中がジャンプ命令のジャンプ先になっている場合や，strcoll 関数及び exit 関数以外の関数を呼び出しているときにはキルコマンドとはみなさない．例えば図 4 では最初に “LOLNOGTFO” への参照があり，strcoll 関数と exit 関数を呼び出しており，コードの途中から実行されることはなく，他の関数は呼び出されていないため “LOLNOGTFO” をキルコマンドとして抽出する．

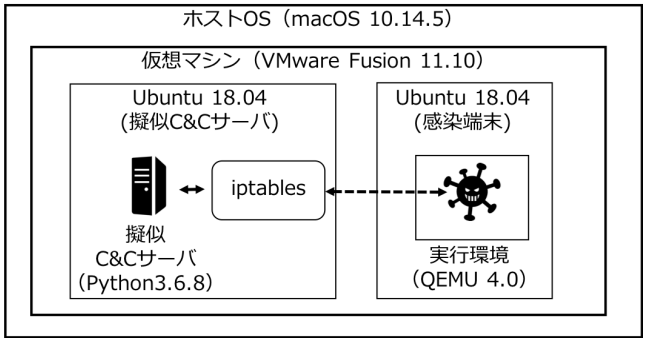

図 5　実験環境

5.　実験

本章では，実験環境と実験結果について述べる．

実験には IoTPOT[15] にて収集された MIPS アーキテクチャで動作する IoT マルウェア 3,137 検体を使用した．収集期間は 2018 年 9 月 14 日から 2019 年 4 月 10 日である．

5.1 節及び 5.4 節の実験環境を**図 5** に示す．実験環境では，macOS に VMware Fusion をインストールし，VMware Fusion 上に擬似 C&C サーバとマルウェア実行環境を用意した．擬似 C&C サーバとマルウェア実行環境は Ubuntu 上で動作する．擬似 C&C サーバは Python で実装し，マルウェア実行環境は QEMU を用いて MIPS アーキテクチャをエミュレートした．なお，今回の実験環境ではマルウェア実行環境から発生する全ての通信を iptables を用いて擬似 C&C サーバへ転送した．

5.2 節及び 5.3 節の実験は IDA Python スクリプトを作成して行った．使用した IDA のバージョンは 7.3.190614，実行環境は Windows 10 Professional（64 ビット日本語版）である．

5.1　初期通信を用いた検体の分類

提案手法は Bashlite を対象としているため，IoTPOT から得られた検体から Bahslite の検体を分類する必要がある．よって，マルウェアから擬似 C&C サーバに送信される初期通信の内容を元に分類を行った．分類はこれまでの我々の研究 [4] において得られた結果や，セキュリティベンダーが公開しているマルウェア解析レポート [14] を元に行った．初期通信の内容を用いたマルウェアの分類結果を**表 1** に示す．なお，表 1 に示した初期通信内容以外の初期通信が発生したマルウェアについては Unknown1 と定義した．また，擬似 C&C サーバへセッションを生成するが，初期通信が発生しなかったマルウェアを Unknown2 と定義した．擬似 C&C サーバへのコネクションを生成しなかったマルウェア（動作しなかったマルウェア）については ERROR と定義した．

表 1　マルウェアの分類結果

分類名	初期通信内容	検体数
Bashlite	BUILD か¥x1b を含む	103
Mirai	0x00 0x00 0x00 0x01	476
Miori	fftt:(null) を含む	61
Tsunami	PASS か NICK を含む	4
Unknown1	不明な初期通信	94
Unknown2	初期通信発生せず	25
ERROR	動作せず	2,374

表 2　抽出した関数の命令パターン

関数	命令パターン	ニーモニック数	検体数
strcoll	$lbu, lbu, \cdots, nop$	11	117
exit	$li, addu, \cdots, nop$	59	110
	$li, addu, \cdots, move$	59	7

表 3　strcoll 関数及び exit 関数の一致

		strcoll		Total
		一致した	一致しない	
exit	一致した	191	2,505	2,696
	一致しない	1	295	296
Total		192	2,800	2,992

表 4　キルコマンド抽出対象検体と結果

	strcoll 関数及び exit 関数が存在	キルコマンド抽出
strip されていない	117	26
strip されている	191	10
Total	308	36

5.2　関数の命令パターン抽出

IoTPOT で収集した 3,137 検体のうち 143 検体は strip されていなかった．これらの検体に対して 4.1 節に記述する提案手法で関数の命令パターンを抽出した．117 検体は strcoll 関数及び exit 関数の両方があり，残りの 26 検体は exit 関数だけがあった．ゆえに**表 2** に示す 117 検体から抽出した関数の命令パターンが有効である．strcoll 関数は 1 つの命令パターンが抽出され，exit 関数は同じニーモニック数の異なる 2 つの命令パターンが抽出された．

5.3　Bashlite のキルコマンド抽出

IoTPOT で収集した 3,137 検体に 4.2 節に記述する提案手法を適用した．なお，Bashlite と分類されなかった検体の中に Bashlite が含まれている可能性を考慮し，3,137 検体を抽出対象とした．提案手法を適用した結果，3,137 検体のうち 2 検体は逆アセンブルに失敗した．5.2 節の strip されていない 143 検体を除く 2,992 検体で，5.2 節で抽出した strcoll 関数及び exit 関数の命令パターンに一致した数を**表 3** に示す．

strcoll 関数及び exit 関数の両方が存在する検体について，**表 4** にキルコマンドを抽出の成否をまとめる．合計で 308 検体のうち，36 検体からキルコマンドを抽出した．

表 5　キルコマンド抽出結果と分類名

ハッシュ値	キルコマンド	strip	分類名
0490a	GURAORERIA		Unknown2
06c37	ECHOBOT		Bashlite
0801e	ECHOBOT	✓	Unknown2
2582a	EXITFAG	✓	Bashlite
259e5	ECHOBOT		Unknown2
28e0c	34892374238947900000000000000000	✓	Bashlite
30023	Bender		Bashlite
4a287	GTFOFAG	✓	Bashlite
4db82	EXITFAG		Bashlite
591c7	ECHOBOT		Bashlite
5988d	ECHOBOT		Bashlite
64570	LOLNOGTFO	✓	Bashlite
6830f	EXITFAG	✓	Bashlite
71585	LOLNOGTFO		Bashlite
78011	EXITFAG		Bashlite
834ca	EXITFAG		Bashlite
8c895	EXITFAG		Bashlite
8cf47	ECHOBOT		Bashlite
9750a	LOLNOGTFO		Bashlite
a80d3	EXITFAG		Bashlite
a8a31	EXITFAG		Bashlite
aaf54	ECHOBOT	✓	Bashlite
b8bf4	ECHOBOT		Bashlite
baff7	EXITFAG		Bashlite
bc048	ECHOBOT		Unknown2
c9f11	LOGOUT	✓	Bashlite
d3a6d	EXITFAG		Bashlite
d7d5f	ECHOBOT		Bashlite
db8d1	EXITFAG		Bashlite
e384f	34892374238947900000000000000000	✓	Bashlite
e76ba	34892374238947900000000000000000		Bashlite
edb27	LOGOUT		Bashlite
f07ba	ECHOBOT	✓	Bashlite
fa54a	LMFAO		Bashlite
fedfb	EXITFAG		Bashlite
ff5cb	KILLMYEYEPEEUSINGHOIC		Bashlite

36 検体のキルコマンドの内容と strip の有無，及び分類結果を**表 5** に示す．キルコマンドの抽出が可能であったマルウェアはそれぞれ，Bashlite が 32 検体，Unknown2 が 4 検体であった．抽出できたキルコマンドの種類と件数を**表 6** に示す．抽出したキルコマンドの件数が最も多かったのは “EXITFAG” の 12 件であり，Bashlite のソースコードに記述されていたデフォルトのキルコマンドの文字列である “LOLNOGTFO” は 3 件であった．

5.4　Bashlite の駆除

提案手法を用いて Bashlite のキルコマンドを抽出した結

```
b'[\x1b[96mINFECTED\x1b[97m] Arch: \x1b[96mMIPS \x1b[97m|| Type: BIG_ENDIAN]\n'
```

図 6 Bashlite 初期通信例 1

```
b'\x1b[34m[\x1b[31mUltron\x1b[34m] [MIPS]\n'
```

図 7 Bashlite 初期通信例 2

表 6 キルコマンド件数

キルコマンド	件数
34892374238947900000000000000000	3
Bender	1
ECHOBOT	11
EXITFAG	12
GTFOFAG	1
GURAORERIA	1
KILLMYEYEPEEUSINGHOIC	1
LMFAO	1
LOGOUT	2
LOLNOGTFO	3

表 7 IoT マルウェア駆除結果

種別	検体数	駆除数	割合
Bashlite	103	29	28.2%
Unknown1	94	4	4.3%
Unknown2	25	3	12.0%
Total	222	36	16.2%

果を擬似 C&C サーバの駆除情報読み込みモジュールに適用し，Bashlite，Unknown1，Unknown2 に分類された 222 検体に対して駆除実験を行った．実験における駆除の条件は 2 つある．1 つ目は，擬似 C&C サーバとのセッションが切断された場合である．なお，セッションの切断の確認は擬似 C&C サーバ側で検体とのセッションが切断されたことを確認する．2 つ目は，擬似 C&C サーバと通信を行ったプロセスが終了した場合である．なお，プロセスの終了の確認は VMware Fusion の持つゲスト OS のプロセス一覧を取得する機能を用いて，検体の実行時とセッションの切断後のプロセス一覧の比較を行い確認する．この 2 つの条件を満たした場合に検体を駆除したとみなす．駆除実験では，抽出した 10 種類のキルコマンド（表 6）を用いた．なお，キルコマンドは擬似 C&C サーバへのセッションが開始し，初期通信を受信した後に送信する．また，3 章で述べた通り，複数のキルコマンドが存在する場合，全てのキルコマンドを送信し終えるかセッションが切断されるまで順番に送信する．

IoT マルウェアの駆除結果を**表 7** に示す．表 7 より，検体の駆除割合は Bashlite が 28.2%，Unkown1 が 4.3%，Unknown2 が 12.0%であった．

次に，キルコマンドの抽出が可能であった 36 検体の駆除結果を**表 8** に示す．表 8 より，Bashlite は 90.6%で駆除することができた．その内抽出したキルコマンドで駆除できた検体は 19 検体であった．Unknown2 は 75.0%で駆除することができた．その内抽出したキルコマンドで駆除できた検体は存在せず，抽出したキルコマンドではなく他の検体のキルコマンドで駆除できた検体は 3 検体であった．提案手法を用いてキルコマンドの抽出が可能であった検体の駆除率は 88.9%であった．

6. 考察

本章では，5 章で述べた実験結果の考察について述べる．

6.1 マルウェアの初期通信を用いた検体の分類

Bashlite の初期通信の内容の一部を**図 6**，**図 7** に示す．Bashlite の通信パターンの多くは図 6 の様に，アーキテクチャ（MIPS）やデータ格納方式（BIG ENDIAN）を含んでいた．また，図 7 の様にマルウェア名と思われる文字列（Ultron）が含まれてる検体もいくつか存在した．Unknown1 に分類された初期通信にも同様の特徴を持つ検体が存在した．これらの検体は Bashlite の亜種と想定される．この様に，動的解析によって得られた初期通信の内容を分析することで，マルウェアの種別を特定し，分類精度の向上を図ることが可能と考えられる．

6.2 関数の命令パターン抽出とキルコマンド抽出

提案手法では，命令のオペランドを無視し，ニーモニックの列を関数の命令パターンとして抽出した．この比較手法は簡単に実装できる．しかし同じ動作であっても命令が 1 つでも異なれば別の関数と判定される．他の関数の判定方法としては，羽田らが提案した BinGrep[9] がある．これを応用すれば，関数を特定する精度が高まる可能性がある．

実験では関数の命令パターンを検体から抽出した．その結果，exit 関数では 2 種類，strcoll 関数では 1 種類のわずかな命令パターンしか得られなかった．しかし関数は C 言語のランタイムライブラリの関数が対象であるため，検体だけから抽出する必要はない．マルウェアに限らず，各種ディストリビューションに含まれる既存のライブラリや，一般に出回っている実行可能プログラムなどから広く収集すれば，多くの命令パターンを得られる可能性がある．命令パターンの数を増やすことでも，関数を特定する精度が高まる可能性がある．

表 8　キルコマンドの抽出が可能であった IoT マルウェアの駆除結果

種別	検体数	抽出したキルコマンドで駆除できた数	他の検体から抽出したキルコマンドで駆除できた数	Total	割合
Bashlite	32	19	10	29	90.6%
Unknown2	4	0	3	3	75.0%
Total	36	19	13	32	88.9%

実験では exit 関数に比べて，strcoll 関数の命令パターンに一致した数は少なかった．strcoll 関数の命令パターンは 1 種類しか抽出できていないため，他の命令パターンが存在する可能性はある．または，strcoll 関数はインライン展開されている可能性も考えられる．その場合，キルコマンドの抽出手法を見直す必要がある．

提案手法のように機械語に基づいた手法では，アーキテクチャ毎に命令パターンやキルコマンドの抽出手法を構築する必要がある．制御フローグラフなどの機械語に依らない特徴を用いれば，アーキテクチャが異なっている場合でも共通の手法でキルコマンドを抽出できる可能性はある．

また，提案手法とは異なる駆除情報の抽出するためのアプローチとして，シンボリック実行 [11], [12] を利用することも考えられる．

6.3　Bashlite の駆除

駆除実験で用いたキルコマンドで駆除できた検体とできなかった検体について，以下のケースが存在した．

- ケース 1
  どのようなキルコマンドを送信しても，セッションが切断され，プロセスも終了した検体
- ケース 2
  抽出したキルコマンドを送信することで，セッションは切断されるが，プロセスは残っている検体
- ケース 3
  他の検体から抽出したキルコマンドを送信することで，セッションは切断されるが，プロセスは残っている検体
- ケース 4
  どのようなキルコマンドを送信しても，セッションが切断されず，プロセスも残っている検体

ケース 1 に相当するハッシュ値 8cf47 の検体を静的解析した．静的解析の結果では，想定されていないコマンドや通信の形式が誤っているなどの条件で意図的にプロセスが終了するコードは確認されなかった．そのため解析を妨害するためにプロセスが終了したわけではなく，検体のバグまたは実験環境の問題が原因であると考えられる．

ケース 2 に相当するハッシュ値 baff7 の検体，ケース 3 に相当するハッシュ値 ff5cb の検体，及びケース 4 に相当するハッシュ値 0490a の検体を静的解析した．静的解析の結果では，検体は孫プロセス（子プロセスの子プロセス）または子プロセスがマルウェアとしての実質的な動作を行い，それ以外のプロセスは何もせずに終了する．また，それらの検体では抽出したキルコマンドで終了する機能を確認できた．これらのケースでプロセスが残っていた理由としては，実験環境は QEMU を用いてエミュレートしているためプロセスの作成や終了に時間がかかっている，あるいはプロセスの終了に問題があるなどの理由でゾンビプロセスになっていると考えられる．

本稿では，駆除出来たかどうかの判断をプロセスの有無によって行っていた．しかし，ケース 2，3，4 の様にプロセスが存在していたとしても，実際には終了の途中である等が考えられる．このため，exit のシステムコール等を観測し，プロセスが確実に終了した事を確認できる様な仕組みを検討する必要がある．

静的解析を行った検体では確認されなかったが，複数のプロセスが連携して動作する場合，駆除するためには全てのプロセスを終了させる必要がある．全てのプロセスを終了させるためには，BOT 型のマルウェアの多くが持つ機能である Shell コマンド実行機能を利用し，OS の持つコマンドである kill コマンドを実行して関連するプロセスを終了させる方法が考えられる．あるいは，他のマルウェアをダウンロードして実行する機能を保有するマルウェアでは該当の機能を利用し，マルウェアのプロセスを終了するプログラムを感染機器にダウンロードし実行させる手法が考えられる．

7.　まとめと今後の課題

本章では，まとめと今後の課題について述べる．

本稿では，IoT マルウェア駆除のためのキルコマンド等の自動抽出手法として，Bashlite を対象にキルコマンドの自動抽出手法を提案した．実験の結果，提案手法を用いて有効なキルコマンドを抽出可能であることを明らかにした．しかし，Bashlite に分類された検体の一部からのキルコマンドを抽出する結果にとどまった．今後は，6.2 節で述べた様な自動抽出の精度を高める手法について検討する．また，Bashlite の駆除実験において，現状のプロセス終了の判断基準では，駆除が成功していても失敗したと判断している可能性があるため，実験環境を改良する必要がある．

加えて，Bashlite 以外の IoT マルウェアである Mirai，Tsunami 等の自動抽出手法の検討や，特定のアーキテクチャに依存しない自動抽出手法についても検討する．

謝辞 本研究は，国立研究開発法人情報通信研究機構の委託研究「Web 媒介型攻撃対策技術の実用化に向けた研究開発」の成果の一部です．ご協力いただいた皆様に感謝致します．

参考文献

[1] IoT デバイスを狙うマルウェア「Mirai」とは何か??その正体と対策，https://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html

[2] threatpost，https://threatpost.com/bashlite-family-of-malware-infects-1-million-iot-devices/120230/

[3] New IoT/Linux Malware Targets DVRs Forms Botnet，https://unit42.paloaltonetworks.com/unit42-new-iotlinux-malware-targets-dvrs-forms-botnet/

[4] 三須 剛史，桃井達明，“擬似 C&C サーバを用いた IoT マルウェア駆除手法の提案”，暗号と情報セキュリティシンポジウム 2019，セッション 3E2-2，2019.

[5] 三須 剛史，高田一樹，“擬似 C&C サーバを用いた IoT マルウェア駆除手法の検討”，セッション B-1-9，2019.

[6] 篠宮一真，山村翔，荒木翔平，張一凡，胡博，神谷和憲，谷川真樹，浜田泰幸，高橋健司，“IoT マルウェアと通信可能な悪性サーバの探索”，コンピューターセキュリティシンポジウム 2018，セッション 3B1-4，2018

[7] 星澤裕二，太刀川剛，岡田晃一郎，“動的解析による BOT コマンドの自動抽出”，マルウェア対策研究人材育成ワークショップ 2008，セッション M6-1013，2008

[8] 岡田隼人，森井昌克，中尾康二，“BOT コードの静的解析によるネットワークアクセス情報抽出について”，電子情報通信学会技術研究報告. ISEC, 情報セキュリティ，Vol.107，No.345，pp.37-41，2007-11-14

[9] 羽田大樹，後藤厚宏，“BinGrep : 制御フローグラフの比較を用いた関数の検索によるマルウェア解析の効率化の提案”，情報処理学会論文誌，Vol.58，No.5，pp.1151-1162，2017-05-15

[10] 与那嶺俊，門林雄基，“IoT マルウェアの機能判別を目的とした静的解析と動的解析に基づく解析手法”，第 30 回コンピュータシステム・シンポジウム（ComSys2018），ポスターセッション，2018

[11] James C. King, “Symbolic Execution and Program Testing”, Commun. ACM, Vol.19, No.7, pp.385–394(1976)

[12] Cristian Cadar, Patrice Godefroid, Sarfraz Khurshid, Corina S. Păsăreanu, Koushik Sen, Nikolai Tillmann, Willem Visser, “Symbolic Execution for Software Testing in Practice: Preliminary Assessment”, Proceedings of the 33rd International Conference on Software Engineering (ICSE ’11), New York, NY, USA, pp.1066–1071(2011)

[13] angr，https://angr.io/

[14] New Miori Variant Uses Unique Protocol to Communicate with C&C，https://blog.trendmicro.com/trendlabs-security-intelligence/new-miori-variant-uses-unique-protocol-to-communicate-with-cc/

[15] Y. M. P. Pa et al., “IoTPOT: A Novel Honeypot for Revealing Current IoT Threats”, J. Information Processing, vol. 24, no. 3, pp. 522-533, 2016.